FINAL PROJECT REPORT

**Simulation and Visualization of Ball Movement & Collision on Stellaris Launchpad**

Prepared by:

Rahul D Prabhu

Sujan Rao

1. **OBJECTIVES**

To use the Launchpad to simulate moving balls and their collisions with obstacles under the effect of the force of gravity and moving the obstacle using the accelerometer (ADXL345).

1. **REQUIREMENTS**

Software Requirements:

* Code Composer Studio V5
* Eclipse IDE (JAVA)

Hardware Requirement:

* Stellaris Launchpad EK-LM4F120XL
* Accelerometer ADXL345

1. **PROJECT DESCRIPTION**

In this project Stellaris Launchpad is used to stimulate and visualize the ball movement and their collision with obstacle under the influence of gravity and moving the obstacle using the accelerometer.

A ball is generated at regular time interval with a velocity initialized to zero under the effect of gravity. The ball position and velocity are computed using the algorithm under various constraints. The ball position and speed generated needs to be communicated to PC to visualize it using RXTXComm(PC side) and UART(microprocessor side). Microprocessor and PC are connected through a USB cable.

In order to visualize the ball movement we need to get the position of the ball in timely basis. Hence a timer along with an interrupt was used. The timer was preloaded with a particular time interval and interrupt was generated after the expiration of that interval. InterruptHandler computes the position of the ball and communicates it to PC using UART.

To move the obstacle an accelerometer was used and using I2C communication x-axis accelerometer value was obtained at the microcontroller which was then passed to the PC. The value obtained from the accelerometer at regular interval was scaled and added with the lines x, y value to move the line (obstacle) in real-time.

Finally a JAVA GUI was used to visualize the ball movement. The positions generated by ball motion were passed to the GUI. There were functions to update these values as soon as they are received through UART.

We were able to determine the maximum number of the balls that the microcontroller could handle and the utilization of the microcontroller CPU.

1. **ALGORITHM:**
2. **Distance and Velocity:**

Velocity v = u + a\*t

Distance s= u\*t + 1/2\*a\*t\*t.

Velocity and distance are split into x and y component.

For X component gravity will be zero. So the below formulas are used.

Vx’= Vx - g\*t. Since g = 0.

Vx’ = Vx.

For Y component gravity needs to be considered.

Vy’ = Vy - g\*t.

Initial velocityof the ball is set to 0.

Vx will only be changing at the time of collision.

Vy is dependent on Gravity. Vy’=Vy-g\*t.

Vx component will be zero when ball is free falling.

After collision, the Vx component and Vy component changes. Then the Ball takes the trajectory path.

1. **Collision Detection.**

For Collision Detection the distance between lines and point are computed using the below formula:

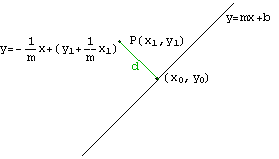


Fig 1: Collision detection between ball and a line

Distance = http://math.ucsd.edu/~wgarner/math4c/derivations/distance/distptline_files/empty.gifhttp://math.ucsd.edu/~wgarner/math4c/derivations/distance/distptline_files/eq0006M.gif

Where, A = slope of line.

B= -1

C = Y-m\*X

Using the above formula we compute distance between ball and line. If distance < radius of ball, new Vx and Vy are computed.

1. **Updating velocities after Collision has been detected:**

Velocities at the time of collision is obtained by using the following formula.

Vx’= E\*cos(2\*α-γ)=E\*cos(α)\*cos(γ)+ sin(α)\*sin(γ).

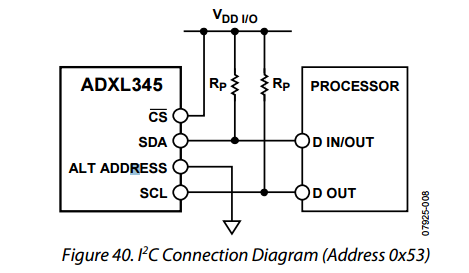
Vy’= E\*sin(2\*α-γ)=E\*sin(2\* α)\*cos(γ)-cos(2\* α)\*sin(γ)

Where E = sqrt(Vx\*Vx+Vy\*Vy)

The algorithm is designed to avoid multiple collision. This is achieved by using flags to detect multiple collision.

1. **Algorithm Flow**

Working of I2C for the ADXL345 accelerometer (Slave device)



ROM\_GPIOPinConfigure(GPIO\_PA6\_I2C1SCL);

ROM\_GPIOPinConfigure(GPIO\_PA7\_I2C1SDA);

The SCL pin for the accelerometer was set to Pin 6 on the GPIO A and SDA to pin 7 and the direction was accordingly set. The chip select was tied to VDD. Pull up resistors for the circuit was recommended for better pull up and pull down voltages. The FIFO was bypassed and the info was serially received.

The registers in the accelerometer were defined as follows

**#define** ADXL345\_POWER\_CTL 0x2d

**#define** ADXL345\_INT\_ENABLE 0x2e

**#define** ADXL345\_INT\_MAP 0x2f

**#define** ADXL345\_INT\_SOURCE 0x30

**#define** ADXL345\_DATA\_FORMAT 0x31

**#define** ADXL345\_DATAX0 0x32

**#define** ADXL345\_DATAX1 0x33

**#define** ADXL345\_DATAY0 0x34

**#define** ADXL345\_DATAY1 0x35

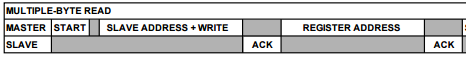
**#define** ADXL345\_DATAZ0 0x36

**#define** ADXL345\_DATAZ1 0x37

**#define** ADXL345\_FIFO\_CTL 0x38

**#define** ADXL345\_FIFO\_STATUS 0x39

The power on for the accelerometer was set at register shown above.





The register address was passed on to the slave device as shown above multiple byte read was employed. The data was received in bytes; the data from the accelerometer received was 6 x 2 bytes each for x, y and z. Since the bytes are received serially the bytes were left shifted to get the values for x y and z respectively as shown below.

i2c\_RcvBuf(I2C\_ID\_ADXL345, ADXL345\_DATAX0, 6, buf);

x = (s16) ((((u16) buf[1]) << 8) | (u16) buf[0]);

\*pxraw = x;

y = (s16) ((((u16) buf[3]) << 8) | (u16) buf[2]);

\*pyraw = y;

z = (s16) ((((u16) buf[5]) << 8) | (u16) buf[4]);

\*pzraw = z;

The data is started with X0 at register number 0x32 and the rest follows the bits are left shifted to get the values each for x y and z. The received values are 8 bits so the int values vary from -256 to +256. To match our computation for the movement of obstacles we had to scale the values from 0 to 500 since the size of our window was roughly 1000 by 1000.

The scaling was done as shown below

**if**(gAXRaw<0)

{ fraction=(**float**)gAXRaw/256;

output=250-**abs**(fraction\*250);

}

**else** **if**(gAXRaw>0)

{ fraction=(**float**)gAXRaw/256;

output=(fraction\*250)+250;

}

Here the condition first check the raw value of the accelerometer then decides what to do with the value

-256 0 +256

0 250 +500

The values are passed on to the main function in the ball collision program and based on these values the obstacle is moved accordingly. Say there are 2 lines line 1 and line 2 with slopes m1 and m2 respectively. At each instance of “time” (where time is defined by the clock period by which the timer function is called in our case it is 4000000 per second) the slope is calculated based on the current value of the position of the line which is given as follows

float m1 = ((Line1\_Y2)-(Line1\_Y1))/((Line1\_X1+output+300)-(Line1\_X1+output));

500+output

500 +output +300

Once the slope is calculated the ball is initiated and the distance of the ball from the line is calculated using the formula

Distance = http://math.ucsd.edu/~wgarner/math4c/derivations/distance/distptline_files/empty.gifhttp://math.ucsd.edu/~wgarner/math4c/derivations/distance/distptline_files/eq0006M.gif

Where, A = slope of line.

The point being x0 y0 and the line being Ax+By+C=0.In our case since we are using the slope equation we are using y=mx+c clearly both equations are same

Comparing both equations

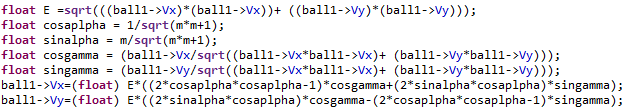
clearly, B = -1; A = m; and C = Y1 - m\*(X1+output) since we know Y1 and X1+ output lies on the line now calculating the distance using the above formula we have,

Distance = (fabs(A\*X+B\*Y+C))/(sqrt(A\*A+B\*B));

Setting the equations for motion with gravity we have



If the ball is with the distance lesser than the radius of the ball we have to initiate the acceleration accordingly using the angle if incidence as alpha and the angle of reflection as gamma we have the following equations



The same is done for line 2 as well and if the ball goes out of reach of the window it disappears.

In this timer a CPU tick count is checked every time the computation is done it is called by the ROM function as shown below.

ui32Value = MAP\_TimerValueGet(g\_pui32CPUUsageTimerBase[1],TIMER\_A);

This value is compared to the previous value of the tick which was initialized to 0xFFFFFFFF based on the difference we can tell how much percent of the CPU the timer was used for computation

ui32Usage=((((g\_ui32CPUUsagePrevious - ui32Value)\*6400)/4000000)\* 1024);

40000000 is the ratio of the clock rate to the rate at which the function was called.

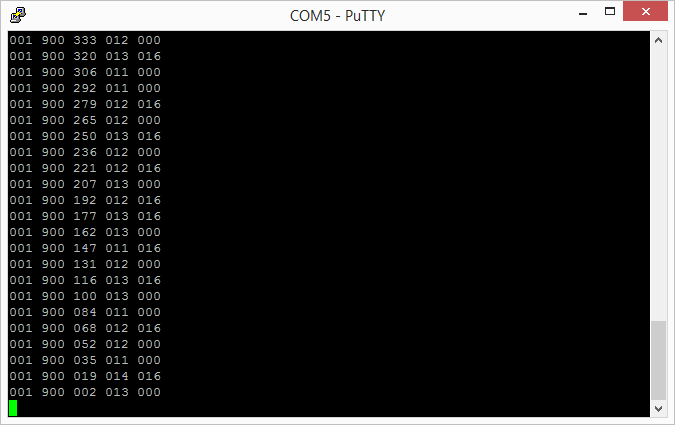
The previous value was then stored in memory

g\_ui32CPUUsagePrevious = ui32Value;

This 32 bit unsigned long int value was stored in the memory in a 16.16 fixed point format and was then converted to readable form either by dividing by 2^32 or right shifting 16 times.

percent= ui32Usage >> 16;

All these values were sent through the UART one line each with the index of the ball Xposition, Yposition, accelerometer output, cpu\_percent.



The values were then read line by line using a buffered reader in Java and separating each by a “ “ space

in1 = **new** BufferedReader(**new** InputStreamReader(inputStream));

**while** (inputStream.available() > -1) {

String line = in1.readLine();

String[] tokens = line.split(" ");

*name* = Integer.*parseInt*(tokens[0]);

*X* = Integer.*parseInt*(tokens[1])\*3/4;

*Y* = 750-Integer.*parseInt*(tokens[2])\*3/4 **int** accx1=Integer.*parseInt*(tokens[3]);

**int** perc=Integer.*parseInt*(tokens[4]);

graphicvisual.mov(*name*, *X*, *Y*,accx1);

**if**(perc!=0)

{

System.***out***.println("cpuusage:"+perc+"%");

} }

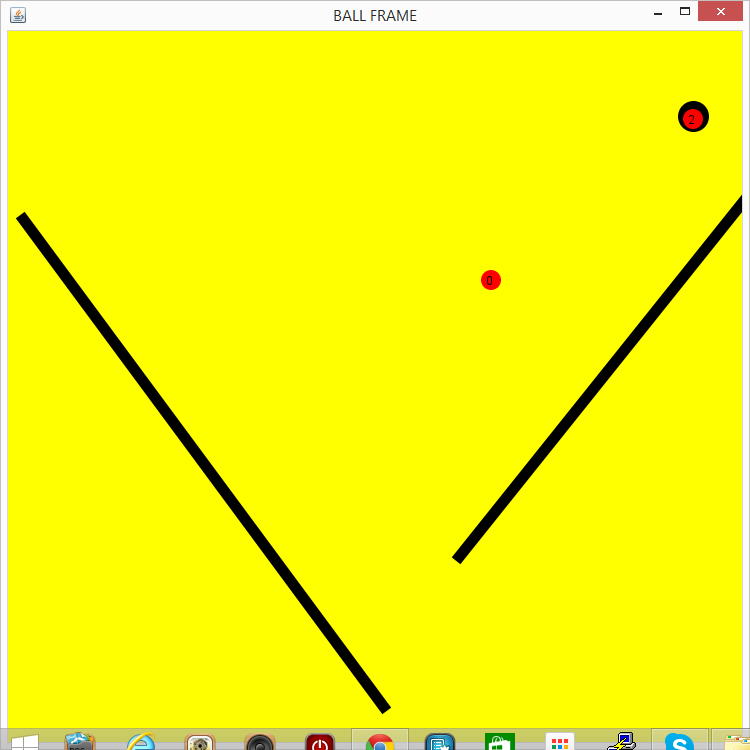
The values were then stored in an array and were used to move the GUI.

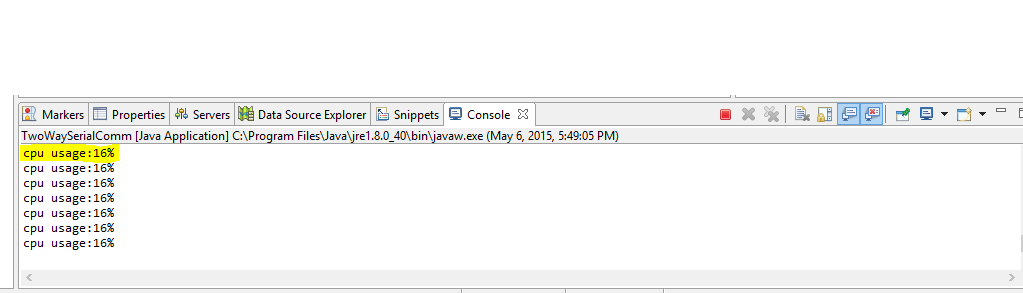
1. **RESULTS**

A total of 15 ball was generated. The ball movement slowed down as the number of ball increased because of the increased computation speed, more memory utilization and more values being transmitted through serial port.

Also, we were able to move the obstacle using the accelerometer using I2C communication between the accelerometer and the microcontroller board.

The GUI for the project



The CPU percentage was printed in the console 

1. **CONCLUSION**

The further improvement for this project would be implementing a ball to ball collision and to movie the obstacle in any direction. This project helped us learn implementation of serial communication and use of board to calculate values of computationally complex equations.