Department of Information Technology

Digitala Projekt
Frysboxregulator

Andreas Gustavsson E05 Simon Saffer D02

10 december 2007

Abstract

The project consisted of constructing a regulator for a freezer. The regulator keeps the freezer temperature within a certain determined temperature interval. This is done by continually measuring the temperature. If it is too high, the compressor of the freezer is turned on untill the temperature has reached the desired level. The temperature wich the regulator strives to reach and maintain is the reference temperature which can be adjusted by two different buttons, one increasing and the other decreasing the reference temperature. To avoid wearing out the compressor, it is not activated untill the temperature has gone one degree above the desired temperature. Once active, the compressor continues until the temperature has dropped two degrees below the reference temperature.

The regulation is done by an AVR ATMega16 processor, which continually polls a DS1820 thermometer to recieve the current temperature. Then we calculate wheather or not the current temperature is within the acceptable interval, and appropriate action is then taken. The value for the current temperature and the reference temperature is displayed on a HD44780U display. The reference temeperature can be set at any time by pressing the buttons.

Innehåll

1 Inledning

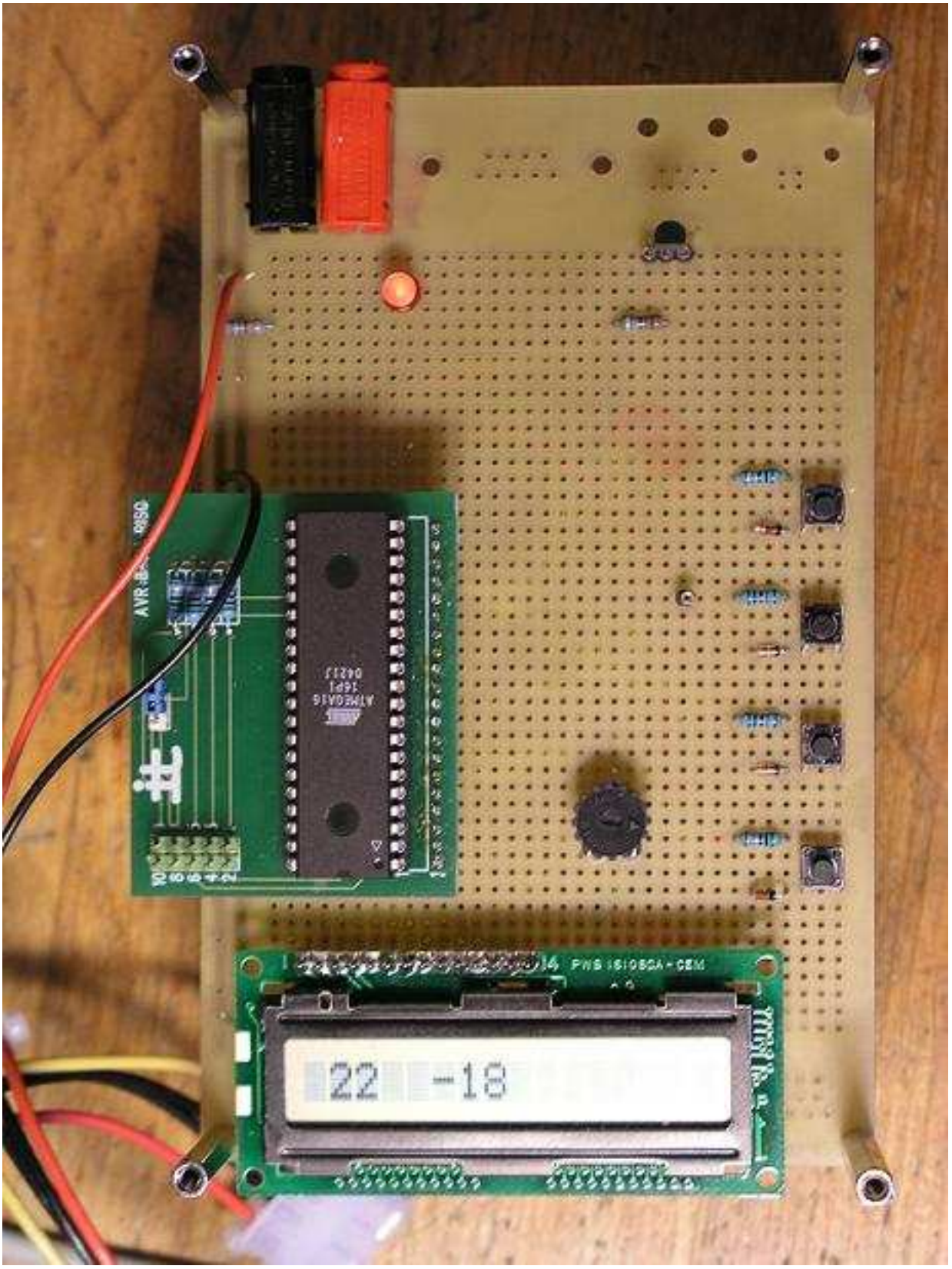
Vårt projekt gick ut på att konstruera en regulator för en frys. De kriterier som regulatorn behövde uppfylla var självklart att alltid hålla sig tillräckligt nära referenstemperaturen men att samtidigt vara så energisnål som möjligt och undvika slitage på kompressorn i den allra högsta möjliga utsträckningen.
Regleringsalgoritmen i sig är väldigt enkel eftersom kompressorn bara har ett av-läge och ett på-läge. Det enda svåra här skulle vara att balansera mellan de ovanstående kraven.
De största utmaningarna som uppgiften medförde var att kunna på rätt sätt begära och ta emot data från termometern som använder ett och samma ben för både in och utdata (1-wire bus), och att kunna skriva datan till skärmen. Sedan behövde vi också göra ett par designval som t.ex. vad som skulle hanteras med avbrott och vad som skulle göras genom polling.
I denna rapporten kommer vi inte fullständigt gå igenom processerna för att kommunicera med termometern eller skärmen, då databladen ger en utförlig beskrivning av de data-utbytena som behöver göras. För att få en mer detaljerad uppfattning om vad som händer så hänvisar vi till kommentarerna av programkoden som finns i Apendix B.

2 Konstruktion och Genomförande

Systemet byggdes upp successivt, vi började med att bygga upp systemet fysiskt på ett prototypkort, den kretsmässiga lösningen av konstruktionen är ganska enkel, den del som vi fick tänka till lite på var hur vi skulle få knapparna att fungera utan att scanna igenom dem hela tiden, det skulle medföra mycket mer jobb för processorn. Knapparna löste vi med ett externt avbrott som startar en genomsökning av knappsatsen. Inputpinnarna till knapparna är satta höga och då en knapp trycks ner så dras både avbrottspinnen och inputpinnen till jord. Eftersom alla knapparna är anslutna till samma avbrottspinne, så för att kunna särskilja knapparna används en diod mellan knappen och avbrottspinnen (se kretsschema). Denna lösning på knapparna har också ett problem som kallas debounce d.v.s. att då en knapp trycks ner så hinner processorn registrera knapptryckningen flera gånger. Därför startas en timer vi knapptryckningen som hindrar ett nytt avbrott att exekveras under en viss tid. Temperaturgivaren är en Maxim DS 1820 som har ett digitalt 1-wire interface som bygger på pulslängder. Efter en initieringssekvens kan man läsa data från givaren. När temperaturgivaren skickar data till processorn skickar den ett 16-bitars två-komplementstal där de första åtta bitarna är teckenbitar. I och med att temperaturgivaren skickar två-komplementsdata fick vi lite problem med att jämföra ett referensvärde med den aktuella temperaturen. Detta löste vi genom att konvertera det negativa talet till ett positivt tal. För att hålla reda på om det är plus eller minusgrader i frysen kontrollerar vi endast teckenbitarna från temperaturgivaren.
LCDn som vi använder i projektet är en standard alfanumerisk lcd med en rad med 16 tecken. Eftersom vi använder en Atmega16 med hela 32 I/O-pinnar så kunde vi slösa 8 pinnar till databussen till lcdn, annars kan man köra en 4-bitars buss om man inte har tillgång till så många I/O-pinnar.

3 Resultat och Utvärdering

Enligt kravspecifikationerna skulle antal dörröppningar registreras, det fanns det tyvär inte tid för men som temperaturregulator fungerar den stabilt och vi har även testat att lägga ner regulatorn i en frys för att kontrollera att regulatorn slår till då temperaturen blir en grad för varm och att den slår av då temperaturen blir två grader under referenstemperaturen. Regulatorn kan tyvär bara reglera minusgrader men det kan nog lösas i framtiden om man vill reglera en kyl tillexempel. I själva temperaturregleringen använder vi ett hysteresband på 3 grader, eftersom om vi hade reglerat temperaturen mer exakt så som om man skulle PI-reglera temperaturen, detta hade gett en väldigt exakt temperatur utan stationärt fel pga. integrationsdelen. Men då skulle kompressorn få arbeta väldigt många och korta stunder. Den lösningen kommer att ge ett stort slitage på kompressorn.
I bilden nedan är konstruktionen på hela regulatorkretsen där en trimpotentiometer används till att reglera kontrasten till displayen och en lysdiod som visar att kompressorn är tillslagen eftersom temperaturen är för hög. Av knapparna använder vi endast två stycken för att höja respektive tänka referenstemperaturen. De två andra knapparna skulle användas till att nollställa antalet dörröppningar samt total öppen tid på dörren. På displayen syns den nuvarande temperaturen till vänster och referenstemperaturen till höger. Den nästan transistorliknande komponenten är den digitala temperatursensorn.



Figur 1: Regulatorn i all sin prakt

4 Referenser

Datablad för AVR ATMega16

`http://www.it.lth.se/datablad/Processors/ATmega16_adv.pdf`

Datablad för DS1820

`http://www.it.lth.se/datablad/sensors/1820.pdf`

Datablad för HD44780U

`http://www.it.lth.se/datablad/display/hd44780.pdf`

För kommunikationen med temperaturmätaren och lcd-displayen modifierade vi Joakim Ansby's bibliotek för **DS1820** och **HD44780U**.

5 Appendix

5.1 Apendix A - Programkod

5.1.1 main.c

```
#include <avr/io.h>
#include <stdint.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include "lcd_lib.h"
#include "1wire.h"
/* define CPU frequency in Mhz here if not defined in Makefile */
#ifndef F_CPU
#define F_CPU 8000000UL
#endif
uint8_t ref;

uint16_t gettemp(){ // initierar tempmätaren.
reset();
writebyte(0xCC);
writebyte(0x44);
delay_ms(500);
reset();
writebyte(0xCC);
writebyte(0xBE);
return (readbyte()|(readbyte()<<8));

}


void printtemp(){ // läser in tmp från tempmätaren.
static uint16_t tmp;
tmp = gettemp();
if (tmp & _BV(15)){ //bit 15 i tmp måste vara 1 alla andra kvittar. dvs om tmp är negativt
lcd_jumpto(0,0); //är bit 15 = 1.
lcd_putc('-');
tmp=0xffff-tmp+1;
}
else
lcd_putc(' ');
lcd_dec2(tmp/16);
}

uint8_t tmptransform(){ // konvertera det negativa temp värdet så att vi kan jämföra det m
static uint16_t tmp; // referensvärdet
tmp = gettemp();
if(tmp & _BV(15)){
tmp=0xffff-tmp+1;
```

```
tmp = tmp/16;
tmp = 0x7F-tmp; //ändrat till +
}
else {
tmp = tmp/16;
tmp = 0x7F + tmp;
}
return tmp;
}

ISR(INT0_vect){ // Avbrottsrutin söker av knappsats
GICR &=~_BV(INT0);
if(!(PINB & _BV(PB4))){ //&-ar B-bussen med bit 4
if(ref < 30){
ref = ref+1;
eeprom_write_byte(0,ref);
}
}
else if(!(PINB & _BV(PB5))){
if(ref > 0){
ref = ref-1;
eeprom_write_byte(0,ref);
}
}
TCNT1 = 0;
TIMSK |= _BV(TOIE1);
}

ISR(TIMER1_OVF_vect){ // timer för att undvika debounce på knapparna.
if((PINB & _BV(PB4)) && (PINB & _BV(PB5))){
TIMSK &=~_BV(TOIE1);
GICR |= _BV(INT0);
}
}

void startmotor(){ // Startar kompressorn
DDRD |= _BV(PD7);
PORTD |= _BV(PD7);
}

void stopmotor(){ // Stoppar komressorn
DDRD |= _BV(PD7);
PORTD &=~_BV(PD7);
}


int main(void){
PORTD |= _BV(PD2); // sätter pinne 2 på B-bussen till utgång.
DDRB &=~ (_BV(PB4) | _BV(PB5) | _BV(PB6) | _BV(PB7)); //ingångspinnar till knappsatsen.
MCUCR |= _BV(ISC01);
```

```
GICR |= _BV(INT0); // Aktiverar externt avbrott på pinne INT0
TCCR1A = 0; // Aktiverar timerinterrupt till debounce av knappar
TCCR1B = _BV(CS11); // Aktiverar timerinterrupt till debounce av knappar
sei(); // Startar interrupts
ref = eeprom_read_byte(0);
init_lcd();
lcd_dec2(gettemp()/16);
if(ref > 30){ //Sriver iställd temperatur till eeprom-minnet
eeprom_write_byte(0,18);
ref = eeprom_read_byte(0);
}
while(1) { //evig loop
// calibratedelay();
lcd_puts("               "); //tömmer skärmen
lcd_jumpto(0,0); //sätter markören till startposition
printtemp(); //skriver ut temperaturen från temp-mätaren på skärmen
lcd_puts("  -");
lcd_dec2(ref); //skriver ut referenstemperaturen
delay_ms(60);
if(tmptransform() < (127 - ref-2)){ // om temperaturen är 2 grader under ref så stannar mo
stopmotor();
}
if(tmptransform() > (127 - ref)){ // starta motorn då temp är en grad varmare är ref
startmotor();
}
}
return 0;

}
```

5.1.2 1wire.h

```
#include <avr/pgmspace.h>
#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#include <stdlib.h>


#define WDDR DDRB
#define WPORT PORTB
#define WBIT 3
#define WPIN PINB
#define DA 1 //7
#define DB 21  //64
#define DC 20
#define DD 1
#define DE 1
#define DF 18
```

```
#define DG 1
#define DH 240
#define DI 23
#define DJ 136

inline void delayus(uint16_t us);
void write(uint8_t bit);
uint8_t read();
uint8_t reset();
void writebyte(uint8_t byte);
uint8_t readbyte();
void calibratedelay();
```

5.1.3 lcd.h

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

#include <math.h>

#define DATADDR DDRA
#define DATAPORT PORTA
#define DATAPIN PINA
#define CMDDDR DDRB
#define CMDPORT PORTB
#define EBIT 1
#define RSBIT 2
#define RWBIT 0

void init_lcd();
void lcd_busy();
void lcd_write(unsigned char command);
void lcd_putc(unsigned char command);
void lcd_puts(char* text);
void lcd_jumptosec();
void lcd_jumpto(unsigned char x,unsigned char y);
void lcd_clear();
void lcd_off();
void lcd_on();
void lcd_dec(int value);
void lcd_printtemp(uint16_t temp);
int bintodec(int* data);
int bintoword(int* data);
void lcd_writeblack();
void lcd_decdigit(int value);
void lcd_dec2(int value);
```

```
void lcd_hex(unsigned char data);
void lcd_hexint(int data);
void delay_ms(int ms);   // wait "ms" milliseconds
```

5.1.4 1wire.c

```
/*
Write 1 bit   Send a '1' bit to the 1-Wire slaves (Write 1 time slot)
Drive bus low, delay A
Release bus, delay B

Write 0 bit  send a '0' bit to the 1-Wire slaves (Write 0 time slot)
Drive bus low, delay C
Release bus, delay D

Read bit  Read a bit from the 1-Wire slaves (Read time slot)
Drive bus low, delay A
Release bus, delay E
Sample bus to read bit from slave
Delay F

Reset  Reset the 1-Wire bus slave devices and ready them for a command
Delay G
Drive bus low, delay H
Release bus, delay I
Sample bus, 0 = device(s) present, 1 = no device present
Delay J
*/

#include <avr/pgmspace.h>
#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include "1wire.h"
/*howto read temperature from ds18s20
uint16_t gettemp(){
reset();
writebyte(0xCC);
writebyte(0x44);
delay_ms(3);
reset();
writebyte(0xCC);
writebyte(0xBE);
return readbyte();

}
```

```
*/
void delayus(unsigned int);
void write(uint8_t bit);
void writebyte(uint8_t byte);
uint8_t read();
uint8_t readbyte();
uint8_t reset();

inline void delayus(uint16_t us){
for(uint16_t i = 0; i < us; i++){
// for(uint8_t k = 0; k < 1; k++){
// asm  volatile ("nop");

// }
}
}


void write(uint8_t bit){
WDDR |= _BV(WBIT);
WPORT &=  ~_BV(WBIT);
if(bit)
delayus(DA);
else if(!bit)
delayus(DC);
WDDR &= ~_BV(WBIT);
if(bit)
delayus(DB);
else if(!bit)
delayus(DE);
}

void writebyte(uint8_t byte){

cli();
for(uint8_t i = 0; i < 8; i++){
write(byte & 0x1);
byte >>=1;
}
sei();

}
uint8_t readbyte(){
cli();
 uint8_t data = 0;
for(uint8_t i = 0; i < 8; i++){
data |= read() << i;
}
sei();
return data;
```

```
}

uint8_t read(){
static uint8_t bit = 0;
WDDR |= _BV(WBIT);
WPORT &=  ~_BV(WBIT);
delayus(DA);
WDDR &= ~_BV(WBIT);
delayus(DE);
if(WPIN & _BV(WBIT))
bit = 1;
else if(!(WPIN & _BV(WBIT)))
bit = 0;

delayus(DF);

return bit;
}

uint8_t reset(){
static uint8_t bit = 0;
delayus(DG);
WDDR |= _BV(WBIT);
WPORT &=  ~_BV(WBIT);
delayus(DH);
WDDR &= ~_BV(WBIT);
delayus(DI);
if(WPIN & _BV(WBIT))
bit = 1;
else if(!(WPIN & _BV(WBIT)))
bit = 0;
delayus(DJ);
return bit;
}

void calibratedelay(){
WDDR |= _BV(WBIT);
for(uint8_t i = 0; i < 100; i++){
WPORT |= _BV(WBIT);
delayus(DG);
WPORT &= ~_BV(WBIT);
delayus(DG);
}
}
```

5.1.5 lcd.c

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

```
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <avr/pgmspace.h>
#include "lcd_lib.h"
#include <math.h>

void init_lcd(){
//if called right after power-up, we'll have to wait a bit (fine-tune for faster execution
delay_ms(1000); //tell the LCD that it's used in 8-bit mode 3 times, each with a delay inb
lcd_write(0x30);
delay_ms(10);
lcd_write(0x30); //now: 8 bit interface, 5*7 font, 1 lines.
delay_ms(10);
lcd_write(12); //display on, cursor on (blinking)
delay_ms(10);
lcd_write(0x06); //cursor auto-inc
delay_ms(10);
lcd_write(0x10); //cursor auto-inc
delay_ms(10);
lcd_write(0x01); //now clear the display, cursor home
delay_ms(10);
}


void lcd_busy(){
//while(lcd_getAdr() == 0x80);
delay_ms(40);
}

void lcd_write(unsigned char command){
lcd_busy(); //kollar klar
CMDDDR = _BV(RSBIT)|_BV(EBIT)|_BV(RWBIT); // Sets RS,R/W and E pins to output
CMDPORT &= ~(_BV(RSBIT)|_BV(RWBIT)|_BV(EBIT));  // Sets RS and RW low = Instruction input
DATADDR = 0xff;                         // Sets databus to output
DATAPORT = command;                       //Put data on bus
CMDPORT |= _BV(EBIT);     //Pulse E
delay_ms(1);                          //wait a while
CMDPORT &= ~_BV(EBIT);              //Set e low again

}
void lcd_putc(unsigned char command){
lcd_busy();
lcd_busy(); //kollar klar
CMDDDR = _BV(RSBIT)|_BV(EBIT)|_BV(RWBIT); // Sets RS,R/W and E pins to output
CMDPORT &= ~(_BV(RWBIT)|_BV(EBIT));  // Sets RW low = Write
CMDPORT |= _BV(RSBIT);              // Set RS high = Data input mode
DATADDR = 0xff;                         // Sets databus to output
DATAPORT = command;                       //Put data on bus
CMDPORT |= _BV(EBIT);     //Pulse E
delay_ms(1);                          //wait a while
```

```
CMDPORT &= ~_BV(EBIT);              //Set e low again
}

void lcd_puts(char* text){
while(*text){
lcd_putc(*text++);
}
}


void lcd_jumptosec(){
lcd_write(0x0C0);

}

void lcd_jumpto(unsigned char x,unsigned char y){
unsigned char adress=0;
if(y == 0){
adress = 128+x;
}
else if( y == 1){
adress = 0x40+128+x;
}
else if( y == 2){
adress = 0x14+128+x;
}
else if( y == 3){
adress = 0x54+128+x;
}
lcd_write(adress);
}

void lcd_clear(){

lcd_write(0x01); //now clear the display, cursor home
}

void lcd_off(){
lcd_write(8);
}
void lcd_on(){
lcd_write(12);
}

void lcd_dec(int value){
//value = value*10;
//value = value/2;
// 1  2  3  4
// f1 f2 f2 f4
int f[4];
```

```
f[0]= value/1000;
f[1] = (value-(f[0]*1000))/100;
f[2] = (value - (f[1]*100)-(f[0]*1000))/10;
f[3] = value - (f[1]*100)-(f[0]*1000)-(f[2]*10);

for (int i=0; i<=3; i++){
if(i == 3){
//lcd_putc(',');
}
if(f[i] == 0){
lcd_putc('0');
}
if(f[i] == 1){
lcd_putc('1');
}
if(f[i] == 2){
lcd_putc('2');
}
if(f[i] == 3){
lcd_putc('3');
}
if(f[i] == 4){
lcd_putc('4');
}
if(f[i] == 5){
lcd_putc('5');
}
if(f[i] == 6){
lcd_putc('6');
}
if(f[i] == 7){
lcd_putc('7');
}
if(f[i] == 8){
lcd_putc('8');
}
if(f[i] == 9){
lcd_putc('9');
}

}

}

void lcd_printtemp(uint16_t temp){
//value = value*10;
//value = value/2;
// 1  2  3  4
// f1 f2 f2 f4
```

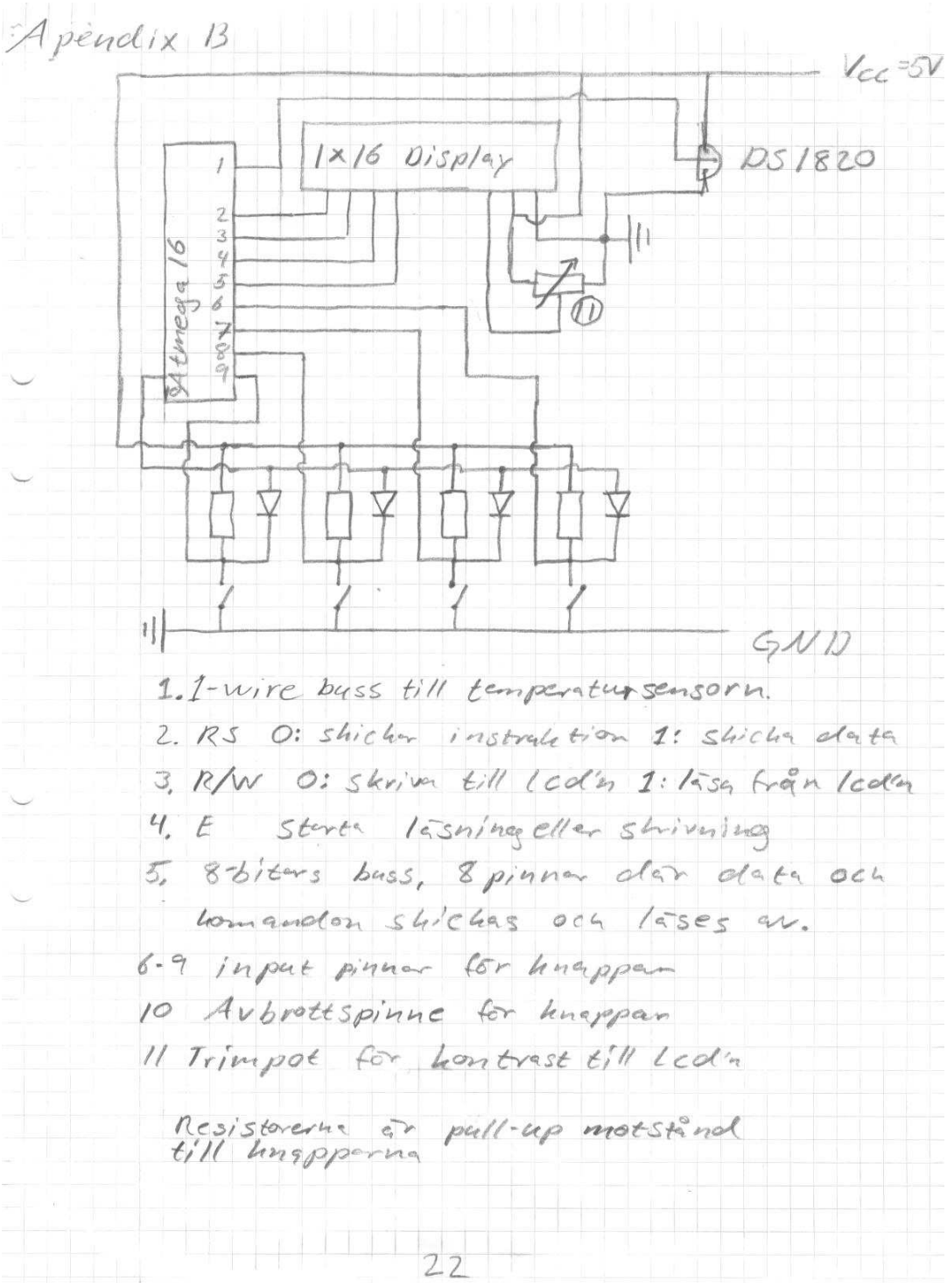
```
int f[3];
f[0] = (temp/100);
f[1] = (temp - (f[0]*100))/10;
f[2] = temp - (f[0]*100)-(f[1]*10);

for (int i=0; i<=2; i++){
if(i == 2){
lcd_putc(',');
}
if(f[i] == 0){
lcd_putc('0');
}
if(f[i] == 1){
lcd_putc('1');
}
if(f[i] == 2){
lcd_putc('2');
}
if(f[i] == 3){
lcd_putc('3');
}
if(f[i] == 4){
lcd_putc('4');
}
if(f[i] == 5){
lcd_putc('5');
}
if(f[i] == 6){
lcd_putc('6');
}
if(f[i] == 7){
lcd_putc('7');
}
if(f[i] == 8){
lcd_putc('8');
}
if(f[i] == 9){
lcd_putc('9');
}

}

}
int bintodec(int* data){
int temp=0;
for(int i = 0; i<9; i++){
temp |= (data[i] << i);
}
return temp;
}
```

```
  int bintoword(int* data){
int temp=0;
for(int i = 0; i<16; i++){
temp |= (data[i] << i);
}
return temp;
}



void lcd_writeblack(){
lcd_write(64);
for(int i = 0; i < 8; i++){
lcd_putc(255);
}
lcd_jumpto(0,0);
}

void lcd_decdigit(int value){
//value = value*10;
//value = value/2;
// 1  2  3  4
// f1 f2 f2 f4
unsigned char f[4];
f[0]= value/10;
f[1] = (value-(f[0]*10));


for (int i=0; i<=1; i++){
if(i == 3){
//lcd_putc(',');
}
if(f[i] == 0){
lcd_putc('0');
}
if(f[i] == 1){
lcd_putc('1');
}
if(f[i] == 2){
lcd_putc('2');
}
if(f[i] == 3){
lcd_putc('3');
}
if(f[i] == 4){
lcd_putc('4');
}
if(f[i] == 5){
```

```
lcd_putc('5');
}
if(f[i] == 6){
lcd_putc('6');
}
if(f[i] == 7){
lcd_putc('7');
}
if(f[i] == 8){
lcd_putc('8');
}
if(f[i] == 9){
lcd_putc('9');
}

}
}


void lcd_dec2(int value){
//value = value*10;
//value = value/2;
// 1  2  3  4
// f1 f2 f2 f4
unsigned char f[4];
f[0]= value/10;
f[1] = (value-(f[0]*10));


for (int i=0; i<=1; i++){
if(i == 3){
//lcd_putc(',');
}
if(f[i] == 0 && i != 0){
lcd_putc('0');
}
else if(f[i]==0 && i == 0)
lcd_putc(' ');
if(f[i] == 1){
lcd_putc('1');
}
if(f[i] == 2){
lcd_putc('2');
}
if(f[i] == 3){
lcd_putc('3');
}
if(f[i] == 4){
lcd_putc('4');
}
```

```
if(f[i] == 5){
lcd_putc('5');
}
if(f[i] == 6){
lcd_putc('6');
}
if(f[i] == 7){
lcd_putc('7');
}
if(f[i] == 8){
lcd_putc('8');
}
if(f[i] == 9){
lcd_putc('9');
}

}
}

void lcd_hex(unsigned char data){
unsigned char aHex[] = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
lcd_putc(aHex[((data>>4)&0x0F)]);
lcd_putc(aHex[(data&0x0F)]);

}
void lcd_hexint(int data){
unsigned char aHex[] = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
lcd_putc(aHex[((data>>16)&0x0F)]);
lcd_putc(aHex[((data>>8)&0x0F)]);
lcd_putc(aHex[((data>>4)&0x0F)]);
lcd_putc(aHex[(data&0x0F)]);
}
void delay_ms(int ms) {   // wait "ms" milliseconds
    for(int i = 0; i < ms; i++) {
        for(int x = 0; x < 10; x++) {

        asm volatile("nop");
    asm volatile("nop");
    asm volatile("nop");
    asm volatile("nop");
    asm volatile("nop");
    asm volatile("nop");
        }
    }
}
```

5.2 Apendix B - Kretsschema



Figur 2: Kretsschema