CHALMERS, S2

Styrteknik PLC

Morgan Osbeck, Göran Hult

2016

Del 1

Innehåll

Kap 1. Inledning

Människan anses av naturen vara lat och, om man drar slutsatser av den tekniska utvecklingen, så söker hon i alla fall att finna lösningar som befriar henne från tunga, tråkiga och monotona arbetsuppgifter. Verksamheter runt omkring oss automatiseras i allt högre grad och förutsättningen för detta är möjligheten att styra all den utrustning som skall göra arbetet för oss. Det är här styrtekniken med PLC-styrningar kommer in, det kunskapsområde som behandlar tekniken att styra de mer eller mindre komplicerade tekniska processer vi har skapat runt omkring oss.

Det första man tänker på i sammanhanget är kanske automatiserade tillverkningsprocesser som t ex hopsättning av en bilkaross i Volvos karosseriverkstad. Där finns utgångsmaterialet i form av pressade plåtsjok. Där finns fixturer som placerar de olika plåtsjoken i rätt läge i förhållande till varandra. Där finns svetsutrustning som fogar samman de olika delarna till en hel kaross. För att sedan få denna tillverkningsprocess att löpa krävs en styrning. Ett styrsystem beordrar hanteringsutrustning att lägga in plåtar i fixturen, kollar via närvarogivare att plåtarna ligger rätt innan det beordrar robotarnas svetstänger att lägga punktsvetsar på önskade ställen m.m. Det sammanlagda antalet in - och utgångar hos det styrsystem som styr denna del av hopsättningen är fler tusen stycken. Utgångar är beordringssignalvägar från styrsystem till processen och ingångar är kvittenssignalvägar från process till styrsystem.

Men inte bara industriella tillverkningsprocesser använder sig av PLC-teknik för att styra och reglera. Fastigheters värme- och ventilationssystem styrs idag av PLC-system liksom infrastrukturanläggningar som vattendistribution med tryck- och flödesregleringar av olika pumpstationer i nätet liksom vägtunnlar med styrning av fläktar, belysning och trafikövervakning. Dessa anläggningar tar normalt upp stora geografiska områden, ett fastighetsbolag har fastigheter på många ställen i stan och vattenledningsnätet täcker hela kommuner, vilket gör behovet stort av att kommunicera information mellan de utspridda styrsystemen och operatörsstationer placerade i centralt kontor.

Denna första del av kompendiet vill ge en grund för att lösa PLC-styrning av automatiserade förlopp och att bringa struktur i dessa lösningar. Förkunskaper som krävs för att tillgodogöra sig materialet är grundläggande logik och även grunden i reglerteknik i den del som berör PID-regulatorn. PLC-tekniken baseras idag på en programmeringsstandard, IEC 61131-3 och de första kapitlen baseras helt på denna standard. Det finns dock en uppsjö av PLC-fabrikat på marknaden och alla har sin programutvecklingsmiljö där de flesta stödjer sig på standarden men har också en del fabrikatsspecifika funktioner utöver standarden. I denna skrift är Mitsubishis PLC-system och deras utvecklingsmiljö GX IEC Developer det system och den utvecklingsmiljö som använts som ”referenssystem” dvs som miljö för programexempel och för exemplifiering av hur fabrikanternas egna funktioner kan komplettera standarden. Standarden kom till efter att PLC:er funnits på marknaden i 20-talet år och då är det inte lätt att enas om en heltäckande standard som alla tillverkare är beredda att anamma fullt ut.

Kap 2. Funktionsbeskrivningar

Inför utveckling och konstruktion av en maskin eller process är funktionsbeskrivningen som underlag för styrningen av maskinen eller processen en viktig del där många aktörer är inblandade. Vid upphandling av utvecklings- och konstruktionstjänster ställs en kravspecifikation upp där det är mycket viktigt att vara tydlig vid kravformuleringen. En viktig del i denna kravspecifikation är att funktionsbeskrivningen är entydig, att beställaren på ett otvetydigt sätt för utföraren kan redogöra för det önskade beteendet hos den önskade produkten, för att därmed undvika missuppfattningar, extraarbete och framtida tvister. Ju tidigare en svaghet eller ett fel upptäcks i en utvecklingsprocess desto billigare är det att åtgärda. Om man exempelvis i funktionsbeskrivningen för styrningen av en produktionsanläggning för ”gosenallebjörnar” har med fastnästning av nallebjörnens vänstra öra men missar proceduren att slutligen sy fast det med en rejäl söm. Om detta upptäcks vid igångkörning av anläggningen kommer detta att förorsaka en del kostnader i form av ändrat processflöde, ändring av programvara, ändring av produktionsutrustning och försenad produktionsstart. Ännu värre är om det inte upptäcks i det steget heller utan ett antal tusen nallebjörnar har hunnit ut på marknaden och ett barn har bitit loss ett öra och kanske satt i halsen med följd stora tidningsrubriker och förlorad varumärkesstatus och därmed stora ekonomiska verkningar. Viktigt är alltså att kravspecifikationer och i dessa ingående funktionsbeskrivningar blir riktiga från början.

För att en funktionsbeskrivning skall bli bra och uppfylla sitt syfte krävs det att sättet att beskriva funktionen på är sådant att alla parter som har kunskap, som är relevanta för att nå rätt funktion, också skall förstå beskrivningssättet av funktionen lika väl som att de som skall realisera funktionsbeskrivningen i form av t ex utrustning och styrprogram skall uppfatta funktionsbeskrivningen rätt dvs förstå beskrivningssättet och inte riskera misstolkningar p g a beskrivningssättet.

Kravet på en funktionsbeskrivning för styrning av en process är således att:

- Den kan förstås av alla personalkategorier som är involverade i processens styrning såsom driftpersonal, underhållspersonal, projektansvariga, styrsystemprogrammerare, konstruktörer.

- Den utgör ett entydigt och utrustningsneutralt underlag för utformning och programmering av styrutrustningen.

- Den skall kunna användas och återkopplas till vid projektering, konstruktion, programmering, igångkörning, felsökning och underhåll av processen.

- Den utgör ett underlag för anläggningens dokumentation.

Här följer några sätt att beskriva funktioner hos en processtyrning med huvudvikten lagd på funktionsdiagrammet i avsnitt 2.4.

2.1. Verbal funktionsbeskrivning

Det mest naturliga sättet för gemene man är att beskriva en funktion i verbal form. För enklare och mer övergripande beskrivningar kan ofta den verbala formen bli mest

överskådlig för alla inblandade parter. Övergripande beskrivningar kan vara säkerhets-funktioner såsom nödstopp och skyddsutrustnings interagerande med styrningen, underhållsfunktioner såsom drifttidsövervakningar för olika processkomponenter m m. Dessa övergripande funktioner måste naturligtvis noggrant integreras i styrprogrammet men kan ofta med fördel separeras ifrån beskrivningen av den grundläggande funktionen hos processen. En verbal beskrivning kan med fördel kompletteras med figurer för bättre förståelse.

Att använda verbal beskrivning för att funktionsbeskriva komplexa styrförlopp är inte att rekommendera då beskrivningen blir ordrik, svårläst och också svår att få entydig och korrekt.

Exempel 2.1*:*Verbal beskrivning av beteendet hos en markis.

Figur 2.1: Markis över uteplats.

Markisen bestyckas med givare och manöverdon enligt

Figur 2.2 nedan.

Bestyckning:

MAN/AUTO: Vred för val av manuell / automatisk markismanövrering.

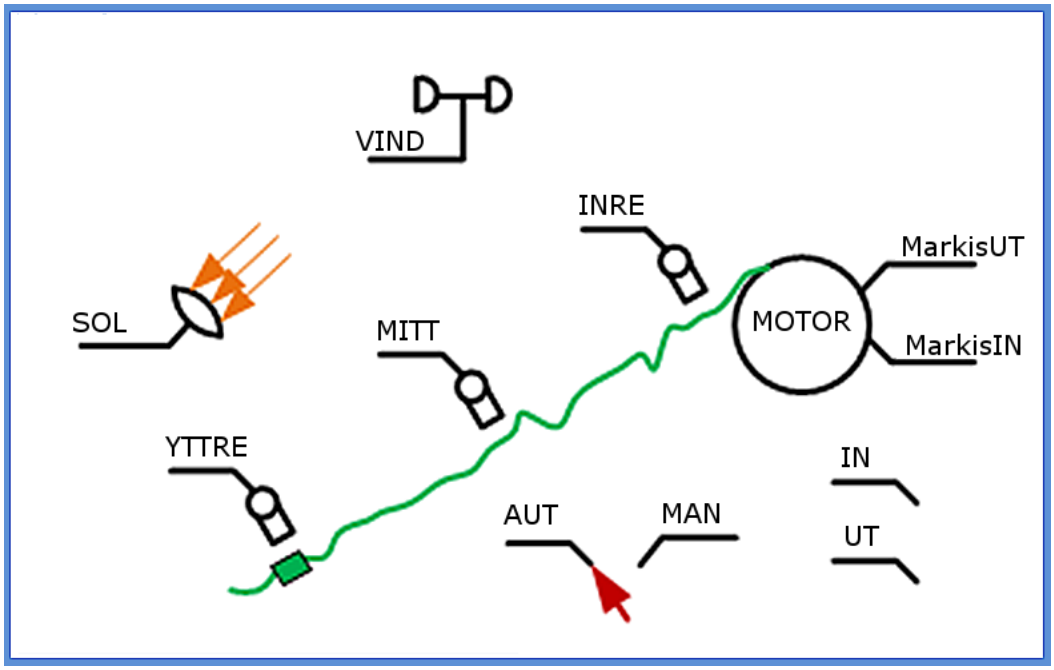
IN / UT: Tryckknapp för manuell körning in / ut.

YTTRE/MITT/INRE: Givare som känner av yttre / mittre / inre läge hos markis.

SOL: Givare som känner av solintensitet – omslag inställbart.

VIND Givare som känner av vindstyrka – omslag inställbart.

MOTOR: Motor för ut- och inkörning av markisen.



Figur 2.2: Givare och manöverdon för styrautomatik till markis.

Funktion:

1. Med omkopplare väljs MAN/AUT – manuell/automatisk manövrering.

2. I MAN-läge körs markisen in /ut med tryckknappar IN / UT. Påverkad tryckknapp innebär körning av markis.

3. I AUT-läge styrs markisen enligt pkt 4-7.

4. Om solintensitet hög och vindstyrka låg kör markis till yttre läge.

5. Om solintensitet hög och vindstyrka hög kör markis till mittläge.

6. Om solintensitet låg och oavsett vindstyrka kör markis till inre läge.

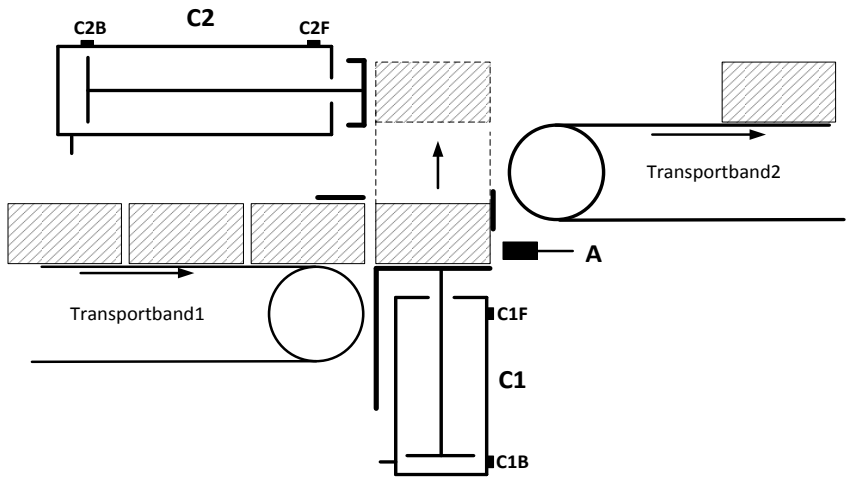
7. Om markis står stilla och ändå inget av inre / mitt / yttre gränsläge är påverkat kör inåt.

Vi stannar där för nu är grundfunktionen beskriven men ytterligare förreglingar behövs för att gardera systemet mot oönskat beteende vi bortfall av någon givarfunktion m m.

2.2. Följddiagram.

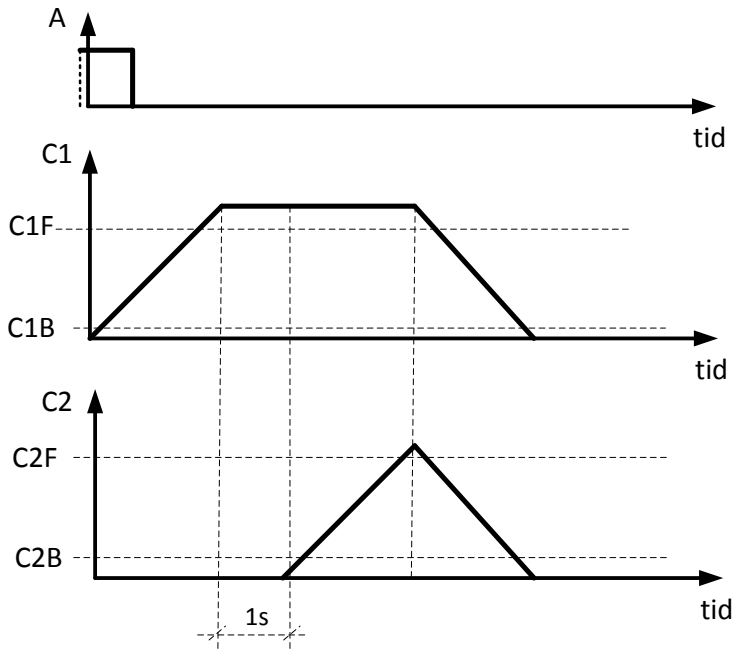
Följddiagram eller väg-tid-diagram är ett sätt att beskriva en tidssekvens av händelser. Framför allt används det för beskrivning av förflyttningssekvenser där traditionellt pneumatiska cylindrar förflyttar, stoppar, håller fast osv material eller komponenter i en behandlingskedja.

Vi betraktar ett förlopp enligt Figur 2.3, där lådor transporteras på transportband1, detekteras av en givare A, lyfts av cylinder C1 till nivån för band2 (C1F) stabiliseras där i 1 sekund varefter cylinder C2 skjuter ut lådan (C2F) på transportband2 varefter cylindrarna går tillbaka till ursprungslägena (C1B resp C2B).



Figur 2.3: Vertikalförflyttning av lådor.

Följddiagrammet nedan, Figur 2.4 beskriver styrföljden för de två cylindrarna då signal ges från givare A. Detta beskriver bara själva styrföljden. Ytterligare funktioner som är inblandade i styrningen är troligen att något driftvillkor är uppfyllt dvs att hela anläggningen, där denna del ingår, är i driftläge. Vidare bör framgå vad som skall hända med styrningen efter nödstopp av anläggningen. Dessa ytterligare krav på styrningen kan med fördel beskrivas verbalt.

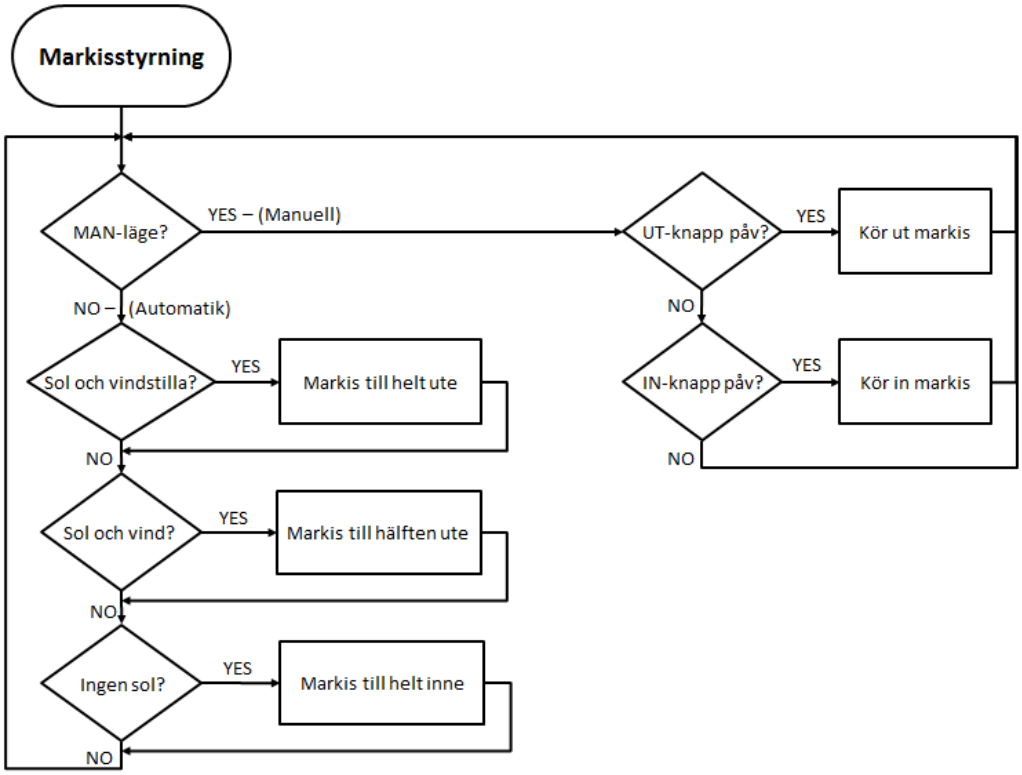


Figur 2.4: Följddiagram för cylinderrörelser vid vertikalförflyttning.

2.3. Flödesschema.

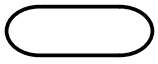
Flödesschema är det klassiska sättet att beskriva programflöden och används också i styr-sammanhang för att beskriva förlopp som skall omsättas till program för styrsystem.

Det önskade beteendet hos markisstyrningen i avsnitt 2.1 beskrivs i flödesschemaform i Figur 2.5 nedan. Detta flödesschema är beteendebeskrivande utan att beteendet rent tekniskt skall lösas med givare och ställdon. Denna typ av flödesschema kan alltså tjäna som diskussionsunderlag för alla kategorier av inblandade intressenter i utvecklingen av markisen.

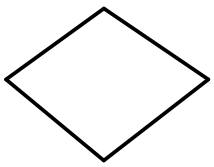


Figur 2.5: Flödesschema för markisstyrning - beteendebeskrivande.

För att bygga upp flödesdiagram används en ett antal symboler varav de viktigaste fem är följande:

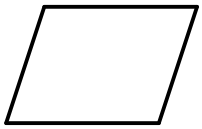


Programstart och -stopp anges med denna symbol.

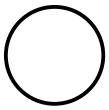


Symboliserar val eller beslut och utgörs normalt av en fråga och har därmed mer än en utgång, vanligen två – ”ja” resp. ”nej”.

Symboliserar process vilket innefattar händelser och beräkningar som inte omfattas av andra symboler.

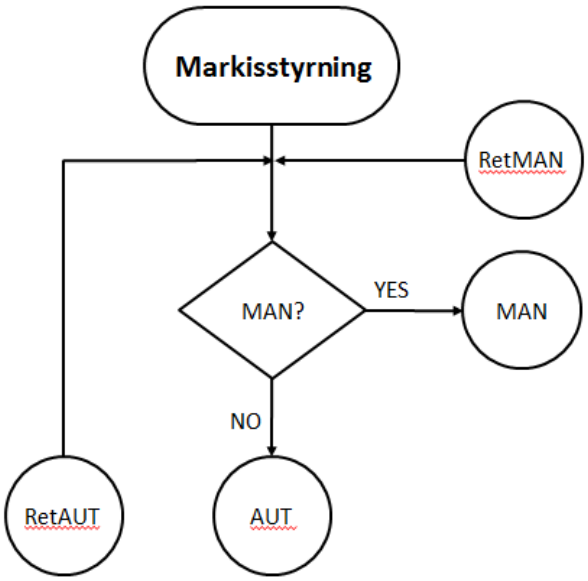


Input och output anges med denna symbol. Vid flödesschema för processtyrprogram förutsätts att kontinuerlig in- och utmatning av processignaler sker varför inte denna symbol används för detta.



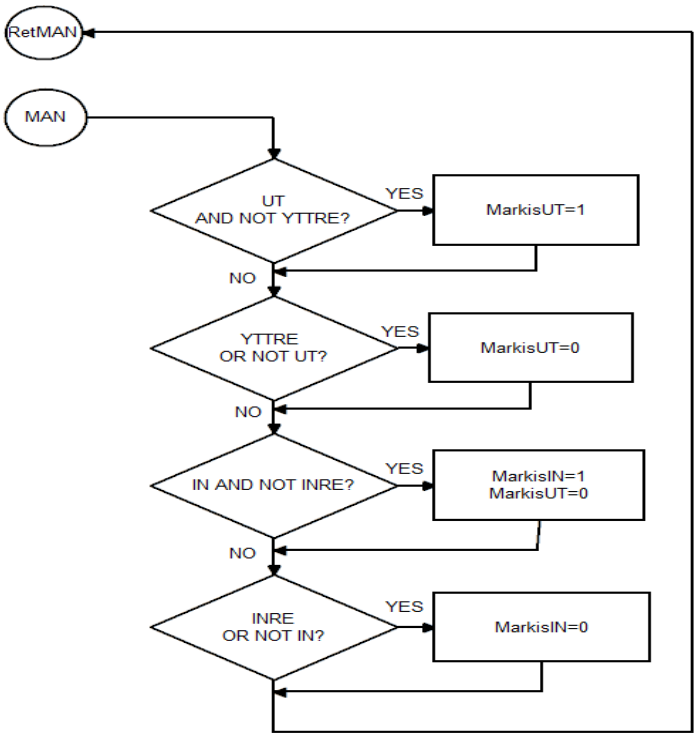
Anger förbindelse till annat flödesschema. Används då hela schemat inte får plats på en A4-sida.

Vi återvänder till markisstyrningen och lägger ett styrtekniskt perspektiv på flödesschemat. Med de signalbenämningar som framgår av Figur 2.2 kan nu styrningen beskrivas enligt flödesdiagrammen Figur 2.6 – Figur 2.8. Läsbarheten hos flödesschemat ökar om det går att på ett strukturerat sätt dela upp flödena i mindre delar. Här redovisas manuella körningen för sig i Figur 2.7, den automatiska styrningen i Figur 2.8 samt växlingen mellan manuell och automatik i Figur 2.6.

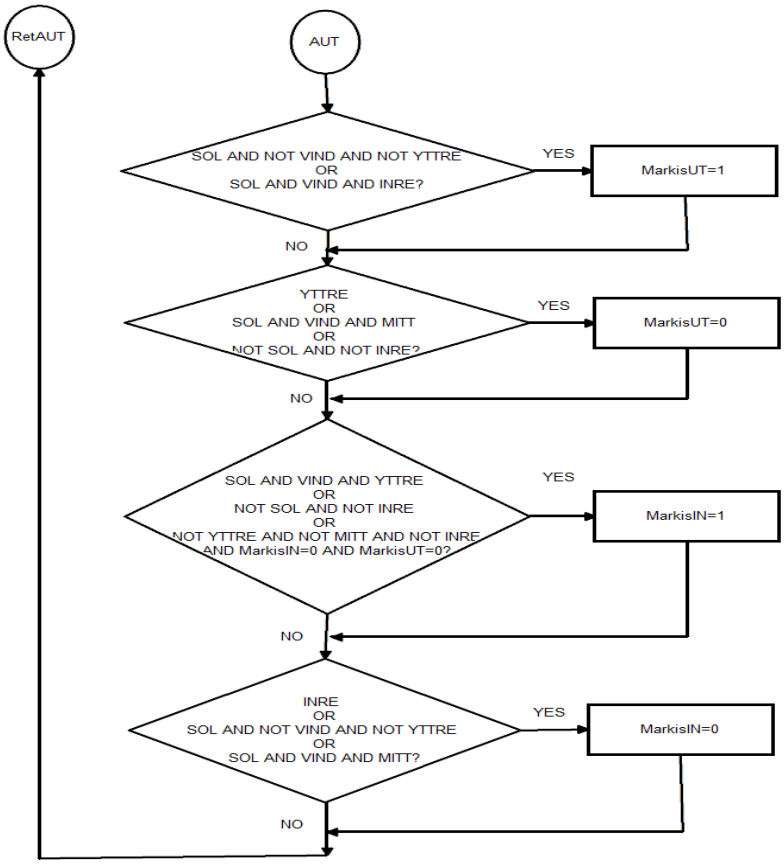


Figur 2.6: Flödesschema för markisstyrning, val man/aut - styrsignalsbaserat.

Detta är en så kallad förreglingsstyrning, till skillnad mot följdstyrning, vilket innebär att om vissa villkor är uppfyllda skall något specifikt hända, vid andra villkor skall något annat hända o s v. Det medför att då ett beslut gjorts som resulterar i någon av vägarna ut ur beslutssymbolen kommer, efter eventuell händelse, flödet att fortsätta nedåt till nästa beslutsvillkor. Flödet stannar aldrig upp utan en loop löper runt och avfrågar varje beslutsvillkor kontinuerligt. Denna typ av styrningar realiseras med logisk kombinatorik i form av reläkopplingar, logikkretsar eller program baserade på logiska uttryck eller grindar.

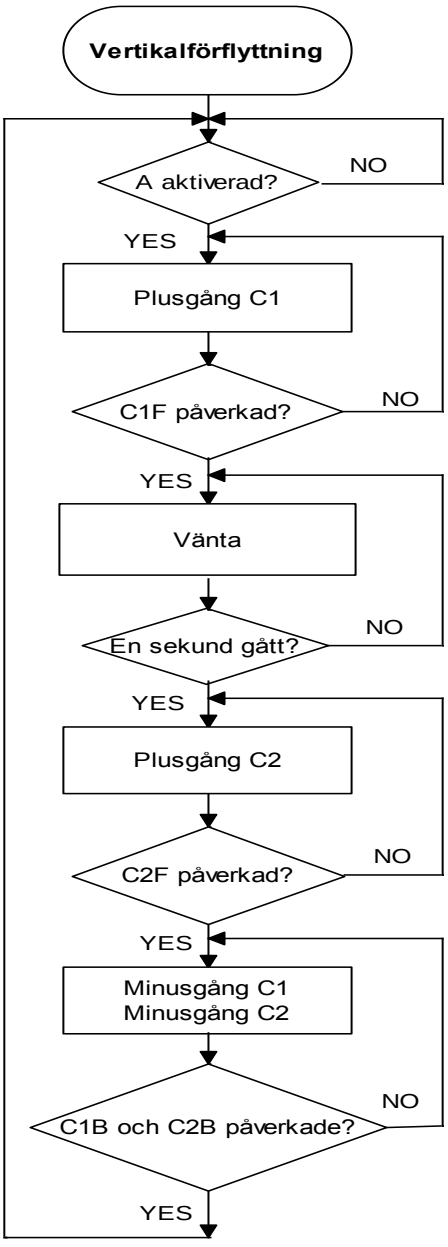


Figur 2.7: Flödesschema för markisstyrning, manuell körning - styrsignalsbaserat.



Figur 2.8: Flödesschema för markisstyrning, automatisk styrning - styrsignalsbaserat.

Betraktas istället vertikalförflyttningen enligt avsnitt 2.2 vilket utgör en följdstyrning eller sekvensstyrning så resulterar det i följddiagram enligt Figur 2.9. Här bromsas flödet upp genom ett antal återhopp så länge en händelse inte är slutförd. Omsätts detta flödesschema till ett styrprogram kommer loopningen i NO-loopen normalt att innebära ett uthopp till andra rutiner (ej med i detta flödesschema) för att snart återvända hit för ytterligare en beslutskontroll. Detta för att styrsystemet troligen har mer än denna delprocess att hålla reda på och då får inte programmet låsas i väntan på att givare skall påverkas utan mycket annat kan hinnas med i denna väntan. Ett annat sätt att beskriva sekvenser är med funktionsdiagram enligt nästa avsnitt som utgör bakgrund till ett grafiskt programmerings-språk för sekventiella förlopp, SFC.



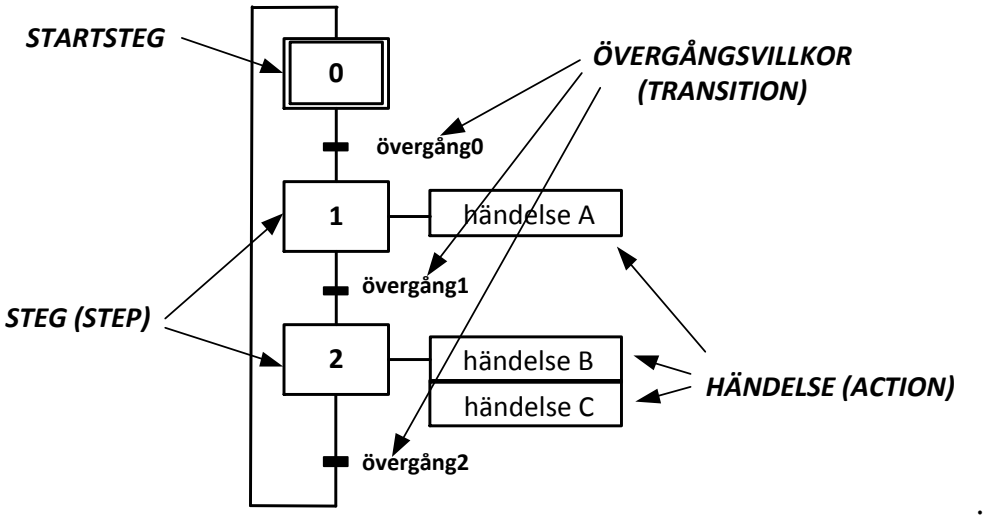
Figur 2.9: Flödesschema för följdstyrningen av två cylindrar - styrsignalsbaserat.

2.4. Funktionsdiagram.

Funktionsdiagrammet eller F-diagrammet (eng. Sequence Function Chart) är ett standardiserat (IEC 848) grafiskt beskrivningssätt för styrning av processer. Ursprungligen togs det fram av det franska företaget Telemecanique, numera Schneider Electric, som ett beskrivningssätt att använda i samband med deras pneumatiska sekvensregister. Telemecanique mönsterskyddade detta beskrivningssätt under benämningen GRAFCET. Att det ursprungligen togs fram för sekvensstyrningar gör att det är mycket användbart i den typen av styrning men är även möjligt att använda vid beskrivning av förreglingar. Sekvensstyrning innebär förlopp av en serie händelser där en händelse kvitteras innan nästa händelse tar vid.

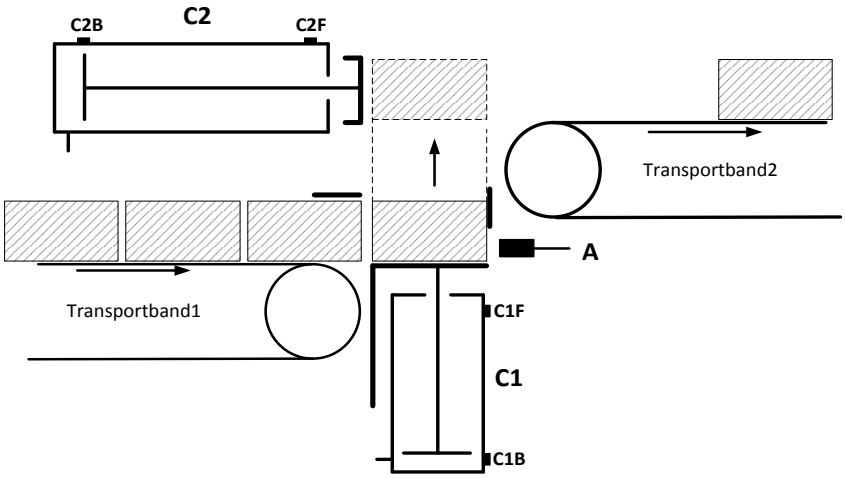
Funktionsdiagrammet uppfyller de krav som kan ställas på funktionsbeskrivningar nämligen att det skall enkelt förstås av alla personalgrupper, vara utrustningsneutralt, och kunna användas vid projektering aven anläggning såväl som vid programmering, igång-körning, underhåll, felsökning och dokumentation. Funktionsdiagrammet är också bas för ett av de standardiserade programmeringssätten för programmerbara styrsystem.

I Figur 2.10 visas funktionsdiagrammets principiella uppbyggnad med några få steg från början i en sekvens. Rutorna beskriver steg (step) eller tillstånd hos sekvensen som placeras utmed en förloppslinje som alltid ritas vertikalt men kan förgrenas horisontellt. Startsteget utgör startpunkten vilket oftast är processens viloläge innan förloppet startats. Startsteget ritas med dubbel ram medan övriga med enkel. För att förflyttning skall ske från ett steg till nästa måste övergångsvillkoret (transition) vara uppfyllt. Övergångsvillkoret kan vara allt från ett enkelt villkor till ett omfattande logisk samband men oavsett omfattning kan övergångsvillkoret endast resultera i att det är sant eller falskt. För att ett steg skall bli aktivt krävs att föregående steg är aktivt och att övergångsvillkoret till nästa steg är sant. Då nästa steg blivit aktivt avaktiveras föregående steg. Endast ett steg kan alltså vara aktivt vid en tidpunkt med undantag då parallella processer genomförs, se senare. Startsteget är aktivt från början och avaktiveras då första steget aktiveras. Till varje steg och även startsteget kan kopplas en eller flera händelser (actions). Dessa händelser kan också vara villkorade vilket framgår senare.

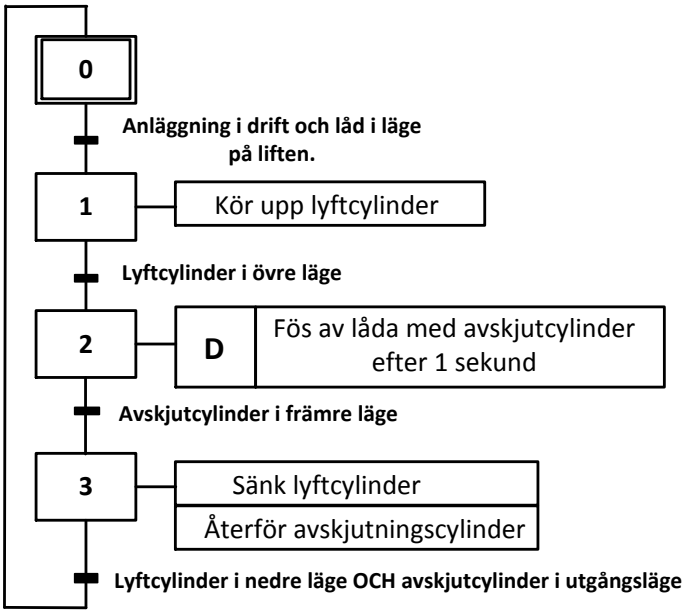


Figur 2.10: Funktionsdiagram (Funtion Chart) – principiell uppbyggnad..

Vi återknyter till den vertikallyftprocess som beskrevs i avsnitt 2.2 och återfinns i Figur 2.11 nedan. I Figur 2.12 ges ett beteendebaserat funktionsdiagram för förloppet och i Figur 2.13 ges ett styrsignalbaserat där det förutsätts att cylindrarna styrs via dubbelt styrda arbetsventiler med två styrsignaler vardera, VC+ resp VC-. I Figur 2.14 återges två alternativ av samma styrsignalbaserade funktionsdiagram men nu med förutsättningen att cylindrarna styrs via arbetsventiler med fjäderretur med en styrsignal vardera, VC för aktivering av plusgång.

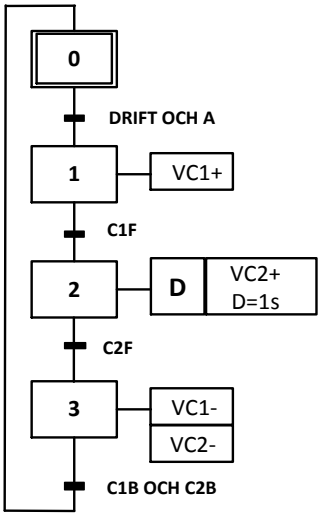


Figur 2.11: Vertikalförflyttning av lådor.

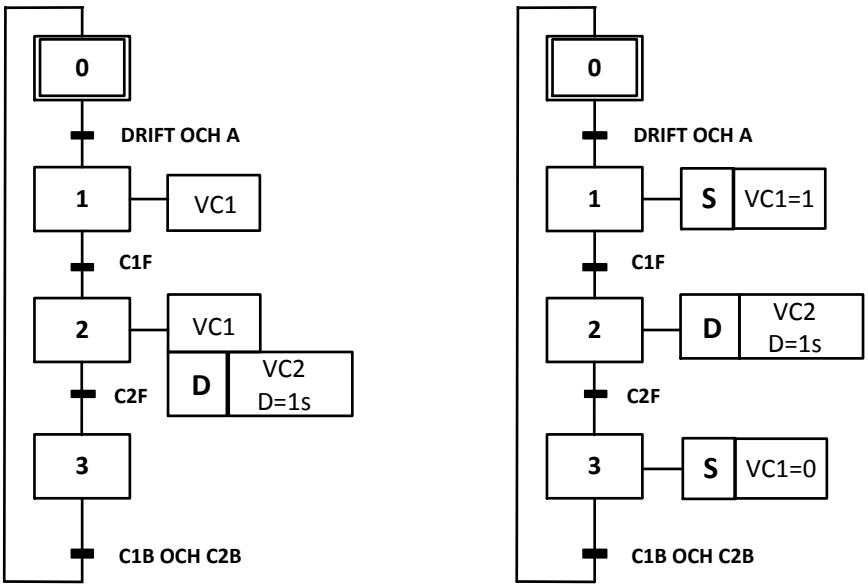


Figur 2.12: Funktionsdiagram – beteendebaserat.

Det beteendebaserade funktionsdiagrammet följer F-diagramstrukturen med klartext varför det är tolkningsbart för stora personalkategorier och bra diskussionsunderlag vid fastställande av beteende.



Figur 2.13: Funktionsdiagram – styrsignalbaserat för dubbelt styrda ventiler.

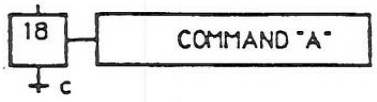
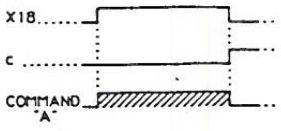
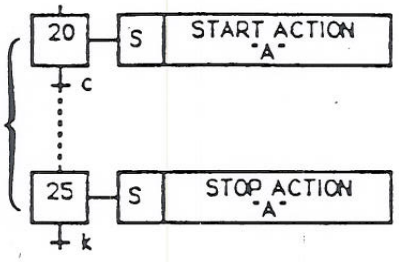
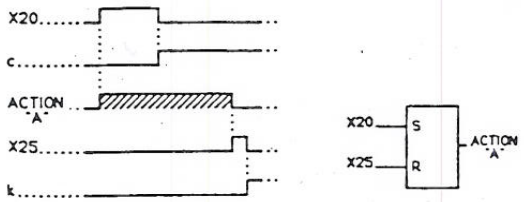
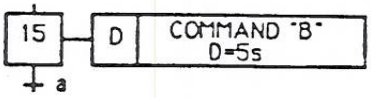
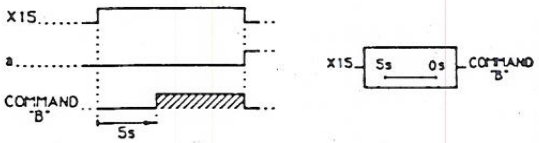
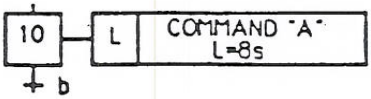
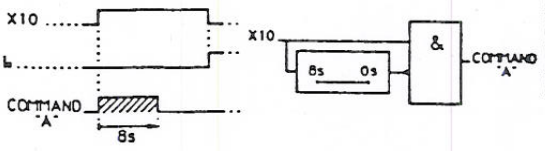


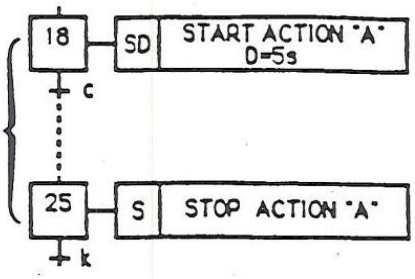
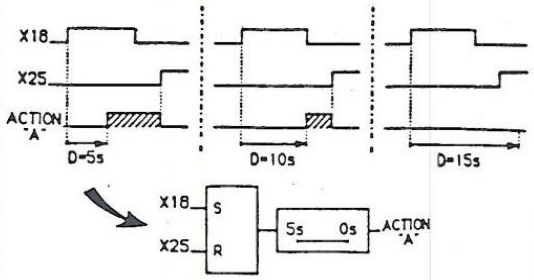
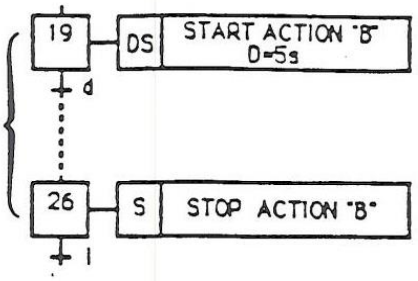
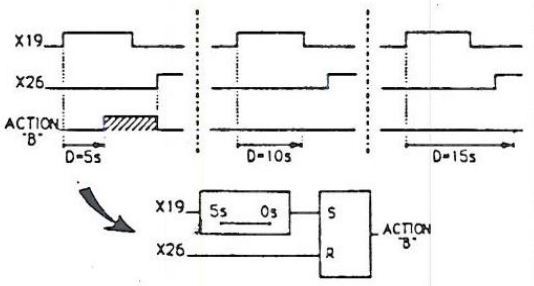
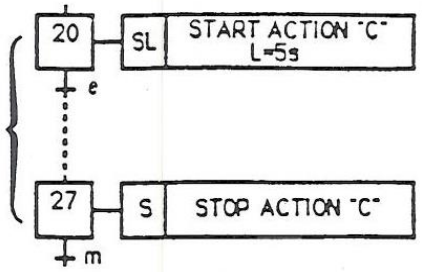
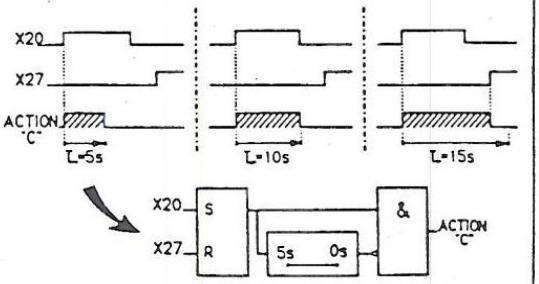
Figur 2.14: Funktionsdiagram – styrsignalbaserat för fjäderreturventiler.

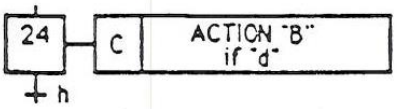
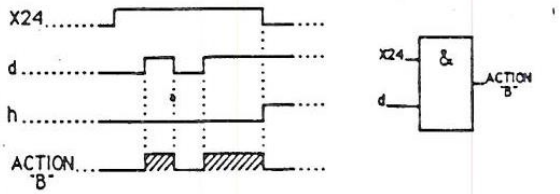
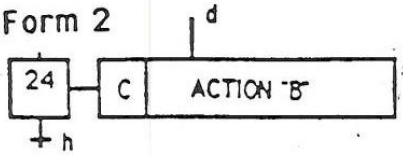
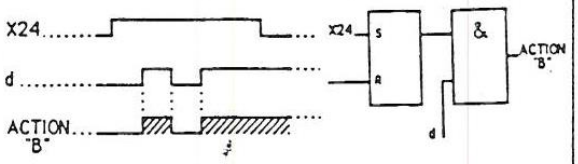
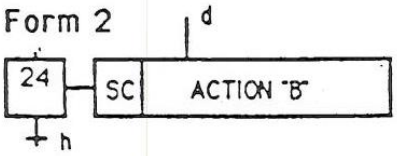
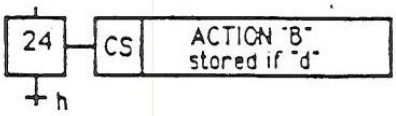
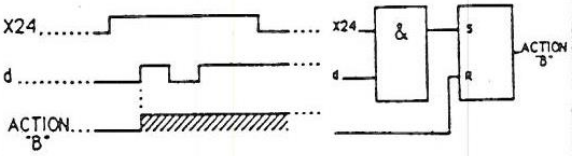
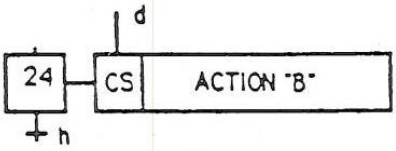
Som framgår av de tre styrsignalbaserade funktionsdiagrammen så är strukturen exakt den samma som för det beteendebaserade men nu bestyckat med processignaler. Dessutom är signalgivningen beroende av vilken typ av givare och ställdon som processen bestyckas med. Observera att i vänstra alternativet i Figur 2.14 finns ingen signalgivning i steg 3 vilket innebär att fjäderreturen ser till att båda cylindrarna går minus vilket skulle ske. Detta leder dock till att ett styrsignalbaserat funktionsdiagram blir mer svårtolkat än beteendebaserade.

Det har också dykt upp några modifierare i form av D och S. D står för delay och betyder att en tidsfördröjning skall löpa ut innan händelsen sker. Fördröjningens storlek anges i händelserutan med D=x s. Modifieraren S står för stored, dvs kom ihåg tillståndet hos signalen i följande steg. Här 1-ställs VC1 i steg 1, hålls i detta tillstånd, och nollställs i steg 3. Alternativet till S är att upprepa samma signal att vara aktiv i flera steg som i vänstra alternativet i steg 1 och 2.

Här följer tre sidors utdrag ur standarden IEC848 hur olika händelser (actions) kan beskrivas:

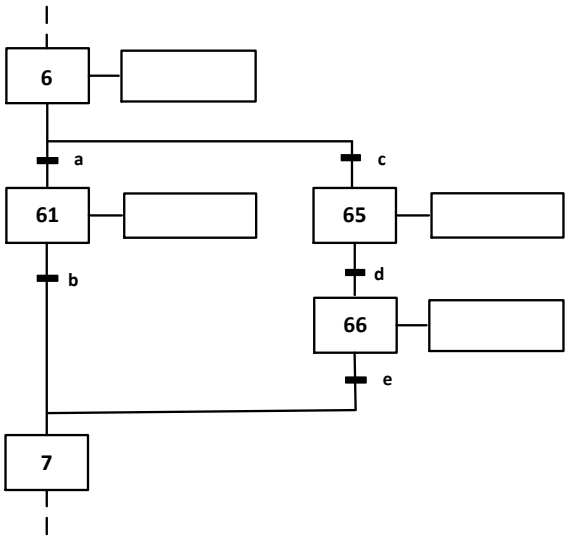
No.	Symbol	Description
5.2		Not stored command 
5.3		Stored action 
5.4		Not stored but delayed command 
5.5		Not stored but time limited command 

No.	Symbol	Description
5.6		Stored and delayed action 
5.7		Delayed and stored action 
5.8		Stored and time limited action 

No.	Symbol	Description
6.1	Form 1 	Conditional action 
6.2	Form 2 	
6.3	Form 1 	Stored and conditional action 
6.4	Form 2 	
6.5	Form 1 	Conditional and stored action 
6.6	 Form 2	

Alla processer utgörs inte av en enda rak sekvens som följer en förloppslinje utan kan förgrenas på olika sätt till alternativa eller parallella sekvenser.

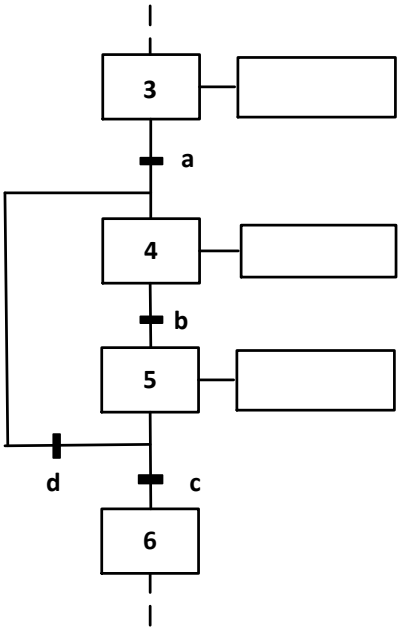
Alternativa sekvenser:



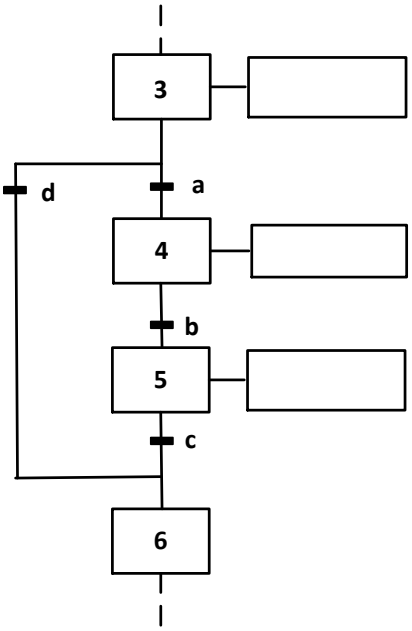
Endast en väg är möjlig, om a är sant efter steg 6 följer steg 61, om är c aktivt följer steg 65, om både a och c aktivt väljs från vänster dvs steg 61.

Hopp – en form av alternativa sekvenser:

Bakåthopp - repeterad sekvens

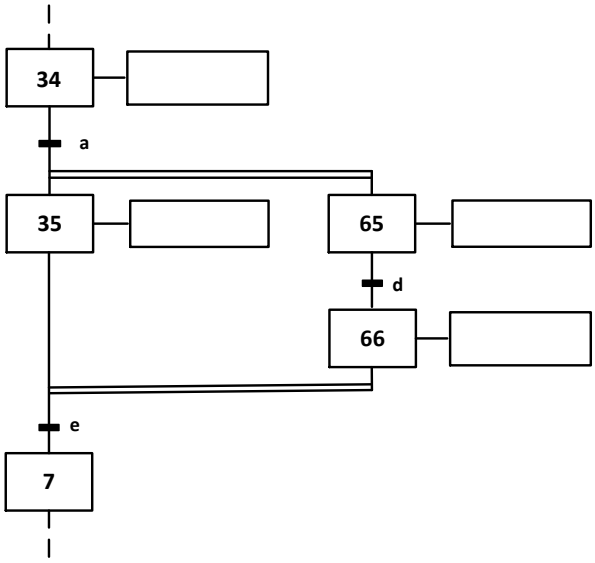


Framåthopp - sekvenspassage



Observera att det alltid skall finnas ett och endast ett övergångvillkor mellan två på varandra följande steg.

Parallella sekvenser:

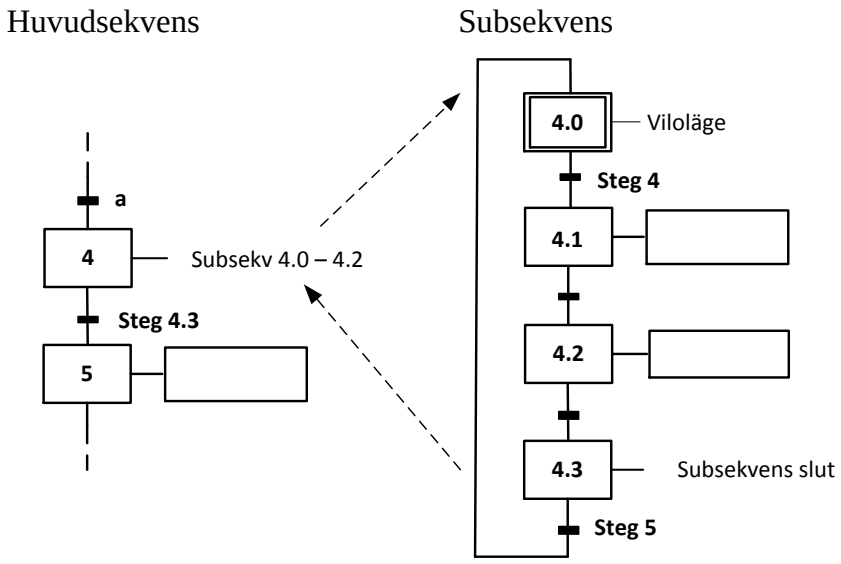


Vid parallella förlopp väljs båda vägarna, från steg 34 och aktiv signal a aktiverar både steg 35 och steg 65. Vid parallella förlopp är alltså ett steg i varje förgrening aktivt. Därefter genomlöps var och en av grenarna i sitt tempo. Vid avslutning av parallella förlopp inväntar alla parallella grenar varandra och går vidare via gemensamt övergångsvillkor. I detta fall krävs för att komma till steg 7 att förloppet befinner sig i steg 35 och i steg 66 samt att övergångsvillkor e är uppfyllt.

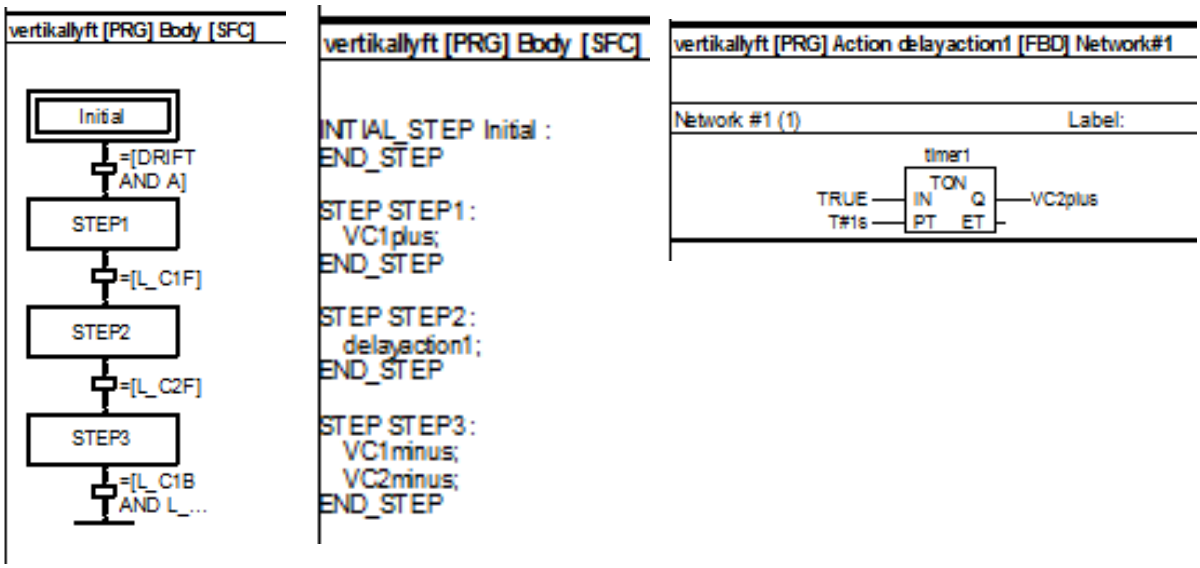
De horisontella förloppslinjerna vid förgreningar utgörs av en enkel linje vid alternativ förgrening och av dubbellinje vid parallell förgrening.

Subsekvenser:

För att minska detaljpackningen i ett funktionsdiagram kan en händelse i ett steg också utgöras av en sekvens, en subsekvens, vilken kan beskrivas i ett eget funktionsdiagram Observera hur stegen används som övergångsvillkor föra att styra sekvensflödet.



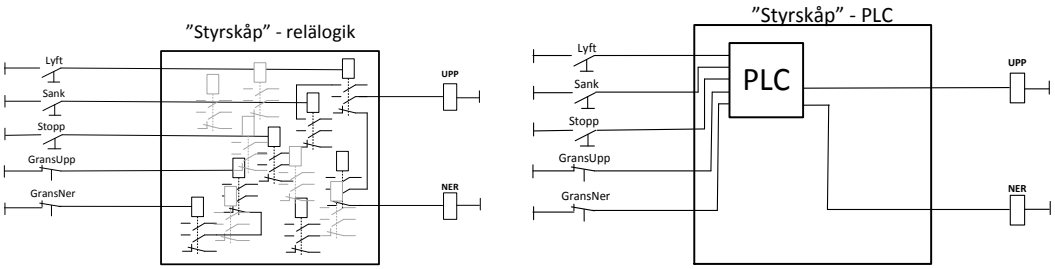
Som tidigare nämnts utgör funktionsdiagrammet basen för ett av de standardiserade (IEC 61131-3) programspråken för programmerbara styrsystem, PLC. Som avslutning på avsnittet visas i Figur 2.15 ett utdrag ur dokumentationen för ett program skrivet i språket Sequence Function Chart (SFC) som utgör en styrning av vertikalförflyttningen enligt Figur 2.13. Närmare presentation av SFC-programmering kommer i senare avsnitt men en titt i programdokumentationen gör att programmets funktion kan anas med kunskap kring funktionsdiagram som bakgrund.



Figur 2.15: Dokumentation av PLC-program skrivet i SFC-språk.

Kap 3. Programmerbara styrsystem - uppbyggnad.

Fram till 1970-talet var den förhärskande tekniken att åstadkomma styrning av processer att använda reläteknik. Signalledningar från givare ute i processen kopplades in till reläer i ett skåp där logiken byggdes upp genom kopplingar mellan reläernas slavbrytare och resultatet skickades via styrsignalledningar ut för att aktivera olika don såsom kontaktorer för manövrering av elmotorer eller magnetventiler för cylindermanövrering. Anläggningarna var i princip uppbyggda enligt Figur 3.1 (vänster) nedan men en större anläggning kunde innefatta hundratals givare- och styrsignaler och tusentals reläer som fyllde hela rum med reläskåp.

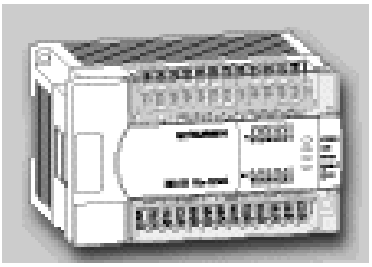


Figur 3.1: Jämförelse reläsystem(vänster) och PLC-styrning (höger).

Så kom transistorn, datorn och mikrokontrollern och därmed andra möjligheter att bygga logik än via reläer. I mitten av 1970-talet dök de första PLC-systemen upp på markanden vilket innebar att innehållet i styrskåpen byttes ut men signalledningar ute i processen var de samma, se Figur 3.1. Nu med ytterligare drygt 30 års utveckling bakom sig finns mycket kraftfulla PLC:er som kan uträtta mycket mer än de gamla reläsystemen och ingå i de omfattande informationssystem som krävs för dagens automationsnivåer.

Graden av automatisering av de industriella processerna går hand i hand med tillgängligheten till styrutrustning som är flexibel, lätthanterlig och billig. Utvecklingen av mikrodatorn öppnade vägen för det programmerbara styrsystemet, PLC, som är ett datorbaserat styrsystem anpassat till styrning av maskiner och processer i industriell miljö. Från att från början varit utrustning som tog över den logiska styrning som tidigare utfördes av reläsystem expanderade arbetsuppgifterna till att ta hand om uppgifter som tidigare utfördes av separata instrument såsom PID-reglering av återkopplade system och recepthantering vid batch-processer. Idag finns allt från mindre PLC-system för logisk styrning av enskilda maskiner till komplexa kraftfulla PLC:er som utöver sin styruppgift är en spelare i fabriksnätverk av andra PLC:er, operatörsstationer, underhållssystem och affärssystem.

Ett programmerbart styrsystem skiljer sig från ett vanligt datorsystem genom att programmeringsspråken är anpassade till användningsområdet för att uppnå hög produktivitet vid programutveckling. Strävan har varit att utveckla ett förhållande människa/utrustning baserat på användarens tekniska erfarenhetsvärld där användaren av styrsystem primärt skall vara processkunnig och sekundärt datorspecialist. Utvecklingen har dock lett till att PLC-systemen mer och mer integreras i datorsystem med kommunikation både med intelligenta givare och don ute i processen och med överordnade datorer och databaser för produktionsplanering, underhållsplanering m m. Som automationsingenjör krävs idag kunskaper kring industriella processer såväl som inom datakommunikation och datorsystem.



Figur 3.2: Två typer av PLC-system. Ett kompakt och ett moduluppbyggt expanderbart.

Utmärkande för ett programmerbart styrsystem är också att elektronik och chassi är dimensionerat att tåla den mekaniska miljö (vibrationer, stötar), kemiska miljö (gaser, fukt) och elektriska miljö (elektriska och magnetiska fält) som ofta är betydligt besvärligare på industrigolvet än i de normala datorsystemens kontorsmiljö. Vidare är kraven att installation och även utbyggnad av systemen skall vara enkel, att systemet skall kunna kommunicera med andra PLC-system, med operatörssystem för människa-maskin-kommunikation, med decentraliserade I/O-enheter mm.

PLC är en förkortning av Programmable Logic Controller och är i dag den gängse benämningen. En kort period i början av 1980-talet var den använda benämningen PC-system, Programmable Controller, men den fick ge vika då samma akronym, PC, i betydelse Personal Computer tog över och ju blev ett välkänt begrepp för den stora allmänheten. Även beteckningen PBS (Programmerbara Binära System) har förekommit.

På svenska marknaden finns idag flera tiotals system av olika modell från ett 20-tal olika leverantörer. Som större tillverkare kan nämnas ABB (Sverige), Siemens, Beckhoff (Tyskland), Mitsubishi, Hitachi (Japan), Rockwell (Frankrike), Allen Bradley, Honeywell (USA). Ett flertal tillverkare av styrutrustning marknadsför också system från redan nämnda tillverkare under eget namn.

Storleken på systemen varierar mycket. Små mycket lätthanterliga system med ett tiotal digitala in- och utgångar är så prisbilliga (2-5000 kr) att de i allt högre grad utnyttjas på områden som tidigare var förbehållna mindre reläsystem. De större systemen kan hantera flera tusen in-och utgångar, både digital och analoga, vilket gör att de kan användas för att lösa mer omfattande styrproblem. Dessa kan också klara av aritmetik, integration med operatörssystem med hantering av dynamiska processbilder samt kommunikation med över- och underordnade system. Trenden går numera åt mer och mer decentraliserade system med nätverk av ett antal PLC:er, kommunicerande via seriell fältbuss med distribuerade sensorer och aktuatorer som hanterar var sin del i en större process.

Programmerbara styrsystem har ingen längre historia bakom sig. Det första PLC-systemet utvecklades i slutet av 60-talet för användning inom bilindustrin. Under 70-talets andra hälft kom de små mikrodatorbaserade PLC-systemen och även de hierarkiska näten av PLC i kombination med processdatorer för styrning av hela fabriker. På 80-talet kom de små prisbilliga systemen med några få in- och utgångar som konkurrerar med reläsystem med endast ett tiotal reläer.

Det är inte bara process- och tillverkningsindustrin som använder PLC-styrningar utan PLC har också idag en stor marknad vad gäller styrning av fastighetsinstallationer och infrastrukturanläggningar som t ex trafikstyrning och -övervakning, ventilation, belysning mm i tunnlar.

De mindre systemen används ofta för maskinstyrningar. De placeras nära den maskin de skall styra vilket gör att man får ett väl avgränsat system som är lätt att programmera och lätt att installera och felsöka i.

Exempel på sådana styrobjekt är:

- Transportsystem.
- Förpackningsmaskiner.
- Pumpanläggningar
- Ventilationssystem

De större PLC-systemen används för att styra och övervaka hela processer. I dessa system ingår ofta även hantering av analoga storheter och i systemet inbyggda regulatorer för återkopplad reglering av analoga storheter. Möjlighet för operatör att följa processen på bildskärm är ofta möjlig i dessa system liksom att via tangentbord eller operatörspanel kunna kommunicera med processen. En annan vanligt förekommande möjlighet är kommunikation med centraldator för produktionsstatistik, produktionsplanering och rapportering.

Exempel på användning av "större" system är:

- Kraftverk, kraftvärmeverk.
- Kemiska processer t.ex raffinaderi.
- Massatillverkning och pappersmaskiner.
- Tillverknings- och monteringsliner för bilar, kylskåp m m.
- Styrning ventilation, belysning, trafik mm i vägtunnlar.
- Belysning, värme, ventilation, låssystem i fastigheter.

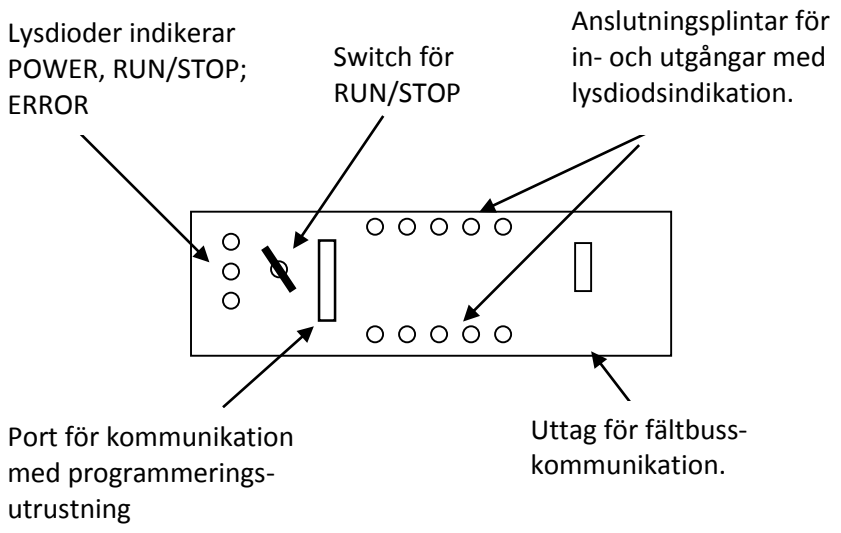
I vilka styrsammanhang är det då olämpligt att använda PLC-system? En viktig egenskap hos PLC-system är att de är flexibla och därför lätt anpassningsbara till olika applikationer. Maskiner som skall mångfaldigas i stora mängder t ex kopieringsmaskiner, bakmaskiner, tvättmaskiner o dyl. är ju inte i behov av ett styrsystem som är flexibelt och därmed dyrare

än alternativ där man inte lagt ner kostnader för att få flexibilitet. I sådana applikationer är därför skräddarsydda, mikrokontrollerbaserade styrenheter i stor upplaga en betydligt billigare lösning.

3.1. PLC-systemet – hårdvarans uppbyggnad.

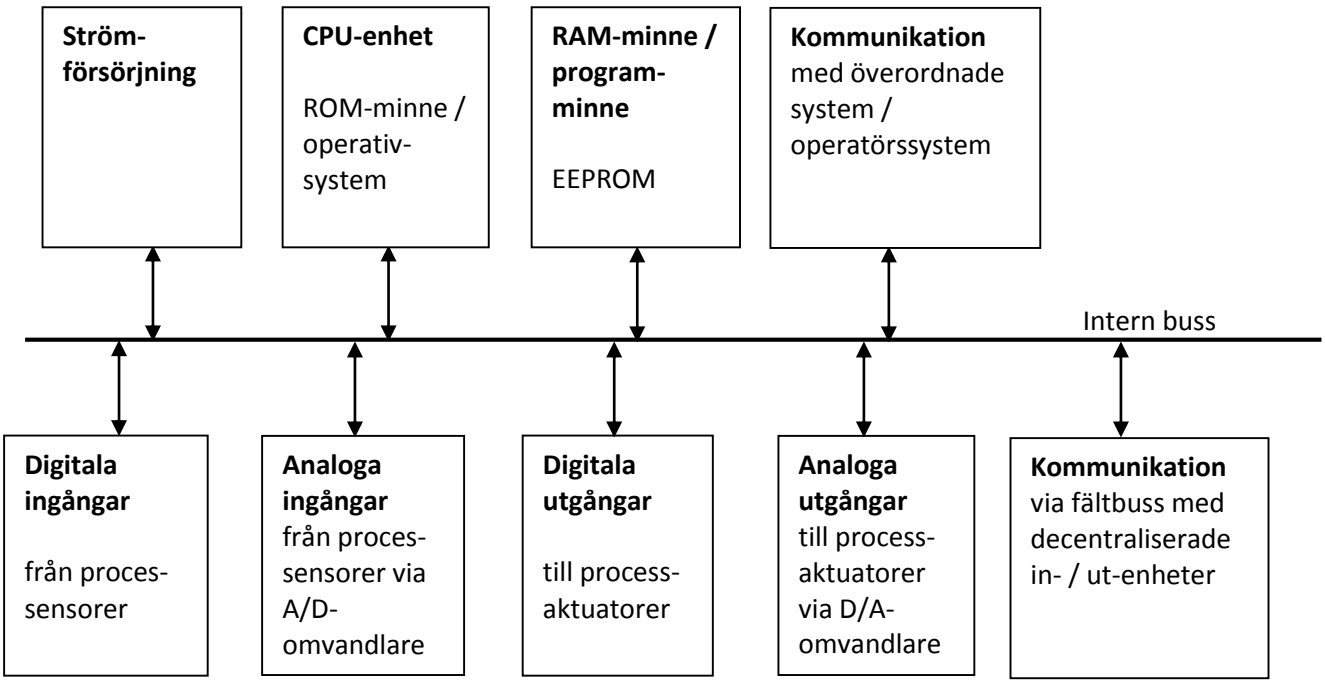
PLC-system är gjorda för montage i elskåp i anslutning till den process de är satta att styra. Ofta är de anpassande för upphängning på DIN-skena som är en standardskena för upphängning av olika typer av utrustning i elskåp. Av Figur 3.3 framgår vad som normalt finns i form av anslutningar, indikeringar m m för kommunikation med yttervärlden. Fel vid styrprogramexekveringen indikeras normalt med en lysdiod men inkoppling av programutvecklingsverktyget är sedan nödvändigt för att läsa av felkoder. Lysdioder för indikering av signalstatus på in- och utgångar är mycket bra hjälpmedel vid felsökning. Om fel uppstår kan man med hjälp av dessa indikeringar lokalisera felet till antingen någon yttre koppling eller till styrprogrammet. Genom att koppla in programutvecklingsverktyget som normalt är en PC-baserad programvara kan monitorering av programmet göras där aktuell signalstatus kan avläsas vilket ger ytterligare möjligheter till fördjupad felsökning.

Enda manövreringsmöjligheten är normalt start/stopp av programexekvering samt reset vilket innebär återgång till programstart och eventuellt nollställning av valda minnen / register. Nollställning av alla minnen /register görs med en högre nivå av reset för att minimera risken att av misstag tömma all i PLC lagrad driftinformation.



Figur 3.3: Schematisk bild över PLC:ets yttre.

I Figur 3.4 nedan beskrivs i blockschemaform funktionella uppbyggnaden av ett PLC-system. Som synes så skiljer det sig inte direkt ifrån ett blockschema över vilket dator som helst.



Figur 3.4: Blockschema för PLC-system.

3.1.1. Strömförsörjningsenhet.

Strömförsörjningsenheten försörjer all intern elektronik med lämplig spänning, normalt 5 V. Den utnyttjas ibland också till att leverera kraft för försörjning av givarenheter m.m som ansluts till styrsystemets ingångar. Däremot kräver oftast aktuatorer såsom kontaktorer och magnetventiler kopplade till utgångarna så hög effekt att en extern kraftkälla används för detta.

3.1.2. Centralenheten.

CPU:n (Central Processing Unit) är själva processorenheten som styr verksamheten i PLC-systemet med hjälp av operativsystemet och av styrprogrammet. Det organiserar alltså flödet av data via en parallell kommunikationsbuss till och från de olika anslutna enheterna, utför logiska och aritmetiska operationer och administrerar minnet. Figur 3.5 visar specifikationen för några PLC-centralenheter.

ROM (Read Only Memory) innehåller operativsystemet som behövs för att initiera systemet. Dessutom innehåller det översättaren som översätter de PLC-instruktioner som skrivs in till systemet till för CPU:t begripliga styrkoder. Operativsystemet är inplanterat vid leverans och kan endast uppdateras via tillverkarens försorg.

RAM (Random Access Memory) används för att lagra de instruktioner som skrivs in till systemet från programmeringsenheten. RAM-minnet är flyktigt vilket innebär att det tappar sitt innehåll vid spänningsbortfall. Ofta är därför detta minne försett med batteriupp-backning (backup).

Parallellt med RAM-minnet finns ofta möjlighet att bränna in samma instruktioner som finns i RAM-minnet i ett EPROM (Eraseable Programmable Read Only Memory). Detta minne behåller sin information tills man raderar den med UV-ljus eller på elektrisk väg (EEPROM). EPROM:et är ofta monterat i en kassett som kan pluggas in i PLC-systemet vilket gör att man kan ha en fungerande programvara lagrad i en sådan kassett. Detta kortar ner ställtiden när man t.ex vill ändra produktionen från en detalj till en annan som kräver andra styrprogram för de i produktionen inblandade PLC-systemen. Har man en gång gjort programmen är det bara att byta EEPROM-kassett. Idag är ofta PLC-systemet nätverks-anslutet och programvara kan enkelt underhållas eller bytas ut genom nedladdning via nätet. Normalt är det ett internt nätverk inom företaget med spärrad kontakt mot Internet för att förhindra intrång från obehöriga.

Table 2.3 Q02(H)CPU, Q06HCPU, Q12HCPU, Q25HCPU Performance Specifications

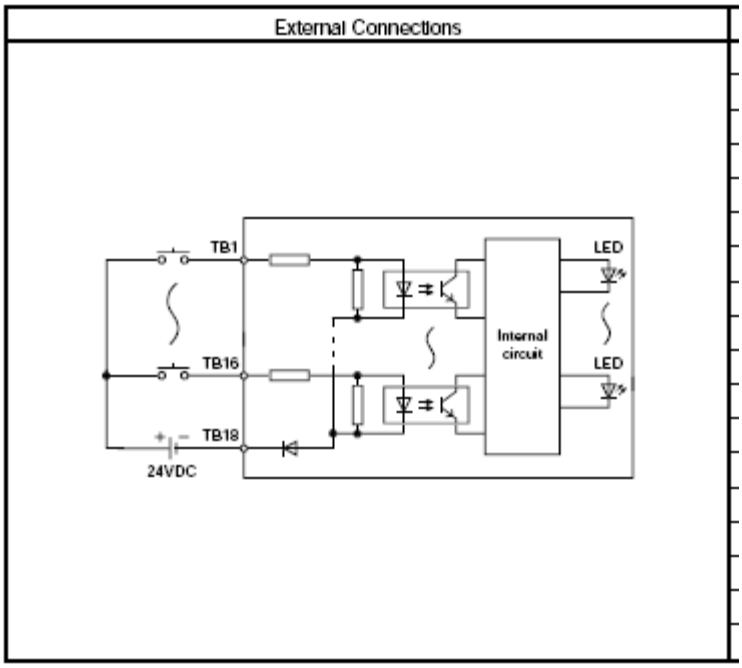
Item			Type					Remarks
			Q02 CPU	Q02H CPU	Q06H CPU	Q12H CPU	Q25H CPU	
Control method			Repeated operation using stored program					——
I/O control mode			Refresh mode					Direct I/O is available by direct I/O specification (DX□, DY□)
Programming language (Language dedicated to sequence control			Relay symbol language, logic symbolic language, MELSAP3 (SFC), MELSAP-L, Function block and structured text					——
Processing speed (sequence instruction)		LD X0	0.079μs	0.034μs				——
		MOV D0 D1	0.237μs	0.102μs				——
Constant scan (Function that uniforms scan time)			0.5 to 2000 ms (can be specified in 0.5ms increments)					Parameter setting
Program capacity *2	Program memory (Drive 0)		28k step		60k step	124k step	252k step	——
Memory capacity	Memory card (RAM) (Drive 1)		Capacity of the memory card (max. 2Mbyte)					——
	Memory card (ROM) (Drive 2)		Capacity of the memory card loaded (Flash card: max. 4Mbyte, ATA card: max. 32Mbyte)					——
	Standard RAM (Drive 3)		64kbyte	128kbyte *5		256kbyte *3		——
	Standard ROM (Drive 4)		112 kbyte		240 kbyte	496 kbyte	1008 kbyte	——
	CPU shared memory*4		8kbyte					——
Max. number of files stored	Program memory		28		60	124	252 *1	——
	Memory card (RAM)		256					——
	Memory card (ROM)	Flash card	288					——
		ATA card	512					——
	Standard RAM		2					Only one file each for file register and local device
	Standard ROM		28		60	124	252	——
Number of standard ROM write times			Max. 100 thousand times					——

Figur 3.5: Teknisk specifikation för några av Mitsubishi-systems CPU.

3.1.3. Digitala ingångsenheter.

Ingångsenheterna för digitala signaler anpassar kommunikationssignalerna mellan PLC-systemets buss och processens givare. PLC-systemet har interna spänningsnivån 5 V och mycket låg effektnivå medan signaler från processen har betydligt högre spänningsnivå (ofta 24 V DC eller 230 V AC) och effektnivå. De högre nivåerna hos processignalerna beror på att de annars skulle störas i den ofta dåliga elektriska miljö de skall fungera inom. I

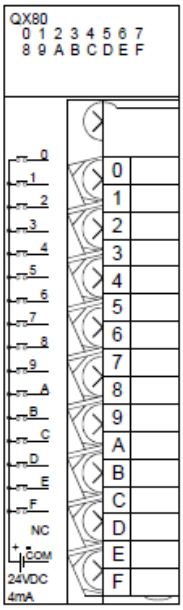
processen finns t.ex. stora elektriska motorer med matarkablage som alstrar starka magnetiska fält. Dessa fält kan i sin tur inducera spänningar i annat kablage som ingår i samma installation t.ex signalkablar från givare ute i processen. Skulle spännings- och effektnivån i dessa signalvägar vara på samma nivå som datorsystemets interna signalnivåer skulle signalerna lätt kunna störas och signalöverföringen inte vara tillförlitlig. Även med högre spänningsnivåer kommer stora transienta störningar att induceras i signalledningarna varför in- och utenheternas uppgift också är att filtrera bort transienter så att de inte når den interna bussen. Figur 3.6 beskriver spänningsdelning av 24 V DC insignaler och filtrering via optokopplare som skiljer processignalerna galvaniskt från interna bussens signaler.



Figur 3.6: Signalanpassning digital ingångar, PLC (Mitsubishi manual).

Av den tekniska specifikationen för en digital ingångsenhet, Figur 3.7, framgår bl a att enheten har 16 ingångar. För 24 V ingången gäller att inspänning över 19 V uppfattas som en logisk etta medan inspänning under 11 V uppfattas som logisk nolla. Mellan 11 och 19 V är då logiska nivån obestämd. Ingångarnas omslagstider är inställbara men grund-inställningen är 10 ms. Korta omslagstider innebär att korta transienta störningar kan slå igenom medan långa omslagstider ger bortfiltrering av störningar men nackdelen att signalomslag uppfattas efter en längre tid. Tiden 10 ms motsvarar ungefär program-cykeltiden hos ett normalprogram och därmed slöar den inte ner hela styrsystemets svarstider men har ändå en viss filterverkan.

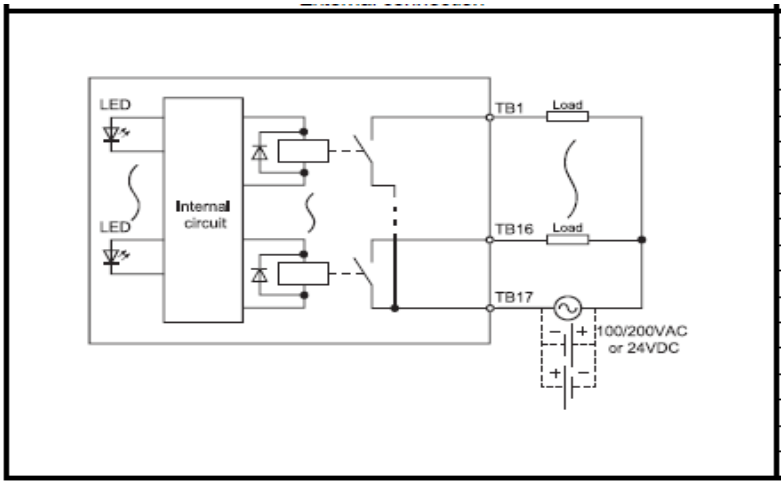
I sammanhanget skall också nämnas att det finns moduler med interuptingångar och snabbräknaringångar. Interuptingångar används då en ingångssignal initierar ett snabbt händelseförlopp som PLC-programmets normala cykeltid inte skulle klara att observera och styra i den takt som krävs. Att ingången genererar ett interrupt innebär att normala styr-programmet avbryts tillfälligt och en kortare och därmed snabbare programrutin startar som behandlar det snabba förloppet. Höghastighetsräknaringången klarar att räkna pulser från en pulsgivare med en frekvens som är högre än normala ingångarnas snabbaste omslagstid (se nästa avsnitt om PLC-systemets arbetssätt).

Specifications \ Type		DC input module (Negative common type) QX80	Appearance
Number of input points		16 points	
Isolation method		Photocoupler	
Rated input voltage		24VDC (+20/-15%, ripple ratio within 5%)	
Rated input current		Approx. 4mA	
Input derating		No	
ON voltage/ON current		19V or higher/3mA or higher	
OFF voltage/OFF current		11V or lower/1.7mA or lower	
Input impedance		Approx. 5.6kΩ	
Response time	OFF to ON	1ms/5ms/10ms/20ms/70ms or less (configured in PLC parameter) *1 (Default: 10ms)	
	ON to OFF	1ms/5ms/10ms/20ms/70ms or less (configured in PLC parameter) *1 (Default: 10ms)	
Dielectric withstand voltage		560VAC rms/3 cycles (altitude 2000m (6557.38ft.))	
Insulation resistance		10MΩ or more by insulation resistance tester	
Noise immunity		By noise simulator of 500Vp-p noise voltage, 1 μs noise width and 25 to 60Hz noise frequency	
		First transient noise IEC61000-4-4: 1kV	
Protection degree		IP2X	
Common terminal arrangement		16 points/common (common terminal: TB18)	
Number of occupied I/O points		16 points (I/O assignment is set as a 16-point input module.)	
Operation indicator		ON indication (LED)	
External connections		18-point terminal block (M3×6 screws)	
Applicable wire size		0.3 to 0.75mm^2 core (2.8mm (0.11in.) OD max.)	
Applicable crimping terminal		R1.25-3 (sleeved crimping terminals cannot be used.)	
Internal current consumption (5VDC)		50mA (TYP. all points ON)	
Weight		0.16kg	

Figur 3.7: Specifikation digital ingångsmodul för PLC (Mitsubishi manual).

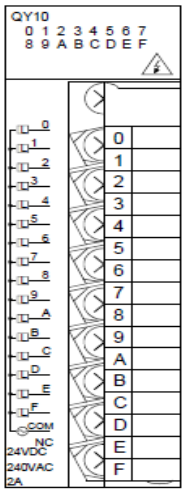
3.1.4. Digitala utgångsenheter.

Digitala utgångsmoduler matas med yttre spänning som normalt är antingen 24 V DC eller 230 V AC. I Figur 3.8 matas utgångarna via reläer och därmed är både DC- och AC-matningar möjliga. Parallellt med reläspolen ligger en diod, en s k frihjulsdiod, för att släcka den transient som annars skapas då strömmen genom spolen bryts. Av specifikationen, Figur 3.9, framgår bl a att varje utgång får belastas med maximala strömmen 2 A, att omslagstiden hos utgången ligger kring 10 ms och omslagstakten är maximerad till 1 omslag/sekund. I Figur 3.10 visas två andra typer av utgångar, transistor resp. TRIAC. Transistorn manövrerar DC-matningar medan TRIAC hanterar AC. Tittar man i specifikationen för dessa utgångsmoduler finner man att omslagstiderna är 10 gånger snabbare än reläutgångarna medan de tål betydligt lägre strömmar, 0,1A för transistor och 0,6 A för TRIAC.

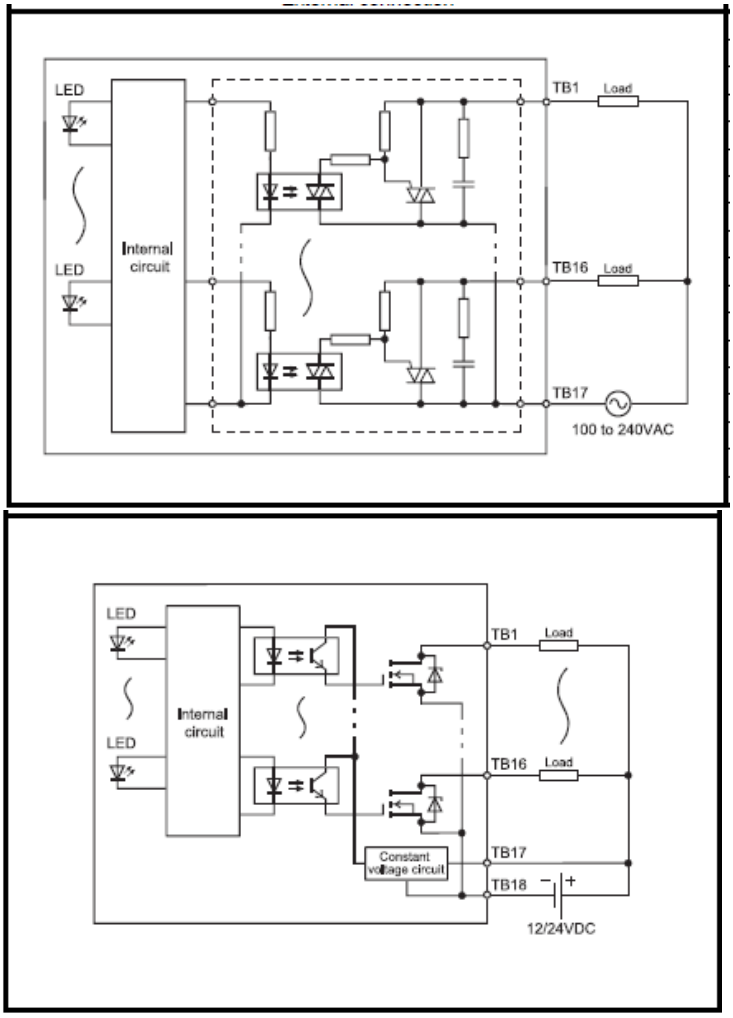


Figur 3.8: Signalanpassning digitala reläutgångar, PLC (Mitsubishi manual).

Specifications \ Type		Contact output module QY10	Appearance
Number of output points		16 points	
Isolation method		Relay	
Rated switching voltage, current		24VDC 2A (resistive load) 240VAC 2A (cos ϕ =1) /point, 8A/common	
Minimum switching load		5VDC 1mA	
Maximum switching load		264VAC 125VDC	
Response time	OFF to ON	10ms or less	
	ON to OFF	12ms or less	
Life	Mechanical	20 million times or more	
	Electrical	Rated switching voltage/current load More than 100 thousand times or more	
		200VAC 1.5A, 240VAC 1A (COS ϕ =0.7) 100 thousand times or more 200VAC 0.4A, 240VAC 0.3A (COS ϕ =0.7) 300 thousand times or more	
		200VAC 1A, 240VAC 0.5A (COS ϕ =0.35) 100 thousand times or more 200VAC 0.3A, 240VAC 0.15A (COS ϕ =0.35) 300 thousand times or more	
		24VDC 1A, 100VDC 0.1A (L/R=7ms) 100 thousand times or more 24VDC 0.3A, 100VDC 0.03A (L/R=7ms) 300 thousand times or more	
Maximum switching frequency		3600 times/hour	
Surge suppressor		No	
Fuse		No	
Dielectric withstand voltage		2830VAC rms/3 cycles (altitude 2000m (6557.38ft.))	
Insulation resistance		10MΩ or more by insulation resistance tester	
Noise immunity		By noise simulator of 1500Vp-p noise voltage, 1 μs noise width and 25 to 60Hz noise frequency	
		First transient noise IEC61000-4-4: 1kV	
Protection degree		IP1X	
Common terminal arrangement		16 points/common (common terminal: TB17)	
Number of occupied I/O points		16 points (I/O assignment is set as a 16-point output module.)	
Operation indicator		ON indication (LED)	
External connections		18-point terminal block (M3×6 screws)	
Applicable wire size		0.3 to 0.75mm^2 core (2.8mm (0.11in.) OD max.)	
Applicable crimping terminal		R1.25-3 (sleeved crimping terminals cannot be used.)	
Internal current consumption (5VDC)		430mA (TYP. all points ON)	
Weight		0.22kg	



Figur 3.9: Specifikation digital reläutgångsmodul för PLC (Mitsubishi manual).



Figur 3.10: Transistor- och TRIAC- utgångar, PLC (Mitsubishi manual).

3.1.5. Analoga in- och utgångsenheter.

Ingångsenheter för analoga signaler innehåller A/D-omvandlare (ADC) och levererar på kommando från centralenheten analoga signalvärden, ofta 0 – 10 V eller 4 – 20 mA, på digitaliserad form med en upplösning på 8 till 14 bitar. Intervallen hos analoga ingången är normalt inställbara. Antag t ex att analoga området 0-10 V omvandlas till digitala området 0 – 8000. Då skulle en insignal på 2,34 V ge ett digitalt värde på 2,34/10·8000 = 1872 att hantera i styrprogrammet. A/D-omvandlaren är konfigurerbar också vad gäller omvandlingstider och när omvandling skall ske men normalt sker kontinuerlig omvandling där det omvandlade värdet läggs i ett minnesregister som sedan kan läsas av PLC-programmet.

Utgångsenheter för analoga signaler innehåller D/A-omvandlare (DAC) och levererar utifrån ett digitalt värde på 8 till 14 bitar analoga signalvärden, ofta 0 – 10 V eller 4-20 mA till utgången. Intervallen hos analoga utgången är normalt inställbara. Antag t ex att digitala området 0 – 8000 omvandlas till analoga området 4-20 mA, som är industristandard för analoga processignaler. Då skulle ett digitalt värde 2000 ge ett värde på 8 mA på den analoga utgången. D/A-omvandlaren är konfigurerbar också vad gäller omvandlingstider

och när omvandling skall ske men normalt sker kontinuerlig omvandling där det värde som skall omvandlas läggs i ett minnesregister. Värdet i minnesregistret omvandlas kontinuerligt och läggs ut på utgången.

3.1.6. Kommunikationsmoduler.

Kommunikationsmoduler möjliggör seriell kommunikation oftast via Ethernet, RS232, RS485, fabrikatsspecifika fältbussar (slutna protokoll) eller via olika typer av generella fältbussar (öppna protokoll) som t ex Interbus, Profibus, Modbus, CANopen. Kommunikation kan ske med yttre enheter såsom:

- Överordnat styrsystem eller annat PLC-system i samma nätverk.
- Operatörspanel eller operatörssystem (SCADA-system) för människa-maskin-kommunikation.
- Decentraliserad in/ut-enhet (dec I/O) vilka inte är anslutna direkt till PLC-enheten utan placerade som noder ute i processen varifrån centralenheten kan läsa in signal-tillstånden från/till flera in-/utgångar seriellt d v s på en enda ledare istället för en ledare per in-/utsignal.
- Överföring av information från / till fabrikens affärssystem för produktionsplanering och underhållsplanering.
- Intelligenta givare som levererar mätdata på seriell form.
- Databas för t ex lagring av driftsdata eller hämtning av recept.

Det förekommer alltså en mängd kommunikationssätt, fältbussar, utgående från lika många kommunikationsprotokoll – beskrivningar av hur kommunikationen skall gå till. Vilken typ som används är beroende av bl a typ av data som skall överföras, hur snabbt det skall ske, hur säkert det skall ske samt på tradition inom branschen. Ethernet, det kommunikationssätt som används på Internet, är dominerande framför allt när det gäller administrativ dataöverföring. Fältbussar används där kraven på snabbhet och dataframkomlighet är viktig. Tendensen är dock att Ethernetvarianter på sikt tar över där tidigare en uppsjö av olika fältbussar dominerat när det gäller snabb överföring av processdata mellan decentraliserade enheter, operatörssystem och centralenhet.

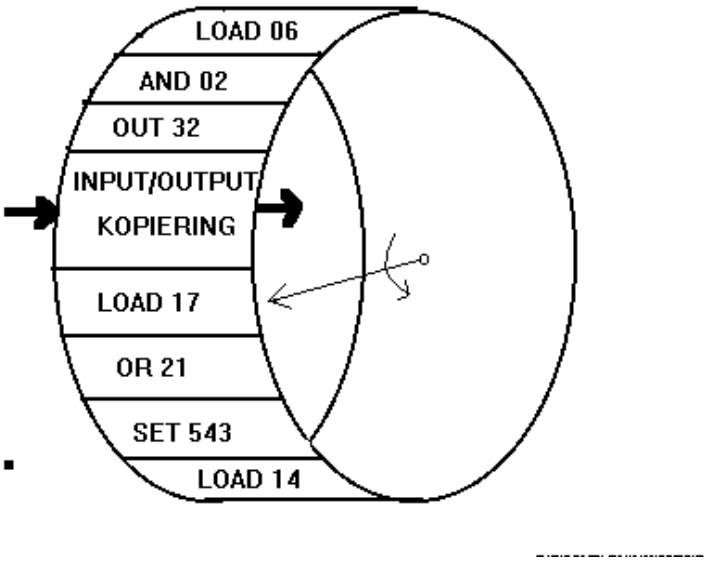
Konfiguration av kommunikationsmoduler skiljer sig åt mellan olika fabrikat och olika kommunikationsprotokoll. Viss konfiguration kan vara integrerad i den PLC-program-meringsmiljö som används men ofta gäller att speciell konfigurationsprogramvara är nödvändig.

3.2. PLC-systemets arbetssätt - mjukvaran.

En viktig egenskap hos PLC-system är att de snabbt skall kunna läsa in och behandla data som behövs för att styra ett eller flera förlopp. Till de mer tidsödande operationerna i ett datorsystem är in- och utmatning av signaler via in/ut-enheterna.

För att få så snabbt arbetssätt som möjligt tillämpar man en teknik som kallas input-output-kopiering. Vid början av en programcykel kopieras alla insignalerna från in-enheterna till ett internt minne. Därefter bearbetas instruktion för instruktion i styrprogrammet och de resulterande utsignalerna lagras efter hand också i ett internt utgångsminne. Då alla instruktioner genomlöpts kopieras det interna utgångsminnet till utenheterna som verkställer styrsignalerna ut mot processen. Därmed är en programcykel (scan-cykel) genomförd och en ny börjar direkt med att på nytt läsa in-enheterna till interna ingångsminnet o s v. Arbetssättet illustreras i Figur 3.11. Normalt finns också möjlighet till villkorliga hopp förbi ett antal programrader i programmet. Hoppen påverkar dock inte cykeltiden.

Styrsystemets cykeltid är tiden från det att läsning av in-enheter börjar tills utenheterna aktiverats. Cykeltiden beror naturligtvis av processorns arbetssätt och klockfrekvens samt av hur långt styrprogrammet är men ligger normalt på någon eller några tiotal millisekunder. Detta är oftast en tillräcklig snabbhet. Vid pulsräkning från en pulsgivare för t ex exakt positionsbestämning kan dock snabbheten vara för dålig varför man får tillgripa in-enheter i form av en snabbräknarenhet som räknar pulser vid mycket högre frekvens. Styrsystemet kan från en sådan enhet via bussen läsa av om önskad position har uppnåtts eller passerats. Alternativt levererar räknarenheten vid uppnått angivet räknarvärde ett interupt som avbryter den löpande programexekveringen och utför en önskad interuptrutin som verkställer vad som skall hända efter uppnått räknarvärde varefter program-exekveringen återvänder dit där avbrottet skedde. Ett extra utgångskort kan också användas som aktiveras direkt av snabbräknarenheten för att slippa tidsfördröjningen man ändå får genom att blanda in centralenheten.



Figur 3.11: Styrsystemets cykliska arbetssätt.

Arbetssättet med ingångs-utgångs-kopiering innebär att man får ta hänsyn till detta vid programmeringen. En utgång får bara användas en enda gång i programmet. Om t ex en varningslampa skall tändas antingen för villkor 1 eller för villkor 2. i programkoden och man skriver först att villkor 1 skall aktivera utgången för lampan och lite senare i

programmet att villkor 2 skall aktivera samma utgång så är det endast det senare villkor 2 som kommer att gälla. Resultatet av villkor 2 kommer alltid att skriva över resultatet från villkor 1 i det interna utgångsminnet. Resultatet av villkor 1 når alltså aldrig utgångarna. I stället måste man skriva programmet så att om villkor 1 ”eller” villkor 2 är uppfyllt så aktivera utgång för varningslampa. Utgången används då bara en gång. Det är alltså väsentligt att man tar hänsyn till arbetssättet med ingångs-utgångs-kopiering vid programmeringen annars uppstår lätt logiska fel.

Det språk som används för programmering av PLC-system skall vara enkelt, kraftfullt och anpassat för typen av styrning. Tidigare var inte alla tillverkare eniga om hur ett sådant språk ser ut utan en flora av olika språkvarianter finns, som dock inte vid närmare beskådande skiljer sig så mycket från varandra. På 1990-talet arbetades en standard fram för programmering av PLC-system (IEC 61131-3) vilken de flesta leverantörer nu anpassat sig till vilket gör att man i framtiden kommer att känna igen sig i programutvecklings-miljöerna oavsett vilket fabrikat av PLC-system man arbetar med.

Fem olika sätt att koda instruktionsprogrammen, programspråk, är specificerade i standarden och är språktyper som förekommit tidigare men nu fått en enhetlig form. En del tillverkare tillhandahåller alla sätten medan andra har begränsat sig till något eller några.

De fem olika programmeringssätten bygger i grunden alla på logiska (Booleska) instruktioner där tillståndskombinationer av ingångsvärden och tidigare mellanlagrade tillstånd resulterar i motsvarande utgångstillstånd. Logiska grundinstruktioner som AND, OR, ANDNOT och ORNOT utökas med SET och RESET av vippor (minnen) samt av räknar- och tidsfördröjningsinstruktioner. Ytterligare instruktioner finns i större eller mindre omfattning beroende på styrsystemets komplexitetsgrad. De fem språken är Ladderprogrammering, Funktionsblockprogrammering, Instruktionslista, Funktionsdiagram samt Strukturerad text. Här följer presentation av olika instruktioner och efter hand presenteras de olika programmeringssätten enligt standard IEC 61131-3. Programmeringsmiljön som de följande exemplen är programmerade i är GX IEC Developer som är en IEC 61131-3 baserad programutvecklingsmiljö och endast instruktioner enligt standarden används varför det som presenteras i detta avsnitt gäller allmänt för alla utvecklingsmiljöer som stödjer standarden. Men först en presentation av de olika signaler som skall hanteras i PLC-systemet.

3.2.1. PLC-systemets signaluppsättning och beteckningsstandard.

Kommunikation mellan PLC-system och till processen kopplade givare och don av on/off-typ sker via digitala in- och utgångar. Via ingångar %IX mottar systemet signaler från externa switchande givare eller kontakter. De förekommer både som slutande (NO =normally open) och brytande kontakter (NC =normally closed) beroende bl.a på säkerhetsaspekter. Beteckning %IX är IEC-standardbeteckning på ingångar.

Med utgångarna %QX överför PLC-systemet styrsignaler till styr- eller indikeringsdon som t.ex kontaktorer, magnetventiler eller indikeringslampor. Utgångsstatusen kan också användas internt i PLC-programmet på samma sätt som en ingång eller en minnescell. Det finns inga begränsningar på hur många gånger en in- eller utgång får användas som villkor i ett program. Beteckning %QX är IEC-standardbeteckning på utgångar.

Adressering av ingångskanaler görs med beteckning %IXn där n är en löpande numrering av ingångarna. Om det gäller ett moduluppbyggt PLC där ett CPU placeras på ett bakplan

och sedan kompletteras med de funktionsmoduler som behövs så sker numreringen utifrån de adressplatser som föregående moduler upptagit. Om första modulen är en ingångsmodul med 16 ingångar adresseras dessa med %IX0 - %IX15. Är nästa modul en utgångsmodul med 16 utgångar adresseras dessa %QX16 - %QX31. Är sedan nästa en ingångsmodul med 32 ingångar adresseras dessa %IX32 - %IX63 osv. Två placeringsexempel finns beskrivna i Figur 3.12 nedan. I exemplen som följer används systemsammansättning enligt den övre delen i Figur 3.12 d v s ingångar %IX0-%IXF (obs att vid modulplats 0 skrivs inte nollan ut) och utgångar %QX10 - %QX1F.

Nätdel	CPU	Digital ingångs-modul 16 kanal Adresser: %IX0-%IX15	Digital utgångs-modul 16 kanal Adresser: %QX16 -%QX31	... osv

Nätdel	CPU	Digital ingångs-Modul 16 kanal Adresser: %IX0-%IX15	Digital ingångs-Modul 16 kanal Adresser: %IX16 -%IX31	Digital utgångs-modul 32 kanal Adresser: %QX32 -%QX63	.... osv

Figur 3.12: In- och utgångsadresser i två olika systemsammansättningar.

Internt finns ett stort antal enbits minnesflaggor som betecknas %MX0.n där n är ett löpnummer. Det finns också enbits latchade minnesflaggor betecknade %MX8.n. Det som skiljer är att %MX0-flaggorna nollställs vid ”normal” reset av CPU eller spänningsbortfall medan %MX8 behåller sitt tillstånd och en speciell svåråtkomlig latch-resetkrävs för att nollställa dessa.

Det finns också ett antal enbits specialminnesflaggor vars beteende är förutbestämt. De ofta använda presenteras i Figur 3.13 nedan och utgörs dels av klocksignaler (pulståg) av olika frekvens och dels av en flagga som är ettställd endast första exekveringscykeln och därför bra att använda vid initieringar.

Beskrivning	Adress
TRUE första scancykel efter RUN	%MX10.402
10 Hz klockpulståg	%MX10.410
5 Hz klockpulståg	%MX10.411
1 Hz klockpulståg	%MX10.412
0,5 Hz klockpulståg	%MX10.413

Figur 3.13: Några specialminnesflaggor och dess adresser.

Alla hittills nämnda variabler är enbits och därmed av datatypen BOOL. Internt finns också ett stort antal 16-bitars minnesregister som betecknas %MW0.n. De är i de följande exemplen decimalt numrerade. Det finns oftast något eller några tusental benämnda %MW0.0, %MW0.1,….%MW0.456, %MW0.457…… - %MW0.nn. Dessa register är av datatypen INT eller WORD. Man kan också adressera dessa 16-bitars register bitvis. Vill man t ex adressera bit nr 7 i register %MW.345 används adressen %MX0.345.7 vilken då är av typen BOOL.

Här följer en tabell över grundläggande variabeltyper:

Variabel	IEC-adress	Datatyp	Värde	Beskrivning
Digitala ingång	%IXn	BOOL	TRUE FALSE	Digitala ingångar, n=decimalt löpnummer
Digital utgång	%QXn	BOOL	TRUE FALSE	Digitala utgångar, n=decimalt löpnummer
Minnesflagga	%MX0.n	BOOL	TRUE FALSE	Interna enbits minnesflaggor, nollställs vid CPU-reset
Minnesflagga, batteriuppback	%MX8.n	BOOL	TRUE FALSE	Interna enbits minnesflaggor, behåller sitt tillstånd vid enkel CPU-reset
Specialminne	%MX10.n	BOOL	TRUE FALSE	Se tabell Figur 3.13
16-bits register	%MW0.n	INT WORD	-32768 …. +32767 0 …. 65535	16-bitars minnesregister
32-bits register	%MW0.n	DINT DWORD		32-bitars minnesregister, tar upp adress n och n+1

Figur 3.14: Grundläggande variabler i PLC-system.

Det finns också datatyper DINT och DWORD där står D för Double d v s två 16-bitars register slås ihop till ett samverkande 32-bitars vilket innebär att betydligt större tal kan hanteras. Vid typ DINT och DWORD upptas två 16-bits registerplatser som adresseras %MD0.n där n är löpnummer. Deklaration av adressplats görs dock till en adress men då ockuperas också adressen närmast över t ex om %MD0.35 adresseras med en DINT kommer 16-bitars registerplatserna %MW0.35 och %MW0.36 att ockuperas.

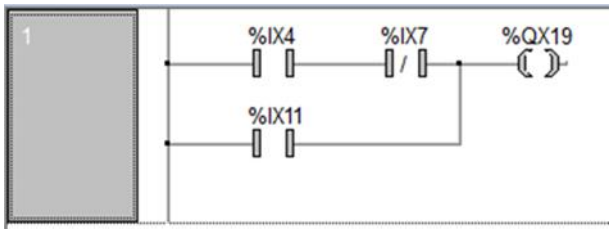
3.2.2. Ladderprogrammering (LD –Ladder Diagram)

Före PLC-systemens genombrott på 70-talet byggdes i stort sett all styrutrustning som reläsystem. En förutsättning för detta genombrott var att den nya styrutrustningen skulle vara attraktiv för den stora stab av ingenjörer som sysslade med reläsystemskonstruktion. För att åstadkomma detta skapades ladderprogramspråket vilket är ett grafiskt programmeringsspråk som bygger på en efterapning av reläschemat. Programmet byggs upp som ett relälinjeschema där insignalerna till de logiska operationerna ligger som slutande eller brytande kontakter och resulterande utsignaler ligger som belastningar i respektive krets. Programmeringssättet etablerade sig alltså mycket tidigt och är fortfarande mycket vanligt förekommande. Uppskattningsvis är 70-80 % av all hittills utvecklad PLC-kod skriven på ladderform. Ladderdiagrammet anses ge en mer överskådlig bild över styrningen än vad instruktionslista och funktionsblock gör.

Här följer några exempel på ladderprogram.

Exempel 3.1:
Kombinatoriska villkoret $\%QX19 = \%IX4 * \overline{\%IX7} + \%IX11$

Ger ladderschema:

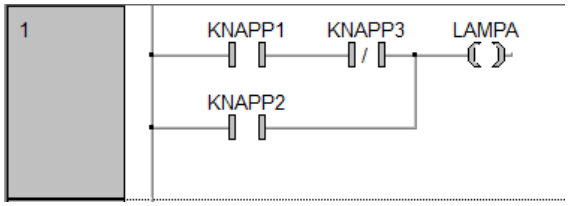


Inför vi identifierare för de olika signalerna enligt globala variabellistan nedan där Identifier-benämningen är en mera processnära benämning än ingångsbeteckningen (IEC-Adressen) så blir programmet mera lättolkat. (Kolumnen med MIT-adress anger adresser specifika för fabrikatet Mitsubishi från tiden före standarden. Dessa är uppbyggda efter funktionsmodulernas placering och med hexadecimal numrering 0-F av portarna på varje modul. Y13 är då utgångsport (Y) modulplats 1 och port nr 3, X4 är ingångsport (X), modulplats 0 (underförstått)och port nr 4.)

Global Variable List

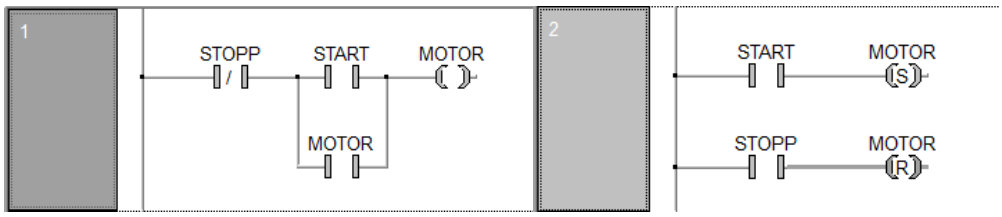
	Class	Identifier	MIT-Addr.	IEC-Addr.	Type
0	VAR_GLOBAL	LAMPA	Y13	%QX19	BOOL
1	VAR_GLOBAL	KNAPP1	X4	%IX4	BOOL
2	VAR_GLOBAL	KNAPP2	XB	%IX11	BOOL
3	VAR_GLOBAL	KNAPP3	X7	%IX7	BOOL

Samma program presenteras då i editorn enligt:

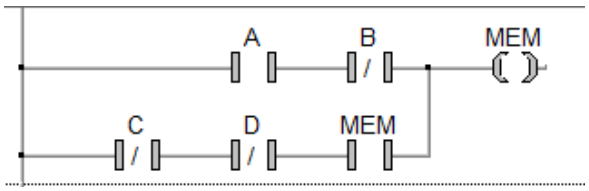


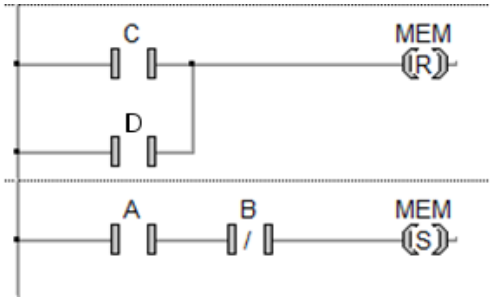
Programspråket är alltså analogt med reläschema där snedstrecket i symbolen för KNAPP3 innebär invertering av signalen.

Exempel 3.2:
Relähållkrets blir i ladderform enligt vänstra nätverket nedan. Ett alternativt sätt att beskriva samma minnesfunktion i ladder visas till höger. I och med den beskrivningsformen har man lämnat kopplingen till reläschema men får mera lättolkade SET- och RESET-villkor. S innebär alltså SET (1-ställ) och R står för RESET (0-ställ). Att resetvillkoret placerats efter setvillkoret innebär att minnet blir reset-dominant.



Exempel 3.3:
Ett ytterligare exempel på logiskt minne men med setvillkor $\mathrm{A} \cdot \overline{\mathrm{B}}$ *, resetvillkor* $\mathrm{C} + \mathrm{D}$ *och setdominant som visas i två varianter.*



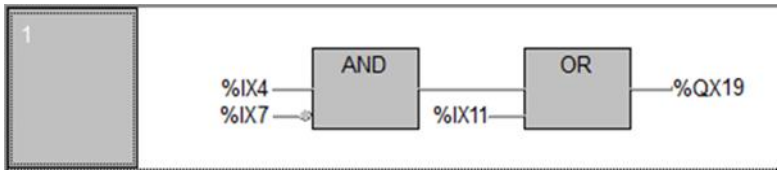


3.2.3. Funktionsblock (FBD – Function Block Diagram)

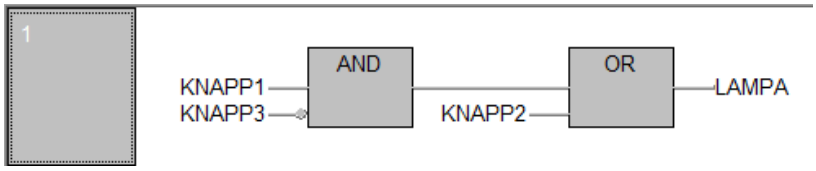
FBD beskriver logiken som kopplade logiska funktionsblock. I följande exempel används de mest grundläggande funktionerna men som kommer att framgå av fortsättningen finns ett stort antal funktionsblock tillgängliga i utvecklingsmiljöerna som används vid både FBD, LD och SFC-programmering. Här följer exempel på FBD-program som bygger på LD-exemplen ovan.

Exempel 3.4:
Liksom i Exempel 3.1 används det kombinatoriska villkoret
$\%\mathrm{QX19} = \%\mathrm{IX4} * \overline{\%\mathrm{IX7}} + \%\mathrm{IX11}$

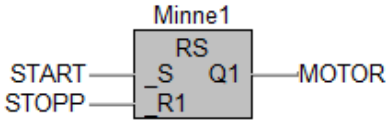
Ger FBD-program:



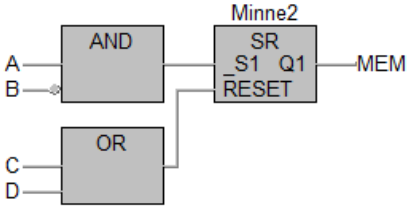
Alternativt med användning av deklarerade variabler med Identifier-benämningar:



Exempel 3.5:
FBD-program analogt med Exempel 3.2. Funktionsblocket kräver allokering av minnesplats varför det ges ett Instance-namn, här Minne1, som gör funktionsblocket unikt med egen identitet. RS innebär RESET-dominant.



Exempel 3.6:
FBD-program analogt med Exempel 3.3. SR innebär SET-dominant minne.



3.2.4. Instruktionslistan (IL – Instruction List)

Instruktionslistprogram består av en serie av instruktioner som matas in efter varandra. Instruktionen består av en operationsdel och operaranddel. Operationsdelen bestämmer vad som skall göras dvs. oftast någon logisk operation. Operanden anger vilken signal som operationen skall utföras på. Här följer på IL-program analoga med tidigare LD- och FBD-program, Exempel 3.1 - Exempel 3.3 respektive Exempel 3.4 - Exempel 3.6.

Exempel 3.7:

```
LD      KNAPP1
ANDN    KNAPP3
OR      KNAPP2
ST      LAMPA
```

Exempel 3.8:

```
LD      START
S       MOTOR
LD      STOPP
R       MOTOR
```

Exempel 3.9:

```
LD      C
OR      D
R       MEM
LD      A
ANDN    B
S       MEM
```

Mindre PLC-system programmerades tidigare ofta direkt via programmeringsdosa kopplad till PLC-systemet. Då används instruktionslistan som programmeringsform då en sådan lista är lätt att knappa in via en enkel knappsats på programmeringsdosan och programmet kan presenteras rad för rad på en enkel display. Med datorbaserade programmeringsmiljöer har användningen av instruktionslista minskat väsentligt. IL-program kommer inte att beröras ytterligare i denna skrift.

3.2.5. SFC och ST.

Ytterligare två programspråk finns i standarden:

- SFC – Sequence Function Chart – Funktionsdiagramprogrammering
- ST – Structured Text – Programmering i strukturerad text, ett högnivåspråk liknande C

SFC tas upp i senare separata avsnitt.

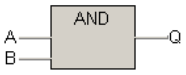
Kap 4. Instruktionsuppsättning i standard IEC 61131-3.

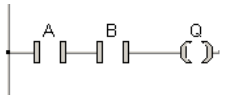
Föregående avsnitt behandlade hur grundläggande logik programmeras i de tre IEC-standardspråken LD, FBD och IL. När vi nu går vidare för att titta på ytterligare tillgängliga instruktioner finner man att dessa inte kan beskrivas i LD utan utgörs av funktionsblock vilka vid FBD-programmering faller in naturligt men också integreras i LD-program om man föredrar detta. Det som skiljer LD- och FBD-programmering är alltså hur man beskriver grundläggande logik och minnesfunktioner. Av den anledningen kommer nu ytterligare funktioner att beskrivas. Några exempel ges både i LD och FBD medan några presenteras i FBD-program varvid det överlåts till läsaren att implementera dem i LD-program om så önskas.

4.1. Logik.

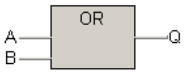
De logiska grundfunktionerna AND, OR, NOT och XOR utgör basen för kombinatoriska villkor och hur de hanteras i programspråken FBD och LD är redan exemplifierat i Kap 3. Likaså har olika sätt att implementera minnen (vippor), både SET-dominanta och RESET-dominanta exemplifierats där. För fullständighetens skull visas grundfunktionerna nedan både som FBD och LD:

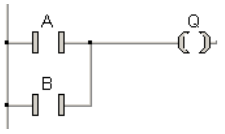
AND



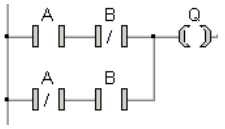


OR



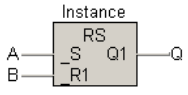


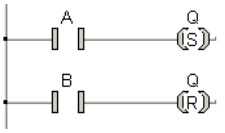
XOR



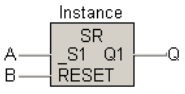
NOT

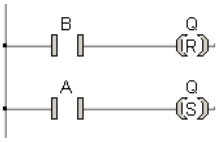
RS-vippa
(RESET-dominant)





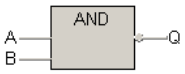
SR-vippa
(SET-dominant)

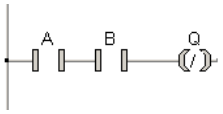




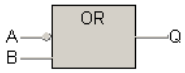
Ingångar och utgångar kan inverteras vilket gör att NOT-blocket inte behöver användas så ofta. Se exempel nedan:

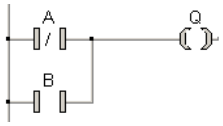
NAND





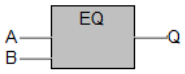
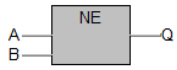
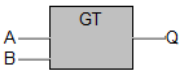
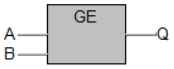
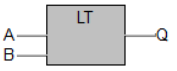
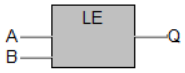
$Q = \overline{A} + B$





Som nämndes i Kap 3 finns också tillgång till register d v s möjlighet att lagra och hantera numeriska värden. Dessa register adresseras %MW0.n och är i de flesta PLC av storleken 16 bitar och rymmer därmed värden mellan 0 och 65535 om registervariabeln deklareras som typ WORD (unsigned) eller värden mellan -32768 och +32767 om den deklareras som typ INT (signed).

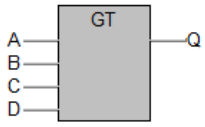
Möjligheten att hantera numeriska värden innebär att behov uppstår att på olika sätt jämföra olika registervärden med varandra varför det finns ett antal instruktioner (funktionsblock) för att utföra dessa jämförelser. Följande instruktioner finns.

EQ - EQual	- lika med,	$A = B$	
NE - Not Equal	- skiljt från,	$A \neq B$	
GT - Greater Than	- större än,	$A > B$	
GE - Greater or Equal	- större eller lika med,	$A \geq B$	
LT - Less Than	- mindre än,	$A < B$	
LE - Less or Equal	- mindre än eller lika med,	$A \leq B$	

För alla dessa funktionsblock är ingångsvariablerna BOOL, INT, WORD, DINT, DWORD tillåtna. Ingångsvariablerna måste dock vara av samma typ d v s om A är av typ INT så måste B vara av typ INT o s v. Utgången, Q, är alltid en BOOL som är TRUE om jämförelsen är sann, i annat fall FALSE.

Jämförande instruktioner finns som jämför ett registervärde med ett annat registervärde enligt ovan men jämförelse kan också göras med en konstant eller med ett konstant värde, se senare Exempel 4.1.

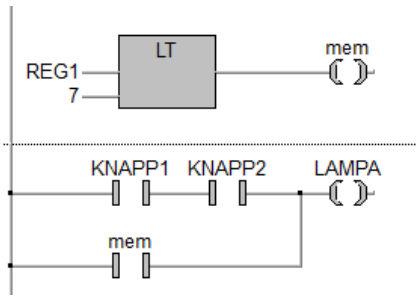
Dessa jämförelsegrindar kan ha fler än två ingångar. För nedanstående exempel gäller att $Q = TRUE$ om $A > B > C > D$.

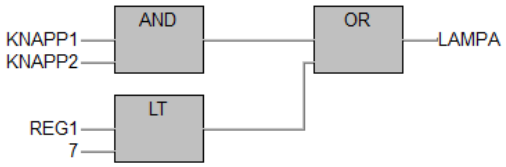


De nu introducerade grindarna är inte realiserbara med relälogik och därmed har vi också lämnat möjligheten att beskriva dem med ladderlogik. Exempel 4.1 nedan visar två analoga program i LD respektive FBD där det framgår att de två språken när man kommer in på instruktioner utanför grundläggande binär logik så används samma grindspråk i LD som i FBD. Variabellistan presenteras också för att visa de olika typ-deklarationerna.

Exempel 4.1:
En lampa, LAMPA, skall lysa om KNAPP1 och KNAPP2 påverkas eller om ett tal REG1 är mindre än 7.

	Class	Identifier	MIT-Addr.	IEC-Addr.	Type
0	VAR_GLOBAL	LAMPA	Y13	%QX19	BOOL
1	VAR_GLOBAL	KNAPP1	X4	%IX4	BOOL
2	VAR_GLOBAL	KNAPP2	X5	%IX5	BOOL
3	VAR_GLOBAL	REG1	D34	%MW0.34	INT
4	VAR_GLOBAL	mem	M67	%MX0.67	BOOL

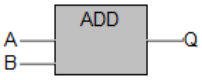




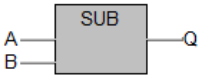
4.2. Beräkningar

Utöver att på olika sätt jämföra olika variabler är det ju också intressant att kunna utföra algebraiska beräkningar. Därför finns följande funktionsblock tillgängliga.
(Samtliga dessa funktionsblock finns med Enable-ingång för villkorad exekvering)

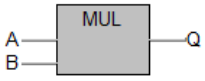
ADD - Addera - $Q = A + B$



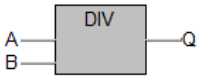
SUB - Subtrahera - $Q = A - B$



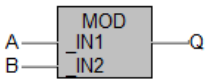
MUL - Multiplicera - $Q = A \cdot B$



DIV - Dividera - $Q = A \div B$



MOD - Modulus - $Q = A \; mod \; B$

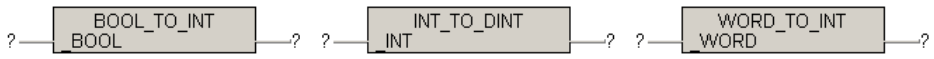


De fyra räknesätten får anses kända men kanske mer okända MOD är förknippad med division på det sättet att den ger den rest som inte kommer med efter en DIV-operation som endast ger heltalsdelen av divisionen utan avrundning.

För alla variablerna A, B och Q gäller att tillåtna datatyper är INT eller DINT. Alla skall vara av samma typ. Observera att ingångsoperander som håller sig inom den deklarerade typens tillåtna intervall kan ge resultat som ligger utanför tillåtna intervallet. Om man t ex vid multiplikation deklarerar variablerna som INT och resultatet av multiplikationen överskrider 16 bitar så kommer de högre bitarna att gå förlorade och resultatet blir alltså felaktigt.

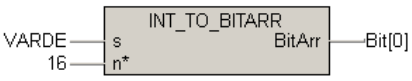
4.3. Typomvandlingar

I sammanhanget kan påpekas att det finns ett antal tillgängliga funktionsblock för typomvandling t ex följande:



Om man vill utvinna de binära bitarna från en en heltalvariabel t ex en INT (16-bitar) kan man göra det med INT_TO_BITARR blocket.

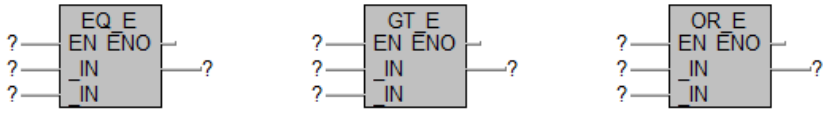
	Class	Identifier	Type
0	VAR	VARDE	INT
1	VAR	Bit	ARRAY [0..15] OF BOOL



Elementen Bit[0]-Bit[15] innehåller nu bitarna i variabeln VARDE.

4.4. Block med enable-ingång (EN / ENO).

Nedan visas några av de ovan redan presenterade blocken men nu med ytterligare ingång och en ytterligare utgång.



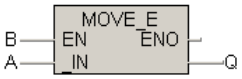
Dessa block har ett tillägg i benämningen, _E. De två ingångarna med märkning _IN är de två ingångar som jämförs enligt jämförelseinstruktionen och resultatet av jämförelsen läggs på nedre omärkta utgången. Men resultatet av jämförelsen länkas endast ut till utgången om enable-ingången, EN, är aktiverad d v s matas med en BOOL som är TRUE. Om EN= FALSE behåller utgången det tillstånd den hade senast EN=TRUE oavsett vad som därefter händer på ingångarna _IN. ENO-utgången är endast en vidarekoppling av EN-ingången och behöver inte anslutas, därav inget ? på ENO-utgångsbenet.

4.5. Förflyttningar

Förflyttning av värde från en variabel eller konstant till en annan variabel utan typomvandling utförs med MOVE och MOVE_E blocken.

I en POU skriven med FBD har blocket MOVE samma funktion som en trådförbindelse och behöver därmed inte användas.

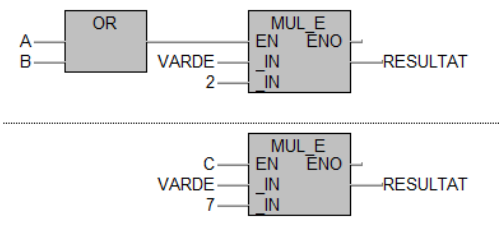
I en POU skriven med SFC har blocket MOVE funktionen att i en Action skriven i FBD minnas tillståndet i följande steg (Stored action).

MOVE	$A \rightarrow Q$	
MOVE_E	$A \rightarrow Q$ endast om $B = TRUE$	

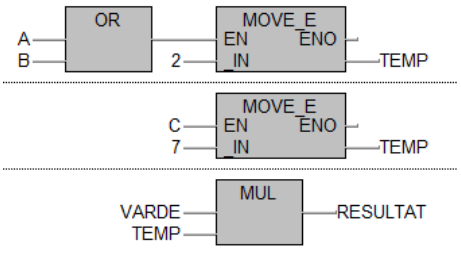
Exempel 4.2:
Ett värde lagrat i INT-variabeln VARDE skall om A eller B påverkas multipliceras med 2 men om C påverkas multipliceras med 7. Resultatet lagras i INT-variabel RESULTAT.

För båda alternativen gäller att om någon av A eller B är påverkade och C samtidigt är påverkad är det C som får genomslag och därmed multiplikation med 7 eftersom koden exekveras uppifrån och ned.

Alternativ 1:

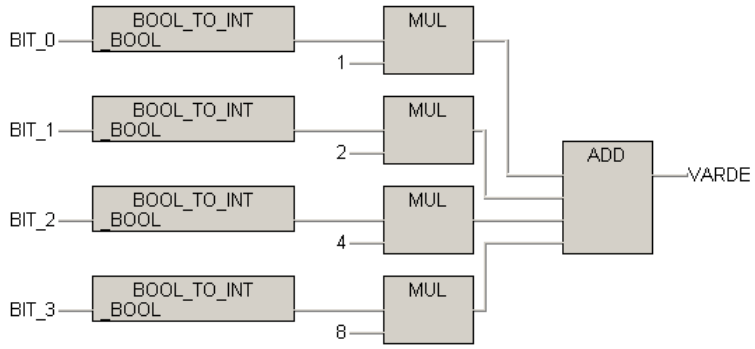


Alternativ 2:



Exempel 4.3:
Ett fyra bitars tal (0-15) kommer in till ett PLC via fyra digitala ingångar kopplade till BOOL-variablerna BIT_0, BIT_1, BIT_2 och BIT_3. Programmet skall överföra detta fyrabitarsvärde till en INT-variabel, VARDE.

Lösningsalternativ A:

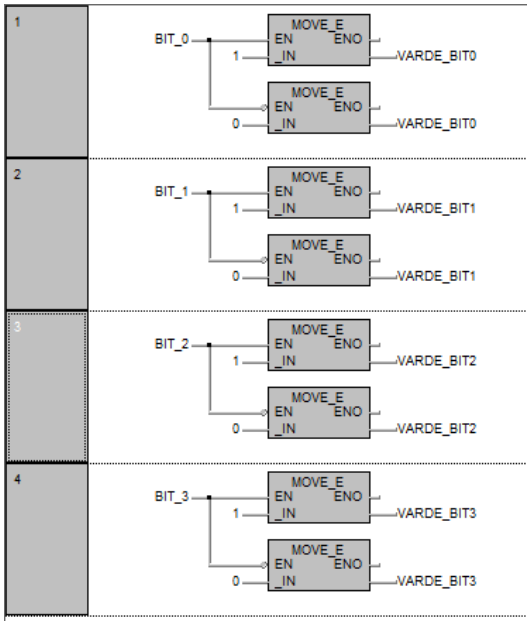


De fyra BOOL-variablerna BIT_0, BIT_1, BIT_2 och BIT_3 typomvandlas till INT-variabler. Därefter multipliceras respektive bit med sin vikt varefter de summeras.

Lösningsalternativ B:

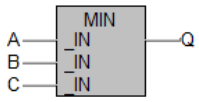
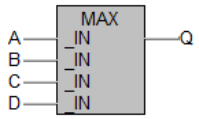
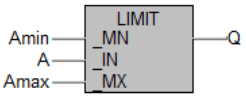
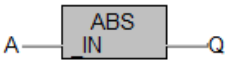
Här åstadkoms samma sak med hjälp av bitadresseringsmöjligheten i ett register (se avsnitt 3.2.1). VARDE är adresserat enligt variabellistan nedan, till %MW0.0 och då når man respektive bit i detta INT-register med bitadresserna %MX0.0.0 , %MX0.0.1…..osv. Varje ingångsbit (BIT_0 osv) 1-ställer eller 0-ställer sin bit i VARDE. Observera den lilla inversringen på ingången till nedre MOVE-grinden i varje nätverksruta. INT-variableln VARDE syns alltså inte i koden men får sitt värde via adresseringen i variabellistan. Observera att denna bitvisa adressering inte är möjlig i alla PLC:er (t ex inte i Mitsubishi A1S men i Q02)

	Class	Identifier	MIT-Addr	IEC-Addr.	Type
0	VAR_GLOBAL	BIT_0	X0	%IX0	BOOL
1	VAR_GLOBAL	BIT_1	X1	%IX1	BOOL
2	VAR_GLOBAL	BIT_2	X2	%IX2	BOOL
3	VAR_GLOBAL	BIT_3	X3	%IX3	BOOL
4	VAR_GLOBAL	VARDE	D0	%MW0.0	INT
5	VAR_GLOBAL	VARDE_BIT0	D0.0	%MX0.0.0	BOOL
6	VAR_GLOBAL	VARDE_BIT1	D0.1	%MX0.0.1	BOOL
7	VAR_GLOBAL	VARDE_BIT2	D0.2	%MX0.0.2	BOOL
8	VAR_GLOBAL	VARDE_BIT3	D0.3	%MX0.0.3	BOOL


1
BIT_0
MOVE_E
EN
ENO
1
_IN
VARDE_BIT0
MOVE_E
EN
ENO
0
_IN
VARDE_BIT0
2
BIT_1
MOVE_E
EN
ENO
1
_IN
VARDE_BIT1
MOVE_E
EN
ENO
0
_IN
VARDE_BIT1
3
BIT_2
MOVE_E
EN
ENO
1
_IN
VARDE_BIT2
MOVE_E
EN
ENO
0
_IN
VARDE_BIT2
4
BIT_3
MOVE_E
EN
ENO
1
_IN
VARDE_BIT3
MOVE_E
EN
ENO
0
_IN
VARDE_BIT3

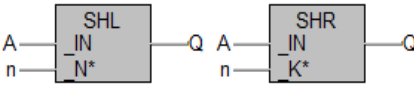
4.6. Ytterligare registerhantering

Här följer ytterligare några funktionsblock för registerhantering.

MIN	minsta värdet A,B,C till Q	
MAX	största värdet A,B,C,D till Q	
LIMIT	om Amin < A < Amax så Q=A om A ≤ Amin så Q = Amin om A ≥ Amax så Q = Amax	
ABS	Q = |A|	

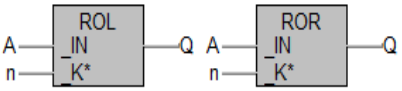
Blocken nedan är för förskjutning av innehållet i ett register deklarerat som WORD eller DWORD. Skiftning SHL och SHR innebär att bitinnehållet i registret skiftas vänster respektive höger så många steg som anges med konstanten n. De bitar som skiftas ut försvinner och de som töms blir nollställda.

SHL, SHR - shiftning av A (se text)



Rotation ROL och ROR innebär att bitinnehållet i registret roteras åt vänster respektive åt höger så många steg som anges med konstanten n. Den bit som skiftas ut läggs i andra änden i den bit som töms.

ROL; ROR - rotation av A (se text)

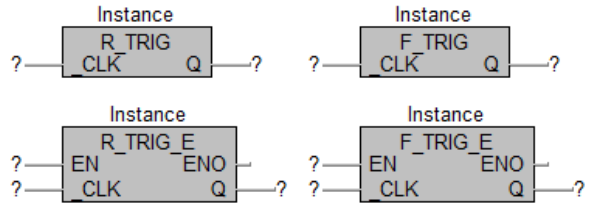


För alla ovan presenterade registerhanteringsinstruktioner finns för var och en också varianten med enable-ingång vilken gör att utförandet av instruktionen är villkorad.

4.7. Flankavkänningar.

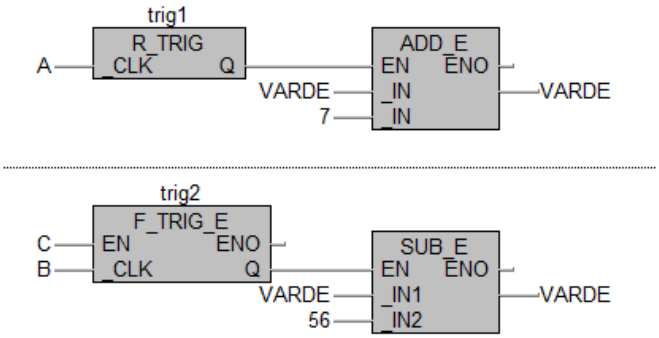
Inte så sällan är det intressant att trigga vissa instruktioner att utföras endast vid positiv eller negativ flank hos den signal som används som enablesignal. Positiv flank (rising edge) är när en binär signal (BOOL) slår om från låg till hög, FALSE till TRUE. Negativ flank (falling edge) är när en binär signal (BOOL) slår om från hög till låg. IEC-standarden

använder sig av ett flankavkännande block som kopplas till efterföljande blocks enableingång.



Exempel 4.4:
Register VARDE skall öka sitt värde med 7 varje gång det kommer en positiv flank på ingång A. Register VARDE skall minska sitt värde med 56 vid negativ flank på ingång B under förutsättning att flagga C är aktiv.

Kod:



Variabellista:

	Class	Identifier	Type
0	VAR	A	BOOL
1	VAR	B	BOOL
2	VAR	C	BOOL
3	VAR	VARDE	INT
4	VAR	trig1	R_TRIG
5	VAR	trig2	F_TRIG_E

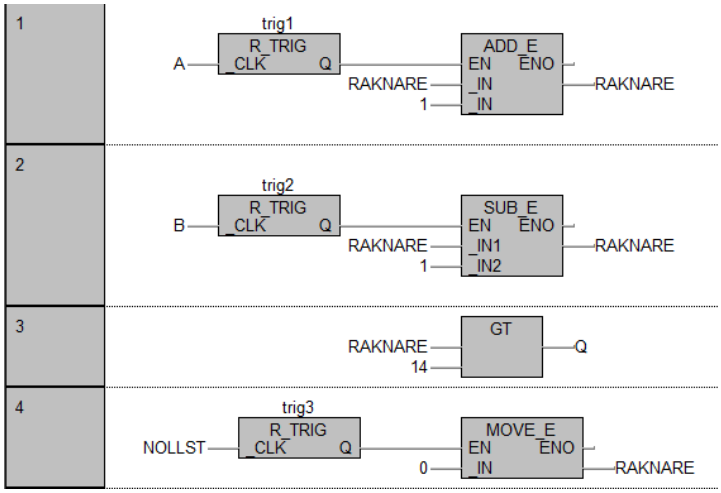
Beteckningarna trig1 och trig2 är Instance för de två Function Blocks som används i Exempel 4.4. Som tidigare nämnts är en del funktionsblock (kallade Function Block) uppbyggda av flera instruktioner med interna variabler. För att ett Function Block skall bli unikt skapas en kopia med unika interna variabler genom att ge blocket ett namn, Instance. Denna Instance måste deklareras i variabellistan. Identifierarna trig1 och trig2 deklareras alltså som R_Trig respektive F_Trig_E i variabellistan eftersom triggningen med A var ovillkorlig men triggningen med B var under villkor att C var påverkad.

Observera att om inte triggning hade använts i Exempel 4.2 där VARDE räknas upp skulle uppräkning ske varje exekveringsvarv som EN-ingången var aktiverad vilket skulle innebära att VARDE ökade med något tusental per sekunds påverkan av EN.

4.8. Räknare.

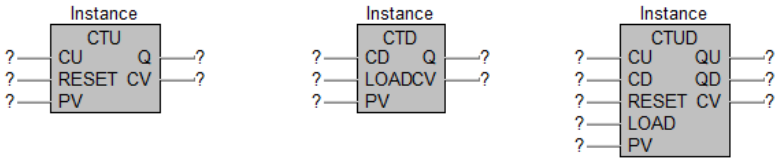
Att räkna pulser från pulsgivare för att bestämma antal passerade paket på en transportbana, varvtal på en axel e dyl är naturligtvis intressant i styrsammanhang. Med hjälp av de funktionsblock som redan är presenterade kan en räknarfunktion byggas och ett visst antal hos räknaren avkodas. Detta illustreras med följande Exempel 4.5*:*

Exempel 4.5:
Varje positiv flank på ingång A ökar register RAKNARE med ett. Varje positiv flank på ingång B minskar register RAKNARE med ett. Om innehållet i register RAKNARE är större än 14 aktiveras utgång Q. Register RAKNARE nollställs av ingång NOLLST.



Triggningen av nollställningssignalen, NOLLST, är inte nödvändig men kan göras för att förhindra att en längre påverkan på NOLLST förorsakar att räknepulser går förlorade.

I standard IEC 61131-3 finns också färdiga räknefunktionsblock, CTU, CTD och CTUD vilket står för Counter Triggered Upward, Counter Triggered Downward respektive Counter Triggered Upward / Downward.



PV står för Preset Value (inställt värde), CV står för Current Value (aktuellt värde), Q är en boolesk utsignal.

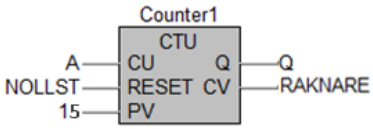
För CTU gäller att om RESET=TRUE så nollställs CV, Q=FALSE och ingen uppräkning är möjlig. Då RESET=FALSE ökas CV med 1 för varje positiv flank in på CU, då CV≥PV sätts Q=TRUE.

För CTD gäller att om LOAD=TRUE så sätts CV=PV, Q=FALSE och ingen nedräkning är möjlig. Då LOAD=FALSE minskas CV med 1 för varje positiv flank in på CD, då CV≤0 sätts Q=TRUE.

CTUD är en kombination av CTU och CTD med gemensamt PV. Utgång QU=TRUE då CV≥PV och QD=TRUE då CV≤0.

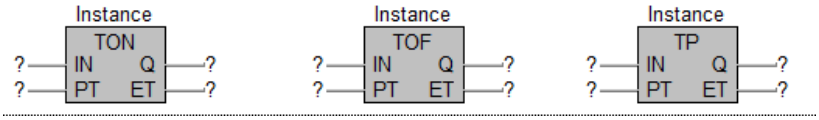
För alla ovan presenterade räknarfunktionsblock finns för var och en också varianten med enable-ingång vilken gör att utförandet av instruktionen är villkorad.

Exempel 4.6:
Här följer en lösning på samma problem som i Exempel 4.5 men utan möjlighet till nedräkning, nu löst med tillgängliga räknarfunktionsblock. (Med ett CTUD-block kunde också nedräkningsfunktionen lösts.)



4.9. Tidskretsar.

Att ta tid och skapa tidsfördröjningar för att bl a skapa tidsutrymme för händelser att ske i processen är en nödvändig funktion att ha tillgång till. Ett PLC innehåller ett antal timers som adresseras via Instance-benämningen hos följande tidsfunktionsblock. I standard IEC 61131-3 finns färdiga tidsfunktionsblock, TON, TOF och TP vilket står för Timer On Delay, Timer Off Delay respektive Timer Pulse.



PT står för Preset Time (inställd tid) vilken anges i datatyp TIME. Formen på tids-angivelsen ser exempelvis ut enlig T#2h34m45s700ms eller T#7s. ET är också typ TIME och ger hittills förfluten tid (Elapsed Time).

För TON gäller att om IN=FALSE så nollställs ET, Q sätts omedelbart FALSE och timern räknar inte. Då IN=TRUE räknas tiden upp i ET och då ET>PT sätts Q=TRUE. Tillslaget hos IN fördröjs alltså tiden PT innan den läggs på Q.

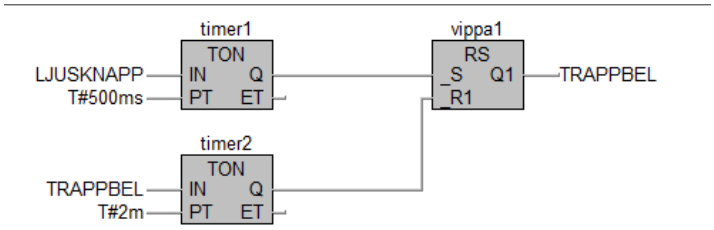
För TOF gäller att om IN=TRUE så sätts Q omedelbart TRUE. Då IN=FALSE fortsätter Q att ligga TRUE ytterligare tiden PT varefter Q=FALSE. IN läggs alltså ut på Q och frånslaget hos Q fördröjs tiden PT efter det att IN=FALSE.

För TP gäller att vid positiv flank på IN läggs Q=TRUE under tiden PT oavsett om signalen på IN är kortare eller längre än PT. ET visar hur lång tid som förflutit av pulsens längd.

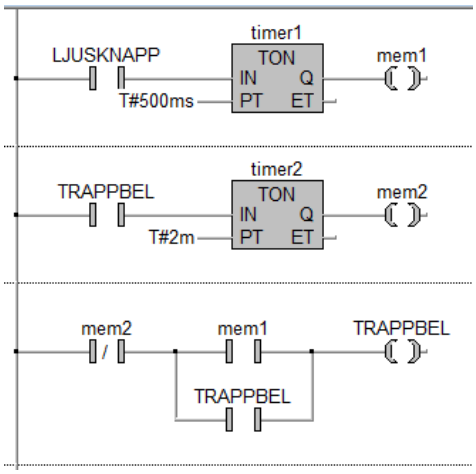
För alla ovan presenterade timerfunktionsblock finns för var och en också varianten med enable-ingång vilken gör att utförandet av instruktionen är villkorad.

Exempel 4.7:
Då ingången LJUSKNAPP aktiveras tänds TRAPPBELYSNING och fortsätter lysa 2 minuter. För att förhindra ofrivilliga nuddningar av ljusknappen har en tidsfördröjning på 0,5 sekunder lagts in innan den reagerar med att tända ljuset.

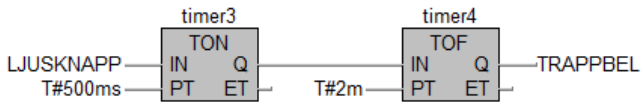
Lösningsalternativ A:
Observera att utsignalen TRAPPBEL kan återkopplas till ingången på timer2 och därmed via RS-vippan ”släcka sig själv” efter 2 minuter.
ET-utgången behöver inte anslutas. Den kan vara intressant att ansluta till en variabel av typ TIME för visning i ett operatörssystem.
Detta alternativ innebär att om LJUSKNAPP är aktiverad längre än 2 minuter kommer en ny 2-minutersperiod att starta utan att TRAPPBEL släcks.



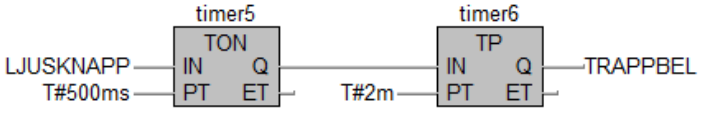
Lösningsalternativ B:
LD-lösning analog med FBD-lösningen i alternativ A.



Lösningsalternativ C:
Här utnyttjas ett tidsfördröjt frånslag vilket gör att belysningsperioden blir 2 minuter ytterligare efter det att ljusknappen släpps.

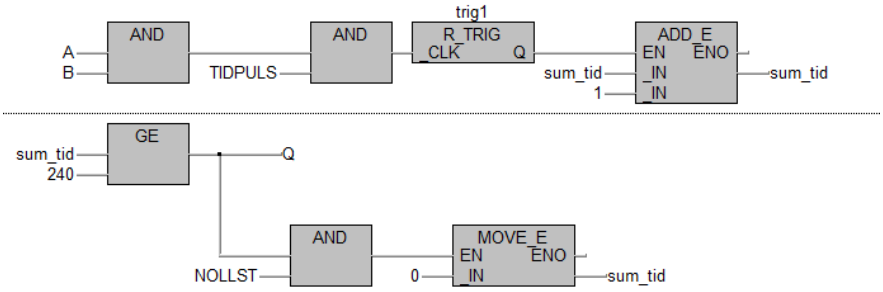


Lösningsalternativ D:
Detta alternativ ger alltid en 2 minuter lång belysning varefter ljusknappen måste släppas och åter påverkas för att en ny 2-minutersperiod skall påbörjas.



När insignalen, IN, på ovan presenterade TON-timer nollställs så nollställs också ET. Önskar man mäta totala tiden för ett diskontinuerligt förlopp, en händelse som dyker upp då och då, kan något av de interna klockpulstågen utnyttjas för att skapa en tidtagning av ett diskontinuerligt förlopp enligt följande exempel.

Exempel 4.8:
Den samlade tiden som två ingångar, A och B, båda är aktiverade mäts. När den samlade tiden överstiger 240 sekunder aktiveras utsignal Q. Då insignal NOLLST påverkas nollställs tidtagningen och Q avaktiveras. Nollställning är inte möjlig om inte avkodade tiden uppnåtts och Q därmed är aktiv.



Global variabellista:

	Class	Identifier	MIT	IEC-Addr.	Type
0	VAR_GLOBAL	A	X1	%IX1	BOOL
1	VAR_GLOBAL	B	X0	%IX0	BOOL
2	VAR_GLOBAL	Q	Y10	%QX16	BOOL
3	VAR_GLOBAL	TIDPULS	SM412	%MX10.412	BOOL
4	VAR_GLOBAL	NOLLST	X2	%IX2	BOOL

Header – Lokal variabellista:

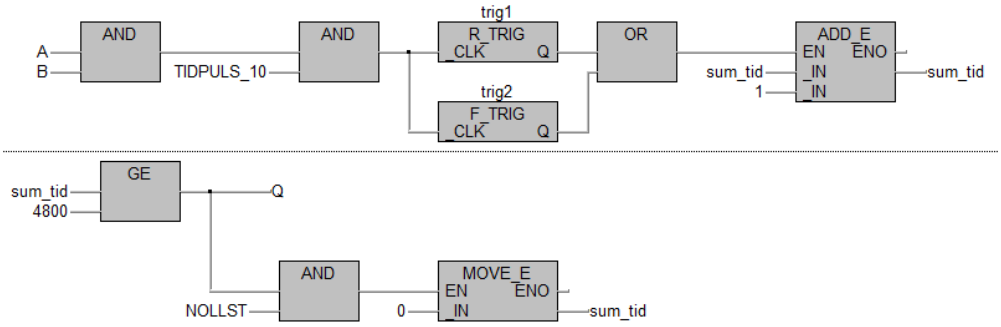
	Class	Identifier	Type
0	VAR	trig1	R_TRIG
1	VAR	sum_tid	INT

Med denna typ av tidtagning kan också flera olika tider avkodas om så önskas.

I variabellistan framgår att TIDPULS är kopplad till adress %MX10.412som enligt Figur 3.13 är en 1 Hz pulståg som då ger en positiv flank per sekund. Räkningen av pulser sker i en INT-variabel (men kunde också göras i TIME-variabel).

Som synes används två olika variabellistor. Globala variabellistan tar upp variabler där man som programmerare väljer destinationsadress vilket är nödvändigt för bl a fysiska in- och utsignaler samt i detta fall val av klockpuls. För arbetsvariablerna sum_tid och trig1 kan systemet själv välja adress. Mera om detta kommer att behandlas i senare avsnitt.

Noggrannheten på denna klockning kan bli dålig. Till- och frånslag från A och B kan ju komma när som hels under sekunden och ett fel på upp till närmare en sekund är möjlig för varje tillslag. Ett alternativ är att använda klockpulssignalen %MX10.410 som jobbar med 10 Hz samt koda av både positiv och negativ flank. Då fås istället uppräkning av sum_tid med 20 ggr/sekund och därmed bättre noggrannhet. Avkodningen av 240 sekunder får då ske med 20·240=4800. Se modifierad kod nedan.



Denna lösning kräver dock en programcykeltid på mindre än 5 ms för att tidräkningen skall hänga med. Inga varningar ges för detta men PLC:ts operativsystem kan via monitoreringsfunktion i utvecklingsmiljön ge besked om programcykeltiden.

4.10. A/D- och D/A-omvandling.

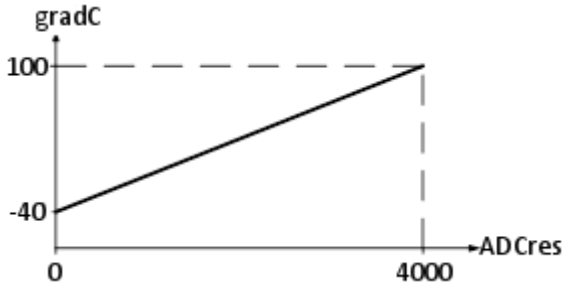
Kodning av A/D-omvandling (ADC) och D/A-omvandling (DAC) skiljer sig mellan olika PLC-fabrikat och ingår alltså inte i någon standard. En A/D-omvandling resulterar dock alltid i att det A/D-omvandlade värdet hamnar i ett register av typ INT eller WORD och kan därefter hanteras som vilket register som helst. ADC- och DAC-enheterna är ofta konfigurerbara med avseende på insignalsomfång och utsignalsomfång.

Insignal till en ADC kan vara både ström och spänningssignal där strömsignalering med industristandarden 4-20 mA är vanlig men även t ex spänningssignalering 0-10V m fl förekommer. Utsignal är då ett digitalt siffervärde t ex 0-255 (8-bitars omvandlare), 0-1023 (10-bitars) eller 0-4095 (12-bitars) men skalan kan också justeras till jämna värden och därmed inte utnyttja hela omvandlarens upplösning som t ex 0-4000.

Exempel 4.9:
En temperaturgivare är kalibrerad att för temperaturintervallet -40 till 100 °C ge en utsignal 4-20mA. Denna strömsignal kopplas till en ADC-ingång hos ett PLC där

insignalsintervall 4-20 mA omvandlas till utsignalsintervall 0-4000. I ett register, benämnt gradC, skall temperaturen i hela °C ligga. Det A/D-omvandlade värdet från temperatur-givaren ligger i register benämnt ADCres. Båda registren antas vara av typ INT.

Lösning:
Sambandet mellan gradC och ADCres kan beskrivas grafiskt enligt



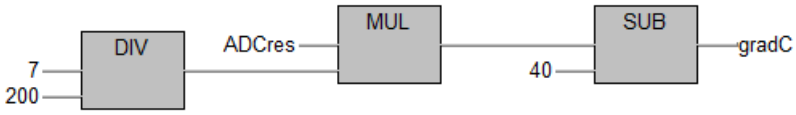
Med tvåpunktformeln kan ekvationen för den räta linjen bestämmas:

$$gradC - (-40) = \frac{100 - (-40)}{4000 - 0} \cdot (ADCres - 0)$$

vilket tillsnyggat ger:

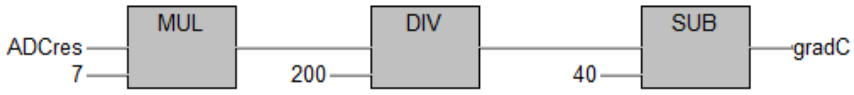
$$gradC = \frac{7}{200} \cdot ADCres - 40$$

En variabel typ INT består av 16 bitar varav en teckenbit och kan därmed hantera tal mellan -32768 och +32767. I den beräkningskedja som skall programmeras får alltså inte något värde riskera att hamna utanför detta intervall. Vidare skall man sträva efter att ha så stora intervall för mellanresultat som möjligt för att inte tappa noggrannhet. Vi börjar med en dålig lösning:



Detta är kanske den mest ”rakt på” lösningen utifrån det matematiska uttrycket ovan men man inser snart att resultatet av första divisionen alltid <1 vilket innebär att resultatet av denna heltalsdivision är noll. Detta gör att gradC = -40 oavsett vad ADCres är.

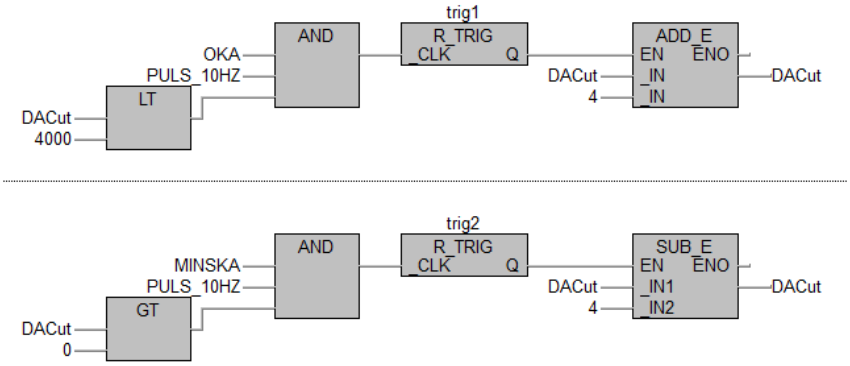
Ny och bättre lösning:



Med denna lösning blir resultatet i första mellanled i intervallet 0 – 28000 vilket ryms i 16 bitar. I denna lösning finns ett avrundningsfel kvar. Resulterade gradC är heltalsdelen av den temperatur som beräknas d v s om temperaturen var t ex 72,8 °C så blir resultatet här 72 °C.

Exempel 4.10:
En DAC är kalibrerad så att det digitala värdet, lagrat i DACut, med omfång 0-4000 ger en utsignal 0-10V. När en digital insignal OKA aktiveras skall analoga utsignalen öka med 0,1 V/sekund vilket motsvarar att DACut skall öka med 40 enheter/sekund. När en digital insignal MINSKA påverkas skall utsignalen minska i samma takt. DACut får inte gå utanför gränserna 0-4000.

Lösning:



4.11. Datatyperna – ARRAY, REAL.

Utöver datatyperna BOOL, INT, DINT, WORD, DWORD, och TIME som presenterats tidigare finns i standarden också typerna REAL och ARRAY. Typen REAL innebär hantering av ett flyttal och därmed användbart enbart när PLC-processorn kan hantera flyttal (Mitsubishi Q02 men inte A1S). De hittills presenterade funktionsblocken fungerar i de flesta fall inte mot REAL. Denna datatyp är mest använd vid mer omfattande beräkningar som då normalt utförs i programspråket Structured Text (ST) som presenteras i senare avsnitt.

Datatypen ARRAY innebär att vektorer i upp till 3 dimensioner kan hanteras. Typdeklarationen görs i variabellistan och en ARRAY deklareras t ex som

	Class	Identifier	Type
0	VAR	VEKTOR_A	ARRAY [0..3] OF INT
1	VAR	VEKTOR_B	ARRAY [0..3,0..4] OF BOOL
2	VAR	VEKTOR_C	ARRAY [0..1,0..3,0..2] OF WORD

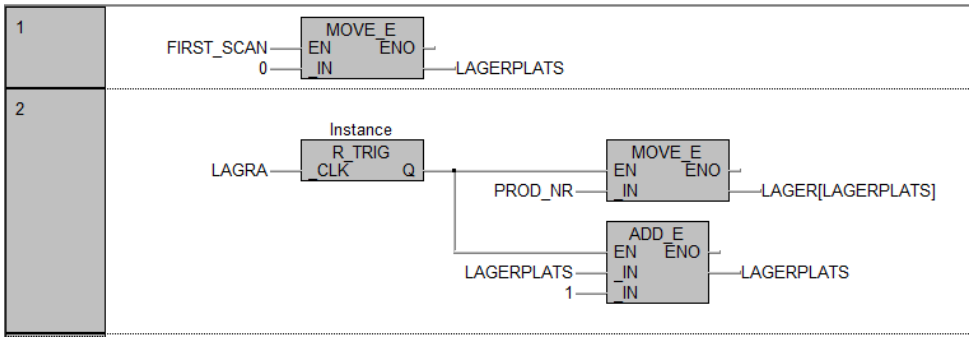
I listan är VEKTOR_A endimensionell med 4 element av typen INT. VEKTOR_B är tvådimensionell med 4x5 element av typen BOOL medan VEKTOR_C är tredimensionell med 2x4x3 element av typen WORD

Exempel 4.11:
Array LAGER[LAGERPLATS] skall hålla reda på vilken typ av produkt som lagras in på lagrets fyra platser. Typen av produkt som lagras in finns i WORD-variabeln PROD_NR och är ett artikelnummer som ryms inom WORD-variabelns 16 bitar. På något sätt har artikelnumret hamnat i PROD_NR för den artikel som står i tur att lagras. En BOOL-flagga LAGRA aktiveras då lagring skall ske. Lagret fylls på från lagerplats 0 och i nummerordning uppåt. Hur det sedan töms o s v lämnar vi därhän.

Lösning:

Global Variable List

	Class	Identifier	MIT	IEC-Addr.	Type
0	VAR_GLOBAL	FIRST_SCAN	SM402	%MX10.402	BOOL
1	VAR_GLOBAL	LAGRA	X5	%IX5	BOOL
2	VAR_GLOBAL	LAGER	D0	%MW0.0	ARRAY [0..3] OF WORD
3	VAR_GLOBAL	LAGERPLATS	D30	%MW0.30	WORD
4	VAR_GLOBAL	PROD_NR	D40	%MW0.40	WORD



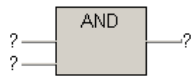
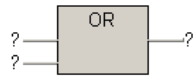
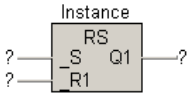
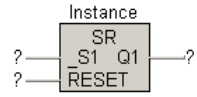
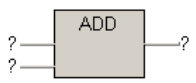
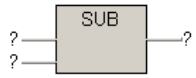
LAGER-vektorn har i variabellistan fått destinationsadress %MW0.0 vilket är startadressen för vektorn. De fyra elementen (artikelnumren) kommer alltså att hamna i de fyra adresserna %MW0.0 till %MW0.3.

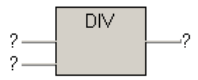
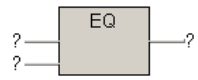
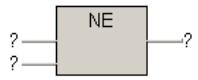
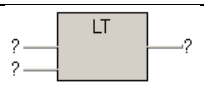
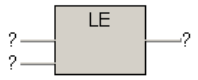
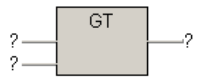
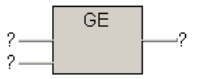
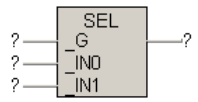
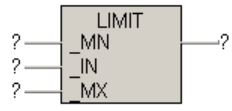
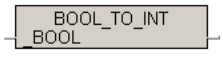
Hur lagras då en tvådimensionell array? Antag att arrayen TVARR[0..2,0..3] dvs 3×4 element deklareras med startdestinationsadress %MW100.0. De olika elementen hamnar då enligt följande i PLC:ets registerarea:

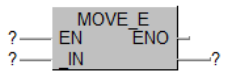
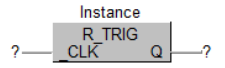
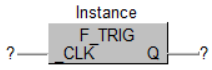
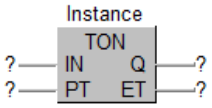
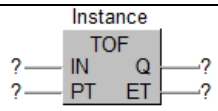
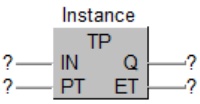
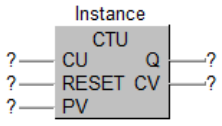
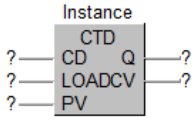
TVARR[0,0] i %MW100.0	TVARR[1,0] i %MW100.4	TVARR[2,0] i %MW100.8
TVARR[0,1] i %MW100.1	TVARR[1,1] i %MW100.5	TVARR[2,1] i %MW100.9
TVARR[0,2] i %MW100.2	TVARR[1,2] i %MW100.6	TVARR[2,2]i %MW100.10
TVARR[0,3] i %MW100.3	TVARR[1,3] i %MW100.7	TVARR[2,3] i %MW100.11

4.12. Lista över vanliga IEC 61131-3 funktionsblock.

Nedanstående tabell visar de vanligaste funktionsblocken som finns tillgängliga i IEC 61131-3. De flesta blocken finns också med enablefunktion dvs EN-ingång och ENO-utgång. In- och utgångar kan inverteras genom att klicka alldeles innanför anslutningen till blocket.
I tabellen visas GX IEC Developers grafiska symboler.

Funktion	**GX IEC Dev.**	**Kort förklaring**
AND		Den logiska funktionen ”OCH” jämför bit för bit av ingångarna och lägger resultatet på utgången till höger. In- och utgångar kan inverteras genom att klicka alldeles innanför anslutningen till blocket. Fler ingångar kan läggas till genom att ”dra” i blockets nederkant
OR		Den logiska funktionen ”ELLER” jämför bit för bit. In- och utgångar kan inverteras. Fler ingångar kan läggas till genom att ”dra” i blockets nederkant.
XOR		Den logiska funktionen ”EXCLUSIVT ELLER”
NOT		Den logiska funktionen ”NOT”. Utsignalen är inversen av insignalen. Eftersom in- och utgångar på de flesta block kan förses med inverteringsringar behöver detta block sällan användas.
RS		RS-vippan sätter TRUE på utgången när SET är TRUE, och FALSE på utgången när RESET1 är TRUE. Om båda ingångarna är TRUE samtidigt dominerar RESET1 (1:an betyder dominans).
SR		SR-vippan fungerar som ovanstående, men om båda ingångarna är TRUE samtidigt dominerar SET1.
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.
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.
LIMIT		Insignalen IN skickas vidare till utgången om den ligger inom intervallet MN≤IN≤MX. Annars får utgången värdet MN respektive MX i stället.
Typkonv-ertering		Typkonverteringsblock finns i en mängd varianter.
MOVE		Värdet flyttas från ingången till utgången på blocket. Kan användas i actions (I SFC) för att minnas tillståndet hos variabler i efterföljande steg. (Stored action)

MOVE_E		Som namnet antyder flyttas värdet från ingången till utgången på blocket när ”Enable” (EN) är TRUE.
R_TRIG		Utgången Q blir TRUE när ingången just fått värdet TRUE (positiv flank). Sedan återvänder utgången till FALSE vid nästa ”scan-cycle”. Alla funktionsblock som har ett minne, d.v.s. utgången beror inte enbart på ingångarnas momentana värde, måste ha ett instansnamn.
F_TRIG		Fungerar motsvarande för negativ flank. Utgången blir TRUE precis när ingången växlat till FALSE, men återvänder nästa ”scan-cycle” till FALSE.
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 ett fördröjt frånslag på Q när IN slås från. På ET visas tiden som har gått från frånslaget. Om fördröjning önskas på både positiv och negativ flank kan TON och TOF seriekopplas.
TOF		Blocket ”Timer Pulse” ger en puls av längd PT när IN aktiveras oberoende av varaktighet av IN. På ET visas tiden som har gått från tillslaget.
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 ≥ 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.

4.13. Grundläggande datatyper IEC 61131-3

Datatyp	**Värdeområde**	**Exempel på konstanter**
BOOL	FALSE TRUE	FALSE TRUE
INT	-32768 …. +32767	0 -23 876
WORD	0 …. 65535	0 4567
DINT	-2147483648…. 2147483647	0 -456789 779544
DWORD	0…. 4294967295	0 680777
TIME	T#-24h ….. T#24h	T#2h34m45s25ms T#200ms
REAL	3.4E±38	0.0 2.54 -7.865E4
STRING	Max 50 tecken	”HELLO”
ARRAY	Max 3 dimensioner	

4.14. Identifiers och reserverade nyckelord IEC 61131-3

Identifiers (variabelnamn, funktionsnamn, programnamn mm) är ”case insensitive” d v s det görs ingen skillnad på versaler och gemener.

- Identifiers får bestå av bokstäver (EJ å, ä ,ö), siffror och ”underscores _”
- Identifiers måste börja med en bokstav eller ”underscore _”

Följande ord är reserverade nyckelord i IEC 61131-3 och får inte användas som identifiers:

ABS ACOS ACTION ADD AND ANDN ANY ANY_BIT ANY_DATE ANY_INT ANY_NUM ANY_REAL ARRAY ASIN AT ATAN

BOOL BY BYTE

CAL CALC CALCN CASE CD CDT CLK CONCAT CONFIGURATION CONSTANT COS CTD CTU CTUD CU CV

DATE DATE_AND_TIME DELETE DINT DIV DO DS DT DWORD

ELSE ESIF END_ACTION END_CASE END_CONFIGURATION END_FOR END_FUNCTION END_FUNCTION_BLOCK END_IF END_PROGRAM END_REPEAT END_RESOURCE END_STEP END_STRUCT END_TRANSITION END_TYPE END_VAR END_WHILE EN ENO EQ ET EXIT EXP EXPT

FALSE F_EDGE F_TRIG FIND FOR FROM FUNCTION FUNCTION_BLOCK

GE GT

IF IN INITIAL_STEP INSERT INT INTERVAL

JMP JMPC JMPCN

L LD LDN LE LEFT LEN LIMIT LINT LN LOG LREAL LT LWORD

MAX MID MIN MOD MOVE MUL MUX

NE NEG NOT

OF ON OR ORN

P PRIORITY PROGRAM PT PV

Q Q1 QU QD

R R1 R_TRIG READ_ONLY READ_WRITE REAL RELEASE REPEAT REPLACE RESOURCE RET RETAIN RETC RETCN RETURN RIGHT

ROL ROR RS RTC R_EDGE

S S1 SD SEL SEMA SHL SHR SIN SINGLE SINT SL SQRT SR ST STEP STN STRING STRUCT SUB

TAN TASK THEN TIME TIME_OF_DAY TO TOD TOF TON TP TRANS TRUE TYPE

UDINT UINT ULINT UNTIL USINT

VAR VAR_ACCESS VAR_EXTERNAL VAR_GLOBAL VAR_INPUT VAR_IN_OUT VAR_OUTPUT

WHILE WITH WORD

XOR XORN

4.15. Reserverade nyckelord Mitsubishi

Följande ord är reserverade och specifika för Mitsubishi och får inte användas som identifiers:

B0	B1	B2 ….
C0	C1	C2 ….
CC0	CC1	CC2 ….
CN0	CN1	CN2 ….
D0	D1	D2 ….
F0	F1	F2 ….
J0	J1	J2 ….
L0	L1	L2 **.....**
M0	M1	M2 ….
P0	P1	P2 ….
S0	S1	S2 ….
SB0	SB1	SB2 ….
ST0	ST1	ST2 ….
SW0	SW1	SW2 ….
T0	T1	T2 ….
TC0	TC1	TC2 ….
U0	U1	U2 ….
V0	V1	V2 ….
W0	W1	W2 ….
X0	X1	X2 ….
Y0	Y1	Y2 ….
Z0	Z1	Z2 ….

Kap 5. Specifikt för PLC-fabrikat Mitsubishi.

I Kap 4 presenterades tillgängliga instruktioner och variabler för PLC-programmering enligt standard 61131-3. Programutvecklingsmiljön enligt samma standard beskrivs senare i Kap 6 och illustreras med utvecklingsmiljön GX IEC Developer vars utvecklingsmiljö är riktad till PLC av fabrikat Mitsubishi men stödjer standarden.

Tillverkare av PLC-system har ansvar gentemot sina gamla kunder och användare från före standardens tillkomst. De kompletterar därför sina utvecklingsmiljöer med instruktioner och variabelbeteckningssätt som gällde för fabrikatet före standardens tillkomst och som kunderna är vana vid att använda. Eftersom detta skrivna material i första hand vänder sig till studenter som möter PLC-fabrikatet Mitsubishi i sin utbildning presenteras i detta avsnitt fabrikatets specifika instruktioner som finns med i utvecklingsmiljön GX IEC Developer och därför kan vara motiverat att känna till vid användandet av denna programmeringsmiljö. Många av dessa instruktioner kan också visa sig praktiska och smidiga att använda. Att de inte är med som standardinstruktioner beror på att framtagningen av en standard efter att många aktörer är etablerade på marknaden är ett tagande och givande och ett passande så att ingen aktör skall få konkurrensfördel.

5.1. Mitsubishis signalbeteckningar.

Mitsubishis ursprungliga adresseringssätt skiljer sig ifrån IEC-standardens vilket beskrivs i nedanstående tabell där exempeladresser angivits. I utvecklingsmiljön GX IEC kan man välja vilken adressering man vill använda, IEC eller MIT.

En fördel med MIT-adressering jämfört med IEC-adressering är att MIT har hexadecimal numrering och då varje modul (ingångsmodul, utgångsmodul osv) har 16 och ibland 32 adressplatser innebär det att t ex 1 i adressen Y16 pekar på att adressen pekar mot modul nr 2 efter CPU-modulen på bakplanet. Standard IEC tillämpar decimal numrering varvid denna koppling till modulposition faller bort.

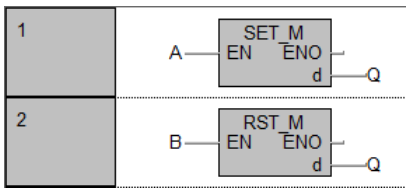
Variabel	**Datatyp**	**IEC-adress**	**MIT-adress Q02_CPU**	**MIT-adress A1S_CPU**
Digital ingång	BOOL	%IX10	XB	XB
Digital utgång	BOOL	%QX21	Y16	Y16
Minnesflagga	BOOL	%MX0.238	M238	M238
Minnesflagga, batteriuppbackad	BOOL	%MX8.124	L124	L124
Specialminne	BOOL	%MX10.402	SM402	M9032
16-bits register	INT, WORD	%MW0.324	D324	D324
32-bits register	DINT, DWORD	%MW0.34+%MW0.35	D34+D35	D34+D35
Specialregister	WORD	%MD10.210	SM210	D9025

Det finns ett antal specialminnesflaggor varav några användbara redovisas i tabell nedan med adresser. För klockpulstågen i tabellen anges periodtid för pulstågen som Tsv.

Beskrivning	**IEC-adress**	**MIT-adress Q02**	**MIT-adress A1S**
TRUE första scan-cykeln efter RUN	%MX10.402	SM402	M9038
10 Hz klockpulståg ,Tsv 0,1s	%MX10.410	SM410	M9030
5 Hz klockpulståg, Tsv 0,2s	%MX10.411	SM411	M9031
1 Hz klockpulståg, Tsv 1s	%MX10.412	SM412	M9032
0,5 Hz klockpulståg, Tsv 2s	%MX10.413	SM413	M9033
Klockpulståg Tsv i sekunder skrivs i register	%MX10.415 %MD10.415	SM415 SD415	_

5.2. Logiska instruktioner – Mitsubishispecifika.

Vad gäller minnesfunktioner har IEC-standard RS- och SR-funktionsblocken redan presenterats. Mitsubishi har två fristående block, ett för SET och ett för RST vilket gör att set och reset kan separeras till olika platser i koden. Dessa block motsvarar –(S)- och –(R)- vid ladderprogrammering. Det är framför allt användbart i SFC-programmering där man då i ett steg kan ettställa en signal för att sedan i något senare steg nollställa variabeln. Figur 5.1 nedan visar de båda blocken och med den inbördes placeringen är de båda blocken analoga med ett RS-block d v s reset-dominant vippa. Med omvänd ordning, RST överst, blir det SR-funktion. Ändelsen _M i SET_M och RST_M är den ändelse som kännetecknar alla Mitsubishispecifika funktionsblock.

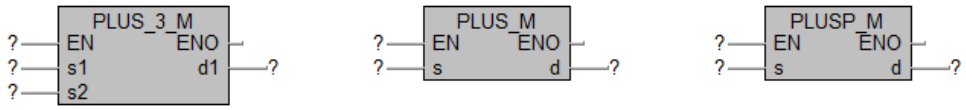


Figur 5.1: Mitsubishispecifika SET- och RST-block

Det kan nämnas att vid kompilering av kod i GX IEC Developer översätts först till Melsec-kod som är Mitsubishis egna utvecklingsmiljö med den ursprungliga instruktions-uppsättningen. Melsec-koden översätts i nästa steg till maskinkod för PLC-processorn. Det innebär att t ex RS- och SR-blocken är SET_M- och RST_M- block paketerade på lämpligt sätt. Möjligheten att göra sådana ”paketeringar” och på det viset skapa egna funktionsblock är centralt i IEC-standarden och är viktig för rationell programutveckling.

5.3. Beräknings- och förflyttningsinstruktioner– Mitsubishispecifika.

När det gäller algebraiska operationer finns ett flertal Mitsubishi-specifika funktionsblock som kan rationalisera programutvecklingen. När det gäller addition finns följande additionsrelaterade block:

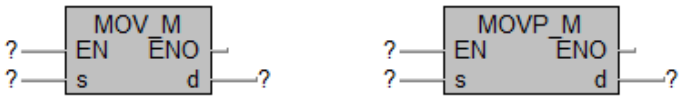


Den första, PLUS_3_M är identisk med IEC-blocket ADD_E dvs då EN aktiv utförs d1 = s1+s2. Block nr 2, PLUS_M, innebär att destinationsregistret vid d ökas med värdet (register eller konstant) kopplat till s då EN=TRUE. För båda dessa gäller att operationen utför varje programcykel som EN=TRUE. I det tredje blocket, PLUSP_M, är en positiv flanktriggning inbyggd i EN-ingången vilket innebär att destinationsregistret ökas endast vid positiv flank in till EN.

Motsvarande block finns för de övriga räknesätten benämnda MINUS_3_M, MINUSP_3_M, MINUS_M, MINUSP_M, MULTI_3_M, DIVIDP_3_M ..o s v,

Utöver dessa finns två block för att vid positiv flank öka respektive minska ett register med 1.

Förflyttning av registervärden eller konstanter gjordes med IEC-instruktionerna MOVE eller MOVE_E. Mitsubishispecifika block med motsvarande funktion ses nedan där som synes också inbyggd positiv flanktriggning finns med.



En finess är också att med dessa block kan bitinformation överföras till 16-bitars register enligt följande exempel.

Exempel 5.1:
Vi tittar på en alternativ lösning till Exempel 4.3 som var formulerat som följer. Ett fyra bitars tal (0-15) kommer in till ett PLC via fyra digitala ingångar kopplade till BOOL-variablerna BIT_0, BIT_1, BIT_2 och BIT_3. Programmet skall överföra detta fyrabitarsvärde till en INT-variabel, VARDE.

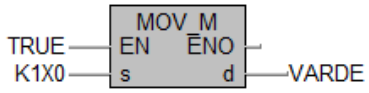
Lösning:

Med följande lösningssätt måste tilldelningen av ingångsvariablerna vara känd och utnyttjas i koden. Antag tilldelning enligt:

Global Variable List

	Class	Identifier	MIT	IEC-Addr.	Type
0	VAR_GLOBAL	BIT_0	X0	%IX0	BOOL
1	VAR_GLOBAL	BIT_1	X1	%IX1	BOOL
2	VAR_GLOBAL	BIT_2	X2	%IX2	BOOL
3	VAR_GLOBAL	BIT_3	X3	%IX3	BOOL
4	VAR_GLOBAL	VARDE	D0	%MW0.0	INT

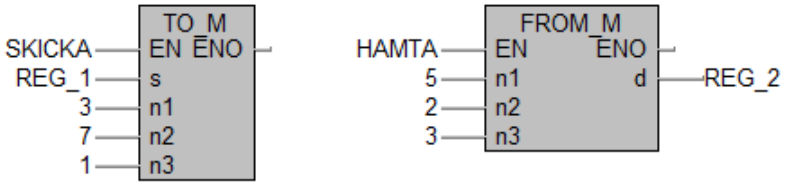
Nu kan överföring av de 4 bitarna göras med en enda instruktion:



K1X0 innebär att med början från X0 och uppåt tas 4 bitar. X0 läggs då i lägsta biten, X1 i näst lägsta o s v. Dessa överförs ovillkorlig (TRUE) varje exekveringsvarv till registret VARDE. (För att ge ytterligare ett exempel så innebär K3M8 att med början från interminne M8 och uppåt överförs 12 bitar (3x4)där M8 blir lägsta bit o s v.)

Vissa moduler i Mitsubishi-systemet kallas ”intelligenta moduler”. En ”intelligent modul” kan vara A/D- , D/A-omvandlarmoduler, kommunikationsmoduler för seriell kommunikation eller motorstyrningsmoduler.. Dessa intelligenta moduler innehåller en egen registeruppsättning med egna adresser.

För förflyttning av registervärden mellan CPU-delens minne till en ”intelligent modul” hos PLC:t används följande två instruktioner.



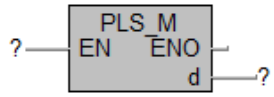
Modulerna adresserad´s efter den ordning till höger om CPU-modulen som de är placerade på PLC:ts monteringsunderlag, det så kallade bakplanet.

TO_M förflyttar, då SKICKA=TRUE, registervärdet REG_1 till modul på adressplats 3 på bakplanet och till registeradress 7 i denna modul. Adressplats på bakplanet är normalt samma som positionen på bakplanet räknat från CPU-modulen och med början plats 0. Men en modul kan ta upp två adressmodulplatser. Modulnumrering för aktuellt styrsystem fås genom uppkoppling av systemet mot GX IEC Developer och kommendera Debug – System Monitor.

FROM_M innebär att, då HAMTA=TRUE, hämtas registervärde från modulplats 5, adress 2 till REG_2. Men eftersom det i detta fall står 3 vid n3 så innebär det att 3 register hämtas, adress 2 till REG_2, adress 3 till CPU-minnesadress ovanför REG_2s tilldelade adress och adress 3 till CPU-minnesadress två steg ovanför REG_2s tilldelade adress. Normalt läses ett register i taget dvs n3 sätts normalt till 1.

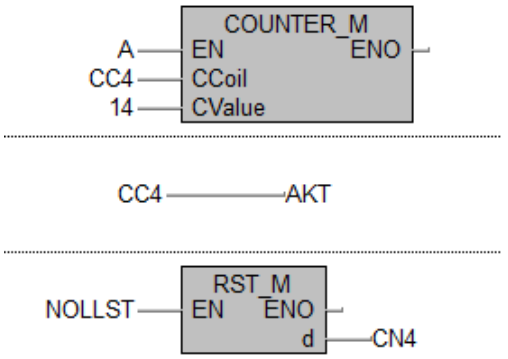
5.4. Flankavkännande instruktioner – Mitsubishispecifika.

Utöver de integrerade flanktriggningar som beskrivits i föregående avsnitt finns ett block för positiv flanktriggning, PLS_M vilket helt motsvarar R_TRIG.



5.5. Räknarinstruktioner – Mitsubishispecifika.

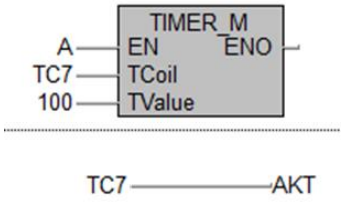
Basblocket för alla IEC-räknarfunktionsblock i Mitsubishisystem är blocket COUNTER_M. Systemet innehåller ett antal räknare (counters), olika antal beroende på CPU-typ. För varje positiv flank på EN-ingången kommer ett till blocket kopplat register CNn att räknas upp med ett där n är räknarens löpnummer. I figuren har räknare nummer 4 använts.



När värdet kopplat till ingång CValue uppnåtts hos CNn kommer en flagga CCn att tändas. På COUNTER_M-blocket är CCn kopplat till en ingång CCoil som inte är möjlig att koppla vidare. Istället anropas CCn för att aktivera den händelse som skall aktiveras av räknaren. I figur nedan aktiverar CC4 signalen AKT. CN4 nollställs av NOLLST och därmed nollställs även CC4.

5.6. Timerinstruktioner – Mitsubishispecifika.

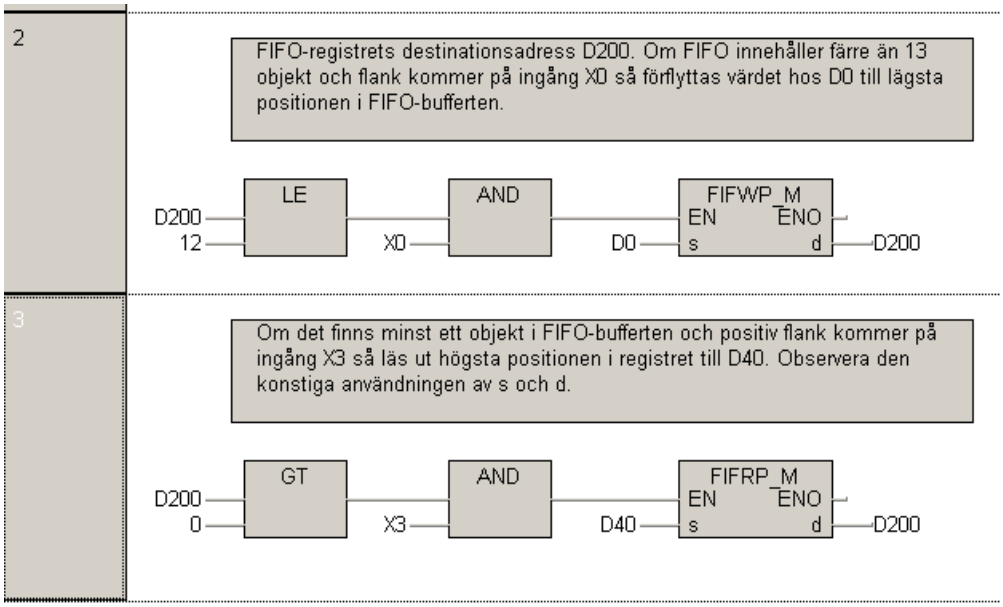
Basblocket för alla IEC-timerfunktionsblock i Mitsubishisystem är blocket COUNTER_M. Systemet innehåller ett antal timers, CCn, olika antal beroende på CPU-typ. Då insignal A till EN-ingången är aktiv pågår en tidtagning med upplösning 100 ms eller 10 ms. Då antalet intervall har uppnått TValue, i detta fall 100 aktiveras TCoil, i detta fall TC7. Då EN-ingången går låg nollställs tidräkningen och TCoil går låg, i detta fall TC7. TC0-TC199 räknar i 100 ms-intervall och TC200-TC255 i 10 ms-intervall.



På TIMER_M-blocket är TCn kopplat till en ingång TCoil som inte är möjlig att koppla vidare. Istället anropas TCn, i detta fall TC7 för att aktivera den händelse som skall aktiveras av timern, i detta fall AKT.

5.7. FIFO-register.

FIFO-register (first in - first out) används för buffring av data t ex vid lagring av komponentidentiteter i en buffert eller för att hålla reda på beställningsköer. Med hjälp av instruktionerna FIFW, FIFWP och FIFR, FIFRP kan man skriva till respektive läsa från ett sådant register. När ett värde skrivs till ett FIFO -register läggs det längst ner i bufferten och då man läser ur registret så tas värdet längst upp i bufferten och alla kvarvarande värden stegar upp ett steg. FIFO-registret adresseras till en adress t ex D200 i vilken lagras antalet värden som finns i registret. D200 räkas alltså upp med ett då skrivning sker till registret och räknas ner med ett då läsning sker ur registret. I nästa D-register, här D201, finns det värde som befinner sig högst upp i bufferten, D202 det näst längst upp osv. Observera att här är det oftast nödvändigt att använda FIFWP_M resp FIFRP_M dvs positiv flanktriggning för annars är risken stor att värden bara rasar in eller ut då enable-villkoret är uppfyllt. Nedan följer ett exempel med kommentarer. Observera den något konstiga användningen av source (s) och destination (d) i FIFR instruktionen.



5.8. A/D- och D/A-omvandling i Mitsubishisystem Q02.

För att kommunicera med omvärlden med analoga signaler utrustas PLC-systemet med moduler för just analoga signaler in och ut. Dessa moduler innehåller då A/D- och D/A-omvandlare och kommunikation med PLC:t sker via ett antal från CPU läs- eller skrivbara register i modulen. Till system Q02 finns några alternativa moduler tillgängliga. Här presenteras dock A/D-modul Q64AD som har 4 analoga ingångar med valbarhet vad gäller signaltyp (ström eller spänning), signalomfång och upplösning enligt tabell nedan.

Signaltyp		Normal upplösning		Hög upplösning	
		Digitalt värde	Upplösning	Digitalt värde	Upplösning
Spänning	0 till 10 V	0 till 4 000	2.5 mV	0 till 16 000	0.625 mV
	0 till 5 V		1.25 mV	0 till 12 000	0.416 mV
	1 till 5 V		1.0 mV		0.333 mV
	- 10 till +10 V	-4 000 till 4 000	2.5 mV	-16 000 till 16 000	0.625 mV
	Egen signaltyp		0.375 mV	-12 000 till 12 000	0.333 mV
Ström	0 till 20mA	0 till 4 000	5 µA	0 till 12 000	1.66 µA
	4 till 20 mA		4 µA		1.33 µA
	Egen signaltyp	-4 000 till 4 000	1.37 µA	-12 000 till 12 000	1.33 µA

Vidare presenteras en D/A-modul Q64DA med 4 analoga utgångar och med samma egenskaper som ingångarna vad gäller omfång, upplösning och omvandlingstid. Modulerna har också mer avancerade funktioner i form av medelvärdesbildning mm.

Arbetsområdet hos in- och utgångarna är inställbart. (Hos de vi använder i labbet är både in och utgångar normalt kalibrerade för digitalt 0 - 4000 motsvarande analogt 0 - 5 V.) Kommunikationen med modulen kräver att modulens placering på bakplanet är känd. På labsystemen sitter modulen i ordning CPU – 1 st digital in – 1 st digital ut – AD – DA vilket medför att AD-modulen upptar modulplats 2 och DA modulplats 3. För det enklaste användarfallet gäller att de kanaler som skall användas enablas samt att analoga signaler som A/D-omvandlats kan läsas till systemet i digital form och att digitala värden kan läsas ut för D/A-omvandling och placeras på analoga utgången.

Inställning av signaltyp och omfång för de fyra kanalerna både för Q64DA och Q64AD ställs med en så kallad switch enligt:

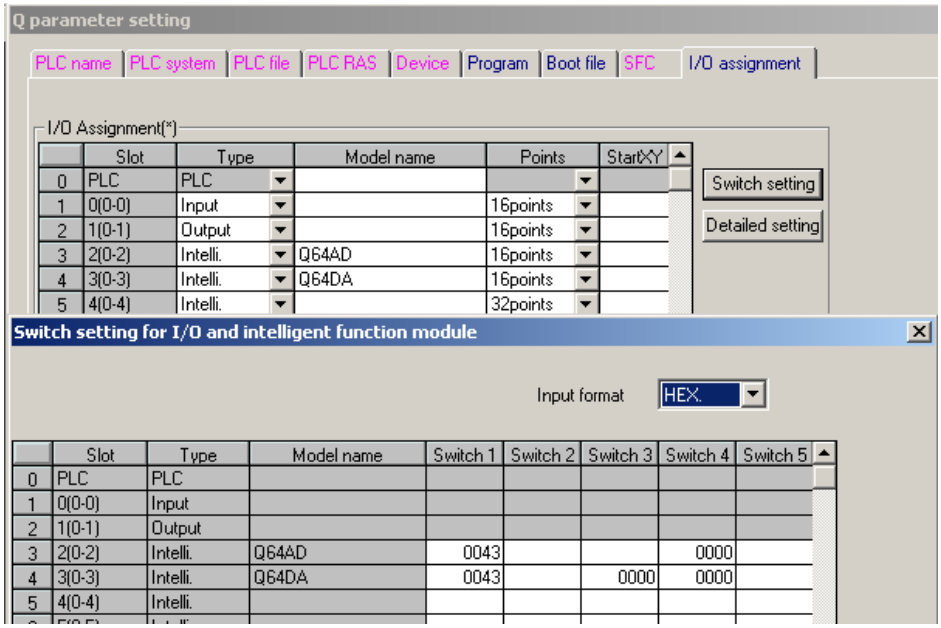
Signaltyp	**Hex-värde i Switch 1**
4 – 20 mA	0
0 - 20 mA	1
1 - 5 V	2
0 - 5V	3
-10V till 10V	4
0 – 10V	5

Switcharna ställs in via GX IEC Developer – *Parameter – PLC – I/O assignment* och sedan *Switch*. Kommunikationen med modulen kräver att modulens placering på bakplanet är känd. På labsystemen sitter modulerna i ordning CPU – X80 (16 digitala in) – Y10 (16 digitala ut) – Q64AD – Q64DA vilket medför att A/D-modulen upptar modulplats 2 och D/A modulplats 3.
Med samma konfiguration för både Q64AD- och Q64DA-modul:

CH1 0-5V ⇒ Switch 1=3,
CH2 -10 till 10V ⇒ Switch 1=4
CH3 o CH4 4-20mA ⇒ Switch 1= 0

Konfigurationen ser ut enligt nedan:

.



Switch 4 som är satt till 0000 innebär att normalt omvandlingsläge och normal upplösning valts.

För att denna konfigurering skall slå igenom måste PLC:ts CPU resetas efter det att konfigurationen laddats över till PLC:t.

Möjlighet finns att medelvärdesbilda över viss tid eller för ett visst antal omvandlingar men för detta hänvisas till manual. För det enklaste användarfallet med direkt avläsning av A/D-omvandlade värdet resp. direkt utläsning av värde för D/A-omvandling gäller att de kanaler som skall användas enablas samt att analoga signaler som A/D-omvandlats kan läsas till systemet i digital form och att digitala värden kan läsas ut för D/A-omvandling och placering på den fysiska analoga utgången.

Som nämnts ovan har modulerna ett antal från CPU:t läs- och/eller skrivbara register. Skrivning till register görs med instruktionen TO_M medan läsning från görs med FROM_M. De register som presenteras här är för att aktivera (enable) olika kanaler samt att

läsa in A/D-omvandlat värde samt skicka ut värde för D/A-omvandling. Utöver detta finns en del konfigurationsregister som förbises här.

I Q64AD-modulens register nr 0 används endast de fyra lägsta bitarna. Dessa har följande funktion:

- Bit 0 – 0/1 ⇒ enable / disable analog inkanal CH1
- Bit 1 – 0/1 ⇒ enable / disable analog inkanal CH2
- Bit 2 – 0/1 ⇒ enable / disable analog inkanal CH3
- Bit 3 – 0/1 ⇒ enable / disable analog inkanal CH4

För att överföra till Q64AD-modulen vilka kanaler som skall öppnas ges värden till TO_M-instruktionen enligt:

- s – decimala värdet som motsvarar de kanaler som skall enablas enligt ovan.
- n1 – positionsvärdet på den plats modulen är placera på bakplanet.
- n2 – det register nr som informationen skall läggas i (här 0)
- n3 – hur många register som skall överföras.

I register 11 i Q64AD-modulen finns det A/D-omvandlade värdet från CH1 att hämta, i register 12 hämtas CH2 osv. Instruktion för att hämta detta värde sker med FROM_M-funktion.

- d – Benämningen på D-register där det A/D-omvandlade resultatet läggs.
- n1 – positionsvärdet på den plats modulen är placera på bakplanet.
- n2 – det register nr som informationen skall hämtas från.
- n3 – hur många register som skall överföras.

I register 1 i Q64DA-modulen placeras det värde som skall D/A-omvandlas och hamnar som analogt utvärde CH1. Register 2 till CH2 osv. Instruktion för att lägga ut detta värde sker med TO_M-funktion.

- s – Benämningen på D-register som innehåller värdet för D/A-omvandling.
- n1 – positionsvärdet på den plats modulen är placera på bakplanet.
- n2 – det register nr som informationen skall läggas i .
- n3 – hur många register som skall överföras.

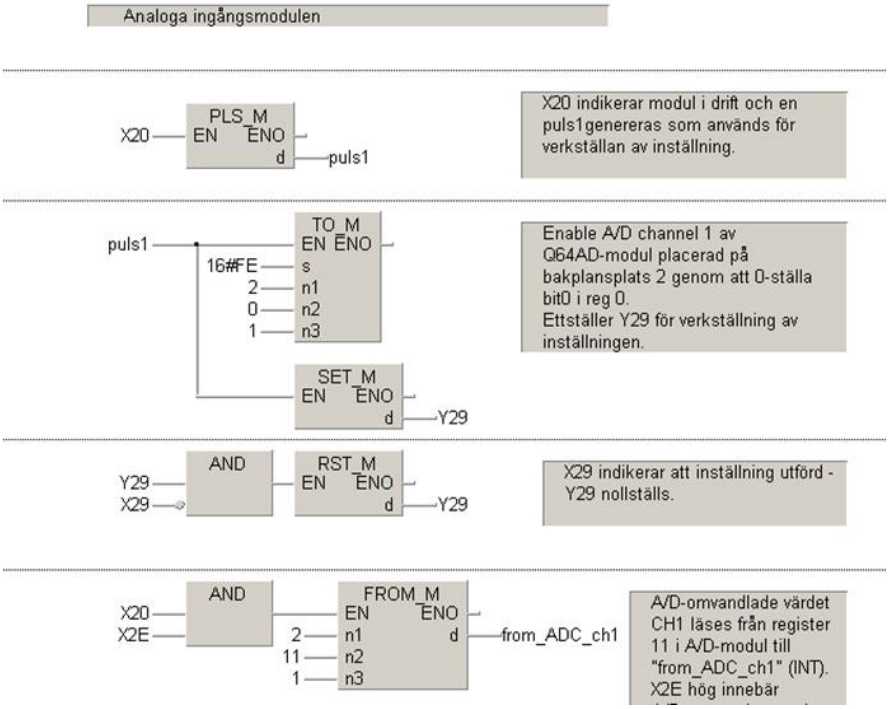
En ytterligare funktion hos Q64DA-modulen är att ytterligare en flagga måste aktiveras för att analoga utsignalen skall släppas ut på plinten. Denna flagga är Yx1 för CH1, Yx2 för CH2 osv. där x är modulens placering på bakplanet. Om denna flagga är nollställd kommer signalen 0 V alt 0 A ligga ut. Anledningen till detta är att man med säkerhetslogik enkelt

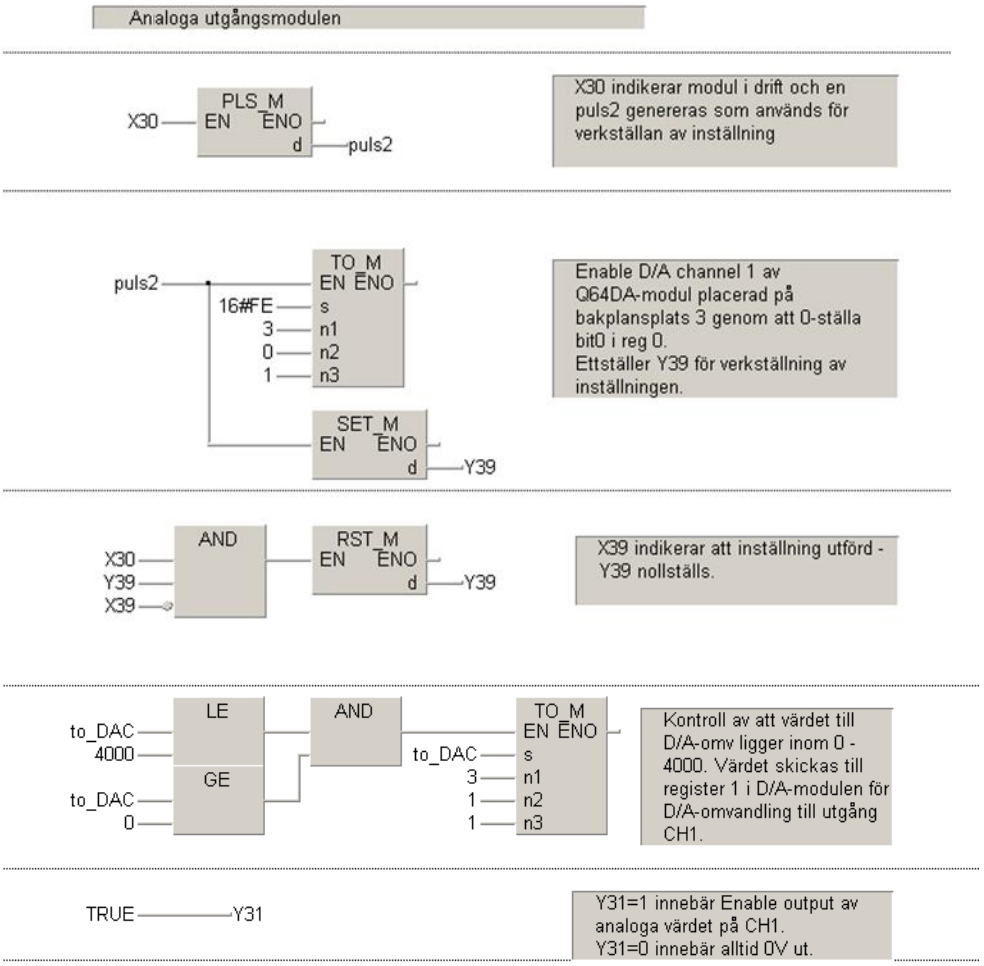
skall kunna nolla utsignalen som kan innebära t ex stoppa varvtalsstyrda fläkten, stänga ventilen e dyl.

Här följer ett exempel på detta.

Exempel 5.2:
Instruktionerna nedan gäller adressering av Q64AD-modul med 4 analoga kanaler in på modulplats 2 och en Q64DA med 4 analoga kanaler ut med placering modulplats på bakplanet.

n1, n2 och n3 i instruktionerna nedan anger modulplats, buffertadress i modulens minne respektive antal 16-bitars register som sänds alt. hämtas. Kommentarrutorna i respektive Network anger vad de olika blocken har för uppgift. Endast kanal 1 utnyttjas på de två modulerna.





5.9. A/D- och D/A-omvandling i Mitsubishisystem A1S.

För att kommunicera med omvärlden med analoga signaler utrustas PLC-systemet med moduler för just analoga signaler in och ut. Dessa moduler innehåller då A/D- och D/A-omvandlare och kommunikation med PLC:t sker via ett antal från CPU läs- eller skrivbara register i modulen. Till system A1S finns några alternativa moduler tillgängliga. Här presenteras dock modul ADA som har 2 analoga ingångar -10 till 10 V alternativt -20 till 20 mA med en maximal upplösning på 0,83 mV alternativt 3,33 μA. Vid denna upplösning är omvandlingstiden 3 ms/kanal. Upplösningen är inställbar om man vill ha snabbare omvandlingstid. Vidare har ADA-modulen en analog utgång med samma egenskaper som ingångarna vad gäller omfång, upplösning och omvandlingstid. Modulen har också mer avancerade funktioner i form av medelvärdesbildning och funktionsföljning. Det senare innebär att analoga utsignalen fås som funktion av de båda analoga ingångsvärdena.

Arbetsområdet hos in- och utgångarna är inställbart. (Hos de vi använder i labbet är både in och utgångar normalt kalibrerade för digitalt 0 - 4000 motsvarar analogt 0 - 10 V.) Kommunikationen med modulen kräver att modulens placering på bakplanet är känd. Fysiskt tar modulen upp en plats men adressmässigt tar den två modulplatser. (På labsystemen sitter modulerna i ordning CPU – 1 st digital in – 1 st digital ut - ADA vilket medför att ADA-modulen upptar modulplats 2.) För det enklaste användarfallet gäller att de

kanaler som skall användas enablas samt att analoga signaler som A/D-omvandlats kan läsas till systemet i digital form och att digitala värden kan läsas ut för D/A-omvandling och placeras på analoga utgången.

Som nämnts ovan har ADA-modulen ett antal från CPU:t läs- och/eller skrivbara register. Skrivning till register görs med instruktionen TO_M medan läsning från görs med FROM_M. De register som presenteras här är för att aktivera (enable) olika kanaler samt att läsa in A/D-omvandlat värde samt skicka ut värde för D/A-omvandling. Utöver detta finns en del konfigurationsregister som förbises här.

I ADA-modulens register nr 0 används endast de tre lägsta bitarna. Dessa har följande funktion:

- Bit 0 – enable / disable analog inkanal CH1
- Bit 1 – enable / disable analog inkanal CH2
- Bit 2 – enable / disable analog utkanal CH3

För att överföra till ADA-modulen vilka kanaler som skall öppnas ges värden till TO_M-instruktionen enligt:

- s – decimala värdet som motsvarar de kanaler som skall enablas enligt ovan.
- n1 – positionsvärdet på den plats modulen är placera på bakplanet. (här 2)
- n2 – det register nr som informationen skall läggas i (här 0)
- n3 – hur många register som skall överföras.

Register 10 i ADA-modulen är det register där värde läggs för D/A-omvandling och sedan läggas ut som analog signal på analoga utgången, CH3. Instruktion för överföring av detta värde sker också med TO_M-instrutionen enligt:

- s – Benämningen på D-register som innehåller värdet för D/A-omvandling.
- n1 – positionsvärdet på den plats modulen är placera på bakplanet.(här 2)
- n2 – det register nr som informationen skall läggas i (här 10)
- n3 – hur många register som skall överföras.

En ytterligare funktion hos ADA-modulen är att ytterligare en flagga måste aktiveras för att analoga utsignalen skall släppas ut på plinten. Denna flagga är Y30 om modulen är placerad på plats 2 (Y70 om placerad plats 6 osv). Om denna flagga nollställd kommer signalen 0 V alt 0 A ligga ut. Anledningen till detta är att man med säkerhetslogik enkelt skall kunna nolla utsignalen som kan innebära t ex stoppa varvtalsstyrda fläkten, stänga ventilen e dyl.

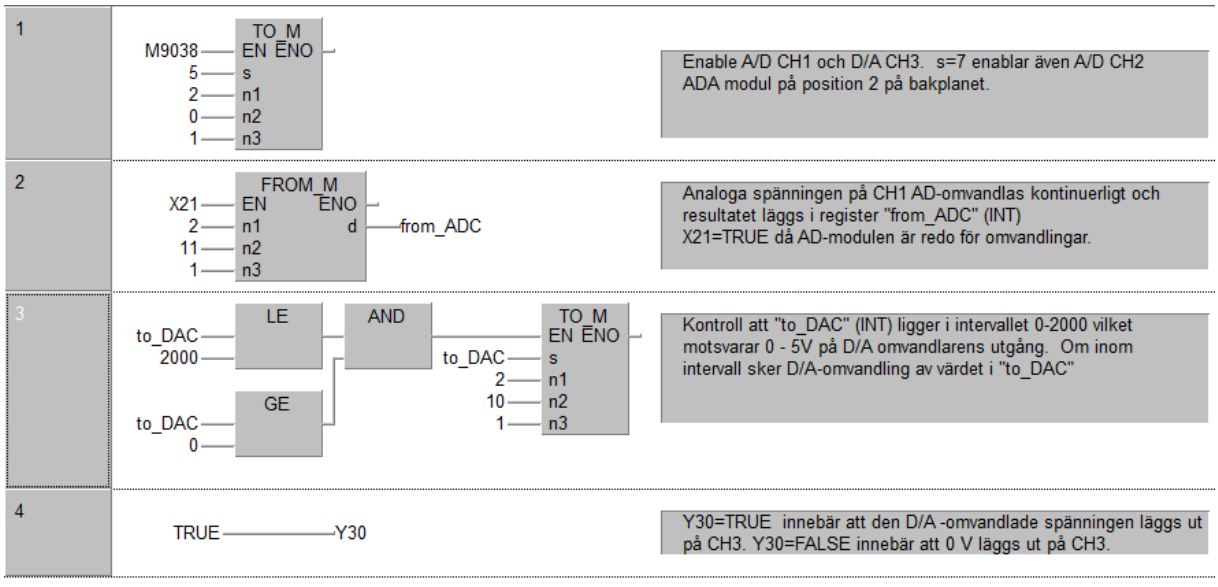
Register 11 är det register i ADA-modulen där det A/D-omvandlade värdet för analoga inkanalen CH1. Register 12 är motsvarande för CH2. A/D-omvandlingen sker kontinuerligt (konfigurerbart). A/D-conversion ready-flaggan är X21 (tvåan anger modulplats). Dessa register kan sedan läsas in till CPU:et med FROM_M-instruktion där :

- d – Benämningen på D-register där det A/D-omvandlade resultatet läggs.
- n1 – positionsvärdet på den plats modulen är placera på bakplanet.
- n2 – det register nr som informationen skall hämtas från (här 11 alt 12)
- n3 – hur många register som skall överföras.

Här följer ett exempel på detta.

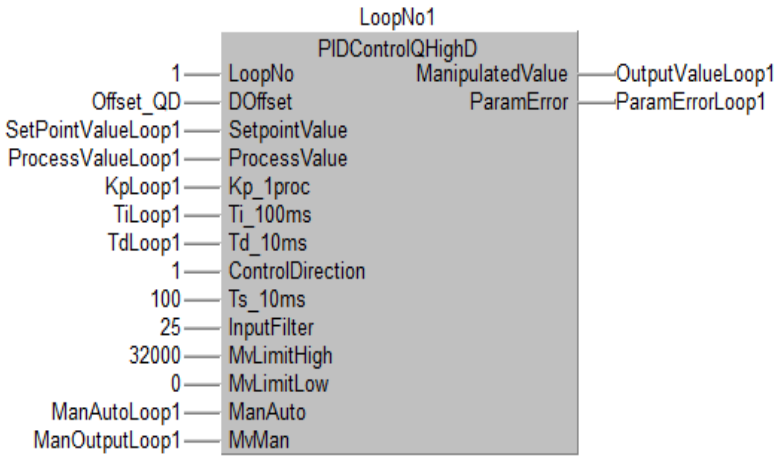
Exempel 5.3:
Instruktionerna nedan gäller adressering av ADA-modul med två analoga inkanaler och en analog utkanal med placering på modulplats 2 på bakplanet.

n1, n2 och n3 i instruktionerna nedan anger modulplats, buffertadress i modulens minne respektive antal 16-bitars register som sänds alt. hämtas. Kommentarrutorna i respektive Network anger vad de olika blocken har för uppgift.



5.10. PID-regulatorn i Q02-systemet.

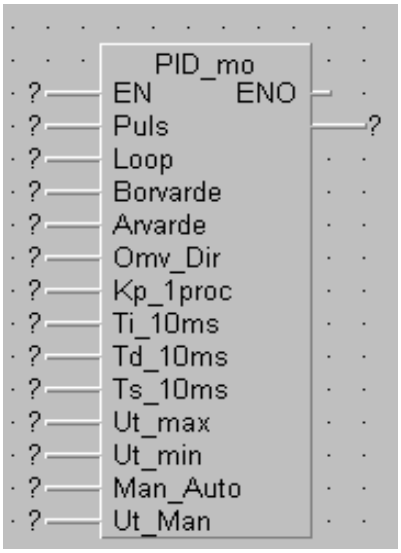
För Q-systemen finns följande funktionsblock för PID-reglering tillgängligt. PID-regulatorns allmänna funktion anses vara känd och därmed de olika ingångs- och utgångsparametrarna hos blocket.



SetPointValue	– Börvärde
ProcessValue	– Ärvärde – hämtas normalt från A/D-omvandlare
Kp	– proportionell förstärkning i %
Ti	– integrationstid i antal 100 ms
Td	– deriveringstid i antal 10ms
ControlDirection	– 1=omvänd, 0=direkt – där omvänd innebär ökande styrsignal vid ökande reglerfel (börvärde – ärvärde)
Ts	– samplingstid i antal 10 ms
MvLimit	– begränsning av styrsignal
ManAuto	– 1=manuell 0=automatik
MvMan	– styrvärde vid manuell inställning
ManipulatedValue	– styrsignal – skickas normalt till D/A-omvandling

5.11. PID-regulatorn i A1S-systemet.

I GX IEC Developer finns möjligheten att utveckla egna funktionsblock. Ett exempel på sådana är PID-regulatorblocket i A1S-systemet för reglering av återkopplade analoga processer.



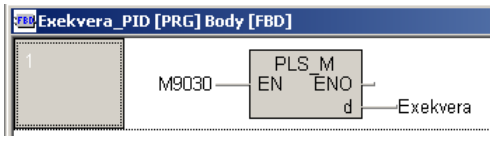
In-/utgång	Typ	Funktion
Puls	BOOL	Flanktriggad signal var 100:e ms, exekv.
Loop	INT	Löpnummer på reglerloop (1-30)
Borvarde	INT	Börvärde regulator (0-12000)
Arvarde	INT	Ärvärde regulator (0-12000)
Omv_Dir	DINT	Direkt(0) / Omvänd(1) funktion [1]
Kp_1proc	DINT	P-konstant i % ex. 100 %=1ggr; (1-100000 %)
Ti_100ms	DINT	Integr.tid ex. 100=10s; (0.1-3000s) [2]
Td_10ms	DINT	Deriv.tid ex. 100= 1s; (0.00-300s) [3]
Ts_10ms	DINT	Samplingstid ex. 100 = 1s; (0.01-60 s) (Dock, 0.1s minsta tekniskt möjliga!)
Ut_max	DINT	Högst önskade utsignal (0-12000)
Ut_min	DINT	Minst önskade utsignal (0-12000)
Man_Auto	INT	Manuell(1) / Auto(0) utsignal
Ut_Man	DINT	Manuell utsignal (0-12000)
Ut	INT	Utsignal (0-12000)

[1] Vid omvänd funktion ökar utsignalen vid ökande reglerfel (börvärde – ärvärde)
[2] Om ingen I-verkan önskas, skriv ett värde > 100000s.
[3] Om ingen D-verkan önskas, skriv värde 0s.

OBS!

1. Sätt av 2k filregister i parametrarna. R0-R1050 används av regulatorerna!! Ställ in detta i Navigatorns PLC_Parameters – MemoryParam...

2. För ingången Puls ovan, gör exempelvis ett program enligt nedan och anslut variabeln Exekvera till nämnda ingång!



5.12. Realtidsklockan.

För att kunna göra tidsstyrningar, dvs. starta en aktivitet ett visst klockslag, en viss dag eller en viss veckodag finns en inbyggd realtidsklocka som håller reda på år, månad, dag, timme, minut, sekund och veckodag. Klockan finns integrerad i PLC-systemets processor och kan läsas och ställas av PLC-programmet via följande register i Q02 (inom parantes i A1S).

SD210 (D9025)	-	år och månad
SD211 (D9026)	-	dag och timma
SD212 (D9027)	-	minut och sekund
SD213 (D9028)	-	veckodag

I de register som innehåller dubbel information t.ex D9025 ligger år i de 8 högsta bitarna och månad i de 8 lägsta bitarna i registret. Data är lagrade i BCD-kod (Binary Coded Decimal) vilket görs för att lätt kunna lägga ut dem till t.ex en display. Skall de användas för tidsstyrning i programmet är det dock lämpligt att ha värdena som heltal varför omvandling till INT-typ är att föredra. Stöd för detta finns i instruktioner som BCD_TO_INT respektive INT_TO_BCD.

Vad gäller veckodag är de numrerade enligt 0=söndag, …6=lördag. Veckodagen ges med fyra siffror där de två högsta avser århundrade. Exemelvis värdet 2005 innebär en fredag på 2000-talet.

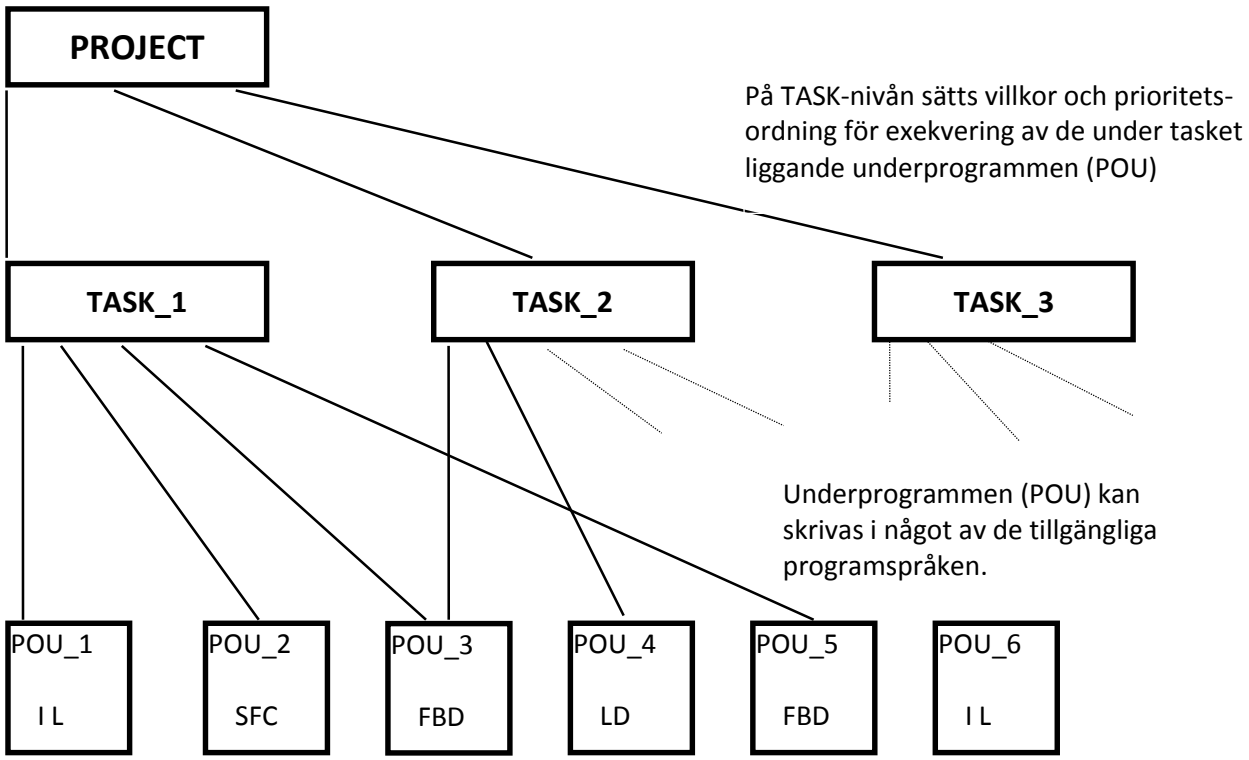
Två stycken specialminnesceller används för att kommunicera SD210-SD213 (D9025 - D9028) med realtidsklockan. SM210 (M9025) används för att lägga in data i realtidsklockan dvs för att ställa den. SM213 (M9028) används för att läsa klockan dvs när denna flagga aktiveras kommer aktuella tidsdata att läggas i SD210-SD213 (D9025 - D9028).

Kap 6. Utvecklingsmiljön enligt standard IEC 61131-3.

Som redan nämnts finns numera en standard, IEC 61131, som skall göra tillvaron lättare när det gäller att programmera PLC av olika fabrikat. Mitsubishi i Tyskland har utvecklat en editor, GX IEC Developer som stödjer denna standard men också kompletterat den med en mängd instruktioner utöver standarden vilka har sitt ursprung i Mitsubishis tidigare programeditor.

Av IEC 61131:s fem alternativa programmeringssätt är alla tillgängliga i nuvarande version av GX IEC Developer nämligen ladderdiagram (LD), funktionsblock (FBD), funktionsdiagram (SFC=Sequential Function Chart), strukturerad text (ST) samt instruktionslista (IL). Programmeringssättet enligt denna standard håller alltså dörrarna öppna för alla de sätt som PLC programmerats på genom tiderna. Miljön vill dock uppmuntra till ett strukturerat sätt att programmera varför vi här innan vi går in på de olika instruktionssätten först bekantar oss med hur ett projekt struktureras.

6.1. Programstrukturen i GX IEC Developer.



Figur 6.1: Programstrukturen i GX IEC Developer.

Hela styrlösningen samlas under ett **project** och arbetar med ett bibliotek för varje project. Själva programfilen heter detsamma i alla project (softctrl.pro) varför uppmärksamhet krävs vid backup av project. Strukturen hos ett project framgår av Figur 6.1 ovan.

Som framgår av Figur 6.1 ovan så kan ett projekt delas upp i en eller flera delar som benämns TASK. För dessa olika Task kan man styra under vilket villkor och med vilken prioritet de skall exekveras. De olika exekveringsvillkor (event) som kan användas är:

- kontinuerlig exekvering – d v s så fort processorn hinner.
- händelse (dvs yttre händelse som kommer in via ingång eller händelse som upptäcks via exekvering av annat task)
- med jämna tidsintervall
- interrupthändelse som genereras via de olika interruptpekarna I1-I31.(se nedan)

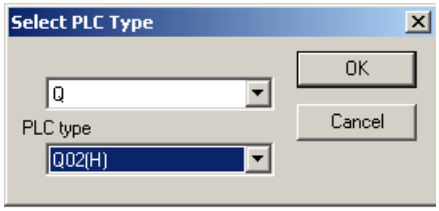
Exekveringsvillkoret ställs in genom att markera aktuellt Task i Navigator och gå till Object – Information.

Om olika Task har samma exekveringsvillkor kan exekveringsordningen styras via prioritet 0 - 31 där 0 innebär högsta prioritet. Mera om exekveringsordning finns i avsnitt 6.6.

Varje Task består i sin tur av ett eller flera POU (Program Organisation Unit) som skrivs i något av de fyra tillgängliga programmeringssätten instruktionslista (IL), ladderdiagram (LD), funktionsblock (FBD) eller funktionsdiagram (SFC).

6.2. Skapa project.

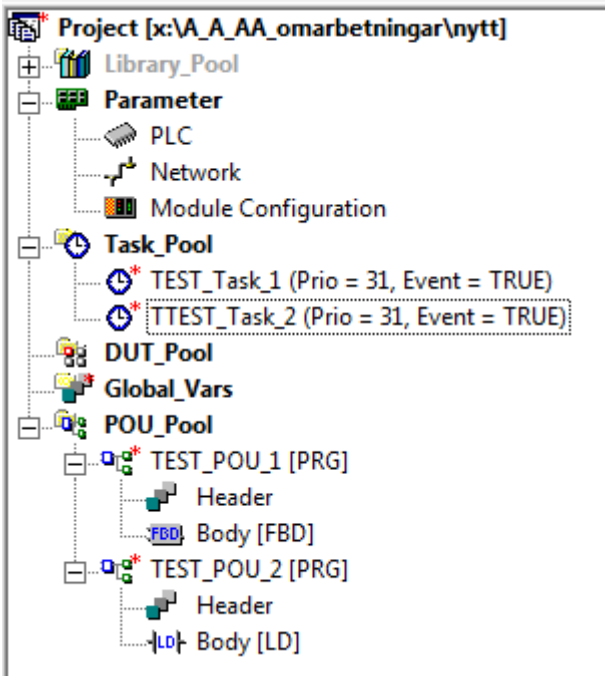
Välj *New* under menyn *Project*. Då kommer detta fönster upp:



Välj enligt figur om Q02-system skall användas eller AnS – A1S om A1S-system skall användas. Välj destination var projektet skall sparas. Välj Empty Project för att få börja med ett helt tomt projekt. Menyn för hantering av editorn ligger nu på en list i överkant medan *Project Navigator* dyker nu upp vid vänsterkanten. Resten är arbetsyta för programskapande.

6.3. Navigatorn.

I *Project Navigator* läggs nu gången upp hur man tar fram en styrlösning. I navigator enligt figur nedan är redan inlagt två stycken Tasks och två stycken POU i Task_Pool respektive POU_Pool.



Library_Pool innehåller biblioteket av instruktioner och funktionsblock som finns tillgängliga i systemet. Det är dock lättare att söka dem på andra vägar vilket framgår senare. Här kan också egenutvecklade funktionsblock placeras, vilket behandlas senare.

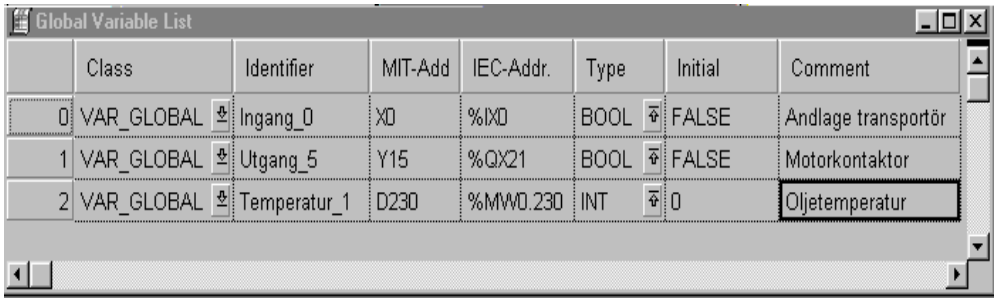
DUT_Pool (Data Unit Type) kan utnyttjas för att deklarera en global variabelgrupp som görs för förenklad upprepad användning vid styrning av flera likadant uppbyggda processdelar. För närmare förklaring hänvisas till manual.

Nu återstår tre nivåer att presentera i navigatorn nämligen *Global_Vars*, *POU_Pool* och *Task_Pool*. Dessa är de centrala för att man efter deklaration av systemet skall kunna åstadkomma ett styrprogram.

6.4. Globala variabler.

De globala variablerna deklareras alltid som VAR_GLOBAL och en benämning ges mot en absolut adress i PLC-minnet eller mot en in- eller utgång till systemet. De globala variablerna gäller för hela projektet och kan nås från olika POU (underprogram) och gör det möjligt att utbyta data mellan olika POU och TASK.

Ett exempel på global variabellista visas nedan. I IEC-standarden har man föreskrivit att in- och utgångar, minnen, register m.m skall ha en viss adressbenämning. Användare av olika fabrikat är dock inarbetade på andra fabrikatsspecifika adressbenämningar varför man i denna editor kan använda IEC-adress alternativt Mitsubishi-adress. Skriver man den ena så ges den andra. Tidigare i denna skrift är det IEC-adresser som presenterats i beskrivningen av operanderna i avsnitt 3.2.1. Variabler, både lokala och globala, kan skapas löpande under programkonstruktionen och placeras då in i dessa listor.



Typ av variabel deklareras i *Type* där man kan välja mellan BOOL, INT, DINT, WORD, DWORD, REAL och ARRAY där en array kan bestå av 1, 2 eller 3 dimensioner av data. INT är ett 16-bitars register som kan anta värden från -32768 till +32767. DINT är ett 32-bitars dito. WORD är en 16-bitars sträng som kan anta värden 0 till 65535. Observera att många registerhanterande instruktioner bara klarar datatyperna INT och DINT.

I *Initial* skall man enligt standarden kunna ge ett initialvärde för de olika variablerna vid programladdning. GX IEC stödjer dock inte denna facilitet. Initialvärden är alltså alltid FALSE för boolska variabler och 0 för register. Vill man tilldela andra initialvärden kan detta göras med hjälp av %MX10.402 eller SM402 (Q02) alt. M9038 (A1S) som är en specialflagga som är ettställd endast första scan-cykeln efter exekveringsstart.

Det finns i GX IEC möjlighet att smita förbi den globala variabellistan. De adresser, som inte är lämnade till systemet, är globala och kan anropas direkt utan att vara upptagna i globala variabellistan. Vilka adresser som finns tillgängliga går att finna under PLC i Navigatorn. Med adresser menas IEC-adresser av typ %MX0.89, %IX12, %QX23, %MW0.567 eller som motsvarande MIT-adresser av typ M89, XC, Y18, D567. Detta är dock inte att rekommendera då globala variabellistans syfte är att ge mer processnära namn till de olika variablerna för att därmed göra programkoden mera lättolkad och i mindre behov av kommentarer.

6.5. Skapa delprogram POU.

Under menyraden finns en rad med verktygsikoner varav en är märkt POU. För att skapa ett nytt POU tryck på denna och en dialogruta kommer fram där man anger vad man vill döpa POU:t till och sedan välja i vilken form av de fyra möjliga man vill skriva underprogramet (*Body*). Det uppträder nu två underrubriker till POU i navigatorn nämligen *Header* och *Body*.

6.5.1. Lokala variabellistan (Header).

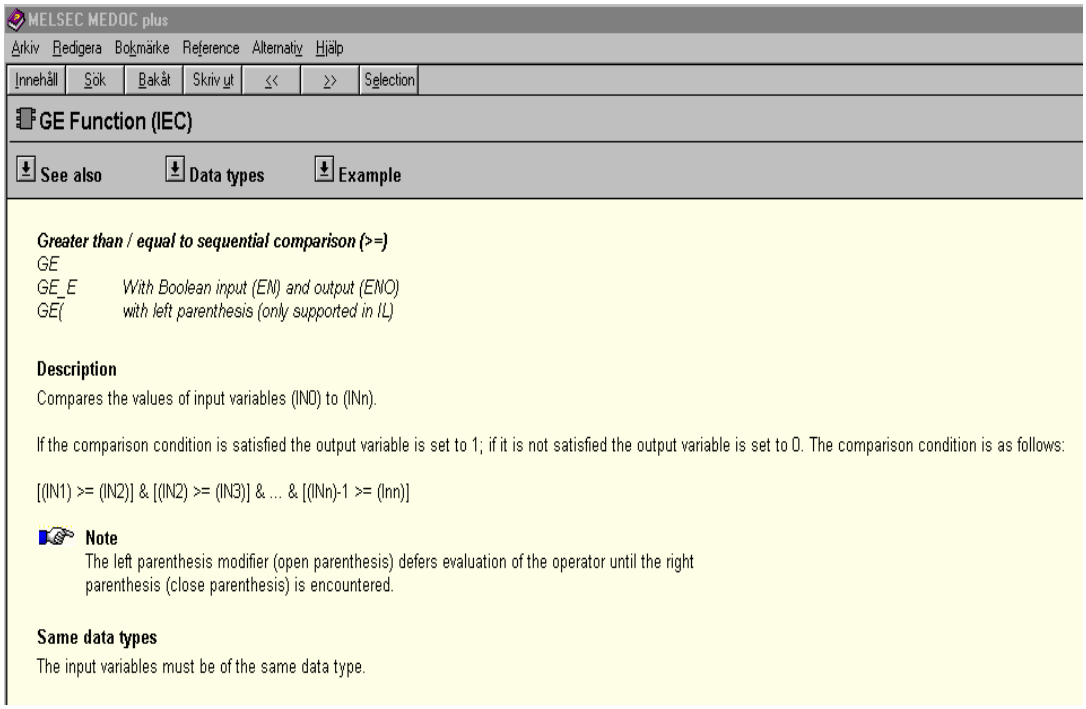
Lokalt använda variabler i detta POU skall vara deklarerade i dess Header-lista. Dubbelklicka på Header under aktuellt POU i Navigatorn för att skapa denna lista för lokala variabler. Dessa skall inte knytas till någon adress utan systemet väljer själv ur de variabler som är lämnade till systemet enligt tidigare beskrivning. Variabler, både lokala och globala, kan skapas löpande under programkonstruktionen och placeras då in i dessa listor.

6.5.2. Instruktionslista (IL).

Går man sedan in i Body under aktuellt POU hamnar man i en fri editor vilket innebär att man kan skriva in önskad kod i vilken Windowseditor som helst och sedan kopiera in programmet via klippbordet.

Det mörka fältet längst till vänster indikerar något som kallas *Network*. När man skriver instruktionslista kan man skriva hela programmet i ett Network. Nytt Network krävs bara vid hopp i programmet då hoppet sker till ett nytt Network försett med Label som utgör aktuell pekare. Dubbelklicka på Network och skriv in aktuell pekare.

Själva programmet utgörs av två kolumner. Den första utgör instruktioner följt av andra kolumnens operander som finns deklarerade i headern. Med F2 får man upp en lista med tillåtna instruktioner. Hjälpfunktionen i denna dialogruta är lite konstig varför det är bättre att gå in i hjälp via Help och Overview i huvudmenyn. I hjälpen finner man bl.a alla instruktioner förklarade med hur de fungerar och vilka variabeltyper som kan användas. Exempel visas nedan.



Programmet exekverar som en bit-ackumulator vilket innebär att resultatet av en instruktion lagras direkt efter exekvering i ackumulatorn. I ackumulatorn finns alltså alltid resultatet av föregående instruktion.

I programlistan kan kommentarer skrivas in antingen i tredje kolumnen eller på ny rad. Kommentarerna skall börja med (* och sluta med *).

En instruktionslista kan se ut som följer. Observera att alla använda operatorer inte är deklarerade i Headern avsnittet ovan.

UNDER_1 [PRG] Body [IL]

```
(*OBS! Alla operanderna är inte deklarerade i Headern i avsnitt 11.5.5.1 *)
LD      Ingang_0    (*Logiskt villkor          *)
AND     Minne_1     (*Utgang_5=Ingang_0 and Minne_1  *)
ST      Utgang_5

LD      Ingang_1    (* Ingang_1 and invers Ingang_2  *)
ANI     Ingang_2
S       Minne_6     (* sätter Minne_6              *)

LD      Ingang_7    (* Ingang_7 and (Minne_1 or Utgang_8)  *)
AND(    Minne_1
OR      Utgang_8
)
R       Minne_6     (*  resetar Minne_6    *)
```

Instruktionslistan är användbar och snabb för att skriva in enkel logik. Då man kommer in på funktioner som timers, räknare, jämförare osv är det enklare och klarare att använda ladderdiagram eller funktionsblock.

De vanligaste tillgängliga instruktionerna är LD, AND, ANI, OR, ORI, ST, STN, S, R.

6.5.3. Ladderdiagram eller Relälista (LD).

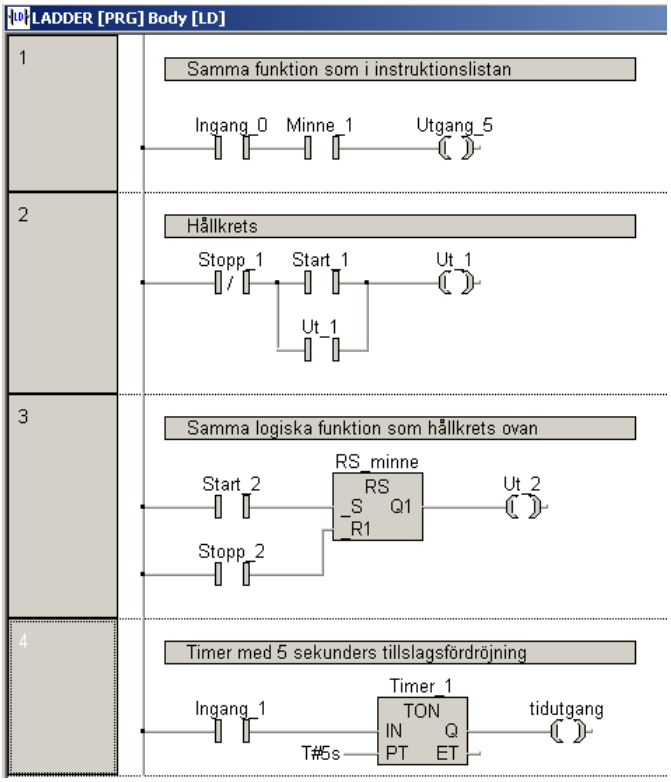
Ladderdiagrammet eller relälistan är en grafisk beskrivning av styrlogiken som är uppbyggd enligt elschemaritning av relälogik. Relälogik var det sätt man löste styrningar med före elektronikens inträde på området. Eftersom traditionen och yrkeskunskapen från relätiden har flyttats över på moderna programmerbara system har detta resulterat i att man räknar med att 70 % - 80 % av all PLC-kod som finns är skriven i ladderform. Med nya kanske mer tilltalande sätt att programmera så är det ändå viktigt att känna till ladderkoden och kunna tolka den då man skall gå in i redan befintlig programdokumentation.

Som framgår av exemplet nedan kan man nu lägga in mer avancerade funktioner i ladder-koden än vad man kunde realisera med reläteknik tidigare. I ladderdiagrammet kan man enligt IEC 61131 lägga in samma funktionsblock som vid programmering just med funktionsblock (FBD), se nästa avsnitt.

Laddereditorn är en grafisk editor där kontakter och spolar fritt kan placeras, flyttas och kopieras mellan olika *Networks.* **Observera att endast en krets får finnas i varje Network.** Ett nytt Network öppnas ovanför eller under det aktiverade med två alternativa ikoner i verktygsfältet.

För att negera en kontakt dvs ändra från NO till NC dubbelklicka på kontakten för att få dialogruta. För att välja funktionsblock så välj verktygsikon med ”IC-krets-symbol”.

Förbindelser mellan komponenterna fås genom att högerklicka och välja Interconnect Mode. Klicka på utgång, flytta musmarkör till ingång och klicka. För att ansluta signal till funktionsblock använd verktygsikon VAR- och -VAR beroende på in- eller utgång. Kommentarer läggs in genom att aktivera verktygsikon med pratbubbla och rita kommentarruta.



6.5.4. Funktionsblock (FBD).

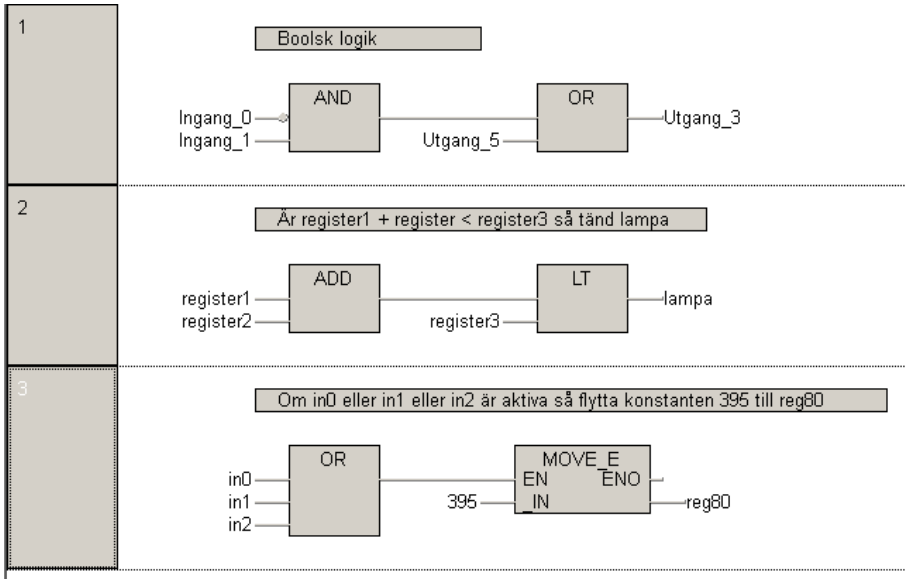
Funktionsblockeditorn är uppbyggd och används på samma sätt som ladderdiagrameditorn. Skillnaden är att booleska logiken byggs upp med logiska block istället för trådade kontakter och enskilda ingångar och utgångar ansluts med enbart signalbeteckning i stället för med kontaktsymbol respektive spolsymbol. För övrigt gäller alltså en krets per Network, funktionsblockval med IC-kretsikon, anslutning med VAR-ikon, pratbubbla för kommentarruta. Förbindelser mellan komponenterna fås genom att högerklicka och välja Interconnect Mode.

Invertering av in- eller utgång görs genom att dubbelklicka på anslutningen till blocket.

Vissa *functions*, som har benämning som slutar _E, har längst upp en ingång märkt EN (enable). Detta block exekveras endast om signalen till EN är TRUE. På dessa block finns också en utgång ENO vilken slaviskt följer signalen till EN. Denna utsignal kan användas till eventuellt efterföljande blocks EN-ingång.

I biblioteket över functions finns dels Standard_Lib och dels Manufacturers_Lib. Standard_Lib innehåller de funktioner som skall finnas enligt IEC-standarden. Tillverkaren (i detta fall Mitsubishi) är dock intresserad att ha kvar sina ”gamla” funktioner som dess programmerare är vana att hantera varför man har kompletterat med ett Manufacturers_Lib. Dessa funktioner kan ofta vara användbara och är väl dokumenterade i Help-manualer. En del av dem presenteras i avsnitt 5.2.

Här några exempel på kretsar i funktionsblockform vilka utgör ett POU:



Observera att exekveringen av Networken sker uppifrån och ner, stannar aldrig upp och väntar på något, utan upprepas cykliskt i den takt som styrs av exekveringsvillkoret för det Task som POU:et tillhör.

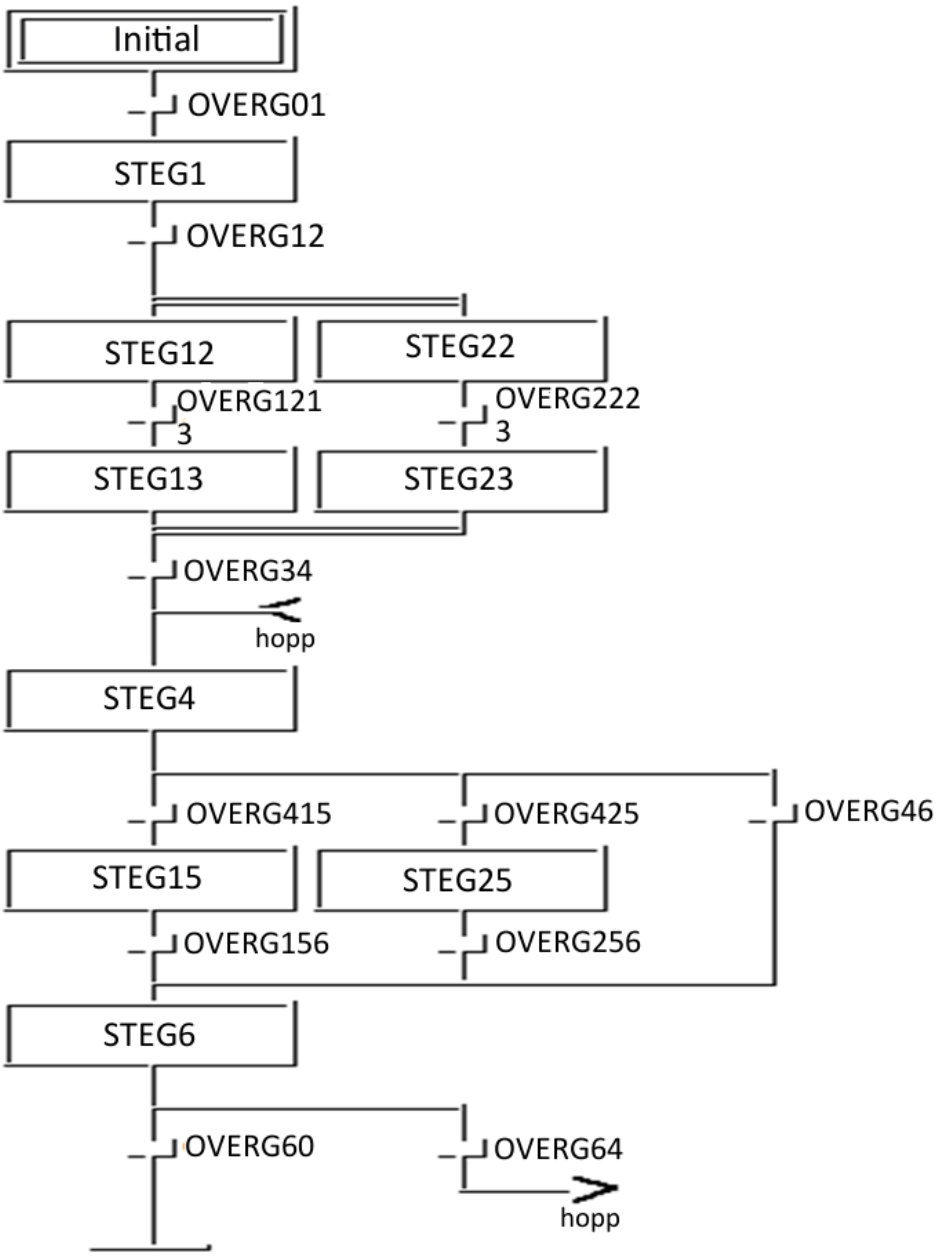
6.5.5. Funktionsdiagram eller Grafcet (SFC).

För att beskriva sekventiella förlopp finns numera en standardiserad form (IEC 848) kallad funktionsdiagram som är beskriven tidigare i kapitlet ”Funktionsbeskrivningar”. Funktionsdiagramformen går också under namnet Grafcet.

Det är naturligt att funktionsdiagrammets lättbegripliga presentation av en problemlösning är lämplig att använda för att programmera styrlösningar till sekventiella förlopp. Därför blev en av de standardiserade programmeringsformerna för PLC just funktionsdiagrammet. Benämningen på detta språk är Sequence Function Chart (SFC) och en programeditor för detta språk finns i GX IEC Developer. Denna editor är en fast grafisk editor som följer ett visst fast mönster. När ett POU skapas som skall programmeras i SFC finner man att i Navigatorn uppträder tre underrubriker, Header och Body är kända sedan tidigare men nu finns också Action. I Body bygger man upp själva funktionsdiagramstrukturen medan man i olika Actions lagrar de händelser som skall ske i de olika stegen.

Nedan visas en SFC-body där namn har satts på de olika stegen och övergångsvillkoren. Grafiskt byggs diagrammet upp med musen genom att aktivera grunddiagrammet på ”rätt” ställe och sedan lägga till steg, övergångsvillkor, parallella och alternativa förgreningar med hjälp av verktygsikonerna. Bättre än att försöka förklara varje steg är att uppmana användaren att testa sig fram. Avslutningen med ett tvärstreck efter OVERG60 innebär återhopp till startsteget (Initial).

I SFC-programexemplet nedan visas dels en parallellförgrening och senare i sekvensen en alternativförgrening. Möjligheter finns också till återhopp uppåt i funktionsstegen som framgår av ”hopp” i figuren.



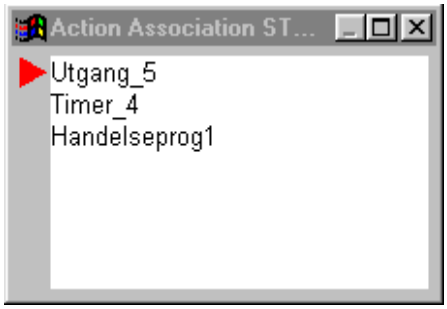
SFC-programmet översätts till en vanlig instruktionslista vid kompileringen. Det innebär att ett SFC-POU exekveras på samma sätt som andra. Att förloppet befinner sig i ett visst steg innebär inte att exekveringen stannar upp i väntan på att aktuellt övergångsvillkor skall uppfyllas utan exekveringen av alla andra i projektet ingående POU:n fortlöper kontinuerligt.

Som synes i funktionsdiagramexemplet ovan kan också hopp utföras i förloppet. Undvik dock enligt god programmeringssed att gör hopp eftersom det ofta minskar programmets läsbarhet.

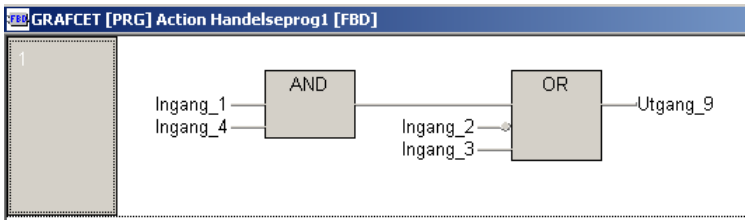
Händelser:

Varje steg i programmet har tilldelats ett namn som inte skall deklareras i den lokala variabellistan. Till varje steg kan man knyta händelser eller om händelsen är villkorad ett PLC-program som kan skrivas i valfri editor. Det görs på följande sätt:

 Klicka t ex på STEG1 vilket gör den rutan aktiverad vilket visas genom att en svart ram uppträder kring STEG1. Tryck den ikon som visas här intill. Då framträder Action-rutan enligt nedan där tre rader skrivs in så att direktaktiveringen av en händelse (variabler), Utgang_1 sker samt aktivering av två PLC-program, Timer_4 och Handelseprog1 .

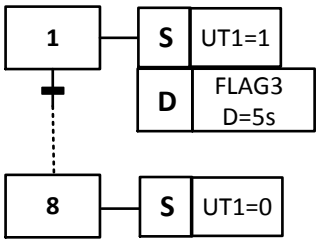


 Med markören placerad på Handelseprog1 i Action-rutan aktiveras ikon som visas här intill. Då ges möjlighet att välja PLC-editor och skriva in sin villkorade händelse för steg1. Denna händelse kan se ut enligt nedan. Tittar man i Navigatorn upptäcker man nu att Handelseprog1 har hamnat i Action Pool i POU:t Grafcet. Observera att för att i detta fall Utgang_9 skall aktiveras måste det logiska villkoret Action Handelseprog1 vara uppfyllt samt förloppet befinna sig i STEG1 som denna Action är kopplad (associerad) till. Händelserna som är deklarerade i ett steg exekveras endast om förloppet befinner sig i det steget.



När det gäller händelser som är bestyckade med modifierare av typ D och S d v s Delay av händelse resp ”Stored” händelse kan de hanteras på olika sätt. Vi betraktar några olika lösningar i följande exempel.

Exempel 6.1:
Skriv SFC-programlösning för följande händelser som här beskrivs på funktionsdiagramform.

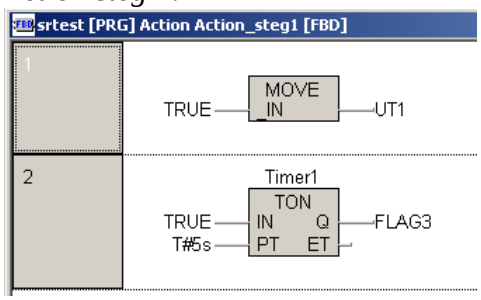


Lösning alternativ 1:

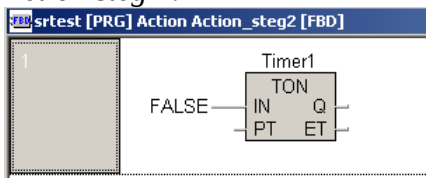
Lösning med IEC-standardinstruktioner där Actions skrivs i FBD. Notera att i detta fall behövs ett steg 2 som följer direkt på steg 1. Förklaring till detta följer efter figurerna. För minnesfunktionen används instruktionen MOVE för att till UT1 lägga värdet TRUE i steg1 respektive FALSE i steg 8.

Följande Actions deklareras i steg1, steg 2 respektive steg 8.

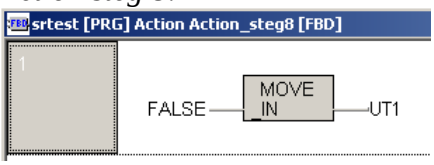
Action steg 1:



Action steg 2:



Action steg 8:

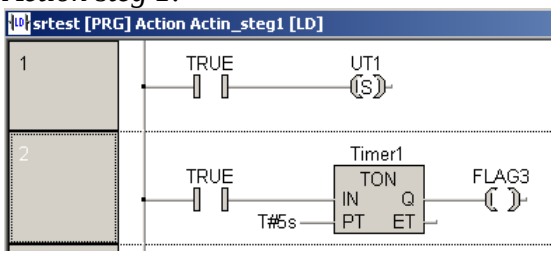


Respektive Action exekveras först när man befinner sig i steget vilket innebär att UT1 ettställs då steg 1 nås och förblir ettställt tills steg 8 nås då den nollställs. I steg 1 aktiveras Timer1. Om förloppet dröjer kvar i steg 1 mer än 5 sekunder kommer FLAG3 att ettställas. Lämnas nu steg 1 kommer Timer1 inte att exekveras mer men för rätt funktion krävs att Timer1 nollställs i nästa direkt följande steg (steg 2) varför steg 2 måste förses med en kopia av Timer1 men med FALSE på ingången varvid denna exekveras och timerns tidräknare nollställs.

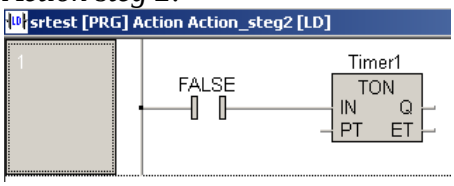
Lösning alternativ 2:

Alternativ lösning med IEC-standardinstruktioner där Actions skrivs i LD där tillgång finns till Set- resp Reset-instruktioner som är separerade. Vad gäller tidsfunktionen är denna lösning identisk med lösningsalternativ 1.
Följande Actions deklareras i steg1, steg2 respektive steg 8.

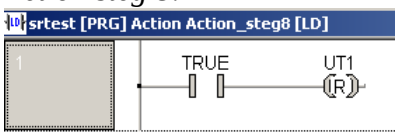
Action steg 1:



Action steg 2:



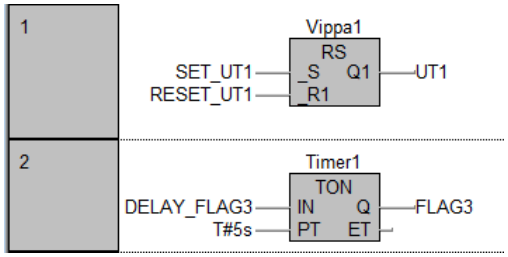
Action steg 8:



Lösning alternativ 3:

Fortfarande lösning med IEC-block men nu används ett FBD-POU som mål för händelseflaggor som aktiveras i olika Actions. Denna metod tillämpas konsekvent då stora system programmeras där funktionsblock skapas för de olika objekten som ingår i systemet.

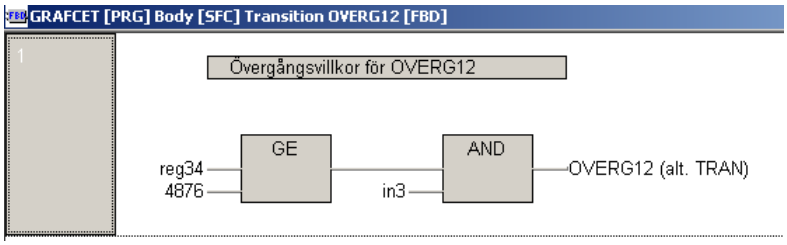
I SFC-programmet aktiveras i steg 1 en BOOL- flagga SET_UT1 och en DELAY_FLAG3 och i steg 8 en flagga RESET_UT1. Dessa flaggor deklareras som Globala variabler eftersom de skall verka över POU-gränser. Sedan skaps ett nytt POU med programspråk FBD, I detta läggs två networks för modifierad styrning av UT1 och FLAG3 som påverkas ifrån SFC-programmet:



Övergångsvillkor:

Mellan två på varandra följande steg finns ett övergångsvillkor. Den benämning man satt på övergångsvillkoret är namnet på det PLC-program som skrivs i valfri editor. För att komma till nästa steg skall man skriva ett program som aktiverar en utgång med den unika benämningen *TRAN* eller med samma namn som övergångsprogrammet.
Tillvägagångssättet är detsamma som villkorad händelse enligt ovan. Aktivera det aktuella övergångsvillkoret och tryck ikonknapp  så kommer dialogruta för *New Transition* där val av editor görs.

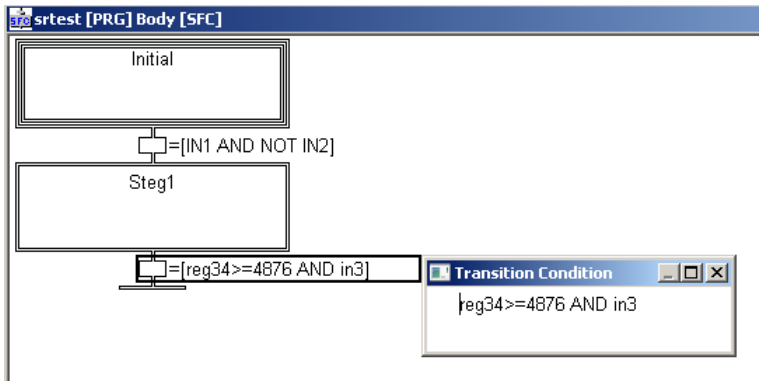
Nedan visas ett exempel för OVERG12.



Om förloppet befinner sig i STEG1 och övergångsvillkoret (transition) OVERG12 är uppfyllt kommer förloppet att hamna i STEG12 och STEG22.

En brist finns i programvaran som gör att om övergångsvillkoret är allt för omfattande (flera ingångar och grindar) så vägrar kompilatorn och ger felmeddelandet ”Only one OUT instruction is allowed in a transition for the current CPU type”. I detta läget tvingas man skapa ett eget POU för övergångsvillkoret och använda resultatvariabeln som övergångsvillkor. Med detta förfarande är kompilatorn åter medgörlig.

Ett alternativt sätt att skriva övergångsvillkor är att markera övergången och i menyn välja Tools – Edit Transition Condition och i den skriva in övergångsvillkoret i strukturerad text-form. Detta kan se ut enligt:



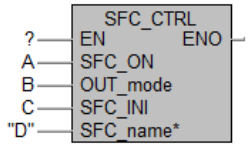
Man formulerar övergångsvillkoret på samma sätt som if-satser i andra högnivåspråk d v s resultatet ska vara antingen TRUE eller FALSE. Använd parenteser så att det framgår klart i vilken ordning operationerna ska utföras.

Följande operatorer kan användas:

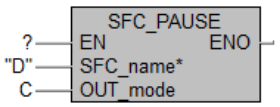
Priority level	Operation	Operator symbol	Comments
1	Brackets	( )	
2	Function call	Fun( )	
3	Exponentiation	**	
4	Negation Complement	- NOT	Unary minus
5	Multiplication Division Modulo	* / MOD	Remainder of integer division
6	Addition Subtraction	+ -	Binary minus
7	Comparison	<, >, <=, >=	
8	Equality Inequality	= < >	
9	Logical AND	AND, &	
10	Logical Exclusive OR	XOR	
11	Logical OR	OR	

Master Control för SFC:

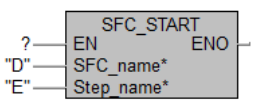
Önskemål kan finnas att kunna stoppa sekvensen var som helst i förloppet. För detta ändamål finns ett antal funktionsblock tillgängliga för ”Master Control” över en SFC-slinga. Dessa och deras funktion anges nedan.



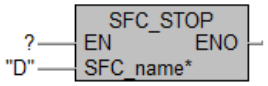
För SFC_CTRL gäller att om A=TRUE så exekveras SFC-POU:et med namn ”D” normalt. Om A=FALSE och B=FALSE kommer förloppet att stoppas i aktuellt steg (PAUSED) och alla out-aktiviteter att nollställas. Om A=FALSE och B=TRUE kommer de däremot att behålla sitt värde. C=TRUE medför reset av POU:et och återgång till Initial-steget.



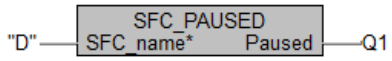
För SFC_PAUSE gäller att om EN =TRUE kommer förloppet att stoppas i aktuellt steg (PAUSED). Om C=FALSE kommer alla out-aktiviteter att nollställas medan om C=TRUE kommer de att behålla sitt värde.



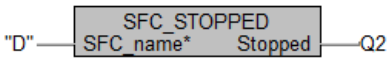
För SFC_START gäller PAUSE:ad sekvens enligt ovan startar igen on EN = TRUE och då från steg med benämning ”E”.



För SFC_STOPP gäller SFC ”D” stoppas, out-aktiviteter nollställs och reset ger återgång till Initial-steg.



Flagga Q1 ettställs då SFC ”D” pausats med SFC_PAUSE enligt ovan. Detta kan användas för att flagga av att denna master control gått in.



Flagga Q2 ettställs då SFC ”D” stoppats SFC_STOP enligt ovan. Detta kan användas för att flagga av att denna master control gått in.

6.5.6. Strukturerad text.

Det femte möjliga programspråket Structured Text, ST, är ett textbaserat språk liknande C och Pascal och är lämpligt för bl a beräkningsrutiner. Detta språk behandlas dock inte djupare här.

6.5.7. Kontroll av inskriven kod.

 Med denna knapp kan kontroll av program och benämningslistor göras. Kontrollen innebär att aktiverat fönster kontrolleras av kompilatorn och eventuella felmeddelanden rapporteras. Samtidigt sparas de ändringar som gjorts sedan föregående kontroll eller sedan fönstret aktiverades. Det senare sker även då fönstret stängs. Kontrollen innebär dock endast kontroll av syntaxen så någon kompilering sker inte.

6.6. Skapa TASK och kompilera projektet.

Ett projekt består av ett eller flera Tasks. Till ett Task knyts ett eller flera delprogram (POU). Aktivera Navigatorfönstret och klicka sedan på knapp märkt TSK i verktygsikonraden. Ge detta nya TASK ett namn och dubbelklick sedan på detta namn i Navigatorn. Ett fönster kommer då upp där man kan skapa en lista på de POU som skall ingå i Tasket. Exekveringen av de olika POU:na kommer nu att ske i den ordning som de läggs i listan.

Ett Task kan sedan exekveras på olika villkor. Markera aktuellt Task i Navigatorn, välj sedan Objekt i menyn och där Information. Där kan villkor för exekvering ställas in enligt:

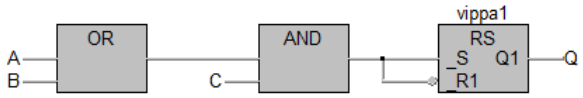
3. Varje programvarv. Sätt Event till TRUE.

4. På händelse. Sätt Event till I/O-adressen eller benämningen på det villkor som skall aktivera exekveringen.

5. Tidsintervall. Sätt Event till FALSE samt Interval till det önskade tidsintervallet mellan programexekveringsstarterna. Denna tid måste vara längre än scancykeltiden.

Vid icke kontinuerlig exekvering enligt pkt 2-3 ovan, dvs då exekveringsvillkoret för ett Task inte är TRUE (kontinuerlig exekvering) utan utförs på händelse eller med jämna tidsintervall uppför sig systemet något underligt. Möjligheten finns där för att lasta av centralenheten men för övrigt är avsikten att styrlogiken skall fungera normalt. Den funktion som fanns tillgänglig för att realisera denna IEC 61131-facilitet i Mitsubishi-systemet var en Master Control-funktion som frikopplade utvalda delar av programkoden från exekvering. Det innebar att då faciliteten icke kontinuerlig exekvering tillämpas så faller alla M-flaggor (%MX-flaggor) och Y-utgångar (%QX-utgångar), som inte är styrda med SET/RESET-instruktion, så fort den tids- eller villkorsstyrda exekveringen utförts. Det innebär att om diskontinuerlig exekvering skall tillämpas så måste alla kombinatoriska uttryck låsas med SET och RESET för att förväntad funktion skall uppnås.

Exempel 6.2:
Logiska villkoret Q = C och (A eller B) måste skrivas på följande sätt om det ligger i ett POU under en TASK som exekveras på händelse eller med jämna tidsintervall.



Vidare väljs också exekveringsprioritet (*Priority*) mellan 0 och 31 för Tasket där 0 är högsta prioritet. Vid kompilering medför detta att det Task med högst prioritet läggs överst i programlistan.

Exekveringsordningen blir följande:

6. TASK med högst prioritet, dvs lägst angivet prioritetsvärde (0-31), exekveras först. Default har alla TASK lägsta prioritet 31.

7. De TASK med samma prioritet exekveras i den ordning de läggs i Navigatorn.

8. De POU som ligger under ett TASK exekveras i den ordning de läggs i Navigatorn.

9. Varje POU består av en instruktionslista eller av ett eller flera Networks i ladder (LD) eller funktionsblock (FBD) och dessa exekveras i ordning uppifrån och ner.

10. Exekveringen sker cykliskt dvs om och om igen i den ordning som ges av pkt 1-4.

11. Interrupt avbryter tillfälligt exekveringsordningen då interruptrutinen körs. Exekveringen återvänder efter interrupt tillbaka till det ställe i programcykeln den befann sig då interruptet kom.

Kompilering av hela projektet sker sedan i meny *Project* rubrik *Build* eller *Build all*. Endast de POU som är knutna till ett Task blir kompilerade. Vid kompileringen skapas den kod som kan laddas ner till PLC-systemet. Med *Build* kompileras endast det som är ändrat i projektet sedan föregående kompilering. Man vet vilka Task och POU som är kompilerade

genom att den röda asterisk som finns vid rubrikerna i Navigatorn försvinner vid lyckad kompilering.

6.7. Överföring av program och OnLine-funktioner.

I *Online-menyn* väljer man *Transfer Setup* för att ställa in hur överföring skall ske. För både Q02-systemen och A1S-systemen använder vi serieporten och RS232.

I *Online-menyn* kan man också starta monitorering av program som finns i aktiverat fönster och följa statusen hos olika signaler. *Entry Data Monitor* kan också väljas för att läsa statusen hos valfria adresser.

Man kan också göra onlineändringar av PLC-programmet. Med *Projekt - Online Program Change* kompileras det modifierade programmet och överförs till PLC:t då detta är i RUN mode.

6.8. Simulering av program.

I GX IEC Developermiljön finns en simulator som aktiveras genom Online – GX Simulator. Aktivering av denna funktion innebär att överföring sker till en simulerad PLC-enhet och signaler kan styras och monitoreras via en monitoreringsfunktion i programkoden och via *Entry Data Monitor*. OBSERVERA att det inte är möjligt att simulera POU:er skrivna i SFC.

6.9. Komma igång exempel.

Här följer en snabb steg för steg-genomgång av ett programexempel som utför följande:

a) Ingångarna In1, In2 och In3 kopplade till de fysiska ingångarna X1, X2 resp X3 används. Om In1 varit aktiv i 4 sekunder och In2 ej är aktiv eller om In3 är aktiv skall utgång Y14 benämnd Utgang14 aktiveras.

b) Ett registervärde Pott kopplat till register D54 skall räknas uppåt med en enhet per sekund. Registret får inte överskrida värdet 12. Pott skall nollställas om Utgang14 aktiveras enligt a).

1. Starta GX IEC Developer v 7.04 och aktivera Project – New.
2. Välj CPU-typ Q – Q02(H) alt. AnS - A1S.
3. Ge projektet ett namn och biblioteksplacering.
4. I New Project Startup Option väljs Empty Project.
5. Vi kör detta exempel med defaultinställningar varför vi i Project Navigator inte bryr oss om PLC Parameter.

6. Välj Global Variables i Project Navigator. Asterix före någon rubrik innebär att den enheten inte är kompilerad.

7. Lägg in i Global Variable List som VAR_GLOBAL Identifier In1 med MIT-adress X1 så fås automatiskt motsvarande IEC-standardadress och typen sätts automatiskt till BOOL.

8. Lägg till sex nya rader med hjälp av verktygsknapp  (tredje från höger).

9. Systemet serverar nu ett förslag på de sex följande raderna där de två första är helt acceptabla. Redigera om de två sista raderna så att de får följande utseende. Alla variabler som är kopplade till en in- eller utgångsadress måste deklareras som Globala variabler. I detta fall gäller det alltså In1, In2, In3 och Utgang14. De andra tre, Pott, Timer1 och Trig1, behöver egentligen i detta fall inte deklareras som globala utan det hade varit tillfyllest att deklarera dem som lokala variabler i Headern.

Global Variable List

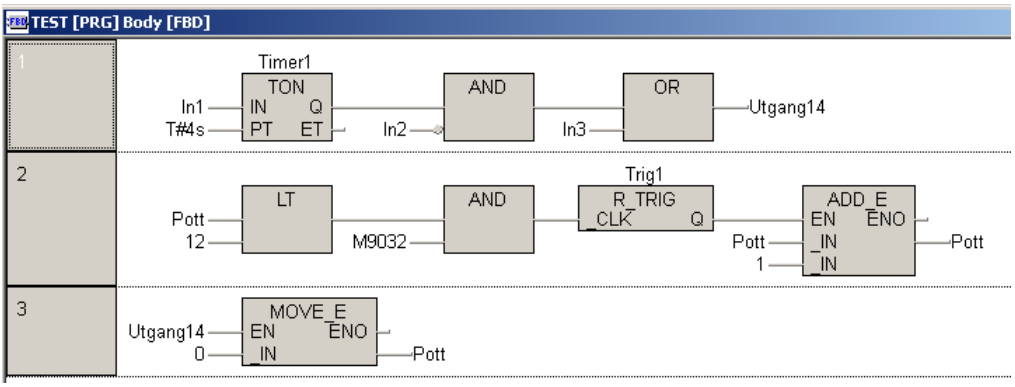
	Class	Identifier	MIT-Addr.	IEC-Addr.	Type
0	VAR_GLOBAL	In1	X1	%IX1	BOOL
1	VAR_GLOBAL	In2	X2	%IX2	BOOL
2	VAR_GLOBAL	In3	X3	%IX3	BOOL
3	VAR_GLOBAL	Utgang14	Y14	%QX20	BOOL
4	VAR_GLOBAL	Pott	D54	%MW0.54	INT
5	VAR_GLOBAL	Timer1			TON
6	VAR_GLOBAL	Trig1			R_TRIG

10. Gå till Project Navigator, markera POU-pool, klicka på verktygsknapp märkt POU. Ge POU:t ett namn och välj Function Block Diagram. Sedan OK.

11. Dubbelklicka i Project Navigator på POU-pool så kommer några undernivåer fram. Dubbelklicka där på Header vilket öppnar en ruta för lokala variabellistan. Några ytterligare variabler behöver vi inte i detta fall. Gå tillbaka till Project Navigator och dubbelklicka Body. En Network-ruta finns nu i Body-rutan. Välj i verktygsknappraden en knapp med en ”IC-kapselsymbol” på. En dialogruta öppnas. Välj Operator Type till All Types och välj där en AND. Behåll Number of Pins vid defaultvärdet 2. Tryck Apply och gå upp i Network-rutan och klicka in funktionsblocket. Testa också att du kan flytta på blocket med musen. För att invertera ingång In2 klicka vid ”roten” av ingångsbenet. Hämta också ett OR-block och ett TON-block och placera på lämpligt ställe.

12. Klicka på ? på in och utgångar. Högerklicka. Via dialogruta kan In1, In2, In3, Utgang14 läggas på sina ställen enligt figur nedan. Detta kan också skrivas in via tangentbordet. Markera ingången PT på TON-blocket och skriv T#4s vilket innebär 4 sekunders fördröjning. Återstår att knyta ihop de olika blocken. Högerklicka och välj Interconnect Mode. Klicka på utgång, flytta musen och klicka på ingång. När alla förbindelser är gjorda högerklicka och välj Select Mode.

13. Skapa två nya network med verktygsknapp  och lägg in block enligt figur nedan. ADD_E blocket innebär att Pott ökar med en enhet för varje positiv flank som läggs på EN-ingången. M9032 är ett pulståg med frekvensen 1 Hz.



14. Vi skall nu koppla det skapade delprogrammet, POU:et, till ett Task. Gå till Projekt Navigator, markera Task och klicka på verktygsknappen märkt TSK. Ge tasket ett namn.

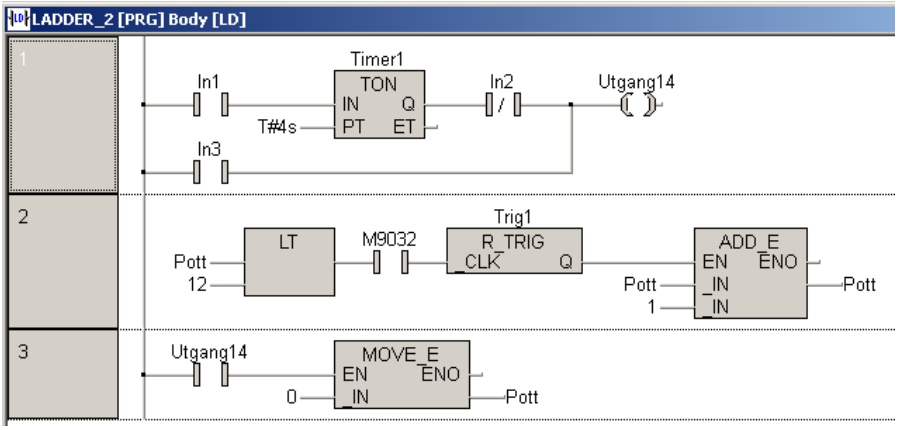
15. Dubbelklicka på det antagna tasknamnet i Project Navigator. En dialogruta kommer fram där önskade POU:n ansluts till detta Task. I detta fall finns bara ett val.

16. Nu är projektet klart för kompilering. Gå till Project i huvudmenyn och aktivera Rebuild all. Förhoppningsvis inga fel annars gå in och korrigera.

17. Dags för överföring till PLC-systemet. Välj huvudmenyns Project och Transfer och där Dopwnload to PLC. (Lyckas det inte så gå till Online-menyn välj Transfer Setup - Ports - Setup och ställ in rätt port.)

18. Monitoreringar och on-line-ändringar kan sedan göras enligt avsnitt 6.7.

19. Skulle samma problem lösas med ladderprogrammering LD skulle denna programmeringsform väljas under punkt 110 ovan. Det färdiga programmet skulle se ut enligt nedan. Som synes är det endast de logiska grindarna av typ AND och OR och negeringar som beskrivs med serie och parallellkopplade kontakter i ett ladderdiagram. För övrigt används samma funktionsblock som i FBD.



20. För att skriva SFC-program är förfarandet ungefär detsamma. Det som skiljer är uppbyggnaden av själva funktionsdiagrammet.

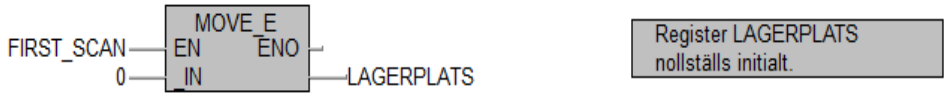
6.10. Dokumentation.

Vid all form av utvecklingsarbete är dokumentation en nödvändighet. Det gäller i högsta grad i programutvecklingssammanhang. Är ett program inte dokumenterat på ett tillfredsställande sätt kan det vara mycket tidskrävande att redan efter en relativt kort tid återvända för att göra en modifiering, även om man själv utvecklat programmet. Om sedan någon som inte varit med i utvecklingsarbetet skall göra modifieringen kan det vara ännu besvärligare om bra dokumentation saknas.

I utvecklingsmiljön GX IEC finns goda möjligheter till dokumentation. Dels skall man hela tiden arbeta med relevanta benämningar på de variabler eller signaler som används. Var alltså noggrann med att hitta processnära benämningar som därmed direkt gör programlogiken mera lättläst.

Därutöver finns det mycket goda möjligheter att lägga in kommentarer där så önskas.

Genom att aktivera ikonen  kan en kommentarruta ”ritas” i arbetsfältet genom att aktivera vänster musknapp. Resultatet blir enligt nedan.



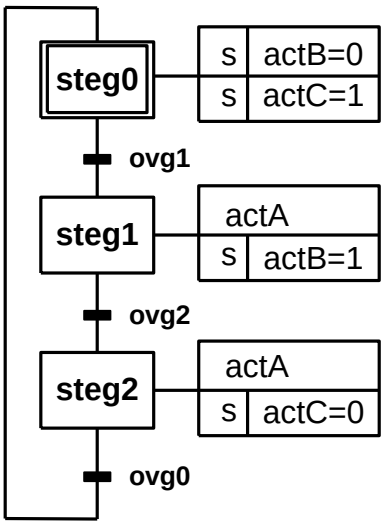
Utskrift av delar eller hela GX IEC Developer-projekt kan göras via Print-funktionen under Projekt i menyn. Utskrift sker av det som är markerat i Navigatorn. Om hela projektet markeras blir default-utskriften omfattande. Genom att ställa önskad omfattning i Print Options kan man styra omfattningen av utskriften. Utskriften sker på eget format men kan göras till en PDF-writer så att dokumentationen sedan kan kopplas till annat dokument.

Kap 7. Sekvensstyrningar i LD och FBD.

Programmering av sekventiella förlopp är numera möjligt att göra i språket SFC, Sequence Function Chart. Sekventiella förlopp har dock utförts länge med reläer och sedan, innan SFC-språket dök upp, med LD- eller FBD-kod i PLC-styrningar. Följande avsnitt beskriver programmeringsmönstret för hur en sekvenskedja kodas på ett systematiskt sätt utgående från funktionsdiagrammet. Systematiken är viktig annars virrar man lätt bort sig i koden. Den nedan beskrivna metoden för LD-kod är också överförbar till relälösningar för sekvensiella förlopp.

7.1. Funktionsdiagrammet.

Funktionsdiagrammets uppbyggnad är reglerad i tidigare presenterad standard IEC 848. Det som följer i detta kapitel utgår från ett enkelt sekventiellt förlopp beskrivet i funktions-diagram i Figur 7.1.



Figur 7.1: Funktionsdiagram för ett enkelt sekventiellt förlopp

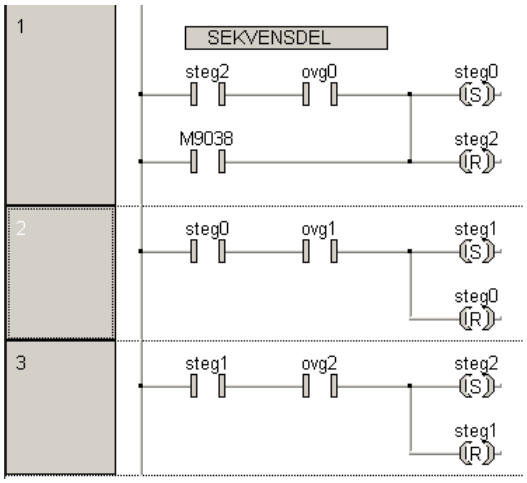
Vi skall nu titta på hur denna styrföljd kan realiseras I PLC-program med några olika metoder. Nedan visas Globala Variabellistan för de POU som presenteras senare.

	Class	Aut	Identifier	MIT-Addr.	IEC-Addr.	Type
0	VAR_GLOBAL		ovg0	X0	%IX0	BOOL
1	VAR_GLOBAL		ovg1	X1	%IX1	BOOL
2	VAR_GLOBAL		ovg2	X2	%IX2	BOOL
3	VAR_GLOBAL		actA	Y1A	%QX26	BOOL
4	VAR_GLOBAL		actB	Y1B	%QX27	BOOL
5	VAR_GLOBAL		actC	Y1C	%QX28	BOOL
6	VAR_GLOBAL		steg0	M0	%MX0.0	BOOL
7	VAR_GLOBAL		steg1	M1	%MX0.1	BOOL
8	VAR_GLOBAL		steg2	M2	%MX0.2	BOOL

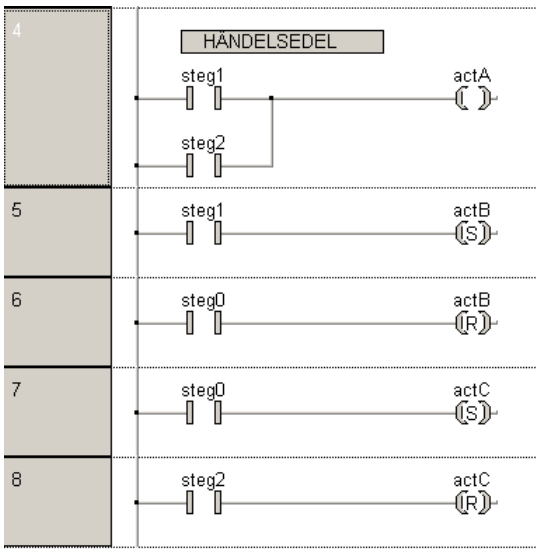
7.2. Lösning som Ladderprogram.

Nedan framgår hur man delar upp förloppet i en sekvensdel och en händelsedel. Anledningen till det är att en och samma händelse kan ske i flera olika steg och då kan de inte aktiveras separat i varje aktuellt steg eftersom då endast den sista påverkan av händelsen skulle slå igenom. PLC:t arbetar så att insignaler läses in, sedan exekveras hela koden varefter utsignalerna läggs ut. Detta upprepas sedan cykliskt. I sekvensdelen kodar man alltså att steg och övergångar genomförs.

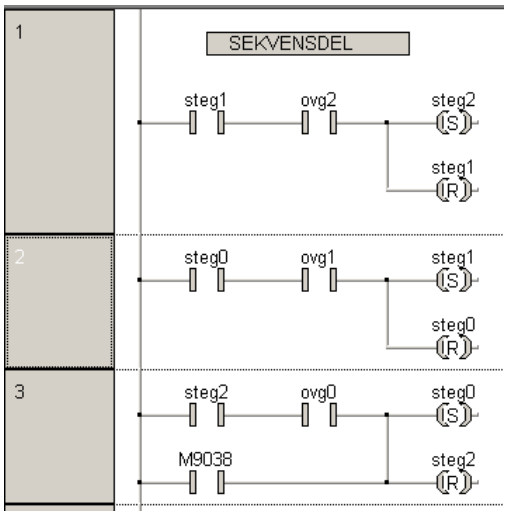
En svaghet i sekvensdelen nedan är att om t ex **ovg2** skulle vara aktiv då **ovg1** blir aktiv skulle **steg1** passeras direkt utan att någon exekvering av händelsedelen sker. Detta medför att dessa händelser i det fallet inte blir genomförda. Detta sätt att koda innebär att man måste försäkra sig om att ett stegs efterföljande övergångsvillkor verkligen kontrollerar att stegets händelse är utförd.



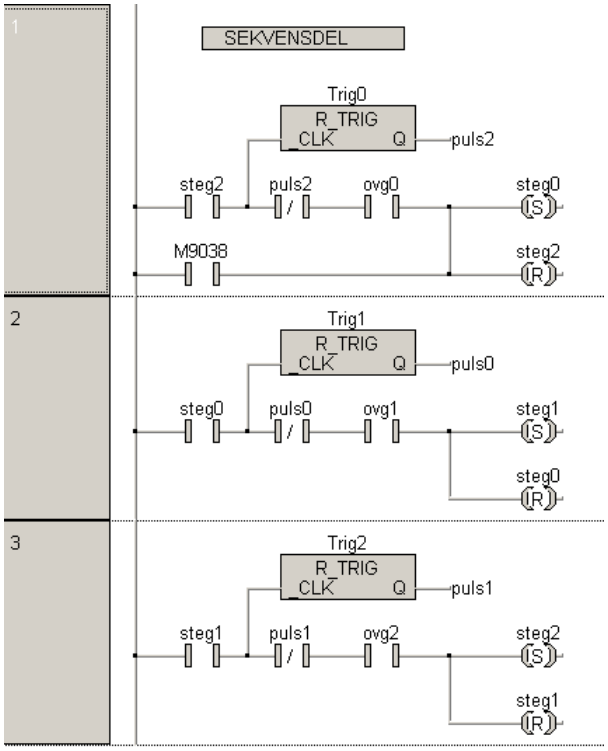
I Händelsedelen tas varje händelse upp en gång och de steg som skall aktivera denna händelse samlas till en gemensam OR-grind. Detta säkrar att händelser alltid blir utförda.



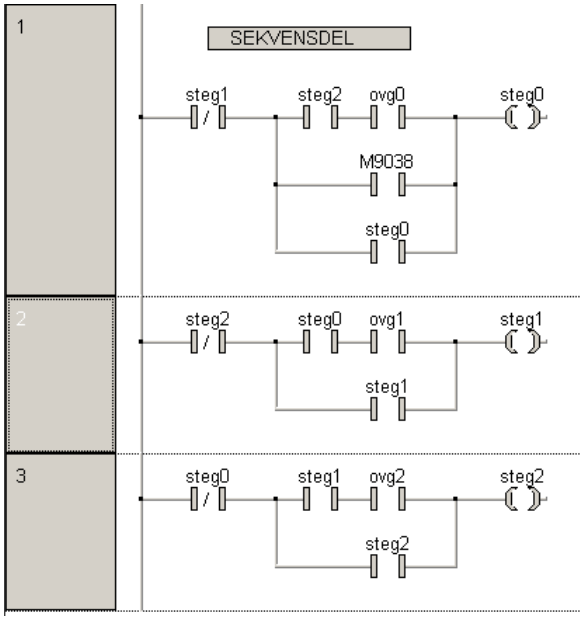
Med en ändring av sekvensdelen till att behandla stegen i ordning nerifrån och upp enligt nedan elimineras problemet med risk för överhoppad händelse till att sista stegets, här steg2:s, händelse inte blir exekverad om ovg0 är aktiv då ovg2 blir aktiv. Särskild uppmärksamhet måste alltså ges sista steget i sekvenskedjan vid kodning av stegen nerifrån och upp i funktionsdiagrammet. Händelsedelen blir här densamma.



Genom att utnyttja en flanktriggning vid övergången kan alla övergångar säkras för passage utan att händelser exekveras. Metoden innebär att två exekveringsvarv måste genomlöpas vid varje övergång vilket ökar fördröjningen något. Denna teknik framgår av koden nedan.

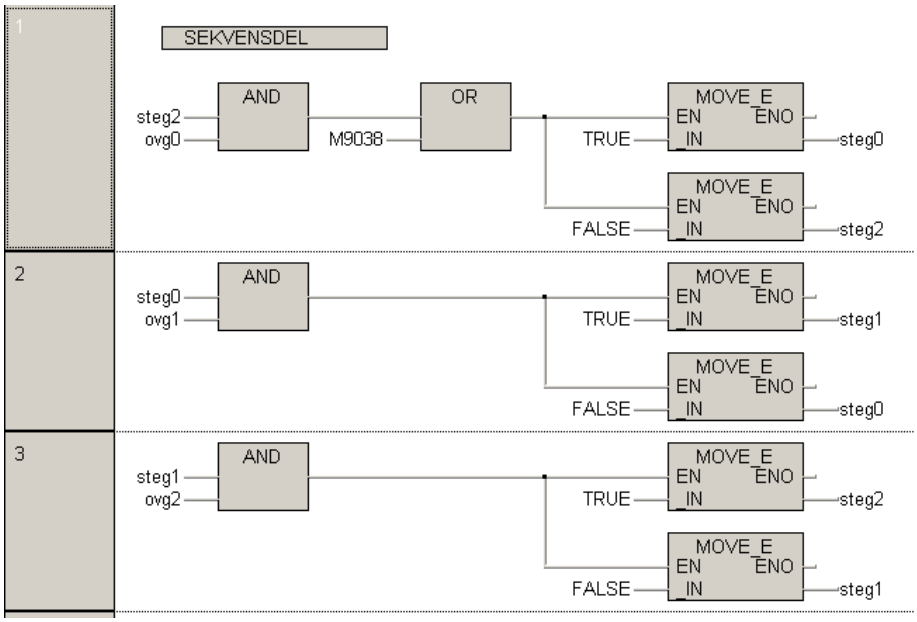


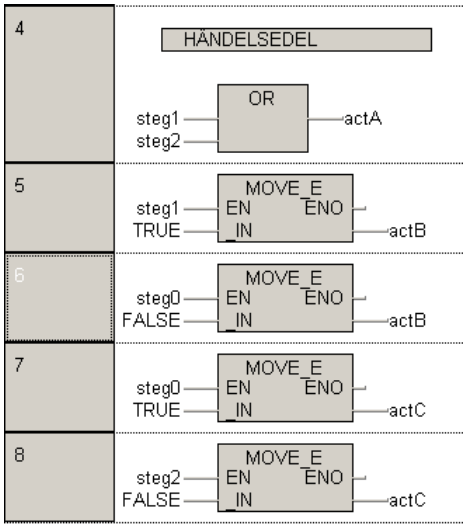
Sekvensdelen kan naturligtvis också kodas i LD med hållkretsar. Nedan givna exempel är jämförbart med första lösningen ovan där risk fanns för missade utförda händelser om två övergångsvillkor efter varandra är uppfyllda samtidigt. Detta sätt att koda är det som ligger nära relälösningar från tider före PLC. Dessa relälösningar innebar alltså att det krävdes ett relä för varje steg i sekvensen.



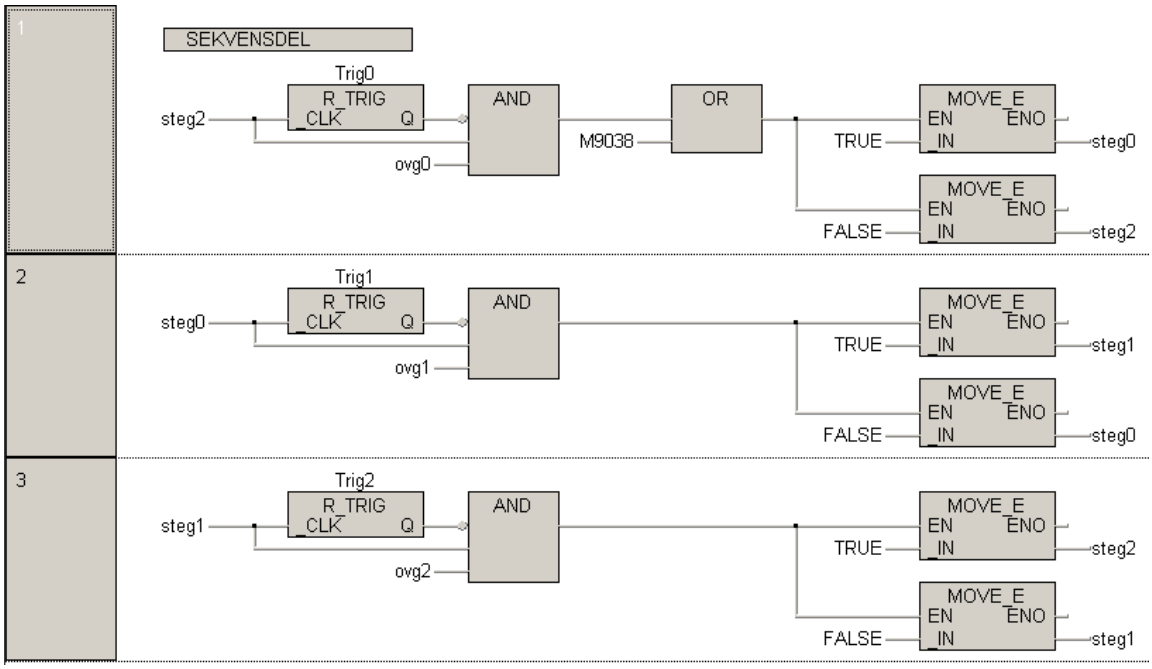
7.3. Lösning som Funktionsblock-program.

Föredrar man att koda samma styrföljd i FBD enligt IEC kan koden se ut enligt nedan. Det första exemplet har samma uppbyggnad som första LD-exemplet ovan.





Nedan visas en lösning med övergångssäkring av händelser enligt det tidigare LD-exemplet men nu kodat i FBD.



Sakregister

M

N

O

P

R

S

T

U

V,W

X

Ö