INTERFACING A ACCELEROMETER WITH A MICROCONTROLLER

Name – Yashu Garg

Enroll no. – 101236

Supervisor – Prof. Vivek Sehgal

May, 2014

Submitted
In
Partial Fulfillment of Degree of Bachelor of Technology
DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

AND

INFORMATION TECHNOLOGY

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,

WAKNAGHAT.

CERTIFICATE

This is to certify that the work entitled **Interfacing an Accelerometer with a Microcontroller** submitted by **Yashu Garg(101236)** in partial fulfillment for award for degree of Bachelor of Technology in Information Technology of JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY has been carried out under my supervision. This work has not been submitted partially or wholly to any other University for any award of this or any other degree.

Prof. Vivek Sehgal
(Associate Professor)
Department of Computer Science Engineering and Information Technology
Jaypee University of Information Technology
Waknaghat

Acknowledgement

"It is not possible to prepare a project without the assistance & Encouragement of other people. This one is certainly no exception."

On the very outset of this report, we would like to extend our sincere & heartfelt obligation towards all the personages who have helped us in this endeavor. Without their active guidance, help, cooperation & encouragement, we would not have made headway in the project.

We would like to show our greatest appreciation to **Prof. Vivek Sehgal.** We feel motivated every time we get his encouragement. For his coherent guidance throughout the tenure of the project, we feel fortunate to be taught by **Prof. Vivek Sehgal**, who gave us his unwavering support. Besides being our mentor, he taught us that there is no substitute for hard work.

We will be always in debt of **Prof. Punit Gupta** for providing us his timely help and guidance.

We owe our heartiest thanks to **Brig. (Retd.) S.P. Ghrera** (HOD, CES/IT Department) who has always inspired us to take initiatives and showed us the path for achieving our goal.

In the light of new developments and recent findings, we devote the task that was asked from us at Jaypee University of Information Technology to **"INTERFACING A ACCELEROMETER WITH A MICROCONTROLLER"**.

Yashu Garg(101236)

Table of Contents

`

CHAPTER 1 - INTRODUCTION

What is a MICROCONTROLLER?

A **microcontroller** is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. Microcontrollers are designed for embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications.

Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, toys and other embedded systems.

Looking Inside the MICROCONTROLLER

- Read Only Memory (ROM)

Read Only Memory (ROM) is a type of memory used to permanently save the program being executed. The size of the program that can be written depends on the size of this memory. ROM can be built in the microcontroller or added as an external chip, which depends on the type of the microcontroller. Both options have some disadvantages. If ROM is added as an external chip, the microcontroller is cheaper and the program can be considerably longer. At the same time, a number of available pins is reduced as the microcontroller uses its own input/output ports for connection to the chip. The internal ROM is usually smaller and more expensive, but leaves more pins available for connecting to peripheral environment. The size of ROM ranges from 512B to 64KB.

- Random Access Memory (RAM)

Random Access Memory (RAM) is a type of memory used for temporary storing data and intermediate results created and used during the operation of the microcontrollers. The content of this memory is cleared once the power supply is off. For example, if the program performs an addition, it is necessary to have a register standing for what in everyday life is called the “sum”. For that purpose, one of the registers in RAM is called the "sum" and used for storing results of addition. The size of RAM goes up to a few KBs.

- Electrically Erasable Programmable ROM (EEPROM)

The EEPROM is a special type of memory not contained in all microcontrollers. Its contents may be changed during program execution (similar to RAM), but remains permanently saved even after

the loss of power (similar to ROM). It is often used to store values, created and used during operation (such as calibration values, codes, values to count up to etc.), which must be saved after turning the power supply off. A disadvantage of this memory is that the process of programming is relatively slow. It is measured in milliseconds.

- Special Function Registers (SFR)

Special function registers are part of RAM memory. Their purpose is predefined by the manufacturer and cannot be changed therefore. Since their bits are physically connected to particular circuits within the microcontroller, such as A/D converter, serial communication module etc., any change of their state directly affects the operation of the microcontroller or some of the circuits. For example, writing zero or one to the SFR controlling an input/output port causes the appropriate port pin to be configured as input or output. In other words, each bit of this register controls the function of one single pin.

- Program Counter

Program Counter is an engine running the program and points to the memory address containing the next instruction to execute. After each instruction execution, the value of the counter is incremented by 1. For this reason, the program executes only one instruction at a time just as it is written. However…the value of the program counter can be changed at any moment, which causes a "jump" to a new memory location. This is how subroutines and branch instructions are executed. After jumping, the counter resumes even and monotonous automatic counting +1, +1, +1…

- Central Processor Unit (CPU)

As its name suggests, this is a unit which monitors and controls all processes within the microcontroller and the user cannot affect its work. It consists of several smaller subunits, of which the most important are:

- **Instruction decoder** is a part of the electronics which recognizes program instructions and runs other circuits on the basis of that. The abilities of this circuit are expressed in the "instruction set" which is different for each microcontroller family.
- **Arithmetical Logical Unit (ALU)** performs all mathematical and logical operations upon data.
- **Accumulator** is an SFR closely related to the operation of ALU. It is a kind of working desk used for storing all data upon which some operations should be executed (addition, shift etc.). It also stores the results ready for use in further processing. One of the SFRs, called the Status Register, is closely related to the accumulator, showing at any given time the "status" of a number stored in the accumulator (the number is greater or less than zero etc.).

- Oscillator

Even pulses generated by the oscillator enable harmonic and synchronous operation of all circuits within the microcontroller. It is usually configured as to use quartz-crystal or ceramics resonator for frequency stabilization. It can also operate without elements for frequency stabilization (like RC oscillator). It is important to say that program instructions are not executed at the rate imposed by the oscillator itself, but several times slower. It happens because each instruction is executed in several steps. For some microcontrollers, the same number of cycles is needed to execute any instruction, while it's different for other microcontrollers. Accordingly, if the system uses quartz crystal with a frequency of 20MHz, the execution time of an instruction is not expected 50nS, but 200, 400 or even 800 nS, depending on the type of the microcontroller!

- Timers/Counters

Most programs use these miniature electronic "stopwatches" in their operation. These are commonly 8- or 16-bit SFRs the contents of which is automatically incremented by each coming pulse. Once the register is completely loaded, an interrupt is generated!

If these registers use an internal quartz oscillator as a clock source, then it is possible to measure the time between two events. If the registers use pulses coming from external source, then such a timer is turned into a counter.

- Watchdog timer

The Watchdog Timer is a timer connected to a completely separate RC oscillator within the microcontroller.

If the watchdog timer is enabled, every time it counts up to the program end, the microcontroller reset occurs and program execution starts from the first instruction. The point is to prevent this from happening by using a special command. The whole idea is based on the fact that every program is executed in several longer or shorter loops.

If instructions resetting the watchdog timer are set at the appropriate program locations, besides commands being regularly executed, then the operation of the watchdog timer will not affect the program execution.

If for any reason (usually electrical noise in industry), the program counter "gets stuck" at some memory location from which there is no return, the watchdog will not be cleared, so the register's value being constantly incremented will reach the maximum et voila! Reset occurs!

- **Interrupt** - electronics is usually faster than physical processes it should keep under control. This is why the microcontroller spends most of its time waiting for something to happen or execute. In other words, when some event takes place, the microcontroller does something. In order to prevent the microcontroller from spending most of its time endlessly checking for logic state on input pins and registers, an interrupt is generated. It is the signal which informs the central processor that something attention worthy has happened. As its name suggests, it interrupts regular program execution. It can be generated by different sources so when it occurs, the microcontroller immediately stops operation and checks for the cause. If it is needed to perform some operations, a current state of the program counter is pushed onto the Stack and the appropriate program is executed. It's the so called interrupt routine.

- **Stack** is a part of RAM used for storing the current state of the program counter (address) when an interrupt occurs. In this way, after a subroutine or an interrupt execution, the microcontroller knows from where to continue regular program execution. This address is cleared after returning to the program because there is no need to save it any longer, and one location of the stack is automatically available for further use. In addition, the stack can consist of several levels. This enables subroutines' nesting, i.e. calling one subroutine from another.

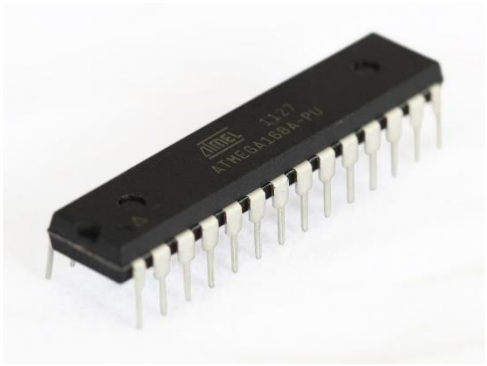
There most commonly used Microcontroller in the world today

ATMEL AVR

The **AVR** is a modified Harvard architecture 8-bit RISC single chip microcontroller which was developed by Atmel in 1996. The AVR was one of the first microcontroller families to use on-chip flash memory for program storage, as opposed to one-time programmable ROM, EPROM, or EEPROM used by other microcontrollers at the time.

Basic families

AVRs are generally classified into following:



- **TinyAVR** — the ATtiny series
  - 0.5–16 kB program memory
  - 6–32-pin package
  - Limited peripheral set
- **MegaAVR** — the ATmega series
  - 4–512 kB program memory
  - 28–100-pin package
  - Extended instruction set (multiply instructions and instructions for handling larger program memories)
  - Extensive peripheral set
- **XMEGA** — the ATxmega series
  - 16–384 kB program memory
  - 44–64–100-pin package (A4, A3, A1)
  - Extended performance features, such as DMA, "Event System", and cryptography support.
  - Extensive peripheral set with ADCs

Microchip PIC

PIC is a family of modified Harvard architecture microcontrollers made by Microchip Technology, derived from the PIC1650,originally developed by General Instrument's Microelectronics Division. The name PIC initially referred to "**Peripheral Interface Controller**" now it is "**PIC**" only.

PICs are popular with both industrial developers and hobbyists alike due to their low cost, wide availability, large user base, extensive collection of application notes, availability of low cost or free development tools, and serial programming (and re-programming with flash memory) capability.

The PIC architecture is characterized by its multiple attributes:

- Separate code and data spaces (Harvard architecture).
- A small number of fixed length instructions
- Most instructions are single cycle execution (2 clock cycles, or 4 clock cycles in 8-bit models), with one delay cycle on branches and skips
- One accumulator (W0), the use of which (as source operand) is implied (i.e. is not encoded in the opcode)
- All RAM locations function as registers as both source and/or destination of math and other functions.[6]
- A hardware stack for storing return addresses
- A small amount of addressable data space (32, 128, or 256 bytes, depending on the family), extended through banking
- Data space mapped CPU, port, and peripheral registers

.

Philips LPC

LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors (formerly Philips Semiconductors). The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals. The legacy LPC families were based on the 8-bit 80C51 core.[2] As of February 2011, NXP had shipped over one billion ARM processor-based chips.

Motorola's Free scale 68HC11

The **68HC11** (**6811** or **HC11** for short) is an 8-bit microcontroller (µC) family introduced by Motorola in 1985. Now produced by Free scale Semiconductor, it descended from the Motorola 6800 microprocessor. It is a CISC microcontroller. The 68HC11 devices are more powerful and more expensive than the 68HC08microcontrollers, and are used in barcode readers, hotel card key writers, amateur robotics, and various other embedded systems. The MC68HC11A8 was the first MCU to include CMOS EEPROM.

Difference between Microcontrollers & Microprocessors

Microprocessor	Microcontroller
It is just a processor. Memory and I/O components have to be connected externally	Micro controller has external processor along with internal memory and i/O components
Since memory and I/O has to be connected externally, the circuit becomes large.	Since memory and I/O are present internally, the circuit is small.
Cannot be used in compact systems and hence inefficient	Can be used in compact systems and hence it is an efficient technique
Cost of the entire system increases	Cost of the entire system is low
Due to external components, the entire power consumption is high. Hence it is not suitable to used with devices running on stored power like batteries.	Since external components are low, total power consumption is less and can be used with devices running on stored power like batteries.
Most of the microprocessors do not have power saving features.	Most of the micro controllers have power saving modes like idle mode and power saving mode. This helps to reduce power consumption even further.
Since memory and I/O components are all external, each instruction will need external operation, hence it is relatively slower.	Since components are internal, most of the operations are internal instruction, hence speed is fast.
Microprocessor have less number of registers, hence more operations are memory based.	Micro controller have more number of registers, hence the programs are easier to write.
Microprocessors are based on von Neumann model/architecture where program and data are stored in same memory module	Micro controllers are based on Harvard architecture where program memory and Data memory are separate
Mainly used in personal computers	Used mainly in washing machine, MP3 players

Microcontroller Used

The Microcontroller used as a part of this project is **ATMEL's AtMega 328P.**

ATMEGA 328P MICROCONTROLLER

ATMEGA 328P

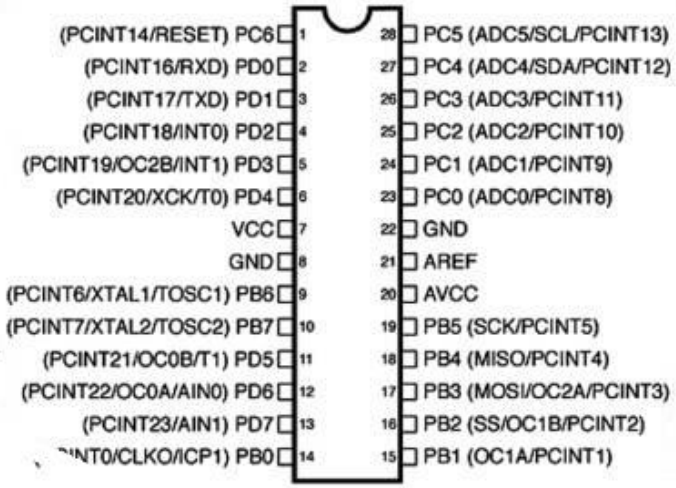
The **ATmega328P** is a single chip micro-controller created by Atmel and belongs to the mega series.
The high-performance Atmel 8-bit AVR RISC-based microcontroller combines

- Flash memory - 32 KB ISP
- EEPROM - 1 KB
- SRAM - 2 KB
- 23 general purpose I/O lines,
- 32 general purpose working registers
- 3 flexible timer/counters with compare modes, internal and external interrupts,
- Serial Programmable USART
- A byte-oriented 2-wire serial interface
- SPI serial port
- 6-channel 10-bit A/D converter
- Internal oscillator

- Software selectable power saving modes.

The device operates between 1.8-5.5 volts. By executing powerful instructions in a single clock cycle, the device achieves throughputs approaching 1 MIPS per MHz, balancing power consumption and processing speed.

ATmega168/328 Pin Mapping



What is an Accelerometer?

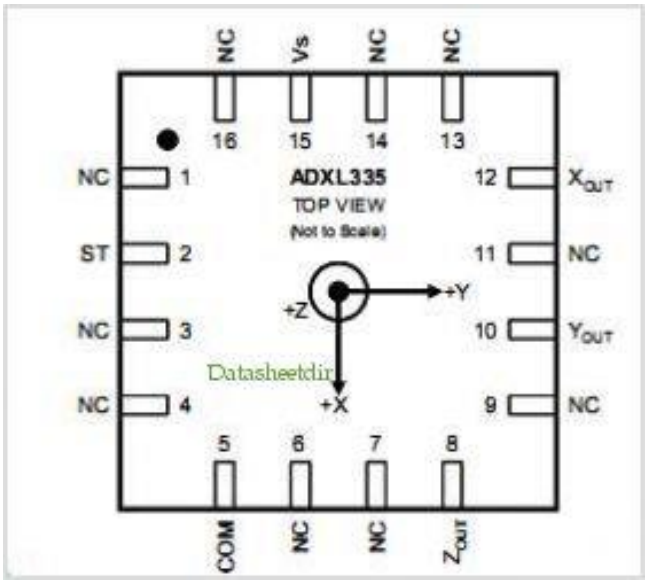
An **accelerometer** is a device that measures acceleration. The acceleration measured by an accelerometer is not necessarily the coordinate acceleration (rate of change of velocity).
The accelerometer is a built-in electronic component that measures tilt and motion. It is also capable of detecting rotation and motion gestures such as swinging or shaking.
The most common use for it is to activate auto screen rotation on mobile devices when the user changes their orientation from portrait to landscape or vice-versa.
Another modern application for the accelerometer is to control the mobile device music player with gestures (Sony Ericsson Shake control or Samsung Motion play technologies).
Accelerometers are also utilized for enriching the gaming controls (navigating by tilting the device instead of by pressing keys).
Another popular mobile phone feature based on an accelerometer is turn-to-mute. It allows user to mute an incoming call, silence an alarm or pause the mobile music player simply by turning the device face down.

Accelerometer Used

The Accelerometer used as a part of this project is **ADXL 335.**

ADXL 335

The ADXL335 is a small, thin, low power, complete 3-axis accelerometer with signal conditioned voltage outputs. The product measures acceleration with a minimum full-scale range of ±3 *g*. It can measure the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion, shock, or vibration.



PIN DIAGRAM – ADXL335

The user selects the bandwidth of the accelerometer using the CX, CY, and CZ capacitors at the XOUT, YOUT, and ZOUT pins. There are few bandwidths that can be selected to suit the task needed. They range from 0.5 Hz to 1600 Hz for the X and Y axes and from 0.5 Hz to 550 Hz for the Z axis.

The ADXL335 contains a Polysilicon surface-micro machined structure built on top of a silicon wafer. Polysilicon springs suspend the structure over the surface of the wafer and provide a resistance against acceleration forces. A differential capacitor, consisting of independent fixed plates and plates attached to the moving mass, measures the deflection of the structure. Acceleration unbalances the capacitor, which in turn, results in a sensor output with amplitude proportional to the acceleration experienced.

.

Mechanical Sensor

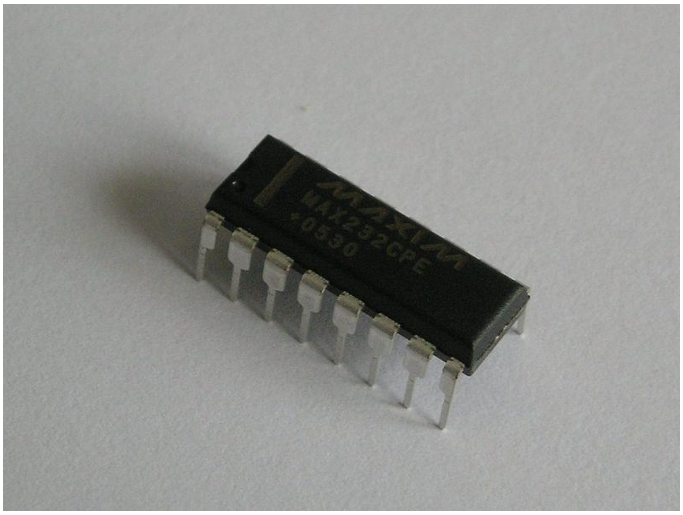
The ADXL335 uses a single structure for sensing the three axes. Therefore, the axes' sense direction is orthogonal and has little cross-axis sensitivity. Mechanical misalignment is the principal source of cross-axis sensitivity. However, this misalignment can be calibrated out at the system level.

·Performance

Innovative design techniques make sure that high performance is achieved by the ADXL335. For this reason, there is no quantization error or no monotonic behavior, and temperature hysteresis is very low (usually less than 3mg over the -25℃ to +70℃ temperature range).

MAX232

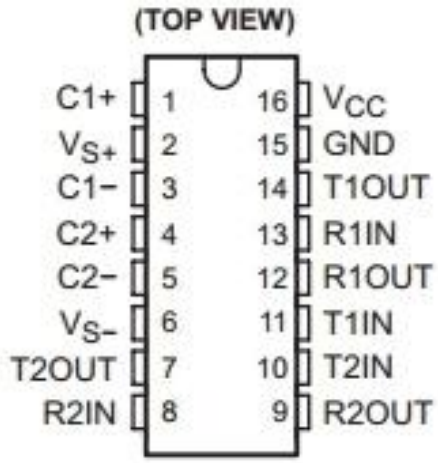
The **MAX232** is an IC, that converts signals from an RS-232 serial port to signals suitable for use in TTL compatible digital logic circuits. The MAX232 is a dual driver/receiver and typically converts the RX, TX, CTS and RTS signals.



The drivers provide RS-232 voltage level outputs (approx. ± 7.5 V) from a single + 5 V supply via on-chip charge pumps and external capacitors. This makes it useful for implementing RS-232 in

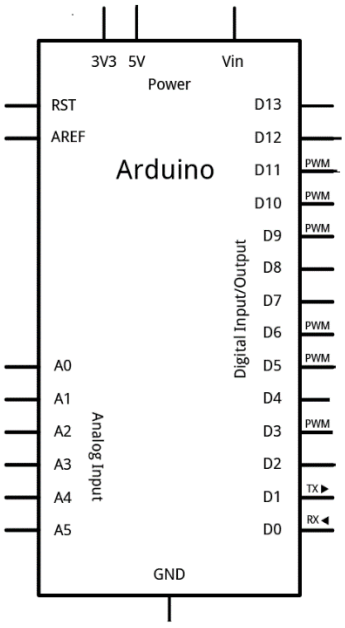
devices that otherwise do not need any voltages outside the 0 V to + 5 V range, as power supply design does not need to be made more complicated just for driving the RS-232 in this case.

The receivers reduce RS-232 inputs (which may be as high as ± 25 V), to standard 5 V TTL levels. These receivers have a typical threshold of 1.3 V, and a typical hysteresis of 0.5 V.



Programmer

The programmer used is **ARDUINO UNO BOARD.** The Arduino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.



Pin diagram Arduino Uno

CHAPTER 2 –PREVIOUS STUDY

As this project was altogether new and different, it needed the study of basics of Hardware, circuits. Thus I had to go through the basic libraries for coding in Arduino, Implemented the basic circuits & programs and then went through with the Project .

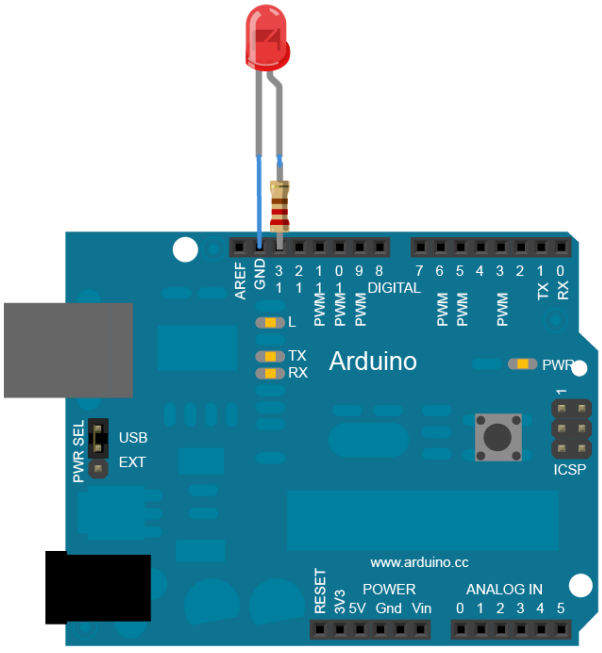
The basic programs implemented were

Blinking an LED

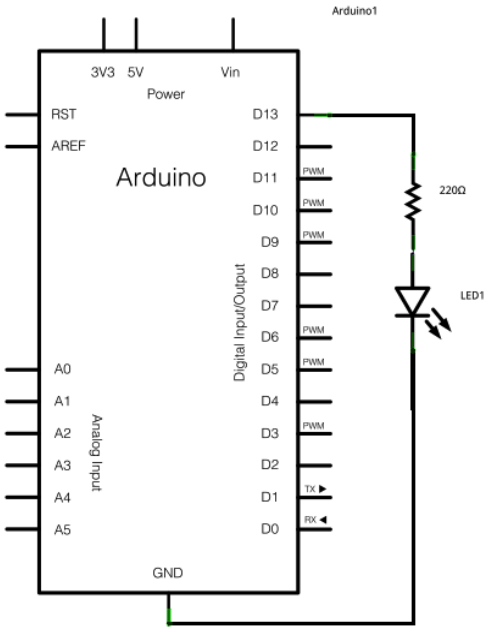
Hardware Required

- Arduino Uno Board
- LED'S
- AtMega 328P

Circuit



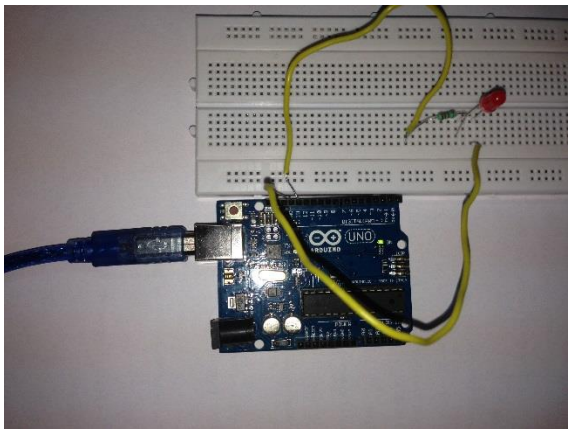
Schematic



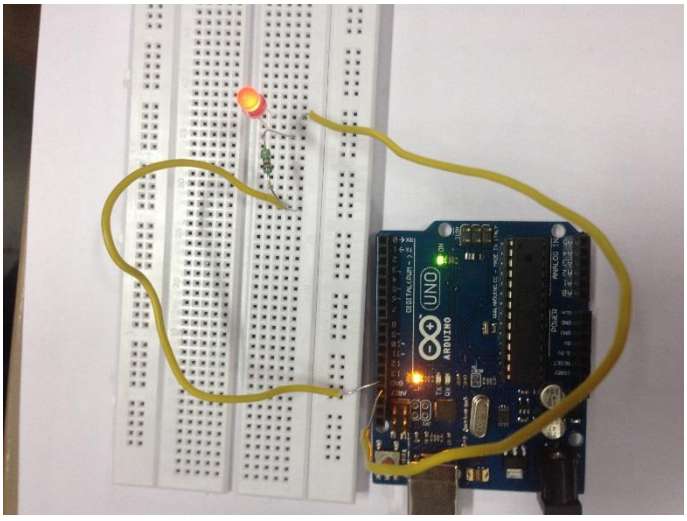
Code

```
 #include <Blink.h>
int led = 13;
void setup()
{
   pinMode(led, OUTPUT);
}
void loop()
{
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

Results



ORIGNAL CIRCUIT



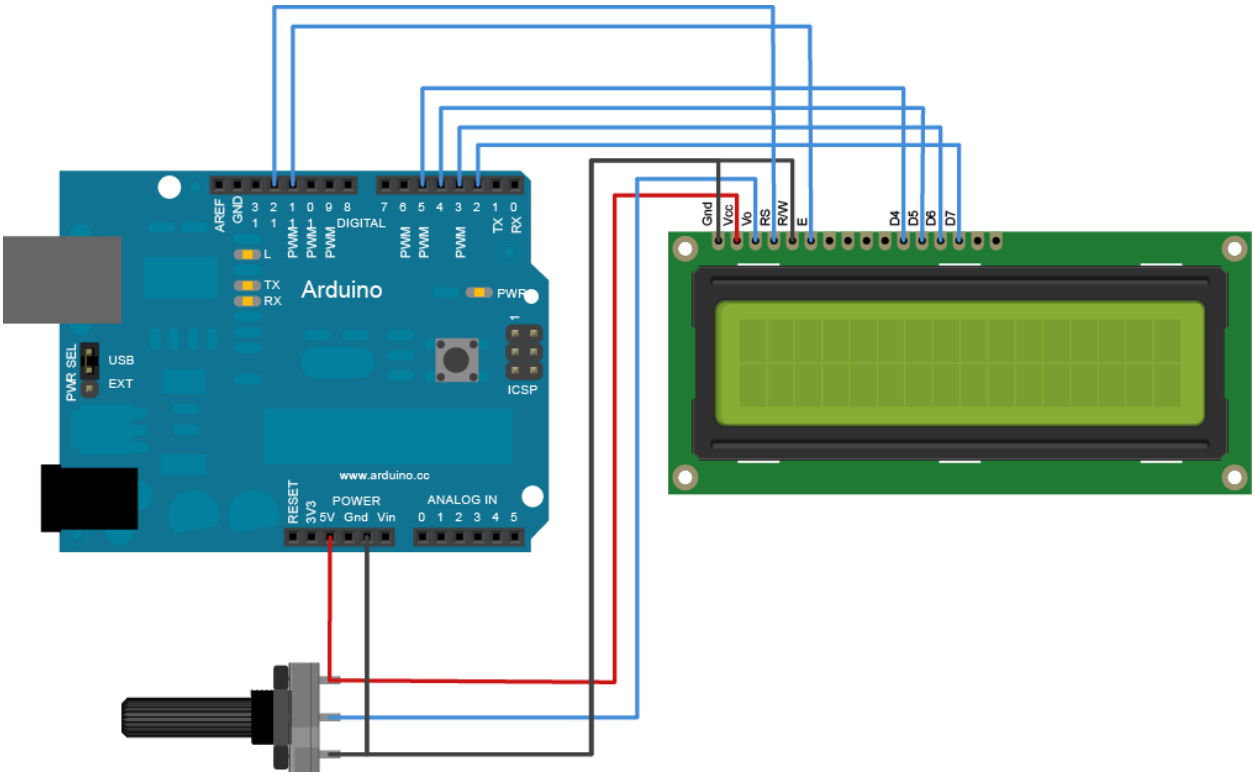
BLINKING LED

Displaying On a LCD Screen

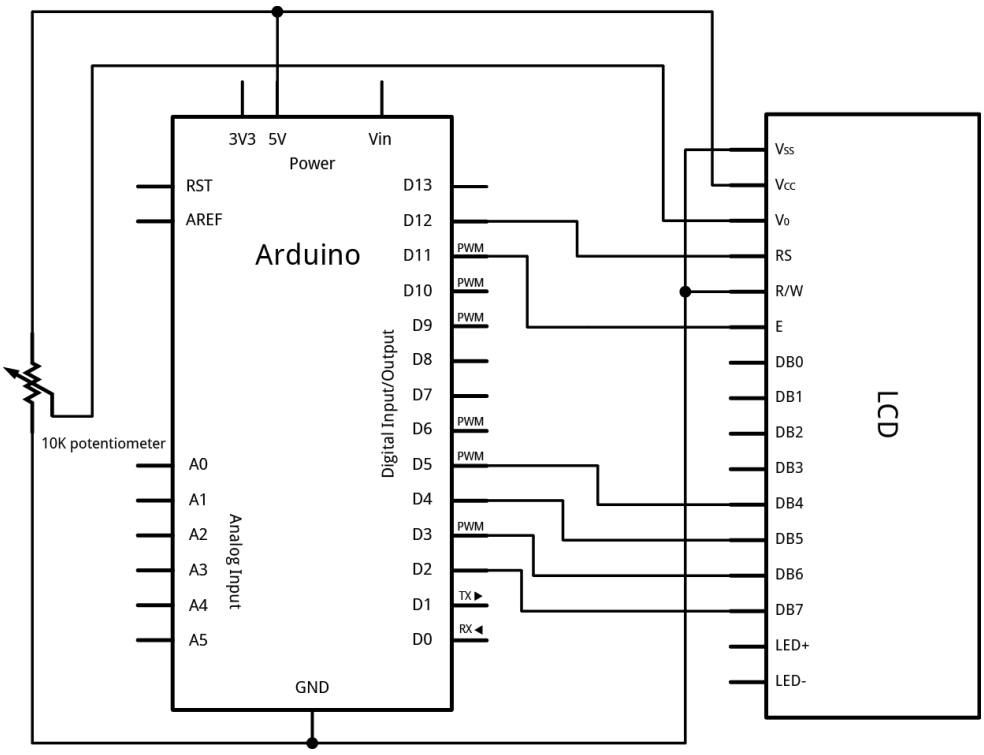
Hardware Required

- Arduino Board
- LCD Screen
- Pin headers to solder to the LCD display pins
- 10k Variable Resistance
- Breadboard
- Hook-up wire

Circuit



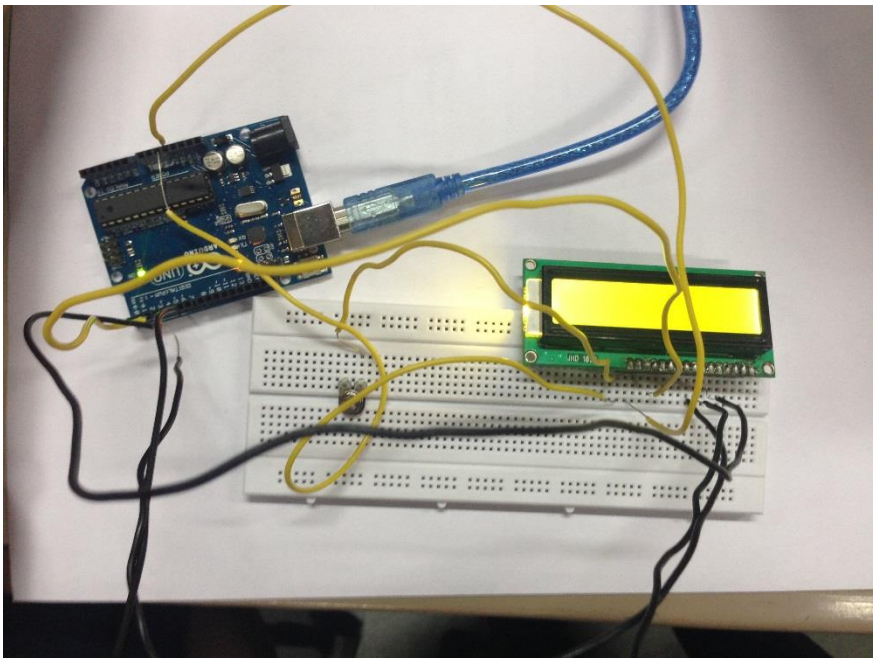
Schematic



Code

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
  lcd.begin(16, 2);

  lcd.print("hello, world!");
}

void loop()
{
  lcd.setCursor(0, 1);
  lcd.print(millis()/1000);}
```

Results



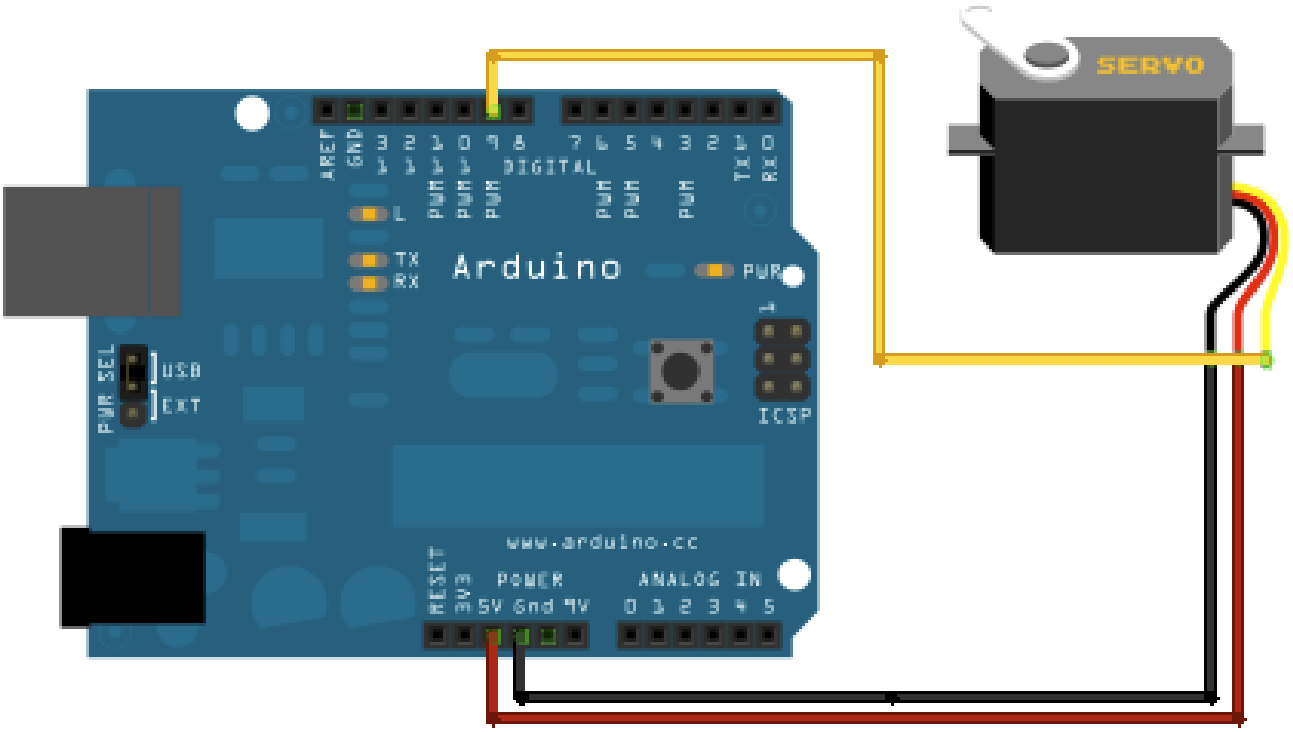
DISPLAYING ON A LED

Interfacing Servo Motor with Arduino Uno

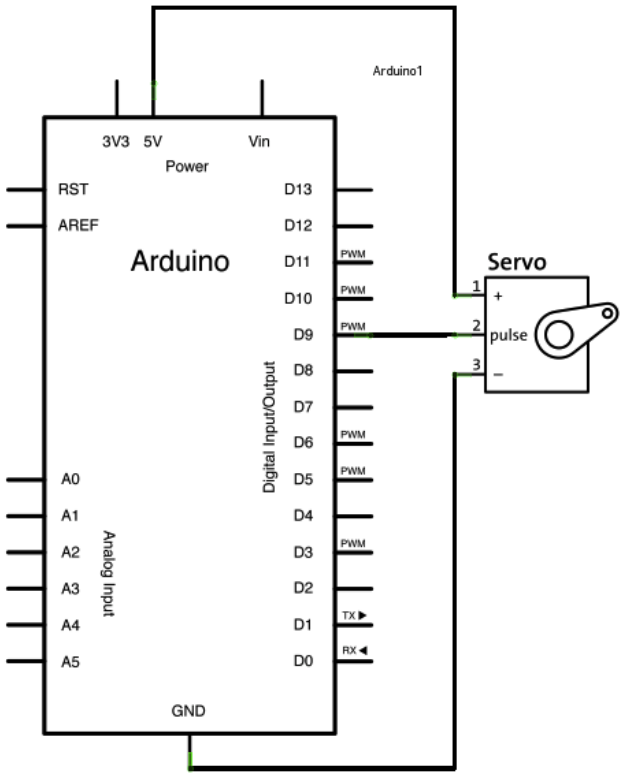
Hardware Required

- Arduino Uno Board
- Servo Motor
- Hook Up Wires

Circuit



Schematic



Code for Arduino

```
#include <Servo.h>

Servo servo1; Servo servo2;


void setup() {

  pinMode(1,OUTPUT);
  servo1.attach(14); //analog pin 0
  //servo1.setMaximumPulse(2000);
  //servo1.setMinimumPulse(700);

  servo2.attach(15); //analog pin 1
  Serial.begin(19200);
  Serial.println("Ready");

}
```

```
void loop() {

  static int v = 0;

  if ( Serial.available()) {
    char ch = Serial.read();

    switch(ch) {
      case '0'...'9':
        v = v * 10 + ch - '0';
        break;
      case 's':
        servo1.write(v);
        v = 0;
        break;
      case 'w':
        servo2.write(v);
        v = 0;
        break;
      case 'd':
        servo2.detach();
        break;
      case 'a':
        servo2.attach(15);
        break;
    }
  }

  Servo::refresh();

}
```

Processing Code

```
import processing.serial.* ;


int gx = 15;
int gy = 35;
int spos=90;

float leftColor = 0.0;
float rightColor = 0.0;
```

```
Serial port;
void setup()
{
  size(720, 720);
  colorMode(RGB, 1.0);
  noStroke();
  rectMode(CENTER);
  frameRate(100);

  println(Serial.list());

  port = new Serial(this, Serial.list()[1], 19200);
}

void draw()
{
  background(0.0);
  update(mouseX);
  fill(mouseX/4);
  rect(150, 320, gx*2, gx*2);
  fill(180 - (mouseX/4));
  rect(450, 320, gy*2, gy*2);
}

void update(int x)
{
  spos= x/4;
  port.write("s"+spos);
  leftColor = -0.002 * x/2 + 0.06;
  rightColor =  0.002 * x/2 + 0.06;
  gx = x/2;
  gy = 100-x/2;

}
```

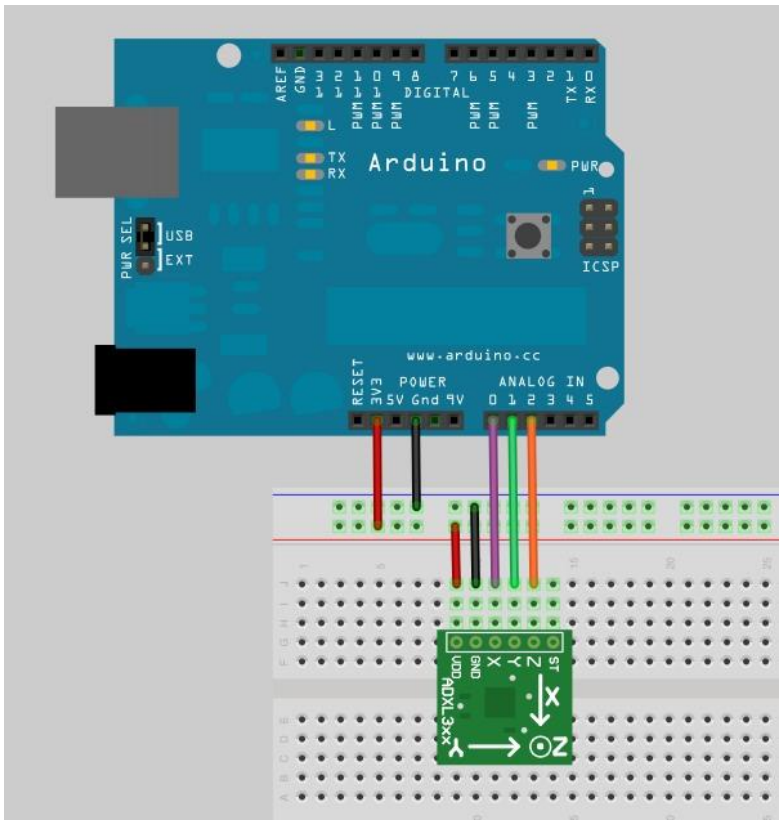
Chapter 3: INTERFACING A ACCELEROMETER WITH A MICROCONTROLLER

Hardware Required

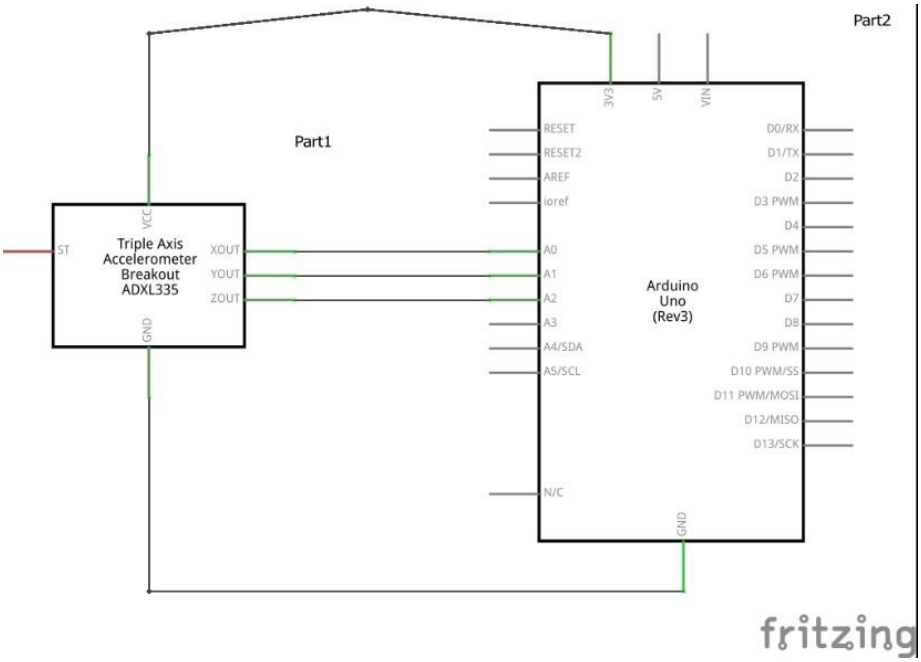
- Arduino Uno Board
- ADXL 335 Accelerometer
- Hook Up Wires

Circuit

- Connect the Vcc of ADXL 335 to the 3.3V pin of the UNO Board
- The X out , Y out, Z out of ADXL 335 to the Analog 0 ,1 ,2 Pins of the UNO Board
- Connect the Gnd Pin of ADXL 335 to Gnd Pin of the UNO Board



Schematic



Working

An **accelerometer** is a device that measures acceleration. The acceleration measured by an accelerometer is not necessarily the coordinate acceleration (rate of change of velocity).
The accelerometer is a built-in electronic component that measures tilt and motion. It is also capable of detecting rotation and motion gestures such as swinging or shaking.

Whenever an accelerometer is aligned in a certain direction it gives the degree of tilt in that direction. This is what our Project does.
After making all the connections shown above, we tilt the accelerometer in different orientations and notice that there is some changes in the reading shown in the Serial Monitor.
Accelerometers are widely used in our motion Gaming Consoles, Mobile Phones etc.

Code for Arduino Uno

```
#include < accelerometer.h>
const int xpin = 0;
const int ypin = 1;
const int zpin = 2;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.print("BEG");
  Serial.print("X"); Serial.print(analogRead(xpin));
  Serial.print("Y"); Serial.print(analogRead(ypin));
  Serial.print("Z"); Serial.print(analogRead(zpin));
  Serial.println();
  delay(50);
}
```

Communication Code to retrieve data

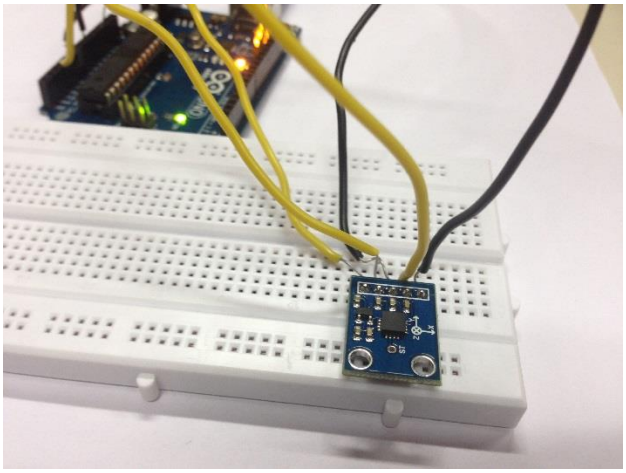
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.Threading;

namespace ADXL335
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

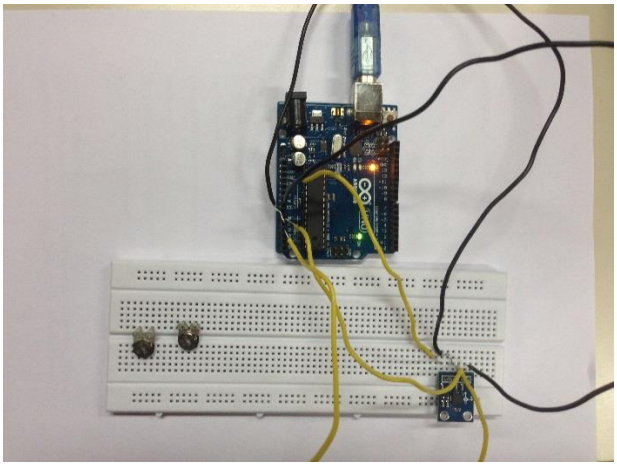
```
private SerialPort serial_port_;
private Thread     thread_;

private void refresh_button_Click(object sender, EventArgs e)
{
   portlist_comboBox.Items.Clear();
   foreach (string s in SerialPort.GetPortNames())
   {
      portlist_comboBox.Items.Add(s);
   }
}

private void connect_button_Click(object sender, EventArgs e)
{
   if ("Connect" == connect_button.Text)
   {
      if (-1 == portlist_comboBox.SelectedIndex)
      {
         MessageBox.Show("Select com port");
         return;
      }
      string port_name = portlist_comboBox.SelectedItem.ToString();
      serial_port_ = new SerialPort(port_name, 9600);

      serial_port_.Open();
      thread_ = new Thread(read_serial_port);
      thread_.Start();

      connect_button.Text = "Disconnect";
   }
   else
   {
      try
      {
         thread_.Abort();
         serial_port_.Close();
      }
      catch
      {
      }
      connect_button.Text = "Connect";
   }
}

private void read_serial_port()
```

```
{
   while (serial_port_.IsOpen)
   {
      try
      {
         string str = serial_port_.ReadLine();
         set_axis(str);
      }
      catch (TimeoutException) { }
   }
}

delegate void SetTextCallback(string newText);
private string str_;

private void set_axis(string text)
{
   if (this.X_progressBar.InvokeRequired)
   {
      SetTextCallback d = new SetTextCallback(set_axis);
      this.Invoke(d, new object[] { text });
   }
   else
   {
      str_ += text;
      int i1 = str_.IndexOf("BEG");
      int i2 = str_.IndexOf("BEG", 1 + i1);
      if ((-1 != i1) && (-1 != i2))
      {
         string current = str_.Substring(0, i2);
         current = current.Substring(current.IndexOf("BEG"));
         str_ = str_.Substring(i2);

         int beg_index = current.IndexOf("BEG");
         int x_index = current.IndexOf("X");
         int y_index = current.IndexOf("Y");
         int z_index = current.IndexOf("Z");

         if ((0 != beg_index) || (3 != x_index) || (-1 == y_index) || (-1 == z_index))
         {
               MessageBox.Show("Error " + current + ", beg_index = " +
            beg_index.ToString()      + ",   x_index = " + x_index.ToString() + ", y_index = "
            + y_index.ToString() + ", z_index = " + z_index.ToString());
            return;
         }
```

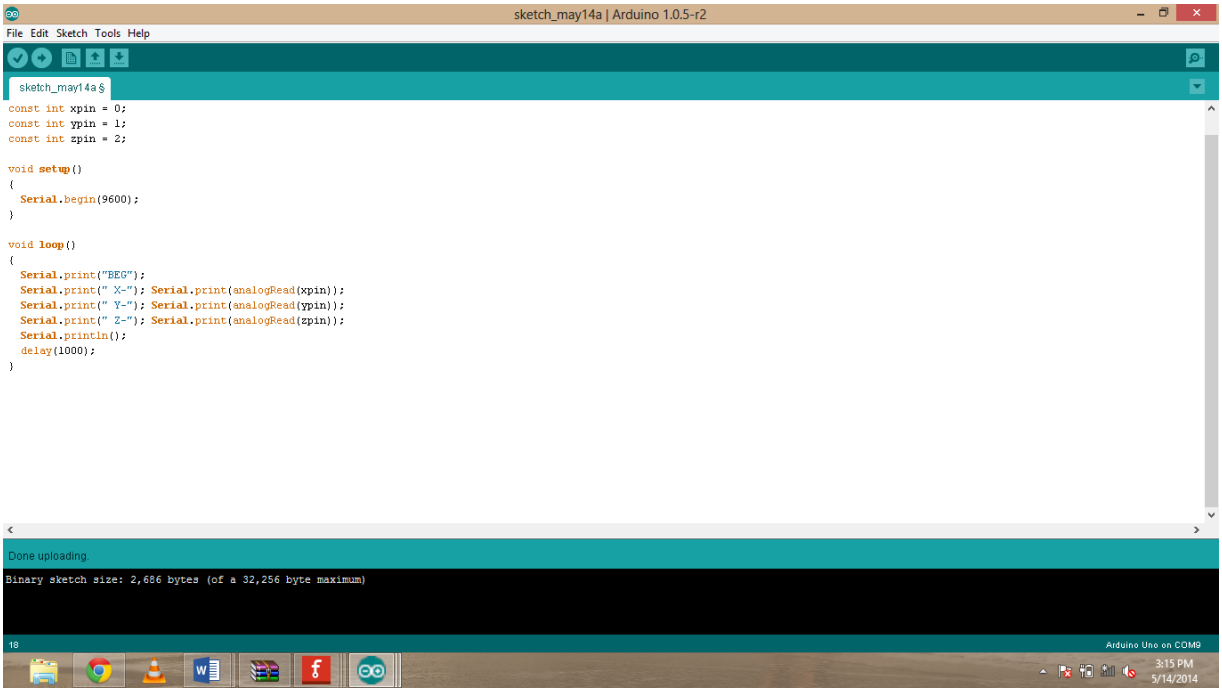
```
                int x = Convert.ToInt32(current.Substring(x_index + 1, y_index - x_index - 1));
                int y = Convert.ToInt32(current.Substring(y_index + 1, z_index - y_index - 1));
                int z = Convert.ToInt32(current.Substring(z_index + 1));

                if (x > X_progressBar.Maximum) x = X_progressBar.Maximum;
                if (y > Y_progressBar.Maximum) y = Y_progressBar.Maximum;
                if (z > Z_progressBar.Maximum) z = Z_progressBar.Maximum;

                if (x < X_progressBar.Minimum) x = X_progressBar.Minimum;
                if (y < Y_progressBar.Minimum) y = Y_progressBar.Minimum;
                if (z < Z_progressBar.Minimum) z = Z_progressBar.Minimum;

                X_progressBar.Value = x;
                Y_progressBar.Value = y;
                Z_progressBar.Value = z;
            }
        }
    }

    private void Form1_FormClosed(object sender, FormClosedEventArgs e)
    {
        try
        {
            serial_port_.Close();
        }
        catch
        {
        }
    }
  }
}
```

Results using Serial Monitor

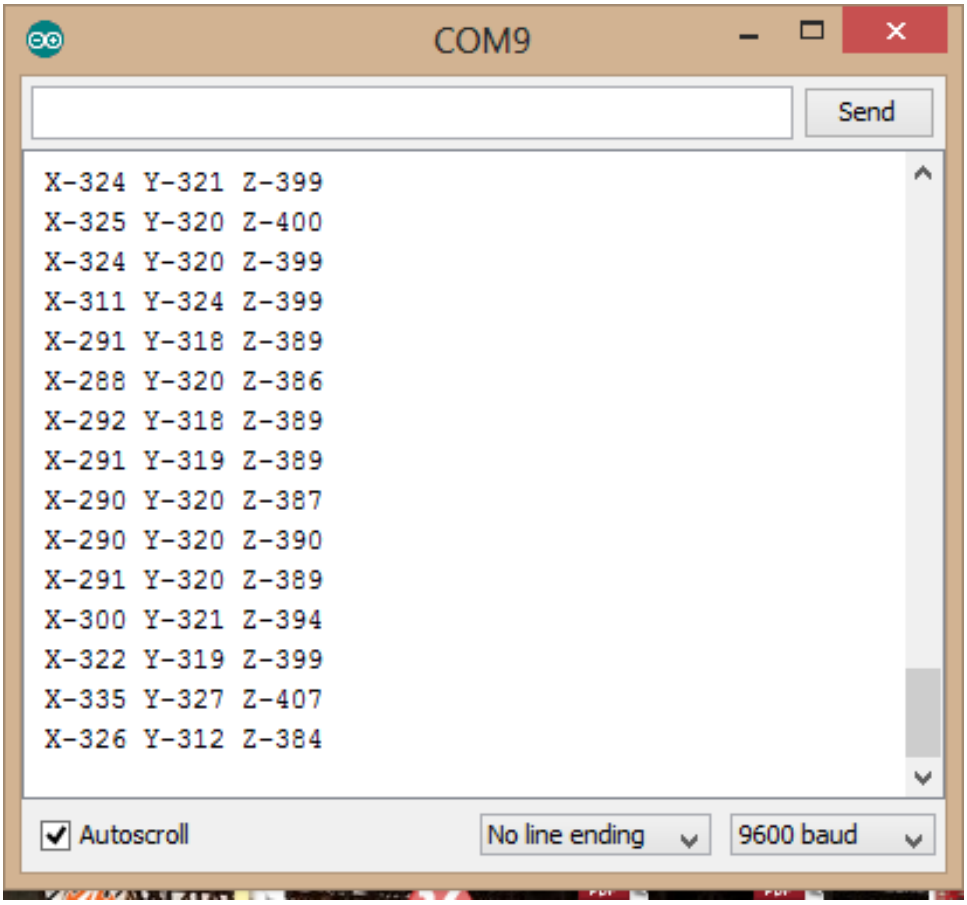




CIRCUIT WITH ADXL 335 ACCELEROMETER



CODE FOR ARDUINO


COM9
Send
X-324 Y-321 Z-399
X-325 Y-320 Z-400
X-324 Y-320 Z-399
X-311 Y-324 Z-399
X-291 Y-318 Z-389
X-288 Y-320 Z-386
X-292 Y-318 Z-389
X-291 Y-319 Z-389
X-290 Y-320 Z-387
X-290 Y-320 Z-390
X-291 Y-320 Z-389
X-300 Y-321 Z-394
X-322 Y-319 Z-399
X-335 Y-327 Z-407
X-326 Y-312 Z-384
Autoscroll
No line ending
9600 baud

CHAPTER 4 – FUTURE SCOPE

Accelerometers are used in tilt sensing applications in Cell phones and also find use in Motion gaming consoles. The various other fields in which accelerometers can be useful are:

Vibration Analysis

Measuring the frequency, strength, and signature of vibrations is useful in many machine health and industrial monitoring applications. The accelerometers are capable of determining the above parameters over a wide frequency and displacement range.

Other applications where vibration monitoring maybe useful are:

- Structural Vibration - analysis and identification of vibration sources and problems in structures
- Product Testing - vibration and shock testing to identify potential design problems
- Acceptance Testing - testing and analysis to ensure products comply with specified vibration tolerance limits
- Workplace Vibration - measurement and analysis of vibration from hand tools and other equipment

Inertial Navigation

In inertial navigation acceleration sensors can be used for making distance measurements. Inertial measurements are frequently required in the tracking of planes, boats, and automobiles over long distances and longtime constants. Inertial navigation is an extremely demanding application for sensors and many factors contribute to the performance of an inertial navigation system. Alignment, scale factor errors, and offset errors are crucial, because a constant error in these readings will result in a quadratic ally growing position error as given in the following equation:

$$PosError = \frac{1}{2} * AccError * T^2$$

Tilt / Angle Sensing

Angle sensing is the measurement of angles with an acceleration-based sensor. In most cases, these measurements are made using the Earth's G field as a reference. For angles less than 200, you can approximate the sine function with a linear response. Then the relationship between angle and Vout is:

Angle = (Vout-Offset_Voltage)/Scale Factor

CHAPTER 5 – CONCLUSION

Through this Project, I got the basic idea of working with Microcontrollers and its applications. Accelerometers are devices that are used to sense tilt & motion. This project taught me about Accelerometers as well as interfacing it with Microcontrollers. The working & function of Components like MAX 232 and Arduino Uno board were also understood really well. Thus this Project enriched my knowledge and was found to be very beneficial.

CHAPTER 6 – TOOLS & TECHNIQUES USED

Proteus - It is a software for microprocessor simulation, schematic capture, and printed circuit board (PCB) design. It is developed by Lab center Electronics. It combines the ISIS schematic capture and ARES PCB layout programs to provide a powerful, integrated and easy to use suite of tools for professional PCB Design.

Arduino IDE - The Arduino integrated development environment (IDE) is a cross-platform application written in Java, and is derived from the IDE for the Processing programming language and the Wiring projects. It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as syntax highlighting, brace matching, and automatic indentation, and is also capable of compiling and uploading programs to the board with a single click. A program or code written for Arduino is called a "sketch".

Frit zing - Frit zing is an open source software initiative to support designers and artists ready to move from physical prototyping to actual product. It was developed at the University of Applied Sciences of Potsdam. The software is created in the spirit of Processing and Arduino and allows a designer, artist, researcher, or hobbyist to document their Arduino-based prototype and create a PCB layout for manufacturing.

CHAPTER 7 – REFRENCES

- **[1] Edward A. Lee and Sanjit A. Seshia, Introduction to Embedded Systems, A Cyber-Physical Systems Approach**

- **[2] Sangiovanni-Vincentelli**, A., **Zeng**, H., **Di Natale**, M., **Marwedel**, **Embedded Systems Development**

WEB REFRENCES

- **http://autosysprogs.blogspot.in/2011/02/adxl335-accelerometer.html**

- **http://medialappi.net/lab/equipment/sensors/adxl335/**

- **http://en.wikipedia.org/**

- **http://www.arduino.cc/**