Burglar Alarm Project

Student Name: Istvan Nagy
Student ID: 117106574
Module code: CS3514
Other team member: Anca Faur

1. Introduction

Burglar Alarm is a project developed on an Arduino board, using additional hardware components such as an LCD, IR receiver, remote control, potentiometer, switches, resistors and a buzzer. The project simulates a real life alarm with four types of zones: entry-exit, digital, analog and continuous monitoring. Each zone is programmable and can be converted into another zone. With the help of the menus, using the remote control, the user can navigate the alarm, check the time, date, zones and history of events. He/she can modify zone attributes and stop the alarm. In order to program a zone, an engineer password is required. The project supports any type of conversion. For each type of zone there is an alarm condition, and if that condition is satisfied, the alarm will go off.

2. Requirement. Analysis. Design

2.1 What are we building?

The purpose of the project is to build a prototype for a Burglar Alarm. This alarm is required to have certain features, however the developers were encouraged to come up with new ideas for obtaining a satisfactory result.

2.2 Features and Limitations

The Burglar alarm project has the following features:

1.Time of the day (clock menu): displayed on LCD, shows the clock in real-time (hour, minutes and seconds), it can be set by the user to a valid time.

2.Date (date menu): displayed on LCD, shows the current date (year, month and day), which can be set by the user to a valid date.

3.Alarm zones: 4 zones with programmable zone types. The four zones are displayed under the zones menu. There are four types of zones: Entry-Exit, Alarm Digital, Alarm Analogue and Continuous monitoring.

- Entry/Exit: The user has the ability to set the password of the zone and the exit/enter time. If the user wants to change this zone, the system will require the password in order to apply the changes. The user has a given time to enter the password in case the zone is activated. In case the user gives the wrong password or the timer reaches zero, the alarm will go off.
- Alarm Digital: The user has the ability to set the digital zone to active HIGH or active LOW. In case the zone is activated(HIGH or LOW), the alarm will go off.
- Alarm Analogue: The user can set the threshold value for this zone. If the threshold value is exceeded, the alarm will go off.
- Continuous monitoring: The alarm will go off in case of a HIGH to LOW transition, for example in case of cutting a wire.

4.Programming the zones: The engineer can change the type of the zone. Every type of zone can be converted to another one, if the hardware is available. The system will ask for an engineer password, and if a valid password is given, a selected zone can be converted into another. The system will ask for the specific parameters in case of a conversion(password and time limit in case of entry/exit, active high or low in case of digital, threshold in case of analogue).

5.Engineer password (engineer menu): The engineer has the possibility to change the engineer password. The system will ask for the old engineer password in this case, and will let the engineer select a new password only in the case in which a good password is given.

6.Record alarm: When an alarm condition occurs, the zone, time and date of the alarm are saved in the EEPROM. These events can be viewed in the history menu.

7.History: menu containing all the alarms which occurred. Each alarm is described by an event, containing the zone number, the time and the date when the alarm went off. The user has the possibility to delete the history, after which the history will be empty and the deleted data cannot be recovered.

8.Saving the settings: The settings for the zones are saved to EEPROM each time the program is loaded and also when a zone is converted to another zone. Besides this, the engineer password is also saved in the EEPROM memory.

9.Remote control: is used to navigate all the menus, entering the entry/exit password, entering the engineer password and stopping the alarm.

Limitations:

- Size of the EEPROM (512 bytes). This gives a limitation in the number of events which can be stored if an alarm condition occurs.
- SRAM (2K). The program uses only the memory needed to store the data structures and perform the necessary calculations.

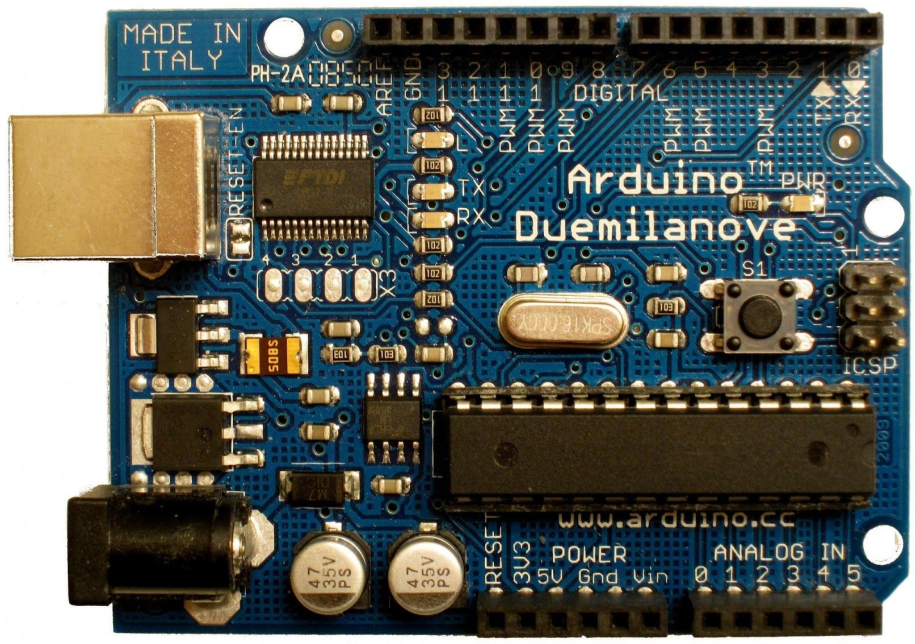
- Flash (32K). The program occupies 40% of the available memory.

2.3 Major functional blocks

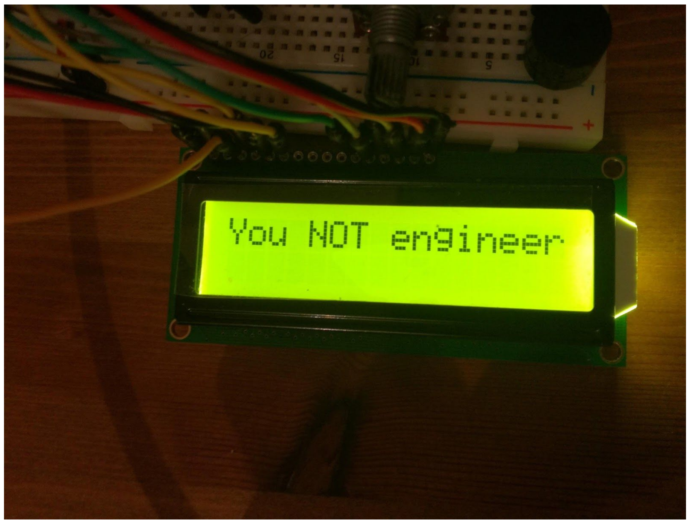
1. **Arduino Duemilanove**: microcontroller board based on the ATmega168 processor. Microcontrollers are “small computers” on a single-integrated circuit containing a single processor core, memory (Flash, SRAM and EEPROM) and programmable input/outputs peripherals.

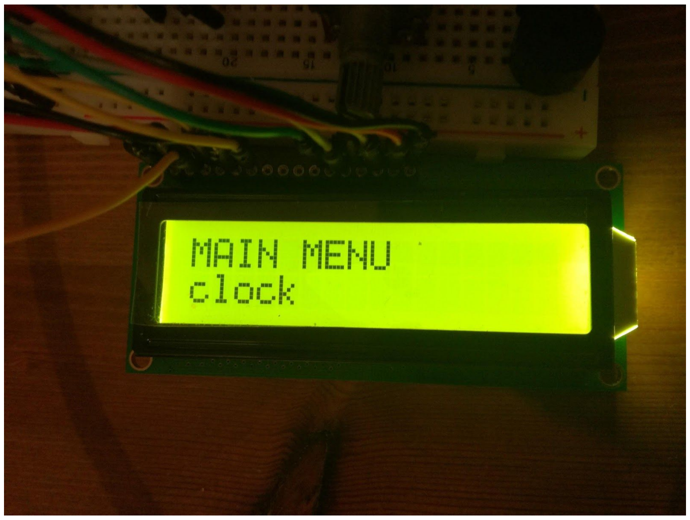
Our project uses all types of memory available on the microcontroller, the Flash memory is used for storing sketch of our program, the SRAM memory is used in runtime when manipulating different variables and we are using the EEPROM in order to safely save the alarm events and the configuration of our zones when the microcontroller is powered off (non-volatile). The inputs pins are used when simulating different sensors inputs (we used switches and circuit settings to simulate alarm sensors) and the Arduino outputs are used for displaying the user interface on the LCD screen and for controlling a buzzer which represents our alarm alert.

For more information about the board:
https://www.arduino.cc/en/Main/ArduinoBoardDuemilanove.



2. **LCD LCM1602C:** 16x2 characters display used for creating a friendly user interface. Spreadsheet available on:
http://www.datasheetcafe.com/lcm1602c-datasheet-pdf

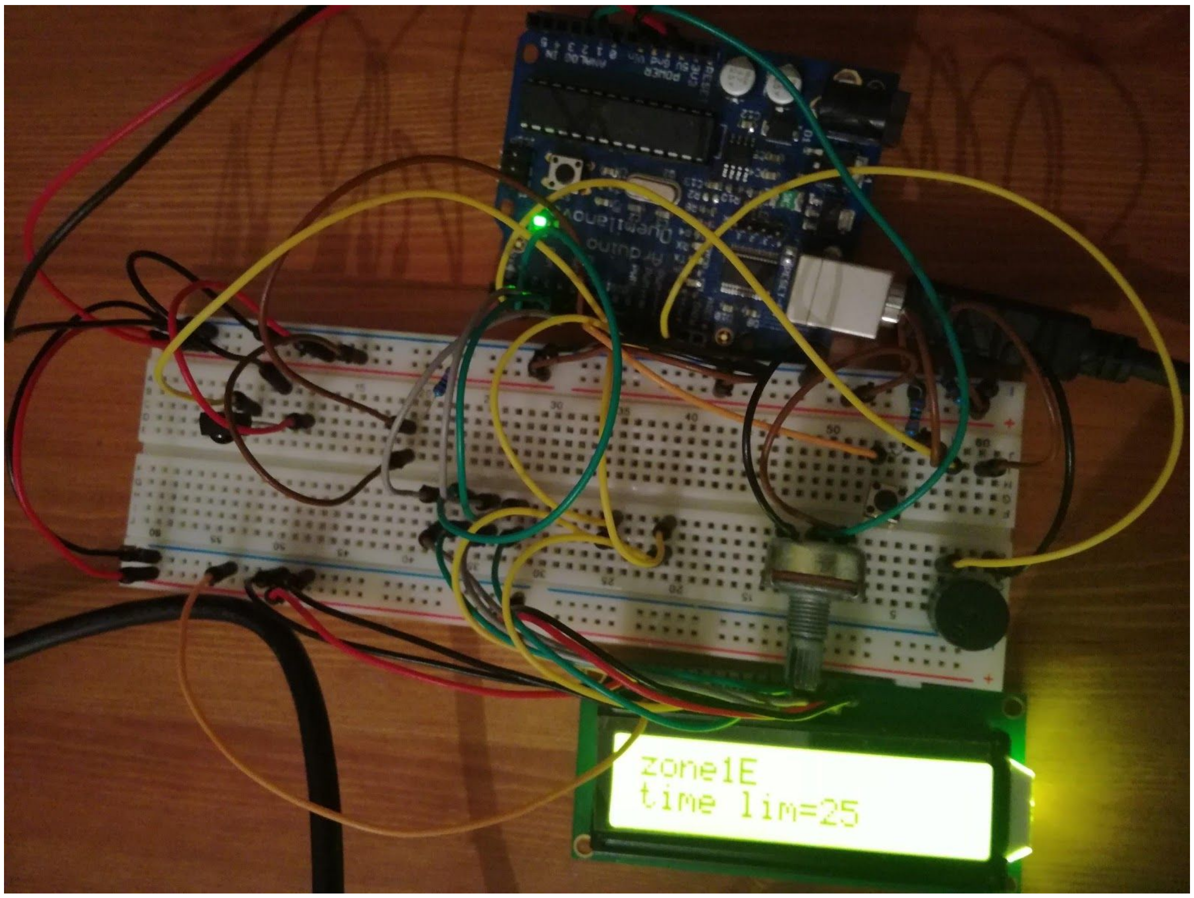




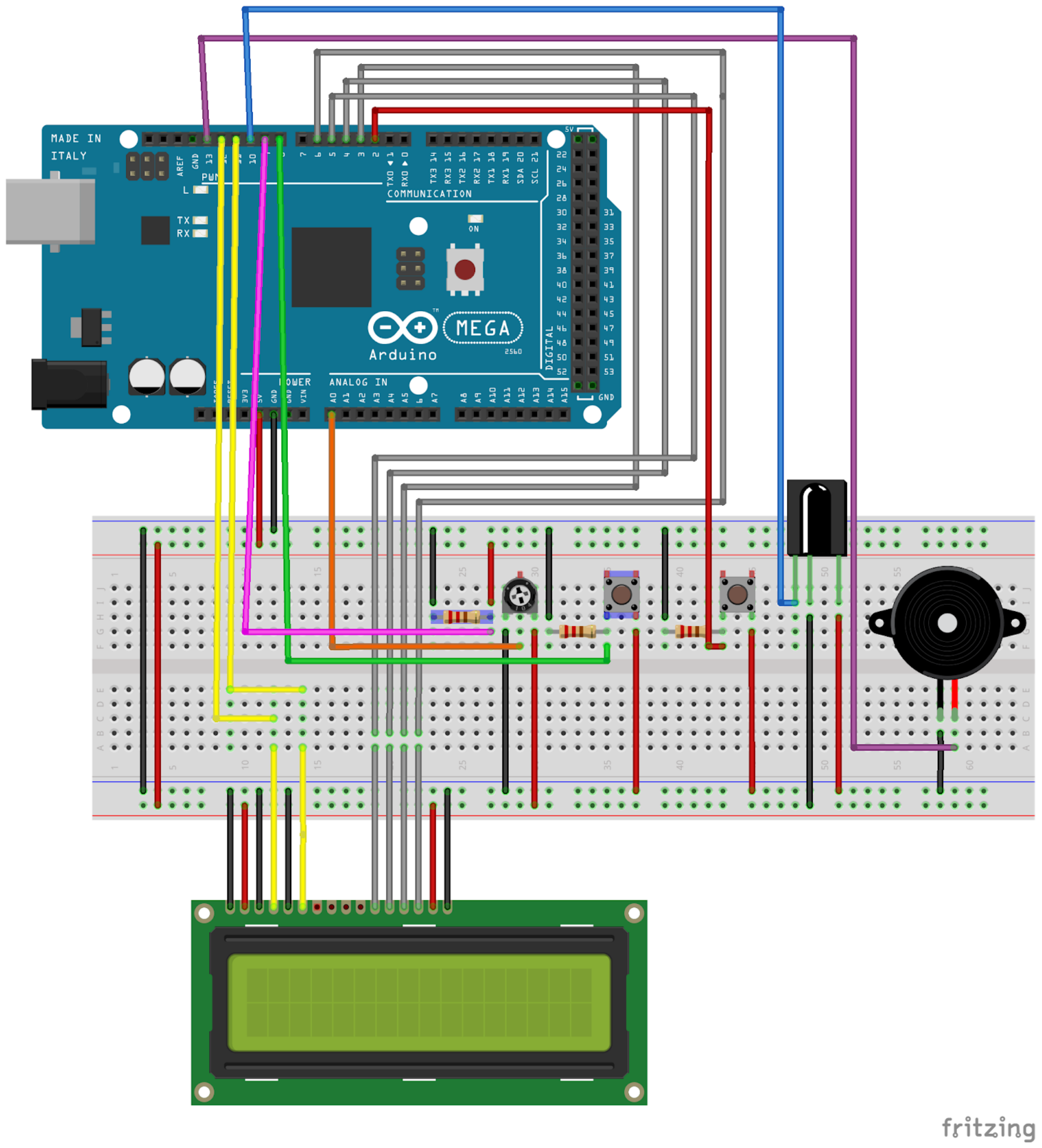
3. **Infrared remote controller**: The remote controller's LED emits infrared light according to a protocol when the user presses any key on the 25 available on the remote. A photo sensor reads the incoming IR data and transforms it in a binary waveform that can be read by the Arduino using an analog pin. The remote is also used for enabling the user to easily interact with the system. The simple convention for our key-choice is:

EQ = Enter
Mode = Require Setting (eg. for Date, Clock, Zone parameters)
Plus = More Advanced Setting (engineer can change the zone type)
Minus = Clear saved events (clear the history of alarm events)
Play = Next Option (cycling all menus)
Re-play = Returning to Main Menu

Picture of the whole circuit



2.4 Circuit diagram



2.5 Pc Connected

The project needs to be connected to the PC in order to have a power source for the Arduino board.

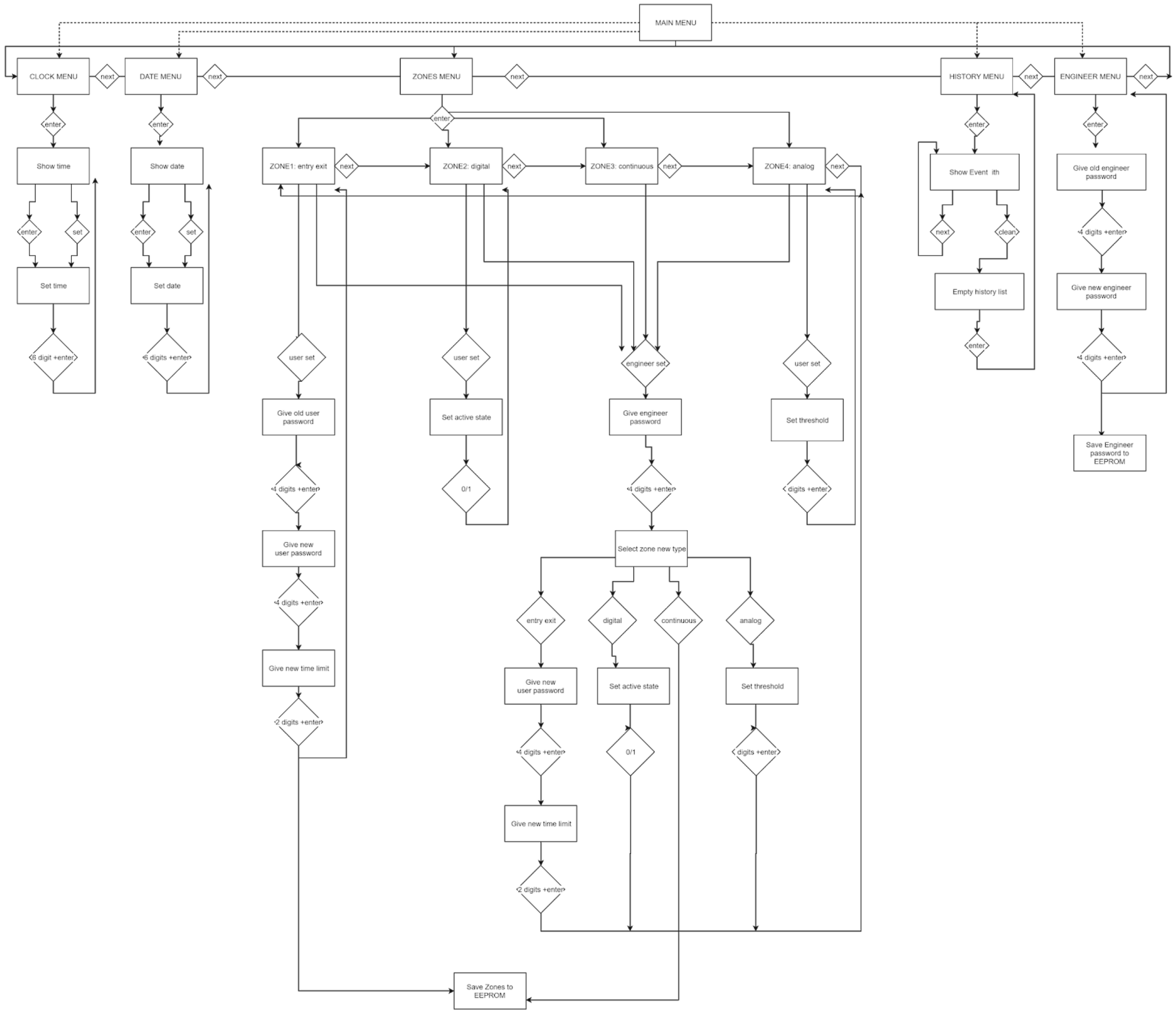
2.6 Memory usage

Sketch uses 15278 bytes 49% of maximum 30720.
Global variables use 1019 bytes(49% of dynamic memory), 1029 bytes left to local variables.

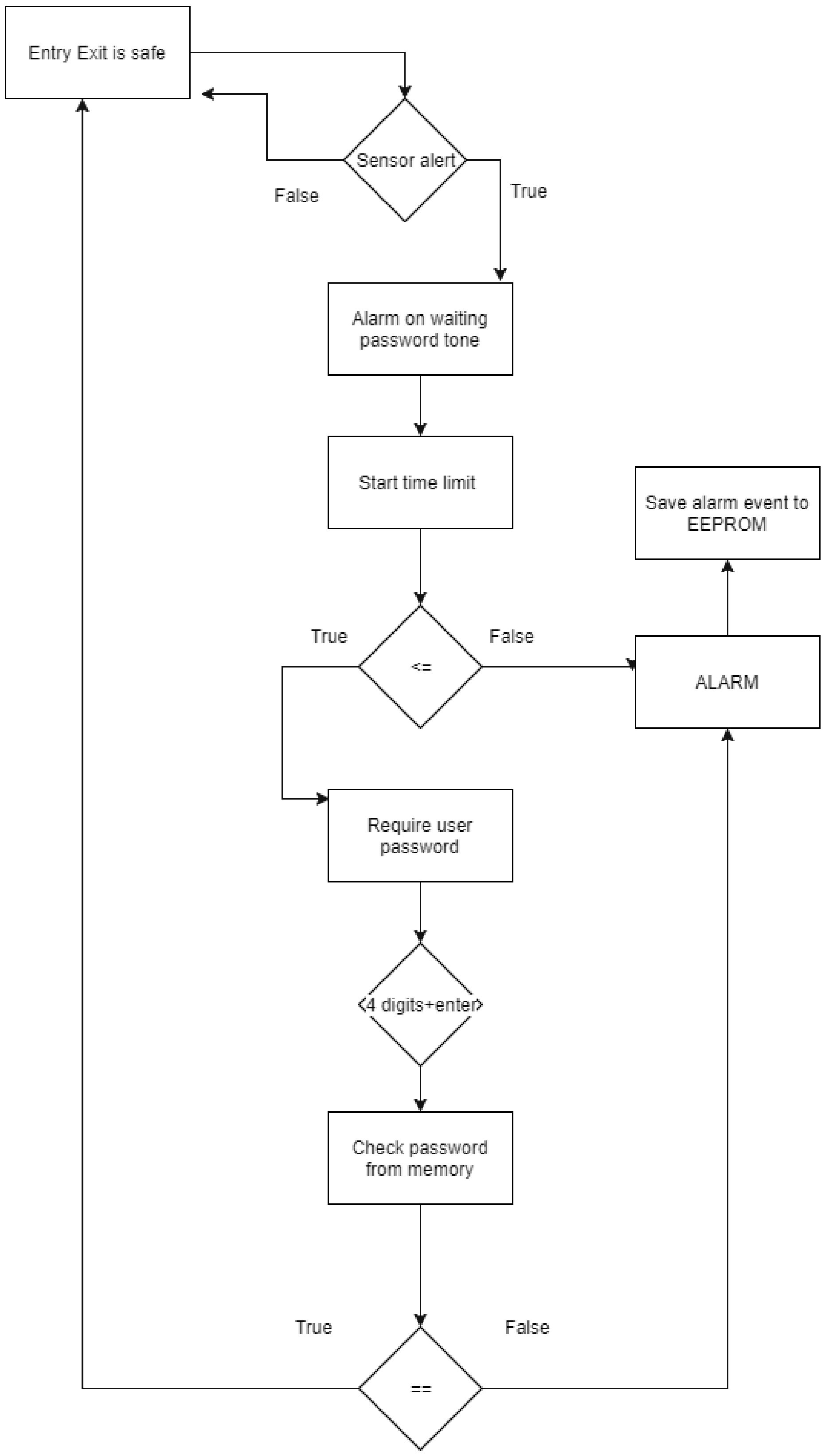
3. Implementation

3.1 Flow Chart

3.1.a. Flow Chart explaining the user and engineer features



3.1.b. Flow Chart explaining the behavior of the Entry Exit Zone



3.2 Coding Decisions

Setup Function

The setup function is the function which will be executed only once when the program is loaded. In the setup function the following settings are present:

1.Read the zones from the EEPROM memory: is a function which reads the four zones previously saved in memory. It uses the EEPROM.read() function and reads byte by byte every value, and from the read bytes it updates the array of structures which is basically the list of all the zones.

2. Read in the engineer password from EEPROM: another function which reads in the password of the engineer from a well defined position in the EEPROM memory.

3. Begin the LCD with 16 columns and 2 rows.

4. Set the pins to be either INPUT or OUTPUT pins.

5. Start the timer which generates an interrupt whenever the value of the timer equals the value of the compare register. Basically, the compare register is set in such a way, that the timer will generate an interrupt every second, thus having a clock.

6. Enable the IR receiver to receive button presses.

Loop Function

The loop function is the function which gets executed over and over again. Basically, it does three important steps on each iteration, which are described below.

1.Print on LCD: the function takes care of printing on the LCD. It checks the current menu by a switch and prints a corresponding message on the LCD.

Example of printing on LCD in case of the main menu:
case MENU_MAIN:

```
lcd.print("MAIN MENU");
lcd.setCursor(0, 1);
switch (menuSelect) {
  case MENU_CLOCK: lcd.print("clock"); break;
  case MENU_DATE : lcd.print("date");  break;
  case MENU_ZONES: lcd.print("zones"); break;
  case MENU_HISTORY: lcd.print("history"); break;
  case MENU_ENGINEER: lcd.print("engineer"); break;
}
break;
```

2.Perform task: updates the time of the system and in case a button was pressed on the remote control, it will either make a menu transition or receive a digit (in case of entering a password).

Example of perform task in case of the zones menu, where we can switch between zones with the PLAY button, we can configure a zone with the MODE button, we can go back with the REW button and we can program the zone with the PLUS button. Here, all the menu transitions are specified.

```
 case MENU_ZONES:
       if (results.value == IR_NEXT_PLAY) {
         zoneSelect++;
         lcd.clear();
         if (zoneSelect >= MAX_ZONES + ZONE_1)
           zoneSelect = ZONE_1;
       }
       else if (results.value == IR_SET_MODE) {
         lcd.clear();
         index = zoneSelect - ZONE_1;
         switch (zones[index].type) {
           case 'E': menu = MENU_REQUIRE_PASSWORD; break;
           case 'A': menu = MENU_SET_THRESHOLD; break;
           case 'D': menu = MENU_SET_DIGITAL; break;
           default: menu = MENU_MAIN;
         }
       }
       else if (results.value == IR_AGAIN_REW) {
         menu = MENU_MAIN;
         lcd.clear();
       }
       else if (results.value == IR_CHANGE_PLUS) {
         index = zoneSelect - ZONE_1;
         settingZone = index;
         menu = MENU_REQUIRE_ENG_PASS;
         lcd.clear();
       }
       break;
```

3.Check zones: function which iterates over all zones and depending on the zone type, it will check whether or not an alarm condition occurred. If so it will trigger the alarm and save the event to the EEPROM memory.

Example of checking an analogue type zone. We read the value from the sensor, and if the value is greater than the threshold, we activate the alarm by going to the MENU_ALARM and save the event which occurred in the EEPROM.

```
if (zones[i].type == 'A') {
        int val = analogRead(zones[i].pin);
        if (val >= zones[i].value) {
          if (!setAlarm) {
            menu = MENU_ALARM;
            setAlarm = true;
            saveEventToEEPROM(i);
            lcd.clear();
            alarmZone = i;
          }
        }
      }
```

Define Statements

We decided to use define statements to make our life easy. They were really important part of our code when we were creating all the different menus. Each menu is described by a unique identifier which is used in the switch block to identify it.

Also, we used them to define the pins on the arduino, the pins on the LCD, the values received by the remote control when decoding,

An example of how we used define statements in our code can be seen below, in case of the menus.

```
#define MAX_MENUS 5
#define MENU_MAIN 0
#define MENU_CLOCK 1
#define MENU_DATE 2
#define MENU_ZONES 3
#define MENU_HISTORY 4
#define MENU_ENGINEER 5
#define MENU_SET_TIME 11
#define MENU_SET_DATE 12
#define MAX_ZONES 4
#define ZONE_1 21
#define ZONE_2 22
#define ZONE_3 23
#define ZONE_4 24
#define MENU_ALARM 30
#define MENU_GIVE_PASSWORD 31
#define MENU_REQUIRE_PASSWORD 32
#define MENU_REQUIRE_ENG_PASS 33
#define MENU_SET_PASS 34
#define MENU_SET_LIMIT 35
#define MENU_SET_THRESHOLD 36
#define MENU_SET_DIGITAL 37
```

```
#define MENU_SET_ENG_PASS 38
#define MENU_SELECT_TYPE 40
```

Structures

We used two structures for storing the time and the date.

```
struct timeStruct {
  byte hour = 0;
  byte minutes = 0;
  byte seconds = 0;
} myTime;

struct dateStruct {
  byte year = 17;
  byte month = 11;
  byte day = 27;
} myDate;
```

Reading and Writing to/from the EEPROM

In many parts of the code we have to read or write data to the EEPROM. There are several functions which implement this functionality: saveEventToEEPROM(), readEventFromEEPROM(), saveZonesToEEPROM(), readZonesFromEEPROM(), saveEngPassToEEPROM(), readEngPassToEEPROM(). They all access the EEPROM memory at a well specified offset and use the EEPROM.read() and EEPROM.write() functions.
Here is an example of reading an event from the EEPROM, a function which takes a pointer to an event which will be changed, and also an offset which represents the address in memory from where the reading process starts.

```
void readEventFromEEPROM(struct eventStruct* event, int offset) {
  event->zone = EEPROM.read(offset);
  event->alarm_time.hour = EEPROM.read(offset + 1);
  event->alarm_time.minutes = EEPROM.read(offset + 2);
  event->alarm_time.seconds = EEPROM.read(offset + 3);
  event->alarm_date.year = EEPROM.read(offset + 4);
  event->alarm_date.month = EEPROM.read(offset + 5);
  event->alarm_date.day = EEPROM.read(offset + 6);
}
```

3.3 Innovative code pieces

Structure for a zone

We used a structure for a zone. It stores the:

- pin number - byte
- the type (char from the set : {‘a’, ‘d’, ‘e’, ‘c’}, representing analog, digital, entry-exit and continuous monitoring)
- Value which is an int : it represents the threshold in case of analogue zone, the active HIGH or LOW property in case of digital and entry-exit, the transition which activates the alarm in case of continuous.
- a pointer to another structure which only applies if the zone is of type ‘e’(entry-exit). If not, the pointer is left NULL and it is not used. The param is of type entryStruct which contains a time limit(byte) and a password(array of four bytes).

Below is the structure, which also defines default values for the fields.

```
struct zone {
  byte pin = BUTTON1;
  char type = 'D';
  int value = 0;
  entryStruct *param = NULL;
};
```

Reallocating memory

Every time the user wants to access the memory, the space for the array of events is reallocated. This is because the array can change dynamically in case we are deleting events or in the case we trigger alarms. We don’t know the exact number of alarms that occurred at a given time, so we have to realloc the space. After the space is reallocated, we read the events from the EEPROM one by one.

Here is the piece of code that does this functionality.

```
byte eventNumber = EEPROM.read(EVENTS_OFFSET);
events = (struct eventStruct*) realloc(events, eventNumber * sizeof(struct eventStruct));
for (int i = 0; i < eventNumber; i++) {
readEventFromEEPROM(&events[i], EVENTS_OFFSET + 1 +i * sizeof(struct eventStruct));
}
```

4. Evaluation

4.1 How good is the project ?

For the purpose of the assignment, that of introducing us to microcontroller programing, our project is a success. Even if it is a prototype, it has pretty advanced features, it is use friendly with the help of the LCD and the remote control and the results (alarming situations) are reliable.

4.2 Known Problems and Improvements

4.2.1. Known Problems

As this project is a prototype of a burglar alarm which has the purpose of introducing us to microcontrollers programming, we didn't aim to obtaining a perfect behavior. That's the reason of the date defect of our system. Even if our user is allowed to change the date and the alarm memorizes its input, the program we have written doesn't change the date when the hour reaches 00:00. We didn't solve this problem as we noticed that there are many conditions (month of 30 days or 31 days, how many days has February in the current year?).

In this version of the project, the date and the clock are not saved to EEPROM so the program must be always reconfigured by the user if the system is turned off. We just didn't think about saving these on the EEPROM as well.

A problem may occur if the user leaves more than approximately 60 alarm events to be saved to the EEPROM. The EEPROM memory would've been consumed and we would try to write on a nonexistent memory address. This problem can be theoretically solved by cycling the history event EEPROM memory, but this solution needs more thinking to fit in our design as we are saving the number of events and then events structures in that area.

Another defect of our current project is the hardware setting. As the components belong to a Beginner Arduino kit, the "good for a lot, but excellent at nothing" rule applies . Some components cannot be matched perfectly and these cause our setting to be very sensitive. For example, our switches keep jumping on the breadboard and our potentiometer should be carefully positioned in order to close the circuit.

4.2.1. Improvements

Our project just simulates the existence of sensors using switches, one wire on the breadboard and a simple potentiometer. In order to be able to build-up a reliable program for a alarm, that program should be tested in a real-life environment. The interaction between the environment and the sensors, the possible interaction between different the sensors are very important in order to evaluate how successful is your alarm.

Other improvements ideas:

- Make alarm send a message to the Master when alarm is triggered
- Make alarm call the Garda Station automatically

- Allow user to select the tone of the alarm
- Allow user to temporarily set off one of the alarming zones
- Send user photos of the monitoring area in case the alarm is triggered
- Power on the Arduino using a battery

5. Conclusion

Together with my teammate, we managed to implement the basic and the enhanced functionalities of this project. We learnt how to structure a bigger project like this, decomposing the tasks into steps and menus and combining the hardware and the software. The most interesting part of the project was reprogramming a zone. Our project supports any type of conversion, from a selected zone to any other zone type. Moreover, we recorded every event that occured in the EEPROM memory, and we enjoyed playing with the history of events, which is accessible to the user via a menu and can be deleted anytime.

We tried to make the project very user friendly, implementing plenty of menus and messages such as: “Changes saved”, “Hello master!”, “You NOT engineer”. From the start of the project, we tried to emphasise the functionality of our project, by implementing enhanced functionalities in a usable way.

One thing we were not satisfied with was the code of our project. The loop function has a logical structure, but some functions like performTask, which takes in the user input from the remote control and makes the transitions between menus, are very long and complex. Another thing that made our life difficult was that we implemented everything in one source file. As the project got bigger, it was harder to add new functionality and to maintain it.

5. Appendices

Code:

```
#include <avr/interrupt.h>
#include <EEPROM.h>

#define MAX_BOUNCE_TIME 5

// alarm pins (for the reading the "sensors" and controlling the buzzer )
#define BUTTON1 8
#define BUTTON2 2
#define BUZZ_PIN 13
#define CONTINUOUS_PIN 7
#define ANALOG_PIN 0

//---------LCD---------
#include <LiquidCrystal.h>
#define LCD_RS 12
#define LCD_EN 11
#define LCD_D4 5
#define LCD_D5 4
#define LCD_D6 3
#define LCD_D7 6
LiquidCrystal lcd(LCD_RS, LCD_EN, LCD_D4, LCD_D5, LCD_D6, LCD_D7);

//---------Remote control---------
#include <IRremote.h>
#define RECV_PIN 10
#define  IR_0        0xff6897
#define  IR_1        0xff30cf
#define  IR_2        0xff18e7
#define  IR_3        0xff7a85
#define  IR_4        0xff10ef
#define  IR_5        0xff38c7
#define  IR_6        0xff5aa5
#define  IR_7        0xff42bd
#define  IR_8        0xff4ab5
#define  IR_9        0xff52ad
#define  IR_DELETE_MINUS     0xffe01f
#define  IR_CHANGE_PLUS      0xffa857
#define  IR_ENTER_EQ  0xff906f
#define  IR_ON_OFF    0xffa25d
#define  IR_SET_MODE  0xff629d
```

```
#define  IR_MUTE      0xffe21d
#define  IR_NEXT_PLAY 0xffc23d
#define  IR_AGAIN_REW 0xff22dd
#define  IR_FF        0xff02fd
IRrecv irrecv(RECV_PIN);
decode_results results;

int keyToInt(int key_pressed) {
  switch (key_pressed) {
    case IR_0: return 0;
    case IR_1: return 1;
    case IR_2: return 2;
    case IR_3: return 3;
    case IR_4: return 4;
    case IR_5: return 5;
    case IR_6: return 6;
    case IR_7: return 7;
    case IR_8: return 8;
    case IR_9: return 9;
    default: return -1;
  }
}

//counter, used for counting down from a given value
volatile int counter = 0;
//counts the number of seconds using a timer interrupt
volatile int sec = 0;

// flags
volatile boolean countDown = false;
volatile boolean setAlarm = false;
boolean changeEngPass = false;

// structures
struct entryStruct {
  byte timeLimit;
  byte password[4];
};

struct zone {
  byte pin = BUTTON1;
  char type = 'D';
  int value = 0;
  entryStruct *param = NULL;
};
struct zone zones[4];

struct timeStruct {
  byte hour = 0;
  byte minutes = 0;
  byte seconds = 0;
} myTime;

struct dateStruct {
  byte year = 17;
  byte month = 11;
  byte day = 27;
} myDate;

struct eventStruct {
  byte zone;
  timeStruct alarm_time;
  dateStruct alarm_date;
};
struct eventStruct* events;

//-----MENUS---------
#define MAX_MENUS 5

#define MENU_MAIN 0
#define MENU_CLOCK 1
#define MENU_DATE 2
#define MENU_ZONES 3
#define MENU_HISTORY 4
#define MENU_ENGINEER 5

#define MENU_SET_TIME 11
#define MENU_SET_DATE 12

#define MAX_ZONES 4
#define ZONE_1 21
#define ZONE_2 22
#define ZONE_3 23
#define ZONE_4 24

#define MENU_ALARM 30
```

```
#define MENU_GIVE_PASSWORD 31
#define MENU_REQUIRE_PASSWORD 32
#define MENU_REQUIRE_ENG_PASS 33
#define MENU_SET_PASS 34
#define MENU_SET_LIMIT 35
#define MENU_SET_THRESHOLD 36
#define MENU_SET_DIGITAL 37

#define MENU_SET_ENG_PASS 38
#define MENU_SELECT_TYPE 40

// used in menus navigation
int menu = 0;
int menuSelect = MENU_CLOCK;
int zoneSelect = ZONE_1;
int eventSelect = 0;

// used to hold the engineer current password
byte eng_pass[4] = {0, 0, 0, 0};
// used whenever we are receiving user digits input
byte numbers[6] = {0, 0, 0, 0, 0, 0};
// used in printToLcd function for telling which alarm zone is triggered
byte alarmZone = -1;

// used for dissabling one zone checking when the zone is under setting
int settingZone = -1;

// offsets in EEPROM
#define EVENTS_OFFSET 36
#define ENG_OFFSET 508

// read engineer password from EEPROM and save it to a global array
void readEngFromEEPROM() {
  for (int i = 0; i < 4; i++) {
    eng_pass[i] = EEPROM.read(ENG_OFFSET + i);
  }
}

// save engineer password to EEPROM (when changed)
void saveEngToEEPROM() {
  for (int i = 0; i < 4; i++) {
    EEPROM.write(ENG_OFFSET + i, eng_pass[i]);
  }
```

```
}

// read one event from the EEPROM from given address
void readEventFromEEPROM(struct eventStruct* event, int offset) {
  event->zone = EEPROM.read(offset);
  event->alarm_time.hour = EEPROM.read(offset + 1);
  event->alarm_time.minutes = EEPROM.read(offset + 2);
  event->alarm_time.seconds = EEPROM.read(offset + 3);
  event->alarm_date.year = EEPROM.read(offset + 4);
  event->alarm_date.month = EEPROM.read(offset + 5);
  event->alarm_date.day = EEPROM.read(offset + 6);
}

// create alarm event for zone i and save it to EEPROM
void saveEventToEEPROM(int i) {
  struct eventStruct event;
  byte eventNumber;
  int offset;
  eventNumber = EEPROM.read(EVENTS_OFFSET);
  event.zone = i + 1;
  event.alarm_time.hour = myTime.hour; event.alarm_time.minutes = myTime.minutes;
event.alarm_time.seconds = myTime.seconds;
  event.alarm_date.year = myDate.year; event.alarm_date.month = myDate.month;
event.alarm_date.day = myDate.day;
  offset = EVENTS_OFFSET + 1 + eventNumber * sizeof(struct eventStruct);
  eventNumber++;
  EEPROM.write(EVENTS_OFFSET, eventNumber);

  EEPROM.write(offset, event.zone);
  EEPROM.write(offset + 1, event.alarm_time.hour);
  EEPROM.write(offset + 2, event.alarm_time.minutes);
  EEPROM.write(offset + 3, event.alarm_time.seconds);
  EEPROM.write(offset + 4, event.alarm_date.year);
  EEPROM.write(offset + 5, event.alarm_date.month);
  EEPROM.write(offset + 6, event.alarm_date.day);
}

// read Zones from EEPROM - used in setup() function
int readZonesFromEEPROM() {
  int offset = 0;
  for (int i = 0; i < 4; i++) {
    zones[i].pin = EEPROM.read(offset);
    zones[i].type = EEPROM.read(offset + 1);
    zones[i].value = word(EEPROM.read(offset + 3), EEPROM.read(offset + 2));
    offset += 4;
    if (zones[i].type == 'E') {
      zones[i].param = (struct entryStruct*)malloc(sizeof(struct entryStruct));
      zones[i].param->timeLimit = EEPROM.read(offset);
      offset++;
      for (int j = 0; j < 4; j++) {
        zones[i].param->password[j] = EEPROM.read(offset);
        offset++;
      }
    }
  }
  return offset;
}

// save all Zones to EEPROM, required whenever one zone has been changed
// all require saving in the case a zone changes type from A/D/C to E or vice-versa
void saveZonesToEEPROM() {
  int offset = 0;
  for (int i = 0; i < 4; i++) {
    Serial.println(offset);
    EEPROM.write(offset, zones[i].pin);
    EEPROM.write(offset + 1, zones[i].type);
    EEPROM.write(offset + 2, lowByte(zones[i].value));
    EEPROM.write(offset + 3, highByte(zones[i].value));
    offset += 4;
    if (zones[i].type == 'E') {
      EEPROM.write(offset, zones[i].param->timeLimit );
      offset++;
      for (int j = 0; j < 4; j++) {
        EEPROM.write(offset, zones[i].param->password[j]);
        offset++;
      }
    }
  }
}

void setup() {
  Serial.begin(9600);
  readZonesFromEEPROM();

  //saveEngToEEPROM();
  readEngFromEEPROM();
  Serial.print(eng_pass[0]); Serial.print(eng_pass[1]); Serial.print(eng_pass[2]);
Serial.print(eng_pass[3]);

  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  pinMode(BUTTON1, INPUT);
  pinMode(BUTTON2, INPUT);
  pinMode(BUZZ_PIN, OUTPUT);
  pinMode(CONTINUOUS_PIN, INPUT);
  pinMode(ANALOG_PIN, INPUT);

  //clear the interrupts
  cli();
  //set the timer
  TCCR1A = 0;
  TCCR1B = 0;
  OCR1A = 15625;
  TCCR1B |= (1 << WGM12);
  TCCR1B |= (1 << CS10);
  TCCR1B |= (1 << CS12);
  TIMSK1 |= (1 << OCIE1A);
  sei();

  //enable IR
  irrecv.enableIRIn();
}

// alarm
void playSong() {
  digitalWrite(BUZZ_PIN, HIGH);
}

// correct display of numbers on the LCD
void printNumber(int number) {
  if (number < 10) {
    lcd.print("0");
  }
  lcd.print(number);
}

// history event show
void printEvent(struct eventStruct event) {
  lcd.setCursor(0, 0);
  lcd.print("zone:");
  lcd.print(event.zone);
  lcd.print(" ");
  printNumber(event.alarm_time.hour);
  lcd.print(":");
  printNumber(event.alarm_time.minutes);
  lcd.print(":");
  printNumber(event.alarm_time.seconds);
  lcd.setCursor(0, 1);
  printNumber(event.alarm_date.year);
  lcd.print("/");
  printNumber(event.alarm_date.month);
  lcd.print("/");
  printNumber(event.alarm_date.day);
}

// main function that manages the LCD print with respect to current menu
void printOnLCD() {
  int index;
  lcd.setCursor(0, 0);
  switch (menu) {
    case MENU_CLOCK:
      lcd.print("CLOCK:");
      lcd.setCursor(0, 1);
      printNumber(myTime.hour);
      lcd.print(":");
      printNumber(myTime.minutes);
      lcd.print(":");
      printNumber(myTime.seconds);
      break;
    case MENU_DATE:
      lcd.print("DATE:");
      lcd.setCursor(0, 1);
      printNumber(myDate.year);
      lcd.print("/");
      printNumber(myDate.month);
      lcd.print("/");
      printNumber(myDate.day);
      break;
    case MENU_SET_TIME:
      lcd.print("SET TIME");
      lcd.setCursor(0, 1);
      for (int j = 0; j < 6; j++) {
        lcd.print(numbers[j]);
        if (j == 1 || j == 3) lcd.print(":");
      }
      break;
    case MENU_SET_DATE:
      lcd.print("SET DATE");
      lcd.setCursor(0, 1);
      for (int j = 0; j < 6; j++) {
        lcd.print(numbers[j]);
        if (j == 1 || j == 3) lcd.print("/");
      }
      break;
    case MENU_ZONES:
      index = zoneSelect - ZONE_1;
      lcd.print("zone"); lcd.print(index + 1); lcd.print(zones[index].type);
      lcd.setCursor(0, 1);
      switch (zones[index].type) {
        case 'E': lcd.print("time lim="); lcd.print(zones[index].param->timeLimit); break;
        case 'D': lcd.print("active="); lcd.print(zones[index].value); break;
        case 'A': lcd.print("threshold="); lcd.print(zones[index].value); break;
        case 'C': lcd.print("continuous watch"); break;
      }
      break;
    case MENU_HISTORY:
      if (EEPROM.read(EVENTS_OFFSET) > 0)
        printEvent(events[eventSelect]);
      else
        lcd.print("empty");
      break;
    case MENU_MAIN:
      lcd.print("MAIN MENU");
      lcd.setCursor(0, 1);
      switch (menuSelect) {
        case MENU_CLOCK: lcd.print("clock"); break;
        case MENU_DATE : lcd.print("date");  break;
        case MENU_ZONES: lcd.print("zones"); break;
        case MENU_HISTORY: lcd.print("history"); break;
        case MENU_ENGINEER: lcd.print("engineer"); break;
      }
      break;
    case MENU_ALARM:
      lcd.print("ALARM!!!");
```

```
      lcd.setCursor(0, 1);
      lcd.print("zone ");
      lcd.print(alarmZone + 1);
      break;
    case MENU_GIVE_PASSWORD:
      lcd.print("Enter Pass ");
      printNumber(counter);
      break;
    case MENU_REQUIRE_PASSWORD:
      lcd.print("Enter Pass ");
      break;
    case MENU_REQUIRE_ENG_PASS:
      lcd.print("Engineer pass");
      break;
    case MENU_SET_PASS:
      lcd.print("Give new pass ");
      break;
    case MENU_SET_ENG_PASS:
      lcd.print("New eng pass ");
      break;
    case MENU_SET_LIMIT:
      lcd.print("Give time limit ");
      break;
    case MENU_SET_THRESHOLD:
      lcd.print("Give threshold");
      break;
    case MENU_SET_DIGITAL:
      lcd.print("Set active 0/1");
      break;
    case MENU_SELECT_TYPE:
      lcd.print("Select type");
      lcd.setCursor(0, 1);
      lcd.print("0=D 1=E 2=C 3=A");
      break;
    default: break;
  }
}

// state machine where states are menus; transition are driven by user input
void performTask() {
  myTime.seconds = sec;
  if (myTime.seconds >= 60) {
    myTime.seconds = 0;
    myTime.minutes++;
    sec = 0;
  }
  if (myTime.minutes >= 60) {
    myTime.minutes = 0;
    myTime.hour++;
  }
  if (myTime.hour >= 24) {
    myTime.hour = 0;
    myTime.minutes = 0;
    myTime.seconds = 0;
  }
  if (menu == MENU_ALARM) {
    playSong();
  }
  if (irrecv.decode(&results)) {
    Serial.println("key pressed");
    int digit;
    static int index = 0;
    switch (menu) {
      case MENU_MAIN:
        if (results.value == IR_NEXT_PLAY) {
          menuSelect++;
          lcd.clear();
          if (menuSelect > MAX_MENUS)
            menuSelect = MENU_CLOCK;
        }
        else if (results.value == IR_ENTER_EQ) {
          if (menuSelect == MENU_HISTORY) {
            eventSelect=0;
            byte eventNumber = EEPROM.read(EVENTS_OFFSET);
            events = (struct eventStruct*) realloc(events, eventNumber * sizeof(struct eventStruct));
            for (int i = 0; i < eventNumber; i++) {
              readEventFromEEPROM(&events[i], EVENTS_OFFSET + 1 + i * sizeof(struct
eventStruct));
            }
          }
          if (menuSelect == MENU_ENGINEER) {
            changeEngPass = true;
            menu = MENU_REQUIRE_ENG_PASS;
          } else menu = menuSelect;

          lcd.clear();
        }
        break;
      case MENU_ZONES:
        if (results.value == IR_NEXT_PLAY) {
          zoneSelect++;
          lcd.clear();
          if (zoneSelect >= MAX_ZONES + ZONE_1)
            zoneSelect = ZONE_1;
        }
        else if (results.value == IR_SET_MODE) {
          lcd.clear();
          index = zoneSelect - ZONE_1;
          switch (zones[index].type) {
            case 'E': menu = MENU_REQUIRE_PASSWORD; break;
            case 'A': menu = MENU_SET_THRESHOLD; break;
            case 'D': menu = MENU_SET_DIGITAL; break;
            default: menu = MENU_MAIN;
          }
        }
        else if (results.value == IR_AGAIN_REW) {
          menu = MENU_MAIN;
          lcd.clear();
        }
        else if (results.value == IR_CHANGE_PLUS) {
          index = zoneSelect - ZONE_1;
          settingZone = index;
          menu = MENU_REQUIRE_ENG_PASS;
          lcd.clear();
        }
        break;
      case MENU_SELECT_TYPE:
        switch (results.value) {
          case IR_0:
            zones[index].type = 'D';
            zones[index].param = NULL;

            menu = MENU_SET_DIGITAL;
            lcd.clear();
            break;

          case IR_1:
            zones[index].type = 'E';
            zones[index].param = (struct entryStruct*) malloc(sizeof(entryStruct));
            zones[index].value = 1;
            menu = MENU_SET_PASS;
            lcd.clear();
            break;

          case IR_2:
            zones[index].type = 'C';
            zones[index].param = NULL;
            zones[index].value = 0;
            menu = MENU_ZONES;
            settingZone = -1;
            lcd.clear();
            saveZonesToEEPROM();
            break;

          case IR_3:
            if (zones[index].pin != ANALOG_PIN) {
              lcd.clear();
              lcd.print("HW only zone4");
              delay(1000);
              lcd.clear();
              menu = MENU_ZONES;
            } else {
              zones[index].type = 'A';
              zones[index].param = NULL;
              menu = MENU_SET_THRESHOLD;
              lcd.clear();
            }
            break;
        }
        break;
      case MENU_HISTORY:
        if (results.value == IR_ENTER_EQ) {
          menu = MENU_MAIN;
          lcd.clear();
        }
        else if (results.value == IR_NEXT_PLAY) {
          eventSelect++;
          if (eventSelect >= EEPROM.read(EVENTS_OFFSET))
            eventSelect = 0;
        }
        else if (results.value == IR_DELETE_MINUS) {
          EEPROM.write(EVENTS_OFFSET, 0);
```

```
      lcd.clear();
      lcd.print("history deleted");
      delay(1000);
      lcd.clear();
      menu = MENU_MAIN;
      eventSelect=0;
    }
    break;
  case MENU_CLOCK:
    if (results.value == IR_ENTER_EQ) {
      menu = MENU_MAIN;
      menuSelect = MENU_CLOCK;
      lcd.clear();
    } else if (results.value == IR_SET_MODE) {
      menu = MENU_SET_TIME;
      lcd.clear();
    }
    break;
  case MENU_DATE:
    if (results.value == IR_ENTER_EQ) {
      menu = MENU_MAIN;
      menuSelect = MENU_DATE;
      lcd.clear();
    } else if (results.value == IR_SET_MODE) {
      menu = MENU_SET_DATE;
      lcd.clear();
    }
    break;
  case MENU_SET_TIME...MENU_SET_DATE:
    static int i = 0;
    digit = keyToInt(results.value);
    if (results.value == IR_AGAIN_REW) {
      lcd.clear();
      i = 0;
      for (int j = 0; j < 6; j++)
        numbers[j] = 0;
    }
    else if (results.value == IR_ENTER_EQ) {
      if (menu == MENU_SET_TIME) {
        myTime.hour = numbers[0] * 10 + numbers[1];
        myTime.minutes = numbers[2] * 10 + numbers[3];
        sec = numbers[4] * 10 + numbers[5];
        myTime.seconds = sec;
```

```
          menu = MENU_CLOCK;
        } else {
          myDate.year = numbers[0] * 10 + numbers[1];
          myDate.month = numbers[2] * 10 + numbers[3];
          myDate.day = numbers[4] * 10 + numbers[5];

          if (myDate.month > 12)  myDate.month=12;
          if (myDate.day > 31)  myDate.month=31;

          menu = MENU_DATE;
        }
        i = 0;
        for (int j = 0; j < 6; j++)
          numbers[j] = 0;
        lcd.clear();
      }
      else if (digit >= 0 && digit <= 9) {
        numbers[i] = digit;
        i++;
      }
      break;
    case MENU_ALARM:
      if (results.value == IR_ON_OFF) {
        menu = MENU_MAIN;
        digitalWrite(BUZZ_PIN, LOW);
        lcd.clear();
        setAlarm = false;
      }
      break;
    case MENU_GIVE_PASSWORD...MENU_REQUIRE_ENG_PASS:
      //MENU_GIVE_PASSWORD =alarm alert in entry/exit uses alarmZone ;
MENU_REQUIRE_PASSWORD =mode on entry zone ; MENU_REQUIRE_ENG_PASS
=change eng pass/ configure a zone
      static int j = 0;
      digit = keyToInt(results.value);
      if (results.value == IR_ENTER_EQ) {
        j = 0;
        boolean same = true;
        for (int k = 0; k < 4; k++) {
          Serial.print(menu); Serial.print(" "); Serial.print(numbers[k]); Serial.println(eng_pass[k]);
          if (menu == MENU_REQUIRE_ENG_PASS) {
            if (numbers[k] != eng_pass[k])
              same = false;
          } else if (menu == MENU_GIVE_PASSWORD) {
            if (numbers[k] != zones[alarmZone].param->password[k]) same = false;
          } else if (menu == MENU_REQUIRE_PASSWORD) {
            if (numbers[k] != zones[index].param->password[k]) same = false;
          }
        }
        if (same == false) {
          Serial.print("WRONG PASSWORD!");
          if (menu == MENU_GIVE_PASSWORD) {
            countDown = false;
            setAlarm = true;
            saveEventToEEPROM(alarmZone);
            menu = MENU_ALARM;
            lcd.clear();
          } else {
            lcd.clear();
            if (menu == MENU_REQUIRE_PASSWORD)
              lcd.print("Wrong pass, lads!");
            else
              lcd.print("You NOT engineer");
            delay(1000);
            lcd.clear();
            if (menu == MENU_REQUIRE_ENG_PASS) menu = MENU_MAIN;
            else menu = MENU_ZONES;  // two cases: MENU_REQUIRE_PASS(plus) &&
MENU_GIVE_PASS(alert on entry/exit zone)
          }
        } else {
          Serial.print("GOOD PASSWORD!");
          if (menu == MENU_GIVE_PASSWORD) {
            lcd.clear();
            lcd.print("Hello Master!");
            delay(1000);
            countDown = false;
            menu = MENU_MAIN;
            lcd.clear();
            setAlarm = false;
            digitalWrite(BUZZ_PIN, LOW);
          }
          else if (menu == MENU_REQUIRE_PASSWORD) {
            menu = MENU_SET_PASS;
            lcd.clear();
          }
          else if (menu == MENU_REQUIRE_ENG_PASS) {
            if (changeEngPass) menu = MENU_SET_ENG_PASS;
            else menu = MENU_SELECT_TYPE;
            lcd.clear();
          }
        }
      }
      if (digit >= 0 && digit <= 9) {
        numbers[j] = digit;
        lcd.setCursor(j, 1);
        lcd.print("*");
        Serial.println(digit);
        j++;
      }
      break;
    case MENU_SET_PASS:
      digit = keyToInt(results.value);
      if (results.value == IR_ENTER_EQ) {
        for (int k = 0; k < 4; k++) {
          zones[index].param->password[k] = numbers[k];
        }
        menu = MENU_SET_LIMIT;
        lcd.clear();
        i = 0;
      } else if (digit >= 0 && digit <= 9) {
        lcd.setCursor(i, 1);
        lcd.print(digit);
        numbers[i] = digit;
        i++;
      }
      break;
    case MENU_SET_ENG_PASS:
      digit = keyToInt(results.value);
      if (results.value == IR_ENTER_EQ) {
        for (int k = 0; k < 4; k++) {
          eng_pass[k] = numbers[k];
        }
        menu = MENU_MAIN;
        changeEngPass = false;
        saveEngToEEPROM();
        lcd.clear();
        i = 0;
      } else if (digit >= 0 && digit <= 9) {
        lcd.setCursor(i, 1);
        lcd.print(digit);
        numbers[i] = digit;
        i++;
      }
      break;
    case MENU_SET_LIMIT:
      digit = keyToInt(results.value);
      if (results.value == IR_ENTER_EQ) {
        zones[index].param->timeLimit = numbers[0] * 10 + numbers[1];
        lcd.clear();
        lcd.print("Changes saved");
        delay(1000);
        menu = MENU_ZONES;
        lcd.clear();
        i = 0;
        settingZone = -1;
        saveZonesToEEPROM();
      } else if (digit >= 0 && digit <= 9) {
        lcd.setCursor(i, 1);
        lcd.print(digit);
        numbers[i] = digit;
        i++;
      }
      break;
    case MENU_SET_THRESHOLD:
      digit = keyToInt(results.value);
      if (results.value == IR_ENTER_EQ) {
        int value = 0;
        for (int k = 0; k < i; k++) {
          value = value * 10 + numbers[k];
        }
        zones[index].value = value;
        menu = MENU_ZONES;
        settingZone = -1;
        lcd.clear();
        i = 0;
        saveZonesToEEPROM();
      } else if (digit >= 0 && digit <= 9) {
        lcd.setCursor(i, 1);
        lcd.print(digit);
        numbers[i] = digit;
        i++;
      }
```

```
        break;
      case MENU_SET_DIGITAL:
        digit = keyToInt(results.value);
        if (digit == 0 || digit == 1) {
          zones[index].value = digit;
          menu = MENU_ZONES;
          settingZone = -1;
          lcd.clear();
          i = 0;
          saveZonesToEEPROM();
        }
       break;
      default: break;
    }
    irrecv.resume();
  }
}

// interrupt service routine for counting seconds
ISR(TIMER1_COMPA_vect) {
  sec++;
  if (countDown) {
    counter--;
  }
}

boolean debounce(int pin) {
  if (digitalRead(pin)) {
    delay(MAX_BOUNCE_TIME);
    return digitalRead(pin);
  }
  return false;
}

// function that checks if the alarm activate conditions are met on one of the zones
// activate conditions depend on zones types
void checkZones() {
  for (int i = 0; i < 4; i++) {
    byte stateNow = 0;
    if (zones[i].pin == ANALOG_PIN) {
      int readPot = analogRead(zones[i].pin);
      if (readPot > 512) stateNow = HIGH;
      else stateNow = LOW;
    } else stateNow = debounce(zones[i].pin);

    if (i != settingZone) {
      if (zones[i].type == 'E') {
        if (!countDown && (stateNow == zones[i].value)) {
          Serial.print("zone E: countDown="); Serial.print(countDown); Serial.print("pin=");
Serial.println(debounce(zones[i].pin));
          counter = zones[i].param->timeLimit;
          countDown = true;
          menu = MENU_GIVE_PASSWORD;
          alarmZone = i;
          lcd.clear();
        }
        if (countDown && counter == 0) {
          countDown = false;
          setAlarm = true;
          //-----create event
          saveEventToEEPROM(i);
          menu = MENU_ALARM;
          lcd.clear();
        }
        if (countDown) {
          digitalWrite(BUZZ_PIN, HIGH);
          delay(200);
          digitalWrite(BUZZ_PIN, LOW);
          delay(200);
        }
      }
      else if (zones[i].type == 'C') {
        if (stateNow == zones[i].value) {
          if (!setAlarm) {
            menu = MENU_ALARM;
            setAlarm = true;
            saveEventToEEPROM(i);
            lcd.clear();
            alarmZone = i;
          }
        }
      }
      else if (zones[i].type == 'A') {
        int val = analogRead(zones[i].pin);
        if (val >= zones[i].value) {
          if (!setAlarm) {
            menu = MENU_ALARM;
            setAlarm = true;
            saveEventToEEPROM(i);
            lcd.clear();
            alarmZone = i;
          }
        }
      }
      else if (zones[i].type == 'D') {
        if (stateNow == zones[i].value) {
          Serial.print("sunt D si cred debounce(zones[i].pin) == zones[i].value D=");
          Serial.println(zones[i].value);
          if (!setAlarm) {
            menu = MENU_ALARM;
            setAlarm = true;
            saveEventToEEPROM(i);
            lcd.clear();
            alarmZone = i;
          }
        }
      }
    }
  }
}

// very clean loop
void loop() {
  printOnLCD();
  performTask();
  checkZones();
}
```