KTH Industriell teknik
och management

Huvudmotorreglering i TCD-maskin

Författare:	Percy Villegas T
	Basel Lwai
Uppdragsgivare:	Talaris AB, Flen
Handledare:	Björn Nordin (Talaris AB)
	Lars Johansson (KTH TMT)
Godkännandedatum:	2009-03-16

KTH Industriell teknik och management
Tillämpad maskinteknik
Examensarbete inom <elektronik, robotik och mekatronik>

KTH TMT, Campus Telge

Sammanfattning

Syftet med detta examensarbete är att reglera huvudmotor i en TCD ”Teller Cash Dispenser” maskin.
Arbetet var uppdelat i olika del moment framförallt:

- Programmering av Atmega32, Usart, PWM, ADC, TIMER, INTERRRUPT behandlas i detta projekt.
- C# programmering, ett program i C# som styr motorn. Där finns Start, Stop, Hastighet och Ström knappar, och det finns några textfält som bekräftar kommunikationen (handskakningar).
- Drivsteg för att reglera en 24V DC motor.
- P-reglering och PI- reglering.

Huvuddelen av projektet gick ut på att reglera en motor som oberoende av belastning ska hålla en konstant hastighet. Den andra delen var att göra ett enkelt program ”interface” som kunde styra motorn.

Förord

Denna rapport är en del av examensarbetet som ingår i utbildningen ”Robotik, Mekatronik och Elektronik” på KTH syd, Campus Telge. Vår handledare från skolans sida var Lars Johansson. Uppdraget var genomfört för Talaris AB i utvecklingsavdelningen under perioden från 14 januari till 14 mars 2009. Handledaren i Talaris var ingenjör och Programutvecklare Björn Nordin och kontaktpersonen Ingenjör direktör Lars Svensson.

Arbetet utfördes i fyra olika faser. Den första fasen handlade mest om att förbereda/klargöra och forma en helhetsbild om vad företaget för väntade sig och vad vi kunde åstadkomma. Den andra fasen var i stort sätt informationssökning i form av fakta och c-kod och sätta grund för kod som skall utvecklas och sammanslås under fas 3. Under fas 2 inhämtades nya kunskaper i C# och reglering i C-kod. Under fas 3 som är den största och viktigaste fasen i projektet, genomfördes all arbete för att åstadkomma slutprodukten enligt uppdragets specifikation. Under 4:e och avslutande fasen skrevs rapporten och förberedelse av presentationen påbörjades.

Projektets arbete har flutit på ganska bra under den 1:a fasen. I slutet av fas 2 hade byggandet av drivsteg, och test av ADC ”Analog Digital Omvandlare” tagit längre tid än förväntat och dessutom hade de uppmätta värdena på hastigheten stor variation. Slutförandet av dessa tester ledde till att fas 3 blev förskjuten en vecka. Detta har lett till att vi under genomförandefasen nästan helt kunnat koncentrera oss på motorreglering, och att fas 4, då rapportskrivningen skulle skrivas kunde börjas tidigare än planerat. Projektets arbete har lyckats uppfylla alla uppgifter som ställts. För att kunna tillgodogöra sig rapportens innehåll behöver läsaren ha allmänna kunskaper om C-programmering och mikroprocessorer.

Acknowledgements

First of all, we would like to express our appreciation and gratefulness to our informal supervisor, Engineering Manager Lars Svenson, for giving us the their trust throughout our thesis period in Talaris AB. Our formal supervisor and Embedded SW Björn Nordin have given as code bits and many good tips that appeared to be important to the project. Without him we would be lost. We would also like to thank our supervisor at the Royal Institute of Technology “KTH”, Doktor Lars Jonsson for his good comments and support about Motor regulation. Last but not least, thanks to Talaris AB for giving us this opportunity to do our thesis.

Basel Lwai & Percy Villegas T

Mars, 2009

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Anledningen till att Talaris gick ut med detta uppdrag var att man såg möjligheten att förbättra strömregulatorns funktion och därmed sänka kostnaderna för maskinens tillverkning. Den nuvarande strömregulatorn är för stor och kostsam.

1.2 Problemdefinition

Nedan framgår de moment som i början av projektet identifierades som nödvändiga att klara av. Momenten är fyra till antalet.

1.2.1 Egenskaper

- Sensorns noggrannhet.
- Prioriteringar/ processorhastighet?
- Strömförsörjning?

1.2.2 Programmering

- Kommunikation
- Sammanfogning av delmoment/olika programmeringsdelar
- Kommunikation mellan alla enheter, Atmega32, PC osv.
- Debuggning/felsökning
- Protokollkommunikation

1.2.3 Humaninterface

- Hur skall AVR-processorn programmeras och provköras?
- Hur skall kommunikationen med PC utformas?
- Hur skall kommunikationen med kretskortet utformas?
- Hur ska PC-applikationens (i C#) datainmatning/utmatning skötas?

1.2.4 Huvudprocessorkortet

- Hur fungerar Amega32-processorn.
- Hur ska vi få reda på konstanterna i den matematiska modellen för PI reglering?
- Hur skall PWM till motorn se ut?
- Hur ser signalerna från sensorn ut och hur skall de behandlas?
- Hur skall de matematiska reglerprocesserna programmeras?
- Hur skall felhanteringen gå till?
- Hur skall hårdvaran som ska styra motorns hastighet dimensioneras/konstrueras?

1.3 Kravspecifikation

1.3.1 Hårdvara

- Lämplig hårdvara skall dimensioneras för att kunna styra motorns hastighet via PWM och utläsa driftströmmen med A/D omvandlare.
- Prototyp av hårdvara enligt ovanstående. För styrlogik används STK500 som plattform.

1.3.2 Mjukvara

- C-program i Atmega32 på STK500 som innehåller:
  Kontrollerad start (mjukstart) och stopp av motor via seriellt kommando.
  Felmeddelande vid misslyckad uppstart eller annat fel, kvittens vid lyckad start.
  Utläsning av motorns hastighet och ström via seriellt kommando/svar

  Förutsättningar:
  - Kommandon utformas enligt befintlig kommandostruktur i samråd med Talaris handledare.
  - Klockfrekvens, timerintervall väljs i samråd med Talaris handledare för kompabilitet med övrig programvara.

- En enkel PC-applikation i C# för att skicka Start/Stop/Mät kommandon till STK500 via RS232.

1.4 Mål

Målet med examensarbetet är att ta fram en ny modul för en ny TCD-controller/TCD-maskin (Teller Cash Dispenser), som driver och reglerar mekanismens huvudmotor till en lämplig hastighet.

1.5 Avgränsningar

Programkoden skall inte skrivas i andra språk än C.

Gränssnittet för presentation på PC av motorns beteende/kontroll behöver inte innehålla mer är Start, Stop samt Mät knappar.

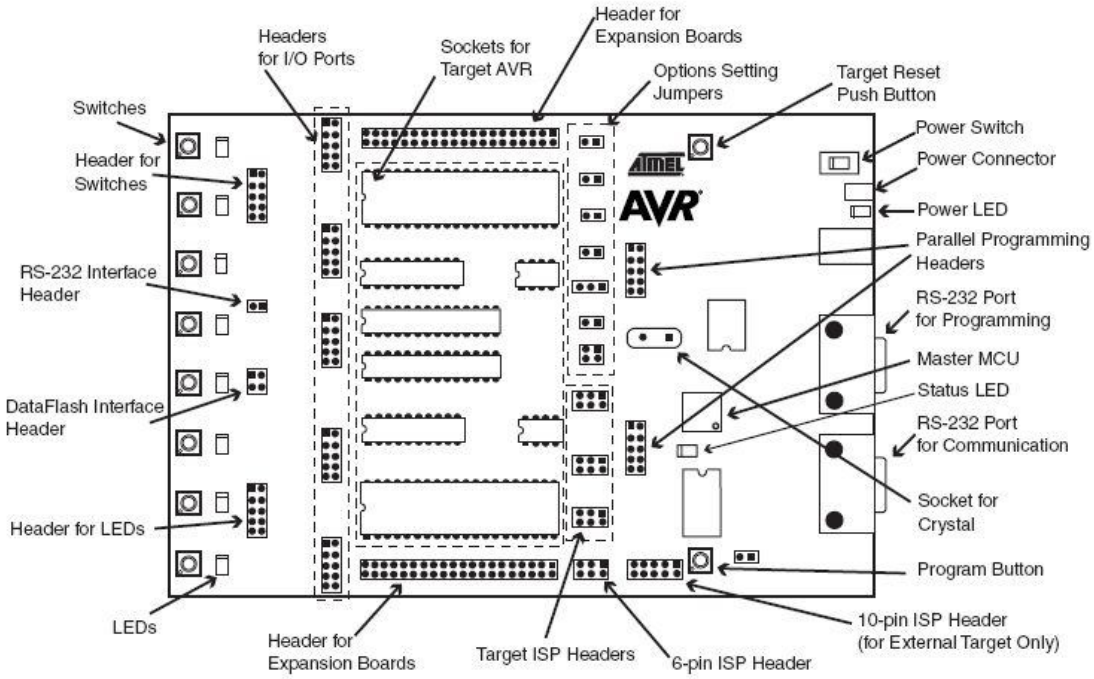
1.6 Lösningsmetod

Arbetet skall utföras i projektform enligt Campus Telges projekthandbok. Information från TCD-maskinens specifikation ligger till grund för olika områden tex kommunikation. Med hjälp av Talaris tillhandahållen hårdvara och mjukvara kommer programmering och testning ske modulvis. Resultaten kommer sedan att samköras och testas i slutprodukten.

- Påläsning om hur USART fungerar, samt implementera det i programkoden.
- Föra diskussion med handledaren om hur signalerna från sensorn ser ut och utforma funktioner som kan behandla dessa.
- Ta reda på hur den matematiska regleringen skall fungera genom att simulera sensorvärden och analysera motorns reaktion.
- För programmeringen av kommunikation och kontroll av motorn används ett grafiskt programmeringsspråk, C#.

2 Huvudprocessorkort

STK500 kretskortet är ett laborationskort för mikrocontrollers av typen Atmel AVR. Figuren visar vart de ingående och utgående signalerna skall kopplas samt en kortfattad beskrivning av varje ports ändamål **[7]**.

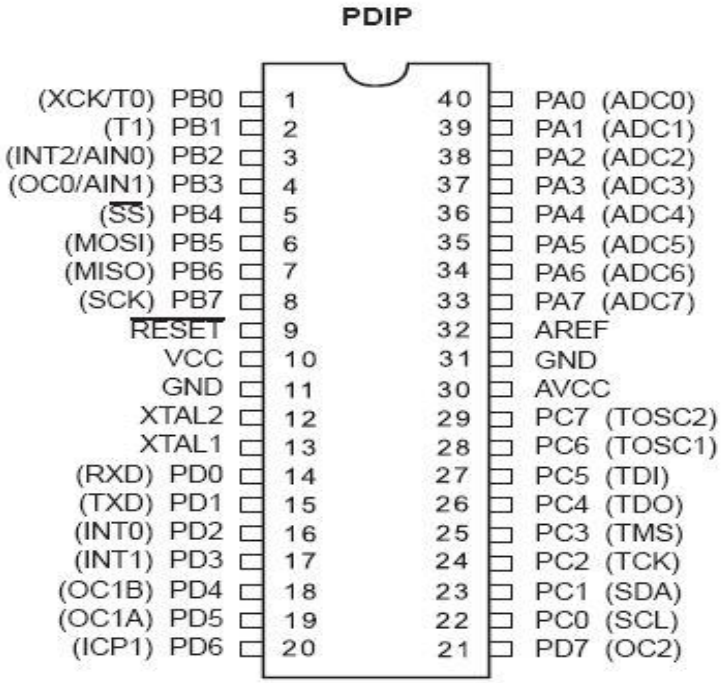


För mer information hänvisas till databladet för STK500**[7]**.

3 Mikroprocessor

Atmega32 är en lämplig mikroprocessor för uppdraget då den uppfyllde de kriterier som fanns, bl.a. beträffande antal in- och utgångar, Timer, ADC och USART etc.

Atmega32 är en 8-bitars AVR-processor som har 32 I/O ben och kan köras i upp till 16 MHz.



För mer information om Atmega32 hänvisas till databladet **[6]**.

3.1 AVR programmering

Processorn programmerades i C. Koden utvecklades och kompilerades i CodeVision AVR och simulerades i AVR Studio.

3.2 Timer

Det finns tre timers i Atmega32 Timer0, Timer1 och Timer2 **[5]**. Timer0 och Timer2 har 8 bitar och Timer1 har 16 bitar. Att timern har åtta bitar innebör att det kan räkna till 255 och sedan börjar från noll och sexton bitar att den kan räkna till 65535 och sedan börjar från noll.

Kristallen som används är på 14745600 Hz. Den är grundklockan för mikroprocessorn. Denna klocka har valts på begäran av Taralis framför allt för att uppnå kompatibilitet till andra delar av maskinen.

CodeVision AVR möjliggör inställning av timern enligt önskemål.

Timer0 används för att ge ett fast PWM (Pulse Width Modulation), Timer1 för att kunna ta reda på tiden som har passerat mellan varje flank som ges av sensor och Timer2 för att läsa ADC.

3.2.1 Timer0

Timer0 ger fast PWM på 7228 Hz, detta genom att först dividera grundklockans frekvens med åtta: 14745600/ 8 = 1843200 och därefter divideras detta värde med 255: 1843200/255= 7228,23. Denna fasta frekvens bestämdes genom samtal med Talaris handledare, då de använde en tre gånger större frekvens på den äldre modellen av maskinen och för att transistorn och AVR Timer har sina begränsningar.

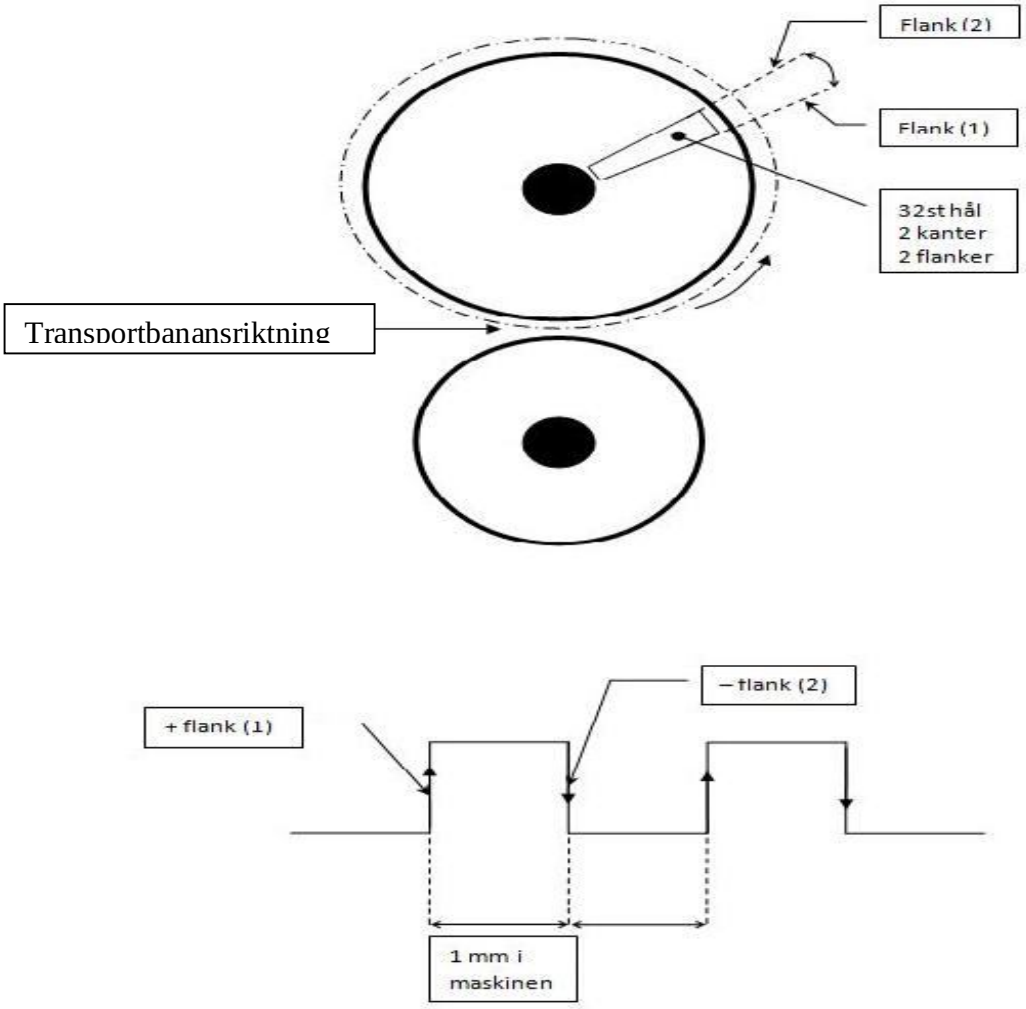
Dutycycle har ett intervall mellan 0 – 255. Man kan ändra på dutycycle genom att sätta olika värde på 0CR0, t.ex. 0CR0 = 100 ger en ducycycle på 0,39 (100 / 255) eller 39%. Se **4.5** som handlar om PWM.

För mera information om timerinställningar och Prescaler se tabellen för Timer Counter0 i Appendix A.

3.2.2 Timer1

Vi läser hastighet genom att läsa en sensor som ger två lägen: på och av. När den går på (5V) är det positiv flank och när den går av är det negativ flank (0V). Sensor är en optisk läsgaffel med schmittrigger utgång.

Genom att mätta den tid som går mellan varje flank kan vi sätta ihop två tider och mätta den tid som går mellan varje positiv flank. Sträckan mellan den positiva och den negativa flanken är lika stora 1mm. Se också den grafiska förklaringen på bilden nedan.



Överst på bilden visas hjulet med 64 flanker (omkrets = 64mm).
Nederst på bilden visas avståndet mellan varje flank och avståndet på två positiva flanker motsvarar 2mm, vilket används för hastighetsmätningen.

3.2.3 Timer2

Timer2 triggar avläsning av ADC:n (Analog Digital Converter). Med hjälp av Timer2 läses medelvärdet på spänningen som motsvarar Motorströmmens medelvärde. Om man försöker bromsar motorn ökar strömmen och därmed spänningen. Om systemen bromsas så ska motorn uppnå den önskade hastigheten på mindre än en sekund. Lyckas inte detta stängs motorn av för att begränsa strömmen.

För att kunna klara av detta läser mikroprocessorn kontinuerligt 451 ggr/s via ADC1. För att få det värdet ska grundklockans frekvens delas med 128 ”Prescaler”, som ytterligare ska delas med 255, 14745600/128 = 115200 : 115200/255 = 451.

För mer information om timerinställningar och Prescaler **[5]**, se tabellen för Timer Counter 2 i Appendix A.

3.3 Usart

Usart är den del som skickar eller tar emot information från PC **[5]**. Till exempel skickar vi information om att starta/stänga motor från PC och sänder information om hastighet och ström till PC.

Sända och motta kan man göra i 8 eller 16 bitar. Í detta projekt sänds och mottages informationen i 8 bitar med hjälp av funktioner Getchar(), som ”tar emot” och putchar() (Sänder).

Här ett exempel på hur informationen om att starta motor tas emot och kvitteras genom att sända tillbaka ett meddelande.

Nedan skrivna kodbit körs alltid i mikroprocessorn (dvs. som bakgrundsprocess):

```
while(1){

    sendBuffert[0] =  getchar();  //här tas första byten emot från PC:n
    sendBuffert[1] =  getchar (); //här tas andra byten emot från PC:n

   StartMotor();    // Gå till  StartMotor() och se om meddelandet motsvarar start
   MotorSpeed();  // Gå  till MotorSpeed() och se om meddelandet motsvarar hastighet
   StopMotor();    // Gå till  StopMotor()och se om meddelandet motsvarar stopp
};
```

Här är StartMotor() funktionen, som sätter sendBuffert[] med 0OK om hastigheten överstiger 226 eller FERROR om hastigheten är mindre än 226.

```
void StartMotor(void ){

  if ((sendBuffert[0]=='M')&&(sendBuffert[1]=='1')) { //  Om PC:n sänder “M1” starta

   OCR0=70;          // Här sätts DutyCycle till 0,27 (70/255= 0,27). Motor startar
   if (velocity > 226){
      sendBuffert[0]= '0';
      sendBuffert[1]= 'O';
      sendBuffert[2]= 'K' ;
      sendBuffert[3]= '\n';
      SendAnswer();
  }
  else{
```

```
        sendBuffert[0]= 'F';
        sendBuffert[1]= 'E';
        sendBuffert[2]= 'R';
        sendBuffert[3]= 'R';
        sendBuffert[4]= 'O';
        sendBuffert[5]= 'R';
        sendBuffert[6]= '\n';
        SendAnswer() ;     // Här kallas SendAnswer(), som skickar information till PC:n
    }
  }
 }
```

Följande funktion sänder buffert sendBuffert[] till PC och sätter alla sina index till "\n"

```
void SendAnswer(void){
     buffertIndex =0;

   while (sendBuffert[ buffertIndex] != '\n')
   {
      putchar(sendBuffert[buffertIndex]); //här sänds svar till PC
     sendBuffert[buffertIndex]='\n';          // Alla index till "\n"
      buffertIndex++;
   }
   }
```

På motsvarande sätt fungerar stoppknappen och all annan kommunikation mellan PC och Usart.

Kommunikationsparametrar är 8 databitar, 1 stoppbit och ingen paritet.
Dessa bitar sätts i registret UCSRC och sändnings hastighet är 9600 bitar per sekund och Baudrate "bitar per sekund" sätts i register UUBRRH och UBRRL.

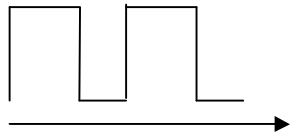
3.4 ADC (Analog Digital Converter)

Här omvandlas analog signal till digital, det är 8 bitar på varje inläsning från ADC1. För att starta AD omvandlingen sätter man ADCSRA|=0x40 och det är också viktig att vänta tills själva omvandlingen är klar. Detta görs genom en while loop, while ((ADCSRA & 0x10)==0). All dessa bitar sätts automatisk av CodeVisionAVR men tänk på att kontrollera databladet, så att koden åstadkommer det du vill.
ADC inläsning används för att strömmen inte ska bli alltför stor och att belastningen inte ska vara alltför påfrestande. Enkel uttryckt används ADC i projektet för att hålla koll på den ström som passerar genom motorn.

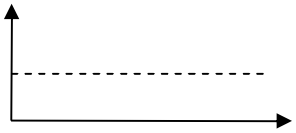
Signalen som läses in i ADC1 passerar genom ett filter och en OP-förstärkare. Filtret eliminerar de höga frekvenserna och gör signalen mer stabil. Förstärkaren svarar för att signalen förstärks och att inströmmen till mikroprocessorn hålls begränsad, mera om detta i Motorslutsteg/Drivsteg **5.**

3.5 PWM (Pulse Width Modulation)

PWM är en metod att styra inkopplade enheter. Dessa enheter kan vara tröga komponenter som t.ex. motorer. En pulsbreddsmodulerad signal kan bara anta en nolla eller etta. Där en etta är en signal för att enheten ska starta och en nolla för att enheten ska stanna. Genom att modifiera pulsens duty cycle alltså hur länge signalen är hög eller låg under en period får man en hastighet mellan 0-100%.



Pulsbreddsmodulering



Viktat medelvärde

PWM används för att kunna styra motorn. I detta projekt betyder 255 full fart, 127 halv fart och 0 stopp. Ökad PWM betyder ökad fart på motorn. Det är Timer0 som levererar fast PWM-signal i vårt projekt, det innebör att frekvensen 78228 Hz inte ändras.

Det är Duty Cycle i PWM vi reglerar för att kunna åstadkomma önskad hastighet. OCR0 är ett register som anger PWM med åtta bitar, så ledes blir maxvärdet 255.

4 Motorslutsteg/drivsteg

En lämplig prototyp av en styrkrets dimensionerades för att kunna styra motorns hastighet via PWM och utläsa driftsströmmen med A/D omvandlaren.
Valet av effekttransistor har grundats på hur mycket den ska tåla innan den går sönder (bränns) vid maxström och vilken stigtid och falltid den har.

Resistor R1 på bilden nedan används för att kunna räkna fram spänning. Genom att göra två mättningar på den äldre maskinen, dels när maskinen är normal

belastad och dels när den är bromsad, så kan ett lagom värde på R1 räknas fram.
Genom effektformeln: P = I * U och ohms lag: U = R * I beräknas effektmotståndet (resistor R1 i schemat på bilden nedan)

Beräkningar på den gamla maskinen med 40V spänning:

Gamla Maskinen bromsad:

8A * 40V = 320W

Gamla Maskinen normalbelastad:

2A * 40V = 80W

Anpassade beräkningar för den nya maskinen med 24V spänning:

Strömberäkningen ger:

320W / 24V = 13,3A

80W / 24V = 3,3A

Spänningsberäkning ger:

13,3A * 0,1Ω = 1,33V

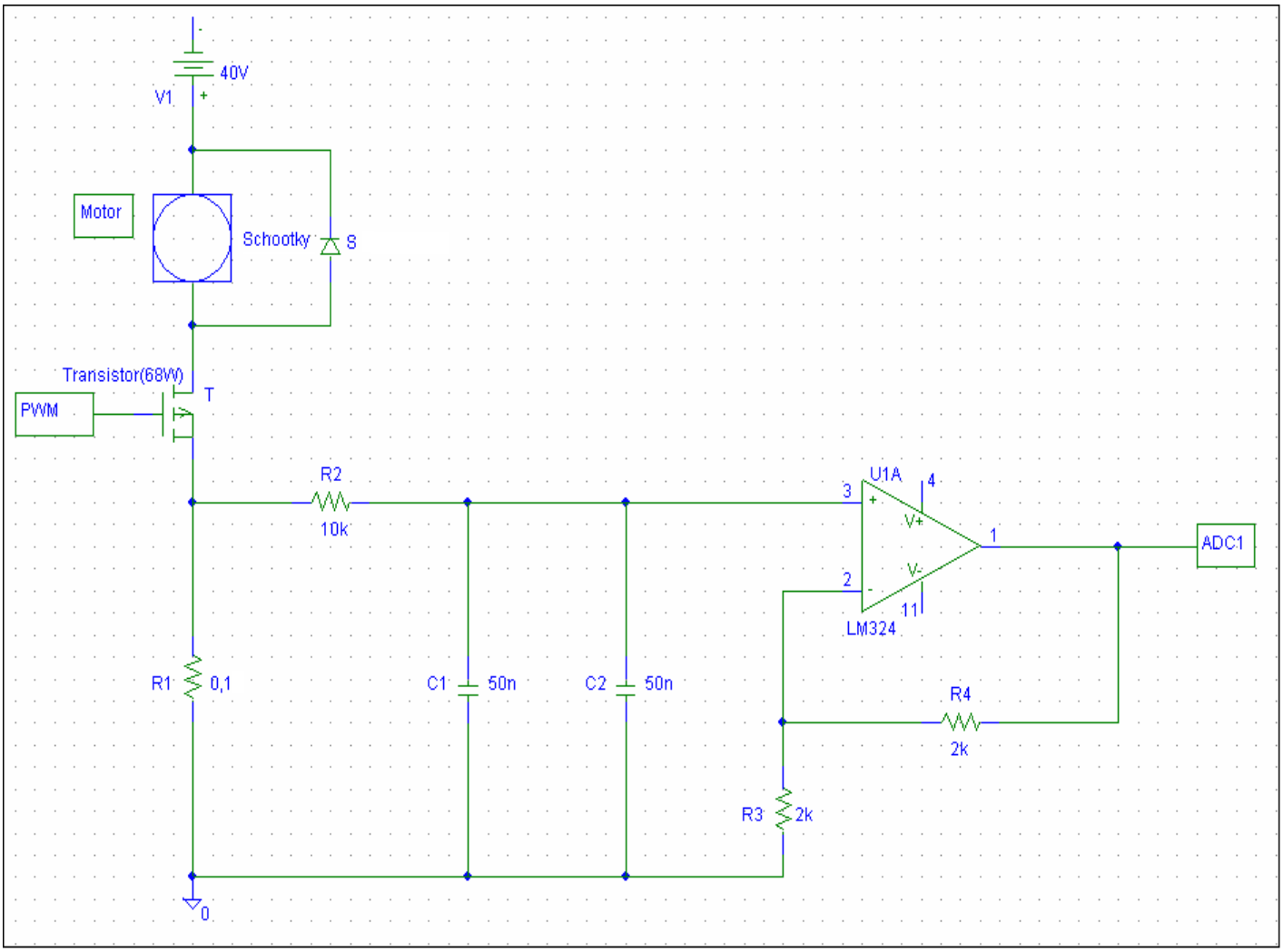
3,3A * 0,1Ω = 0,33V

Effektmotståndet R1 valdes så att spänningen inte överstiger 5 V efter två gångers förstärkning, för max spänning in i mikroprocessor är 5V.

Effektberäkningen ger:

Största Effekt	13,3A * 1,33V = 17,689W
Minsta Effekt	3,3 A * 0,33V = 1,089W

R1 tål en effekt på 38 W, alltså klarar komponenten effektkraven.

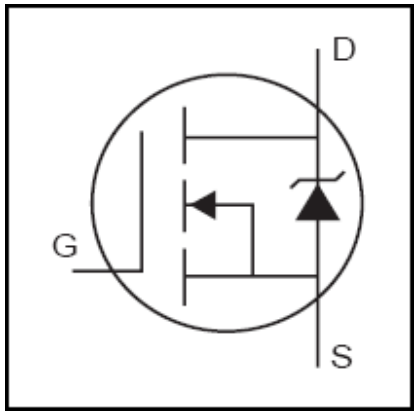


Bilden ovan visar drivsteget med komponenter som presenteras i tabellen nedan

Nedan listas de komponenter som ingår i styrkretsen:

Modellkod	Betäckning	Antal	Hårdvara	Info
IRLR2705	T	1	Effekttransistor	P_{tot} = 68W
MP930	R1	1	Effektmotstånd	P_{tot} = 38W, R = 0,1 Ω
LM324 DIP14	LM324	1	OP-förstärkare	Matningsspänning: ±1,5 – ±16 V, Strömförbrukning: ≤3 mA
MBR760	S	1	Schottkydiod	I_f = 7,5A, V_{rrm} = 60V
	R2	1	Motstånd	R = 10kΩ
	R3, R4	2	Motstånd	R = 2kΩ
	C1, C2	2	Kondensator	C = 50nF

4.1 Effekttransistorn IRLR2705



En viktig komponent i framtagningen av drivsteget är transistorn. När spänningen på G (PWM) är 0V så är transistorn öppen vilket betyder att det inte går någon ström genom transistorn. Däremot när spänningen på G är 5V, så går strömmen vidare till jord genom S (Jord), så att strömmen flyter genom transistorn vilket i sin tur sätter igång motorn.

Ett krav var att transistorn skulle klara av en frekvens på ca 10 KHz. Genom att se på tillhörande datablad ser man att stigtiden är 100ns (tr) och falltiden är 29ns (tf) för denna transistor. Beräkningen nedan visar att kraven kan uppfyllas med denna transistor:

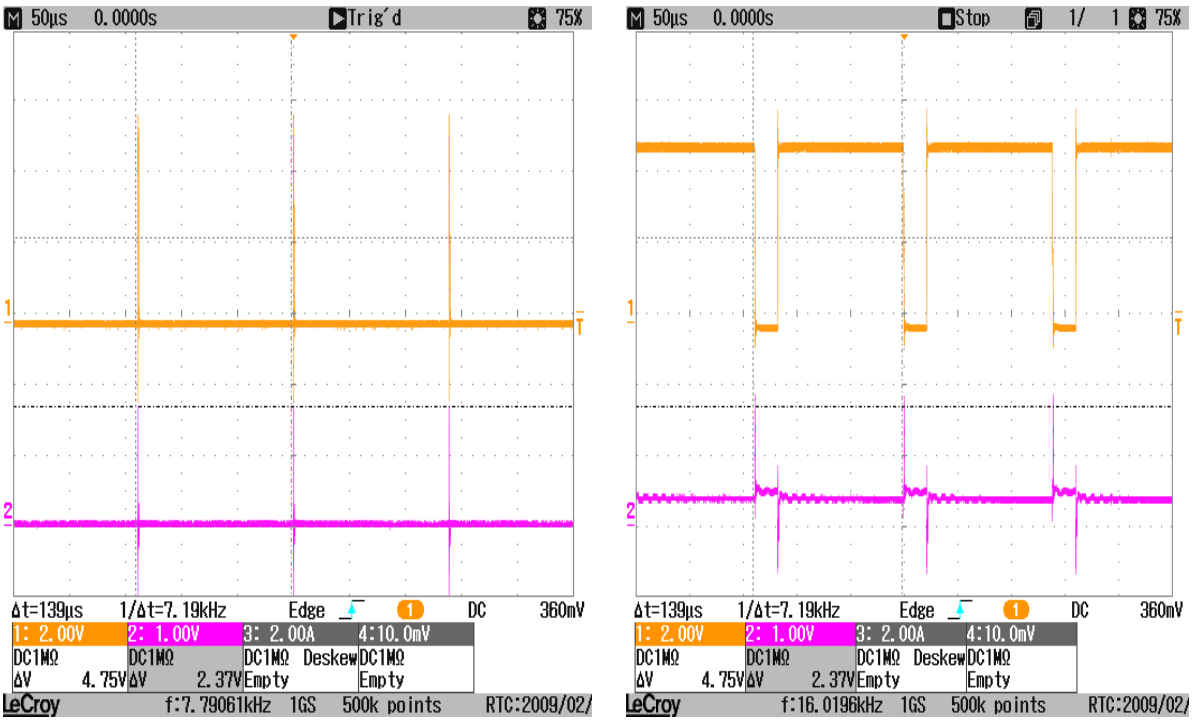
Med hjälp av den allmänna formeln f = 1/t kan man göra utvärderingen:

Periodtiden 1/10000 = 0,0001 s
Stigtiden 0,0001/0,0000001 = 1000 ggr mindre än periodtiden
Falltiden 0,0001/0,000000029 = 3448,27 ggr mindre än periodtiden

Genom ett enkelt resonemang konstateras att stigtid och falltid utan problem ryms inom använd periodtid.

4.2 Strömmätning

Innan spänningen går in till mikroprocessorn så passerar den filtret och förstärkaren i nämnd ordning (se schemat om drivsteget i samma kapitel ovan ” motorslutsteg/drivsteg” för att tydligt förstå strömmätningen). Spänningen ska motsvara den motorströmmen. När strömmen når sin toppvärde slås maskinen av.

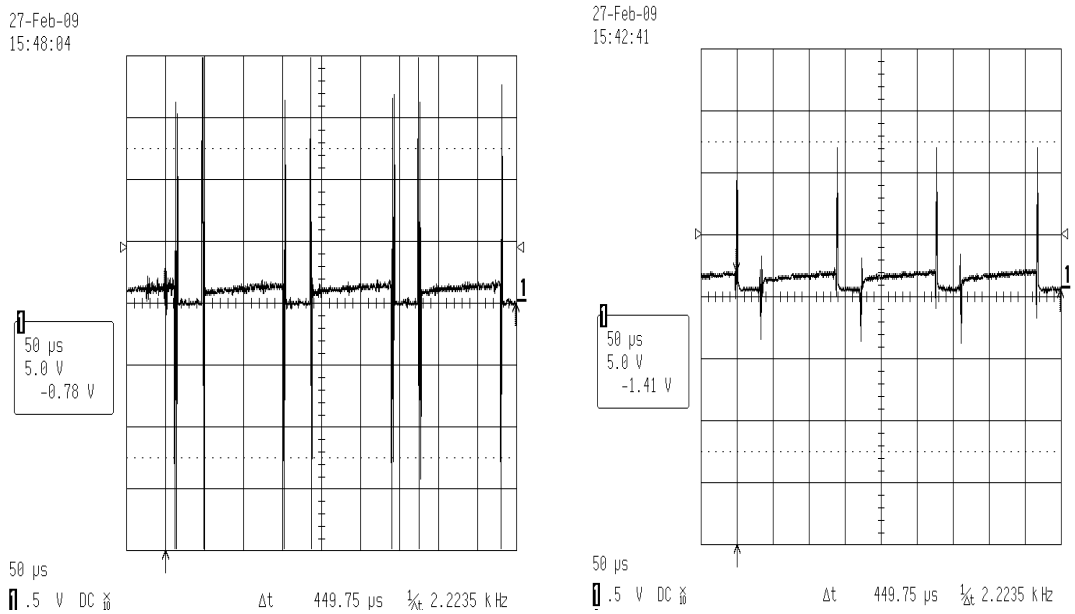


Översta signalen PWM
Understa signalen Ström

Bilden till vänster i ovanstående figur visar maskinens tillstånd när den är avslagen och bilden till höger visar maskinens tillstånd när den är påslagen.

4.3 Filtrering

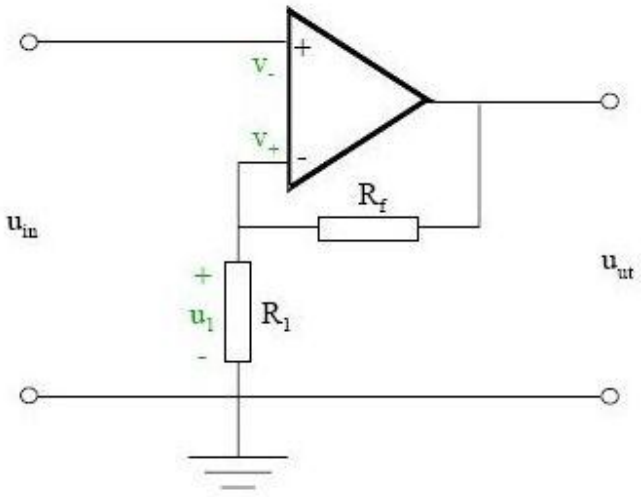
En lämplig RC-länk av typen lågpassfilter har lagts till för att filtrera störningarna innan signalen når förstärkaren. Det som utmärker denna typ av filter är att den dämpar svängningar med hög frekvens, medan svängningar med låg frekvens och andra långsamma förändringar tillåts passera i stort sett oförändrade dvs. släpper igenom signaler med frekvens som ligger under ett bestämt värde och kraftigt dämpar övriga signaler.



Bilden till vänster i ovanstående figur presenterar mättningen av spänningen innan filtret och bilden till höger presenterar signalen sedan den har passerat filtret. Efter filtret går signalen in i förstärkaren.

4.4 Operationsförstärkare

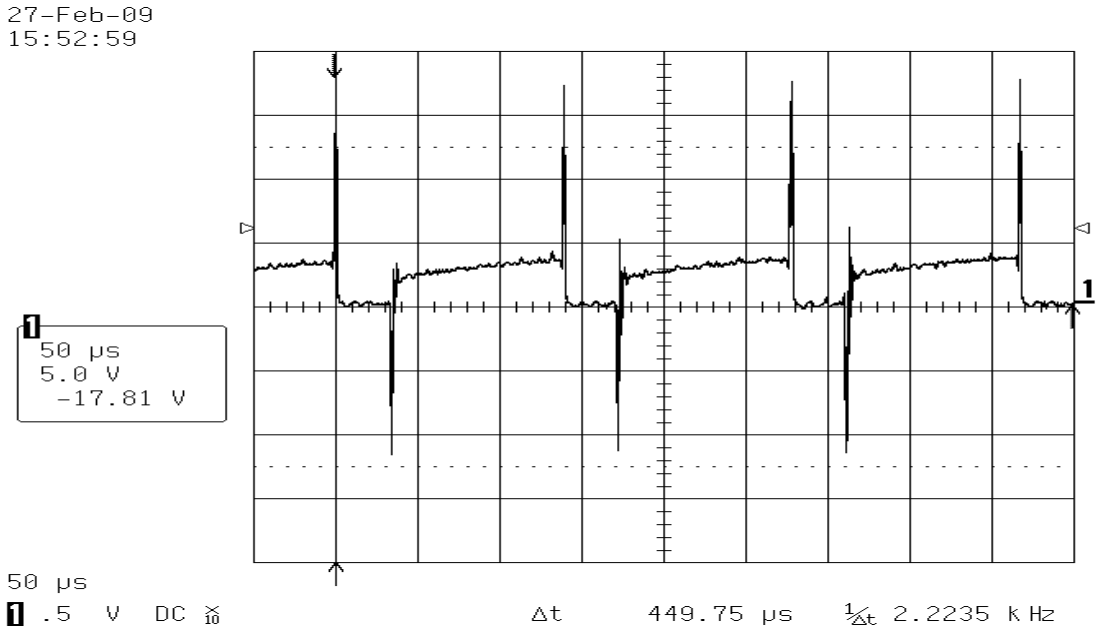
Operationsförstärkaren LM324 består av fyra oberoende operationsförstärkare med hög förstärkning. Den kan arbeta från enkel eller dubbel matningsspänning. Förutom att förstärkaren förstärker spänningen begränsar den alla spänningsfall över 5 V. På bilden nedan ses en Icke inverterande förstärkare [2].



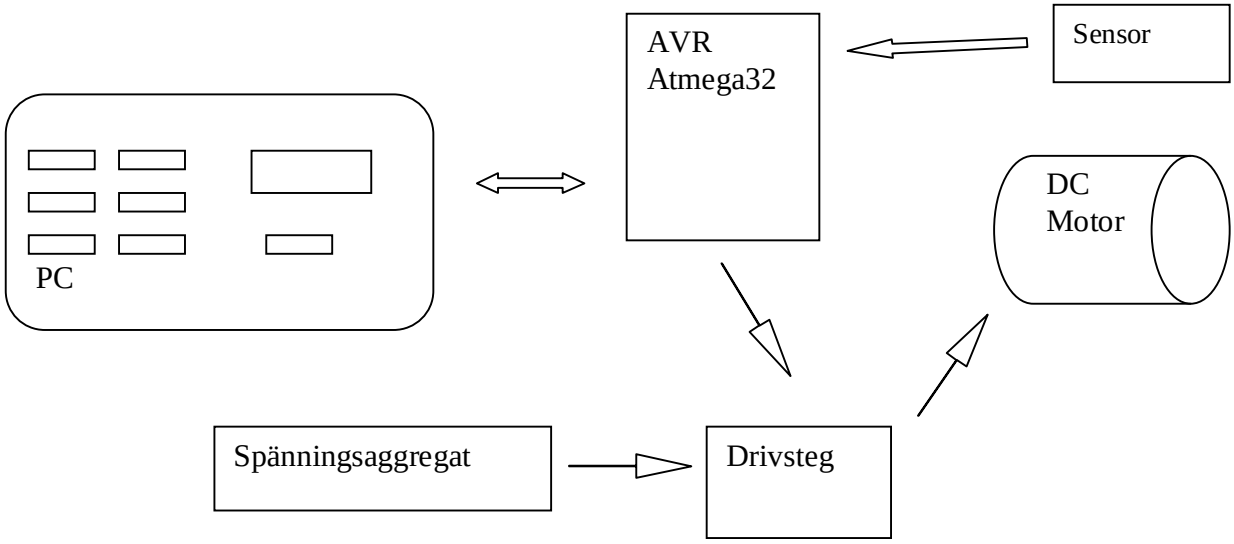
Spänningsdelning: $U_1 = \frac{R_1}{R_1 + R_f} U_{ut}$

$V_- = V_+$ blir $U_{in} = U_1$: $A_v = \frac{R_1 + R_f}{R_1} = 1 + \frac{R_f}{R_1}$

Bilden nedan visar spänningsignalen som efter filtrering och förstärkning är klar att gå in i mikroprocessorn via ADC1.



5 Kommunikation mellan enheterna



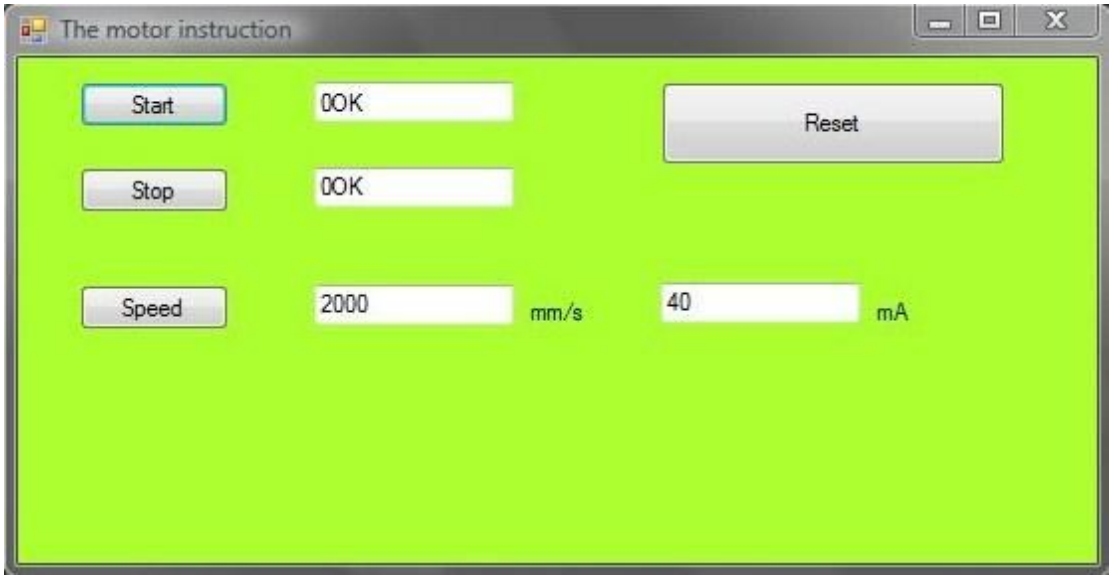
Ovan presenteras en helhetsbild av hårdvaroschema

6 Gränssnitt för enhetsstyrning ”C#”

Programmet som tar emot och sänder information till mikroprocessorn är skrivet i C#. Utvecklingsmiljön är en gratisversion av Microsoft Visual C#. Nedan presenteras den del av koden där man ställer in Comport, BaudRate (bit per sekund), DataBits (bitar där man sänder informationen), parity och stopBits [4].

```
serialPort1.PortName = "COM1";          //  val av ComPort
serialPort1.BaudRate = 9600;            // BaudRate ”Bitar per sekund”
serialPort1.DataBits = 8;
serialPort1.Parity = Parity.None;       //Paritet
serialPort1.StopBits = StopBits.One;    //Stopbitar
```

Det är viktigt att Usart i mikroprocessorn har samma format när man sänder eller mottar information dvs. att Usart och serialPort1 i detta fall ska ha samma BaudRate, DataBits, Parity och StopBits.



Ovan ses det slutliga gränssnittet för styrning av motorn

När man trycker på en knapp, låt oss säga stoppknappen i panelen, så genererar det en händelse som möjliggör tillträde till funktionen ”Button2_Click”enligt följande programkod:

```
 private void button2_Click(object sender, EventArgs e)
   {

    //konfiguration av serieport
   serialPort1.Close();
   textBox1.Text = "";         // ta bort gamla värdet från textBox1
   serialPort1.Open();

   serialPort1.Write("M0");    // ”M0” är stopp i AVR
   BackColor = System.Drawing.Color.YellowGreen; //Det blir mörkgrönt
   TexboxChoose = "12";        // “12”  är stop i C#
}
```

Efter att knappen Stopp i panelen trycks in och ”M0” sänts förväntas ett svar från AVR mikroprocessor. Eftersom programmet i C# nu förväntar sig ett svar till Stoppknappen är det möjligt att ha en if-sats som verifierar att stoppknappen tryckts in.

```
else if (TexboxChoose == "12")     //Ja om stopp har tryckts
       {
      textBox1.Text = data.Trim(); //Sätt svaret från AVR i textBox1
      TexboxChoose = "0";

       }
```

Programmet i C# kan hela tiden ta emot information från mikroprocessorn, vilket äger rum framförallt för att ta emot varningar om att motorströmmen är för stor. Detta innebör att en varning presenteras i ett meddelandefönster om systemet ”motor” inte kan komma till den önskade hastigheten inom en sekund.

Programmet i C# innehåller en timer för att skicka kontinuerlig informationen till mikroprocessorn. Timer skickar information varje tredje sekund och samtidigt som den skickar informationen börjar den räkna till en sekund innan den kallar på en händelse och får den inte svar så vissas ett meddelande om detta på skärmen och hela skärmen blir blå. Om det får svar under den sekunden så forsätter programmet sända information var tredje sekund.

6.1 Kommunikation

Information skickas genom en enkel gjord kabel som består av tre små kablar: en för jord, en för att sänd och en för att motta information. Denna kabel är mera känd som RS232. Informationen går mellan PC och Mikroprocessorn via denna kabel. Det förtjänar att nämnas att inställningar måste göras noggrant så att den korrekta informationen kommer fram[4].
Här är parametrarna som används i C#:

```
serialPort1.PortName = "COM1";           // Skriv ComPort
serialPort1.BaudRate = 9600;             // BaudRate ”Bitar per sekund”
serialPort1.DataBits = 8;                // Databitar
serialPort1.Parity = Parity.None;        //Paritet
serialPort1.StopBits = StopBits.One;    //StoppBitar
```

Här är inställningar på parametrar i USART i mikroprocessorn:

```
//Kommunikations parametrana: 8 Data, 1 Stop, No Parity
// USART Taremot: On
// USART Sänder On
// USART Mode: Asynchronous
// USART Baud Rate: 9600
UCSRA=0x00;
UCSRB=0xD8;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x5F;
```

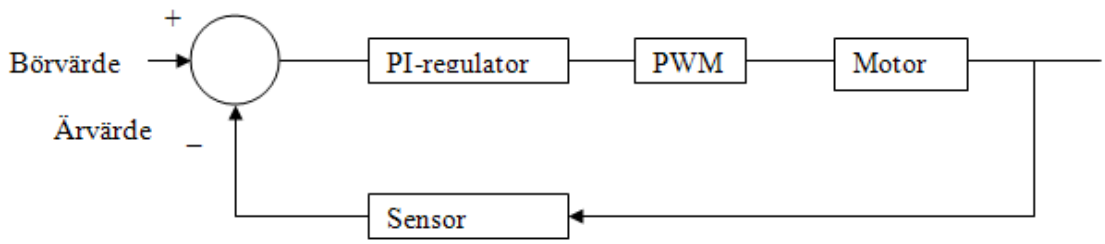
Det finns en timer i C# som skickar information varje tredje sekund till mikroprocessorn för att verifiera att de har kontakt med varandra. Om Usart inte svarar på detta meddelande stoppas mikroprocessorn och gränssnittet visar felmeddelande.

7 Reglering

Avsikten med regleringen är att motor ska hålla önskad hastighet oberoende av belastning. Det finns två olika TCD-maskiner: en med tre kasetter och en med sex kasetter. Det innebär att påfrestningen på motorn blir större i den stora maskinen. För att kunna komma till den önskade hastigheten, som är 2m/s, måste man reglera. Till att börja med prövades enkel P-reglering som sedan utvecklades till PI-reglering som, om det hade behövts, kunde ha utökats till PID-reglering. Detta möjligtgör en bred användning av denna motor och dessutom går det bra att ställa önskad hastighet genom ändring av börvärdet.

Man använder en regulator för att kunna få ett system som medger kontroll av både in- och utparametrar. Tyngdpunkten i uppdraget låg förmodligen i att modellera den process som skulle regleras. Det var svårt att beräkna parametrarna. Det hade nog varit lättare att prova ut dem.

Nedanstående bild visar reglersystemet:



7.1 P-reglering

En if–else sats användes för P reglering den första med stort K och den andra med mindre K för att undvika ryckig gång. Det stora K gör den grova regleringen: när felet är stort tar den över för att snabbt reducera felet. Det mindre K värdet gör det förfinande jobbet och möjligtgör en jämn och konstant hastighet. Denna kombination av if –else sats har visat sig ge en tillfredställande reglering.
Efter varje varv så kontrolleras om felet är nära noll, eller om PWM måste ändras. Felet i varje varv är ett genomsnitt av alla fel under varvet. Varför det görs på detta sätt beror framförallt på att felet mellan varje positiv flank är för varierande. Det visade sig att variationen kunde ligga på en intervall av hundra värden, detta framförallt på grund av systemets eller enhetens mekanism.

En allmän formel för P-regulatorn är:

$u = K(ysp-y) + ub$

Eftersom $e = ysp - y$ kan formeln skrivas om till:

$u = K\,e + ub$

u = styrsignal
K = P-konstant
ysp = börvärdet
y = ärvärdet
ub = offset för att styrsignalen ska hålla en viss nivå då reglerfelet är noll.
e = reglerfelet

7.2 PI-reglering

Vi gick vidare med PI-reglering, där värdet på styrsignalen ”PWM” också bygger på de förflutna värdena [3][1].

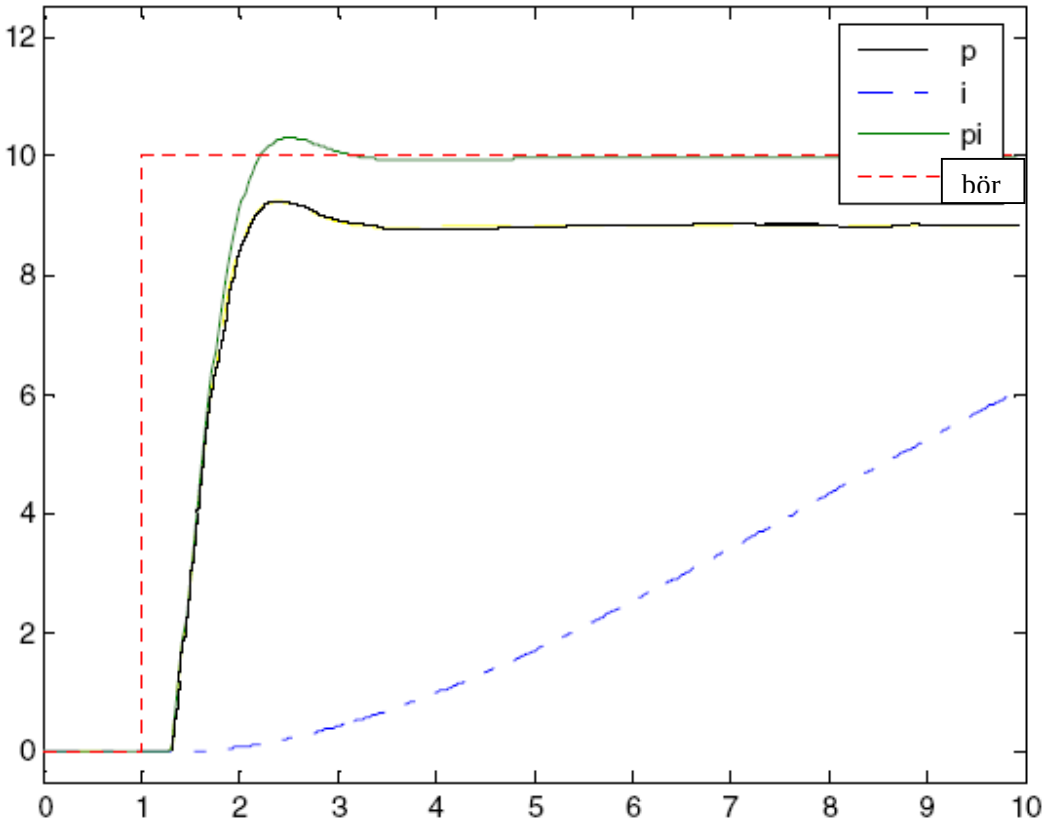
Nedan finns första kodalternativet för regleringen av maskinen. För varje varv utförs denna kodbit vilken justerar OCR0 (PWM):

```
if (RotationError > -40 || RotationError > 15){
        P_regulation = K_VALUE1 * RotationError; // P_reglering varje varv
         I_regulation = Sum * I_VALUE;   // summan av K_VALUE2 och RotationError
         PI =  I_regulation + P_regulation;  //
          Sum = PI ;

    if ((PI) > 255){
                OCR0 = 255;
             }

        else if ((PI) < 0){
                OCR0 = 0;
             }
        else{
                OCR0 = PI;
             }

            }

else{
      P_regulation = K_VALUE2 * RotationError;
       I_regulation = Sum * I_VALUE;              //summan av K_VALUE2 och Rotation Error
        PI = I_regulation + P_regulation;
         Sum = PI;

             if (( PI) > 255){                    //PI_regulation
```

```
        OCR0 = 255;
      }
      else if (( PI) < 0){
        OCR0 = 0;
      }

      else{
        OCR0 = PI;
      }
}
```

Koden består av två huvuddelar, en *if*- och en *else*-sats. *if* -satsen behandlar alla värden som är större än -40 och reglerar dessa med ett stort K-värde (K_VALUE1). *else*-satsen hanterar alla andra värden dvs värden som är mindre än -40 med ett mindre K-värde (K_VALUE2). K-värdet bestämdes dels genom experiment och dels genom vad som är allmänt känt när det gäller reglering t.ex. att K-värdet inte kan vara för stort. Annars uppstår stor översväng och eventuellt instabilitet.

Bilden nedan visar P-reglering, I-reglering och PI-reglering. Notera att reglerfelet elimineras med PI-reglering



Integration tar hänsyn till tidigare reglerfel. Summeringen av felen fortsätter tills man uppnår börvärdet. Därför används PI-reglering.

Andra kodalternativet PI_reglering

```
P_regulation = K_VALUE1 * RotationError;  //P reglering varje varv
 Sum = Sum + RotationError;               //Summeringen av de tidigare felen
I_regulation = Sum * I_VALUE;             //I reglering varje varv
PI = I_regulation + P_regulation;

 if ((PI) > 255){                         //Skydd mot för stor PI värde
    OCR0 = 255;
    PI = 225;
 }

 else if ((PI) < 0){                      //Skydd mot för liten PI värde
    OCR0 = 0;
    PI = 0;
 }

 else{
    OCR0 = (unsigned char)PI;             //Endast positive värde
 }
```

Alla konstanter experimenterades fram, framför allt för att kunna uppnå verklig precision i systemet.

8 Slutsats

Projektet har varit utvecklande och mycket lärorikt, eftersom vi behövde skaffa nya kunskaper och definitiv använda kunskaper vi skaffat under vår utbildning ”Robotik Mekatronik och Elektronik”. Som vår handledare sa när han fick reda på vad vi skulle göra för examens arbete ” Det sammanfattar verkligen allt ni har lärt er under utbildningen ”. Taralis, Lars Svensson, Björn Nordin och Åke Eriksson har verkligen gett oss de bästa förutsättningar för att klara detta projekt. Vi vill verkligen tacka dem för den introduktion de gav oss nästan en månad innan projektet sattes igång. Detta gav oss verkligen tid att forma en tydlig idé och läsa på det som var nytt t.ex. C# och AVR mm.

Efter detta projekt som berört områden som reglerteknik, programmering, analogteknik, mätteknik och Matematik så känner vi stolthet över att ha gjort detta projekt och kanske kommer det att finns lite av oss i den nya TCD maskinen.
Två stolta killar tackar och önskar alla i företaget

”ha ett långt liv med gud på er sida här på jorden”

9 Resultat

Projektet handlar om att reglera huvudmotor i en TCD-maskin, alltså ska motor hålla en önskad hastighet oberoende av belastning. För att klara detta använde vi oss av PI-reglering.

Ett mål var att kunna hålla en hastighet på 2 m/s, och med tanke på de svårigheter som påträffades och löstes under projektets gång får resultatet anses bra. Att företaget Talaris sett och godkänt projektet innebär att det har uppfyllt alla krav i kravspecifikationen.

Det andra viktiga målet var att styrprogrammet till motorn skulle göras i C#. Trots vår okunskap i C# i början av projektet lyckades vi göra ett välfungerande styrprogram.

Ytterligare ett mål var att vi skulle få lära oss ny teknik och förbereda oss för det som förväntas av ingenjörer i vardagen. Uppdraget var tydligt och lärorik och med rätt stöd i företaget kunde det inte ha slutat bättre.

Tack

Percy Villegas T
Basel Lwei

10 Referenser

[1] Modern Reglerteknik av Thomas, Bertil.
Kapitel **4**
4.4 Proprotionell reglering
4.5 Integrerande reglering
4.6 PI-reglering
4.7 Deriverande verkan och PID-reglering
ISBN: *91-47-05085-3*

[2] Analog Elektronik av Molin, Bengt.
Kapitel **5**
Exempel **5.1** RC lågpasslänk
Underkapitel
2.3Förstärkarkopplingar
Icke inverterande förstärkarekoppling
ISBN: *91-44-01435-X*

[3] Realisering och inställning av PID-regulatorer (2007-02-19) av Jonas Wahlfrid.
Avsnitt 4.2 Tidsdiskret PID-regulator.
http://dspace.mah.se/dspace/bitstream/2043/6484/1/PIDJonasWahlfrid2007_02_19.pdf

[4] Serial Port (RS-232 Serial COM Port) in C# .NET
Underrubrik: Let's Have the Code.
http://msmvps.com/blogs/coad/archive/2005/03/23/SerialPort-_2800_RS_2D00_232-Serial-COM-Port_2900_-in-C_2300_-.NET.aspx

[5] Extreme electronics, AVR Tutorials
Avsnitten om Timers: Timers in Comare Mode – Part I, Part II & Part III.
Avsnittet om USART: Using the USART of AVR Microcontrollers.
http://extremeelectronics.co.in/avr-tutorials/

[6] 8-bit Microcontroller with 32K Bytes In-System Programmable Flash, Atmega32A
http://www.atmel.com/dyn/resources/prod_documents/doc8155.pdf

[7] AVR STK500 User Guide
http://www.atmel.com/dyn/resources/prod_documents/doc1925.pdf

11 Appendix

Appendix A **Teknisk specifikation**

Appendix B **Flödesschema**

Appendix C **Bilder**

Appendix D **Programkod**

Appendix A – Teknisk specifikation

11.1 Räknare

11.1.1 Timer counter 0

Table 14-6. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Sid 86 i databladet för Atmega32 **[6]**

11.1.2 Timer counter 1

Table 16-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Sid 115 i databladet för Atmega32 **[6]**

11.1.3 Timer counter 2

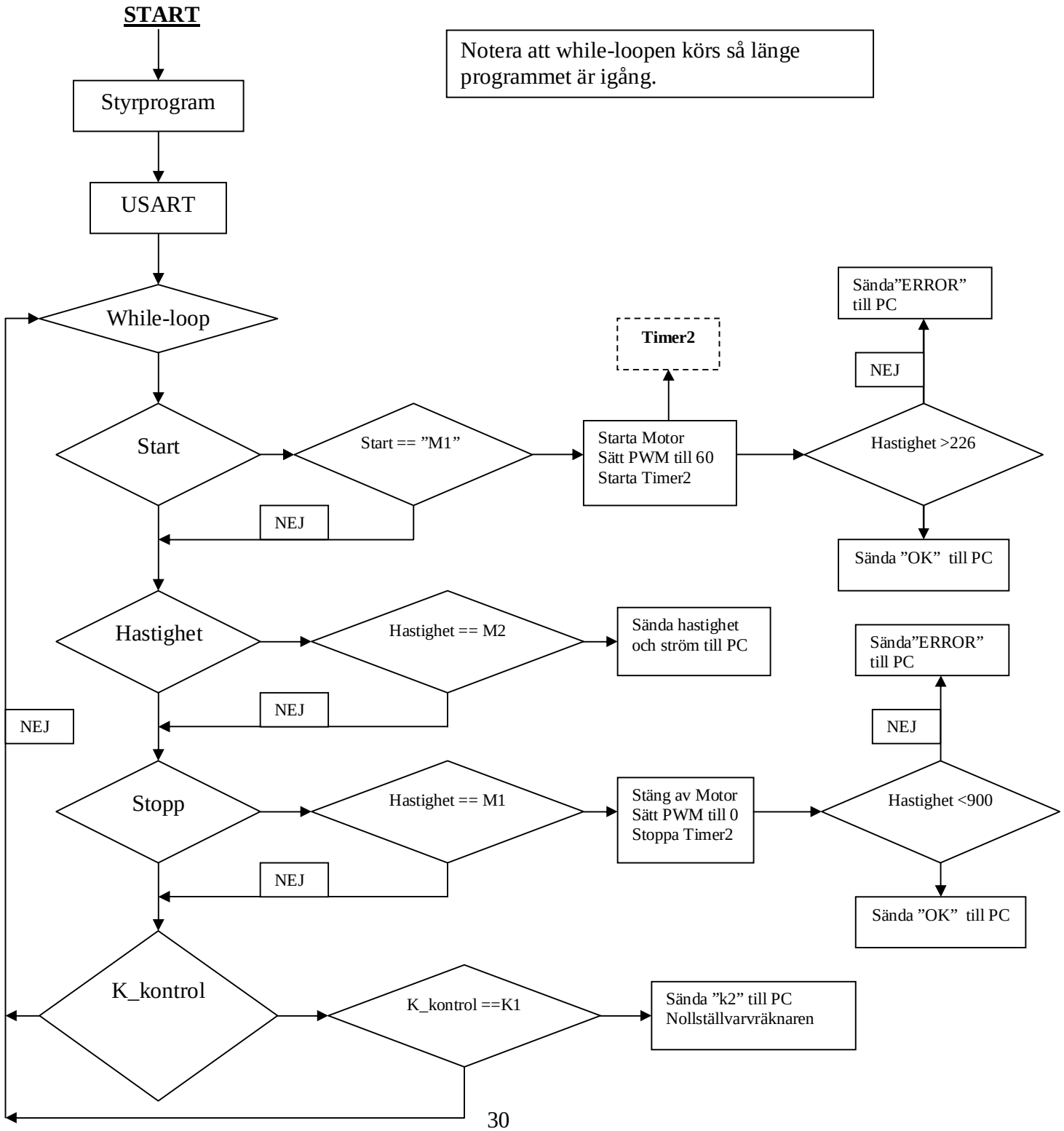
Table 17-6. Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_{T2S}/(No prescaling)
0	1	0	clk_{T2S}/8 (From prescaler)
0	1	1	clk_{T2S}/32 (From prescaler)
1	0	0	clk_{T2S}/64 (From prescaler)
1	0	1	clk_{T2S}/128 (From prescaler)
1	1	0	clk_{T2S}/256 (From prescaler)
1	1	1	clk_{T2S}/1024 (From prescaler)

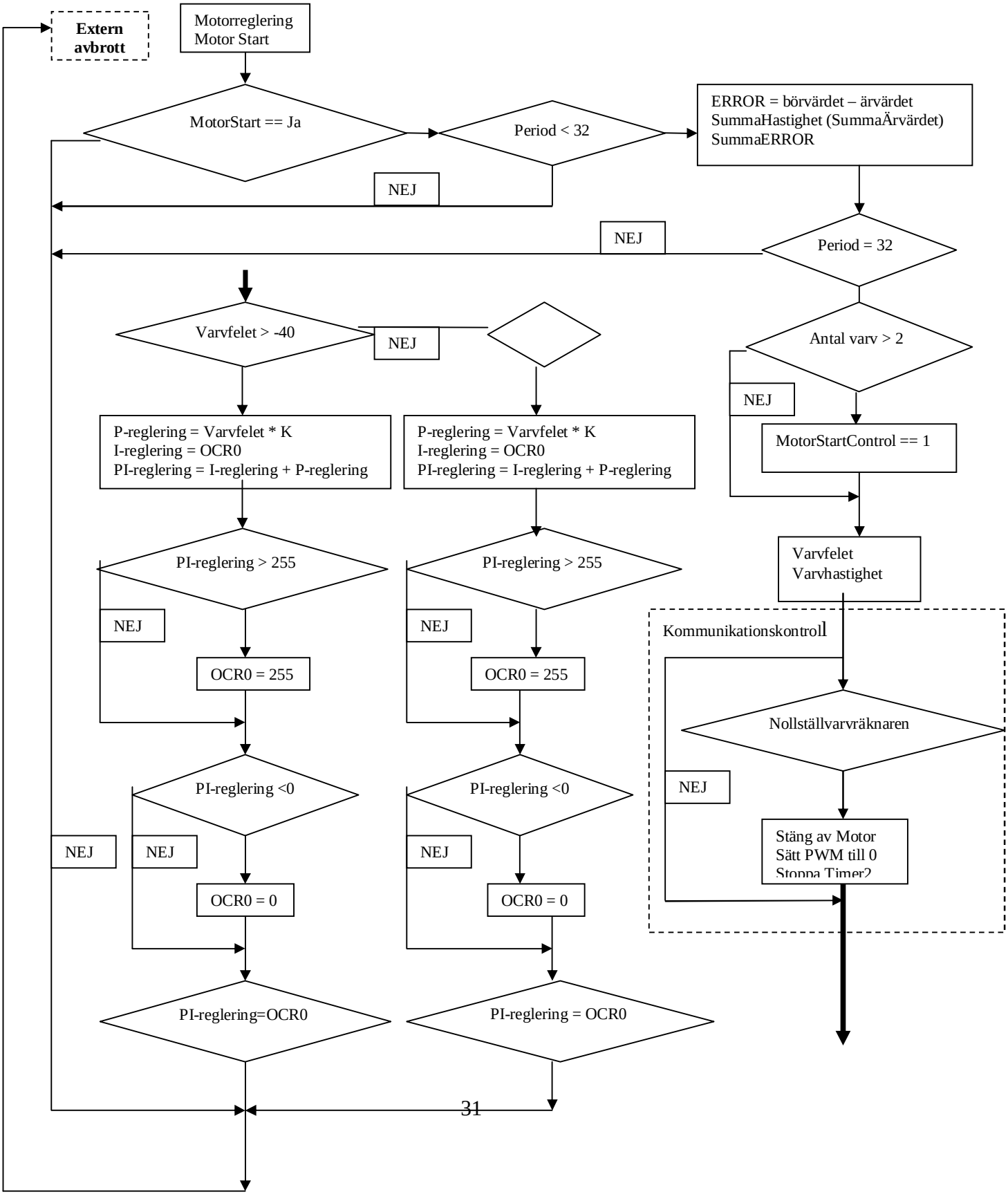
Sid 134 i databladet för Atmega32 **[6]**

11.2 Appendix B - Flödesschema

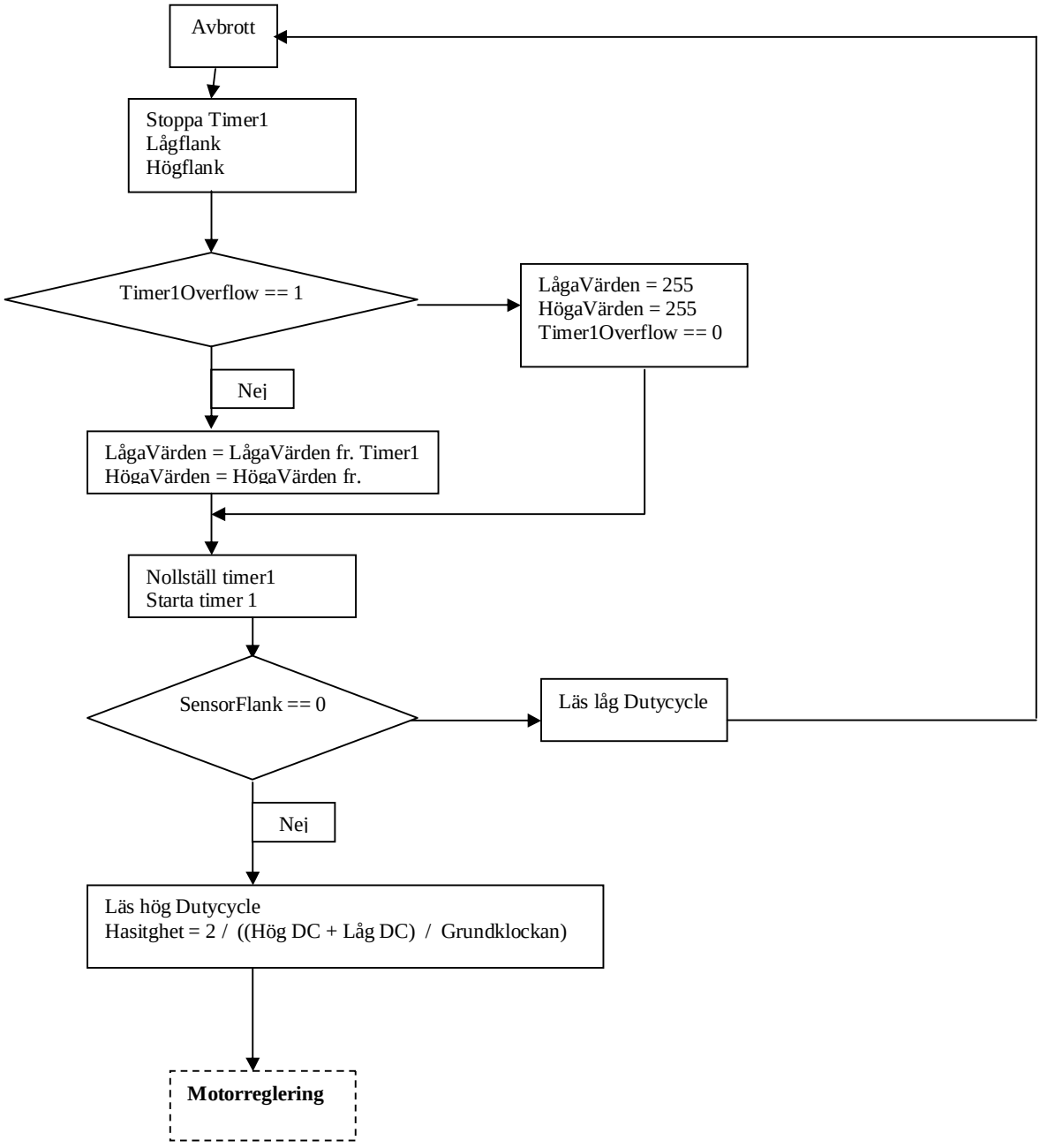
11.2.1 Startsekvensen



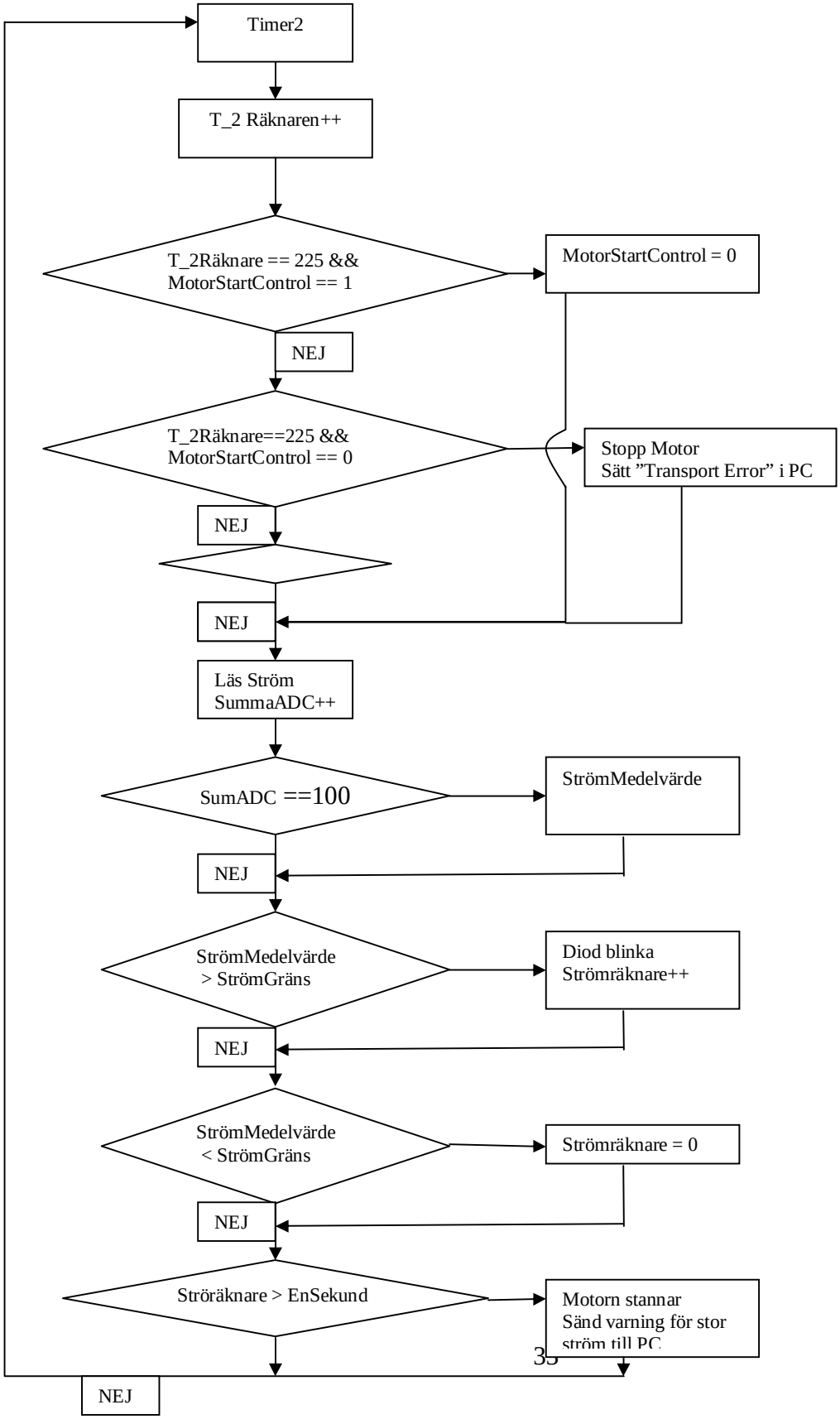
11.2.2 Motorreglering



11.2.3 Extern avbrott



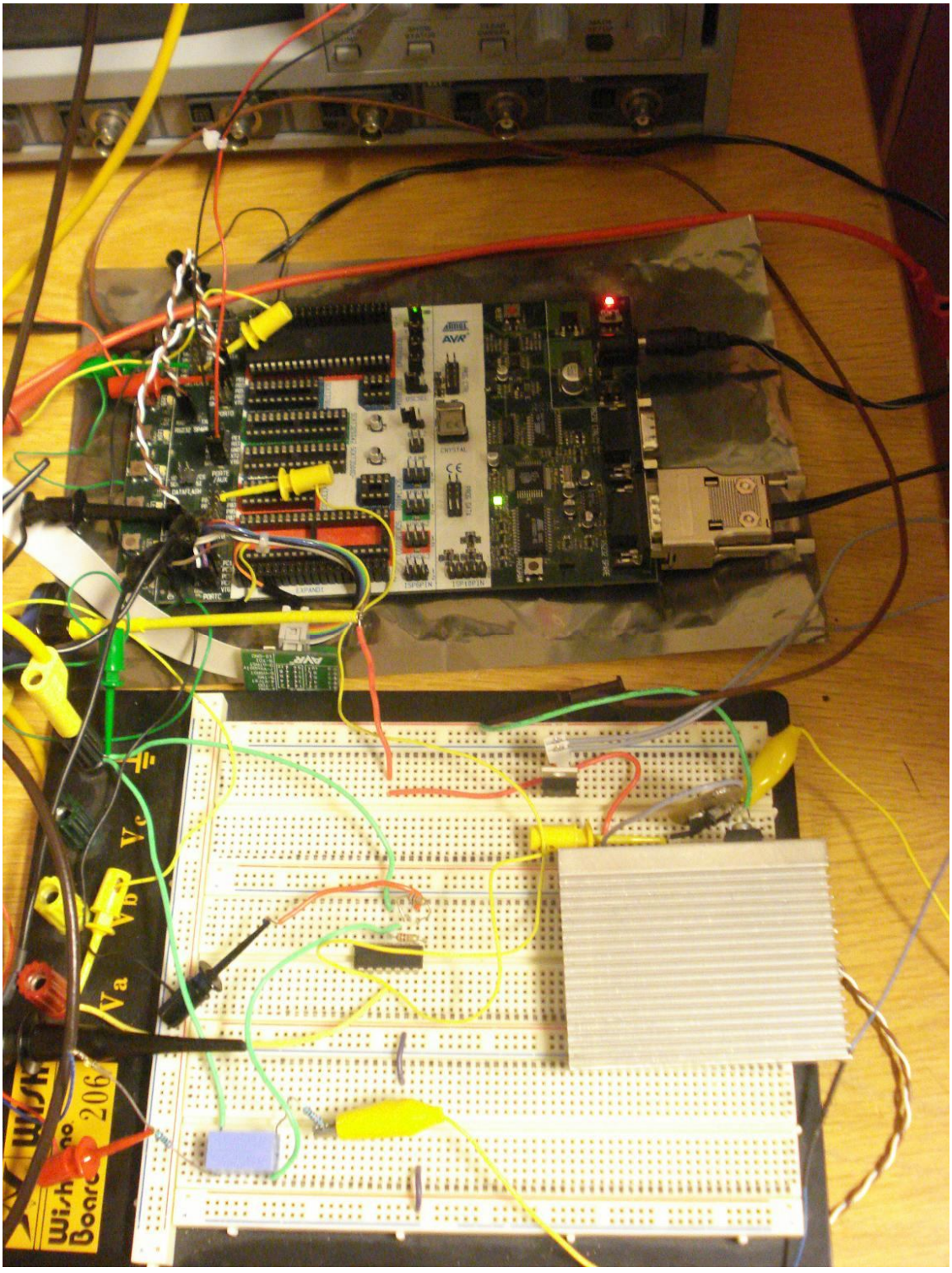
11.2.4 Timer2


Timer2
T_2 Räknaren++
T_2Räknare == 225 &&
MotorStartControl == 1
MotorStartControl = 0
NEJ
T_2Räknare==225 &&
MotorStartControl == 0
Stopp Motor
Sätt ”Transport Error” i PC
NEJ
NEJ
Läs Ström
SummaADC++
SumADC ==100
StrömMedelvärde
NEJ
StrömMedelvärde
> StrömGräns
Diod blinka
Strömräknare++
NEJ
StrömMedelvärde
< StrömGräns
Strömräknare = 0
NEJ
Ströräknare > EnSekund
Motorn stannar
Sänd varning för stor
ström till PC
NEJ

Appendix C – Bilder

Envånings TCD maskin, som rymmer tre kasetter.

Tvåvånings TCD maskin, som rymmer tre kasetter.



Huvudprocessorkortet och drivsteget

11.3 Appendix D – Programkod

11.3.1 Kod för huvudmotorreglering

```
/*****************************************************
This program was produced by the
CodeWizardAVR V2.03.8a Standard
Automatic Program Generator
© Copyright 1998-2008 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project : Examensarbete 2009  "Huvudmotorreglering"
Version : Final (090313)
Date    : 04/03/2009
Author  : Percy Villegas & Basel Lwai
Company : Talaris
Comments:


Chip type           : ATmega32
Program type        : Application
Clock frequency     : 14.745600 MHz
Memory model        : Small
External RAM size   : 0
Data Stack size     : 512
*****************************************************/

#include <mega32.h>

#define VELOCIY_CONTROL   2000          // Setpoint
#define CURRENT_LIMIT     100           // Max possible limit

#define ALT1

#ifdef ALT1
#define  K_VALUE1        0.019         // alt 1 =0.019
#define  I_VALUE         0.017         //  alt 1 = 0.017
#endif

#ifdef ALT2
#define  K_VALUE1        0.05        //    alt 2 = 0.05
#define  K_VALUE2        0.002       //    alt 2 = 0.002
#define  I_VALUE         1           //      alt 2 = 1
#endif


#define ADC_VREF_TYPE 0x40

#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
```

```
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)

//USART Receiver buffer
#define RX_BUFFER_SIZE 16
char rx_buffer[RX_BUFFER_SIZE];

#if RX_BUFFER_SIZE < 256
unsigned char rx_wr_index,rx_rd_index,rx_counter;
#else
unsigned int rx_wr_index,rx_rd_index,rx_counter;
#endif

//This flag is set on USART Receiver buffer overflow
bit rx_buffer_overflow;

//USART Receiver interrupt service routine
interrupt [USART_RXC] void usart_rx_isr(void){
   char status, data;
   status = UCSRA;
   data = UDR;
   if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN)) == 0){
      rx_buffer[rx_wr_index] = data;

      if (++rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;

         if (++rx_counter == RX_BUFFER_SIZE){
            rx_counter = 0;
            rx_buffer_overflow=1;
         };
   };
}

#ifndef _DEBUG_TERMINAL_IO_

//Get a character from the USART Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void){
char data;
while (rx_counter==0);
data = rx_buffer[rx_rd_index];
if (++rx_rd_index == RX_BUFFER_SIZE) rx_rd_index = 0;
#asm("cli")
--rx_counter;
#asm("sei")
return data;
}
#pragma used-
```

```
#endif

//USART Transmitter buffer
#define TX_BUFFER_SIZE 16
char tx_buffer[TX_BUFFER_SIZE];

#if TX_BUFFER_SIZE < 256
unsigned char tx_wr_index, tx_rd_index, tx_counter;
#else
unsigned int tx_wr_index, tx_rd_index, tx_counter;
#endif

//USART Transmitter interrupt service routine
interrupt [USART_TXC] void usart_tx_isr(void){
   if (tx_counter){
      --tx_counter;
      UDR = tx_buffer[tx_rd_index];
      if (++tx_rd_index == TX_BUFFER_SIZE) tx_rd_index=0;
   };
}

#ifndef _DEBUG_TERMINAL_IO_

//Write a character to the USART Transmitter buffer
#define _ALTERNATE_PUTCHAR_
#pragma used+
void putchar(char c){
while (tx_counter == TX_BUFFER_SIZE);
#asm("cli")

   if (tx_counter || ((UCSRA & DATA_REGISTER_EMPTY) == 0)){
      tx_buffer[tx_wr_index] = c;
      if (++tx_wr_index == TX_BUFFER_SIZE) tx_wr_index = 0;
         ++tx_counter;
   }
   else
      UDR = c;
      #asm("sei")
}

#pragma used-
#endif

//Standard Input/Output functions
#include <stdio.h>

//Declare your global variables here
unsigned char sendBuffert [50];
unsigned char MotorStart;
unsigned long int ProcessError, ProcessErrorSum;
unsigned long int index32 = 0, indexErrorChanges = 0;
unsigned long int adc;

float P_regulation = 0, I_regulation = 0;
int PI = 0;
```

```
unsigned long int low_time = 0, high_time = 0;
int velocity ,  Contact =0;
unsigned long int Sumvelocity = 0, velocity32 = 0;
int RotationError = 0;

#define F_CPU 14745600

int buffertIndex = 0;
int MotorStartControl = 0, Timer2Counter = 0, RotationCounter = 0;
unsigned long int index = 0 ;
unsigned long int Timer1Overflow;
unsigned int Current_timer = 0;
unsigned int SumAdc100 = 0;
unsigned int current = 0;

float  Sum = 0;

/*****************************************************************************************
****
Read ADC 8 bits
*****************************************************************************************
****/
unsigned int read_adc(unsigned char adc_input){
   ADMUX=adc_input|ADC_VREF_TYPE;

   //Start the AD conversion
   ADCSRA|=0x40;

   //Wait for the AD conversion to complete
   while ((ADCSRA & 0x10) == 0);
      ADCSRA|= 0x10;
   return ADCW;
}


void SendAnswer(void){
  buffertIndex = 0;

   while (sendBuffert[buffertIndex] != '\n'){
      putchar(sendBuffert[buffertIndex]);
      sendBuffert[buffertIndex] = '\n';
      buffertIndex++;
   }
}


/*****************************************************************************************
**************************
 Timer2 output compare interrupt service routine
 This interrupt is called  115200/ 255 = 451.7647 /secounds
 This timer starts when you press start button in c# and it gets to StartMotor(void) function in this program
 This timer stops in four different cases
  -When you stop the motor
  -When the current exides its limit
```

```
*****************************************************************************************
*************************/

interrupt [TIM2_COMP] void timer2_comp_isr(void){
   Current_timer++;
   Timer2Counter++;

    if (Timer2Counter == 200 && MotorStartControl == 1){          //If motor exided two rotations in 45% of
a secound
       Timer2Counter = 0;
       MotorStartControl = 0;
    }

    else if (Timer2Counter == 200  && MotorStartControl == 0){    //If motor hasn't rotated two times in a
half secound, stop the motor and view a message in C#

       OCR0 = 0x00;
       MotorStart = 'N';
       sendBuffert[0] = 'A';
       sendBuffert[1] = '9';
       sendBuffert[2] = '\n';
       SendAnswer();

       TCCR2=0x00;

       Timer2Counter = 0;

    }

    else{

    }

    SumAdc100++;
    adc = adc + read_adc(1);

    if (SumAdc100 == 100){
          current = (adc / 100);
          adc = 0;
          SumAdc100 = 0;
    }

    PORTD = 0xF0;

      if (current > CURRENT_LIMIT ){
          PORTD = 0x00;
          Current_timer++;
      }

      if (current < CURRENT_LIMIT ){
          Current_timer = 0;
      }

      if (Current_timer > 451){
          OCR0 = 0x00;
```

```
        Current_timer = 0;
        current = 0;

        Timer2Counter = 0;
        TCCR2 = 0x00;
        PORTD = 0xF0;
        MotorStart = 'N';
        sendBuffert[0]= 'F';
        sendBuffert[1]= 'F';
        sendBuffert[2]= '\n';
        SendAnswer();
      }
}

void SerialCommunication(void){

      Contact++;


      if (Contact > 130){
         OCR0 = 0x00;
        PI = 0;
        MotorStart = 'N';
        TCCR2 = 0x00;
        Timer2Counter = 0;
        Contact = 0;
      }

}

/*****************************************************************************************
**********************
The motor starts, OCR0 = 0x60 for the regulation begins equal to PWM 96. The max value for PWM is 255

M1 is a start command for TCD

Timer2 begins TCCR2=0x05;   14745600/128 = 115200 Hz
*****************************************************************************************
**********************/

void StartMotor(void){
unsigned int ii = 0;

   if ((sendBuffert[0]=='M')&&(sendBuffert[1]=='1')){

      Contact = 0;
      MotorStart = 'J';
      OCR0 = 60;   //60
      //PI = 60;
      Sum = 1500;
      TCCR2 = 0x05;

      while (ii < 65535){          // delay for motor stop
        ii++;
      }
```

```
        if (velocity > 226){

           sendBuffert[0]= '0';
           sendBuffert[1]= 'O';
           sendBuffert[2]= 'K' ;
           sendBuffert[3]= '\n';
           SendAnswer();
        }

        else{

           sendBuffert[0]= 'F';
           sendBuffert[1]= 'E';
           sendBuffert[2]= 'R';
           sendBuffert[3]= 'R';
           sendBuffert[4]= 'O';
           sendBuffert[5]= 'R';
           sendBuffert[6]= '\n';
           SendAnswer();
        }

   }
}

/*****************************************************************************
Velocity calculations
m  is for dividing send value in Textbox in C#
*****************************************************************************/


void MotorSpeed(void){
int velocityHundredth, velocityTen, velocityOne, velocityTenC, velocityOneC;

   if ((sendBuffert[0]=='M')&&(sendBuffert[1]=='2')){

      sendBuffert[0]    =   (Sumvelocity  / 1000) + 48;

      velocityHundredth  =    (int) Sumvelocity % 1000;
      sendBuffert[1]    =   (velocityHundredth / 100) + 48;

      velocityTen       =   velocityHundredth % 100;
      sendBuffert[2]    =   (velocityTen / 10) + 48;

      velocityOne       =   velocityTen %10;
      sendBuffert[3]    =    velocityOne + 48;


      sendBuffert[4]    =   'm';                  //m  is for dividing send value in Textbox in C#

      sendBuffert[5]    =   (current / 100) + 48;

      velocityTenC      =    current % 100;

      sendBuffert[6]    =   (velocityTenC / 10) + 48;
```

```
        velocityOneC      =    (int) velocityTenC % 10;
        sendBuffert[7]    =    velocityOneC + 48;
        sendBuffert[8]= '\n';
        SendAnswer();
    }
}


/*****************************************************************************
The motor stop, OCR0=0x00  equal to PWM 0
MO is a stop command for TCD
*****************************************************************************/

void StopMotor(void){
unsigned int i = 0;

    if ((sendBuffert[0]=='M')&&(sendBuffert[1]=='0')){
        Timer2Counter = 0;
        OCR0 = 0x00;
        MotorStart = 'N';
        TCCR2 = 0x00;

        while (i < 65535){                  //Delay for motor stop
            i++;
        }

          if ((velocity / 3) < 900){
           sendBuffert[0]= '0';
           sendBuffert[1]= 'O';
           sendBuffert[2]= 'K';
           sendBuffert[3]= '\n';
           SendAnswer();
          }

          else{
           sendBuffert[0]= 'F';
           sendBuffert[1]= 'E';
           sendBuffert[2]= 'R';
           sendBuffert[3]= 'R';
           sendBuffert[4]= 'O';
           sendBuffert[5]= 'R';
           sendBuffert[6]= '\n';
           SendAnswer();
          }
    }
}


void CommunicationControl(void ){


     if ((sendBuffert[0]=='K')&&(sendBuffert[1]=='1')){     //Communication Control with C#

        Contact =  0;
```

```
        sendBuffert[0]= 'K';
        sendBuffert[1]= '2';
        sendBuffert[2] = '\n';
        SendAnswer();
      }
}

// Timer 1 overflow interrupt service routine
interrupt [TIM1_OVF] void timer1_ovf_isr(void){

Timer1Overflow = 1;
Sumvelocity = 225;
}

//External Interrupt 0 service routine, read the sensor
interrupt [EXT_INT0] void ext_int0_isr(void){

unsigned int count_hi=0, count_lo=0;


TCCR1B=0x00;          //Stop counter timer1

if (Timer1Overflow == 1){
   count_lo = 0xFF;    //Read counter
   count_hi = 0xFF;    //Read counter
   Timer1Overflow = 0;
   index++;
}

else{
   count_lo = TCNT1L;      //Read counter
   count_hi = TCNT1H;      //Read counter
}
TCNT1H=0x00;              //Zero counter registers for next period
TCNT1L=0x00;
TCCR1B=0x01;              //Start counter again (prescaler divisor 1)

if (PIND.2 == 0){         //Sensor has gone low, read out how long the low period was
   count_hi = (count_hi << 8);
   low_time = count_lo + count_hi;
}

else{                     //sensor has gone high, read out how long the high period was
   count_hi = (count_hi << 8);
   high_time = count_lo + count_hi;

   velocity = 2/((float)(high_time +low_time)/F_CPU); // the velocity in each period



        if ( MotorStart == 'J')     //If the button Start is push in the C# program
        {
```

```
if (index32 < 32  ){

  ProcessError = VELOCIY_CONTROL - velocity;
  velocity32 = velocity32 + velocity;
  index32++;
  ProcessErrorSum = ProcessErrorSum + ProcessError;
  indexErrorChanges++;



   if (index32 == 32){   //Index32 is equal to 32 when the motor has gone one rotation

      RotationCounter++;

      if (RotationCounter > 2){
        RotationCounter = 0;
        MotorStartControl = 1;
      }



      index32 = 0;
      RotationError = (float)ProcessErrorSum/(float)32;
      Sumvelocity = (float)velocity32/(float)32;
      velocity32 = 0;




/*****************************************************************************
Regulation Alt.1
*****************************************************************************/

#ifdef ALT1


    P_regulation = K_VALUE1 * RotationError;


  Sum = Sum + RotationError;
  I_regulation = Sum * I_VALUE;
  PI = I_regulation + P_regulation;

   if ((PI) > 255){
     OCR0 = 255;
     PI = 225;
   }

   else if ((PI) < 0){
     OCR0 = 0;
     PI = 0;
```

```
 }

 else{
    OCR0 = (unsigned char)PI;
 }



#endif

/*****************************************************************************
Regulation Alt.2
*****************************************************************************/
#ifdef ALT2
    if (RotationError > -40 || RotationError > 15){
  P_regulation = K_VALUE1 * RotationError;

      Sum = PI ;

       I_regulation = Sum * I_VALUE;

         PI =  I_regulation + P_regulation;
          //SumBuffert[indexErrorSumBuffert] = PI;

 if ((PI) > 255){    //  PI_regulation
    OCR0 = 255;
 }

 else if ((PI) < 0){
    OCR0 = 0;
 }

 else{
    OCR0 =  (unsigned char)PI;
 }

}


 else{
     P_regulation = K_VALUE2 * RotationError;
      Sum = PI ;

       I_regulation = Sum * I_VALUE;

       PI =  I_regulation + P_regulation;
        // SumBuffert[indexErrorSumBuffert] = PI;

 if (( PI) > 255){        //PI_regulation is equal to  OCR0 + P_regulation
   OCR0 =  255;
 }
 else if (( PI) < 0){
   OCR0 = 0;
 }
```

```
            else{
              OCR0 = PI;
            }
           }

#endif

            SerialCommunication();


           //indexErrorSumBuffert++;
            ProcessErrorSum = 0;
            RotationError = 0;
         }


     }

    }

}


}



void main(void)
{


// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=Out Func2=In Func1=In Func0=Out
// State7=T State6=T State5=T State4=T State3=0 State2=T State1=T State0=T
PORTB=0x00;

DDRB=0xFF;
//DDRB=0x0B;
//PORTB=0xF0;
//DDRB=0xF0;


// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;
```

```
// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0xF0;
DDRD=0xF0;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 1843.200 kHz
// Mode: Fast PWM top=FFh
// OC0 output: Disconnected
TCCR0=0x6A;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x02;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;



// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value 14745600/128 Hz
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
//TCCR2=0x05;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0xFF;
```

```
// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Any change
// INT1: Off
// INT2: Off
GICR|=0x40;
MCUCR=0x01;
MCUCSR=0x00;
GIFR=0x40;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x84;

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 9600
UCSRA=0x00;
UCSRB=0xD8;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x5F;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// ADC initialization
// ADC Clock frequency: 230.400 kHz
// ADC Voltage Reference: AREF pin
// Only the 8 most significant bits of
// the AD conversion result are used
ADMUX=ADC_VREF_TYPE;
ADCSRA=0x85;


// Global enable interrupts
#asm("sei")

while (1)   // This loop is activated all the time
     {


     sendBuffert[0] =   getchar();
     sendBuffert[1] =  getchar ();


        StartMotor();
        MotorSpeed();
```

```
        StopMotor();

        CommunicationControl();

 };
}
```

11.3.2 Kod för motorstyrning (C#)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

namespace Formt{

    delegate void ButtonEventHandler(object sender, EventArgs e);
    delegate void SetTextDeleg(String text);
    public partial class Form1 : Form{

        String TexboxChoose = "O",TexboxChoos = "i";
        int i = 0;
        public Form1(){

            InitializeComponent();

            serialPort1.DataReceived += new
SerialDataReceivedEventHandler(port_DataReceived);
                    button1.Click += new EventHandler(button1_Click);
                    button2.Click += new EventHandler(button2_Click);
                    button3.Click += new EventHandler(button3_Click);
                    button4.Click += new EventHandler(button4_Click);
                    timer1.Tick += new EventHandler (OnTimerEvent);

                    serialPort1.PortName = "COM1";
                    serialPort1.BaudRate = 9600;
                    serialPort1.DataBits = 8;
                    serialPort1.Parity = Parity.None;
                    serialPort1.StopBits = StopBits.One;

                    serialPort1.Open();
        }

        private void OnTimerEvent(object source, EventArgs e){

            if (timer1.Interval == 1000){
                BackColor = System.Drawing.Color.Aqua;
                timer1.Enabled = false;
                MessageBox.Show("Communication ERROR, see RS232!!! \n");
            }
            else{
                serialPort1.Close();
                serialPort1.Open();
                serialPort1.Write("K1");

                timer1.Interval = 1000;
```

```
                timer1.Enabled = true;
            }

}

        private void button1_Click(object sender, EventArgs e){

                timer1.Interval = 3000;
                timer1.Enabled = true;
                serialPort1.Close();
                textBox2.Text = "";

                BackColor = System.Drawing.Color.GreenYellow;

                serialPort1.Open();

                //write data to serial port
                serialPort1.Write("M1");
                TexboxChoose = "11";
              }

                 private void button4_Click(object sender, EventArgs e)
                 {
                     textBox1.Text = "";
                     textBox2.Text = "";
                     textBox3.Text = "";
                     textBox4.Text = "";
                 }


                 private void port_DataReceived(object sender,
SerialDataReceivedEventArgs e){

                    String data = serialPort1.ReadExisting();
                    this.BeginInvoke(new SetTextDeleg(si_DataReceived),
new object [] {data});
                }

                private void si_DataReceived(string data){

                    Char[] delimiterChars = {'m'};
                    String[]words = data.Split(delimiterChars);

                    if (TexboxChoose == "11"){

                        textBox2.Text = data;
                        TexboxChoose = "0";
                    }

                    else if (data == "K2"){

                        timer1.Interval = 3000;
                        timer1.Enabled = true;
                    }

                    else if (data == "A9"){
```

```
                    MessageBox.Show("Transport Error!!! \n");
                }

                else if (TexboxChoose == "12"){

                    textBox1.Text = data.Trim();
                    TexboxChoose = "0";
                }

                else{

                for (int IndexSplit = 0; IndexSplit < words.Length;
IndexSplit++){
                    if (IndexSplit == 0){

                        if (words[0] == "FF"){

                            BackColor =
System.Drawing.Color.YellowGreen;
                            MessageBox.Show("Start the motor again
if the light is OFF!!! \n If the light is ON  call the support!!!");
                        }

                        else{
                            textBox3.Text = words[0];
                        }
                    }

                        if (IndexSplit == 1){

                            textBox4.Text = words[1];
                        }
                    }
                }

            }


            private void button2_Click(object sender, EventArgs e){

                timer1.Enabled = false;

                //comfiguring the serial port
                serialPort1.Close();
                textBox1.Text = "";
                serialPort1.Open();
                serialPort1.Write("M0");
                BackColor = System.Drawing.Color.YellowGreen;
                TexboxChoose = "12";
            }

            private void button3_Click(object sender, EventArgs e){

                //comfiguring the serial port
```

```
                serialPort1.Close();
                textBox3.Text = "";
                serialPort1.Open();
                serialPort1.Write("M2");
                TexboxChoose = "13";
            }

            private void Form1_Load(object sender, EventArgs e){
            }

        }

            public class Buttonn{

            public string clicked(){

                int Ampere = 10, Resistans = 5, Volt;
                String Volta;
                Volt = Ampere * Resistans;
                Volta = Convert.ToString(Volt);

                return Volta;
            }
        }
}
```