Rapport i Digitala Projekt (EDI021)

Grupp 6
Daniel Raneland, dt05dr1

2009-03-01

1 Sammanfattning

Idag är det väldigt vanligt att man har en termometer hemma som mäter temperaturen både inomhus och utomhus. Det blir allt vanligare med avancerade funktioner på dessa, såsom tryckmätare eller statistik. Mitt projekt gick ut på att bygga en digital termometer som ritar upp en graf över temperaturen de senaste 12 timmarna.

Själva termometern styrs av en AVR ATMega16, en microprocessor med bland annat inbyggt minne och AD-omvandlare. Projektet resulterade i en prototyp som skulle kunna serietillverkas, prototypen är ihopsatt på ett kopplingsbräde och komponenterna är sammankopplade med sladdar.

Innehåll

2 Inledning

Kursen Digitala Project (EDI021) är en praktisk kurs där man designar och bygger ett digitalt system, själva systemet konstrueras som en prototyp på ett kopplingsbräde.

Eftersom jag inte har någon som helst erfarenhet av digitala projekt tidigare valde jag att göra något ganska enkelt - en väderstation. Detta kändes som en ungefär lagom utmaning då det innefattar ganska få digitala och analoga komponenter, men ända tillräckligt många för att man ska lära sig något utav det.

2.1 Krav

De inledande kraven på väderstationen var att den skulle kunna:

- Avläsa inohumtemeperatur och visa denna
- Avläsa utomhustemperatur och visa denna
- Spara högsta/lägsta inomhustemperatur och visa dessa
- Spara högsta/lägsta utomhustemperatur och visa dessa
- Spara största skillnaden mellan utomhus- och inomhustemperatur och visa denna

Under projektets gång ändrades dock detta till att väderstationen ska kunna:

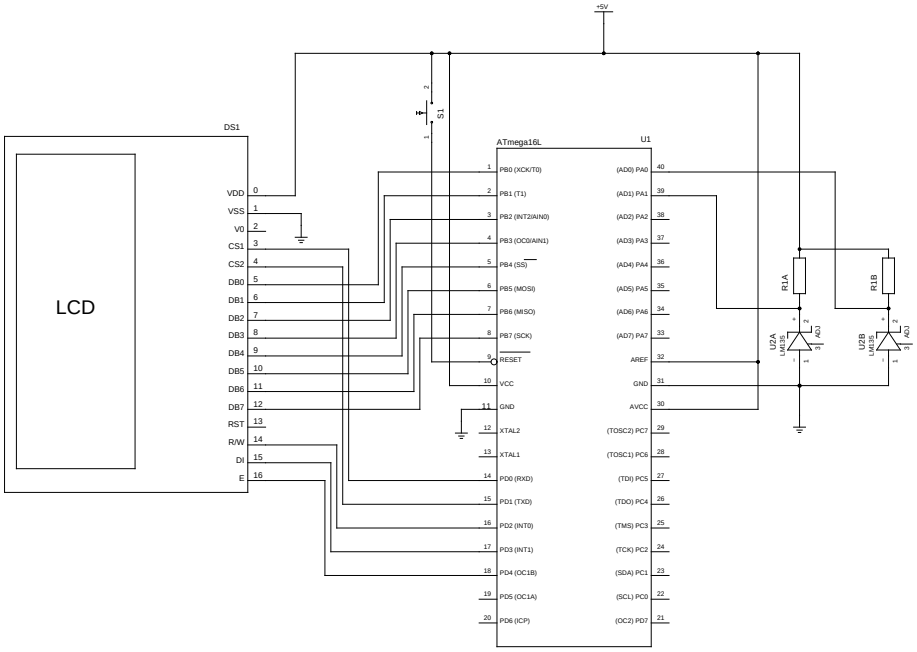
- Avläsa inohumtemeperatur och visa denna
- Avläsa utomhustemperatur och visa denna
- Rita upp en graf över uppmätta temperaturer de senaste 12 timmarna

2.2 Komponenter

De komponenter som ingår i väderstation är följande:

- 1st AVR ATMega16 microprocessor
- 1st vanlig tryckknapp
- 1st färdigkopplad 128x64 LCD display med anslutna kontrollers
- 2st National Semiconductor LM335 temperaturmätare
- Några motstånd

Komponenterna är ihopkopplade enligt figur 1.



Figur 1: Kopplingsschema för väderstationen

2.2.1 AVR Atmega16

AVR Atmega16 är en komplett microprocessor som innehåller bland annat minne, AD-omvandlare och 3 st digitala räknare som kan användas för att generera periodiska avbrott. Microprocessorn kan arbeta i 1, 2, 4, eller 8 Mhz och den fria mjukvaran AVR Studio gör att det är mycket enkelt att arbeta med den.

2.2.2 LM335

LM335 är en temperaturmätare från National Semiconductor som arbetar linjärt i Kelvin. Detta innebär att man väldigt enkelt kan avläsa temperaturen med den inbyggda AD-omvandlaren på ATMega16.

2.2.3 LCD-display

Displayen är av okänt märke och sitter ihopkopplad med två st KS108B kontrollkretsar. Dessa styr vars en halva av displayen och kommunikationen sker via tre styrsignaler och en databuss på åtta bitar.

3 Konstruktion

3.1 Hårdvara

De fysiska komponenterna i väderstationen sattes ihop på ett kopplingsbräde och virades ihop för att det skulle vara lätt att ändra om något blev fel. Genom att inte använda de ben på microprocessorn som börjar på PD kunde jag koppla in microprocessorn till datorn via ett *JTAG* interface[1]. Detta gjorde att det blev mycket smidigt att debugga kod och testa kommunikationen med de andra komponenterna.

3.2 Mjukvara

Det enda i väderstationen som behövde programmeras var microprocessorn, detta gjordes i C. Programmet är uppdelat i två headerfiler och tre källkodsfiler. Dessa listas i tabell 1.

Själva programmet fungerar så att microprocessorn läser av de analoga signalerna kontinuerligt medans uppdateringen av displayen är kopplat till ett avbrott som sker varje sekund. I avbrottsrutinen räknas en intern klocka upp och ett sampel tas för grafen om det har gått 432 sekunder sedan det förra samplet togs[2]. Hela källkoden till programmet finns i appendix A.

[1] JTAG står för Joined Test Action Group och är en standard för att ansluta till inbyggda system

[2] Eftersom grafen består av 100 sampel innebär 432 sekunder att grafen täcker 12 timmar

Namn	Beskrivning
ascii.h	Deklarerar de funktioner som finns i *ascii.c*
lcd.h	Deklarerar de funktioner som finsn i *lch.c*
ascii.c	Definierar funktioner för att skriva ascii-tecken till LCD-displayen
lcd.c	Definierar funktioner för att kommunicera med LCD-displayen
main.c	Definierar main-metoden samt metoder för att initialisera microprocessorn och hantera avbrott

Tabell 1: Förklaring av de filer som programmet består av

4 Vidareutveckling

Programmet till microprocessorn är skrivet så att man lätt kan byta ut funktionaliteten, bland annat så är rutinerna för LCD-displayen helt generella och går att använda i andra projekt med liknande sammankoppling, man behöver inte ens koppla in LCD-displayen på samma portar utan dessa ställs in i headerfilen *lcd.h*.

Det är mycket enkelt att koppla in flera analoga signaler till väderstationen, själva programmet stödjer upp till åtta insignaler genom att ändra en variabel. Dock måste man ändra lite i koden om man inte har linjära insignaler.

5 Resultat

Själva väderstationen fungerar utmärkt, på grund av att kabeln mellan LCD-displayen och microprocessorn är för lång syns däremot en del störningar på displayen, dessa försvinner dock om man byter till en kortare kabel.

Eftersom kraven på väderstationen ändrades under projektets gång, detta eftersom jag fick en bättre inblick i vad jag egentligen ville göra med min väderstation, så uppfyller väderstationen bara de senare listade kraven.

Överlag så är jag mycket nöjd med projektet, det har varit roligt att göra något praktiskt och jag känner att jag har lärt mig väldigt mycket.

6 Referenser

Datablad för ATMega16
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Processors/ATmega16.pdf
Datablad för LM335
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Sensors/lm335.pdf
Datablad för KS108B
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Display/ks0108b.pdf

A Källkod

Här listas källkoden för väderstationen.

A.1 ascii.h

```
#ifndef ASCII_H
#define ASCII_H
#include <avr/io.h>

#define ERROR_CHAR 93
#define START_CHAR 32
#define CHAR_WIDTH 5
#define CHAR_HEIGHT 8

void asciitest();
void putChar( const char c );

#endif // ASCII_H
```

A.2 lcd.h

```
#ifndef LCD_H
#define LCD_H
#include <avr/io.h>

// Define LCD Commands
#define LCDPORT PORTB
#define LCDCMD PORTD
#define CS1 0
#define CS2 1
#define RW 2
#define RS 3
#define E 4
#define RESET 5

// Define window boundaries
#define WINDOW_HEIGHT 9
#define WINDOW_WIDTH 16

// The active controller
static volatile uint8_t CS = CS1;
static volatile uint8_t X;
static volatile uint8_t Y;

void putString( const char* c );
```

```
void display( uint8_t d );
void wait( uint8_t c );
void clearLCD( void );
void move( uint8_t x, uint8_t y );
void write( uint8_t byte );

#endif // LCD_H
```

A.3 ascii.c

```
#include <avr/io.h>
#include "ascii.h"
#include "lcd.h"

/**
 * Outputs all printable characters in the ASCII table
 * tot he LCD. This is useful when debugging changes in
 * the character definitions.
 */
void asciitest() {
        clearLCD();
        move( 0, 0 );
        putString( " !\"#$%'&()*+,-./01234" );
        move( 0, 1 );
        putString( "56789:;<=>?@ABCDEFGHI" );
        move( 0, 2 );
        putString( "JKLMNOPQRSTUVXYZ[\\]^_" );
        move( 0, 3 );
        putString( "`abcdefghijklmnopqrst" );
        move( 0, 4 );
        putString( "uvxyz{|}~" );
}

/**
 * Output a character to the LCD.
 * This method supports all printable characters
 * in the ASCII table, to reduce the amount of code,
 * lowercase letters are converted to upper case.
 *
 * @param c A character
 */
void putChar( const char c ) {
    uint8_t num;
    if ( 'a' <= c && c <= 'z' ) {
        num = c - START_CHAR + 'A' - 'a';
    }
```

```
else {
    num = c - START_CHAR;
}
switch ( num ) {
    case 0:
        write( 0x00 ); // , , , , , , , ,
        write( 0x00 ); // , , , , , , , ,
        write( 0x00 ); // , , , , , , , ,
        write( 0x00 ); // , , , , , , , ,
        write( 0x00 ); // , , , , , , , ,
        break;
    case 1:
        write( 0x00 ); // , , , , , , , ,
        write( 0x00 ); // , , , , , , , ,
        write( 0x5F ); // , # , # # # # #
        write( 0x00 ); // , , , , , , , ,
        write( 0x00 ); // , , , , , , , ,
        break;
    case 2:
        write( 0x00 ); // , , , , , , , ,
        write( 0x07 ); // , , , , , # # #
        write( 0x00 ); // , , , , , , , ,
        write( 0x00 ); // , , , , , , , ,
        write( 0x07 ); // , , , , , # # #
        break;
    case 3:
        write( 0x54 ); // , # , # , # , ,
        write( 0x3E ); // , , # # # # # ,
        write( 0x55 ); // , # , # , # , #
        write( 0x3E ); // , , # # # # # ,
        write( 0x15 ); // , , , # , # , #
        break;
    case 4:
        write( 0x04 ); // , , , , , # , ,
        write( 0x2A ); // , , # , # , # ,
        write( 0x7F ); // , # # # # # # #
        write( 0x2A ); // , , # , # , # ,
        write( 0x10 ); // , , , # , , , ,
        break;
    case 5:
        write( 0x22 ); // , , # , , , # ,
        write( 0x15 ); // , , , # , # , #
        write( 0x2A ); // , , # , # , # ,
        write( 0x54 ); // , # , # , # , ,
        write( 0x22 ); // , , # , , , # ,
        break;
```

```
case 6:
    write( 0x00 ); // , , , , , , , ,
    write( 0x00 ); // , , , , , , , ,
    write( 0x07 ); // , , , , , # # #
    write( 0x00 ); // , , , , , , , ,
    write( 0x00 ); // , , , , , , , ,
    break;
case 7:
    write( 0x38 ); // , , # # # , , ,
    write( 0x44 ); // , # , , , # , ,
    write( 0x52 ); // , # , # , , # ,
    write( 0x22 ); // , , # , , , # ,
    write( 0x50 ); // , # , # , , , ,
    break;
case 8:
    write( 0x00 ); // , , , , , , , ,
    write( 0x1C ); // , , , # # # , ,
    write( 0x22 ); // , , # , , , # ,
    write( 0x41 ); // , # , , , , , #
    write( 0x00 ); // , , , , , , , ,
    break;
case 9:
    write( 0x00 ); // , , , , , , , ,
    write( 0x41 ); // , # , , , , , #
    write( 0x22 ); // , , # , , , # ,
    write( 0x1C ); // , , , # # # , ,
    write( 0x00 ); // , , , , , , , ,
    break;
case 10:
    write( 0x0A ); // , , , , # , # ,
    write( 0x06 ); // , , , , , # # ,
    write( 0x0F ); // , , , , # # # #
    write( 0x06 ); // , , , , , # # ,
    write( 0x0A ); // , , , , # , # ,
    break;
case 11:
    write( 0x08 ); // , , , , # , , ,
    write( 0x08 ); // , , , , # , , ,
    write( 0x3E ); // , , # # # # # ,
    write( 0x08 ); // , , , , # , , ,
    write( 0x08 ); // , , , , # , , ,
    break;
case 12:
    write( 0x00 ); // , , , , , , , ,
    write( 0x00 ); // , , , , , , , ,
    write( 0xE0 ); // # # # , , , , ,
```

```
    write( 0x60 ); // , # # , , , , ,
    write( 0x00 ); // , , , , , , , ,
    break;
case 13:
    write( 0x08 ); // , , , , # , , ,
    write( 0x08 ); // , , , , # , , ,
    write( 0x08 ); // , , , , # , , ,
    write( 0x08 ); // , , , , # , , ,
    write( 0x08 ); // , , , , # , , ,
    break;
case 14:
    write( 0x00 ); // , , , , , , , ,
    write( 0x00 ); // , , , , , , , ,
    write( 0x60 ); // , # # , , , , ,
    write( 0x60 ); // , # # , , , , ,
    write( 0x00 ); // , , , , , , , ,
    break;
case 15:
    write( 0x00 ); // , , , , , , , ,
    write( 0x60 ); // , # # , , , , ,
    write( 0x18 ); // , , , # # , , ,
    write( 0x06 ); // , , , , , # # ,
    write( 0x01 ); // , , , , , , , #
    break;
case 16:
    write( 0x3E ); // , , # # # # # ,
    write( 0x51 ); // , # , # , , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x45 ); // , # , , , # , #
    write( 0x3E ); // , , # # # # # ,
    break;
case 17:
    write( 0x00 ); // , , , , , , , ,
    write( 0x42 ); // , # , , , , # ,
    write( 0x7F ); // , # # # # # # #
    write( 0x40 ); // , # , , , , , ,
    write( 0x00 ); // , , , , , , , ,
    break;
case 18:
    write( 0x46 ); // , # , , , # # ,
    write( 0x61 ); // , # # , , , , #
    write( 0x51 ); // , # , # , , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x46 ); // , # , , , # # ,
    break;
case 19:
```

```
    write( 0x41 ); // , # , , , , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x36 ); // , , # # , # # ,
    break;
case 20:
    write( 0x30 ); // , , # # , , , ,
    write( 0x2C ); // , , # , # # , ,
    write( 0x22 ); // , , # , , , # ,
    write( 0x7F ); // , # # # # # # #
    write( 0x20 ); // , , # , , , , ,
    break;
case 21:
    write( 0x4F ); // , # , , # # # #
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x30 ); // , , # # , , , ,
    break;
case 22:
    write( 0x3E ); // , , # # # # # ,
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x30 ); // , , # # , , , ,
    break;
case 23:
    write( 0x01 ); // , , , , , , , #
    write( 0x01 ); // , , , , , , , #
    write( 0x79 ); // , # # # # , , #
    write( 0x05 ); // , , , , , # , #
    write( 0x02 ); // , , , , , , # ,
    break;
case 24:
    write( 0x36 ); // , , # # , # # ,
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x36 ); // , , # # , # # ,
    break;
case 25:
    write( 0x06 ); // , , , , , # # ,
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
```

```
    write( 0x3E ); // , , # # # # # ,
    break;
case 26:
    write( 0x00 ); // , , , , , , , ,
    write( 0x00 ); // , , , , , , , ,
    write( 0x6C ); // , # # , # # , ,
    write( 0x6C ); // , # # , # # , ,
    write( 0x00 ); // , , , , , , , ,
    break;
case 27:
    write( 0x00 ); // , , , , , , , ,
    write( 0x00 ); // , , , , , , , ,
    write( 0xEC ); // # # # , # # , ,
    write( 0x6C ); // , # # , # # , ,
    write( 0x00 ); // , , , , , , , ,
    write( 0x00 ); // , , , , , , , ,
    break;
case 28:
    write( 0x08 ); // , , , , # , , ,
    write( 0x14 ); // , , , # , # , ,
    write( 0x14 ); // , , , # , # , ,
    write( 0x22 ); // , , # , , , # ,
    write( 0x22 ); // , , # , , , # ,
    break;
case 29:
    write( 0x14 ); // , , , # , # , ,
    write( 0x14 ); // , , , # , # , ,
    write( 0x14 ); // , , , # , # , ,
    write( 0x14 ); // , , , # , # , ,
    write( 0x14 ); // , , , # , # , ,
    break;
case 30:
    write( 0x22 ); // , , # , , , # ,
    write( 0x22 ); // , , # , , , # ,
    write( 0x14 ); // , , , # , # , ,
    write( 0x14 ); // , , , # , # , ,
    write( 0x08 ); // , , , , # , , ,
    break;
case 31:
    write( 0x02 ); // , , , , , , # ,
    write( 0x01 ); // , , , , , , , #
    write( 0x59 ); // , # , # # , , #
    write( 0x05 ); // , , , , , # , #
    write( 0x02 ); // , , , , , , # ,
    break;
case 32:
```

```
    write( 0x3E ); // , , # # # # # ,
    write( 0x41 ); // , # , , , , , #
    write( 0x4D ); // , # , , # # , #
    write( 0x4D ); // , # , , # # , #
    write( 0x2E ); // , , # , # # # ,
    break;
case 33:
    write( 0x7E ); // , # # # # # # ,
    write( 0x09 ); // , , , , # , , #
    write( 0x09 ); // , , , , # , , #
    write( 0x09 ); // , , , , # , , #
    write( 0x7E ); // , # # # # # # ,
    break;
case 34:
    write( 0x7F ); // , # # # # # # #
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x36 ); // , , # # , # # ,
    break;
case 35:
    write( 0x1C ); // , , , # # # , ,
    write( 0x22 ); // , , # , , , # ,
    write( 0x41 ); // , # , , , , , #
    write( 0x41 ); // , # , , , , , #
    write( 0x41 ); // , # , , , , , #
    break;
case 36:
    write( 0x7F ); // , # # # # # # #
    write( 0x41 ); // , # , , , , , #
    write( 0x41 ); // , # , , , , , #
    write( 0x22 ); // , , # , , , # ,
    write( 0x1C ); // , , , # # # , ,
    break;
case 37:
    write( 0x7F ); // , # # # # # # #
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x41 ); // , # , , , , , #
    write( 0x41 ); // , # , , , , , #
    break;
case 38:
    write( 0x7F ); // , # # # # # # #
    write( 0x09 ); // , , , , # , , #
    write( 0x09 ); // , , , , # , , #
    write( 0x01 ); // , , , , , , , #
```

```
    write( 0x01 ); // , , , , , , , #
    break;
case 39:
    write( 0x1C ); // , , , # # # , ,
    write( 0x22 ); // , , # , , , # ,
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x79 ); // , # # # # , , #
    break;
case 40:
    write( 0x7F ); // , # # # # # # #
    write( 0x08 ); // , , , , # , , ,
    write( 0x08 ); // , , , , # , , ,
    write( 0x08 ); // , , , , # , , ,
    write( 0x7F ); // , # # # # # # #
    break;
case 41:
    write( 0x41 ); // , # , , , , , #
    write( 0x41 ); // , # , , , , , #
    write( 0x7F ); // , # # # # # # #
    write( 0x41 ); // , # , , , , , #
    write( 0x41 ); // , # , , , , , #
    break;
case 42:
    write( 0x41 ); // , # , , , , , #
    write( 0x41 ); // , # , , , , , #
    write( 0x41 ); // , # , , , , , #
    write( 0x21 ); // , , # , , , , #
    write( 0x1F ); // , , , # # # # #
    break;
case 43:
    write( 0x7F ); // , # # # # # # #
    write( 0x08 ); // , , , , # , , ,
    write( 0x14 ); // , , , # , # , ,
    write( 0x22 ); // , , # , , , # ,
    write( 0x41 ); // , # , , , , , #
    break;
case 44:
    write( 0x7F ); // , # # # # # # #
    write( 0x40 ); // , # , , , , , ,
    write( 0x40 ); // , # , , , , , ,
    write( 0x40 ); // , # , , , , , ,
    write( 0x40 ); // , # , , , , , ,
    break;
case 45:
    write( 0x7F ); // , # # # # # # #
```

```
    write( 0x06 ); // , , , , , # # ,
    write( 0x18 ); // , , , # # , , ,
    write( 0x06 ); // , , , , , # # ,
    write( 0x7F ); // , # # # # # # #
    break;
case 46:
    write( 0x7F ); // , # # # # # # #
    write( 0x03 ); // , , , , , , # #
    write( 0x1C ); // , , , # # # , ,
    write( 0x60 ); // , # # , , , , ,
    write( 0x7F ); // , # # # # # # #
    break;
case 47:
    write( 0x3E ); // , , # # # # # ,
    write( 0x41 ); // , # , , , , , #
    write( 0x41 ); // , # , , , , , #
    write( 0x41 ); // , # , , , , , #
    write( 0x3E ); // , , # # # # # ,
    break;
case 48:
    write( 0x7F ); // , # # # # # # #
    write( 0x09 ); // , , , , # , , #
    write( 0x09 ); // , , , , # , , #
    write( 0x09 ); // , , , , # , , #
    write( 0x06 ); // , , , , , # # ,
    break;
case 49:
    write( 0x3E ); // , , # # # # # ,
    write( 0x41 ); // , # , , , , , #
    write( 0x51 ); // , # , # , , , #
    write( 0x61 ); // , # # , , , , #
    write( 0x7E ); // , # # # # # # ,
    break;
case 50:
       write( 0x7F ); // , # # # # # # #
    write( 0x09 ); // , , , , # , , #
    write( 0x09 ); // , , , , # , , #
    write( 0x09 ); // , , , , # , , #
    write( 0x76 ); // , # # # , # # ,
    break;
case 51:
    write( 0x26 ); // , , # , , # # ,
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x49 ); // , # , , # , , #
    write( 0x32 ); // , , # # , , # ,
```

```
    break;
case 52:
    write( 0x01 ); // , , , , , , , #
    write( 0x01 ); // , , , , , , , #
    write( 0x7F ); // , # # # # # # #
    write( 0x01 ); // , , , , , , , #
    write( 0x01 ); // , , , , , , , #
    break;
case 53:
    write( 0x3F ); // , , # # # # # #
    write( 0x40 ); // , # , , , , , ,
    write( 0x40 ); // , # , , , , , ,
    write( 0x40 ); // , # , , , , , ,
    write( 0x3F ); // , , # # # # # #
    break;
case 54:
    write( 0x07 ); // , , , , , # # #
    write( 0x38 ); // , , # # # , , ,
    write( 0x40 ); // , # , , , , , ,
    write( 0x38 ); //, , # # # , , ,
    write( 0x07 ); // , , , , , # # #
    break;
case 55:
    write( 0x3F ); // , , # # # # # #
    write( 0x40 ); // , # , , , , , ,
    write( 0x38 ); // , , # # # , , ,
    write( 0x40 ); // , # , , , , , ,
    write( 0x3F ); // , , # # # # # #
    break;
case 56:
    write( 0x63 ); // , # # , , , # #
    write( 0x14 ); // , , , # , # , ,
    write( 0x08 ); // , , , , # , , ,
    write( 0x14 ); // , , , # , # , ,
    write( 0x63 ); // , # # , , , # #
    break;
case 57:
    write( 0x07 ); // , , , , , # # #
    write( 0x08 ); // , , , , # , , ,
    write( 0x70 ); // , # # # , , , ,
    write( 0x08 ); // , , , , # , , ,
    write( 0x07 ); // , , , , , # # #
    break;
case 58:
    write( 0x61 ); // , # # , , , , #
    write( 0x51 ); // , # , # , , , #
```

```
    write( 0x49 ); // , # , , # , , #
    write( 0x45 ); // , # , , , # , #
    write( 0x43 ); // , # , , , , # #
    break;
case 59:
    write( 0x00 ); // , , , , , , , ,
    write( 0x7F ); // , # # # # # # #
    write( 0x41 ); // , # , , , , , #
    write( 0x41 ); // , # , , , , , #
    write( 0x00 ); // , , , , , , , ,
    break;
case 60:
    write( 0x01 ); // , , , , , , , #
    write( 0x06 ); // , , , , , # # ,
    write( 0x18 ); // , , , # # , , ,
    write( 0x60 ); // , # # , , , , ,
    write( 0x00 ); // , , , , , , , ,
    break;
case 61:
    write( 0x00 ); // , , , , , , , ,
    write( 0x41 ); // , # , , , , , #
    write( 0x41 ); // , # , , , , , #
    write( 0x7F ); // , # # # # # # #
    write( 0x00 ); // , , , , , , , ,
    break;
case 62:
    write( 0x04 ); // , , , , , # , ,
    write( 0x02 ); // , , , , , , # ,
    write( 0x01 ); // , , , , , , , #
    write( 0x02 ); // , , , , , , # ,
    write( 0x04 ); // , , , , , # , ,
    break;
case 63:
    write( 0x40 ); // , # , , , , , ,
    write( 0x40 ); // , # , , , , , ,
    write( 0x40 ); // , # , , , , , ,
    write( 0x40 ); // , # , , , , , ,
    write( 0x40 ); // , # , , , , , ,
    break;
case 64:
    write( 0x00 ); // , , , , , , , ,
    write( 0x01 ); // , , , , , , , #
    write( 0x02 ); // , , , , , , # ,
    write( 0x04 ); // , , , , , # , ,
    write( 0x00 ); // , , , , , , , ,
    break;
```

```
            // a-z goes here, lowercase letters are converted to uppercase
            // and definiton of lowercase letters is therefore optional.
            // Don't forget to remove the conversion from lowercase to uppercase
            // if adding lowercase letters
        case 91:
            write( 0x00 ); // , , , , , , , ,
            write( 0x08 ); // , , , , # , , ,
            write( 0x36 ); // , , # # , # # ,
            write( 0x41 ); // , # , , , , , #
            write( 0x00 ); // , , , , , , , ,
            break;
        case 92:
            write( 0x00 ); // , , , , , , , ,
            write( 0x00 ); // , , , , , , , ,
            write( 0x7F ); // , # # # # # # #
            write( 0x00 ); // , , , , , , , ,
            write( 0x00 ); // , , , , , , , ,
            break;
        case 93:
            write( 0x00 ); // , , , , , , , ,
            write( 0x41 ); // , # , , , , , #
            write( 0x36 ); // , , # # , # # ,
            write( 0x08 ); // , , , , # , , ,
            write( 0x00 ); // , , , , , , , ,
            break;
        case 94:
            write( 0x08 ); // , , , , # , , ,
            write( 0x04 ); // , , , , , # , ,
            write( 0x04 ); // , , , , , # , ,
            write( 0x08 ); // , , , , # , , ,
            write( 0x04 ); // , , , , , # , ,
            break;
        default:
            write( 0x7F ); // , # # # # # # #
            write( 0x41 ); // , # , , , , , #
            write( 0x41 ); // , # , , , , , #
            write( 0x41 ); // , # , , , , , #
            write( 0x7F ); // , # # # # # # #
            break;
    }
}
```

A.4 lcd.c

```
#include "lcd.h"
#include "ascii.h"
```

```
#include <avr/io.h>
#include <avr/interrupt.h>

/**
 * Turns on/off the display on the LCD.
 * @param d If the display should be on or off (1 or 0)
 */
void display( uint8_t d ) {
    cli();
    LCDPORT = ( 0x3E | d );
    LCDCMD = ( 1<<RESET | 1<<CS1 | 1<<CS2 );
    LCDCMD |= 1<<E;
    LCDCMD = ( 1<<RESET | 1<<CS1 | 1<<CS2 );
    sei();

}

/**
 * Print a string characters on the screen.
 * Prints a string of characters from the ASCII table on the screen.
 *
 * @param s An null-terminated string of ASCII characters
 */
void putString( const char s[] ) {
    uint8_t i = 0;
    char c = s[0];
    while ( c != '\0' ) {
        // Write the char
        putChar( c );
        // Spacing between chars
        write( 0 );
        i++;
        c = s[i];
    }
}

/**
 * Clear the LCD
 * Clears the LCD by writing zeroes to all pages
 */
void clearLCD( void ) {
    uint8_t x = 0;
    uint8_t y = 0;
    for ( y = 0; y < 8; y++ ) {
        move( 0, y );
        for ( x = 0; x < 128; x++ ) {
```

```
            write( 0 );
        }
    }
}

/**
 * Set the LCD to a specific page.
 * @param x X-coordinate, 0-128
 * @param y Y-cooridnate, 0-8
 */
void move( uint8_t x, uint8_t y ) {
    // Set x
    X = x;
    Y = y;
    if ( x < 64 ) {
        CS = CS1;
    }
    else {
        CS = CS2;
        x = x-64;
    }
    cli();
    LCDPORT = ( 1<<6 | x );
    LCDCMD = ( 1<<RESET | 1<<CS );
    LCDCMD = ( 1<<RESET | 1<<CS | 1<<E );
    LCDCMD = ( 1<<RESET | 1<<CS );

    // Set y
    LCDPORT = ( 1<<7 | 1<<5 | 1<<4 | 1<<3 | y );
    LCDCMD = ( 1<<RESET | 1<<CS );
    LCDCMD = ( 1<<RESET | 1<<CS | 1<<E );
    LCDCMD = ( 1<<RESET | 1<<CS );
    sei();

}

/**
 * Wait for a LCD controller to be ready.
 * Sets the LCD controller in status mode and waits until
 * the busy flag is low.
 *
 * @param c The controller number (0 or 1)
 */
void wait( uint8_t c ) {
    // Set PORTB to input
    DDRB = 0;
```

```
    uint8_t cs;
    if ( c == 0 ) {
        cs = CS1;
    }
    else {
        cs = CS2;
    }


    uint8_t flag = 0x80;

    // Wait for busy flag to go low
    while ( flag & 0x80 ) {
        cli();
        // Enable the controller and tell it to output status
        LCDCMD = 1<<cs | 1<<RW | 1<<RESET;
        LCDCMD = 1<<cs | 1<<RW | 1<<RESET | 1<<E;
        LCDCMD = 1<<cs | 1<<RW | 1<<RESET;
        flag = PINB;
        sei();

    }


    // Set PORTB to output again
    DDRB = 0xFF;
}

/**
 * Write display data to an LCD controller.
 * Writes the screen to the LCD controller.
 *
 * @param c The controller number (0 or 1)
 */
void write( uint8_t byte ) {

    cli();
    LCDPORT = byte;
    // Write the data
    LCDCMD = ( 1<<CS | 1<<RS | 1<<RESET );
    LCDCMD = ( 1<<CS | 1<<RS | 1<<RESET | 1<<E );
    LCDCMD = ( 1<<CS | 1<<RS | 1<<RESET );
    sei();
    //wait( CS );
    X++;
    if ( X == 64 ) {
```

```
        move( X, Y );
    }

}
```

A.5 main.c

```
#define F_CPU 4000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdio.h>
#include "ascii.h"
#include "lcd.h"

#define NUM_INPUTS 2
// This will give us 12h of history (graph coverage)
#define SAMPLE_INTERVAL 432
#define NUM_SAMPLES 100
#define GRAPH_START 25
#define GRAPH_BOTTOM -20
#define GRAPH_STEP 3
#define GRAPH_TOP 49
#define GRAPH_HEIGHT 23
static float VperKelvin[NUM_INPUTS] = { 0.0101, 0.01005 };
static uint8_t AdcInput = 0;
static float Temps[NUM_INPUTS];
static uint8_t Fake[15] = { 6, 5, 12, 20, 13, 23, 7, 1, 0, 2, 3, 5, 12, 9, 7 };
static uint8_t Samples[NUM_INPUTS][NUM_SAMPLES];
static uint8_t Hours = 0;
static uint8_t Minutes = 0;
static uint8_t Seconds = 0;
static uint16_t Count = 0;
static uint8_t SamplesTaken = 0;
static uint8_t n = 0;

void initADC( void ) {
    // Set Vref to AREF, keep result right-justified and set input source to ADC0
    ADMUX = 0;
    // Enable the ADC, start the first conversion and enable interrupts
    ADCSRA = 1<<ADEN | 1<<ADSC;
    // Wait for the first conversion to complete and discard it
    while ( ( ADCSRA & (1<<ADIF) ) == 0 ) {
        // Reset the flag
        ADCSRA = ( ADCSRA | 1<<ADIF );
        // Start the next conversion
```

```
        ADCSRA = ( ADCSRA | 1<<ADSC );
    }
}


void initInterrupts( void ) {

    // Enable global interrupts
    sei();

    // Enable the timer1 compare interrupt A
    TIMSK = TIMSK | 1<<OCIE1A;
}


void initTimers( void ) {

    // Start timer 1, set to CTC (OCR1A) with 1/256 prescaler
    TCCR1A = 0;
    TCCR1B = 1 <<WGM12 | 1<<CS12;
    // Set the top value for timer1A (interrupt is caused when this value is hit)
    // 4Mhz / 256 / 15625 = 1Hz
    OCR1A = 15625;

    // Reset timer 1
    TCNT1 = 0;
}

void initPorts( void ) {
    // Setup the LCD data buss
    DDRB = 0xFF;
    DDRD = 1<<CS1 | 1<<CS2 | 1<<RESET | 1<<E | 1<<RS | 1<<RW;
    DDRA = 0;
    PORTA = ~( 1<<1 | 1 );
    LCDPORT = 0;
    LCDCMD = 0;
}

/**
 * Display the graph and temperature readings on the LCD
 */
void redraw( void ) {

    clearLCD();

    int8_t in = Temps[0];
```

```
int8_t out = Temps[1];

// In temperature
move( 0, 0 );
putString( "In:" );
// Handle negative numbers
if ( in < 0 ) {
    in = -in;
    putChar( '-' );
}
if ( in >= 10 ) {
    putChar( (uint8_t) ( in / 10 ) + '0' );
    write( 0 );
}
putChar( (uint8_t) in % 10 + '0' );
write( 0 );
putChar( 'C' );
// This erases any trailing characters
putChar( ' ' );
putChar( ' ' );

// Out temperature
move ( 0, 4 );
putString( "Out:" );
// Handle negative numbers
if ( out < 0 ) {
    out = -out;
    putChar( '-' );
}
if ( out >= 10 ) {
    putChar( (uint8_t) ( out / 10 ) + '0' );
    write( 0 );
}
putChar( (uint8_t) out % 10 + '0' );
write( 0 );
putChar( 'C' );
// This erases any trailing characters
putChar( ' ' );
putChar( ' ' );

// Output the Graphs
// Labels
move( 0, 1 );
putString( "+50" );
// A _ at the top of the row
write( 0x1 );
```

```
write( 0x1 );
write( 0x1 );
write( 0x1 );
write( 0x1 );
move( 0, 3 );
putString( "-20_" );
move( 0, 5 );
putString( "+50" );
// A _ at the top of the row
write( 0x1 );
write( 0x1 );
write( 0x1 );
write( 0x1 );
write( 0x1 );
move( 0, 7 );
putString( "-20_" );

// Draw the graphs
uint8_t page;
uint8_t i;
uint8_t x;
uint8_t height;
int8_t bits;
uint8_t byte;

// Loop through inputs and samples, outputting each graph one page
// at a time
for ( i = 0; i < NUM_INPUTS; i++ ) {
    for( page = 0; page < 3; page++ ) {
        move( GRAPH_START, 3 - page + 4 * i );
        for ( x = 0; x < SamplesTaken; x++ ) {
            height = Samples[i][x] + 1;
            // Calculate how many bits we should show in this page
            bits = height - page * 8;
            // A page is maximum 8 bits
            if ( bits > 8 ) {
                bits = 8;
            }
            // We must output 0 bits, else the graph will be crooked
            else if ( bits <= 0 ) {
                bits = 0;
            }
            // Output the page, adjust for empty row at the bottom page
            byte = 0xFF<<(8 - bits);
            if ( page == 0 ) {
                byte &= 0x7F;
```

```
                }
                write( byte );
            }
        }
    }
}

 int main( void ) {

    initPorts();
    initTimers();
    initADC();
    initInterrupts();

    // Turn on the display
    display( 1 );
    while ( 1 ) {

        // Check if conversion is complete
        if ( ADCSRA & (1<<ADIF) ) {
            // Reset the flag
            ADCSRA = ( ADCSRA | 1<<ADIF );

            // Adjust the input, give the old value a significance of 99% and the new va
            // This will provide a more stable value less influenced by analog noise
            Temps[AdcInput] = Temps[AdcInput] * 0.9 + ( ADC * 5.24 / 1024 / VperKelvin[A

            // Increase the input, wrap if overflow
            AdcInput = ( AdcInput + 1 ) % NUM_INPUTS;

            // Set the ADMUX to the new adcInput
            ADMUX = ( ADMUX & ( BIT7 | BIT6 | BIT5 ) ) | AdcInput;

            // Start the next conversion
            ADCSRA = ( ADCSRA | 1<<ADSC );
        }
        _delay_ms( 50 );
    }
}

/**
 * Handle interrupt when counter 1 has reached TOP.
 * This method increases the clock with one second and updates
 * the LCD. If enough time have passed a sample is also taken.
 */
void SIG_OUTPUT_COMPARE1A( void ) {
```

```
Seconds += 1;
if ( Seconds == 60 ) {
    Seconds = 0;
    Minutes += 1;
    if ( Minutes == 60 ) {
        Minutes = 0;
        Hours = ( Hours + 1 ) % 24;
    }
}

// Increase count
Count++;
// Take a sample for the graph
if ( Count == SAMPLE_INTERVAL ) {
    float t;
    n = ( n + 1 ) % 15;
    if ( SamplesTaken < NUM_SAMPLES ) {
        // Append samples
        uint8_t i = 0;

        for ( i = 0; i < NUM_INPUTS; i++ ) {
            t = Temps[i];
            if ( t < GRAPH_BOTTOM ) {
                t = GRAPH_BOTTOM;
            }
            else if ( t > GRAPH_TOP ) {
                t = GRAPH_TOP;
            }
            Samples[i][SamplesTaken] = (int) ( ( t - GRAPH_BOTTOM ) / GRAPH_STEP );
        }
        SamplesTaken++;
    }
    else {
        // Push Samples backwards
        uint8_t i = 0;
        uint8_t j = 0;
        for ( i = 0; i < NUM_INPUTS; i++ ) {
            for( j = 0; j < NUM_SAMPLES - 1; j++ ) {
                Samples[i][j] = Samples[i][j+1];
            }
            t = Temps[i];
            if ( t < GRAPH_BOTTOM ) {
                t = GRAPH_BOTTOM;
            }
            else if ( t > GRAPH_TOP ) {
                t = GRAPH_TOP;
```

```
                }
                Samples[i][NUM_SAMPLES-1] = (int) ( ( t - GRAPH_BOTTOM ) / GRAPH_STEP );
            }
        }
        Count = 0;
    }
    redraw();
    sei();
}
```