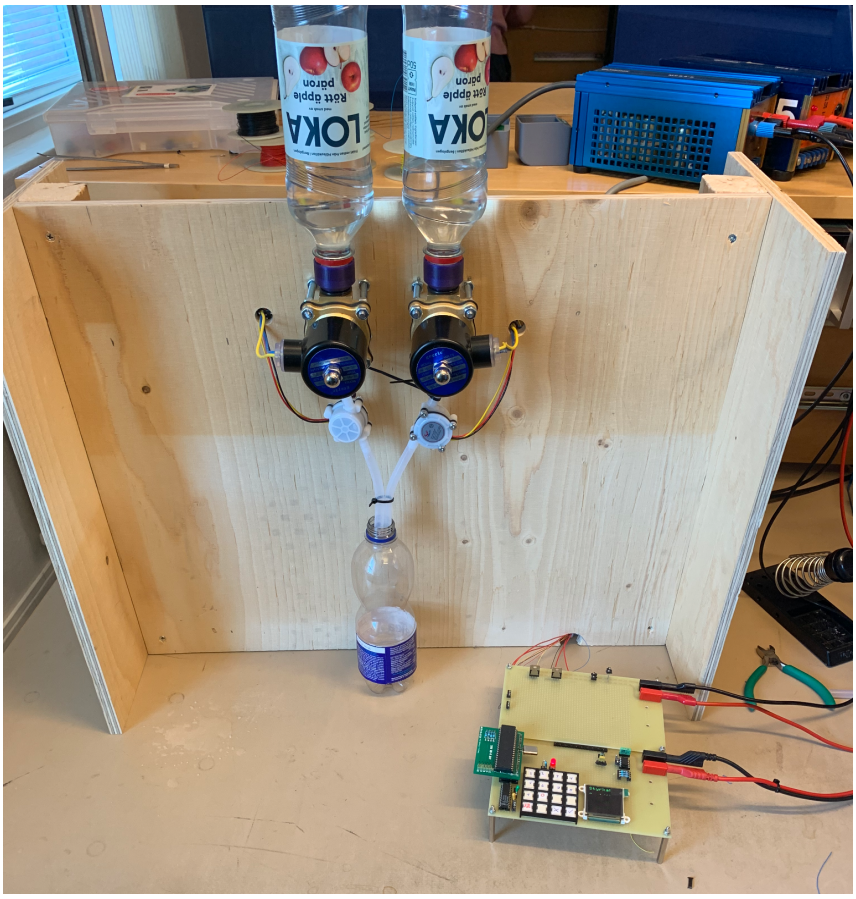
Lunds Tekniska Högskola

EITF12 - Digitala Projekt

Bartender 2000



Studenter: Axel Strömbäck, Henrik Wexell och Fabian Rosén

Handledare: Christoffer Cederberg

Kursansvarig: Bertil Lindvall

Abstract

The purpose of this project was to build a simple prototype of something from scratch. This group decided to build an automatic drink-mixer, named Bartender 2000. This report describes the functionality, method of operating and layout of Bartender 2000. Construction of Bartender 2000 was successful and ended up with a working prototype. This project is part of the course EITF12 - Digital projects at Lund University.

1 Introduktion

1.1 Syfte

Kursen EITF12 Digitala Projekt gick ut på att illustrera industriellt utvecklingsarbete genom byggandet av en enklare konstruktion. Konstruktionen utgår ifrån processorn AVR ATmega16 och del av kursen går även ut på att programmera denna konstruktion. Den konstruktion som gruppen valde att bygga var en drinkblandare, nedan kallad Bartender 2000.

1.2 Kravspecifikation

Konstruktionen skall:

- Ha en display med tillhörande knappsats för val av dryck.
- Ha ett UI bestående av en meny där dryck kan väljas med hjälp av knappsatsen. UI skall även kräva användaren på en PIN-kod för att nå denna meny.
- Hälla upp korrekta drinkar på ett konsekvent sätt.
- Endast fungera bestämda tider på dygnet.
- Kräva PIN-kod vid inlogg.

1.3 Metod

Byggandet av hårdvaran samt kodningen av mjukvaran kan ses som process uppdelad i flera mindre delsteg där gruppen tillsammans med handledare successivt konstruerade Bartender 2000 steg för steg. Designbeslut överlades med handledare för att komma fram till den bästa lösningen på olika konstruktionsutmaningar.

Till en start kopplades endast processor, knappsats och display samman med hjälp av tillhörande komponenter för att få dessa att fungera. Därefter skrevs nödvändig kod för att få dessa att samspela. Parallellt med detta testades funktioner löpande för att snabbt identifiera potentiella fel som behövde rättas till. När gruppen väl fått dessa att fungera och samspela som önskat fortsatte arbetet med att utveckla interface för menyn. Därefter kopplades ventiler och flödesmätare på parallellt med skrivandet av nödvändig kod för att få dessa att fungera. Samtidigt monterades pumpar och flödesmätare på en ställning. Slutligen färdigställdes menyn, klockan och inlogg.

2 Hårdvara

2.1 Ingående komponenter

För konstruktionen användes följande huvudsaklig hårdvara:

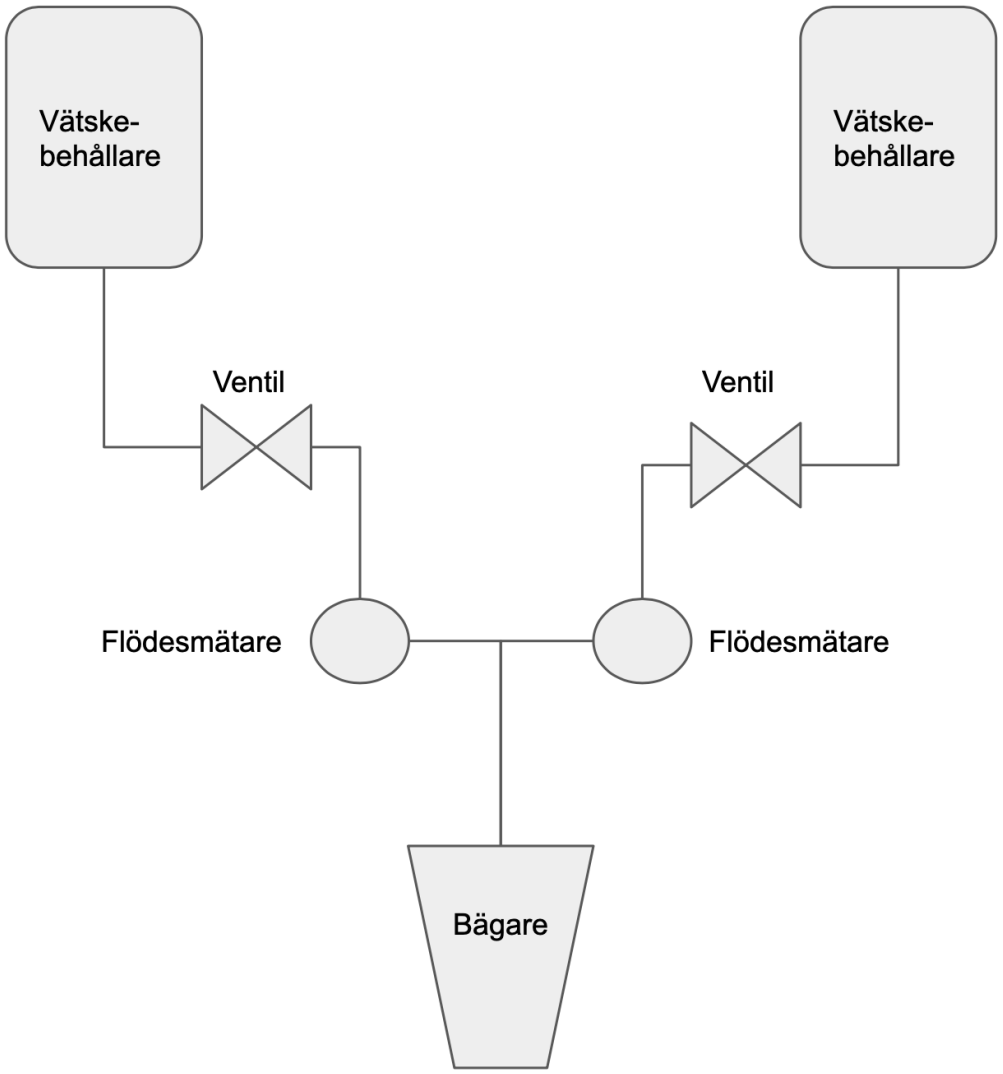
- 1st AVR AT Mega16
    - Processorn är den centrala komponenten som styr övriga delar av konstruktionen. För att programmera processorn användes en JTAG ICE som fick sin input av kod från Atmel studios.
- 1st knappsats med 16 knappar
    - Analog knappsats som lästes av med hjälp av key encodern.
- 1st MM74C923 16-Key Encoder
    - Modulen som läste av den analoga inputen från knappsatsen för att sedan skicka vidare den till processorn som digital information.
- 1st uOLED-128-G1
    - En digital LED-skärm med inbyggda funktioner som kontrollerades av processorn.
- 2 st YF-S401 flödesmätare
    - Flödesmätare för att mäta mängden vätska som strömmar ut från ventilerna.
- 2 st geerte 2W-160-15 ventiler
    - Ventiler för att släppa på flödet av vätska.

Övriga komponenter som ingick i bygget var:

- Kondensatorer
- Resistorer
- Oscillatorer
- Transistor
- Sladdar
- 3D-printade kompenter för fästande av flaskor och slangar
- Övrigt byggmaterial för ställning
- Sladdar
- Slangar
- Flaskor

2.2 Övergripande konstruktion

Blockschema över hur hårdvarukomponenterna som hanterar vätska sitter ihop kan ses i *figur 1* och *bild 1*. Komponenterna som hanterar dryck sitter ihop så att överst är två behållare för vätska som är kopplade till varsin ventil. Ventilerna är i sin tur kopplade till varsin flödesmätare som mynnar ut i två slangar under vilka man ställer bägaren som man vill ha drycken hälld i.



Figur 1: Blockschema över hårdvarukomponenter som hanterar vätska.

Bild 1: Konstruktionen sedd framifrån.

På *bild 2* syns ovansidan av kopplingsbrädet. De hårdvarukomponenter som utnyttjas vid användning av Bartender 2000 är knappsatsen, displayen och lampan. En mer detaljerad beskrivning av hur allt är kopplat ges i *figur 2*.

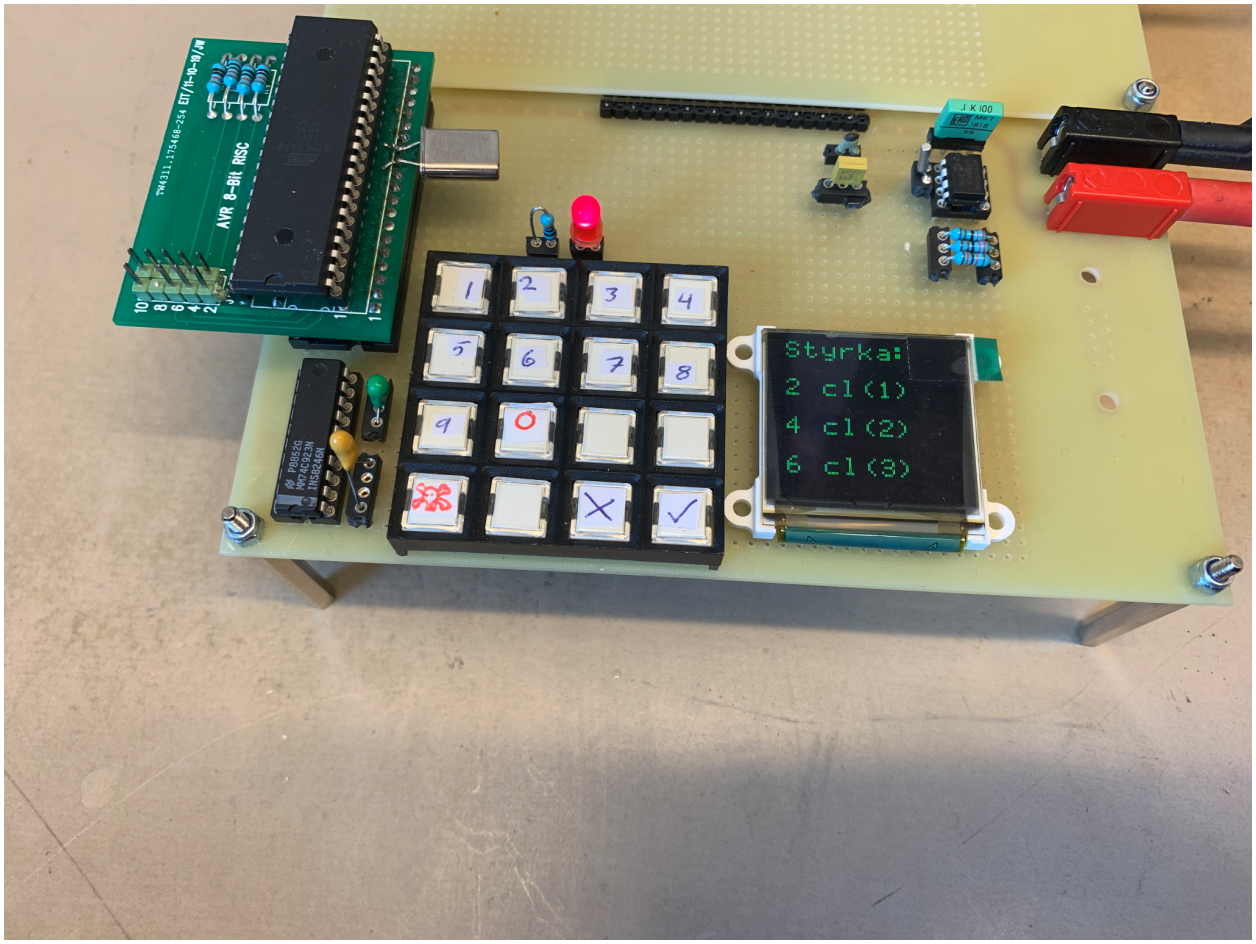


Bild 2: Ovansidan av kopplingsbordet.

3 Funktion

3.1 Interface

Vid aktivering av Bartender 2000 ges användaren först möjlighet att skriva in vad klockan är. Det finns även en lampa på kopplingsbordet som lyser för att indikera att Bartender 2000 är påslagen. Är klockan något mellan kl 13:00-23:59 ges användaren möjlighet att skriva in PIN-kod för att sedan komma åt drinkmenyn. Väl inne i drinkmenyn gör man val av drink med hjälp av knappsatsen i enlighet med instruktionerna från skärmen. På grund av begränsad tillgång i antal ventiler går det endast att välja olika starka drinkar av samma sort. Efter val av drink hälls önskad drink upp av Bartender 2000.

3.2 Funktion av hårdvara

När användaren har gjort ett val att få en specifik drink blandad så öppnas rätt ventiler under en lagom lång tidsperiod för att hälla upp korrekt mängd av ingående volymer. För att assistera så detta blir rätt finns det även flödesmätare som säkerställer att korrekt mängd vätska hälls upp i glaset.

3.2 Funktion av mjukvaran

Detta avsnitt är till för att beskriva de ingående klasserna i mjukvaran samt deras respektive funktioner och hur de hänger ihop.

3.2.1 main

Klassen som innehåller main-metoden och som allt utgår ifrån.

3.2.2 menu

Denna klass är till för att hantera drinkmenyn. Den innehåller de olika menyvalen och det är även från denna klass som ventilerna styrs.

3.2.3 timedisplay

Är till för att hantera tiden. Inmatning av tiden samt funktion som håller baren stängd mellan kl00-13 finns här.

3.2.4 login

Hanterar inloggningssidan där man behöver fylla i PIN-kod.

3.2.5 i2c

Hjälpklass för RTC.

3.2.6 rtc

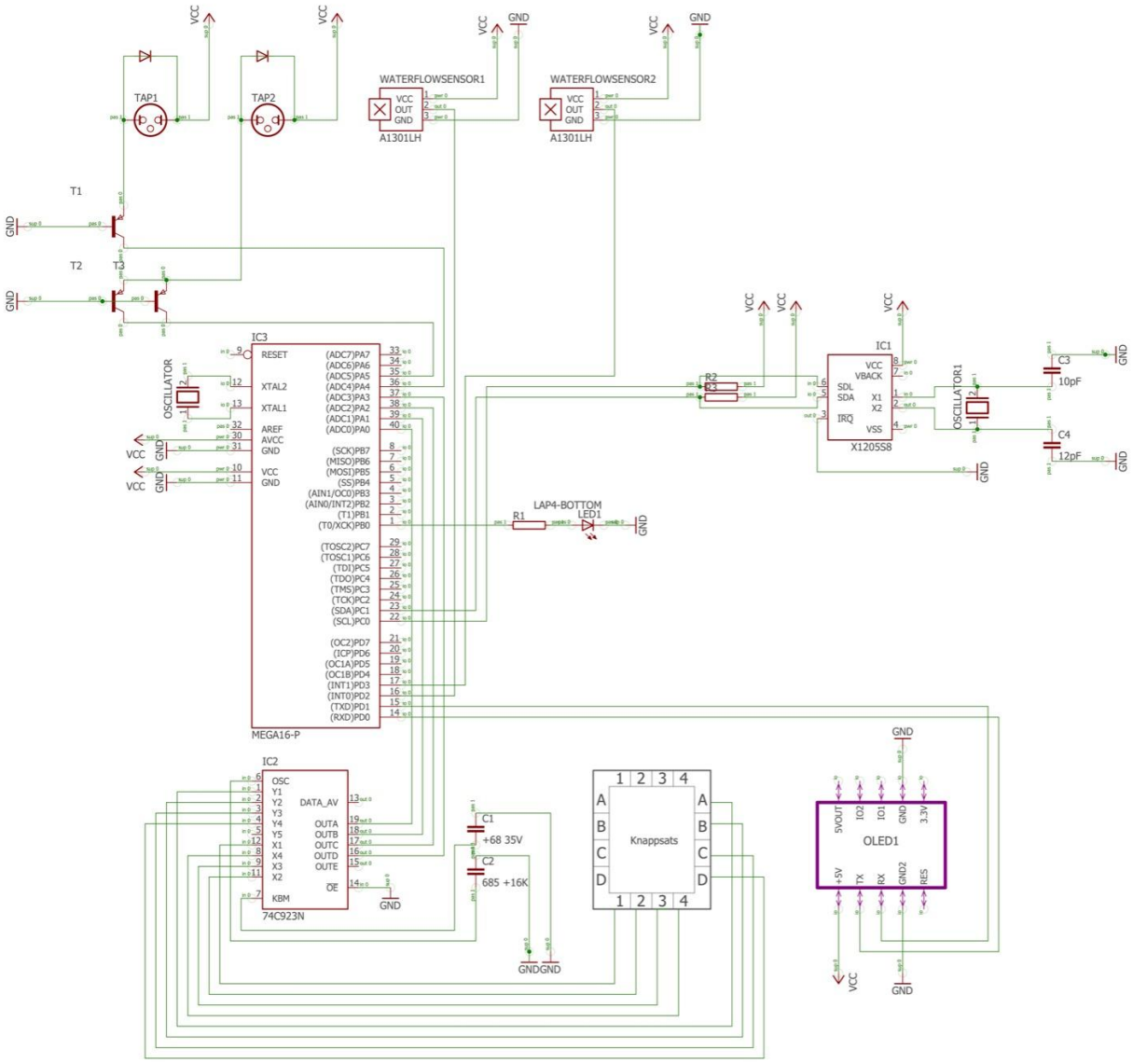
Innehåller metoder för att läsa av och skriva till klockan.

3.2.7 uart

Klass till för att hantera och styra skärmen.

4 Kopplingsschema

Kopplingsschema för Bartender 2000 ges i *figur 2*.



Figur 2: Kopplingsschema.

5 Resultat

Vid färdigställande fungerade Bartender 2000 som väntat. Samtliga punkter från kravspecifikationen uppfylldes och konstruktionen fungerade väl på ett konsekvent vis. Även om drinkmenyn blev liten, på grund av brist på ventiler och flödesmätare, så gick det att få en variation av olika drinkar blandade från de två dryckeskomponenter som gick att blanda ifrån.

En del av funktionen som hade kunnat fungera bättre var knappsatsen som inte var speciellt responsiv. Det var dock en nödvändighet att göra knappsatsen så icke-responsiv med hjälp av en kondensator för att motverka problem med ofrivilliga signaler från knappsatsen som skedde spontant utan denna kondensator.

6 Diskussion

Projektet har varit lärorikt för att förstå de utmaningar som finns vid konstruktionen av även lättare konstruktioner. Kursen har gett hel gruppen nya insikter kring processen för produktframtagning samt en helt ny förståelse för hur hårdvara och mjukvara integreras.

Den största utmaningen med projektet var hur nytt allt var för alla medlemmar i gruppen. Programmeringen i C var en utmaning till en start men något som gruppen allteftersom lärde sig under projektet gång. När gruppen väl blivit bekant i Atmel studios flöt själva programmeringen på bra men en utmaning var att förstå hur skärmen skulle styras för att få denna att fungera.

När det kommer till kopplingen av hårdvara så flöt den på bra och gruppen lärde sig snabbt att tolka manualer för att förstå hur sladdar skulle dras. Något som var en utmaning dock var när det behövdes resistorer, kondensatorer, transistorer och dylikt. Gruppens brist på förkunskap inom elektronik gjorde sig väl påmind men denna utmaning överkom gruppen med hjälp av handledning från handledarna.

Även om Bartender 2000 uppfyllde kravspecifikationen och fungerade tillfredsställande så finns det förbättringspotential. Något som hade kunnat förbättras är knappsatsens responsivitet som inte alltid var så god som man hade kunnat önska. Mer testning med olika starka kondensatorer hade möjligtvis kunnat mynna ut i en mer optimalt fungerande knappsats. Ett annat förbättringsområde är en utökning av antalet vätskebehållare och ventiler för att utöka mängden möjliga dryckeskombinationer. Ingående dryckeskomponenter fick i detta projekt begränsas till två på grund av brist på fler ventiler.

7 Appendix

7.1 Källkod

7.1.1 f_cpu.h

```
/*
```

```
 * f_cpu.h
 *
 * Created: 2022-05-11 12:14:36
 *  Author: ax6446st-s
 */


#ifndef F_CPU_H_
#define F_CPU_H_

#define F_CPU    16000000UL



#endif /* F_CPU_H_ */
```

7.1.2 main.c

```
/*
 * GccApplication1.c
 *
 * Created: 2022-03-29 08:37:03
 * Author : ax6446st-s
 */
#include "f_cpu.h"
#include <avr/interrupt.h>
#include <avr/io.h>
#include <util/delay.h>
#include <inttypes.h>
#include <compat/twi.h>

#include "rtc.h"
 #include "timedisplay.h"
 #include "uart.h"
 #include "menu.h"
 #include "login.h"

#define LED1     (1 << PB0)

volatile uint8_t knapp;
volatile int pour1 = 0;
volatile int pour2 = 0;
volatile char rData = 0;
volatile uint8_t knappflagga;
volatile rtc_time_t time;

void ext_int2_init();
```

```
int main(void)
{
	DDRB |= LED1; //lampor
	DDRA &= ~((1 << PA0) | (1 << PA1) | (1 << PA2) | (1 << PA3));
//knappar

	knapp = 0;

	PORTB &= ~LED1;
	_delay_ms(3000); // delay is needed for OLED screen startup

	uart_init_oled();
	ext_int2_init();


	sei();  //  enables global interrupt
	display_clear_screen();
	_delay_ms(50);
	twi_init();
	init_taps();

	PORTB |= LED1;
	set_time();
	rtc_start_clock();
	display_time();
	_delay_ms(50);
	login();
	menu();

}
void ext_int2_init()
{
	MCUCSR |= (1<<ISC2); // Rising edge triggers interrupt 2.
	GICR |= (1<<INT2);
}

ISR(INT2_vect)
{
	knapp = PINA & 0b00001111;
	knappflagga = 1;
}



ISR(INT0_vect)
```

```
{
	pour1++;
}

ISR(INT1_vect)
{
	pour2++;
}


ISR(USART_RXC_vect) //interrupt service routine for receiver
{
	rData = UDR; //receive data
}
```

7.1.3 login.h

```
/*
 * login.h
 *
 * Created: 2022-05-13 17:19:31
 *  Author: ax6446st-s
 */


#ifndef LOGIN_H_
#define LOGIN_H_

#include "uart.h"
#include <avr/io.h>
void login();


extern volatile uint8_t knapp;
extern volatile uint8_t knappflagga;

#endif /* LOGIN_H_ */
```

7.1.4 login.c

```
/*
 * login.c
 *
 * Created: 2022-05-13 17:18:59
 *  Author: ax6446st-s
 */
#include "login.h"
```

```
void login()
{
	display_clear_screen();
	unsigned char text[] = {0x00, 0x06, 'P', 'I', 'N', ':', 0x00};
	unsigned char star[] = {0x00, 0x06, '*', 0x00};
	display_move_cursor(0,0);
	Uart_Tstr(text, sizeof(text));
	int password[4] = {0, 0, 0, 0};
	int correctPassword[4] = {2, 2, 2, 2};
	int pin = 1;
	int counter = 0;
	while(pin){
		if(knappflagga){
			knappflagga = 0;
			if (knapp == 14){ //erase input
				display_clear_screen();
				display_move_cursor(0,0);
				Uart_Tstr(text, sizeof(text));
				counter = 0;
				for(int i = 0; i < 4; i++){
					password[i] = 0;
				}
				}else if (knapp == 15){ //check if input is correct
				counter = 0;
				int wrongPass = 0;
				for(int i = 0; i < 4; i++){
					if(password[i] != correctPassword[i]){
						wrongPass = 1;
					}
				}
				if(wrongPass){
					display_clear_screen();
					display_move_cursor(0,0);
					Uart_Tstr(text, sizeof(text));
					counter = 0;
					for(int i = 0; i < 4; i++){
						password[i] = 0;
					}
					}else{
					pin = 0;
				}
				}else if(counter<4) { //enter digit to PIN
				password[counter] = knapp;
				counter++;
				Uart_Tstr(star, sizeof(star));
			}
```

```
            }
      }
}
```

7.1.5 menu.h

```
/*
 * menu.h
 *
 * Created: 2022-05-13 15:51:16
 *  Author: ax6446st-s
 */


#ifndef MENU_H_
#define MENU_H_

#include "uart.h"
#include <avr/io.h>
#include <util/delay.h>
#include "f_cpu.h"

#define TAP1     (1 << PA4)
#define TAP2     (1 << PA5)

void menu();
void yes_no(unsigned char x);
void init_taps();
void close_tap1();
void close_tap2();

extern volatile uint8_t knapp;
extern volatile uint8_t knappflagga;
extern volatile int pour1;
extern volatile int pour2;


#endif /* MENU_H_ */
```

7.1.6 menu.c

```
/*
 * menu.c
 *
 * Created: 2022-05-13 15:54:13
 *  Author: ax6446st-s
 */
```

```
#include "menu.h"


void init_taps()
{
	DDRA |= TAP1;
	DDRA |= TAP2;

	/*initiate  flow meter interrupts*/
	MCUCR |= (1<<ISC00) | (1<<ISC10);
	GICR |= (1<<INT0);
	GICR |= (1<<INT1);

	PORTA &= ~TAP1;
	PORTA &= ~TAP2;

}
void menu()
{
	unsigned char menu[4][12] = {
		{0x00, 0x06, 'S', 't', 'y', 'r', 'k', 'a', ':', 0x00},
		{0x00, 0x06, '2', ' ', 'c', 'l', '(' , '1', ')', 0x00},
		{0x00, 0x06, '4', ' ', 'c', 'l', '(' , '2', ')', 0x00},
		{0x00, 0x06, '6', ' ', 'c', 'l', '(' , '3', ')', 0x00},
		};
	unsigned char menurows[4][6] = {
		{0xFF, 0xE4, 0x00, 0x00, 0x00, 0x00},
		{0xFF, 0xE4, 0x00, 0x02, 0x00, 0x00},
		{0xFF, 0xE4, 0x00, 0x04, 0x00, 0x00},
		{0xFF, 0xE4, 0x00, 0x06, 0x00, 0x00}
	};
	display_clear_screen();
	for(int i = 0; i<4; i++){
		_delay_ms(200);
		Uart_Tstr(menurows[i], sizeof(menurows[i]));
		_delay_ms(200);
		Uart_Tstr(menu[i], sizeof(menu[i]));
	}
	while(1)
	{
		if(knappflagga){
			knappflagga = 0;
			if(knapp < 3 || knapp == 12)
				yes_no(knapp);
		}
	}
}
```

```
void yes_no(unsigned char x)
{
	if(knapp==12){
		unsigned char val[] = {0x00, 0x06, 'X' , 'c', 'l', ' ', 'G',
'i', 'n',  '?', 0x00};
		display_clear_screen();
		display_move_cursor(0,0);
		Uart_Tstr(val, sizeof(val));

	}else{
		unsigned char choice = 2 * x + 0x32; //går att lösa mycket
snyggare, skiftar 0x00 till rätt ASCII-tecken
		unsigned char val[] = {0x00, 0x06, choice , 'c', 'l', ' ',
'G', 'i', 'n',  '?', 0x00};
		display_clear_screen();
		display_move_cursor(0,0);
		Uart_Tstr(val, sizeof(val));

	}
	int str = 0;
	if(knapp == 0)
		str = 150;
	else if(knapp == 1)
		str = 350;
	else if (knapp == 2)
		str = 535;
	else if(knapp == 12)
		str = 750;


	while(1) {
		if(knappflagga){
			knappflagga = 0;
			if(knapp == 14){
				PORTA &= ~TAP1;
				PORTA &= ~TAP2;
				menu();
			}else if(knapp == 15){
					while(pour1 < str){
						PORTA |= TAP1;
					}
					close_tap1();
					while(pour2 < 1500)
					{
						PORTA |= TAP2;
					}
					close_tap2();
```

```
                }
            }
        }
}
void close_tap1(){
      PORTA &= ~TAP1;
      pour1 = 0;
}
void close_tap2(){
      PORTA &= ~TAP2;
      pour2 = 0;
}
```

7.1.7 timedisplay.h

```
/*
 * timedisplay.h
 *
 * Created: 2022-05-12 14:44:00
 *  Author: ax6446st-s
 */


#ifndef TIMEDISPLAY_H_
#define TIMEDISPLAY_H_

#include "rtc.h"
#include "uart.h"
#include <util/delay.h>

void display_time();
void set_time();
void printDigit();

unsigned char unSerializeOnes(uint8_t t);
unsigned char unSerializeTens(uint8_t t);

extern volatile uint8_t knapp;
extern volatile uint8_t knappflagga;
extern volatile int set_s;
extern volatile int set_min;
extern volatile int set_h;

#endif /* TIMEDISPLAY_H_ */
```

7.1.8 timedisplay.c

```
/*
 * timedisplay.c
 *
 * Created: 2022-05-12 14:43:33
 *  Author: ax6446st-s
 */

#include "timedisplay.h"
volatile rtc_time_t time;
volatile int counter = 0;
volatile int time_digits[4] = {0, 0, 0, 0};

void display_time()
{
	display_clear_screen();
	_delay_ms(50);
	rtc_read_clock(&time);

	if(time.hour < 13){
		display_move_cursor(2,0);
		unsigned char baren[] = {0x00, 0x06, 'O', 'P', 'E','N', 'S', '
', 'A', 'T', ':' ,0x00 };
		Uart_Tstr(baren, sizeof(baren));

		_delay_ms(50);

		display_move_cursor(3,0);
		unsigned char openingTime[] = {0x00, 0x06, '1', '3', ':','0',
'0', 0x00 };
		Uart_Tstr(openingTime, sizeof(openingTime));
	}
	unsigned char clear_time[] = {0xFF, 0xCE, 0x00, 0x00, 0x00, 0x00,
0x00, 0x7C, 0x00, 0x15, 0x00, 0x00};

	while(time.hour<13){

		rtc_read_clock(&time);
		unsigned char hourOnes = unSerializeOnes(time.hour) + 0x30;
		unsigned char hourTens = unSerializeTens(time.hour) + 0x30;
		unsigned char minutesOnes = unSerializeOnes(time.min) + 0x30;
		unsigned char minutesTens = unSerializeTens(time.min) + 0x30;
		unsigned char secondsOnes = unSerializeOnes(time.sec) + 0x30;
		unsigned char secondsTens = unSerializeTens(time.sec) + 0x30;

		Uart_Tstr(clear_time, sizeof(clear_time));
```

```
            _delay_ms(50);

            display_move_cursor(0,0);
            unsigned char displayTime[] = {0x00, 0x06, hourTens, hourOnes,
':', minutesTens, minutesOnes, ':', secondsTens, secondsOnes, 0x00};
            Uart_Tstr(displayTime, sizeof(displayTime));

            _delay_ms(950);



      }
}

void set_time()
{
      unsigned char ple [] = {0x00, 0x06, ':', 0x00};
      display_move_cursor(0,0);
      unsigned char enter_time[] = {0x00, 0x06, 'S', 'E', 'T', ' ', 'T',
'I', 'M', 'E', ':' ,0x00 };
      Uart_Tstr(enter_time, sizeof(enter_time));
      _delay_ms(50);
      display_move_cursor(1,2);
      Uart_Tstr(ple, sizeof(ple));
      display_move_cursor(1,0);
      int clockTime = 1;
      counter = 0;
      int input_error = 0;
      while(clockTime){
            if(knappflagga){
                  knappflagga = 0;
                  if (knapp == 14){ //erase input
                        display_clear_screen();
                        display_move_cursor(0,0);
                        _delay_ms(50);
                        Uart_Tstr(enter_time, sizeof(enter_time));
                        display_move_cursor(1,2);
                        _delay_ms(50);
                        Uart_Tstr(ple, sizeof(ple));
                        display_move_cursor(1,0);
                        counter = 0;
                        for(int i = 0; i < 4; i++){
                              time_digits[i] = 0;
                        }
                        }else if (knapp == 15){ //check if input is correct
                        input_error = 0;
```

```
                        if(counter<3)
                              input_error = 1;
                        counter = 0;

                        if(input_error){
                              display_clear_screen();
                              display_move_cursor(0,0);
                              _delay_ms(50);
                              Uart_Tstr(enter_time, sizeof(enter_time));
                              display_move_cursor(1,2);
                              _delay_ms(50);
                              Uart_Tstr(ple, sizeof(ple));
                              display_move_cursor(1,0);
                              for(int i = 0; i < 4; i++){
                                    time_digits[i] = 0;
                              }
                        }else{
                              set_h = (time_digits[0]) * 10 +
time_digits[1];
                              set_min = (time_digits[2]) * 10 +
time_digits[3];
                              clockTime = 0;
                        }
                  }else if(counter < 4 && knapp < 10) { //enter digit to
PIN
                        if(counter == 0 && (knapp < 2 || knapp == 9)){
                              printDigit();
                        }
                        else if(counter == 1 && time_digits[0] < 2){
                              printDigit();
                        }
                        else if(counter == 1 && (time_digits[0] == 2) &&
(knapp < 3 || knapp == 9)){
                              printDigit();
                        }
                        else if(counter == 2 && (knapp < 5 || knapp == 9)){
                              printDigit();
                        }
                        else if (counter == 3){
                              printDigit();
                        }
                  }
            }
      }
}

void printDigit(){
```

```
            if (knapp == 9){
                  time_digits[counter] = 0;
                  counter++;
                  if(counter==3)
                  display_move_cursor(1,3);
                  unsigned char digit[] =  {0x00, 0x06, '0',0x00 };
                  Uart_Tstr(digit, sizeof(digit));
            } else {
                  time_digits[counter] = knapp +1;
                  counter++;
                  if(counter==3)
                  display_move_cursor(1,3);
                  unsigned char digit[] =  {0x00, 0x06, knapp + 0x31 ,0x00
};
                  Uart_Tstr(digit, sizeof(digit));
            }
}


unsigned char unSerializeOnes(uint8_t t)
{
      int ones = t%10;
      return ones;
}
unsigned char unSerializeTens(uint8_t t)
{
      int tens = t/10;
      return tens;
}
```

7.1.9 i2c.h

```
/*
 * i2c.h
 *
 * Created: 2022-05-11 11:23:26
 *  Author: ax6446st-s
 */


#ifndef I2C_H_
#define I2C_H_

#include "f_cpu.h"
#define SCL_CLOCK100000UL
#define SLA_R          0b11011111
#define SLA_W          0b11011110
```

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

void twi_init(void);
unsigned char twi_start(unsigned char address);
void twi_start_wait(unsigned char address);
unsigned char twi_rep_start(unsigned char address);
void twi_stop(void);
unsigned char twi_write(unsigned char data);
unsigned char twi_read_ack(void);
unsigned char twi_read_nack(void);
uint8_t read_specific_address(uint8_t address);



#endif /* I2C_H_ */
```

7.1.10 i2c.c

```
/*
 * i2c.c
 *
 * Created: 2022-05-11 11:23:01
 *  Author: ax6446st-s
 */
#include "i2c.h"


void twi_init(void) {
	TWSR = 0;
	TWBR = ((F_CPU/SCL_CLOCK)-16)/2;

}

unsigned char twi_start(unsigned char address) {
	uint8_t twst;

	TWCR = 0;
	TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN); // set interrupt flag -
become master - like in init, starts interface
	while (!(TWCR & (1<<TWINT))); // if TWINT is not set, get stuck
	twst = TW_STATUS & 0xf8;
	if ((twst != TW_START)) return 1;

	// send device address
	TWDR = address;
```

```
	TWCR = (1 << TWINT) | (1 << TWEN);

	// wait for transmission
	while (!(TWCR & (1 << TWINT)));

	// check value of TWI Status Register. Mask prescaler bits.
	twst = TW_STATUS & 0xF8;
	if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) ) return 1;

	return 0;
}

unsigned char twi_rep_start(unsigned char address) {
	return twi_start(address);
}

void twi_stop(void) {
	TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);  // we are done,
interrupt flag, stop bit generates stop, activates interface like before
	// wait until stop condition is executed and bus released
	while(TWCR & (1<<TWSTO));
}


unsigned char twi_write(unsigned char data) {
	uint8_t twst;
	TWDR = data; // data registry containing next byte to transmit
	TWCR = (1<<TWINT) | (1<<TWEN); // interrupt flag that we are done -
reactivates interface
	while(!(TWCR & (1<<TWINT))); // like before, if TWINT is not set,
get stuck because it means we are not done
	twst = TW_STATUS & 0xF8;
	if( twst != TW_MT_DATA_ACK) return 1;
	return 0;
}

unsigned char twi_read_ack(void) {
	TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA); // interrupt flag, we are
done - enable again - setting to one generates an ack pulse on twi bus
	while(!(TWCR & (1<<TWINT))); // wait for interrupt!
	return TWDR; // return the data on twi bus (is this the ACK?)
}

unsigned char twi_read_nack(void) {
	TWCR = (1<<TWINT) | (1<<TWEN); // interrupt flag, done - enable
again
	while(!(TWCR & (1<<TWINT))); // wait for interrupt to be set
```

```
	return TWDR; // have not generated ack, maybe this is empty now?
}

uint8_t read_specific_address(uint8_t address) {
	uint8_t data;
	twi_start(SLA_W);
	twi_write(address);
	twi_start(SLA_R);
	data = twi_read_nack();
	twi_stop();
	return data;
}
```

7.1.11 rtc.h

```
/*
 * rtc.h
 *
 * Created: 2022-05-11 11:17:25
 *  Author: ax6446st-s
 */


#ifndef RTC_H_
#define RTC_H_

#include <avr/io.h>
#include "i2c.h"

#define HOUR_24		0
#define HOUR_12		1
#define SEC_REG		0x00
#define MIN_REG		0x01
#define HOUR_REG 0x02


typedef struct {
	uint8_t sec;
	uint8_t min;
	uint8_t hour;
} rtc_time_t;

void rtc_read_clock(rtc_time_t *);
void rtc_start_clock(void);
void rtc_set_clock_format(uint8_t format);
uint8_t serialize_seconds(int raw_seconds);
uint8_t serialize_minutes(int raw_minutes);
```

```
uint8_t serialize_hours(int raw_hours);



#endif /* RTC_H_ */
```

7.1.12 rtc.c

```
/*
 * rtc.c
 *
 * Created: 2022-05-11 11:11:52
 *  Author: ax6446st-s
 */

#include "rtc.h"
volatile int set_s = 00;
volatile int set_min = 40;
volatile int set_h  = 12;

void rtc_set_clock_format(uint8_t format) {
      uint8_t currentData = read_specific_address(HOUR_REG);
      twi_start(SLA_W);
      twi_write(HOUR_REG);
      uint8_t dataToSend = currentData;
      if (format) {
            // 12 hour
            dataToSend |= (HOUR_12 << 6);
            } else {
            // 24 hour
            dataToSend |= (HOUR_24 << 6);
      }
      twi_write(dataToSend);
      twi_stop();
}

void rtc_start_clock(void) {
      twi_start(SLA_W);
      twi_write(0x00);
      twi_write(((set_s /10) << 4)| (set_s % 10) |(1 << 7));
      twi_write((set_min / 10 << 4) | (set_min % 10));
      twi_write((set_h / 10 << 4) |(set_h % 10));
      twi_stop();
}
```

```
void rtc_read_clock(rtc_time_t *time) {
	int raw_seconds;
	int raw_minutes;
	int raw_hours;
	twi_start(SLA_W);
	twi_write(0x00);
	twi_start(SLA_R);
	raw_seconds = twi_read_ack();
	raw_minutes = twi_read_ack();
	raw_hours = twi_read_nack();
	time->sec = serialize_seconds(raw_seconds);
	time->min = serialize_minutes(raw_minutes);
	time->hour = serialize_hours(raw_hours);
	twi_stop();
}

uint8_t serialize_seconds(int raw_seconds) {
	uint8_t ones = (uint8_t) (raw_seconds & 0b00001111);
	uint8_t tens = (uint8_t) (raw_seconds & 0b01110000);
	tens = (tens >> 4);
	return ones + tens*10;
}

uint8_t serialize_minutes(int raw_minutes) {
	uint8_t ones = (uint8_t) (raw_minutes & 0b00001111);
	uint8_t tens = (uint8_t) (raw_minutes & 0b01110000);
	tens = (tens >> 4);
	return ones + tens*10;
}

uint8_t serialize_hours(int raw_hours) {
	uint8_t ones = (uint8_t) (raw_hours & 0b00001111);
	uint8_t tens = (uint8_t) (raw_hours & 0b00110000);
	tens = (tens >> 4);
	return ones + tens*10;
}
```

7.1.13 uart.h

```
/*
 * uart.h
 *
 * Created: 2022-05-13 15:47:22
 *  Author: ax6446st-s
 */
```

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

void uart_init_oled();
void Uart_Tchar(unsigned char x);
void Uart_Tstr(unsigned char *msg, int len);
void display_move_cursor(uint16_t row, uint16_t col);
void display_clear_screen();

#endif /* UART_H_ */
```

7.1.14 uart.c

```
/*
 * uart.c
 *
 * Created: 2022-05-13 15:47:11
 *  Author: ax6446st-s
 */
#include "uart.h"

void uart_init_oled()
{
	//UCSRA |= (1<<U2X); //select double speed
	UCSRB = (1<<RXEN)|(1<<TXEN)|(1<<RXCIE); // Receiver / transmitter
enable, receiver interrupt enable
	UCSRC |= (1<<URSEL)|(3<<UCSZ0); //8bit data leng,
	UCSRC &= ~(3<<UPM0); // parity none
	UCSRC |= (1<<USBS); // 1 stop bit
	UCSRC &= ~(1<<UMSEL); // asynchronous mode
	UBRRH = 0x00;	// baud rate
	UBRRL = 103;	// baud rate

	unsigned char screen_saver[] = {0x00, 0x0C, 0x00, 0x00};
	unsigned char text_width[] = {0xFF, 0x7C, 0x00, 0x02};
	unsigned char text_height[] = {0xFF, 0x7B, 0x00, 0x02};

	Uart_Tstr(screen_saver, sizeof(screen_saver)); //Settings for screen
	Uart_Tstr(text_width, sizeof(text_width));	//Settings for
screen
	Uart_Tstr(text_height, sizeof(text_height));	//Settings for screen

}

void Uart_Tchar(unsigned char x)
```

```
{
      UDR = x;
      while(!(UCSRA & (1 << UDRE)));     // TXC
}

void Uart_Tstr(unsigned char *msg, int len)
{
      for (int i = 0; i < len; i++){
            Uart_Tchar(msg[i]);
      }
}

void display_move_cursor(uint16_t row, uint16_t col)
{
      unsigned char cursor[] = {0xFF, 0xE4,
            (unsigned char) row >> 8, (unsigned char) row,
      (unsigned char) col >> 8, (unsigned char) col};
      Uart_Tstr(cursor, sizeof(cursor));
}
void display_clear_screen()
{
      unsigned char clear_screen[] = {0xFF, 0xCE, 0x00, 0x00, 0x00, 0x00,
0x00, 0x7C, 0x00, 0x7C, 0x00, 0x00};
      Uart_Tstr(clear_screen, sizeof(clear_screen));
}
```