CS3514 C Programming for Microcontrollers Report

Colm Cahalane 28/11/2015

Contents

1 Introduction

The goal of this project was to construct a burglar alarm with some basic functionality using C and the Arduino framework for microcontrollers.

Some basic restrictions and requirements that were to be placed on the project were:

- The burglar alarm should be standalone and shouldn't need a computer to run.
- The burglar alarm should have a settable clock with date and time.
- The burglar alarm must support multiple zones:
  - A digital zone, which can be active on high or low.
  - An analog zone, with a variable threshold that can be set by user.
  - An entry-exit zone that allows a user to enter a PIN number and not set off the alarm - or allow a user to arm the system and exit during a grace period.
  - A continuous monitoring zone that triggers on the high $\rightarrow$ low transition.
- The burglar alarm must have a number of persistent settings stored in memory.
- The burglar alarm should have a basic PIN login system.
  - There should be a separate administrator level for making settings.
- The burglar alarm should have the ability to store logs in EEPROM. A registered administrator should be able to view these logs on screen.

We were given a basic Arduino Duemilanove kit with which to construct the project which included an IR Remote and receiver, 2x16 character LCD Display, buttons to use as triggers, a potentiometer for analogue input, some LED lights and buzzers, a breadboard and a collection of wires.

2 Requirements\Analysis\Design

2.1 What are we building?

As our group interpreted the project specifications, hardware-wise, we're looking to build a device that relays sensors for the four zones and a receiver for the IR remote to the Arduino.

The digital zones and entry-exit zones here will be activated by buttons, as that's the most simple implementation of such. Ideally, we'd like to have some sort of IR sensor for the entry-exit zone, but this is complicated by our use of IR for the remote.

The Continuous Monitoring Zone behaves like a sort of anti-tamper device; it's attached directly to a source of 5V. If the connection is broken and a high $\rightarrow$ low transition occurs on this signal, an interrupt should be triggered in software.

In return, output is to be handled by a single-pin output for a buzzer or LED and a more detailed output through the LCD display.

Ideally, for digital inputs, it would be best that we could implement these as triggering interrupts.

The goal of the project is that a user can interface with the alarm's options and settings clearly using the LCD and remote, so inputs and outputs related to operating the device should be kept very simple.

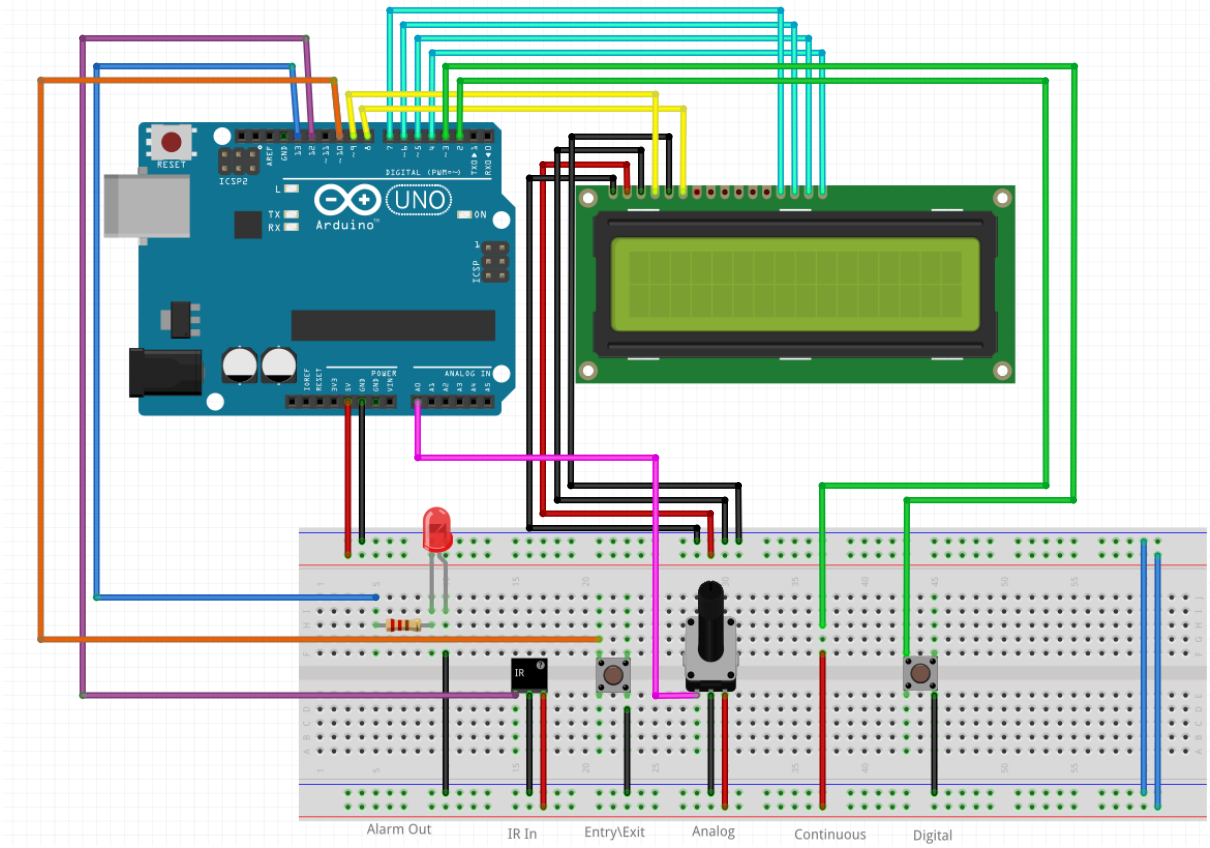
Most of the alarm's other functionality, such as timekeeping, permanent storage etc. can be implemented purely in software.

In the below table and figure we'll describe the hardware implementation and design we settled on as a group.

Table 1: A list of the Arduino's pins in use, and how we're using them.

Pin #	Mode	Function
2	Input (Interrupt)	Continuous Monitoring
3	Input (Interrupt)	Digital Zone
4	Output (LiquidCrystal)	LCD
5	Output (LiquidCrystal)	LCD
6	Output (LiquidCrystal)	LCD
7	Output (LiquidCrystal)	LCD
8	Output (LiquidCrystal)	LCD
9	Output (LiquidCrystal)	LCD
10	Input	Entry/Exit Zone
12	Input (IRRemote)	IR Sensor
13	Output	Alarm Output
A0	Analog Input	Analog Zone

Figure 1: A diagram, generated using Fritzing, that shows the hardware design we decided on.



2.2 What are the features we're building?

We'll need to build software that monitors and responds to the aforementioned zones.

As indicated in the introduction, we're looking to build a number of useful features on top of the hardware that we've designed. We'll need to be able to have user login and verification, two different passwords (one for admin and one for "normal" user).

We'll need an implementation of a clock and allow the user to set the time. Timekeeping is important for our implementation of the entry/exit zone as it has certain specific hours of activity each day as well as

We'll need a way to implement user settings and make sure that defaults are working in a way that makes sense.

The settings that we'll need to implement are:

- User Password
- Admin Password
- Threshold for Analog Zone (0 - 1024)
- Active Hours for Entry\Exit zone
- Digital Zone - active high or active low

We'll need to develop a simple user privileges system and login mode to ensure that these settings are protected. Also, we shouldn't allow the alarm to be unset or disarmed until a valid login has occurred.

If any of the zone conditions are triggered, an alarm should start sounding and only be disabled if a logged-in user silences it. When an alarm sounds, we want there to be a record of this stored in permanent storage.

We'll also want these logs to be viewable by a logged-in user.

2.3 What are the limitations we'll face?

2.3.1 16x2 Screen

The 16x2 screen, allowing for 32 characters on screen at any time and of which one row is generally used for the clock, means that the user experience must suffer for functionality; we weren't able to provide for a full user menu and navigation experience. In the software design we've built commands must be learnt off or read from a manual. This is less than ideal.

2.3.2 Timekeeping

The Arduino doesn't have much functionality for timekeeping built in that would work in a way that suits us. While we can keep count of the number of milliseconds since the Arduino was started using `millis()`, this doesn't work within interrupts and is not an adequate solution to implement a settable clock.

We'd have to implement a clock ourselves through a carefully written interrupt, or we could use the `Time.h` library.

2.3.3 Limited number of interruptable pins

Though it would suit us to have all digital inputs handled as interrupts, the Arduino Duemilanove board only allows pins 2 and 3 to trigger interrupts. As such, we can't attach an interrupt to each of the digital, entry-exit and continuous monitoring zones. We must choose these carefully.

2.3.4 EEPROM

Permanent storage on an Arduino board is here provided by a library which allows for writing from and reading to EEPROM storage based on addresses. It is therefore important to us to divide the space available into EEPROM into blocks and designate specific addresses to specific functions. We must also make sure that at the same time that we have a way of checking that the values in EEPROM are set, and if not, imposing some defaults so that invalid information is not placed into the program.

2.4 Dividing program into functional blocks.

Based on this description of the software, we need to start to break down the program into functional blocks.

- A setup function that runs when the Arduino is first turned on.
- A function which runs to check if the settings stored in EEPROM are valid and if not, set defaults.
- Interrupt service routines for when the digital or continuous zones are breached.
- The program's main execution loop.
  - A section of this loop to poll the entry-exit zone and if it has been tripped, prompts the user to log in during a countdown - otherwise triggers alarm.
  - A section of this loop to poll the analog zone and trip it if necessary
  - A function allowing a user to log in/out
  - A function allowing a logged in admin to view logs
  - A function that displays a log at an address in EEPROM
  - A function allowing a logged in admin to arm/disarm the alarm
  - A function allowing a logged in user to deactivate an active alarm.
  - A function allowing a logged in admin to set the time
  - A function allowing a logged in admin to change options stored in EEPROM
  - A function which prints the current time.
- A set of functions available to write logs to EEPROM if an alarm is triggered.
- A set of functions to allow settings in EEPROM to be read and changed.
- A set of functions to interpret user input from the IR remote and return values.

3 Implementation

3.1 Control-Flow Diagram

Figure 2: A general control-flow diagram layout for the program.

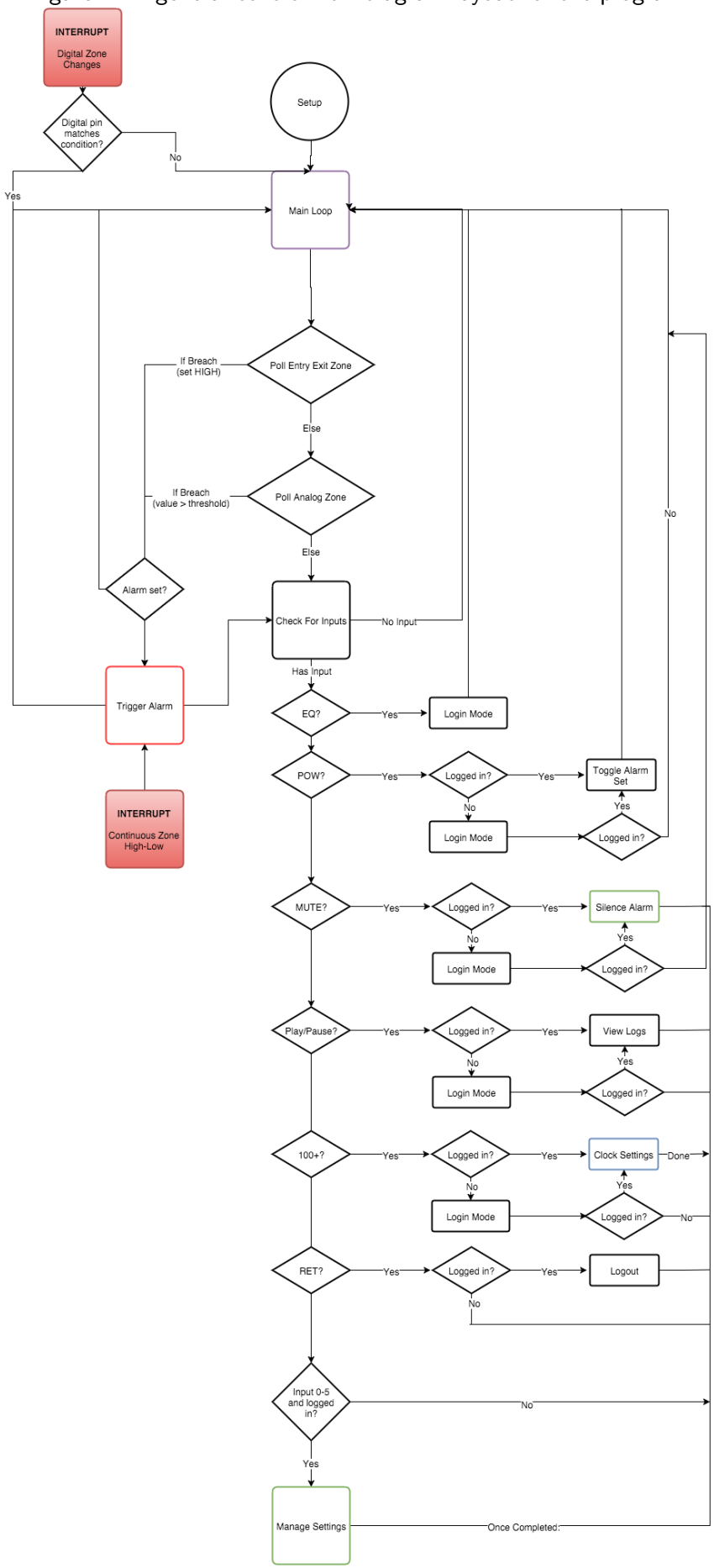


Figure 3: A control-flow diagram the program's settings stored in EEPROM.

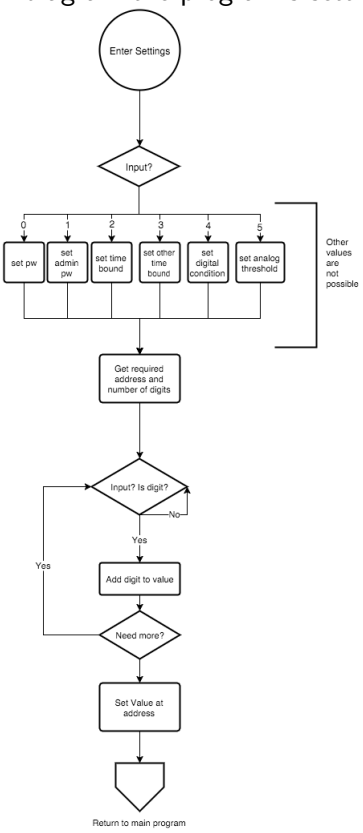
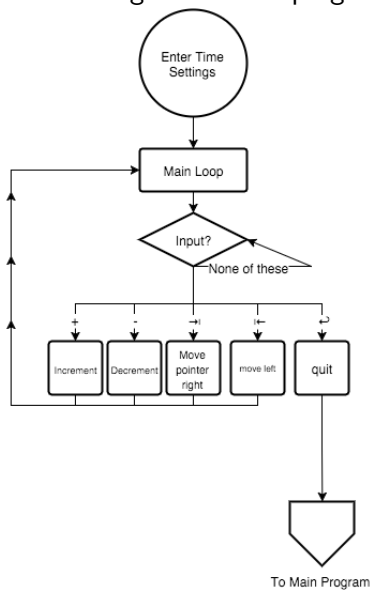


Figure 4: A control-flow diagram for the program's time settings.



3.2 Coding Decisions

3.2.1 Timekeeping

As outlined above in the limitations section, we had to make a decision on using a timekeeping library or write our own with interrupts. Our decision was to use `Time.h` but the limitations of the Arduino and the small 16x2 screen meant that a user can only set the time by using the + and - buttons to increment and decrement the day/month/hour/minute etc. rather than enter the dates

Table 2: Locations of values stored in EEPROM and their types

Location	Information	Type
0-1	Password	unsigned int
2-3	Administrator Password	unsigned int
4-5	Number Of Breaches	unsigned short
6-8	'S' 'E' 'T' values	char[]
15	lower-bound hour for entry/exit zone	unsigned short
17	upper-bound hour for entry/exit zone	unsigned short
20	Trip condition for digital zone (high/low like a boolean)	unsigned short
30-32	Threshold value for analog zone	unsigned int
100-511	Logging	6 byte segments as follows:
L0-3	Time of Breach	unsigned long
L4-5	Zone ID	unsigned short

with numeric keys. Still, it makes up for this by giving us a universal date format that we can use in creating/storing logs, and the TimeElements structure for printing/setting time with very readable and maintainable code.

If we were to use the interrupt-based approach, this code would have been used to create a clock-based interrupt which would trigger once per second:

```
1   TCCR1A = 0;
2   TCCR1B = 0;
3   OCR1A = 15625;
4   TCCR1B |= (1 << WGM12);
5   TCCR1B |= (1 << CS10);
6   TCCR1B |= (1 << CS12);
7   TIMSK1 |= (1 << OCIE1A);
8   sei();
```

3.2.2 EEPROM Mapping and defaults

We had to map out specific areas of EEPROM and the types of values that could be stored there but it took us a while to come up with a good solution for the first-boot and ensuring that the values in EEPROM were legitimate. We decided on mapping a sequence of addresses that would contain the values 'S', 'E', 'T', allowing us to have a pretty reliable indicator for if the device had been set up with the burglar alarm before. If the SET values aren't present, we'd run a function which would flash defaults.

3.2.3 User Interaction

As shown in the Control-Flow Diagram, the user can log in to the device, and then enter a number of settings modes from within the main loop. If the user is not logged in and attempts to enter a settings mode, they will be first asked to log in.

Handling most of this user interaction from within the main loop and not constructing an additional menu simply means that we can continue to monitor the analog and entry-exit zones actively as the user interacts with the device.

3.2.4 Interrupts

We use both interrupt pins to the board to ensure that active monitoring takes place at all times on certain pins. The continuous zone is set to interrupt on the high $\rightarrow$ low transition and instantly triggers the alarm. The digital zone is set to interrupt on any change and if the alarm is set, is not currently active and the digital zone switches to the condition specified by the user, the alarm is triggered.

We can not set an interrupt-based system to monitor the analog pin, and the entry-exit zone works better as a non-interrupt due to its tie-ins with the user account system and user interaction functions.

3.3 Interesting pieces of code

Here I'm including some particular segments of the codebase for the project that I find rather interesting or valuable in achieving elements of the project specification. Of course, the complete codebase for the project is available in the appendixes.

3.3.1 Getting a digit value from the IR sensor

```
1  int getDigitFromIR(){
2    while( 1 ){ // Loop until a button press is received
3      irrecv.resume();
4      while( !irrecv.decode(&results) ) { /* Wait for input! */ }
5      switch(results.value)
6      {
7        case 0xFF6897: return 0;
8        case 0xFF30CF: return 1;
9        case 0xFF18E7: return 2;
10       case 0xFF7A85: return 3;
11       case 0xFF10EF: return 4;
12       case 0xFF38C7: return 5;
13       case 0xFF5AA5: return 6;
14       case 0xFF42BD: return 7;
15       case 0xFF4AB5: return 8;
```

```
      case 0xFF52AD: return 9;
      default:       break; // Other button press or undefined; reloop
    }
  }
}
```

3.3.2 Checking if the EEPROM is set up right and setting defaults

```
int settingsSet( ){
  char first_time[3];
  EEPROM.get( FIRST_TIME_SET, first_time );

  // Check if the values found match our chosen string.
  return !(first_time == "SET");
}

void defaults(){
  unsigned int password = 1234,
               admin_password = 5678,
               analog_threshold = 100;

  unsigned short number_breaches = 0,
                 lower_bound_hour = 20,
                 upper_bound_hour = 22,
                 digital_trip_condition = 1;

  char first_time[] = "SET";

  // Store our default values in memory.
  EEPROM.put( FIRST_TIME_SET, first_time );
  EEPROM.put( PASSWORD, password );
  EEPROM.put( ADMIN_PASSWORD, admin_password );
  EEPROM.put( ANALOG_THRESHOLD, analog_threshold );

  EEPROM.put( NUMBER_OF_BREACHES, number_breaches );
  EEPROM.put( LOWER_TIME_BOUND, lower_bound_hour );
  EEPROM.put( UPPER_TIME_BOUND, upper_bound_hour );
  EEPROM.put( DIGITAL_CONDITION, digital_trip_condition );
}
```

3.3.3 Printing UNIX time values

```
void printWithLeadingZero(int val){
  if(val < 10){
```

```
    lcd.print('0');
  }
  lcd.print(val);
}

void convertUnixToReadable( unsigned long int input_time ){
  TimeElements full_time;
  breakTime( input_time, full_time );

  printWithLeadingZero(full_time.Hour);
  lcd.print( ":" );
  printWithLeadingZero(full_time.Minute);
  lcd.print( " " );
  printWithLeadingZero( full_time.Day );
  lcd.print("/");
  printWithLeadingZero( full_time.Month );
  lcd.print("/");
  lcd.print( full_time.Year + 1970 );
}
```

3.3.4 User interface for modifying time

```
void changeTime(){
    TimeElements t;
    time_t newTime;
    breakTime(now(), t);

    int settingsMode = 0;
    short exitLoop = 0;

    while( !exitLoop ){
      newTime = makeTime(t);

      lcd.clear();
      lcd.setCursor(0,0);

      convertUnixToReadable(newTime);

      lcd.setCursor(0,1);

      if(settingsMode == HOUR){
        lcd.print("Setting HOUR");
      } else if(settingsMode == MINUTE){
        lcd.print("Setting MINUTE");
      } else if(settingsMode == DAY){
```

```
24          lcd.print("Setting DAY");
25        } else if(settingsMode == MONTH){
26          lcd.print("Setting MONTH");
27        } else if(settingsMode == YEAR){
28          lcd.print("Setting YEAR");
29        }
30
31        irrecv.resume();
32        while( !irrecv.decode(&results) ) { /* Wait for input! */ }
33        switch(results.value){
34                                // +4 should be -1, but here we avoid
                                      nevative modulo
35          case 0xFF22DD: /* PREV */ settingsMode = (settingsMode+4)%5;
                    break;
36          case 0xFF02FD: /* NEXT */ settingsMode = (settingsMode+1)%5;
                    break;
37          case 0xFFE01F: /*  -  */
38              if(settingsMode == HOUR){
39                t.Hour--;
40              } else if(settingsMode == MINUTE){
41                t.Minute--;
42              } else if(settingsMode == DAY){
43                t.Day--;
44              } else if(settingsMode == MONTH){
45                t.Month--;
46              } else if(settingsMode == YEAR){
47                t.Year--;
48              }
49             break;
50          case 0xFFA857: /*  +  */
51              if(settingsMode == HOUR){
52                t.Hour++;
53              } else if(settingsMode == MINUTE){
54                t.Minute++;
55              } else if(settingsMode == DAY){
56                t.Day++;
57              } else if(settingsMode == MONTH){
58                t.Month++;
59              } else if(settingsMode == YEAR){
60                t.Year++;
61              }
62             break;
63          case 0xFFB04F: /* RET */ exitLoop = 1; lcd.clear(); break;
64        }
65      }
66
67      setTime(newTime);
```

```
}
```

3.3.5 Logging functions

```
/**
 * Append log to memory
 * @param time_of_breach Unix timestamp of current time
 * @param zone           Zone number that was breached
 */
void appendLog( unsigned long int time_of_breach, unsigned short zone ) {

  unsigned short number_of_breaches;
  EEPROM.get( NUMBER_OF_BREACHES, number_of_breaches );

  // Increase the number of breaches
  number_of_breaches++;
  EEPROM.put( NUMBER_OF_BREACHES, number_of_breaches );

  int memory_address = LOG_MEMORY_START + (( LOG_MEMORY_START + (
      LOG_LENGTH * number_of_breaches) ) % 500);

  // Write our log to EEPROM
  EEPROM.put( memory_address, time_of_breach );
  memory_address += sizeof(time_of_breach);
  EEPROM.put( memory_address, (short) zone );
}

/**
 * Allows user to navigate the stored log
 * @param current_log The current log to be printed
 */
void printLog( short current_log ){
  unsigned short number_of_breaches;
  EEPROM.get( NUMBER_OF_BREACHES, number_of_breaches );

  if( current_log <= number_of_breaches && current_log != 0){
    // If we have a log to show
    int memory_address = LOG_MEMORY_START + (( LOG_MEMORY_START + (
        LOG_LENGTH * current_log) ) % 500);

    unsigned long int time_of_breach;
    unsigned short zone;

    // Get log info
```

```
39      EEPROM.get( memory_address, time_of_breach );
40      memory_address += sizeof(time_of_breach);
41      EEPROM.get( memory_address, zone );
42
43      lcd.clear();
44      switch(zone){
45        case DIGITAL_ZONE:
46            lcd.print( "DIGITAL ZONE");
47          break;
48        case ANALOG_ZONE:
49            lcd.print("ANALOG ZONE");
50          break;
51        case CONTINUOUS_ZONE:
52            lcd.print( "CONTINUOUS ZONE");
53          break;
54        case ENTRY_EXIT_ZONE:
55            lcd.print("ENTRY/EXIT ZONE");
56          break;
57        default:
58            lcd.print( "UNKNOWN ZONE" );
59          break;
60      }
61      lcd.setCursor(0,1);
62      convertUnixToReadable( time_of_breach );
63
64      irrecv.resume();
65      while( !irrecv.decode(&results) ) { /* Wait for input! */ }
66      switch(results.value)
67      {
68        case 0xFF22DD: printLog( current_log - 1 );    break;
69        case 0xFF02FD: printLog( current_log + 1 );    break;
70        case 0xFFB04F: lcd.clear(); /* If return, just let it go */ break
              ;
71        default: printLog(current_log); // Other button press or
              undefined
72      }
73      irrecv.resume();
74    } else{
75      lcd.clear();
76      lcd.print("NO LOGS");
77      delay( 1000 );
78
79      if( current_log > 0 )
80        // If current log isn't 0, send them back a log
81        printLog( current_log - 1 );
82    }
83  }
```

4 Evaluation

4.1 What went right.

Generally speaking I consider this project to be a success. All of the most important features are implemented, the device is secure, and the code and hardware layout is generally elegant. I'm reasonably proud of how we got around some of the Arduino's specific nuances in terms of timekeeping and persistent storage in EEPROM.

Our user experience does have room for improvement (it's not immediately intuitive) but it is quite fast and responsive meaning that once a user learns their way around the system, it's fairly pleasant to do so.

Viewing logs, changing the date/time, logging in, arming the system and all of the sensors work as intended with few consequences.

We were able to demonstrate that the project implemented all of its intended functionality during the lab demonstrations.

4.2 Room to improve.

Given time or given the opportunity to rise above some of the issues with Arduino, some improvements could be made.

The user experience is a bit complicated. Although some of the commands correlate nicely to buttons on the remote control (ON/OFF arms or disarms the alarm, MUTE silences an ongoing alarm signal, etc.) many do not (for instance command 0 is to set the user password). We should've spent time building a menu based system for options and settings.

One notable issue is that if the analog threshold is exceeded as the system is arming, the alarm will start going off immediately after the countdown has reached zero. We should warn the user about active zones before arming the system.

One feature I wished to implement but failed to build was the ability to synchronise the clock to another device using serial communication.

It would also make sense to implement setting the clock using the number pad rather than the + / - buttons alone.

Given other hardware and further time, there would be some interesting additional functionalities we could build. Using a GSM or Ethernet shield, we could make this a connected device that would interface with other devices. It'd be interesting to implement some web server functionality and allow for the viewing of logs or the administration of the device over the network.

5 Appendix 1: Complete Code

Contains previously mentioned code and comments.

```
#include <EEPROM.h>
#include <Time.h>
#include <IRremote.h>
#include <LiquidCrystal.h>

/**
 * @author Colm Cahalane <113326986>
 * @author Evan Smith <113300626>
 *
 * Build a burglar alarm --- The task for the project
 * was to build a device, using an arduino, that could
 * monitor several different types of zones, log alarm
 * trips and allow for user and admin interaction via
 * an LCD. When an alarm condition is met, a buzzer
 * or LED will go off indicating as such.
 *
 * Pins
 * ====
 *
 * Digital
 * -------
 *
 * 2 (interrupt) : Continuous Zone
 * 3 (interrupt) : Digital Zone
 * 4 : LCD Screen
 * 5 : LCD Screen
 * 6 : LCD Screen
 * 7 : LCD Screen
 * 8 : LCD Screen
 * 9 : LCD Screen
 * 10: Entry/Exit Zone
 * 12: IR Sensor
 * 13: ALARM
 *
 * Analog
 * ------
 * 0 : Analog Zone 1
 *
 * IR Remote Layout
 * ----------------
 *
 * EQ : Enter password (4-Digit Pin)
```

```
 * Return : Return to standard menu
 * Play/Pause : Navigate Log
 *   Next     : Next log
 *   Prev     : Previous log
 * Mute       : Turn off alarm
 * Power      : Set/unset the alarm
 * 0  : Set user password
 * 1  : Set admin password
 * 2  : Set lower bound hour for entry/exit zone
 * 3  : Set upper bound hour for entry/exit zone
 * 4  : Set digital activate condition
 * 5  : Set analog threshold
 *
 * EEPROM Mapping
 * --------------
 * 0 - 1 : password (unsigned int)
 * 2 - 3 : admin password (unsigned int)
 * 4 - 5 : number of breaches (unsigned short)
 * 6 - 8 : first-time settings ("set" or anything else)
 *
 *  Entry/Exit (Zone 0)
 *    15     : lower-bound hour (unsigned short)
 *    16     : upper-bound hour (unsigned short)
 *
 *  Digital (Zone 1)
 *    20     : trip condition (unsigned short)
 *
 *  Analog  (Zone 2)
 *    30 - 32 : threshold (unsigned int)
 *
 * 100 - 511 : Logging
 *   Bit Mapping (6 bytes each):
 *   0 - 3 : time (unsigned long int)
 *   4 - 5 : zone (unsigned short)
 *
 *
 */

#define PASSWORD              0     // 4 digit pin
#define ADMIN_PASSWORD        2     // 4 digit pin
#define NUMBER_OF_BREACHES    4
#define FIRST_TIME_SET        6

// ~~~~~ ENTRY / EXIT ZONE ~~~~~
#define ENTRY_EXIT_ZONE       0
#define ENTRY_EXIT_PIN        10
#define LOWER_TIME_BOUND      15     // Hour (2 digits max)
```

```
90  #define UPPER_TIME_BOUND      16     // Hour (2 digits max)
91
92  // ~~~~~~~ DIGITAL ZONE ~~~~~~~~
93  #define DIGITAL_ZONE          1
94  #define DIGITAL_CONDITION     20    // HIGH (1) or LOW (0)
95  #define DIGITAL_ZONE_PIN      3
96
97   // ~~~~~~~ ANALOG ZONE ~~~~~~~~
98  #define ANALOG_ZONE           2
99  #define ANALOG_THRESHOLD      30    // short between 0 - 255
100 #define ANALOG_ZONE_PIN       0
101
102
103  // ~~~ CONTINUOUS MON ZONE ~~~~
104 #define CONTINUOUS_ZONE       3
105 #define CONTINUOUS_ZONE_PIN   2
106
107  // ~~~~~ TIME SETTING MODE ~~~~
108 #define HOUR 0
109 #define MINUTE 1
110 #define DAY 2
111 #define MONTH 3
112 #define YEAR 4
113
114 // ~~~~~~~~~ LOGS ~~~~~~~~~~~~~~
115 #define LOG_MEMORY_START  100
116 #define LOG_LENGTH        6
117
118 // ~~~~~~~~~~ IR ~~~~~~~~~~~~~~~
119 #define IR_RECV_PIN   12
120 IRrecv irrecv(IR_RECV_PIN);
121 decode_results results;
122
123 #define ALARM_PIN     13
124
125 // initialize the library with the numbers of the interface pins
126 LiquidCrystal lcd(9, 8, 7, 6, 5, 4);
127
128 // Used to check if current user is an admin
129 unsigned short is_admin = 0;
130
131 // 0 is disabled ; 1 is enabled
132 unsigned short alarm_set = 0;
133
134 // 0 alarm is idle ; 1 alarm is ringing
135 volatile unsigned short alarm_active = 0;
136
```

```
137 // 0 not logged in ; 1 logged in
138 unsigned short is_user_logged_in = 0;
139
140 /**
141  * Prints the current time to the LCD
142  */
143 void printTime(){
144   lcd.setCursor(0, 0);
145   convertUnixToReadable(now());
146 }
147
148 /**
149  * Pads and prints an integer with 0s
150  * @param val Integer to pad
151  */
152 void printWithLeadingZero(int val){
153   if(val < 10){
154     lcd.print('0');
155   }
156   lcd.print(val);
157 }
158
159 /**
160  * Set the time
161  */
162 void changeTime(){
163     TimeElements t;
164     time_t newTime;
165     breakTime(now(), t);
166
167     int settingsMode = 0;
168     short exitLoop = 0;
169
170     while( !exitLoop ){
171       newTime = makeTime(t);
172
173       lcd.clear();
174       lcd.setCursor(0,0);
175
176       lcd.print( hour(newTime) );
177       lcd.print(':');
178       printWithLeadingZero( minute(newTime) );
179
180       lcd.print(' ');
181
182       printWithLeadingZero( day(newTime) );
183       lcd.print('/');
```

```
184         printWithLeadingZero( month(newTime) );
185         lcd.print('/');
186         lcd.print( year(newTime) );
187 
188         lcd.setCursor(0,1);
189 
190         if(settingsMode == HOUR){
191           lcd.print("Setting HOUR");
192         } else if(settingsMode == MINUTE){
193           lcd.print("Setting MINUTE");
194         } else if(settingsMode == DAY){
195           lcd.print("Setting DAY");
196         } else if(settingsMode == MONTH){
197           lcd.print("Setting MONTH");
198         } else if(settingsMode == YEAR){
199           lcd.print("Setting YEAR");
200         }
201 
202         irrecv.resume();
203         while( !irrecv.decode(&results) ) { /* Wait for input! */ }
204         switch(results.value){
205                                 // +4 should be -1, but here we avoid
                                        nevative modulo
206           case 0xFF22DD: /* PREV */ settingsMode = (settingsMode+4)%5;
                      break;
207           case 0xFF02FD: /* NEXT */ settingsMode = (settingsMode+1)%5;
                      break;
208           case 0xFFE01F: /*  -  */
209               if(settingsMode == HOUR){
210                 t.Hour--;
211               } else if(settingsMode == MINUTE){
212                 t.Minute--;
213               } else if(settingsMode == DAY){
214                 t.Day--;
215               } else if(settingsMode == MONTH){
216                 t.Month--;
217               } else if(settingsMode == YEAR){
218                 t.Year--;
219               }
220              break;
221           case 0xFFA857: /*  +  */
222               if(settingsMode == HOUR){
223                 t.Hour++;
224               } else if(settingsMode == MINUTE){
225                 t.Minute++;
226               } else if(settingsMode == DAY){
227                 t.Day++;
```

```
228             } else if(settingsMode == MONTH){
229               t.Month++;
230             } else if(settingsMode == YEAR){
231               t.Year++;
232             }
233            break;
234         case 0xFFB04F: /* RET */ exitLoop = 1; lcd.clear(); break;
235       }
236     }
237
238     setTime(newTime);
239 }
240
241 int getDigitFromIR(){
242   while( 1 ){
243     irrecv.resume();
244     while( !irrecv.decode(&results) ) { /* Wait for input! */ }
245     switch(results.value)
246     {
247       case 0xFF6897: return 0;
248       case 0xFF30CF: return 1;
249       case 0xFF18E7: return 2;
250       case 0xFF7A85: return 3;
251       case 0xFF10EF: return 4;
252       case 0xFF38C7: return 5;
253       case 0xFF5AA5: return 6;
254       case 0xFF42BD: return 7;
255       case 0xFF4AB5: return 8;
256       case 0xFF52AD: return 9;
257       default:       break; // Other button press or undefined; reloop
258     }
259   }
260 }
261
262 /**
263  * Attempt to log in a user, prompting
264  *   them for a user or admin password
265  * @return 1 if user logged in ; 0 otherwise
266  */
267 int loginMode() {
268   lcd.clear();
269   lcd.print( "Login Mode");
270   if( !is_user_logged_in || !is_admin ){
271     // if admin is already logged in, bypass login
272
273     lcd.clear();
274     if( is_user_logged_in && !is_admin ){
```

```
275       lcd.print( "Enter admin pin");
276     } else {
277       lcd.print( "4 Digit Pin");
278     }
279     delay(50);
280 
281     int pin_entered = 0;
282     unsigned int password, admin_password;
283     EEPROM.get( PASSWORD, password );
284     EEPROM.get( ADMIN_PASSWORD, admin_password );
285 
286     lcd.setCursor(0, 1);
287     for(int i = 0; i < 4; i++){
288       int received_value = getDigitFromIR();
289       pin_entered *= 10;
290       pin_entered += received_value;
291       lcd.print('*');
292 
293       // Minor delay to prevent debouncing "0"s
294       delay(50);
295       irrecv.resume();
296     }
297 
298 
299     lcd.setCursor(0,1);
300     if( !is_user_logged_in && pin_entered == password ){
301       is_user_logged_in = 1;
302       lcd.print( "LOGGED IN" );
303     } else if( pin_entered == admin_password ){
304       is_admin = 1;
305       is_user_logged_in = 1;
306       lcd.print( "LOGGED IN" );
307     } else{
308       lcd.print( "FAILED LOGIN" );
309     }
310     delay(1500);
311   } else{
312     lcd.print("You are admin");
313   }
314   lcd.clear();
315 
316   return is_user_logged_in;
317 }
318 
319 /**
320  * Append log to memory
321  * @param time_of_breach Unix timestamp of current time
```

```
322  * @param zone           Zone number that was breached
323  */
324 void appendLog( unsigned long int time_of_breach, unsigned short zone )
        {
325
326   unsigned short number_of_breaches;
327   EEPROM.get( NUMBER_OF_BREACHES, number_of_breaches );
328
329   // Increase the number of breaches
330   number_of_breaches++;
331   EEPROM.put( NUMBER_OF_BREACHES, number_of_breaches );
332
333   int memory_address = LOG_MEMORY_START + (( LOG_MEMORY_START + (
          LOG_LENGTH * number_of_breaches) ) % 500);
334
335   // Write our log to EEPROM
336   EEPROM.put( memory_address, time_of_breach );
337   memory_address += sizeof(time_of_breach);
338   EEPROM.put( memory_address, (short) zone );
339 }
340
341 /**
342  * Allows user to navigate the stored log
343  * @param current_log The current log to be printed
344  */
345 void printLog( short current_log ){
346   unsigned short number_of_breaches;
347   EEPROM.get( NUMBER_OF_BREACHES, number_of_breaches );
348
349   if( current_log <= number_of_breaches && current_log != 0){
350     // If we have a log to show
351     int memory_address = LOG_MEMORY_START + (( LOG_MEMORY_START + (
            LOG_LENGTH * current_log) ) % 500);
352
353     unsigned long int time_of_breach;
354     unsigned short zone;
355
356     // Get log info
357     EEPROM.get( memory_address, time_of_breach );
358     memory_address += sizeof(time_of_breach);
359     EEPROM.get( memory_address, zone );
360
361     lcd.clear();
362     switch(zone){
363       case DIGITAL_ZONE:
364           lcd.print( "DIGITAL ZONE");
365         break;
```

```
366       case ANALOG_ZONE:
367           lcd.print("ANALOG ZONE");
368         break;
369       case CONTINUOUS_ZONE:
370           lcd.print( "CONTINUOUS ZONE");
371         break;
372       case ENTRY_EXIT_ZONE:
373           lcd.print("ENTRY/EXIT ZONE");
374         break;
375       default:
376           lcd.print( "UNKNOWN ZONE" );
377         break;
378     }
379     lcd.setCursor(0,1);
380     convertUnixToReadable( time_of_breach );
381 
382     irrecv.resume();
383     while( !irrecv.decode(&results) ) { /* Wait for input! */ }
384     switch(results.value)
385     {
386       case 0xFF22DD: printLog( current_log - 1 );    break;
387       case 0xFF02FD: printLog( current_log + 1 );    break;
388       case 0xFFB04F: lcd.clear(); /* If return, just let it go */ break
              ;
389       default: printLog(current_log); // Other button press or
              undefined
390     }
391     irrecv.resume();
392   } else{
393     lcd.clear();
394     lcd.print("NO LOGS");
395     delay( 1000 );
396 
397     if( current_log > 0 )
398       // If current log isn't 0, send them back a log
399       printLog( current_log - 1 );
400   }
401 }
402 
403 /**
404  * Prints out unix time in a human-readable format
405  * @param  input_time Unix time input
406  */
407 void convertUnixToReadable( unsigned long int input_time ){
408   TimeElements full_time;
409   breakTime( input_time, full_time );
410 
```

```
411   printWithLeadingZero(full_time.Hour);
412   lcd.print( ":" );
413   printWithLeadingZero(full_time.Minute);
414   lcd.print( " " );
415   printWithLeadingZero( full_time.Day );
416   lcd.print("/");
417   printWithLeadingZero( full_time.Month );
418   lcd.print("/");
419   lcd.print( full_time.Year + 1970 );
420 }
421 
422 /**
423  * Exit admin mode
424  */
425 void exitAdmin( ){
426   is_admin = 0;
427   logout( );
428 }
429 
430 /**
431  * Remove logged in status
432  */
433 void logout( ){
434   is_user_logged_in = 0;
435   lcd.clear();
436   lcd.print("Logged out");
437   delay(700);
438 }
439 
440 /**
441  * Change whether the alarm can be active or not
442  */
443 void toggleAlarmSet( ){
444   if( !alarm_active ){
445     lcd.clear();
446     // We set the alarm at the end of the function to
447     // avoid interrupts triggering the alarm
448     unsigned short temp_alarm = !alarm_set;
449 
450     if( temp_alarm ){
451       logout( );
452       for (int i = 9; i < 10 && i >= 0; i--){
453         lcd.clear();
454         lcd.print(i);
455         delay(1000);
456       }
457       lcd.clear();
```

```
458         lcd.print( "ALARM SET" );
459         delay(800);
460       } else{
461         lcd.print( "ALARM UNSET" );
462         delay(800);
463       }
464
465       alarm_set = temp_alarm;
466     }
467 }
468
469 /**
470  * Change whether alarm is ringing or not
471  */
472 void toggleAlarm( ){
473   alarm_active = !alarm_active;
474
475   lcd.clear();
476   lcd.setCursor(0,1);
477   if( alarm_set ){
478     if( alarm_active ){
479       lcd.print( "ALARM ACTIVE     " );
480       digitalWrite( ALARM_PIN, HIGH );
481     } else {
482       lcd.print( "ALARM DEACTIVATED" );
483       digitalWrite( ALARM_PIN, LOW );
484       delay(1500);
485       lcd.clear();
486     }
487   }
488 }
489
490
491 /**
492  * Trip the digital zone if conditions are met
493  */
494 void digitalZoneTrip( ){
495   volatile unsigned short trip_condition;
496   EEPROM.get( DIGITAL_CONDITION, trip_condition );
497
498   if( trip_condition ){
499     if( digitalRead( DIGITAL_ZONE_PIN ) == HIGH && !alarm_active &&
          alarm_set ){
500       toggleAlarm( );
501       appendLog( now(), DIGITAL_ZONE );
502     }
503   } else {
```

```
504     if( digitalRead( DIGITAL_ZONE_PIN ) == LOW && !alarm_active &&
            alarm_set ){
505       toggleAlarm( );
506       appendLog( now(), DIGITAL_ZONE );
507     }
508   }
509 }
510
511 /**
512  * Trip the continuous zone
513  */
514 void contZoneTrip( ){
515   toggleAlarm( );
516   appendLog( now(), CONTINUOUS_ZONE );
517 }
518
519 /**
520  * Trip the analog zone if higher than threshold
521  */
522 void analogZoneTrip( ){
523   unsigned int threshold;
524   EEPROM.get( ANALOG_THRESHOLD, threshold );
525
526   if( analogRead( ANALOG_ZONE_PIN ) > threshold && !alarm_active &&
          alarm_set ){
527     toggleAlarm( );
528     appendLog( now(), ANALOG_ZONE );
529     delay(200);
530   }
531 }
532
533 /**
534  * Allows users to set permanent option
535  *   values (stored in EEPROM)
536  * @param option Option number from IR Remote
537  */
538 void setOption( short option ){
539   unsigned int address;
540   unsigned short digits;
541   switch( option ){
542     case 0:
543         lcd.print("PASSWORD");
544         address = PASSWORD;
545         digits = 4;
546       break;
547     case 1:
548         lcd.print("ADMIN_PASSWORD");
```

```
549           address = ADMIN_PASSWORD;
550           digits = 4;
551         break;
552       case 2:
553           lcd.print("LOWER TIME (Hour)");
554           address = LOWER_TIME_BOUND;
555           digits = 2;
556         break;
557       case 3:
558           lcd.print("UPPER TIME (Hour)");
559           address = UPPER_TIME_BOUND;
560           digits = 2;
561         break;
562       case 4:
563           lcd.print("DIGITAL COND 0/1");
564           address = DIGITAL_CONDITION;
565           digits = 1;
566         break;
567       case 5:
568           lcd.print("ANALOG THRESH 1-255");
569           address = ANALOG_THRESHOLD;
570           digits = 3;
571         break;
572       default:
573           return;
574         break;
575   }
576
577   if( digits > 2 ){
578     // If there are more than 2 digits, we'll need an int
579     unsigned int final_value = 0;
580     lcd.setCursor(0,1);
581     for(int i = 0; i < digits; i++){
582       int received_value = getDigitFromIR();
583         final_value *= 10;
584         final_value += received_value;
585         lcd.print(received_value);
586         // Minor delay to prevent debouncing "0"s
587         delay(50);
588         irrecv.resume();
589     }
590     EEPROM.put( address, final_value );
591   } else {
592     // If there are less than 2 digits, we can use a short
593     unsigned short final_value = 0;
594     lcd.setCursor(0,1);
595     for(int i = 0; i < digits; i++){
```

```
      int received_value = getDigitFromIR();
        final_value *= 10;
        final_value += received_value;
        lcd.print(received_value);
        // Minor delay to prevent debouncing "0"s
        delay(50);
        irrecv.resume();
    }
    EEPROM.put( address, final_value );
  }
}

/**
 * Places default values in memory if it's the
 * first time
 */
void defaults(){
  unsigned int password = 1234,
               admin_password = 5678,
               analog_threshold = 100;

  unsigned short number_breaches = 0,
                 lower_bound_hour = 20,
                 upper_bound_hour = 22,
                 digital_trip_condition = 1;

  char first_time[] = "SET";
  EEPROM.put( FIRST_TIME_SET, first_time );
  EEPROM.put( PASSWORD, password );
  EEPROM.put( ADMIN_PASSWORD, admin_password );
  EEPROM.put( ANALOG_THRESHOLD, analog_threshold );

  EEPROM.put( NUMBER_OF_BREACHES, number_breaches );
  EEPROM.put( LOWER_TIME_BOUND, lower_bound_hour );
  EEPROM.put( UPPER_TIME_BOUND, upper_bound_hour );
  EEPROM.put( DIGITAL_CONDITION, digital_trip_condition );
}

/**
 * Check if settings exist
 * @return 0 if not first time; 1 if first time
 */
int settingsSet( ){
  char first_time[3];

  EEPROM.get( FIRST_TIME_SET, first_time );

```

```
643   return !(first_time == "SET");
644 }
645
646 void setup() {
647   pinMode(ALARM_PIN, OUTPUT);
648   pinMode(CONTINUOUS_ZONE_PIN, INPUT);
649   pinMode(DIGITAL_ZONE_PIN, INPUT);
650   pinMode(ENTRY_EXIT_PIN, INPUT);
651
652   digitalWrite( ALARM_PIN, LOW );
653   digitalWrite( CONTINUOUS_ZONE_PIN, LOW );
654   digitalWrite( DIGITAL_ZONE_PIN, LOW );
655
656   attachInterrupt( digitalPinToInterrupt(DIGITAL_ZONE_PIN),
          digitalZoneTrip, CHANGE  );
657   attachInterrupt( digitalPinToInterrupt(CONTINUOUS_ZONE_PIN),
          contZoneTrip, FALLING );
658
659   int first_time = settingsSet( );
660
661   if( first_time ){
662     defaults();
663   }
664
665   alarm_set = 0;
666
667   setTime( 1447854337 );
668
669   irrecv.enableIRIn();
670
671   lcd.begin(16, 2);
672 }
673
674 void loop() {
675
676   /**
677    * Trip the Entry/Exit zone as necessary
678    */
679   if( digitalRead(ENTRY_EXIT_PIN) == HIGH ){
680     unsigned short lower, upper, currentHour;
681     EEPROM.get( LOWER_TIME_BOUND, lower );
682     EEPROM.get( UPPER_TIME_BOUND, upper );
683     currentHour = hour();
684
685     lcd.clear();
686     lcd.print("Plz login (EQ)");
687     long start_time = millis();
```

```
688     lcd.setCursor(0,1);
689     while( (millis() - start_time) < 20000 ){
690
691       irrecv.resume();
692       delay(300);
693       if(irrecv.decode(&results)){
694         if( results.value == 0xFF906F ){
695           if( !is_user_logged_in ){
696            loginMode( );
697           }
698           lcd.clear();
699           lcd.print("Crisis averted");
700           delay(800);
701           break;
702         }
703       }
704       lcd.clear();
705       lcd.print("Plz login (EQ)");
706       lcd.setCursor(0,1);
707       lcd.print( 20 - ((millis() - start_time) / 1000) );
708     }
709     irrecv.resume();
710
711     if( !alarm_active && alarm_set && !( currentHour <= lower &&
            currentHour >= upper) ){
712       // If current hour is not between the upper and lower bound
713       // then activate the alarm
714       toggleAlarm( );
715       appendLog( now(), ENTRY_EXIT_ZONE );
716       delay( 200 );
717     }
718   }
719
720   analogZoneTrip( );
721
722   if( irrecv.decode(&results) ) {
723     switch(results.value)
724     {
725       case 0xFF906F:
726           // EQ
727           loginMode();
728         break;
729       case 0xFFA25D:
730           // POW
731           if( !alarm_active && ( is_user_logged_in || loginMode() ) )
732             toggleAlarmSet( );
733         break;
```

```
734         case 0xFFE21D:
735             // MUTE
736             if( alarm_active && ( is_user_logged_in || loginMode() ) ){
737               toggleAlarm();
738             }
739           break;
740         case 0xFFC23D:
741             /* PLAY/PAUSE */
742             if( is_user_logged_in || loginMode() )
743               printLog( 1 );
744           break;
745         case 0xFF9867:
746             /* 100+ */
747             if(is_user_logged_in || loginMode() ){
748               changeTime();
749             }
750           break;
751 
752         case 0xFFB04F:
753             // RET
754             if( is_admin ){
755               exitAdmin( );
756             } else if( is_user_logged_in ){
757               logout();
758             }
759           break;
760 
761         /* SET OPTION VALUES */
762         case 0xFF6897:
763             if( is_admin || ( loginMode( ) && is_admin ) )
764               setOption( 0 );
765           break;
766         case 0xFF30CF:
767             if( is_admin || ( loginMode( ) && is_admin ) )
768               setOption( 1 );
769           break;
770         case 0xFF18E7:
771             if( is_admin || ( loginMode( ) && is_admin ) )
772               setOption( 2 );
773           break;
774         case 0xFFFFFF:
775             if( is_admin || ( loginMode( ) && is_admin ) )
776               setOption( 3 );
777           break;
778         case 0xFF10EF:
779             if( is_admin || ( loginMode( ) && is_admin ) )
780               setOption( 4 );
```

```
781           break;
782         case 0xFF38C7:
783             if( is_admin || ( loginMode( ) && is_admin ) )
784               setOption( 5 );
785           break;
786
787         default:
788       }
789
790       irrecv.resume(); // Receive the next value
791     } else {
792         printTime();
793     }
794
795 }
```

6 Appendix 2: User Manual

The RainbowTables Burglar Alarm is an arduino-based alarm system. As such, it will need to be powered by some external power source or by USB, but it will not need to be connected to a computer to function. As soon as it is powered on, it will set some default settings.

It by default monitors four zones, an entry-exit zone, a digital zone, an analog zone, and an anti-tampering device located within the alarm.

To log into the system, you can press the EQ button, and enter one of the default passwords; for basic user it's 1234 and for administrator it's 5678.

Once you're logged in as administrator, you can change these passwords. For user password, press 0 to change. For administrator, press 1.

There are some other settings you can change from this screen. Pressing 2 allows you to set the lower bound hour for the entry-exit zone, and pressing 3 allows you to set the upper bound hour. What this means is that you can set the time during which the entry-exit zone is monitoring actively and can trigger an alarm. These bounds are set by giving a number from 0 to 24 indicating the hours at which it is active. By default, it is active from 20:00 to 22:00.

You can also set if the digital zone should be triggered on a switch to low or high. This can be changed by pressing 4. By default, it activates on High. If you wish for it to activate on Low, set this value to 0. Set this value back to 1 if you would like to restore the default behaviour.

You can also set the sensitivity of the analog zone - a number from 0 to 1024 indicating the value at which the analog zone is triggered. 0 represents no voltage being carried, and 1024 represents a 5 volt charge. You can set this value by pressing 5. By default, this value is set to 100.

To set the clock on the device, press the 100+ button. Navigate between the Day/Month/Year-/Hour/Minute settings using the Previous and Next buttons and use the + and - buttons to adjust these values. Press the return button to save settings.

To arm the alarm system, press the power button. Note that you will be logged out and given a countdown for which to exit the area as monitoring will begin after this period.

If you trigger an alarm, you can press the mute button to silence it. You will be asked to log in if you are not already. If you trigger the Entry-Exit zone, you have a twenty second window to log in with a valid password (by pressing EQ) before the alarm becomes triggered.

When an alarm is triggered, a log is recorded of the incident. To view the logs, press the Play-Pause button. Using the Previous and Next buttons, you can navigate through a log of breaches.

To log out, simply press the return button.

If the device is powered off or plugged out, the clock will be reset but the settings and logs will remain.

7 Appendix 3: Project Plan

1. Review description and decide on possible hardware design
2. Create a basic hardware design (with Fritzing) which implements each sensor zone.
3. Ensure that the pin layout chosen fits some requirements (interrupts limitation etc.)
4. Construct this hardware.
5. Design a basic implementation of software that monitors and reacts to the sensor zones.
6. Test this implementation of hardware and software.
7. Implement and test an LCD clock on top of the alarm system.
8. Implement and test a basic user interaction system using the IR remote on top of the above.
9. Implement and test a user account and privileges system, allowing users to activate/deactivate the alarm, set the clock, etc.
10. Map out locations within EEPROM where values will be stored.
11. Implement and test integrity checking and defaults.
12. Allow users to change these settings and store their values in EEPROM. Test this.
13. Implement logging functionality and test.
14. Run a complete test of all functionality.
15. Make necessary changes to respond to any limitations or issues that arose in testing.
16. Finalize code and design, test, and present in class.