LET085 Styrteknik - föreläsning 3

Förra gången: Olika användbara funktionsblock och tillgängliga variabler. Variabler deklareras globalt eller lokalt i POU:ns header.

Exempel:- räknare 1

Efter 13 pulser från PULS-givare ska LAMPA tändas. LAMPA släcks och en räknare nollställs med insignal NOLLST.

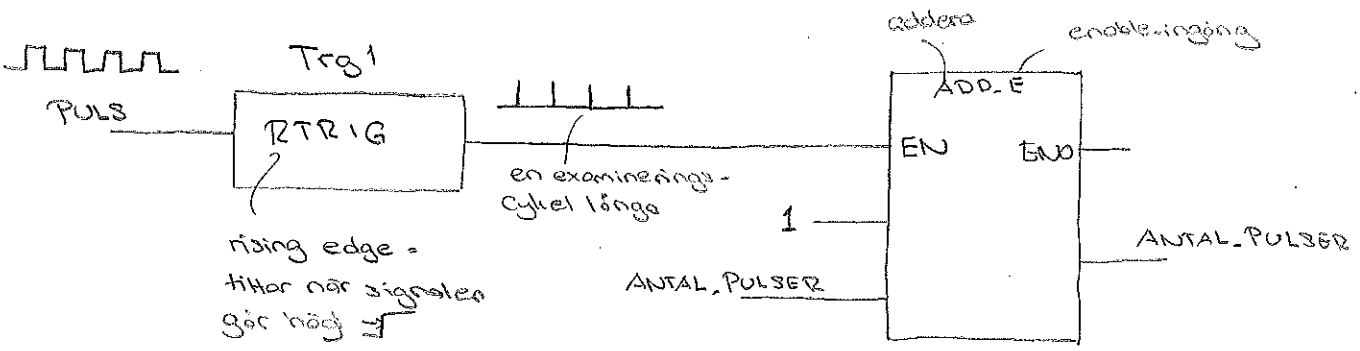
Global variable list

Identifier	Address	Type	
PULS	X0	BOOL	
NOLLST	X1	BOOL	
LAMPA	Y1A	BOOL	
ANTAL_PULSER	D0	INT	- kan sättas i Header istället om man vill.

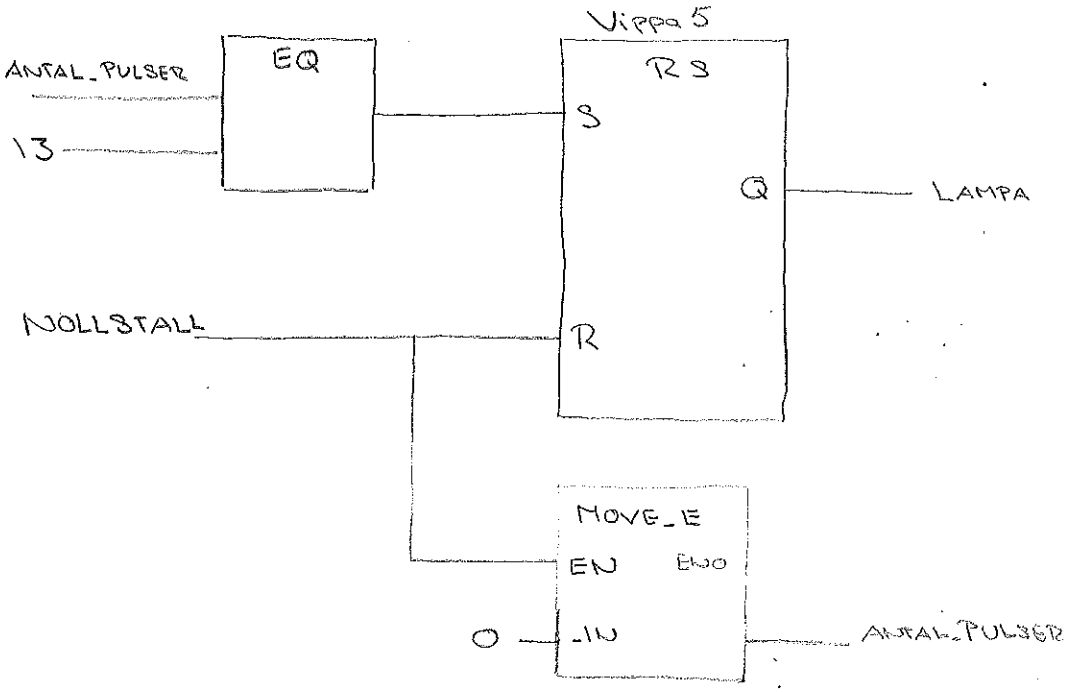
max 16 tecken i variabelnamnen.

7 16-bitars register

POU-body = Delprogram



(Vill vi istället räkna nedgångar på pulserna använd F_TRIG - falling edge)

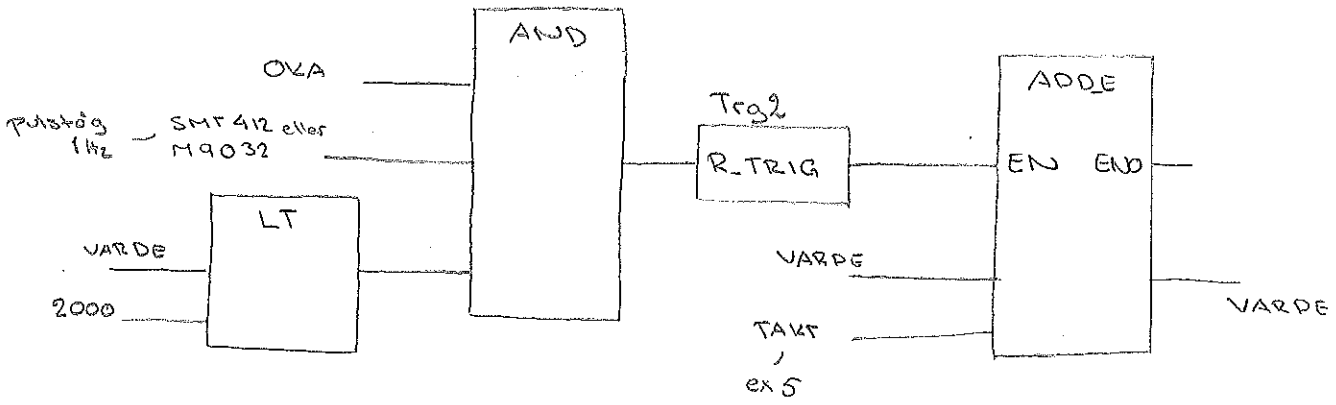


POU-Header - Lokala variabler i detta POU.

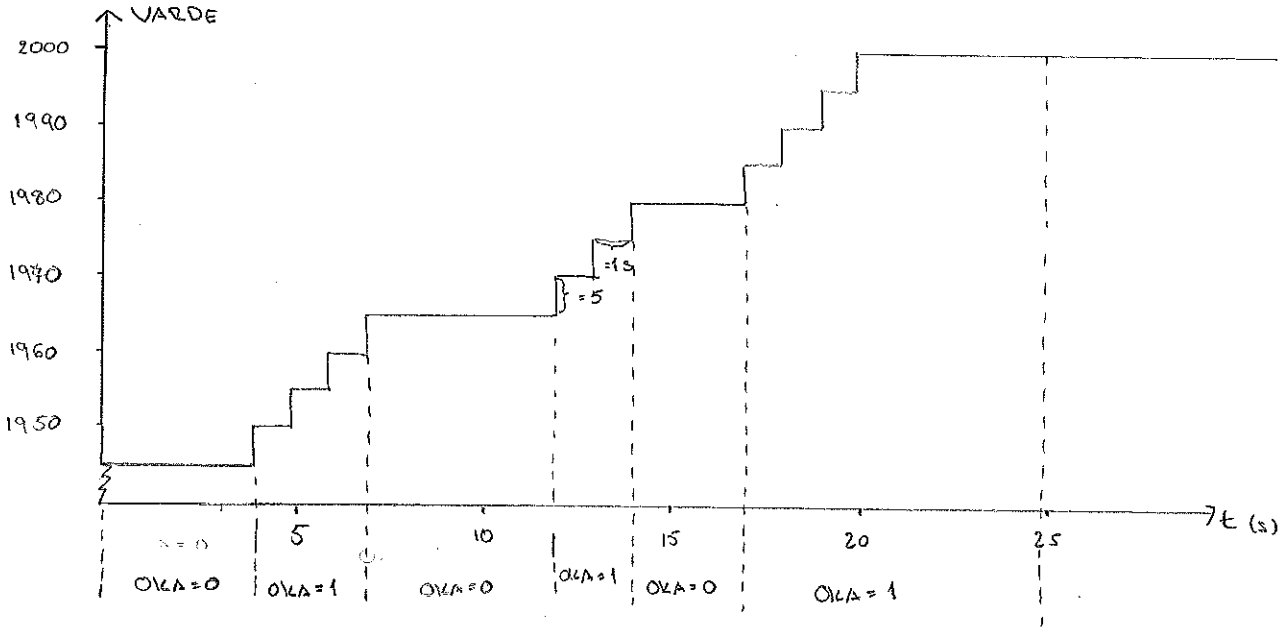
Identifier	Type	
Trg1	R_TRIG	
Vippa5	RS	
ANTAL_PULSER	INT	- Om den inte redan är deklarerad global.

Exempel räknare 2.

Rampning uppåt av variabel VARDE (INT) när en flagga OKA aktiveras.



Antag TAKT = 5 (skrivs in i programmet)



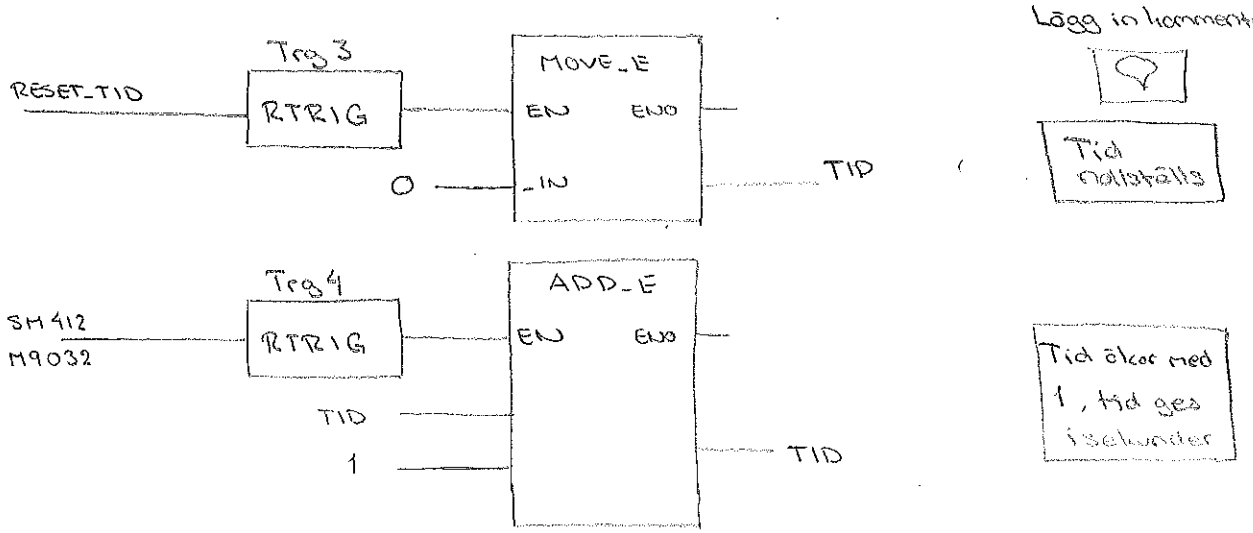
VARDE ökar nu med 5 enheter/s upp till max 2004

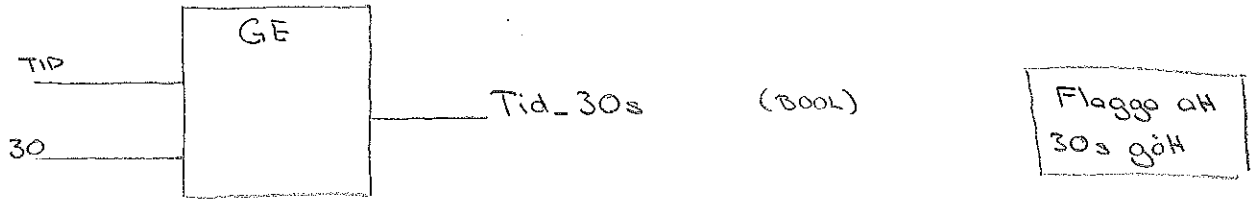
Kan på motsvarande sätt använda SUB_E

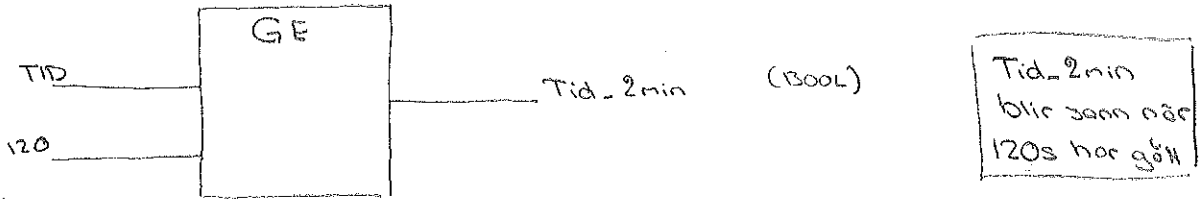
Exempel - räknare 3

En kortlöpande klocka, TID (INT) räknas upp och kan lösas av.

Klockan nollställs med RESET_TID (BOOL)



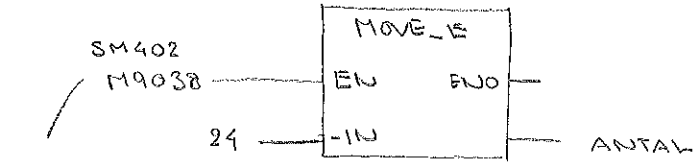




Vid behov av initiering till annat än 0 (eller FALSE) (5)
använd inte kolumnen "Initial" i variabellisterna.
Det fungerar inte alltid.

Identifier	Address	Type	Initial
ANTAL	D214	INT	~~24~~ - använd inte!

Gör istället:

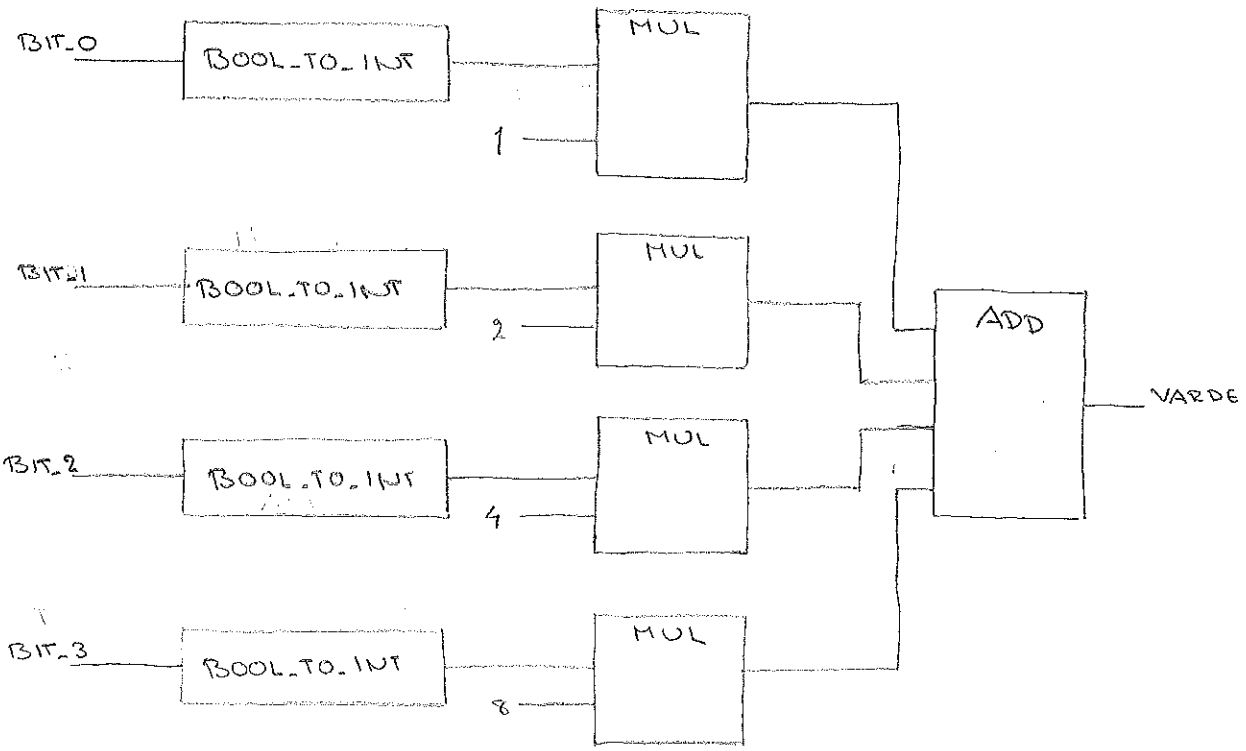


är TRUE
första exekveringsvarvet
därefter FALSE.

Exempel - talomvandling.

En 4-bitars tal läses in från 4st digitala ingångar.
Minst signifikanta bit betecknas här BIT_0 och
mest signifikanta bit är BIT_3, däremellan finns
BIT_1 och BIT_2 samtliga är BOOL.
Vi vill översätta det binära talet till INT-variabeln TAL.

Här kan vi utnyttja typomvandlingen BOOL_TO_INT som
omvandlar en BOOL som är TRUE till en INT som är 1
och en BOOL som är FALSE till en INT som är 0.
(Det finns även INT_TO_DINT och WORD_TO_INT exempelvis)



FIFO-register, First In First Out

Består först maximalt antal tillåtna element i kön.

I detta exempel används 8.

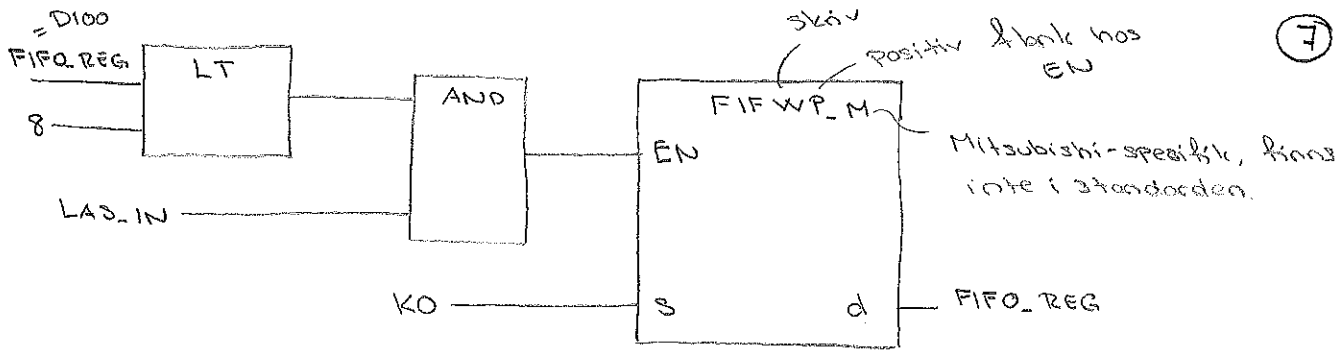
Nya beställningar läses in i variabeln KO (INT).

I detta exempel är först KO = 4 och därefter KO = 7.

För att inte ett stort antal 4 och sedan 7 ska läggas i kön måste vi också ha en BOOL-variabel LAS_IN som säger när det är tillåtet att lägga till ett nytt värde i kön

Består också en minnesadress där kölagringen börjar

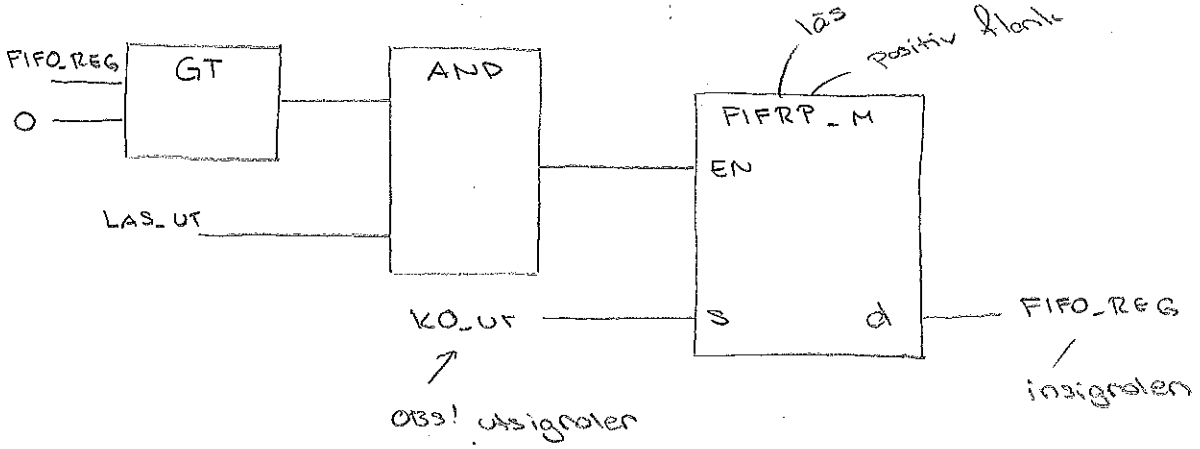
Här används adressen D100 som givits namnet FIFO_REG i den globala variabellistan. (INT)



Om KO = 4 och sedan KO = 7 kommer FIFO_REG att se ut som följer varje gång LAS_IN går från FALSE till TRUE (positiv flank →⎍)

		Första LAS_IN	andra LAS_IN	
D100	0	1	2	← antal i kön
D101	0	7	7	
D102	0	0	4	
D103	0	0	0	
D104	0	0	0	
D105	0	0	0	
D106	0	0	0	
D107	0	0	0	osv.
D108	0	0	0	

Nu kan vi börja läsa ut ur kön. Så länge vi har minst ett element kvar i kön kan vi läsa från den. Läsning ur kön görs vid positiv flank på LAS_UT (BOOL) och den aviösta variabeln hamnar i KO_UT (INT)



För varje gång LAS_UT aktiveras kommer då kön att förändras:

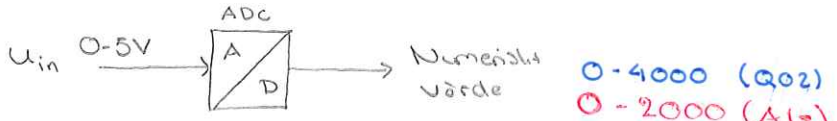
		LAS_UT skickas till
D100	2	1
D101	7	4
D102	4	0
D103	0	0
D104	0	0
D105	0	0
D106	0	0
D107	0	0
D108	0	0

till KO_UT

Analoga signaler i PLC

Förutom digitala ingångar (X0 - XF) och utgångar (Y10 - Y1F) som är 0 eller 24V kan vi använda analoga in- och utgångar. Intervallen för de analoga in- och utgångarna är ställbart. I våra labb har vi ställt in 0 - 5V på utgången motsvarande 0-4000 i Q02 och 0-2000 i AIS (heltal)

Vi får då i A/D - omvandlaren för analoga insignaler: ADC

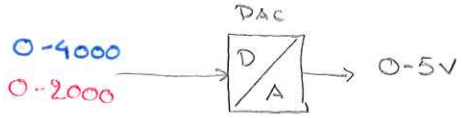


Exempelvis $U_{in} = 2{,}0\,V \Rightarrow$ 1600 / 800

Den analoga inmodulen innehåller egen intelligens med ett antal 16-bitars register som kommunicerar till / från CPU samt A/D - omvandlaren.

D/A-omvandlaren DAC sitter på de analoga utgångarna

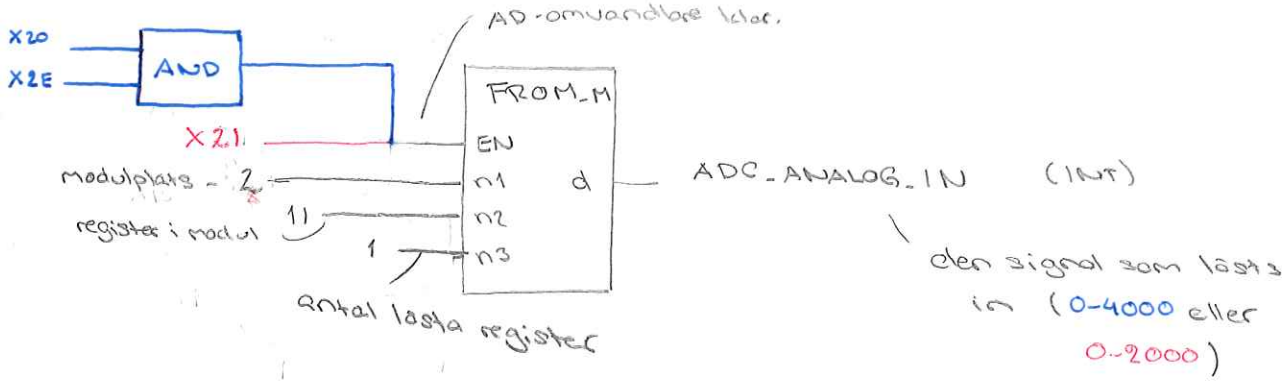
EH värde 0-4000 / 0-2000 omvandlas till analogt 0-5V och läggs ut på analog utgång.



Har egen intelligens och ett antal 16-bitars register.

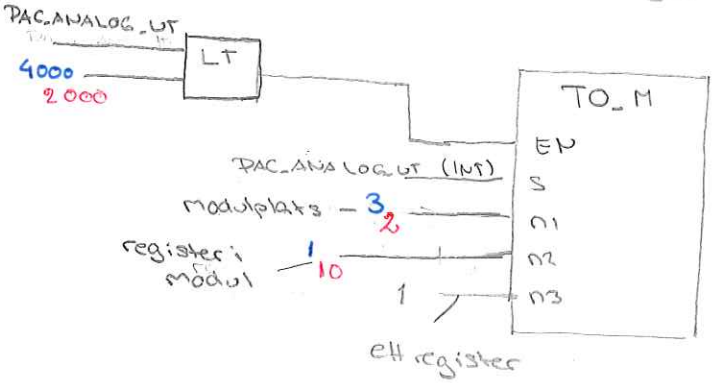
Vi behöver nu ett sätt att kommunicera med analogmodulen.

FROM_M läser från en intelligent modul (ADC) och skickar till CPU.

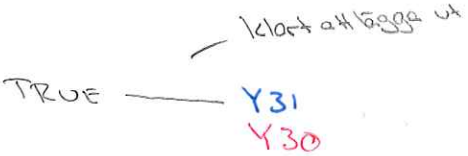


Se s. 70 resp. 73 i boken.

TO_M skriver till intelligent modul (DAC)



DAC_ANALOG_UT är den signal (INT) som ska läggas ut analogt



Se s. 71 resp 73 i boken