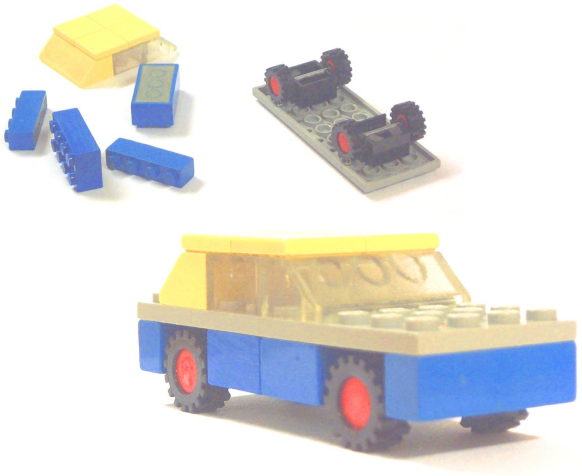
Styrning av legofabrik

Denna version: 2023-03-23



REGLERTEKNIK
AUTOMATIC CONTROL
LINKÖPING

Namn: _____

Personnr: _____

Datum: _____

Godkänd: _____

Innehåll

Introduktion

Denna laboration handlar om *sekvensstyrning*, vilket handlar om att skapa en grupp instruktioner som ska utföras i en viss sekvens för att lösa ett givet problem. Sekvensstyrning är väldigt användbart för att reglera produktionskedjor inom industrin.

Syftet med denna laboration är dels att ge insikt i problem kring sekvensstyrning i allmänhet, dels att visa hur ett sekvensstyrningsproblem kan lösas med hjälp av industriella verktyg i synnerhet. Er uppgift är att i en grupp om två studenter konstruera och implementera ett program som styr en bilfabrik i LEGO. Ni kommer att utveckla kod för en av två fabriker: fabrik A (Fig. 2) eller fabrik B (Fig. 3)

Bilfabriken finns färdig i Laboteket och är inkopplad på en PLC som i sin tur kommunicerar med en vanlig dator där själva programutvecklingen sker. För att lösa denna typ av problem används idag vanligen PLC-styrsystem (eng. Programmable Logic Controller). Dessa utgörs av relativt enkla men robusta datorsystem. I laborationen ska ni programmera en PLC med hjälp av funktionsdiagram (GRAFCET) och reläscheman.

Laborerandet sker till stor del självständigt på icke schemalagd tid, varefter en fungerande bilfabrik uppvisas för laborationsassistenten vid ett examinationstillfälle. Bilfabriken ska då fungera enligt kravspecifikationen i avsnitt 6.2. Vid denna examination ska ni dessutom kunna förklara och motivera lösningen samt självkritiskt reflektera över eventuella brister och svaga punkter i systemet. Detaljer rörande examinationen kan ni se i avsnitt 6.

Detta dokument är organiserat på följande vis. Avsnitt 1 beskriver i detalj varje station i fabrik A och B. I avsnitt 2 beskrivs programvaran i generella drag och i avsnitt 3 beskrivs sedan mer praktiskt hur den används. Ett exempel på detta ges i avsnitt 4. Slutligen beskrivs examinationsmomenten i avsnitt 6 och förberedelseuppgifterna i avsnitt 7.

1 Laborationsutrustning

Varje laborationsgrupp om två studenter förfogar över:

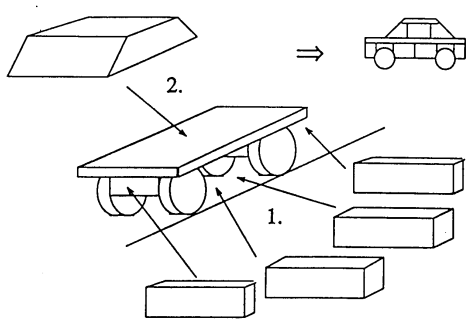
1. En *halv* bilfabrik i LEGO (benämnd LEGO A eller LEGO B).

2. En styrenhet som består av en industriell PLC (ABB PM571) på vilken styralgoritmen ska exekveras, nätaggregat, relän för motorstyrning samt elektronik för spänningsanpassning och för förstärkning av sensorsignaler.

3. En vanlig dator med utvecklingsverktyg för PLC-program (*CoDeSys*).

I legofabrikerna finns det sensorer med namn Sn där n är ett heltal och motorer med namn En. Dessa sensorer och motorer är kopplade till in- och utgångar på PLC:n så att man i styrprogrammet kan läsa av värdet på en sensor genom att titta på variabeln med samma namn (t.ex. S0 eller S3). Likaså kan man köra en motor framåt genom att ettställa variabeln EnF (t.ex. E1F) och bakåt genom att ettställa EnB (t.ex. E1B). I några fall heter motorvariablerna EnU (körning uppåt) och EnD (körning nedåt) eftersom det ger en bättre bild av deras funktion och om en motor bara kan köras åt ett håll heter dess variabel En.

En komplett legofabrik består av *två* separata fabrikshalvor: *LEGO A* som monterar bilens underrede (chassibitar) samt *LEGO B* som monterar bilens tak. Styrsystemen för LEGO A och LEGO B är oberoende av varandra och kommunicerar endast med en optisk länk för att synkronisera produktionen. Ni väljer redan från början vilken fabrikshalva ni ska programmera, dvs. *antingen* LEGO A *eller* LEGO B. De två fabrikshalvorna är dock ganska lika så det är samma principiella styrproblem som ska lösas i LEGO A som i LEGO B.

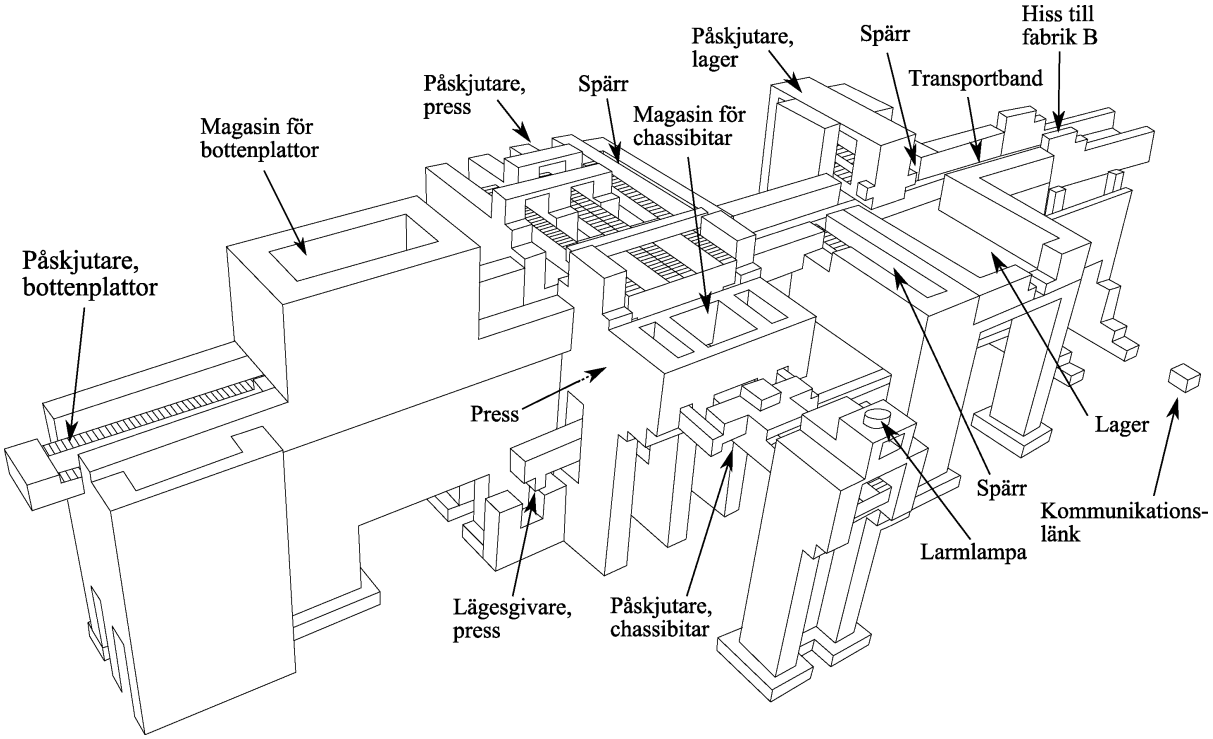


Figur 1: Två arbetsmoment ger en färdig bil: 1. Chassibitar monteras på en bottenplatta i LEGO A. 2. Tak monteras i LEGO B.

1.1 Beskrivning av de två fabrikerna

LEGO A

Figur 2 visar LEGO A. I denna figur går tillverkningsprocessen i riktning från vänster till höger och de olika momenten i tillverkningen beskrivs enligt följande lista.



Figur 2: LEGO A: Montering av chassibitar.

1. Transportbandet (E0) sträcker sig genom hela LEGO A och är en gemensam resurs som förflyttar bilar mellan de olika stationerna. Transportbandet är konstruerat så att det kan gå kontinuerligt och oberoende av om någon spärr är utskjuten eller ej. Detta gör att stationerna längs transportbandet i princip kan arbeta oberoende av varandra, vilket gör parallell bearbetning möjlig. Transportmotorn kan endast köras åt ett håll, dvs.

transportera bilarna framåt.

2. Det finns en alarmlampa (ALARM) monterad ovanpå pressen. Denna lampa kan användas för att göra operatören uppmärksam på att något behöver åtgärdas i fabriken.

3. Den första stationen är ett magasin för bottenplattor med en påskjutare (E1) för att skjuta ut bottenplattor på transportbandet. Påskjutarmotorn kan köras framåt (påskjutning av platta) och bakåt. Att påskjutaren är i sitt främre ändläge detekteras av S0=1 och i sitt bakre läge av S1=1. Ett tomt magasin detekteras av S2=1.

4. Den andra stationen är pressen där de fyra chassibitarna ska pressas fast på bottenplattan underifrån. Här finns en spärr (E5) som stoppar bilen i rätt läge. Att spärren är i sitt främre ändläge (och kan stoppa en bil) detekteras av S10=1 och i sitt bakre läge av S11=0. Närvaron av en bil indikeras av sensorn S3 som är placerad strax innan pressen. I presstationen finns en kloliknande påskjutare (E4) som kan skjuta bort bilen från transportbandet till pressen och även dra tillbaka den. Att denna påskjutare är i sitt främre ändläge detekteras av S4=1 och i sitt bakre läge av S5=0. Det finns även ett magasin för chassibitar och en påskjutare (E2) som kan ladda in bitar i pressen. Att denna påskjutare är i sitt främre ändläge detekteras av S6=1 och i sitt bakre läge av S7=0. Den bör förbli framskjuten under hela pressförloppet för att garantera att bitarna pressas rakt upp. Pressen måste vara i sitt nedre läge för att det ska gå att skjuta ut bitar. Pressen pressar fast bitar underifrån och drivs av motorn E3. Denna motor kan bara rotera i en riktning och har inte några egentliga ändlägen. Att pressen befinner sig i sitt nedre läge detekteras dock av S8=0. Det finns även en rotationsgivare (S9) som kommer att beskrivas mer senare i Sec. 3.5.

5. Den tredje stationen är en diagnosstation vid vilken en bil kan stoppas av en spärr (E6). Att spärren är i sitt främre ändläge detekteras av S17=1 och i sitt bakre läge av S18=0. Närvaron av en bil kan detekteras av S13=0. I diagnosstationen finns även fyra sensorer (S12, S14, S15 och S16) som blir ettställda om motsvarande legobit saknas på bilen.

6. Den fjärde stationen är en lagerstation där defekta bilar (som saknar någon chassibit) kan föras åt sidan för vidare manuell bearbetning. Även här kan en bil stoppas av en spärr (E8). Att spärren är i sitt främre ändläge detekteras av S22=1 och i sitt bakre läge av S23=1. Närvaron av en bil vid lagerstationen kan detekteras med sensorn S19 som är placerad strax innan stationen. Påskjutaren (E7) som kan föra bilar åt sidan har ett främre ändläge som detekteras av S20=0 och ett bakre läge som detekteras av S21=0.

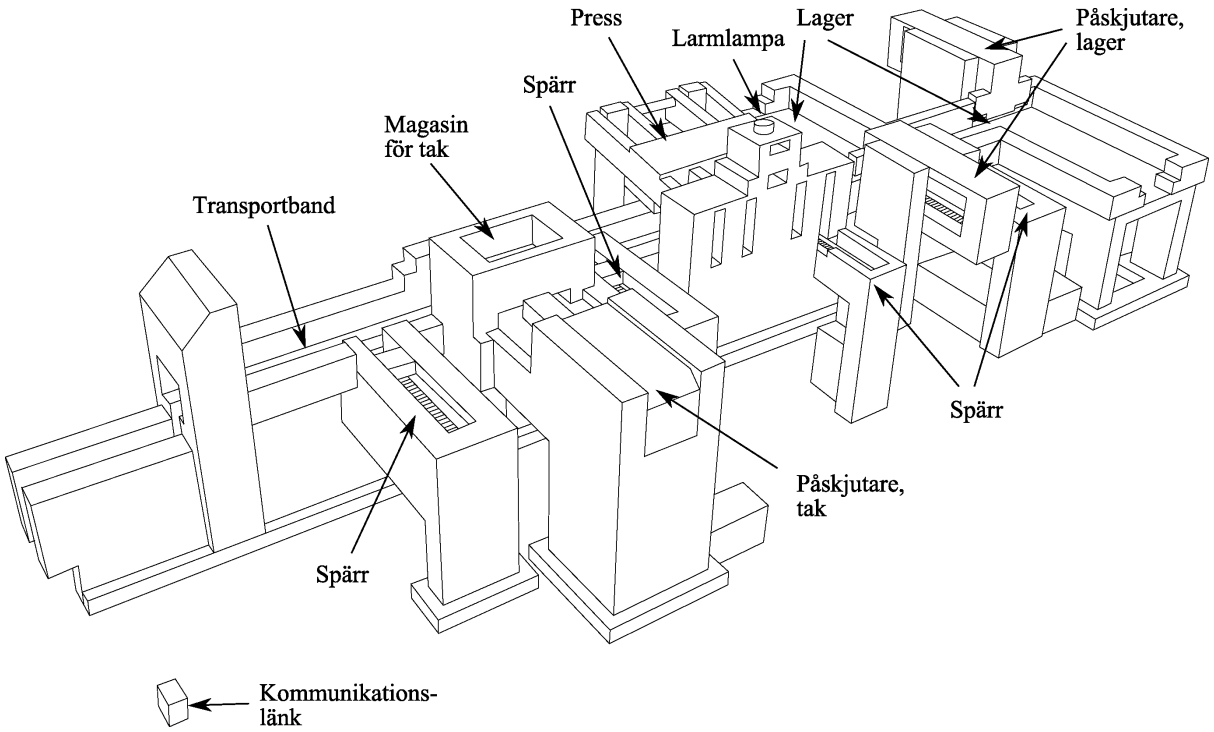
7. Den femte och sista stationen är en hiss som levererar bilar till LEGO B. Närvaron av en bil här kan detekteras av sensorn S24 som är placerad strax innan stationen. Hissen (E9) har ett övre ändläge som detekteras av S25=1 och ett nedre som detekteras av S26=0.

8. För kommunikation med LEGO B finns en sensor S27 som blir 1 när LEGO B är redo att ta emot en bil.

LEGO B

LEGO B förses med chassin från LEGO A som anländer från vänster i Figur 3. I denna figur går tillverkningsprocessen sedan i riktning från vänster till höger och de olika momenten i tillverkningen beskrivs enligt följande lista.

1. Det finns en lysdiod (LOBBY_READY) som ska användas för att kommunicera med LEGO A. Denna lysdiod ska tändas när lobbyn är redo att ta emot en bil.



Figur 3: LEGO B: Montering av tak.

2. Transportbandet (E0) sträcker sig genom hela LEGO B och är en gemensam resurs som förflyttar bilar mellan de olika stationerna. Transportbandet är konstruerat så att det kan gå kontinuerligt och oberoende av om någon spärr är utskjuten eller ej. Detta gör att stationerna längs transportbandet i princip kan arbeta oberoende av varandra, vilket gör parallell bearbetning möjlig. Transportmotorn kan endast köras åt ett håll, dvs. transportera bilarna framåt.

3. Det finns en alarmlampa (ALARM) monterad ovanpå pressen. Denna lampa kan användas för att göra operatören uppmärksam på att något behöver åtgärdas i fabriken.

4. Den första stationen är lobbyn dit bilchassin anländer från LEGO A. Här finns en spärr (E1) som kan se till att en bil inte åker vidare förrän nästa station är redo att ta emot den. Att spärren är i sitt främre ändläge (och kan stoppa en bil) detekteras av S2=1 och i sitt bakre läge av S1=0. Närvaron av en bil vid stationen detekteras av S0=0.

5. Den andra stationen är ett magasin för biltak. En bil kan stoppas här med en spärr (E3) vars främre ändläge detekteras av S7=1 och vars bakre läge av S8=1. Närvaron av en bil vid stationen detekteras av S3=0. I stationen finns en påskjutare (E2) som kan skjuta ut ett tak på ett tomt chassi. Att påskjutaren är i sitt främre ändläge detekteras av S5=0 och i sitt bakre läge av S4=1. Ett tomt magasin detekteras av S6=1.

6. Den tredje stationen är pressen där taket ska pressas fast på bilen. Här finns en spärr (E5) vars främre ändläge detekteras av S12=1 och vars bakre läge av S13=1. Närvaron av en bil vid stationen detekteras av S9=0. Pressen drivs av motorn E4 som kan köras åt två håll och vars övre läge detekteras av S10=1. Det finns även en rotationsgivare (S11) som kommer att beskrivas mer senare i Sec. 3.5.

7. Den fjärde stationen är en lagerstation där defekta bilar (som saknar tak) kan föras åt

sidan för vidare manuell bearbetning. Även här kan en bil stoppas av en spärr (E7). Att spärren är i sitt främre ändläge detekteras av S17=1 och i sitt bakre läge av S18=1. Närvaron av en bil vid lagerstationen kan detekteras med sensorn S14 som är placerad strax innan stationen. Påskjutaren (E6) som kan föra åt sidan bilar har ett främre ändläge som detekteras av S15=0 och ett bakre läge som detekteras av S16=0.

8. Den femte och sista stationen är även den en lagerstation där färdigproducerade och korrekta bilar kan lagras i väntan på leverans till kund. Närvaron av en bil vid denna lagerstation kan detekteras med sensorn S19 som är placerad strax innan stationen. Påskjutaren (E8) som kan föra ut bilar i lagret har ett främre ändläge som detekteras av S20=0 och ett bakre läge som detekteras av S21=0.

1.2 Styrenhet

Det styrsystem som ni ska utveckla i den här laborationen ska exekveras i en styrenhet som består av följande delar

- En industriell PLC (ABB PM571).
- Ett nätaggregat som förser PLC:n med 24V likspänning.
- En spänningsomvandlare som ger 9V likspänning.
- Relän som är kopplade så att man kan köra motorerna åt två håll med hjälp av de binära styrsignalerna från PLC:n.
- Ett kretskort som dels förstärker sensorsignalerna och dels anpassar strömmen till lysdioderna i fabrikerna.
- Ett nödstopp som bryter strömmen till motorerna när det trycks in. Nödstoppet är även kopplat till PLC:n så att det avbryter exekveringen av programmet (se avsnittet om kompilering och körning på sidan 12).

Styrenheterna är redan kopplade till både fabrikerna och datorerna när ni kommer till laborationen så ni behöver inte själva utföra något kopplingsarbete.

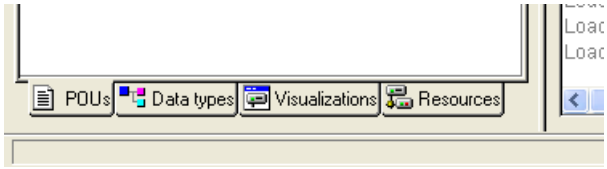
2 Mjukvara

CoDeSys är ett program med vilket man kan utveckla programkod och operatörsgränssnitt till PLC:er. Ett exempel på hur programmeringsmiljön i CoDeSys ser ut visas i Figur 7. CoDeSys har bland annat stöd för funktionsdiagram (**SFC**, Sequential Function Chart), reläscheman (**LD**, Ladder Diagram), blockdiagram (FBD, Function Block Diagram) och strukturerad text (ST, Structured Text). Man har full frihet att för olika uppgifter i ett projekt välja det programspråk som är mest lämpligt. I denna laboration rekommenderar vi dock att ni håller er till funktionsdiagram och reläscheman eftersom de är programspråk som många använder och som ger relativt tydliga och lättöverskådliga program. Ett komplett PLC-program i CoDeSys kallas för ett projekt och består av flera delar som man kan hitta under någon av de fyra flikarna POUs, Data types, Visualizations och Resources i den vänstra delen av fönstret (se Figur 4).

Observera att vissa delar av projektet redan har implementerats, som Data types, Visualizations och Resources. **Er uppgift är att skriva delarna Funktionsdiagram(SFC) och reläscheman(LD).** Håll i åtanke att de två delarna komplimenterar varandra och att det är viktigt att förstå hur de interagerar med varandra för att kunna klara av uppgiften.

OBS.

Ni behöver långtifrån lära er allt om CoDeSys. I de programskelett ni förslagsvis utgår från finns de flesta grundinställningar redan gjorda så att ni redan från början kan koncentrera er på programmeringen. Detta avsnitt är till för att ni ska kunna komma igång snabbt utan att behöva läsa någon manual. Om ni senare märker att ni behöver komplettera med mer uttömmande information finns det mycket att läsa under hjälpmenyn i programmet.



Figur 4: Flikarna i den nedre vänstra delen av CoDeSys-fönstret (jfr. Figur 7).

2.1 Reläscheman (LD)

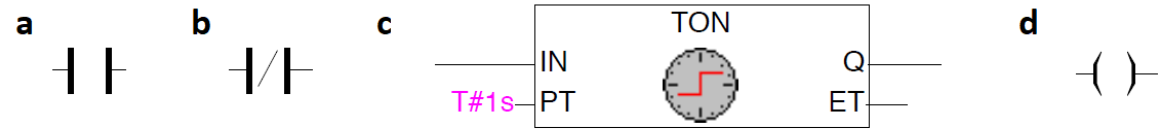
Reläscheman hittar ni i ”Automatic_Conditional”. Dessa används för att implementera logik som, *Öm detta villkor är uppfyllt, så ska denna variabel sättas till 1"*. De är uppbyggda av ett antal logiska villkor som grafiskt representeras av en linje i CoDeSys (se till höger i Figur 12). Variabeln till höger på linjen sätts till 1 om alla villkoren till vänster är uppfyllda. I praktiken bör ni endast behöva ett fåtal symboler för att skapa era egna reläscheman. Dessa presenteras i Figur 5. Generellt bör ni ha två avsikter då ni designar era reläscheman:

- Att se till så att ni endast aktiverar en motor om den inte redan är i sitt ändläge, som i Figur 6(a). Håll i åtanke att programet kommer att krascha om ni inte gör detta. Alla motorer som utför förflyttningar mellan väldefinierade ändlägen är försedda med ändlägesgivare. Om en motor stoppas i sitt ändläge utan att spänningen bryts, stiger strömmen snabbt till ett värde som kan medföra överhettning. Därför ska styrprogrammen vara skrivna så att spänningen till motorn omedelbart bryts då den aktuella ändlägesgivaren aktiveras. Var uppmärksam på att vissa ändlägesgivare kvitterar med 0 (aktivt låg) medan andra kvitterar med 1 (aktivt hög). Vilka signaler som är aktivt låga respektive höga anges i appendix A.1/B.1 (LEGO A/LEGO B).
- Att sätta värdet för övergångsvillkor(se mer i avsnitt 2.2) i funktionsdiagramet, som i Figur 6(b).

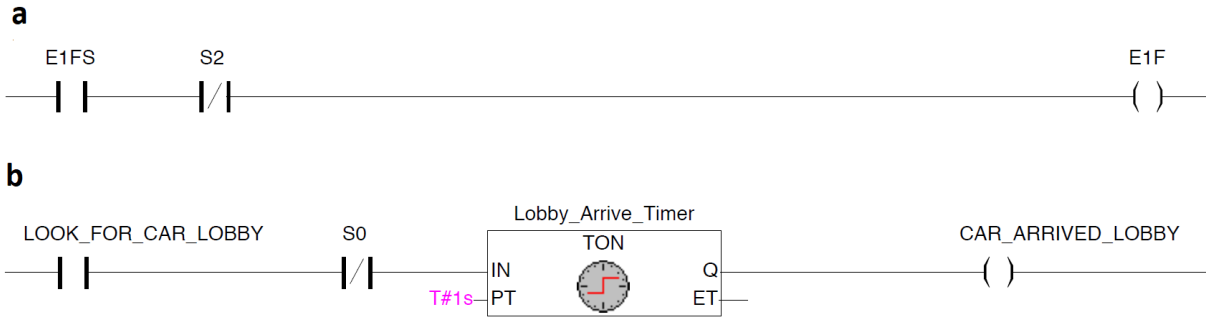
2.2 Funktionsdiagram (SFC)

Funktionsdiagram (SFC) beskriver en grupp instruktioner som ska repeteras. I Figur 7 kan ett enkelt exempel ses. I exemplet kan två huvudsakliga delar ses: de horisontella strecken mellan stegen, som representerar övergångsvillkor, och instruktionerna tillhörande varje steg. Från början befinner sig programmet i steget Init, som inte har några tillhörande instruktioner, dvs. inget kommer att hända. När sedan övergångsvillkoret RUN blir 1 kommer programmet att gå vidare till nästa steg (Step1). Där kommer det att förbli fram till dess att dess övergångsvillkor D1 uppfylls. Under tiden som programmet befinner sig is Step1 så kommer det att börja utföra de två instruktioner som är associerade med det blocket. De instruktionerna som finns att välja bland är

- N (Non-stored) används för att sätta en variabel till 1. Observera att variabeln omedelbart kommer att återgå till att vara 0 efter att programmet har lämnat det blocket.



Figur 5: De huvudsakliga symbolerna i ett reläschema. (a), (b) och (c) är logiska villkor som placeras på vänstra sidan av strecket (nätverket). Villkoret (a) är uppfyllt då dess insignal är 1, (b) är uppfyllt då dess insignal är 0, och (c) är uppfyllt då dess insignal är uppfyllt under en viss (här betyder T#1s 1 sekund). Symbolen (d) sätts istället på högra sidan av nätverket och sätter värdet på sin variabel till 1 om alla villkor till vänster är uppfyllda.



Figur 6: Två exempelnätverk från reläschemat. (a) Det första nätverket aktiverar E1F (motorn som stänger grind 1) om E1FS är 1 och S2 är 0 (dvs. då det finns ett objekt framför sensorn). (b) Det andra nätverket sätter variabeln CAR_ARRIVED_LOBBY till 1 om LOOK_FOR_CAR_LOBBY är 1 och S0 har varit 0 i åtminstone 1 sekund. Timern är här nödvändig eftersom det enkelt kan komma någon liten störning som gör att sensorn ser något under en kort tid, vilket annars skulle sätta variabeln till 1.

- L (tidsbegränsade handlingar, time-Limited) används för att sätta en variabel till 1 under en viss begränsad tid, varefter variabeln återgår till att vara 0.
- D (tidsfördröjda handlingar, time-Delayed) används för att sätta en variabel till 1 efter en viss fördröjning (den kommer vara 0 fram till dess).

I detta exempel kommer alltså E1FS bli 1 omedelbart då programmet kommer till Step1, och D1 kommer bli 1 en sekund efter att programmet kommit till Step1. Observera att när man använder tidsfördröjda eller tidsbegränsade handlingar måste man också skriva in en tid till höger om handlingstypen. Tiden ska vara angiven på formatet ”T#1s”, ”T#700 ms”, etc.

Då D1 blir till 1 kommer övergångsvillkoret att vara uppfyllt och vi kommer därmed att gå vidare till nästa steg Step3. I Step3 kommer E1BS vara 1, och vi kommer förbli i Step3 tills S2 blir 1. På detta sätt kommer programmet att utföra en instruktion i taget, tills det når botten på koden. Där kan ni se den lilla triangeln där det står "Init". Detta menar att programmet kommer att *hoppa* till blocket "Init", vilket är i början av programmet, varefter programmet börjar om från början. Det är också möjligt att välja att hoppa till något annat steg i programmet, vilket kan vara användbart i vissa situationer.

OBS.

Använd endast instruktionerna **N**, **L** och **D** i era funktionsdiagram. Andra instrutioner såsom **S** (Set), **R** (Reset) och **C** (Conditional), fungerar inte. CoDeSys har, liksom många andra program för sekvensstyrning, inte direkt stöd för villkorliga handlingar i funktionsdiagram. Dessa måste istället implementeras i något annat programspråk, till exempel i ett reläschema.

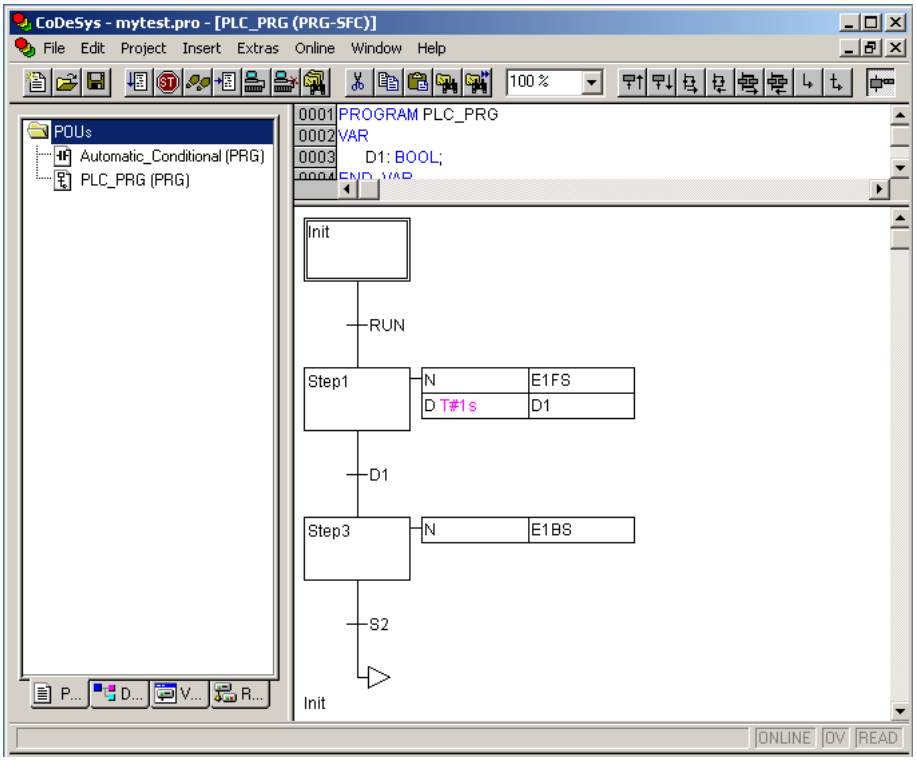
Håll i åtanke att CoDeSys kommer att lämna steget så fort som övergångsvillkoret är uppfyllt (även om vissa instruktioner i blocket innehåller tidsfördröjda handlingar). Det är därför viktigt att försäkra sig om att det finns tid att utföra alla instruktioner inom varje block innan nästa övergångsvillkor blir sant. Detta görs bäst genom att använda sig av ett övergångsvillkor som blir sant först när instruktionerna har uppfyllts.

I vissa fall kan man även ha förgreningar i övergångsvillkoren, som exempelvis i Förberedelseuppgift 2. I sådana fall är det viktigt att tänka på att aldrig ha komplementära övergångsvillkor (som S1 och NOT S1) eftersom programmet då omedelbart kommer lämna det nuvarande steget utan att utföra instruktionerna associerade med det steget.

Tänk på att ert program kommer bestå av flera filer (som alla motsvarar varsin station) som kommer köras samtidigt. De olika funktionsdiagrammen i appendix A.3/B.3 svarar mot fabrikshalvans olika stationer och exekveras helt parallellt. För att det inte ska bli kaos i fabriken krävs att stationerna är synkroniserade. Utan synkronisering kan det exempelvis hända att en station släpper iväg en bil till den efterföljande stationen trots att denna inte är klar med sin bearbetning av föregående bil. Synkroniseringen sker med globala hjälpvariabler. I appendix A.1/B.1 finns ett antal fördefinierade hjälpvariabler listade och vid behov kan ni deklarera fler. En mycket enkel synkroniseringsprocedur är att en station meddelar föregående station så snart den är redo att ta emot en bil för bearbetning genom att sätta en hjälpvariabel (t.ex. PRESS_READY) till 1. Så snart stationen erhållit den bil som då levereras, sätts hjälpvariabeln till 0, och förblir 0 ända tills stationen är klar med bearbetningen och redo att ta emot ytterligare en bil, se Figur 11 för ett generellt exempel.

Bra kod

I denna laboration kommer ni att implementera många olika villkor. Det är därför viktigt att använda de förskrivna variablerna som finns tillgängliga i tabellerna A.1/B.1 eftersom dessa



Figur 7: Ett exempel på ett funktionsdiagram i CoDeSys.

kommer att göra koden mycket mer lättläslig. Som ett exempel är det mycket svårt att förstå vad övergångsvillkoren ”S3 AND NOT S1” eller ”S14 AND S16” betyder. Istället är det bra att använda intuitiva namn som ”LOOK_FOR_CAR_LOBBY” eller ”CAR_ARRIVED_STORAGE1”, och sedan lägga till nätverk i reläschemat som gör så att

”LOOK_FOR_CAR_LOBBY” blir 1, om ”S3 AND NOT S1” uppfylls
”CAR_ARRIVED_STORAGE1” blir 1, om ”S14 AND S16” uppfylls.

Detta kräver ett visst extra arbete initialt men det kommer också spara er en del tid senare då ni får problem och ska försöka felsöka ert program. **Generellt är ett lättläst funktionsdiagram ett tecken på ett välgjort projekt**.

3 Implementering

Det är nu dags att kolla på en del praktiska aspekter av implementationen.

3.1 Reläscheman (LD)

Som nämnts tidigare kallas varje streck i reläschemat för ett *nätverk*. För att lägga till fler nätverk till reläschemat högerklickar ni i reläschemat och väljer *add Network*.

Tidsfördröjningar i reläscheman

Ibland kan det vara smidigt att tidsfördröja signaler i ett reläschema. I CoDeSys kan detta göras genom att man lägger in en timer (TON, Timer ON) genom att klicka på blocket med en klocka i verktygsfältet. Timerblocket som då dyker upp måste ges ett unikt namn, vilket man gör om man skriver något lämpligt istället för frågetecknen ovanför blocket. När man anger ett nytt namn får man upp en dialogruta som man bara ska klicka OK i.

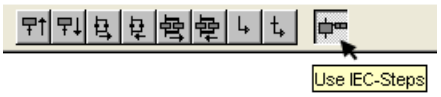
Ett TON-block har två ingångar. IN är den booleska ingång till vilken ingångsvillkoret ska kopplas. PT (PresetTime) anger fördröjningstiden, vilken ska anges på formatet ”T#1s”, ”T#700 ms”, etc. Q är den intressanta booleska utgången, vilken typiskt kopplas till en spole som påverkar en viss variabel. TON-blocket fungerar så att Q blir ettställd när IN har varit ettställd minst den tid som har angivits vid PT. Ett enkelt reläschema som innehåller en timer visas i Figur 6(b).

OBS.

Utmed bilens väg genom fabriken finns ett antal fotosensorer utplacerade för att hålla reda på bilens läge. Dessa sensorer har relativt låg precision – de utgör närvarodetektorer snarare än positionsdetektorer. Detta förklarar behovet av transportbanans mekaniska spärrar, vilka är nödvändiga för att uppnå tillräcklig precision vid positioneringen inför de olika arbetsmomenten. För att bilen ska hinna nå fram till spärren ordentligt, samt för att filtrera bort eventuella störningar, måste signalerna från fotosensorerna fördröjas med en tillslagsfördröjd timer, se till exempel Figur 6.

3.2 Funktionsdiagram (SFC)

För att funktionsdiagrammen i CoDeSys ska fungera enligt standarden IEC 61131-3 måste man klicka på ”Use IEC-Steps”-knappen längst till höger i verktygsfältet i den övre delen av CoDeSys-fönstret när funktionsdiagrammet har öppnats. Denna knapp visar ett steg med en handling kopplad till sig (se Figur 8).

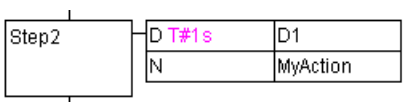


Figur 8: Verktygsfältet när funktionsdiagram editeras i CoDeSys (jfr. Figur 7).

Funktionsdiagrammet kan därefter editeras genom att man lägger till nya steg eller förgreningar med de övriga knapparna i högra delen av verktygsfältet. Varje steg i ett funktionsdiagram ska ha ett unikt namn som kan skrivas in genom att man klickar på steget. Stegnamnen får inte innehålla svenska tecken. Befintliga steg och övergångar kan tas bort genom att man markerar dem och klickar på Delete-tangenten. För att funktionsdiagrammet inte ska bli felaktigt är det bara möjligt att ta bort ett steg tillsammans med övergången före eller efter steget genom att markera dem med Shift-tangenten intryckt.

Handlingar i funktionsdiagram

Handlingar (eng. actions) kan kopplas till IEC-steg genom att man högerklickar på steget och väljer *Associate Action*. Ett nytt IEC-steg har alltid en handling kopplad till sig och man kan maximalt ha nio handlingar kopplade till samma steg. En handling illustreras av en ruta bredvid steget (se Figur 9). I den vänstra halvan av denna ruta anges det vilken typ av handling det är och i den högra halvan finns namnet på den variabel som ska aktiveras i det aktuella steget.



Figur 9: Ett exempel på en handling i ett funktionsdiagram i CoDeSys (jfr. Figur 7).

Ett generellt tips är att undvika lagrade handlingar så långt det är möjligt eftersom dessa

försämrar programmets läsbarhet. Handlingar kan tas bort från ett steg genom att man högerklickar på steget och väljer *Clear Action/Transition*.

Övergångsvillkor i funktionsdiagram

Ett övergångsvillkor (eng. transition condition) kan kopplas till en viss övergång genom att man skriver in ett logiskt uttryck till höger om övergången. Ett exempel på ett övergångsvillkor är

(NOT S2) AND TimeOut,

där S2 och TimeOut är variabler i PLC-programmet. Nyckelord som NOT, AND och OR ska vara skrivna med versaler men programmet ändrar dem automatiskt till det om ni skriver in dem med gemener.

3.3 RL och SFC

Nya variabler

När man anger en ny variabel i ett funktionsdiagram eller ett reläschema kommer det upp en dialogruta där man kan ställa in hur den nya variabeln ska deklareras. Normalt är det bara att klicka OK i denna ruta. Variabeldeklarationerna som hör till ett visst delprogram visas om man dubbelklickar på det aktuella programmet under POUs-fliken. Globala variabler deklareras under Resources-fliken. Observera att flera globala variabler redan har definierats, se Tabell A.1 eller Tabell B.1 för de globala variablerna deklarerade för station A respektive B. Som en tumregel kan det vara bra att använda alla de globala variablerna.

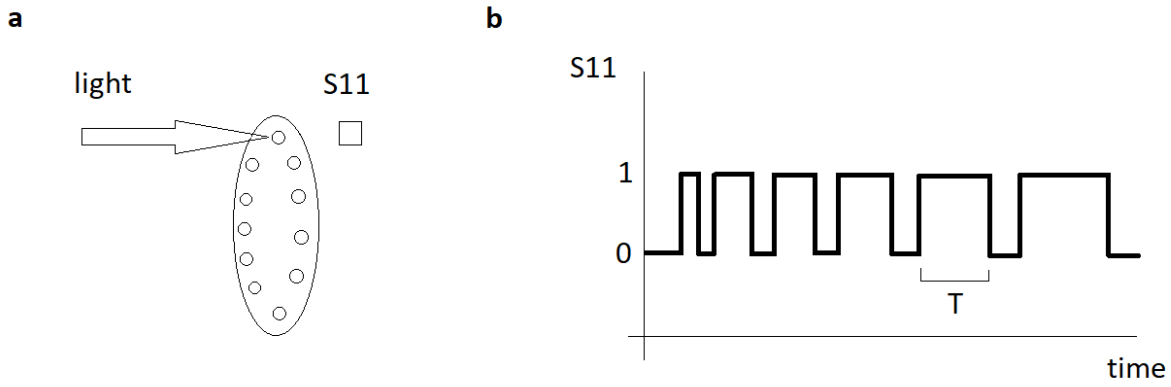
Hantering av gemensamma resurser

Transportbandet (E0) och alarmlampan (ALARM) är utgångar från PLC:n som man behöver kunna styra från olika delprogram. För att man inte ska hamna i fall där ett program vill ettställa E0 eller ALARM samtidigt som ett annat vill nollställa den finns det ett antal globala hjälpvariabler som man bör använda istället. Om en station vill köra transportbandet ska den ettställa en av variablerna E0A, E0B, ..., E0F och bandet kommer då att köras oavsett vad övriga variabler har för värden, dvs. det räcker att en av dessa variabler är ettställd för att bandet ska köras. Genom att det finns en bandstyrningsvariabel per steg där bandet ska köras undviker man konflikter i programmet. På motsvarande sätt finns det hjälpvariabler som styr när alarmlampan ska tändas. Dessa hjälpvariabler heter BASE_ALARM och DEFECT_ALARM i LEGO A respektive ROOF_ALARM, STORAGE1_ALARM och STORAGE2_ALARM i LEGO B.

3.4 Kompilering och körning

Innan man kan köra ett projekt på PLC:n måste dess kod kompileras. Detta gör man genom att välja *Project/Build* i menyraden (eller genom att klicka på F11-tangenten). Därefter väljer man *Online/Login* (Alt+F8) för att kopiera programmet till PLC:n och *Online/Run* (F5) för att starta det. I legoprogrammet finns det ett grafiskt gränssnitt med vilket man kan köra enskilda motorer eller starta det automatiska programmet (när det är skrivet). Under körning kan man i gränssnittet även se vilka sensorer som är aktiverade, något som kan vara mycket användbart vid felsökning. Man kan också följa exekveringen av ett enskilt delprogram genom att dubbelklicka på det under POUs-fliken. Exekveringen av ett program stoppas med *Online/Stop* (Shift+F8) och det är också ofta lämpligt att nollställa det med *Online/Reset* så att det befinner sig i sitt starttillstånd nästa gång man startar det. Onlineläget lämnas med *Online/Logout* (Ctrl+F8).

Det finns ett motorskydd inbyggt i programmet som ska se till att motorerna inte förstörs om de körs trots att de är i sina ändlägen. Om motorskyddet aktiveras låses programmet



Figur 10: Sketch på hur pressen fungerar. (a) Pressen är kopplad till en cylinder med hål i sig. Cylindern roterar då pressen är igång. Framför cylindern är en ljuskälla som pekar på sensor S11. Ljuset från ljuskällan kommer endast att nå sensorn då ett hål i cylindern är framför sensorn (b) Eftersom cylindern roterar kommer S11 ha formen av en rektangelvåg. Då pressen trycker på bilen kommer pressen att möta mer och mer motstånd vilket kommer sakta ner den, därmed kommer värdet T att öka.

och startstegen i alla funktionsdiagram aktiveras omedelbart (och alla andra steg deaktiveras). Dessutom kan man i det grafiska gränssnittet se en röd cirkel vid den motor som gav upphov till stoppet. I detta läge kan varken det automatiska eller manuella programmet startas om man inte först nollställer PLC:n med menyvalet *Online/Reset*. I många fall är det också lämpligt att återgå till programmeringsläget för att åtgärda det som gav upphov till att motorskyddet behövde aktiveras.

Om nödstoppet trycks in under körning blir effekten densamma som då motorskyddet har aktiverats förutom att man inte behöver nollställa PLC:n för att kunna starta programmet igen.

3.5 Rotationsgivare

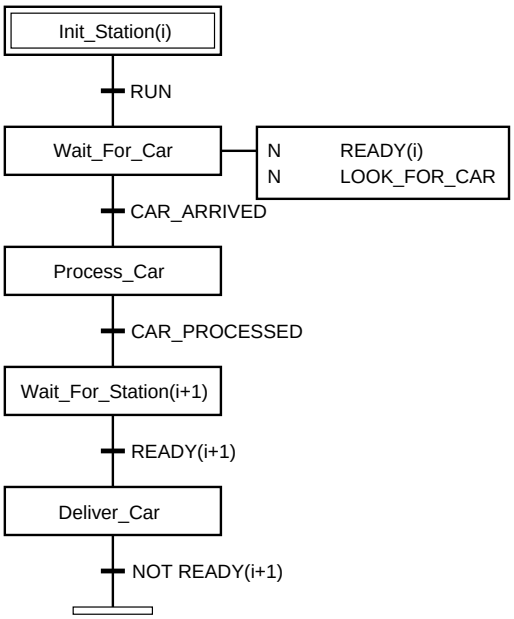
Både fabrik A och fabrik B har en rotationsgivare. I Figur 10 kan ni se hur den fungerar samt hur programmet räknar ut T. Ni behöver inte implementera detta själva. Pressen använder utöver detta två variabler, $TACHO$ och $TACHOTIME$. Deras beteende beskrivs av

$$TACHO = \begin{cases} 1, & \text{om } T \geq TACHOTIME \\ 0, & \text{annars.} \end{cases} \tag{1}$$

Variabeln $TACHO$ är den som ni kommer använda i ert program. I fabrik B kan den användas för att avgöra att pressen har pressat fast taket tillräckligt hårt och att det momentet nu är klart. I fabrik A används $TACHO$ endast för att stoppa pressen då den har stannat. För att pressen ska fungera bra i fabrik B är det nödvändigt att välja $TACHOTIME$ så att taket pressas fast ”lagom” hårt. Om $TACHOTIME$ är för stort riskerar pressen att inte sluta förrän den trycker väldigt hårt vilket kan leda till hårdvaruproblem, medans ett för lågt värde riskerar att taket inte trycks fast ordentligt. Ni kan hitta $TACHOTIME$ under ”Resources->Global_Variables”.

OBS.
Innan pressen startas kommer $TACHO$ att vara 1. Om endast $TACHO$ används som slutvillkor kommer pressen att omedelbart sluta. Ett sätt att hantera detta skulle vara att introducera en tidsbegränsad variabel ”START_PRESS_UP” och lägga till ”NOT_START_PRESS_UP” som slutvillkor. Det finns också flera andra sätt att hantera detta.

4 Ett sammanfattande exempel

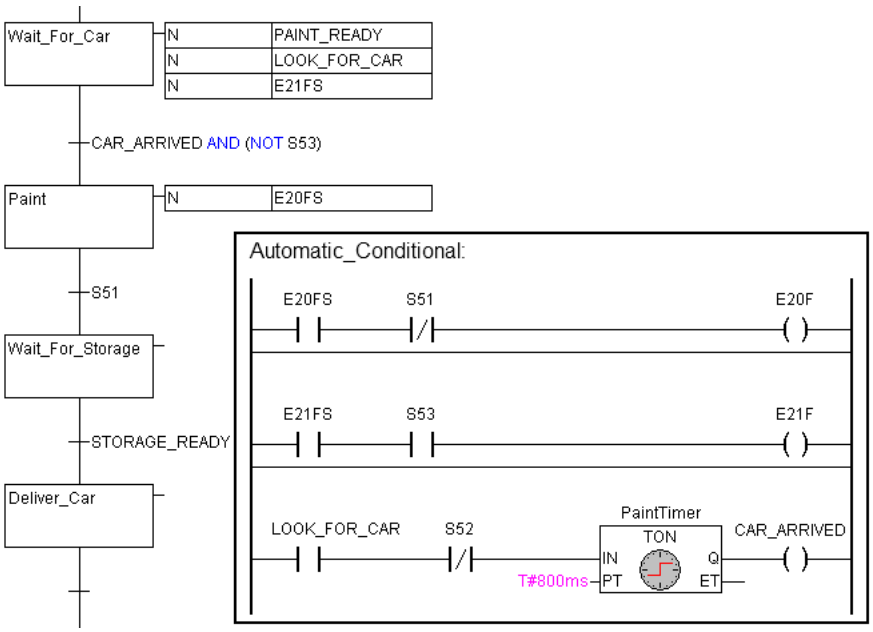


Figur 11: Generellt funktionsdiagram för delprogram. Notera hur station i väntar på att station $i+1$ ska bli redo innan bilen levereras. Varför väntar station i (efter leveransen) även på "NOT READY($i+1$)"? Fundera noga igenom vad som ska hända i de olika stegen. I vilket steg ska stationen återställas? När ska transportbandet köras?

För att sammanfatta avsnittet om utvecklingsverktyget CoDeSys och för att ge er lite inspiration när ni ska komma igång med programmeringen följer här ett exempel på hur en fiktiv station för målning av bilar skulle kunna implementeras. Alla detaljer i programmeringen är dock inte medtagna här. Tänk på att detta exempel följer strukturen som visas i Figur 11, vilket kan vara en bra bas för alla stationerna.

Antag att det finns en station där bilen ska målas innan den skickas vidare till ett lager. Vid målningsstationen finns det en spärr som kan köras fram med E21F. Att spärren är i sitt främre ändläge detekteras av S53=0. Vidare kan närvaron av en bil vid målningsstationen detekteras av fotosensorn S52=0. Målningen sker automatiskt genom att man ettställer E20F tills hela bilen är målad, vilket detekteras av S51=1. I Figur 12 visas några steg i ett funktionsdiagram som styr målningsstationen.

I steget Wait_For_Car förs spärren vid målningsstationen fram (E21FS). Motorn ska stoppas så snart spärrens ändlägesgivare S53 blir noll och logiken som ser till att så blir fallet är implementerad i ett reläschema i delprogrammet Automatic_Conditional. I steget Wait_For_Car skickas även en signal (PAINT_READY) till föregående station att målningsstationen är redo att ta emot en bil för bearbetning. Det förutsätts naturligtvis att transportbandet E0 körs under leveransfasen. Vi vill stanna kvar i steget Wait_For_Car ända tills vi säkert vet att en bil har kommit fram och tills spärrmotorn har nått sitt ändläge. Övergångsvillkoret efter steget blir därför uppfyllt först när *båda* dessa saker har inträffat. För att få en tillförlitlig närvaroindikering får signalen från fotosensorn gå genom en tillslagsfördröjd timer (TON). Härigenom filtreras eventuella störningar från fjädrarna på transportbandet bort. Den logik som ettställer flaggan CAR_ARRIVED för att indikera säker närvaro av en bil är implementerad i Automatic_Conditional. I detta delprogram används en instans av TON som vi har valt att kalla PaintTimer. "T#800 ms" betyder att fördröjningen (Preset Time, PT) ska vara 800 ms, dvs. S52



Figur 12: Ett exempel på hur den fiktiva målningsstationen kan programmeras.

måste vara 0 och variabeln LOOK_FOR_CAR måste vara 1 under minst 800 ms för att variabeln CAR_ARRIVED ska bli 1.

Då en bil säkert är framme vid målningsstationen går vi till steget Paint, i vilket målningsverktyget körs tills hela bilen är målad. Precis som med E21F i föregående steg styr man inte E20F direkt i funktionsdiagrammet utan bara via en hjälpvariabel E20FS. Med hjälp av reläschemat i Automatic_Conditional får man en säkrare ettställning av E20F som kräver både att E20FS är 1 och att S51 är 0. Även om man här i princip skulle kunna använda en standardhandling för ettställning av E20F så skulle det innebära en risk eftersom oförsiktig ändring av övergångsvillkoret efter Paint-steget skulle kunna leda till att motorn förstörs. Grundregeln är därför att alltid använda reläscheman med kontakter som stannar motorn när den når sitt ändläge.

Då målningen är klar väntar vi i steget Wait_For_Storage på att den efterföljande lagerstationen ska bli klar att ta emot en ny bil, vilket i så fall indikeras av flaggan STORAGE_READY.

5 Att komma igång...

Innan ni börjar programmera

Vid varje fabrikshalva finns en dator där ni kan logga in på era studentkonton. Gå därefter till `K:\TSRT07\labs\` och kopiera katalogen `lego` till studentkontot. Döp inte om katalogen eftersom det kommer att krävas att den finns med oförändrat namn när ni ska kompilera ert program. Starta CoDeSys och öppna (*File/Open*) filen `factory_a.pro` eller `factory_b.pro` i katalogen som ni nyss kopierade.

Ett första program

Vi ska här bekanta oss med utvecklingsverktyget CoDeSys genom att skriva in och testa det enkla program som beskrivs i förberedelseuppgift 3.

1. Öppna delprogrammet Automatic_Operation under POUs-fliken och komplettera funktionsdiagrammet så att det stämmer överens med er lösning till förberedelseuppgift 3.

2. Kompilera programmet med *Project/Build* (F11), rätta eventuella fel och kompilera på nytt tills alla fel är rättade.

3. Kopiera programmet till PLC:n med *Online/Login* (Alt+F8).

4. Starta programmet med *Online/Run* (F5).

5. Öppna det grafiska gränssnittet under Visualizations-fliken.

6. Testa nu programmet genom att byta till automatisk mod och klicka på startknappen. Titta på delprogrammet Automatic_Operation under POUs-fliken. Vilket steg är aktivt nu? Klicka även på stoppknappen och observera vad som händer.

7. Stoppa och nollställ PLC:n med *Online/Reset* och avbryt monitoreringen med *Online/Logout* (Ctrl+F8). Ni kan nu antingen korrigera eventuella fel och därefter repetera stegen ovan, eller gå vidare med nästa delprogram.

8. Ni kan hitta skeletten för SFC filerna i Sec. A.3 och Sec. B.3. Det är naturligtvis tillåtet att ändra på dessa om ni kan hitta alternativa lösningar.

Att tänka på

- Börja med att köra igenom fabriken manuellt genom att använda knapparna i det grafiska gränssnittet för att förstå hur den fungerar och var alla fotosensorer sitter.

- Under drift indikeras fotosensorernas värden både i det grafiska gränssnittet och av dioder på PLC:n. Styrsignalernas värden indikeras både av dioder på PLC:n och på reläerna som sitter på styrenheten. Dessa indikatorer kan vara mycket användbara vid felsökning.

- Att läsa namnen på de globala variablerna i tabell. A.1 och tabell B.1 can ge er an ledtråd i hur ni implementerar era funktionsdiagram.

- Tänk på att om möjligt hålla er till samma fabrikshalva (nr 1, 2 eller 3) genom hela laborationstiden. Detta kan nämligen underlätta programmeringen eftersom de olika fabrikerna kan upplevas som något olika ”individer”.

- Bilfabrikerna får inte byggas om på något sätt. Om utrustningen inte fungerar anmäler ni detta i webbformuläret på webbadressen ovan. Genomtänkta förbättringsförslag mottages tacksamt, men eventuella förändringar i hårdvaran får endast utföras av institutionens personal.

- Det är inte tillåtet att äta mat i Laboteket eller att ta med några drycker till laborationsplatserna. Det främsta motivet för denna regel är er egen säkerhet (styrenheterna kan kortslutas av utspillda vätskor) men också för att öka livslängden på utrustningen och att se till att laborationsplatserna är fräscha.

- Visa hänsyn till era medstudenter. Följ regeln som nämndes ovan om hur mycket tid man får boka. Tänk också på att det kan pågå annan undervisning i Laboteket samtidigt som ni är där för att arbeta med legofabriken och se till att laborationsplatsen är ren och städad när ni lämnar den.

6 Examination

6.1 Laborationens upplägg

Det praktiska upplägget för laborationen är följande:

1. På en *föreläsning* innan laborationen gås viss teori igenom och laborationens upplägg förklaras.
   Att förbereda: Läs igenom motsvarande kapitel i kurskompendiet.

2. Vid ett *introduktionstillfälle (2h)* i Laboteket (med obligatorisk närvaro) visar laborationsassistenten fabrikerna och förklarar laborationens utförande.
   Att förbereda: Läs igenom detta häfte noggrant och lös de tre förberedelseuppgifterna. Dessa är obligatoriska och lösningarna kommer att kontrolleras av laborationsassistenten vid introduktionstillfället.
   Uppgifter under tillfället: Bekanta er med fabriken, bland annat genom att provköra den manuellt från det grafiska gränssnittet. Implementera förberedelseuppgift 3, och fortsätt sedan på själva laborationsuppgiften om tid finns över.

3. Den stora delen av laborationen utgörs av *eget arbete*, varvid erforderlig laborationstid för programutveckling bokas i Laboteket. Det är tillåtet att boka *max 4 timmar* totalt per grupp åt gången, se bokningslistan i pärmen som hör till varje fabrik. Skriv noga i *LiU-ID* på listan. När dessa timmar har avverkats, får laborationsgruppen naturligtvis boka 4 nya timmar, o.s.v. Dessa regler gäller även helgtiderna, och vi har dem för att alla ska få en likvärdig chans att sitta vid fabrikerna.

4. Det kommer att finnas *resurstillfällen* i Laboteket kl 12.00 på måndagar och 12.30 tisdag-fredag under laborationstiden. En laborationsassistent finns då tillgänglig för frågor och diskussion om någon har anmält behov av detta via det frågeformulär som finns på följande websida:

   `http://www.control.isy.liu.se/student/laboteket/legolab/`

   Där finns även ett formulär som kan användas för anmälning av fel på utrustningen. Sådana fel kommer att åtgärdas så snart som möjligt, medan svar på frågor i första hand ges på nästa resurstillfälle. Webbsidan innehåller även en del aktuell information om laborationen.

5. *Examinationstillfället (1h eller 2h beroende på kurs)* i Laboteket (med obligatorisk närvaro) äger rum cirka en vecka efter introduktionstillfället. Laborationsassistenten kontrollerar då att kravspecifikationen i avsnitt 6.2 är uppfylld. Dessutom ska varje laborationsgrupp *implementera ett fel* hos en annan grupp, samt få ett fel implementerat i sin egen kod. Felet ska hittas och åtgärdas inom 30 minuter.

6.2 Kravspecifikation för examination

För att ni ska kunna examineras med godkänt resultat på laborationen krävs det att styrprogrammet uppfyller ett antal krav (samt närvaro på introduktions- och examinationstillfällena). Punkterna 1–9 nedan beskriver de krav som gäller i kursen TSRT07 Industriell reglerteknik. Motsvarande krav för övriga kurser finns beskrivna i en separat fil på kurshemsidan.

1. När nödstoppet trycks in ska samtliga motorer stanna omedelbart. Vidare ska alla funktionsdiagram få sina startsteg aktiverade och alla andra steg deaktiverade. (Denna funktionalitet finns redan implementerad i programskelettet som ni utgår ifrån, så det enda ni behöver göra är kontrollera att ni inte har gjort något som har satt denna funktionalitet ur funktion.)

2. Den del av bilfabriken ni valt (LEGO A eller LEGO B) ska kunna köras i normal drift och alla stationer ska kunna arbeta parallellt. Det innebär att det samtidigt ska kunna finnas en bil under bearbetning vid varje station (station = den del av en fabrikshalva som utför ett arbetsmoment).

3. Programmet ska hantera att det kan ta godtyckligt lång tid för en bil att anlända till nästkommande station.

4. Då tillverkningen startas ska motorernas lägen sakna betydelse, likaså om bottenplattorna/taken är slut eller lagret är fullt. Vid starten antas att inga delvis tillverkade bilar finns i fabriken.

5. Vid normalstopp, dvs. då man klickar på stoppknappen i det grafiska gränssnittet, ska varje station göra färdigt den bil den håller på med och sedan stanna. Även transportbandet ska stanna, ej nödvändigtvis omedelbart. Efter ett normalstopp ska bearbetningen återupptas automatiskt då man klickar på startknappen.

6. Felaktiga bilar, dvs. bilar som saknar någon chassibit i LEGO A och bilar som saknar tak i LEGO B, ska automatiskt sorteras bort.

7. LEGO A ska kunna hantera situationerna att bottenplattorna tar slut och att lagret med defekta bilar blir fullt. När något av detta händer ska motsvarande delprogram stanna i ett larmläge som lämnas först när problemet har åtgärdats och man har klickat på startknappen i det grafiska gränssnittet. Arbetet vid övriga stationer ska naturligtvis inte stanna bara för att bottenplattorna är slut eller lagret är fullt utan produktionen vid fabrikshalvans stationer ska fortsätta så långt det är möjligt.

8. LEGO B ska kunna hantera att biltaken tar slut och att något av billagren blir fullt. När något av detta händer ska motsvarande delprogram stanna i ett larmläge som lämnas först när problemet har åtgärdats och man har klickat på startknappen i det grafiska gränssnittet. Arbetet vid övriga stationer ska naturligtvis inte stanna bara för att taken är slut eller lagret är fullt utan produktionen vid fabrikshalvans stationer ska fortsätta så långt det är möjligt.

9. Er fabrikshalva (LEGO A eller LEGO B) ska vid examinationstillfället demonstreras tillsammans med den grupp som är bokad på den andra fabrikshalvan (LEGO B eller LEGO A). Fabrikshalvorna ska vara synkroniserade, vilket här innebär att LEGO A inte får leverera bilar till LEGO B om denna inte är redo att ta emot.

Vid examinationstillfällets början ska ni vara klara att demonstrera fabriken. Eventuella fel på utrustningen ska då vara inrapporterade. Examinationen består, förutom demonstrationen, i att ett (1) hemligt fel läggs in i ert styrprogram. För att bli godkända måste ni lokalisera och åtgärda felet inom 30 minuter. Sabotaget ombesörjs av en annan laborationsgrupp, varför ni ska vara inställda på att själva implementera ett fel i någon annan laborationsgrupps styrprogram. Att uppfinna ett sådant fel *är en del av examinationen*, varför följande krav ställs:

- Felet ska vara utslagsgivande i den bemärkelsen att då man kör fabriken ska det vara uppenbart att något är fel.
- Felet får inte vara trivialt, ej heller svårt att upptäcka för den som förstår styrprogrammet.
- Felet måste implementeras i mjukvara eftersom det inte är tillåtet att bygga om något i fabrikerna ens tillfälligt.
- Endast ett fel ska implementeras, dvs. endast en sak får ändras i programmet.

Inför examinationstillfället ska ni även ha reflekterat över följande frågeställningar.

- Om ni hade gjort laborationen igen, skulle ni då ha löst uppgifterna på något annat sätt?
- Har er arbetstid varit optimalt fördelad över laborationsveckan?

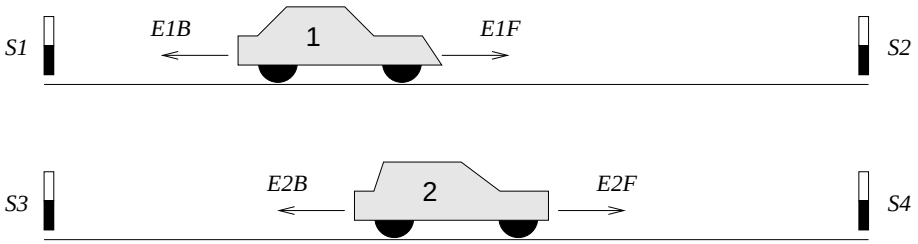
Omförhandling av kraven

Programmeringsuppgiften i denna laboration kan liknas vid ett litet projekt och i alla projekt finns det en risk för oförutsedda händelser. För att kunna hantera detta finns det därför en möjlighet för er att omförhandla kravspecifikationen ovan om det skulle behövas. En sådan omförhandling ska ske med en laborationsassistent på ett resurstillfälle *senast dagen innan ert examinationstillfälle* och ni måste noga kunna motivera varför det inte är möjligt att uppfylla något eller några av kraven. Bokför därför redan från början hur många timmar ni lägger ner på laborationen och tänk på att inte skjuta upp för mycket av arbetet till slutet av laborationsveckan. Det är inte möjligt att förhandla om kraven för att man har planerat sitt arbete dåligt och fått ont om tid på slutet.

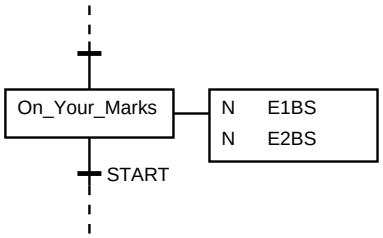
7 Förberedelseuppgifter

Uppgift 1: Återställning

Figur 13 visar två bilar som ska köra ikapp från ändlägesgivarna S1/S3 till S2/S4. Innan tävlingen kan börja måste bilarna dock backa till S1/S3, vilket ska göras i steget On_Your_Marks i funktionsdiagramsfragmentet i Figur 14. Rita reläschemat Automatic_Conditional som backar båda bilarna samtidigt när E1BS och E2BS är 1. Respektive motor (E1B/E2B) ska genast slås av då bilen når sitt ändläge. Då båda bilarna nått startpositionen ska flaggan READY bli 1. Formulera även övergångsvillkoret START så att det blir 1 då båda bilarna är färdiga att starta och domaren ger signalen GO.



Figur 13: Motorerna E1F/E2F kör bilarna framåt, E1B/E2B bakåt. Givarna S1,. . . ,S4 indikerar att respektive ändposition är nådd.



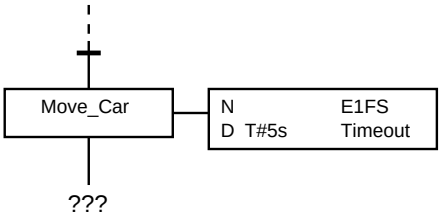
Figur 14: I On_Your_Marks ska bilarna i Figur 13 backa till startläget S1/S3.

Uppgift 2: Timeout

Antag att vi vill köra bil 1 i Figur 13 fram till S2, men att bilen eventuellt kan bli hindrad. Motorn E1F ska därför slås av om bilen inte nått fram inom 5 sekunder. En god början på ett funktionsdiagram som ska klara kraven ges i Figur 15. Det är er uppgift att göra fortsättningen. Lägg till en alternativförgrening som gör att exekveringen går vidare om bilen ohindrad når fram. Om bilen hindras ska motorn slås av och exekveringen stanna i ett larmsteg, i vilket flaggan ALARM ska vara 1. Larmsteget lämnas då signalen START blir 1, varefter exekveringen ska fortsätta som om inget hinder påträffats alls. (Denna typ av övervakning ger möjlighet att detektera fulla lager i LEGO A och LEGO B.)

Uppgift 3: Driftstyrning

I appendix A.3/B.3 (LEGO A/LEGO B) hittar ni skelettet till ett program för att tillverka legobilar. Koncentrera er först på Automatic_Operation, vars uppgift är att hantera drift och normalstopp. Se kravspecifikationen för en förklaring av vad som menas med ett normalstopp. Er uppgift är att färdigställa Automatic_Operation så att följande krav uppfylls:



Figur 15: Fortsättningsvis bör man ha en alternativförgrening som hanterar Timeout.

> När man trycker på startknappen i det grafiska gränssnittet så att START ettställs ett kort ögonblick ska den interna hjälpvariabeln RUN bli 1. RUN ska sedan förbli 1 ända tills stoppknappen i det grafiska gränssnittet trycks in och STOP ettställs ett kort ögonblick. Startsteget ska då åter bli aktivt.

Under laborationen kommer ni därefter komplettera övriga diagram så att det blir fullständigt specificerat med avseende på handlingar och övergångar. I programskeletten är övergångsvillkoret FALSE inlagt provisoriskt på flera ställen. Detta byter ni förstås ut mot era egna!

OBS.
Använd RUN, som indikerar att fabriken är igång, som startvillkor för det första blocket i varje station.

Lycka till!

A LEGO A

A.1 Variabler för LEGO A

Ingångar, LEGO A	Namn
Påskjutare bottenplattor (främre/bakre ändl.)	S0/S1
Magasin för bottenplattor tomt	S2
Närvaroindikering vid press	•S3
Påskjutare (bil) vid press (främre/bakre ändl.)	S4/•S5
Påskjutare (bitar) vid press (främre/bakre ändl.)	S6/•S7
Press (nedre läge)	•S8
Rotationsgivare press	S9
Spärr vid press (främre/bakre ändl.)	S10/•S11
Närvaroindikering vid diagnosstation	•S13
Indikering av saknad chassibit	S12, S14, S15, S16
Spärr vid diagnosstation (främre/bakre ändl.)	S17/•S18
Närvaroindikering vid lagerstation	•S19
Påskjutare vid lagerstation (främre/bakre ändl.)	•S20/•S21
Spärr vid lagerstation (främre/bakre ändl.)	S22/S23
Närvaroindikering vid hiss	•S24
Hiss (övre/nedre ändl.)	S25/•S26
LEGO B redo	S27

Med • indikeras aktivt låg ingång.

Utgångar, LEGO A	Namn
Transportband	E0*
Påskjutare för bottenplattor (fram/back)	E1F/E1B
Påskjutare (bitar) vid press (fram/back)	E2F/E2B
Press	E3
Påskjutare (bil) vid press (fram/back)	E4F/E4B
Spärr vid press (fram/back)	E5F/E5B
Spärr vid diagnosstation (fram/back)	E6F/E6B
Påskjutare vid lagerstation (fram/back)	E7F/E7B
Spärr vid lagerstation (fram/back)	E8F/E8B
Hiss (upp/ned)	E9U/E9D
Alarmlampa	ALARM**

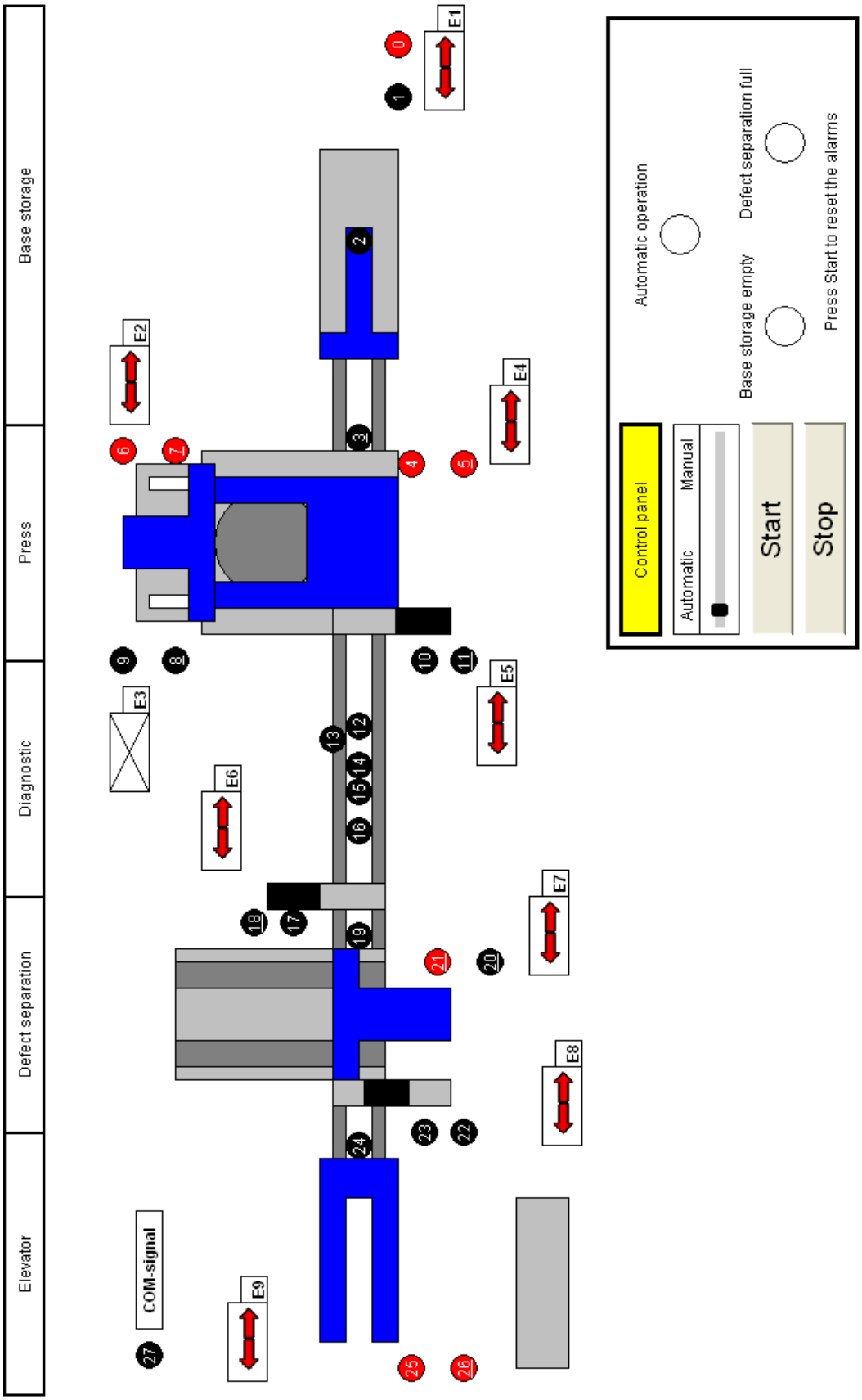
* E0 ettställs med E0A, E0B, E0C, E0D, E0E eller E0F.

** ALARM ettställs med BASE_ALARM eller DEFECT_ALARM.

START	LOOK_FOR_CAR_PRESS	E3S
STOP	CAR_DETECTED_PRESS	E9US
RUN	CAR_ARRIVED_PRESS	E9DS
PRESS_READY	LOOK_FOR_CAR_DIAGNOSTICS	BASE_ALARM
DIAGNOSTICS_READY	CAR_ARRIVED_DIAGNOSTICS	DEFECT_ALARM
DEFECT_READY	LOOK_FOR_CAR_DEFECT	
ELEVATOR_READY	CAR_DETECTED_DEFECT	
LOOK_FOR_PLATES	CAR_ARRIVED_DEFECT	
NO_BASE_PLATES	LOOK_FOR_CAR_ELEVATOR	
START_PRESS	CAR_DETECTED_ ELEVATOR	
LOOK_FOR_BRICKS	CAR_ARRIVED_ELEVATOR	
MISSING_BRICKS	E0A, E0B, E0C, E0D, E0E, E0F	
FAULTY_CAR	EkFS, EkBS, k = $1, 2, 4, 5, 6, 7, 8$	

Tabell 1: Globala variabler för LEGO fabrik A.

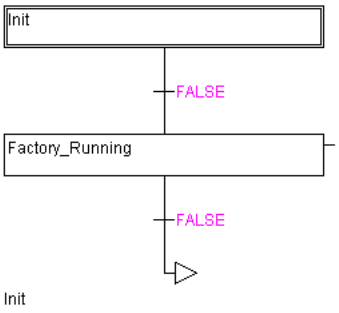
A.2 Grafiskt gränssnitt för LEGO A



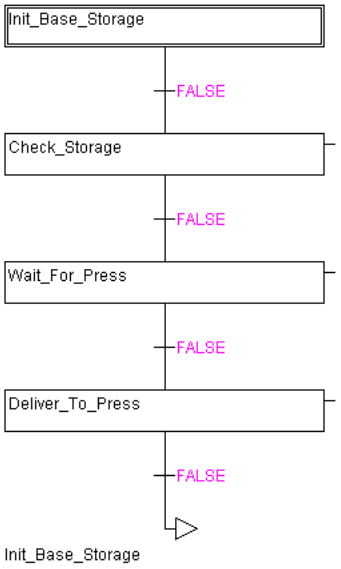
Figur 16: Interface of factory A.

A.3 Programmallar för LEGO A

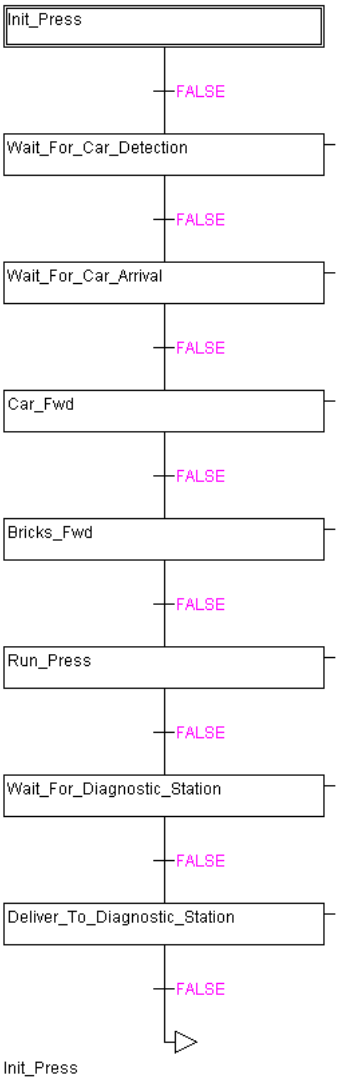
Automatic_Operation:



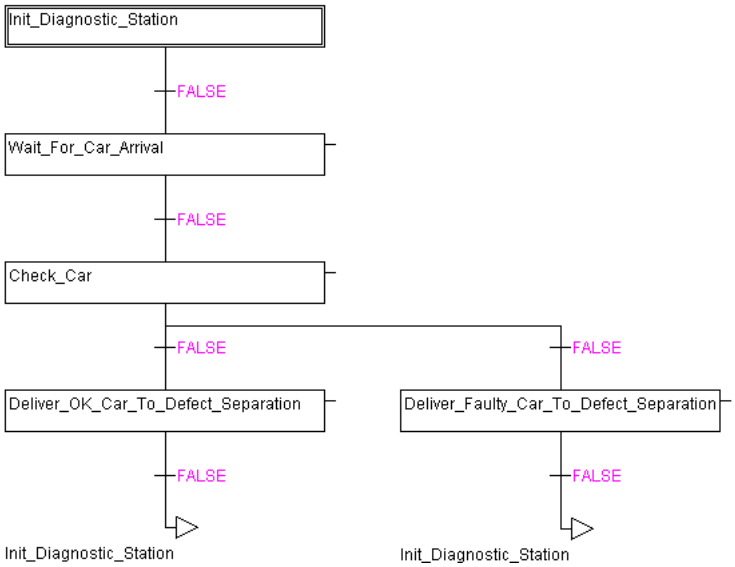
Base_Storage:



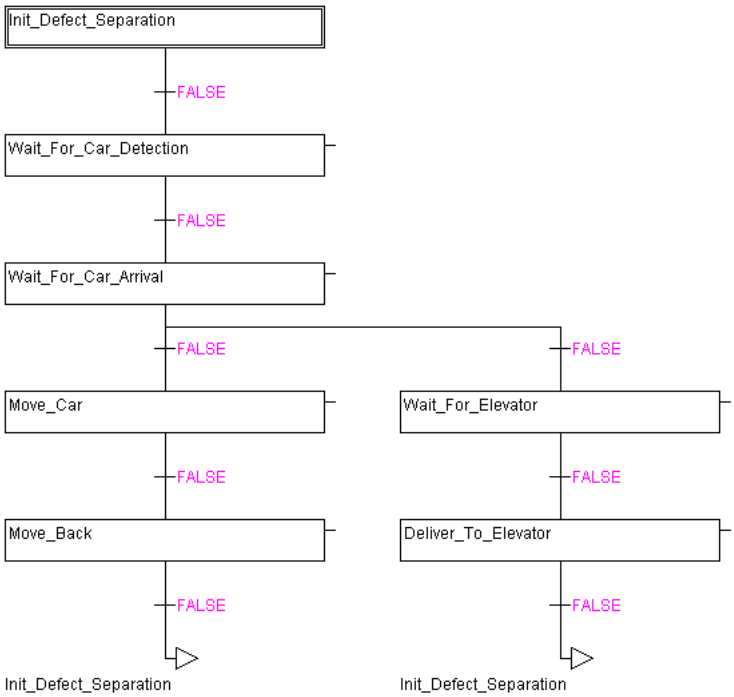
Press:



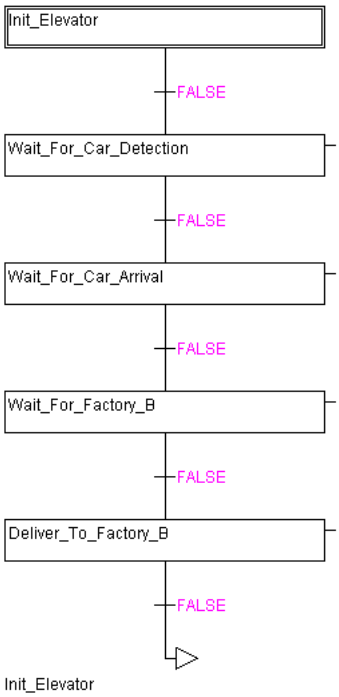
Diagnostic_Station:



Defect_Separation:



Elevator:



B LEGO B

B.1 Variabler för LEGO B

Ingångar, LEGO B	Namn
Närvaroindikering vid lobby	•S0
Spärr vid lobby (främre/bakre ändl.)	S2/•S1
Närvaroindikering vid takmagasin	•S3
Påskjutare för tak (främre/bakre ändl.)	•S5/S4
Magasin för tak tomt	S6
Spärr vid takmagasin (främre/bakre ändl.)	S7/S8
Närvaroindikering vid press	•S9
Press (övre läge)	S10
Rotationsgivare press	S11
Spärr vid press (främre/bakre ändl.)	S12/S13
Närvaroindikering vid lager 1	•S14
Påskjutare vid lager 1 (främre/bakre ändl.)	•S15/•S16
Spärr vid lager 1 (främre/bakre ändl.)	S17/S18
Närvaroindikering vid lager 2	•S19
Påskjutare vid lager 2 (främre/bakre ändl.)	•S20/•S21

Med • indikeras aktivt låg ingång.

START	LOOK_FOR_CAR_LOBBY	E0AS
STOP	CAR_ARRIVED_LOBBY	E4US
RUN	LOOK_FOR_CAR_ROOF	E4DS
ROOF_READY	CAR_ARRIVED_ROOF	ROOF_ALARM
PRESS_READY	LOOK_FOR_CAR_PRESS	STORAGE1_ALARM
STORAGE1_READY	CAR_ARRIVED_PRESS	STORAGE2_ALARM
STORAGE2_READY	LOOK_FOR_CAR_STORAGE1	
LOOK_FOR_ROOFS	CAR_DETECTED_STORAGE1	
NO_ROOFS	CAR_ARRIVED_STORAGE1	
START_PRESS_UP	LOOK_FOR_CAR_STORAGE2	
START_PRESS_DOWN	CAR_DETECTED_STORAGE2	
TACHO	CAR_ARRIVED_STORAGE2	
TACHOTIME	E0A, E0B, E0C, E0D, E0E, E0F	
FAULTY_CAR	EkFS, EkBS, k = $1, 2, 3, 5, 6, 7, 8$	

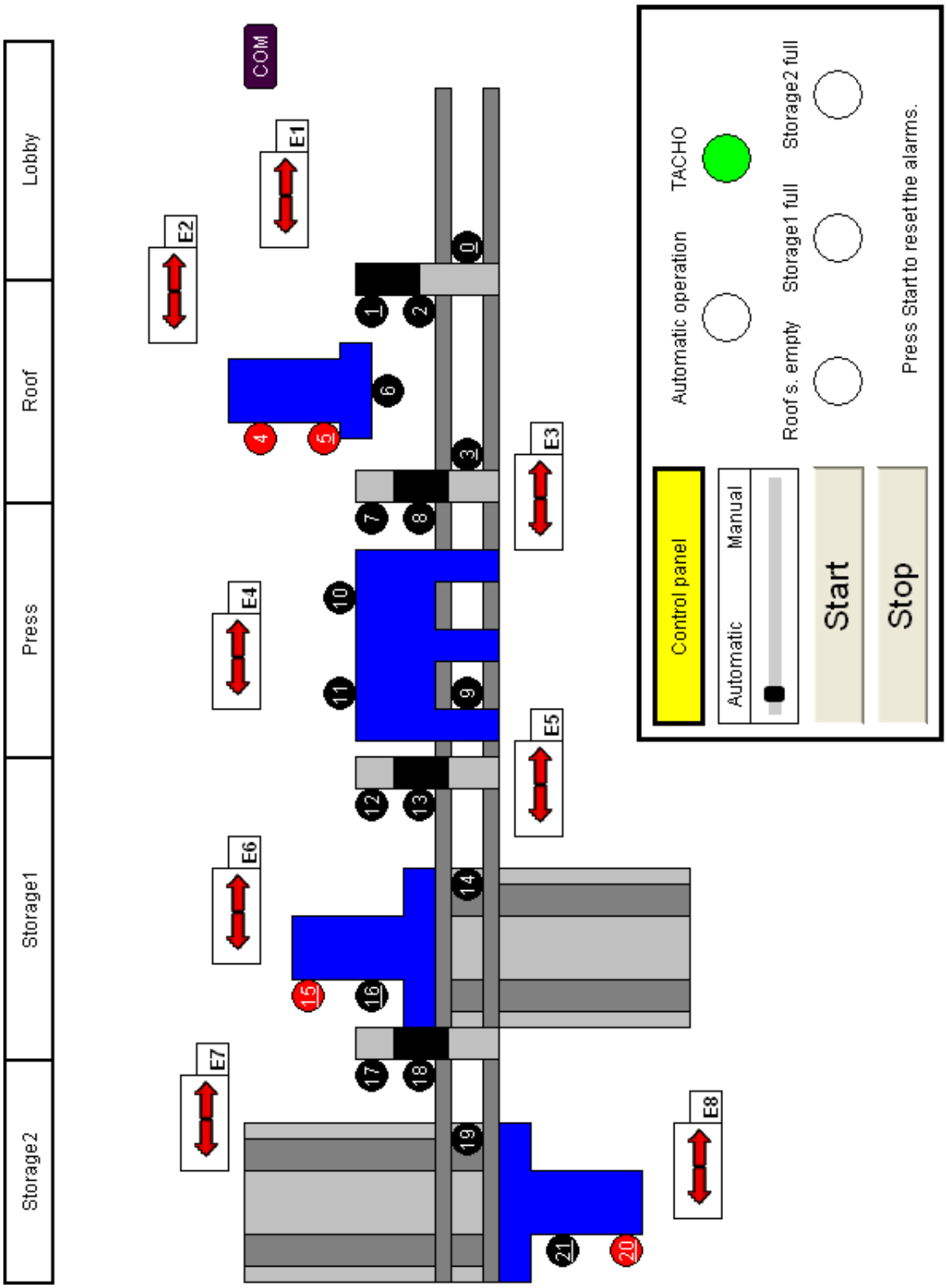
Tabell 2: Globala variabler för LEGO fabrik B.

Utgångar, LEGO B	Namn
Transportband	E0*
Spärr vid lobby (fram/back)	E1F/E1B
Påskjutare för tak (fram/back)	E2F/E2B
Spärr vid takmagasin (fram/back)	E3F/E3B
Press (upp/ned)	E4U/E4D
Spärr vid press (fram/back)	E5F/E5B
Påskjutare vid lager 1 (fram/back)	E6F/E6B
Spärr vid lager 1 (fram/back)	E7F/E7B
Påskjutare vid lager 2 (fram/back)	E8F/E8B
Signal till LEGO A	LOBBY_READY
Alarmlampa	ALARM**

* E0 ettställs med E0A, E0B, E0C, E0D, E0E eller E0F.

** ALARM ettställs med ROOF_ALARM, STORAGE1_ALARM eller STORAGE2_ALARM.

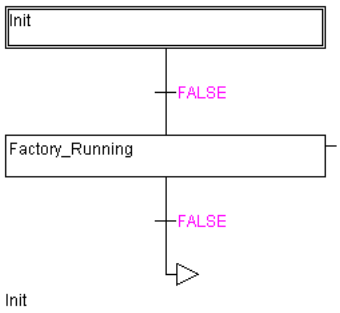
B.2 Grafiskt gränssnitt för LEGO B



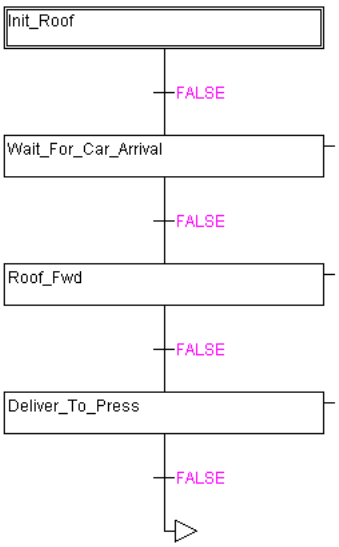
Figur 17: Interface of factory B.

B.3 Programmallar för LEGO B

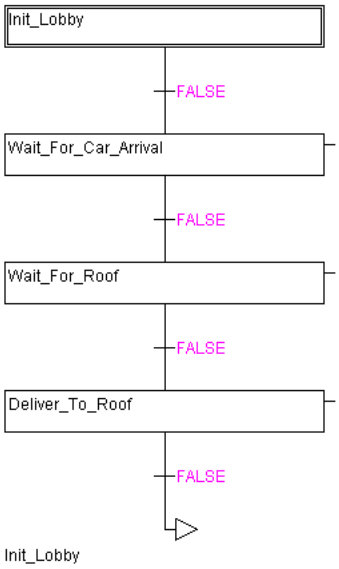
Automatic_Operation:



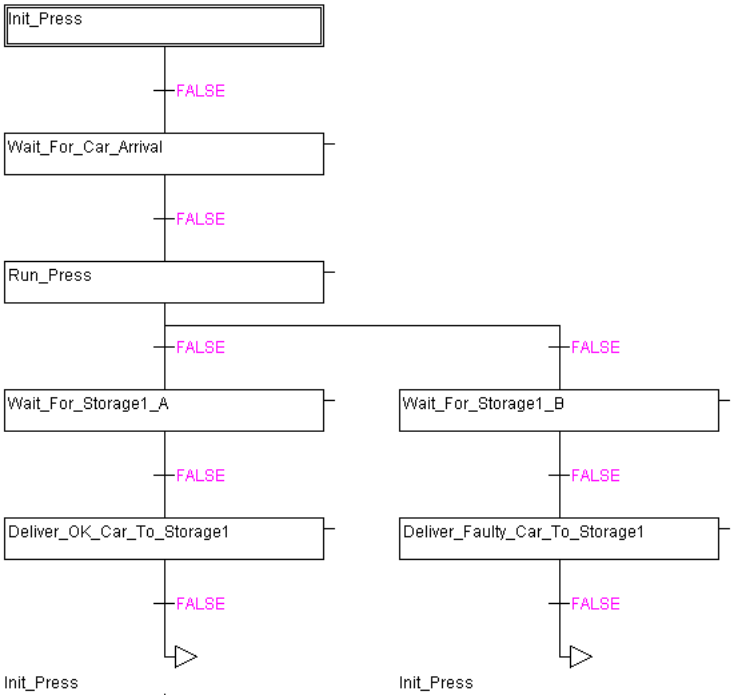
Roof:



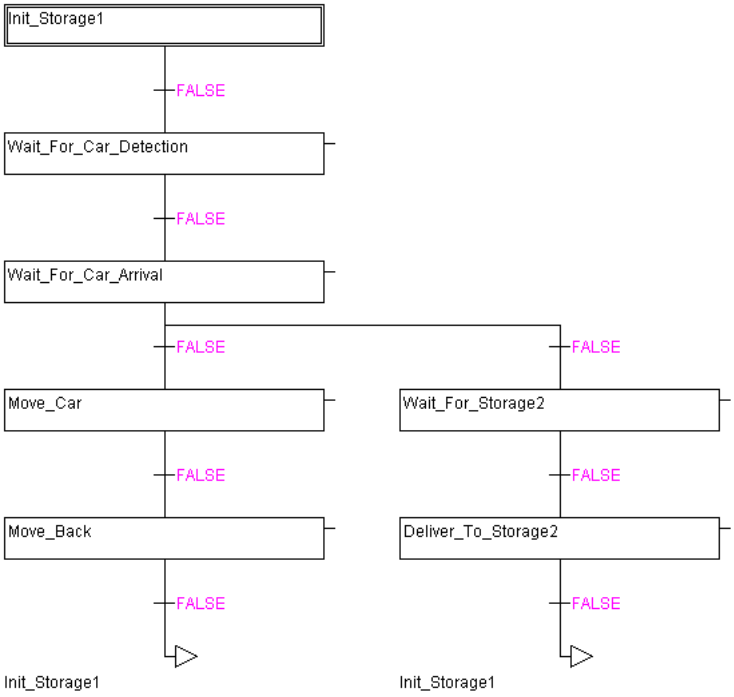
Lobby:



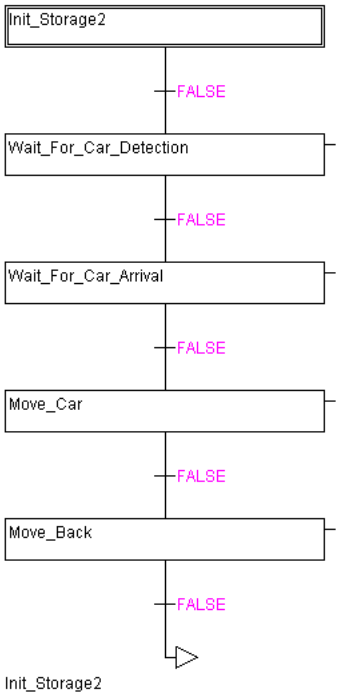
Press:



Storage1:



Storage2:


Init_Storage2
FALSE
Wait_For_Car_Detection
FALSE
Wait_For_Car_Arrival
FALSE
Move_Car
FALSE
Move_Back
FALSE
Init_Storage2