Abdullah Aljandali
Class: EE354 - Digital Systems
Project 2: Hourglass, Pachinko, and Water Waves
Date: 12/5/2018

Project Description

For this project I simulated sand falling through an hourglass, pachinko, and water waves using two 8 x 8 LED matrix displays (total of 128 LEDs) and an accelerometer configured as a level sensor. The game must use the ARM cortex m4 microcontroller, and the code must be in C and Assembly. The game container must fit within a container that is no larger than 6" x 8" x 2". Nothing may extend beyond this boundary – this include switch handles, knobs, LEDs, etc. The game must be completely self-contained and battery operated. The game must be sturdy enough to survive a six foot drop onto a concrete floor.

Features:

- The final product meets the size requirements. It fits into a 6" x 8" x 2" container
- The game is self contained and battery operated
- Battery is replaceable without needing to disassemble the container
- This product uses the ARM stm32F446 Nucleo board to drive the game
- The box contains 4 buttons (Power, Reset, Game mode 1, Game mode 2)
- The box is water-proof and relatively shock-proof and does not rattle when shaken

Novel features

- The box is 3d printed and uses screws for security
- The game change on their own every now and then
- The hardware is made on a breadboard with the LEDS on the breadboard, making it require much less space

Items Demonstrated as working

- Hour-glass simulation
- Water-fall simulation
- Games react to tilt
- Switching between games

Power Requirements

The voltage was supplied to the circuit by a 9 volt battery that was dropped into 5 volts using a voltage regulator. The current was measured to be 145mA. Power = Voltage * Current = 1.305 Watts.

Discussion of Safety, Reliability, Production, and Environment

Safety Factors

- All internal components are secured and protected by a 3d printed plastic lid
- The box is completely closed with screws
- In case the box needs to be opened, all wires are glued with electric tape for safety.
- The battery has its own separate section of the box
- The box is light weight, and won't hurt if it falls down.

Reliability Factors

- The only issue is that the female header pins on the ARM processor are somewhat unreliable. Though in a practical situation, this problem would be solved with soldering (which we couldn't use on the ARM for this project).

Economic Factors

- The product is cheap as it uses a small breadboard and wires that cost around 5$ together
- It uses minimum plastic needed to keep it strong but at the same time as thin as possible

Manufacturability Factors

- The box is 3d printed. There is a 3d design which allows for automatic manufacturing of boxes
- This project could have also used a PCB to make mass production much easier but that was given up for the economic factor (a breadboard would cost 5$ compared to 60$ for the PCBs). I designed a PCB in ExpressPCB website which sells 3 PCBs for 60$. Though, I did not need all three PCBs,therefore, I decided to use a breadboard instead.
- The product is easy to assemble as there aren't many wires because the LEDS go straight on a breadboard. It only needs to be connected to the ARM Cortex

Environmental Factors

This project does not consider the environment. In fact, environmental factors were given up in exchange for manufacturability and economic factors. Although plastic and hot glue are harmful for the environment, they are cheap. Also, using plastic allows for easier manufacturability as the 3d printed design can be mass produced. The product also uses a 9v volt battery even though the components require only 5v which results in power less.

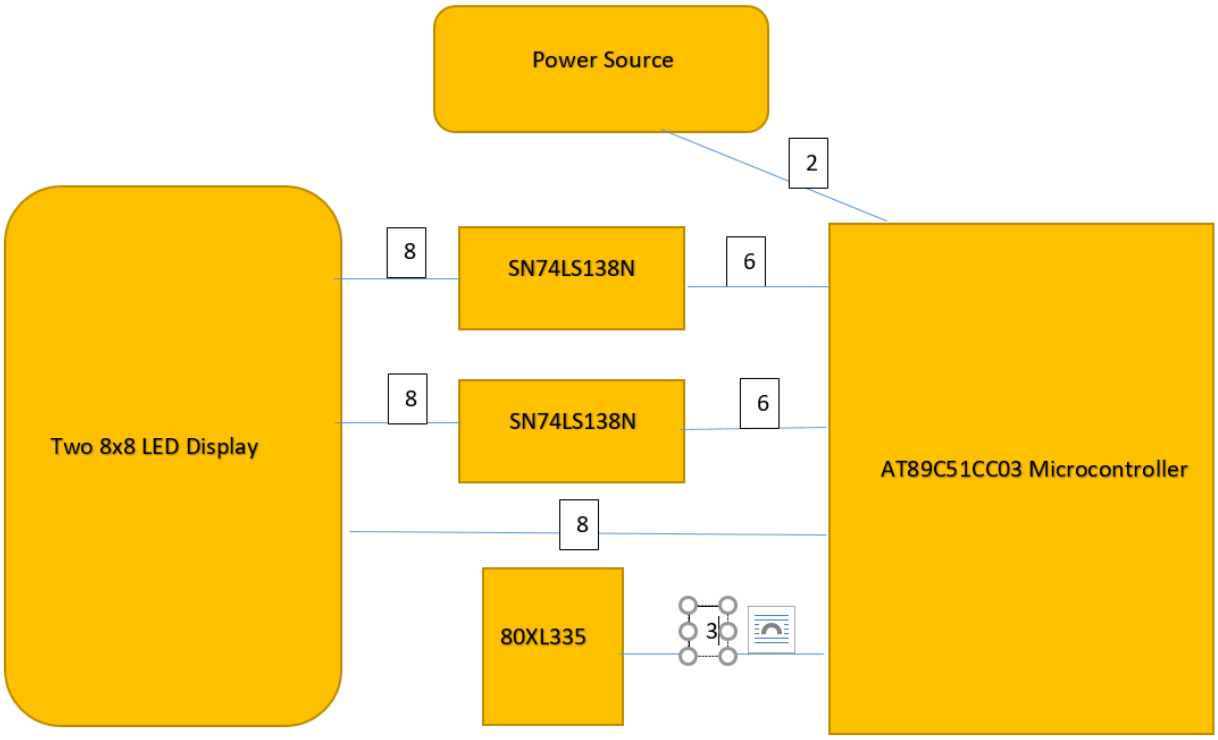
Hardware Details

Components used:

- AT89C51CC03 Microcontroller - x1
- CLM-1588B LED LED Matrix- x2
- SN74LS138N Decoders -x2
- 360 Ohms Resistor - x8

- Generic Switch -x4
- Generic Push Button -x2
- Small solderable circuit board -x1
- 9v Battery with switchable Battery Holder -x1
- 80XL335 Level sensor -x1

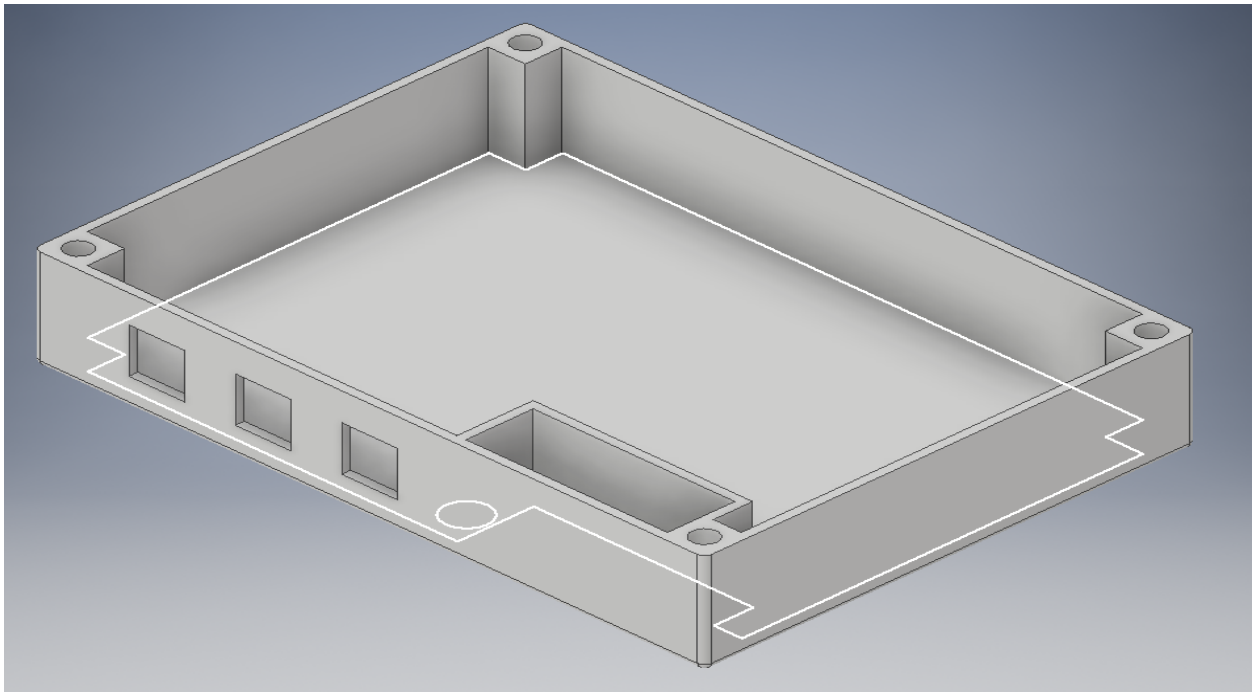
System Diagram:



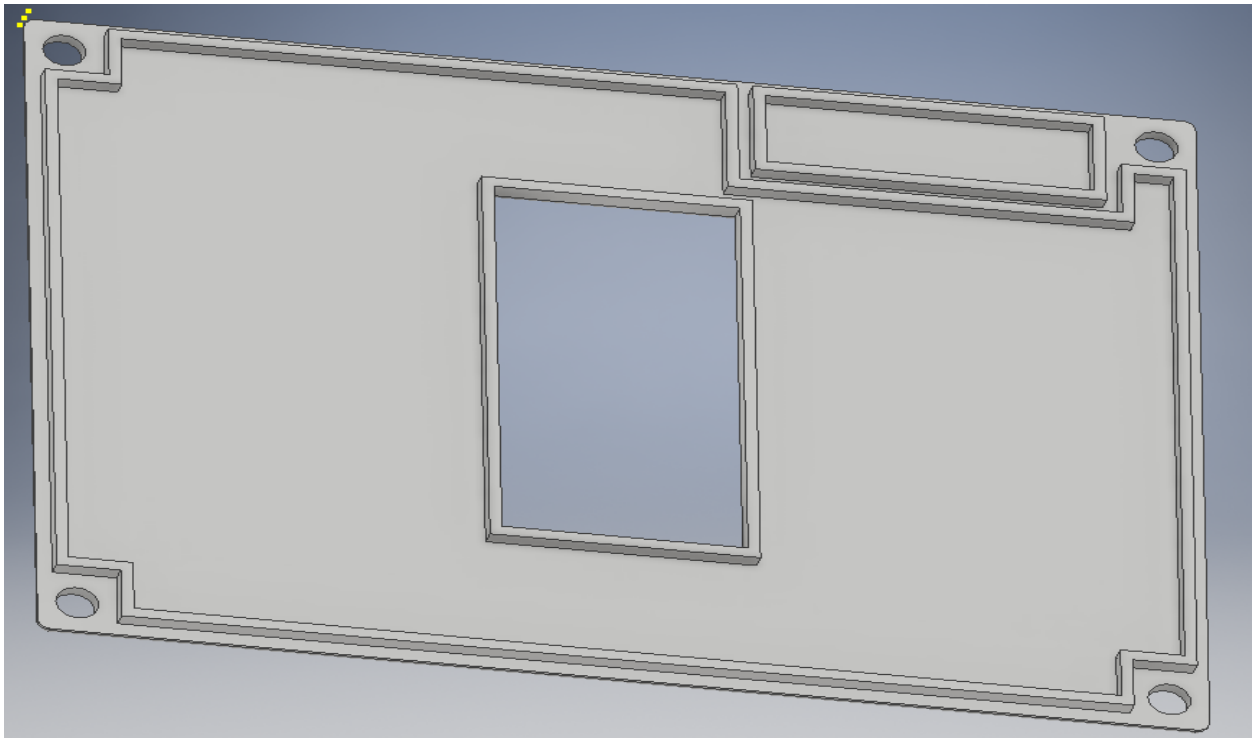
Mechanical Diagram:

The physical dimensions are measured to be 7.5” x 5.5” x 1.8”

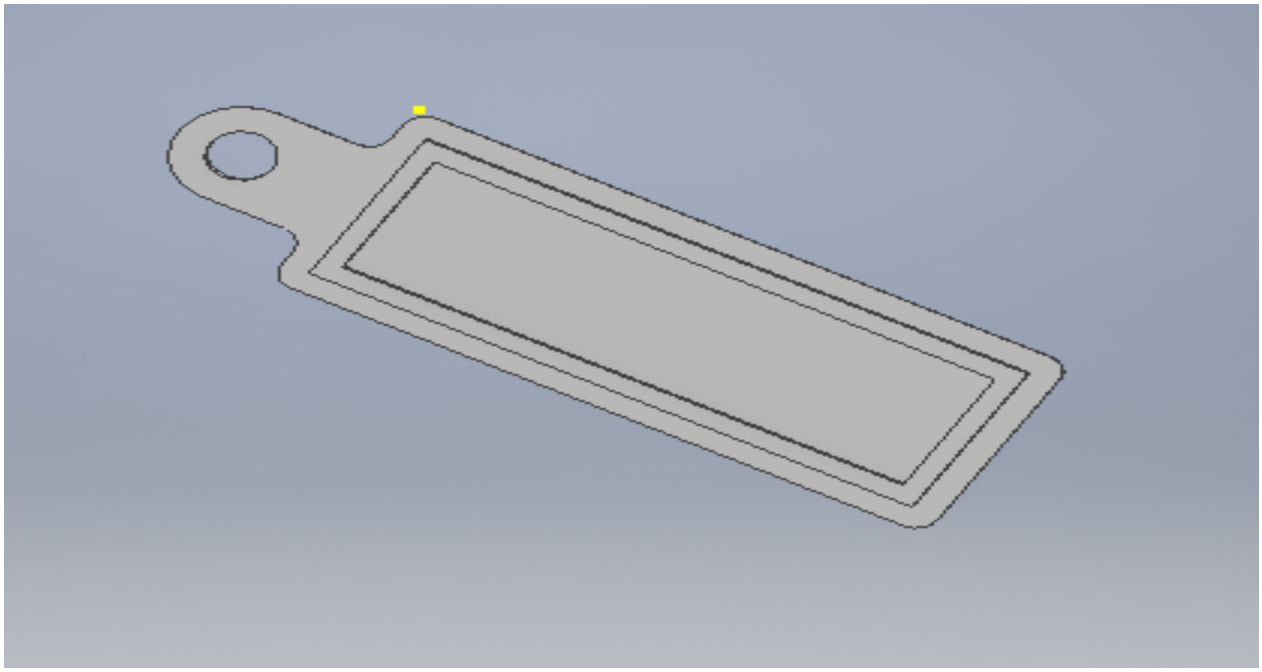
Box Base:

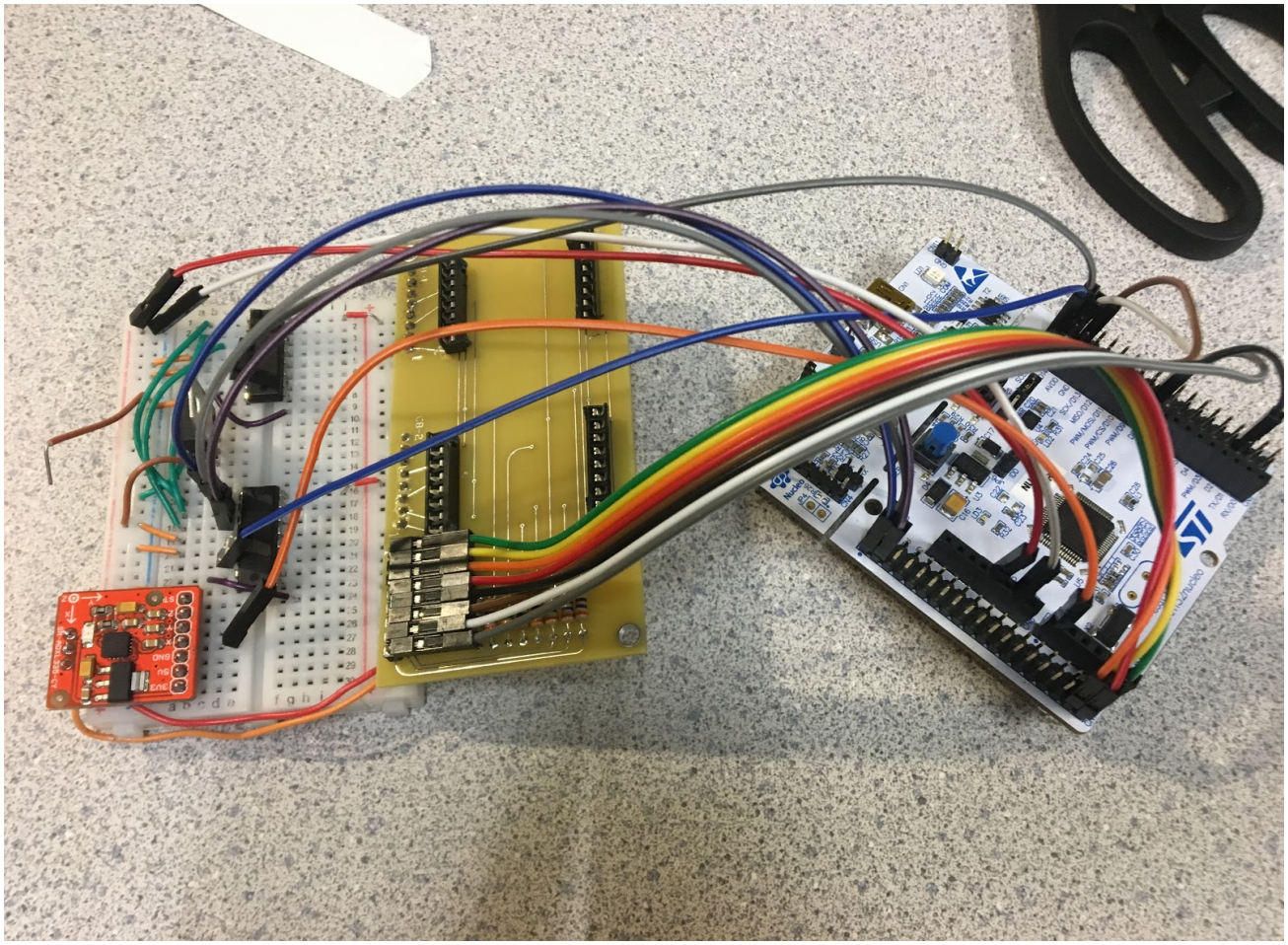


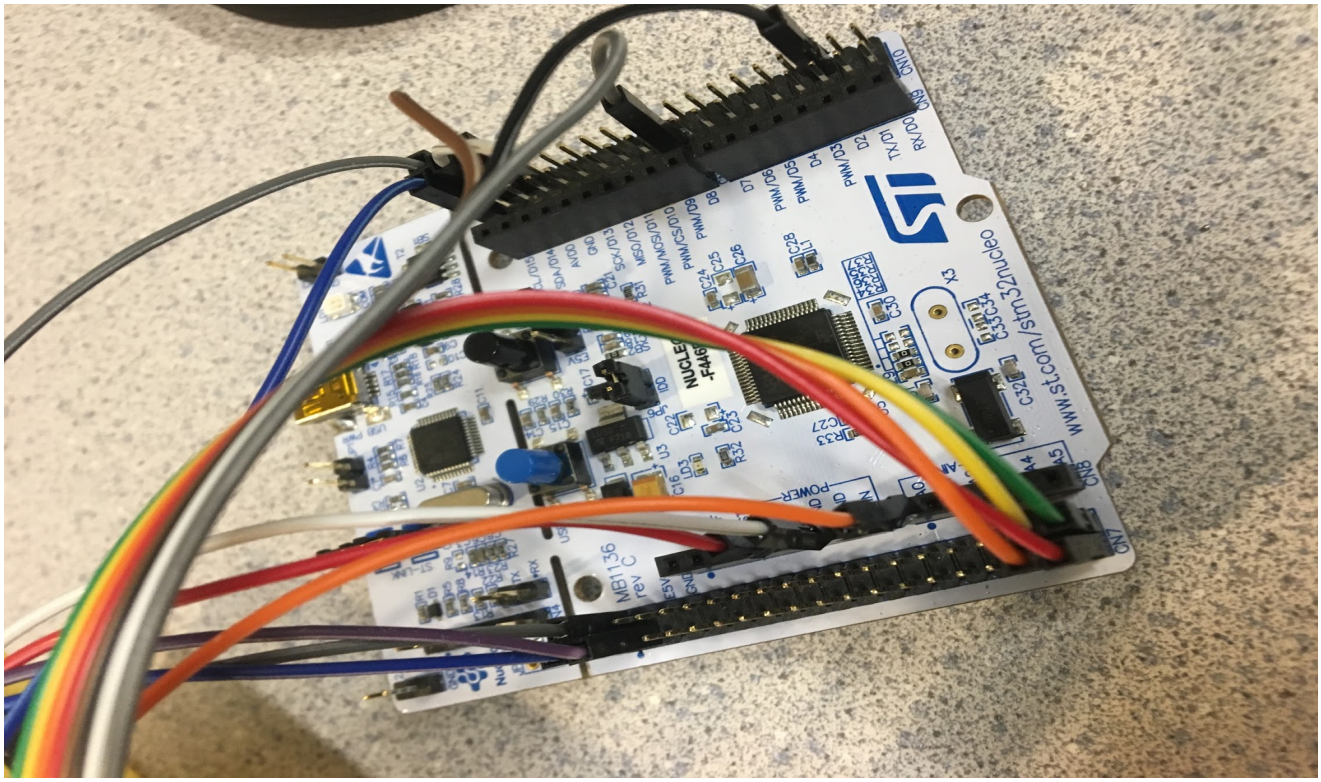
Box lid:

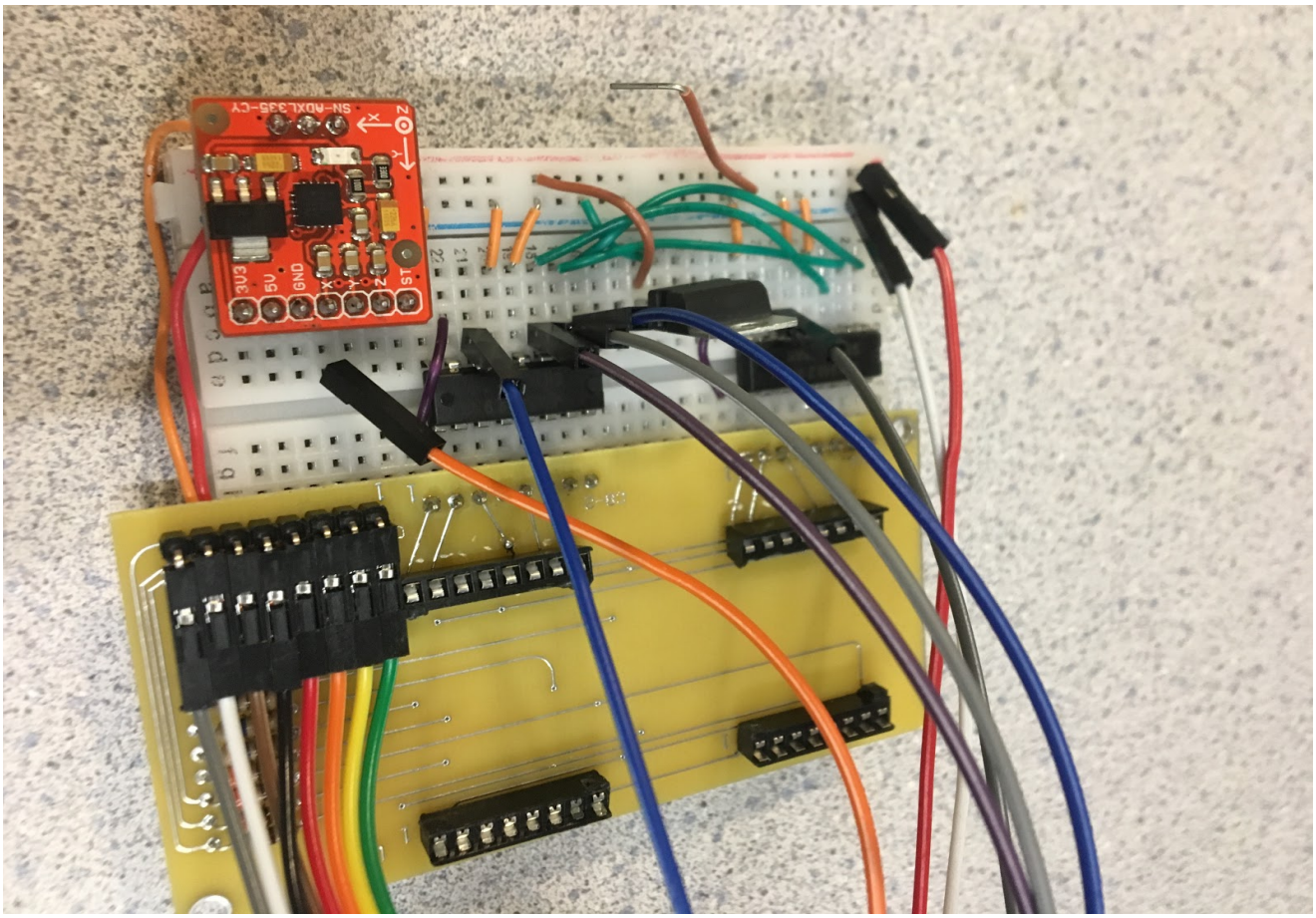


Battery lid:






www.st.com/stm32nucleo
MB1136
rev C


SN-ADXL335-CY
3V3
5V
GND
X
Y
Z
ST

Software Details

Keil uVision5 was used to compile the C code and send it to the ARM board.

Pseudocode

```
Initialize global constants
Initialize global variables
unsigned char leds[16][8]
unsigned char rows[16]

Main code{
        Enable Port A Clock
        Enable Port C Clock
        Make PC port an output
        Set pull up pull down off
        Start conversion initial
        Enable IDR for Port A

        while(run forever){
                Check buttons for game mode
                Clear LED memory map
                Place game outline in memory map
                AD conversion
                Load in value from accelerometer
                if(x < 1900 ){ // if the box is tilted up
                        if(mode == 1)
                          moveSandUp();
                        else if (mode == 2)
                          moveWaterUp();
                        else if(mode ==3)
                           movePachUp();

         }
        else if (x > 2200){ //if the box is tilted down
            if(mode == 1)
             moveSandDown();
            else if (mode == 2)
```

```
            moveWaterDown();
           else if(mode ==3)
            movePachDown();
       }
      for(int i =0; i<100; i++)
       Led_out();//displays the rows using the current array values

 }
void Led_out(){
      Take in value from port C
      for ( int i = 0; i < 8; i++)
      {
            Enable encoder 1 //PC8
            Send row i to encoder //PC10-12
            Send data at this row from the array // PC0-7
            delay
      }
      for ( int i = 0; i < 8; i++)
      {
            Enable encoder 2 //PC9
            Send row i to encoder //PC10-12
            Send data at this row from the array // PC0-7
            delay
      }
}
void moveSandDown(){
for( unsigned char  i = 14; i != 255; i--){
   for( unsigned char j = 7; j != 255; j--){
      if the led is on
            If led below it is off
                  Move there
            Else if led below it and to the right is off
                  Move there
            Else if led below it and to the left is off
                  Move there
}

void moveSandUp(){
      for( unsigned char  i = 14; i != 255; i--){
```

```
   for( unsigned char j = 7; j != 255; j--){
     if the led is on
          If led above it is off
               Move there
          Else if led above it and to the right is off
               Move there
          Else if led above it and to the left is off
               Move there
}
void moveWaterDown(){
for( unsigned char  i = 14; i != 255; i--){
  if((i <15 && i > 10)||(i < 4)){
   for( unsigned char j = 7; j != 255; j--){
    If here is on{
     if led below it is off
          Move There
     Else if led below it and to the right is off
          Move there
     Else if led below it and to the left is off
          Move there
     }
   }
}
void moveWaterUp(){
for( unsigned char  i = 1; i != 16; i++){
  if((i <16 && i > 11)||(i < 4)){
   for( unsigned char j = 0; j != 8; j++){
    If here is on{
     if led above it is off
          Move There
     Else if led above it and to the right is off
          Move there
     Else if led above it and to the left is off
          Move there
     }
   }
}

void resetHour(){
```

```
     Reset the row array to look like an hourglass
     Reset the leds array to look like an hourglass
}
//resets the arrays to make to the initial water glass
void resetWater(){
     Reset the row array to look like an waterglass
     Reset the leds array to look like an waterglass
}
```

C Source Code

```
//Author: Abdullah Aljandali
//School: University of Evansville
//Course: EE354
//Instructor: Dr. Dick Blandford
//Assignment: Project 2: Hourglass Simulation

#include "stdlib.h"
#include "stm32f446.h"

//outputs to the decoders and driver what pins to turn on
void Led_out();
//simulates falling sand by changing the values of the LED arrays
void moveSandDown();
//simulates falling sand when the box is tilted up
void moveSandUp();
//simulates falling water by changing the values of the LED arrays
void moveWaterDown();
//simulates falling water when the box is tilted up
void moveWaterUp();
//simulates pachinko by changing the values of the LED arrays
void movePachDown();
//simulates pachinko when the box is tilted up
void movePachUp();
//resets the arrays to make to the initial hour glass shape
```

```
void resetHour();
//resets the arrays to make to the initial water glass
void resetWater();
//resets the arrays to make to the initial plachinko game
void resetPach();
unsigned char rands[250];
unsigned char rows [16]; // used for output
unsigned char leds [16][8];
//0 is light off
//1 is light on
//2 is obstacle/wall

int mode = 2; //decides which of the three games. 1
// 1 for hour glass\
// 2 for waterfall
// 3 for pachinko

int main()
  {

  //stores random variables to be used for pachinko
  for(int i=0; i<250;i++)
   rands[i] = rand();
      int x = 0;
  //Enable Port A clock
  RCC_AHB1ENR |= 1;
  //Enable Port C clock
      RCC_AHB1ENR |= 1<<2;
  RCC_APB2ENR |= 0x100;
      // A/D clock
      GPIOA_MODER |= 0x3FF;
  //Entire PC port is output
  GPIOC_MODER |= 0x555555555;
  //Set Pull up pull down off
  GPIOA_PUPDR &= 0xFFFFFFFF3;
      ADC1_CR2 |= 1; //A/D enable
  ADC1_SQR3 = 1;
  //Start conversion initial
    ADC1_CR2 |= 1<<30;
```

```
// button for game hourglass
int btn_hourGlass = GPIOA_IDR & (1<<2);
    // button for game waterglass
int btn_waterFall = GPIOA_IDR & (1<<3);
    // button for reset
int btn_Reset = GPIOA_IDR & (1<<4);

//if the switch is on
if(btn_hourGlass  !=0)
 mode = 1;
if(btn_waterFall  !=0)
 mode = 2;


if(mode == 1)
 resetHour();
else if(mode==2)
 resetWater();
else if(mode == 3)
 resetPach();
 while(1)
  {
  //if they click the reset button
  //reset the current game
  if(btn_Reset !=0){
   if(mode == 1)
    resetHour();
   else if(mode==2)
    resetWater();
   else if(mode == 3)
    resetPach();
   }

   ADC1_CR2 |= 1<<30;//Start conversion
   unsigned int TimeOut = 0;
   while(((ADC1_SR &(1<<1))== 0) && TimeOut < 100000)
   {
    TimeOut++;//Timeout variable in case of error
```

```
      }
      //Load in A/D value
      x = ADC1_DR;
      if(x < 1900 ){ // if the box is tilted up
       if(mode == 1)
        moveSandUp();
       else if (mode == 2)
        moveWaterUp();
       else if(mode ==3)
        movePachUp();
      }
      else if (x > 2200){ //if the box is tilted down
       if(mode == 1)
        moveSandDown();
       else if (mode == 2)
        moveWaterDown();
       else if(mode ==3)
        movePachDown();
      }
      for(int i =0; i<100; i++)
       Led_out();//displays the rows using the current array values
 }
}
//resets hour glass to initial shape
void resetHour(){

 //update the rows array used for output
  rows[0] = 0xFF;
  rows[1] = 0xFF;
  rows[2] = 0xFF;
  rows[3] = 0xFF;
  rows[4] = 0x7E;
  rows[5] = 0x3C;
  rows[6] = 0;
  rows[7] = 0;
  rows[8] = 0;
  rows[9] = 0;
  rows[10] = 0;
  rows[11] = 0;
```

```
  rows[12] = 0;
  rows[13] = 0;
  rows[14] = 0;
  rows[15] = 0;

 //update the leds array used for logic
 for(unsigned char i = 0; i < 16; i++){
  for(unsigned char j = 0; j<8 ; j++){
    if( i < 7)
     leds[i][j] = 1;
    else
     leds[i][j] = 0;
  }
 }

 leds[5][0] = 2; leds [5][7] = 2;
 leds[6][0] = 2; leds [6][1] =2; leds[6][6] = 2; leds[6][7] = 2;
 leds[7][0] = 2; leds [7][1] = 2; leds[7][2] = 2; leds[7][5] = 2; leds
[7][6] = 2; leds[7][7] = 2;
 leds[8][0] = 2; leds [8][1] = 2; leds[8][2] = 2; leds[8][5] = 2; leds
[8][6] = 2; leds[8][7] = 2;
 leds[9][0] = 2; leds [9][1] =2; leds[9][6] = 2; leds[9][7] = 2;
 leds[10][0] = 2; leds [10][7] = 2;
}
//resets water glass to initial shape
void resetWater(){

 //resets the rows array to the initial
  rows[0] = 0xFF;
  rows[1] = 0xFF;
  rows[2] = 0xFF;
  rows[3] = 0;
  rows[4] = 0;
  rows[5] = 0;
  rows[6] = 0;
  rows[7] = 0;
  rows[8] = 0;
  rows[9] = 0;
  rows[10] = 0;
```

```
  rows[11] = 0;
  rows[12] = 0;
  rows[13] = 0;
  rows[14] = 0;
  rows[15] = 0;

 //resets the leds arrow to the initial shape
 for(unsigned char i = 0; i < 16; i++){
  for(unsigned char j = 0; j<8 ; j++){
    if( i < 3) //first three rows are on
     leds[i][j] = 1;
    else if( i==4 && j < 7)//the fifth row is an obstacle
     leds[i][j]=2;
    else if( i==11 && j >0)//the twelfth row is an obstacle
     leds[i][j]=2;
    else
     leds[i][j] = 0;//other LEDs are off
  }
 }
}
//resets water glass to initial shape
void resetPach(){

 //resets the rows and leds arrow
 //to the initial pachinko game
 //which is basically all lights off
  rows[0] = 0;
  rows[1] = 0;
  rows[2] = 0;
  rows[3] = 0;
  rows[4] = 0;
  rows[5] = 0;
  rows[6] = 0;
  rows[7] = 0;
  rows[8] = 0;
  rows[9] = 0;
  rows[10] = 0;
  rows[11] = 0;
  rows[12] = 0;
```

```
  rows[13] = 0;
  rows[14] = 0;
  rows[15] = 0;

 for(unsigned char i = 0; i < 16; i++){
  for(unsigned char j = 0; j<8 ; j++){
     leds[i][j] = 0;
  }
 }
}
//simulates falling water when the box is tilted up
void moveWaterUp(){

// go throw all rows except the top row because it cannot go up
anymore
for( unsigned char  i = 1; i != 16; i++){
//if i is a row in the regions that allows water to move in the left
direction
  if((i <16 && i > 11)||(i < 4)){
   for( unsigned char j = 0; j != 8; j++){ // go throw columns left to
right
     if(leds[i][j] == 1){ //if current led is lighted up
      //if the led above is empty
      if(leds[i-1][j] == 0){

       leds [i-1][j] = 1;
       leds [i][j] = 0;
       rows [i] &= ~(1<<j);
       rows [i-1]  |=  1<<j;
      }
      //if the led above it and to the left is empty
      else if(j >0 && leds[i-1][j-1] == 0){
       leds [i-1][j-1] = 1;
       leds [i][j] = 0;
       rows [i] &= ~(1<<j);
       rows [i-1]  |=  1<<(j-1);
      }
      //if the led to the left is empty
       else if(j >0 && leds[i][j-1] == 0){
```

```
       leds [i][j-1] = 1;
       leds [i][j] = 0;
       rows [i] &= ~(1<<j);
       rows [i]  |=  1<<(j-1);
      }
     }
   }
  }
//if i is a row in the regions that allows water to move in the right
direction
  else{
   for( unsigned char j = 7; j != 255; j--){ // go throw columns right
to left
    if(leds[i][j] == 1){//if current led is lighted up
     //if the led above is empty
     if(leds[i-1][j] == 0){
      leds [i-1][j] = 1;
      leds [i][j] = 0;
      rows [i] &= ~(1<<j);
      rows [i-1]  |=  1<<j;
     }
     //if the led above it and to the left is empty
     else if(j < 7 && leds[i+1][j+1] == 0 && i!=5){
      leds [i-1][j+1] = 1;
      leds [i][j] = 0;
      rows [i] &= ~(1<<j);
      rows [i-1]  |=  1<<(j+1);
     }
      //if the led to the left is empty
      else if(j < 7 && leds[i][j+1] == 0){
      leds [i][j+1] = 1;
      leds [i][j] = 0;
      rows [i] &= ~(1<<j);
      rows [i]  |=  1<<(j+1);
     }
    }
   }

  }
```

```
 }
}
//simulates falling water when the box is tilted down
void moveWaterDown(){
 for( unsigned char  i = 14; i != 255; i--){ // go throw all rows
except the bottom because it cannot go down
  //if i is a row in the regions that allows water to move in the
right direction
  if((i <15 && i > 10)||(i < 4)){
   for( unsigned char j = 7; j != 255; j--){ // iterate columns right
to left
    //if current led is on
    if(leds[i][j] == 1){
     //if led below it is off
     if(leds[i+1][j] == 0){
      leds [i+1][j] = 1;
      leds [i][j] = 0;
      rows [i] &= ~(1<<j);
      rows [i+1]  |=  1<<j;
     }
     //if led below it and to the right is off
     else if(j < 7 && leds[i+1][j+1] == 0){
      leds [i+1][j+1] = 1;
      leds [i][j] = 0;
      rows [i] &= ~(1<<j);
      rows [i+1]  |=  1<<(j+1);
     }
     //if led below it and to the left is off
      else if(j < 7 && leds[i][j+1] == 0 ){
      leds [i][j+1] = 1;
      leds [i][j] = 0;
      rows [i] &= ~(1<<j);
      rows [i]  |=  1<<(j+1);
     }
    }
   }
  }
//if i is a row in the regions that allows water to move in the left
direction
```

```
  else{
   for( unsigned char j = 0; j != 8; j++){ // iterate columns left to
right
     if(leds[i][j] == 1){ //if current led is on
      //if led below it is off
      if(leds[i+1][j] == 0){
       leds [i+1][j] = 1;
       leds [i][j] = 0;
       rows [i] &= ~(1<<j);
       rows [i+1]  |=  1<<j;
      }
      //if led below it and to the left is off
      else if(j >0 && leds[i+1][j-1] == 0 && i!=10){
       leds [i+1][j-1] = 1;
       leds [i][j] = 0;
       rows [i] &= ~(1<<j);
       rows [i+1]  |=  1<<(j-1);
      }
      //if led to the left is off
       else if(j >0 && leds[i][j-1] == 0){
       leds [i][j-1] = 1;
       leds [i][j] = 0;
       rows [i] &= ~(1<<j);
       rows [i]  |=  1<<(j-1);
      }
     }
   }
  }
 }
}
//simulates pachinko when the box is tilted down
void movePachDown(){
// int rnd;
// for(int i =0; i<16; i++){
//  //for(unsigned long int l = 0; l <99999; l++);
//  rnd = rand()%8;
//  leds[i][rnd] = 1;
//  rows[i] |= (1<<rnd);
//  if(i<15 && leds[i+1][rnd] !=1){
```

```
//   leds[i][rnd] = 0;
//   rows[i] &= ~(1<<rnd);
//  }
//  for(int k = 0; k <10000; k++)
//   Led_out();
// }
 /* You have to declare an array of a certain size,
 can be anything that you want it to be, then you
 need to fill it up at the start of the main program
 with random numbers using the rand()after that you
 have to start off by adding one at the top of the
 leds(at row=0) and then you have to iterate through
 the led array inside of this iteration you are going
 to look and see if there is an led on at that location,
 however you do that,if there is not one on there then
 you will go through the rest of the array, if there
 is one then you use the random array that you setup
 earlier you go into it at the position that you will
 also have globally saved, and you will take that number
 and mod it with three if that number is a 0 then look
 down and to the left to see if there is an led displaying
 at that location, if not then put it there,otherwise go
 through the next two options, if it is a 1 then do the
 same thing but starting directly below, and if it is a
 2 then do the same thing but starting below and to the
 right the final thing that you do is continue on through
 the rest of the array and do the same thing over and over again*/
}
//simulates pachinko when the box is tilted up
void movePachUp(){
}


//simulates falling sand when the box is tilted down
void moveSandDown(){
// go throw all rows except the bottom because it cannot go down
  for( unsigned char  i = 14; i != 255; i--){
   // go throw all the columns
   for( unsigned char j = 7; j != 255; j--){
```

```
    // if the current LED is lit up
    if(leds[i][j] == 1){
     // if the LED under it is empty, move there
     if(leds[i + 1][j] == 0){
      leds [i+1][j] = 1;
      leds [i][j] = 0;
      rows [i] &= ~(1<<j); //update current row
      rows [i+1]  |=  1<<j; //update row below
     }
     //otherwise check below and to the right
     else if(leds[i + 1][j + 1] == 0 && j < 7){
      leds [i+1][j+ 1] = 1;
      leds [i][j] = 0;
      rows [i] &= ~(1<<j);//update current row
      rows [i+1]  |=  1<< (1 + j);//update row below
     }
     //otherwise check below and to the left
     else if(leds[i + 1][j-1] == 0 && j >0){
      leds [i+1][j-1] = 1;
      leds [i][j] = 0;
      rows [i] &= ~(1<<j);//update current row
      rows [i+1]  |=  1<<(j-1);//update row below
     }
    }
   }
  }
}
//simulates falling sand when the box is tilted up
void moveSandUp(){

 // go throw all rows except
 //the top because it cannot go up anymore
  for( unsigned char  i = 1; i != 16; i++){
  // go throw all the columns
   for( unsigned char j = 7; j != 255; j--){
   // if the current LED is lit up
    // if the LED under it is empty, move there
    if(leds[i][j] == 1){
     if(leds[i - 1][j] == 0){
```

```
      leds [i-1][j] = 1;
      leds [i][j] = 0;
      rows [i] &= ~(1<<j);//update current row
      rows [i-1]  |=  1<<j;//update row below
     }
     //otherwise check below and to the right
     else if(leds[i - 1][j + 1] == 0 && j < 7){
      leds [i-1][j+ 1] = 1;
      leds [i][j] = 0;
      rows [i] &= ~(1<<j);//update current row
      rows [i-1]  |=  1<<(j+1);//update row below
     }
     //otherwise check below and to the left
     else if(leds[i - 1][j-1] == 0 && j >0){
      leds [i-1][j-1] = 1;
      leds [i][j] = 0;
      rows [i] &= ~(1<<j);//update current row
      rows [i-1]  |=  1<<(j-1);//update row below
     }
    }
   }
  }
}
//outputs to the decoders and driver what pins to turn on
void Led_out(void){
 unsigned int tmp = 0;
 //Store output value in tmp
 tmp = GPIOC_ODR;
 //PC10-12 = high, decoder row 0 off
 tmp |= 1<<9;//PC9 On, Decoder 2 off
 tmp &= ~(1<<8);//PC8 Off, Decoder 1 on

 for ( int i = 0; i < 8; i++)
 {

 tmp = GPIOC_ODR;
 tmp &= ~(0x1c00);//Clears decoder bits
 tmp |= i<<10;//Set i into decoder pins
```

```
 tmp &= ~(0xFF );//Clear driver values
 tmp |= rows[i] ;//Force value of driver

  tmp |= 1<<9;//PC9 On, Decoder 2 off
 tmp &= ~(1<<8);//PC8 Off, Decoder 1 on

 //output the result
 GPIOC_ODR = tmp;
 //make a delay
 for (int k = 0; k < 100; k++);
 }

 for (int i = 0; i < 8; i++)
 {
 tmp = GPIOC_ODR;
 tmp &= ~(0x1c00);//Clears decoder bits
 tmp |= i<<10;//Set i into decoder pins

 tmp &= ~(0xFF );//Clear driver values
 tmp |= rows[i+8] ;//Force value of driver


 tmp &= ~(1<<9);//PC9 off, Decoder 2 on
 tmp |= 1<<8;//PC8 on, Decoder 1 off

 //output the result
 GPIOC_ODR = tmp;
 //make a delay
 for (int k = 0; k < 100; k++);

 }
}
```