Pausevarsler

Prosjektoppgave i ING1507

Kandidatnr. xx

Kandidatnr. yy

1. Introduksjon

1.1. Prosjektbeskrivelse

Ved bruk av ATmega32 mikrokontrollere og en rekke komponenter, skulle vi ut ifra de ulike aspektene vi har lært gjennom kurset *Datamaskinarkitektur* lage en modul/system som passer i et smarthjem.

Innhold

2. Prosjektidé

2.1. Utfordring

På skolen har vi pause fra undervisningen med omtrent 45 minutter intervaller, dette er det en fastsatt plan på når pausene skal komme. Det er kadett kommandør (KK) som har dette ansvaret, jobben er å følge med på klokken og varsle foredragsholder med 5 minutter før en pause starter. Utfordringen for KK er at hen glemmer å følge med på klokken da dette vil ta over for fokuset på det instruktøren sier. Som resulterer i at pausene ikke kommer når de skal, som da igjen gjør at kadettene sliter med å fokusere når de ikke får avbrekk.

2.2. Løsning

Løsningen vi har kommet opp med er å lage et produkt som varsler KK samt foredragsholderen når disse pausene skal komme, og hvor lenge det er til.

2.2.1. Virkemåte

Det er et system bestående av en kk-modul og en lærer-modul. Disse kommuniserer med hverandre over blåtann.

2.2.2. KK-modul

Dette er masteren. Den har en Real Time Clock, som er programmert med en alarm som sender et signal ved skolestart. Mikrokontrolleren har ekstern interrupt på dette signalet og vekkes ved dette signalet. Da går den over i å lese av med et intervall på 1 sekund ved hjelp av Compare Match på Timer1 på ATmega32. Når timen starter sender den lengden på timen over blåtann til lærer-modulen. Når pausen starter, sender den lengden på pausen til lærer-modulen. Når skoledagen er over, sender den stop-kommando til lærer-modulen. All konfiurering av skolestart, pauser og skoleslutt konfigureres altså på KK-modulen. KK-modulen har også en LCD som viser gjenværende minutter og sekunder av pausen når det er pause, og gjenværende tid av timen når det er time.

2.2.3. Lærer-modul

Dette er slaven. Denne har RX-interrupt fra blåtannmodulen. Den sover fram til den får melding over blåtann. Da viser den på 7-segment-skjermen gjenværende tid til time/pause, basert på lengden på pausen/timen den får tilsendt hvor den trekker fra tid ved bruk av den interne klokken i mikrokontrolleren. Når det er pause hever en servo et flagg for å tydeliggjøre at det er pause. Når pausen er over, senkes flagget.

3. Metode

Kobling ble gjort for de ulike komponentene for å kunne opprette funksjonaliteten beskrevet innledningsvis. Deretter ble en og en komponent testet og det ble laget kode for rett funksjonalitet for denne komponenten. Deretter ble disse kodeutsnittene satt sammen i en sammensatt kode for funksjonaliteten på kk-modulen og en sammensatt kode for lærer-modulen. Mye inspirasjon, og informasjon om hvordan flere av komponentene som ble brukt i prosjektet fungerer, ble hentet fra Leksjonene i ING1507 [1], samt databladene til de ulike komponentene.

3.1. Utstyr

3.1.1. KK-modul

- 1x ATmega32 mikrokontroller
- 1x HM-10 blåtannmodul
- 1x DS3231 Real Time Clock
- 1x 16x2 LCD

- 2x koblingsbrett
- Flere ledninger og motstander

3.1.2. Lærer-modul

- 1x ATmega32 mikrokontroller
- 1x HM-10 blåtannmodul
- 1x 7-segment-skjerm
- 1x Servo
- 3x koblingsbrett
- Flere ledninger og motstander

4. Teori

4.1. USART

Blåtannmodulen kommuniserer med en baud-rate på 9600. Dette skaper et nøyaktighetsproblem på ATmega32 uten videre konfigurering. Normal konfigurasjon av USART på ATmega32 med standard klokkehastighet på 1MHz og baud rate på 9600, gir en baud-prescaler med −7% avvik fra en baud-verdi på 9600 (ref. Figur 1). Dette kommer av utregningen på baud-prescaler som gir et så lavt tall at nøyaktigheten blir for dårlig til å nøyaktig kunne nå baudraten på 9600 med en heltalls baud-prescaler. For å overkomme dette dobles USART-klokkehastigheten ved å sette U2X-bit i UCSRA. Da må også utregningen av prescaler dobles. Dette gir et avvik på bare 0.2% på baud-raten, noe som er pålitelig nok til å få lest av data korrekt i svært stor grad.

Table 68. Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 1.0000MHz				f_{osc} = 1.8432MHz				f_{osc} = 2.0000MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max (1)	62.5 Kbps		125 Kbps		115.2 Kbps		230.4 Kbps		125 Kbps		250 Kbps	

1. UBRR = 0, Error = 0.0%

Figur 1: Avvik i baudrate på ATmega32 (ref. ATmega32 datablad [2])

4.2. HM-10 Blåtannmodul

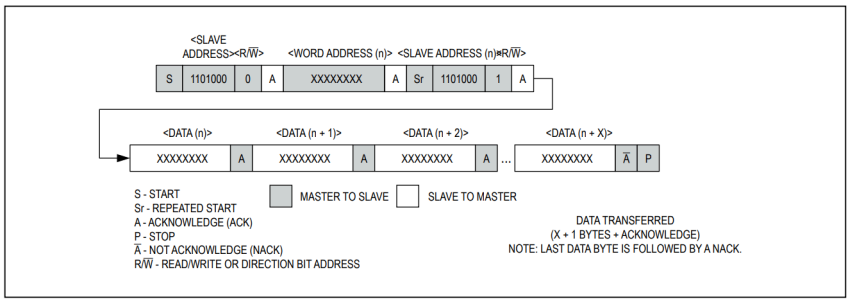
En HM-10 blåtannmodul sender UART data over bluetooth. Den har en intern krets som tar seg alt prosessering ved overføring over bluetooth og fungerer med standardinnstillinger så snart den blir koblet opp til en spenning på 3-6v. Den kan konfigrureres ved hjelp av AT-kommandoer for å endre innstillinger som baud-rate, enhetsnavn, eller om den skal være master eller slave. Det er verdt å merke seg at den interne kretsen kun er 3.3v tolerant. Logikken i ATmega32 er 5v. Dette går fint for signalet fra TX på blåtannmodul til ATmega32, da ATmega32 kan regirstrere et signal på 3.3v. Problematikken er på signalet fra TX på ATmega32 til blåtannmodulen. Den indre kretsen i blåtannmodulen kan skades dersom dette signalet er 5v. Dette løses ved å bruke en spenningsdeler med ulike motstander (se koblingsskjema) på dette signalet fra TX på mikrokontrolleren.

4.3. DS3231 Real Time Clock

DS3231 er en ekstremt nøyaktig klokke som fungerer over I2C. Denne har et minnebatteri, som gjør at klokken fortsetter å gå rett selv om ekstern strøm til enheten kobles fra, dette gjør at den vil kunne vise rett tid i svært lang tid. Enheten har en krystall og en temperatur-kompensert krystall. RTC-en holder styr på sekunder, minutter, timer, ukedag, dato, måned og år. Enheten har også to programmerbare alarmer. For å hente ut tiden fra RTC senders først adressen til RTC hvor least significant bit (LSB) er satt til 0 over I2C fra masteren (ATmega32 er masteren i dette scenarioet). Dette gjør at RTC-en begynner, og er klar til å respondere på kommende meldinger. Deretter sender adressen vi ønsker å lese fra ("word address") dette er adressen i Figur 2 (for eksempel adresse = 03h for å lese av dagen). Deretter sendes repeated start, etterfulgt av adressen til RTC-en hvor LSB er satt til 1. Dette gjør at RTC-en sender byten med data fra denne adressen. Dersom master så sender ACK sendes byten fra neste adresse osv. For å avslutte sendes NACK og Stop. Denne prosessen følger vi i Figur 3, og er implementert i Oppføring 1.

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{AM}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59
09h	A1M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 1 Hours	1–12 + $\overline{AM}$/PM 00–23
0Ah	A1M4	DY/DT	10 Date		Day				Alarm 1 Day	1–7
					Date				Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 2 Hours	1–12 + $\overline{AM}$/PM 00–23
0Dh	A2M4	DY/$\overline{DT}$	10 Date		Day				Alarm 2 Day	1–7
					Date				Alarm 2 Date	1–31
0Eh	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—
0Fh	OSF	0	0	0	EN32kHz	BSY	A2F	A1F	Control/Status	—
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	—
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	MSB of Temp	—
12h	DATA	DATA	0	0	0	0	0	0	LSB of Temp	—

Figur 2: Tidsregistere i DS3231 (ref. DS3231 datablad [3])



Figur 3: Prosess for avlesning DS3231

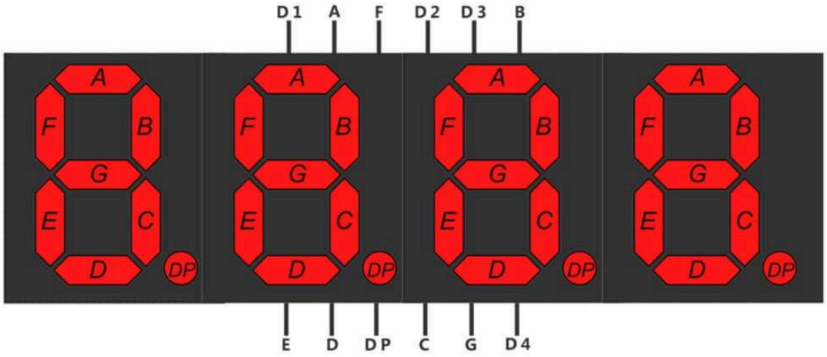
```
void RTC_Read_Clock(char read_clock_address) {
    I2C_Start(RTC_Write_address); // Start I2C communication with RTC
    I2C_Write(read_clock_address); // Write address to read
    I2C_Repeated_Start(RTC_Read_address);

    second = BDC2value(I2C_Read_Ack());
    minute = BDC2value(I2C_Read_Ack());
    hour = BDC2value(I2C_Read_Nack() & 0b00111111); //Last communication so nack.
The two MSB are not relevant
    I2C_Stop();
}
```

Oppføring 1: Kode for avlesning DS3231

4.4. 7-segment-skjerm

En 7-segment-skjerm er satt sammen av 7 led-lys med felles anode eller katode, hver av led-lysene har en egen pin. Hvis det er flere segmenter satt sammen vil det være en felles anode/katode for hvert segment og alle led-lysene for samme posisjon/samme bokstav er koblet sammen (ref. Figur 4). For å vise en bokstav på skjermen må de riktige pinnene settes høye eller lave.



Figur 4: 7-segment-skjerm

5. Kodeprinsipper

5.1. KK-modul

5.1.1. Alarm på DS3231

Utklipp hentet fra databladet til DS3231 [3]

For prosjektet skal ATmega32 vekkes ved alarmen på DS3231 til kl.08.00. Alarmen settes ved å skrive til alarm-registrene (ref. Figur 2). For å få alarmen til å trigge på sekunder, minutter og timer, settes A1M4 i DS3231-register til 1 (ref. Figur 5) For å oppnå at SQW-pinnen på DS3231 settes til GND når alarmen trigges, slås Interrupt Control på ved å sette INTCN til 1. For at den skal trigge på alarm 1, slås også A1IE på. (ref. Figur 6). Alarm-initieringen er implementert i Oppføring 2. Setting av alarm er implementert i Oppføring 3. Når flagget for alarmen blir satt, vekkes ATmega32 og går over i å lese av klokken. Flagget må nullstilles manuelt, og gjøres ved å sette bit-en for A1F i Figur 7 til 0. Dette er implementert i Oppføring 4.

DY/$\overline{DT}$	ALARM 1 REGISTER MASK BITS (BIT 7)				ALARM RATE
	A1M4	A1M3	A1M2	A1M1	
X	1	1	1	1	Alarm once per second
X	1	1	1	0	Alarm when seconds match
X	1	1	0	0	Alarm when minutes and seconds match
X	1	0	0	0	Alarm when hours, minutes, and seconds match
0	0	0	0	0	Alarm when date, hours, minutes, and seconds match
1	0	0	0	0	Alarm when day, hours, minutes, and seconds match

Figur 5: Konfigurering av alarm på DS3231

Control Register (0Eh)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE
POR:	0	0	0	1	1	1	0	0

Figur 6: Register for å konfigurere blant annet alarm på DS3231

```
    void RTC_Alarm_Init(){
        I2C_Start(RTC_Write_address);
        I2C_Write(0xE);   // Bet at the control register position (See datasheet)
        I2C_Write(0b00000101);  // Enabling interrupt Control and Alarm 1
Interrupt Enable
        I2C_Stop();
}
```

Oppføring 2: Kode for å initiere alarm på DS3231

```
void RTC_Alarm1_Time(char _hour, char _minute, char _second){ // The alarm
triggers at the spesified time every day.
    I2C_Start(RTC_Write_address);
    I2C_Write(7);         // Set alarm1 time in register 7
    I2C_Write(value2BDC(_second));
    I2C_Write(value2BDC(_minute));
    I2C_Write(value2BDC(_hour));
    I2C_Write(0b10000000); // Set A1M4 to trigger at that time set
    I2C_Stop();
}
```

Oppføring 3: Kode for å sette alarm1

Status Register (0Fh)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	OSF	0	0	0	EN32kHz	BSY	A2F	A1F
POR:	1	0	0	0	1	X	X	X

Figur 7: Register for å lese av / nullstille flagg

```
void RTC_Alarm_Clear(){
    I2C_Start(RTC_Write_address);
    I2C_Write(0xF);
    I2C_Write(0b10001000);  // Write 0 at the BSY, A2F and A1F flag.
    I2C_Stop();
}
}
```

Oppføring 4: Kode for å nullstille alarm-flagget

6. Diskusjon

6.1. Blåtannmodul

Konfigurering av modulen var utfordrende, da det er vanskelig å vite nøyaktig hvordan modulen opererer, særlig med lite dokumentasjon. Slike blåtannmoduler er også moduler det er finnes utallige kopier av der ulike produsenter har ulike standarder. Dette gjør det spesielt vanskelig å finne hvilken type modul med hvilken konfigurasjon man har, og deretter å finne adekvat dokumentasjon for denne. (Utseende på en HC-06, HC-05 og HM-10-modul er tilnærmet identisk, noe som gjør det vanskelig å identifisere hvilken man har). I vårt tilfelle fant vi med mye leting ut at våres modul var av typen HM-10, de var imidlertid ikke helt etter standard, for flere av AT-kommandoene som var å finne i ufullstendig dokumentasjon på internett returnerte våres blåtann moduler med “Error”. Tross dette var vi i stand til å endre rolle mellom slave og master. I vårt prosjekt ble blåtannmodulen brukt ved KK-modulen satt til å være master, og lærer-modulen slave. Etter det vi kunne finne av dokumentasjon på internett om automatisk oppkobling i slave-master-par fungerte ingen av kommandoene for å sette opp at master-modulen automatisk skulle koble seg opp til slave-modulen ved oppstart. Vi måtte derfor konkludere med at med mangel på rett dokumentasjon for våre blåtannmoduler var ikke automatisk sammenkobling ved oppstart mulig. Dette resulterte i at man ved oppstart må koble master modulen til seriell tilbobling til PC, for å via AT-kommandoer søke etter nære blåtann-moduler og velge rett, for så å koble opp til denne.

6.2. Virkemåte KK-modul

Det er hensiktsmessig at masteren har mest mulig nøyaktig tid, da Lærer-modulen settes etter denne. Derfor ble det brukt intern timer med Compare Match for å lese av klokken med 1 sekunds intervall. Dette gjør at KK-modulen har mest mulig nøyaktig tid, ettersom DS3231-klokken heller ikke har mer nøyaktighet enn 1-sekunds intervall, og mer enn dette er jo heller strengt tatt ikke nødvendig for våres formål. Noen ytteligere optimaliseringer ble også implementert for å kunne spare energi: 1. Alarmen på DS3231 settes til kl.08.00. Slik at når tidspunktet er 08.00 settes SQW-pinnen på DS3231 til GND. KK-modulen har dette signalet som et interrupt-signal, slik at den er i sleep fram til starten av dagen. Deretter leser den av klokken over I2C ved hvert tidsintervall. I2C-kommunikasjonen gjør den med bruk av while mens den sender og venter på ACK, grunnen til dette er at vi er nødt til å ha kontroll på hvor vi er i kommunikasjonsprosessen, da vi først sender adressen vi leser fra, for så å lese av et bestemt antall verdier, der vi får neste verdi ved å sende ACK og må ha kontroll på når vi skal sende NACK og STOP når all klokkedata er mottatt. Ved pause-/timestart senders lengden på pausen eller timen over USART.

7. Feilkilder

USART på ATmega32 ble konfigurert til å bruke dobbel hastighet. Dette gjorde at baud-prescaler fikk en verdi som ga den 0.2% avvik ved baud-rate på 9600. Selv om avviket er lite, er det ikke neglisjerbart. Dette kombinert med eventuelle feil under overføring med blåtann, gjør at det er en viss mulighet for at data som sendes fra KK-modulen blir mottatt feil på Lærer-modulen, og at denne derfor vil vise feil gjenværende tid.

8. Konklusjon

Prosjektet har vært lærerikt, og vi har fått en god forståelse for hvordan de ulike komponentene vi har brukt fungerer. Vi har også fått en god forståelse for hvordan grunnprinsippene i USART, PWM, I2C og interrupt fungerer og hvordan dette kan brukes i praksis. Vi har også fått en god forståelse for hvordan man lager mer sammensatte systemer, og hvilke utfordringer som kommer med dette, noe som har gitt en dypere forståelse for grunnprinsippene ved at problemene har måttet løses i praksis.

Bruk av nye komponenter som DS3231 har også gitt mer innsikt i hvordan datablad kan brukes til å forstå hvordan en komponent fungerer, og hvordan kode må skrives for å kunne bruke denne komponenten.

Referanser

[1] Marshed Kassim Mohammed, “Leksjoner i datamaskinarkitektur.” 2024.

[2] Atmel Corporation, “ATmega32/L Datasheet.” [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/doc2503.pdf

[3] Analog Devices, “DS3231 Extremely Accurate I2C-Integrated RTC/TCXO/Crystal.” [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/DS3231.pdf

9. Vedlegg

9.1. KK_modul

9.1.1. KK_module.c

```
/*
THIS IS THE MASTER
  CONNECTION
  Wire up master to arduino
  type AT+INQ until 0x884AEA4C7E09 shows
  type AT+CONN[index of that adress]
  no connected!

 * Functionallity:
 KK_module:
 - alarm set at 08.00 in the weekdays, when this triggers, then it reads always
from it in while
 - print the time on the lcd when reading
 - is bluetooth master so sends the data to the slave when close to break

 teacher_module:
 - shows the time on the 7 segmet display
 - raise the flag when break
 - lowers the flag when break over
 */

#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <string.h>
#include "screen.h"
#include "I2C.h"

#define RTC_Write_address 0b11010000
#define RTC_Read_address 0b11010001
#include "RTC.h"


long USART_BAUDRATE = 9600;

#include "USART.h"

void timer_init();

int read_continious_clock = 0; // Boolean if clock should be read


int main(void)
{

  GICR |= 1 << INT1; // Enable INT1 (Alarm interrupt)
  MCUCR |= (1 << ISC11);
 MCUCR &= ~(1 << ISC10); //Detect falling edge
```

```
    I2C_Init();
    RTC_Clock_Write(7,59,50); // Set the time
    RTC_Date_Write(1,5,6,24); // Year is the 2 last digits in the year

 RTC_Alarm_Init();
  RTC_Alarm1_Time(7,59,58);   // Wake up 2 second before start of day, to be ready

    LCD_Init();     /* Initialize LCD */
 timer_init();
  USART_Init(USART_BAUDRATE);   // Intitialize USART with spesified baud rate
  LCD_String_xy(0,0,"Init");
 sei();

    while (1) {
    }

}

#define NUMBER_OF_BREAKS 5
int break_times[NUMBER_OF_BREAKS][3] = {
 {8, 0, 0},        // Start of day, (have breaklength of 0)
 {8, 1, 1},        // Break {starthour, startminute, length}
  {8, 3, 2},
 {11, 0, 45},
 {15, 30, -1}        // End of day, (have breaklength of -1)
};


// When alarm the SQW pin on the board gets high, detect this using interrulpt
// When the alarm has been triggered, and interrupted the program we need to set
the flag to 0 again
// The signal from DS3231 gets low when alarm
ISR(INT1_vect){
  LCD_String_xy(0,0,"Alarm");
  RTC_Read_Date(3);   // Read the date
  if (weekday >= 6){    // Weekday >= 6 is saturday and sunday. go back to
sleep ;=)
    read_continious_clock = 0;
  } else {
    read_continious_clock = 1; // Start to read form the clock
    TIMSK |= 1 << OCIE1A; // Enable Output Compare A Match Interrupt
  }
}


volatile int message_sent = 0; // Boolean to keep track of if we have transmitted
the message with either break- or lecture time.
volatile int period_length = 0;
volatile int period_end_minutes;
// Trigger at set interval
ISR(TIMER1_COMPA_vect){
  if (read_continious_clock){
    RTC_Read_Clock(0);
```

```
    int minutes_since_midnight = hour*60 + minute;
    char buffer[20];
    sprintf(buffer, "%02d:%02d:%02d", hour, minute, second);
   LCD_String_xy(0,0,buffer);

    char message[5];

    for (int i = 0; i < NUMBER_OF_BREAKS; i++){
      if ((minutes_since_midnight == break_times[i][0]*60 + break_times[i][1]) &&
break_times[i][2] != 0){ // If the time now is equal to the time the break starts.
And it is not the start of the day (break length = 0)
        if (break_times[i][2] == -1){   // End of day has duration = -1
          // End of day, go back to sleep
          TIMSK &= ~(1 << OCIE1A); // Disable Output Compare A Match Interrupt
          RTC_Alarm_Clear();
          read_continious_clock = 0;
          break;
        }

        if (message_sent == 1){         // This is reversed, as opposed to when
the break is over, this is since after we send break length it should send lecture
length
          period_length = break_times[i][2];
          period_end_minutes = minutes_since_midnight + period_length;
          sprintf(message, "b%d", period_length);   // Send the message of current
breaklength, identified by a "b" at index 0 in the string.
          USART_Transmit_String(message);

          LCD_Clear();
          LCD_String_xy(0,0,buffer);    // To print the current time immedeately
after the clear
          LCD_String_xy(1,7, "PAUSE");
          message_sent = 0;
        }
      } else if (minutes_since_midnight == break_times[i][0]*60 + break_times[i]
[1] + break_times[i][2]){ // The time now is the start of the break time + the
break length, so the break is just over
        if (message_sent == 0){
          int next_period_start_minute = break_times[i+1][0]*60 + break_times[i+1]
[1];
          period_length = next_period_start_minute - minutes_since_midnight;
          period_end_minutes = minutes_since_midnight + period_length;
          sprintf(message, "l%d", period_length);   // Send the message of current
breaklength, identified by a "b" at index 0 in the string.
          USART_Transmit_String(message);

          LCD_Clear();
          LCD_String_xy(0,0,buffer);
          LCD_String_xy(1,7, "TIL PAUSE");
          message_sent = 1;
        }
      }
    }

    char message_left[20];
```

```
    int total_seconds_left = (period_end_minutes * 60 + 59) -
((minutes_since_midnight+1) * 60 + second);
    int minutes_left = total_seconds_left / 60;
    int seconds_left = total_seconds_left % 60;

    if (minutes_left >= 100){
      sprintf(message_left, "%03d:%02d", minutes_left, seconds_left);
    } else{
      sprintf(message_left, "%02d:%02d", minutes_left, seconds_left);
    }
    LCD_String_xy(1,0, message_left);
  }
}

void timer_init(){
  TCCR1B |= (1<<CS11) | (1<<CS10); // 64 prescaler (Use a low prescaler to make
the Count more accurate
  TCCR1B |= 1<<WGM12;       // CTC (compare output mode)

  // Count which is equivalent to 1 sec:
  // 1 * F_CPU/prescaler
  uint16_t Count = 15625;
  OCR1A = Count;    // Put value in Output Compare Register
  TIMSK &= ~(1 << OCIE1A); // Disable Output Compare A Match Interrupt
}
```

9.1.2. I2C.h

```
/* This code is modified from https://www.electronicwings.com/avr-atmega/atmega
1632-i2c */

/* Define bit rate */
#define BITRATE(TWSR) ((F_CPU/SCL_CLK)-16)/(2*pow(4,(TWSR&((1<<TWPS0)|
(1<<TWPS1)))))

void I2C_Init()     /* I2C initialize function */
{
  //TWBR = BITRATE(TWSR=0x00);  /* Get bit rate register value by formula */
 TWBR = 2;
  PORTC |= (1<<PC0)|(1<<PC1);   // Set pull up resistor on SCL and SDA
}

uint8_t I2C_Start(char write_address)/* I2C start function */
{
  uint8_t status;   /* Declare variable */
 TWCR=(1<<TWSTA)|(1<<TWEN)|(1<<TWINT); /* Enable TWI, generate START */
 while(!(TWCR&(1<<TWINT)));  /* Wait until TWI finish its current job */
  status=TWSR&0xF8;   /* Read TWI status register */
 if(status!=0x08)    /* Check weather START transmitted or not? */
    return 0;     /* Return 0 to indicate start condition fail */
  TWDR=write_address;   /* Write SLA+W in TWI data register */
 TWCR=(1<<TWEN)|(1<<TWINT);  /* Enable TWI & clear interrupt flag */
  while(!(TWCR&(1<<TWINT)));  /* Wait until TWI finish its current job */
  status=TWSR&0xF8;   /* Read TWI status register */
 if(status==0x18)    /* Check for SLA+W transmitted &ack received */
```

```
    return 1;     /* Return 1 to indicate ack received */
  if(status==0x20)    /* Check for SLA+W transmitted &nack received */
    return 2;     /* Return 2 to indicate nack received */
 else
   return 3;     /* Else return 3 to indicate SLA+W failed */
}

uint8_t I2C_Repeated_Start(char read_address) /* I2C repeated start function */
{
  uint8_t status;   /* Declare variable */
 TWCR=(1<<TWSTA)|(1<<TWEN)|(1<<TWINT);/* Enable TWI, generate start */
  while(!(TWCR&(1<<TWINT)));  /* Wait until TWI finish its current job */

 status=TWSR&0xF8;   /* Read TWI status register */
 if(status!=0x10)    /* Check for repeated start transmitted */
   return 0;     /* Return 0 for repeated start condition fail */

 TWDR=read_address;    /* Write SLA+R in TWI data register */
 TWCR=(1<<TWEN)|(1<<TWINT);  /* Enable TWI and clear interrupt flag */
  while(!(TWCR&(1<<TWINT)));  /* Wait until TWI finish its current job */

 status=TWSR&0xF8;   /* Read TWI status register */
 if(status==0x40)    /* Check for SLA+R transmitted &ack received */
    return 1;     /* Return 1 to indicate ack received */
  if(status==0x48)    /* Check for SLA+R transmitted &nack received */
   return 2;     /* Return 2 to indicate nack received */
 else
   return 3;     /* Else return 3 to indicate SLA+W failed */
}

uint8_t I2C_Write(char data)  /* I2C write function */
{
 uint8_t status;   /* Declare variable */
 TWDR=data;      /* Copy data in TWI data register */
 TWCR=(1<<TWEN)|(1<<TWINT);  /* Enable TWI and clear interrupt flag */
  while(!(TWCR&(1<<TWINT)));  /* Wait until TWI finish its current job */
  status=TWSR&0xF8;   /* Read TWI status register */
 if(status==0x28)    /* Check for data transmitted &ack received */
   return 0;     /* Return 0 to indicate ack received */
  if(status==0x30)    /* Check for data transmitted &nack received */
    return 1;     /* Return 1 to indicate nack received */
 else
   return 2;     /* Else return 2 for data transmission failure */
}

char I2C_Read_Ack()    /* I2C read ack function */
{
  TWCR=(1<<TWEN)|(1<<TWINT)|(1<<TWEA); /* Enable TWI, generation of ack */
 while(!(TWCR&(1<<TWINT)));  /* Wait until TWI finish its current job */
  return TWDR;      /* Return received data */
}

char I2C_Read_Nack()    /* I2C read nack function */
{
 TWCR=(1<<TWEN)|(1<<TWINT);  /* Enable TWI and clear interrupt flag */
```

```
  while(!(TWCR&(1<<TWINT)));  /* Wait until TWI finish its current job */
  return TWDR;    /* Return received data */
}

void I2C_Stop()     /* I2C stop function */
{
  TWCR=(1<<TWSTO)|(1<<TWINT)|(1<<TWEN);/* Enable TWI, generate stop */
 while(TWCR&(1<<TWSTO)); /* Wait until stop condition execution */
}
```

9.1.3. RTC.h

```
/* This code is modified from https://www.electronicwings.com/avr-atmega/real-
time-clock-rtc-ds1307-interfacing-with-atmega16-32 */

uint8_t second, minute, hour, weekday, date, month, year;

uint8_t BDC2value(char BDC) {
  uint8_t value = 0;
 uint8_t tens = (BDC & 0b11110000) >> 4; // Extract the tens from the upper nibble
  uint8_t ones = (BDC & 0b00001111);        // Extract the ones from the lower
nibble
  value = tens * 10 + ones;         // Combine tens and ones to get the integer
value
  return value;
}

void RTC_Read_Clock(char read_clock_address) {
  I2C_Start(RTC_Write_address); // Start I2C communication with RTC
  I2C_Write(read_clock_address); // Write address to read
  I2C_Repeated_Start(RTC_Read_address);

 second = BDC2value(I2C_Read_Ack());
  minute = BDC2value(I2C_Read_Ack());
  hour = BDC2value(I2C_Read_Nack() & 0b00111111); //Last communication so nack.
The two MSB are not relevant
 I2C_Stop();
}

void RTC_Read_Date(char read_date_address) {
  I2C_Start(RTC_Write_address);
  I2C_Write(read_date_address);
  I2C_Repeated_Start(RTC_Read_address);

 weekday = I2C_Read_Ack();
  date = I2C_Read_Ack();
 month = I2C_Read_Ack();
  year = I2C_Read_Nack();
  I2C_Stop();
}

char hour2BDC(uint8_t _hour){
 // See datasheet for DS3231 how hour is decoded in register
  char BDC = 0;
```

```
 if (_hour >= 10){
    if (_hour >= 20){
      BDC |= 1 << 5; //bit 5 represents 20 hour
      _hour -= 20; //We need the to get just the rest
      } else {
     BDC |= 1 << 4; //bit 4 represents 10 hour
      _hour -= 10;
   }
  }
  //Add in the rest
  BDC |= (_hour & 0b00001111); // The 4 last bits are as normal
  return BDC;
}

char value2BDC(uint8_t value){ // For minutes and seconds
 char BDC = 0;
  uint8_t tens = value/10;  // Get the number of tens in the value
 value -= tens*10;       // value is now the rest
 BDC |= tens << 4;   // The tens is put in from the bit 4
 BDC |= (value & 0b00001111);
 return BDC;
}

void RTC_Clock_Write(char _hour, char _minute, char _second){
 I2C_Start(RTC_Write_address); //Address defined in main
  I2C_Write(0); // Be at the 0 location (second)
 I2C_Write(value2BDC(_second));
 I2C_Write(value2BDC(_minute));
 I2C_Write(value2BDC(_hour));
 I2C_Stop();
}

void RTC_Date_Write(char _weekday, char _date, char _month, char _year){
  I2C_Start(RTC_Write_address);
  I2C_Write(3);   // (First bit sent tells the location to be at) Bet at the 3.
location (weekday)
 I2C_Write(_weekday);
 I2C_Write(value2BDC(_date));
 I2C_Write(value2BDC(_month));
  I2C_Write(value2BDC(_year));
 I2C_Stop();
}

void RTC_Alarm_Init(){
  I2C_Start(RTC_Write_address);
  I2C_Write(0xE);   // Bet at the control register position (See datasheet)
  I2C_Write(0b00000101);  // Enabling interrupt Control and Alarm 1 Interrupt
Enable
 I2C_Stop();
}

void RTC_Alarm1_Time(char _hour, char _minute, char _second){ // The alarm
triggers at the spesified time every day.
  I2C_Start(RTC_Write_address);
  I2C_Write(7);         // Set alarm1 time in register 7
```

```
 I2C_Write(value2BDC(_second));
 I2C_Write(value2BDC(_minute));
 I2C_Write(value2BDC(_hour));
 I2C_Write(0b10000000); // Set A1M4 to trigger at that time set
 I2C_Stop();
}

void RTC_Alarm_Clear(){
 I2C_Start(RTC_Write_address);
  I2C_Write(0xF);
  I2C_Write(0b10001000);  // Write 0 at the BSY, A2F and A1F flag.
 I2C_Stop();
}
```

9.1.4. USART.h

```
void USART_Init(long USART_BAUDRATE)
{
  UCSRB |= (1 << RXEN) | (1 << TXEN); /* Turn on transmission and reception */
 UCSRC |= (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1);/* Use 8-bit char size */
  UCSRA |= 1 << U2X;        /* Double the baudspeed */
 UCSRB |= 1 << RXCIE;    // Interrupts for RX
 long BAUD_PRESCALE = (((F_CPU / (USART_BAUDRATE * 8UL))) - 1);   // Specifies a
Baud Rate Prescale. Multiply by 2 since we use double baud speed
 //long BAUD_PRESCALE = 12;  // Read from the datasheet for 9600
  UBRRL = BAUD_PRESCALE;      /* Load lower 8-bits of the baud rate */
 UBRRH = (BAUD_PRESCALE >> 8);   /* Load upper 8-bits*/
}

unsigned char USART_Receive()
{
  while ((UCSRA & (1 << RXC)) == 0);/* Wait till data is received */
 return(UDR);    /* Return the byte */
}

unsigned char USART_Receive_buffer()
{
  return(UDR);
}

void USART_Transmit(unsigned char data){
  while ((UCSRA & (1 << UDRE)) == 0);   //Wait until the transmitter is ready
  UDR = data; //Put the data in the register
}

void USART_Transmit_String(char str[]){
  int i = 0;
  do {
    char character = str[i];
    USART_Transmit(character);
    i++;
  } while (str[i] != '\0');
}
```

9.1.5. screen.h

```
#define LCD_Data_Dir DDRB   /* Define LCD data port direction */
#define LCD_Command_Dir DDRA /* Define LCD command port direction register */
#define LCD_Data_Port PORTB    /* Define LCD data port */
#define LCD_Command_Port PORTA /* Define LCD data port */

#define RS PA2          /* Define Register Select (data/command reg.)pin */
#define RW PA1          /* Define Read/Write signal pin */
#define EN PA0         /* Define Enable signal pin */

#include <stdio.h>

void LCD_enable_pulse(){
 LCD_Command_Port |= (1<<EN);  /* Enable pulse */
 _delay_us(1);
  LCD_Command_Port &= ~(1<<EN);
  _delay_us(1530); // was 3 ms
}

void LCD_Command(unsigned char cmnd)
{
 LCD_Data_Port= cmnd;
 LCD_Command_Port &= ~(1<<RS); /* RS=0 command reg. */
  LCD_Command_Port &= ~(1<<RW); /* RW=0 Write operation */
 LCD_enable_pulse();
}

void LCD_Char (unsigned char char_data)  /* LCD data write function */
{
  LCD_Data_Port= char_data;
  LCD_Command_Port |= (1<<RS);  /* RS=1 Data reg. */
 LCD_Command_Port &= ~(1<<RW); /* RW=0 write operation */
 LCD_enable_pulse();
}

void LCD_String (char *str)    /* Send string to LCD function */
{
  int i;
 for(i=0;str[i]!=0;i++)    /* Send each char of string till the NULL */
 {
    LCD_Char (str[i]);
 }
}

void LCD_String_xy (uint8_t row, uint8_t pos, char *str)  /* Send string to LCD
with xy position */
{
 if (row == 0 && pos<16)
  LCD_Command(pos|0b10000000);          /* Command of first row and required
position<16 */
  else if (row == 1 && pos<16)
 LCD_Command(pos|0b11000000);          /* Command of first row and required
position<16 */
  LCD_String(str);                /* Call LCD string function */
}
```

```
void LCD_Num(int num)     // Send a number to LCD
{
  char str[16];         // Max screen length 16
  sprintf(str, "%d", num);    // Makes num into str
  LCD_String(str);
}

void LCD_Num_xy(uint8_t row, uint8_t pos, int num)    // Send a number to LCD with
xy position
{
 if (row == 0 && pos<16)
  LCD_Command(pos|0b10000000);          // Command of first row and required
position<16
 else if (row == 1 && pos<16)
 LCD_Command(pos|0b11000000);          // Command of first row and required
position<16
 LCD_Num(num);
}

void LCD_Clear()
{
  LCD_Command (0b1);    // clear display
 LCD_Command (0b10000000);   // cursor at home position
}

void LCD_Newline() {
 LCD_Command(0b11000000);    // Go to 2nd line
}

void LCD_Init (void)     // LCD Initialize function
{
 LCD_Command_Dir = 0xFF;   // Make LCD command port direction as o/p
  LCD_Data_Dir = 0xFF;    // Make LCD data port direction as o/p
 _delay_ms(20);        // LCD Power ON delay always >15ms

 LCD_Command (0b00111000);     // Initialization of 16X2 LCD in 8bit mode
 LCD_Command (0b1100);     // Display ON Cursor OFF
 LCD_Command (0b0110);     // Auto Increment cursor
 LCD_Clear();
}
```