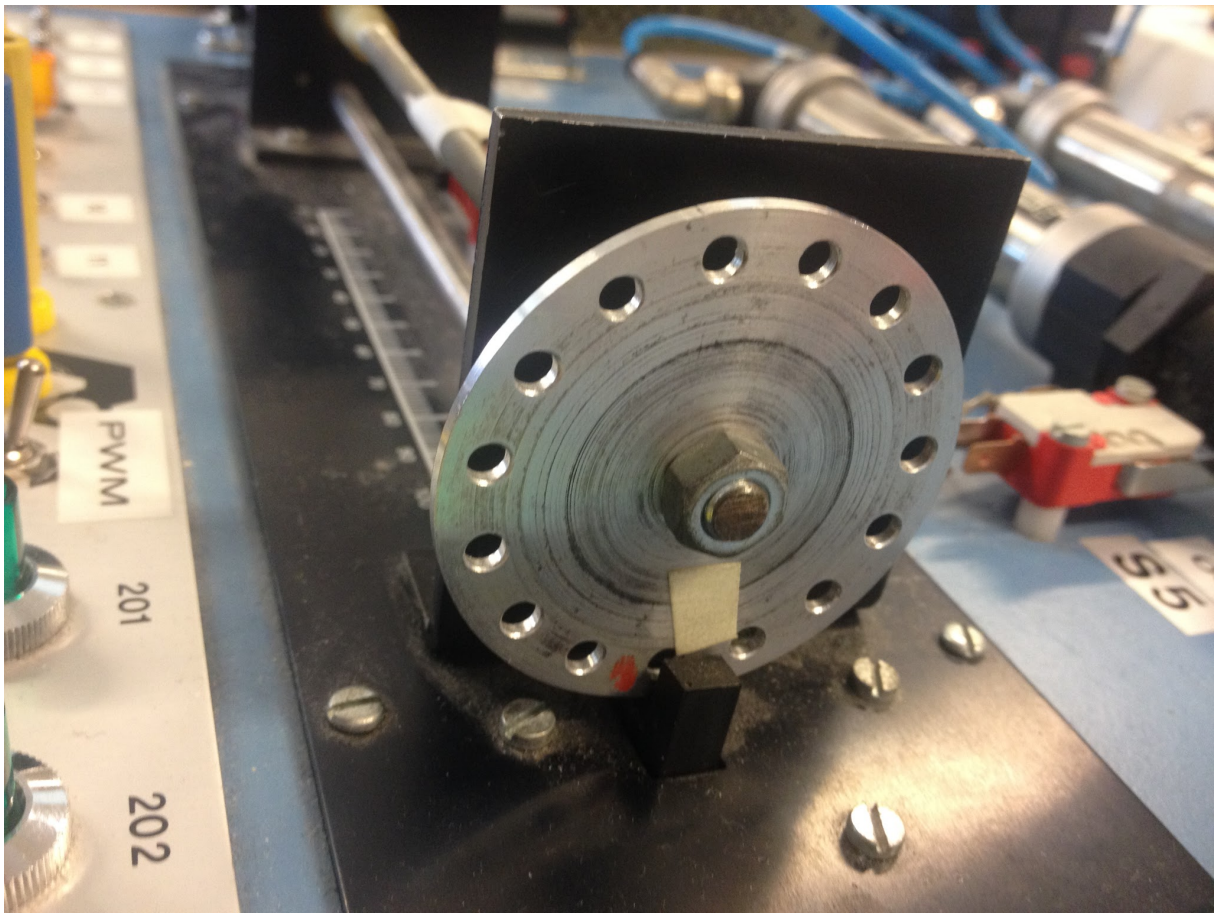
CHALMERS



Tilluftssystem

- automatiseringen av en mekanisk process

Projektarbete inom kursen LET085 Styrteknik

Andreas Johansson 19960813-8872 **TIELL**
John Croft 19930814-7959 **TIELL**

Institutionen för signaler o system
Avdelningen för reglerteknik, automation och mekatronik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2016

Sammanfattning

I denna rapport dokumenteras uppritningen och konstruktionen av ett automatiskt självreglerande feedback system i form av ett primitivt tilluftssystem. Syftet var att redovisa ett fungerande system som uppfyller kriterierna ställda av en unik specifikation. Arbetet behandlade huvudsakligen två typer av digitala styrenheter som skulle kommunicera med varandra och annan periferell hårdvara: en Programmable Logic Controller (PLC) och en mikrokontroller (MCU). Arbetet var till följd uppdelad i två övergripande delar som behandlade varje enhet och sina periferienheter för sig. Implementeringen i hårdvara krävde nya kunskaper inom olika elektronikkomponenters funktion och sätten som de kan användas för att behandla elektriska signaler, både digitala och analoga, för att göra kommunikation möjligt mellan varierande typer av elektriska kretsar. På mjukvarusidan krävdes bekantskap med det grafiska programspråket *funktion block diagram*, dock på en mycket grundläggande nivå, samt en tydlig förståelse av C-språket. Arbetet som beskrivs i denna rapport resulterade i ett fullt fungerande system som överensstämmer med specifikationen men som i vissa fall ger några mindre önskvärda lösningar till de ställda problemen.

Innehåll

1 Inledning.

Det här projektarbetet görs inom kursen *Styrteknik* som bygger på tidigare kurser inom programmering, elektronik och datavetenskap. Projektet är det sista praktiska arbetet under årskurs 1 av elektroingenjörsprogrammet.

1.1 Bakgrund.

Automatiska kontrollsystem används för att automatisera mekaniska och elektroniska processer. Från sina helt mekaniska föregångare har de ständigt utvecklats till dagens sofistikerade datorstyrda system. Idag är automatiseringen större än någonsin och många företag väljer att minska sina utgifter genom att automatisera så mycket av deras verksamhet som möjligt. Det vanligaste verktyget för dagens industriell automatisering är PLC:n och därför är PLC-programmering ett gynnsamt redskap för den blivande ingengören. Detta medför både konstruering av och felsökning av industriella processer och enheter.

1.2 Syfte.

Syftet med projektet är att som student kunna dra nytta av kunskaper från tidigare kurser för att slutligen designa och konstruera hårdvaran och mjukvaran till ett datoriserat styrsystem. Projektet är utformat för att se hur väl studenten tagit åt sig grunderna i elektronik, datavetenskap samt programmering. Dessutom skall studenten utifrån en teknisk specifikation, söka upp och komplettera kunskapsbrister på egen hand.

1.3 Mål.

Målet med projektet är att skapa ett styrsystem som fullföljer den utsedda specifikationen. Detta projekt behandlar ett tilluftsystem och för att lättare förstå funktionen så skall man tänka sig en byggnads tilluftsystem med en fläkt och tre spjällar som öppnar och stänger tre ventiler och en kontrollpanel för in och utdata. Styrsystemet skall använda en av användaren definierat *referensvärde* för att reglera fläkhastigheten och sedan självkorrigera eventuell avvikelse.

För att uppfylla specifiaktionen så skall tilluftsystemet i sin helhet uppnå följande konstruktionsmål:

- Ett justerbart analogt värde 0-5V skall skickas till PLC:n som utgör systemets *referensvärde* eller motorvarvtal. Värdet skall tolkas av systemet som ett tal med intervall 0-15 och enheten [varv/5sekunder].
    - *Referensvärdet* skall matas ut i binär representation på 4 lampor.
- Vid momentan tryckning av startknapp så skall första spjällen öppna helt och motorns varvtal sakta rampa upp mot *referensvärdet.*
- Efter att motorvarvtalet ‘rampats upp’ skall systemet konstant övervaka skillnaden mellan *referensvärdet* och det aktuella uppmätta varvtalet och korrigera eventuell avvikelse med avseende på *referensvärdet*.
- Det aktuella varvtalet skall mätas med samma enhet och intervall som *referensvärdet* och visas som ett decimalt tal på två sjusegmentsdisplayer.
    - Varvtalsintervallet är uppdelad i 4 områden och beroende på vilket område det uppmåtta varvtalet befinner sig i skall spjällarna stå i olika positioner.
        i. Spällarna skall slå om 1,5 sekunder efter att det uppmätta varvtalet hamnar inom ett område.
        ii. Vid ändring av position så skall eventuell öppnande spjäll och tillhörande ventil öppnas helt innan eventuellt stängande spjällar stänger.

- Vid momentan tryckning av stoppknapp skall motorn stanna och alla spjällar stängas.
- Systemet skall övervaka omgivningens temperatur och om den överskrider 30C, larma genom att blinka en ‘8’ på sjusegmentsdisplayen för decimala ‘ettorna’ medan sjusegmentsdisplayen för ‘tiorna’ är släckt.
- Motorn skall drivas av en PWM signal från MCU:n. Frekvensen skall ligga inom området 200-300 Hz.
- Ingång RA0 på MCU:n skall ej användas som analog ingång. Resten av I/O portarna får användas valfritt.

Dessa konstruktionsmål är verifierbara genom observation och med hjälp av extern mätutrustning.

1.4 Avgränsningar.

Projektets infallsvinkel är mestadels på systemets uppbyggnad med kopplingar och samarbetet mellan MCU och PLC samt programmeringen av dessa två komponenter. I rapporten undviker vi redovisning av interna register, mattematiska uträkningar och komponentdetaljer då vi förmedlar mestadels om hur komponenterna används i sin helhet, hur systemet är uppbyggt och även des funkonalitet.

Under arbetet så var tid en begränsande faktor eftersom vi bara hade tillträde till systemet i ca 40 timmar totalt. En materiell begränsning var valet av elektronikkomponenter. Vissa delar av arbetet kunde ha förenklats genom att använda komponenter som vi inte hade tillgång till.

2 Teoretisk referensram.

Det här kapitlet ger en kort genomgång av hårdvaran som behandlas i detta projekt. Tekniska detaljer tas upp, men även saker som hjälper till att sätta in komponenterna i sina tekniska och ibland historiska sammanghang och kan motivera anledningen till varför dessa komponenter används till att automatisera en mekanisk process.

2.1 Industriella styrsystem, PLC

Figur 1Mitsubishi PLC

En Programmable Logic Controller (PLC) är en enhet liknande den prestanda i ungefär samma klass. PLC är dock fysisk förstärkt då de miljö och förutsättningar såsom värme, fukt, mekanisk stress etc. Sl många andra programmbara enheter är att PLC:n använder sig utav unika för PLC:n. Dessa språk baserades ursprungligen på den tidiga logiken men har sedan dess utvecklats och blivit mer sofistikerade i av PLC sker genom att skriva själva programmet på en PC och sedan överföra progammen till PLC:n. PLC:n används lämpligast för automatisering av repetativa mekaniska processer då den alltid exekverar sitt program i kontinuerlig cykel. Till skillnad från en vanlig MCU är en PLC också helt modulär, vilket synns tydligt i bilden till höger, och olika funktioner kan läggas till som hårdvarumoduler utan att behöva skräddarsy en extern lösning.

För att kunna motstå elektriska störningar använder PLC:n en högre logiknivå än vanliga mikrokontrollers, 24V jämfört med den vanligare 5V eller 3,3V logiken. För att kunna tolka logiska nollor och ettor krävs desutom att logiksignalen har en ström i milliampär (mA) storleken.

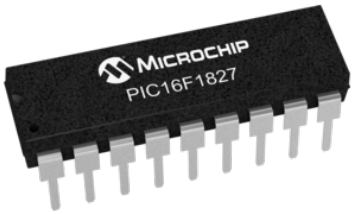
Utvecklingen av PLC:n började på sent 60-tal som ersättare åt den regerande elektromekaniska reläteknikеn [1]. Innan PLC:n uppfanns implementerades booleanska

logiksystem med reläomkopplare. Ett fullständigt system kunde innehålla tusentals med reläer och eftersom varenda en skulla manuellt handlindas och kopplas på rätt plats så var kostnaderna och kraven på ingenjörer enorma. För att 'programmera' om ett sådant system innebar i många fall att stora delar fick omarbetas fullständigt, beroende på komplexiteten.

På grund av det stora användsingsområdet för automatisering har PLC:n växt till att bli standardutrustning inom modern industri. Företag automatiserar de enklare processerna för att både spara tid och pengar då PLC:n erbjuder i många fall ett billigare och mer tidseffektiv alternativ än mänsklig arbetskraft. Dock medför PLC styrning risker, speciellt om den styr en last med hög effekt, då felprogrammering eller hårdvarufel kan ge farliga och eventuellt kostsamma konsekvenser

2.2 Mikrokontroller, PIC1827.

En mikrokontroller (MCU eller *microcontroller unit*) är en dator i den bemärkelsen att den har en central processorenhet, minne och input/output (I/O) portar, men på en och samma integrerad krets. Utöver detta kan det finnas olika periferienheter på kretsen som utökar funktionaliteten. Fördelen med att ha alla dessa vanligtvis diskreta moduler på samma krets är att energikraven, tillverkningskostnaderna och den fysiska storleken blir mycket mindre.



Figur 2Typisk mikrokontroller

Microchip(™) PIC16F1827 är på många vis en generisk MCU med ett mycket brett användningsområde på grund av sina många inbyggda funktioner. Den har 16 adresserbara I/O portar arrangerade i två 8-bitars register, port A och port B. Varje bit är en fysisk ingång/utgång till mikrokontrollern och refereras RA0-RA7 respektive RB0-RB7. Dessa bitar kan programmeras som antingen analoga eller digitala ingångar eller utgångar, eller så går det att använda speciella hårdvarufunktioner.

Kretsen kan använda antingen en extern eller intern oscillator och har en maximal frekvens på 32 MHz, men kan justeras så lågt som 31.25 KHz vilket även sänker energiförbrukningen. Den inställda oscillatorfrekvensen påverkar tidsberoende funktioner i mikrokontrollern som 'Interrupt Timers' och Pulse Width Modulation (PWM) signaler eftersom dessa beror på en viss klockfrekvens som är direkt beroende av oscillatorfrekvensen.

PIC18F1827 använder sig av 5V matningsspänning och logik och är därmed inte direkt elektriskt kompatibel med PLC:n som använder 24V logik. Dessutom är den förhållandevis känslig för elektriska störningar vilket kan leda till logiska 'misstolkningar' eller till och med att kretsen tar skada. Det krävs därför att signaler från externa enheter anpassas för att alltid hålla dem inom toleranserna för mikrokontrollern.

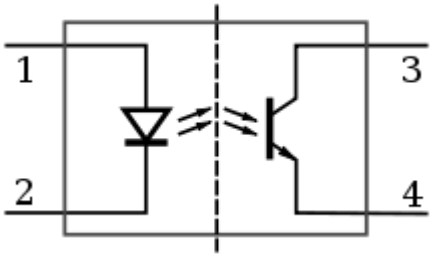
Kretsen kan programmeras i assemblerspråk eller C, det sistnämnda är dock att föredra eftersom kompilatorn sköter saker som minneshantering automatiskt. Programmering och kompilering görs genom Microchips *Integrated Development Environment* (IDE) som även har en *debugging*-suite som kan köra en interaktiv simulering av den skrivna koden på emulerad hårdvara och visa mikrokontrollerns interna register och andra variabler under programmets körning.

2.3 Beskrivning av elektronikkomponenter.

Det här kapitlet beskriver kortfattat övriga elektronikkomponenter som används i systemet.

- Optokopplare

Optokopplare används främst för att överföra signaler mellan galvaniskt isolerade kretsar. De består av en ljusdiod och en phototransistor som matas med var sin separata spänningskälla. Detta gör dem även lämpade som spänningsomvandlare mellan olika spänningar.



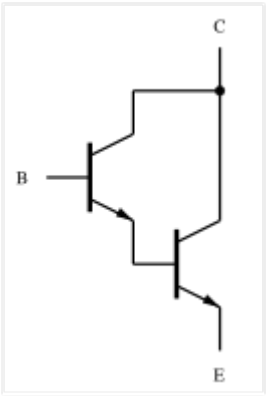
Figur 3 Optokopplare kretsschema

- NTC (Negative Temperature Coefficient) Termistor

NTC termistorer är elektriska motstånd som följer principen att resistansen minskar i takt med att termistorns temperatur ökar. Relationen är inte nödvändningvist linjär och kan i vissa fall definieras av komplicerade ekvationer.

- Darlington Transistor

Darlington transistorer består av två bipolära transistorer i 'serie' fast med en gemensam kollektor. Den första transistorn förstärks av den andra i serien vilket leder till en total strömförstärkning som är mycket högre än vad vanliga transistorer kan åstadkomma. Detta gör dem lämpliga för att driva högre elektriska laster såsom små motorer eller solenoider.



Figur 4 Darlington Transistor kretsschema

- Frihjulsdiod

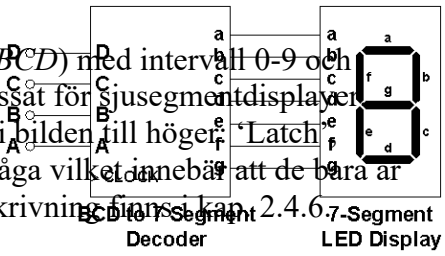
En fruhjulsdiod är en diod parallelkopplad med en motor eller annan induktiv last så att den leder i motsatt riktning till matningsspänningen. När strömmen till en induktiv last bryts uppstår en hög spänningspuls (*inductive spike*) vilket kan skada komponenter. För att åtgärda detta används en diod som en 'kortslutning' mellan terminalerna på lasten, vilket skapar en väg för spänningspulsen att ta tills energintill slut omvandlas helt till värme i lasten och kopplingarna.

- Sjusegmentsdisplay

Ett sjusegmentsdisplay är egentligen en array med 8 ljusdioder som antingen har gemensam anod eller katod. 7 dioder formar siffran och den sista utgör punkten. Punkten används inte i detta arbete.

- BCD till sjusegment avkodare (74*HC4511*)

En integrerad krets som tar ett 4-bitars BCD tal (även kallat *NBCD*) med intervall 0-9 och omvandlar den till det motsvarande decimala talet specialanpassat för sjusegmentdisplayer. Den har ytterligare tre speciella signalingångar som inte visas i bilden till höger: 'Latch' 'Blank' och 'Lamp Test'. Dessa hårdvarufunktioner är aktivt låga vilket innebär att de bara är aktiva när ingångarna är kopplade till jord. En utförligare beskrivning finns i kap. 2.4.6.



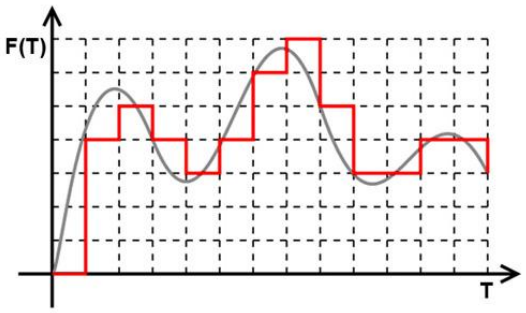
Figur 5 BCD till sjusegment-avkodare blockschema

2.4 Beskrivning av signalbehandling.

För att kunna styra olika enheter så krävs ofta en omvandling från en signal till en annan typ av signal. Här beskrivs de typer av signalbehandling som använts under uppbyggnaden av systemet.

2.4.1 Analog-till-Digital (A/D)

A/D omvandling används i både PLC:n och MCU:n för att ta in ett analogt värde (0-5V) och tolka det som ett digitalt heltal med storlek beroende på A/D omvandlarens upplösning. PLC:n har en upplösning som alltid är på 4000 diskreta steg medans MCU:n har en 10-bitars upplösning vilket innebär 1024 steg. För att förenkla programmeringen så väljer vi dock att slopa de två minst signifikanta bitarna och istället få ett 8-bitars tal, eller 256 diskreta steg.



Figur 6 Varje ruta representerar ett diskret steg i den givna signalupplösningen

PLC:n har två 'kanalar' som används för A/D omvandling medans PIC:en har 12 möjliga ingångar som kan programmeras för att behandla analoga signaler.

När en analog signal matas in i MCU:n så änvänds en strömbegränsande resistor och en keramisk kondensator för att skydda ingångarna samt stabilisera signalen. MCU:ns höga inresistans gör att signalen inte påverkas avsevärt av den strömbegränsande resistorn.

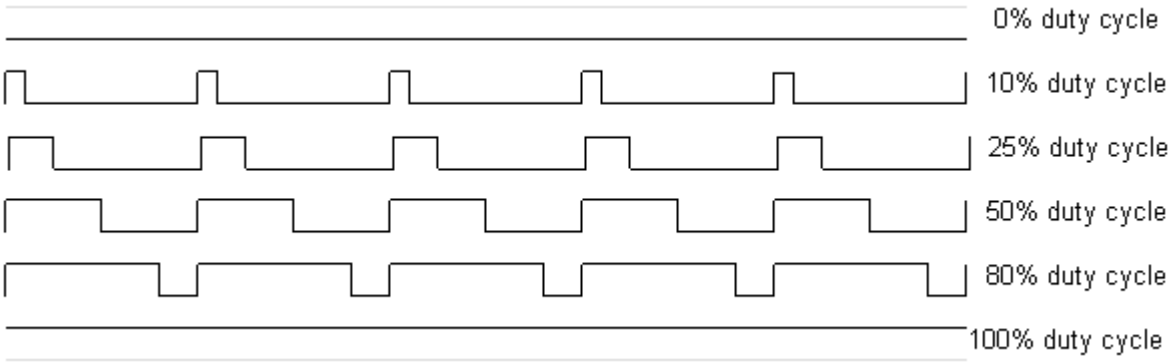
A/D omvandling används i stor utsträckning för att läsa av analog sensorutrustning. I det här fallet används en NTC termistor som varierar inspänningen beroende på temperaturen.

2.4.2 Digital-till-Analog (D/A)

D/A omvandling kan beskrivas som inversen till A/D omvandling i med att ett heltal inom något intervall omvandlas till en motsvarande analog signal inom ett annat intervall, ofta en spänning inom. I detta arbete används D/A omvandling i PLC:n för att omvandla ett heltal 0-4000 till en spänning 0-5V, vilket utgör styrsignalen för motorn. Detta tillåter finjusteringar i signalen.

2.4.3 Pulsbreddmodulering (PWM)

Pulsbreddmodulering eller *Pulse Width Modulation* (PWM) är en teknik som använder en logisk signal som snabbt växlar tillstånd. Förhållandet mellan tiden signalen är hög och när den är låg ger ett medelvärde som kan användas för att variera effekten över en last, i det här fallet en DC motor. PWM signalen är cyklisk och har en bestämd frekvens. Under varje period så är signalen hög eller låg en viss procent av tiden, detta kallas för 'pulsfaktorn' (*duty cycle*) och gör det möjligt att styra signalens medelvärde. Som ett exempel så kan man anta en matningsspänning på 5V och en PWM signal med godtycklig frekvens och en pulsfaktor på



Figur 7 PWM-signaler hur de syns på ett oscilloskop

50%. Eftersom 5V signalen är då bara till 50% av tiden och bara hälften av effekten överförs till lasten så erhålles ett medelvärde på 2,5V.

Fördelen med PWM styrning är att den har en mycket hög verkningsgrad eftersom signalen alltid är antingen helt till eller från. När signalen är låg så går ingen ström (bortsett från eventuella läckströmmar) och när signaler är hög så blir eventuell spänningsfall försumbart med hjälp av kompetent kretsdesign

2.4.4 Spänningsomvandling & störningsskydd

Om sammankopplade komponenter använder olika logiska spänningar så måste signaler mellan dem omvandlas så att de hamnar inom komponentens toleransnivåer. Alla spänningsomvandlingar görs med optokopplare som är galvaniskt isolerande. Detta har även fördelen att elektriska störningar från en komponent inte kan påverka en annan, vilket är speciellt viktigt inom ett industriellt sammanhang med förhållandevis extrema elektriska förhållanden.

2.4.5 Effektanpassning

Vid vanlig operation så kan MCU enheter bara driva ett tiotal milliwatt (mW) med sina signalutgångar. För att driva en större last krävs det ofta att signalen från styrenheten förstärks. I detta system används en darlington-transistor som ger en strömförstärkning på ca 750 gånger. Detta innebär att bara ett par mA från styreneheten kan driva en motor som kräver ett tiotal watt.

2.4.6 BCD-till-Sjusegment avkodning

Normalt krävs en signal för varje diod i sjusegmentdisplayen som skall drivas; med en så kallad ‘avkodare’ så används färre signaler för att skicka samma datainnehåll. BCD-omvandlaren är en kombinatorisk krets som tar in ett 4-bitars BCD tal (0-9) med fyra signaler och matar ut motsvarande tal på sjusegmentdisplayen genom sina 8 utgångar.

Övriga funktioner är en ‘Lamp Test’ som sätter alla utgångar höga oavsett vilket tal som matas in, ‘Blank’ som på motsvarande sätt sätter alla utgångar låga vilket släcker displayen och ‘Latch’ som, i inaktivt läge, låser värdet från ingångarna och gör att utsignalerna inte längra påverkas av BCD-bitarna. ‘Latch’ funktionen gör det möjligt att selektivt byta mellan olika BCD-avkodare av denna typ och därmed olika displayer. Detta används i ‘multiplexing’ där displayerna uppdateras var för sig i en hög frekvens vilket upplevs av ögat som att alla displayer uppdateras parallellt. ‘Blank’ och ‘Lamp Test’används i detta arbete för att på ett enkelt sätt kunna blinka displayer genom att växla mellan dem, ‘Latch’ används dock inte eftersom bara en display behöver skrivas till och därför finns inget behov av att välja display. ‘Latch’ ansluts darför inte till MCU:n.

Eftersom den ena sjusegmentdisplayen bara skall visa en etta eller visa blankt så ‘hårdkopplades’ en ‘1’ på BCD-ingångarna genom att bit 0 kopplas till VCC och bitarna 1-3 kopplas till jord. Enbart ingången för ‘Blank’ funktionen kopplades till MCU:n (se *BCD1* i kap. 4.3 *Kretsschema*). MCU:n kan då effektivt växla mellan att visa en etta eller släcka displayen helt med enbart ‘Blank’ signalen.

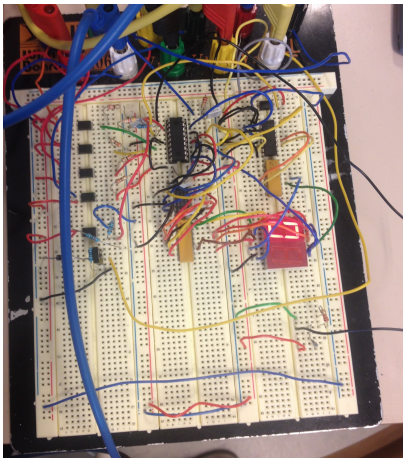
3 Metod.

Metodiken som tillämpades vid konstruktionen av tilluftsystemet kan beskrivas som en upprepandeprocess som var baserad på prövningar och successiva finslipningar. Denna

process ledde till en längre konstruktion av arbetet än förväntat vid brist på tidigare kunskap inom ämnet.

Projektet baserades runt PLC styrenheten vilket ledde till att projektet grundades med uppkoppling och programmering utav PLC. PLC kopplades upp till kopplingsdäcket där extern hårdvara satt, och grundfunktionerna programmerades. Grundfunktionerna i hårdvara som PLC skulle utnyttja var DAC, ADC och digitala utsignaler och insignaler. Då alla funktioner visade klartecken exekverades det samlade programmet som i sin helhet och felsöktes vid eventuella felbeetende som uppstod mellan funktionerna. När PLC:n hade sina basfunktioner fungerande kunde vi börja med kopplingsplattan och programmeringen av MCU:n.

Det fanns tydliga instruktioner på hur MCU:n skulle fungera med avseende på förutvalda ingångar och utgångar samt specifika C-kods kommandon, unika för just PIC16F1827. Med denna informationen var det möjligt att skapa ett ungefärligt kretschema på hur MCU skulle användas. Implementering av funktioner var till stor del gjord i den inbyggda hårdvaru- och mjukvarusimulatorn i IDE:n (*Integrated Development Environment*) innan koden fördes över till MCU:n. För att koppla MCU:ns olika funktioner i hårdvara användes en kopplingsplatta av *breadboard* varianten. Programmets funktioner testades var för sig för att bekräfta att de uppfyllde sitt syfte och eventuellt styrde hårdvaran korrekt, samt följde specifikationen. C-programmet kunde då testas i sin helhet och felsökas vid eventuella fel.

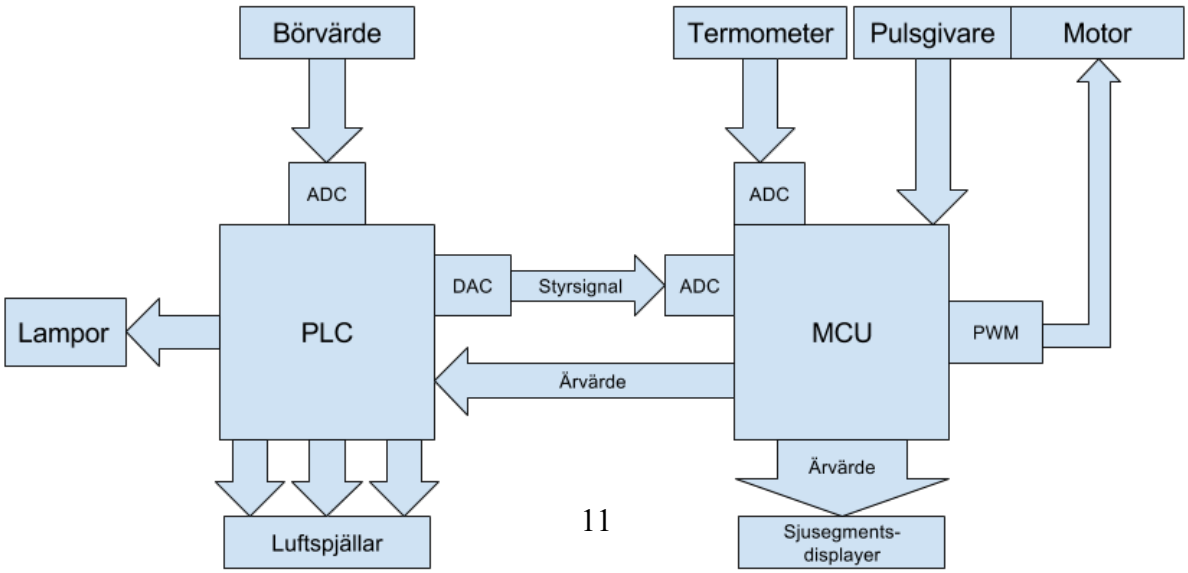


Figur 8 En typisk 'breadboard' *kopplingsplatta.*

Med både PLC- och MCU-programmen fullt fungerande individuellt, började sammankopplingen mellan dem. Detta hanterades i hårdvara med av hjälp optokopplare då nivåskillnaden i spänning var mellan 24V och 5V. Vid hopkoppling testades hela systemet som en enda enhet och felsöktes därefter. Vid lyckad kommunikation mellan de två enheterna implementerades vidare en slags ’sampling’ mekanism för att synkronisera dataöverföringen mellan dem. Med detta var systemet i princip färdigkonstruerad och testades mot specifikationskriterierna för att bekräfta att den uppfyllde dem vid alla lägen.

4 Genomförande.

Detta kapitel tar upp funktionsbeskrivningarna för både PLC och MCU individuellt. Kod genomgång och händelseförlopp för enheterna beskrivs tydligt samt hur de kommunicerar med varandra.

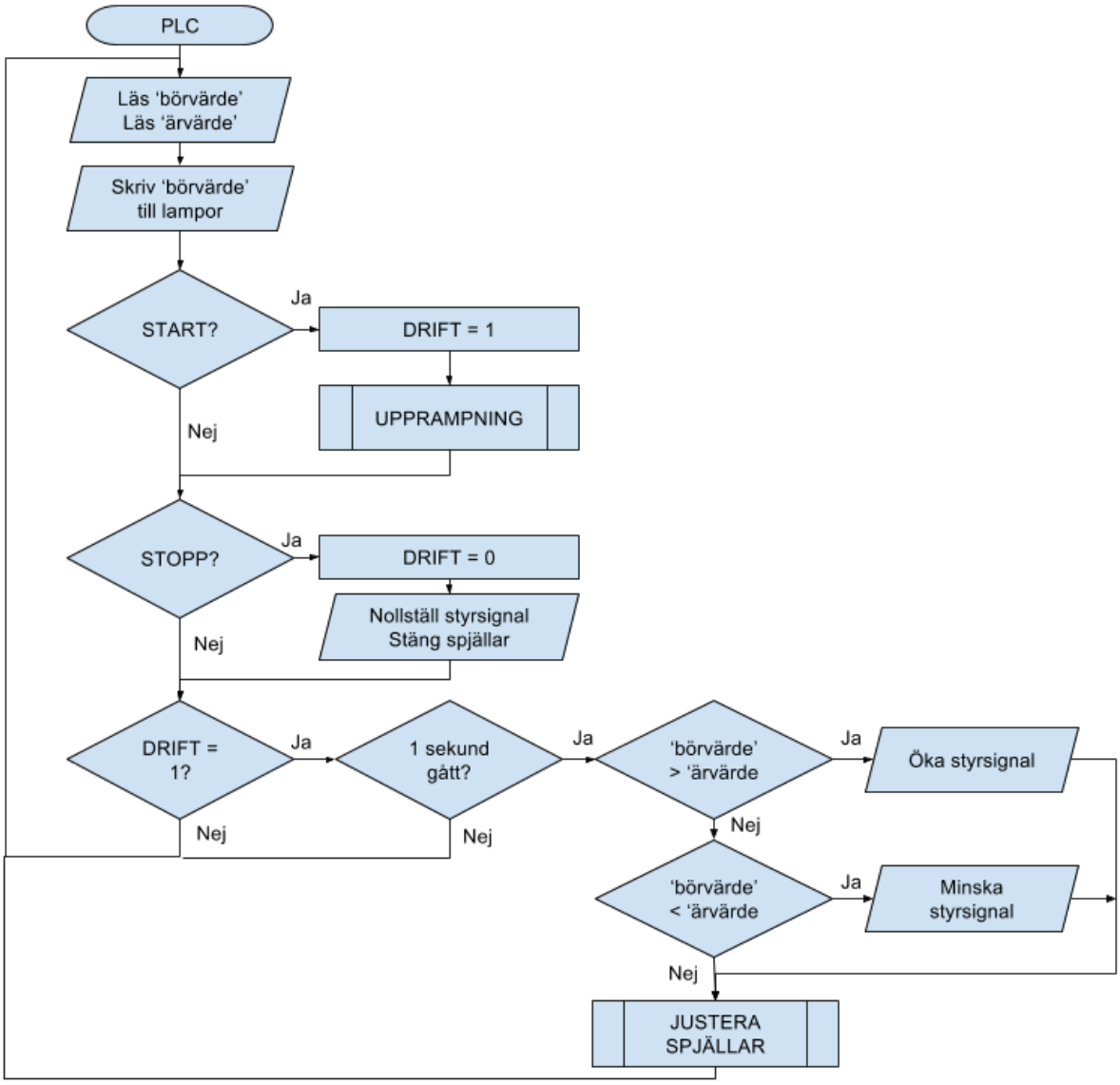


Figur 9 Här syns ett blockschema för hela systemet. 'Börvärdet' representerar det önskade värdet och 'ärvärdet' representerar det aktuellt uppmätta värdet

4.1 Funktionsbeskrivning – programmerbart styrsystem, PLC.

PLC:n har i uppgift att:

- Läsa det analoga *referensvärdet* och omvandla det till ett 4-bitars tal (0-15).
    - Skriva det omvandlade talet till 4 lampor representativa av dem 4 bitarna.
- Skicka en styrsignal till MCU:n för att indirekt driva motorn. Storleken på styrsignalen skall speglas i motorns varvtal.
- Med jämna mellanrum läsa det uppmätta varvtalet från MCU:n och jämföra den med *referensvärdet.*
    - Öka eller minska styrsignalen till MCU:n beroende på om det uppmätta värdet är större eller mindre än *referensvärdet.*
- Justera dem tre spjällarnas positioner beroende på det uppmätta varvtalet.
- Vid en momentan tryckning av startknapp, avvika från huvudprogrammet och istället sakta rampa upp varvtalet på motorn från stillastående tills det når *referensvärdet* och sedan återgå till huvudprogrammet.
- Vid momentan tryckning av stoppknapp stanna motorn och stänga alla tre spjällar.



Figur 10 Flödesschema för PLC-delen av systemet

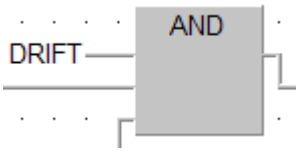
'Börvärdet' i flödesschemat ovan är en analog signal som ställs in av användaren och kan varieras mellan 0-5V med en extern potentiometer. Signalen tolkas då av PLC:ns A/D omvandlare som ett heltal mellan 0-4000. Talet delas med 250 för att erhålla ett 4-bitars heltal 0-15. Funktioner behövs i programmet för att välja specifika kanaler och eftersom PLC:n är modulär, även specifiera modulplatsen som AD-omvandlaren är installerad på. ADC funktionen visas i *Bilaga 4, sid. 2-3*.

Styrsignalen till motorn använder DA-omvandling för att skriva ett heltal 0-4000 som en analog spänning 0-5V. På samma sätt som i AD-omvandlingen måste stödfunktioner tillgås för att specifiera kanal och modulplats. DAC funktionen visas i *Bilaga 4, sid. 4*.

'Ärvärdet' är det uppmätta varvtalet i MCU:n och överförs som ett 4-bitars tal på 4 ingångar på PLC:n. Dessa bitar omvandlas till ett decimalt heltal 0-15 i programdelen "PIC_to_PLC" (se *bilage 4, sid. 5*) för att kunna användas i resten av programmet.

När 'börvärdet skall skrivas till 4 lampor via 4 utgångar (se *bilaga 4, sid. 3*) så tillämpas motsatta funktion till "PIC_to_PLC".

PLC-programmet kör alltid som ett enda stycke i en oändlig loop, men med hjälp av 'flaggbitar' så kan vi erhålla konstruktioner som funktionsmässigt liknar funktioner på andra högnivåspråk som C. I det här programmet är 'flaggbitar' booleanska enbitars variabler som används för att villkorligen aktivera viss funktionalitet. Detta görs genom att sätta dem på ENABLE ingången på många av funktionsblocken eller användningen av en logisk AND grind. Till höger visas ett exempel på en funktion som bara aktiveras när flaggbiten DRIFT är hög.

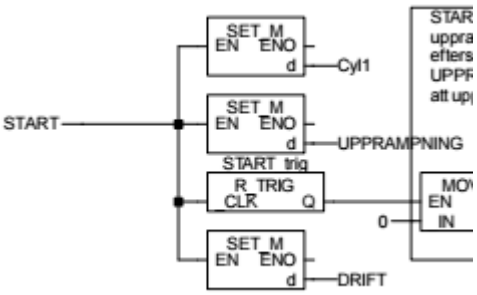


Figur 11

Programmet kan anta tre olika lägen:

- 'stopp'-läge där alla spjällar är stängda och styrsignalen till MCU:n är satt till 0. Övriga peripherella funktioner som visning av 'börvärdet' på lamporna är opåverkade.
- 'upprampning'-läge där systemet sakta rampar upp varvtalet på motorn från noll tills det att det uppmätta varvtalet är lika med 'börvärdet' eller *referensvärdet.*
- 'kör'-läge, läget den antar under normal drift, som kontinuerligt jämför det uppmätta varvtalet från motorn och varierar styrsignalen för att få den att följa 'börvärdet' så nära som möjligt.

'Upprampnings'-läge och 'kör'-läge är dock skilda från stopp'-läge i med att de båda innebär att motorn faktiskt är i drift. I programmet används därför en flaggbit "DRIFT" för att kunna villkorligen stänga av funktionerna från 'upprampning'- och 'kör'- läge när 'stopp'-läge är aktiverat. I *bilaga 4, sid. 6* visas hur signaler från momentanknappar med etiketter START och STOPP sätter DRIFT variabeln antingen LÅG eller HÖG. En till flaggbit förekommer här, "UPPRAMPNING", som skiljer 'kör'-läge från 'upprampning'-läge på samma sätt som DRIFT variabeln gjorde med 'stopp'-läget.



Figur 12 Aktivering av flaggbitar. SET_M blocken sätter variablerna höga

Både 'upprampning'-läget och 'kör'-läget består av liknande funktioner. Vid båda är takten av exekveringen bestämd av en slags klockpuls "SAMPE_PULSE" som pulsar HÖG i 1Hz. Trots benämningen är även den en flaggbit som villkorligen aktiverar dessa funktioner. Vid upprampning av motorn (se *bilaga 4, sid. 7*) är flaggbiten "UPPRAMPNING" hög och styrsignalen ökas med 25 steg (eller ca 0,03V) varje klockpuls tills 'ärvärdet' och 'börvärdet' är lika, varpå UPPRAMPNING variabeln sätts låg och funktionen avaktiveras.

I 'kör'-läget (se *bilaga 4, sid. 9*) så används samma klockpuls men funktionen är bara aktiv när 'ärvärdet' och 'börvärdet'har olika värden och UPPRAMPNING biten är låg. Detta

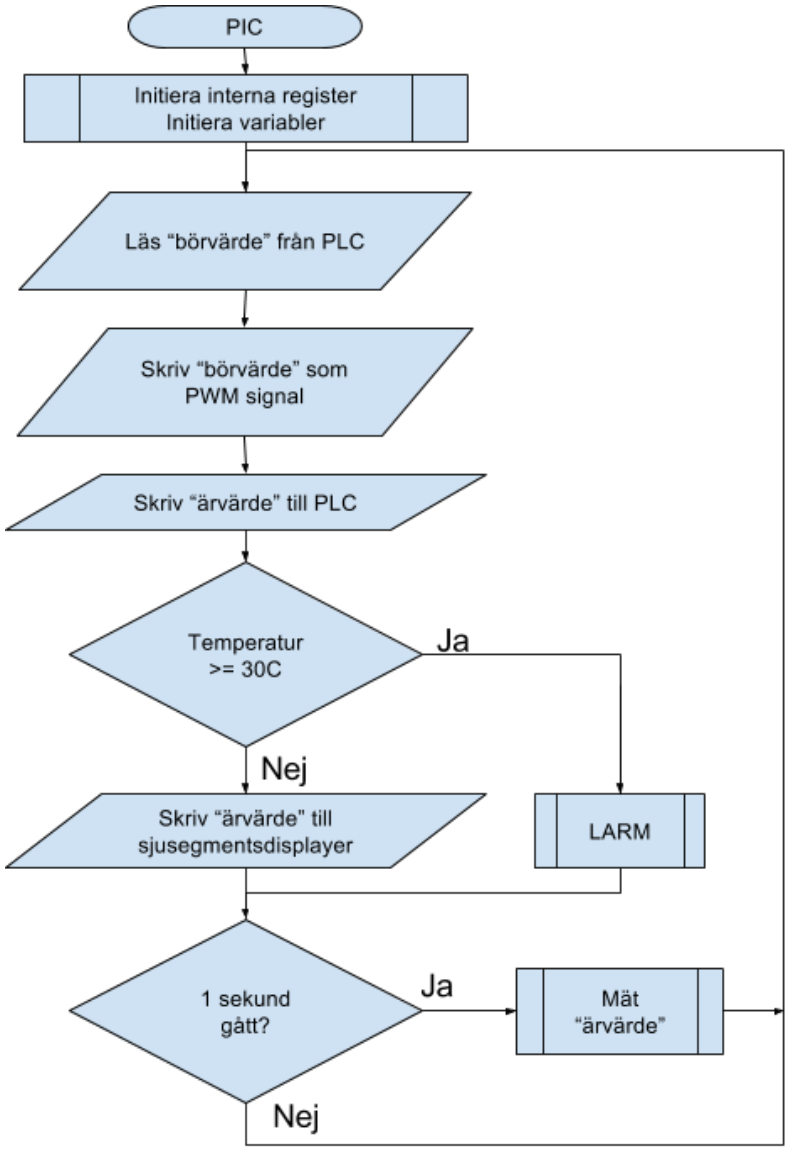
säkerställer att funktionerna inte ‘krockar’. Funktionen är även kapabel att subtrahera steg från styrsignalens värde för att minska motorns varvtal. Villkorliga gränssättande grindar används för att hålla styrsignalens värde och därmed motorvartalet inom specifikationen.

Klockpulsen genereras av en intern 1Hz pulsgivande register SM412 (se bilaga 4, sid. 9). Funktionen är egentligen anpassad för att includera en *prescaler*-liknande funktion vilket skulle multiplicera fördröjningstiden mellan varje klockpuls, men i programmet används den i 1:1 läge, dvs ingen extra fördröjning. Att allting sker i 1Hz är centralt till kommuniceringen med MCU:n eftersom det synkroniserar deras dataöverföringstakter och gör att PLC:n inte använder inaktuell data.

Beroende på det uppmätta varvtalet från MCU:n ställs tre pistonger i olika positioner med tryckluft reglerad av tre spjällar. För att detektera vilka positioner de befinner sig i används återfjädrande knappsensorer som kan känna av när varje pistong är antingen helt öppen eller stängd genom att de blir höga eller låga. Dessa sensorer används för att kunna öppna en öppnande pistong helt innan en stängande stänger sig. Positionerna beror på vilket intervall varvtalet ligger inom (se *bilaga 4, sid.7* för detaljer). Första spjällen skall, enligt specifikation, alltid vara öppen så länge programmet är i drift, det vill säga ‘upprampning’- eller kör’- läge (se *bilaga 4, sid. 6*).

Innan pistongerna ändrar position vid ett ändrat varvtal, så skall de enligt specifikationen vänta 1,5 sekunder. Om varvtalet fortfarande befinner sig inom samma intervall så skall positionen ändras, om inte så skall inget hända. Detta beteende implementerades i programmet med ett *TON* funktionsblock som väntar en förbestämd tid efter insignalen blir hög, och om insignalen fortfarande är hög efteråt, ändrar utsignalen till hög så länge som insignalen inte blir låg igen.

4.2 Funktionsbeskrivning – mikrokontroller, PIC1827.



PIC16F1827 mikrokontrollern har i uppgift att:

- omvandla styrsignalen från PLC:n till en motsvarande PWM signal som i sin tur styr motorhastigheten.
- med jämna mellanrum mäta det aktuella varvtalet (‘ärvärdet’) och skriva ut det till PLC:n och två sjusegmentsdisplayer.
- övervaka omgivningens temperatur och exekvera en larm-subrutin ifall temperaturen blir för hög.
    - i larm-subrutinen, skriva en blinkande ‘8’ till den ‘låga’ sjusegmentsdisplayen, displayerna ska i detta fall inte visa ‘ärvärdet’

Figur 13 Flödesschema över PIC-delen av systemet

Programmet är skrivet i C och är uppdelad i tre huvuddelar: initieringsfunktion, main-funktion och interruptrutin.

Initieringsfunktionen (*bilaga 3*) kallas av main-funktionen innan något annat och kör bara en gång. Den initierar interna register som styr det grundläggande beteendet hos mikrokontrollerns hårdvara som, till exempel, vilka ben som är ingångar eller utgångar. Main-funktionen (*bilaga 2*) upprepas i en oändlig *while*-loop så länge mikrokontrollern är igång och kallar på dem övriga subrutinerna. Interruptrutinen (*bilaga 3*) exekveras periodvis beroende på en intern klocka och kan inte kallas av några andra funktioner.

I header-filen (*bilaga 1*) finns alla defines, macron och funktionsprototyper. Här har även globala variabler deklarerats. Eftersom globala variabler ska användas i flera källfiler så

är det lämpligt att deklarera dem i header-filen och sedan ‘importera’ dem med ‘extern’ nyckelordet. Globala variabler används nästan exklusivt för att interruptrutinen skall kunna påverka programmet utanför sitt *scope*.

Under rubriken “Configuration Bits” (*bilaga 1*) finns många hårdvaruinställningar. Dessa är till mestadels tagna från tillverkarens dokumentation och kan betraktas som standardinställningar förutom att CLKOUTEN är ändrat till OFF. Denna inställning gör att microkontrollern inte lägger ut en pulståg (Fosc/4) på RA6 och tillåter användningen av porten som en vanlig I/O pin.

Eftersom många av funktionerna i programmet är förhållandevis tidskänsliga så är det viktigt att på ett bra sätt kunna mäta tid. Det här programmet använder sig av timerbaserade interrupts. Principen är att ett speciellt registerpar (TMR1L & TMR1H) på totalt 16 bitar inkrementerar ett tal i takt med MCU:ns klockfrekvens, och när talet spiller över vid 0xFFFF så genereras en interruptsignal. Om dessa register initieras till ett känt värde så är det möjligt att ställa in en mycket exakt fördröjning innan interrupten genereras. I detta program valdes ca 500ms. Programmet inkrementerar variabeln *halfsec* varje gång interruptrutinen exekveras för att erhålla en effektiv tidtagning i exakta halvsekunder.

Analog signalomvandling implementerades i funktionen ‘AD_omv’ som följer databladets egna exempel för hur det bör utformas. I *bilaga 1* visas hur funktionen tar porten där insignalen skall omvandlas som parameter, och returnerar det omvandlade värdet som ett 8-bitars osignerat heltal. Inparametern är ett 8-bitars osignerat binärt tal där endast en bit får vara hög. Den höga biten representerar ‘kanalen’ eller porten som ska omvandlas. Till exempel, 00000000 representerar AD kanal 0, medans 00010000 representerar AD kanal 5. Detta innebär dock att funktionen bara kan adressera 9 AD kanaler totalt, trots att MCU:n har fler att tillgå.

4.2.1 PWM

Innan PLC:ns styrsignal kan omvandlas till en PWM signal måste PWM-inställningar först väljas i initieringsfunktionen. Här används speciellt två ekvationer tagna från tillverkarens datablad för att välja pulsbredd och periodtid.

EQUATION 24-1: PWM PERIOD

$$PWM\ Period = [(PRx) + 1] \bullet 4 \bullet TOSC \bullet (TMRx\ Prescale\ Value)$$

Note 1: TOSC = 1/FOSC

Figur 15

EQUATION 24-2: PULSE WIDTH

$$Pulse\ Width = (CCPRxL:CCPxCON\langle 5:4\rangle) \bullet TOSC \bullet (TMRx\ Prescale\ Value)$$

Figur 14

För att ställa in rätt egenskaper krävs att man skriver till flera olika interna register givna av ekvationerna under initieringenfasen av programmet. För att veta exakt vilka register och värden används i detta program, se “Pulse Width Modulation” i *bilaga 3*. Specifikationen krävde att frekvensen på PWM-signalen (Fpwm) skulle vara mellan 200-300 Hz. Den enda möjliga frekvensen vid klockfreckvensen (Fosc) 4MHz som uppfyllde kravet visade sig vara ca 245 Hz och valdes därmed till det. I huvuddelen av programmet kan pulsbredden (och därmed spänningen på utsignalen) ändras genom att skriva ett 8-bitars tal till registret CCPR3L (enligt *EQUATION 24-2* ovan).

4.2.2 Varvtalsbestämning

Varvtalet mäts genom att funktionen *measure_rps()* varje sekund läser antalet pulser som tagits emot från pulsgivaren på motoraxeln. För att kunna ta emot och räkna externa pulser krävs det att interna registrerna CPSCON0 & OPTION_REG konfigureras under initieringensfasen. 8-bitars registret TMR0 räknar därefter externa pulser, men kan även skrivas till direkt i programmet, vilket gör det möjligt att nollställa den.

Pulsgivaren skickar 15 pulser per motorvarv. Måttenheten enligt specifikationen är varv-per-5-sekunder (varv/5s) men det är mycket opraktiskt att vänta så länge mellan mätningar, vi kompenserar därför med att mäta varje sekund och extrapolera värdet över 5 sekunder. Eftersom extrapolering på det här viset ökar eventuella felmarginaler så valdes 1s som en kompromiss mellan prestanda och noggranhet.

Det erhållna varvtalet (‘*arvarde’* i bilderna till höger) måste enligt specifikationen vara ett heltal 0-15, därför så sparas eventuella rester från divisionen inför nästa mätning med en modulo-operation.

Om funktionen skulle returnera ett avvikande heltal som är större än 15 så sätts den automatisk till 15 för att hålla den inom specifikationen.

```
unsigned char measure_rps(void){
    unsigned char arvarde;

    arvarde = TMR0; //Läser anta
    arvarde *= 5;   //Pulser ext

    arvarde /= 15;  //Delat med

    TMR0 = TMR0 % 3;        //TMR


    return arvarde;
```

Figur 16

```
 if (arvarde > 15)
 {
     arvarde = 15;
 }
```

Figur 17

4.2.3 Temperaturövervakning

MCU:n övervakar omgivningens temperatur genom funktionen ‘therm_check’ (se bild till höger) där den läser en analog spänning från en NTC termistor spänningsdelarkrets. För programmet innebär detta att ju högre det uppmätta värdet är, ju högre temperaturen.

Om ett värde högre än programmets fördefinierade gränsvärde, THERM_MAX (~30C i detta program), uppnås så initieras larmfunktionen.

Enligt specifikationen så skall larmet påverka sjusegmentsdisplayerna medans resten av systemet förblir opåverkad: displayen för dem låga siffrorna (0-9) skall visa en blinkande ‘8’ och displayen för dem höga siffrorna (10-) skall stängas av helt.

Detta beteende implementeras i flera steg i programmet. Första steget visas i *figur 18*: när en temperatur som överskrider maxgränsen läses så sätts båda displayerna i ‘blank’ läge (med omvänd logik), med andra ord de släcks, och den globala variabeln *alarm* sätts till ‘1’. Om det lästa temperaturen ligger under gränsen så händer det motsatta och displayerna fortsätter att fungera enligt normalt.

```
void therm_check(void){

    therm = AD_omv(2);

    if (therm >= THERM_MAX)
    {
        BL1 = 0;     //Displ
        BL0 = 0;
        alarm = 1;
    }else{
        BL0 = 1;     //Ifall
        LT = 1;      //'Lamp
        alarm = 0;
    }
}
```

Figur 18

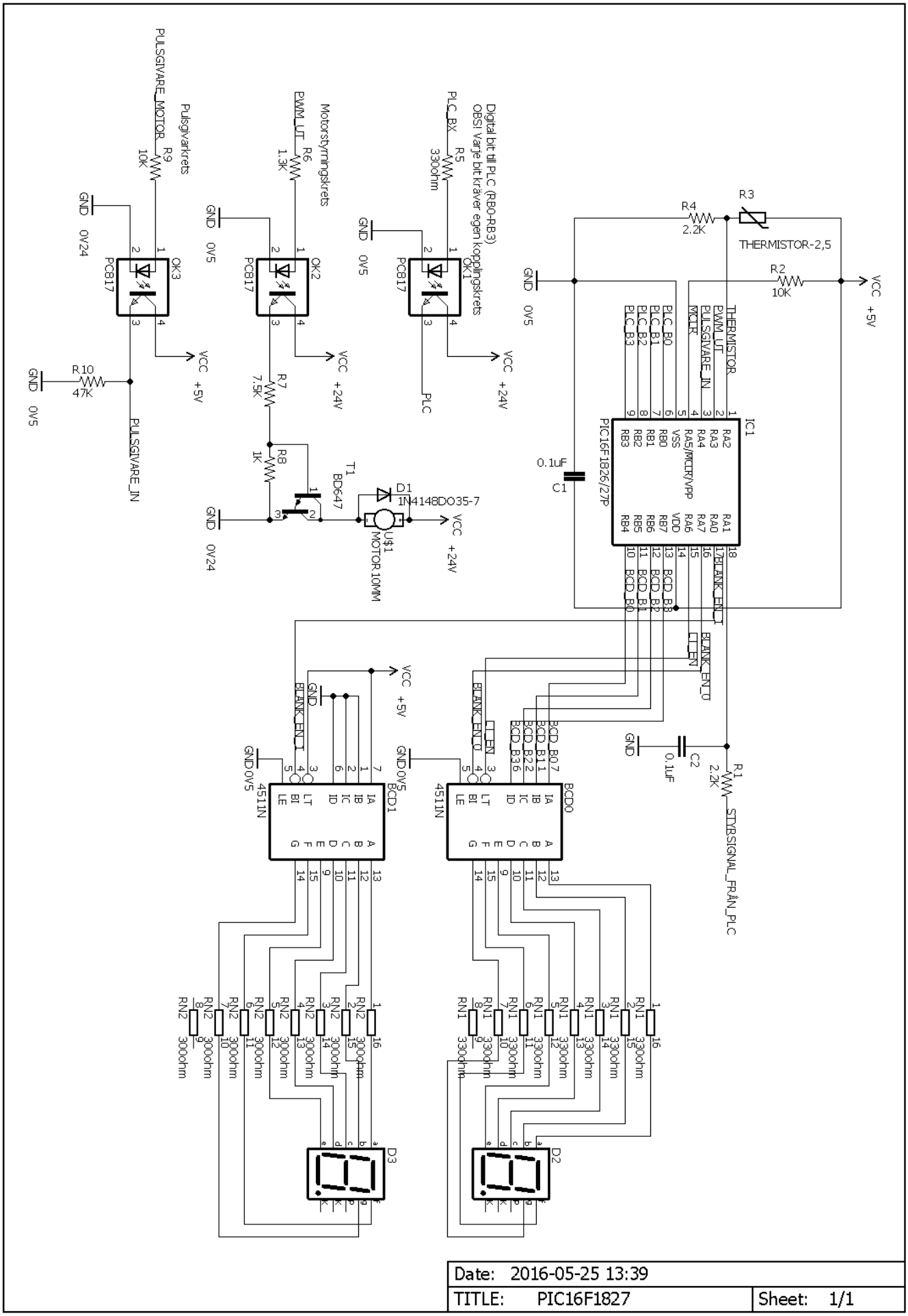
Nästa steg ligger i interruptrutinen: varje gång interruptrutinen exekveras (varje 500ms) så undersöks *alarm* variabeln från förra steget och om den är skild från 0 så växlas tillståndet på ‘Lamp Test’ (’*LT’ i figur 19*) funktionen. ‘Lamp Test’ är inversen av ‘blank’, en hårdvarufunktion som tänder alla segment på sjusegmentsdisplayen för decimala ‘ettorna’ och är ett behändigt sätt att ‘skriva’ en ‘8’ på displayen. Den har dock hårdvarumässig prioritet över ‘blank’ funktionen. På så sätt växlas tillståndet mellan en ‘8’ och blankt kontinuerligt och vi erhåller en ‘8’ som blinkar i 1

```
if(alarm)
    LT = !LT;
```

Figur 19

Hz på endast den ena sjusegmentsdisplayen så länge som temperaturen är över den fördefinierade maximala temperaturen.

4.3 Kretsschema.



Date: 2016-05-25 13:39		
TITLE: PIC16F1827		Sheet: 1/1

5 Resultat.

Alla målen uppnåddes på ett vis som fulföljde specifikationen. Systemet har kapaciteten att både rampa upp och sedan korrigera motorns varvtal med avseende på *referensvärdet* på ett snabbt och noggrant sätt. Spjällarna och alla periferella funktioner, som till exempel utmatning av data till lampor och sjusegmentsdisplayer och larmet, fungerar inom specifikationens ramar. PWM frekvensen från MCU:n bekräftades ligga inom det specifierade området (200-300Hz) på ca 245 Hz vilket mättes med oscilloskop.

6 Slutsatser och Kommentarer.

De största problemen som vi stötte på under arbetet gäller PLC:ns sätt att styra motornvarvtalet. Vi märkte snabbt att vid PLC:ns 'korrigering' av styrsignal för att öka eller sänka varvtalet så kunde det uppstå en situation där styrsignalen och därmed varvtalet hamnade utanför det tillåtna intervallet. Detta var en begränsning av den specifierade 4-bitars databussen som bara kan hantera varvtal upp till 15 (varv/5 sekunder). Lösningen var att se till att PLC:n korrigerar styrsignalen i en förhållandevis liten grad varje gång för att öka noggranheten.

Relaterat till detta är hur PLC:n kommunicerar med MCU:n för att ta emot det uppmätta varvtalet. Eftersom PLC:n ska reagera på det uppmätta varvtalet och korrigera styrsignalen till motorn, och det tar tid att faktiskt mäta varvtalet, så uppstod en situation där PLC:n reagerade flera gånger på samma, och därmed inaktuella, mätvärde. Egentligen ska PLC:n bara kunna reagera en gång per varje 'feedback'-värde. Lösningen som vi kom fram till är att periodvis, i detta fall varje sekund, läsa av och reagera på datan från MCU:n så att den hinner få en bra mätning. Genom att mätningen och läsning görs i så nära takt som möjligt så får vi det önskade beteendet. Denna lösning medförde dock sina egna problem och risken är överhängande att PLC:n och MCU:n kan bli osynkroniserade. Ett bättre alternativ skulle vara att implementera en funktion som explicit kommunicerar till PLC:n när det finns ett uppdaterat värde att hämta.

På grund av att dessa två problem inte har någon egentlig 'rätt svar' som passar alla system, så fick vi testa oss fram till en lämplig kombination av prestanda i form av uppdateringsfrekvensen och noggranhet i form av korrigeringsmagnituden.

Av processen lärde vi oss att det kan vara till fördel att börja planera med hjälp av flödesscheman, diagram och kretscheman vid ett mycket tidigare stadium för att ha alla delar klara och tydliga. I detta projekt var det svårt att hålla reda på alla olika delar vilket ledde till flera designmissar, omkonstruktioner och generellt slarvig kodskrivning.

Bilagor.

Här samlas alla bilagor som använts under projektet i sina ursprungliga former.

Bilaga 1 - PIC16F1827 - tilluft.h

```
//Header filer-------------------------------------------------------------
#include <xc.h>
//Macros & defines---------------------------------------------------------
#define _XTAL_FREQ 4000000   // __delay_ms/us(x)

#define BL0 LATAbits.LATA7  //Blank Enable display 0.    Aktiv LÅG.
#define BL1 LATAbits.LATA0  //Blank Enable display 1.    Aktiv LÅG.
#define LT LATAbits.LATA6   //Visar en '8' på display 0. Aktiv LÅG.
/* OBS! LT tar prioritet över BL0 i hårdvara. */

/* BCD bitar [0-9] till 7-segment display drivaren */
#define D0 LATBbits.LATB4
#define D1 LATBbits.LATB5
#define D2 LATBbits.LATB6
#define D3 LATBbits.LATB7

/* Binära bitar [0-15] till PLC */
#define b0 LATBbits.LATB0
#define b1 LATBbits.LATB1
#define b2 LATBbits.LATB2
#define b3 LATBbits.LATB3

/* Här kalibreras gränsvärdet för temperaturen som ska utlösa larmet.
 * Värdet är en analog signal 0-255 */
/* Högre värde == högre temperatur */
#define THERM_MAX 127

//Configuration Bits-------------------------------------------------------
//Se C:\Program Files\Microchip\xc8\Vx.xx\docs\pic_chipinfo
//Oscillator Intern, Watchdog timer disabled, Power-Up timer enabled
//Brownout Reset Disable, Low-Voltage Programming Disabled, Unprotect memory,
//MasterClear Input Enabled, Debug disabled,FCM/IESO disabled
#pragma config CPD=OFF, BOREN=OFF, IESO=OFF, FOSC=INTOSC, FCMEN=OFF, MCLRE=ON,\
 WDTE=OFF, CP=OFF, PWRTE=ON, CLKOUTEN=OFF //Config Word 1
#pragma config PLLEN=OFF, WRT=OFF, STVREN=ON, BORV=LO, LVP=OFF //Config Word 2

//Global Variables - Deklaration-------------------------------------------
unsigned char halfsec, therm, alarm;
/*halfsec - räknar antalet halvsekunder*/
/*therm - uppmätta termometervärde*/
/*alarm - flaggbit som används i ISR för att avgöra om display ska blinka.
 *                                  1:a blinka, 0:a statisk */
```

```
//Funktionsprototyper-------------------------------------------------------
void init(void);
/* Initiala värden i kritiska interna register sätts */
void interrupt isr(void);
/* Interrupt subrutin. Timer baserat. Exekveras varje 0.5 sekunder. */
char AD_omv(char ADkanal);
/* Omvandlar en analog signal till digital på den specifierade ingången.
 * Ingångar refereras med binära tal t.ex. ingång 3 == 0b00001000 == 0x08 */
void BIN_out(unsigned char arvarde);
/* Lägger ut "ärvärdet" som ett BIN tal på utgångar b0-b3 */
void BCD_out(unsigned char arvarde);
/* Lägger ut "ärvärdet" som ett BCD tal på utgångar D0-D3 */
unsigned char measure_rps(void);
/* Mäter varvtalet [enhet varv/5sek] */
void therm_check(void);
/* Läser av termometern och utlöser larmet om värdet överskrider THERM_MAX */
```

Bilaga 2 - PIC16F1827 - main.c

```
#include <xc.h>
#include "tilluft.h"

//Globala Variabler - Ursprunglig declaration i .h-fil----------------------
extern unsigned char halfsec, therm;

//Huvudprogram--------------------------------------------------------------
void main(void) {
   init();

   BL0 = 1;    //Låg 7seg disp. på.
   BL1 = 0;    //Hög 7seg disp. av.

   alarm   = 0;
   halfsec = 0;

   unsigned char arvarde = 0;

   /* Main Loop */
   while(1)
   {
      CCPR3L = AD_omv(1);     //Det som uppmätts på AN1 (börvärdet) matas direkt
                              //ut som en PWM signal [0-255] på RA3.

      if (halfsec == 2)       //Här kan väljas hur ofta koden inom if-satsen ska exekveras.
                             //I detta fall varje sekund.
      {
```

```
        halfsec = 0;                    //Nollställ räknaren
        arvarde = measure_rps();    //Mät varvtal.

        if (arvarde > 15)       //Tal över 0xF är otillåtna,
                                //t.ex. 0b00001 dvs. > 4-bitars tal
        {
           arvarde = 15;       //Begränsas till 15 oavsett.
        }
      }

      BIN_out(arvarde);       //Ärvärdet matas ut till PLC:n och display drivaren.
      BCD_out(arvarde);

      therm_check();          //Temperaturen mäts och jämförs med THERM_MAX.
                              //Löser ut larmet om temperaturen för hög.
   }
}
```

Bilaga 3 - PIC16F1827 - tiluft.c

```
//Header Filer-------------------------------------------------------
#include <xc.h>
#include "tilluft.h"

//Globala Variabler--------------------------------------------------
extern unsigned char therm;
extern unsigned char halfsec;
extern unsigned char alarm;

//Funktioner---------------------------------------------------------
void init() {
   OSCCON = 0b01101000;    //Fosc = 4MHz == 1101
   LATA   = 0x00;              //Nollställer alla bitar i PORTA
   LATB   = 0x00;              //Nollställer alla bitar i PORTB
   ANSELA = 0b00000110;    //0 = Digital, 1 = Analog. RA1 o RA2 analoga ingångar.
   ANSELB = 0x00;
   TRISA  = 0b00110110;    //Ingångar: MCLR(RA5), T0CKI(RA4), AN1(RA1), AN2(RA2).
                           //Resten utgångar.
   TRISB  = 0b00000000;    //Sätter hela PORTB till utgångar.

   //Pulsräknare-------------------------------------------------------------------------
   TMR0       = 0x00;          //Nollställer pulsräknaren
   CPSCON0    = 0X00;          //Välj external clock source. Dvs.
                               //vi räknar nu pulser utifrån.
   OPTION_REG  = 0b00101000;    //ext. clock source, rising edge, prescaler disabled(1:1)

   //Timer1 & overflow interrupt---------------------------------------------------------
   T1CON  = 0b00110001;     //Clock Source=Fosc/4, Prescaler 1:8, Internal Clock, Timer1 ON
   T1GCON = 0b00000000;    //Gate control register; all fucntions disabled.
```

```
    TMR1L  = 0xDF;          //Dessa två register utgör ett 16-bitars initialvärde
                            //som sedan räknas upp i den takt som bestäms av T1CON och T1GCON.
    TMR1H  = 0x0B;          //ISR exekverar när talet spiller över.
                            //På så sätt kan vi bestämma hur mycket vi vill fördröja ISR.

    PIE1   = 0x01;                  //TMR1 overflow interrupt enable
    INTCON = 0b11000000;    //Global and peripheral interrupt enable

    //ADC----------------------------------------------------------------------------
    ADCON1 = 0b01000000;    //Vänsterjusterat, A/D Conversion Clock Fosc/4
    ADCON0 = 0b00000001;    //ADON

    //Pulse Width Modulation---------------------------------------------------------
    //OBS! CCP3(RA3) används till PWM!
    CCP3CON = 0b00001100;       //2 LSB förkastade för att få ett 8-bitars tal, CCP3 i PWM mode
    CCPTMRS = 0x00;             //Välj så att CCP3 använder Timer 2.
    PR2     = 254;              //Då CCPR3L := 255, önskar vi tp := Tpwm,
                                        //därför 1020 := (PR2+1)*4.
                                        //Komplett formel i datablad.
    T2CON   = 0b00000110;       //Timer 2 ON, Prescaler=16. Ger Fpwm = 245 Hz.
}

char AD_omv(char ADkanal){
    ADCON0 = (ADCON0 & 0b10000011) | (ADkanal << 2);  //Aktiverar rätt AD kanal.
    __delay_us(5);              //delay 5us för Tacq.
    ADCON0bits.GO = 1;          //AD omvandling startar.
    while(ADCON0bits.GO);       //Väntar på registertillståndet att förändras.

    return ADRESH;          //Returnerar 8 MSB. (2 LSB slopas)
}

void BIN_out(unsigned char arvarde){
    //bit 3
    b3 = arvarde / 8;
    arvarde %= 8;
    //bit 2
    b2 = arvarde / 4;
    arvarde %= 4;
    //bit 1
    b1 = arvarde / 2;
    arvarde %= 2;
    //bit 0
    b0 = arvarde;
}

void BCD_out(unsigned char arvarde){

    unsigned char tior, ettor;
```

```
   /* Tvåsiffriga DECIMALA tal representeras av två tal 0-9 */


   if (alarm == 0)     //Om larmet är av, så vill vi ovillkorligen stänga av
                       //"Lamp Test" hårdvarufunktionen.
      LT = 1;

   tior = arvarde / 10;
   if (tior && !alarm) //Om larmet är AKTIV så får inte BL1 påverkas av funktionen
   {
      BL1 = 1;    //Om tior är 0 (t.ex. 00-09) så släcks den höga displayen.
   }else{
      BL1 = 0;    //Om tior >0 (t.ex. 10-15) så tänds den höga displayen för att visa en etta.
   }

   ettor = arvarde % 10;

   /* ettor matas ut som ett BCD tal [0-9] på den låga displayen */
   D3 = ettor / 8;
   ettor %= 8;
   D2 = ettor / 4;
   ettor %= 4;
   D1 = ettor / 2;
   ettor %= 2;
   D0 = ettor;
}

void interrupt isr(void){
   if (PIR1bits.TMR1IF && PIE1bits.TMR1IE)     //Vilken interrupt är aktuell
  {
      halfsec++;          //Räknar upp antalet halvsekunder som gått.
                          //Vi är begränsade till 0.5 sekunder av hårdvaran.
                          //Detta därför att timer modulen bara rymmer 16-bitar.

      if(alarm)
          LT = !LT;       //Display 0 (ettor) blinkar varje halvsekund.
                          //Använder 'Lamp Test' hårdvarufunktionen i BCD drivaren
                          //för att tända alt. släcka alla segment (excl. punkt) samtidigt.

                          //Fungerar i med att displayer släcks av 'BLANK_ENABLE'
                          //men tillståndet kan växlas till det motsatta med 'LAMP_TEST'
                          //som tar prioritet.

      TMR1L = 0xDF;               //Återställer TMR1 så att
      TMR1H = 0x0B;               //delay == 0,5 s

      PIR1bits.TMR1IF = 0;        //Nollställer/kvitterar interruptflaggan.
   }
}
```

```
unsigned char measure_rps(void){
   unsigned char arvarde;

   arvarde = TMR0;      //Läser antalet pulser som skett sedan förra läsningen.
   arvarde *= 5;        //Pulser extrapolerade över 5 sek för att få enheten [pulser/5sek]

   arvarde /= 15;       //Delat med 15 pulser för att få [varv/5sek]

   TMR0 = TMR0 % 3;     //TMR0 får värdet av resten.


   return arvarde;
}

void therm_check(void){

   therm = AD_omv(2);              //Läs av termometermodulen.

   if (therm >= THERM_MAX)     //THERM_MAX kalibreras i .h-filen.
   {
      BL1 = 0;    //Display 1 (tior) och display 0 (ettor) släcks.
      BL0 = 0;
      alarm = 1;
   }else{
      BL0 = 1;    //Ifall 'display 0' har släckts av annan funktion, måste den tändas igen.
      LT = 1;     //'Lamp Test' hårdvarufunktionen stängs av
      alarm = 0;
   }
}
```

Bilaga 4 - PLC Q02 program

Project

Tilluft

Project Number	**N/A**
Project Manager	**N/A**
PLC Type	**Q02**
Comment	**Styrteknik LET085**

				Date			2016-05-25 13:05:30	E:\Sync\Chalmers\Styrtekni...\Prog_main
				Drawn				
				Appr.				STYRTEKNIK LET085 Page: 1
Rev	Change	Date	Name	Rel.				

Comp (Prio = 31, Event = TRUE)

	POU name	Comment
0	ADC	ADC
1	Rampning	Fläkt_ramp
2	DAC	DAC
3	Varvtalsjustera	Jämför mellan börvärde och ärvärde och justerar varvtalet därefter.
4	PIC_to_PLC	Tar in digitalt värde 0-F genom fyra inportar.

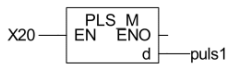
Global Variable List

	Class	Identifier	MIT-Addr.	IEC-Addr.	Type	Initial	Comment	Remark
0	VAR_GLOBAL	BORVARDE_A	D0	%MW0.0	INT	0	Önskade värdet. Analog. Ställs in på POT.	
1	VAR_GLOBAL	ARVARDE_D	D3	%MW0.3	INT	0	Ärvärde digitalt 0-F. Från PIC.	
2	VAR_GLOBAL	BORVARDE_D	D4	%MW0.4	INT	0	Börvärde digital 0-F. Från PLC till lampor	
3	VAR_GLOBAL	STYR_TILL_PIC	D5	%MW0.5	INT	0	Värde 0-4000 som konverteras till analog spänning och ges till PIC.	
4	VAR_GLOBAL	Cyl1	Y13	%QX19	BOOL	FALSE		
5	VAR_GLOBAL	Cyl2	Y14	%QX20	BOOL	FALSE		
6	VAR_GLOBAL	Cyl3	Y15	%QX21	BOOL	FALSE		
7	VAR_GLOBAL	Cyl1Bak	XA	%IX10	BOOL	FALSE		
8	VAR_GLOBAL	Cyl1Fram	XB	%IX11	BOOL	FALSE		
9	VAR_GLOBAL	Cyl2Bak	XC	%IX12	BOOL	FALSE		
10	VAR_GLOBAL	Cyl2Fram	XD	%IX13	BOOL	FALSE		
11	VAR_GLOBAL	Cyl3Bak	XE	%IX14	BOOL	FALSE		
12	VAR_GLOBAL	Cyl3Fram	XF	%IX15	BOOL	FALSE		
13	VAR_GLOBAL	lampa1	Y16	%QX22	BOOL	FALSE	lampor	
14	VAR_GLOBAL	lampa2	Y17	%QX23	BOOL	FALSE	lampor	
15	VAR_GLOBAL	lampa3	Y18	%QX24	BOOL	FALSE	lampor	
16	VAR_GLOBAL	lampa4	Y19	%QX25	BOOL	FALSE	lampor	
17	VAR_GLOBAL	PICb0	X0	%IX0	BOOL	FALSE		
18	VAR_GLOBAL	PICb1	X1	%IX1	BOOL	FALSE		
19	VAR_GLOBAL	PICb2	X2	%IX2	BOOL	FALSE		
20	VAR_GLOBAL	PICb3	X3	%IX3	BOOL	FALSE		
21	VAR_GLOBAL	puls1	M0	%MX0.0	BOOL	FALSE	init puls ADC	
22	VAR_GLOBAL	puls2	M1	%MX0.1	BOOL	FALSE	init puls DAC	
23	VAR_GLOBAL	START	X4	%IX4	BOOL	FALSE	Momentan START knapp	
24	VAR_GLOBAL	STOPP	X5	%IX5	BOOL	FALSE	Momentan STOPP knapp	
25	VAR_GLOBAL	VARVTAL	D2	%MW0.2	INT	0	Enhet [antal varv/5s]. Representeras 0-F.	
26	VAR_GLOBAL	UPPRAMPNING	M2	%MX0.2	BOOL	FALSE		
27	VAR_GLOBAL	DRIFT	M3	%MX0.3	BOOL	FALSE	Om låg, skickar ingen styrsignal til PIC:en	
28	VAR_GLOBAL	NOLLST	M4	%MX0.4	BOOL	FALSE	Nollställer sample_puls	
29	VAR_GLOBAL	SAMPLE_PULSE_COUNTER	D6	%MW0.6	INT	0	Räknar upp 5 pulser i 1Hz. Triggar sample var 5:e sekund	
30	VAR_GLOBAL	SAMPLE_PULSE	M5	%MX0.5	BOOL	FALSE	Puls som aktiverar stickprov av värdet på digitala databussen.	

ADC [PRG] Header

ADC [PRG] Body [FBD] Network#1

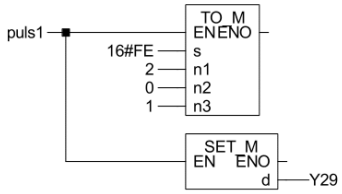
Network #1 (1) Label: Title:



Rising edge flank trigger.
Functionally equivalent to R_TRIG.

ADC [PRG] Body [FBD] Network#2

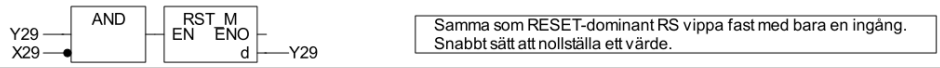
Network #2 (1) Label: Title:



Enables the DAC/ADC module. bits 0-4 enable/disable ch.1-4. 0/1 -> enable/disable.

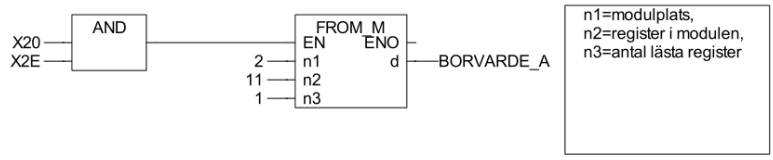
ADC [PRG] Body [FBD] Network#3

Network #3 (1) Label: Title:



ADC [PRG] Body [FBD] Network#4

Network #4 (1) Label: Title:

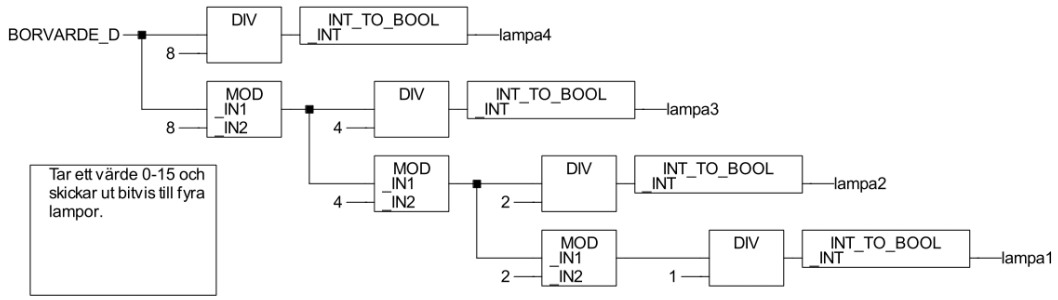


ADC [PRG] Body [FBD] Network#5

Network #5 (1) Label: Title:

ADC [PRG] Body [FBD] Network#6

Network #6 (1) Label: Title:



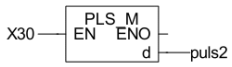
				Date			2016-05-25 13:05:30	E:\Sync\Chalmers\Styrtekni...\Prog_main
				Drawn				ADC [PRG] Body [FBD] Network#3
				Appr.				STYRTEKNIK LET085 Page: 3
Rev	Change	Date	Name	Rel.				

DAC [PRG] Header

	Class	Identifier	Type	Initial	Comment
0	VAR	VIPPA1	RS		
1	VAR	VIPPA2	RS		
2	VAR	TIMER1	TON		

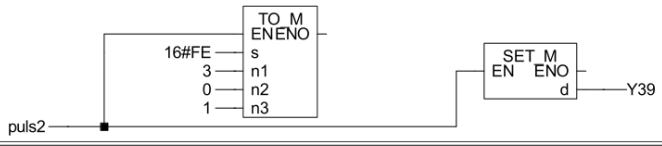
DAC [PRG] Body [FBD] Network#1

Network #1 (1) Label: Title:



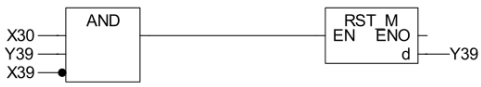
DAC [PRG] Body [FBD] Network#2

Network #2 (1) Label: Title:



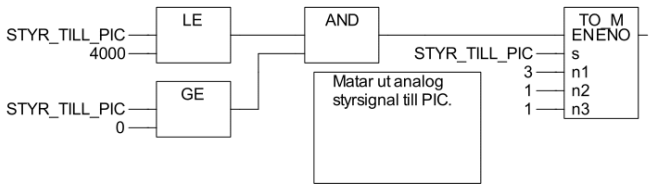
DAC [PRG] Body [FBD] Network#3

Network #3 (1) Label: Title:



DAC [PRG] Body [FBD] Network#4

Network #4 (1) Label: Title:



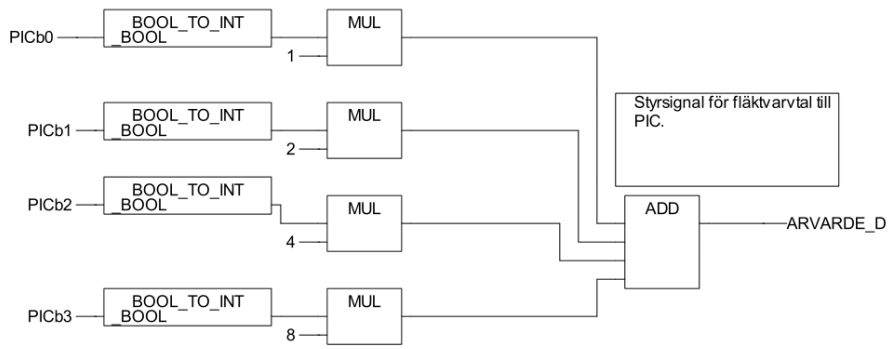
DAC [PRG] Body [FBD] Network#5

Network #5 (1) Label: Title:

PIC_to_PLC [PRG] Header

PIC_to_PLC [PRG] Body [FBD] Network#1

Network #1 (1) Label: Title:



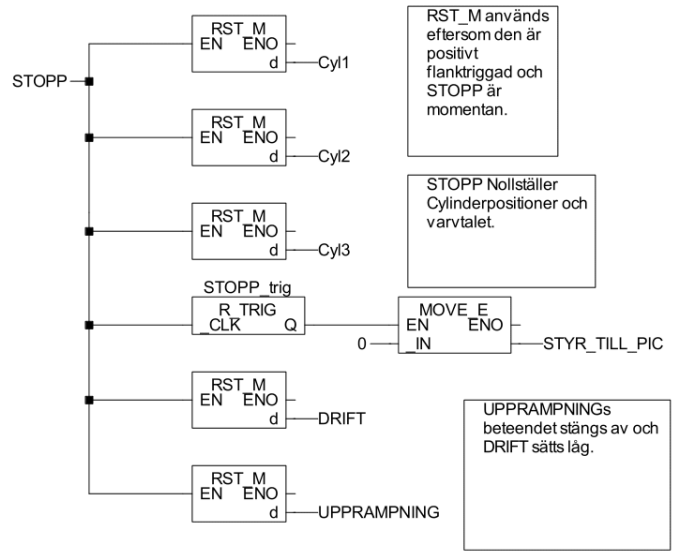
				Date			2016-05-25 13:05:30	E:\Sync\Chalmers\Styrtekni...\Prog_main
				Drawn				PIC_to_PLC [PRG] Header
				Appr.				STYRTEKNIK LET085 Page: 5
Rev	Change	Date	Name	Rel.				

Rampning [PRG] Header

	Class	Identifier	Type	Initial	Comment
0	VAR	RAMPNING_TRIG	R_TRIG		
1	VAR	Timer_1	TON		
2	VAR	Timer_2	TON		
3	VAR	Timer_4	TON		
4	VAR	Timer_3	TON		
5	VAR	STOPP_trig	R_TRIG		
6	VAR	START_trig	R_TRIG		

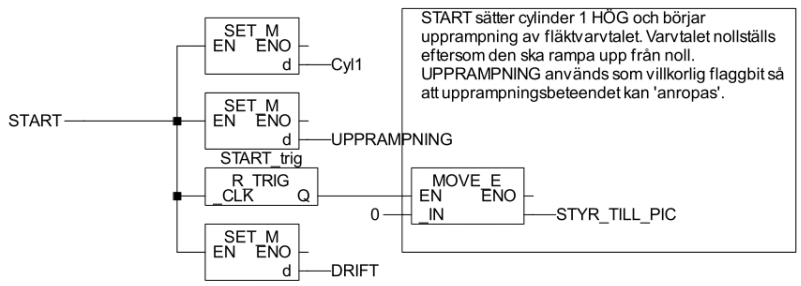
Rampning [PRG] Body [FBD] Network#1

Network #1 (1) Label: Title:



Rampning [PRG] Body [FBD] Network#2

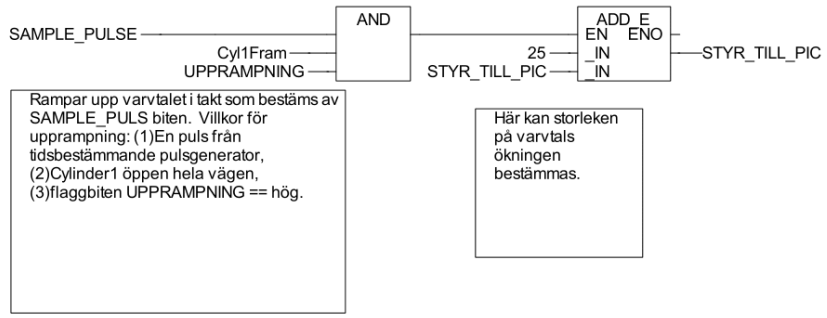
Network #2 (1) Label: Title:



				Date		2016-05-25 13:05:30	E:\Sync\Chalmers\Styrtekni...\Prog_main	
				Drawn			Rampning [PRG] Header	
				Appr.			STYRTEKNIK LET085	Page: 6
Rev	Change	Date	Name	Rel.				

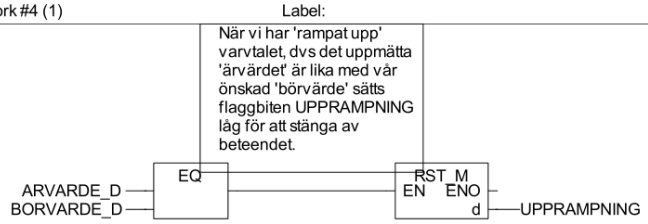
Rampning [PRG] Body [FBD] Network#3

Network #3 (1) Label: Title:



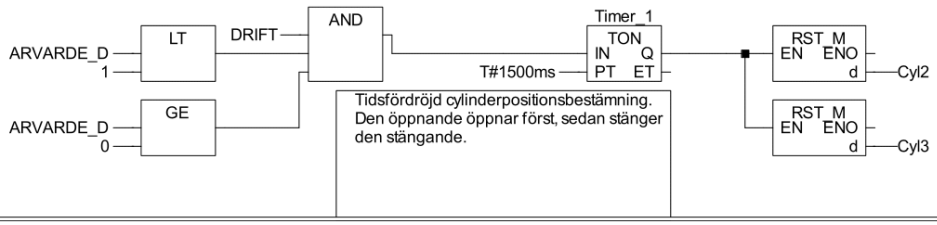
Rampning [PRG] Body [FBD] Network#4

Network #4 (1) Label: Title:



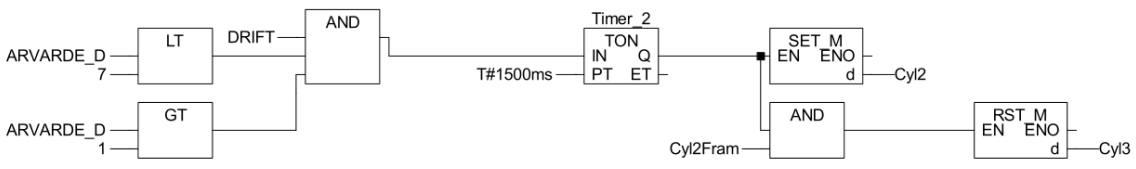
Rampning [PRG] Body [FBD] Network#5

Network #5 (1) Label: Title:



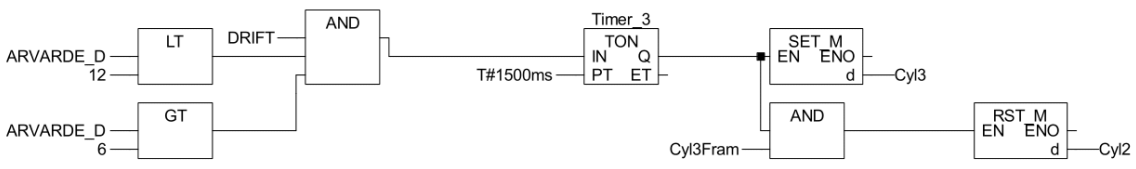
Rampning [PRG] Body [FBD] Network#6

Network #6 (1) Label: Title:



Rampning [PRG] Body [FBD] Network#7

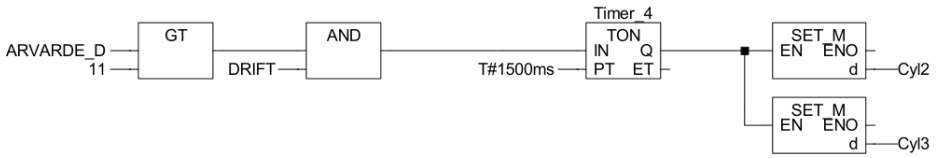
Network #7 (1) Label: Title:



				Date			2016-05-25 13:05:30	E:\Sync\Chalmers\Styrtekni...\Prog_main
				Drawn				Rampning [PRG] Body [FBD] Network#3
				Appr.				STYRTEKNIK LET085 Page: 7
Rev	Change	Date	Name	Rel.				

Rampning [PRG] Body [FBD] Network#8

Network #8 (1) Label: Title:



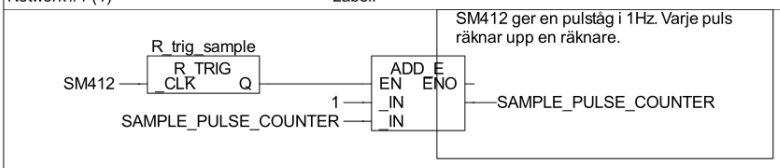
				Date			2016-05-25 13:05:30	E:\Sync\Chalmers\Styrtekni...\Prog_main
				Drawn				Rampning [PRG] Body [FBD] Network#8
				Appr.				STYRTEKNIK LET085 Page: 8
Rev	Change	Date	Name	Rel.				

Varvtalsjustera [PRG] Header

	Class	Identifier	Type	Initial	Comment
0	VAR	RS_sample	RS		
1	VAR	R_trig_sample	R_TRIG		

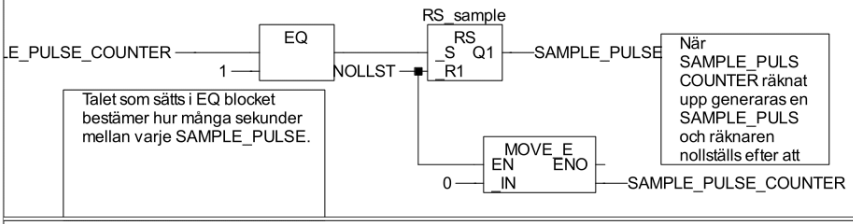
Varvtalsjustera [PRG] Body [FBD] Network#1

Network #1 (1) Label: Title:



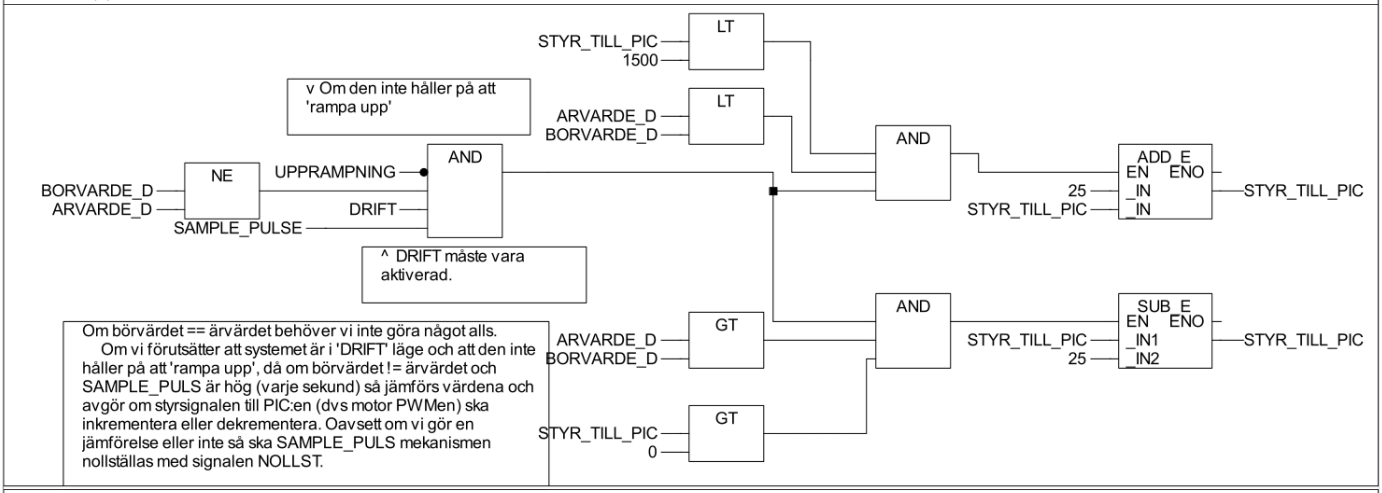
Varvtalsjustera [PRG] Body [FBD] Network#2

Network #2 (1) Label: Title:



Varvtalsjustera [PRG] Body [FBD] Network#3

Network #3 (1) Label: Title:



Varvtalsjustera [PRG] Body [FBD] Network#4

Network #4 (1) Label: Title:

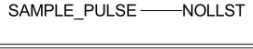
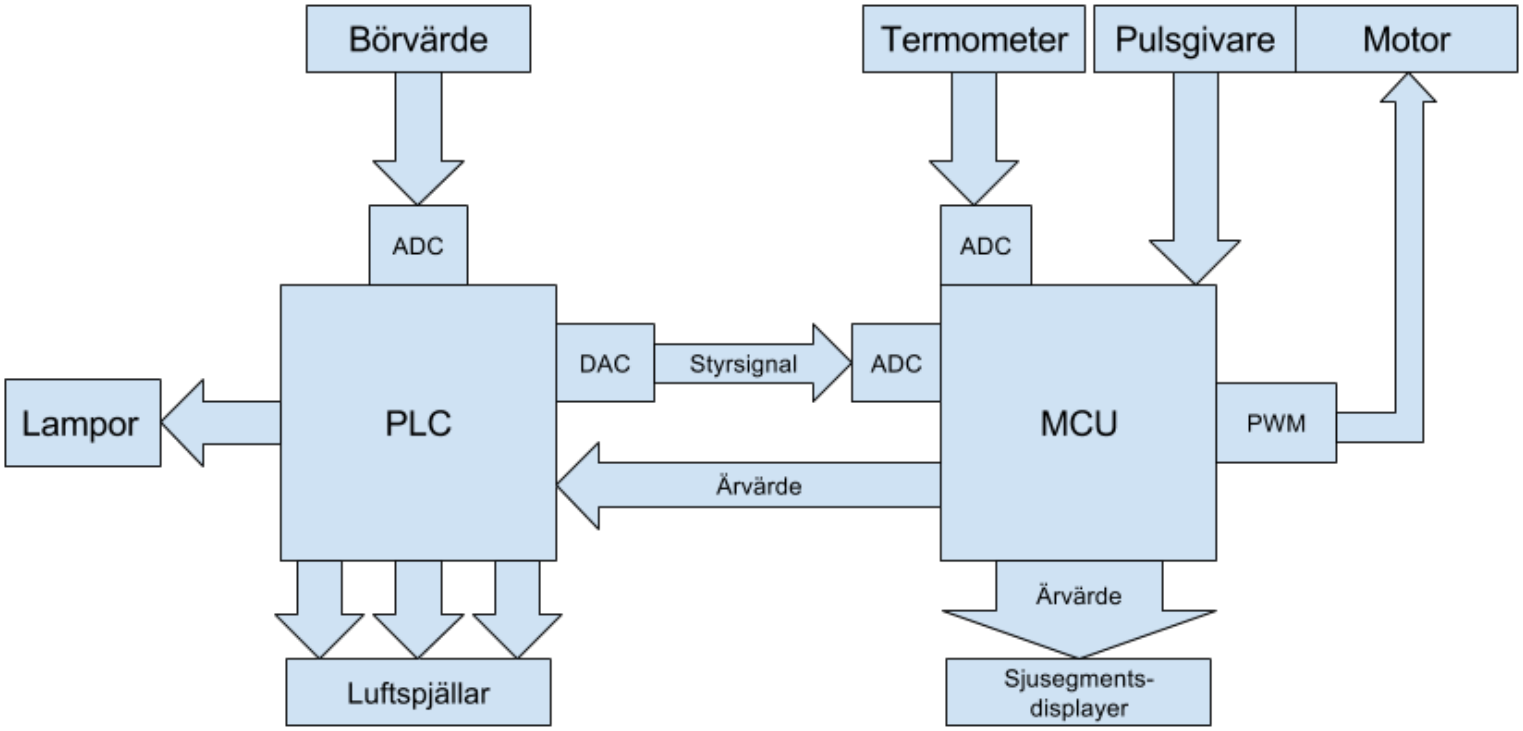
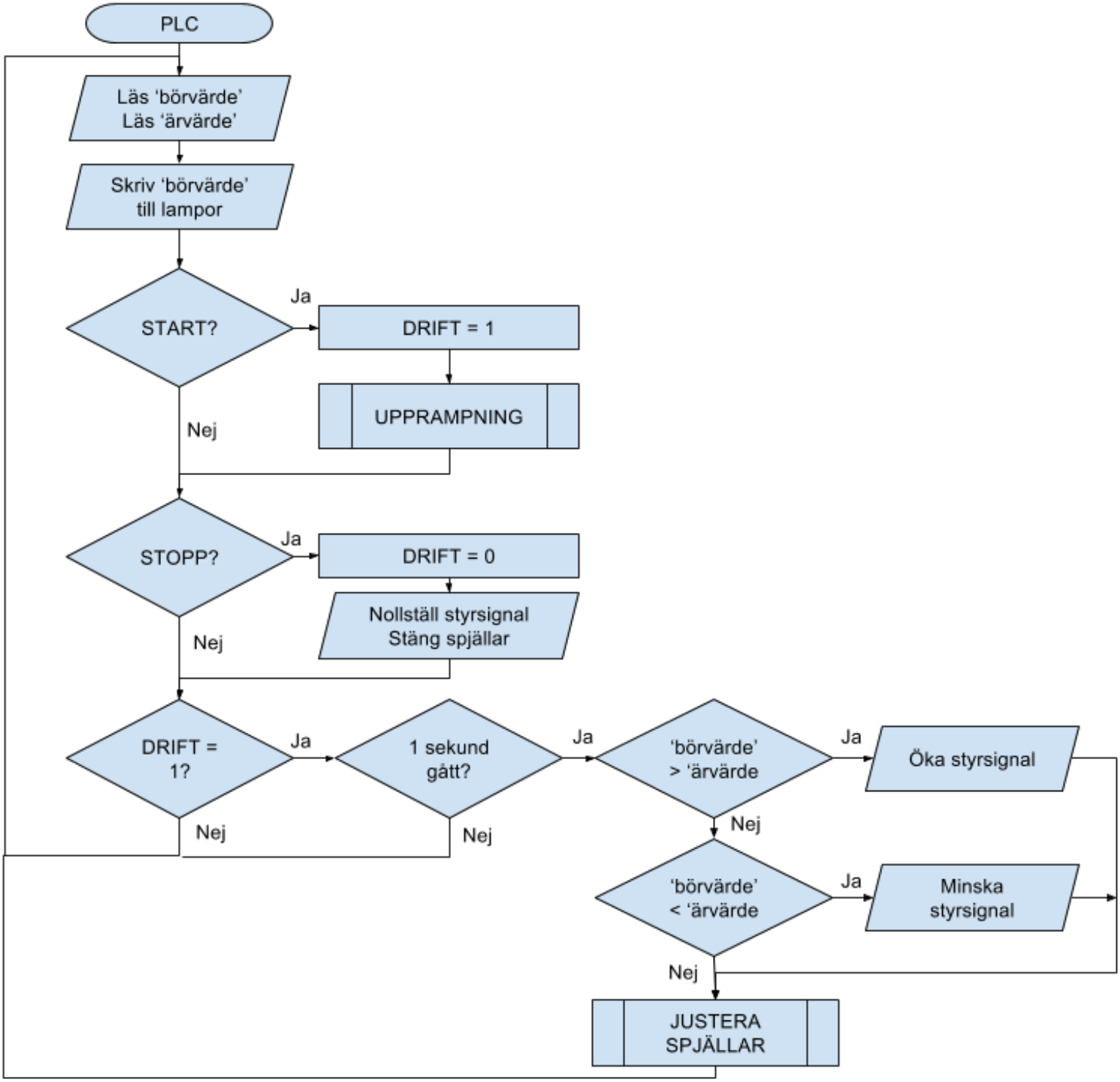


Table of Contents

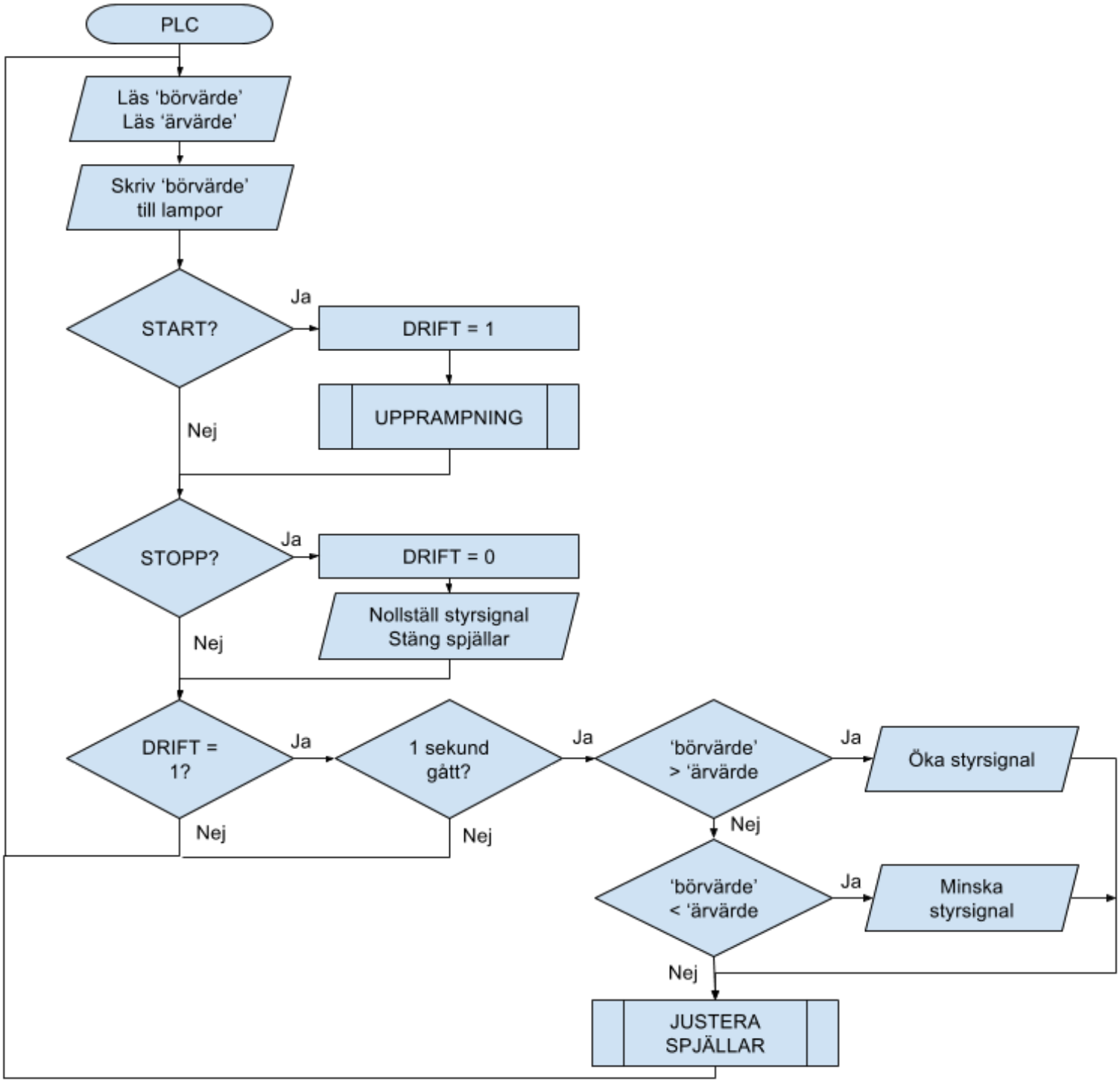
Bilaga 5 – Blockschema



Bilaga 6 – Flödesschema PLC



Bilaga 7 – Flödesschema MCU



Källförteckning.

[1] K. Erickson, "Programmable logic controllers", *IEEE Potentials*, vol. 15, no. 1, pp. 14-17, 1996.