# CS3514 - Burglar Alarm Project

Evan Smith <113300626> and Colm Cahalane

19/11/2015

## Contents

# 1 Introduction

**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.

# 2 Requirements / Analysis / Design

## 2.1 Limitations

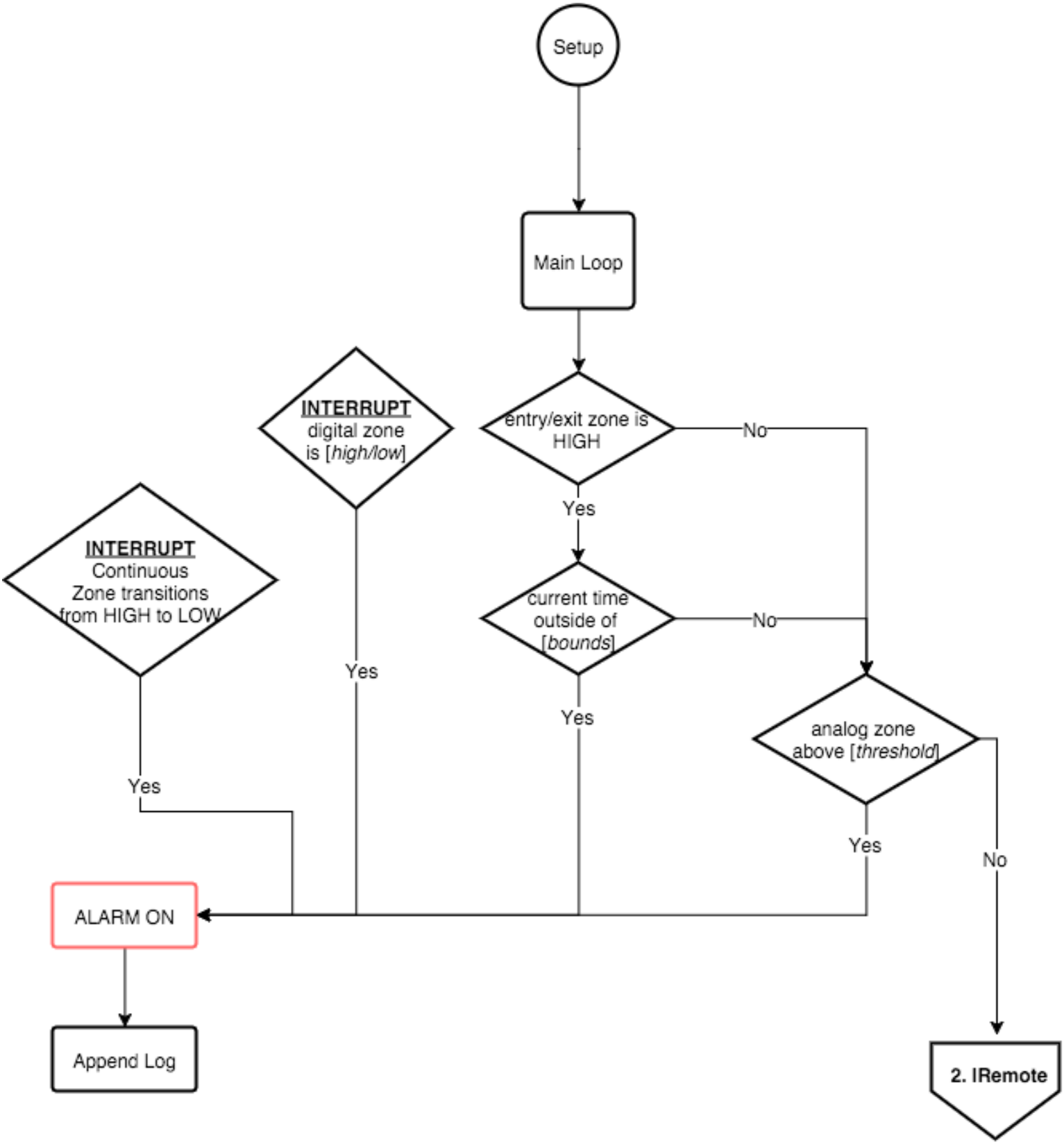
- 512MB permanent memory for logging and settings storage
- Only the admin is allowed to change settings

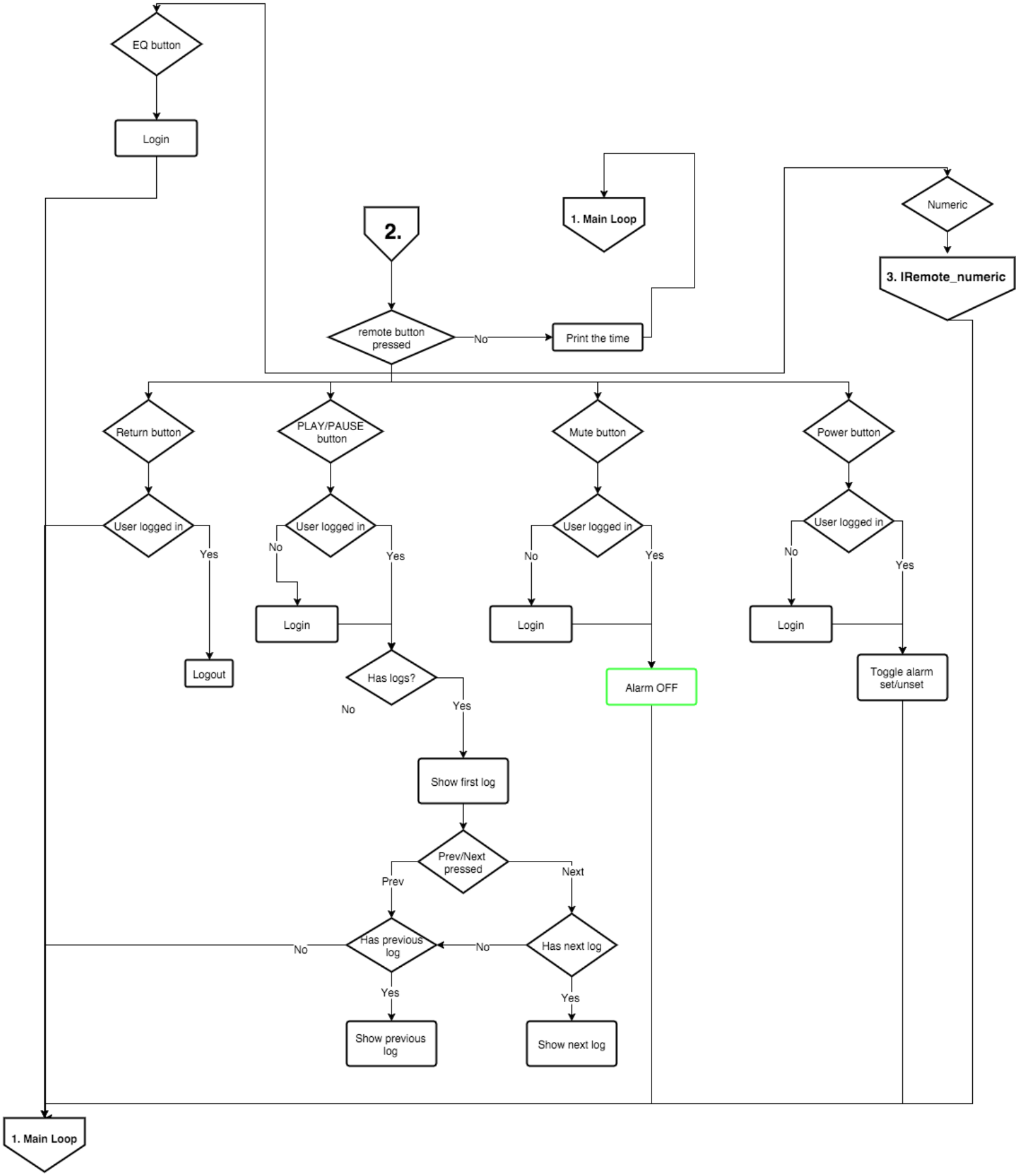
## 2.2 Features

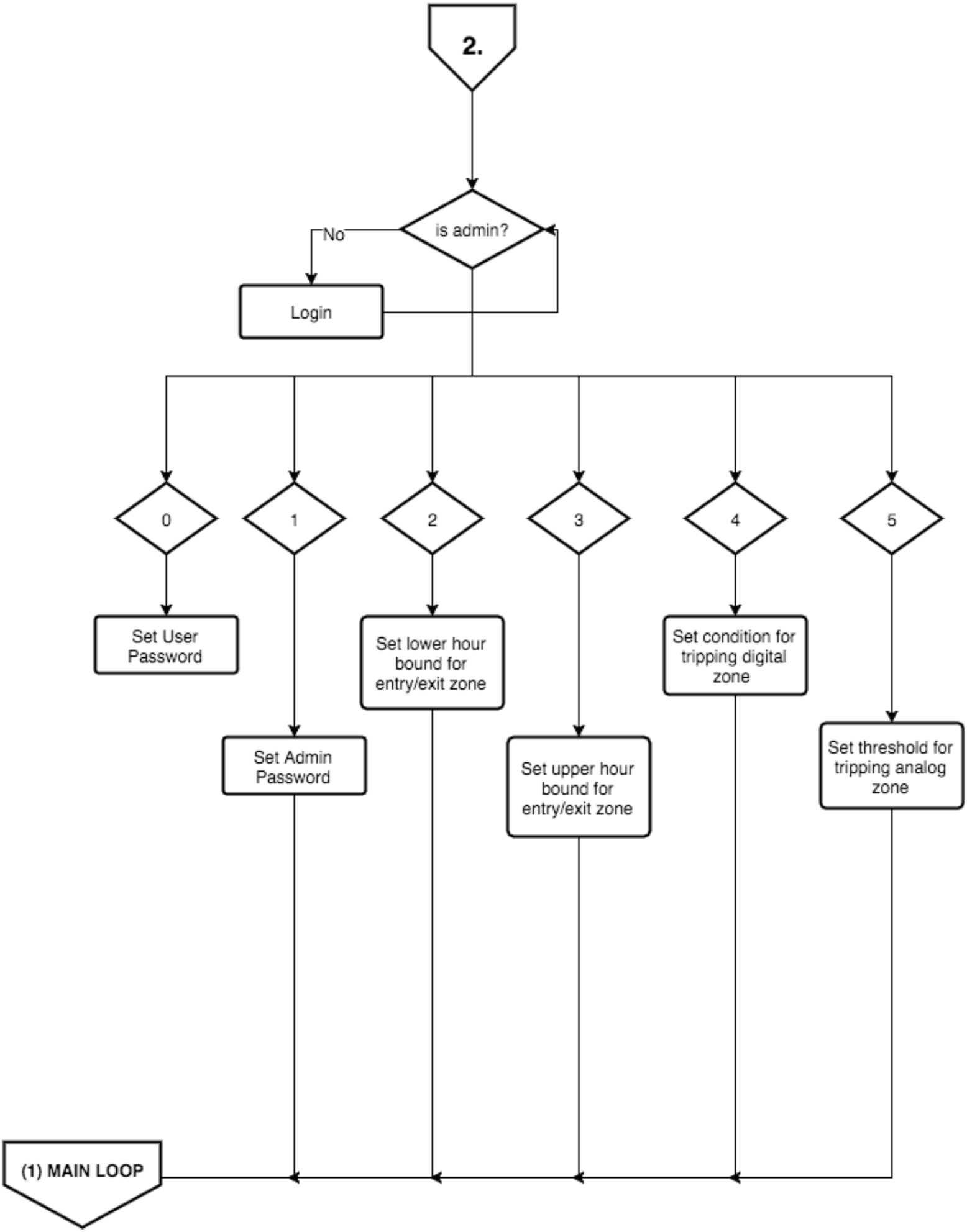
- The date and time is displayed by default on the LCD
- There are 4 zones hooked up to the alarm:
    1. **Entry/Exit:** This is linked to the front door. When tripped, it allows a certain amount of time to disable the alarm before it starts ringing.
    2. **Digital:** This would be linked to sensors attached to the windows or other vulnerable aspects of the house. When contact is broken, it sets off the alarm. E.G. If a burglar broke in by prying open the window.
    3. **Analog:** Connected to analog sensors such as a thermometer for detecting heat or variable motion sensors. This is tripped after the sensor signal breaches a certain threshold.
    4. **Continuous Monitoring:** This alarm is tripped when the signal transitions from high to low, used primarily to ensure no tampering occurs with the burglar alarm.
- A number of zones are programmable and administrators can set a variety of options like:
    - User password
    - Admin password
    - Do-not-disturb times for the entry/exit zone
    - Digital zone trip condition
- Whenever a zone is tripped, the event is logged to permanent storage
- Settings, logs, passwords and alarm configuration can be done with an Infra Red Remote

# 3 Implementation

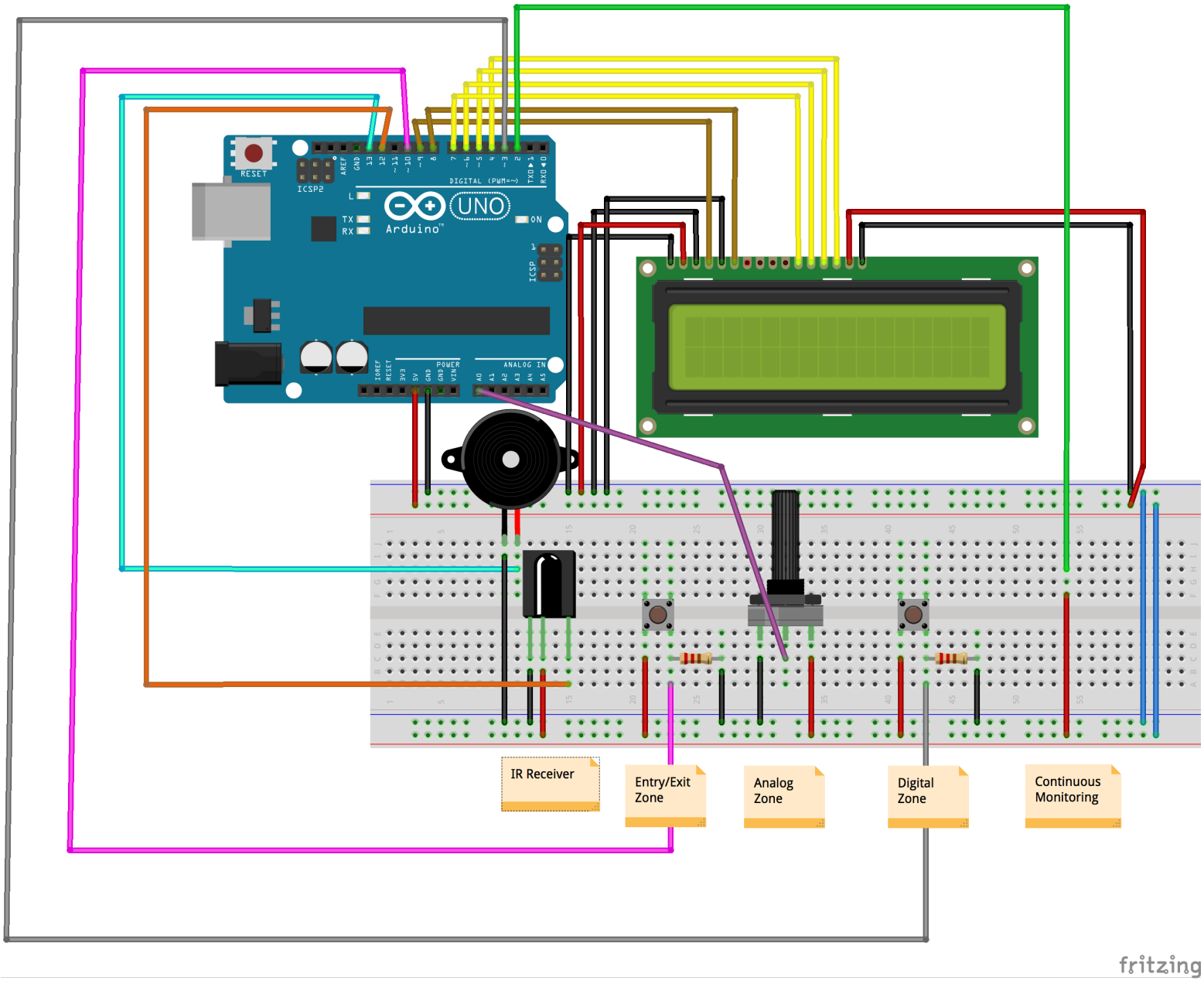
## 3.1 Flow chart




EQ button
Login
2.
1. Main Loop
Numeric
3. IRemote_numeric
remote button pressed
No
Print the time
Return button
PLAY/PAUSE button
Mute button
Power button
User logged in
Yes
No
Logout
Login
Has logs?
Alarm OFF
Toggle alarm set/unset
Show first log
Prev/Next pressed
Prev
Next
Has previous log
Has next log
Show previous log
Show next log
1. Main Loop


2.
is admin?
No
Login
0
1
2
3
4
5
Set User Password
Set Admin Password
Set lower hour bound for entry/exit zone
Set upper hour bound for entry/exit zone
Set condition for tripping digital zone
Set threshold for tripping analog zone
(1) MAIN LOOP

## 3.2 Circuit Diagram




## 3.3 Coding decisions

### 3.3.1 Global Variables

We decided to have five global variables:

```
1 // initialize the library with the numbers of the interface pins
2 LiquidCrystal lcd(9, 8, 7, 6, 5, 4);
3
4 // Used to check if current user is an admin
5 unsigned short is_admin = 0;
6
```

```
7  // 0 is disabled ; 1 is enabled
8  unsigned short alarm_set = 0;
9
10 // 0 alarm is idle ; 1 alarm is ringing
11 // Volatile because needed in interrupts
12 volatile unsigned short alarm_active = 0;
13
14 // 0 not logged in ; 1 logged in
15 unsigned short is_user_logged_in = 0;
```

All of these variables are used extensively within other functions for manipulation of the components overall, which is the reason they've been globalised.

### 3.3.2 Debouncing IR

At many points in our code, we add a delay after receiving a numeric signal from the IR Remote in order to prevent extraneous zeroes corrupting the intended number.

### 3.3.3 Default Settings on First Time

When the microcontroller is first setup, we should populate the EEPROM with default settings. Depending on the use of the microcontroller before this program was uploaded, the EEPROM may have old values still stored in its memory so we chose a sentinel value - a 3-byte char - to tell whether the defaults had been set or not.

### 3.3.4 Digital and Continuous Zones are interrupts

We chose to make the digital and continuous zones as interrupts to simplify the code and make the alarm more responsive.

The continuous zone interrupts on a falling signal only so when the signal goes from HIGH to LOW, the alarm is set off as the wire connection has been broken.

The digital zone interrupts whenever the signal changes and then checks it against the user-defined condition. If the condition is met, the alarm goes off.

```
1 attachInterrupt( digitalPinToInterrupt(DIGITAL_ZONE_PIN),
      digitalZoneTrip, CHANGE  );
2 attachInterrupt( digitalPinToInterrupt(CONTINUOUS_ZONE_PIN),
      contZoneTrip, FALLING );
```

### 3.3.5 Check For Admin

```
if( is_admin || ( loginMode( ) && is_admin ) )
```

Whenever we needed an admin login to change settings, etc., we used the fact that C lazy-evaluates to our advantage. If the admin was already logged in, he passed into the statement immediately but if we had to make them login, we used the fact that C evaluates logic conditions from left to right. So first the admin would login and if they were successful, then the `is_admin` flag would be set and we could let them in.

### 3.3.6 Logs Loop

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

When storing a log, if we've reached the maximum space (in this case it's 500 bytes to be safe), we loop around and continue writing logs at the beginning of the memory.

### 3.3.7 Increased Extensibility for Logging

```
EEPROM.put( memory_address, time_of_breach );
memory_address += sizeof(time_of_breach);
EEPROM.put( memory_address, (short) zone );
```

By using sizeof, we don't have to store a constant with the size of the time variable so we could easily increase the `LOG_LENGTH` constant, add the size of the `zone` variable to the memory address and then write a new variable/value as part of the log.

# 4 Evaluation

## 4.1 How successful was it?

The burglar alarm works as expected but the continuous zone was hard to conceptually understand. We chose a more low-tech, physical approach which achieved the same outcome but was probably not what was sought.

The EEPROM works exceptionally well and leaves a lot of room for additional settings if needed later.

The analog zone works exactly as expected with the potentiometer as an example.

Attaching the digital and continuous zone to interrupts simplified our code and gives a much more instantaneous response to actions.

The entry/exit zone also works quite nicely despite requiring us to poll for it in the loop. The countdown upon entering the house also works a lot better than expected although the count

stops while trying to login so you could theoretically enter login multiple times to keep the alarm occupied while you disabled it.

### 4.2 Any improvements?

We spent a portion of the project misunderstanding that both short and int occupy the same number of bytes, I would probably stick solely to using int data types now knowing this.

Now understanding the role of the continuous monitoring zone more closely, I would probably look at embedding software and circuity in other zones to ensure they're operating as expected instead of our proposed approach.

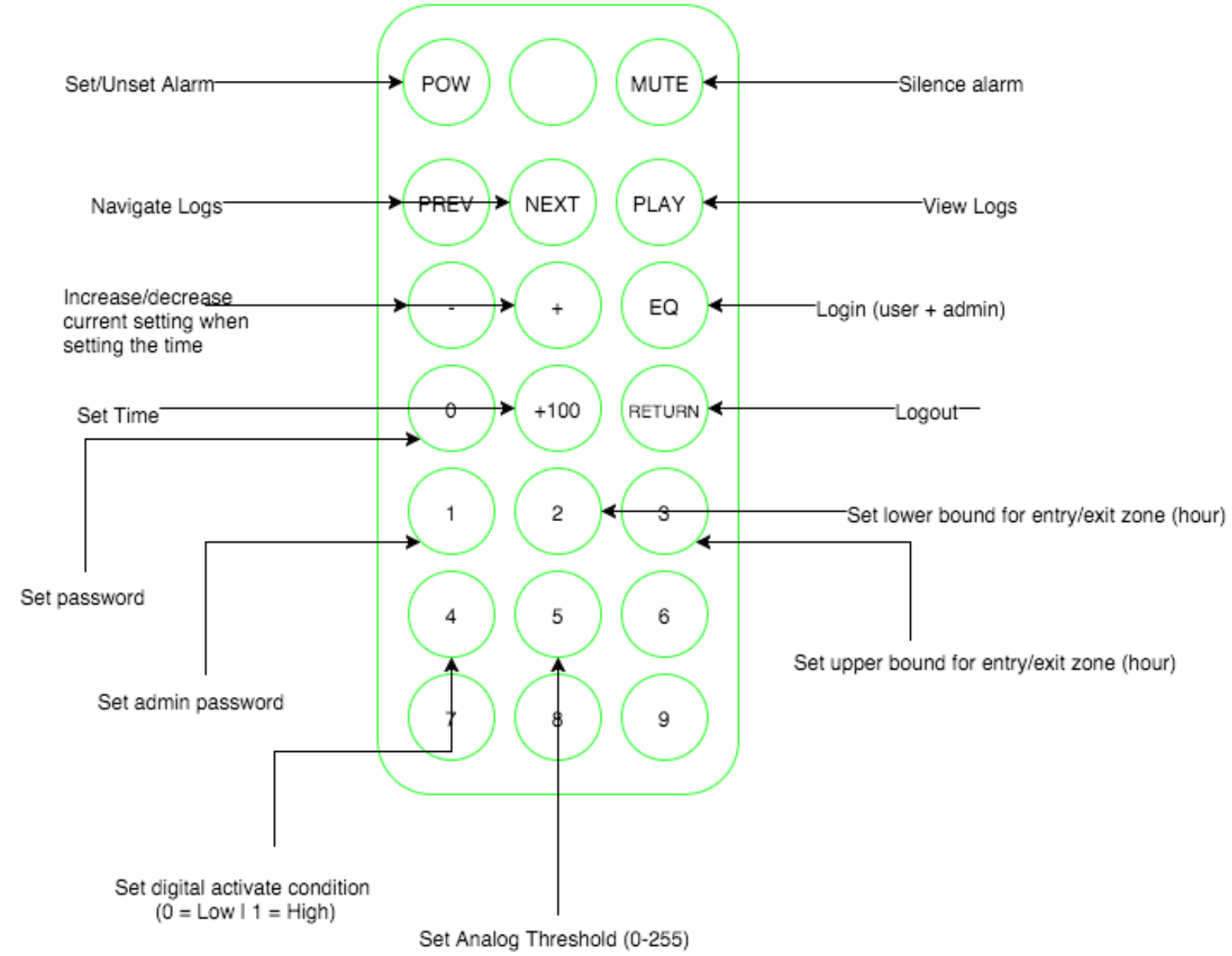
I would probably create an interrupt for timing so that we could continue to count time as someone's trying to login to the entry/exit zone. I would also set the alarm off on a failed login to avoid a brute force attack.

Logs loop but start muddying the order. We could probably fix this by also storing a sequence number and navigating through them that way.

Because so many of our settings (EG: Digital condition and the upper/lower bound hours) only needed less than 1 byte for storage but took up 2 bytes as ints, we could probably use a single int to store two settings.

## 5 Appendices

## 5.1 User manual




### 5.1.1 How do I silence the alarm?

When the alarm is active/sounding, you can silence it using the "Mute" button in the top-right corner of the remote. You will be asked to login (enter your passcode) to deactivate it unless you have recently logged in (*and have not SET the alarm*).

### 5.1.2 When will the alarm activate?

- **Analog Zone:** When the desired threshold is exceeded, the alarm will activate.
- **Digital Zone:** When the chosen condition is met (HIGH/LOW), the alarm will activate.
- **Entry/Exit Zone:** When you enter the house, you will be given 20 seconds to login and deactivate the alarm. The alarm **will not** activate if you enter your home between the two

hours you've selected in the options.

- **Continuous Zone:** When the seal of the box is broken, the two wires connected to the lid of the container will separate and the alarm will sound. This is an anti-tampering measure as the user should never be altering the direct wiring of the alarm after setup.

Whenever a zone is tripped, a log of the zone, the date and the time is written to memory for you to browse later. To view logs, hit the play/pause button and use the prev and next buttons to navigate through them.

## 5.2 Project plan

1. Map out the EEPROM memory
2. Assign functions to IR Remote buttons
3. Map all necessary pins to relevant components
4. Declare constants at the top of the code
5. Begin outlining the code
6. Assemble the hardware for testing
7. Test, re-iterate
8. Finalise the code and hardware and test again

Throughout the project, we used Github to organise our code and traded the arduino back and forth as we tackled separate hardware problems and ideas.

### 5.2.1 EEPROM Mapping

```
/**
 * 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)
 *
 */
```

### 5.2.2 IR Button Mapping

```
/**
 * 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
 */
```

### 5.2.3 Pin Mapping

```
/**
 * 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
 */
```

## 6 Code

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

/**
 * @author Colm Cahalane
 * @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
 *
```

```
35  * Analog
36  * ------
37  * 0 : Analog Zone 1
38  *
39  * IR Remote Layout
40  * ----------------
41  *
42  * EQ : Enter password (4-Digit Pin)
43  * Return : Return to standard menu
44  * Play/Pause : Navigate Log
45  *   Next     : Next log
46  *   Prev     : Previous log
47  * Mute       : Turn off alarm
48  * Power      : Set/unset the alarm
49  * 0  : Set user password
50  * 1  : Set admin password
51  * 2  : Set lower bound hour for entry/exit zone
52  * 3  : Set upper bound hour for entry/exit zone
53  * 4  : Set digital activate condition
54  * 5  : Set analog threshold
55  *
56  * EEPROM Mapping
57  * --------------
58  * 0 - 1 : password (unsigned int)
59  * 2 - 3 : admin password (unsigned int)
60  * 4 - 5 : number of breaches (unsigned short)
61  * 6 - 8 : first-time settings ("set" or anything else)
62  *
63  *  Entry/Exit (Zone 0)
64  *    15     : lower-bound hour (unsigned short)
65  *    16     : upper-bound hour (unsigned short)
66  *
67  *  Digital (Zone 1)
68  *    20    : trip condition (unsigned short)
69  *
70  *  Analog  (Zone 2)
71  *    30 - 32 : threshold (unsigned int)
72  *
73  * 100 - 511 : Logging
74  *   Bit Mapping (6 bytes each):
75  *   0 - 3 : time (unsigned long int)
76  *   4 - 5 : zone (unsigned short)
77  *
78  *
79  */
80
81  #define PASSWORD              0     // 4 digit pin
```

```
82  #define ADMIN_PASSWORD        2     // 4 digit pin
83  #define NUMBER_OF_BREACHES    4
84  #define FIRST_TIME_SET        6
85
86  // ~~~~~ ENTRY / EXIT ZONE ~~~~~
87  #define ENTRY_EXIT_ZONE       0
88  #define ENTRY_EXIT_PIN        10
89  #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++){
596       int received_value = getDigitFromIR();
597         final_value *= 10;
598         final_value += received_value;
599         lcd.print(received_value);
600         // Minor delay to prevent debouncing "0"s
601         delay(50);
602         irrecv.resume();
603     }
604     EEPROM.put( address, final_value );
605   }
606 }
607
608 /**
609  * Places default values in memory if it's the
610  * first time
611  */
612 void defaults(){
613   unsigned int password = 1234,
614                admin_password = 5678,
615                analog_threshold = 100;
616
617   unsigned short number_breaches = 0,
618                  lower_bound_hour = 20,
619                  upper_bound_hour = 22,
620                  digital_trip_condition = 1;
621
622   char first_time[] = "SET";
623   EEPROM.put( FIRST_TIME_SET, first_time );
624   EEPROM.put( PASSWORD, password );
625   EEPROM.put( ADMIN_PASSWORD, admin_password );
626   EEPROM.put( ANALOG_THRESHOLD, analog_threshold );
627
628   EEPROM.put( NUMBER_OF_BREACHES, number_breaches );
629   EEPROM.put( LOWER_TIME_BOUND, lower_bound_hour );
630   EEPROM.put( UPPER_TIME_BOUND, upper_bound_hour );
631   EEPROM.put( DIGITAL_CONDITION, digital_trip_condition );
632 }
633
634 /**
```

```
635  * Check if settings exist
636  * @return 0 if not first time; 1 if first time
637  */
638 int settingsSet( ){
639   char first_time[3];
640 
641   EEPROM.get( FIRST_TIME_SET, first_time );
642 
643   return !(first_time == "SET");
644 }
645 
646 void setup() {
647   Serial.begin(9600);
648 
649   pinMode(ALARM_PIN, OUTPUT);
650   pinMode(CONTINUOUS_ZONE_PIN, INPUT);
651   pinMode(DIGITAL_ZONE_PIN, INPUT);
652   pinMode(ENTRY_EXIT_PIN, INPUT);
653 
654   digitalWrite( ALARM_PIN, LOW );
655   digitalWrite( CONTINUOUS_ZONE_PIN, LOW );
656   digitalWrite( DIGITAL_ZONE_PIN, LOW );
657 
658   attachInterrupt( digitalPinToInterrupt(DIGITAL_ZONE_PIN),
         digitalZoneTrip, CHANGE  );
659   attachInterrupt( digitalPinToInterrupt(CONTINUOUS_ZONE_PIN),
         contZoneTrip, FALLING );
660 
661   int first_time = settingsSet( );
662 
663   if( first_time ){
664     defaults();
665   }
666 
667   alarm_set = 0;
668 
669   setTime( 1447854337 );
670 
671   irrecv.enableIRIn();
672 
673   lcd.begin(16, 2);
674 }
675 
676 void loop() {
677 
678   /**
679    * Trip the Entry/Exit zone as necessary
```

```
680     */
681   if( digitalRead(ENTRY_EXIT_PIN) == HIGH ){
682     unsigned short lower, upper, currentHour;
683     EEPROM.get( LOWER_TIME_BOUND, lower );
684     EEPROM.get( UPPER_TIME_BOUND, upper );
685     currentHour = hour();
686 
687     lcd.clear();
688     lcd.print("Plz login (EQ)");
689     long start_time = millis();
690     lcd.setCursor(0,1);
691     while( (millis() - start_time) < 20000 ){
692 
693       irrecv.resume();
694       delay(300);
695       if(irrecv.decode(&results)){
696         if( results.value == 0xFF906F ){
697           if( !is_user_logged_in ){
698            loginMode( );
699           }
700           lcd.clear();
701           lcd.print("Crisis averted");
702           delay(800);
703           break;
704         }
705       }
706       lcd.clear();
707       lcd.print("Plz login (EQ)");
708       lcd.setCursor(0,1);
709       lcd.print( 20 - ((millis() - start_time) / 1000) );
710     }
711     irrecv.resume();
712 
713     if( !alarm_active && alarm_set && !( currentHour <= lower &&
          currentHour >= upper) ){
714       // If current hour is not between the upper and lower bound
715       // then activate the alarm
716       toggleAlarm( );
717       appendLog( now(), ENTRY_EXIT_ZONE );
718       delay( 200 );
719     }
720   }
721 
722   analogZoneTrip( );
723 
724   if( irrecv.decode(&results) ) {
725     switch(results.value)
```

```
726     {
727       case 0xFF906F:
728           // EQ
729           loginMode();
730         break;
731       case 0xFFA25D:
732           // POW
733           if( !alarm_active && ( is_user_logged_in || loginMode() ) )
734             toggleAlarmSet( );
735         break;
736       case 0xFFE21D:
737           // MUTE
738           if( alarm_active && ( is_user_logged_in || loginMode() ) ){
739             toggleAlarm();
740           }
741         break;
742       case 0xFFC23D:
743           /* PLAY/PAUSE */
744           if( is_user_logged_in || loginMode() )
745             printLog( 1 );
746         break;
747       case 0xFF9867:
748           /* 100+ */
749           if(is_user_logged_in || loginMode() ){
750             changeTime();
751           }
752         break;
753
754       case 0xFFB04F:
755           // RET
756           if( is_admin ){
757             exitAdmin( );
758           } else if( is_user_logged_in ){
759             logout();
760           }
761         break;
762
763       /* SET OPTION VALUES */
764       case 0xFF6897:
765           if( is_admin || ( loginMode( ) && is_admin ) )
766             setOption( 0 );
767         break;
768       case 0xFF30CF:
769           if( is_admin || ( loginMode( ) && is_admin ) )
770             setOption( 1 );
771         break;
772       case 0xFF18E7:
```

```
773             if( is_admin || ( loginMode( ) && is_admin ) )
774               setOption( 2 );
775           break;
776         case 0xFFFFFF:
777             if( is_admin || ( loginMode( ) && is_admin ) )
778               setOption( 3 );
779           break;
780         case 0xFF10EF:
781             if( is_admin || ( loginMode( ) && is_admin ) )
782               setOption( 4 );
783           break;
784         case 0xFF38C7:
785             if( is_admin || ( loginMode( ) && is_admin ) )
786               setOption( 5 );
787           break;
788 
789         default: Serial.println("unrecognised");
790       }
791 
792       irrecv.resume(); // Receive the next value
793     } else {
794         printTime();
795     }
796 
797 }
```