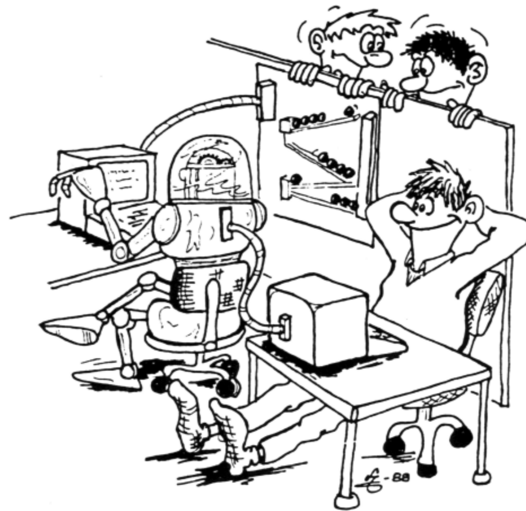
Industriautomation

Tentamen SSY 066, fredag 13/01–2017, fm

Lärare: Kristofer Bengtsson, 0768 979561



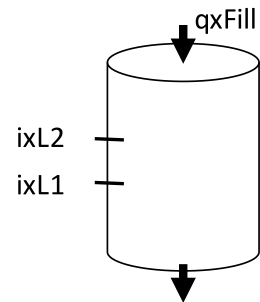
Fullständig lösning ska lämnas på samtliga uppgifter. I förekommande fall av tvetydigt formulerade tentamensuppgifter ska den föreslagna lösningen och eventuella antaganden motiveras. Examinator förbehåller sig rätten att godkänna rimligheten i antaganden och motiveringar.

Totalt omfattar tentamen 40 poäng. För betygen tre, fyra och fem krävs 16, 24 resp. 32 poäng. Lösningar anslås direkt efter tentatillfället på kursens hemsida i Pingpong. Granskning av rättningen sker måndag den 6 februari kl. 12:30 – 13:30 på institutionen.

OBS. Inga hjälpmedel är tillåtna.

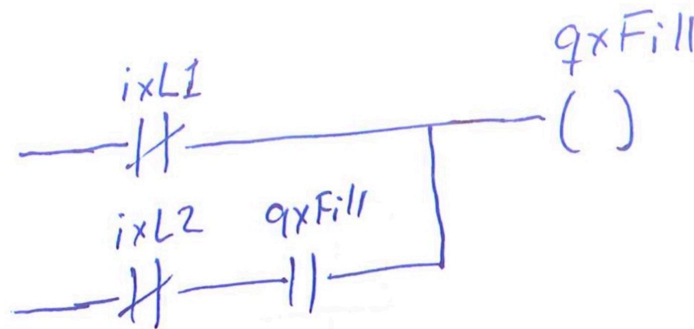
Uppgift 1

Målet i denna uppgift är att reglera en tank med vatten. Tanken fylls på med en slang längst upp och töms långsamt ut kontinuerligt ur ett litet hål i botten. Din uppgift är att hålla nivån på vattnet mellan två nivåer ixL1 och ixL2. Om vattnet är ovanför ixL2 skall du sluta fylla och när vattnet kommer under L1 skall du fylla på igen.



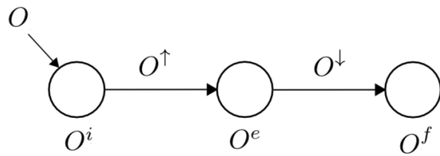
Implementera endast ett ladder-nätverk (ladder rung) för att styra utsignalen qxFill. Du får alltså inte använda set och reset coils.

(3 poäng)



Uppgift 2

Du har tre operationer som modelleras med tre tillstånd, - init, executing och finished. De tre operationerna kan gå mellan sina tillstånd enligt följande graf.



Rita upp tillstånden och övergångarna som beskriver hur dessa operationer kan exekvera i förhållande till varandra (grafen som bildas när dessa operationer körs) om de tre operationerna har följande guards (här används ==) och actions (här används :=):

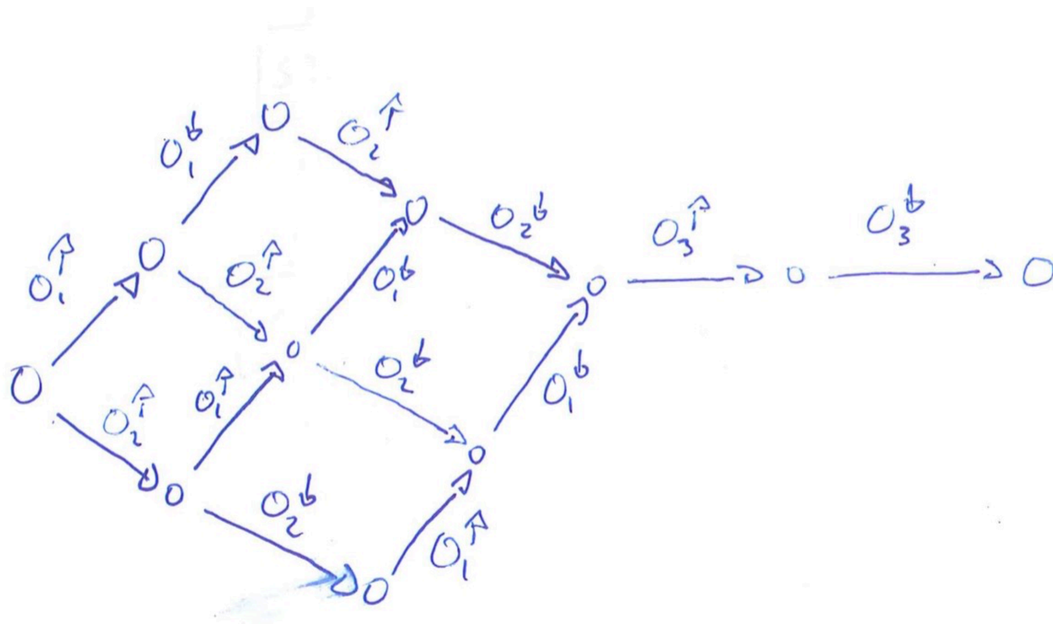
O1: precondition: v == 0, postcondition: v := 1

O2: precondition: k == 0, postcondition: k := 1

O3: precondition: v == 1 ∧ k == 1, postcondition: v := 2

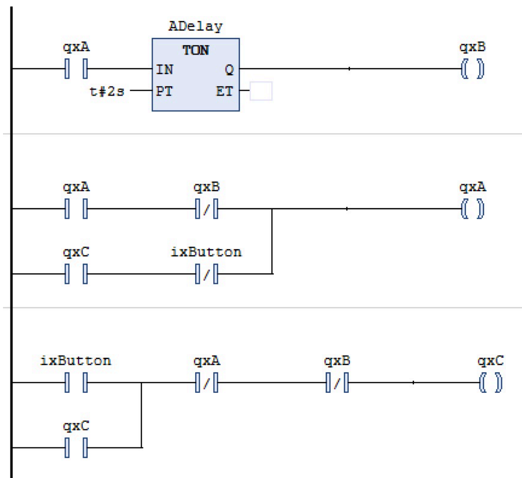
Initialtillstånd på variablerna är 0.

(3 poäng)



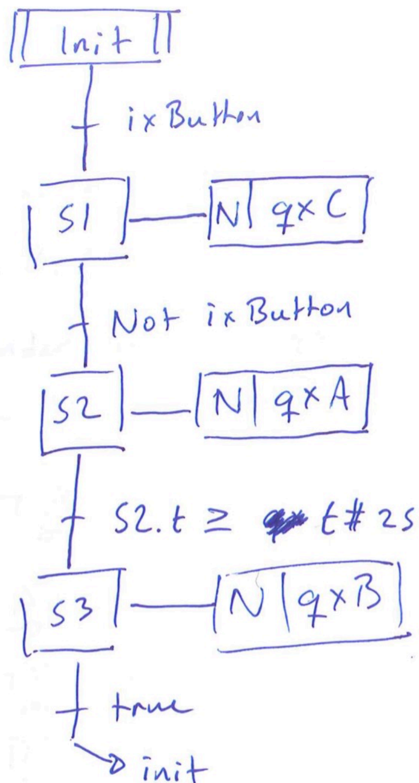
Uppgift 3

Följande ladderkod styr ett system med ingången ixButton och utgångarna qxA, qxB, och qxC enligt följande:



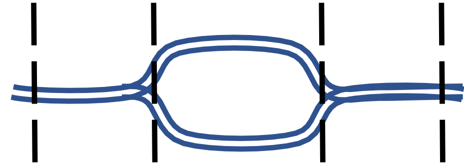
Implementera samma beteende med hjälp av SFC. Alla signaler börjar som false.

(4 poäng)



Uppgift 4

Autonoma fordon behöver kunna ta olika typer av beslut när de kör ute i trafiken. En hel del av dessa beslut kan beskrivas med hjälp av tillstånd och operationer. I denna uppgift skall vi titta på en ”korsning” som består av tre sektioner:



- I den vänstra sektionen kan det endast finnas en bil åt gången,
- I sektionen i mitten kan två bilar mötas (högertrafik gäller och det kan max vara två bilar i sektionen samtidigt)
- I den högra sektionen kan endast en bil befinna sig.
- Till höger och vänster om korsningen kan oändligt många bilar befinna sig

Målet med denna uppgift är att ta fram en styrning av fordonen som beslutar om de är möjligt att åka in eller ut ur sektionerna. Styrningen kan alltså stoppa bilar i en sektion tills nästa sektion är ledig. Fordonen kommer alltid köra rakt igenom, vilket betyder att om man åker in från vänster, åker man ut till höger och vise versa. De kan alltså inte backa eller vända.

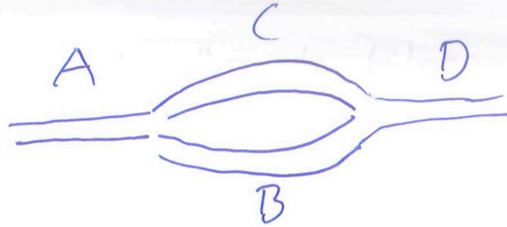
4a) Definiera lämpliga variabler och operationer för att beskriva en styrning av systemet där fordon åker igenom korsningen från båda hållen. En operation (tex att flytta ett fordon från en sektion till en annan) tar ingen tid och när den exekverat kan den direkt utföras igen om dess villkor är uppfyllda. Du skall göra egna antaganden och begränsningar av systemet, men motivera alltid dem. Din lösning måste tillåta att det finns flera fordon i korsningen samtidigt. Kom ihåg att styrning är diskret så operationerna uppdaterar tillståndet i diskreta steg. Ett tips: Det är vanligt att behöva skriva följande guard och action: ¬v / v := true, och v / v := false. Om du vill kan du använda följande notation: v^+: (¬v / v := true), och v^- : (v / v := false).

(5 poäng)

4b) Det finns ett problem med korsningen ovan som gör att det kan låsa sig så att fordon varken kan åka in eller ut ur korsningen. I vilken situation sker det och hur kan det undvikas? Uppdatera dina villkor på operationerna ovan så att det aldrig kan ske, men att du **ändå tillåter så många bilar i korsningen som möjligt**. Om din lösning i uppgift a redan hanterar detta problem, skriv ner vilka villkor som hanterar problemet

(3 poäng)

4a



Vi använder följande variabler:

$\underset{\rightarrow}{A}, \underset{\leftarrow}{A}, B, C, \underset{\rightarrow}{D}, \underset{\leftarrow}{D}$

Pilen under visar riktning på bilen. Alla är boolska. Vi har nu följande operationer:

InA: $\neg \underset{\leftarrow}{A} \wedge \underset{\rightarrow}{A}^{+}$ (vilket betyder $\neg \underset{\leftarrow}{A} \wedge \neg \underset{\rightarrow}{A} \;/\; \underset{\rightarrow}{A} := true$)

AB: $\underset{\rightarrow}{A}^{-} \wedge B^{+}$

BD: $\neg \underset{\leftarrow}{D} \wedge B^{-} \wedge \underset{\rightarrow}{D}^{+}$

DOut: $\underset{\rightarrow}{D}^{-}$

InD: $\neg \underset{\rightarrow}{D} \wedge \underset{\leftarrow}{D}^{+}$

DC: $\underset{\leftarrow}{D}^{-} \wedge C^{+}$

CA: $\neg \underset{\rightarrow}{A} \wedge C^{-} \wedge \underset{\leftarrow}{A}^{+}$

AOut: $\underset{\leftarrow}{A}^{-}$

4b Om det är 4 bilar i korsningen o båda bilen i A och i D är påväg in, kan inga bilar köra mer. Detta löses med extra guards på InA och InD.

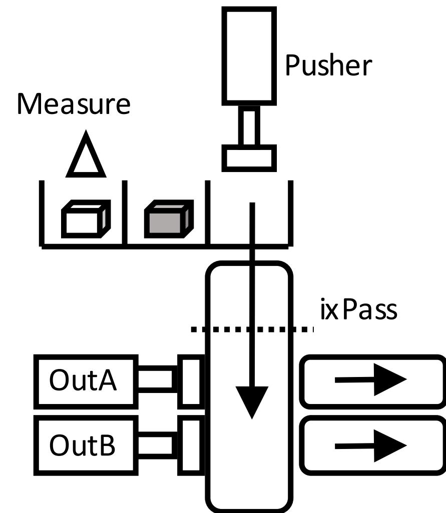
$$\text{InA}: \neg \underset{\leftarrow}{A} \wedge \underset{\rightarrow}{A^{+}} \wedge \underline{\neg (C \wedge B \wedge \underset{\leftarrow}{D})}$$

$$\text{InD}: \neg \underset{\rightarrow}{D} \wedge \underset{\leftarrow}{D^{+}} \wedge \underline{\neg (C \wedge B \wedge \underset{\rightarrow}{A})}$$

Observera att det är ok med 4 bilar i korsningen om någon i A eller D är påväg ut.

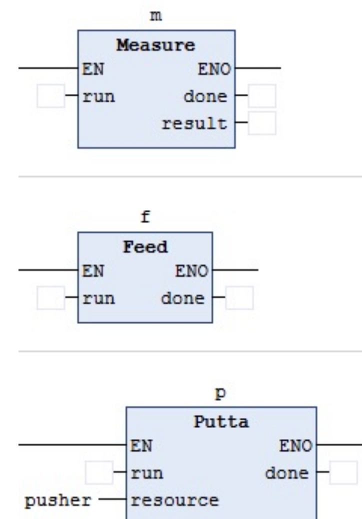
Uppgift 5

I denna uppgift skall du styra ett litet system som sorterar kuber. Det består av två delar, ett transportband där kuberna mäts och tre kolvar som puttar kuberna med hög fart in i rätt utbana.



Mätsystemet, Measure, mäter om det finns en kub på plats och i så fall om det är en A-kub eller en B-kub. När mätningen är gjord flyttas kuben åt höger, ett steg i taget, tills den befinner sig framför utputtaren, Pusher. Pusher puttar till kuben så den åker ut i banan med hög fart, där antingen OutA eller outB puttar ut kuben i rätt utbana beroende på dess typ.

En snäll programmera har redan implementerat några funktionsblock som du skall använda i din kod. För att mäta kuber finns **funktionsblocket Measure**, som mäter kuben när dess insignal **run** blir sann. När den mätt färdigt, blir utsignalen **done** sann, och resultatet, **result,** visar resultat där 0 = ingen kub, 1 = en kub av typ A, 2 = en kub av typ B. När insignalen run sätts false, blir också done false och det går att mäta igen.



Funktionsblocket Feed flyttar kuberna ett steg åt höger vid **run** och sätter **done** när flytten är klar. done blir false när run blir false.

Pusher, OutA och OutB styrs av var sin instans av **funktionsblocket Putta**. Detta funktionsblock har som insignal **run** och **resource,** där run triggar puttningen och där resource säger vilken av de tre puttarna det är. Putta-blocket har endast en utsignal, **done**, som blir sann när puttaren har puttat klart. Utsignalen blir false när run blir false så att det går att putta igen. Du behöver inte instansiera dessa block då de redan finns deklarerade. De globala variablerna **p.run** (pusher), **OA.run** (OutA) och **OB.run** (OutB) och **p.done**, **OA.done** och **OB.done** är kopplade till in och utgångarna på de tre blocken.

Det finns också en insignal som du skall använda, **ixPass**, vilket blir sann när en kub passerar. Den används för att veta tiden tills antingen OutA eller OutB skall putta. Vi återkommer till den senare. Uppgiften består av 4 delar, skriva koden för mätningen och att mata fram kuber, skriva koden för att putta ut kuber, skriva kod som använder dina två funktionsblock för att mata ut ett antal kuber och slutligen kod som kan styra din kod från två externa system.

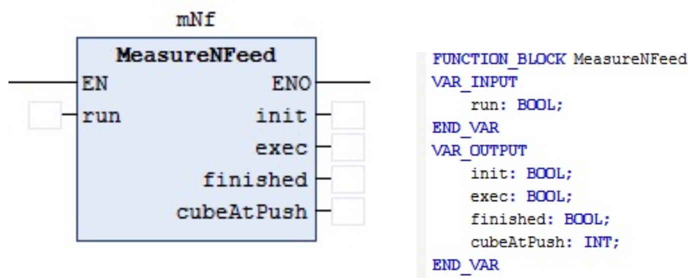
forts på nästa sida

5a)

Nu skall du implementera funktionsblocket MeasureNFeed. Denna operation skall vid signalen **run**, först mäta om det finns en kub på mätpositionen och i så fall vilken sort, och därefter flytta transportbandet ett steg åt höger. Kuben som just mättes (om det var någon) åker till mitten-positionen och den som var på mittenpositionen åker till ut-positionen. (Eventuellt kommer det också in en ny kub till mätpositionen). När detta är klart skall operationen uppdatera sin utsignal cubeAtPush med om det är en kub vid ut-positionen och i så fall vilken. (där 0 = ingen kub, 1 = en kub av typ A, 2 = en kub av typ B).

Operationens tillstånd, init, exec och finished skall också uppdateras på rätt sätt. Du måste se till att operationen minst är en scancykel i varje operationstillstånd.

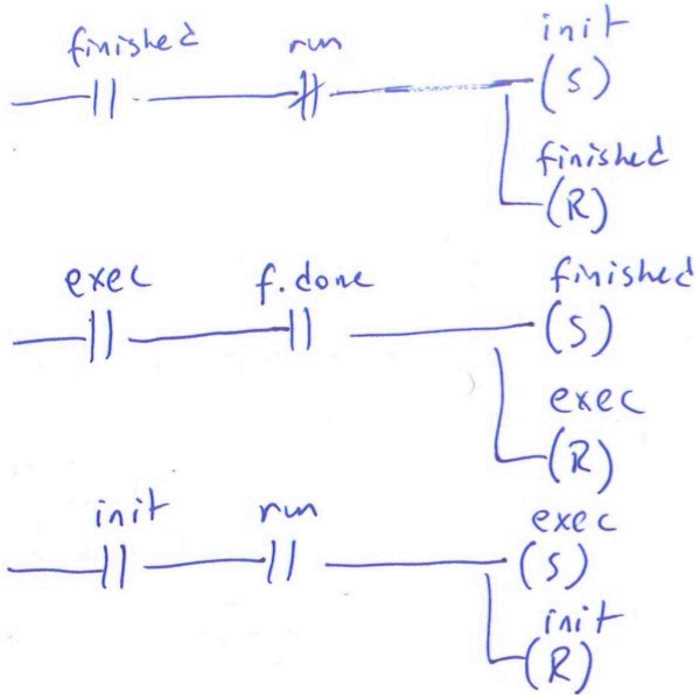
Du skall implementera logiken i ladder logic.

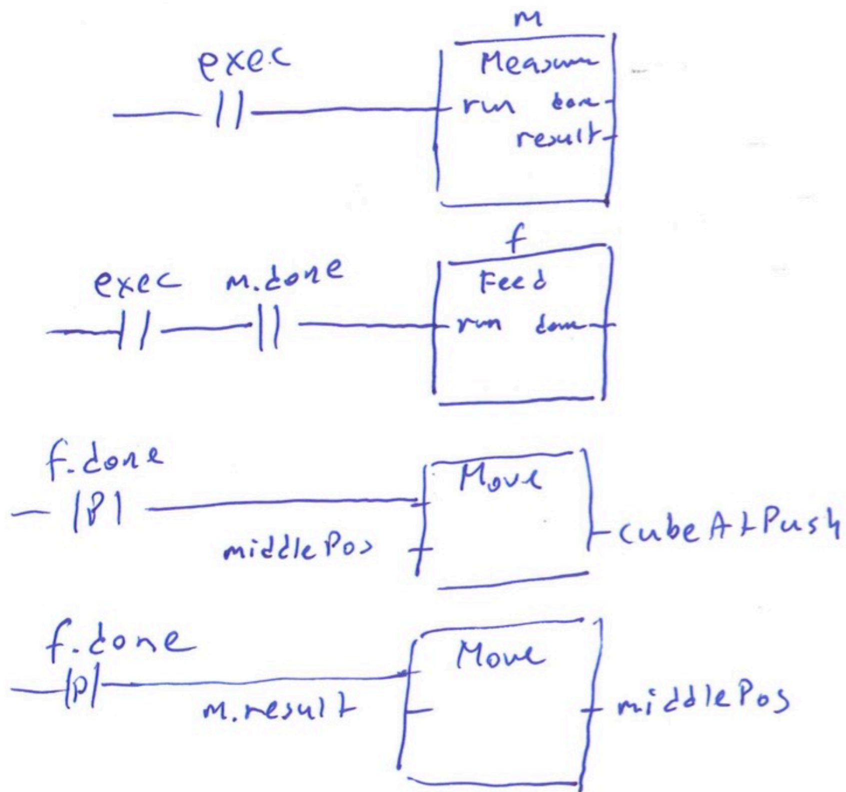


(6 poäng)

forts på nästa sida

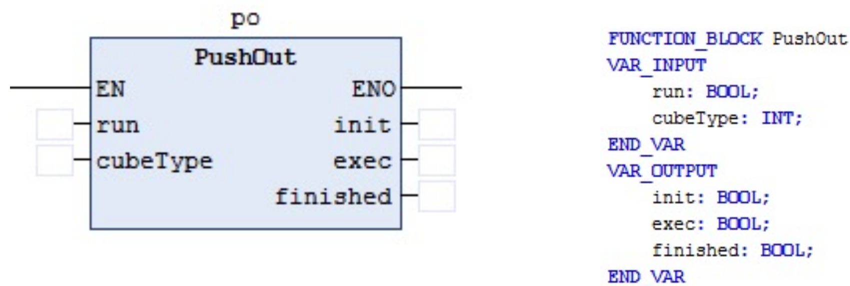
5a VAR middlePos : Int





5b)

Nästa steg är att bygga operationen som puttar ut kuberna på rätt utbana. Den heter PushOut och skall implementeras med en SFC. När run signalen blir sann och om cubeType inte är 0 (då finns det ingen kub) skall koden trycka till kuben så att den åker ut på banan. När kuben passerar ixPass, tar det 0.2 sek tills kuben är framme vid OutA och 0.6 sek innan kuben är framme vid OutB. Om kuben är A (1 som input), skall OutA putta ut den och om den är B (2 som input) skall OutB putta ut den. Använd de globala variablerna p.run (pusher), OA.run (OutA) och OB.run (OutB) och p.done, OA.done och OB.done i SFCn.



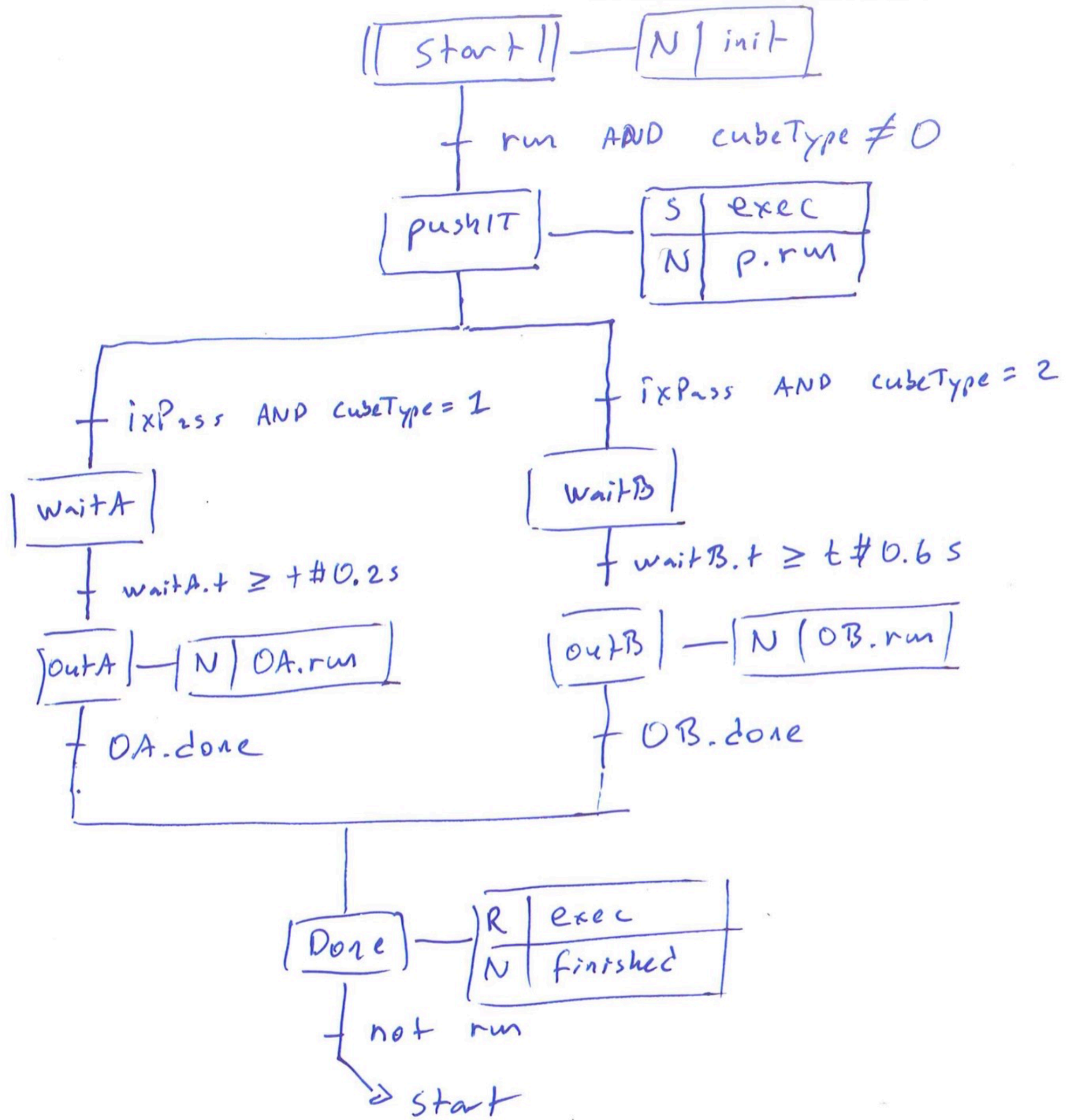
Din SFC skall givetvis uppdatera init, exec och finished på rätt sätt och gå tillbaka till init när run blir false. Hur din kod hanterar tillstånden om cubeType = 0 är upp till dig (din lösning kommer att påverka nästa steg)

Du skall implementera logiken i SFC.

(5 poäng)

forts på nästa sida

5b

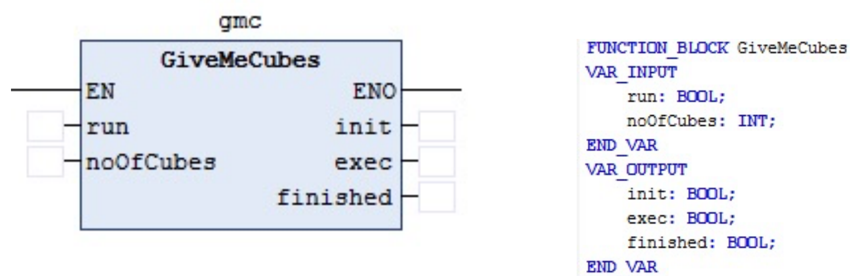


5c)

Nu skall vi implementera funktionsblocket GiveMeCubes. Den använder tidigare funktionsblock (MeasureNFeed och PushOut) för att mata ut kuber ur systemet. Den som använder GiveMeCubes vill tex få ut 10 kuber och sätter då noOfCubes till 10 och sätter run till sann. Din kod skall då mata ut 10 kuber och sedan sätta finished till sann.

Observera att du inte vet om det finns en kub vi mätpositionen, så det räcker inte med att bara köra dina block 10 gånger. Det som räknas är antal A eller B som kommer ut. Din kod skall hålla på tills det är färdigt.

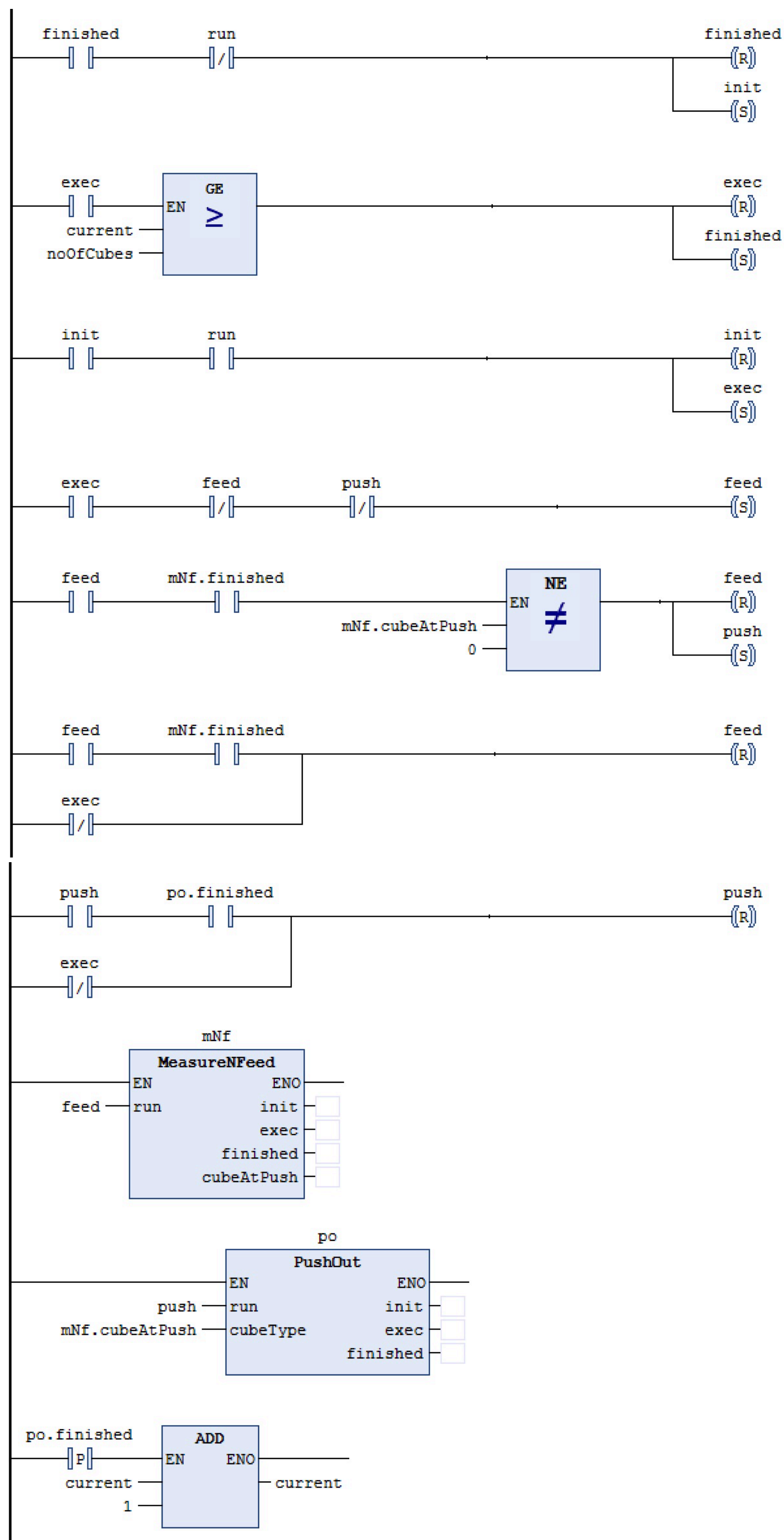
Du skall implementera logiken i ladder logic.



(6 poäng)

forts på nästa sida

En möjlig lösning 5c, current: INT, feed, push: BOOL



5d)

Nu är vi framme vid sista steget. Ditt senaste block GiveMeCubes, behöver kunna styras av två olika externa system. Dessa system uppdaterar och läser av olika variabler för att interagera med ditt block.

System 1 sätter sys1Run, sys1NoOfCubes och läser av sys1Finished
System 2 sätter sys2Run, sys2NoOfCubes och läser av sys2Finished

Vi behöver med andra ord kod som agerar ”request handler” der ovan signaler kopplas ihop med ditt block. Den som är först att sätta run får köra först, men kan inte avbrytas av den andra. Din kod får inte heller uppdatera fel finished signal, utan skall bara interagera med det system som för tillfället styr.

Ett annat scenario som din kod skall klara är om tex system 1 glömmer att plocka ner sys1Run efter att GiveMeCubes är färdig, då skall system 2 ändå kunna starta. Dock skall sys1Finished vara satt tills sys1Run plockas ner och system 1 kan inte starta blocket igen förrän detta sker.

Du kan välja om du skall implementera i ladder eller SFC. Du behöver inte instansiera GiveMeCubes i din kod, utan kan direkt sätta insignalerna gmc.run, gmc.noOfCubes och läsa av gmc.finished.

(5 poäng)

En möjlig lösning 5d. Var: sys1, sys2, exec: BOOL. cubes: INT