Fan Controller
Digital projects (EITF01) Lund University, Faculty of Engineering

John Hedestig, Karl Fogelström
Handledare: Bertil Lindvall

Abstract

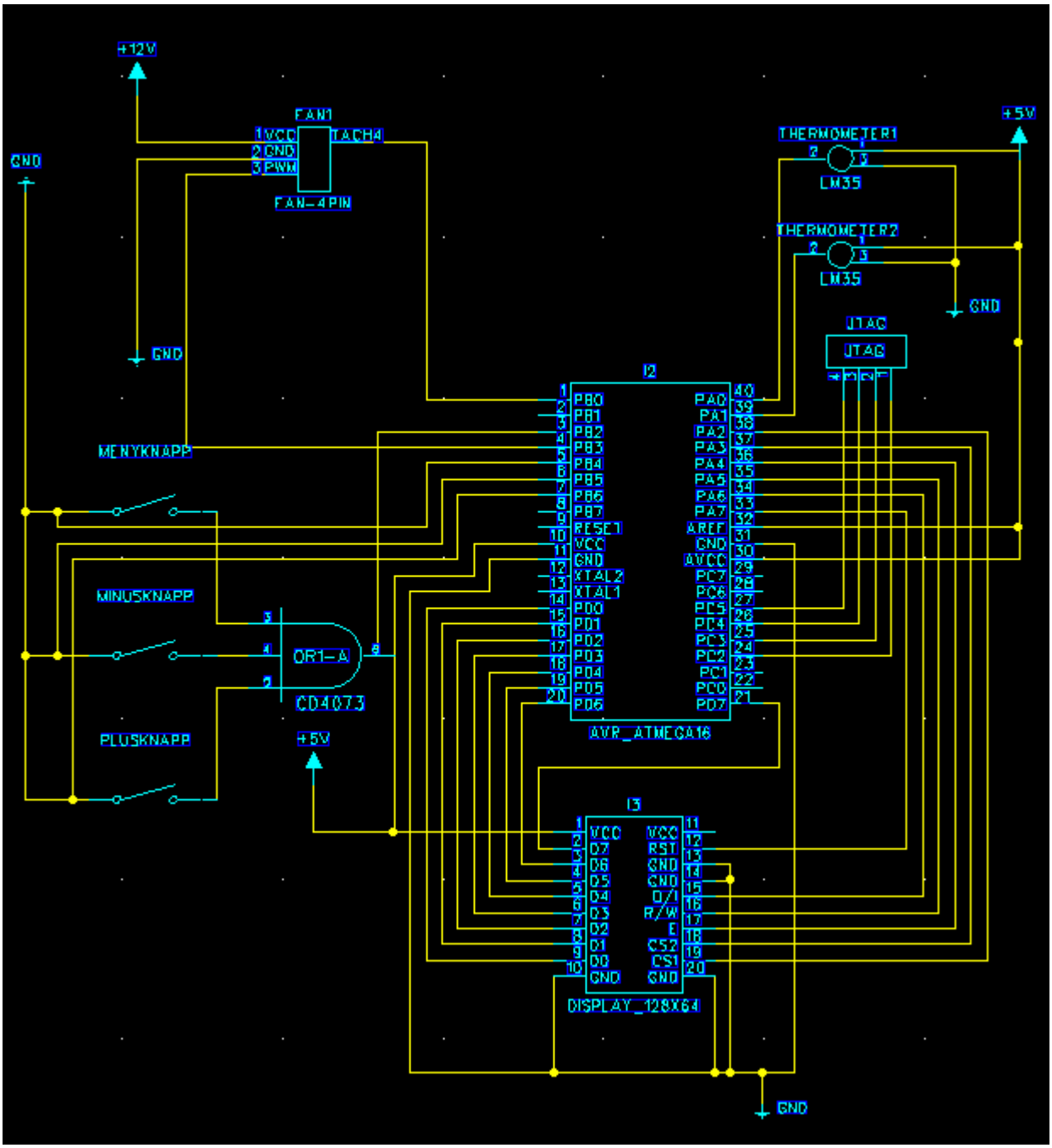
The aim of this digital project was to construct a controller for a 4-pin processor fan that controlled the speed of the fan depending on a chosen target temperature and a measured temperature. From the beginning the ambition of the project had a larger scope but had to be downscaled due to lack of time and choice of processor.

Contents

Inledning

I kursen Digitala projekt för industriell ekonomi fick vi i uppgift att bygga en apparat innehållande bl.a. en micro-controller (ATMega16) och andra elektroniska komponenter, beroende på vilken typ av projekt man valde att göra. Vårt val föll på ett system för att kontrollera datorfläktar. Ursprungligen var målen ganska högt satta, med ambitionen att kontrollera två fläktar med avseende på antingen måltemperatur eller ett varvtal. Detta visade sig bli både oss och vår processor en övermäktig uppgift så projektet skalades ner till att styra en fläkt med avseende på en måltemperatur. Båda termometrarna behölls dock för att kunna ha en referenstemperatur på den omgivande luften i rummet.

Kopplingsschema



Kravspecifikation

I projektets absolut tidigaste skede formulerades en ursprunglig kravspecifikation med många ambitiösa krav och ännu fler eventuella krav i mån av tid. Inför denna kurs hade vi nästintill obefintliga kunskaper i elektronik och detta visade sig vara lite övermäktiga krav, så efter lite testande och en del överläggande med vår handledare nåddes så småningom en slutgiltig kravspecifikation.

Ursprunglig kravspecifikation

Funktionella krav

- Display:
    - (Bakgrundsbelyst)
    - I standby ska den visa temperatur och varvtal på alla fläktar
        - Eventuellt alla samtidigt, eller
        - Visa fläkt 1 i 3 sek sedan fläkt 2 sek etc etc.
- Applikation ska rymmas i 5,25-tums mått
- Knappsats:
    - 3 knappar
    - Knapp 1:
        - Byta mellan varvtal/temperatur/(fläkt)
        - Hålla in 3 sekunder: Byta läge måltemperatur/larmtemperatur
    - Knapp 2: -1 grader Celsius
    - Knapp 3: +1 grader Celsius
    - Knapp 1-2-3: Stänga av larmfunktion
- Kontrollera minst 2 fläktar
- Strömförsörjning via vanligt datornätaggregat +5V
- Larmfunktion:
    - (ljud?)
    - Blinkande display
    - (Meddelande?)
    - Minnas larmen?
    - Larmar när:
        - Aktuell temp överskrider måltemp för ofta (80%)
        - Överskrider akuta temperaturpunkten (Stänga av datorn?)
- (Historisk):
    - Grafisk
    - Min-Max
    - Tidpunkt
- För stationära datorer
- Start/Stop med datorn

Kvalitativa krav

- Temperaturuppdatering display maximalt var 5:e sekund
- Display:
    - Lättförståelig, behöver ingen instruktion
- Knappsats:
    - Lättförståelig, behöver ingen instruktion
- Maximal Temperaturmätavvikelse +/- 0.5
- Maximal reglertemperaturavvikelse (över målvärdet):
    - Fläktar "utan" fördröjning: 20 sekunder
    - Fläktar med fördröjning (ex: chassifläkt m.m.): 1 min
    - Förutsatt tillfredsställande installation och rumstemperatur.
- Varvtalet på fläkten skall vara tillfredsställande jämt för att inte orsaka onödigt störande ljud
- Installerbar på alla sorters stationära datorer med ledig 5,25 plats och ledig +5V strömförsörjning.

- Typkrav på fläktar:
    - 4-pinnars PWM
- Typkrav på temperaturmätare:
    - 3-pinnars Analog

Slutgiltig kravspecifikation

Funktionella krav

- Display:
    - I standby ska den visa:
        - Rumstemperatur
        - Temperatur vid fläkt
        - Måltemperatur
        - Fläktens procentuella effekt
- Knappsats:
    - 3 knappar
    - Knapp 1: Sätt måltemperatur till aktuell fläkttemperatur
    - Knapp 2: Måltemperatur -1 grad Celsius
    - Knapp 3: Måltemperatur +1 grad Celsius
- Kontrollera 1 fläkt
- Strömförsörjning 5V och 12V
- Grafisk: Visa graf över vad rums- och fläkttemperaturen har varit den senaste tiden

Kvalitativa krav

- Temperaturuppdatering display maximalt var 5:e sekund
- Display:
    - Lättförståelig, behöver ingen instruktion
- Knappsats:
    - Lättförståelig, behöver ingen instruktion
- Maximal Temperaturmätavvikelse +/- 0.5
- Maximal reglertemperaturavvikelse (över målvärdet): Nå rimliga målvärden inom en halv minut
- Varvtalet på fläkten skall vara tillfredsställande jämt för att inte orsaka onödigt störande ljud
- Typkrav på fläktar:
    - 4-pinnars PWM
- Typkrav på temperaturmätare:
    - 3-pinnars Analog

Konstruktion

På ett kretskort monterades en ATMega16 och tre knappar. Dessa kopplades till processorn och dit kopplades även en display, en fläkt och två termometrar, en som hängde fritt i luften och en som satt på fläkten.

Hårdvara

Här följer en kort beskrivning av den hårdvara som användes.

Processor
Som processor användes en ATMega16, en 40-pinnars micro-controller med klockfrekvens på 16MHz, 1kB RAM och 16kB internt flashminne.

Display
Displayen är en LCD-skärm med 128x64 pixlar.

Fläkt
Fläkten som användes var en 4-pinnars Intel pentium-4- fläkt.

Termometrar
Termometrarna var av typen LM35, vilka har 3 pinnar och ger en analog signal där varje 10mV är en grad Celsius.

Knappar
De tre knapparna är programmerade att generera avbrott vid nedtryckning.

Mjukvara
Mjukvaran består av en display rutin, en grafritare, och en PI-regulator inklusive respektive rutin för att hämta externa variabler (temp etc.).

För att fastställa temperaturen användes A/D-omvandlare på port ADC0 och ADC1. Den ena temperaturen motsvarade rumstemperaturen och den andra fästes på fläkten för att kunna regleras via PI-regulatorn. Vi körde ett antal testrundor för att fastställa vad motsvarande rumstemperatur motsvarade för ADC-värde och kompenserade detta med hjälp av OFFSET_ERROR variabeln. I och med att värdet varierade kraftigt så var vi tvungna att ta ett medelvärde vid varje temperaturmätning för att inte störa PI-regulatorn vilket kunde orsaka onödiga varvtalsändringar/oljud.

För att styra varvtalet på fläkten användes Counter0 som en PWM-generator. När fläkten får en logisk 1:a på PWM-pinnen så bryts strömmen och logisk 0:a så öppnas den. Med denna kunskap skapade vi en PWM-generator med klockan CLK och Counter0. Att sätta värdet OCR0 till hälften av MAX (0xFF) skapade en PWM signal som bröt strömmen 50% av tiden och vi körde därmed på halva effekten. Sattes OCR0 till MAX-1 (0xFE) så kördes fläkten i >99% hastighet o.s.v.

PI-regulatorn använde värdet från A/D-omvandlaren för att styra PWM-generatorn (OCR0) och därmed få en reglering mellan temperatur och varvtal. För att fastställa konstanterna i PI-regulatorn användes Ziegler-Nichols metoden vilket beskrivs närmare under Metod. Vi använde oss av summor för att estimera integraldelarna i PI-regulatorn.

$$U(k) = Kp * E(k) + Ki * (Eint + Ek), \qquad Eint = SUM(E(l),0,k-1)$$

Knapparna användes för att göra en målvärdesändring på target-temperaturen på fläkten. I och med en diskrepans från första kravspecifikation och den slutgiltiga så fick vi över en knapp (meny-knappen) vilket lämnades utan någon specifik funktion i slutgiltiga versionen.

Counter2 användes för att skapa en realtidsklocka. Interna klockan CLK delades med en prescaler och OCR2 sattes till 131 för att skapa ett compare match interrupt varje ms.

$8mhz: 131/(8 * 1024 * 1024/64) = 0.000999451s \approx 1ms$

GraphDrawer ritade upp en serie av värden för temperaturen. Uppdateringsintervallet för grafen var 1gång i sekunden vilket resulterade i en 62 sekunders historik. Intervallet för rumstemperaturen var [0,48] och fläkten [0,120]. I och med rumsbrist på skärmen kunde vi inte skriva in en numerisk skala på displayen.

Metod och Resultat

För att avgöra konstanterna på den proportionella delen och integraldelen av reglerfunktionen utfördes ett Ziegler-Nichols-test. Först hettades fläkttermometern upp med hjälp av en lödpenna och måltemperaturen sattes till 55 grader. Sedan höjdes Kp-konstanten tills fläktregleringen började svänga. Svängningen skedde med ~4 sekunders intervall och Kp var 8. Detta gav upphov till konstanter som båda var 4.

Table 1. Ziegler-Nichols, Source: Wikipedia

Ziegler–Nichols method			
Control Type	K_p	K_i	K_d
P	K_u / 2	-	-
PI	K_u / 2.2	T_u / 1.2	-
PID	K_u / 1.7	T_u / 2	T_u / 8

För att avgöra regulatorns effektivitet värmdes fläkttermometern upp till 100 grader Celsius med hjälp av en lödpenna. När fläkten och regulatorn sedan sattes igång körde den fläkten på maximal effekt tills den nådde sitt målvärde, vartefter den efter några få sekunder hade hittat en konstant fläkthastighet för att hålla den temperaturen. Den lägsta temperatur som kunde åstadkommas var dock 45 grader, då den var 100 utan fläkt, så om måltemperaturen valdes lägre än detta körde fläkten på maxeffekt oavbrutet utan att nå sin måltemperatur.

Slutsats

Kursen och projektet har lärt oss oerhört mycket om både elektronik och om hur man bygger en prototyp vid utveckling av en elektronisk produkt.

Vi lyckades inte riktigt implementera all funktioner vi hade hoppats men vi är ändå mycket nöjda med den slutgiltiga produkten. Vi känner även att vi nu behärskar elektroniken, programmeringen och komponenterna i sådan utsträckning att vi skulle kunna genomföra liknande projekt på egen hand i framtiden.

Referenser

- 8-bit Microcontroller with 16K Bytes In-System Programmable Flash
  www.atmel.com/atmel/acrobat/doc2466.pdf
- http://en.wikipedia.org/wiki/Ziegler%E2%80%93Nichols_method
- Brief Graphic LCD Font; Pascal Stang

Appendix I - Källkod

Buttons.c

```
#include <avr/interrupt.h>
#include "Buttons.h"
#include "Global.h"

/*
 * Initialize routine for buttons
 */
void init_buttons (void)
{
	/* Enable pull-up resistors */
	BUTTON_PORT |= (BUTTON_INT2|BUTTON_MENU|BUTTON_MINUS|BUTTON_PLUS);

	/* Set data direction input */
	BUTTON_DDR &= ~(BUTTON_INT2|BUTTON_MENU|BUTTON_MINUS|BUTTON_PLUS);

	/* Enable interrupt INT2 */
	GICR = (1<<INT2);

	/* Enable INT2 interrupt on rising edge */
	MCUCSR = (1<<ISC2);
}

/* Interrupt routine for pressed button */
ISR(INT2_vect)
{
	if ((BUTTON_PIN & BUTTON_MENU) ^ BUTTON_MENU)
	{
		menu_button();
	}
	else if ((BUTTON_PIN & BUTTON_MINUS) ^ BUTTON_MINUS)
	{
		minus_button();
	}
	else if ((BUTTON_PIN & BUTTON_PLUS) ^ BUTTON_PLUS)
	{
```

```
                plus_button();
        }

}

/*
 * Menu button routine
 */
void menu_button ()
{

}

/*
 * Minus button routine
 */
void minus_button ()
{
        reg_set_R(reg_R-1);
}

/*
 * Plus button routine
 */
void plus_button ()
{
        reg_set_R(reg_R+1);
}
```

Counter0.c

```
/*
 * Counter/Timer0 is used as a PWM generator for the FAN.
 */
#include <avr/io.h>
#include <stdint.h>
#include "Counter0.h"

/*
 * Initialize Timer/Counter0 PWM routine
 */
void init_PWM0 ()
{
        /* Set data direction T0 input T1 output, disable pullup */
        COUNTER0_DDR |= COUNTER0_OC0;
        COUNTER0_DDR &= ~(COUNTER0_T0);
        COUNTER0_PORT &= ~(COUNTER0_T0);

        /* Set Fast PWM-mode, Clock on Clk/1024 */
```

```
	TCCR0 |= (1<<WGM00)|(1<<COM01)|(1<<WGM01)|(1<<CS02)|(1<<CS00);

	/* Set capacity to max */
	OCR0 = 0xFF-1; // 254 max vid 8 bitar
	TIFR |= (1<<TOV0);

}
```

Counter2.c

```
/*
 * Counter/Timer2 is used a realtime clock for the system.
 */

#include <avr/interrupt.h>
#include <stdint.h>
#include "Counter2.h"
#include "FanController.h"
#include "Global.h"

/*
 * Initialize routine for the real time clock
 */
void init_realTimeClock(void)
{
	/* Set Clear timer on compare, prescaling clk/64 */
	TCCR2 = (1<<WGM21)|(1<<CS22);

	/*
	 * Set TOP to 131 to achieve Compare match interrupt every ms
	 * 8mhz: 131/(8*1024*1024/64)=0.000999451s
	 */
	OCR2 = 131;

	/* Enable compare match interrupt */
	TIMSK = (1<<OCIE2);

	/* Reset realtime clock */
	time = 0;
}

/* Increases Time by one ms every interrupt */
ISR (TIMER2_COMP_vect)
{
	time++;

	/* Update main loop every 500ms */
	if (!(time%500)) {
		upd_main = 1;
```

```
		}
		/* Update graph */
		if (!(time%1000)) {
			upd_graph = 1;
		}
}
```

Fancontroller.c

```
#include <avr/io.h>
#include <stdint.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include "Global.h"
#include "FanController.h"
#include "Buttons.h"
#include "Counter0.h"
#include "Counter2.h"
#include "GraphDrawer.h"
#include "Lcd.h"
#include "Thermometer.h"


int main (void)
{
		/* Run initializing routines for internal hardware components */
		init_realTimeClock();
		init_buttons();
		init_PWM0();
		init_display();
		init_ADConverter();
		init_FanController();

		/* Global interrupt enable */
		sei();

		/* Start the hardware display */
		init_harddisplay();
		clear_display(0xFF);

		/* Initialize GraphDrawer */
		init_GraphDrawer();

		/* Sleep until interrupt occurs */
		while (1) {
			if (upd_graph) {
				update_graph();
				upd_graph = 0;
			}
```

```
                if (upd_main) {
                        update_main();
                        upd_main = 0;
                }
        }
        return 0;
}
/*
 * Main update loop controlled by real time clock:
 * - Update PI-regulator
 * - Update Display
 */
void update_main() {

        /* Update PI-regulator */
        reg_update();

        /*Clear left screen */
        print_line5x7("                ",0);
        print_line5x7("                ",1);
        print_line5x7("                ",2);
        print_line5x7("                ",3);
        print_line5x7("                ",4);
        print_line5x7("                ",5);
        print_line5x7("                ",6);
        print_line5x7("                ",7);

        /* Update display */
        char str[11];

        print_line5x7("ROOM ^C:      TEMP ROOM",0);
        print_line5x7(uint_2_str(0xFFFFFFFF & read_ADC(TEMP2_CHIP), str ,8),1);

        print_line5x7("FAN ^C:",2);
        print_line5x7(uint_2_str(0xFFFFFFFF & read_ADC(TEMP1_CHIP), str ,8),3);

        print_line5x7("TARGET ^C: TEMP FAN",4);
        print_line5x7(uint_2_str(0xFFFFFFFF & (reg_R),str,8),5);

        print_line5x7("FAN PWR %:",6);
        print_line5x7(uint_2_str(0xFFFFFFFF & (OCR0*100/254),str,8),7);
}
/*
 * If error occurs BADISR_vect catches it
 */
ISR(BADISR_vect)
{
        clear_display(3);
```

```
        print_line5x7("BADISR_vect", 7);
}
/*
  * Initialize routine for the PI-regulator
  */
void init_FanController ()
{
        reg_Kp = 4;
        reg_Ki = 4;
        reg_Eint = 0;
        reg_R_lowerlimit = 0x0F;
        reg_E = 0;
        reg_R = read_ADC(TEMP2_CHIP); // Fetch room temperature
        reg_Y = 0;
        reg_enable = 0x03; // Enable PI

}
/*
  * Defines a new R value for the regulator
  */
void reg_set_R (uint8_t value)
{
        if (reg_R_lowerlimit-1 < value && value < 0xFF) {
                reg_R = value;
                reg_Eint = 0; // To prevent windup
        }
}
/*
  * Updates the PI-regulator
  */
void    reg_update ()
{
        int16_t reg_U = 0;

        /* Collect new input data for the PI-regulator */
        reg_Y = read_ADC(TEMP1_CHIP);
        reg_E = (int16_t)(reg_Y - reg_R);

        /* Calculate P-regulator part */
        if (reg_enable & 0x01) {
                reg_U += (int16_t)reg_Kp*reg_E;
        }

        /* Calcuate    I-regualtor part */
        if (reg_enable & 0x02) {
                reg_U += (int16_t)reg_Ki*(reg_Eint+reg_E);
                reg_Eint += reg_E;
        }
```

```
	if (reg_U < 0) {
		reg_U = (int16_t)reg_R_lowerlimit;
	}
	if (reg_U >= 0xFF) {
		reg_U = (int16_t)0xFF-1;
	}
	OCR0 = reg_U;
}
```

GraphDrawer.c

```
#include <stdint.h>
#include "GraphDrawer.h"
#include "Global.h"
#include "Lcd.h"
#include "Thermometer.h"

/*
 * Initialize routine for the GraphDrawer
 */
void init_GraphDrawer () {

	/* Room temperature Graph */
	display_page (1);
	display_column(0,DISPLAY_CS2);
	write_instr (0xFF, DISPLAY_CS2);
	write_instr (0x01, DISPLAY_CS2);

	display_page (2);
	display_column(0,DISPLAY_CS2);
	write_instr (0xFF, DISPLAY_CS2);
	write_instr (0x01, DISPLAY_CS2);

	display_page (3);
	display_column(0,DISPLAY_CS2);
	write_instr (0xFF, DISPLAY_CS2);
	write_instr (0x81, DISPLAY_CS2);
	write_instr (0x80, DISPLAY_CS2);
	write_instr (0x80, DISPLAY_CS2);
	write_instr (0x80, DISPLAY_CS2);
	write_instr (0x80, DISPLAY_CS2);
	write_instr (0x80, DISPLAY_CS2);
	write_instr (0x80, DISPLAY_CS2);

	write_instr (0x80, DISPLAY_CS2);
	write_instr (0x80, DISPLAY_CS2);
	write_instr (0x80, DISPLAY_CS2);
	write_instr (0x80, DISPLAY_CS2);
	write_instr (0x80, DISPLAY_CS2);
```

```
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);

write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);

write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);

write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);

write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);

write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
```

```
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);

/* Fan temperature Graph */
display_page (5);
display_column(0,DISPLAY_CS2);
write_instr (0xFF, DISPLAY_CS2);
write_instr (0x01, DISPLAY_CS2);

display_page (6);
display_column(0,DISPLAY_CS2);
write_instr (0xFF, DISPLAY_CS2);
write_instr (0x01, DISPLAY_CS2);

display_page (7);
display_column(0,DISPLAY_CS2);
write_instr (0xFF, DISPLAY_CS2);
write_instr (0x81, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);

write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);

write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
write_instr (0x80, DISPLAY_CS2);
```

```
		write_instr (0x80, DISPLAY_CS2);

		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);

		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);

		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);

		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);

		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
		write_instr (0x80, DISPLAY_CS2);
}
/*
```

```
 * Update the graph
 * Each value are shifted left once
 */
void update_graph() {

	int i;

	uint16_t fan_temp = read_ADC(TEMP1_CHIP);
	uint16_t room_temp = read_ADC(TEMP2_CHIP);

	for (i=0; i<61; i++) {
		fan_graph[i] = fan_graph[i+1];
		room_graph[i] = room_graph[i+1];
	}
	if (fan_temp > 120) {
		fan_temp = 120;
	}
	if (room_temp > 48) {
		fan_temp = 48;
	}
	fan_graph[61] =    (fan_temp/5);
	room_graph[61] = (room_temp/2);

	/* Room Graph */
	display_column(2,DISPLAY_CS2);
	display_page (1);
	for (i=0; i<62; i++) {
		if (room_graph[i] >16) {
			write_instr (0x80>>(room_graph[i]-17), DISPLAY_CS2);
		} else {
			write_instr (0x00, DISPLAY_CS2);
		}
	}
	display_column(2,DISPLAY_CS2);
	display_page (2);
	for (i=0; i<62; i++) {
		if (room_graph[i] > 8) {
			write_instr (0x80>>(room_graph[i]-9), DISPLAY_CS2);
		} else {
			write_instr (0x00, DISPLAY_CS2);
		}
	}
	display_column(2,DISPLAY_CS2);
	display_page (3);
	for (i=0; i<62; i++) {
		if (room_graph[i] >1) {
			write_instr (0x80>>(room_graph[i]-1)|0x80, DISPLAY_CS2);
		} else {
```

```
                write_instr (0x80, DISPLAY_CS2);
            }
        }

        /* Fan Graph */
        display_column(2,DISPLAY_CS2);
        display_page (5);
        for (i=0; i<62; i++) {
            if (fan_graph[i] >16) {
                write_instr (0x80>>(fan_graph[i]-17), DISPLAY_CS2);
            } else {
                write_instr (0x00, DISPLAY_CS2);
            }
        }
        display_column(2,DISPLAY_CS2);
        display_page (6);
        for (i=0; i<62; i++) {
            if (fan_graph[i] > 8) {
                write_instr (0x80>>(fan_graph[i]-9), DISPLAY_CS2);
            } else {
                write_instr (0x00, DISPLAY_CS2);
            }
        }
        display_column(2,DISPLAY_CS2);
        display_page (7);
        for (i=0; i<62; i++) {
            if (fan_graph[i] >1) {
                write_instr (0x80>>(fan_graph[i]-1)|0x80, DISPLAY_CS2);
            } else {
                write_instr (0x80, DISPLAY_CS2);
            }
        }
}
```

Lcd.c

```
#include <stdint.h>
#include "Lcd.h"
#include <avr/io.h>
#include "Font5x7.h"
#include <avr/interrupt.h>
#include "Global.h"

/*
 * Initialize routine for display
 */
void init_display (void)
{
        /* Set data direction on output pins */
```

```
	DISPLAY_INSTR_DDR |=
DISPLAY_RS|DISPLAY_RW|DISPLAY_E|DISPLAY_CS2|DISPLAY_CS1|DISPLAY_RESET;
	DISPLAY_DATA_DDR = 0xFF;

	/* Reset display RST=1 */

	DISPLAY_INSTR_PORT &= ~DISPLAY_RESET;
	DISPLAY_INSTR_PORT |= DISPLAY_RESET;
}
/*
 * Start routine for display
 */
void init_harddisplay (void)
{
	/* Turn on display */
	display_power(1);

	/* Set Z to Max */
	write_cmd(DISPLAY_INSTR_Z,DISPLAY_CS1);
	write_cmd(DISPLAY_INSTR_Z,DISPLAY_CS2);

	/* Set X-page to 0 */
	display_page(0);
	display_column(0, DISPLAY_CS1);
	display_column(0, DISPLAY_CS2);
	y_koord = 0;
}
/*
 * Power function for both screens
 * on = 1: Turn On
 * on = 0: Turn Off
 */
void display_power (uint8_t on)
{
	/* Turn on display RS=RW=0 */
	cli();
	write_cmd(DISPLAY_INSTR_POWER|on,DISPLAY_CS1);
	write_cmd(DISPLAY_INSTR_POWER|on,DISPLAY_CS2);
	sei();

}
/*
 * Sets the page address X
 * page: DISPLAY_CS1 or DISPLAY_CS2
 */
void display_page (uint8_t page)
{
```

```
	/*Set page address X */
	write_cmd(DISPLAY_INSTR_X | page,DISPLAY_CS1);
	write_cmd(DISPLAY_INSTR_X | page,DISPLAY_CS2);
}
/*
 * Sets the column address Y
 * column: [0,63]
 * page: DISPLAY_CS1 or DISPLAY_CS2
 */
void display_column (uint8_t column, uint8_t chip)
{
	/* Chip select */
	DISPLAY_INSTR_PORT &= ~(DISPLAY_CS1|DISPLAY_CS2);
	DISPLAY_INSTR_PORT |= chip;

	/*Set column address Y */
	write_cmd(DISPLAY_INSTR_Y | column,chip);

}
/*
 * Write timing data
 */
void write_instr (uint8_t data, uint8_t chip)
{

      /* Chip select */
	DISPLAY_INSTR_PORT &= ~(DISPLAY_CS1|DISPLAY_CS2);
	DISPLAY_INSTR_PORT |= chip;

	/* Start E=RS=1 RW=0 */
	DISPLAY_INSTR_PORT |= DISPLAY_E|DISPLAY_RS;
	DISPLAY_INSTR_PORT &= ~DISPLAY_RW;

	/* DB0-DB7=data */
	DISPLAY_DATA_PORT = data;

	/* E=0 */
	DISPLAY_INSTR_PORT &= ~DISPLAY_E;

	/* ~CS1,CS2,CS RW=0 */
	DISPLAY_INSTR_PORT ^= DISPLAY_CS1|DISPLAY_CS2|DISPLAY_RW|DISPLAY_RS;

	/* E=1 */
	DISPLAY_INSTR_PORT |= DISPLAY_E;

	/* E=0 */
	DISPLAY_INSTR_PORT &= ~(DISPLAY_E);
```

```
	/* RW=RS=1, CS1=CS2=0 */
	DISPLAY_INSTR_PORT ^= DISPLAY_CS1|DISPLAY_CS2|DISPLAY_RW|DISPLAY_RS;
}
/*
 * Write timing cmd
 */
void write_cmd (uint8_t cmd, uint8_t chip)
{
	cli();
	DISPLAY_DATA_DDR = 0xFF;
	DISPLAY_DATA_PORT = cmd;

	DISPLAY_INSTR_PORT &= ~(DISPLAY_CS1|DISPLAY_CS2);
	DISPLAY_INSTR_PORT |= chip;

	DISPLAY_INSTR_PORT &= ~(DISPLAY_RW|DISPLAY_RS);
	DISPLAY_INSTR_PORT |= DISPLAY_E;

	DISPLAY_INSTR_PORT &= ~(DISPLAY_E);
	sei();
}

/*
 * Write routine for 8pixels
 */
void write_display (uint8_t data)
{
	if (y_koord < 64)
	{
		write_instr(data,DISPLAY_CS1);
	}
	else
	{
		write_instr(data,DISPLAY_CS2);
	}
	y_koord%127;
	y_koord++;
}


/*
 * Prints a 5x7 pixel string (max 21 char) to line [0,7]
 */
void print_line5x7 (char *str, uint8_t line)
{

	int pos,j,i;
	unsigned char value;
```

```
	/* Set page to line and both columns to 0 */
	display_page(line);
	display_column(0, DISPLAY_CS1);
	display_column(0, DISPLAY_CS2);
	y_koord = 0;

	cli();

	for (i=0; i<22; i++) {

		if    (*str == 0x00) { break;}

		for (j=0; j<5; j++)
		{
			pos = (*str-32)*5;
			pos += j;


			value = Font5x7[pos];

			write_display(value);
		}
		write_display(0x00);
		str++;
	}
	sei();
}
/*
 * Clear rows [0,7]
 * row: 0x01:0x08
 *		0xFF clears entire screen
 */
void clear_display(uint8_t row)
{
	switch (row) {
		case 0xFF:	{
			print_line5x7("                                ",0);
			print_line5x7("                                ",1);
			print_line5x7("                                ",2);
			print_line5x7("                                ",3);
			print_line5x7("                                ",4);
			print_line5x7("                                ",5);
			print_line5x7("                                ",6);
			print_line5x7("                                ",7);
			break;
		}
		case 0x00: print_line5x7("                                ",0); break;
```

```
                case 0x01: print_line5x7("                    ",1); break;
                case 0x02: print_line5x7("                    ",2); break;
                case 0x03: print_line5x7("                    ",3); break;
                case 0x04: print_line5x7("                    ",4); break;
                case 0x05: print_line5x7("                    ",5); break;
                case 0x06: print_line5x7("                    ",6); break;
                case 0x07: print_line5x7("                    ",7); break;
        }
}

/*
 * uint_2_str returns a string representation of
 * an unsigned integer value. Does not support signed
 * values!
 *
 * str should be defined as char[] of length 11
 *
 * max_length defines the max length of the string
 * if value exceeds max_length the upper values
 * are cut off. [1,10]
 */
char* uint_2_str (uint32_t input, char* str, uint8_t max_length)
{
        uint32_t divider = 1000000000;
        int i = 0;
        char *j = 0;

        str[10] = 0x00; //To ensure the string is cut

        if (max_length > 10 || max_length < 1) {
                max_length = 10; // If faulty input
        }
        if (input == 0) {
                str[0] = 0x30;
                str[1] = 0x00;
        } else {

                j=str;
                while (divider) {
                        *j = input/divider;
                        j++;
                        input %= divider;
                        divider /= 10;
                }

                j=str;
                while (*j == 0 && j<str+10) {
                        i++;
```

```
                j++;
            }
            for (j=str; j<str+10; j++) {
                switch (*j) {
                    case 0: *j = 0x30; break;
                    case 1: *j = 0x31; break;
                    case 2: *j = 0x32; break;
                    case 3: *j = 0x33; break;
                    case 4: *j = 0x34; break;
                    case 5: *j = 0x35; break;
                    case 6: *j = 0x36; break;
                    case 7: *j = 0x37; break;
                    case 8: *j = 0x38; break;
                    case 9: *j = 0x39; break;
                }
            }

            if (i+max_length < 10) {
                return &str[10-max_length];
            }
        }
        return &str[i];
}
```

Thermometer.c

```
#include <stdint.h>
#include <avr/io.h>
#include "Thermometer.h"

/*
 * Initializes the AD converter on ADC0 and ADC1
 */
void init_ADConverter ()
{

    /* Set enable AD-conversion, prescaler=1/128 */
    ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);

    /* Disable comparator */
    ACSR = (1<<ACD);

    /* Enable internal Vref 2.56V */
    ADMUX = (1<<REFS1)|(1<<REFS0);

}
/*
 * Reads the current ADC value
 */
```

```
uint16_t read_ADC(uint8_t chip)
{
	int i;
	uint16_t value = 0;

	/* Chip select */
	ADMUX &= 0xE0;
	ADMUX |= chip;

	for (i=0; i<8; i++) {

		ADCSRA |= (1<<ADSC);

		while (!(ADCSRA & (1<<ADIF))) {} // Wait for ADC to finish

		ADCSRA |= (1<<ADIF); // Set ADIF to clear it

		value += ADC+OFFSET_ERROR; // Takes offset error into account
	}

	return value/8;
}
```

Buttons.h

```
#ifndef BUTTONS_H
#define BUTTONS_H

#define BUTTON_PORT		PORTB
#define BUTTON_DDR		DDRB
#define BUTTON_PIN		PINB
#define BUTTON_INT2		(0x01<<2)
#define BUTTON_MENU			(0x01<<4)
#define BUTTON_MINUS		(0x01<<5)
#define BUTTON_PLUS		(0x01<<6)

/*
 * Initialize routine for buttons
 */
void init_buttons (void);

/*
 * Menu button routine
 */
void menu_button ();

/*
 * Minus button routine
 */
```

```
void minus_button ();

/*
 * Plus button routine
 */
void plus_button ();

#endif
```

Counter0.h

```
#ifndef COUNTER0_H
#define COUNTER0_H

/* Define Counter 0 port */
#define COUNTER0_PORT        PORTB
#define COUNTER0_DDR         DDRB
#define COUNTER0_PIN PINB

/* Define Counter0 pins */
#define COUNTER0_T0          (0x00)
#define COUNTER0_OC0         (0x01<<3)

/*
 * Initialize Timer/Counter0 PWM routine
 */
void init_PWM0 ();

#endif
```

Counter2.h

```
#ifndef COUNTER2_H
#define COUNTER2_H

/*
 * Initialize routine for the real time clock
 */
void init_realTimeClock(void);

#endif
```

Fancontroller.h

```
#ifndef FANCONTROLLER_H
#define FANCONTROLLER_H


/*
 * Main update loop controlled by real time clock:
 * - Update PI-regulator
```

```
 * - Update Display
 */
void update_main();

/*
 * Initialize routine for the PI-regulator
 */
void init_FanController ();

/*
 * Defines a new R value for the regulator
 */
void reg_set_R (uint8_t value);

/*
 * Updates the PI-regulator
 */
void    reg_update ();

#endif
```

Font5x7.h

```
/*! \file font5x7.h \brief Graphic LCD Font (Ascii Characters). */
//*****************************************************************************
//
// File Name     : 'font5x7.h'
// Title         : Graphic LCD Font (Ascii Charaters)
// Author                : Pascal Stang
// Date                  : 10/19/2001
// Revised               : 10/19/2001
// Version               : 0.1
// Target MCU    : Atmel AVR
// Editor Tabs   : 4
//
//*****************************************************************************

#ifndef FONT5X7_H
#define FONT5X7_H

// standard ascii 5x7 font
// defines ascii characters 0x20-0x7F (32-127)

static unsigned char Font5x7[] = {
        0x00, 0x00, 0x00, 0x00, 0x00,// (space)
        0x00, 0x00, 0x5F, 0x00, 0x00,// !
        0x00, 0x07, 0x00, 0x07, 0x00,// "
        0x14, 0x7F, 0x14, 0x7F, 0x14,// #
        0x24, 0x2A, 0x7F, 0x2A, 0x12,// $
```

```
0x23, 0x13, 0x08, 0x64, 0x62,// %
0x36, 0x49, 0x55, 0x22, 0x50,// &
0x00, 0x05, 0x03, 0x00, 0x00,// '
0x00, 0x1C, 0x22, 0x41, 0x00,// (
0x00, 0x41, 0x22, 0x1C, 0x00,// )
0x08, 0x2A, 0x1C, 0x2A, 0x08,// *
0x08, 0x08, 0x3E, 0x08, 0x08,// +
0x00, 0x50, 0x30, 0x00, 0x00,// ,
0x08, 0x08, 0x08, 0x08, 0x08,// -
0x00, 0x60, 0x60, 0x00, 0x00,// .
0x20, 0x10, 0x08, 0x04, 0x02,// /
0x3E, 0x51, 0x49, 0x45, 0x3E,// 0
0x00, 0x42, 0x7F, 0x40, 0x00,// 1
0x42, 0x61, 0x51, 0x49, 0x46,// 2
0x21, 0x41, 0x45, 0x4B, 0x31,// 3
0x18, 0x14, 0x12, 0x7F, 0x10,// 4
0x27, 0x45, 0x45, 0x45, 0x39,// 5
0x3C, 0x4A, 0x49, 0x49, 0x30,// 6
0x01, 0x71, 0x09, 0x05, 0x03,// 7
0x36, 0x49, 0x49, 0x49, 0x36,// 8
0x06, 0x49, 0x49, 0x29, 0x1E,// 9
0x00, 0x36, 0x36, 0x00, 0x00,// :
0x00, 0x56, 0x36, 0x00, 0x00,// ;
0x00, 0x08, 0x14, 0x22, 0x41,// <
0x14, 0x14, 0x14, 0x14, 0x14,// =
0x41, 0x22, 0x14, 0x08, 0x00,// >
0x02, 0x01, 0x51, 0x09, 0x06,// ?
0x32, 0x49, 0x79, 0x41, 0x3E,// @
0x7E, 0x11, 0x11, 0x11, 0x7E,// A
0x7F, 0x49, 0x49, 0x49, 0x36,// B
0x3E, 0x41, 0x41, 0x41, 0x22,// C
0x7F, 0x41, 0x41, 0x22, 0x1C,// D
0x7F, 0x49, 0x49, 0x49, 0x41,// E
0x7F, 0x09, 0x09, 0x01, 0x01,// F
0x3E, 0x41, 0x41, 0x51, 0x32,// G
0x7F, 0x08, 0x08, 0x08, 0x7F,// H
0x00, 0x41, 0x7F, 0x41, 0x00,// I
0x20, 0x40, 0x41, 0x3F, 0x01,// J
0x7F, 0x08, 0x14, 0x22, 0x41,// K
0x7F, 0x40, 0x40, 0x40, 0x40,// L
0x7F, 0x02, 0x04, 0x02, 0x7F,// M
0x7F, 0x04, 0x08, 0x10, 0x7F,// N
0x3E, 0x41, 0x41, 0x41, 0x3E,// O
0x7F, 0x09, 0x09, 0x09, 0x06,// P
0x3E, 0x41, 0x51, 0x21, 0x5E,// Q
0x7F, 0x09, 0x19, 0x29, 0x46,// R
0x46, 0x49, 0x49, 0x49, 0x31,// S
0x01, 0x01, 0x7F, 0x01, 0x01,// T
```

```
        0x3F, 0x40, 0x40, 0x40, 0x3F,// U
        0x1F, 0x20, 0x40, 0x20, 0x1F,// V
        0x7F, 0x20, 0x18, 0x20, 0x7F,// W
        0x63, 0x14, 0x08, 0x14, 0x63,// X
        0x03, 0x04, 0x78, 0x04, 0x03,// Y
        0x61, 0x51, 0x49, 0x45, 0x43,// Z
        0x00, 0x00, 0x7F, 0x41, 0x41,// [
        0x02, 0x04, 0x08, 0x10, 0x20,// "\"
        0x41, 0x41, 0x7F, 0x00, 0x00,// ]
        0x00, 0x06, 0x09, 0x09, 0x06,// ^ We changed this row to include the degree char from extended
Ascii
        0x40, 0x40, 0x40, 0x40, 0x40,// _
        0x00, 0x01, 0x02, 0x04, 0x00,// `
        0x20, 0x54, 0x54, 0x54, 0x78,// a
        0x7F, 0x48, 0x44, 0x44, 0x38,// b
        0x38, 0x44, 0x44, 0x44, 0x20,// c
        0x38, 0x44, 0x44, 0x48, 0x7F,// d
        0x38, 0x54, 0x54, 0x54, 0x18,// e
        0x08, 0x7E, 0x09, 0x01, 0x02,// f
        0x08, 0x14, 0x54, 0x54, 0x3C,// g
        0x7F, 0x08, 0x04, 0x04, 0x78,// h
        0x00, 0x44, 0x7D, 0x40, 0x00,// i
        0x20, 0x40, 0x44, 0x3D, 0x00,// j
        0x00, 0x7F, 0x10, 0x28, 0x44,// k
        0x00, 0x41, 0x7F, 0x40, 0x00,// l
        0x7C, 0x04, 0x18, 0x04, 0x78,// m
        0x7C, 0x08, 0x04, 0x04, 0x78,// n
        0x38, 0x44, 0x44, 0x44, 0x38,// o
        0x7C, 0x14, 0x14, 0x14, 0x08,// p
        0x08, 0x14, 0x14, 0x18, 0x7C,// q
        0x7C, 0x08, 0x04, 0x04, 0x08,// r
        0x48, 0x54, 0x54, 0x54, 0x20,// s
        0x04, 0x3F, 0x44, 0x40, 0x20,// t
        0x3C, 0x40, 0x40, 0x20, 0x7C,// u
        0x1C, 0x20, 0x40, 0x20, 0x1C,// v
        0x3C, 0x40, 0x30, 0x40, 0x3C,// w
        0x44, 0x28, 0x10, 0x28, 0x44,// x
        0x0C, 0x50, 0x50, 0x50, 0x3C,// y
        0x44, 0x64, 0x54, 0x4C, 0x44,// z
        0x00, 0x08, 0x36, 0x41, 0x00,// {
        0x00, 0x00, 0x7F, 0x00, 0x00,// |
        0x00, 0x41, 0x36, 0x08, 0x00,// }
        0x08, 0x08, 0x2A, 0x1C, 0x08,// ->
        0x08, 0x1C, 0x2A, 0x08, 0x08 // <-
};

#endif
```

Global.h

```
#include <stdint.h>

#ifndef GLOBAL_H
#define GLOBAL_H

#define		TEMP1_CHIP			0x01
#define		TEMP2_CHIP			0x00

/* LCD Global variables */
uint8_t y_koord;

/* Counter2 Global variables */
uint32_t time;

/* Main routine variables */
uint8_t upd_graph;
uint8_t upd_main;

/* PI-regulator:
 *
 * The temperature (ADC-value) is used to regulate
 * the percentage of maximum power to the fan.
 *
 * u(t)=Kp*(Y(t)-R(t)+Ki*Integrate((Y(x)-R(x)),0,t) <=>
 * u(t)=Kp*E(t)+Ki*Integrate(E(x),0,t), E(t)=Y(t)-R(t) <=>
 * U(k)=Kp*E(k)+Ki*SUM(E(l),0,k) <=>
 * U(k)=Kp*E(k)+Ki*(Eint+Ek), Eint=SUM(E(l),0,k-1)
 * U will correspond to a OCR0 value [1,254]
 */

int16_t		reg_E;				// E(k) defines the offset between Y(k) and R(k)
uint16_t	reg_Y;				// Y(k) defines the current ADC value [0,1023]
uint8_t		reg_R;				// R(k) defines the reference value
int16_t		reg_Eint;			// Eint is the sum of the previous E(k) since start or last
reference value change
uint16_t	reg_Kp;				// Kp is the constant for the P-regulator
uint16_t	reg_Ki;				// Ki is the constant for the I-regulator
uint8_t		reg_R_lowerlimit;	// To prevent FAN damage
uint8_t		reg_enable;			// Bit 0 Set = P-regulator part enable, Bit 1 Set =
I-regulator part enable

#endif
```

GraphDrawer.h

```
#include <stdint.h>

#ifndef GRAPHDRAWER_H
```

```
#define GRAPHDRAWER_H

uint16_t room_graph[62];
uint16_t fan_graph[62];

/*
 * Initialize routine for the GraphDrawer
 */
void init_GraphDrawer ();

/*
 * Update the graph
 * Each value are shifted left once
 */
void update_graph();

#endif
```

Lcd.c

```
#ifndef LCD_H
#define LCD_H

/* Define hardware pins for the display */
#define DISPLAY_INSTR_PORT   PORTA
#define DISPLAY_INSTR_DDR    DDRA
#define DISPLAY_INSTR_PIN    PINA
#define DISPLAY_CS1          (0x01<<2)
#define DISPLAY_CS2          (0x01<<3)
#define DISPLAY_E                (0x01<<4)
#define DISPLAY_RW               (0x01<<5)
#define DISPLAY_RS               (0x01<<6)
#define DISPLAY_RESET            (0x01<<7)

/* Define data pins for the display */
#define DISPLAY_DATA_PORT   PORTD
#define DISPLAY_DATA_DDR    DDRD
#define DISPLAY_DATA_PIN    PIND
#define DISPLAY_DB0         (0x01<<0)
#define DISPLAY_DB1         (0x01<<1)
#define DISPLAY_DB2         (0x01<<2)
#define DISPLAY_DB3         (0x01<<3)
#define DISPLAY_DB4         (0x01<<4)
#define DISPLAY_DB5         (0x01<<5)
#define DISPLAY_DB6         (0x01<<6)
#define DISPLAY_DB7         (0x01<<7)

/* Define data port instructions */
#define DISPLAY_INSTR_POWER 0x3E
```

```
#define DISPLAY_INSTR_X         0xB8
#define DISPLAY_INSTR_Y                 0x40
#define DISPLAY_INSTR_Z                 0xC0

/* Define status read instruction */
#define DISPLAY_STATUS                  0x02

/*
  * Initialize routine for display
  */
void init_display (void);

/*
  * Power function for both screens
  * on = 1: Turn On
  * on = 0: Turn Off
  */
void display_power (uint8_t on);

/*
  * Hardware initialize routine for display
  */
void init_harddisplay (void);

/*
  * Sets the page address X
  * page: DISPLAY_CS1 or DISPLAY_CS2
  */
void display_page (uint8_t page);

/*
  * Sets the column address Y
  * column: [0,63]
  * page: DISPLAY_CS1 or DISPLAY_CS2
  */
void display_column (uint8_t column, uint8_t chip);

/*
  * Write timing data
  */
void write_instr (uint8_t data, uint8_t chip);

/*
  * Write timing cmd
  */
void write_cmd (uint8_t cmd, uint8_t chip);

/*
```

```
 * Write routine for 8pixels
 */
void write_display (uint8_t data);

/*
 * Prints a 5x7 pixel string (max 21 char) to line [0,7]
 */
void print_line5x7 (char *str, uint8_t line);

/*
 * Clear rows [0,7]
 * row: 0x01:0x08
 *                  0xFF clears entire screen
 */
void clear_display(uint8_t row);

/*
 * uint_2_str returns a string representation of
 * an unsigned integer value. Does not support signed
 * values!
 *
 * str should be defined as char[] of length 11
 *
 * max_length defines the max length of the string
 * if value exceeds max_length the upper values
 * are cut off. [1,10]
 */
char* uint_2_str (uint32_t input, char* str, uint8_t max_length);

#endif
```

Thermometer.c

```
#ifndef THERMOMETER_H
#define THERMOMETER_H

#define         OFFSET_ERROR            -26     // Defines offset error
#define         TEMP1_CHIP                      0x01
#define         TEMP2_CHIP                      0x00


/*
 * Initializes the AD converter on ADC0 and ADC1
 */
void init_ADConverter ();

/*
 * Reads the current ADC value
 */
```

```
uint16_t read_ADC(uint8_t chip);

#endif
```