LUNDS TEKNISKA HÖGSKOLA
Lunds universitet

WIFS

-Wireless Flexometer System
ett digitalt projekt under VT 2005

Utfört vid institutionen för Informationsteknologi, LTH

Handledare: Bertil Lindvall

Teknologer: Thomas Ahlström - E01
Peter Bengtsson - E01
Johan Henricson - E02

Gruppnr.: 10

Datum: 2005-03-01

Abstract

This is a report from a project carried out at the Lund Institute of Technology in Sweden during the spring of 2005. The objective of the project is to design a standard (named WIFS) for measuring, transmitting and displaying a variety of weather-parameters such as temperatures, air pressure and wind strength and to illustrate it by designing a couple of units using it.
The standard's basic thought is to locate all of the expensive intelligence inside a main central unit into wich up to eight sensors may be plugged in. The central unit do all of the AD-conversion, signal processing and data packaging needed to wirelessly broadcast the information to any number of more or less advanced recievers. Each reciever may display (or log) one or more of the parameters (chosen by the reciever designer).

To illustrate the standard two sensors, one central unit and one display unit is constructed. The central, and display units are based on the Atmel AVRMega16 microprocessor (programmed with the AVR-gcc C-compiler). The different parts of the project are described and explained in general and all source code, functions etc. can be found as appendix.
The report is entirely in Swedish.

Keywords

WIFS, Weather-station, Sensor-design, Wireless, LCD, AVR, C

Innehållsförteckning

1. Inledning

1.1 Projektets omfattning

Detta projekt har utförts vid Lunds Tekniska Högskola under våren 2005.
Syftet med det har varit att, främst i inlärande syfte, designa en standard för att mäta ett antal specificerade väderparametrar, samt att trådlöst överföra dessa till ett valfritt antal mottagare (broadcast). Två mätare, en centralenhet och en mottagarenhet för visning av data från sändarna konstruerades för att testa, illustrera och verifiera standarden. Projektet har löpt under en läsperiod (sju veckor) och motsvarar fem akademiska poäng.

1.2 Idé och grundtanke.

Tanken är att konstruera en central för väderobservationer av olika sort.
Basenheten, centralen, har åtta ingångar av lämplig, väl specificerad, art. På dessa ingångar kopplas termometrar, hygrometrar, barometrar m.m. Maximalt åtta enheter kan anslutas men det finns inget krav på minsta antal.
Signalerna från dessa sensorer läses, omvandlas, behandlas, och sätts samman till lämpliga datapaket som trådlöst skickas ut till en eller flera mottagare.

Tanken är att skapa en form av standardenhet till vilken man kan ansluta den typ och mängd av enheter man själv behöver, såväl mätare som visare - don efter person, alltså!
Poängen är att en konsument skaffar en ”central” och sen kan bygga på med det antal termometrar, barometrar och vindmätare han anser sig behöva. Då resultatet av mätningarna skickas ut till vem som behagar lyssna (broadcast) kan även ”visarna” införskaffas efter tycke och smak. Man kunde tänka sig att en konsument vill mäta temperaturen vid solstolen i trädgården och få den presenterad såväl i köket som på nattduksbordet. På nattduksbordet vill han dock även kunna se luftfuktigheten och vad det kan tänkas bli för väder idag.
Alltså skaffar han en central, en termometerenhet, en fuktmätare, en tempvisare och en temp/fuktighetsvisare.
Så länge man har sin central är möjligheterna i princip oändliga; man kan tänka sig specialdesignade termometrar (visare) av kända toppdesigners, termometrar som visar max/min-värden, presentation av data på LCD-displayer eller analoga instrument. Eller varför inte logga väderdata i sin PC med hjälp av en speciell USB/RS-232-mottagare för att skapa statistik och rita grafer! Kanske en sändare man kan skicka SMS till för att få aktuella mätarställningar tillbaka.

1.3 Realisering

Tanken är att givarenheterna (termometrarna etc.) är trådbundet kopplade till centralen. Signalerna ligger mellan 0 och 5V, vilket alltså ska motsvara ett lagom mätintervall.
Centralen A/D-omvandlar insignalerna, gör nödvändig behandling, sätter ihop allt till ett datapaket och sänder via ett trådlöst interface ut detta paket. Varje paket förses med kontrollbitar för att kunna identifiera sig och intyga att det kommer fram i samma skicka som det utsändes.
Avläsning, behandling, paketering och sändning görs regelbundet och med ganska korta intervall varför informationen bör komma fram förr eller senare även om en hel del paket kan komma att försvinna på vägen. Väderparametrar varierar överlag ganska långsamt och inte ens ett ganska stort antal försvunna paket bör kunna störa pålitligheten.
Mottagarenheten ska ta emot signalen, ta vara på den data den vill och kan visa, omvandla denna och presentera den på önskat sätt. Genom denna valfrihet kan givare och visare göras precis så billiga eller dyra som önskas vilket ger större valfrihet åt såväl tillverkare som konsumenter.

A/D-omvandlingen, den seriella kommunikationen med de trådlösa enheterna och interfacet mot displayer etc. samt det cykliska förloppet (vilket innebär en förhållandevis kort kod) gör att valet faller på AVRMega16-processorer i central och visare.

1.4 Projektets beståndsdelar

Tre stora teoretiska huvudområden framträder i:

a) insamling/mottagning/behandling av data (givare→central),
b) behandling/paketering/utsändande av information (central→visare) och
c) mottagning/behandling/presentation av information (visare→användare).

Rent fysiskt faller det sig dock mer naturligt att dela in konstruktionen i mätgivare, centralenhet, mottagarenhet och överföring mellan enheter. Det är också så arbetsgången läggs upp och därför har även denna rapport orienterats efter denna struktur.

2. Mätgivarna

2.1 Allmänt om mätgivare

För att kunna mäta storheter som temperatur och tryck måste någon typ av elektronisk komponent, som omvandlar en icke-elektronisk storhet till en elektronisk, användas. Idag finns det både analoga och digitala sensor i alla möjliga prisklasser att tillgå, men för att kunna utnyttja AVR:ns inbygga AD-omvandlar och 8-kannals multiplexer har uteslutande analoga sensorer använts. De flesta mätgivare avger mycket svaga signaler som måste förstärkas och anpassas till AD-omvandlarens dynamiska område. I vårt fall har AD-omvandlaren ett dynamisk område på 5 V och det utvalda mätområdet måste därför ligga inom detta.

Många givares utsignal består av en spänning som är proportionell mot den storhet som mäts. Att överföra en sådan signal långa avstånd innebär många nackdelar. Dels har man spänningsfall i ledningarna, dels kommer man, beroende i vilken miljö man mäter, få problem med störningar. Spänningsfallet går att kompensera för, men störningar är svårare att helt utestänga. Skärmade eller partvinnade ledningar ger ett skydd, men detta är oftast inte tillräckligt. Ett sätt att komma runt detta är att istället för att använda sig av en spänning som informationsbärare använda sig av en ström som inte är lika känslig för störningar.

Konstruktion innehåller en innetermometer och en tryckgivare som kommer att beskrivas mer utförligt nedan. Båda dessa storheter mäts inomhus och på kort avstånd från AD-omvandlaren så därför har någon signalomformare (transmitter) inte konstruerats.

2.2 Temperatur

Som temperatursensor används en LM35, vars utspänning är direkt proportionell mot temperaturen i Celsius (10mV / ºC). Mätområdet bestämdes i kravspecifikationen (0 ºC – 40 ºC) och som förstärkande element används en icke-inverterande förstärkarkoppling för att anpassa signalerna till AD-omvandlarens dynamiska område.

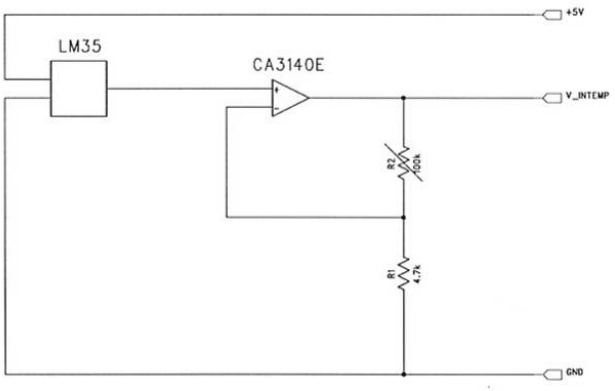


Fig 2.1; *Kopplingsschema över temperatursensorn.*

För en icke-inverterande förstärkare gäller följande enkla samband:

$$V_{INTEMP} = A \cdot V_{sensor} \quad \text{(ekv. 2.1)}$$

$$A = \left(1 + \frac{R_2}{R_1}\right) \quad \text{(ekv. 2.2)}$$

Förstärkningen (A) kan sedan bestämmas genom att anpassa signalnivåerna.

Temperatur / ºC	V_{sensor} **/ mV**	V_{INTEMP} **/ V**
0	0	0
40 ºC	400	5

Tavell 2.1; *Tabellen visar sensorns (LM35) utspänning och spänningen ut från operationsförstärkaren vid de båda gränstemperaturerna.*

$$A = \frac{5}{0.400} = 12{,}5 ggr \quad \text{(ekv. 2.3)}$$

Genom att välja R_2 till en trimpotentiometer kan förstärkningen trimmas in exakt.

2.3 Tryck

Precis som temperatursensorn har tryckgivaren (MPX5100AP) en utspänning som är proportionell mot den mätande storheten. Trycket mäts absolut och sensorn klara med en felmarginal på ±2,5 kPa av att mäta tryck mellan 15 och 115 kPa. Ett så stort mätområde är inte nödvändigt vid mätning av lufttryck som normalt ligger någonstans mellan 90 – 110 kPa utan nollnivån ”lyfts” här upp för att ligga på 90 kPa. Att använda samma konstruktion som för temperatursensorn fungerar därför inte, utan en mer sofistikerad lösning måste konstrueras. Figur 2.2 visar hur detta har lösts genom att använda en instrumentförstärkare för att kunna subtrahera bort en referensspänning motsvarande 90 kPa och sedan förstärka signalen för att anpassa den till AD-omvandlaren.

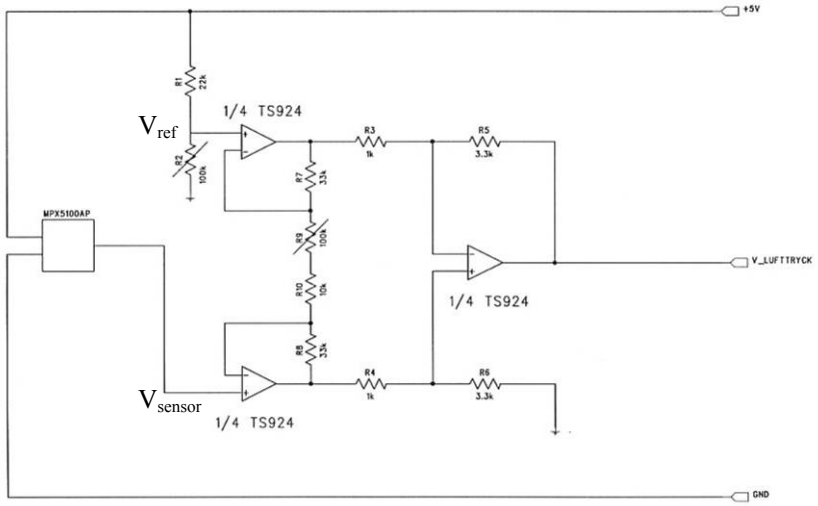


Fig2.2; *Kopplingsschema över den instrumentförstärkare som används vid mätning av tryck.*

Spänningen ut från tryckgivaren (MPX5100AP) ges av följande samband:

$V_{sensor} = V_{cc}\left(P \cdot 0{,}009 - 0{,}095\right)$, där P mäts i kPa. (ekv. 2.4)

En instrumentförstärkare mäter differentiellt och om $R_3 = R_4$, $R_6 = R_5$ och $R_7 = R_8$ kan ekvationerna nedan ställas upp. Förstärkningen kan sedan bestämmas med hjälp av tabell 2.2.

$$V_{TRYCK} = A(V_{sensor} - V_{ref}) \quad \text{(ekv. 2.5)}$$

$$A = \left(1 + 2\frac{R_7}{R_9 + R_{10}}\right)\left(\frac{R_5}{R_3}\right) \quad \text{(ekv. 2.6)}$$

Tryck / kPa	V_{sensor} / V	V_{TRYCK} / V
90	3,57	0
110	*4,70*	5

Tabell 2.2; *Spänningarna anpassas för att utnyttja hela AD-omvandlarens dynamiska område. Värden för V_{sensor} är beräknade med V_{CC} = 5,25V.*

$$A = \frac{V_{TRYCK}}{V_{sensor} - V_{ref}} = \frac{5}{4{,}70 - 3{,}57} = 4{,}42 ggr \quad \text{(ekv. 2.7)}$$

Återigen har en trimpotentiometer använts för att kunna trimma in förstärkningen exakt, dessutom ligger en resistans (R_{10}) i serie med trimpotentiometern för att hindra att resistansen vrids ner för lågt. För att ställa in rätt referensspänning används ytterligare en trimpotentiometer.

3. Centralenheten

3.1 Funktionsbeskrivning

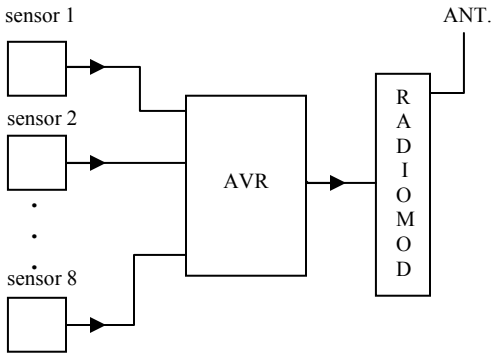


Fig 3.1; *Blockschema över centralenheten.*

Centralenhetens funktion är mycket enkel. Beroende på om en sensor är inkopplad eller inte så ska ett värde läsas in till AD-omvandlaren för att sedan direkt skickas iväg, via uarten, till radiomodulen. Därefter kontrolleras nästa ingång till AD-omvandlaren och när alla inkopplade sensorer har gåtts igenom och skickats så väntar centralenheten i 3 sekunder för att sedan upprepa samma procedur igen.Genom att skicka data ofta minskar risken att mottagarenheten blir utan paket under en längre tid och därmed kanske visar felaktig information.

Centralenhetens hjärta består av en AVR (ATmega16) från Atmel som innehåller både AD-omvandlare och uart, därför är radiomodulen, förutom sensorerna, den enda externa komponent som anslutits. Vilken av sensorerna som ska vara ansluten till AD-omvandlaren bestäms av en 8-kanals multiplexer som också den sitter internt i AVR:n. AD-omvandlaren har en 10-bitars upplösning, men endast 8 bitar utnyttjas för att kunna få plats med all data i två datapaket. Datapaketen kommer att beskrivas utförligare i kap. 4, överföringen.

3.2 Användning av radiomoduler vid sändning

Att skicka information trådlöst var för bara några år sedan en ren omöjlighet för icke-radiotekniker. Idag, tack vare radiomodulernas frammarsch på marknaden, har detta blivit relativt enkelt. Radiomodulerna arbetar på det fria frekvensbandet 433 MHz, använder sig av ASK som moduleringsteknik och har en maximal överföringshastighet på 100bps.

AVR:n innehåller, som tidigare nämnts, en uart (Universial Asynchronous serial Receiver and Transmitter) vilket gör det enkelt att skicka och ta emot data seriellt. Hur man skickar data (formatet) och med vilken hastighet (baudrate) sätts i uartens kontrollregister. När ingen data skickas ligger uart:ens TxD-utgång hög för att markera att ingen kommunikation sker. Detta fungerar dåligt tillsammans med radiomodulerna och därför inverteras signalen. I figur 3.2 kan ett typiskt datapaket ut från uart:en studeras. Eftersom detta paket inverteras måste det på mottagarsidan inverteras tillbaka för att kunna tolkas rätt av mottagarenhetens uart. Hur detta fungerar beskrivs utförligare i kapitel 5.4.

Förutom den digitala ingången och matningsspänning så återfinns endast en antenningång på sändarmodulen. ¼-vågsantenn, som i detta fall blir ungefär 17 cm, fungerar alldeles utmärkt som antenn.

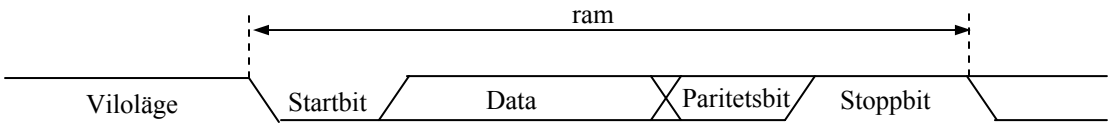


Fig 3.2; När inga ramar skickas från uart:en (viloläge) så ligger TxD-utgången hög vilket ställer till problem för radiomodulen och signalen måste därför inverteras.

4. Överföring

Givarna till väderstationen ger, efter A/D-omvandling, ett åtta bitar långt datapaket motsvarande den spänning givaren levererar.

Vid överföring av data mellan de två enheterna delas de åtta bitar långa datapaketen upp i två paket. Då varje paket som skickas mellan enheterna kan innehålla åtta bitar vardera används de fyra resterande tecknen till att adressera paketen. Väderstationen kan som mest administrera åtta givare varvid vi får åtta värden som delas upp i sexton olika paket.

Paketen som skickas kan delas upp i tre partier. De fyra första tecknen motsvarar hälften av datan från enheten, det femte tecknet talar om huruvida det är paket ett eller två från givaren och de tre sista tecknen beskriver från vilken givare paketet kommer.

Vid mottagning kontrolleras om de två senast mottagna paketen hör ihop d.v.s. att paketen kommer från samma enhet och att paketen har nummer ett och två. När det bekräftats att de två paketen hör ihop kan vi använda de första paketens adress för att utläsa vilken enhet paketen kommer ifrån.
Nu återstår endast att ta de fyra tecknen data från det första paketet och slå ihop med de fyra tecken data från det andra paketet.

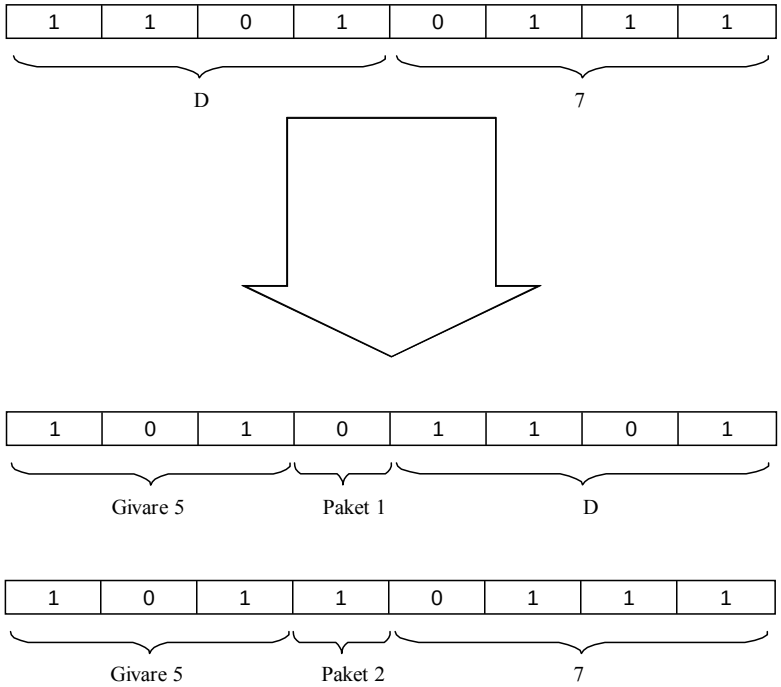


Fig. 4.1; *Exempel där värdet D7 skickas från givare numer fem (vindstyrka).*

5. Mottagarenhet

5.1 Funktionsbeskrivning

Denna mottagarenhet är av allmännast möjliga typ. Den kan ta emot samtlig data WIFS-standarden tillåter, lagra densamma och presentera den på en 4x40-tecken (fyra rader med fyrtio tecken vardera) stor LCD-display. På mottagarens front återfinns fyra menyknappar för navigering och inställning av hur värdena presenteras. Även en reset-knapp för att starta om proceduren finns utifall att något skulle gå galet finns att tillgå.

Tanken bakom mottagarenheten är att den hela tiden håller sin UART-reciever öppen för data. Då ett helt datapaket detekterats kontrolleras dess användbarhet. Verkar allt vara i sin ordning packas mätdatan upp och lagras i minnet (i en tämligen stor matris, se källkod). Från sin plats i minnet hämtas sedan datan då den efterfrågas, räknas om i lämplig enhet och presenteras i sin önskade form på displayen

5. 2 Användarinterface

Tanken bakom menysystemet är, förstås, att det ska vara så enkelt och överskådligt som möjligt. Knapparna ska inte vara för få så att varje knapp får olika funktioner beroende på vilken meny man står i, vad man tryckt förut eller vilken veckodag det är. Men de ska heller inte vara för många så att man måste fundera över vilken det är man bör välja. Efter moget övervägande fastslås att det optimala antalet är fyra;

- en, **NAVIGATE**, för att välja vilket mätvärde man vill titta på
- en, **SELECT**, för att växla mellan Översiktsläge (”Overview”) och Detaljerat läge (”Detailed View”)
- en, **CLEAR**, för att nollställa max/min-värden.
- en, **UNIT**, för att välja vilken enhet valt mätvärde ska visas i.

Det finns två typer av vyer att välja mellan, detaljerad och översikt. När mottagaren startas ställs den in på översiktsvyn (Overview) där samtliga inkopplade mätares mätvärden presenteras i valda enheter. Det detaljerade läger skiljer sig något beroende på vilken mätare som är vald. En skiss över de olika menyerna och knapparnas funktion finns i Appendix B

5.3 Tekniska detaljer

Större delen av mottagarmodulen bygger på inställningar, konfigurationer och mer eller mindre standardmässiga förfaranden (se koppl.schema i App.A.2) men en detalj som kan vara värd ett närmare omnämnande är hur de fyra menyknapparna kopplas till processorn.
Alla fyra knapparna är, via pull up-resistorer, kopplade till varsin SR-vippa som sätts (”S”) då en knapp tyckts ned. Trycks en knapp ned så är alltså signalen efter dess SR-vippa hög tills det att den nollställs av en yttre signal. Utgångarna från dessa fyra SR-vippor genererar med hjälp av en fyra-ingångars OR-grind en signal som talar om huruvida någon knapp tryckts ned eller ej.

Utöver detta så är också varje SR-utgång (”*Q*”) kopplad direkt till en ingångsport på processorn, samt har sitt Reset-stift (”*R*”) förbundet med de övrigas och är gemensamt med dessa kopplad till en utgång från processorn.
Detta förfarande har flera fördelar. Dels så undviks kontaktstuds på ett enkelt och effektivt sätt, dels så kan man direkt i mjukvara bestämma hur lång tid som ska förflyta innan en inhållen knapp ska tolkas som ytterligare ett tryck (genom att välja tiden innan vippan resetas och därmed kan sättas på nytt).
Men framför allt så kan signalen från OR-grinden användas för att generera ett avbrott (Interrupt) i processorn vilket gör att programstrukturen kan utgå ifrån en evighets-loop som bara avbryts då användaren vill förändra något (trycker på en knapp) eller då ny data är inkommen (UART Recieve Complete-Interrupt)

5.4 Användning av radiomodulerna vid mottagning

Den radiomodul som sitter på mottagarenheten innehåller, förutom antenningång, både en digital och en linjär utgång. Som nämndes kort i kapitlet om sändning så har datapaketet inverterats innan sändningen och måste därför inverteras igen på mottagarsidan för att kunna tolkas av mottagarenhetens uart. Att direkt ansluta den digitala utgången till en inverterare fungerade otillfredsställande eftersom radiomodulen inte lämnar en stabil nollnivå i viloläge. Den linjära utgången ligger stabil, dock med en överlagrad DC-spänning vilket syns i figur 5.1.

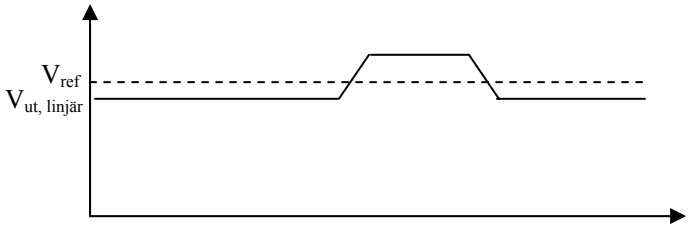


Fig 5.1; *Radiomodulens linjära utgång jämförs i en komperator med en referensnivå för att bilda en standardiserad dataram som uart:en kan tolka.*

Konstruktionen i figur 5.2 eliminerar DC-spänningen och inverterar dessutom signalen. En referensspänning, som ställs in med trimpotentiometern, ansluts till komperatorns ena ingång och radiomodulens linjära utgång till den andra. Utgången kan därefter direkt anslutas till RxD-ingången på uart:en. Komperatorn som används är en LM311 vars utgång är av typen öppen kollektor och en yttre resistor, $R_{2,}$, måste därför anslutas upp till V_{CC}.

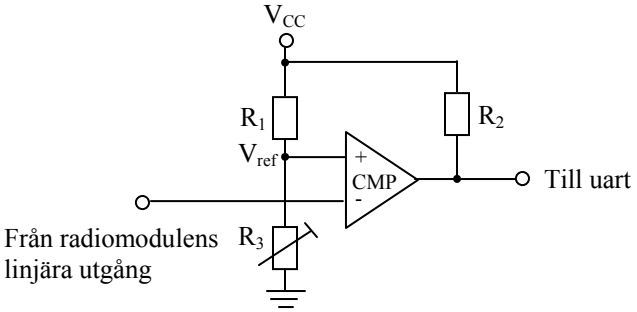


Fig 5.2; *Schema över komperatorkopplingen som ansluts mellan radiomodulen och uart:ens dataingång.*

6. Utvärderande diskussion

6.1 Resultat

Den trådlösa förbindelsen har visat sig vara projektets akilleshäl. Då de komponenter som institutionen förfogar över inte gick att använda fick vi som ersättning använda ett alternativ från Kjell & Co. De nya radiomodulerna var dock inte av den kvalité att de genast gick att koppla in till processorn.

Det faktum att ingen av gruppmedlemmarna hade någon stor erfarenhet av programspråket C har överraskande nog inte varit något större problem. Likheten med Java och bra referenslitteratur att söka i har hjälp oss igenom programmeringen.

För att slutföra projektet har vi i så stor mån som möjligt försökt att dela upp det i mindre problem och testat dessa efter hand. T.ex. har vi haft stor nytta av att testa genom att använda kabel istället för trådlös förbindelse.

Alla problem och mål vi ställde upp före projektets start i kravspecifikationen har vi lyckats uppfylla, dock har vi inte haft tid över för rubriken ”I mån av tid”.

6.2 Vidareutveckling

Projektet har givit en bild hur vår centralenhet kan användas, tanken är nu att detta lätt ska kunna vidareutvecklas.

När en ny typ av givare kopplas in på centralenheten känner enheten att en givare kopplats in och skickar iväg den spänning den levererar tillsammans numret på den port den är ansluten till. Det som krävs för att tillverka en givare anpassad för vår centralenhet är:

- att den använder 3.5 mm 3-poliga s.k. phonokontakter
- att den drivs på 5 volt och
- att den levererar en spänning mellan 0 och 5 volt för att representera sitt mätvärde.

Enheten kan, med nuvarande standard, stödja upp till åtta givare.

På samma sätt går de lätt att utveckla egna visarenheter, allt som krävs är att enheten läser av de paket den är intresserad av. Exempel på visarenheter är många: grafiska displayer, numeriska displayer, lysdioder för att visa vindriktning och stegmotorer som styr visare (för att t.ex. efterlikna analoga termometrar). Systemen behöver inte använda Atmels utrustning, en konstruktion som kan läsa av 433 MHz i kombination med mjukvara som läser av datapaketen enligt våra föreskrifter är allt som krävs.

7. Källförteckning

[1] Franco, Sergio Design with Operational Amplifiers and Analog Integrated Circuits
2002, McGraw Hill, New York

[2] Lindahl, Per-Erik & Sandqvist, William. Mätgivare.
1996, Studentlitteratur, Lund

[3] Kernigan, Brian W. & Ritchie, Dennis M. The C Programming Language
Second Edition, 1996, Prentice Hall, New Jersey

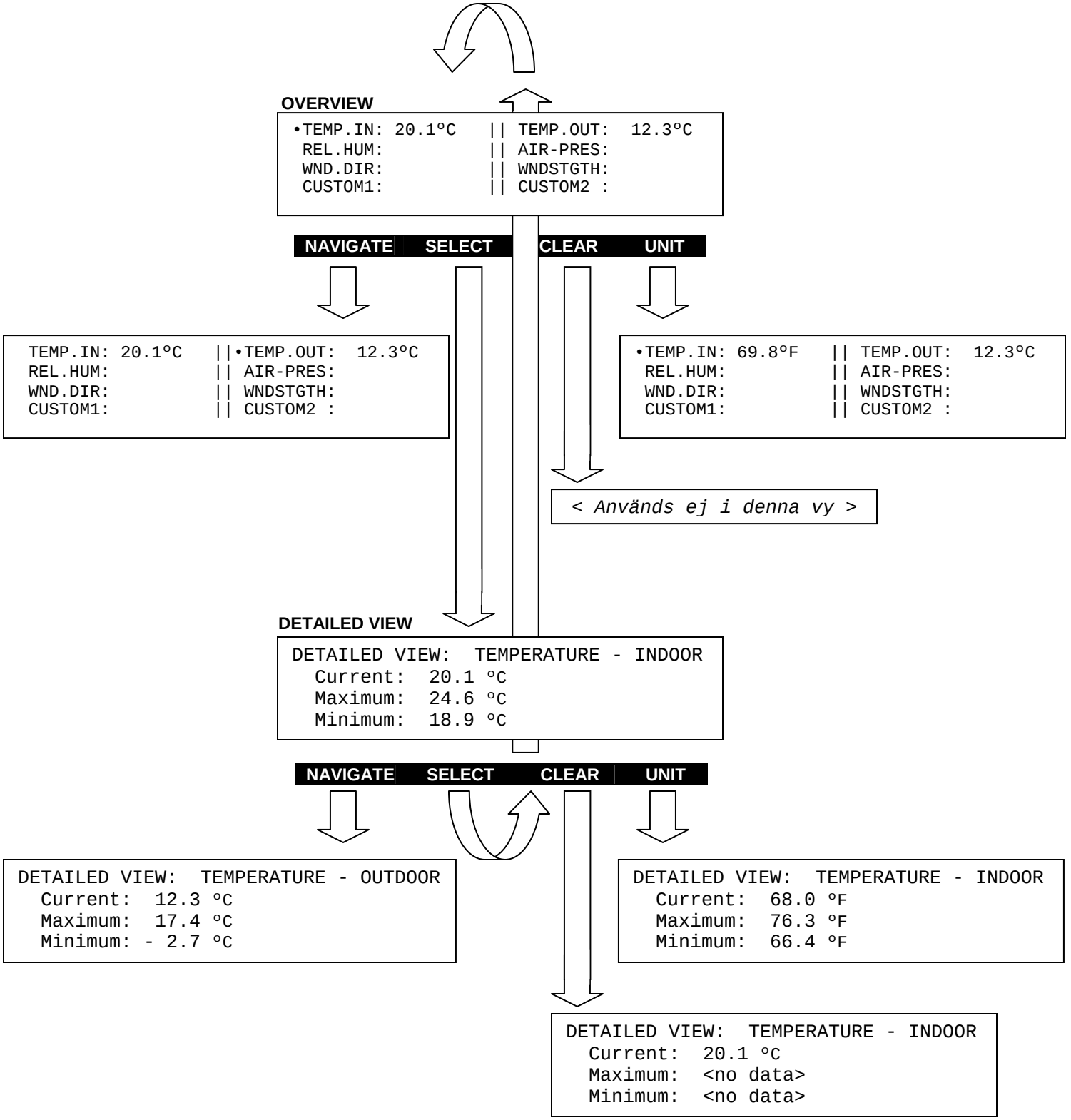
[4] Waite, Mitchel & Prata, Stephen & Martin, Donald. Programmering i C
1986, Pagina Förlags AB Borås

[5] Atmel, Datablad Atmel AVR ATMega16
2004, www.atmel.com/literature

[6] Diverse datablad och övriga, av tillverkare tillhandahållna, instruktioner

[7] http://www.avrfreaks.com (2005-01-24 till 2005-02-28)

Appendix B, Användarinterface - principskiss



Appendix C: Källkod - Centralenhet

C.1 Huvudprogram

```
/*****************************
* Program för centralenheten *
* version 3.0 2004-02-18     *
*****************************/

#include <avr/io.h>
#include <avr/delay.h>
#include "definitioner.h"
#include "funktioner.h"


int main(void){
      intiAVR();        //Initierar AVR:n
      initADC();        //Initierar ADC:n
      initUARTtx();     //Initierar UARTEN för Tx-mode

      for(;;){
            collectDataAndSend(); //Hämtar sensordata och skickar
            delay_s(3); //Väntar ungefär 3 sek
      }
}
```

C.2 Definitionsfil

```
/***********************************
* Definitioner för centralenheten   *
* version 3.0 2004-02-18            *
***********************************/

//Definition av funktioner
void intiAVR(void);
void initADC(void);
void initUARTtx(void);
int pow(int x, int y);
void delay_ms(int ms);
void delay_s(int s);
void sendSensorData(int data, int ap);
```

C.3 Funktionsfil

```
/*******************************
* FUNKTIONER FÖR CENTRALENHETEN *
* version. 3.0 2004-02-18       *
*******************************/

//Deklaration av konstanter
const int maxNbrOfSensors = 4;

/*************************************************************
* initAVR                                                    *
* Ställer ingångar och utgångar rätt hos AVR:n               *
*************************************************************/
void intiAVR(void){
      sbi(DDRD,PD5);
}

/*************************************************************
* initADC                                                    *
* 1. ADLAR = 1          - Vänsterjusterar resultatet         *
* 2. ADPS0,1,2 = 011    - ADC klockfrek. = AVRklockfrek / 8  *
*************************************************************/
void initADC(void){
      sbi(ADMUX,ADLAR);
      ADCSRA = 0x03;
}

/*************************************************************
* initUART                                                   *
* Format: 8e2, baud rate: 2400                               *
*************************************************************/
void initUARTtx(void){
      UBRRL = 0x19;     //Ställer BAUDRATE - 2400
      UCSRA = 0x00;     //Nollställer hela UART reg. A
      UCSRC = 0xAE;     //8e2 - 8 databitar, even parity, 2 stoppbitar
      UCSRB = 0x08;     //Sätter på transmittern
}


/*************************************************************
* int pow(int x, int y)                                      *
* ans = x^y                                                  *
*************************************************************/
int pow(int x, int y){
      int ans=1;
      if ((x<0 || y<0)){ //x och y får inte vara mindre än 0
            return -1;
      }

      for(int i=0; i<y; i++){
            ans=ans*x;
      }
      return ans;
}
```

```
/************************************************************
* void delay_ms(int ms)                                     *
* Väntar "ms" millisekunder                                 *
* Använder _delay_loop_2()                                  *
************************************************************/
void delay_ms(int ms){
      for (int i=0; i<ms; i++){
            _delay_loop_2(333); //väntar ungefär 1 ms
      }
}

/************************************************************
* void delay_s(int s)                                       *
* Väntar "s" sekunder                                       *
************************************************************/
void delay_s(int s){
      for (int t=0; t<s; t++){
            delay_ms(1000); //väntar ungefär 1 s
      }
}

/************************************************************
* void sendSensorData(int data, int ap)                     *
* Delar upp sensorvärdet i två paket.                       *
*                                                           *
* PAKET 1: Ap Ap Ap 0 Data7 Data6 Data5 Data4               *
* PAKET 2: Ap Ap Ap 1 Data3 Data2 Data1 DAta0               *
*                                                           *
* Ap är en kod, så att mottageren ska kunna avkoda vilken   *
* sensor som skickar data                                   *
************************************************************/
void sendSensorData(int data, int ap){

      int datafirst=data/16;
      int datasecond=data-(datafirst*16);
      int sendfirst=(ap*2*16)+datafirst;
      int sendsecond=(((ap*2)+1)*16)+datasecond;

      UDR = sendfirst;  //Sänder de MSB för en sensor
      delay_ms(100);
      UDR = sendsecond; //Sänder den LSB för samma sensor
      delay_ms(90);
}
```

```
/*********************************************************
* collectDataAndSend(void)                               *
* Går igenom alla inkopplade sensorer och skickar värdet    *
* enligt datapakten i "sendSensorData"                   *
*********************************************************/
void collectDataAndSend(void){

      //Deklaration av variabler
      int maskCode;
      int sensorData;

      sbi(PORTD,PD5);         //Tänder LED för att visa
      ADMUX = ADMUX & 0xE0;   //Nollställer muxen
      sbi(ADCSRA, ADEN);      //Sätter på ADC:n

      for(int sensorNbr=0; sensorNbr < maxNbrOfSensors; sensorNbr++){
            maskCode = pow(2,sensorNbr);  //Används för att maska fram
                                          // rätt register m.m.
            if((PINB & maskCode) == 0){ //Sensorn inkopplad ==> Hämtar
                                        // data och skickar

                  sbi(ADCSRA,ADSC); //Startar omvandlingen
                  loop_until_bit_is_set(ADCSRA,ADIF);
                  sensorData = ADCH;
                  sbi(ADCSRA,ADIF); //Nollställer ADC “flaggan”
                  sendSensorData(sensorData, sensorNbr);
            }
      ADMUX = ADMUX + 1; //Uppdaterar till nästa sensor
      }
      cbi(ADCSRA,ADEN); //Stänger ADC:n (sparläge)
      cbi(PORTD,PD5);   //Släcker LED
}
```

Appendix D: Källkod - Mottagarenhet

D.1 Huvudprogram

```
#include "definitioner.h"
#include "funktioner.h"
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <avr/pgmspace.h>

int view;
int data[8][4];
int data_sensor;
int Btn;
int dataFromUDR;

/*************************************************
* Screens for overview and detailed view:             *
*************************************************/

char buffert[40];
PGM_P lines[36] PROGMEM = {
    ov1, ov2, ov3, ov4, tI1,  tI2,  tI3,  tI4,  tO1,  tO2,  tO3,  tO4,
      rH1,  rH2,  rH3,  rH4,  aP1,  aP2,  aP3,  aP4,  wD1,  wD2,  wD3,
      wD4,  wS1,  wS2,  wS3,  wS4,  C11,  C12,  C13,  C14,  C21,  C22,
      C23,  C24         };


const char ov1[] PROGMEM = " TEMP.IN:          || TEMP.OUT:         ";
const char ov2[] PROGMEM = " REL.HUM:          || AIR-PRES:         ";
const char ov3[] PROGMEM = " WND.DIR:          || WNDSTGTH:         ";
const char ov4[] PROGMEM = " CUSTOM1:          || CUSTOM2 :         ";

const char tI1[] PROGMEM = "DETAILED VIEW:  TEMPERATURE - INDOOR    ";
const char tI2[] PROGMEM = "  Current:                               ";
const char tI3[] PROGMEM = "  Maximum:                               ";
const char tI4[] PROGMEM = "  Minimum:                               ";

const char tO1[] PROGMEM = "DETAILED VIEW:  TEMPERATURE - OUTDOOR   ";
const char tO2[] PROGMEM = "  Current:                               ";
const char tO3[] PROGMEM = "  Maximum:                               ";
const char tO4[] PROGMEM = "  Minimum:                               ";

const char rH1[] PROGMEM = "DETAILED VIEW:  RELATIVE AIR HUMIDITY   ";
const char rH2[] PROGMEM = "  Current:                               ";
const char rH3[] PROGMEM = "  Maximum:                               ";
const char rH4[] PROGMEM = "  Minimum:                               ";

const char aP1[] PROGMEM = "DETAILED VIEW:  AIR PRESSURE            ";
const char aP2[] PROGMEM = "                                        ";
const char aP3[] PROGMEM = "  Current:                              ";
const char aP4[] PROGMEM = "                                        ";
```

```
const char wD1[] PROGMEM = "DETAILED VIEW:  WIND DIRECTION          ";
const char wD2[] PROGMEM = "                                        ";
const char wD3[] PROGMEM = "  Current:                              ";
const char wD4[] PROGMEM = "                                        ";

const char wS1[] PROGMEM = "DETAILED VIEW:  WIND STRENGTH           ";
const char wS2[] PROGMEM = "   Strength:                            ";
const char wS3[] PROGMEM = "  Cold.eff.:                            ";
const char wS4[] PROGMEM = "Description:                            ";

const char C11[] PROGMEM = "DETAILED VIEW:  CUSTOM FUNCTION #1      ";
const char C12[] PROGMEM = "                                        ";
const char C13[] PROGMEM = " <No custom-function device connected!> ";
const char C14[] PROGMEM = "   Please refer to manual for details   ";

const char C21[] PROGMEM = "DETAILED VIEW:  CUSTOM FUNCTION #2      ";
const char C22[] PROGMEM = "                                        ";
const char C23[] PROGMEM = " <No custom-function device connected!> ";
const char C24[] PROGMEM = "   Please refer to manual for details   ";

SIGNAL(SIG_UART_RECV){                    //tagit emot paket via USART

      int oldDataFromUDR=0;
      extern int dataFromUDR;

      int dataUCSRA=UCSRA;
      int FrameError=mask(dataUCSRA,4);
      int ParityError=mask(dataUCSRA,2);

      if(FrameError==0){
            oldDataFromUDR=dataFromUDR;
            dataFromUDR = UDR;
            adress(oldDataFromUDR,dataFromUDR);
      }
      if(ParityError==0){}

      if( ( (dataFromUDR/16) - (oldDataFromUDR/16) ==1) &&
((oldDataFromUDR/16)-(((oldDataFromUDR/16))*2)==0 ) ){
                        //om helt paket mottaget, uppdatera display
            if (view==1){     DisplayOverview();
            }else{                  DisplayDetails(); }
      }
}
INTERRUPT(SIG_INTERRUPT1)                 //knapptyckning
{
      //Läs av knappar och gör vad som ska göras
      extern int view;
      extern int data[8][4];
      extern int data_sensor;
      extern int Btn;

      int i;
      Btn=Read_Buttons();
      switch ( Btn ){
            case 1 :                      //Navigate
                  if (view==1){           //Overview
                        data_sensor=incr(data_sensor, 7);
```

```
                        DisplayOverview();
                  }else{                        //Detailed view
                        data_sensor=incr(data_sensor, 7);
                        DisplayDetails();             }
                  break;

            case 2 :                            //Select
                  if (view==1){                 //Overview
                        view=2;
                        DisplayDetails();
                  }else{                        //Detailed view
                        view=1;
                        DisplayOverview();
                  }
                  break;

            case 3 :                            //Clear
                  if (view==1){                 //Overview
                        //do nothing
                  }else{                        //Detailed view
                        data[data_sensor][1]=-1;      //nolla max
                        data[data_sensor][2]=-1;      //nolla min
                        DisplayDetails();
                  }
                  break;

            case 4 :                            //Unit


                  i = data[data_sensor][3];
                  i = i+1;
                  if (i>1){   i=-1; }

            data[data_sensor][3] = i;
                  if (view==1){
                        DisplayOverview();
                  }else if (view==2){
                        DisplayDetails();
                  }

                  break;

            default :
                        //Ingen giltig knapp, något har gått fel.
                  break;

      }
      // delay för att man ska hinna släppa knappen
      delay_ms(250);
      // reseta sr-vippa
      cbi(PORTD, SR_Reset);
      sbi(PORTD, SR_Reset);
}

INTERRUPT(__vector_default){  //gå tillbaka direkt, gör inget alls
}
```

```
void DisplayOverview(void){
      extern int data[][];
    extern char buffert[];

      LCD_Place_Cursor(0x80,'u');
      strcpy_P (buffert, lines[0]);
      LCD_SendString(buffert, 'u');

      LCD_Place_Cursor(0xc0,'u');
      strcpy_P (buffert, lines[1]);
      LCD_SendString(buffert, 'u');

      LCD_Place_Cursor(0x80,'l');
      strcpy_P (buffert, lines[2]);
      LCD_SendString(buffert,'l');

      LCD_Place_Cursor(0xc0,'l');
      strcpy_P (buffert, lines[3]);
    LCD_SendString(buffert,'l');


      //Uppdatera data
      LCD_Place_Cursor(0x8a , 'u');       //Temp In

      if(data[0][0]==-1){
            LCD_SendString(" ---",'u');
      }else{                              //skriv ut data


            int z;
            int first;
            switch ( data[0][3] ){  // välj väg beroende på vald enhet
                  case -1 :                     //Celsius
                        z=data[0][0];
                        printCelsius(z, 'u');
                        break;

                  case 0:                       //Fahrenheit
                        z=data[0][0];
                        printFahrenheit(z, 'u');

                        break;

                  case 1:                       //Kelvin
                        z=data[0][0];     //27320=273,2*100
                        printKelvin(z, 'u');
                        break;

                  default:
                        LCD_SendChar(' ','u');
                        LCD_SendChar('d','u');
                        break;

            }
      }
```

```
LCD_Place_Cursor(0xa0 , 'u');             //Temp Ut
if(data[1][0]==-1){
      LCD_SendString(" ---",'u');
}else{                    //Skriv data
}
LCD_Place_Cursor(0xcb , 'u');             //Fukt.
if(data[2][0]==-1){
      LCD_SendString("---",'u');
}else{                    //Skriv data
}


LCD_Place_Cursor(0xe0 , 'u');             //Tryck

if(data[3][0]==-1){
      LCD_SendString(" ---",'u');
}else{                    //Skriv data

      int z=data[3][0];
      switch (data[3][3]){

            case -1:
                  printMbar(z, 'u');
            break;

            case 0:
                  printHpa(z, 'u');
            break;

            case 1:
                  printMmhg(z, 'u');
            break;
      }
}

LCD_Place_Cursor(0x8b , 'l');             //Wind-dir
if(data[4][0]==-1){
      LCD_SendString("---",'l');
}else{
}
LCD_Place_Cursor(0xa1 , 'l');             //Wind-spd
if(data[5][0]==-1){
      LCD_SendString("---",'l');
}else{
}
LCD_Place_Cursor(0xcb , 'l');             //Custom1
if(data[6][0]==-1){
      LCD_SendString("---",'l');
}else{
}
LCD_Place_Cursor(0xe1 , 'l');             //Custom2
if(data[7][0]==-1){
      LCD_SendString("---",'l');
}else{
}
```

```
      char rad;
      char adress;
      if (data_sensor<4){rad='u';
      }else{ rad='l';         }
    if ( (data_sensor==0)|(data_sensor==4) ){adress=0x80;
      }else if ( (data_sensor==1)|(data_sensor==5) ) {adress=0x95;
      }else if ( (data_sensor==2)|(data_sensor==6) ) {adress=0xc0;
      }else if ( (data_sensor==3)|(data_sensor==7) ) {adress=0xd5;
      }
      LCD_Place_Cursor(adress, rad);
      LCD_SendChar(0xa5, rad);
      }

void DisplayDetails(void){
      //                    12345678901234567890123456789012345678901234567890
      extern char buffert[];
      switch (data_sensor){
            case 0:                                 //Temp, inne
                  //Skriver grund
                  LCD_Place_Cursor(0x80, 'u');
                  strcpy_P (buffert, lines[4]);
                  LCD_SendString(buffert, 'u');

                  LCD_Place_Cursor(0xc0, 'u');
                  strcpy_P (buffert, lines[5]);
                  LCD_SendString(buffert, 'u');

                  LCD_Place_Cursor(0x80, 'l');
                  strcpy_P (buffert, lines[6]);
                  LCD_SendString(buffert, 'l');

                  LCD_Place_Cursor(0xc0, 'l');
                  strcpy_P (buffert, lines[7]);
                  LCD_SendString(buffert, 'l');

                  //Skriver värden
                  LCD_Place_Cursor(0xcb, 'u');  //Current
                  if(data[0][0]==-1){
                        LCD_SendString("<No data>",'u');
                  }else{
                        int z;
                        switch ( data[0][3] ){
                        // välj väg beroende på vald enhet
                        case -1 :         //Celsius
                              z=data[0][0];
                              printCelsius(z, 'u');
                              break;
                        case 0:                 //Fahrenheit
                              z=data[0][0];
                              printFahrenheit(z, 'u');
                              break;
                        case 1:                 //Kelvin
                              z=data[0][0];           //27320=273,2*100
                              printKelvin(z, 'u');
                              break;
                        default:
                              LCD_SendChar(' ','u');
```

```
            LCD_SendChar('d','u');
            break;
      }
}
LCD_Place_Cursor(0x8b, 'l');        //Max
if(data[0][1]==-1){
      LCD_SendString("<No data>",'l');
}else{
      int z;
      switch ( data[0][3] ){
      // välj väg beroende på vald enhet
      case -1 :               //Celsius
            z=data[0][1];
            printCelsius(z, 'l');
            break;

      case 0:                 //Fahrenheit
            z=data[0][1];
            printFahrenheit(z, 'l');
            break;

      case 1:                 //Kelvin
            z=data[0][1];     //27320=273,2*100
            printKelvin(z, 'l');
            break;

      default:
            LCD_SendChar(' ','l');
            LCD_SendChar('d','l');
            break;

      }
}
LCD_Place_Cursor(0xcb, 'l');        //Min
if(data[0][2]==-1){
      LCD_SendString("<No data>",'l');
}else{
      int z;

      switch ( data[0][3] ){
      // välj väg beroende på vald enhet
      case -1 :               //Celsius
            z=data[0][2];
            printCelsius(z, 'l');
            break;

      case 0:                 //Fahrenheit
            z=data[0][2];
            printFahrenheit(z, 'l');
            break;

      case 1:                 //Kelvin
            z=data[0][2];     //27320=273,2*100
            printKelvin(z, 'l');
            break;

      default:
```

```
                    LCD_SendChar(' ','l');
                    LCD_SendChar('d','l');
                    break;
              }
        }
        break;
case 1:                                 //Temp, ute
        //Skriver grund
        LCD_Place_Cursor(0x80, 'u');
        strcpy_P (buffert, lines[8]);
        LCD_SendString(buffert, 'u');

        LCD_Place_Cursor(0xc0, 'u');
        strcpy_P (buffert, lines[9]);
        LCD_SendString(buffert, 'u');

        LCD_Place_Cursor(0x80, 'l');
        strcpy_P (buffert, lines[10]);
        LCD_SendString(buffert, 'l');

        LCD_Place_Cursor(0xc0, 'l');
        strcpy_P (buffert, lines[11]);
        LCD_SendString(buffert, 'l');


        //Skriver värden
        LCD_Place_Cursor(0xcc, 'u');        //Current
        if(data[1][0]==-1){
              LCD_SendString("<No data>",'u');
        }else{
              //skriv data och enhet
        }
        LCD_Place_Cursor(0x8c, 'l');        //Max
        if(data[1][1]==-1){
              LCD_SendString("<No data>",'l');
        }else{
              //skriv data och enhet
        }
        LCD_Place_Cursor(0xcc, 'l');        //Min
        if(data[1][2]==-1){
              LCD_SendString("<No data>",'l');
        }else{
              //skriv data och enhet
        }
        break;

case 2:                                 //Luftfuktighet
        //Skriver grund
        LCD_Place_Cursor(0x80, 'u');
        strcpy_P (buffert, lines[12]);
        LCD_SendString(buffert, 'u');

        LCD_Place_Cursor(0xc0, 'u');
        strcpy_P (buffert, lines[13]);
        LCD_SendString(buffert, 'u');

        LCD_Place_Cursor(0x80, 'l');
```

```
		strcpy_P (buffert, lines[14]);
		LCD_SendString(buffert, 'l');

		LCD_Place_Cursor(0xc0, 'l');
		strcpy_P (buffert, lines[15]);
		LCD_SendString(buffert, 'l');

		//Skriver värden
		LCD_Place_Cursor(0xcc, 'u');		//Current
		if(data[2][0]==-1){
			LCD_SendString("<No data>",'u');
		}else{
			//skriv data och enhet
		}
		LCD_Place_Cursor(0x8c, 'l');		//Max
		if(data[2][1]==-1){
			LCD_SendString("<No data>",'l');
		}else{
			LCD_SendString("data funnen!", 'l');
		}
		LCD_Place_Cursor(0xcc, 'l');		//Min
		if(data[2][2]==-1){
			LCD_SendString("<No data>",'l');
		}else{
			//skriv data och enhet
		}
		break;

	case 3:					//Lufttryck
		//Skriver grund
		LCD_Place_Cursor(0x80, 'u');
		strcpy_P (buffert, lines[16]);
		LCD_SendString(buffert, 'u');

		LCD_Place_Cursor(0xc0, 'u');
		strcpy_P (buffert, lines[17]);
		LCD_SendString(buffert, 'u');

		LCD_Place_Cursor(0x80, 'l');
		strcpy_P (buffert, lines[18]);
		LCD_SendString(buffert, 'l');

		LCD_Place_Cursor(0xc0, 'l');
		strcpy_P (buffert, lines[19]);
		LCD_SendString(buffert, 'l');
		//Skriver värden
		LCD_Place_Cursor(0x8c, 'l');		//Current
		if(data[3][0]==-1){
			LCD_SendString("<No data>",'l');
		}else{
			int z=data[3][0];

			switch (data[3][3]){

			case -1:
				printMbar(z, 'l');
			break;
```

```
                case 0:
                        printHpa(z, 'l');
                break;

                case 1:
                        printMmhg(z, 'l');
                break;
        }

                //skriv hi/norm/lo
                LCD_SendChar(' ', 'l');
                LCD_SendChar('(', 'l');
                if ( z>10250 ){
                        LCD_SendString("High",'l');
                }else if ( z>10000 ){
                        LCD_SendString("Normal",'l');
                }else{
                        LCD_SendString("Low",'l');
                }
                LCD_SendChar(')', 'l');
        }

        break;

case 4:                                 //Vindriktning
        //Skriver grund
        LCD_Place_Cursor(0x80, 'u');
        strcpy_P (buffert, lines[20]);
        LCD_SendString(buffert, 'u');

        LCD_Place_Cursor(0xc0, 'u');
        strcpy_P (buffert, lines[21]);
        LCD_SendString(buffert, 'u');

        LCD_Place_Cursor(0x80, 'l');
        strcpy_P (buffert, lines[22]);
        LCD_SendString(buffert, 'l');

        LCD_Place_Cursor(0xc0, 'l');
        strcpy_P (buffert, lines[23]);
        LCD_SendString(buffert, 'l');
        //Skriver värden
        LCD_Place_Cursor(0x8c, 'l');            //Current
        if(data[4][0]==-1){
                LCD_SendString("<No data>",'l');
        }else{
                //skriv data
        }

        break;
case 5:                                 //Vindstyrka
        //Skriver grund
        LCD_Place_Cursor(0x80, 'u');
        strcpy_P (buffert, lines[24]);
        LCD_SendString(buffert, 'u');
```

```
		LCD_Place_Cursor(0xc0, 'u');
		strcpy_P (buffert, lines[25]);
		LCD_SendString(buffert, 'u');

		LCD_Place_Cursor(0x80, 'l');
		strcpy_P (buffert, lines[26]);
		LCD_SendString(buffert, 'l');

		LCD_Place_Cursor(0xc0, 'l');
		strcpy_P (buffert, lines[27]);
		LCD_SendString(buffert, 'l');
		//Skriver värden
		LCD_Place_Cursor(0xcd, 'u');		//strength
		if(data[5][0]==-1){
			LCD_SendString("<No data>",'u');
		}else{
			//skriv data och enhet
		}
		LCD_Place_Cursor(0x8d, 'l');		//coldeff
		if(data[5][1]==-1){
			LCD_SendString("<No data for calc.>",'l');
		}else{
			//beräkna data och enhet
		}
		LCD_Place_Cursor(0xcd, 'l');		//descr
		if(data[5][0]==-1){
			LCD_SendString("<No data for calc.>",'l');
		}else{
			//skriv beskrivning
		}

		break;
	case 6:					//Custom1
		//Skriver grund
		LCD_Place_Cursor(0x80, 'u');
		strcpy_P (buffert, lines[28]);
		LCD_SendString(buffert, 'u');

		LCD_Place_Cursor(0xc0, 'u');
		strcpy_P (buffert, lines[29]);
		LCD_SendString(buffert, 'u');

		LCD_Place_Cursor(0x80, 'l');
		strcpy_P (buffert, lines[30]);
		LCD_SendString(buffert, 'l');

		LCD_Place_Cursor(0xc0, 'l');
		strcpy_P (buffert, lines[31]);
		LCD_SendString(buffert, 'l');
		break;
	case 7:					//Custom2
		//Skriver grund
		LCD_Place_Cursor(0x80, 'u');
		strcpy_P (buffert, lines[32]);
		LCD_SendString(buffert, 'u');

		LCD_Place_Cursor(0xc0, 'u');
```

```
                  strcpy_P (buffert, lines[33]);
                  LCD_SendString(buffert, 'u');

                  LCD_Place_Cursor(0x80, 'l');
                  strcpy_P (buffert, lines[34]);
                  LCD_SendString(buffert, 'l');

                  LCD_Place_Cursor(0xc0, 'l');
                  strcpy_P (buffert, lines[35]);
                  LCD_SendString(buffert, 'l');
                  break;

                  default:                    //Ska aldrig inträffa
                  break;
      }
}


int main(void){

/**********************
*    INITIALISERING    *
**********************/
///Port-initiering
      DDRA = 0xff;                  //Sätter port A som utgångar
      DDRB = 0xf0;                  //B4-7 utgångar, B0-3 ingångar.
/*    PORTC res. för JTAG           */
      DDRD = 0xf6 ;                 //(111*0110) *=DC=1

      cbi(PORTD, SR_Reset);   //Nollställer SR
      sbi(PORTD, SR_Reset);   //Sätter SR_Reset hög
      LCD4044_Init();               //Initierar LCD
      USARTRx_Init();               //Initierar UART-Recieve
      int dummy;
      while (UCSRA & (1<<RXC)){
            dummy = UDR;}           //Tömmer UART:s buffer


///GUI-initiering
      extern int data[8][4];        // kolumner(8 st)=storhet,
rader(4st)=data
            for(int m=0; m<8; m++){
                  for(int n=0; n<4; n++){
                        data[m][n]=(-1.0);
      }     }
      extern int dataFromUDR;
      dataFromUDR=0;
      extern int data_sensor;             // kan anta 0
(TempIn)....7(Custom2)
      //extern int data_type;             // kan anta 0 (curr.), 1
(max.), 2(min.), 3 (unit)
      data_sensor=0;
      extern int view;                    // 1="overview", 2="details"
(VARFÖR FINNS INGA BOOL?!?)
      view=1;
      DisplayOverview();
```

```
//Interrupt-initiering
      timer_enable_int(0);    // disablar alla timer-interrupts
      GICR |= 0x80;                 // aktiverar externt interrupt 1
(endast)
      MCUCR |= 0x0c;                //
      sbi(UCSRB, RXCIE);            // aktiverar USART Recieve-
interrupt
      sei();                              // enablar Global Interrupt i
SREG



/*********************
*    HUVUDPROGRAM  *
*********************/

      while(1){
      while (UCSRA & (1<<RXC)) dummy = UDR;
      }
      return(0);
}
```

D.2 Definitionsfil

```
/*////////////////////////////////////////////////
/     Definitionsfil för WIFS-Mottagarenhet        /
/      v1.0 2005-02-15                             /
////////////////////////////////////////////////*/

/*************************************************
* Pin correspondence; AVRMega16<->L4044 Disp.Driver *
*************************************************/
/*PA0 - DB0
.           .
.           .
.           .
PA7 - DB7

PB4   - RS
PB5   - R/W
PB6   - E1
PB7   - E2
*/

/*************************************************
* Algorithm used for data/character-transfer:       *
*************************************************
  - Take RW low
  - RS as needed for operation
  - Put data on bus
  - Take E high
  - Take E low
*/


/*************************************************
* Functions                                      *
*************************************************/
//General functions
void delay_10us(int us);
void delay_ms(int ms);

//LCD-related functions
void LCD_SendCmd(char c, char rows);
void LCD_SendChar(char c, char rows);
void LCD_SendNbr(int i, char rows);
void LCD_SendString(char *strng, char rows);
void LCD_Display_Clear(char rows);  //also returns cursor to home
void LCD_Place_Cursor(char addr, char rows);
void LCD4044_Init(void);

//Button-related functions
int Read_Buttons(void);

//Program-specific functions
void USARTRx_Init(void);
```

```
int incr(int val, int limit);
void DisplayOverview(void);
void DisplayDetails(void);
void printFahrenheit (int z, char rows);
void printKelvin (int z, char rows);
void printCelsius (int z, char rows);
void printMbar (int z, char rows);
void printHpa (int z, char rows);
void printMmhg (int z, char rows);
//sub-functions
void temp(int data);
void baro(int volt);
void adress(int old, int thenew);
int mask(int data, int bit);
/************************************************
* Definitions                                   *
************************************************/
//Ports and connectors
#define LCD_Data PORTA
#define LCD_Control PORTB  //OBS!
#define RS PB4
#define RW PB5
#define E1 PB6
#define E2 PB7

#define SR_Reset PD2
#define BTN1 PB3
#define BTN2 PB2
#define BTN3 PB1
#define BTN4 PB0
//Commands (RS=RW=0 except for CG/DDRAM)
//All commands defined as stated in the L4044 driver data sheet
#define Cursor_Return_Home    0x02

#define Function_Set     0x20       //Sets interface properties
#define Eightbit_DataLen 0x10       //clear for 4bitDL
#define TwoorFourlines   0x08       //clear for 1-line mode
#define Big_Font         0x02       //clear for std 5x7 dots

#define Entry_Mode_Set  0x04        //Sets cursor behaviour. Can
                                    //be read as well as written
#define Cursor_Inc      0x02        //clear for decrement. Specifies
                                    //direction of cursor movement on
                                    //data write/read
#define Display_Shift   0x01        //clear for don't shift. Shifts
                                    //display on data write
#define Display_OnOff   0x08        //Sets display appearace properties
#define Display_On      0x04        //clear for Display_On
#define Cursor_On       0x02        //clear for Cursor_Off
#define Cursor_Blink    0x01        //clear for Cursor_undersc

#define DispCurs_shift  0x10        //Shifts display or incr/decr
                                    //cursor wo changing DDRAM
#define     DC_Curs_left  0x00      //Cursor left (AC--), displ. Stat.
#define     DC_Curs_right     0x01  //Cursor right (AC++), displ. stat.
#define     DC_DC_left        0x02  //Cursor and display left
#define     DC_DC_right       0x03  //Cursor and display right
```

D.3 Funktionsfil

```
/*////////////////////////////////////////////////
/           Funktionsfil för WIFS-Mottagarenhet            /
/           v1.0  2005-02-16                                      /
////////////////////////////////////////////////*/
#include <avr/io.h>
#include <avr/delay.h>

/************************************************
* General Functions                              *
************************************************/
void delay_10us(int us){
      // Väntar ca us*10 mikrosekunder (verklig delay ~9us)
      for (int i=0; i<us; i++){
            _delay_loop_2(3);
      }
}
void delay_ms(int ms){
      // Väntar ca ms millisekunder (verklig delay ~9,99ms)
      for (int i=0; i<ms; i++){
            _delay_loop_2(333);
      }
}

/************************************************
* LCD-related Functions                          *
************************************************/
void LCD_SendCmd(char c, char rows){
      //Sänder ett Kommando till LCD:n
      //Syntaxex: LCD_SendCmd(Disp_Clear, 'u'), LCD_SendCmd(0x01, 'l');

      cbi(LCD_Control, E2), cbi(LCD_Control, E1), cbi(LCD_Control, RS),
      cbi(LCD_Control, RW);               // (E2) (E1) (RS) (R/W)
      LCD_Data=c;
      if(rows=='u'){
            sbi(LCD_Control, E1);
            delay_10us(1);
            cbi(LCD_Control, E1);
      }else if(rows=='l'){
            sbi(LCD_Control, E2);
            delay_10us(1);
            cbi(LCD_Control, E2);
      }else if(rows=='a'){
            sbi(LCD_Control, E1);
            sbi(LCD_Control, E2);
            delay_10us(1);
            cbi(LCD_Control, E1);
            cbi(LCD_Control, E2);
      }
      delay_10us(5);
}
```

```
void LCD_SendChar(char c, char rows){
      //Skriver ASCII-tecken till displayen på cursorns nuvarande pos.
      //rows=u ger övre radparet, l nedre
      //Syntax: LCD_SendChar(H, u);
      cbi(LCD_Control, E2), cbi(LCD_Control, E1), sbi(LCD_Control, RS),
cbi(LCD_Control, RW); // (E2) (E1) RS (R/W)
      LCD_Data=c;
      if(rows=='u'){
            sbi(LCD_Control, E1);
            delay_10us(1);          //behöver bara vara 500ns
            cbi(LCD_Control, E1);
      }else if(rows=='l'){
            sbi(LCD_Control, E2);
            delay_10us(1);          //behöver bara vara 500ns
            cbi(LCD_Control, E2);
      }else if(rows=='a'){
            sbi(LCD_Control, E1);
            sbi(LCD_Control, E2);
            delay_10us(1);          //behöver bara vara 500ns
            cbi(LCD_Control, E1);
            cbi(LCD_Control, E2);
      }
}
void LCD_SendNbr(int i, char rows){
      switch (i){
      case 0 :
            LCD_SendChar('0', rows);
            break;
      case 1 :
            LCD_SendChar('1', rows);
            break;
      case 2 :
            LCD_SendChar('2', rows);
            break;
      case 3 :
            LCD_SendChar('3', rows);
            break;
      case 4 :
            LCD_SendChar('4', rows);
            break;
      case 5 :
            LCD_SendChar('5', rows);
            break;
      case 6 :
            LCD_SendChar('6', rows);
            break;
      case 7 :
            LCD_SendChar('7', rows);
            break;
      case 8 :
            LCD_SendChar('8', rows);
            break;
      case 9 :
            LCD_SendChar('9', rows);
            break;
      }
}
```

```
void LCD_SendString(char *strng, char rows){
      for(int i=0; (i < strlen(strng)); i++){
                  LCD_SendChar(strng[i], rows);
      }
}
void LCD_Display_Clear(char rows){
      cbi(LCD_Control, E2), cbi(LCD_Control, E1), cbi(LCD_Control, RS),
cbi(LCD_Control, RW);         // (E2) (E1) (RS) (R/W)
      LCD_SendCmd(0x01,rows);
      delay_ms(2);
}
void LCD_Place_Cursor(char addr, char rows){
      //Sätter cursor på char. Radpar definieras genom rows
      //Syntex: LCD_Place_Cursor(0x8e, 'u')
      cbi(LCD_Control,RS), cbi(LCD_Control,RW);
      LCD_Data=addr;
      if(rows=='u'){
            sbi(LCD_Control, E1);
            delay_10us(1);
            cbi(LCD_Control, E1);
      }else if(rows=='l'){
            sbi(LCD_Control, E2);
            delay_10us(1);
            cbi(LCD_Control, E2);
      }else if(rows=='a'){
            sbi(LCD_Control, E1);
            sbi(LCD_Control, E2);
            delay_10us(1);
            cbi(LCD_Control, E1);
            cbi(LCD_Control, E2);
      }
}
void LCD4044_Init(void){
      //Initierar LCD-Controllern L4044 (Måste göras på varje
      //startup/reset)
      //Rensar displayraderna, Aktiverar INTE cursor på övre radpar mm.
      //Syntax: LCD4044_Init();

      LCD_SendCmd( Function_Set + Eightbit_DataLen + TwoorFourlines
      ,'u');
      LCD_SendCmd( Function_Set + Eightbit_DataLen + TwoorFourlines
      ,'u');
      LCD_SendCmd( Entry_Mode_Set + Cursor_Inc ,'u');
      delay_ms(2);
      LCD_SendCmd( Display_OnOff + Display_On ,'u');
      LCD_Display_Clear('u');
      LCD_Place_Cursor( 0x80,'u');

      LCD_SendCmd( Function_Set + Eightbit_DataLen + TwoorFourlines
      ,'l');
      LCD_SendCmd( Entry_Mode_Set + Cursor_Inc ,'l');
      LCD_SendCmd( Display_OnOff + Display_On ,'l');
      LCD_Display_Clear('l');
      LCD_Place_Cursor( 0x80,'l');
}
```

```
/*****************************************************
* Button-related Functions                           *
*****************************************************/
int Read_Buttons(void){
      /*Returnerar 0x0n för knapp nr n.
        Ingen knapp => 0x00                */
      /*Prioritetsordning: 4-3-2-1-(0)  */

      char mask =0x01;
      int knappNr;
      for (knappNr = 1; knappNr < 5; knappNr++) {
            if((PINB & mask) == mask){
                  break;
            }else{
                  mask = mask*2;
            }
            if (knappNr==4){
                  return 0;
            }
      }
      return (5-knappNr);
}


#define spacer
/*****************************************************
* Program-specific functions                         *
*****************************************************/
void USARTRx_Init(void){
      UCSRB = 0x10;     //Sätter på Receivern
      UBRRL = 0x19;     //Ställer BAUDRATEN -
      UCSRA = 0x00;     //Speed
      UCSRC = 0xAE;     //
}

int incr(int val, int limit){
      val = val+1;            //varför funkar inte ++ ??
      if ( val==(limit+1) ){
            val=0;
      }
      return val;
}

void printFahrenheit(int z, char rows){

            float fa = (float)z;
            float fb = (1.8*fa)+3200;
            int F = (int)fb;                    // 06800F

            int f1=F/10000;
            if(f1==0){  LCD_SendChar(' ',rows); //' '
            }else{            LCD_SendNbr(f1, rows);  }

            int f2=(F - f1*10000)/1000;         // 6
            LCD_SendNbr (f2, rows);
            int f3=(F - f1*10000 - f2*1000)/100;// 8
            LCD_SendNbr (f3, rows);
```

```
            LCD_SendChar('.', rows);
            int f4=(F - f1*10000 - f2*1000 -f3*100)/10;
            LCD_SendNbr (f4, rows);
            LCD_SendChar(' ', rows);
            LCD_SendChar(0xdf, rows);
            LCD_SendChar('F', rows);
}

void printKelvin (int z, char rows){

            z= z + 27320;           //27320=273,2*100
            int k1=z/10000;
            LCD_SendNbr ( k1 ,rows);
            // 2
            int k2=(z - k1*10000)/1000;
            LCD_SendNbr ( k2 ,rows);
            // 7
            int k3=(z - k1*10000 - k2*1000)/100;
            LCD_SendNbr ( k3 ,rows);
            // 3
            LCD_SendChar('.', rows);
            // .
            int k4=(z - k1*10000 - k2*1000 - k3*100)/10;
            LCD_SendNbr ( k4 ,rows);
            // 2
            LCD_SendChar(' ', rows);
            LCD_SendChar('K', rows);

}

void printCelsius (int z, char rows){
            int first=z/1000;
            if(first!=0){     LCD_SendNbr(first,rows );
            }else{                  LCD_SendChar(' ', rows);
      }
            LCD_SendNbr (z/100-(first*10), rows);
            LCD_SendChar('.',rows);
            LCD_SendNbr ((z/10)-((z/100)*10),rows);
            LCD_SendChar(' ', rows);
            LCD_SendChar(0xdf, rows);
            LCD_SendChar('C', rows);
}
void printMbar (int z, char rows){

            int first=z/10000;
            if (first!=0){
                  LCD_SendNbr(first, rows);
            }else{
                  LCD_SendChar(' ', rows);
            }
          LCD_SendNbr(z/1000-(first*10),rows);
          int b=(z/1000)-(first*10);
          LCD_SendNbr((z/100)-(first*100)-(b*10),rows);
          int c=(z/100)-(first*100)-(b*10);
          LCD_SendNbr((z/10)-(first*1000)-(b*100)-(c*10),rows);

            LCD_SendString("mb", rows);
```

```
}
void printHpa (int z, char rows){

            int first=z/10000;
            if (first!=0){
                  LCD_SendNbr(first, rows);
            }else{
                  LCD_SendChar(' ', rows);
            }
          LCD_SendNbr(z/1000-(first*10),rows);
          int b=(z/1000)-(first*10);
          LCD_SendNbr((z/100)-(first*100)-(b*10),rows);
          int c=(z/100)-(first*100)-(b*10);
          LCD_SendNbr((z/10)-(first*1000)-(b*100)-(c*10),rows);

            LCD_SendString("hPa", rows);
}
void printMmhg (int z, char rows){
            int first=z/10000;
            if (first!=0){
                  LCD_SendNbr(first, rows);
            }else{
                  LCD_SendChar(' ', rows);
            }
          LCD_SendNbr(z/1000-(first*10),rows);
          int b=(z/1000)-(first*10);
          LCD_SendNbr((z/100)-(first*100)-(b*10),rows);
          int c=(z/100)-(first*100)-(b*10);
          LCD_SendNbr((z/10)-(first*1000)-(b*100)-(c*10),rows);

            LCD_SendString("mmHg", rows);
}

//sub-functions
void adress(int old, int thenew){

      int adressold=old/16;
      int adressnew = thenew/16;
            int order = adressold-((adressold/2)*2);// 0 om paketen I
                                                    //följande ordning
      if((adressnew-adressold==1) && (order==0)){
          int dataold=old-(adressold*16);
          int datanew=thenew-(adressnew*16);
          int sensordata =(dataold*16)+datanew;

            if(adressold==0){
                  temp(sensordata);
                  }
            if(adressold==2){
                  }
            if(adressold==4){
                  }
            if(adressold==6){
                 baro(sensordata);
                  }
            if(adressold==8){
```

```
                }
            if(adressold==10){
                }
            if(adressold==12){
                }
            if(adressold==14){
                }
      }
}

void temp(int volt){
      extern int data[][];
      float x=((volt*5.25)/256)*8;
      float y=(x*100);
      data[0][0] =(int)y;
      if(y > data[0][1]){
            data[0][1]=y;
      }
      if( (y < data[0][2]) | (data[0][2]==(-1))  ){
            data[0][2]=y;
      }

}
void baro(int volt){
      extern int data[][];
      float x=(((volt*5.25)/256)*40)+900;
      float y=(x*10);
      data[3][0]= (int)y;
}
int mask(int data, int bit){

      //Undviker att anvanda pow!!!
      int result=2; //For kompilatorn
      if(bit==0){
         result = data-((data/2)*2);
      }
      if(bit==1){
          int x=data/(2);
         result = x-((x/2)*2);
      }
      if(bit==2){
          int x=data/(4);
         result = x-((x/2)*2);
      }
      if(bit==3){
          int x=data/(8);
         result = x-((x/2)*2);
      }
      if(bit==4){
          int x=data/(16);
         result = x-((x/2)*2);
      }

      if(bit==5){
          int x=data/(32);
         result = x-((x/2)*2);
      }
```

```
        if(bit==6){
            int x=data/(64);
           result = x-((x/2)*2);
        }

        if(bit==7){
           result =data/(128);
        }
        return result;
}
```