EITA15 Projekt, Rapport Grupp 12
Kurs: EITA15 Digitala System

Martin Lysén My Leifsdotter Marko Lauridsen Marcus Thomasson
Daniel Isaksson

Period: 2023-01-27 till 2023-06-09
Labbhandledare: Bertil Lindvall

Abstract

It is of paramount importance that engineer students understand the challenges involved in in designing and producing a product, during the course of this project this group has put together the framework for a product that can be used as an alarm clock that can activate household items that rely on a switch to become active. in the end, this project was not finalized but instead works as a clock and calendar that accurately will keep track of time and date while compensating for months not 30 days long, and leap years.

Innehåll

1 Introduktion

Denna rapport beskriver projektet inom ramen för kursen digitala system (EITA15) på Lunds Tekniska Högskola. Arbetet utfördes under kursen senare halva under vårterminen 2023. I detta dokument kommer detaljeras arbetsgången för grupp 12 projekt det Dumma Smarta HemmetDTH en digital väckarklocka driven av en åtta bitars ATmega1284p-processor om beräkningsenhet. Arbetet utfördes på Helsingborgs Campus. Syftet med detta projekt är att fördjupa elevernas förmåga att planera, avgränsa och utföra ett praktiskt arbete för att ta fram en fungerande prototyp.

Rapportens upplägg beskrivs nedan. Först presenteras avsnittet **Metod** (s.3), som beskriver arbetsgången från idé till konkreta kravspecifikationer. Därefter presenteras avsnittet **Produkt** (s.4), där de olika komponenterna som använts i arbetet beskrivs. Avsnitten **Utförande** (s.6) och **Diskussion** (s.6) beskriver sedan arbetsmomenten mer detaljerat, följt av en analys av resultaten i Diskussionsavsnittet.

2 Metod

I detta avsnitt beskrivs de olika arbetsmomenten som genomfördes i projektet. Arbetet inleddes med en diskussion om hur prototypen skulle fungera. Utifrån diskussioner kunde en kravspecifikation tas fram som konkretiserade arbetsplanen. Hårdvaran planerades först genom ett blockschema, där relationen mellan komponenterna ritades upp. Blockschemat kunde sedan utvecklas till ett kopplingsschema baserat på komponenternas datablad.

Det praktiska arbetet bestod i att konstruera hårdvaran utifrån kopplingsschemat. Komponenterna testades löpande under konstruktionen för att säkerställa att de var korrekt inkopplade. När komponenterna var inkopplade kunde mjukvaran utvecklas. Rapporten skrevs parallellt med arbetet och fylldes på och reviderades kontinuerligt. Projektet avslutades med en muntlig presentation och en praktisk demonstration av resultatet.

2.1 Kravspecifikation

- ska hålla koll på tid och dag, åtminstone en sjudagars-cykel.
- ska ha olika larmtider för vardag och helgdag (åtminstone lördag och söndag).
- ska inkludera en reservkondensator för att skydda mot strömavbrott
- ska helst hålla koll på sommartid och vintertid
- ska slå på ström till kaffekokare
- ska stänga av strömmen till kaffekokaren efter inställt antal minuter.
- ska läsa ljusnivå av en lampa

- ska slå på lampor om det är dags att kliva upp och ljusnivån är för låg
- ska helst anpassa ljusstyrkan till lämplig nivå
- ska slå av lampor om tiden för läggdags har inträffat
- ska visa upp datum, tid, och inställningar på LCD
- ska inkludera knappar för att navigera information och meny
- ska inkludera en funktion för att manuellt starta, släcka kopplade lampor.

3 Produkt

3.1 ATmega1284P

ATmega1284P är en 8-bitars AVR RISC-baserad mikrokontroller som är utrustad med 128 KB programmerbart flashminne, 16 KB SRAM och 4 KB EEPROM. Den har en 12-kanals 10-bitars A/D-omvandlare, två UART-enheter, en JTAG-gränssnitt och flera timer/räknare. ATmega1284P fungerar med en spänningsförsörjning mellan 1,8 V och 5,5 V och kan uppnå en maximal klockfrekvens på 20 MHz. Den är lämplig för en rad olika tillämpningar som kräver högre prestanda och mer minne än vad som finns tillgängligt på mindre mikrokontroller[1].

3.2 LCD

BC2004A-serien är en LCD-skärm med bakgrundsbelysning som är utformad för att visa tecken i fyra rader med 20 tecken per rad. Skärmen drivs av en HD44780-kompatibel styrkrets anslutas till en mikrokontroller via parallellgränssnitt. BC2004A-serien har en inbyggd teckengenerator som stöder en uppsättning av 208 tecken och kan visa specialtecken. Den drivs av en spänning på 5 V och har en strömförbrukning på upp till 2,2 mA vid normal användning.[2].

3.3 Keymapper

MM74C923 är en integrerad krets som används som en tangentbordsavkodare. Kretsen använder en extern oscillator och en räknare för att avkoda vilken tangent som har tryckts ned. MM74C923 ger en BCD-utgång som representerar den avlästa tangenten. Kretsen har även möjlighet att skanna tangentbordet för att upptäcka en tangenttryckning. MM74C923 drivs av en spänning på 5 V och är lämplig för olika tillämpningar som kräver enkel inmatning av användarinput genom ett tangentbord[3].

3.4 Fotosensor

PD202B är en fotodiod som används för att omvandla ljus till en elektrisk signal. Den fungerar genom att det ljus som träffar fotodioden absorberas av materialet och genererar fria laddningsbärare som skapar en ström genom dioden[4].

3.5 RTC, Real Time Clock

DS1307 är en realtidsklocka (RTC) med seriellt gränssnitt. Den har en 56-byte icke-flyktigt minne för att lagra datum och tid som kan kopplas till en kondensator för att kunna lagra information även vid strömavbrott. kretsen är designad för för att kopplas till en extern klockkristall. Komponentens främsta fördel är att den kompenserar för dagar med mindre än 31 dagar automatiskt samt korrigerar för skott år[5].

4 Utförande

Implementering utfördes i steg, först så spolades de olika komponenterna och kopplades till Vcc och dess Avsedda pinnar på processorn. När alla komponenter fästs på brädet så påbörjade arbetet med koden där tangentbordsavkodaren var först ut med att programmeras, för att testa att den implementerats korrekt så testades den mot en LED-Toggle för att se att alla knapparna kunde användas. Efter att funktionalitet bekräftats kunde keymapern kodas. LCD kunde kollas med hjälp av exempelkod som gick igenom startsekvensen och sedan skriva ut ett tecken. Ett standard template användes för att implementera RTC varefter den kunde testas genom att skriva till RTC interna register och sedan validera överföringen genom att jämföra de sända värdena mot de mottagna värdena [6].

Som sista komponentimplementering skulle fotodioden ha testats genom att mäta hur spänningen in till en pin på processorn förändrades som en funktion av ljusintensiteten på komponenten. Som utförandets sista steg skulle casekod implementerats, för att kontrollera överföring a tid till LCD och konfigurering av larmtider med hjälp av knappsatsen.

5 Diskussion

Grunddesignen är mångsidig och har flertalet applikationer för användaren som vill ha en samlingspunkt för kontroll av elektronik i hemmet utan att förlita sig på proprietär hårdvara och mjukvara. Även om prototypen ej kunde färdigställas inom ramen för projektet så kan den med tiden komma att leva upp till sitt namn Det dumma smarta hemmet. I nuläget så kan prototypen endast hålla koll på tid och datum och visa upp dessa på LCD i realtid.

5.1 Problem

Detta projekt var till stor del beroende av att RTC kretsen kunde kommunicera med processorn för att hålla koll på tid och styra larmfunktionen. En full implementering av programmet skulle kräva avbrotts-rutiner för att förhindra att processorn hela tiden begär tid från RTC:n och orsakar en flaskhals. I samma ven så måste avbrotsrutin också implmenteras för att kunna reglera vilket tillstånd displayen skall vara.

En möjlig förbättring till framtida arbetslag som skulle möjliggöra färdigställande av ett projekt av denna typ är att implementera en krets åt gången, i detta fall skulle den första att implementera med fördel varit RTC som då skulle kunnat kontrollerar med debug och en ledlampa. detta skulle troligtvis lett till effektivare felsökning där färre element skulle lett till en klarare problembild.

Referenser

[1] Atmel Corporation. *ATmega1284P 8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash*. Datasheet. [Online; accessed 5-May-2023]. 2013. URL: https://www.microchip.com/wwwproducts/en/ATmega1284P.

[2] Newhaven Display International. *BC2004A Series Datasheet*. https://www.newhavendisplay.com/specs/BC2004A.pdf. [Online; accessed May 6, 2023]. 2018.

[3] National Semiconductor. *MM74C923 Datasheet*. https://www.onsemi.com/pdf/datasheet/mm74c923-d.pdf. [Online; accessed May 7, 2023]. 1997.

[4] Vishay Semiconductors. *PD202B Datasheet*. https://www.tme.eu/Document/74a526e8c3fed872e12f5PD202B.pdf. Accessed: 2023-05-08. 2006.

[5] Maxim Integrated Products. *DS1307 64 x 8, Serial, I2C Real-Time Clock*. https://datasheets.maximintegrated.com/en/ds/DS1307.pdf. Accessed: May 9, 2023. 2011.

[6] Peter Fleury. *Peter Fleury's AVR-GCC Libraries - I2C Master Library*. http://www.peterfleury.epizy.com/doxygen/avr-gcc-libraries/group__pfleury__ic2master.html. Accessed: May 10, 2023.

A Appendix

A.1 Source Code

```
#ifndef KEYMAPPER_H_
#define KEYMAPPER_H_


#include <avr/io.h> // Necessary libraries to include
#include <util/delay.h>

#define Btn_0 0b0000 // Creates all the buttons.
#define Btn_1 0b0100
#define Btn_2 0b1000
#define Btn_3 0b1100
#define Btn_4 0b0001
#define Btn_5 0b0101
#define Btn_6 0b1001
#define Btn_7 0b1101
#define Btn_8 0b0010
#define Btn_9 0b0110
#define Btn_A 0b1010
#define Btn_B 0b1110
#define Btn_C 0b0011
#define Btn_D 0b0111
#define Btn_E 0b1011
#define Btn_F 0b1111

void Keymapper_init(){

   DDRA &= 0b11100001; // Sets in/out for port A
   DDRD |= 0b00000010; // Sets in/out for port D
}

int getForBtn(int btn, int andor){ // Converts port D byte to appropriate don't
    care for buttons
   int first3bits = 0b111 * andor; // andor: 0 = or, 1 = and
   int lastbit = 0b1 * andor;
   int result = (first3bits<<5) | (btn<<1) | lastbit;
   return result;

}
```

```
int getButton(){ // Function for waiting until button is pressed.
   // Runs until a button is pressed, at which point it returns the key pressed as
       hex.
   while(1){
      if(PIND & (1<<2)){
         PIND = PIND | 0b00000010;

         if(PINA == (PINA | getForBtn(Btn_0, 0)) && PINA == (PINA & getForBtn(Btn_0,
             1))){
            //Button 0
            return 0;
         }
         if(PINA == (PINA | getForBtn(Btn_1, 0)) && PINA == (PINA & getForBtn(Btn_1,
             1))){
            //Button 1
            return 1;
         }
         if(PINA == (PINA | getForBtn(Btn_2, 0)) && PINA == (PINA & getForBtn(Btn_2,
             1))){
            //Button 2
            return 2;
         }
         if(PINA == (PINA | getForBtn(Btn_3, 0)) && PINA == (PINA & getForBtn(Btn_3,
             1))){
            //Button 3
            return 3;
         }
         if(PINA == (PINA | getForBtn(Btn_4, 0)) && PINA == (PINA & getForBtn(Btn_4,
             1))){
            //Button 4
            return 4;
         }
         if(PINA == (PINA | getForBtn(Btn_5, 0)) && PINA == (PINA & getForBtn(Btn_5,
             1))){
            //Button 5
            return 5;
         }
         if(PINA == (PINA | getForBtn(Btn_6, 0)) && PINA == (PINA & getForBtn(Btn_6,
             1))){
            //Button 6
            return 6;
         }
```

```
if(PINA == (PINA | getForBtn(Btn_7, 0)) && PINA == (PINA & getForBtn(Btn_7,
    1))){
   //Button 7
   return 7;
}
if(PINA == (PINA | getForBtn(Btn_8, 0)) && PINA == (PINA & getForBtn(Btn_8,
    1))){
   //Button 8
   return 8;
}
if(PINA == (PINA | getForBtn(Btn_9, 0)) && PINA == (PINA & getForBtn(Btn_9,
    1))){
   //Button 9
   return 9;
}
if(PINA == (PINA | getForBtn(Btn_A, 0)) && PINA == (PINA & getForBtn(Btn_A,
    1))){
   //Button A
   return 10;
}
if(PINA == (PINA | getForBtn(Btn_B, 0)) && PINA == (PINA & getForBtn(Btn_B,
    1))){
   //Button B
   return 11;
}
if(PINA == (PINA | getForBtn(Btn_C, 0)) && PINA == (PINA & getForBtn(Btn_C,
    1))){
   //Button C
   return 12;
}
if(PINA == (PINA | getForBtn(Btn_D, 0)) && PINA == (PINA & getForBtn(Btn_D,
    1))){
   //Button D
   return 13;
}
if(PINA == (PINA | getForBtn(Btn_E, 0)) && PINA == (PINA & getForBtn(Btn_E,
    1))){
   //Button E
   return 14;
}
if(PINA == (PINA | getForBtn(Btn_F, 0)) && PINA == (PINA & getForBtn(Btn_F,
    1))){
   //Button F
```

```
            return 15;
        }
      }
   }
}


#endif /* KEYMAPPER_H_ */


#ifndef LCD_H_
#define LCD_H_


#include <avr/io.h>
#include <util/delay.h>
#include "RTC.h"
/* Wiring dependent definitions */
#define LCDcontrolPort PORTA // Output Port Register
#define DDRcontrolPort DDRA // Data Direction Register
#define E 7 // Enable Signal
#define RW 6 // Read Write Select Signal
#define RS 5 // Register Select Signal
#define LCDdataPort PORTB  // Output Port Register
#define DDRdataPort DDRB  // Data Direction Register

/* Values defined for the LCD BC2004A-series */
#define FUNCTIONSET 0x38  // Functionset 00111000 => (8-bit 2-line)
#define DISPLAY_ONOFF 0x0F // Display ONOFF 00001111 => cursor blinking
#define CLEAR_DISPLAY 0x01  // Clear LCD display
#define ExecutionTimeCmd 80 // Delay needed for writing command to LCD
#define ExecutionTimeWrite 50 // Delay needed for writing character to display

#define SECONDS_REGISTER 0x00
#define MINUTES_REGISTER 0x01
#define HOURS_REGISTER 0x02
#define DAYOFWK_REGISTER 0x03
#define DAYS_REGISTER 0x04
#define MONTHS_REGISTER 0x05
#define YEARS_REGISTER 0x06

char sendNumber(uint8_t nbr);
```

```
void write_lcd_cmd(unsigned char command); // sends instruction to LCD
void write_on_lcd(unsigned char character); // prints character on LCD screen
void sendString(char *string_of_characters); // prints string on LCD screen

void lcd_init() {
   DDRcontrolPort |= 1<<E | 1<<RW | 1<<RS; // E RW RS set as output
   DDRdataPort = 0xFF; // data port set as output
   LCDcontrolPort |= 1<<E; // E=1
   LCDcontrolPort &= ~(1<<RW|1<<RS); // RW=0 RS=0
   write_lcd_cmd(FUNCTIONSET); // Functionset, 00111000 (8-bit 2-line)
   write_lcd_cmd(DISPLAY_ONOFF); // Display ON/OFF 00001111 blinking cursor
}

void write_lcd_cmd(unsigned char command){
   LCDcontrolPort |= 1<<E; // Enable 1
   LCDcontrolPort &= ~(1<<RW|1<<RS); // RW=0 RS=0
   LCDdataPort = command; // command to port
   LCDcontrolPort &= ~(1<<E); // Enable 0
   LCDcontrolPort |= 1<<E; // Enable 1
   _delay_ms(ExecutionTimeCmd); // Dependent on execution times
}

void write_on_lcd(unsigned char character){
   LCDcontrolPort |= 1<<E | 1<<RS; // E=1 RS=1
   LCDcontrolPort &= ~(1<<RW); // RW=0
   LCDdataPort = character; // character to port
   LCDcontrolPort &= ~(1<<E); // Enable 0
   LCDcontrolPort |= 1<<E; // Enable 1
   _delay_ms(ExecutionTimeWrite); // Dependent on execution times
}

void sendString(char *string_of_characters){
   while(*string_of_characters>0){
      write_on_lcd(*string_of_characters++);
      //_delay_ms(100); // Just to make it look nice
   }
}

void WriteTime()
{
 uint8_t hours, minutes, seconds;
 DS1307_GetTime(&hours,&minutes,&seconds);
 TwoDigits(hours);
```

```
 write_on_lcd(':');
 TwoDigits(minutes);
 write_on_lcd(':');
 TwoDigits(seconds);
}

void DS1307_GetDate(uint8_t *months, uint8_t *days, uint8_t *years)
// returns months, days, and years in BCD format
{
 *months = rtc_read_register(MONTHS_REGISTER);
 *days = rtc_read_register(DAYS_REGISTER);
 *years = rtc_read_register(YEARS_REGISTER);
}

void WriteDate()
{
 uint8_t months, days, years;
 DS1307_GetDate(&months,&days,&years);
 TwoDigits(months);
 write_on_lcd('/');
 TwoDigits(days);
 write_on_lcd('/');
 TwoDigits(years);
}

void DS1307_GetTime(uint8_t *hours, uint8_t *minutes, uint8_t *seconds)
// returns hours, minutes, and seconds in BCD format
{
 *hours = rtc_read_register(HOURS_REGISTER);
 *minutes = rtc_read_register(MINUTES_REGISTER);
 *seconds = rtc_read_register(SECONDS_REGISTER);
 if (*hours & 0x40) // 12hr mode:
 *hours &= 0x1F; // use bottom 5 bits (pm bit = temp & 0x20)
 else *hours &= 0x3F; // 24hr mode: use bottom 6 bits
}

void TwoDigits(uint8_t data)
// helper function for WriteDate() and WriteTime()
// input is two digits in BCD format
// output is to LCD display at current cursor position
{
 uint8_t temp = data>>4; // get upper 4 bits
 write_on_lcd(sendNumber(temp)); // display upper digit
```

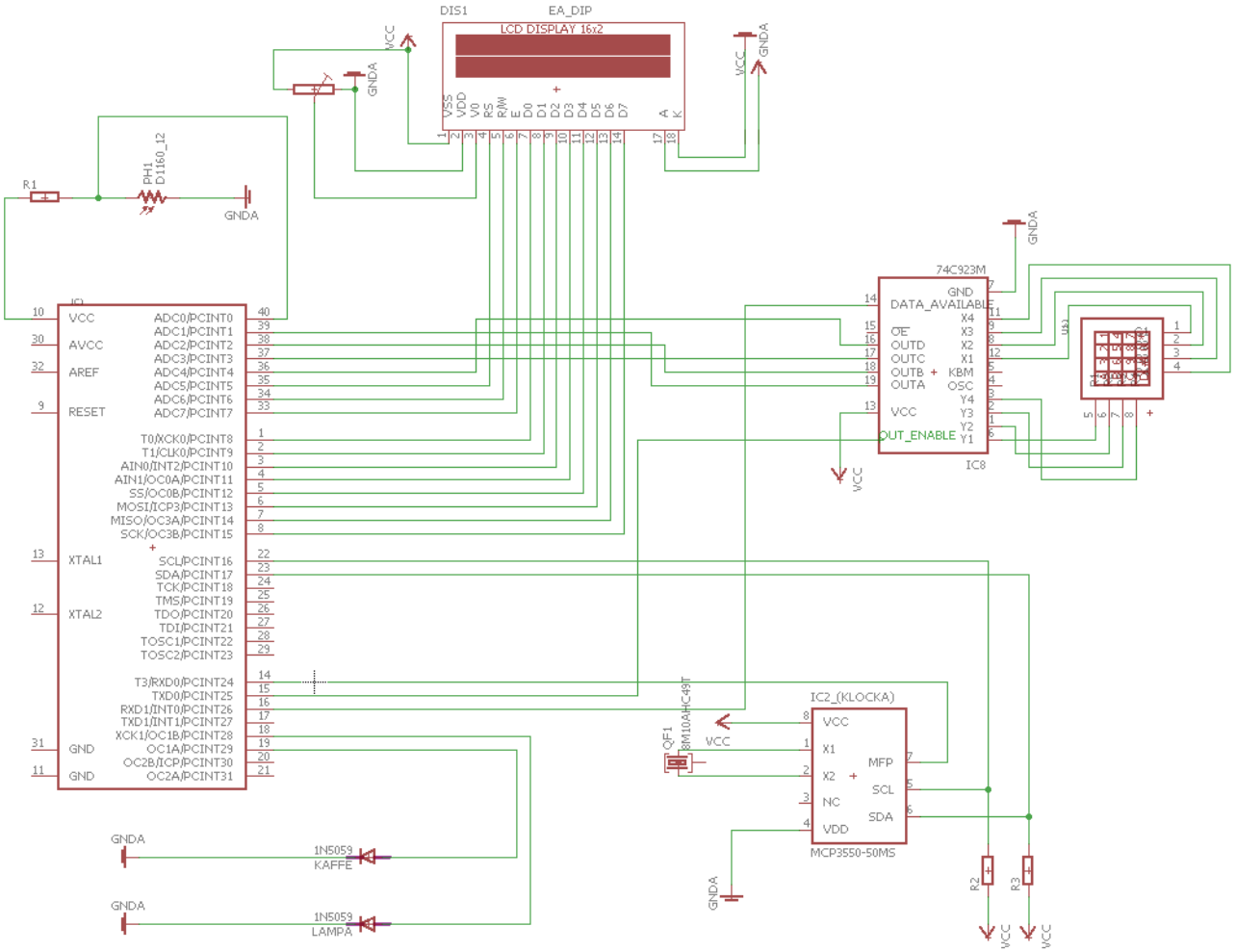
```
 data &= 0x0F; // get lower 4 bits
 write_on_lcd(sendNumber(data)); // display lower digit
}

//convert int to string
char sendNumber(uint8_t nbr){
   switch(nbr){
      case 0:
      return('0');
      break;
      case 1:
      return('1');
      break;
      case 2:
      return('2');
      break;
      case 3:
      return('3');
      break;
      case 4:
      return('4');
      break;
      case 5:
      return('5');
      break;
      case 6:
      return('6');
      break;
      case 7:
      return('7');
      break;
      case 8:
      return('8');
      break;
      case 9:
      return('9');
      break;
   }
}

#endif //LCD_H

#ifndef RTC_H_
```

```
#define RTC_H_

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#define DS1307_ADDRESS 0xD0 // Address of the RTC on the I2C bus
#define SECONDS_REGISTER 0x00
#define MINUTES_REGISTER 0x01
#define HOURS_REGISTER 0x02
#define DAYOFWK_REGISTER 0x03
#define DAYS_REGISTER 0x04
#define MONTHS_REGISTER 0x05
#define YEARS_REGISTER 0x06
void i2c_init(void)
{
   TWSR = 0x00; // Set prescaler to 1
   TWBR = ((F_CPU / 100000UL) - 16) / 2; // Set I2C frequency to 100kHz
}

void i2c_start(void)
{
   TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN); // Send start condition
   while (!(TWCR & (1 << TWINT))); // Wait for TWINT flag to be set
}

void i2c_stop(void)
{
   TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN); // Send stop condition
   while ((TWCR & (1 << TWSTO))); // Wait for TWSTO flag to be cleared
}

void i2c_write(uint8_t data)
{
   TWDR = data; // Load data into data register
   TWCR = (1 << TWINT) | (1 << TWEN); // Start transmission
   while (!(TWCR & (1 << TWINT))); // Wait for TWINT flag to be set
}

uint8_t i2c_read_ack(void)
{
   TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWEA); // Enable acknowledge bit
   while (!(TWCR & (1 << TWINT))); // Wait for TWINT flag to be set
```

```
   return TWDR; // Return data received
}

uint8_t i2c_read_nack(void)
{
   TWCR = (1 << TWINT) | (1 << TWEN); // Disable acknowledge bit
   while (!(TWCR & (1 << TWINT))); // Wait for TWINT flag to be set
   return TWDR; // Return data received
}

void SetTimeDate()
// simple, hard-coded way to set the date 8/13/21013 at 8:51 PM
{
   I2C_WriteRegister(MONTHS_REGISTER, 0x08);
   I2C_WriteRegister(DAYS_REGISTER, 0x31);
   I2C_WriteRegister(YEARS_REGISTER, 0x13);
   I2C_WriteRegister(HOURS_REGISTER, 0x08+0x40); // add 0x40 for PM
   I2C_WriteRegister(MINUTES_REGISTER, 0x00);
   I2C_WriteRegister(SECONDS_REGISTER, 0x00);
}
void rtc_init(void)
{
   i2c_start();
   i2c_write(DS1307_ADDRESS);
   i2c_write(0x00); // Set control register to 0
   i2c_write(0x00); // Enable oscillator and set square-wave output to 1Hz
   i2c_stop();
}
void I2C_WriteRegister(uint8_t deviceRegister, uint8_t data)
{
   i2c_start();
   i2c_write(DS1307_ADDRESS); // send bus address
   i2c_write(deviceRegister); // first byte = device register address
   i2c_write(data); // second byte = data for device register
   i2c_stop();
}
uint8_t rtc_read_register(uint8_t reg)
{
   uint8_t value;

   i2c_start();
   i2c_write(DS1307_ADDRESS);
   i2c_write(reg); // Set register pointer
```

```
    i2c_start(); // Send repeated start condition
    i2c_write(DS1307_ADDRESS | 0x01); // Set RTC address with read bit
    value = i2c_read_nack(); // Read value from register
    i2c_stop();

    return value;
}

#endif /* RTC_H_ */

#define F_CPU 8000000L
#define F_SCL 100000L

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "LCD.h"
#include "RTC.h"
#include "Keymapper.h"

int main(void)
{
    i2c_init();
    lcd_init();
    Keymapper_init();
    /* Replace with your application code */
    SetTimeDate();

    sendString("Startup");
    _delay_ms(1000);
    write_lcd_cmd(CLEAR_DISPLAY); // Clear LCD

    while (1)
    {
      write_lcd_cmd(CLEAR_DISPLAY);
      WriteTime();
      write_on_lcd(' ');
      WriteDate();
      _delay_ms(10000);
    }
}
```

A.2 Image



Figur 1: Kretschema det dumma smarta hemmet