ADVACNED MECHATRONICS - ME-GY 6933

Hands Free Li-fi based Multi-Door Automation

PROJECT -3 REPORT

PROJECT TEAM MEMBERS

- Adarsh Shanmugam - AS13065
- Rebanta Roy – RR3659
- Vaikunth Naarayan Shankar - VNS249

NEW YORK UNIVERSITY

Table of Contents

List of Figures

ABSTRACT

The novel Corona virus outbreak has instilled a great sense of fear in all of us. Despite washing our hands frequently, coming into contact with various objects from our day to day lives poses a great level of risk. One such object is a door handle, realizing that door handles are potential breeding grounds for a variety of infections in hospitals and SCM based warehouses. We propose a solution that allows patients, essential, and healthcare workers to open multiple doors using a smart band. This smart band contains an Arduino Nano along with an inertial measurement unit. The door automation system will run on top of the Propeller Activity Board utilizing its multi-cog functionality. As our previous project, the means of communication between these two microcontrollers would be using Light Fidelity or Li-fi. However, this time we are planning to increase the amount of security using 8-bit data. We are also aiming to increase the response time with quicker operation as this was an issue last time. The Li-fi module can be triggered by either placing the hand in a particular orientation or using a button. We feel amongst the pandemic situation, a hands-free multi-door unlocking system would serve a great deal of purpose.

INTRODUCTION

- **Li-fi**

It is a well known fact that light comes with an immense speed that makes it by far the fastest thing known to man. Hence using it for communication means very high speeds of data transmission, given the right equipments are setup to harness the best out of it. One such technology which uses the above mentioned is Li-fi. Li-Fi is a light based communication system that is capable of transmitting data at high speeds over the visible light, ultraviolet, and infrared spectrums. In its present state, only LED lamps can be used for the transmission of visible light. Using light in communication as medium to transmit data allows Li-Fi to put forward a plethora advantages over the well known established forms of communication like Wi-fi etc. Some of the advantages of Li-fi are:

- Increased bandwidth: Visible Light, Ultraviolet, and Infrared.

- The capability to safely operate in areas otherwise prone to electromagnetic disturbances like the inside of aircraft cabins, hospitals, military etc.
- Higher transmission speeds.

- **Wearable Technology/Smart Watch**

Wearable technology refers to smart electronic devices with micro-controllers that can be integrated into attire or worn on the body as an accessory. Wearable devices such as activity trackers are an example of the Internet of Things (IoT), since "things" such as electronics, software, sensors, and connectivity are effectors that enable objects to exchange data through the internet with a manufacturer, operator, and/or other connected devices, without requiring human intervention and without any loss in the quality of data being transmitted. Wearable technology has a variety of applications which grows as the field itself expands. One such popular example for a wearable technology/device that is very prominent with regards to a variety of aspects and applications is the smart watch.

DESIGN SCHEMATIC

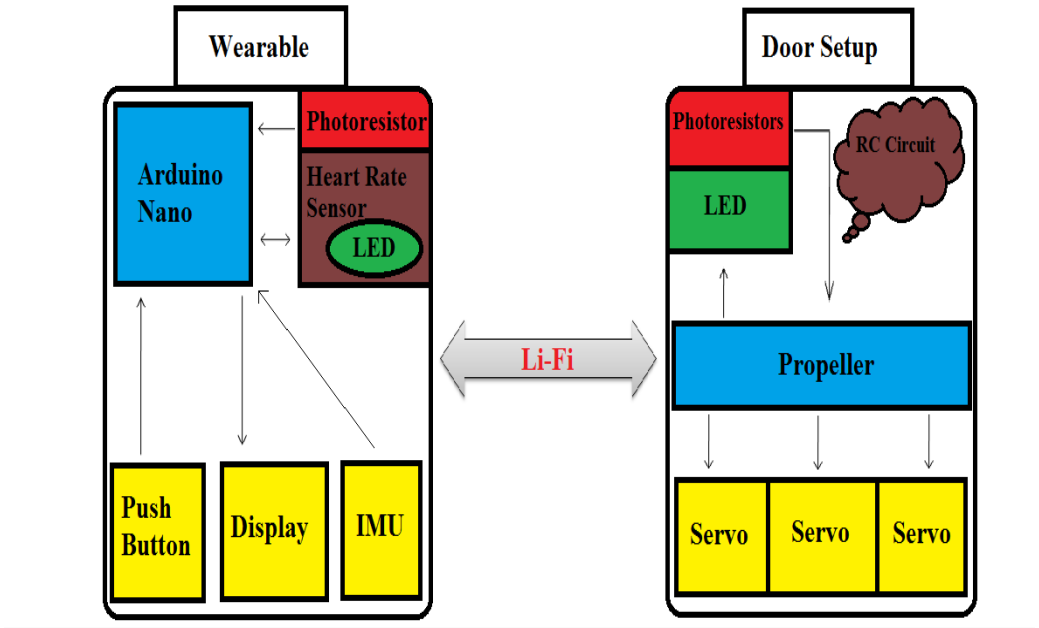

Figure 1: Design Schematic

BILL OF MATERIALS

S.No	Component	Quantity	Cost (in $)
1	Arduino Nano	1	22.00
2	Propeller Activity Board WX	1	79.00
3	MPU-6050	1	4.99
4	Parallax Servo Motors	3	30.49
5	Pulse Sensor	1	24.99
6	Photo-resistors	30	4.15
7	LEDs	10	5.00
8	SSD 1306 – 128x32 OLED Display I2C	1	8.00
9	Capacitors Kit (10pF-100nF)	300	15.99
10	Resistors Kit (220Ω-10kΩ)	150	7.89
11	Push Buttons	2	3.00
12	Jumper Wires Pack	1	5.99
		Total: $211.49	

Key Components

- **Arduino Nano**

Figure 2: Arduino Nano

Figure 3: Arduino Nano Pin Diagram

Arduino Nano is a microcontroller board. The microcontroller used in the Arduino Nano is Atmega328. It has a wide range of applications and is a major microcontroller board because of its small size and flexibility. It has 22 input/output pins in total. Out of the 22 I/O pins 14 are digital pins and 8 are analog pins. Amongst the 14 digital I/O pins, 6 are PWM compatible. It has an on-board oscillator of 16MHz. The Arduino Nano's operating voltage varies from 5V to 12V. It also supports different communications protocols like Serial Protocol, I2C Protocol, SPI Protocol. It comes with a mini USB Pin which is used to upload code.

Memory Specs: Arduino Nano

It has below memories embedded in it which are used for different purposes and are as follows:

- Flash memory of Arduino Nano is 32Kb.
- It has a preinstalled boot loader on it, which takes a flash memory of 2kb.
- SRAM memory of this Microcontroller board is 8kb.
- It has an EEPROM memory of 1kb.

The compact design and small size along with its memory specs make it an ideal choice in a variety of robotics and mechatronics applications.

- **Propeller Activity Board WX**

Figure 4: Propeller Activity Board WX

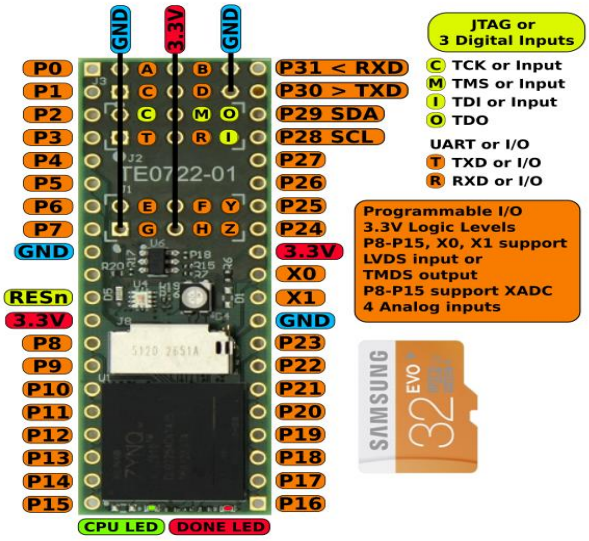

Figure 5: Propeller Pin Diagram

The Propeller Activity Board WX features the 8-core Propeller microcontroller pre-wired to a host of popular peripherals for fast computing and processing. Each of the eight 32-bit cores also called cogs has a central processing unit (CPU) which has access to 512 32-bit long words (2 KB) of instructions and data. Each cog also has access to two dedicated hardware counters and a special video generator for use in generating timing signals. The timing hardware can be used to implement various pulse-width modulation timing signals. Of the 40 available pins, 32 are used for I/O, four for power and ground pins, two for an external crystal, one to enable power outage and brownout detection, and one for reset. All eight cores can access the 32-bit port simultaneously. A special control mechanism is used to

avoid I/O conflicts if one core attempts to use an I/O pin as an output while another tries to use it as input. Any of these pins can be used for the timing or pulse-width modulation output techniques described above. The Propeller Activity Board WX features the 8-core Propeller microcontroller pre-wired to a host of popular peripherals.

- LED and Photo-resistors/ Light dependent Resistors

A light-emitting diode (LED) is a semiconductor light source that emits light when current flows through it. Electrons in the semiconductor recombine with electron holes, releasing energy in the form of photons. The color of the light is determined by the energy required for electrons to cross the band gap of the semiconductor. On the other hand, a **photo-resistor** is an active component that decreases resistance with respect to receiving light on the component's sensitive surface. The resistance of a photo-resistor decreases with increase in incident light intensity; in other words, it exhibits photoconductivity. A photo-resistor can be applied in light-sensitive detector circuits and light-activated and dark-activated switching circuits acting as a resistance semiconductor.

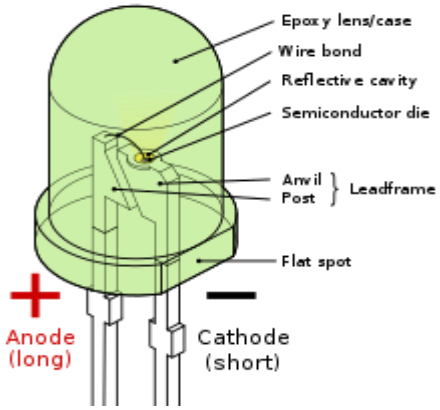

Figure 6: LED

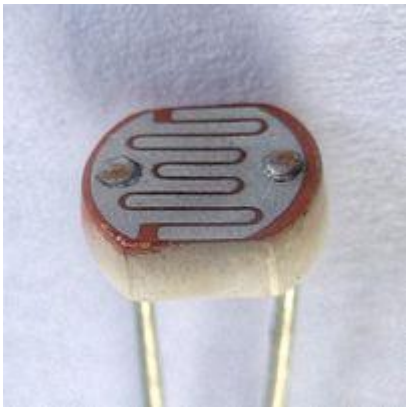

Figure 7: Photo-Resistor

- MPU-6050 (IMU)

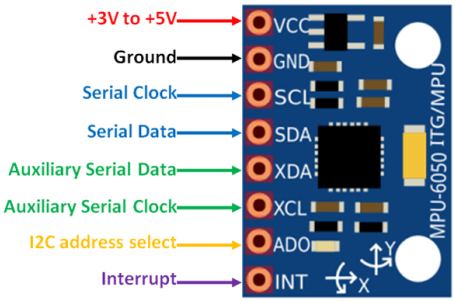

Figure 8: MPU-6050

The MPU-6050 devices combine a 3-axis gyroscope and a 3-axis accelerometer on the same silicon die, together with an onboard Digital Motion Processor (DMP), which processes complex 6-axis Motion Fusion algorithms. The device can access external magnetometers or other sensors through an auxiliary master I^2C bus, allowing the devices to gather a full set of sensor data without intervention from the system processor. For precision tracking of both fast and slow motions, the parts feature a user-programmable gyro full-scale range of ±250, ±500, ±1000, and ±2000 °/sec (dps), and a user-programmable accelerometer full-scale range of ±2g, ±4g, ±8g, and ±16g. Additional features include an embedded temperature sensor and an on-chip oscillator with ±1% variation over the operating temperature range.

- Servo Motor

Figure 9: Parallax Servo Motor

A **servomotor** is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity and acceleration. It consists of a suitable motor coupled to a sensor for position feedback. A servomotor is a closed-

loop servomechanism that uses position feedback to control its motion and final position. The input to its control is a signal (either analogue or digital) representing the position commanded for the output shaft. In this project the servo motors are used to actuate the doors to and fro.

- Pulse Sensor

Figure 10: Pulse Sensor

Pulse Sensor is a well-designed plug-and-play heart-rate sensor for Arduino. It can be used by students, artists, athletes, makers, and game & mobile developers who want to easily incorporate live heart rate data into their projects. The sensor clips onto a fingertip or earlobe and plugs right into Arduino with some jumper cables. The front of the sensor is the pretty side with the Heart logo. This is the side that makes contact with the skin. On the front you see a small round hole, which is where the LED shines through from the back, and there is also a little square just under the LED. This LED is used as a transmitter from the wearable end in Li-Fi communication.

- Display

Figure 11: SSD 1306 – 128 x 32 OLED Displays I2C

These displays are small, only about 1" diagonal, but very readable due to the high contrast of an OLED display. This display is made of 128x32 individual white OLED pixels, each one is turned on or off by the controller chip. Because the display makes its own light, no backlight is required. The driver chip SSD1306, communicates via I2C only. 3 pins are required to communicate with

the chip in the OLED display, two of which are I2C data/clock pins. The OLED and driver require a 3.3V power supply and 3.3V logic levels for communication. There is also a logic level converter and voltage regulator, so this can be operated on 5V which makes it compatible with Arduino. The power requirements depend a little on how much of the display is lit but on average the display uses about 20mA from the 3.3V supply. Built into the OLED driver is a simple switch-cap charge pump that turns 3.3v-5v into a high voltage drive for the OLEDs.

PROTOTYPE WORKING

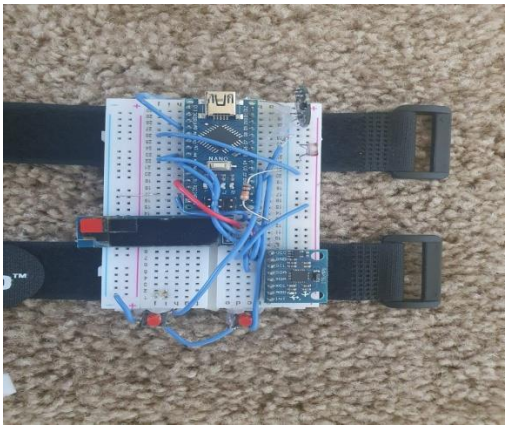

Figure 12: Breadboard model of the wearable

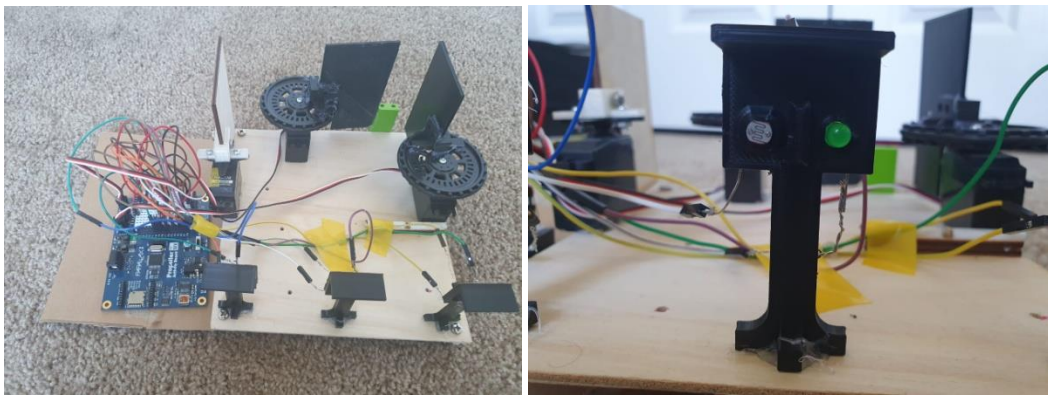

Figure 13: Door Setup

The system can be divided into two parts, the wearable and the door setup. The wearable setup comprises of an Arduino Nano, an IMU, a pulse sensor, an LDR, and an OLED display. The IMU is used to trigger the LED once the calibrated

gesture is performed, thereby making the whole process completely hands free. Whereas the door setup on the other hand consists of three pairs of LED-LDR pairs, Propeller Activity Board WX and servo motors to actuate the doors. The entire communication happens between the two parts through the LEDs and LDRs. The communication between the wearable and door setup can be categorized into the following three parts:

- Indicating the Wearable's and there on the user's presence
- Door recognition
- Door specific sequence

The first part involves the Arduino sending 6 bits of data using the LED present in the pulse sensor. Out of the 6 bits, 4 bits correspond to a sequence for wearable and one each for the start and stop bits. Since the communication is asynchronous, the usage of start and stop bits make it easier to synchronize .This is done to indicate that the wearable and thereby the user is in the vicinity of the door. Arduino Nano triggers the LED when a clockwise motion of angular velocity greater than 4.4rad/sec and an anti-clockwise motion of angular velocity lesser than -4.4rad/sec are recorded by the IMU. This sequence is read by the LDR on the door setup. Once the LDR readings are sent to the propeller by using an RC circuit and the wearable's presence in the vicinity is confirmed, the second part of the communication starts. The second part, which happens to be the recognition of the correct door starts with the propeller sending a door specific 10 bit data using the LED on the door setup, out of which 8 bits corresponds to a sequence and one each for the start and the stop bits. This sequence is read by the LDR on the wearable. This LDR data is fed to the Nano and based on the sequence transmitted, the correct door is identified thereby concluding the second part. The third part involves the Arduino sending a door specific pass code, this is also done in the form of 10 bit data with start and stop bits. This sequence is read by the LDR on the door setup and sent to the propeller. The propeller acts on the sequence. If the obtained sequence is correct, the servo motor responsible for actuating the door is actuated. A feedback is also obtained in the wearable's display. Additionally the wearable device comprises of a display, this makes it an ideal choice for the device to serve as a monitoring device. The user's heart rate,

temperature, etc can be monitored and be seen in the display. There are also push buttons available to navigate through the different interfaces of the display.

DESIGN ASPECTS

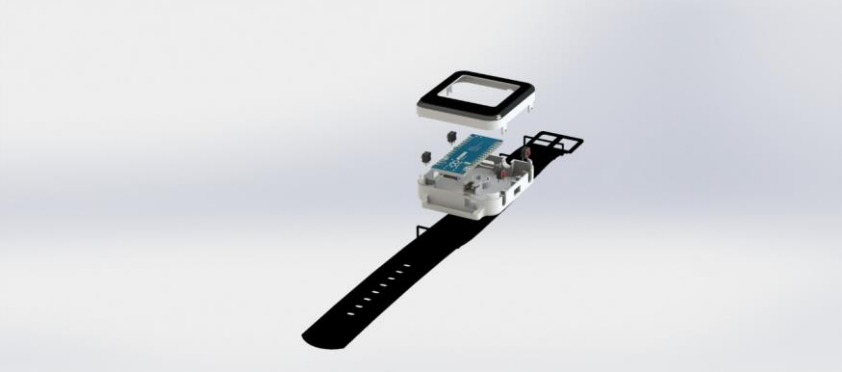

Figure 14: Planned Design for the watch

Figure 15: Door Design

While designing and fabricating a wearable device, ergonomics is always a key aspect to keep in mind as this is a device that will after all be used by people. The following factors are to be considered while designing the case that encloses the device:

- Comfort
- Intuitive and simple user interface design
- Safety & reliability
- Social acceptance and aesthetics

Note: Due to the prevailing conditions as a result of COVID-19, our access to a 3D-Printer was cut short and the proposed design could not be completed and hence a bread board model was presented instead.

ADVANTAGES

- Simple UI that requires no learning curve.
- High Speed, as light is the medium used for communication.
- Cost effective manufacturing.
- Since it operates over three regions of the electromagnetic spectrum (visible light, ultraviolet, and infrared) it's immune to the various forms of disturbances in the spectrum which other forms of communications like Wi-fi suffer from.

DISADVANTAGES

- No battery level indicator.
- If the sensor (photo resistor) is not calibrated to the scenario in which it is used, then ambient light affects the sensory data.
- Noise Sensitive.

FUTURE SCOPE

- Battery Level Indicator.
- Haptic feedback, when the push buttons are pressed.
- Using the Display as a source of light rather than a separate LED in order to simplify the design and reduce the overall size of the watch.
- Use the Nano IoT's on board IMU sensory data to trigger the LED, for a very compact design.

APPENDIX

References:

- https://ieeexplore.ieee.org/abstract/document/8771664
- https://ieeexplore.ieee.org/abstract/document/8269732
- https://ieeexplore.ieee.org/abstract/document/8728476/
- https://www.techworld.com/data/what-is-li-fi-everything-you-need-know-3632764/
- https://en.wikipedia.org/wiki/Wearable_technology
- https://www.theengineeringprojects.com/2018/06/introduction-to-arduino-nano.html
- https://www.parallax.com/product/32912
- https://www.generationrobots.com/media/DetecteurDePoulsAmplifie/PulseSensorAmpedGettingStartedGuide.pdf
- https://www.adafruit.com/product/931
- https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/
- https://www.ledsmagazine.com/leds-ssl-design/materials/article/16701292/what-is-an-led
- https://www.electronics-notes.com/articles/electronic_components/resistors/light-dependent-resistor-ldr.php

Arduino Code:

```
#include <Adafruit_MPU6050.h>

#include <Adafruit_SSD1306.h>

#include <Adafruit_Sensor.h>


#define USE_ARDUINO_INTERRUPTS true

#include <PulseSensorPlayground.h>
```

```
int sensorPin = A1;
int sensorValue = 0;

int h = 12;
int m = 0;
int s = 0;

const int OUTPUT_TYPE = SERIAL_PLOTTER;
const int PULSE_INPUT = A0;
const int PULSE_BLINK = 13;    // Pin 13 is the on-board LED
const int PULSE_FADE = 5;
const int THRESHOLD = 550;   // Adjust this number to avoid noise when idle

PulseSensorPlayground pulseSensor;

Adafruit_MPU6050 mpu;
Adafruit_SSD1306 display = Adafruit_SSD1306(128, 32, &Wire);

void setup() {
  Serial.begin(115200);
  pulseSensor.analogInput(PULSE_INPUT);
  pulseSensor.blinkOnPulse(PULSE_BLINK);
  pulseSensor.fadeOnPulse(PULSE_FADE);

  pulseSensor.setSerial(Serial);
```

```
pulseSensor.setOutputType(OUTPUT_TYPE);
pulseSensor.setThreshold(THRESHOLD);
if (!pulseSensor.begin()) {
   for(;;) {
   // Flash the led to show things didn't work.
   digitalWrite(PULSE_BLINK, LOW);
   delay(50);
   digitalWrite(PULSE_BLINK, HIGH);
   delay(50);
  }
}
// while (!Serial)
Serial.println("MPU6050 OLED demo");
pinMode(12,OUTPUT);
pinMode(2,INPUT_PULLUP);
pinMode(3,INPUT_PULLUP);
if (!mpu.begin()) {
  Serial.println("Sensor init failed");
  while (1)
    yield();
}
Serial.println("Found a MPU-6050 sensor");

// SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128x32
```

```
    Serial.println(F("SSD1306 allocation failed"));
    for (;;)
      ; // Don't proceed, loop forever
  }
  display.display();
  delay(500); // Pause for 2 seconds
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setRotation(0);
}

void loop() {
sensorValue = analogRead(sensorPin);
Serial.println(sensorValue);
sensors_event_t a, g, temp;
mpu.getEvent(&a, &g, &temp);
display.clearDisplay();
display.setCursor(0, 0);
s = s + 1;
delay(1000);
if (s == 60)
  {
    s = 0;
    m = m + 1;
  }
```

```
 if (m == 60)
 {
   m = 0;
   h = h + 1;
 }
 if (h == 13)
 {
   h = 1;
 }
homescreen();
float Y = g.gyro.y;
if(Y >= 4.4 || digitalRead(2)==0)
{
 display.setCursor(0, 18);
 display.print("Triggered!");
 display.display();
 delay(2000);
start_sequence();
while(true)
 {
   long int door_byte = sequence_read_8bits();
   if(door_byte == 11011001)
   {
   delay(5);
   Sequence1();
```

```
delay(200);
display.clearDisplay();
display.setCursor(0, 18);
display.print("Unlocked Door 1");
display.display();
delay(1800);
display.clearDisplay();
break;
}
if(door_byte == 11111001)
{
delay(5);
Sequence2();
delay(200);
display.clearDisplay();
display.setCursor(0, 18);
display.print("Unlocked Door 2");
display.display();
delay(1800);
display.clearDisplay();
break;
}
if(door_byte == 10111011)
{
delay(5);
```

```
    Sequence3();
    display.clearDisplay();
    display.setCursor(0, 18);
    display.print("Unlocked Door 3");
    display.display();
    delay(1800);
    display.clearDisplay();
    break;
    }
  }
}
if(digitalRead(3)==0)
 {
   while(true)
   {
   sensorValue = analogRead(sensorPin);
   heartrate();
   digitalWrite(12,HIGH);
   if(sensorValue>980) // To check if finger is placed
   {
   delay(1500);
   BPM();
   delay(2000);
   }
   if(digitalRead(2)==0)
```

```
   {
    digitalWrite(12,LOW);
    break;}
   }
  }
  display.display();
  delay(100);
}

void homescreen()
{
display.clearDisplay();
display.setCursor(0, 0);
display.setTextSize(2);
display.println(h);
display.setCursor(5, 0);
display.println(":");
display.setCursor(10, 0);
display.println(m);
display.setCursor(10, 0);
display.println("PM");
display.display();
delay(100);

}
```

```
void BPM()
{
int beats;
delay(20);
display.clearDisplay();
display.setCursor(0, 0);
display.setTextSize(2);
display.println("Heart Rate");
display.setCursor(0, 18);
display.setTextSize(2);
pulseSensor.outputSample();
if (pulseSensor.sawStartOfBeat())
{
beats = pulseSensor.getBeatsPerMinute();
}
display.print("BPM: ");
display.print(beats);
display.display();
delay(100);
}
void heartrate()
{
display.clearDisplay();
display.setCursor(0, 0);
```

```
display.setTextSize(2);
display.println("Heart Rate");
display.setCursor(0, 18);
display.print("BPM: ");
display.display();
delay(100);
}

void start_sequence() //1001
{
digitalWrite(12,HIGH); //Start Bit
delay(40);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,LOW); //0
delay(16);
digitalWrite(12,LOW); //0
delay(16);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,LOW); //Stop Bit
}

long int sequence_read_8bits()
{
```

```
long int door_num = 0;
sensorValue = analogRead(sensorPin);
Serial.println(sensorValue);
if(sensorValue<650)
{
 delay(24);
 sensorValue = analogRead(sensorPin);
 if(sensorValue <650)
 {
 Serial.println("Entered for loop");
 for(int i=0;i<8;i++)
      {
      sensorValue = analogRead(sensorPin);
       if(sensorValue < 650)
        {
          door_num = door_num + 1;
        }
       else
        {
          door_num = door_num + 0;
        }
       door_num = door_num*10;
       delay(20);
      }
  Serial.println("Exited for loop");
```

```
  door_num = door_num/10;
  Serial.println(door_num);
 }
}
return door_num;
}

void Sequence1() //11110100
{
digitalWrite(12,HIGH); //Start Bit
delay(40);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,LOW); //0
delay(16);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,LOW); //0
delay(16);
```

```
digitalWrite(12,LOW);//0
delay(16);
digitalWrite(12,LOW); //Stop Bit
}
void Sequence2() //11111000
{
digitalWrite(12,HIGH); //Start Bit
delay(40);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,LOW); //0
delay(16);
digitalWrite(12,LOW); //0
delay(16);
digitalWrite(12,LOW);//0
delay(16);
digitalWrite(12,LOW); //Stop Bit
```

```
}
void Sequence3() //11011100
{
digitalWrite(12,HIGH); //Start Bit
delay(40);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,LOW); //0
delay(16);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,HIGH); //1
delay(16);
digitalWrite(12,LOW); //0
delay(16);
digitalWrite(12,LOW);//0
delay(16);
digitalWrite(12,LOW); //Stop Bit
}
```

Propeller Code:

```
/*
  Blank Simple Project.c
  http://learn.parallax.com/propeller-c-tutorials
*/
#include "simpletools.h"                      // Include simple tools
#include "servo.h"
void door_two(void *par);                     //Function to start scanning for Door 2
void door_three(void *par);                   //Function to start scanning for Door 3
int analogread(int);
int read_4bits(int,int);                      //Common to all the doors
void send_8bits1(void);                       //Door no 1
void send_8bits2(void);                       //Door no 2
void send_8bits3(void);                       //Door no 3
long int read_8bits(int,int);                 //Common to all the doors
void open_door2(void);
void close_door2(void);
void open_door1(void);
void close_door1(void);
unsigned int stack1[40 + 25]; // Stack vars for cog1
unsigned int stack2[40 + 25]; // Stack vars for cog2
int here1,here2,here3;
int main()                                    // Main function
{
  // Add startup code here.
```

```
int cog1 = cogstart(&door_two, NULL, stack1, sizeof(stack1));
int cog2 = cogstart(&door_three, NULL, stack2, sizeof(stack2));
print("First Cog: %d\n", cog1);
print("Second Cog: %d\n", cog2);
while(1)
{
  // Add main loop code here.
  here1 = read_4bits(4,100);
  if(here1 == 1001)
  {
    pause(10);
    send_8bits1();
    while(1)
    {
      long int pass_byte = read_8bits(4,100);
      if(pass_byte == 11110100)
          {
          open_door1();
          pause(2000);
          close_door1();
          break;
          }
    }
  }
}
```

```
}

void door_two(void *par)
{
while(1)
 {
 here2 = read_4bits(2,30);
 if(here2 == 1001)
  {
    pause(10);
    send_8bits2();
    while(1)
    {
     long int pass_byte = read_8bits(2,30);
     if(pass_byte == 11111000)
        {
        open_door2();
        pause(2000);
        close_door2();
        break;
        }
    }
  }
 }
}
```

```
void door_three(void *par)
{
while(1)
 {
 here3 = read_4bits(6,70);
 if(here3 == 1001)
  {
    pause(10);
    send_8bits3();
    while(1)
    {
      long int pass_byte = read_8bits(6,70);
      if(pass_byte == 11011100)
         {
         servo_angle(16, 1800);
         pause(2000);
         servo_angle(16, 900);
         break;
         }
    }
  }
 }
}
```

```
int read_4bits(int a1,int thres)
{
  int sensorValue = analogread(a1);
  int start_byte = 0;
  if(sensorValue < thres)
  {
    pause(34);
    sensorValue = analogread(a1);
    if(sensorValue < thres)
    {
      for(int i=0;i<4;i++)
      {
        sensorValue = analogread(a1);
        if(sensorValue < thres)
        {
          start_byte = start_byte + 1;
        }
        else
        {
          start_byte = start_byte + 0;
        }
        start_byte = start_byte*10;
        pause(20);
      }
     start_byte =  start_byte/10;
```

```
   }
  }
return start_byte;
}

long int read_8bits(int a1, int thres)
{
  int password = 0;
  int sensorValue = analogread(a1);
  if(sensorValue < thres)
  {
    pause(34);
    sensorValue = analogread(a1);
    if(sensorValue < thres)
    {
      for(int i=0;i<8;i++)
      {
      sensorValue = analogread(a1);
        if(sensorValue < thres)
        {
          password = password + 1;
        }
        else
        {
          password = password + 0;
```

```
        }
        password = password*10;
        pause(20);
      }
     password =  password/10;
   }
 }
return password;
}

int analogread(int a)
{
high(a);
int time = rc_time(a,1);
return time;
}

void send_8bits1() //11011001
{
high(8); //Start Bit
pause(30);
high(8); //1
pause(16);
high(8); //1
pause(16);
```

```
low(8); //0
pause(16);
high(8); //1
pause(16);
high(8); //1
pause(16);
low(8); //0
pause(16);
low(8); //0
pause(16);
high(8); //1
pause(16);
low(8); //Stop Bit
}

void send_8bits2() //11111001
{
high(9); //Start Bit
pause(30);
high(9); //1
pause(16);
high(9); //1
pause(16);
high(9); //1
pause(16);
```

```
high(9); //1
pause(16);
high(9); //1
pause(16);
low(9); //0
pause(16);
low(9); //0
pause(16);
high(9); //1
pause(16);
low(9); //Stop Bit
}

void send_8bits3() //10111011
{
high(10); //Start Bit
pause(30);
high(10); //1
pause(16);
low(10); //0
pause(16);
high(10); //1
pause(16);
high(10); //1
pause(16);
```

```
high(10); //1
pause(16);
low(10); //0
pause(16);
high(10); //1
pause(16);
high(10); //1
pause(16);
low(10); //Stop Bit
}

void open_door2()
{
for(int i=1;i<20;i++)
  {
  servo_angle(17,1700);
  pause(10);
  }
  servo_angle(17, 1010);  //stop angle
}

void close_door2()
{
for(int i=1;i<20;i++)
  {
```

```
  servo_angle(17, 0);
  pause(10);
  }
  low(12);
  servo_angle(17, 1010);
}

void open_door3()
{
for(int i=1;i<20;i++)
  {
  servo_angle(12,0);
  pause(10);
  }
  servo_angle(12, 1005);  //stop angle
}

void close_door3()
{
for(int i=1;i<20;i++)
  {
  servo_angle(12, 1700);
  pause(10);
  }
  low(12);
```

```
  servo_angle(12, 1005);
}
```