Mechatronics 2017
Mechanical Engineering Department
Vanderbilt University

Bop It – Final Report

Due Date: 4/24/2017

Jeremy Saslaw

Jeremy Saslaw

Introduction

Mechatronics incorporates mechanical engineering, electrical engineering, and computer science together to create complex systems that can perform a multitude of tasks. More specifically, a mechatronic system can be considered an electromechanical system that uses sensors, a micro-controller, and actuators to determine and model the behavior of a system.

The goal of this project was to design, build, and demonstrate a mechatronic system of any variety with the following primary requirements and design constraints:

- Must be constructed by the student and do something interesting
- Must make use of at least one sensor, one actuator, and a programmable microprocessor (like an Arduino) which runs code as an integral part of the system
- Maximum dimensions of system are 2 feet wide x 2 feet deep x 3 feet high

With the previous requirements in mind, a new-age iteration of the classic ***Bop It*** toy was envisioned.

Bop It is a classic toy and game of the late 90s and early 2000s, where the user has to respond to a command in a timely manner by actuating one of many options, dependent on which was verbally prompted by the system. For example, one iteration of the product was Bop It Extreme 2, which can be seen in Figure 1 below:

Figure 1 - Bop It Extreme 2.[1]

This specific version of Bop It included 5 commands, along with corresponding sensors which allow for the user to perform the tasks:

- "Bop It!" – a depressible button
- "Twist It!" – a twistable lever
- "Pull It!" – a pullable handle
- "Flick It!" – a green zigzag lever
- "Spin It" – a red wheel

Design Overview

The mechatronic new-age Bop It was designed with human factors and usability engineering in mind. The design was intended to be user friendly, simple to use, intuitive, and fun. First and foremost, the physical controller was built to be comfortable to hold, and to contain actuators that were easily identifiable and accessible while holding the controller. The final prototype can be seen below in Figure 2:

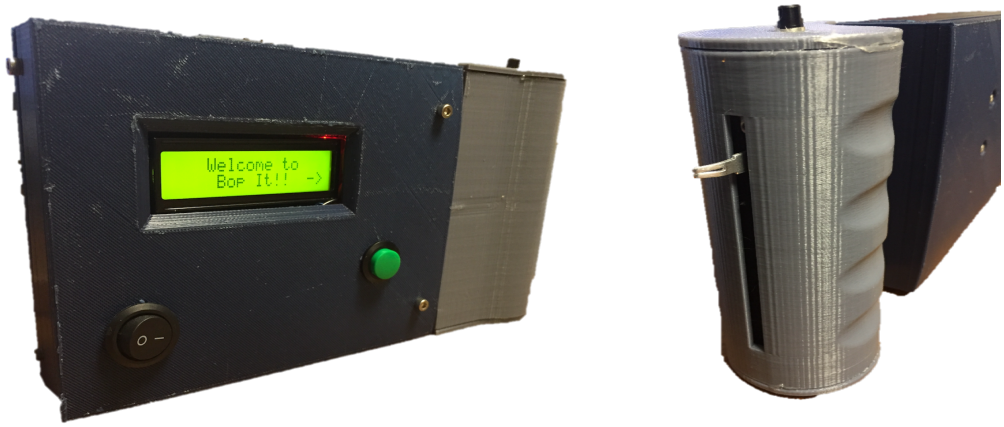

Figure 2 - Final Prototype with two different views.

The new-age Bop It was built with four different commands, each which prompted the user to perform a task that the device was designed to sense using corresponding sensors:

- "Bop It!" – a green button
- "Spin It!" – an encoder
- "Slide It!" – a slide potentiometer
- "Shake It!" – an accelerometer

The device interacts with users via its LCD screen. Once the device is turned on with a simple switch, the LCD screen welcomes the user to the game, and allows for a selection of level. There are three levels, each corresponding to a difficulty ranking as determined by how long the user has to read the prompt and perform the correlated task, as described in Table 1 below:

Table 1: Description of how the levels differentiate in difficulty

Level	Time to Perform Given Task
1	8 seconds
2	4 seconds
3	2 seconds

After starting the game, any of the four tasks are randomly chosen, and the user has the amount of time described in Table 1 to perform said task. If the task is performed in time, then the game provides a new prompt while increasing the score, which can be seen at all times. If the task is not performed in time, the LCD screen lets the user know they lost, while a buzzer starts to buzz and vibrate the controller. At this point the user can restart the game, or turn it off.

Detailed System Description

The overall game was truly a mechatronic project, as it incorporated all three aspects of the subject: mechanical engineering, electrical engineering, and computer science. The various sensors and actuators necessary to operate and control the game were wired up via a breadboard, and hooked up to an Arduino Uno microcontroller. The physical device itself included the construction and design of hardware using 3-D Computer-Aided Design (CAD) and subsequent fabrication involving 3-D printing and more. The code loaded onto this microcontroller allowed for intuitive interaction with the user, and an ability to keep track of certain variables, notably the user's current score.

Sensors

There were five sensors used in the Bop It design: a button, an encoder, a slide potentiometer, an accelerometer, and a switch. The first four sensors correspond to one of the commands given to the user, while the switch allows for powering the device on and off.

The button operates in a relatively simple manner. The physical button can be seen in Figure 3 below, while the circuitry involved in hooking it up can be seen in Figure 4 below:

Figure 3 - Button used for Bop It command.

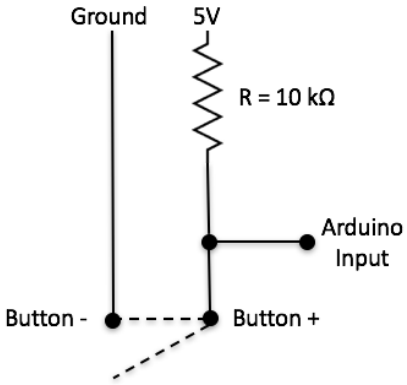

Figure 4 – Button Circuitry.

As seen in Figure 3, the button has two terminals. When the button is not pressed, these two terminals are electrically connected internally. When the button is pressed, an internal connection comes undone and the current flowing between the two terminals goes to 0, thus creating an open circuit. This allows for an easy interaction with the Arduino – the input pin is known to be HIGH when the button is not pressed, while the input pin is known to be LOW when the button is pressed.

An optical encoder is used for the Spin It command, and can be seen in Figure 5 on the following page. The two outer pins are wired into digital input pins 2 and 3 of the Arduino, which are change notification (interrupt) pins, capable of recognizing interrupt requests. The change notification function of the Arduino helps to avoid missing pulses. This function works by watching for a change in a digital input, subsequently telling the microcontroller that a change has occurred, and then automatically started an Interrupt Service Routine (ISR) which is defined in the code.

Figure 5 - Optical Encoder.

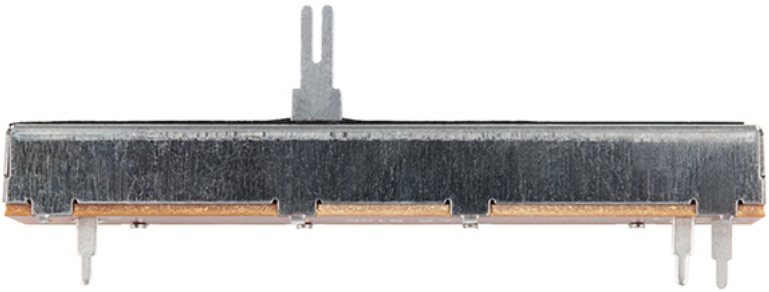

Figure 6 - Slide potentiometer.

The next sensor used is a slide potentiometer, which can be seen above in Figure 6, and acts like a typical potentiometer with 3 pins – one for ground, one for power (5 Volts), and one for the signal. The potentiometer essentially acts like a voltage divider or three-terminal resistor, where the position of the slider corresponds to a voltage ranging from 0 up to the power supply, in this case 5 Volts. This value gets directed to an analog input pin on the Arduino, where the voltage from 0 to 5 corresponds to analog values from 0 to 1023. This allows for mapping the position of the slider in the code, which is described later.

Additionally, in order to track the user's response to the Shake It command, an InvenSense MPU-6050 sensor containing a MEMS accelerometer and a MEMS gyroscope was implemented. This sensor works by using the Arduino Wire library, and connecting the VCC pin to 5 Volts, the GND pin to ground, the SDA pin (data line) to the corresponding data line port on the Arduino (A4), and the SCL pin (clock line) to the corresponding clock line port on the Arduino (A5). The accelerometer aspect of this sensor measures proper acceleration, which is the physical acceleration experienced by an object, or the acceleration relative to free fall. Simply, the accelerometer responds very well to tilting about the x and y axes, as shown on the sensor in Figure 7. The gyroscopic aspect of the sensor was not utilized, as the accelerometer offered enough support in order to verify whether or not the user had shaken the device.

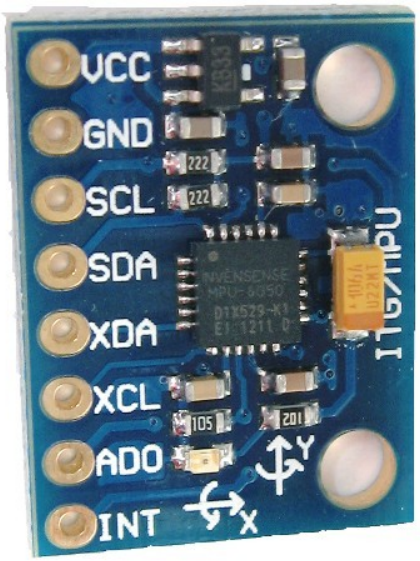

Figure 7 - InvenSense MPU-6050 accelerometer and gyroscope.

The final sensor used is a single-pole, single-throw (SPST) switch, which is used for turning the device on and off. A 9 Volt battery is used to power the device, and when the switch is open the circuit becomes disconnected thus turning the power off, while when the switch is closed the power is turned on, thus turning the device on.

Figure 8 - SPST switch.

Actuators

The first actuator used was an Adafruit 485-772 LCD screen. The LCD screen interacts with the Arduino via the Liquid Crystal library, with the interface pins being initialized with six of the digital input pins. The LCD can be told to output any desired message up to 16 characters long for 2 lines by setting the cursor in the code to the desired spot, and then printing to the screen. The LCD screen can be seen in Figure 2 as a part of the final prototype.

The other actuator used was a simple buzzer. The buzzer has two pins, one for signal and one for ground. When the buzzer is sent a voltage, it buzzes corresponding to how large the voltage is. When the applied voltage is removed, the buzzer remains idle.

Hardware

The hardware used to build Bop It was exclusively designed via 3-D modeling in PTC Creo 2.0, and then subsequently 3-D printed using MakerBot technology. The overall assembly was made of 5 different parts. Figures 9 and 10 show the entire assembly, while Figure 9 also provides reference numbers for each individual part. Figures 11, 12, 13, and 14 then offer detailed views of each part, with the corresponding captions referencing the part number references from Figure 9.

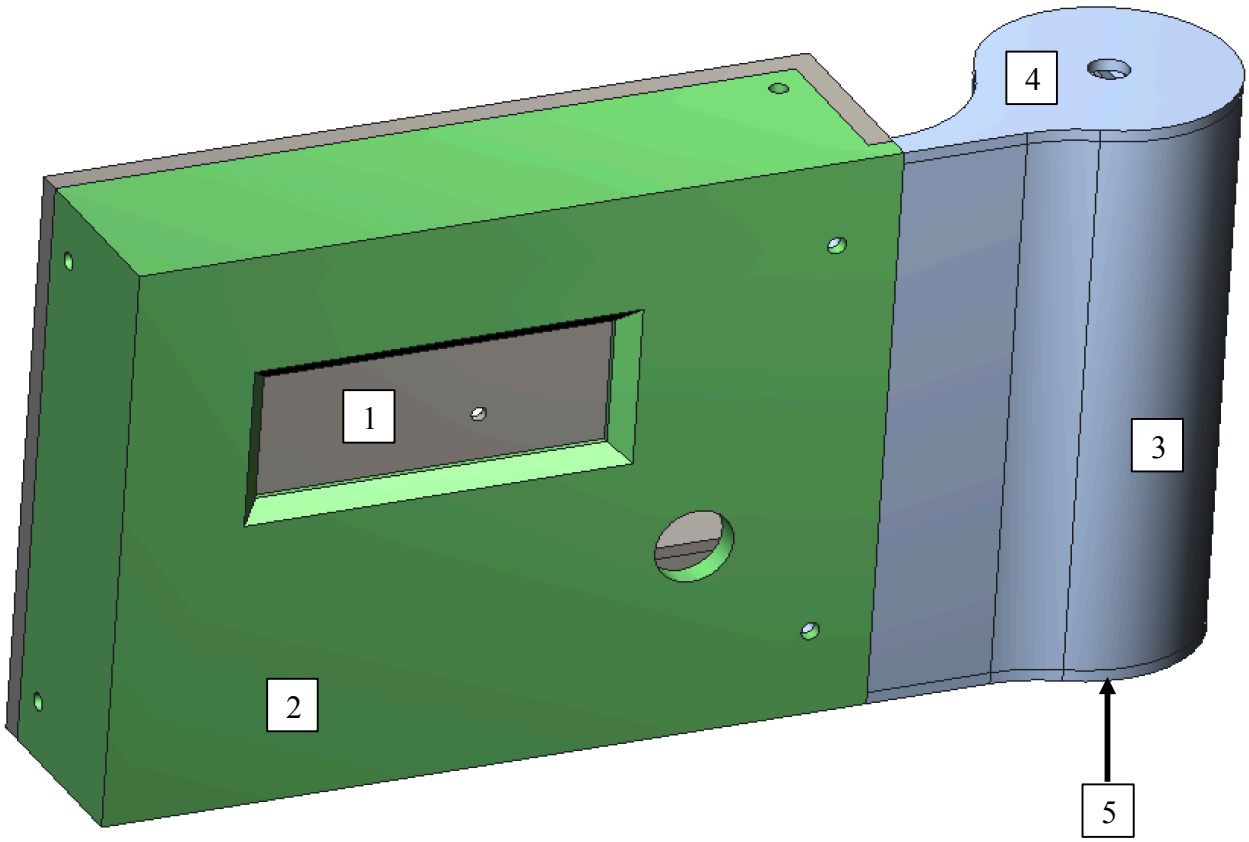

Figure 9 - Front view of final assembly with reference numbers for each part.

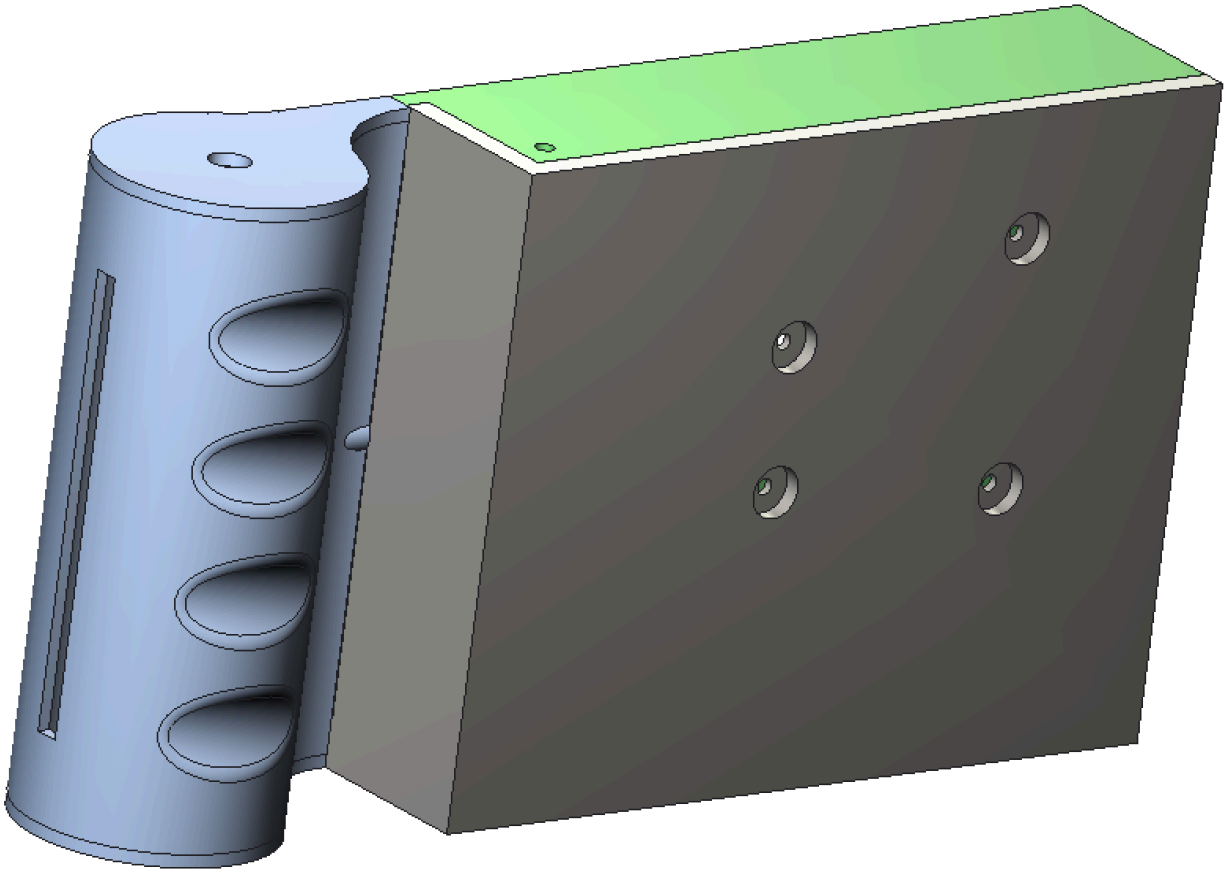

Figure 10 - Back view of final assembly.

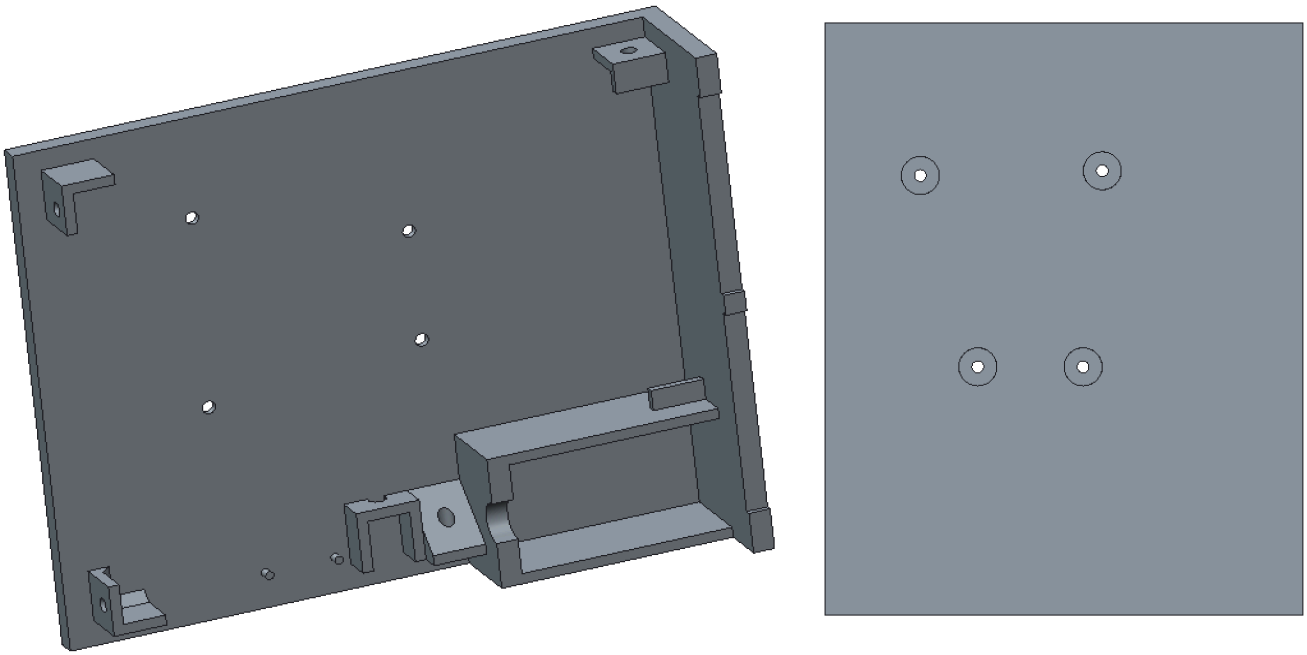

Figure 11 – Part 1 (base), showing the top of the base on the left and bottom of the base on the right.

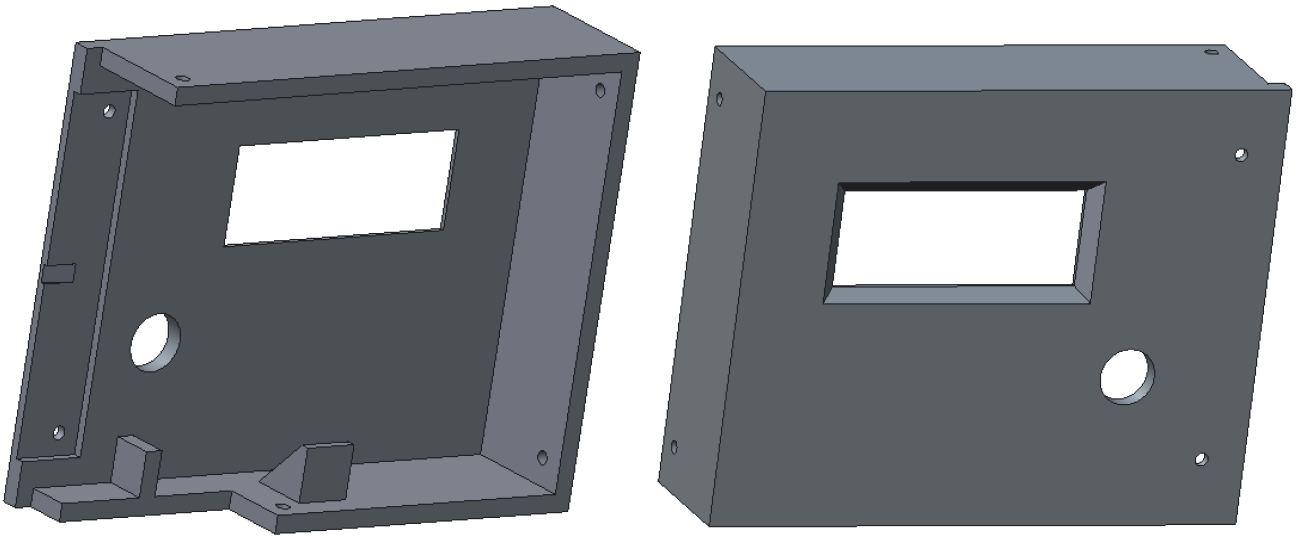

Figure 12 – Part 2 (top), showing the bottom of the part on the left and top of the part on the right.

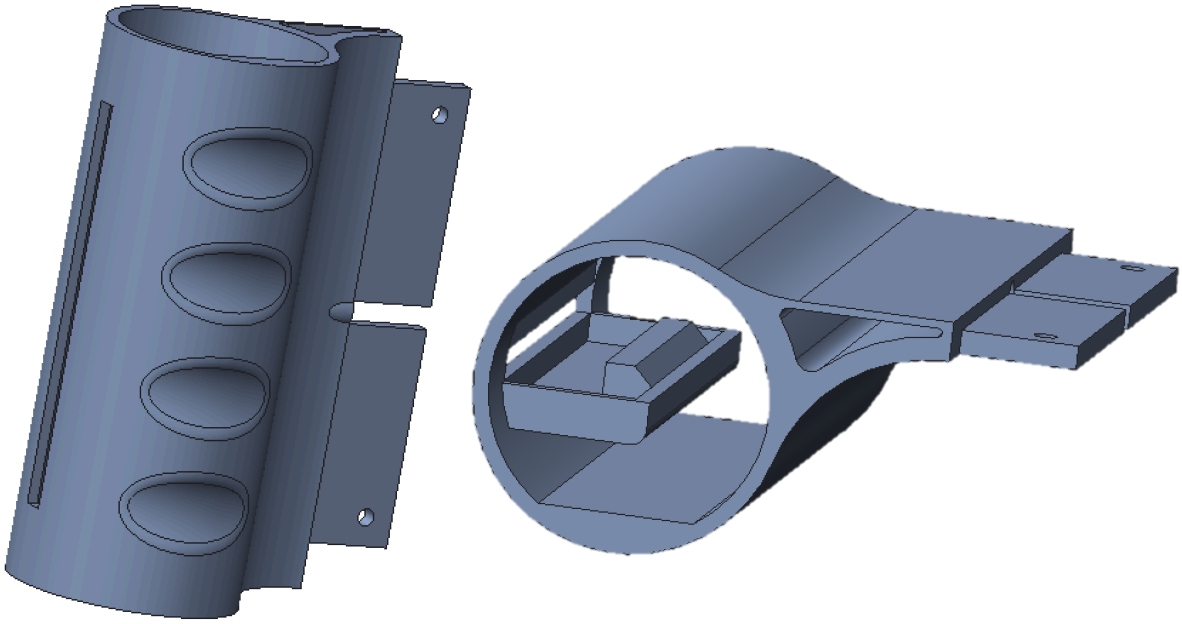

Figure 13 – Part 3 (handle), showing the back of the part on the left and a view into the handle's internal extrusions on the right.

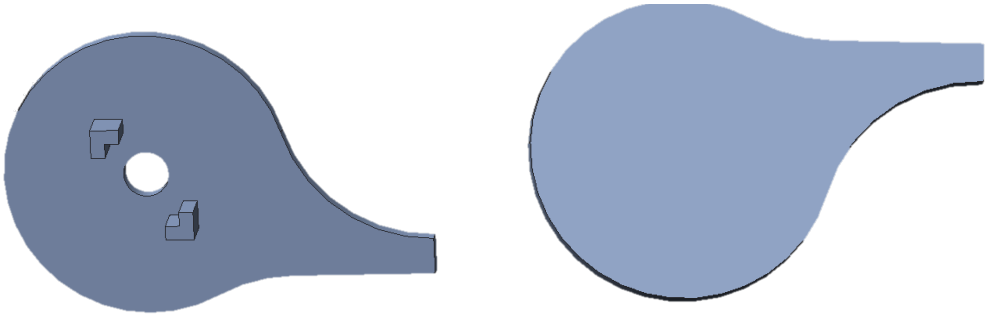

Figure 14 – Part 4 (top cap) on left, Part 5 (bottom cap) on right.

The first step in designing the Bop It "controller" involved determining what orientation and in what manner the user was to hold the game. After deciding on the vertical orientation seen in Figure 9, it was determined to create the controller with two main regions: the rectangular box created by parts 1 and 2, and then the handlebar created by parts 3, 4, and 5. This general setup allowed for the Arduino microcontroller and breadboard to be housed in the box, and then created convenient opportunities to place various external components in the handlebar.

The first consideration made with regards to the internal extrusions involved where to place the Arduino and breadboard. As seen in Figure 11, the base was designed with holes to match those of the Arduino's case, and were countersunk on the back side in order to keep the part flat when placed down. Additionally, there are two ledges in which the breadboard can slide, as seen in Figure 11 as well. By first placing the breadboard into its spot, and then subsequently using bolts to tighten the Arduino in place, both of these components remained stable, and can be seen in place in Figure 15.

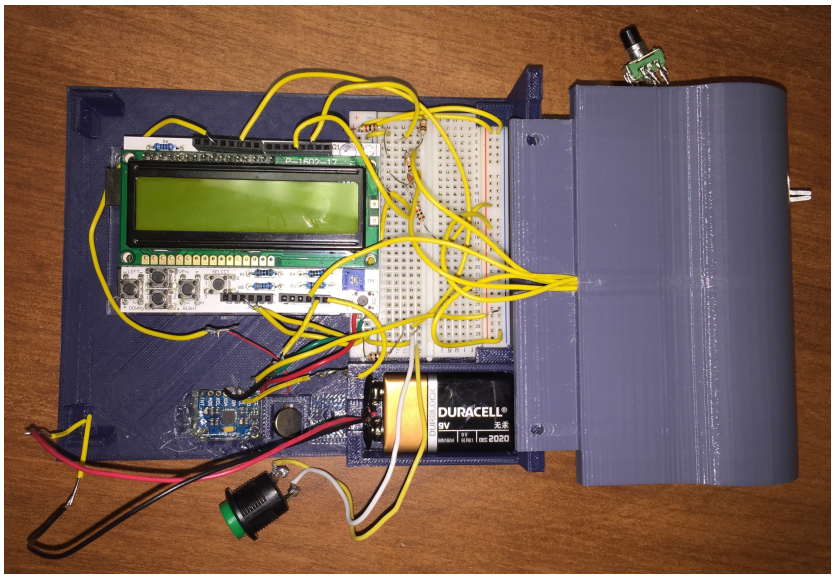

Figure 15 - Inside controller with circuitry implemented.

A battery compartment was additionally implemented in the base as seen in Figure 11, and the component can be seen with a battery in place in Figure 15. An enclosure for the buzzer was also created, with a part from the top ledge (seen in Figure 12) creating a closed box around the buzzer. The final key component of the base are the two pins for the accelerometer to slot onto. This pins act as a guideline for where the accelerometer should say in place, but since movement relative to a stable point is so important to measure shaking, the accelerometer was hot glued into place in order to ensure security.

The top, as seen in Figure 12, offered some key design elements as well, starting primarily with its chamfered hole for the LCD screen to be seen through. A large through hole can be seen diagonally from where the LCD screen lies, and this is where the button was put into place, and secured via nuts on the inside. Finally, several holes for bolts were created to match up with those in the base to close and secure the box. Additionally, a slot for the on and off switch to be placed was forgotten to be placed in the CAD model, so a hole was drilled as seen in Figure 2 of the final prototype.

The handle bar, seen in Figure 13, interfaces with the box via its two holes that come out along the ledge on the right of the figure. There are two key aspects of this part that must be noted, with the first being the finger grips along the exterior surface. Additionally, as seen in the right of

Figure 13, there is a slot for interfacing the sliding potentiometer with the hardware. The pot can be dropped into the hole and locked into place, creating a safe spot for it to not move around during use. An additional hole through the central horizontal plane allows for a connection of wires from the components inside the handlebar with the breadboard.

Figure 14 shows the two caps for the handlebar. External caps were utilized so as to not need to create a 3D printed part which would be a solid extrusion for the whole length. If this were done, the print time would be extremely long and could lead to errors. Additionally, part 4 on the left of Figure 14 has two slots for the encoder to slot up into. These slots, along with hot glue, hold the encoder in place.

Each part was converted into a STL file, which was then sent to the MakerBot 3D printers to fabricate the parts. The 3D printing process can be seen below in Figure 16, where the process of printing the handlebar and the two caps is shown.

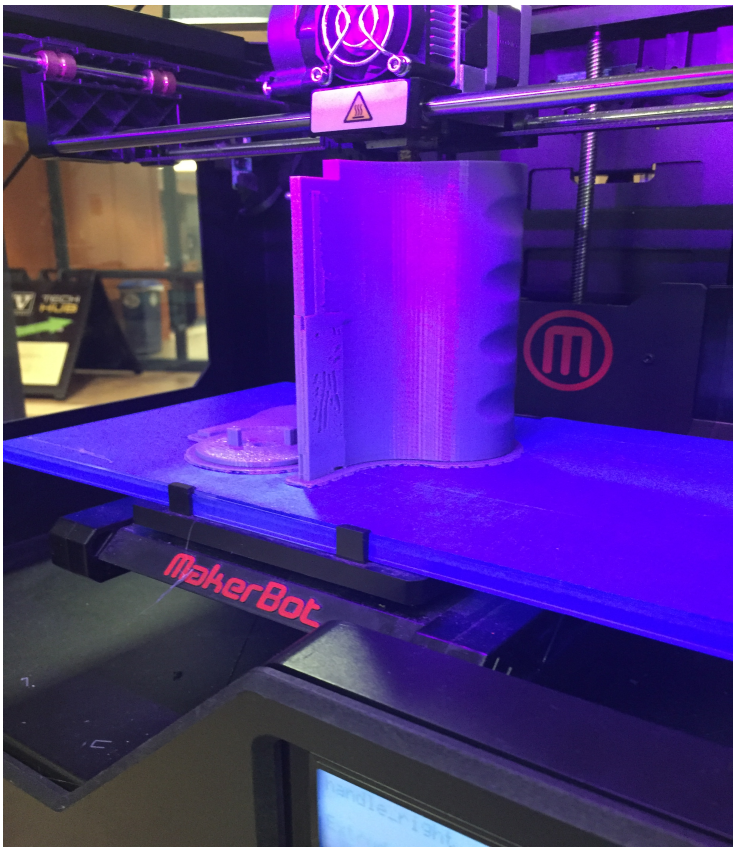

Figure 16 - MakerBot 3D printer printing the handlebar and two caps (parts 3, 4, and 5).

Computer Code

The Arduino code used to run Bop It starts when the power is turned on, by first initializing all pins as digital/analog and input/output based on the desired application, and then attaching interrupts for the two encoder pins. The setup method then prints a welcome message to the LCD screen, which the user can progress through by pressing and then releasing the button. The screen

then prompts the user to select a level, which can be adjusted by spinning the encoder. Interrupt technology is used here, where a change in phase of the encoder interrupts the program and adjusts the variable which tracks encoder position. Once a desired level is selected, the user can progress to starting the game by once again pressing, and then releasing the button.

Before introducing the main loop which constantly runs, there are three additional functions that are used. First is a simple reset function which can be called in the code to reset the device. Additionally, there are two functions (readEncA and readEncB) which are the assigned interrupts for the encoder pins A and B, and function as described previously.

The main loop begins by seeing which level was selected during setup, and applying the given time per move as a variable. The LCD screen then prints the user's score on the 2nd row, and then subsequently a random number generator picks a number from 1 to 4. These numbers correspond to a given command – 1 being Bop It, 2 being Spin It, 3 being Slide It, and 4 being Shake It. An if/else if/else loop is then run, which checks which of the commands has been randomly chosen. If 1 is the random number chosen, then the LCD screen prints "Bop It!" on the first row. A while loop then runs while the time is less than the corresponding level's time, and while the button is not pressed. If the button is pressed in time, the score increases, and a new random command is determined. If the button is not pressed in time, then the LCD screen prints "You lost!!" and the buzzer is turned on, vibrating the controller. The game stays in this state until the button is pressed, which resets the game, or the device is turned off.

Each of the other commands work in similar manners. If the random number of 2 is chosen, the user is instructed to Spin It. If the time runs out, or if the encoder's position variable (checked and modified by interrupts) doesn't surpass 3 or -3 in the corresponding direction, then the user loses. The values of 4 or -4 signify a full click in either the clockwise or counterclockwise direction, and since the user isn't instructed to turn in one specific direction, either response will result in the score increasing and the game continuing.

If the random number of 3 is chosen, then the user is instructed to Slide It. After this, the code first determines if the slider is currently in the top half or bottom half of its allotted movement space. If the slider is in the top half, a variable "upIs0" is set to 0, while if the slider is in the bottom half, the variable is set to 1. At this point, in order to complete the task in the given amount of time based on level, the user must move from whichever half the slider is in, into the other half's top 10% of space. This is accomplished via an analog pin reading the position of the encoder from 0 to 1023. For example, if the slider is ¼ towards the top, the variable keeping track of its position would read something about 255. The code would then sense the Slide It task as accomplished if the slider variable read anything about 923.

Finally, if the random number of 4 is chosen, then the user is instructed to Shake It. At each iteration of the loop, a value is taken for the x, y, and z parts of the accelerometer. It was determined through testing that the x axis of acceleration for the part was the best tracker of whether the user had shaken the device, and thus the Shake It task is considered complete when the user displaces the x value of the accelerometer an absolute value of 8000. This value was determined through trial and error to determine what amount of shaking constitutes an acceptable level to consider the task completed.

Results and Summary

As a whole, the Bop It game works as expected to. All four sensors (button, encoder, slide potentiometer, and accelerometer) work as expected, and accurately sense when the user accomplishes the given task. Additionally, the LCD screen offers a nice and clean way to keep track of score, as well as let the user know what prompt to follow.

After allowing many people to play and test the game, there were a few concepts that could have been improved upon if a further iteration or prototype was implemented. First, the levels seemed a bit easy, so possibly decreasing the amount of time per turn on the given three levels, or adding a fourth more difficult level would be interesting. Additionally, a speaker could be implemented into the game to say the command out loud while also printing it on the LCD. Finally, if this product were to be mass produced, then more comfortable components and hardware could be fabricated, but given the prototype status of this design this would not have been cost effective. All in all, though, the game has been well received and seems to have fulfilled all necessary requirements and design constraints.

References

1 – http://img.dooyoo.co.uk/GB_EN/315/toys-games/toys/hasbro-bop-it-extreme-2.jpg

Appendix A – Components List

Component	Source	Price
3-D Printed hardware	Design Studio	Free
Button	Design Studio	Free
Rotary Encoder	Mechatronics Kit	Free
Slide Potentiometer	Design Studio	Free
Accelerometer	Friend's Old Assignment	Free
SPST Switch	Design Studio	Free
Arduino Uno	Mechatronics Kit	Free
Breadboard	Mechatronics Kit	Free
LCD Screen	Friend's Old Assignment	Free
Buzzer	Design Studio	Free
9V Battery	Mechatronics Kit	Free
Battery Connector	Mechatronics Kit	Free
Wires	Mechatronics Kit	Free
10 kΩ Resistors	Mechatronics Kit	Free
0.047 μF Capacitors	Mechatronics Kit	Free
	Total Cost:	**FREE**

Appendix B – Computer Code

```
#include <LiquidCrystal.h>
#include<Wire.h>


// Initialize the library with the numbers of the interface pins
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

// Initialize inputs
int encApin = 2;
int encBpin = 3;
int buttonpin = 10;
int buzzerpin = 13;
int slidepin = A3;

// Initialize encoder variables
volatile int encCount = 0;

// Initialize button variables
volatile int button = 0; // Not pressed

// Initialize slide variables
volatile int slide = 0;

// Initialize accelerometer variarbles
// SDA is A4 and SCL is A5
const int MPU_addr=0x68;  // I2C address of the MPU-6050
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ,AcXi,AcYi,AcZi;

// Initialize logic variables
int randNumber = 0;
int level = 1;
int score = 0;
int moveTime = 0;

// SETUP
void setup() {

  // Initialize serial communications at 9600 bps
  Serial.begin(9600);

  // Initialize random
  randomSeed(analogRead(A2));

  // LCD SETUP
```

```
// set up the LCD's number of columns and rows:
lcd.begin(16,2);

// ENCODER SETUP
// initialize the encoder terminals as inputs
pinMode(encApin, INPUT);
pinMode(encBpin, INPUT);
// attach interrupts
attachInterrupt(digitalPinToInterrupt(encApin), readEncA, CHANGE);
attachInterrupt(digitalPinToInterrupt(encBpin), readEncB, CHANGE);

// BUTTON SETUP
// initialize the button terminal as input
pinMode(buttonpin, INPUT);

// Slide SETUP
// initialize the slide terminal as input
pinMode(slidepin, INPUT);

// Buzzer SETUP
// initialize the buzzer terminal as input
pinMode(buzzerpin, OUTPUT);
digitalWrite(buzzerpin, LOW);

// Accelerometer SETUP
Wire.begin();
Wire.beginTransmission(MPU_addr);
Wire.write(0x6B);  // PWR_MGMT_1 register
Wire.write(0);     // set to zero (wakes up the MPU-6050)
Wire.endTransmission(true);

// Game Setup
delay(1000);
lcd.setCursor(0,0);
lcd.print("   Welcome to   ");
lcd.setCursor(0,1);
lcd.print("    Bop It!!  ->");
while (digitalRead(buttonpin) == HIGH) {
  ;
}
while (digitalRead(buttonpin) == LOW) {
  ;
}
lcd.setCursor(0,0);
lcd.print("Choose Level By ");
lcd.setCursor(0,1);
```

```
  lcd.print("Spinning:  1  ->");
  delay(500);
  while (digitalRead(buttonpin) == HIGH) {
   if (encCount < -3)  {
     if (level < 3) {
       int newLevel = level + 1;
       level = newLevel;
       lcd.setCursor(0,1);
       lcd.print("Spinning:  ");
       lcd.setCursor(11,1);
       lcd.print(level);
       lcd.setCursor(12,1);
       lcd.print("  ->");
     }
     encCount = 0;
   }
   if (encCount > 3) {
     if (level > 1) {
       int newLevel = level-1;
       level = newLevel;
       lcd.setCursor(0,1);
       lcd.print("Spinning:  ");
       lcd.setCursor(11,1);
       lcd.print(level);
       lcd.setCursor(12,1);
       lcd.print("  ->");
     }
     encCount = 0;
   }
  }
  while (digitalRead(buttonpin) == LOW) {
    ;
  }

}

// RESET function
// Call "resetFunc();" to reset programming
void(*resetFunc) (void) = 0;


// LOOP
void loop() {

  // Determine time per move based on level
  if (level == 1) { moveTime = 8000; }   // level 1 is 8 seconds per move
```

```
if (level == 2) { moveTime = 4000; }   // level 2 is 4 seconds per move
if (level == 3) { moveTime = 2000; }   // level 3 is 2 seconds per move

// Logic code
lcd.clear();
// Display Score
lcd.setCursor(3,1);
lcd.print("Score = ");
lcd.setCursor(11,1);
lcd.print(score);

// Determine random operation (1 = Push button, 2 = Spin Encoder, 3 = Slide, 4 = Accelerometer)
randNumber = random(1,5);

// Push button
if (randNumber == 1) {
  lcd.setCursor(4,0);
  lcd.print("Bop It!");
  int t = 0;
  while (t < moveTime && digitalRead(buttonpin) == HIGH) {
    delay(1);
    int updatedT = t + 1;
    t = updatedT;
  }

  // if button pressed in time
  if (t < moveTime) {
    score++;
    delay(200);
  }

  // if button not pressed in time
  else{
    lcd.setCursor(0,0);
    lcd.print("  You lost!!    ");
    lcd.setCursor(0,1);
    lcd.print("   Score =      ");
    lcd.setCursor(11,1);
    lcd.print(score);
    digitalWrite(buzzerpin, HIGH);

    // Reset game once button pressed
    while (digitalRead(buttonpin) == HIGH) { ; }
    resetFunc();
  }
```

```
}
// Spin encoder
else if (randNumber == 2) {
  lcd.setCursor(4,0);
  lcd.print("Spin It!");
  int t = 0;
  while (t < moveTime) {
    delay(1);
    int updatedT = t + 1;
    t = updatedT;
    if (encCount > 3) {
      encCount = 0;
      break;
    }
    if (encCount < -3) {
      encCount = 0;
      break;
    }
  }

  // if encoder spun in time
  if (t < moveTime) {
    score++;
    delay(200);
  }

  // if encoder not spun in time
  else{
    lcd.setCursor(0,0);
    lcd.print("  You lost!!    ");
    lcd.setCursor(0,1);
    lcd.print("   Score =      ");
    lcd.setCursor(11,1);
    lcd.print(score);
    digitalWrite(buzzerpin, HIGH);

    // Reset game once button pressed
    while (digitalRead(buttonpin) == HIGH) { ; }
    resetFunc();
  }
}


// Slide
else if (randNumber == 3) {
  lcd.setCursor(3,0);
```

```
  lcd.print("Slide It!");
  int t = 0;

  // Determine where slide is currently
  int upIs0;
  if (analogRead(slidepin) > 510) {
    upIs0 = 0; // UP
  }
  else{
    upIs0 = 1; // DOWN
  }

  while (t < moveTime) {
    delay(1);
    int updatedT = t + 1;
    t = updatedT;
    if (upIs0 == 0 && analogRead(slidepin) < 100) {
      break;
    }
    if (upIs0 == 1 && analogRead(slidepin) > 923) {
      break;
    }
  }

  // if slider slid in time
  if (t < moveTime) {
    score++;
    delay(200);
  }

  // if slider not slid in time
  else{
    lcd.setCursor(0,0);
    lcd.print("  You lost!!    ");
    lcd.setCursor(0,1);
    lcd.print("   Score =      ");
    lcd.setCursor(11,1);
    lcd.print(score);
    digitalWrite(buzzerpin, HIGH);

    // Reset game once button pressed
    while (digitalRead(buttonpin) == HIGH) { ; }
    resetFunc();
  }
}
```

```
  // Accelerometer
  else if (randNumber == 4) {
    lcd.setCursor(3,0);
    lcd.print("Shake It!");
    int t = 0;

    // Determine where accelerometer is currently

    Wire.beginTransmission(MPU_addr);
    Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
    Wire.endTransmission(false);
    Wire.requestFrom(MPU_addr,14,true);  // request a total of 14 registers
    AcXi=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)

    while (t < moveTime) {
      delay(1);
      int updatedT = t + 1;
      t = updatedT;
      Wire.beginTransmission(MPU_addr);
      Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
      Wire.endTransmission(false);
      Wire.requestFrom(MPU_addr,14,true);  // request a total of 14 registers
      AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C
(ACCEL_XOUT_L)

      if (abs(AcX-AcXi) > 8000) {
        break;
      }

    }

    // if shaken in time
    if (t < moveTime) {
      score++;
      delay(200);
    }

    // if not shaken in time
    else{
      lcd.setCursor(0,0);
      lcd.print("  You lost!!    ");
      lcd.setCursor(0,1);
      lcd.print("   Score =      ");
      lcd.setCursor(11,1);
```

```
      lcd.print(score);
      digitalWrite(buzzerpin, HIGH);

      // Reset game once button pressed
      while (digitalRead(buttonpin) == HIGH) { ; }
      resetFunc();
    }
  }

  // Button Code
  if (digitalRead(buttonpin) == LOW) {
    button = 1;
  }
  else {
    button = 0;
  }

  // Slide Code
  if (analogRead(slidepin) > 923) {
    slide = 0; // UP
  }
  else if (analogRead(slidepin) < 100) {
    slide = 1; // DOWN
  }

  // Accelerometer Code
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);  // request a total of 14 registers
  AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  Tmp=Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  GyX=Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)

  /*
  Serial.print("AcX = "); Serial.print(AcX);
  Serial.print(" | AcY = "); Serial.print(AcY);
  Serial.print(" | AcZ = "); Serial.print(AcZ);
  Serial.print(" | Tmp = "); Serial.print(Tmp/340.00+36.53);  //equation for temperature in degrees
C from datasheet
  Serial.print(" | GyX = "); Serial.print(GyX);
  Serial.print(" | GyY = "); Serial.print(GyY);
```

```
  Serial.print(" | GyZ = "); Serial.println(GyZ);
  delay(333);
  */

}

// ENCODER INTERRUPT A
void readEncA()  {
  // check for A rising
  if (digitalRead(encApin) == HIGH)  {

    // check for B to see increase or decrease
    if (digitalRead(encBpin) == LOW)  {
      encCount++;
    }
    else  {
      encCount--;
    }
  }

  // A must be falling
  else  {

    // check for B to see increase or decrease
    if (digitalRead(encBpin) == HIGH)  {
      encCount++;
    }
    else  {
      encCount--;
    }
  }
}

// ENCODER INTERRUPT B
void readEncB()  {
 // check for B rising
  if (digitalRead(encBpin) == HIGH)  {

    // check for A to see increase or decrease
    if (digitalRead(encApin) == HIGH)  {
      encCount++;
    }
    else  {
      encCount--;
    }
  }
```

```
  // B must be falling
  else  {

    // check for A to see increase or decrease
    if (digitalRead(encApin) == LOW)  {
      encCount++;
    }
    else  {
      encCount--;
    }
  }
}
```