פרויקט הגשה ברובוטיקה

שעון מעורר חכם

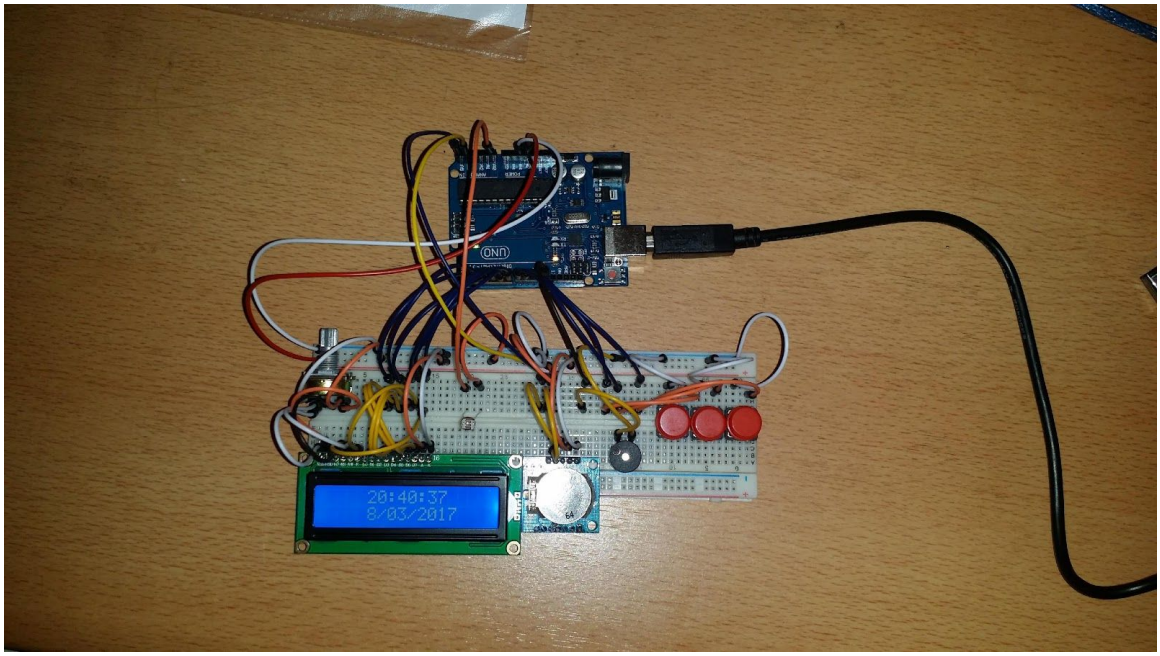
שם הקורס: רובוטיקה למדעי המחשב

מוסד לימודים: המכון הטכנולוגי חולון

מרצה: דר' אליהו מצליח

שם הפרויקט: שעון מעורר חכם

שם המגיש: יעקב סובולב ת.ז 316898162



תוכן עניינים:

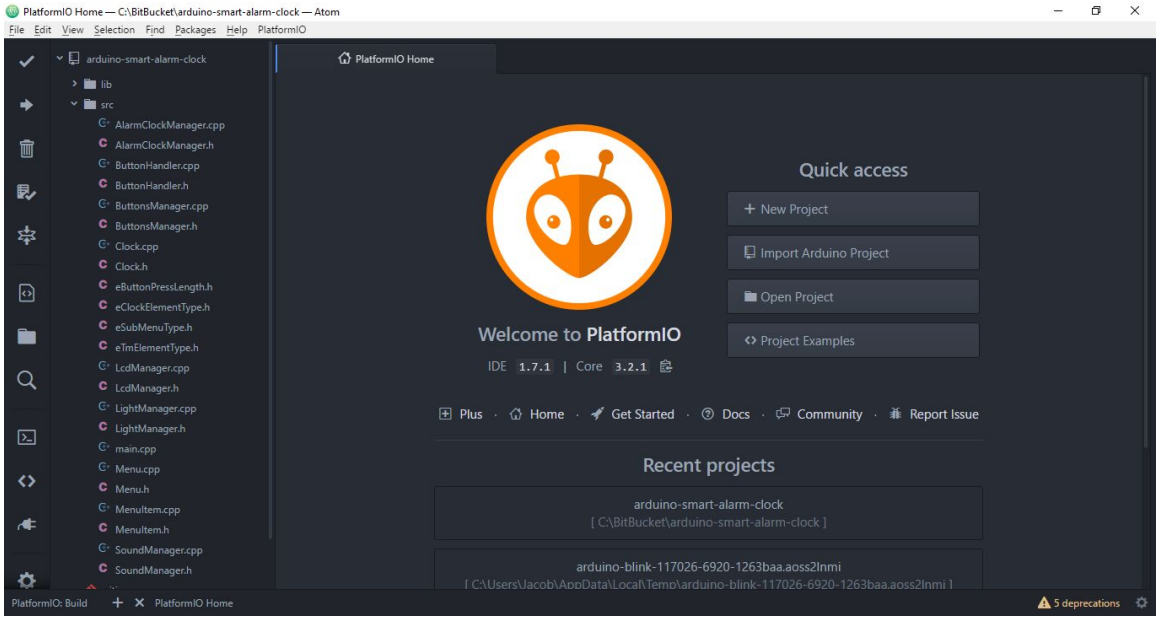
פרק 1 – מבוא:

מהות הפרויקט:

מטרת הפרויקט הוא לבנות שעון מעורר חכם על בסיס רכיב הארדואינו, בדרגש על פיתוח OOP.
רוב פרויקטי הארדואינו אינם ממומשים OOP. ולכן רציתי ליצור פרויקט אשר רוב ההתמקדות שלו היא פיתוח OOP בסביבת ארדואינו.

תיאור סביבת העבודה:

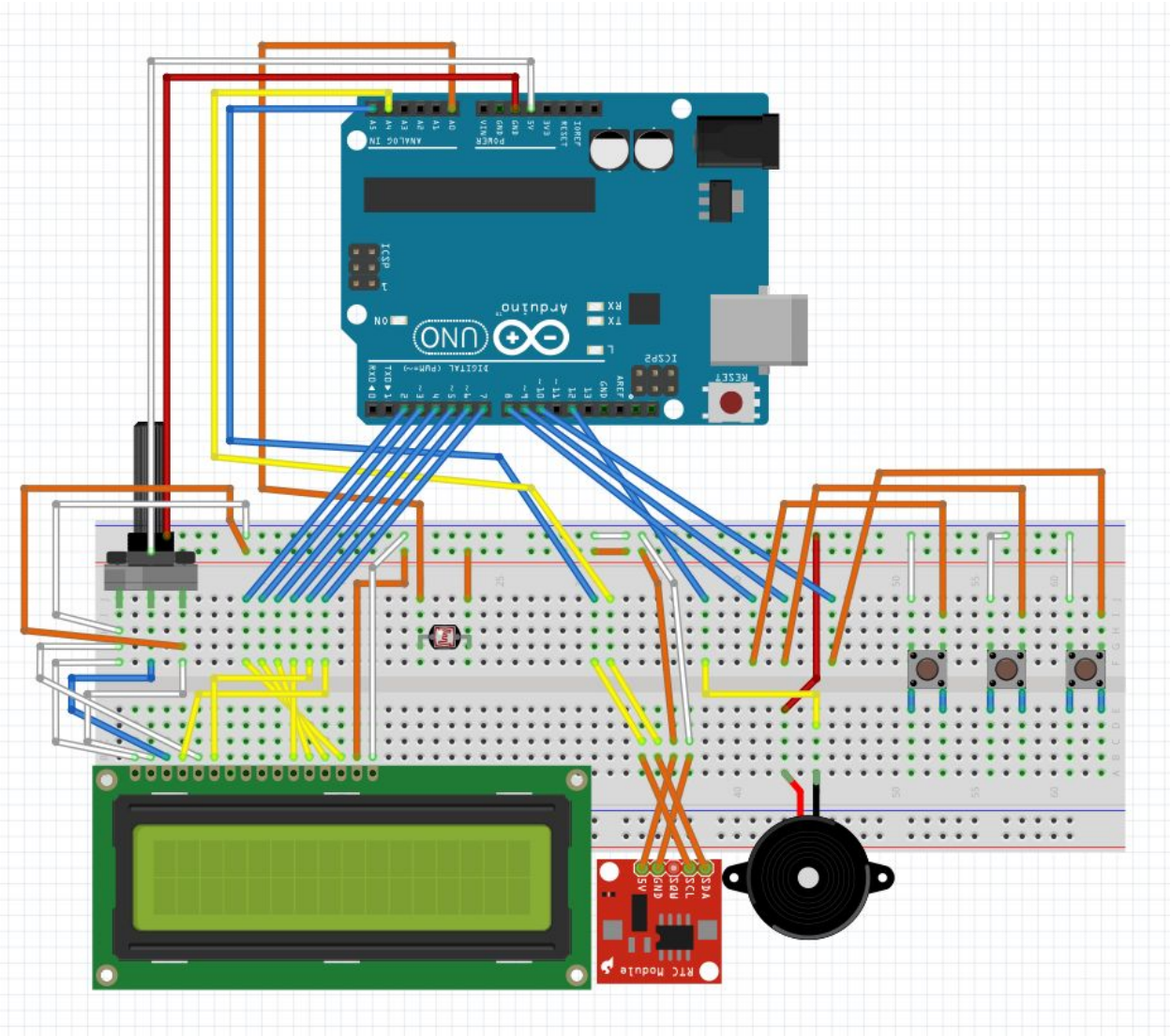
הפיתוח מתבסס על סביבת עבודה של בשם PlatfotmIO אשר מבוסס על עורך טקסט בשם Atom.
הפיתוח הבצע בשפת Cpp בתוספת מספר ספריות לרכיבים השונים.
בחירת סביבת העבודה נעשה מהצורך בכמה שיותר עזרה בפיתוח הקוד הפרויקט בארדואינו.
סביבת העבודה נותנת אפשרות של השלמה אוטומטית, עבודה עם חלונות זה לצד זה, שימוש בספריות מהאינטרנט ועוד.



פרק 2 – תהליך מימוש הפרויקט:

רכיבי החומרה:

1. בקר מרכזי - ארדואינו אונו
2. רכיב תצוגה lcd 1602
3. שלושה רכיבי כפתורים
4. רכיב שמע passive buzzer
5. רכיב שעון זמן אמת TinyRTC
6. רכיב מדידית עוצמת אור



השרטוט הוכן בתוכנה בשם fritzing ניתן להוריד אותה בחינם כאן

התקנת סביבת העבודה:

1. סביבת פיתוח PlatformIO להורדה
2. קומפיילר cpp בשם LLVM להורדה
3. יש להתקין את התכונות עם ההגדרות ברירת המחדל
4. יש להיכנס לקובץ platformio.ini ולשמור אותו כדי שזה יעדכן את הפרויקט עם הספריות המתאימות
5. יש ללחוץ F7 לבחור בבניה\העלאה של הפרויקט.

שלבי הפיתוח:

1. בדיקה של כל הרכיבים
2. חיבור כל הרכיבים בצורה מסודרת
3. תפעול בסיסי של הרכיבים השונים
4. חלוקה של הקוד למחלקות בהתאם לתפקידי עבודה
5. תיקון שגיאות תוך כדי הפיתוח

שפת הקוד:

הפיתוח התבצע בגישה של OOP ופותח עם Cpp, נעשה שימוש במספר ספריות לצורך הקלה על הפיתוח

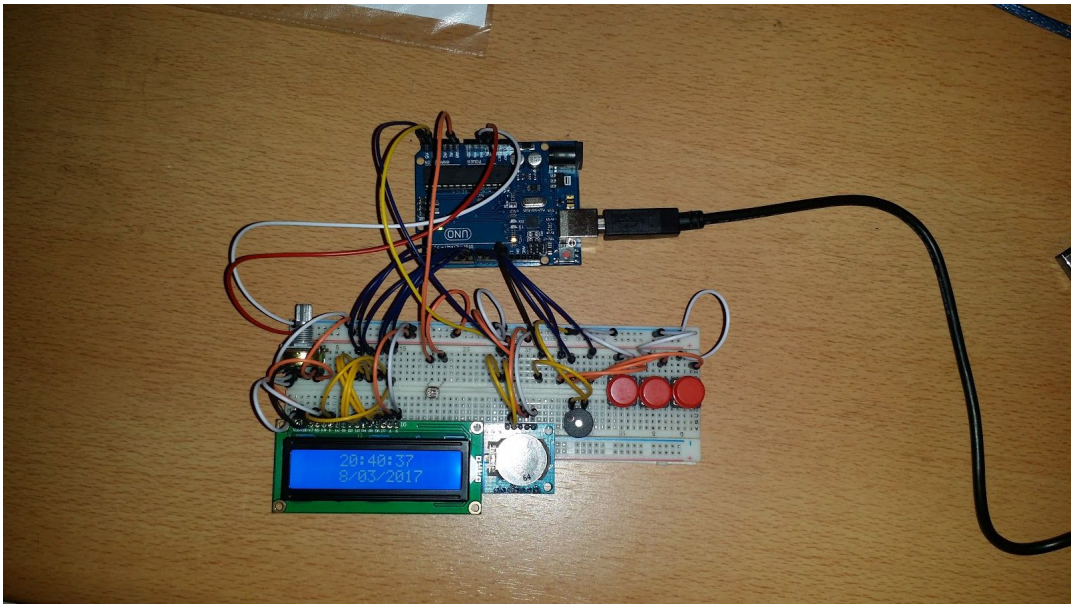
1. TaskScheduler - לעבודה א-סינכורנית של הרכיבים השונים , קלט, פלט, תצוגה, השמעת קול, חיישן עוצמת אור.
2. Time - לעבודה עם השעון הפנימי של הבקר, השמה של הזמן המערכת לבקר ,ושימוש במבנה של זמן לעבודה נוחה.
3. DS3232RTC - לשמירת הזמן והתאריך, וטעינתו וההפרש מאז הרכיב כבה בעת עליית הבקר.

מחלקות ותפקידם:

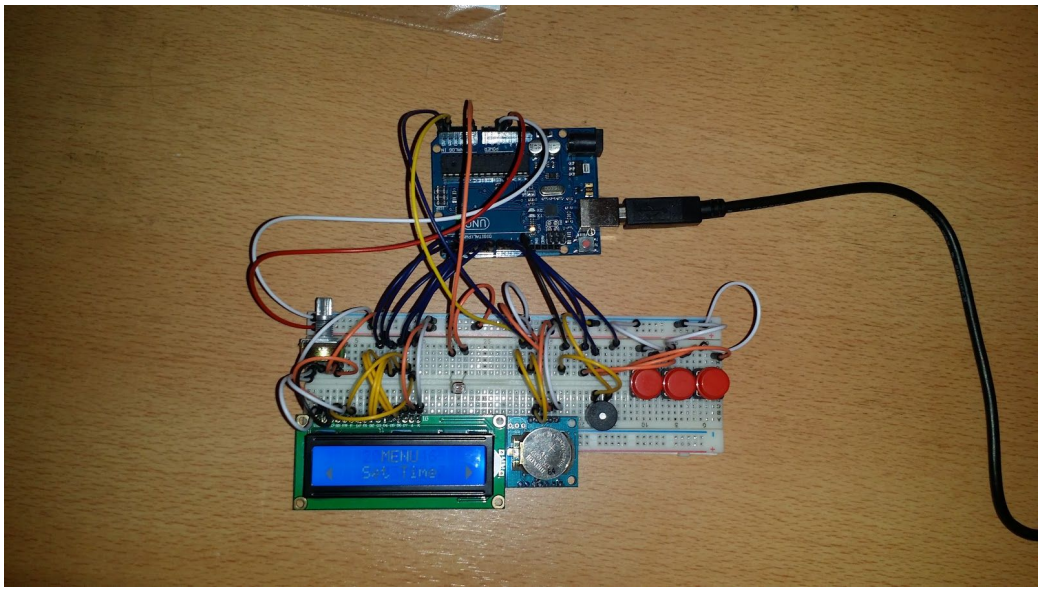
- AlarmClockManager - מחלקה אשר מנהלת את כלל הרכיבים של השעון ועושה האינטרקציה ביניהם
- ButtonHandler - מחלקה אשר מנהלת את אירועי הכפתור הבודד, לחיצה קצרה, לחיצה ארוכה.
- ButtonsManager - מחלקה אשר מאתחלת את שלושת הכפתורים ומאזינה לאירועים של כל כפתור
- Clock - מחלקה אשר מנהלת את השעון, השעון המעורר, שמירה וטעינה של השעון.
- eButtonPressLength - זהו enum אשר מגדיר את סוגי הלחיצות
- eClockElementType - זהו enum אשר מגדיר את סוגי אלמנטי השעון (שעון נוכחי, שעון מעורר)
- eSubMenuType - זהו enum אשר מגדיר את סוגי תתי התפריטים.
- eTmElementType - זהו enum אשר מגדיר את סוגי שדות מבנה השעון
- LcdManager - מחלקה אשר מנהלת את התצוגה של השעון החכם, כותבת את התפריטים, את התת-תפריטים,את השעון,
- -LightManager - מחלקה אשר מנהלת את חיישן עוצמת האור, מדידת הערך הנוכחי, והשמת ערך להתראה.
- main - המחלקה הראשית שממנו הבקר מתחיל את פעולתו, בו מאותחל רכיב השעון אשר מנהל את כלל הרכיבים ומעדכן את שאר המצבים בצורה א-סינכורנית ע"י תזמונים.
- Menu - מחלקה אשר מנהלת את רכיבי התפריט
- MenuItem - מחלקה של רכיב תפריט בודד
- SoundManager - מחלקה אשר מנהלת את השמע השעון

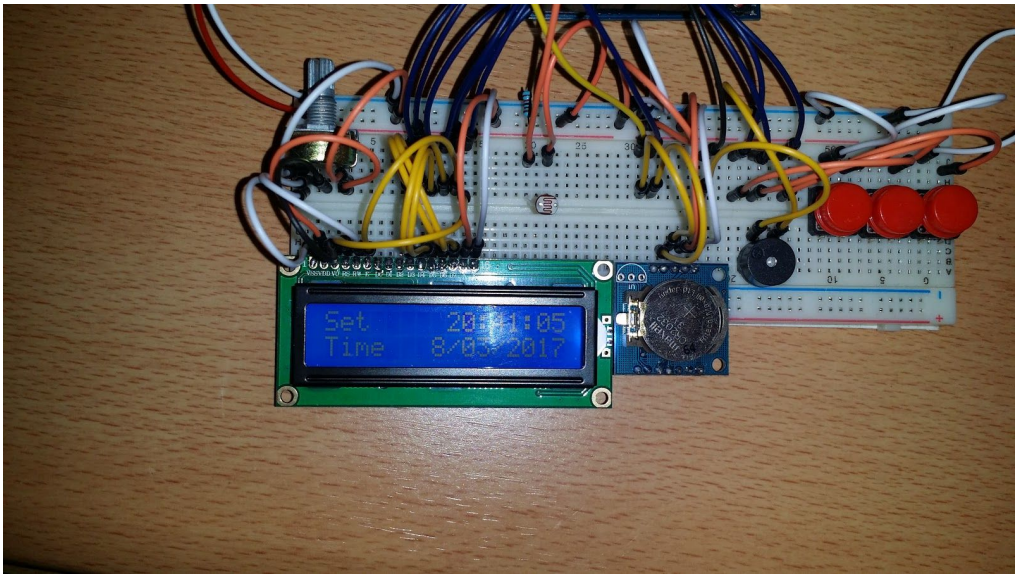
מצבי השעון:

מצב ברירת המחדל בו הוא מציג את השעה הנוכחית והתאריך

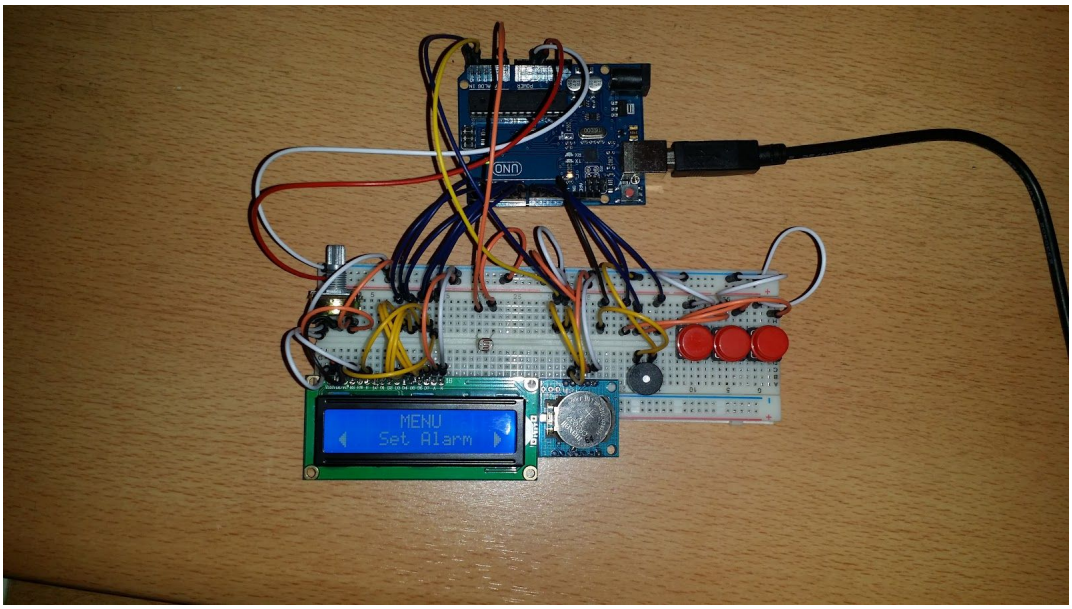


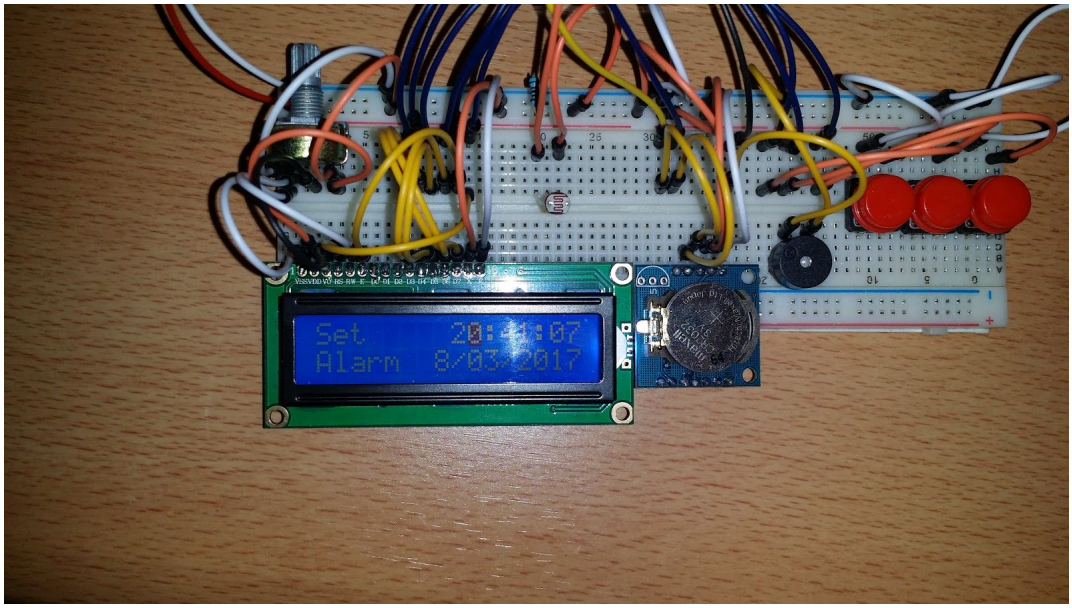
תפריט של השמת השעה והתאריך



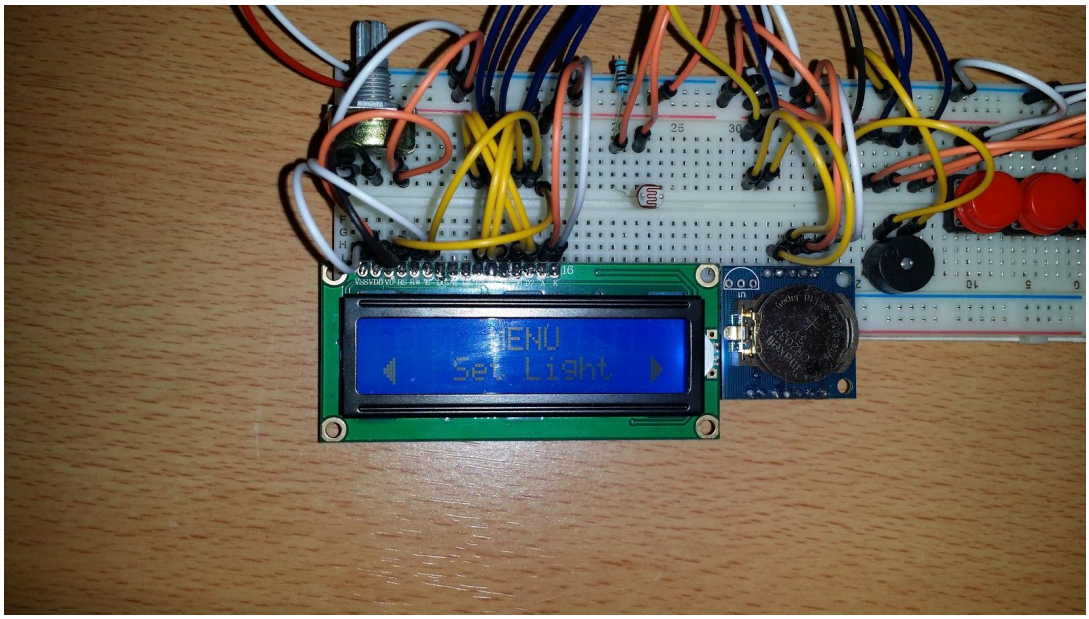


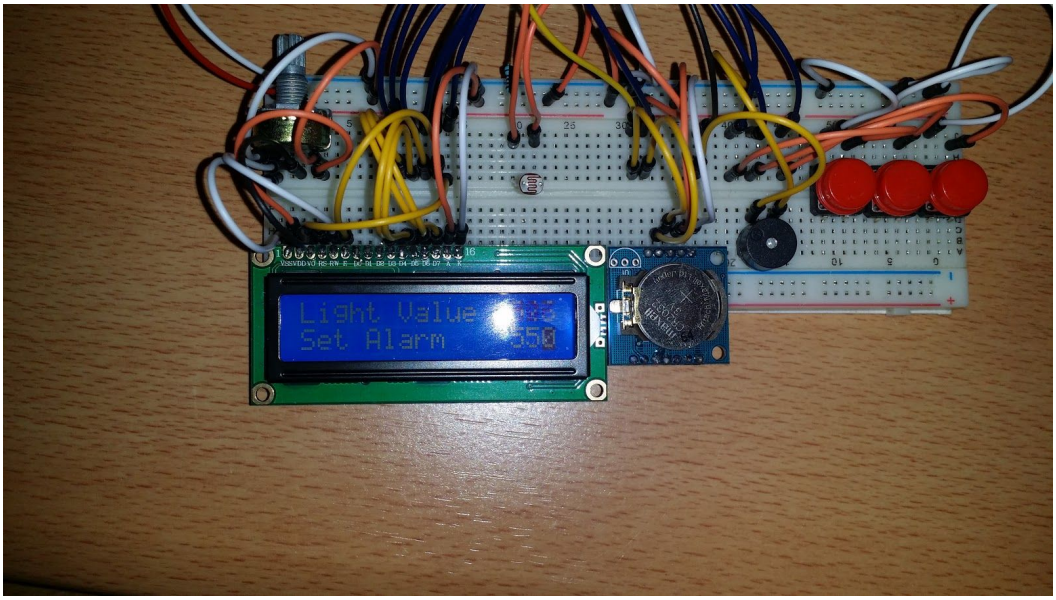
תפריט להכנסת השעה לשעון המעורר



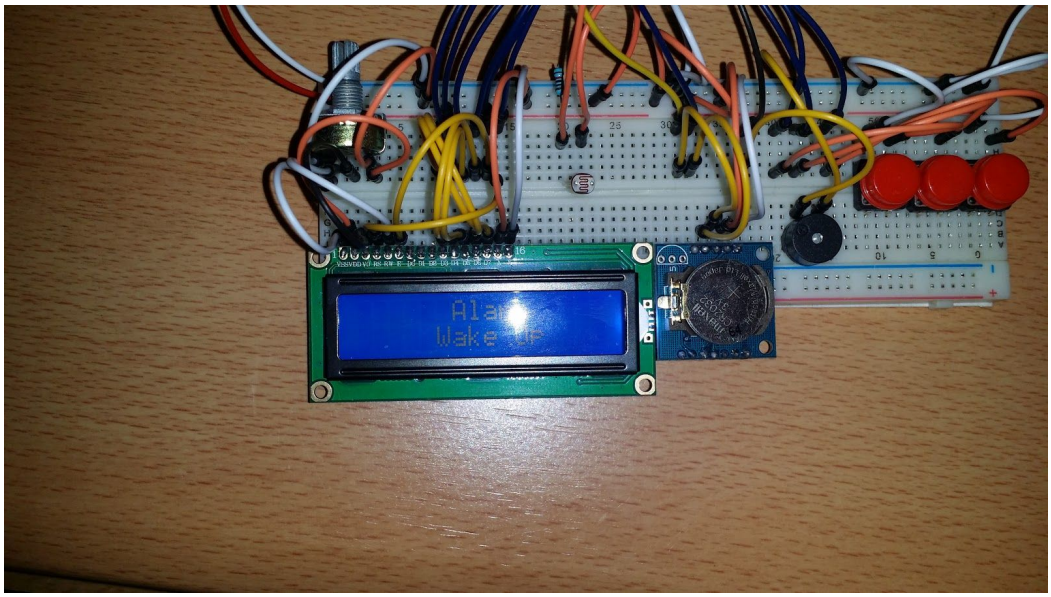


תפריט להפעלה התראה ע"י רכיב עוצמת התאורה





תצוגת מצב התראה



פרק 3 - קוד מקור

AlarmClockManager.h

```
#ifndef ALARM_CLOCK_MANAGER
#define ALARM_CLOCK_MANAGER

#include <Arduino.h>
#include <Wire.h>
#include <DS3232RTC.h>
#include <Time.h>
#include <Menu.h>
#include <ButtonsManager.h>
#include <SoundManager.h>
#include <eButtonPressLength.h>
#include <LcdManager.h>
#include <Clock.h>
#include <eClockElementType.h>
#include <eTmElementType.h>
#include <LightManager.h>

class AlarmClockManager {
private:
  const int _menuSize;
  const int _timeToResetToDefualtMenu;
  const int _menuNumberIterations1;
  const int _menuNumberIterations2;
  const int _menuNumberIterations3;
  const int _menuNumberIterations4;
  const char* _menuName1;
  const char* _menuName2;
  const char* _menuName3;
  const char* _menuName4;


  int _timeFromLastInput;
  bool _insideMenu;
  int _insideMenuIndex;
  bool _alarmTriggered;
```

```
  bool _alarmSet;
  bool _alarmLightSet;


  Menu* _menu;
  ButtonsManager* _buttonsManager;
  SoundManager* _soundManager;
  LcdManager* _lcdManager;
  Clock* _clock;
  LightManager* _lightManager;

  void moveToNextIndexInsideMenu();
  void preformMenuAction();

public:
  AlarmClockManager();
  ~AlarmClockManager();
  void init();
  void printCurrentMenu();
  void handleButtonsInput();
  void playAlarm();
  void checkAlarm();
  void updateLightValue();
};

#endif
```

AlarmClockManager.cpp

```
#include "AlarmClockManager.h"


AlarmClockManager::AlarmClockManager()
: _menuSize(4), _timeToResetToDefualtMenu(50),
_menuNumberIterations1(0), _menuNumberIterations2(6), _menuNumberIterations3(3),
_menuNumberIterations4(1),
_menuName1("Menu 1"), _menuName2("Set Time"), _menuName3("Set
Alarm"),_menuName4("Set Light")
{
  _timeFromLastInput = 0;
  _insideMenu = false;
  _insideMenuIndex = 0;
  _alarmTriggered = false;
```

```
  _alarmSet = false;
  _alarmLightSet = false;

  _menu = new Menu(_menuSize);
  _menu->addItemToMenu(0, _menuName1, _menuNumberIterations1);
  _menu->addItemToMenu(1, _menuName2, _menuNumberIterations2);
  _menu->addItemToMenu(2, _menuName3, _menuNumberIterations3);
  _menu->addItemToMenu(3, _menuName4, _menuNumberIterations4);


  _lcdManager = new LcdManager();
  _buttonsManager = new ButtonsManager();
  _soundManager = new SoundManager();
  _clock = new Clock();
  _lightManager = new LightManager();
}

AlarmClockManager::~AlarmClockManager()
{
}

void AlarmClockManager::init()
{
  _buttonsManager->initAll();
  _soundManager->init();
  _lcdManager->init();
  _clock->init();
  _lightManager->init();
}

void AlarmClockManager::printCurrentMenu()
{

  if (!_alarmTriggered ){
    if (_menu->getCurrentIndex() == 0){
      if(timeStatus() == timeSet) {
        _clock->setTimeElement(now(), eClockElementType::currentTime);
        tmElements_t tm = _clock->getTimeElement(eClockElementType::currentTime);
        _lcdManager->printRealTimeOnLcd(tm, _alarmSet, _alarmLightSet);
      }
    } else {
      if (_insideMenu){
          if (_menu->getCurrentIndex() == 1){
```

```
          // printSetTimeOnLcd();
          if(timeStatus() == timeSet) {
            tmElements_t tm = _clock->getTimeElement(eClockElementType::timeToSet);
            _lcdManager->printInsideMenuWithClock(tm,"Set","Time", _insideMenuIndex);
          }
        }
        else if (_menu->getCurrentIndex() == 2){
          if(timeStatus() == timeSet) {
            tmElements_t tm = _clock->getTimeElement(eClockElementType::timeToAlarm);
            _lcdManager->printInsideMenuWithClock(tm, "Set","Alarm",  _insideMenuIndex);
          }
        }
        else if (_menu->getCurrentIndex() == 3){

_lcdManager->printInsideMenuLight(_lightManager->getCurrentValue(),_lightManager->getAlar
mValue(),_insideMenuIndex);
        }
      }
      else{
        const char* menuStr = _menu->getCurrentMenu()->getName();
        _lcdManager->printMenuTextOnLcd(menuStr);
      }
    }
  }
  else
  {
    _lcdManager->printAlarmTriggeredOnLcd();
  }

}

void AlarmClockManager::handleButtonsInput()
{

  _buttonsManager->handleAll();
  if(!_alarmTriggered){
    if(_buttonsManager->getButtonLastEvent(0) == eButtonPressLength::pressShort){
      if (!_insideMenu){
        _menu->moveIndexUp();
        _timeFromLastInput = 0;
        _lcdManager->clearLcd();
      }
      else
```

```
      {
        if (_menu->getCurrentIndex() == 1)
        {
          _clock->dtCurrentTimeField(1, eClockElementType::timeToSet,
_lcdManager->getTmElementByIndex(_insideMenuIndex));
        } else if (_menu->getCurrentIndex() == 2)
        {
          _clock->dtCurrentTimeField(1, eClockElementType::timeToAlarm,
_lcdManager->getTmElementByIndex(_insideMenuIndex));
        }
        else if (_menu->getCurrentIndex() == 3)
        {
          _lightManager->addToAlarmValue(10);
        }
      }
    }
    else if (_buttonsManager->getButtonLastEvent(1)  == eButtonPressLength::pressShort){
      if (!_insideMenu)
      {
        _menu->moveIndexDown();
        _timeFromLastInput = 0;
        _lcdManager->clearLcd();
      }
      else
      {
        if (_menu->getCurrentIndex() == 1)
        {
          _clock->dtCurrentTimeField(-1, eClockElementType::timeToSet,
_lcdManager->getTmElementByIndex(_insideMenuIndex));
        } else if (_menu->getCurrentIndex() == 2)
        {
          _clock->dtCurrentTimeField(-1, eClockElementType::timeToAlarm,
_lcdManager->getTmElementByIndex(_insideMenuIndex));
        }
        else if (_menu->getCurrentIndex() == 3)
        {
          _lightManager->addToAlarmValue(-10);
        }
      }
    }
    else if (_buttonsManager->getButtonLastEvent(2)  == eButtonPressLength::pressShort){
      if (_menu->getCurrentIndex() == 1){
        if (!_insideMenu){
```

```
        _insideMenu = true;
        _insideMenuIndex = 0;
        _clock->setTimeElement(now(), eClockElementType::timeToSet);
        _lcdManager->clearLcd();
        _lcdManager->setBlink(true);
      }
      else {
        moveToNextIndexInsideMenu();
      }
    }
    else if (_menu->getCurrentIndex() == 2){
      if (!_insideMenu){
        _insideMenu = true;
        _insideMenuIndex = 0;
        _clock->setTimeElement(now(), eClockElementType::timeToAlarm);
        _lcdManager->clearLcd();
        _lcdManager->setBlink(true);
      }
      else {
        moveToNextIndexInsideMenu();
      }
    }
    else if (_menu->getCurrentIndex() == 3){
      if (!_insideMenu){
        _insideMenu = true;
        _insideMenuIndex = 0;
        _lcdManager->clearLcd();
        _lcdManager->setBlink(true);
        _lightManager->setAlarmValue(_lightManager->getCurrentValue());
      }
      else {
        moveToNextIndexInsideMenu();
      }
    }
  }
  else
  {
    if (_timeFromLastInput >= _timeToResetToDefualtMenu && !_menu->isOnDefualtMenu()
&& !_insideMenu){
      _menu->resetToDefualtMenu();
      _lcdManager->clearLcd();
    } else if (!_insideMenu){
      _timeFromLastInput ++;
```

```
      }
    }
  }
  else
  {
    if(_buttonsManager->getButtonLastEvent(0) == eButtonPressLength::pressShort ||
      _buttonsManager->getButtonLastEvent(1) == eButtonPressLength::pressShort ||
      _buttonsManager->getButtonLastEvent(2) == eButtonPressLength::pressShort){
      _lcdManager->clearLcd();
      _alarmTriggered = false;
      _soundManager->setPlayMusic(false);
    }
  }

}



void AlarmClockManager::moveToNextIndexInsideMenu()
{
  if(_insideMenuIndex < _menu->getCurrentMenuIterations() - 1) {
    _insideMenuIndex++;
  }
  else
  {
    preformMenuAction();
    _insideMenu = false;
    _lcdManager->setBlink(false);
    _lcdManager->clearLcd();
    _menu->resetToDefualtMenu();

  }
}


void AlarmClockManager::preformMenuAction()
{
    if (_menu->getCurrentIndex() == 1){
      _clock->setSystemRtcTime();
      _alarmSet = false;
    }
    else if (_menu->getCurrentIndex() == 2){
      _alarmSet = true;
```

```
    } else if (_menu->getCurrentIndex() == 3){
      _alarmLightSet = true;
    }
}

void AlarmClockManager::playAlarm()
{
  _soundManager->playNext();
}

void AlarmClockManager::checkAlarm()
{
  if ( !_alarmTriggered ) {
    if (_alarmSet){
      tmElements_t tmCurrentTime =
_clock->getTimeElement(eClockElementType::currentTime);
      tmElements_t tmAlarmTime = _clock->getTimeElement(eClockElementType::timeToAlarm);
      if (tmCurrentTime.Hour == tmAlarmTime.Hour &&
        tmCurrentTime.Minute == tmAlarmTime.Minute &&
        tmCurrentTime.Second >= tmAlarmTime.Second)
      {
        _lcdManager->clearLcd();
        _soundManager->setPlayMusic(true);
        _alarmTriggered = true;
        _alarmSet = false;
      }
    } else if (_alarmLightSet){
      if (_lightManager->isAlarmValuePassed()){
        _lcdManager->clearLcd();
        _soundManager->setPlayMusic(true);
        _alarmTriggered = true;
        _alarmLightSet = false;
      }
    }

  }
}

void AlarmClockManager::updateLightValue()
{
  _lightManager->updateLightValue();
  // Serial.println(_lightManager->getCurrentValue());
}
```

ButtonHandler.h

```
#ifndef BUTTON_HANDLER
#define BUTTON_HANDLER

#include <Arduino.h>
#include "eButtonPressLength.h"

class ButtonHandler {
  public:

    ButtonHandler(const int i_PinNumber,const int i_LongPressDuration,const char*
i_ButtonName);
    void init();
    eButtonPressLength handle();
    eButtonPressLength getLastEvent();
    const char* getName();
  private:
    const int pinNumber;
    const int longPressDuration;
    const char* buttonName;
    boolean wasPressed;
    int PressCounter;
    eButtonPressLength lastEvent;
};

#endif
```

ButtonHandler.cpp

```
#include "ButtonsManager.h"


ButtonsManager::ButtonsManager()
: _defaultLongPressDuration(25), _numberOfButtons(3),
_defaultButtonPinNumber1(8),  _defaultButtonPinNumber2(9),  _defaultButtonPinNumber3(10),
_defaultButtonName1("button1"), _defaultButtonName2("button2"),
_defaultButtonName3("button3")
```

```
{
  _buttonArr = new ButtonHandler*[_numberOfButtons];
  _buttonArr[0] = new ButtonHandler(_defaultButtonPinNumber1, _defaultLongPressDuration,
_defaultButtonName1);
  _buttonArr[1] = new ButtonHandler(_defaultButtonPinNumber2, _defaultLongPressDuration,
_defaultButtonName2);
  _buttonArr[2] = new ButtonHandler(_defaultButtonPinNumber3, _defaultLongPressDuration,
_defaultButtonName3);
}

ButtonsManager::~ButtonsManager()
{
  if (_buttonArr != NULL){
    for (int i=0;i<_numberOfButtons;i++){
      delete []_buttonArr[i];
    }
    delete []*_buttonArr;
  }
}

void ButtonsManager::initAll()
{
  for (int i=0; i < _numberOfButtons; i++){
    _buttonArr[i]->init();
  }
}

void ButtonsManager::handleAll()
{
  for (int i=0; i < _numberOfButtons; i++){
    _buttonArr[i]->handle();
  }
}


eButtonPressLength ButtonsManager::getButtonLastEvent(int index)
{
  return _buttonArr[index]->getLastEvent();
}
```

ButtonsManager.h

```
#ifndef BUTTONS_MANAGER
#define BUTTONS_MANAGER

#include "ButtonHandler.h"
#include "eButtonPressLength.h"

class ButtonsManager {
private:
  const int _defaultLongPressDuration;
  const int _numberOfButtons;
  const int _defaultButtonPinNumber1;
  const int _defaultButtonPinNumber2;
  const int _defaultButtonPinNumber3;
  const char* _defaultButtonName1;
  const char* _defaultButtonName2;
  const char* _defaultButtonName3;

  ButtonHandler** _buttonArr;

  public:
    ButtonsManager();
    ~ButtonsManager();
    void initAll();
    void handleAll();
    eButtonPressLength getButtonLastEvent(int index);

};


#endif
```

ButtonsManager.cpp

```
#include "ButtonsManager.h"


ButtonsManager::ButtonsManager()
: _defaultLongPressDuration(25), _numberOfButtons(3),
_defaultButtonPinNumber1(8),  _defaultButtonPinNumber2(9),  _defaultButtonPinNumber3(10),
```

```
_defaultButtonName1("button1"), _defaultButtonName2("button2"),
_defaultButtonName3("button3")
{
  _buttonArr = new ButtonHandler*[_numberOfButtons];
  _buttonArr[0] = new ButtonHandler(_defaultButtonPinNumber1, _defaultLongPressDuration,
_defaultButtonName1);
  _buttonArr[1] = new ButtonHandler(_defaultButtonPinNumber2, _defaultLongPressDuration,
_defaultButtonName2);
  _buttonArr[2] = new ButtonHandler(_defaultButtonPinNumber3, _defaultLongPressDuration,
_defaultButtonName3);
}

ButtonsManager::~ButtonsManager()
{
  if (_buttonArr != NULL){
    for (int i=0;i<_numberOfButtons;i++){
      delete []_buttonArr[i];
    }
    delete []*_buttonArr;
  }
}

void ButtonsManager::initAll()
{
  for (int i=0; i < _numberOfButtons; i++){
    _buttonArr[i]->init();
  }
}

void ButtonsManager::handleAll()
{
  for (int i=0; i < _numberOfButtons; i++){
    _buttonArr[i]->handle();
  }
}


eButtonPressLength ButtonsManager::getButtonLastEvent(int index)
{
  return _buttonArr[index]->getLastEvent();
}
```

Clock.h

```
#ifndef CLOCK
#define CLOCK

#include <Arduino.h>
#include <Time.h>
#include <DS3232RTC.h>
#include <eClockElementType.h>
#include <eTmElementType.h>

class Clock{
  private:
    tmElements_t _timeCurrent;
    tmElements_t _timeToSet;
    tmElements_t _timeToAlarm;

  public:
    Clock();
    ~Clock();
    void init();
    void setTimeElement(time_t time, eClockElementType elementClock);
    tmElements_t getTimeElement(eClockElementType elementClock);
    void dtCurrentTimeField(int dt, eClockElementType elementClock, eTmElementType
elementTime);
    void setSystemRtcTime();
};

#endif
```

Clock.cpp

```
#include <Clock.h>

Clock::Clock(){

}

Clock::~Clock(){

}
```

```
void Clock::init(){
  setSyncProvider(RTC.get);
}

void Clock::setTimeElement(time_t time, eClockElementType elementClock)
{
  if (elementClock == eClockElementType::currentTime){
      breakTime(time, _timeCurrent);
  } else if (elementClock == eClockElementType::timeToSet)
  {
    breakTime(time, _timeToSet);
  } else if (elementClock == eClockElementType::timeToAlarm){
    breakTime(time, _timeToAlarm);
  }
}

tmElements_t Clock::getTimeElement(eClockElementType elementClock){
  tmElements_t tmToReturn;
  if (elementClock == eClockElementType::currentTime){
    tmToReturn = _timeCurrent;
  } else if (elementClock == eClockElementType::timeToSet)
  {
    tmToReturn = _timeToSet;
  } else if (elementClock == eClockElementType::timeToAlarm){
    tmToReturn = _timeToAlarm;
  }

  return tmToReturn;
}

void Clock::dtCurrentTimeField(int dt, eClockElementType elementClock, eTmElementType
elementTime)
{
  tmElements_t tm;
  if (elementClock == eClockElementType::currentTime){
    tm = _timeCurrent;
  } else if (elementClock == eClockElementType::timeToSet)
  {
    tm = _timeToSet;
  } else if (elementClock == eClockElementType::timeToAlarm){
    tm = _timeToAlarm;
  }
```

```
if (elementTime == eTmElementType::hours){
  tm.Hour += dt;
  if (tm.Hour  == 24){
    tm.Hour = 0;
  } else if (tm.Hour == -1){
    tm.Hour = 24;
  }
}
else if (elementTime == eTmElementType::minutes){
  tm.Minute += dt;
  if (tm.Minute == 60){
    tm.Minute = 0;
  } else if (tm.Minute == -1){
    tm.Minute = 59;
  }
}
else if (elementTime == eTmElementType::seconds){
  tm.Second += dt;
  if (tm.Second == 60){
    tm.Second = 0;
  } else if (tm.Second == -1){
    tm.Second = 59;
  }
}
else if (elementTime == eTmElementType::days){
  tm.Day += dt;
  if (tm.Month == 2){
    if (tm.Year % 4 == 0 ){
      if (tm.Day  == 30){
          tm.Day = 1;
      } else if (tm.Day == 0){
        tm.Day = 29;
      }
    }
    else {
      if (tm.Day  == 29){
        tm.Day = 1;
      } else if (tm.Day == 0){
        tm.Day = 28;
      }
    }
  }
```

```
    else if (((tm.Month == 4) || (tm.Month == 6) || (tm.Month == 9) || (tm.Month == 11))){
      if (tm.Day  == 31){
          tm.Day = 1;
      } else if (tm.Day == 0){
        tm.Day = 30;
      }

    } else {
      if ( tm.Day == 32){
          tm.Day = 1;
      } else if (tm.Day == 0){
        tm.Day = 31;
      }
    }
  }
  else if (elementTime == eTmElementType::months){
    tm.Month += dt;
    if (tm.Month == 13){
      tm.Month = 1;
    } else if (tm.Month == 0){
      tm.Month = 12;
    }
  }
  else if (elementTime == eTmElementType::years){
    tm.Year += dt;
  }

  if (elementClock == eClockElementType::currentTime){
    _timeCurrent = tm;
  } else if (elementClock == eClockElementType::timeToSet)
  {
    _timeToSet = tm;
  } else if (elementClock == eClockElementType::timeToAlarm){
     _timeToAlarm = tm;
  }
}

void Clock::setSystemRtcTime(){
  time_t t = makeTime(_timeToSet);
  setTime(t);
  RTC.set(t);
}
```

eButtonPressLength.h

```
#ifndef E_BUTTON_PRESS_TYPES
#define E_BUTTON_PRESS_TYPES

enum eButtonPressLength
{
  pressNone=0,
  pressShort,
  pressLong
};

#endif
```

eClockElementType.h

```
#ifndef E_CLOCK_ELEMENT_TYPE
#define E_CLOCK_ELEMENT_TYPE

enum eClockElementType {
  currentTime,
  timeToSet,
  timeToAlarm
};

#endif
```

eSubMenuType.h

```
#ifndef E_SUB_MENU_TYPE
#define E_SUB_MENU_TYPE

enum eSubMenuType {
  ClockMenu,
  SensorMenu
};

#endif
```

eTmElementType.h

```
#ifndef E_TM_ELEMENT_TYPE
#define E_TM_ELEMENT_TYPE

enum eTmElementType {
  seconds,
  minutes,
  hours,
  days,
  months,
  years
};

#endif
```

LcdManager.h

```
#ifndef LCD_MANAGER
#define LCD_MANAGER


#include <Arduino.h>
#include <LiquidCrystal.h>
#include <Time.h>
#include <eTmElementType.h>
#include <eSubmenuType.h>

class LcdManager {
  private:
    const uint8_t _lcdPinNumber1;
    const uint8_t _lcdPinNumber2;
    const uint8_t _lcdPinNumber3;
    const uint8_t _lcdPinNumber4;
    const uint8_t _lcdPinNumber5;
    const uint8_t _lcdPinNumber6;
    const uint8_t _lcdNumOfCols;
    const uint8_t _lcdNumOfRows;

    LiquidCrystal* _lcd;
```

```
    void setBlinkCursor(int index, eSubMenuType subMenuType);

  public:
    LcdManager();
    ~LcdManager();
    void init();
    void printRealTimeOnLcd(tmElements_t tm, bool alarmSet, bool alarmLight);
    void printMenuTextOnLcd(const char* str);
    void printInsideMenuWithClock(tmElements_t tm, const char* str1,const char* str2, int
menuIndex);
    void printInsideMenuLight(int currentValue, int alarmValue, int menuIndex);
    void printAlarmTriggeredOnLcd();
    void setBlink(boolean enabled);
    void clearLcd();
    eTmElementType getTmElementByIndex(int index);

};

#endif
```

LcdManager.cpp

```
#include <LcdManager.h>

LcdManager::LcdManager()
:_lcdPinNumber1(7), _lcdPinNumber2(6), _lcdPinNumber3(5),
_lcdPinNumber4(4), _lcdPinNumber5(3), _lcdPinNumber6(2),
_lcdNumOfCols(16), _lcdNumOfRows(2)
{
  _lcd = new LiquidCrystal(_lcdPinNumber1, _lcdPinNumber2, _lcdPinNumber3,
  _lcdPinNumber4, _lcdPinNumber5, _lcdPinNumber6);
}

LcdManager::~LcdManager()
{

}

void LcdManager::init()
{
  _lcd->begin(_lcdNumOfCols, _lcdNumOfRows);
```

```
byte bellChar[8] = {
  0b00100,
  0b01010,
  0b01010,
  0b01010,
  0b01010,
  0b11111,
  0b00000,
  0b00100
};
byte rightSignChar[8] = {
        0b10000,
        0b11000,
        0b11100,
        0b11110,
        0b11110,
        0b11100,
        0b11000,
        0b10000
};

byte leftSignChar[8] = {
        0b00001,
        0b00011,
        0b00111,
        0b01111,
        0b01111,
        0b00111,
        0b00011,
        0b00001
};
byte lightSensorChar[8] = {
        0b00100,
        0b01010,
        0b10001,
        0b10001,
        0b10101,
        0b10101,
        0b01110,
        0b01110
};
_lcd->createChar(0, bellChar);
_lcd->createChar(1, rightSignChar);
```

```
  _lcd->createChar(2, leftSignChar);
  _lcd->createChar(3, lightSensorChar);
}

void LcdManager::clearLcd(){
  _lcd->clear();
}

void LcdManager::printRealTimeOnLcd(tmElements_t tm, bool alarmSet, bool alarmLight){
  if (alarmSet){
    _lcd->setCursor(0,0);
    _lcd->write(byte(0));
  }
  if (alarmLight){
    _lcd->setCursor(1,0);
    _lcd->write(byte(3));
  }
  char* msg = new char[17];
  _lcd->setCursor(4,0);
  sprintf(msg,"%2d:%02d:%02d",tm.Hour,tm.Minute,tm.Second);
  _lcd->print(msg);
  _lcd->setCursor(3,1);
  sprintf(msg,"%2d/%02d/%02d",tm.Day,tm.Month,tm.Year + 1970);
  _lcd->print(msg);
  delete []msg;
}

void LcdManager::printMenuTextOnLcd(const char* str){
  _lcd->setCursor(6,0);
  _lcd->print("MENU");
  _lcd->setCursor(4,1);
  _lcd->print(str);
  _lcd->setCursor(15,1);
 _lcd->write(byte(1));
 _lcd->setCursor(0,1);
 _lcd->write(byte(2));
}

void LcdManager::printInsideMenuWithClock(tmElements_t tm, const char* str1,const char*
str2, int menuIndex){
  char* msg = new char[17];
  _lcd->setCursor(0,0);
  _lcd->print(str1);
```

```
  _lcd->setCursor(8,0);
  sprintf(msg,"%2d:%02d:%02d",tm.Hour,tm.Minute,tm.Second);
  _lcd->print(msg);
  _lcd->setCursor(0,1);
  _lcd->print(str2);
  _lcd->setCursor(6,1);
  sprintf(msg,"%2d/%02d/%02d",tm.Day,tm.Month,tm.Year + 1970);
  _lcd->print(msg);
  setBlinkCursor(menuIndex,eSubMenuType::ClockMenu);
  delete []msg;
}

void LcdManager::printInsideMenuLight(int currentValue, int alarmValue, int menuIndex){
  char* msg = new char[6];
  _lcd->setCursor(0,0);
  _lcd->print("Light Value");
  _lcd->setCursor(12,0);
  sprintf(msg,"%4d",currentValue);
  _lcd->print(msg);
  _lcd->setCursor(0,1);
  _lcd->print("Set Alarm");
  _lcd->setCursor(12,1);
  sprintf(msg,"%4d",alarmValue);
  _lcd->print(msg);
  setBlinkCursor(menuIndex,eSubMenuType::SensorMenu);
  delete []msg;
}


void LcdManager::printAlarmTriggeredOnLcd()
{
  _lcd->setCursor(6,0);
  _lcd->print("Alarm");
  _lcd->setCursor(5,1);
  _lcd->print("Wake Up");
}

void LcdManager::setBlinkCursor(int index, eSubMenuType subMenuType){
  if (subMenuType == eSubMenuType::ClockMenu){
    if (index == 0){
      _lcd->setCursor(9,0);
    }
    else if (index == 1){
```

```
      _lcd->setCursor(12,0);
    }
    else if (index == 2){
      _lcd->setCursor(15,0);
    }
    else if (index == 3){
      _lcd->setCursor(7,1);
    }
    else if (index == 4){
      _lcd->setCursor(10,1);
    }
    else if (index == 5){
      _lcd->setCursor(15,1);
    }
  } else if (subMenuType == eSubMenuType::SensorMenu){
    if (index == 0){
     _lcd->setCursor(15,1);
   }
  }

}

void LcdManager::setBlink(boolean enabled){
  if (enabled){
    _lcd->blink();
  }
  else {
    _lcd->noBlink();
  }
}

eTmElementType LcdManager::getTmElementByIndex(int index)
{
  eTmElementType elementToReturn = eTmElementType::seconds;
  if (index == 0){
    elementToReturn = eTmElementType::hours;
  }
  else if (index == 1){
    elementToReturn = eTmElementType::minutes;
  }
  else if (index == 2){
    elementToReturn = eTmElementType::seconds;
  }
```

```
  else if (index == 3){
    elementToReturn = eTmElementType::days;
  }
  else if (index == 4){
    elementToReturn = eTmElementType::months;
  }
  else if (index == 5){
    elementToReturn = eTmElementType::years;
  }

  return elementToReturn;
}
```

LightManager.h

```
#ifndef LIGHT_MANAGER
#define LIGHT_MANAGER

#include <Arduino.h>

class LightManager {
  private:
    const int _pinNumber;
    int _currentValue;
    int _alarmValue;

  public:
    LightManager();
    ~LightManager();
    void init();
    void updateLightValue();
    int getCurrentValue();
    void setAlarmValue(int value);
    int getAlarmValue();
    void addToAlarmValue(int valueToAdd);
    bool isAlarmValuePassed();

};

#endif
```

LightManager.cpp

```
#include <LightManager.h>

LightManager::LightManager()
:_pinNumber(0)
{
  _currentValue = 0;
  _alarmValue = 0;
}

LightManager::~LightManager(){

}

void LightManager::init(){
  pinMode(_pinNumber,INPUT);
}

void LightManager::updateLightValue(){
  _currentValue = analogRead(_pinNumber);
}

int LightManager::getCurrentValue(){
  return _currentValue;
}

void LightManager::setAlarmValue(int value){
  _alarmValue = value;
}

int LightManager::getAlarmValue(){
  return _alarmValue;
}

void LightManager::addToAlarmValue(int valueToAdd)
{
  _alarmValue += valueToAdd;
}

bool LightManager::isAlarmValuePassed()
{
```

```
  return _currentValue >= _alarmValue;
}
```

main.cpp

```
#include <Arduino.h>
#include <TaskScheduler.h>
#include <AlarmClockManager.h>
#include <SoundManager.h>

AlarmClockManager *alarmClockManager = new AlarmClockManager();
Scheduler runner;

//
void handleInputCallback();
void printOnLcdCallback();
void checkAlarmCallback();
void playAlarmCallback();
void updateLightCallback();
//
Task handleInputTask(100, TASK_FOREVER, &handleInputCallback);
Task printOnLcdTask(100, TASK_FOREVER, &printOnLcdCallback);
Task checkAlarmTask(1, TASK_FOREVER, &checkAlarmCallback);
Task playAlarmTask(1, TASK_FOREVER, &playAlarmCallback);
Task updateLightTask(500, TASK_FOREVER, &updateLightCallback);
//
void handleInputCallback() {
  alarmClockManager->handleButtonsInput();
}

void printOnLcdCallback() {
  alarmClockManager->printCurrentMenu();
}
//
void checkAlarmCallback() {
  alarmClockManager->checkAlarm();
}

void playAlarmCallback() {
  alarmClockManager->playAlarm();
}
```

```
void updateLightCallback()
{
  alarmClockManager->updateLightValue();
}



void setup()
{
  // Serial.begin(9600);
  alarmClockManager->init();

  runner.addTask(handleInputTask);
  runner.addTask(printOnLcdTask);
  runner.addTask(checkAlarmTask);
  runner.addTask(playAlarmTask);
  runner.addTask(updateLightTask);
  handleInputTask.enable();
  printOnLcdTask.enable();
  checkAlarmTask.enable();
  playAlarmTask.enable();
  updateLightTask.enable();
}

void loop(){
  runner.execute();
}
```

Menu.h

```
#ifndef MENU
#define MENU

#include "MenuItem.h"

class Menu {
private:
  const int _menuSize;
  MenuItem** _menuArr;
  int _menuIndex;
```

```
public:
  Menu(int);
  ~Menu();
  void addItemToMenu(int index, const char* menuName, const int numberOfInputIterations);
  void addItemToMenu(int index, MenuItem* item);
  void printMenuAt(int index);
  void moveIndexUp();
  void moveIndexDown();
  MenuItem* getCurrentMenu();
  void resetToDefualtMenu();
  boolean isOnDefualtMenu();
  int getCurrentIndex();
  const int getCurrentMenuIterations();

};

#endif
```

Menu.cpp

```
#include <Arduino.h>
#include "Menu.h"
#include "MenuItem.h"


Menu::Menu(int menuSize)
: _menuSize(menuSize)
{
  _menuArr = new MenuItem*[menuSize];
  _menuIndex = 0;
}

Menu::~Menu(){
  if (_menuArr != NULL){
    for (int i=0;i<_menuSize;i++){
      delete []_menuArr[i];
    }
    delete []*_menuArr;
  }
}
```

```
void Menu::addItemToMenu(int index, const char* menuName, const int
numberOfInputIterations)
{
  if (index >= 0 && index < _menuSize)
  {
    MenuItem* newItem= new MenuItem(menuName, numberOfInputIterations);
    _menuArr[index] = newItem;
  }
}

void  Menu::addItemToMenu(int index, MenuItem* item)
{
    if (index >= 0 && index < _menuSize)
    {
      _menuArr[index] = item;
    }
}

void Menu::printMenuAt(int index)
{
  if (index >= 0 && index < _menuSize)
  {
    MenuItem *item = _menuArr[index];
    item->printMenu();
  }
}

void Menu::moveIndexUp()
{
  _menuIndex++;
  if (_menuIndex >= _menuSize){
    _menuIndex = 0;
  }
}

void Menu::moveIndexDown()
{
  _menuIndex--;
  if (_menuIndex <= -1){
    _menuIndex = _menuSize - 1;
  }
}
```

```
MenuItem* Menu::getCurrentMenu()
{
  return _menuArr[_menuIndex];
}

void Menu::resetToDefualtMenu()
{
  _menuIndex = 0;
}

boolean Menu::isOnDefualtMenu()
{
  return _menuIndex == 0;
}

int Menu::getCurrentIndex()
{
    return _menuIndex;
}

const int Menu::getCurrentMenuIterations()
{
  MenuItem *item = _menuArr[_menuIndex];
  return item->getNumberOfInputIterations();
}
```

MenuItem.h

```
#ifndef MENU_ITEM
#define MENU_ITEM

#include <Arduino.h>

class MenuItem {
  private:
    const char* _menuName;
    const int _numberOfInputIterations;

  public:
    MenuItem(const char* menuName, const int numberOfInputIterations);
    ~MenuItem();
    const char *getName();
```

```
    void printMenu();
    const int getNumberOfInputIterations();

};

#endif
```

MenuItem.cpp

```
#include "MenuItem.h"

MenuItem::MenuItem(const char* menuName, const int numberOfInputIterations)
: _menuName(menuName), _numberOfInputIterations(numberOfInputIterations)
{
}

MenuItem::~MenuItem()
{
}

const char* MenuItem::getName()
{
  return _menuName;
}


void MenuItem::printMenu()
{
  Serial.print(_menuName);
  Serial.println();
}


const int MenuItem::getNumberOfInputIterations()
{
  return _numberOfInputIterations;
}
```

SoundManager.h

```
#ifndef SOUND_MANAGER
```

```
#define SOUND_MANAGER

#include <Arduino.h>

class SoundManager {
  private:
    const int _buzzerPinNumber;
    const int _startIndexPhase1;
    const int _startIndexPhase3;
    const int _delayLoop1;
    const int _delayLoop2;
    const int _delayBetween;

    int _index ;
    boolean _playMusic;
    int _delayCounter;
    boolean _playedTone;
    int _phase;

  public:

    SoundManager();
    ~SoundManager();
    void init();
    void playNext();
    void setPlayMusic(boolean play);
    boolean getPlayMusic();

};

#endif
```

SoundManager.cpp

```
#include <SoundManager.h>

SoundManager::SoundManager()
: _buzzerPinNumber(12), _startIndexPhase1(200), _startIndexPhase3(800),
_delayLoop1(5), _delayLoop2(10), _delayBetween(1000)
{
  _index = _startIndexPhase1;
  _playMusic = false;
  _playedTone = false;
```

```
  _phase = 0;
}

SoundManager::~SoundManager()
{
}

void SoundManager::init()
{
  pinMode(_buzzerPinNumber, OUTPUT);
}

void SoundManager::playNext()
{
  if (_playMusic){
    if (_phase == 0){
      if (_index < 800){
        if (!_playedTone){
          tone(_buzzerPinNumber,_index);
          _delayCounter = 0;
          _playedTone = true;
        }
        else{
          _delayCounter ++;
          if (_delayCounter >= _delayLoop1){
            _index++;
            _playedTone = false;
          }
        }
      }
      else
      {
        _phase = 1;
        _index = 0;
      }
    }
    else if (_phase == 1){
      _delayCounter ++;
      if (_delayCounter >= _delayBetween)
      {
        _index = _startIndexPhase3;
        _playedTone = false;
        _phase = 2;
```

```
      }
    }
    else if (_phase == 2){
      if (_index > 200){
        if (!_playedTone){
          tone(_buzzerPinNumber,_index);
          _delayCounter = 0;
          _playedTone = true;
        }
        else{
          _delayCounter ++;
          if (_delayCounter >= _delayLoop2){
            _index--;
            _playedTone = false;

          }
        }
      }
      else
      {
        _phase = 3;
      }
    }
    else{
      _phase = 0;
      _index = _startIndexPhase1;
      _playedTone = false;
    }
  }
  else {
    noTone(_buzzerPinNumber);
  }
}

void SoundManager::setPlayMusic(boolean play)
{
  _playMusic = play;
  _index = _startIndexPhase1;
  _phase = 0;
}

boolean SoundManager::getPlayMusic()
{
```

```
  return _playMusic;
}
```

פרק 4 - סיכום

סיכום:

רוב העבודה בפרויקט עם בקוד ולא ברכיבים, אומנם כעת ניתן להוסיף רכיבים נוספים יחסית בקלות ולממשק אותם למערכת הקיימת מכיוון שהכל מחולק למחלקות ולכל אחד יש את התפקיד שלו.

מסקנות והמלצות להמשך:

- רכיב הRTC אשר היה לי בעל פונקציונלית מועטת, יש רכיבים אחרים אשר נותנים את האפשר של הגדרת שעון מעורר וקבלת פסיקה מתי שזה קורה
- רכיב הLCD תופס הרבה פינים, יש מגן של lcd שכולל גם כפתורים