Department of Electronic & Telecommunication Engineering

University of Moratuwa

EN1093 : Laboratory Practice I

Digital Alarm Clock

Group 25

Team Members

190622R O.K.D. Tharindu
190626H P.D. Tharundi
190631T T.A. Thenuwara
190636M A.G.N. Udara

Course Instructor

Randima Senanayake

Abstract

As the final assessment of the EN1093 module, we were given the task of designing a Digital alarm clock. Our Digital alarm clock is a standard Alarm clock with all necessary functionalities and including A simple Timer functionality. We were instructed to use AVR C++ for firmware development and, Proteus for simulations. Also, to use Atmega328P Microcontroller, 16x2 LCD display and for the RTC IC, DS1307 or DS3231. For sound output, we used a simple active piezo buzzer. This document contains the process of creating our Alarm clock, issues we faced, challenges, functionalities, firmware details, and technical specifications.
We also attached a 3D object of our Enclosure design, be sure to use Adobe reader DC for a good experience.

Contents

List of Figures

List of Algorithms

1 Introduction

To utilize what we learned in semester 2 and also with the self-accumulated knowledge throughout the semester, we designed this Digital Alarm clock as a fulfilment for the final assignment of the EN1093 module. It is a fully functional and programmable digital alarm clock with Buzzer as sound output. We are using a DS1307 RTC module with a battery backup power input to store and manage time. We are using I2C protocol to manage the RTC module. Apart from that Atmega 328P will handle all other storage, processing and input/output tasks. 16x2 LCD display is used to display time, dates, alarms and other UI components such as menus and tones. A 4x4 keypad is used for the user inputs and UI controls. A separate button is added for the alarm interruptions. There is a Factory reset button located inside the enclosure which requires a pin to click it. RTC module, LCD display, Piezo Buzzer, keypad, reset and Stop buttons are directly connected to the Atmega 328P Micro Controller. For reprogramming the microcontroller, there is a serial input reserved in the PCB.

1.1 Problem Statement

A Digital Alarm clock, with all the conventional features and minimalist design to reduce the cost but keeping a good user experience.

2 Method

2.1 Steps of the Project

Using teamwork and step by step process, we were able to finish this project. These are the main steps we followed

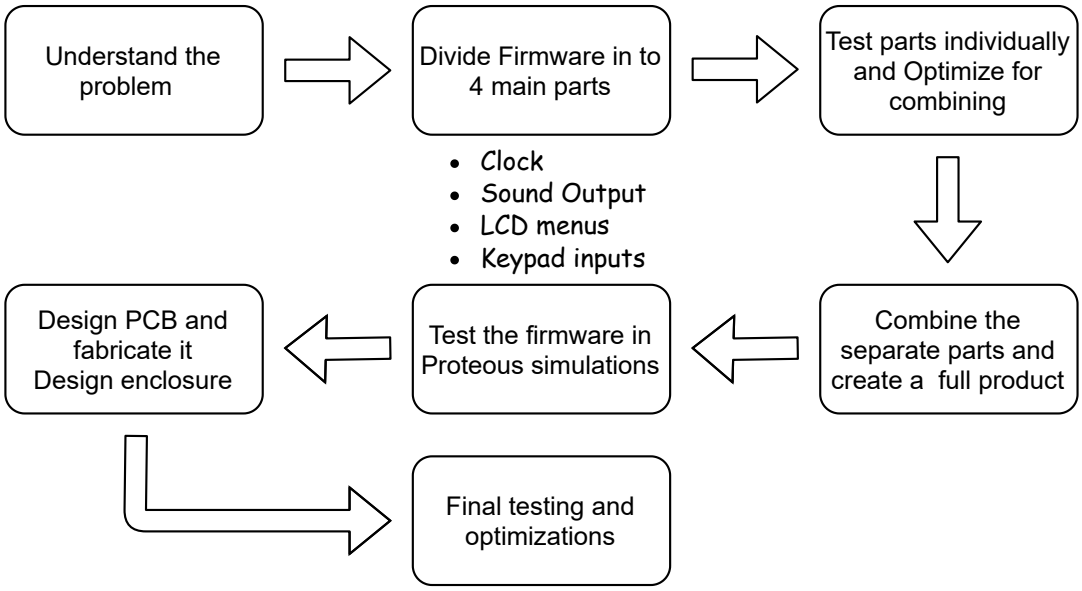


Figure 2.1: Steps

After printing the PCB, the final stage of testings were started. All the components and PCB details is available in the Appendix

2.2 COMPONENTS AND THEIR ALGORITHMS

Here we will look at the main components and how the algorithms and codes work.

2.2.1 The RTC Module

The DS1307 RTC module is working with the I2C protocol with an inbuilt battery chamber so even when the power is not available, The date and time data will be preserved. Using SCL and SDA pins to connect to the microcontroller. The module is temperature dependable so keeping it separated from the main PCB was a solution for the heating. The I2C libraries that we implemented were created by Peter Fleury. The chip maintains seconds, minutes, hours, day, date, month, and year information. The date at the end of the month is automatically adjusted for months with fewer than 31 days, including corrections for leap year.

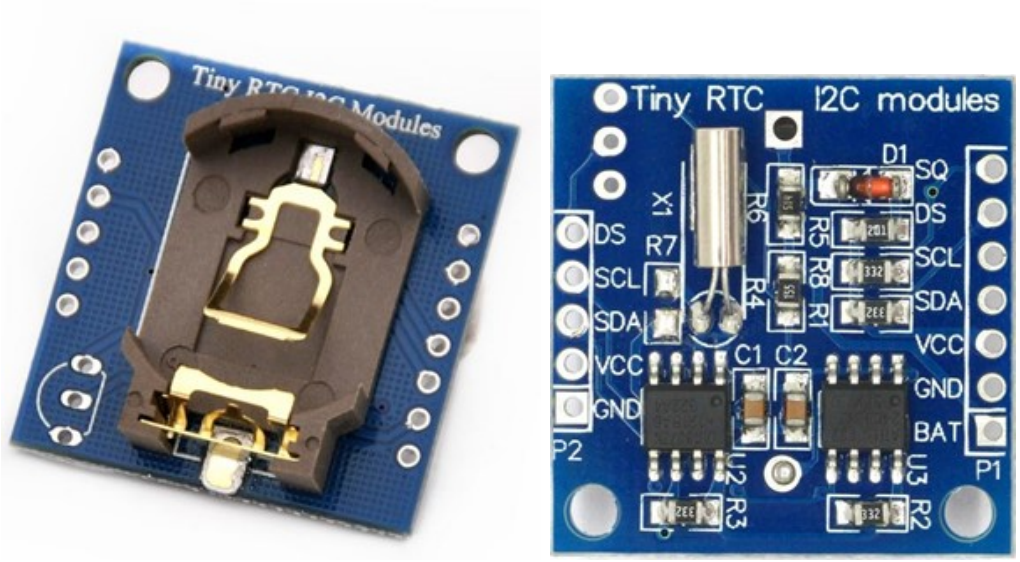
Features

- 32KHz Crystal oscillator
- 3V CR2032 battery backup
- 32 bytes 24C32 EEPROM

Algorithm

With I2C address 0xD0. The module is taking and stores data in 24-hour format.

- ds1307_setdate() Function takes year, month, day, day of week, hour, minute and second in decimal form and send the data to RTC module to rewrite current information.
- ds1307_getdate() Function is used to pull the above information back to the microcontroller.



2.2.2 Buzzer

The Active piezo buzzer is the Sound output device in the Alarm Clock and it used Square waves to generate tones. The pitches header file contains the keys and their designated frequencies. Using this pitch data, time ratios and provided tempo data given in the melodies header file will be used to generate tone sequences. The negative time ratio sequences indicate extended tones. Positive ones are shorter. There are 5 songs and these will be used to trigger the alarm alerts. This data is stored in Program memory to utilize the storage properly. We will use the PD2 pin to connect The buzzer to Atmega 328P.The melodies are adapted from the works of Robson Couto.

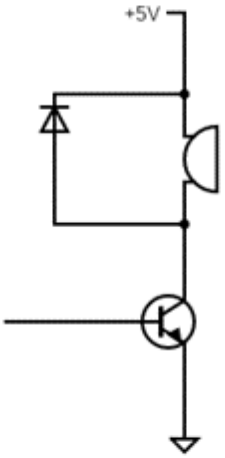
Features

- Active Buzzer
- Using amplifier for better sound volume created with a BJT
- The diode across the buzzer prevent sudden voltage spikes and protect the BJT

Algorithm

To generate a tone out of the buzzer, it needs a signal with that specific frequency. Using simple delays and toggling the output pin we can archive this while generating 50% duty cycle square waves. Since this part is also working as the Alarm output, to implement an interrupt to the system we will use external interrupt pin PD3 to utilize this. Upon the rising edge high, the tone playing function will stop.

- Play_Note() Function is used to generate 50% duty square waves when the relevant frequency and the time duration is given. First, generate the time duration per cycle and then calculate the number of cycles. For each cycle turn 50% of time output high and then turn output low for the remaining 50% cycle. Repeat this for all cycles. If the input frequency was 0, then the output will be 0 for the given duration.
- Play() function will take single integer input which is the song number. After that reading values from the melodies header, it will take notes and duration and trigger Play_Note() function with the predetermined length and tempo will be used to generate the tone duration. The external interrupt button can be used to stop this function and stop the sound from playing.



2.2.3 LCD Display

This 16-character, 2-line parallel liquid crystal display achieves a large viewing area in a compact package. The DDRAM address 0x00 corresponds to the first character of the top line, address 0x0F corresponds to the last character of the top line, address 0x40 corresponds to the first character of the second line, and address 0x4F corresponds to the last character of the second line.

The main functionality of the LCD is to display

- Time/Date
- Control menus
- Tone selections

We are utilizing maximum contrast by grounding the V_0,V_{EE} pin and 4bit configuration mode. Total 6 pins are connected with the microcontroller using a male-female connector. The datasheet will be provided.

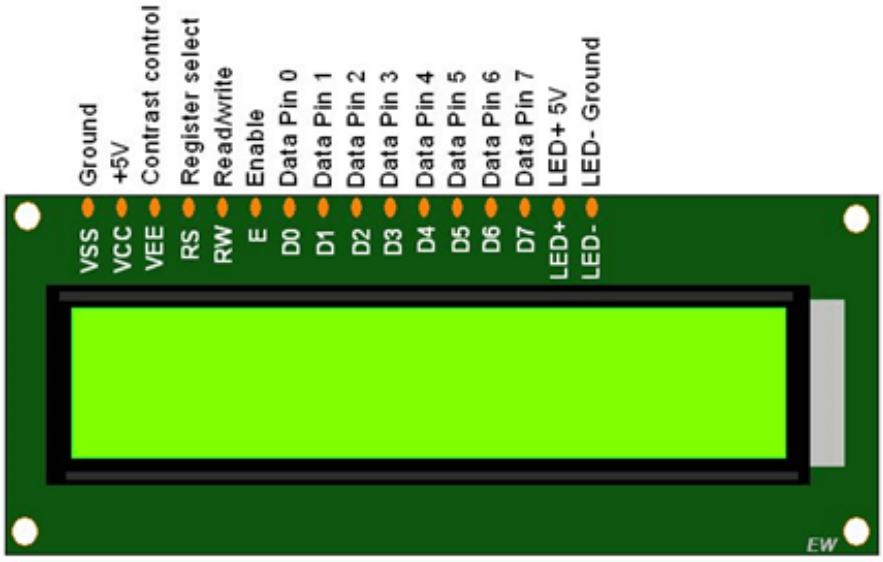
Features

- Using 4Bit mode to utilize pins efficiently
- Blue backlight
- Looks Stylish
- Always-on display

Algorithm

All the menus and content displaying will be done by this algorithm.

- displayTime() will be used to display the time and date on the main screen. The data for this display option will be taken from the RTC module..
- LCD_SetAlarm() Function is used to take user inputs from the 4x4 keypad and set one alarm at a time. To reach this point user will have to choose from "Set Alarm" from the listed menu.
- LCD_Tone() Function will display currently available Alarm tones. The alarm tone names are stored In an array named tone_List. You can select a tone and this function will give an integer output for the selected song index.
- LCD_SetDate() This function will grant the user to update system tie and date with the help of keypad inputs.



2.2.4 Keypad input

We are utilizing a 4x3 matrix keypad for our input purposes. Using number keys as basic number inputs as well as for menu navigation

- 8 :- UP
- 2 :- DOWN
- #:-RETURN/ CANCEL
- * :- ENTER/ SELECT

Features

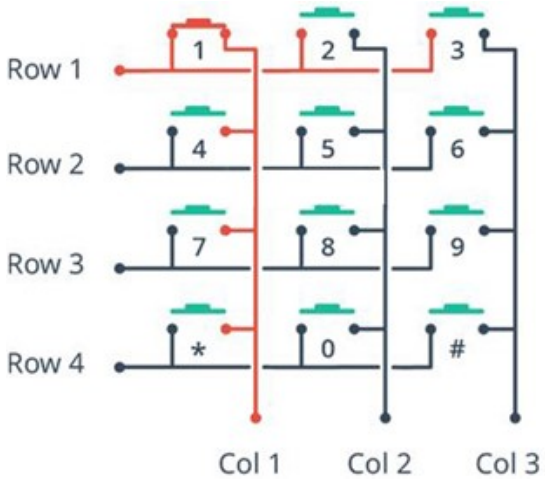
- Active Buzzer
- Using amplifier for better sound volume created with a BJT
- The diode across the buzzer prevent sudden voltage spikes and protect the BJT

use 7 pins for communication with the Microcontroller.

Algorithm

In a nutshell, keypad works as a matrix of push buttons. There are 4 output pins (PC0 to PC3), and the inputs will be toggled in a sequence from top raw to bottom raw. In a instance if a button is pressed we can recognize that row with the toggled on pin. At instance of the button is pressed, that input pin will toggle 5V accordingly. There are 3 input pins (PB0 to PB2). They are connected using a pull down resistor. By the column and raw number, system can recognize the pressed button.

- btnPress() function will check the pins and ports and return the character given in the location reading the matrix data. We will utilize this to get properly control the inputs.



2.3 Features and Overall working Methodology

Here we will look at the features and how the overall algorithm work.

2.3.1 Displaying Time and date

After startup and after the Welcome screen, the date and time data will be constantly be pulling out of the RTC module. And with the LCD display, it will keep updating accordingly. This process doesn't have any specified delays. It will constantly do this process till a key input is given.

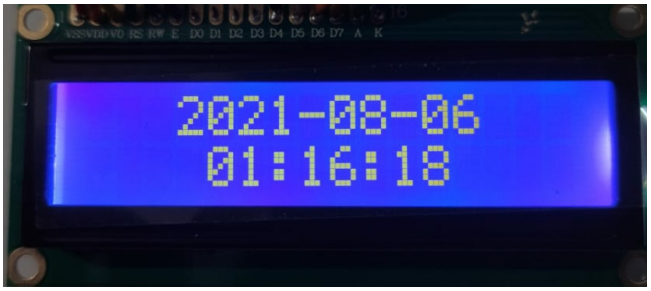


Figure 2.2: Time and date

After a key input, It will navigate to the First list containing

1. SET ALARM
2. SET TIME
3. SEE ALARMS
4. TIMER

2.3.2 Selecting sub menus and features.

Here we will check each scenario.

2.3.2.1 SET ALARM

If you select *SET ALARM*, it will navigate to a new screen where you need to enter the alarm details and you can use the keypad numbers to enter the alarm time. The Alarm should be inserted in hh:mm order. After that pressing " ∗" will take to the next stage to select an alarm tone. To navigate the list up and down, the 2 and 8 buttons are reserved. After choosing the alarm, alarm is set. The algorithm will update a list of arrays with the Alarm Time and Chosen Tone data. These data will be linked by an index.

Figure 2.3: Setting An Alarm LCD Menus

2.3.2.2 SET TIME

If you select $SET\,TIME$, it will open a new screen to set alarm and time. Using keypad inputs date should be added in $yyyy/mm/dd$ format and time in $hh:mm:ss$ format. After setting time this data will be sent to RTC module with the ds1307_setdate() function.

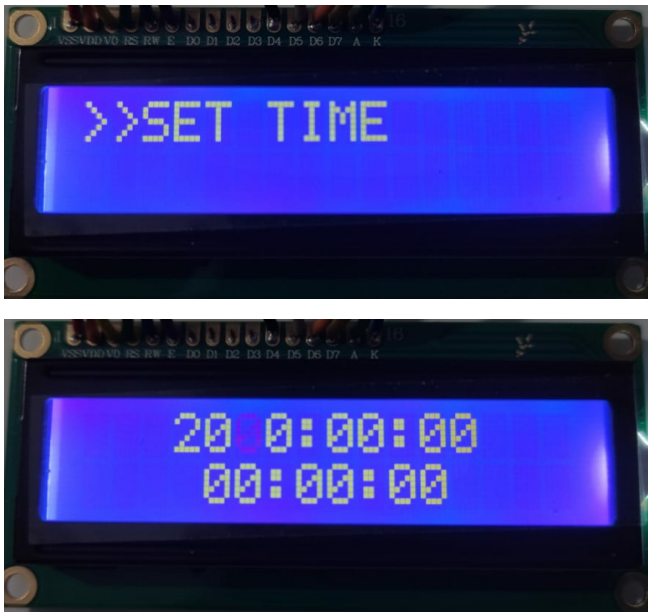


Figure 2.4: Changing LCD Menus

2.3.2.3 SEE ALARMS

If you select *SEE ALARMS*, you can see all the currently available Alarms. If there is Non, A No alarms message will be displayed. From here you can delete alarms as you please. "5" Key is reserved for that purpose. User can remove any and all alarms, if the list is empty, it will say that.

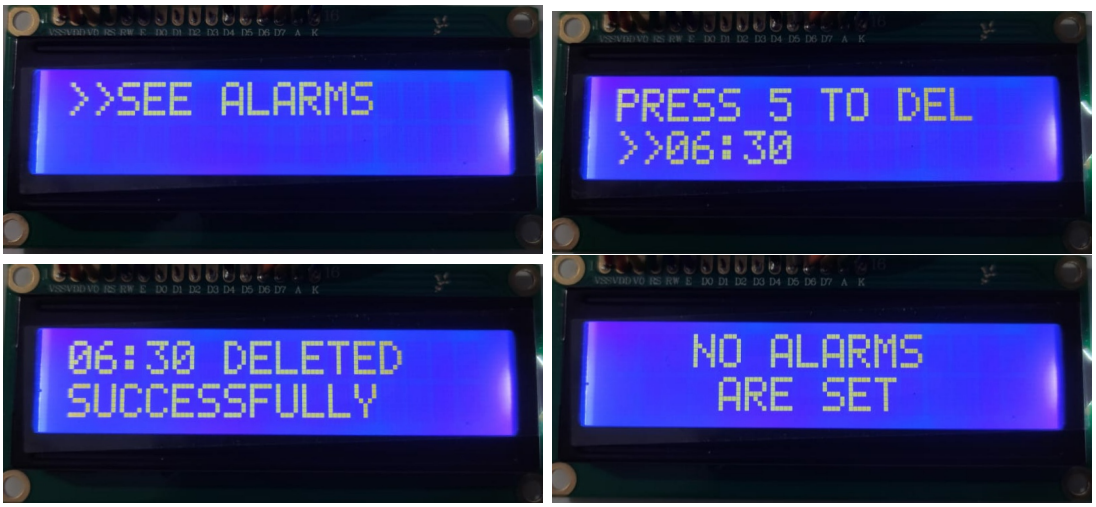


Figure 2.5: Available Alarm list LCD Menus

2.3.2.4 TIMER

TIMER will give you a countdown timer. You can set how long of a timer you need by setting the time in $hh : mm : ss$ format. After the timer reach 00:00:00, Star Wars Imperial march theme will be played.

Figure 2.6: Timer LCD Menus

2.3.3 Alarm interrupting

Since Alarms usually get abused when stopping alarm, we added a separate, durable and big button for this purpose. By using external interrupt pins, the "*sp*" variable will be changed to 0. Using this even when the microcontroller is busy with the for loop, checking this variable can be used to stop the buzzer music from playing.

2.3.4 Factory resetting

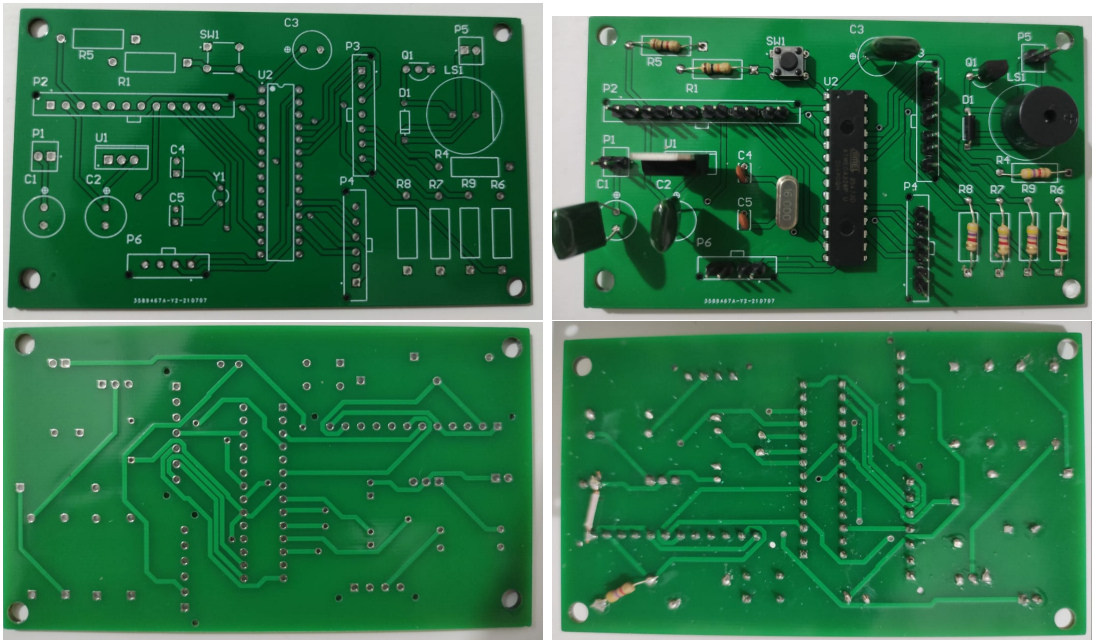
Using the inbuilt reset pin, which is also shared with the programmer, the factory resetting Will be done. The push button will reset everything except the RTC module because it has a separate memory inside. In the enclosure a pin cutout is provided for the user to push the hidden button with a pin. This will stop accidental button presses.

2.3.5 Programmer

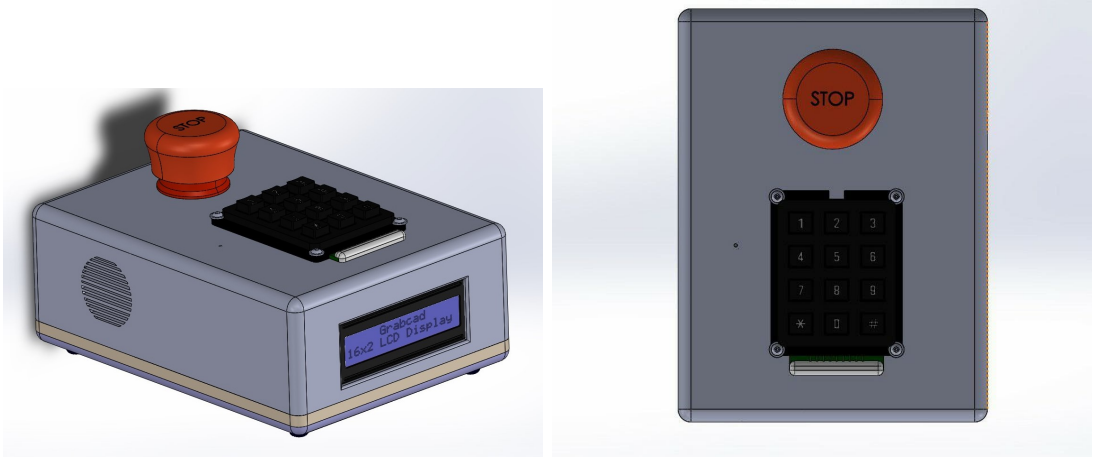
Since our goal was to create a re-programmable digital alarm clock, there is an inbuilt programmer pin reserved header. Using either a USB to serial/TTL programmer or an Arduino device, user can flash firmware. The source code is available in this GitHub Repository.

3 Results

Our PCB design was working as intended but one pin in LCD header was not grounded. The Buzzer had a good volume. The LCD had a contrast issue but was able to fix it by placing a resistor in the pin 3. The current usage was good. The keypad had good responses, the debouncing effect was in an acceptable range. After continuous run over a week, the Time in the RTC module was not affected.



As for our enclosure design, we were not able to 3D print it due to quarantine and other issues.



4 Discussion

As beginners for the embedded designing, we had to learn all the software such as,

- Proteus
- Altium designer
- Solidworks

from the fundamentals.
Also, this was the first time we were introduced to AVR C++ and, we did not have much background in C language.We had to learn AVR C++ from the basics.

4.1 Problems and challenges we faced during this project

- Since the start of the semester, due to COVID 19 pandemic, our team was not able to collaborate in person.
- We had problems with acquiring few parts we needed for our product.
- The deliveries were delayed. So we had to wait for the ordered parts to come for a long time.
- Due to a manufacturing error, our PCB had an unrouted pin.
- Proteus simulations have frequency issues. So we had to alter codes many times for the buzzer. Due to this problem we are using $1MHz$ internal frequency.
- LCD display had too high a contrast level, we had to connect a resistor in order to get proper readability.

4.2 Overall impression and possible improvements

Even though we were not able to 3D print our enclosure design and test it, our design has a good durability and the practicality. The basic box design is suitable for keeping internal components safe from hard impacts. The external button for stopping the alarm is really useful when the user is in a hurry and if the impact is high.The buzzer volume is load and the alarm tones are in a wide range of styles for user to choose.

Since there is no mechanism to change the LCD brightness level, this might become a problem for some scenarios. We will fix this issue and add a option to change screen timeout time.

Overall, the project was a tremendous success, both in its operations and in the lessons obtained from taking on such an involved project. It was a really good experience and gave us the push we needed to start our own projects on embedded systems. We had issues when printing the PCB and finally decided to order the PCB from JLC-PCB. Since from the beginning, when we were writing firmware for our micro products, the codes we wrote were easily combinable and was able to communicate with each parts easily.

There are few features we would like to add to this design. Adding a snooze option could improve user experience. This can be implemented by using the same interrupt button. We can add a option for this by providing a new settings menu too. Giving access to our 9V battery without removing the whole case is also a good feature to adept.

5 Acknowledgement

First of all, we would like to pay our gratitude toward our supervisor/instructor/evaluator Mr. Randima Senanayaka for his guidance and encouragement to overcome this challenge through self study and teamwork. Also, without ENTC family this project would not reach success. We would like to thank all the staff members who helped us throughout this project and semester 2. Our sincere gratitude for all the parties behind providing us with free Solidworks and Altium licenses.
The help, encouragement, tutorials we received from all the guest lecturers from $EN1070$ module is invaluable and we would like to pay our gratitude.

6 Appendix

All the relevant files and progress data is available in this Github repository.

6.1 The PCB

Schematics, PCB and table of content. Used Altium Designer to design these models

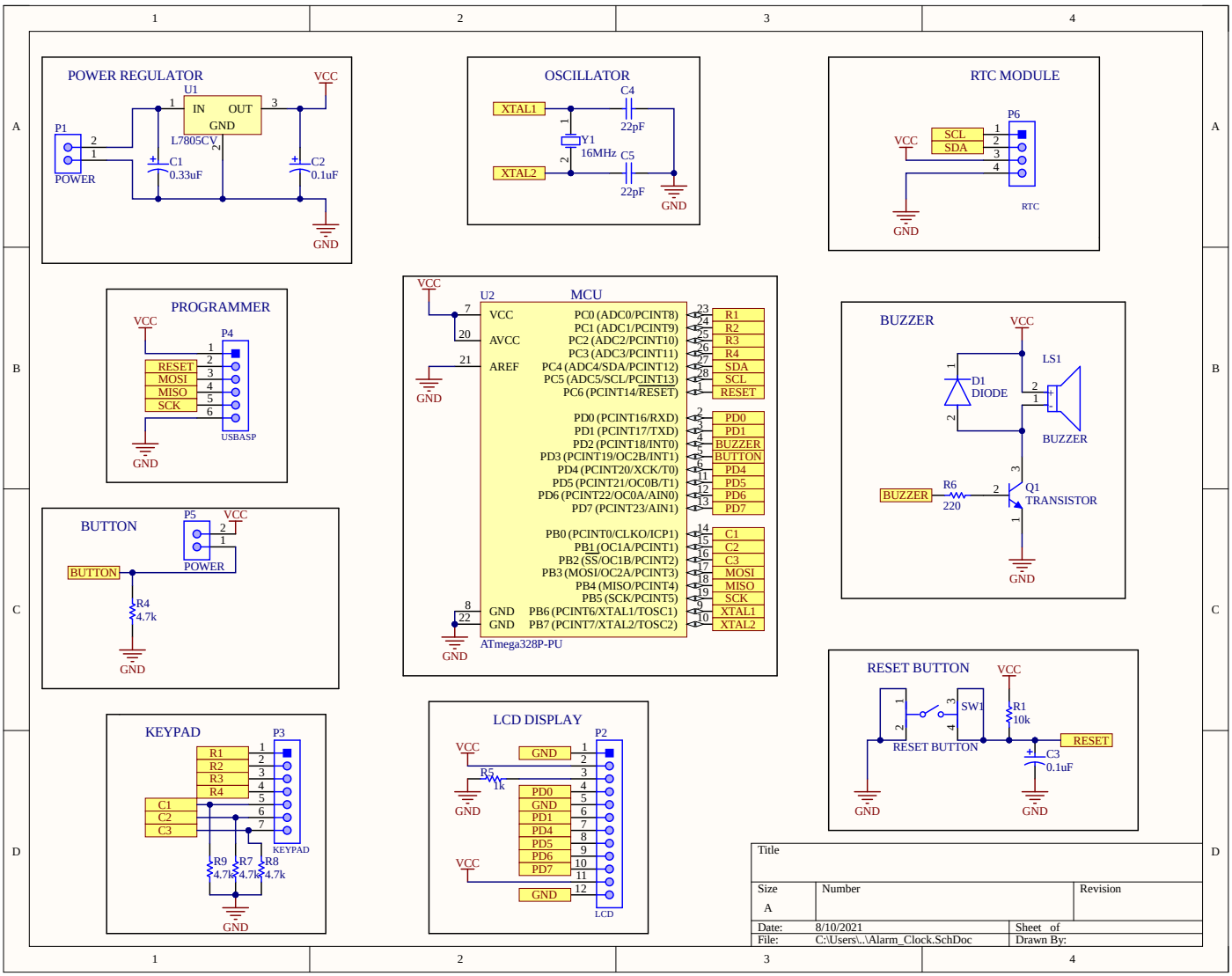


Figure 6.1: Schematics Design

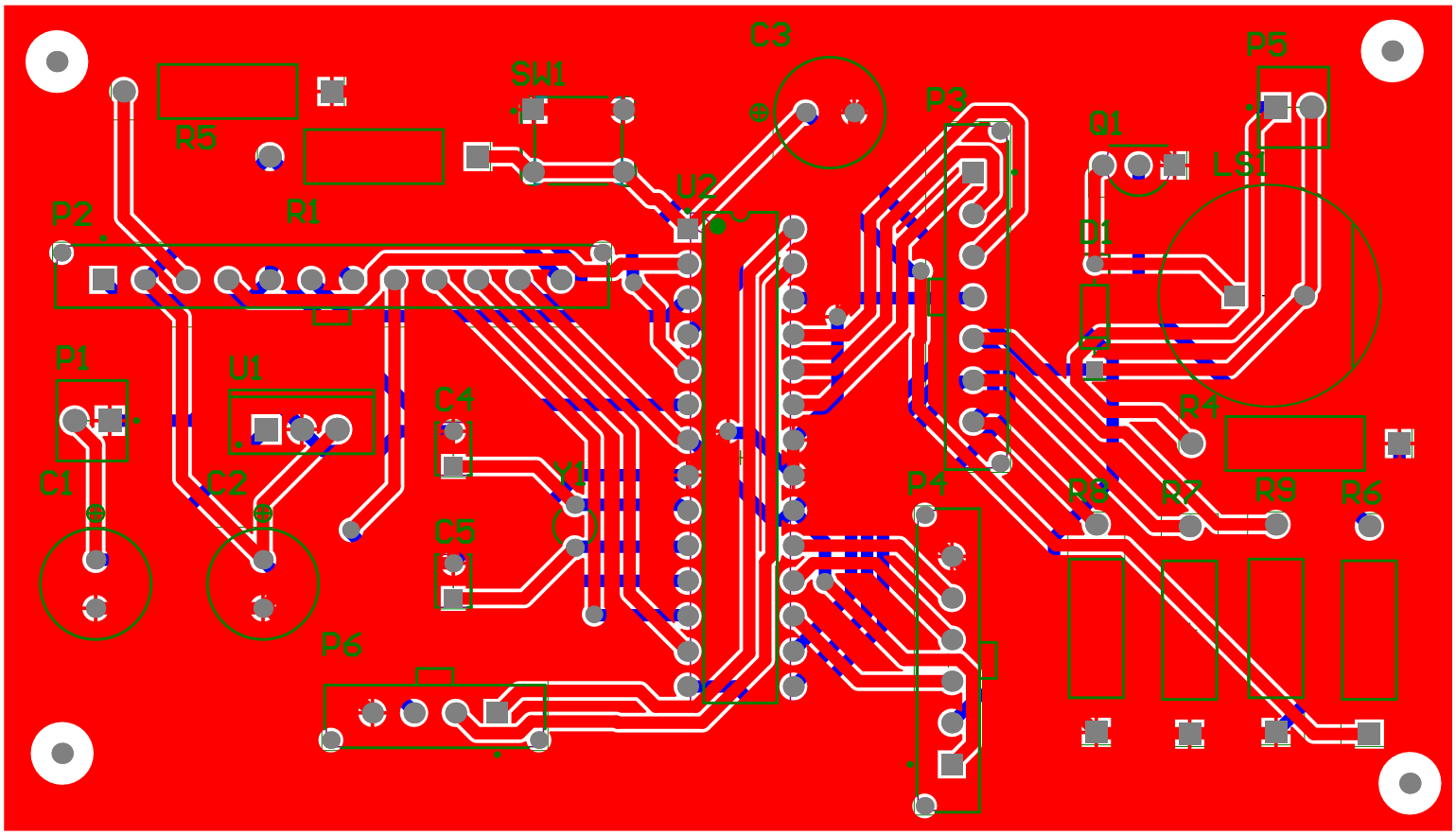


Figure 6.2: PCB Design

Bill of Materials

Bill of Materials for Project [Alarm_Clock.PrjPcb] (No PCB Document Selected)

Source Data From: **Alarm_Clock.PrjPcb**
Project: **Alarm_Clock.PrjPcb**
Variant: **None**

Creation Date: 8/10/2021 1:01 PM
Print Date: 10-Aug-21 1:01:17 PM

Footprint	Comment	LibRef	Designator	Description	Quantity
PNSC-FR-A-	0.33uF	CMP-0744-	C1	Aluminum Electrolytic Capacitor, 1000 uF	1
PNSC-FR-A-	0.1uF	CMP-0744-	C2, C3	Aluminum Electrolytic Capacitor, 1000 uF	2
KEMT-C315-	22pF	CMP-1664-	C4, C5	Capacitor, Ceramic, 0.01 uF, +/- 10%, 100	2
FP-017AG-	DIODE	CMP-07163-	D1	DIODE GEN PURP 15V 150MA DO35	1
FP-	BUZZER	CMP-08413-	LS1	AUDIO MAGNETIC INDICATOR 4-8V TH	1
MOLX-22-27-	POWER	CMP-2000-	P1, P5	Male Header, Pitch 2.54 mm, 1 x 2 Position,	2
66201211122	LCD	CMP-1502-	P2	THT Male Header WR-MPC3, Vertical,	1
66200711122	KEYPAD	CMP-1502-	P3	THT Male Header WR-MPC3, Vertical,	1
66200611122	USBASP	CMP-1502-	P4	THT Male Header WR-MPC3, Vertical,	1
66200411122	RTC	CMP-1502-	P6	THT Male Header WR-MPC3, Vertical,	1
FAIR-TO-92-	TRANSISTOR	CMP-2000-	Q1	NPN General-Purpose Amplifier, 40 V, -55	1
VISH-A1-	10k	CMP-1240-	R1	Power Metal Film Leaded Resistor, 62	1
VISH-A1-	4.7k	CMP-1240-	R4, R7, R8, R9	Power Metal Film Leaded Resistor, 62	4
VISH-A1-	1k	CMP-1240-	R5	Power Metal Film Leaded Resistor, 62	1
VISH-A1-	220	CMP-1240-	R6	Power Metal Film Leaded Resistor, 62	1
FP-1825910-	RESET BUTTON	CMP-03407-	SW1	SWITCH TACTILE SPST-NO 0.05A 24V	1
TO220	L7805CV	CMP-0244-	U1	Positive Voltage Regulator, 12V, 3-Pin TO-	1
28P3	ATmega328P-	CMP-0095-	U2	8-bit AVR Microcontroller, 32KB Flash, 1KB	1
R38	16MHz	XTAL	Y1	Crystal Oscillator	1
					25

Approved **Notes**

Figure 6.3: BOM

6.2 Proteus schematics

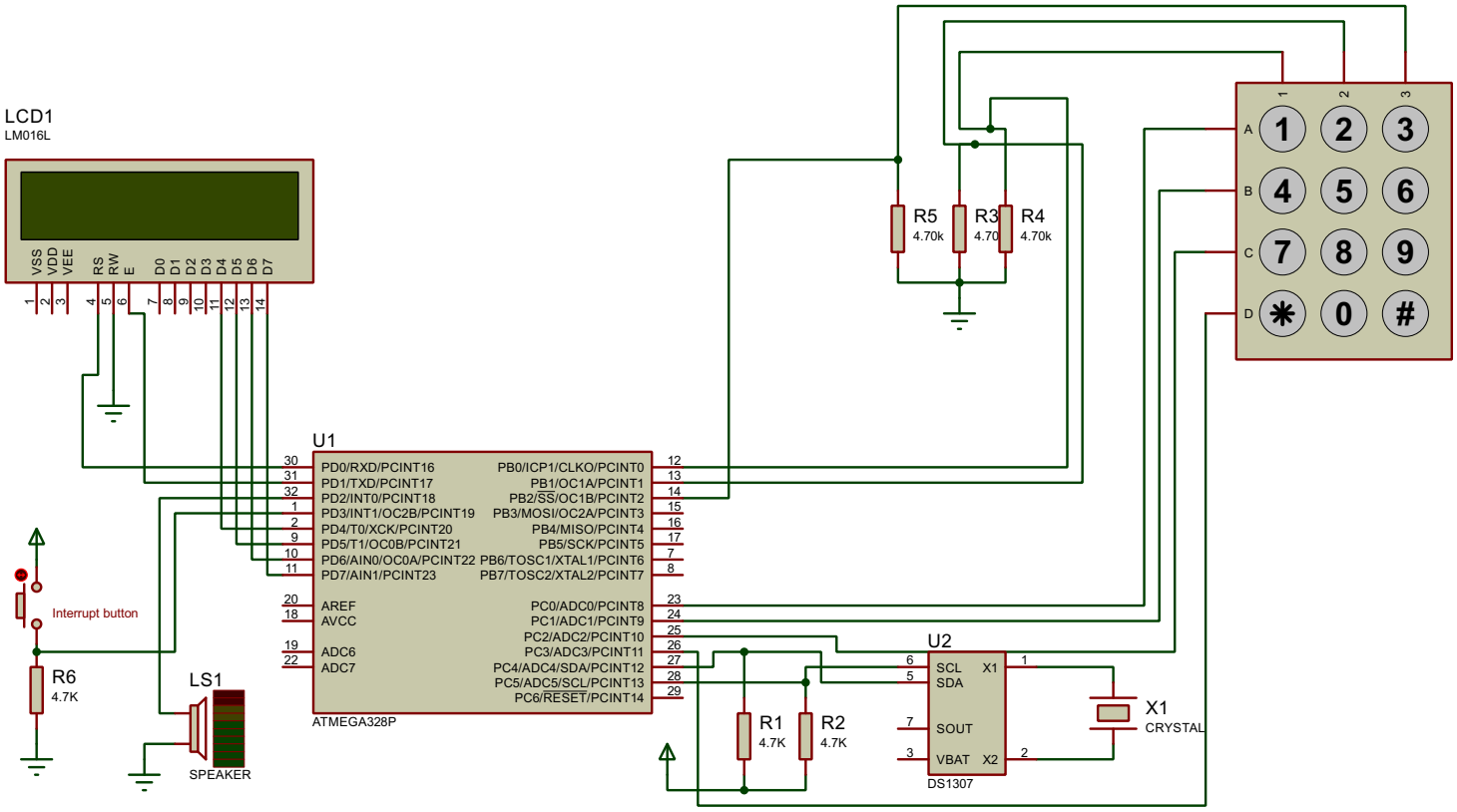


Figure 6.4: Proteus Simulation

6.3 Enclosure

Please use Adobe reader DC to view this. Also for better view, open in full screen by right clicking.

Figure 6.5: 3D Enclosure View

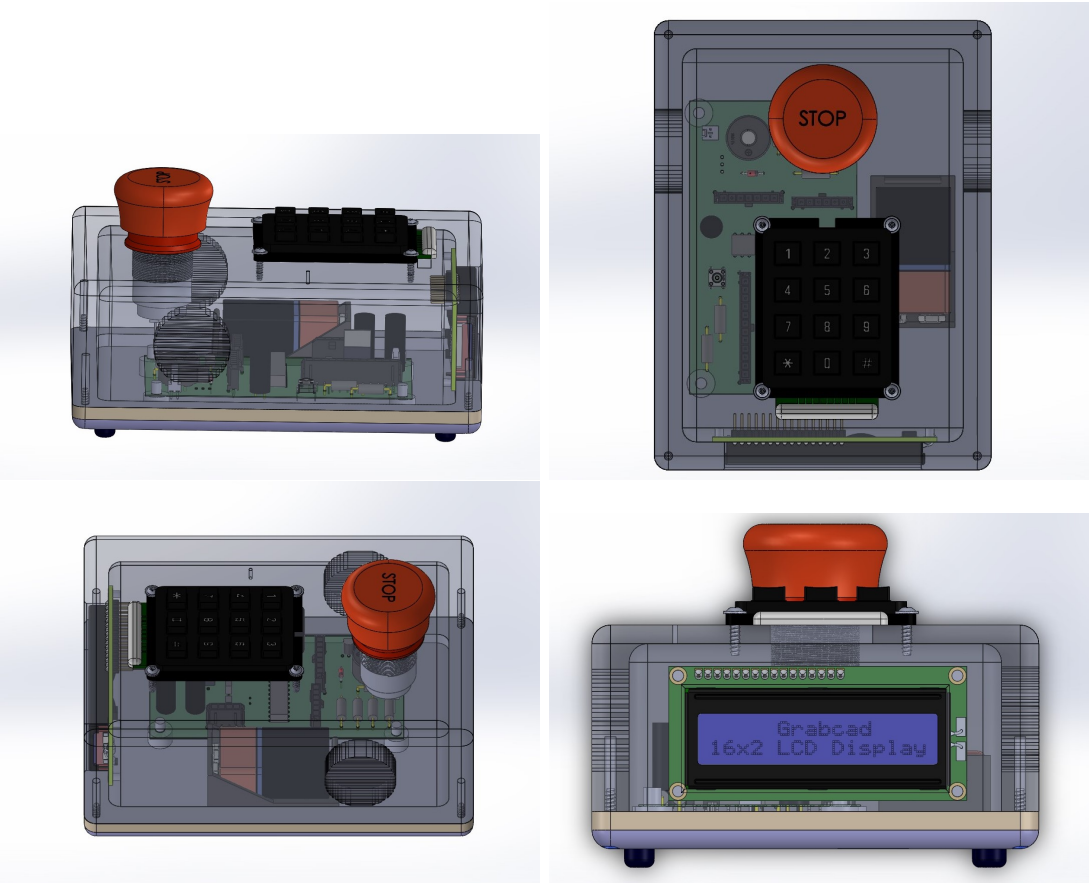


Figure 6.6: Transparent Enclosure views

6.4 AVR Codes

```
#include <avr/io.h>     /* Include AVR std. library file */
#include <avr/interrupt.h>
#include "Display.h"
#include "ds1307.h"
#include "Alarm.h"
#include "Keypad.h"
#include "util/delay.h"
#include "Buzzer.h"
#define BUTTON_DELAY 500

#ifndef F_CPU
#define F_CPU 1000000UL
#endif
int main()
{
  init_buzzer();    // Initialize pins used for buzzer and Interrupt
    button
  LCD_Init();     /* Initialization of LCD*/
  LCD_String("   WELCOME!!!");
  _delay_ms(3000);
  ds1307_init();
  //int clocktime[6] = {21,05,4,12,8,35};
  //setClockTime(clocktime);
  //setAlarm(1209);setAlarm(1223);
  int menu = 0;
  char key;
  while(1){
    checkAlarm();
    key = btnPress();
    if (menu==0){
      displayTime();
    }
    if (key=='*'){
      _delay_ms(BUTTON_DELAY);
      menu ++;
      LCD_Clear();
    }
    if (menu>0){
      LCD_Menu(key,menu);
    }
    if (menu==3 | menu==-1){
      menu = 0;
      LCD_Clear();
    }
    if (key=='#'){
      _delay_ms(BUTTON_DELAY);
      menu --;
      LCD_Clear();
    }
  }
}
```

Algorithm 6.1: Main.ccp

```
/*
* Alarm.h
*
* Created: 5/13/2021 8:39:09 PM
* Author: Dakshina Tharidndu
*/


#ifndef __ALARM_H__
#define __ALARM_H__
#include <avr/io.h>
class Alarm
{
//variables
public:
protected:
private:

//functions
public:
  Alarm();
  ~Alarm();

protected:
private:
  Alarm( const Alarm &c );
  Alarm& operator=( const Alarm &c );

}; //Alarm
extern bool isPress(uint8_t prt);
extern int powerOf(int base, int power);
//extern void getAlarmTime();
extern void ringAlarm(int tone);
extern void setAlarm(int alarm_time);
extern void checkAlarm();
extern void updateAlarmArray(int removeAlarm);
extern void setClockTime(int clockTime[6]);
extern int getDay(int y, int m, int d);
extern void displayTime();
extern void displayZero(uint8_t digit);
extern void setTone(int t);
extern void timer(char key,int state);
extern void timerCountDown();
extern void resetAlarmVariables();
#endif //__ALARM_H__
```

Algorithm 6.2: Alarm.h

```


using namespace std;
#include "stdlib.h"
#include "Alarm.h"
#include "Display.h"
#include "util/delay.h"
#include <avr/io.h>
#include "ds1307.h"
#include "Buzzer.h" //This is used to play the tone with the speaker
#define BUTTON_DELAY 500
#ifndef F_CPU
#define F_CPU 1000000UL
#endif

```

```
// default constructor
Alarm::Alarm()
{
} //Alarm

// default destructor
Alarm::~Alarm()
{
} //~Alarm
int numberOfAlarms = 0;
int alarmArray[10];
int toneArray[10];
int monthsDays[12]={31,28,31,30,31,30,31,31,30,31,30,31};
uint8_t year = 0;
uint8_t month = 0;
uint8_t day = 0;
uint8_t dayofweek = 0;
uint8_t hour = 0;
uint8_t minute = 0;
uint8_t second = 0;
int powerOf(int base, int power){
  int result = 1;
  while(power!=0){
    result*=base;
    power--;
  }
  return result;
}
bool isPress(uint8_t prt){
  if (PIND == 1<< prt){
    return true;
  }
  else{
    return false;
  }
}
void ringAlarm(int tone){
  LCD_Clear();
  LCD_String("RING!!");
  play(tone);       //Trigger buzzer. Can be interrupted by the int1
    button.
  //_delay_ms(2000);
}
void setAlarm(int alarm_time){
  if (alarm_time<2400){
    alarmArray[numberOfAlarms] = alarm_time;
    numberOfAlarms++;
  }
}
void setTone(int t){
  toneArray[numberOfAlarms] = t;
}
void updateAlarmArray(int removeAlarm){
  for (int i=0;i<numberOfAlarms;i++){
    if (i<removeAlarm){
      alarmArray[i] = alarmArray[i];
      toneArray[i] = toneArray[i];
    }
    else{
      alarmArray[i] = alarmArray[i+1];
      toneArray[i] = toneArray[i+1];
    }
```

```
  }
}
void checkAlarm(){
  if (numberOfAlarms != 0){
    ds1307_getdate(&year, &month, &day, &dayofweek, &hour, &minute, &
    second);
    for (int i = 0; i <= numberOfAlarms;i++){
      if (alarmArray[i]/100 == hour && alarmArray[i]%100 == minute){
        ringAlarm(toneArray[i]);
        updateAlarmArray(i);
        numberOfAlarms--;
      }
    }

  }
}
void setClockTime(int clockTime[6]){
  int y = clockTime[0];
  int m = clockTime[1];
  int d = clockTime[2];
  int D = getDay(y,m,d);
  int h = clockTime[3];
  int M = clockTime[4];
  int s = clockTime[5];
  if ((0<y) & (y<99) & (0<m)&(m<13) & (0<d)&(d<32) & (0<=h)&(h<24) & (0<=
    M)&(M<60) & (0<=s)&(s<60)){
    ds1307_setdate(y, m, d, D, h, M, s);
  }
}
int getDay(int y, int m, int d){
  int days = 0;
  for (int i = 1;i<=y;i++){
    if (i%4 == 0){
      days+= 366;
    }
    else{
      days+= 365;
    }

  }
  for (int j = 0;j<m-1;j++){
    if ((y%4==0) & (j == 1)){
      days+=monthsDays[j];
      days+=1;
    }
    else{
      days +=monthsDays[j];
    }
  }
  return (days+d+5)%7;
}
void displayValue(int val){
  char valS[10];
  itoa(val,valS,10);
  LCD_String(valS);
}
void displayZero(uint8_t digit){
  if (digit<10){
    LCD_String("0");
  }
}
void displayTime(){
```

```
137   ds1307_getdate(&year, &month, &day, &dayofweek, &hour, &minute, &second
      );
138 //  char secondS[10];char minuteS[10];char hourS[10];char yearS[10];char
      monthS[10];char dayS[10];
139 //  itoa(second,secondS,10);itoa(minute,minuteS,10);itoa(hour,hourS,10);
      itoa(year,yearS,10);itoa(month,monthS,10);itoa(day,dayS,10);
140   LCD_Home(0);LCD_String("   20");
141   displayZero(year);displayValue(year);LCD_String("-");
142   displayZero(month);displayValue(month);LCD_String("-");
143   displayZero(day);displayValue(day);
144   LCD_Home(1);LCD_String("    ");
145   displayZero(hour);
146   displayValue(hour);
147   LCD_String(":");
148   displayZero(minute);
149   displayValue(minute);
150   LCD_String(":");
151   displayZero(second);
152   displayValue(second);
153   //_delay_ms(1);
154
155 }
156
157 int alarm_var = 0;
158 char alarms[10];
159 void showAlarms(char key,int state){
160   LCD_Home(0);
161   if ((key=='8') & (state==2)){
162     alarm_var++;
163     //LCD_Clear();
164     _delay_ms(BUTTON_DELAY);
165   }
166   if ((key=='2') & (state ==2)){
167     alarm_var--;
168     //LCD_Clear();
169     _delay_ms(BUTTON_DELAY);
170   }
171   if ((key=='5') & (numberOfAlarms!=0)){
172     LCD_Clear();
173     displayTyping(alarmArray[alarm_var]/100,3);LCD_String(":");
      displayTyping(alarmArray[alarm_var]%100,3);
174     LCD_String(" DELETED");LCD_Home(1);LCD_String("SUCCESSFULLY");
175     _delay_ms(3000);
176     LCD_Clear();
177     updateAlarmArray(alarm_var);
178     numberOfAlarms--;
179     _delay_ms(BUTTON_DELAY);
180
181
182   }
183   if (alarm_var>=numberOfAlarms){
184     alarm_var=0;
185   }
186   if (alarm_var<0){
187     alarm_var=numberOfAlarms-1;
188   }
189   if (state==2){
190     LCD_Home(0);
191     if (numberOfAlarms==0){
192       LCD_String("   NO ALARMS");
193       LCD_Home(1);
194       LCD_String("    ARE SET");
```

```
    }
    else{
      LCD_String("PRESS 5 TO DEL");
      LCD_Home(1);
      LCD_String(">>");
      displayTyping(alarmArray[alarm_var]/100,3);LCD_String(":");
    displayTyping(alarmArray[alarm_var]%100,3);

//      displayTyping(alarmArray[(alarm_var+1)%numberOfAlarms]/100,3);
    LCD_String(":");
//      displayTyping(alarmArray[(alarm_var+1)%numberOfAlarms]%100,3);
    }
  }
}
unsigned int timerTime[3];
int timerState = 0;
int temp1 = 0;
int data1 = 0;
int start=0;
void timer(char key, int state){

  LCD_Home(1);
  LCD_String("    ");
  for (int i=0;i<3;i++){
    if ((i!=3) & (i!=0)){
      LCD_String(":");
    }
    if (i==timerState){
      displayTyping(timerTime[i],temp1);
    }
    else{
      displayTyping(timerTime[i],3);
    }
  }

  if (timerState==3){
    LCD_Home(0);

    if (key=='5'){
      start=1;
    }
    if (start==1){
      LCD_String("PRESS # TO STOP ");
      timerCountDown();
    }
    if (start==0){
      LCD_String("PRESS 5 TO START");
    }
    if ((timerTime[2]==0) & (timerTime[1]==0) & (timerTime[0]==0) ){
      start=0;
      timerState=0;
      LCD_Clear();
      LCD_String("TIMES UP!!!");
      play(1);
    }
  }
  else{
    LCD_Home(0);
    LCD_String("ENTER THE TIME");
  }
  if ((int(key)>47) & (int(key)<58) & (state ==2) & (timerState!=3)){
    data1+=(int(key)-48);
```

```
    _delay_ms(BUTTON_DELAY);
    if (temp1 == 0){
      data1*=10;
      temp1 = 1;
      timerTime[timerState] = data1;
    }
    else{
      timerTime[timerState] = data1;
      timerState++;
      data1=0;
      temp1 = 0;
    }
  }

}

int tempSec=60;
void timerCountDown(){
  ds1307_getdate(&year, &month, &day, &dayofweek, &hour, &minute, &second
    );

  if (tempSec!=second){
    timerTime[2]-=1;
    tempSec=second;
  }
  if (timerTime[2]==-1){
    if (timerTime[1]==0){
      if (timerTime[0]!=0){
        timerTime[1]=59;timerTime[0]-=1;
      }
    }
    else{
    timerTime[2]=59;timerTime[1]-=1;
  }
}
}
void resetAlarmVariables(){
  tempSec=60;
  timerTime[0]=0;timerTime[1]=0;timerTime[2]=0;
  timerState = 0;
  temp1 = 0;
  data1 = 0;
  start=0;
}
```

Algorithm 6.3: Alarm.ccp

```
/*
 * Buzzer.h
 *
 * Created: 14/06/2021 16:34:31
 *  Author: Nuwan Udara
 */


#ifndef BUZZER_H_
#define BUZZER_H_

/**
When the duration and note frequency are given this will generate 50%
    duty
cycle square wave with given frequency for the given duration.
**/
```

```
extern void Play_Note(float freq , float dur);

/**
Get the tempo value for the selected Alarm
Input will take the song number and return the assigned tempo
**/
extern int tempochoose(int number);

/**
play function
This will take one input. The song number.
It will take tempo and the chords and duration data from melodies.h
    accordingly
Using Play_Note it will generate sound
**/
extern void play(int number);

/**
Declare input,output pins and interrupt pin methods
**/
extern void init_buzzer();

#endif /* BUZZER_H_ */
```

Algorithm 6.4: Buzzer.h

```
/*
 * Buzzer.cpp
 *
 * Created: 14/06/2021 16:32:48
 *  Author: Nuwan Udara
 */
#define F_CPU  1000000UL //problem with Proteus simulation, so currently
    using 1MHz F_CPU //#define F_CPU  16000000UL
#include <avr/io.h>
#define __DELAY_BACKWARD_COMPATIBLE__    // This is requaired to make
    _delay_ms() take vairables
#include <util/delay.h>          // for delay functions
#include <avr/interrupt.h>          // for clocks and interrupts
#include <avr/pgmspace.h>        //for PROGMEM
#include "pitches.h"          // Pitch data is stored in this
#include "melodies.h"        // Chords and duration data and tempo data
    are stored in here
#include "Buzzer.h"           // The Include File for the main

 //Buzzer declaration
 #define buzzer_DDR DDRD
 #define buzzer_PORT PORTD
 #define buzzer_PIN PORTD2

 // interrupt button declaration
 #define interruptDDR DDRD

 volatile int sp = 1;    // This is a reference to stop the alarm with a
    button press

 void init_buzzer(){
   // pins ports declaration
   buzzer_DDR |= _BV(buzzer_PIN) ;      // make buzzer pin output

   EICRA |= _BV(ISC11) | _BV(ISC10);      // set rising edge method
   EIMSK |= _BV(INT1);           // enable pin PD1 External interrupt

```

```
  sei();                    //  this Enables the Global Interrupt Enable (I-bit) in the Status Register (SREG)

}

void Play_Note(float freq,float dur){
  // variables
  long int i,cycles;
  float half_period,period;

  if (freq != 0){
    period=(1/freq)*1000;
    cycles=dur/period;
    half_period = period/2;

    for (i=0;i<cycles;i++)
    {
      //50% duty cycle
      _delay_ms(half_period);
      buzzer_PORT |= _BV(buzzer_PIN);
      _delay_ms(half_period);
      buzzer_PORT &= ~ _BV(buzzer_PIN);
    }
  }
  else {
    _delay_ms(dur);   // rest
  }
  return;
}

void play(int number){
  sp=1;
  int tempo=tempochoose(number); // choose the right tempo from the list using the function
  int notes = sizeof(melody0[number])/sizeof(melody0[number][0])/2;
  int wholenote = (60000 * 2.5) / tempo;
  int divider = 0;
  int noteDuration = 0;

  for (int thisNote = 0; thisNote < notes *2 ; thisNote = thisNote + 2)
  {
    if (sp==1){
      divider = pgm_read_word_near(melody0[number]+thisNote + 1);

      if (divider > 0) { //positive divider ( positive duration)
        // it is a regular note
      noteDuration = (wholenote) / divider;}

      else if (divider<0){
        noteDuration=(wholenote)/(divider);// negative durations need to be increased and made positive
        noteDuration=noteDuration*1.5;
        noteDuration=-noteDuration;
      }
      else if(divider==0){return;} // this will be used to stop playing
      Play_Note((pgm_read_word_near(melody0[number]+thisNote)), noteDuration*0.9 ); // play the selected pitch
      _delay_ms(noteDuration*0.5);
    }
    else{
      sp=0;
      break;
```

```
    }
  }
}

/** External interrupt for INT0 (PD2) pin ISR function
We will use this to stop currently playing melody **/

ISR (INT1_vect)          //External interrupt Stop button ISR
{
  sp=0;
  _delay_ms(100);
}

int tempochoose(int number){
  if (number==0){return tempos[0];}
  else if (number==1){return tempos[1];}
  else if (number==2){return tempos[2];}
  else{return tempos[3];}
}
```

Algorithm 6.5: Buzzer.ccp

```
#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
```

```
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
#define REST      0
```

Algorithm 6.6: pitches.h

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

const int tempos[] PROGMEM={225,120,85,120,80};

const int melody0[5][400] PROGMEM= {{       //DOOM music
  NOTE_E2, 8, NOTE_E2, 8, NOTE_E3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_D3, 8,
     NOTE_E2, 8, NOTE_E2, 8, //1
  NOTE_C3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_AS2, 8, NOTE_E2, 8, NOTE_E2,
    8, NOTE_B2, 8, NOTE_C3, 8,
  NOTE_E2, 8, NOTE_E2, 8, NOTE_E3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_D3, 8,
     NOTE_E2, 8, NOTE_E2, 8,
```

```
  NOTE_C3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_AS2, -2,

  NOTE_E2, 8, NOTE_E2, 8, NOTE_E3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_D3, 8, NOTE_E2, 8, NOTE_E2, 8, //5
  NOTE_C3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_AS2, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_B2, 8, NOTE_C3, 8,
  NOTE_E2, 8, NOTE_E2, 8, NOTE_E3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_D3, 8, NOTE_E2, 8, NOTE_E2, 8,
  NOTE_C3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_AS2, -2,

  NOTE_E2, 8, NOTE_E2, 8, NOTE_E3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_D3, 8, NOTE_E2, 8, NOTE_E2, 8, //9
  NOTE_C3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_AS2, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_B2, 8, NOTE_C3, 8,
  NOTE_E2, 8, NOTE_E2, 8, NOTE_E3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_D3, 8, NOTE_E2, 8, NOTE_E2, 8,
  NOTE_C3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_AS2, -2,

  NOTE_E2, 8, NOTE_E2, 8, NOTE_E3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_D3, 8, NOTE_E2, 8, NOTE_E2, 8, //13
  NOTE_C3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_AS2, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_B2, 8, NOTE_C3, 8,
  NOTE_E2, 8, NOTE_E2, 8, NOTE_E3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_D3, 8, NOTE_E2, 8, NOTE_E2, 8,
  NOTE_FS3, -16, NOTE_D3, -16, NOTE_B2, -16, NOTE_A3, -16, NOTE_FS3, -16, NOTE_B2, -16, NOTE_D3, -16, NOTE_FS3, -16, NOTE_A3, -16, NOTE_FS3, -16, NOTE_D3, -16, NOTE_B2, -16,

},
{   // imperail march
   NOTE_A4,4, NOTE_A4,4, NOTE_A4,4, NOTE_F4,-8, NOTE_C5,16,

  NOTE_A4,4, NOTE_F4,-8, NOTE_C5,16, NOTE_A4,2,//4
  NOTE_E5,4, NOTE_E5,4, NOTE_E5,4, NOTE_F5,-8, NOTE_C5,16,
  NOTE_A4,4, NOTE_F4,-8, NOTE_C5,16, NOTE_A4,2,

  NOTE_A5,4, NOTE_A4,-8, NOTE_A4,16, NOTE_A5,4, NOTE_GS5,-8, NOTE_G5,16, //7
  NOTE_DS5,16, NOTE_D5,16, NOTE_DS5,8, REST,8, NOTE_A4,8, NOTE_DS5,4, NOTE_D5,-8, NOTE_CS5,16,

  NOTE_C5,16, NOTE_B4,16, NOTE_C5,16, REST,8, NOTE_F4,8, NOTE_GS4,4, NOTE_F4,-8, NOTE_A4,-16,//9
  NOTE_C5,4, NOTE_A4,-8, NOTE_C5,16, NOTE_E5,2,

  NOTE_A5,4, NOTE_A4,-8, NOTE_A4,16, NOTE_A5,4, NOTE_GS5,-8, NOTE_G5,16, //7
  NOTE_DS5,16, NOTE_D5,16, NOTE_DS5,8, REST,8, NOTE_A4,8, NOTE_DS5,4, NOTE_D5,-8, NOTE_CS5,16,

  NOTE_C5,16, NOTE_B4,16, NOTE_C5,16, REST,8, NOTE_F4,8, NOTE_GS4,4, NOTE_F4,-8, NOTE_A4,-16,//9
  NOTE_A4,4, NOTE_F4,-8, NOTE_C5,16, NOTE_A4,2,

},
{ // Game of throne
  NOTE_G4,8, NOTE_C4,8, NOTE_DS4,16, NOTE_F4,16, NOTE_G4,8, NOTE_C4,8, NOTE_DS4,16, NOTE_F4,16, //1
  NOTE_G4,8, NOTE_C4,8, NOTE_DS4,16, NOTE_F4,16, NOTE_G4,8, NOTE_C4,8, NOTE_DS4,16, NOTE_F4,16,
  NOTE_G4,8, NOTE_C4,8, NOTE_E4,16, NOTE_F4,16, NOTE_G4,8, NOTE_C4,8, NOTE_E4,16, NOTE_F4,16,
```

```
  NOTE_G4,8, NOTE_C4,8, NOTE_E4,16, NOTE_F4,16, NOTE_G4,8, NOTE_C4,8,
    NOTE_E4,16, NOTE_F4,16,
  NOTE_G4,-4, NOTE_C4,-4,//5

  NOTE_DS4,16, NOTE_F4,16, NOTE_G4,4, NOTE_C4,4, NOTE_DS4,16, NOTE_F4,16,
    //6
  NOTE_D4,-1, //7 and 8
  NOTE_F4,-4, NOTE_AS3,-4,
  NOTE_DS4,16, NOTE_D4,16, NOTE_F4,4, NOTE_AS3,-4,
  NOTE_DS4,16, NOTE_D4,16, NOTE_C4,-1, //11 and 12

  //repeats from 5
  NOTE_G4,-4, NOTE_C4,-4,//5

  NOTE_DS4,16, NOTE_F4,16, NOTE_G4,4, NOTE_C4,4, NOTE_DS4,16, NOTE_F4,16,
    //6
  NOTE_D4,-1, //7 and 8
  NOTE_F4,-4, NOTE_AS3,-4,
  NOTE_DS4,16, NOTE_D4,16, NOTE_F4,4, NOTE_AS3,-4,
  NOTE_DS4,16, NOTE_D4,16, NOTE_C4,-1, //11 and 12
  NOTE_G4,-4, NOTE_C4,-4,
  NOTE_DS4,16, NOTE_F4,16, NOTE_G4,4,  NOTE_C4,4, NOTE_DS4,16, NOTE_F4
    ,16,

  NOTE_D4,-2,//15
  NOTE_F4,-4, NOTE_AS3,-4,
  NOTE_D4,-8, NOTE_DS4,-8, NOTE_D4,-8, NOTE_AS3,-8,
  NOTE_C4,-1,
  NOTE_C5,-2,
  NOTE_AS4,-2,
  NOTE_C4,-2,
  NOTE_G4,-2,
  NOTE_DS4,-2,
  NOTE_DS4,-4, NOTE_F4,-4,
  NOTE_G4,-1,

  NOTE_C5,-2,//28
  NOTE_AS4,-2,
  NOTE_C4,-2,
  NOTE_G4,-2,
  NOTE_DS4,-2,
  NOTE_DS4,-4, NOTE_D4,-4,
  NOTE_C5,8, NOTE_G4,8, NOTE_GS4,16, NOTE_AS4,16, NOTE_C5,8, NOTE_G4,8,
    NOTE_GS4,16, NOTE_AS4,16,
  NOTE_C5,8, NOTE_G4,8, NOTE_GS4,16, NOTE_AS4,16, NOTE_C5,8, NOTE_G4,8,
    NOTE_GS4,16, NOTE_AS4,16,
  REST,4, NOTE_GS5,16, NOTE_AS5,16, NOTE_C6,8, NOTE_G5,8, NOTE_GS5,16,
    NOTE_AS5,16,
  NOTE_C6,8, NOTE_G5,16, NOTE_GS5,16, NOTE_AS5,16, NOTE_C6,8, NOTE_G5,8,
    NOTE_GS5,16, NOTE_AS5,16,
},
{ //pink panther
  NOTE_E4,-4, REST,8, NOTE_FS4,8, NOTE_G4,-4, REST,8, NOTE_DS4,8,
  NOTE_E4,-8, NOTE_FS4,8,  NOTE_G4,-8, NOTE_C5,8, NOTE_B4,-8, NOTE_E4,8,
    NOTE_G4,-8, NOTE_B4,8,
  NOTE_AS4,2, NOTE_A4,-16, NOTE_G4,-16, NOTE_E4,-16, NOTE_D4,-16,
  NOTE_E4,2, REST,4, REST,8, NOTE_DS4,4,

  NOTE_E4,-4, REST,8, NOTE_FS4,8, NOTE_G4,-4, REST,8, NOTE_DS4,8,
  NOTE_E4,-8, NOTE_FS4,8,  NOTE_G4,-8, NOTE_C5,8, NOTE_B4,-8, NOTE_G4,8,
    NOTE_B4,-8, NOTE_E5,8,
  NOTE_DS5,1,
```

```
  NOTE_D5,2, REST,4, REST,8, NOTE_DS4,8,
  NOTE_E4,-4, REST,8, NOTE_FS4,8, NOTE_G4,-4, REST,8, NOTE_DS4,8,
  NOTE_E4,-8, NOTE_FS4,8,  NOTE_G4,-8, NOTE_C5,8, NOTE_B4,-8, NOTE_E4,8,
    NOTE_G4,-8, NOTE_B4,8,

  NOTE_AS4,2, NOTE_A4,-16, NOTE_G4,-16, NOTE_E4,-16, NOTE_D4,-16,
  NOTE_E4,-4, REST,4,
  REST,4, NOTE_E5,-8, NOTE_D5,8, NOTE_B4,-8, NOTE_A4,8, NOTE_G4,-8,
    NOTE_E4,-8,
  NOTE_AS4,16, NOTE_A4,-8, NOTE_AS4,16, NOTE_A4,-8, NOTE_AS4,16, NOTE_A4
    ,-8, NOTE_AS4,16, NOTE_A4,-8,
  NOTE_G4,-16, NOTE_E4,-16, NOTE_D4,-16, NOTE_E4,16, NOTE_E4,16, NOTE_E4
    ,2,

},
{ // GODFATHER theme
    REST, 4, REST, 8, REST, 8, REST, 8, NOTE_E4, 8, NOTE_A4, 8, NOTE_C5,
    8, //1
    NOTE_B4, 8, NOTE_A4, 8, NOTE_C5, 8, NOTE_A4, 8, NOTE_B4, 8, NOTE_A4,
    8, NOTE_F4, 8, NOTE_G4, 8,
    NOTE_E4, 2, NOTE_E4, 8, NOTE_A4, 8, NOTE_C5, 8,
    NOTE_B4, 8, NOTE_A4, 8, NOTE_C5, 8, NOTE_A4, 8, NOTE_C5, 8, NOTE_A4,
    8, NOTE_E4, 8, NOTE_DS4, 8,

    NOTE_D4, 2, NOTE_D4, 8, NOTE_F4, 8, NOTE_GS4, 8, //5
    NOTE_B4, 2, NOTE_D4, 8, NOTE_F4, 8, NOTE_GS4, 8,
    NOTE_A4, 2, NOTE_C4, 8, NOTE_C4, 8, NOTE_G4, 8,
    NOTE_F4, 8, NOTE_E4, 8, NOTE_G4, 8, NOTE_F4, 8, NOTE_F4, 8, NOTE_E4,
    8, NOTE_E4, 8, NOTE_GS4, 8,

    NOTE_A4, 2, REST,8, NOTE_A4, 8, NOTE_A4, 8, NOTE_GS4, 8, //9
    NOTE_G4, 2, NOTE_B4, 8, NOTE_A4, 8, NOTE_F4, 8,
    NOTE_E4, 2, NOTE_E4, 8, NOTE_G4, 8, NOTE_E4, 8,
    NOTE_D4, 2, NOTE_D4, 8, NOTE_D4, 8, NOTE_F4, 8, NOTE_DS4, 8,

    NOTE_E4, 2, REST, 8, NOTE_E4, 8, NOTE_A4, 8, NOTE_C5, 8, //13

    //repeats from 2
    NOTE_B4, 8, NOTE_A4, 8, NOTE_C5, 8, NOTE_A4, 8, NOTE_B4, 8, NOTE_A4,
    8, NOTE_F4, 8, NOTE_G4, 8, //2
    NOTE_E4, 2, NOTE_E4, 8, NOTE_A4, 8, NOTE_C5, 8,
    NOTE_B4, 8, NOTE_A4, 8, NOTE_C5, 8, NOTE_A4, 8, NOTE_C5, 8, NOTE_A4,
    8, NOTE_E4, 8, NOTE_DS4, 8,

    NOTE_D4, 2, NOTE_D4, 8, NOTE_F4, 8, NOTE_GS4, 8, //5
    NOTE_B4, 2, NOTE_D4, 8, NOTE_F4, 8, NOTE_GS4, 8,
    NOTE_A4, 2, NOTE_C4, 8, NOTE_C4, 8, NOTE_G4, 8,
    NOTE_F4, 8, NOTE_E4, 8, NOTE_G4, 8, NOTE_F4, 8, NOTE_F4, 8, NOTE_E4,
    8, NOTE_E4, 8, NOTE_GS4, 8,

    NOTE_A4, 2, REST,8, NOTE_A4, 8, NOTE_A4, 8, NOTE_GS4, 8, //9
    NOTE_G4, 2, NOTE_B4, 8, NOTE_A4, 8, NOTE_F4, 8,
    NOTE_E4, 2, NOTE_E4, 8, NOTE_G4, 8, NOTE_E4, 8,
    NOTE_D4, 2, NOTE_D4, 8, NOTE_D4, 8, NOTE_F4, 8, NOTE_DS4, 8,

    NOTE_E4, 2 //13

}
};
```

Algorithm 6.7: melodies.h

```
/*
* Display.h
*
* Created: 6/11/2021 6:58:37 PM
* Author: Dakshina Tharidndu
*/


#ifndef __DISPLAY_H__
#define __DISPLAY_H__


class Display
{
//variables
public:
protected:
private:

//functions
public:
  Display();
  ~Display();
protected:
private:
  Display( const Display &c );
  Display& operator=( const Display &c );

}; //Display
extern void LCD_Command( unsigned char cmnd );
extern void LCD_Char( unsigned char data );
extern void LCD_Init (void);
extern void LCD_String (char *str);
extern void LCD_String_xy (char row, char pos, char *str);
extern void LCD_Clear();
extern void LCD_Home(int r);
extern void LCD_Menu(char key,int state);
extern void LCD_SetAlarm(char key,int state);
extern void LCD_SetDate(int key, int state);
extern void LCD_Tone(char key,int state);
extern void resetVariables();
extern void showAlarms(char key,int state);
extern void displayTyping(int val,int pos);
extern void displayValue(int val);
#endif //__DISPLAY_H__
```

Algorithm 6.8: Display.h

```

#ifndef F_CPU
#define F_CPU 1000000UL
#endif
#include <avr/io.h>     /* Include AVR std. library file */
#include <util/delay.h>
#define BUTTON_DELAY 500
#include "Display.h"
#include "Keypad.h"
#include "stdlib.h"
#include "Alarm.h"
#define LCD_Dir  DDRD     /* Define LCD data port direction */
#define LCD_Port PORTD      /* Define LCD data port */
#define RS PB0        /* Define Register Select pin */
#define EN PB1        /* Define Enable signal pin */
```

```
16 // default constructor
17 Display::Display()
18 {
19 } //Display
20
21 // default destructor
22 Display::~Display()
23 {
24 } //~Display
25 void LCD_Command( unsigned char cmnd )
26 {
27   LCD_Port = (LCD_Port & 0x0F) | (cmnd & 0xF0); /* sending upper nibble
       */
28   LCD_Port &= ~ (1<<RS);    /* RS=0, command reg. */
29   LCD_Port |= (1<<EN);    /* Enable pulse */
30   _delay_us(1);
31   LCD_Port &= ~ (1<<EN);
32
33   _delay_us(200);
34
35   LCD_Port = (LCD_Port & 0x0F) | (cmnd << 4);  /* sending lower nibble */
36   LCD_Port |= (1<<EN);
37   _delay_us(1);
38   LCD_Port &= ~ (1<<EN);
39   _delay_ms(2);
40 }
41
42
43 void LCD_Char( unsigned char data )
44 {
45   LCD_Port = (LCD_Port & 0x0F) | (data & 0xF0); /* sending upper nibble
       */
46   LCD_Port |= (1<<RS);    /* RS=1, data reg. */
47   LCD_Port|= (1<<EN);
48   _delay_us(1);
49   LCD_Port &= ~ (1<<EN);
50
51   _delay_us(200);
52
53   LCD_Port = (LCD_Port & 0x0F) | (data << 4); /* sending lower nibble */
54   LCD_Port |= (1<<EN);
55   _delay_us(1);
56   LCD_Port &= ~ (1<<EN);
57   _delay_ms(2);
58 }
59
60 void LCD_Init (void)      /* LCD Initialize function */
61 {
62   LCD_Dir = 0xFF;     /* Make LCD port direction as o/p */
63   _delay_ms(20);      /* LCD Power ON delay always >15ms */
64
65   LCD_Command(0x02);    /* send for 4 bit initialization of LCD  */
66   LCD_Command(0x28);              /* 2 line, 5*7 matrix in 4-bit mode */
67   LCD_Command(0x0c);              /* Display on cursor off*/
68   LCD_Command(0x06);              /* Increment cursor (shift cursor to
       right)*/
69   LCD_Command(0x01);              /* Clear display screen*/
70   _delay_ms(2);
71 }
72
73 //char temp1='/';
74 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 (char row, char pos, char *str)  /* Send string to LCD
     with xy position */
{
  if (row == 0 && pos<16)
  LCD_Command((pos & 0x0F)|0x80); /* Command of first row and required
    position<16 */
  else if (row == 1 && pos<16)
  LCD_Command((pos & 0x0F)|0xC0); /* Command of first row and required
    position<16 */
  LCD_String(str);    /* Call LCD string function */
}

void LCD_Clear()
{
  LCD_Command (0x01);   /* Clear display */
  _delay_ms(2);
  LCD_Command (0x80);   /* Cursor at home position */
}
void LCD_Home(int r){
  if(r==0){
    LCD_Command (0x80);
  }
  else{
    LCD_Command (0xC0);
  }
}
int blink_var=0;
void LCD_Blink(char ch[10]){
  if (blink_var==0){
    LCD_String(ch);
    blink_var=1;
    _delay_ms(100);
  }
  else{
    LCD_String(" ");
    blink_var=0;
    _delay_ms(100);
  }
}
void displayTyping(int val,int pos){
  char val0S[2];char val1S[2];
  itoa(val/10,val0S,10);itoa(val%10,val1S,10);
  if (pos==0){
    LCD_Blink(val0S);
    LCD_String(val1S);
  }
  else if (pos==1){
    LCD_String(val0S);
    LCD_Blink(val1S);
  }
  else{
    LCD_String(val0S);LCD_String(val1S);
  }
```

```
}
int clkTime = 0;
int num = 0;
signed int p = 3;
void LCD_SetAlarm(char key,int state){
  if (p!=-1){
    LCD_Home(0);
    LCD_String("     ");
    displayTyping(clkTime/100,3-p);
    LCD_String(":");
    displayTyping(clkTime%100,1-p);
  }
  //LCD_String("0000");
  if ((int(key)>47) & (int(key)<58) & (state ==2) & (p!=-1)){
    clkTime += (int(key)-48)*powerOf(10,p);
    p--;
    _delay_ms(BUTTON_DELAY);
    if (p==-1){
      LCD_Home(0);
      LCD_String("     ");
      displayTyping(clkTime/100,3);LCD_String(":");displayTyping(clkTime
    %100,3);
      _delay_ms(1000);
      LCD_Home(0);
      LCD_String("SELECT THE TONE");
      _delay_ms(2000);
      LCD_Clear();
    }
  }
  if (p==-1){
    LCD_Tone(key,state);
  }
  if (state==3){
    setAlarm(clkTime);
    LCD_Home(0);
    LCD_String("ALARM IS SET TO");
    LCD_Home(1);
    displayTyping(clkTime/100,3);LCD_String(":");displayTyping(clkTime
    %100,3);
    clkTime=0;num=0;p=3;
    _delay_ms(2000);
    LCD_Clear();
  }
}
int clockTime[6];
int ClockState = 0;
int temp = 0;
int data = 0;
char dataS[10];
void LCD_SetDate(int key, int state){
  LCD_Home(0);
  LCD_String("   20");
  for (int i=0;i<6;i++){
    if (i==3){
      LCD_Home(1);
      LCD_String("    ");
    }
    if ((i!=3) & (i!=0)){
      LCD_String(":");
    }
    if (i==ClockState){
      displayTyping(clockTime[i],temp);
```

```
    }
    else{
      displayTyping(clockTime[i],3);
    }
  }
  if ((int(key)>47) & (int(key)<58) & (state ==2)){
    data+=(int(key)-48);
    _delay_ms(BUTTON_DELAY);
    if (temp == 0){
      data*=10;
      temp = 1;
      clockTime[ClockState] = data;
    }
    else{
      clockTime[ClockState] = data;
      ClockState++;
      data=0;
      temp = 0;
    }
  }
  if ((ClockState == 6) & (state==3)){
    setClockTime(clockTime);
    ClockState = 0;
    LCD_Home(0);LCD_Clear();
    LCD_String("      DONE!");
    _delay_ms(2000);
    LCD_Clear();
  }
}
char tone_List[5][10] = {"DOOM     ","STAR WARS","GOT      ","PANTHER  ",
    "GODFATHER"};
int tone_Var = 0;
void LCD_Tone(char key,int state){
  LCD_Home(0);
  if ((key=='8') & (state==2)){
    tone_Var++;
    //LCD_Clear();
    _delay_ms(BUTTON_DELAY);
  }
  if ((key=='2') & (state ==2)){
    tone_Var--;
    //LCD_Clear();
    _delay_ms(BUTTON_DELAY);
  }
  if (tone_Var>4){
    tone_Var=0;
  }
  if (tone_Var<0){
    tone_Var=4;
  }
  if (state==2){
    LCD_Home(0);LCD_String(">>");
    LCD_String(tone_List[tone_Var%5]);
    //  LCD_Home(1);
    //  LCD_String(menu_List[(menu_Var+1)%3]);
  }
   if (state==3){
     setTone(tone_Var);
   }

}
char menu_List[4][11] = {"SET ALARM ","SET TIME  ","SEE ALARMS","TIMER
```

```
      "};
int menu_Var = 0;
void LCD_Menu(char key,int state){
  LCD_Home(0);
  if ((key=='8') & (state==1)){
    menu_Var++;
    //LCD_Clear();
    _delay_ms(BUTTON_DELAY);
  }
  if ((key=='2') & (state ==1)){
    menu_Var--;
    //LCD_Clear();
    _delay_ms(BUTTON_DELAY);
  }
  if (menu_Var>3){
    menu_Var=0;
  }
  if (menu_Var<0){
    menu_Var=3;
  }
  if (state==1){
    LCD_Home(0);LCD_String(">>");
    LCD_String(menu_List[menu_Var%4]);
    resetVariables();
  //  LCD_Home(1);
  //  LCD_String(menu_List[(menu_Var+1)%3]);
  }
  if ((state >1) & (menu_Var==0)){
    LCD_SetAlarm(key,state);
  }
  if ((state >1) & (menu_Var==1)){
    LCD_SetDate(key,state);
  }
  if ((state >1) & (menu_Var==2)){
    showAlarms(key,state);
  }
  if ((state >1) & (menu_Var==3)){
    timer(key,state);
  }
}
void resetVariables(){
  resetAlarmVariables();
  tone_Var=0;
  clockTime[0]=0;clockTime[1]=0;clockTime[2]=0;clockTime[3]=0;clockTime
    [4]=0;clockTime[5]=0;
  ClockState = 0;
  temp = 0;
  data = 0;
  clkTime = 0;
  num = 0;
  p = 3;
}
```

Algorithm 6.9: Display.cpp

```
/*
* Keypad.h
*
* Created: 6/13/2021 11:40:03 AM
* Author: Dakshina Tharidndu
*/


```

```
#ifndef __KEYPAD_H__
#define __KEYPAD_H__


class Keypad
{
//variables
public:
protected:
private:

//functions
public:
  Keypad();
  ~Keypad();
protected:
private:
  Keypad( const Keypad &c );
  Keypad& operator=( const Keypad &c );

}; //Keypad
extern char btnPress();
#endif //__KEYPAD_H__
```

Algorithm 6.10: Keypad.h

```

#include "Keypad.h"
#include <avr/io.h>
#include <util/delay.h>
#define keyDir DDRC
#define keyPort PORTC
#define keyPin PINB
char keys[4][3] = {{'1','2','3'},{'4','5','6'},{'7','8','9'},{'*','0','#'}};
uint8_t row[4] = {PORTC0,PORTC1,PORTC2,PORTC3};
uint8_t colomn[3] = {PINB0,PINB1,PINB2};
// default constructor
Keypad::Keypad()
{
} //Keypad

// default destructor
Keypad::~Keypad()
{
} //~Keypad


char btnPress(){
  keyDir = 0x0F;
  for (int i=0;i<4;i++){
    keyPort = 1<<row[i];
    for (int j = 0;j<3;j++){
      if (keyPin==(1<<colomn[j])){
        //_delay_ms(100);
        return keys[i][j];
      }
    }
  }
  return '&';
}
```

Algorithm 6.11: Keypad.cpp

```
#include <avr/io.h>
extern uint8_t ds1307_dec2bcd(uint8_t val);
extern uint8_t ds1307_bcd2dec(uint8_t val);
extern void ds1307_init();
extern void ds1307_setdate(uint8_t year, uint8_t month, uint8_t day,
    uint8_t dayofweek, uint8_t hour, uint8_t minute, uint8_t second);
extern void ds1307_getdate(uint8_t *year, uint8_t *month, uint8_t *day,
    uint8_t *dayofweek, uint8_t *hour, uint8_t *minute, uint8_t *second);
```

Algorithm 6.12: ds1307.h

```
#ifndef DS1307_H
#define DS1307_H

#include <avr/io.h>
#include "ds1307.h"
#include "i2cmaster.h"
#include "util/delay.h"
#define DS1307_ADDRESS 0xD0

#ifndef F_CPU
#define F_CPU 1000000UL
#endif


uint8_t ds1307_dec2bcd(uint8_t val) {
  return val + 6 * (val / 10);
}
uint8_t ds1307_bcd2dec(uint8_t val) {
  return val - 6 * (val >> 4);
}
void ds1307_init() {
  i2c_init();
  _delay_us(10);
}
void ds1307_setdate(uint8_t year, uint8_t month, uint8_t day, uint8_t
    dayofweek, uint8_t hour, uint8_t minute, uint8_t second) {


  i2c_start_wait(DS1307_ADDRESS | I2C_WRITE);
  i2c_write(0x00);
  i2c_write(ds1307_dec2bcd(second));
  i2c_write(ds1307_dec2bcd(minute));
  i2c_write(ds1307_dec2bcd(hour));
  i2c_write(ds1307_dec2bcd(dayofweek));
  i2c_write(ds1307_dec2bcd(day));
  i2c_write(ds1307_dec2bcd(month));
  i2c_write(ds1307_dec2bcd(year));
  i2c_write(0x00);
  i2c_stop();
}
void ds1307_getdate(uint8_t *year, uint8_t *month, uint8_t *day,uint8_t *
    dayofweek, uint8_t *hour, uint8_t *minute, uint8_t *second) {
  i2c_start_wait(DS1307_ADDRESS | I2C_WRITE);
  i2c_write(0x00);
  i2c_stop();

  i2c_rep_start(DS1307_ADDRESS | I2C_READ);
  *second = ds1307_bcd2dec(i2c_readAck() & 0x7F);
  *minute = ds1307_bcd2dec(i2c_readAck());
  *hour = ds1307_bcd2dec(i2c_readAck());
  *dayofweek = ds1307_bcd2dec(i2c_readAck());
  *day = ds1307_bcd2dec(i2c_readAck());
```

```
  *month = ds1307_bcd2dec(i2c_readAck());
  *year = ds1307_bcd2dec(i2c_readNak());
  i2c_stop();
}
#endif
```

Algorithm 6.13: ds1307.cpp

```
#ifndef _I2CMASTER_H
#define _I2CMASTER_H   1
/*
    ************************************************************************

* Title:    C include file for the I2C master interface
*           (i2cmaster.S or twimaster.c)
* Author:   Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
* File:     $Id: i2cmaster.h,v 1.10 2005/03/06 22:39:57 Peter Exp $
* Software: AVR-GCC 3.4.3 / avr-libc 1.2.3
* Target:   any AVR device
* Usage:    see Doxygen manual
************************************************************************
    */

#ifdef DOXYGEN
/**
 @defgroup pfleury_ic2master I2C Master library
 @code #include <i2cmaster.h> @endcode

 @brief I2C (TWI) Master Software Library

 Basic routines for communicating with I2C slave devices. This single
    master
 implementation is limited to one bus master on the I2C bus.

 This I2c library is implemented as a compact assembler software
    implementation of the I2C protocol
 which runs on any AVR (i2cmaster.S) and as a TWI hardware interface for
    all AVR with built-in TWI hardware (twimaster.c).
 Since the API for these two implementations is exactly the same, an
    application can be linked either against the
 software I2C implementation or the hardware I2C implementation.

 Use 4.7k pull-up resistor on the SDA and SCL pin.

 Adapt the SCL and SDA port and pin definitions and eventually the delay
    routine in the module
 i2cmaster.S to your target when using the software I2C implementation !

 Adjust the  CPU clock frequence F_CPU in twimaster.c or in the Makfile
    when using the TWI hardware implementaion.

 @note
    The module i2cmaster.S is based on the Atmel Application Note AVR300,
     corrected and adapted
    to GNU assembler and AVR-GCC C call interface.
    Replaced the incorrect quarter period delays found in AVR300 with
    half period delays.

 @author Peter Fleury pfleury@gmx.ch  http://jump.to/fleury

 @par API Usage Example
  The following code shows typical usage of this library, see example
    test_i2cmaster.c
```

```
45
46  @code
47
48  #include <i2cmaster.h>
49
50
51  #define Dev24C02  0xA2      // device address of EEPROM 24C02, see
       datasheet
52
53  int main(void)
54  {
55      unsigned char ret;
56
57      i2c_init();                             // initialize I2C library
58
59      // write 0x75 to EEPROM address 5 (Byte Write)
60      i2c_start_wait(Dev24C02+I2C_WRITE);     // set device address and
       write mode
61      i2c_write(0x05);                        // write address = 5
62      i2c_write(0x75);                        // write value 0x75 to
       EEPROM
63      i2c_stop();                             // set stop conditon =
       release bus
64
65
66      // read previously written value back from EEPROM address 5
67      i2c_start_wait(Dev24C02+I2C_WRITE);     // set device address and
       write mode
68
69      i2c_write(0x05);                        // write address = 5
70      i2c_rep_start(Dev24C02+I2C_READ);       // set device address and
       read mode
71
72      ret = i2c_readNak();                    // read one byte from EEPROM
73      i2c_stop();
74
75      for(;;);
76  }
77  @endcode
78
79  */
80  #endif /* DOXYGEN */
81
82  /**@{*/
83
84  #if (__GNUC__ * 100 + __GNUC_MINOR__) < 304
85  #error "This library requires AVR-GCC 3.4 or later, update to newer AVR-
       GCC compiler !"
86  #endif
87
88  #include <avr/io.h>
89
90  /** defines the data direction (reading from I2C device) in i2c_start(),
       i2c_rep_start() */
91  #define I2C_READ    1
92
93  /** defines the data direction (writing to I2C device) in i2c_start(),
       i2c_rep_start() */
94  #define I2C_WRITE   0
95
96
97  /**
```

```
98  @brief initialize the I2C master interace. Need to be called only once
99  @param  void
100 @return none
101 */
102 extern void i2c_init(void);
103
104
105 /**
106  @brief Terminates the data transfer and releases the I2C bus
107  @param void
108  @return none
109  */
110 extern void i2c_stop(void);
111
112
113 /**
114  @brief Issues a start condition and sends address and transfer direction
115
116  @param    addr address and transfer direction of I2C device
117  @retval   0   device accessible
118  @retval   1   failed to access device
119  */
120 extern unsigned char i2c_start(unsigned char addr);
121
122
123 /**
124  @brief Issues a repeated start condition and sends address and transfer
       direction
125
126  @param   addr address and transfer direction of I2C device
127  @retval  0 device accessible
128  @retval  1 failed to access device
129  */
130 extern unsigned char i2c_rep_start(unsigned char addr);
131
132
133 /**
134  @brief Issues a start condition and sends address and transfer direction
135
136  If device is busy, use ack polling to wait until device ready
137  @param    addr address and transfer direction of I2C device
138  @return   none
139  */
140 extern void i2c_start_wait(unsigned char addr);
141
142
143 /**
144  @brief Send one byte to I2C device
145  @param    data  byte to be transfered
146  @retval   0 write successful
147  @retval   1 write failed
148  */
149 extern unsigned char i2c_write(unsigned char data);
150
151
152 /**
153  @brief    read one byte from the I2C device, request more data from
       device
154  @return   byte read from I2C device
155  */
156 extern unsigned char i2c_readAck(void);
157
```

```
158 /**
159  @brief    read one byte from the I2C device, read is followed by a stop
       condition
160  @return   byte read from I2C device
161  */
162 extern unsigned char i2c_readNak(void);
163 
164 /**
165  @brief    read one byte from the I2C device
166 
167  Implemented as a macro, which calls either i2c_readAck or i2c_readNak
168 
169  @param    ack 1 send ack, request more data from device<br>
170               0 send nak, read is followed by a stop condition
171  @return   byte read from I2C device
172  */
173 extern unsigned char i2c_read(unsigned char ack);
174 #define i2c_read(ack)  (ack) ? i2c_readAck() : i2c_readNak();
175 
176 
177 /**@}*/
178 #endif
```

Algorithm 6.14: i2cmaster.h

```
1  /*
       ************************************************************************
2  * Title:    I2C master library using hardware TWI interface
3  * Author:   Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
4  * File:     $Id: twimaster.c,v 1.3 2005/07/02 11:14:21 Peter Exp $
5  * Software: AVR-GCC 3.4.3 / avr-libc 1.2.3
6  * Target:   any AVR device with hardware TWI
7  * Usage:    API compatible with I2C Software Library i2cmaster.h
8  ************************************************************************
       */
9  #include <inttypes.h>
10 #include <compat/twi.h>
11 
12 #include "i2cmaster.h"
13 
14 
15 /* define CPU frequency in Mhz here if not defined in Makefile */
16 #ifndef F_CPU
17 #define F_CPU 1000000UL
18 #endif
19 
20 /* I2C clock in Hz */
21 #define SCL_CLOCK  100000L
22 
23 
24 /*
       ************************************************************************
25  Initialization of the I2C bus interface. Need to be called only once
26 ************************************************************************
       */
27 void i2c_init(void)
28 {
29   /* initialize TWI clock: 100 kHz clock, TWPS = 0 => prescaler = 1 */
30 
31   TWSR = 0;                         /* no prescaler */
```

```
  TWBR = ((F_CPU/SCL_CLOCK)-16)/2;  /* must be > 10 for stable operation
    */

}/* i2c_init */


/*
    *************************************************************************
  Issues a start condition and sends address and transfer direction.
  return 0 = device accessible, 1= failed to access device
*************************************************************************
    */
unsigned char i2c_start(unsigned char address)
{
    uint8_t   twst;

  // send START condition
  TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

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

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

  // send device address
  TWDR = address;
  TWCR = (1<<TWINT) | (1<<TWEN);

  // wail until transmission completed and ACK/NACK has been received
  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;

}/* i2c_start */


/*
    *************************************************************************
 Issues a start condition and sends address and transfer direction.
 If device is busy, use ack polling to wait until device is ready

 Input:   address and transfer direction of I2C device
*************************************************************************
    */
void i2c_start_wait(unsigned char address)
{
    uint8_t   twst;


    while ( 1 )
    {
      // send START condition
      TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

```

```
87        // wait until transmission completed
88        while(!(TWCR & (1<<TWINT)));
89
90        // check value of TWI Status Register. Mask prescaler bits.
91        twst = TW_STATUS & 0xF8;
92        if ( (twst != TW_START) && (twst != TW_REP_START)) continue;
93
94        // send device address
95        TWDR = address;
96        TWCR = (1<<TWINT) | (1<<TWEN);
97
98        // wail until transmission completed
99        while(!(TWCR & (1<<TWINT)));
100
101       // check value of TWI Status Register. Mask prescaler bits.
102       twst = TW_STATUS & 0xF8;
103       if ( (twst == TW_MT_SLA_NACK )||(twst ==TW_MR_DATA_NACK) )
104       {
105           /* device busy, send stop condition to terminate write
      operation */
106           TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
107
108           // wait until stop condition is executed and bus released
109           while(TWCR & (1<<TWSTO));
110
111           continue;
112       }
113       //if( twst != TW_MT_SLA_ACK) return 1;
114       break;
115     }
116
117 }/* i2c_start_wait */
118
119
120 /*
      *************************************************************************

121  Issues a repeated start condition and sends address and transfer
      direction
122
123  Input:   address and transfer direction of I2C device
124
125  Return:  0 device accessible
126           1 failed to access device
127 *************************************************************************
      */
128 unsigned char i2c_rep_start(unsigned char address)
129 {
130     return i2c_start( address );
131
132 }/* i2c_rep_start */
133
134
135 /*
      *************************************************************************

136  Terminates the data transfer and releases the I2C bus
137 *************************************************************************
      */
138 void i2c_stop(void)
139 {
140     /* send stop condition */
```

```
141   TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
142 
143   // wait until stop condition is executed and bus released
144   while(TWCR & (1<<TWSTO));
145 
146 }/* i2c_stop */
147 
148 
149 /*
    ************************************************************************

150   Send one byte to I2C device
151 
152   Input:    byte to be transfered
153   Return:   0 write successful
154             1 write failed
155 ************************************************************************
    */
156 unsigned char i2c_write( unsigned char data )
157 {
158     uint8_t   twst;
159 
160   // send data to the previously addressed device
161   TWDR = data;
162   TWCR = (1<<TWINT) | (1<<TWEN);
163 
164   // wait until transmission completed
165   while(!(TWCR & (1<<TWINT)));
166 
167   // check value of TWI Status Register. Mask prescaler bits
168   twst = TW_STATUS & 0xF8;
169   if( twst != TW_MT_DATA_ACK) return 1;
170   return 0;
171 
172 }/* i2c_write */
173 
174 
175 /*
    ************************************************************************

176  Read one byte from the I2C device, request more data from device
177 
178  Return:  byte read from I2C device
179 ************************************************************************
    */
180 unsigned char i2c_readAck(void)
181 {
182   TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
183   while(!(TWCR & (1<<TWINT)));
184 
185     return TWDR;
186 
187 }/* i2c_readAck */
188 
189 
190 /*
    ************************************************************************

191  Read one byte from the I2C device, read is followed by a stop condition
192 
193  Return:  byte read from I2C device
194 ************************************************************************
```

```
   */
unsigned char i2c_readNak(void)
{
  TWCR = (1<<TWINT) | (1<<TWEN);
  while(!(TWCR & (1<<TWINT)));

    return TWDR;

}/* i2c_readNak */
```

Algorithm 6.15: i2cmaster.cpp

Bibliography

[1] P. Fleury, *Interfacing a hd44780 based lcd to an avr*. [Online]. Available: http://www.peterfleury.epizy.com/avr-lcd44780.html (visited on 2021).

[2] R. Couto, *Arduino songs*. [Online]. Available: https://github.com/robsoncouto/arduino-songs (visited on 2021).

[3] *Electrical waveforms and electrical signals.* [Online]. Available: https://www.electronics-tutorials.ws/waveforms/waveforms.html (visited on 2021).

[4] *I/o port operations in avr*. [Online]. Available: https://maxembedded.com/2011/06/port-operations-in-avr/ (visited on 2021).

[5] *4x4 keypad interfacing with atmega32 microcontroller*. [Online]. Available: https://www.electronics-tutorials.ws/waveforms/waveforms.html (visited on 2021).

[6] ATMEL, *Atmega 328p data sheet*. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf (visited on 2021).

[7] XIAMEN, *16x2 lcd module*. [Online]. Available: https://components101.com/sites/default/files/component_datasheet/16x2%5C%20LCD%5C%20Datasheet.pdf (visited on 2021).