Examensarbete 10 poäng C-nivå

KONSTRUKTION OCH TILLVERKNING AV

DENSITETSMÄTARE

Reg. kod: Oru-Te-EXE084-EL108/04

Johan Landmark och Oscar Tryggvesson

Elektronikingenjörsprogrammet 120 poäng

Örebro vårterminen 2004

Examinator: Sune Bergelin
Handledare: Thorbjörn Andersson

Örebro universitet
Institutionen för teknik
701 82 Örebro

Örebro University
Department of technology
SE-701 82 Örebro, Sweden

Förord

Vi vill tacka följande personer. **Kjell Mårdensjö** för din hjälp med programmering av AVR-processorn, **Torbjörn Andersson** (Telnek) för ditt aktiva stöd genom hela projektet, **Thomas M. Poms** för dina idéer kring konstruktionen av vår prototyp, för ert stöd i vårt projektarbete.

Örebro 2002-12-11

Johan Landmark

Oscar Tryggvesson

Sammanfattning

Examensarbetet består av att konstruera en prototyp för mätning av luftdensitet i fält. Prototypen ämnar underlätta inställningarna av förgasare då den ger mätvärden kontinuerligt på plats. Förhoppningsvis kommer den att spara tid till användaren då han/hon inte behöver söka efter värden för att sedan räkna ut luftens densitet.

Hjärnan i prototypen består av en AvR mikroprocessor som utför beräkningar och visar resultatet på en LCD-display. Densiteten beräknas utgående från luftfuktighet, lufttryck och temperatur vilket fås från givare inbyggda i konstruktionen.

Kraven för examensarbetet är att bygga en prototyp som klarar av att mäta luftens densitet i temperatur området -10ºC till +50ºC. För en mer detaljerad beskrivning se kravspecifikationen sid. 9.

Målet med vårt projekt är att konstruera prototypen inom 10 veckor samt att uppfylla våra krav.

Resultatet av projektet med hänsyn taget till förutsättningar, anser vi blivit bra. Produkten blev aldrig helt färdigkonstruerad och noggrannheten i mätningen blev inte heller vad vi hade tänkt oss. Trots detta har vi lärt oss mycket inom detta nya område.

Abstract

Our thesis consists in developing a prototype for field measurement that measures air density. The prototype is meant to relieve the job when calibrating fuel injectors when it provides measurement values continuously. Hopefully it will save time for the user thus he/she doesn't need to look for air density values.

The brain in our construction is the AvR microprocessor that performs calculations and presents the result on an LCD-display. The air density is calculated by values given from the sensors.

The objektive was to build a prototype that manages to calculate the air density between -10°C to +50°C. For more detailed specification se page 9.

Our target was to construct the prototyp within 10 weeks and to satesfy our demands.

The project result considering the limited resources in terms of material, we think that the thesis has worked out well. The product never got finished and the accuracy didn't get as good as we hoped. However we learned a lot in this new area.

Innehållsförteckning

Bilder

1 Introduktion

1.1 Bakgrund

Vid karting, gokarttävlingar, trimmas motorernas olika delar för att uppnå optimal prestanda. En av de viktiga aspekterna är förgasarna eftersom det är de som förser motorn med bränsle. När man justerar förgasarna har luftfuktigheten en viss betydelse på hur man ställer in dem. Då företaget Telnik fått i uppgift att tillverka en smart lösning till att mäta luftens densitet så att man på plats direkt kan justera förgasarinställningarna har vi åtagit oss att lösa uppgiften som ett examensarbete och tillverka en mobil prototyp för test i fält.

Den stora fördelen med att kunna mäta luftens densitet mobilt är att man inte behöver ta hänsyn till var i världen man tävlar då mätningen och inställningarna kan göras på plats.
En lösning till problemet skulle spara tid och pengar, vilket är två viktiga aspekter.

1.2 Projekt beskrivning

Projektets slutgiltiga mål är att kunna mäta luftens densitet kontinuerligt.För att åstadkomma detta krävs följande parametrar:

- Luftfuktighet
- Lufttryck
- Temperatur

På uppdrag av Telnik och med anledningar från vår utredning skall en prototyp tillverkas för test i fält. Uppgiften omfattar även en utredning om hur noggrant densiteten kan mätas med de kommersiellt tillgängliga komponenterna med hänsyn tagen till vår budget.

1.3 Förutsättningar

Förutsättningarna för konstruktionen och tillverkningen av prototypen är att använda oss av en AvR processor och allt befintligt material på skolan för att på så sätt sänka kostnaderna.

1.4 Kravspecifikation

1.4.1 Miljötålighet

- Prototypen skall kunna användas i temperaturområdet -10 till +50 grader celsius.
- Prototypen skall vara droppsäker, tåla lättare duggregn samt mild behandling vid användnig.

1.4.2 Enhetskrav

- Prototypens mått får ej överskrida måtten: 150*150*100mm.
- Prototypen skall kunna redovisa de olika parametrarna på en LCD-display samt vara försedd med en knappsats för att kunna mata in data.
- Prototypen skall vara försedd med RS-232 för kommunikation, in/ut, med en stationär enhet.
- Prototypen skall klara av att mäta och lagra data under 24 timmar.
- Batteri för drift i 24 timmar erfordras samt en extern spänningsmatning på 10-36volt.

1.4.2 Budget

Komponentpriser i prisklassen 200 sek, avseende temperatur, lufttryck och luftfuktighetsgivare.

2 Specifikation

2.1 Introduktion

I början av vårt arbete studerade vi vårt problem och försökte komma fram till den billigaste och bästa lösningen. När vi funderade på vilken lösning till problemet som skulle passa bäst tog vi hänsyn till följande aspekter:

- Tidsaspekten, att lösa uppgiften inom 10 veckor.
- Kostnadsaspekten, att kunna realisera vårt mål med högsta kvalitet men ändå hålla nere kostnaderna.
- Funktionalitetsaspekten, att konstruera ett system som är stabilt och med komponenter som då kan ge stabila och exakta värden.
- Materielaspekten, tillgång till befintligt och användbar materiel på skolan, hårdvara samt mjukvara.

När vi utredde vilken lösning som skulle passa oss kom vi fram till att vi skulle använda oss av en mikroprocessor för att utföra beräkningen av luftdensiteten. Vi skulle tillverka ett eller flera kretskort som passade våra komponenter och det skal som vi skulle använda oss av.

2.2 Huvudsyfte

Huvudidén med projektet är att tillverka en smidig mobil mätstation som drivs av kommersiella batterier. Den blir ett enkelt mätinstrument, multimeter, som enkelt kan packas ned och förflyttas utan problem. Väl på plats skall användaren snabbt kunna mäta luftens densitet med några enkla knapptryckningar. Användaren skall inte behöva oroa sig för att prototypen kommer till skada vid användning då den är skyddad mot fukt, damm och tål mild behandling.
Vid användning av prototypen används en enkel knappsats för olika val av mätning samt meny val av vad som skall visas på displayen.

2.3 Använd mjukvara

2.3.1 Scintilla Text Editor

Till en början var Scintilla ett enkelt dokumentationsprogram som sedan utvecklades till ett smidigt program där man kan bygga, kompilera och programmera sin hårdvara. Scintilla klarar av många av dagens använda programmeringsspråk som:
C.C++ och Assembler.

2.3.2 Multisim

Kretsschemat är byggt med hjälp av programmet *Multisim.* Multisim är ett simuleringsprogram för elektronik, där man kan simulera både analoga och digitala kopplingar samt en mix av dessa. Med programmet kan man även exportera (transformera) kretsschema till programmet UltiBoard där man kan konstruera kretskort. Med programmet Multisim kan användaren bygga upp sin koppling på skärmen och testa dess funktion.

3 Teori

3.1 Sensorer

För att klara av att mäta luftdensiteten på ett bra sätt krävs att man använder sensorer som med god noggrannhet mäter sitt specifika område.

När vi valde sensorer, givare, grundade vi våra beslut enligt följande.

- Ett bra pris, som gör att de hamnar inom vår budget.
- Bra linjaritet, för bättre mätnoggrannhet.
- De ska klara av kravspecifikationen.
- Vi undersökte liknande konstruktioner för att se, vilka sensorer som använts vid liknande projekt.

3.1.1 Temperatursensor (SMT160)

Temperatursensorn Smarttemp från Smartec har följande egenskaper:

- Kräver endast en ledig digitalport på din mikroprocessor.
- Inbyggd A/D omvandlare med pulsbreddsmodulerad frekvensutgång vilket ger enkel interface mot omvärlden.
- Linjär utgång.
- Onoggrannheten i intervallet -30 till +100ºC är ± 0.5ºC.
- Matningsspänning: +4.7 till +7.0 Volt.
- Strömförbrukning: <200uA.

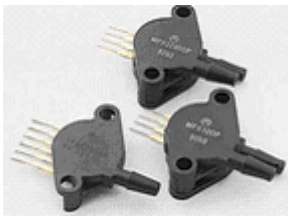
Figur 3.1.1 Temperatursensor (SMT160)

Vid mätning av temperaturen ansluts utsignalen från sensorn till en digital port på mikroprocessorn. Sensorn avger en pulsmodulerad signal som vi med processorn samplar signalen, i vårt fall 40000ggr, till en ”dutycycle”, som sedan matematiskt görs om till temperatur enligt specifikationen från sensorns datablad. Funktionen som används heter ”digital_sampling” och finns som bilaga tillsammans med resterande c-kod.

3.1.2 Trycksensor (mpx5100ap)

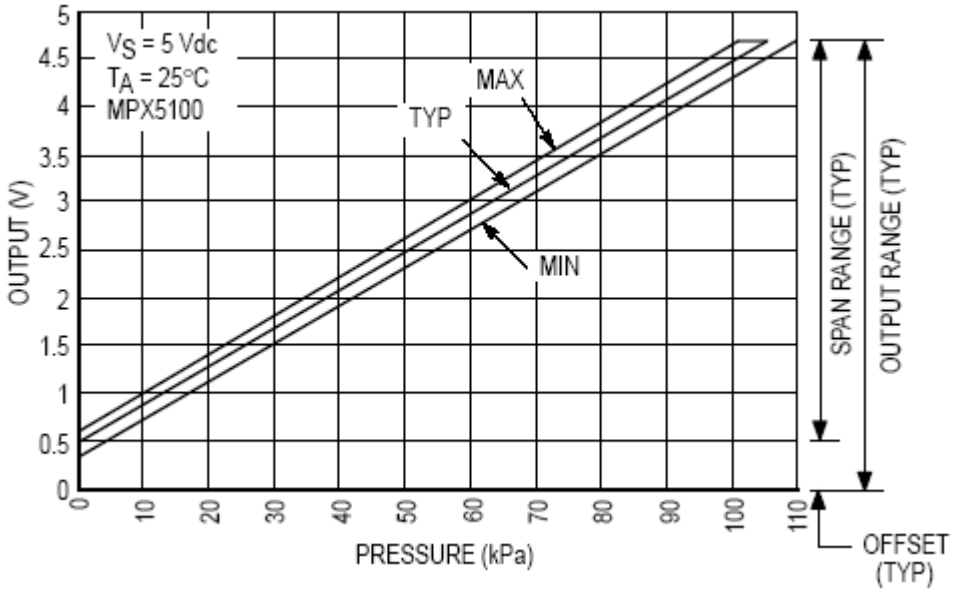
Trycksensorn mpx5100 från Motorola har följande egenskaper:

- Mäter tryckskillnad över 15 till 115kPa.
- Mätfel: ±2,5% mellan 0 till 80°C.
- Kräver endast en ledig analogport på din mikroprocessor.
- Analog utsignal mellan 0.2 till 4.7Volt (0 till 100%RH).
- Inbyggt referenstryck.
- Matningsspänning: +4.75 till +5.25Volt



Figur 3.1.2 Trycksensor (mpx5100ap)

Vid mätning med trycksensor ändvänds en analog ingång på mikroprocessorn som omvandlar likspänningar mellan 0 och 5 volt till ett heltal mellan 0 och 1023. Med andra ord är upplösningen 5/1023=4.88758mv. Komponenten är väldigt känslig gällande matningsspänning (se bild nedan) så därför har vi kompletterat just denna koppling med en mer exakt matningsspännig se avsnittet ”kretskopplingar”.



Figur 3.1.2 Matningsspänningskänslighet

3.1.3 Fuktsensor (SMTHS10)

Fuktsensorn SMTHS10 från Fabr Smartec har följande egenskaper:

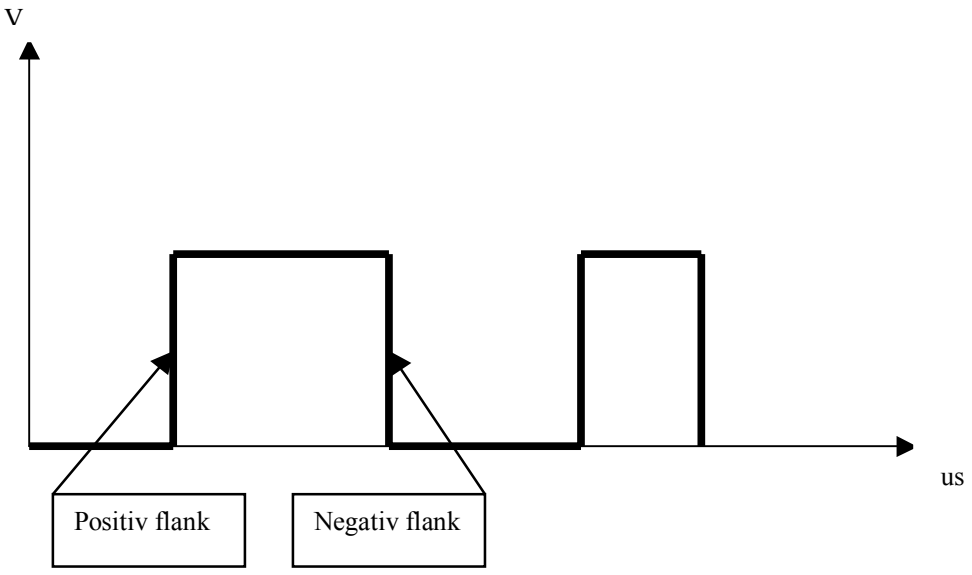
- En kapacitiv fuktsensor som klarar 100 % luftfuktighet utan att ta skada.
- Sensorn har en linjär skala från 0 till 100 %
- Spänning: 5V ~max
- Frekvensområde på 10 till 100 kHz
- Fuktområde mellan 0 och 100% RH
- Kapacitans (60%) på 240pF ±20%
- Kapacitansområde mellan 40pF ±12% (0 till 100% RH)
- Linjaritet runt ±2% RH
- Långtidsstabilitet runt ±3%RH (12mån)
- Reaktionstid på 60 sek
- Temperaturområde mellan -40 till +120ºC

Figur 3.1.3 Fuktsensor (SMTHS10)

Funktion:

En icke-ledande tunn folie förses med ett tunt guldskikt på vardera sida. Denna tunna folie utgör ett dielektrikum och guldskiktet utgör plattorna i en kondensator. Den isolerande folien fungerar som en ”svamp” och absorberar fukt. Detta leder till att ”kondensatorns” kapacitans ändras i förhållande till fukten i folieskiktet.

För att kunna mäta kapacitansförändringen används timerkretsar för att få ett referensmått där pulsens bredd förändras beroende på kapacitansen i fuktsensorn. Pulsens bredd mäts med hjälp av en timer i AvR- kretsen.



Timerfunktionen triggas då den känner av en positiv flank och en tid sparas i ett minne, samma sak händer vid den negativa flanken och en tidsskillnad uppstår. På så vis fås pulsens bredd med hög noggrannhet eftersom processorn arbetar med 8 MHz leder detta till att vi kan mäta pulsens tid ,längd, med en noggrannhet på 125ns.

http://www.tfe.umu.se/courses/elektro/elmat1/AA_DAT2002v36/Seminar%20El_Ding_2003/Gr%20(4)/Gr4_A/fuktgivare%20grupp4.doc

http://www.cyfronika.com.pl/ARCHIWUM/smths10.pdf

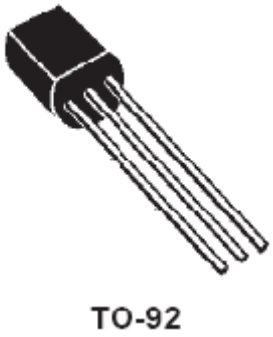
3.2 Övriga komponenter

För att kunna använda sensorerna krävdes följande komponenter som ett komplement för att få önskade utsignaler från dem samt mikrokontroller för att styra systemet.

3.2.1 Spänningsregulator 78L05

Spänningsregulatorn används främst för att trycksensorn ändrar sin utsignal betydligt om matningsspänningen ändras. För att minska felmätningar används denna spänningsregulator. Den har följande egenskaper:

- 6mA strömförbrukning (max)
- Temperatur område –30ºC - 130ºC
- Matningsspänning +4,7V - +7,0V
- Utspänningstolerans ±5%
- Spänningsfall: 1.7 Volt typ

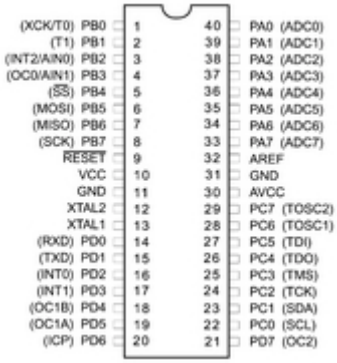


Figur 3.2.1 Spänningsregulator 78L05

3.2.2 AtmelMega16L

Mikrokontrollern från Atmel är mätsystems hjärna. Det är den som programmeras att utföra beräkningar och utföra A/D omvandling. Här är några av mikrokontrollerns egenskaper:

- 16Kbyte programmerbar FLASH minne.
- 512byte EEPROM.
- 8MHz klockfrekvens.
- 32 st. programmerbara I/O pinnar.
- A/D omvandlare



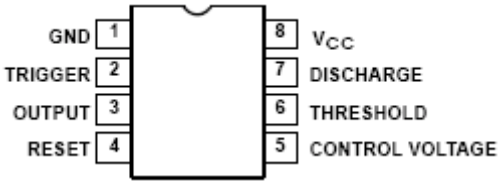
Figur 3.2.2 Atmega16L

Ovanstående egenskaper är bara en liten bråkdel av vad som finns att hämta. För alla detaljer kring mikrokontrollern hänvisar vi till databladet:
http://www.elfa.se/pdf/73/736/07367246.pdf

3.2.3 Timerkrets NE555

Vid uppbyggnaden av konstruktionen använde vi oss av timerkretsar från NEC NE555. Timerkretsen har följande egenskaper:

- Temperaturstabilitet 0.0005%/ ºC.
- Justerbar arbetscykel (dutycycle).
- Arbetsfrekvens större än 500KHz.
- Pulsrepetition från 1us till 1h.
- Pulsbredsmodulation.
- Tid fördröjnings generering.



Figur 3.2.3 Timerkrets NE555

Eftersom man inte kan mäta kapacitansförändringar med mikrokontrollern använde dessa timerkretsar då pulsens bredd ändras i samband med kapacitansförändringar. Efter det att en fyrkantspuls erhållits kan mätningar på denna utföras och konverteras, med beräkningar i processorn, till ett flyt tal och presenteras på LCD-displayen.

3.3 Formler

Vid beräkning av densitet är det den ideella gaslagen som gäller i grund och botten:

3.3.1 Den ideella gaslagen

$P*V = n*R*T$

P= Tryck
V= Volym
n= Antalet molekyler
R= Gaskonstant
T= Temperatur

3.3.2 Densitets formeln

Densitet är helt enkelt antalet molekyler på en viss volym och beskrivs enligt nedan:

$D = n / V$

D = Densitet
n = Antalet molekyler
V = volym

Eftersom luftfuktigheten spelar sin roll i mätandet av densitet måste trycket beräknas för torr respektive fuktig luft. Nedanstående ekvation realiserar detta:

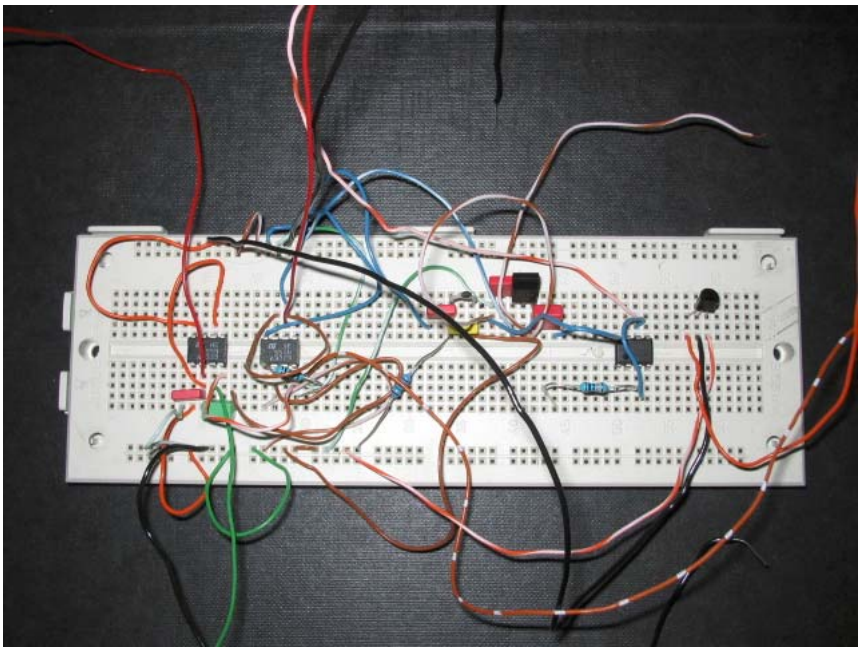
$$D = \left(\frac{P_d}{R_d * T}\right) + \left(\frac{P_v}{R_v * T}\right)$$

D = densitet, kg/m^3
P_d = Trycket för torr luft, Pascal
P_v= Trycket för fuktig luft, Pascal
Rd = Gas konstanten för torr luft, J/(kg*degK) = 287.05
Rv = Gas konstanten för fuktig luft (vaporised), J/(kg*degK) = 461.495
T = Temperatur, degK = deg C + 273.15

Utifrån ovanstående formel beräknas densiteten. En bra sida som beskriver mer detaljrikt är följande: http://wahiduddin.net/calc/density_altitude.htm

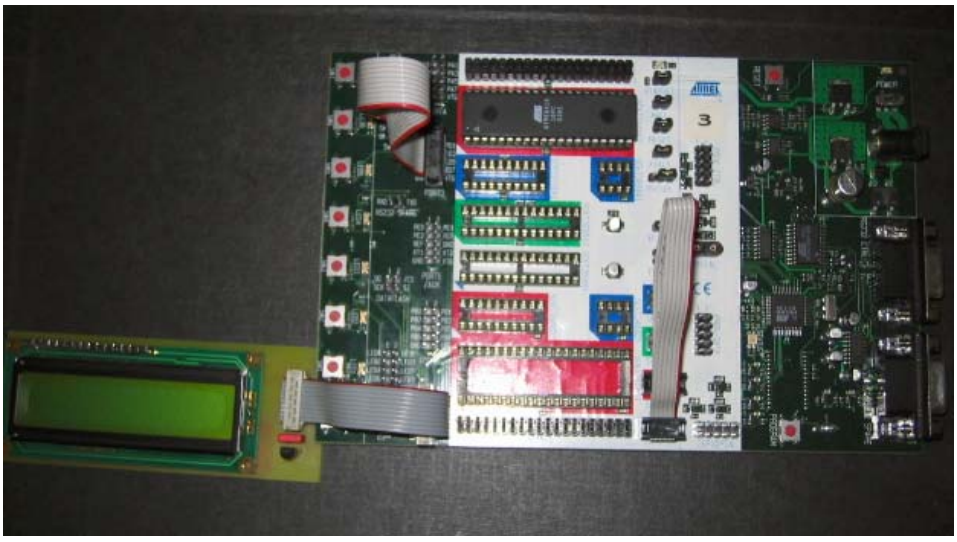
3.4 Bilder

3.4.1 Kopplingsplatta



Ovan ges exempel på hur det kunde se ut då vi testa olika kretskopplingar för att få ut så bra noggrannhet ur sensorerna som möjligt.

3.4.2 AvR-kort



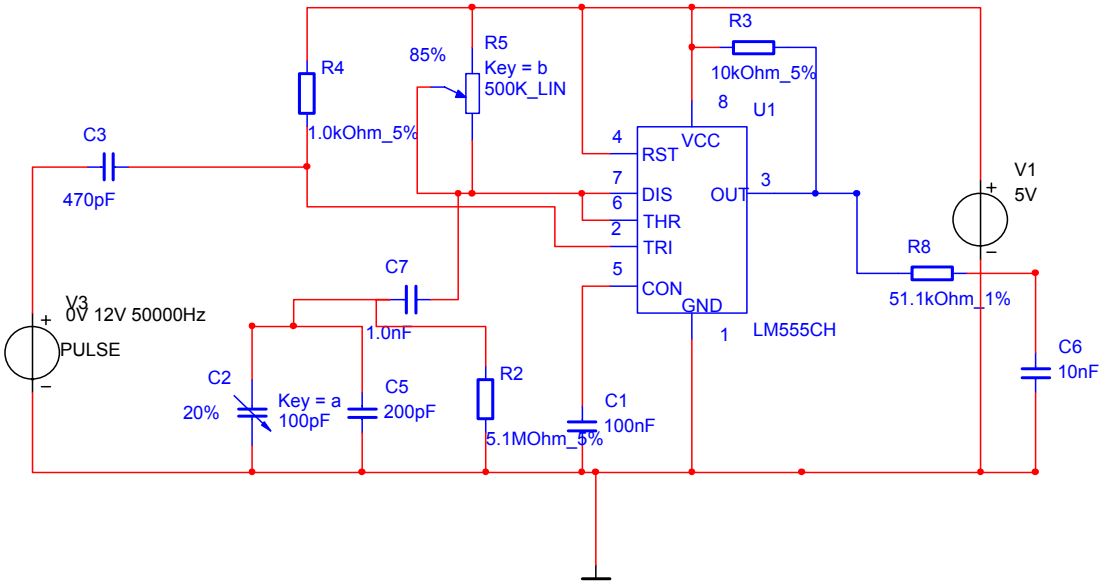
Här ser ni AvR-kortet som vi använde oss av vid programmering av processorn och display för visning av mätvärden från sensorerna.

3.5 Kretslösningar

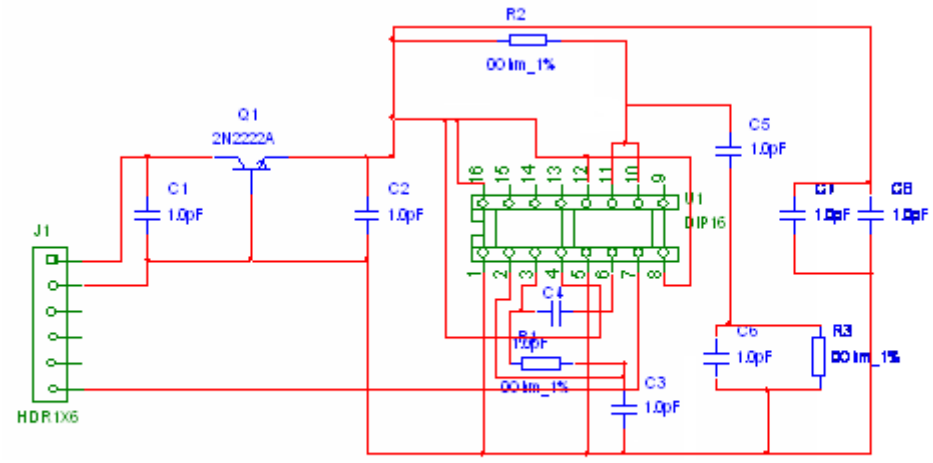
För att förenkla projektet delades vissa områden in i olika delar, därav olika kretsar som realiserar ett visst område.

3.5.1 Luftfuktighetskretsen

Nedanstående kretsschema realiserar en komplett luftfuktighetsgivare med alla dess komponenter. Komponenten C2 i vårt kretsschema realiserar fuktighetssensorn, SMTHS 10, då den avger en kapacitans. Kopplingen som tillverkades i multisim gav oss möjligheten att ändra värden på komponenterna för att få ett så bra resultat som möjligt när den skulle kopplas upp med riktiga komponenter.



3.5.2 Lufttryckskretsen (spänningsmatning)



Vår trycksensor var som tidigare nämnt väldigt känslig när det gällde matningsspänning. Vi tillverkade då en krets för att stabilisera matningsspänningen så att mätvärdena skulle stabiliseras och felmarginalen minska. Nedan ses ett förslag till hur kretslösningen ser ut.

3.6 Noggrannhet

En undersökning kring noggrannheten vid mätning av luftens densitet tillämpades för att kunna se hur noggrant mätningen kunde utföras. Vid beräkningen av det totala felvärdet användes formler från avsnittet formler (kap 3.3).

3.6.1 Maximalt felvärde

Då formeln för luftens densitet ser ut enligt följande:

$$D = \left(\frac{P_d}{R_d * T}\right) + \left(\frac{P_v}{R_v * T}\right)$$

Man kan konstatera att ett max-fel uppstår när temperaturens och fuktens värde är så litet som möjligt, samtidigt som trycket är stort.
Min-fel uppstår då temperaturgivaren och fuktighetsgivaren ger ifrån sig stora värden, samtidigt som trycksensorn indikerar ett för litet värde.

Vi antar att följande värden stämmer:
Tryck: 100kPa
Fukt:40% RH
Temperatur:20 ºC

4 Genomförande

4.1 Arbetsgång

Vi började vårt arbete med att ta reda på vad luftdensitet egentligen är och vilka faktorer som spelar in. När vi sedan fått klart för oss att vi behövde temperatur, lufttryck och luftfuktighet för att kunna räkna ut luftdensiteten, bestämde vi oss inom kort för att koppla dessa givare samman i en AVR-processor.

I uppgiften ingick även att ta reda på hur stor noggrannhet vi kunde få på vår luftdensitetsmätare inom den budgeten vi var begränsade till. För att ta reda på detta sökte vi på Internet för att hitta bättre sensorer än de som skolan hade att erbjuda. Vi fann en fuktsensor, HS1101, som hade mycket hög noggrannhet och mycket litet avvikande fel. Denna var vi dock tvungna att beställa från Tyskland och leveranstiden var på över två veckor, dessutom hade den en aning för högt pris för vad vår budget klarade av. Vi hittade även en passande trycksensor, SM565. Då vi istället fick trycksensorn mpx5100ap från skolan som vi då trodde var bättre[1] använde vi oss av den. Vi funderade även på att bygga ihop fuktgivaren med kretsen DS1616 som hade en inbyggd temperatursensor, men eftersom detta både skulle bli krångligare och överflödigt avvisade vi även detta förslag.

Det slutgiltiga förslaget bestod nu av tre separata sensorer. En temperatursensor med bra noggrannhet och som dessutom är billig, en trycksensor med ett inre referenstryck och en fuktsensor med mycket goda egenskaper.

När vi fått vår AVR-rack provade vi först med några enklare program för att försöka hitta in- och utgångar på processorns portar och att samtidigt lära oss mer om AVR-processorn. Vårt nästa mål var att implementera funktioner för att använda oss av temperatursensorn som vi skulle använda oss av. Temperatursensorn SMTHS10 fick vi att fungera relativt snabbt då det fanns färdiga funktioner och klasser att använda.
När vi kände att kunskapen om vår AVR –krets var tillräckliga påbörjade vi att implementera funktioner för resterande sensorer och uppbyggnaden av ett program som kunde avläsa de olika sensorerna och ta fram luftdensiteten. Allt eftersom arbetet med prototypen fortlöpte utvecklades mjukvaran för att tillfredsställa vår hårdvarukonfigurationen. Arbetet med att få fram högsta möjliga noggrannhet blev mycket tidskrävande då vi var tvungna att komma på egna sätt att använda sensorerna på. Vi studerade många kretslösningar för att få fram så mycket som möjligt från våra sensorer. När vi hittat en lämplig kretslösning återstod programmeringen till processorn för att få fram önskvärd data[2]. Vi gjorde även en kalibrering av sensorerna för temperatur och för fuktighet. För att kalibrera trycksensorn behövdes avancerad utrustning som vår budget inte klarade av. För att testa temperatursensorn använde vi oss av en digital termometer som mätte temperatur med tillräckligt god noggrannhet. För fuktsensorn använde vi oss av kemikalier för att skapa inkapslade miljöer med en exakt luftdensitet.
När detta var avklarat påbörjade vi utveckling av vårt program för att få det driftsäkert och lätt att utveckla vid behov.

4.2 Programuppbyggnad

Huvudprogrammet heter Maektuub V1.0 och är uppbyggt med objektorienterad programmering i Scintilla Text Editor (C++). Programmet fungerar som en tillståndsmaskin där användaren väljer vilken information som skall visas på LCD-displayen genom knapptryckning på AVR-kretsen. Följande tillstånd finns i vårt program:

- Temperaturmätning
- Tryckmätning
- Luftfuktighetsmätning
- Densitet

[1] Detta märktes senare att det inte riktigt stämde.
[2] Mer om detta i Programuppbyggnad 4.2.

4.2.1 Maektuub V1.0 Main.c

Nedan kommer vi att gå igenom programmet del för del med programmeringskod och förklaringar. Main.c fungerar som en oändlig loop som endast ändrar tillstånd då användaren kräver det genom att trycka på en knapp. Nedan visar vi kort hur själva programmet är uppbyggt i c-kod.

```
Interupt
{
input capture
};

While(1)
{
CASE 1
{
sensor_smt160_read(&theSensor,40000,type);                  //TEMPERATUR
lcd4
}
CASE 2
{
sensor_mpx5100_read(&theSensor1);                           //LUFTTRYCK
lcd4
}
CASE 3
{
signal(sig_input_capture1) + smths10                        //LUFTFUKTIGHET
lcd4
}
CASE 4
{
sensor_smt160_read(&theSensor,40000,type);                  //BERÄKNING
sensor_mpx5100_read(&theSensor1);                             AV LUFTDENSITET
signal(sig_input_capture1) + smths10
airdens_calc
lcd4
}
};
```

I stort kan vi genom den enkla kodbeskrivningen förklara programmets uppbyggnad.
Då vi startar vårt program sker, som vanligt, initiering av funktioner, variabler, objekt och konstanter. Programmet är det kodat så att vi hamnar i den första case satsen, CASE 1, av fyra st.

4.2.2 CASE 1 Temperaturmätning

I denna sats mäter vi temperaturen med vår temperatursensor smt160 genom att ropa på funktionen sensor_smt160_read. För att få en större översikt vad som händer när funktionen körs visar vi nedan delar av källkoden för denna funktion.

```
float sensor_smt160_read(sensor_smt160 *this,double n,char type)
{
this->type=type;
this->n=n;
this->duty_cycle=digital_sampling(this->port,this->bit_mask,this->n);
this->temp= (float) this->duty_cycle/(float)this->n;
this->temp=(((this->temp-0.32)/0.0047));
return this->temp;
 }
```

Som vi tidigare nämnt ger denna sensor en pulsbreddsmodulerad signal på utgången. I funktionen ser vi att ”duty_cycle” får värdet av ”digital_sampling” som är pulsbreddsmoduleringen. När vi fått sensorns ”duty_cycle”, omvandlar vi den med formler från sensorspecifikationen för att få ett flyttal som presenterar temperaturen. Detta flyttal är det som returneras från funktionen. Istället för att skriva 30 rader kod kan vi nu förenklat skriva en rad i main , **sensor_smt160_read(&theSensor,40000,type);**, och på så vis få ett snyggare och mer strukturerat program.

4.2.3 CASE 2 Lufttrycksmätning

Ungefär samma tillvägagångssätt här som i föregående case sats. Vi ropar på funktionen sensor_mpx5100 och följande kod körs.

```
io_ai ai1;
float sensor_mpx5100_read(sensor_mpx5100 *this)
{
  this->heltal=io_ai_read(&ai1);
  this->Pa=(this->heltal*0.00488758)/0.04086956;
  return this->Pa;
}
```

Denna sensor returnerar en analog signal mellan 0,2V-4,7V beroende på trycket , 15-115kPa . För att kunna omvandla den signalen till ett flyttal som vi kan använda i våra beräkningar använder vi oss av en analog ingång med 10 bitars upplösning. Efter att insignalens magnitud avlästs på den analoga ingången till processorn beräknar vi exakta antalet kPa som denna insignal motsvarar och returnerar värdet till main funktionen.

4.2.3 CASE 3 Luftfuktighetsmätning

I detta tillstånd mäter vi luftfuktigheten. Då vår konstruktion med luftfuktighetsmätaren ger en variabel pulsbredd på utgången som är ytterst liten, kom vi fram till att om vi ville ha den bästa noggrannheten var vi tvungna att använda oss av processorns ”input capture unit”. En trigger i processorn som känner av positiv respektive negativ flank på en puls. Genom att mäta tiden mellan positiv och negativ flank med klockfrekvensen 16MHz i processorn skulle vi komma så nära målet som möjligt. För att använda ”input capture unit” var vi tvungna att använda oss av specifika kommandon och bygga en avbrottsrutin, som vi kommer att få se nedan.

asm(" SEI ");

Detta kommande betyder att globala avbrott tillåts och processorn hoppar nu till avbrottsrutinen då den känner en positiv flank på ICP, Input Capture Pin. I avbrottsrutinen nedan mäter vi sedan tiden mellan positiv flank och negativ genom att läsa av ICR1, Input Capture Register, och tilldelar en variabel värdet innan vi byter flank och läser av den negativa flankens tidsvärde. För att åstadkomma tidsmätningen kräver programmeringen att en hel del register tilldelas värden. För att visa detta på ett enkelt sätt åskådliggör vi programmkoden för avbrottsrutinen nedan och kommenterar den.

```
SIGNAL(SIG_INPUT_CAPTURE1)
{
if ( bTCCR1B.ices1 )
{
hi=ICR1
bTCCR1B.ices1 = 0;
}
else
{
lo=ICR1;
bTCCR1B.ices1 = 1;
bTIMSK.ticie1 = 0;
}
```

Ovan ser vi en if/else sats, vi börjar med if delen. Om **bTCCR1B.ices1** har värdet ”1” känner ICP, input capture pin, av positiva flanker och om den har värdet ”0” negativa. Vi antar att den har värdet ”1”. Nu kommer variabeln hi att tilldelas tidsvärdet som sparas i ICR-registret. Efter detta byter vi nu till avläsning av den negativa flanken genom att tilldela TCCR1B-registret värdet ”0”, **bTCCR1B.ices1 = 0;**. Den negativa flankens tidsvärde kommer nu att sparas i ICR1 registret och vi har vår tidsskillnad.
Vi ger **TIMSK.ticie1** värdet 0, vilket resulterar i att avbrott inte längre tillåts. Med några enkla else/if satser räknar vi sedan ut tidsskillnaden och då vi vet pulsens bredd mätt i tid kan vi räkna ut motsvarande luftfuktighet.

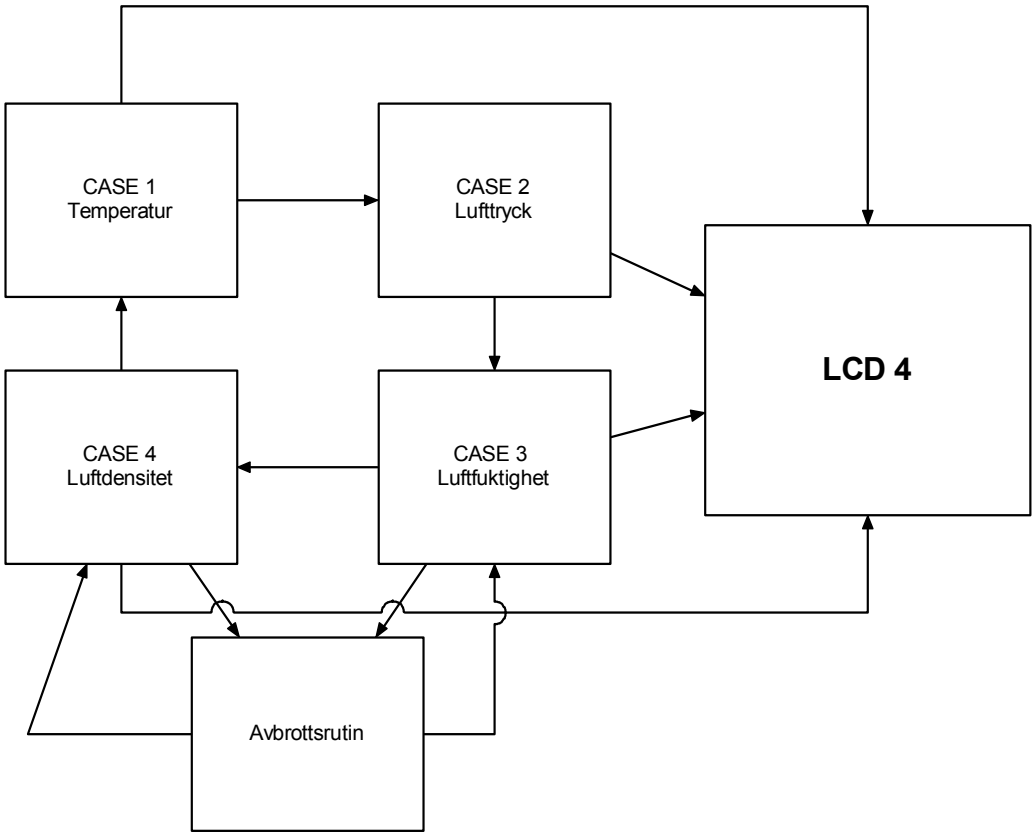
4.2.4 CASE 4 Luftdensitetsberäkning

I detta tillstånd använder vi oss av alla ovanstående funktioner för att få värden på temperatur, lufttryck och luftfuktighet för att kunna utföra luftdensitetsberäkningen. Det enda som är nytt för detta tillstånd är beräkningsdelen. För denna beräkning hänvisar vi till avsnittet ”FORMLER”.

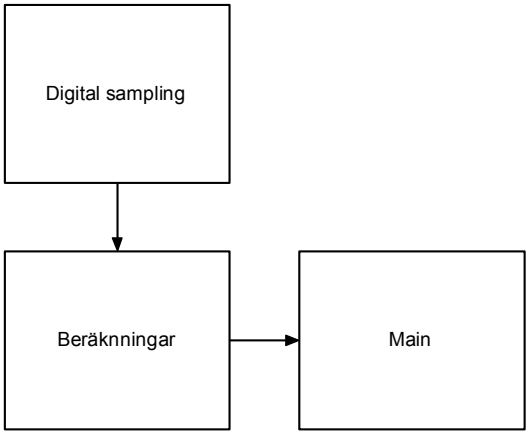
4.3 Flödesscheman

Nedanstående flödesscheman ger en grafisk inblick i hur de olika programmeringsmodulerna är uppbyggda.

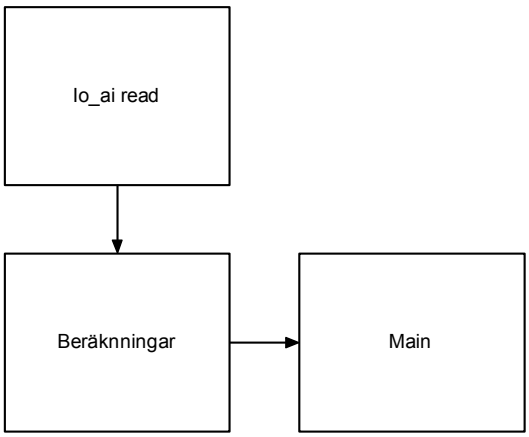
4.3.1 Maektuub V1.0 (main.c)



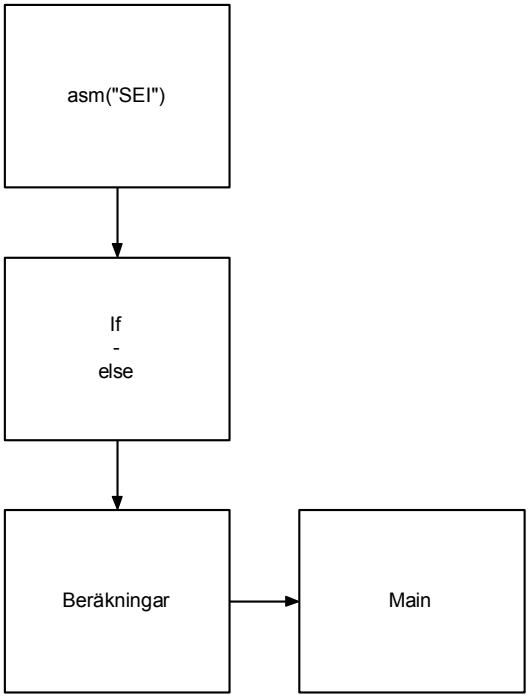
4.3.2 CASE 1 Temperatur Sensor_smt160.c



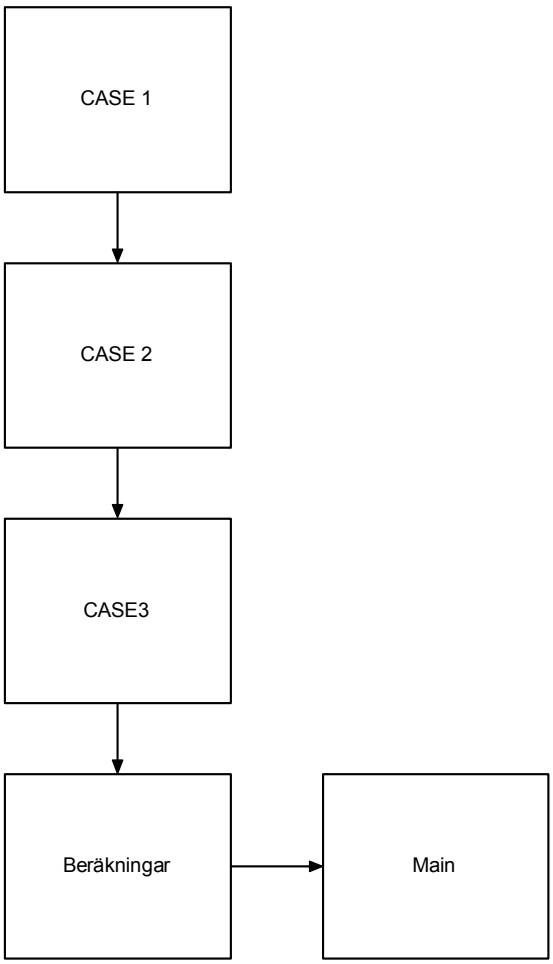
4.3.3 CASE 2 Sensor_mpx5100.c



4.3.4 CASE 3 Sensor_smths10.c



4.3.5 CASE 4 Luftdensitet



5.1 Resultat

5.1 Resultat från mätinstrumentet

Från prototypen får vi ut temperatur med noggrannheten < ± 0.5ºC, fukt med noggrannheten ±0,13% samt lufttryck med en noggrannhet på 2,6kPa. Det totala felet för beräkningen för luftdensiteten blir då ±0,0666kg/m^3

Menyn är enkel att använda sig av och värdena kan avläsas kontinuerligt.
Då detta fortfarande är en prototyp så uppfyller den än inte alla de krav som var satta från början.
Prototypen klarar inte det krav för utomhusbruk då den inte fått någon inkapsling. Prototypen har möjlighet att köras på batterier men för att mätinstrumentet ska komma till sin fulla rätt krävs det att den blir inkapslad och portabel.

5.2 Kalibrering

Vid kalibrering finns det olika tillvägagångssätt. Vi valde metod utgående från vår budget. En bra och mycket noggrann metod är att installera sensorn i en klimatanläggning som håller exakt temperatur, luftfuktighet och lufttryck för att kalibrera sensorn. Detta är dock en dyr variant. Vi använde oss istället av olika saltlösningar för att kalibrera fuktsensorn. Eftersom vår prototyp är kopplat mha. kopplingsplatta så förändrades kapacitansen på fukthetgivaren från dag till dag då vi testade kopplingen. Detta beror på krypkapacitanser i anslutande koppling.
Vid kalibrering av temperatursensorn använde vi oss av en noggrann termometer för att en bra kalibrering och ett litet motstånd som vi tillsatte spänning för att alstra värme. För kalibrering av tryck återstår dock problemet med att hitta en billig och relativ enkel metod

Fuktsensorn smths10

Det är sen tidigare känt att om fuktsensorn inkapslas ovanför en lösning av Magnesium Nitrat vid temperaturen 20 grader Celsius erhålles luftfuktigheten 55% RH. Vid användande av Lithium Klorid vid samma temperatur erhålles 12% RH. Vid användande av Natrium klorid erhålles 75.5 % RH. Ovanstående metod är endast beroende av temperaturen.

Tempsensor smt160

Temperatursensorn vet vi sen tidigare har en noggrannhet i intervallet -30 till +100ºC på ± 0.5ºC. Detta testade vi genom att använda oss av en noggrann digital termometer kopplad till samma motstånd som den sensor vi använde oss av.

5.3 Slutsatts

Som vi märker är det lufttrycksensorn som drar ner noggrannheten. Anledningen till detta beror på att det totala felet från sensorn är hela ±2,5% av Fullscale (0,2 – 4,7V). Detta betyder att felet på bara tryckgivaren blir upp till hela ±2,6 kPa. Det totala felet för uträkningen för luftdensiteten beror därför till stor del av lufttrycksensorns höga felprocent.

Att använda sig av AVR-processorn var betydligt svårare och mer tidskrävande än vi först trodde. Att finna bra funktioner och smarta formler för att använda oss av de sensorer vi valt att använda oss av visade sig vara den tyngsta delen av detta arbete. Det svåra var att finna bra lösningar till hur man på ett bra och noggrant sätt överför data från sensorerna till processorn. Flera problem har vi stött på som tex. hur många volt som processorn kan känna av, hur ofta processorn kan känna av en signal, storleken och noggrannhet på insignaler i form av byte, tid mellan positiv flank till en negativ osv.

5.1 Diskussion

Varför fick vi inte färdig vår prototyp med de krav som var ställda från början? Finns det saker vi kunde ha gjort bättre för att fått bättre förutsättningar att bli klara med prototypen? Vi finner att bättre undersökning borde ha gjorts innan vi bestämde oss för att använda den trycksensor vi valde. Spänningsmatningen till sensorn var tvungen att vara exakt stabil. Om spänningen ändrade sig, om bara minimalt, ändrade sig utsignalen från den rejält. Spänningsstabilisatorn löste problemet och vi kunde sedan fortsätta arbetet med de andra sensorerna.
I efterhand kan vi konstatera att programmeringen och forskningen runt AVR-processorn blev en alldeles för stor och tidskrävande del. Att finna en lösning där noggrannheten på densitetsmätaren ställdes på fokus blev ett allt för stort jobb. Ett alternativ skulle ha varit att densitetsmätaren byggts med enklare komponenter och att i efterhand undersökt om några justeringar till förbättring kunnat ha gjorts. Att bara kasta sig över AVR-processorn var kanske ett annat fel vi gjorde. Kanske fanns det andra processorer vi kunnat ha använt oss av? Problemet och målsättningen för oss blev att hitta den bästa lösningen för att få så låga felprocent från sensorerna. Vi hade sparat mycket tid om vi bara hade valt den enklaste metoden för att lösa uppgiften. Men det gjorde vi inte. Men tack vare det har vi fått en djup inblick i hur det är att arbeta med sensorer. En sensor ska ge oss svar på det vi söker efter, vare sig det är ljus, tryck, vibrationer eller fukt. Att kunna återspegla verkligheten med hjälp av sensorer är mycket svårt. Det sensorerna ger ifrån sig är spänningsförändringar eller kanske pulsförändringar beroende på ur man vill behandla dem. Att finna den bästa lösningen är mycket svår om inte omöjlig. Men då vi hade en budget att hålla oss till så blev den så klart begränsad. Trots detta så fanns det otaliga metoder att använda sig av då det gällde AVR-processorn.

5.2 Förbättringar

En förbättring till problemet med mätnoggrannheten är att använda sig av bättre sensorer helt enkelt. Men då till en större kostnad. Det finns även bättre processorer med högre upplösning på portarna som skulle leda till bättre mätnoggrannhet.

Att löda fast alla komponenter på ett kretskort och stabilisera alla sensorer från stötar som förändrar dess egenskaper skulle öka tillförlitligheten avsevärt. Att använda kortare och avskärmade ledare till och från sensorerna skulle ytterligare öka tillförlitligheten. Alla dessa små detaljer skulle i stort leda till förbättring.

För att associera med det prototypen slutligen ska användas till, nämligen förbättra förbränningen i en go-cart, skulle man även kunna tänka sig att mäta ytterligare en faktor nämligen mängden luft som kommer in till förbränning genom förgasaren. Det finns flera sätt att mäta detta, några mer noggranna än andra. Ett enkelt sätt skulle vara att med hjälp av ett motstånd. Motståndet varierar sin resistans med temperaturen och kan med hjälp av det få ett mått på hur snabbt go-carten färdas. Om man vet hastigheten kan man räkna ut den mängd luft som fångas upp av insuget till förgasaren.
Om man så vill skulle man kunna styra förgasaren eller valfri komponent, för att förbättra dess prestanda.

6 Referenser

http://www.engineersedge.com/basic_conversions.htm
www.elfa.se
www.oru.se

Faktasökning :
AVR-programmering Kell Mårdensjö, 2004.
Tips och idéer om kretslösningar mm Torbjörn Andersson, 2004.