TTK4235: PLS me hard, daddy

Kolbjørn Austreng

Vår 2020

Beskrivelse

PLS står for "Programmerbar Logisk Styringsenhet" og er et utbredt verktøy for å løse automatiseringsoppgaver i industrien. En PLS er en forholdsvis enkel innretning, som består av en prosessor, digitale- og/eller analoge innganger, og eventuelt også en eller flere kommunikasjonsgrensesnitt.

Vanligvis vil man koble en rekke sensorer til PLSens innganger, og en rekke pådragsorganer til PLSens utganger. På selve PLSen implementerer man et sett med logiske funksjoner, slik at man kan sette utgangene basert på hva som inngangene registrerer. Man står fritt til å lage de logiske funksjonene man vil - det være seg rent kombinatoriske funksjoner, sekvensielle funksjoner, eller en blanding av de to.

Eksempler på sensorer er fotoceller, endebrytere, eller nivåfølere. Eksempel på pådragsorganer kan være alt fra motorer, ventiler, lamper, eller releer.

I denne laboppgaven skal vi bruke en PLS fra Siemens; SIMATIC S7-300. Allerede i 1958 ble SIMATIC et registrert varemerke, og den første SIMATIC-PLSen så dagens lys året etter. Dette var den såkalte "SIMATIC G", som baserte seg på germaniumtransistorer - i motsetning til de mer moderne silisiumtransistorene vi har i dag.

SIMATIC S7-300 ble introdusert i 1994, sammen med lillebroren S7-200 og storebroren S7-400. Med disse systemene ble også den nye feltbusstandarden "PROFIBUS" og industrielt ethernett introdusert - som markerte starten på "nettverksalderen" for PLS.

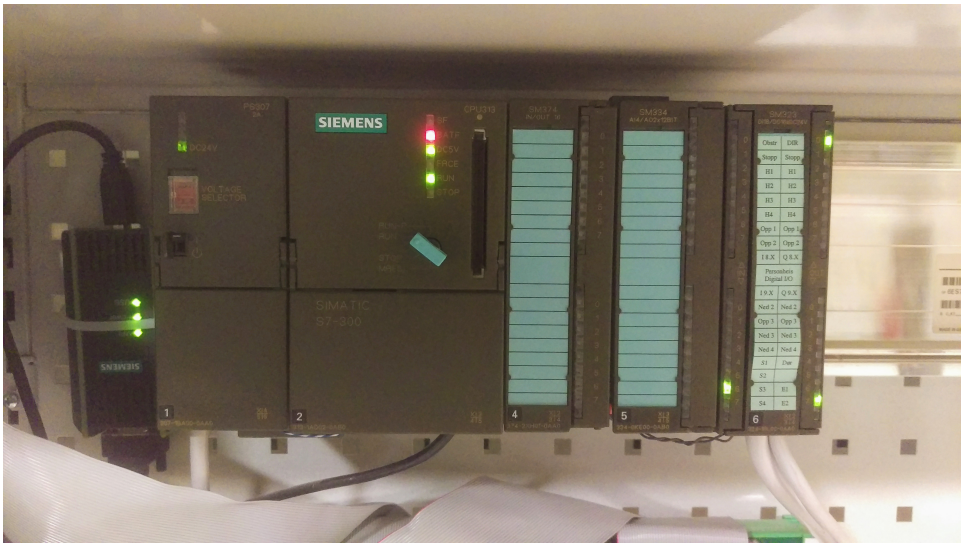
Dette høres kanskje ut som gammel teknologi, noe som forsåvidt er sant, men det er også en grunn til at PLSer ikke egentlig har endret seg særlig siden nittitallet.

I bunn og grunn er PLSer så enkle at "ingenting" kan gå ordentlig galt.

Om du trenger å implementere et automatisk styresystem for et industrielt anlegg, som skal stå i 20 år uten omstart, da er en PLS et godt egnet verktøy. Hadde du prøvd å styre det samme anlegget med en Windows-maskin, ville anlegget ditt krevd en omstart i uka, russiske hackere hadde vært inne allerede ved lanseringsdatoen, og om et par år ville du måtte kjøpe en ny lisens.

PLSer er "kjedelige industrimaskiner" som sjelden får noen særlig heder for innsatsen de gjør, men de er veldig gode til akkurat de oppgave vi trenger dem til.

Overblikk over hardware

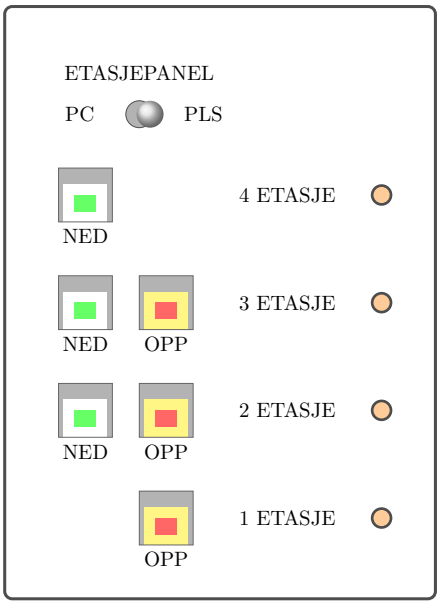


Figur 1: SIMATIC S7-300 brukt på Sanntidssalen.

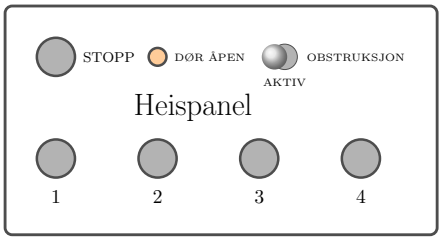
I figur 1 ser dere PLSen brukt på sanntidssalen. Dette er en modulbasert PLS, som betyr at PLSens funksjonaliteter kan utvides ved behov, ved å legge til ekstra moduler. Det er faktisk bare "klossen" som har et 2-tall nede i venstre hjørne som er selve PLSen. Den første klossen er strømforsyning, og de tre andre er moduler som kan gjøre Input/Output.

På labplassen deres står det også en del andre ting. Dere skal ha en heis-modell, som består av en sjakt og en bevegelig heisstol. Det er denne vi skal få til å bevege seg i labopplegget. I tillegg til dette, har dere en motordriver - dette er den store svarte boksen som står ved siden av heismodellen. Før dere gjør noe annet, skrur dere denne på, og passer på at den rød ledningen fra heisen er koblet til `M+`, og den blå ledningen til `M-`.

Til slutt har dere et "heispanel". Dette er illustrert i figur 2 og figur 3. På toppen av etasjepanelet (figur 2) finner dere en bryter som brukes til å skifte styring mellom datamaskin og PLS. I denne laboppgaven skal bryteren stå i "PLS" hele tiden.



Figur 2: Etasjepanel.



Figur 3: Heispanel.

Overblikk over software

Fordi PLSer er gang delte denne jorda med dinosaurene som senere ble oljen i våre hav - er det ingen overraskelse at måten man programmerer dem på har rukket å bli standardisert. Det finnes faktisk en industristandard som forklarer nettopp dette. Den heter `IEC 61131-3`, og definerer fire forskjellige måter man kan fortelle en PLS hva den skal gjøre:

- `Stigediagram (LD)`, eller "Ladder diagram". Dette er et grafisk programmeringsspråk, hvor man lager linjer som representerer programflyt, hvor linjene kan ha brytere som representerer kontrollstrukturer.
- `Funksjonsblokkdiagram (FBD)`. Dette er også et grafisk programmeringsspråk, men her trekker man koblinger mellom utgangen på blokker inn i inngangene til andre blokker.
- `Strukturert tekst (ST)`. Dette er et programmeringsspråk som faktisk bruker tekst - som navnet tilsier. Det ligner litt på et språk som heter "PASCAL". Det er ingen som har hørt noe om PASCAL etter at C ble oppfunnet.
- `Sekvensielle funksjonsdiagram (SFC)`, eller "Sequential function chart". Dette er nok et grafisk programmeringsspråk, men har i tillegg noen elementer for å kunne blande sekvensielle operasjoner med parallelle operasjoner.

Før 2013 fantes det også en femte standardmåte å programmere PLSer på. Denne het `Instruksjonslister (IL)`, og lignet på en eller annen variant av et assemblyspråk. Instruksjonslister ble fjernet i tredje utgave av IEC 61131-3.

Tro det eller ei, men i denne laben skal vi faktisk holde oss til et av de grafiske programmeringsspråkene. Vi skal nemlig benytte oss av funksjonsblokkdiagrammer gjennom hele oppgaven.

Kort om funksjonsblokkdiagrammer

Når dere programmerer en PLS med funksjonsblokkdiagrammer, vil programmet deres være organisert i flere forskjellige blokker. Her er de dere trenger å vite om:

- `Organisasjonsblokker (OB)` representerer grensesnittet mellom operativsystemet og programmet deres. PLSens operativsystem har ansvar for å kalle blokker av denne typen. Vi skal bruke én slik blokk i laboppgaven; en blokk kalt `OB1`, som vil kjøres syklisk av PLSen. Inne i denne blokken vil først alle innganger leses, før eventuelle logiske beregninger gjøres. Til slutt vil alle utganger settes på de fysiske enhetene koblet til

PLSen. Dette gjentar seg et visst antall ganger for hvert PLS-tidssteg. Andre organisasjonsblokker enn `OB1` kjøres bare ved interrupts eller ved registrering av systemfeil, og skal ikke brukes i denne laboppgaven.

- `Funksjoner (FC)` representerer en "ren funksjon", altså en type funksjon som ikke lagrer intern tilstand. Generelt sett ønsker man å unngå ekstra tilstand så lenge man kan unngå det, hvilket forenkler kompleksiteten av programmet.
- `Funksjonsblokker (FB)` er nesten som vanlige funksjoner, men lagrer i tillegg intern tilstand. Foretrekk vanlige funksjoner over disse så langt det lar seg gjøre.
- `Datablokker (DB)` brukes til å lagre data, og kan sees på som statiske variabler.
- `Systemfunksjoner (SFC)` er ferdige rutiner fra biblioteker. Vi kommer ikke til å bruke disse her, men de nevnes for kompletthetens skyld.
- `Systemfunksjonsblokker (SFB)` er det samme som SFCer, bare med data tilknyttet. Vi kommer heller ikke til å bruke disse blokkene i denne oppgaven.

Kort om IO

En PLS uten input og output kan ikke gjøre stort spennende. Innganger og utganger blir tildelt adresser av PLSen, avhengig av hvor de er koblet i PLSens moduler. Adresser ser slik ut:

I$x.y$	Adressen til input bit nummer y, i byte nummer x.
IBx	Adressen til input byte nummer x.
IWx	Adressen til input word nummer x. (Et "word" er 32 bit)

For output fungerer adresseringen på samme måte, men da starter adressen med "`Q`" istedenfor "`I`". Enkelte adresser blir også regnet som "perifere", og får da prefikset "`P`". Et eksempel på dette er adressen til heismotoren, som har adressen `PQW272`.

Dere kan fint hardkode alle adressene i oppgaven, men det ville vært litt det samme som å aldri opprette variabler i vanlig programmering. Det reduserer leseligheten av programmet, og gjør at dere må hardkode masse greier på nytt om innganger- eller utganger skulle endre seg. For å løse dette, skal vi opprette en "symboltabell", som gir adressene navn.

Kort om logikkstyring

Logiske funksjoner kan deles inn i kombinatoriske- og sekvensielle operasjoner. Kombinatoriske funksjoner er alle "rene funksjoner", hvor utgangen av en blokk kun er avhengig av blokkens innganger. Tidligere input har ingenting å si for slike operasjoner. Et eksempel på en kombinatorisk funksjon er den logiske `OR`-porten. Her er utgangen logisk `sann` om minst én av inngangene er logisk `sann`.

På den annen side har vi sekvensielle funksjoner, som har en eller annen form for minne innebygget. Et eksempel på dette er en `SR`-vippe. `SR` står for "Set/Reset", og kan betraktes som en liten bryter som er i stand til å huske ett eneste bit informasjon. Utgangen til `SR`-vippen er ikke bare bestemt av hva inngangene er for noe, men også hvilken av inngangene som sist var logisk `sann`. Om "Set"-inngangen sist var `sann` (og "Reset"-inngangen er `usann`) vil for eksempel vippen fortsette å være `sann`. Sannhetstabellen ser dere her:

R	S	Q_n
0	0	Q_{n-1}
0	1	1
1	0	0
1	1	0

Med andre ord er utgangen til en `SR`-vippe det samme som tilstanden den er i. Derfor kan dere betrakte den som et eksempel på en veldig simpel sekvensiell krets.

Praktisk rundt laben

Seksjon 1 er i stor grad en "tutorial" som introduserer hvordan dere bruker PLSen. Det er ingen egen godkjenning for denne dagen - godkjenning av lab gjøres når dere har vært gjennom begge dagene.

Hvis alt går knirkefritt, for all del, gjør begge labdagene i ett - vi har bare valgt å legge inn litt pusterom i tilfelle dere skulle sette dere fast.

Hvis dere nå ender opp med å stå fast, her er stegene dere skal ta, i den rekkefølgen dere skal ta dem, for å komme ut av videre:

1. Les oppgaveteksten nøye en gang til. Det meste burde være forklart direkte i teksten.
2. Les seksjon 3. Denne har en del hint som kan være greie å ha.
3. Skum gjennom de seksjonene dere tror har det dere trenger fra manualen "Working with STEP 7". Denne ligger ute på blackboard, i tillegg til at hver labplass har en fysisk kopi i stativet ved siden av PLSen. Dette er en ganske god manual, så bruk den flittig om det er noe dere lurer på.
4. Spør studass eller vitass.

1 Oppgave første labdag

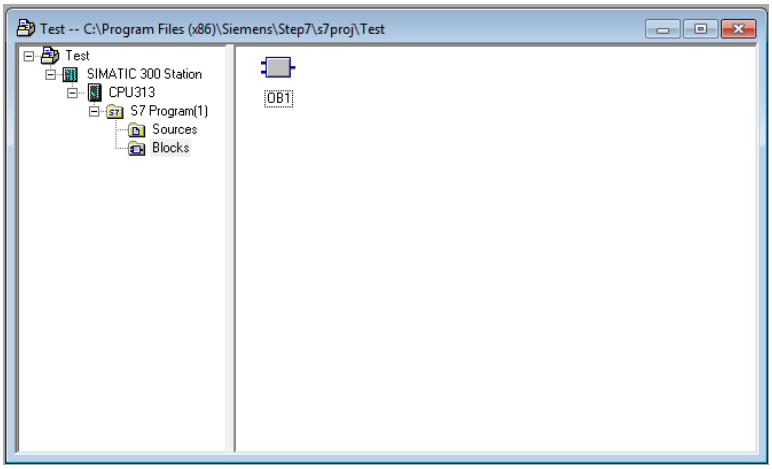
Denne laben gjennomføres dessverre på Windows, så om dere ikke allerede har startet opp Windows, starter dere datamaskinen på nytt og velger Windows i bootmenyen.

Når dere har logget inn, starter dere programmet "SIMATIC Manager", som skal ligge på skrivebordet. Når dette programmet åpner, skal dere se en "project wizard". Hvis denne ikke starter av seg selv, finner dere den under `File`. Her gir dere prosjektet deres et navn som dere er i stand til å finne tilbake til ved neste labdag.

Når dere trykker videre, må dere velge riktig PLS CPU. På Sanntidssalen har vi to forskjellige varianter; `CPU313` og `CPU315-2 DP`. Hvilken type dere har, står øverst i høyre hjørne på den fysiske PLS-blokken.

Når dere får spørsmål om hvilke blokkere dere har lyst til å legge til, velger dere bare `OB1`. Dere skal også velge "Function Block Diagram (FBD)" som programmeringsspråk i det samme vinduet.

Til slutt vil dere ha et prosjektvindu som vist i figur 4.



Figur 4: Prosjektfiler ved ferdig nyopprettet prosjekt.

1.1 Opprett symboltabell

Fra vinduet i figur 4, trykker dere på mappen kalt "S7 Program", og deretter dobbeltklikker dere på "Symbols". Først og fremst omdøper dere `OB1`-blokken fra "Cycle execution" til "main" (fordi vi ønsker å uttrykke vår hjemlengsel til C ♡).

Deretter oppretter dere disse symbolene:

Adresse	Symbolnavn
I 8.0	Obstruction
I 8.1	Stop button
I 8.2	Cab order 1
I 8.3	Cab order 2
I 8.4	Cab order 3
I 8.5	Cab order 4
I 8.6	Up 1
I 8.7	Up 2
I 9.0	Down 2
I 9.1	Up 3
I 9.2	Down 3
I 9.3	Down 4
I 9.4	Floor sensor 1
I 9.5	Floor sensor 2
I 9.6	Floor sensor 3
I 9.7	Floor sensor 4
Q 8.0	Motor direction (0 = up, 1 = down)
Q 8.1	Light stop button
Q 8.2	Light cab order 1
Q 8.3	Light cab order 2
Q 8.4	Light cab order 3
Q 8.5	Light cab order 4
Q 8.6	Light up button 1
Q 8.7	Light up button 2
Q 9.0	Light down button 2
Q 9.1	Light up button 3
Q 9.2	Light down button 3
Q 9.3	Light down button 4
Q 9.4	Light door open
Q 9.5	Unused
Q 9.6	Floor indicator bit 1
Q 9.7	Floor indicator bit 2
PQW272	Motor drive strength

1.1.1 Om etasjeindikatorbit

Fra symboltabellen dere har opprettet ser dere at vi har to bit for å sette etasjeindikatorlysene. Dette er for å spare på utganger fra PLSen. Lysene er enkodet slik:

Bit 2	Bit 1	Etasjeindikatorlys
0	0	Etasje 1
0	1	Etasje 2
1	0	Etasje 3
1	1	Etasje 4

1.1.2 Om motorpådrag

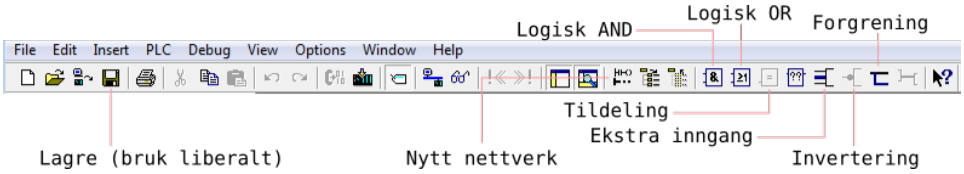
Motorpådraget er det eneste analoge signalet vi setter fra PLSen. Avhengig av hvilken verdi vi setter til motorens adresse, vil motordriveren endre spenningen som fôres til heisen:

Verdi	Motorspenning (V)
`W#16#0000`	0,0
`W#16#3DFF`	2,5
`W#16#7EFF`	5,0

1.2 Tutorial: Lys i stoppknappen

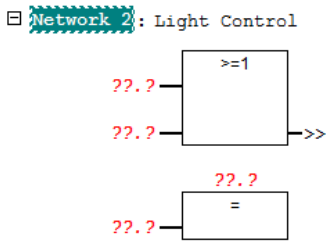
For å introdusere hvordan dere programmerer PLSen, skal vi skru på lyset i heispanelets stoppknapp så lenge en av oppknappene blir holdt inne.

Dobbeltklikk på organisasjonsblokken `OB1`. Dette vil åpne et nytt vindu med ett tomt "nettverk". Et nettverk er simpelthen en gruppering av funksjonaliteter. Dere kan opprette flere nettverk ved behov, men for dette trenger vi bare det første.



Figur 5: Forklaring av noen av symbolene i toppbaren.

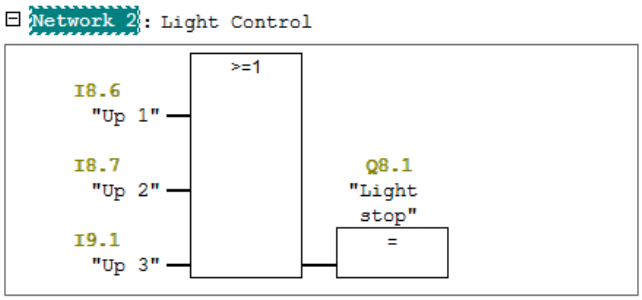
Øverst i vinduet har dere en meny for å legge til vanlige blokker. Denne er forklart i figur 5. Herfra legger dere til en `OR`-blokk, og en tildelingsblokk. Dere vil da ha noe som ligner på figur 6.



Figur 6: `OR`-blokk og tildelingsblokk.

Velg deretter den nye `OR`-blokken, og legg til en ny inngang ved å bruke symbolet for dette fra toppmenyen. Så trekker dere utgangen fra `OR`-blokken til inngangen av tildelingsblokken. For å kunne trekke, må dere plassere musepekeren akkurat på den svarte streken, mellom selve blokken og de to større-enn-tegnene. Husk at dette er et system fra 1994, så den "sømløse brukeropplevelsen" kan enkelte ganger være litt deretter også.

Når OR-blokken har fått tre innganger, og utgangen dens er koblet til inngangen på tildelingsblokken, kan dere trykke på de skumle rød spørsmålstegnene og skrive inn symbolene dere opprettet i symboltabellen. Dere skal ende opp med noe i duren av figur 7.



Figur 7: Ferdig koblet blokker med riktige symboler.

Når dere har kommet så langt, og lagret OB1, er det på tide å programmere PLSen. PLSen kan ikke programmeres hvis den kjører, så sørg for at den blå bryteren på den fysiske PLSen står i posisjonen "STOP". Deretter lukker dere vinduet til OB1 slik at dere ser det samme som i figur 4. Høyreklikk nå på OB1, og last blokken ned til PLSen.

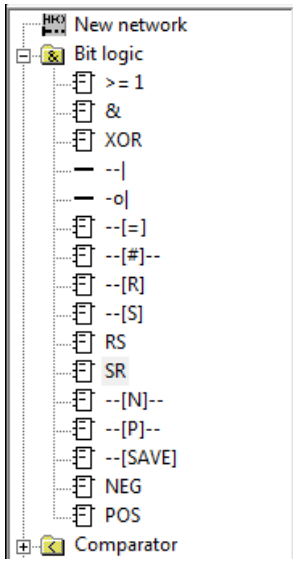
Om SIMATIC Manager klager med "Unable to copy", kan det være at riktig grensesnitt mellom datamaskin og PLS ikke er satt. Da velger dere Options → Set PG/PC Interface, og her velger dere "PC Adapter.MPI.1".

Når PLSen nå forhåpentligvis er programmert riktig, flipper dere den blå bryteren tilbake til "RUN". Så prøver dere å holde inne én eller flere av oppknappene på heispanelet. Nå skal lyset i stoppknappen tennes.

1.3 Tutorial: Gjør stopplyset vedvarende

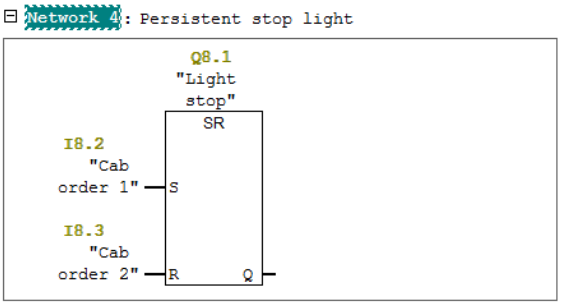
Programmet vi nettopp lagde er et eksempel på en rent kombinatorisk funksjon. Her skal vi introdusere en `SR`-vippe for å huske hvilken knapp som ble trykket sist.

Dobbelttrykk på `OB1` for å åpne den på nytt, og slett det som ligger der fra forrige oppgave (velg en blokk og trykk "Delete" et par ganger). Så går dere i menyen til høyre. Her vil dere se flere mapper med flere kategorier. Åpne mappen kalt "Bit logic". Dere vil nå se det samme som i figur 8.



Figur 8: Meny over blokkene fra "Bit logic".

Her velger dere naturlig nok symbolet kalt "`SR`". Dette symbolet fungerer på akkurat samme måte som tidligere, men her trenger vi ikke en ekstra tildelingsblokk; vi kan putt det ønskede utgangssignalet på toppen av vippen. Et eksempel som bruker første- og andre bestillingsknapp fra heispanelet til å sette stopplyset er illustrert i figur 9.



Figur 9: Vedvarende stopplys ved bruk av `SR`-vippe.

Dere står selvsagt fritt til å kombinere inngangene på vippen med `OR`-blokker, hvis dere for eksempel ønsker å sette stopplyset fra alle oppknappene, og slukke det fra alle nedknappene.

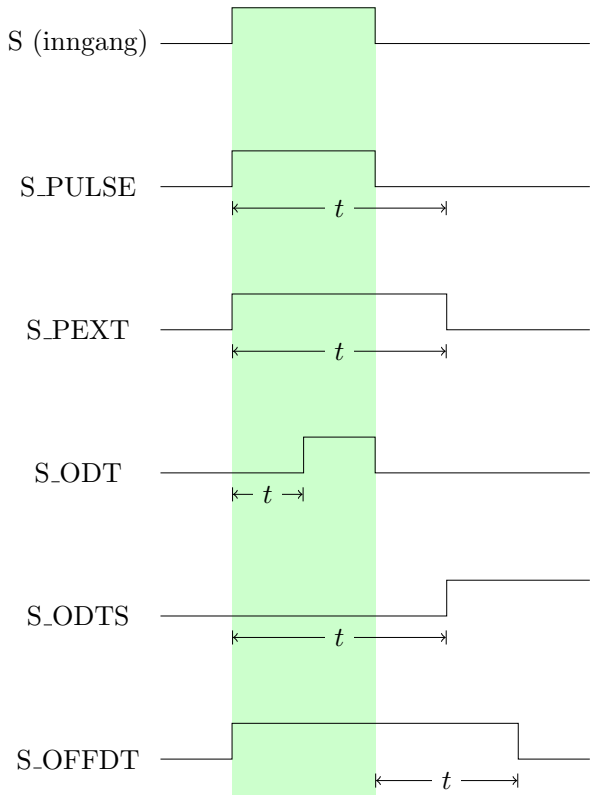
Igjen er prosedyren den samme for å programmere PLSen; lagre `OB1`, sett bryteren i "`STOP`", last ned, og sett bryteren tilbake i "`RUN`". Dere skal nå ha et enkelt sekvensielt program.

1.4 Tutorial: Timere

Ofte skal en oppgave helt- eller delvis gå på tid. For å illustrere hvordan dette gjøres, skal vi implementere følgende:

- Så lenge obstruksjonsbryteren på heispanelet er på, skal stopplyset være tent.
- I tillegg skal stopplyset vente 3 sekunder med å slukke seg etter at vi har skrudd av obstruksjonsbryteren.

For å implementere dette, skal vi bruke en timer. Avhengig av hvilken oppgave vi ønsker å løse, har vi flere forskjellige timere; illustrert i figur 10:



Figur 10: Illustrasjon av forskjellige timere.

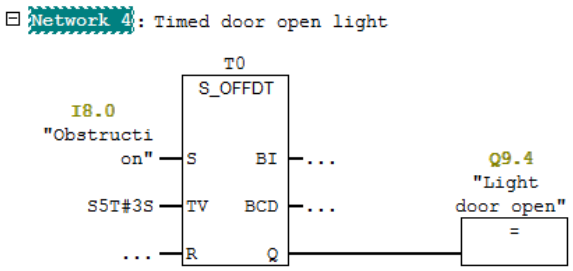
Timer	Forklaring
S_PULSE (Pulse timer)	Danner en puls på t tidsenheter. Pulsen vil aldri være lengre enn varigheten på inngangen.

S_PEXT (Extended pulse)	Danner en puls på t tidsenheter, uavhengig av hvor langvarig pulsen på inngangssignalet er.
S_ODT (On-delay)	Forsinker stigende flanke med t tidsenheter. Signalet går lavt igjen straks inngangssignalet faller.
S_ODTS (Retentive on-delay)	Forsinker stigende flanke med t tidsenheter, men signalet vil ikke falle med inngangssignalet.
S_OFFDT (Off-delay)	Utgangssignalet går høyt sammen med inngangssignalet, men venter t tidsenheter med å falle tilbake etter at inngangssignalet allerede har gått lavt.

Her faller det seg naturlig å bruke en "Off-delay timer", så vi er altså på utkikk etter blokken kalt "`S_OFFDT`". Denne finner dere til venstre i blokkmenyen.

Syntaksen for å fortelle hvor lenge en tidsperiode skal vare er litt finurlig. For å skrive for eksempel "3 sekunder", heter dette "`S5T#3S`" i SIMATIC Manager. "7 sekunder" blir tilsvarende "`S5T#7S`".

Igjen er det bare å huske på at dette er et gammelt industriprogram fra 1994. Med det i bakhodet, kan vi vel egentlig være enige om at brukeropplevelsen kunne vært mye verre enn den er.



Figur 11: Tidsbestemt stopplys ved hjelp av `S_OFFDT`.

I tillegg må alle timerblokker referere en faktisk timerressurs på PLSen. Denne putter dere i toppen av blokken. Et eksempel på en timerressurs er "`T0`", hvor det siste symbolet er tallet "0", ikke en stor O.

I figur 11 ser dere et eksempel på et program som vil holde lyset i stoppknappen tent så lenge obstruksjonsbryteren er på, og fortsette i 3 sekunder etter at vi skrur den av.

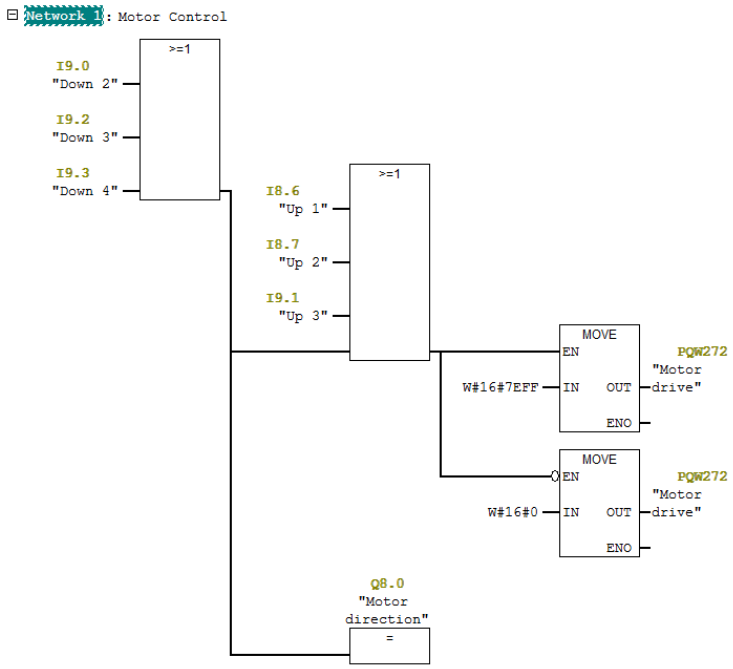
For å finne ut ting som at "`TV`" står for "Timer value", kan dere velge "`S_OFFDT`"-blokken og trykke `F1`. Dere vil da få opp en hjelpeside. Eventuelt vil jeg også minne om manualen "Working with STEP 7", som egentlig er ganske bra, alt tatt i betraktning.

1.5 Tutorial: Motorstyring

Motoren er en analog komponent, og verdien vi skriver til motoren vil gjøres om til en analog spenning i motordriveren. Den enkleste måten å skrive verdier til denne på, er ved å bruke en "`MOVE`"-blokk, som dere også finner i den venstre menyen.

For å styre motoren er det to ting vi trenger å tenke på. Først har vi selve motorspenningen, som dikterer hvor fort heisen vil bevege seg. Så har vi også et retningsbit, som vi kan bruke for å endre hvilken vei motoren går.

Når dere har kommet så langt som dere har nå, burde det meste av brukergrensesnittet være kjent, så jeg presenterer programmet direkte; i figur 12 ser dere et program som vil kjøre heisen opp hvis en hvilken som helst av oppknappene holdes inne - og ned hvis en hvilken som helst av nedknappene holdes inne.



Figur 12: Program for å manuelt styre heisen opp og ned.

Det er spesielt én ting dere bør legge merke til her. Inngangen på den ene `MOVE`-blokken er invertert. Dette gjøres ved å trykke på inngangen, og velge inverteringssymbolet i toppmenyen. Uten denne inverteringen, vil PLSen forsøke å sette forskjellige verdier til motoren samtidig. Ikke gjør det.

1.6 Tutorial: Funksjoner og nettverk

Det går helt fint an å putte all mulig logikk i ett "nettverk". Dette blir på en måte som å putte all koden din direkte i "`main`". Det *lar seg gjøre*, men du kommer til å bli kjeppjaget fra Gløs og puttet i gapestokk om du finner på noe sånt.

Frykt derimot ikke. Den sanne vei til et moralsk korrekt liv uten spirituell fordervelse er ikke langt unna. Alt du trenger å gjøre er å gruppere programmet ditt i logisk sammenhengende strukturer ved å benytte deg av funksjoner og nettverk.

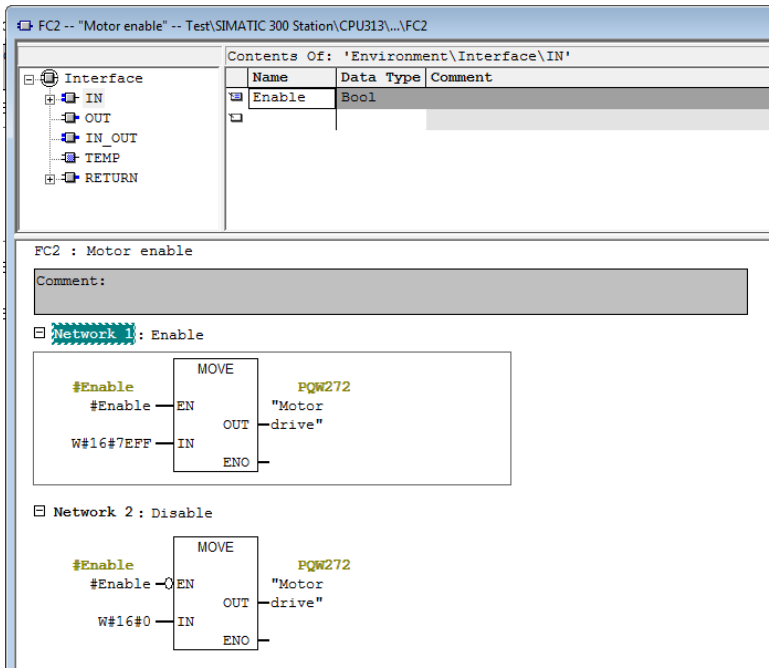
For å illustrere hvordan dere bruker en funksjon, la oss opprette en for å sette motorpådrag. Gå tilbake til vinduet i figur 4. Fra menyen øverst til venstre velger dere `Insert` → `S7 Block` → `Function`. Dette vil opprette en blokk kalt "`FC1`". Dobbeltklikk så på denne for å åpne den.

Øverst i det nye vinduet vil dere se en liste merket med "`Interface`". Denne listen spesifiserer grensesnittet til funksjonen. Dere kan spesifisere så mange innganger- og utganger dere trenger, men her trenger vi ikke mer enn én eneste inngang.

Trykk på symbolet "`IN`" under "`Interface`", og opprett et inngangssignal som dere kaller "`Enable`". Så oppretter dere ett ekstra nettverk inne i funksjonen ved å trykke på symbolet for dette i toppmenyen (se figur 5).

Er ikke funksjonsblokkene tilgjengelig i den vestre menyen? Da er dere sikkert i modien "`Structured Text`". Gå til `View` → `FBD` for å velge "`Function Block Diagram`".

Tanken her er at den nye funksjonen kun skal ta inn et boolsk signal som forteller den om motoren skal gå- eller ikke. I hver av de to nettverkene tar vi for oss ett av tilfellene; den første nettverket er ansvarlig for å starte motoren om `Enable`signalet er høyt, mens det andre er ansvarlig for å stanse motoren straks det går lavt. Dette er illustrert i figur 13.



Figur 13: Motorpådrag bestemt av egen funksjon.

Igjen så bør nok det meste her være noenlunde kjent; som tidligere bruker vi `MOVE`-blokker til å sette rikting spenning på motoren. Som før bruker vi også en invertert inngang på den ene `MOVE`-blokken for å bestemme når motoren skal være i ro.

Legg merke til måten vi bruker "lokale variabler" på inne i funksjoner. Da vi deklarerte "`Enable`" i `IN`-listen til funksjonen, så gjorde dette "`Enable`" til en variabel vi kan referere innad i denne funksjonen. Dette gjøres ved å legge til et hashtegn ("`#`") før navnet på variabelen, som sett på inn-gangene til `MOVE`-blokkene.

Før dere kan bruke denne funksjonen i `OB1`, må dere lagre den ved å trykke diskikonet i toppmenyen.

Når dere har laget ferdig funksjonen, kan dere åpne `OB1`, og finne den nylig opprettede funksjonen i den venstre sidemenyen. Her vil den hete "`FC1`". For å være sikker på at ting fungerer, kan dere prøve å styre motoren ved å bruke denne funksjonen istedenfor "hardkodingen" vi gjorde i oppgaven om motorstyring.

Dere må manuelt laste ned alle funksjoner dere lager til PLSen. Det gjøres ved å høyreklikke på `FC1` i vinduet vist i figur 4, og laste ned funksjonen før dere laster ned `OB1`.

1.7 Konklusjon av første labdag

Dette er slutten av første labdag. Med dette bør dere være greit forberedt til andre labdag, som tar for seg implementeringen av et forenklet heissystem.

Som sagt er det ingen egen godkjenning av denne dagen, og dere står helt fritt til å gå løs på neste labdag nå, om dere har mer tid til overs.

2 Oppgave andre labdag

Her skal dere implementere et forenklet heissystem. For å få godkjent skal dere demonstrere heissystemet til en studass. Spesifikasjonen er slik:

- Heisen skal kontinuerlig kjøre opp- og ned. Når den kommer til 4. etasje, skal den begynne å kjøre nedover; når den kommer til 1. etasje skal den kjøre snu og kjøre oppover.
- Riktig etasjeindikator skal settes når heisen ankommer en etasje. Når heisen er mellom etasjer, skal etasjeindikatoren til den siste etasjen heisen var i fortsatt lyse.
- For hver etasje heisen ankommer, skal den stoppe i 3 sekunder. Samtidig som heisen står stille, skal døren åpne seg. Døren skal lukke seg i det heisen kjører videre.
- Hvis obstruksjonsbryteren skrus på, skal heisen stanse med én gang, og stå i ro helt til bryteren skrus av igjen. Når bryteren skrus av skal heisen fortsette som før, i samme retning den var på vei i før "nødstoppen".
- Dere skal bruke 2 funksjoner i programmet deres utenom `OB1`. En for å sette motorpådrag, og en for å sette riktig etasjeindikator.

Ikke prøv å ta for dere hele spesifikasjonen samtidig, med mindre dere føler dere veldig selvsikre på deres nye PLS-ferdigheter. Her har dere en anbefalt fremgangsmåte:

1. Lag først en funksjon som tar input fra etasjesensorene, og setter riktig etasjeindikatorlys. Her må dere gjøre en enkoding for å tenne riktig lampe. Dette er beskrevet i seksjon 1.1.1. Når dette er gjort, kan dere programmere PLSen, og forsiktig dytte heisstolen opp og ned manuelt for å aktivere etasjesensorene og se at funksjonen fungerer som den skal.
2. Lag så en funksjon som styrer motorpådraget. Den late måten å gjøre dette på, er å kopiere funksjonen fra seksjon 1.6. For å teste denne, kan dere drive motorstyreblokken fra en vilkårlig knapp på heispanelet.
3. Finn en eller annen måte å sette riktig retning på motoren når heisen kommer til endeetasjene. Implementer så dette i et eget nettverk i `OB1`.

4. Velg en fornuftig timer, og bruk dette til å stanse heisen og åpne dørene hver gang den kommer til en etasje. Enn så lenge trenger dere ikke tenke på obstruksjonsbryteren. Det er lurt å alltid teste programmet før dere går videre, særlig nå.

5. Til slutt utvider dere programmet ved å stanse motoren så lenge obstruksjonsbryteren er aktivert.

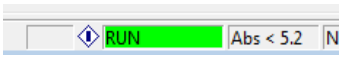
3 Hint

Ser ikke funksjoner ut som de skal?

Det kan hende dere har en annen fremstilling av programmet deres enn funksjonsblokkdiagrammer. Gå til `View` → `FBD`.

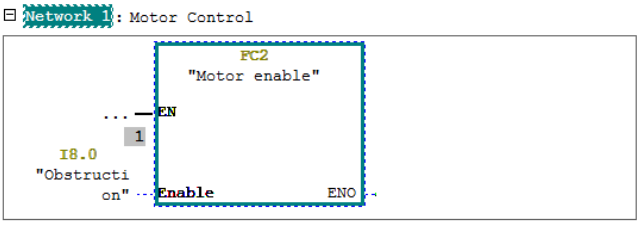
Usikre på hvilke signaler som går hvor?

Åpne debuggeren. Dette gjøres ved å trykke på "brilleikonet" i toppmenyen illustrert i figur 5. Dere vil da se et par endringer i programgrensesnittet. Først vil dere bytte til "online"-modus på PLSen, og dere vil se statusen nede i høyre hjørne. Se figur 14. Dere vil også se hvilke innganger- og



Figur 14: Statusen til PLSen.

utganger som har hvilke verdier mens PLSen kjører. Aktive blokker vil også få et blått omriss. Se figur 15.



Figur 15: Debugger i SIMATIC Manager.

Klager PLSen på for lite minne?

Prøv å dytte den blå bryteren på PLSen ned i "`MRES`". Hold den der til "`STOP`" begynner å blinke sakte; omlag 3 sekunder. Deretter kan dere prøve å programmere på nytt.

Ser hovedvinduet veldig forskjellig ut fra oppgaveteksten?

Prøv `View` → `Large Icons`.

Ingen kommunikasjon med PLSen?

Sjekk grensesnittet mellom PLS og datamaskin. Vi bruker et grensesnitt som heter "Multi-Point Interface" via USB til å programmere PLSen. Gå til `Options` → `Set PG/PC Interface`, og velg "`PC Adapter.MPI.1`".

Unable to copy?

PLSen kan ikke kjøre samtidig som dere programmerer den. Sett den blå bryteren på PLSen i "`STOP`" mens dere programmerer, og dytt den deretter tilbake til "`RUN`" for å kjøre.

Finnes det noen "Stack Exchange" for PLS?

Det sitter en hel del industriknarker på `support.industry.siemens.com` og svarer på spørsmål, om dere skulle ha behov for mer spesifikke ting enn det som står i "Working with STEP 7" - som jeg nok en gang vil minne om at er en ganske så bra manual, sånn helt egentlig, liksom.

Hva er forskjellen på en funksjon- og en funksjonsblokk?

En funksjon (`FC`) er i utgangspunktet en "ren" operasjon, mens en funksjonsblokk (`FB`) rett og slett er en funksjon som har en datablokk (`DB`) tilknyttet seg. Dette gjør at en funksjonsblokk kan lagre sin egen tilstand, som også introduserer økt programkompleksitet. "Fy" hvis du bruker den der du ikke har behov for en.

Hvorfor trenger dere ikke "regtek" på heismotoren?

Vi trenger ikke noen form for tilbakekobling på heismotoren, fordi motoren har et gir med utveksling 1:100. Dette gir et så stort moment at vi fint greier oss uten. I tillegg gir dette oss så stor treghet at heisen ikke kommer til å "gli forbi" etasjer når vi stanser motoren. Artig trivia :)