DESIGNING A TRI-AXIAL ACCELEROMETER INTERFACE FOR THE MEASUREMENT OF IMPACT FORCES CAUSED BY ATHLETIC COLLISIONS

NSF Summer Undergraduate Fellowship in Sensor Technologies
Heather Marandola (Electrical Engineering), Swarthmore College
Advisor: Jim Ostrowski

ABSTRACT

This paper describes the design of an accelerometer system to be used for the transmission of collision forces during the course of a contact sporting event. Discussed in this paper are basic accelerometer and transmission principles, the development and results of data acquisition experiments and the design of three evaluation boards: accelerometer, transmission and reception. My work has led to the design of preliminary calibration and display techniques for dual axis accelerometer outputs, and I have produced evaluation boards which can be utilized in further experimentation. Though a secure transmission link was not established, progress has been made in the development of the transmission scheme to be implemented on the device. Measurement along three axes was not implemented, however, with minor revisions, the evaluation boards are capable of handling one, two, three or four axes. The device is not ready to be installed into the sporting arenas, but some of the principles and design ideas presented in this paper might be of use in future implementation.

1. INTRODUCTION

Contact sport athletes, particularly football and hockey players, are involved in very high impact collisions (see bottom). This project was designed to create a device that can determine the force with which a player is hit and wirelessly transmit that information to a terminal that can display the data in some intelligible form. At present, the accelerometer interface is desired for entertainment purposes. Some hockey organizations such as the Philadelphia Flyers are attracted to the idea of being able to display the magnitude of collisions to the fans. In the future the interface could be used in such applications as training, recruiting, or officiating.

(left) http://www.exploratorium.edu/hockey/checking1.html
(right) http://members.tripod.com/~Yale_Football/yale.html

Accelerometers have been applied as training devices to many other athletic environments. In baseball, accelerometers are placed on the handles of bats to measure bat speed and acceleration as the player swings. Accelerometer devices used in golf measure the force on the head of the club when a ball is driven. Other devices have been used in rowing to monitor arm and shoulder motion. There have been attempts at creating devices to be used in the boxing ring to record the force of a punch to the head.

This project is most similar to the boxing example [1]. Most of the other accelerometer applications do not involve a wireless transmission process but simply store the readings within the device or transmit the data by wire to a display mechanism. Also, many of those devices are not subject to the types of collisions that a hockey or football player might encounter. This project is an attempt at bringing collisions or tackles a little closer to the fans. Those who are lovers of contact sports will understand the entertainment value that this device could have within the sporting industry.

2. GETTING STARTED WITH THE ACCELEROMETER

An accelerometer is a device containing a small mass suspended within an outer casing by small metallic restraints. When the device accelerates, the small mass exerts a force on the restraints, which is output as a voltage. This voltage is proportional to acceleration. Accelerometers are used for applications such as the measuring of seismic disturbances, the deployment of airbags in automobiles and as a method of guidance in aircraft navigation systems.

2.1 Choosing the Accelerometer

Accelerometers have many defining characteristics. They are designed to support a wide range of acceleration magnitudes. For example, Analog Devices manufactures low-cost accelerometers which can handle magnitudes of ±2g's to ± 100g's [2], while Entran Devices manufactures accelerometers that can handle magnitudes of ±2g's to ± 5000g's [3]. As the ranges of the accelerometers vary so do their sensitivities. Analog Device's ±2g accelerometer (ADXL202) has an analog sensitivity of about 312 mV/g [4], while Entran Devices manufactures a high-sensitivity ±2g accelerometer (ECGS-A series) with an internally amplified sensitivity of about 2500 mV/g [5]. Accelerometers are also designed based upon the number of axes they support. Analog Devices has accelerometers that can support one or two axes, while Entran Devices has accelerometers which support one, two or three axes.

Of these considerations in choosing the proper accelerometer, the one most crucial to my application is the selection of the proper 'g'-range to work with. Not many studies have been done on the actual magnitude of an impact that a football or hockey player can be hit with. Studies have shown, however, that a boxer can be hit with peak forces approaching 250 g's [6]. It is not certain whether hockey and football collisions can be approximated by this estimate, but this is the average magnitude that I decided upon. The accelerometers best suited for this application are the Entran Devices miniature accelerometers. The EGA series can easily support 250g accelerations. They are very compact and rugged, which would make it practical to insert them into a hockey or football player's helmet. Also, they support the tri-axial accelerations desired for this application. Unfortunately, these accelerometers are expensive, ranging from $500 to $2000 [7]. For experimentation purposes a lower-range, less expensive accelerometer was used.

Analog Devices' ADXL210 is a bi-axial, ±10g accelerometer with a shock survival of 1000g's. It is a surface mount chip that measures changes in acceleration using pulse width modulation (PWM). The chip contains a surface micromachined polysilicon structure suspended by polysilicon springs over a silicon wafer. The springs provide a resistance against acceleration forces. An acceleration will deflect the structure and unbalance a differential capacitor producing an output square wave with an amplitude proportional to acceleration. This signal is then demodulated, put through a low pass filter and converted to a duty cycle modulated signal. It is in this manner that the ADXL210 outputs a PWM signal [8]. The ADXL210 has a sensitivity of approximately 100 mV/g and is often used in such applications as inertial navigation, seismic monitoring, vehicle security systems, and battery powered motion sensing [9]. This accelerometer is also very small and inexpensive making it a very practical choice for experimentation.

2.2 Constructing An Evaluation Board

The test board was constructed using an evaluation board designed for the ADXL202. These two devices are alike in their pin layout; thus it was possible to remove the ADXL202 from its evaluation board and attach the ADXL210 with no further adaptations. The schematic for the evaluation board is shown below. The period of the duty cycle modulated signal is determined by the value of 'Rset' in Figure 1. The period is described by Equation 1.

Equation 1 [10]:

$$Period = \frac{Rset(\Omega)}{125M\Omega}$$

A 220kΩ reset resistor was used, creating a duty cycle repetition rate of 1.76ms. Capacitors C2 and C3 act as filters for antialiasing and noise reduction. Their values determine the analog 3dB bandwidth or the highest frequency that the system will detect as well as the rms noise performance. The bandwidth and rms noise are described by Equations 2 and 2, respectively.

Equation 2 [11]:

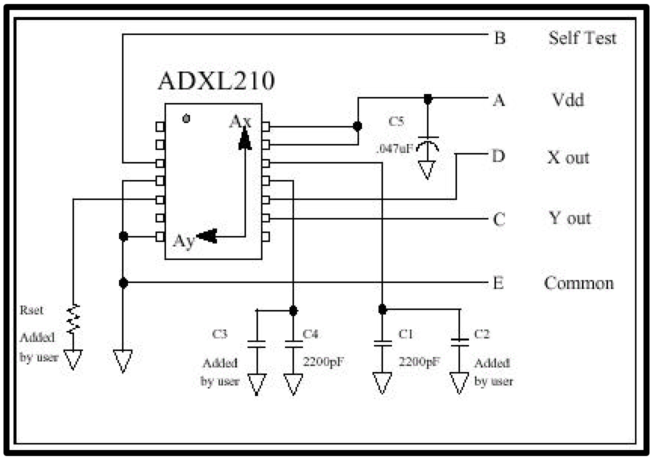
$$F(3dB) = \frac{1}{(2p(32k\Omega)xC(2,3))} = \frac{5mF}{C(2,3)}$$

Equation 3 [12]:

$$Noise(rms) = (\frac{500mg}{Hz})x(\sqrt{BWx1.5})$$

C2 and C3 were set to .47 μF, producing a bandwidth of 10 Hz and a RMS noise performance of 1.9mg. The evaluation board schematic as provided by Analog Devices is shown in Figure 1.

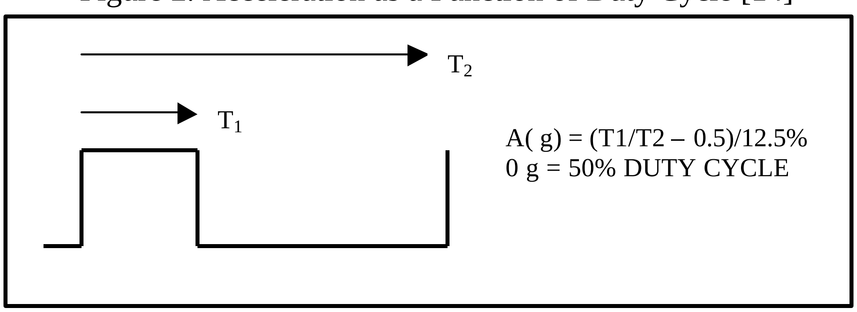
Figure 1. ADXL210 Evaluation Board [13]



3. EVALUATING THE ACCELEROMETER DATA

When examined on an oscilloscope, at zero g's, the ADXL210 produces a 50% duty cycle. The duty cycle or pulse width varies in width as the accelerometer is turned or shaken. T_2, the period of the duty cycle modulated signal described by Equation 1, remains constant. The acceleration as a function of duty cycle is shown below:

Figure 2. Acceleration as a Function of Duty Cycle [14]



A Scenix microcontroller was used to evaluate the width of the changing pulse and generate interpretable data to be evaluated at a computer terminal. To implement the microcontroller it was necessary to spend time gaining an understanding of the Scenix assembly language. I began by performing the simple task of making a light blink on and off. I upgraded that code by implementing a switch, and then further upgraded that code to make the light blink on and off at two different rates. By gaining an understanding of the very basics, I was able to walk through the accelerometer PWM code already generated by sophomore University of Pennsylvania student Daniel Walker.

The code is designed to find the time of the starting edge of a pulse, then find the time of the stopping edge, subtract the two and output the result. This process is completed within an interrupt routine. The interrupt routine is executed each time an edge (rising or falling, depending on which is needed) occurs during operation.

A few considerations had to be made which complicated the program. The program is designed to send the data to the computer in parallel. The parallel connection to the computer is eight bits wide and the data that needed to be sent was sixteen bits. Thus it was necessary to send the first byte through the RC port, wait for it to be received, then send the second byte. The sending and receiving is done in the main part of the code while the interrupts are not occurring. Portions of the main code from the microcontroller and the computer terminal are shown below:

```
; Sending the y-acceleration byte within the main code of the microcontroller

:byte1                                  ; The first 8 bits of the y-accel value
        mov     w,y_time                ; Go get the value of y-time determined in the interrupt
        mov     rc,w                    ; Put the value onto the rc data port (an 8 bit port)

        clrb    rb.7                    ; Tell computer that pic is ready to send data

        sb      ra.0                    ; Has the computer responded? Has the data been sent?
        jmp     :byte1                  ; If the computer does not have the data then wait for it to get it

:byte2
        mov     w,y_time+1              ; If byte is sent, go and get the second 8 bits of the y-accel value
        mov     rc,w                    ; Put the value onto the rc data port

        setb    rb.7                    ;       ...just like the above

        snb     ra.0                    ;               "
        jmp     :byte2                  ;               "
```

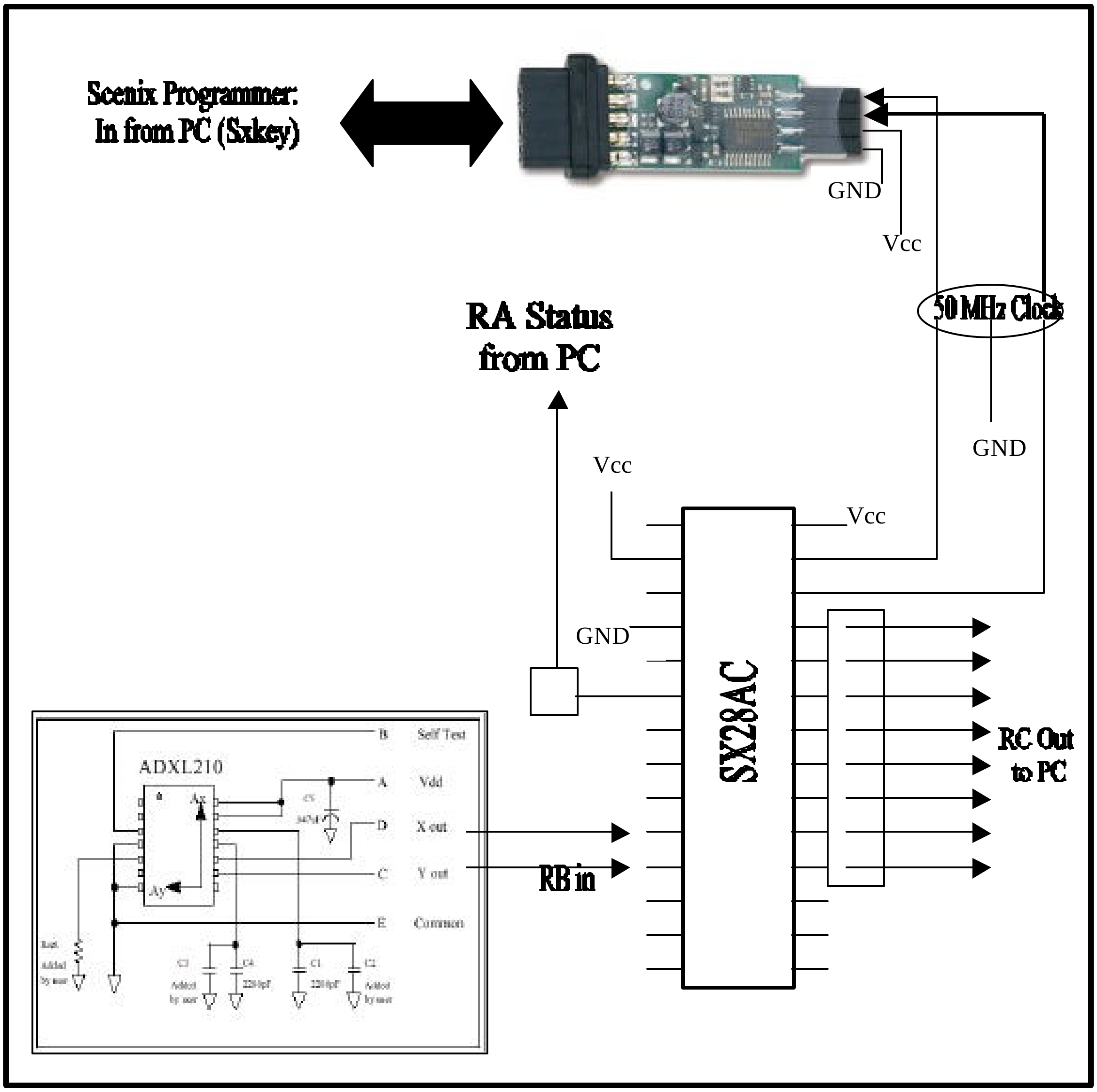
```
; Receiving the y-acceleration byte at the computer terminal

while(!kbhit()) {
        …
        …
        …
        …
 outp(0x37A,keep|33);                        ; tell the microcontroller the 1st byte is being received…
                                             ;                    (corresponds to PIC's ra.0)
 while(inp(0x379)&128==0);                   ; when the 1st byte is in the rc port…   (corresponds to PIC's rb.7)
 buf2=inp(0x378);                            ; go get the 1st y-accel byte… (corresponds to PIC's rc port)
 outp(0x37A,keep|32);                        ; tell the microcontroller the 2nd byte is being received
 while((unsigned char)(inp(0x379)&128)!=0);  ; when the 2nd byte is in the rc port…
 buf2|=inp(0x378)<<8;                        ; go get the 2nd y-accel byte

 printf("%u   %u \n",buf,buf2);              ; display the x-accel and y-accel each as a single num
 delay(1);                                   ; wait .001 seconds
 }
}
```

The code was initially assembled to perform calculations along one axis and I modified it to simultaneously perform calculations along two axes. The code will later need to be modified to include three axes. The completed code, along with detailed comments, can be found in Appendix (a). The experimental setup is shown below:

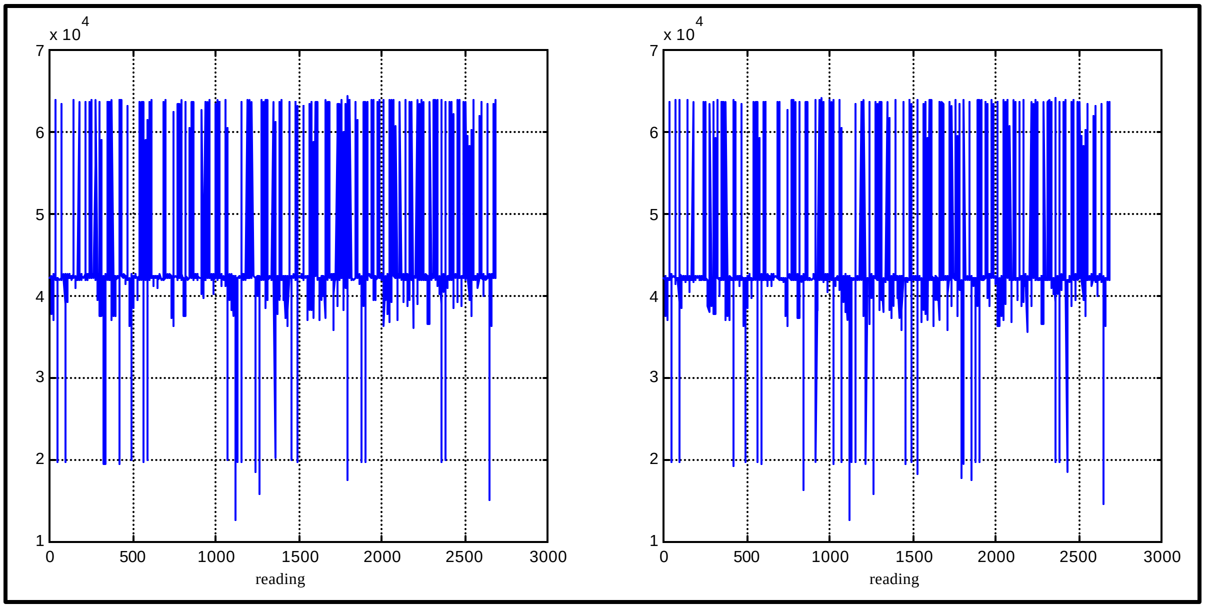
Figure 3. Experimental Setup #1: Accelerometer Evaluation [15]



3.1 Collecting the Raw Data

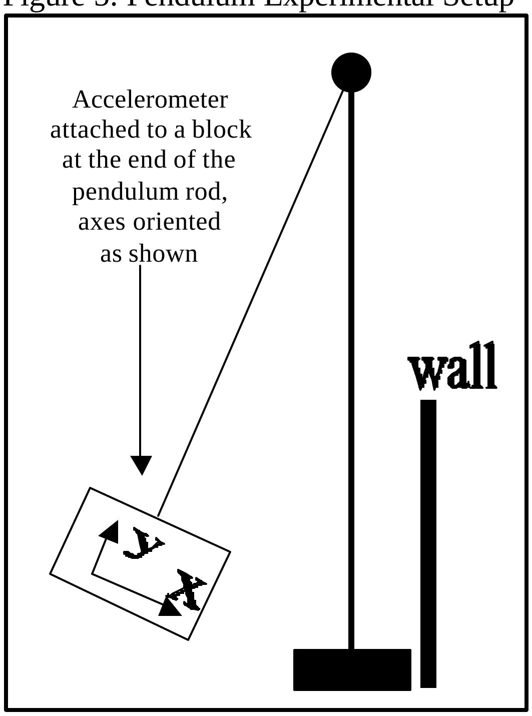
Using the evaluation board it was possible to collect data representing acceleration measurements. I conducted several pendulum experiments designed to display the effects of gradual and sharp changes in acceleration. Figure 4 displays the raw x acceleration and y acceleration data, respectively, when no acceleration force is acting along the axes of the device.

Figure 4: Zero Gravity Acceleration Measurement (x-accel left, y-accel right)



To gain an understanding of the type of output that the accelerometer would produce when accelerating, I designed a simple pendulum experiment. The setup is shown in Figure 5.

Figure 5: Pendulum Experimental Setup



The first experiment was to release the pendulum from a known angle and allow it to swing unobstructed. The pendulum was released from angles of 30, 45, 67, and 90 degrees. Figures 6.a and b, display the 30 and 67 degree trials. The 45 and 90 degree trials were omitted because they are similar to the experiments from 30 and 67 degrees, differing in scale. Figure 6.b includes labels relating the swing of the pendulum to the accelerometer output.

The second experiment was to insert an obstruction – the wall in Figure 5 – at the base of the pendulum. The pendulum was released from known angles of 30, 45, 67, and 90 degrees and allowed to collide into the wall, bounce off, collide again, etc. Figures 7.a and b display the 30 and 67 degree trials. The 45 and 90 degree trials were again omitted because they are similar to the experiments from 30 and 67 degrees, differing in scale. Figure 7.b includes labels relating the swing of the pendulum to the accelerometer output.

The units on all of these figures of raw data have been omitted because they have not been scaled into acceleration values; thus the numbers represent magnitude only.

Figure 6.a: Pendulum Free Fall Drop / 30 degrees (x-accel left, y-accel right)

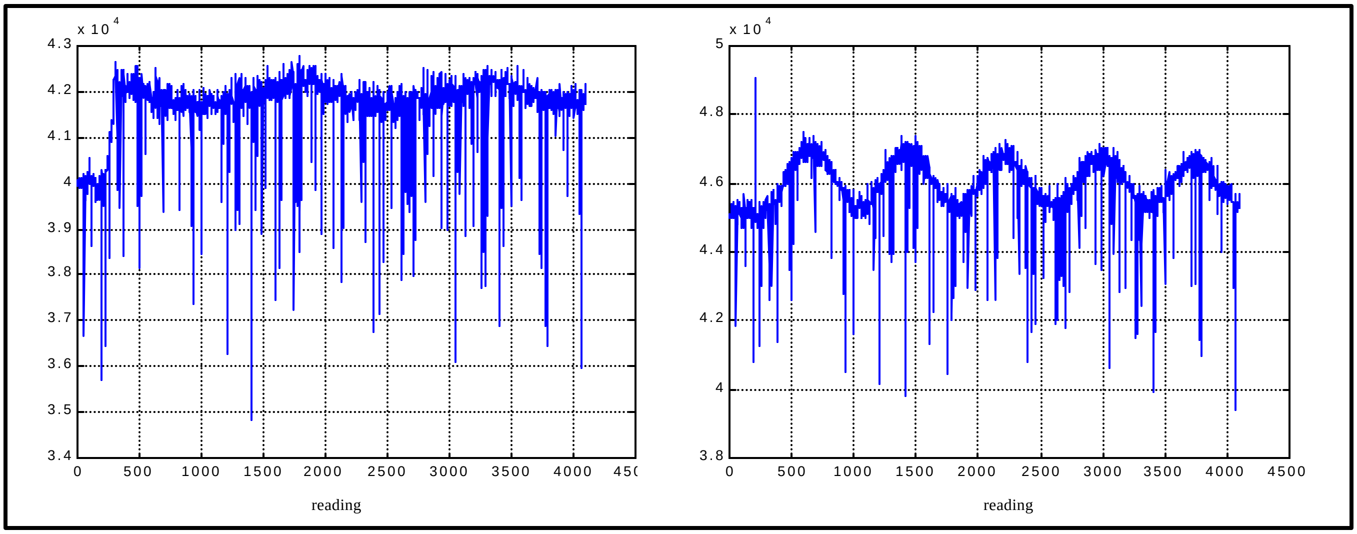
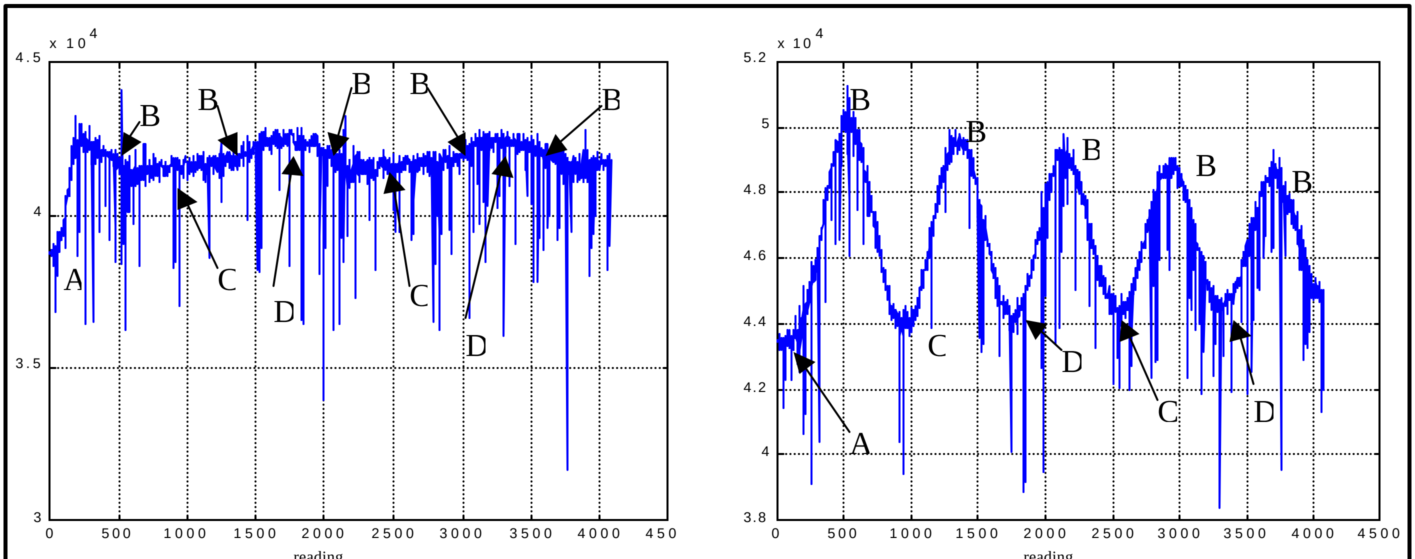
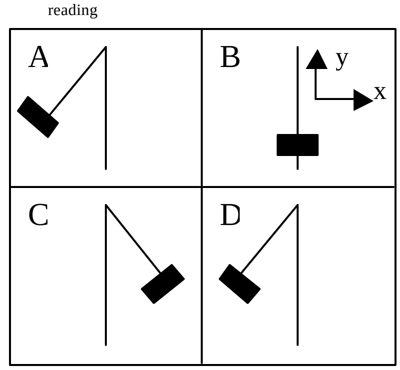


Figure 6.b: Pendulum Free Fall Drop / 67 degrees (x-accel left, y-accel right)





'A' indicates the release point of the pendulum (see right, top left)

'B' indicates the low point of the pendulum's swing (see right, top right)

'C' indicates one high point in the pendulum's swing (see right, bottom left)

'D' indicates other high point in the pendulum's swing (see right, bottom right)

(where the x and y axes are oriented on the block as depicted in position 'B')

Figure 7.a: 30 Degree Pendulum Drop / obstruction (x-accel left, y-accel right)

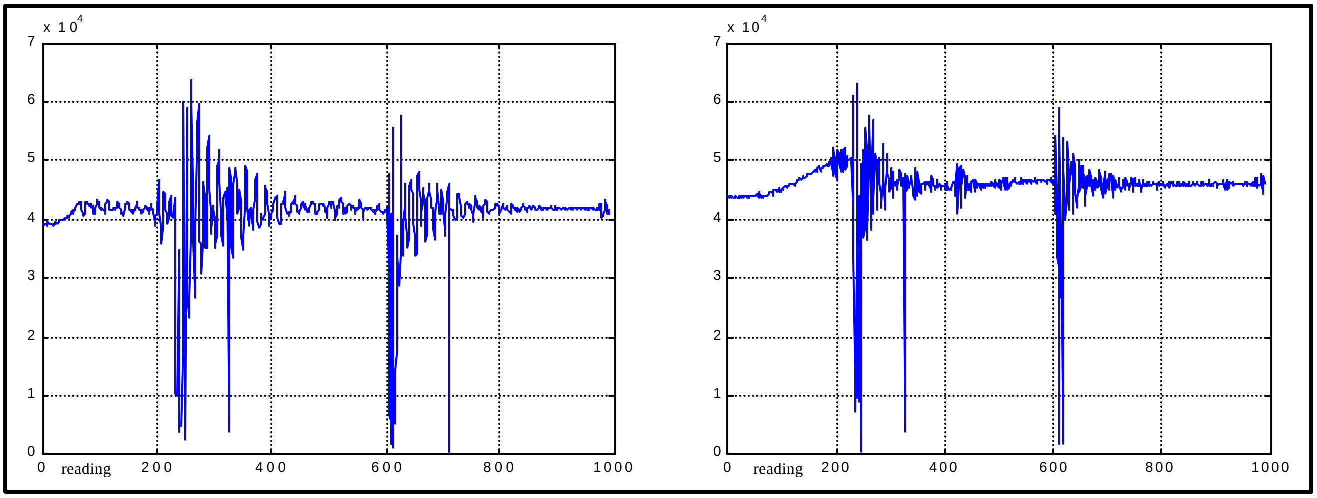
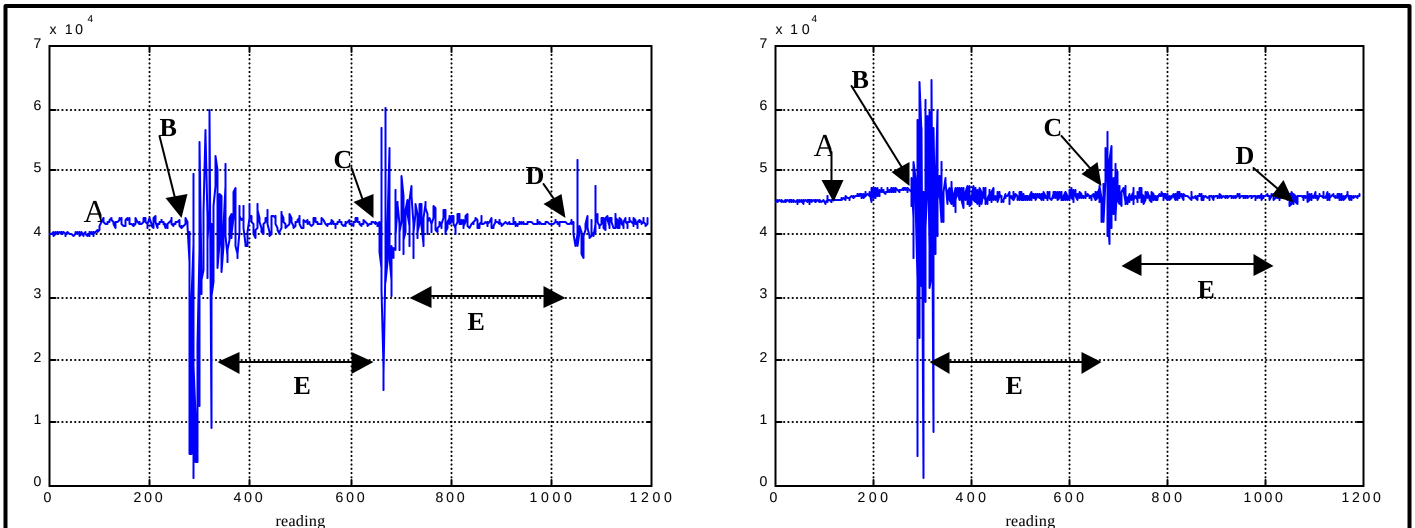


Figure 7.b: 67 Degree Pendulum Drop / obstruction (x-accel left, y-accel right)



'A' indicates the release point of the pendulum
'B' indicates the point of the first impact with the wall
'C' indicates the point of the second impact with the wall
'D' indicates the point of the third impact with the wall
'E' indicates the pendulum's recoil period

As experimental Figures 4, 6, and 7 show, that there was a significant amount of error. The following section describes the filtering techniques that were experimented with to try to reduce this error.

3.2 Filtering the Data

Three different filtering techniques were implemented on the accelerometer data through Matlab: a moving average filter, a low pass butterworth filter, and a median filter. The moving average filter and the low pass butterworth filter yielded unsatisfactory results. Large jumps in the accelerometer output caused by mechanical error, like those displayed in Figure 4, were not eliminated using the moving average filter. Rather, these errors averaged into the final output. The butterworth filter did not filter out as much of the noise as was needed and it also significantly reduced the magnitude of the desired signal.

The filter that was chosen was a fifteen point median filter. This filter successfully eliminated nearly all of the large jumps in the accelerometer output, but did not reduce the magnitude of the desired signal. Fifteen points were chosen as a middle ground. More points would have eliminated more of the desired output signal than was desired; fewer points would have allowed more noise to remain in the output signal. Figure 8, displays the zero gravity data of Figure 4 after being passed through the median filter. Figure 9 displays the 67 degree free fall data of Figure 6.b after being passed through the median filter. The Matlab code used to implement the median filter can be found in Appendix (b).

Figure 8: Filtered Zero Gravity Acceleration Data (x-accel left, y-accel right)

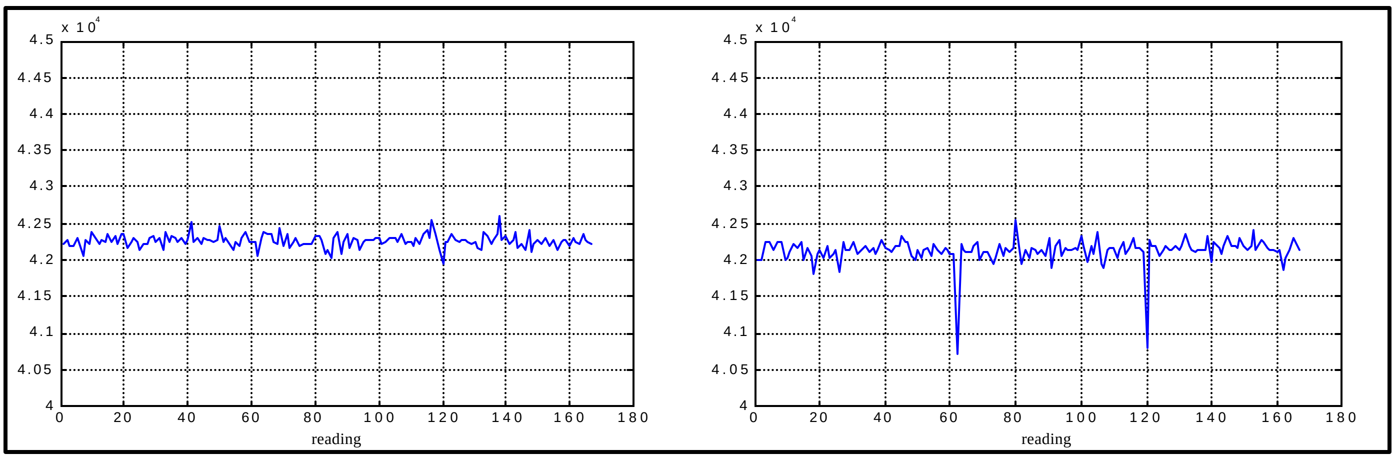
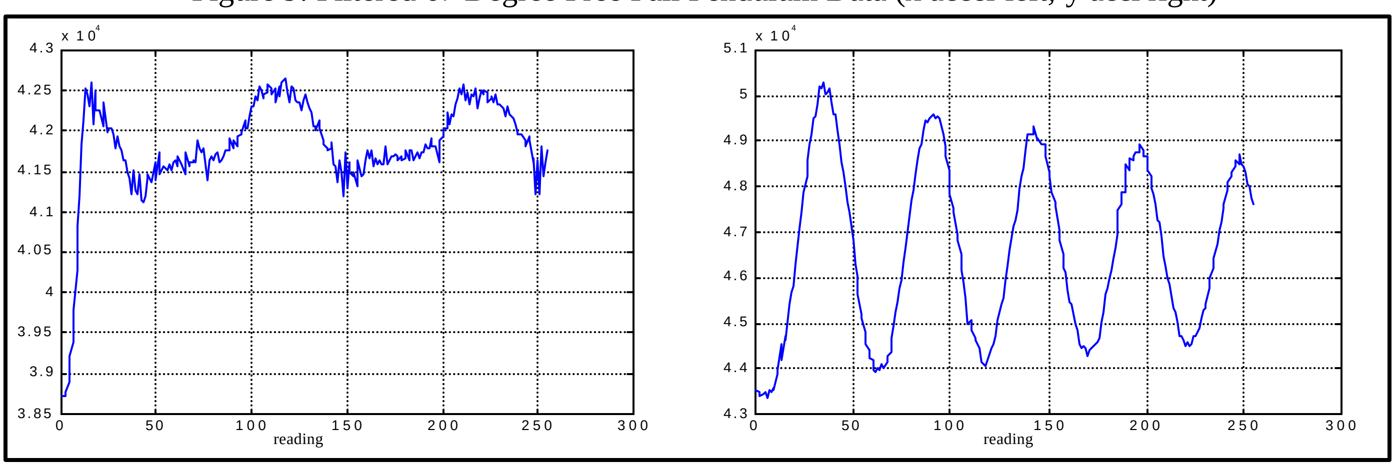


Figure 9: Filtered 67 Degree Free Fall Pendulum Data (x-accel left, y-accl right)

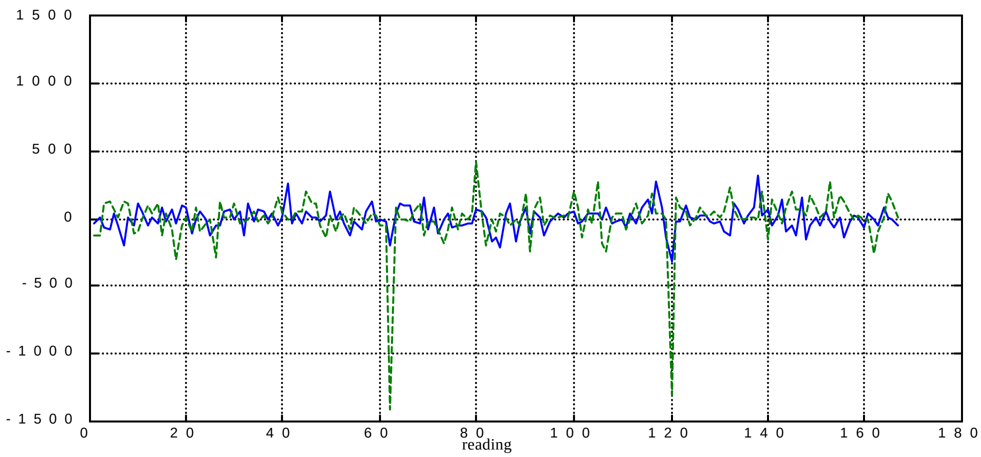


The above figures display the greatly reduced noise of the x and y acceleration outputs. There is still noise present, but the median filter very effectively eliminated much of the noise from the output signal.

3.3 Scaling The Data

It was necessary to scale the data to make the magnitude of the output interpretable in terms of gravity (g's). The scaling was performed by taking readings from five different accelerometer positions. The first position was taken with the accelerometer placed in the zero gravity position used to generate Figures 4 and 8. By averaging the filtered output over the entire sample period, x and y zeroing values of 42300 and 42127, respectively, were established. Using these values both the x and y acceleration outputs were centered around zero. Figure 10 displays the zero acceleration trial after the centering values have been applied.

Figure 10: Zero Gravity Trial / centering (x-accel '____', y-accel '- - - -')



After filtering and zeroing the accelerometer output, it was possible to establish a scale to convert the magnitudes into terms of acceleration using the four other accelerometer positions. The accelerometer positions and the appropriate outputs are shown in Figures 11 and 12. Position one corresponds to 1g or 9.8 m/s^2 acting on the x-axis; position two corresponds to -1g or -9.8 m/s^2 acting on the x-axis; position three corresponds to 1g acting on the y-axis; position four corresponds to -1g acting on the y-axis. To calculate the x-acceleration scale, the average magnitude from position two (-1g) was subtracted from the average magnitude from position one (1g). The resulting magnitude was equal to 2g. The same procedure was used to create the y-acceleration scale. The resulting scaling factors can be found in Table 1.

Figure 11: Developing a Scaling Factor for the X-Acceleration Output

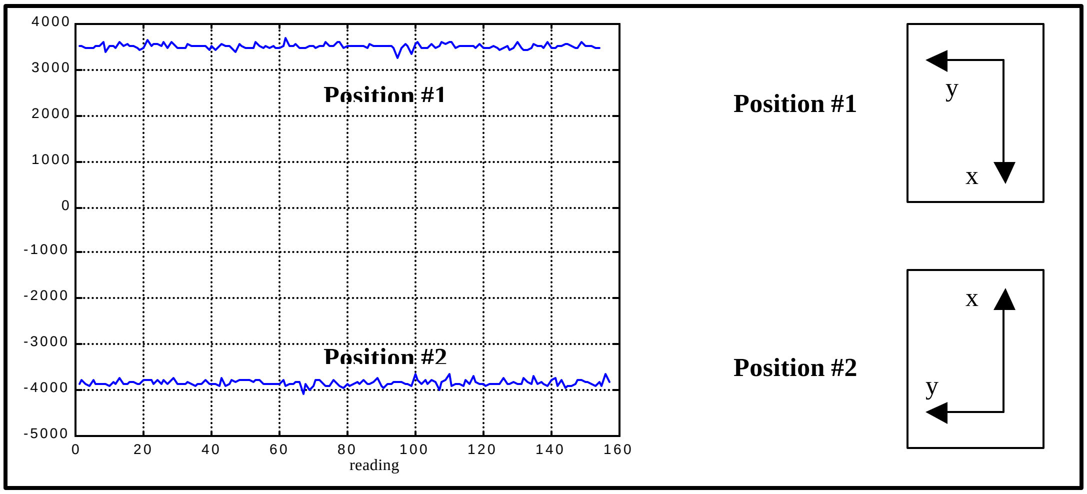


Figure 12: Developing a Scaling Factor for the Y-Acceleration Output

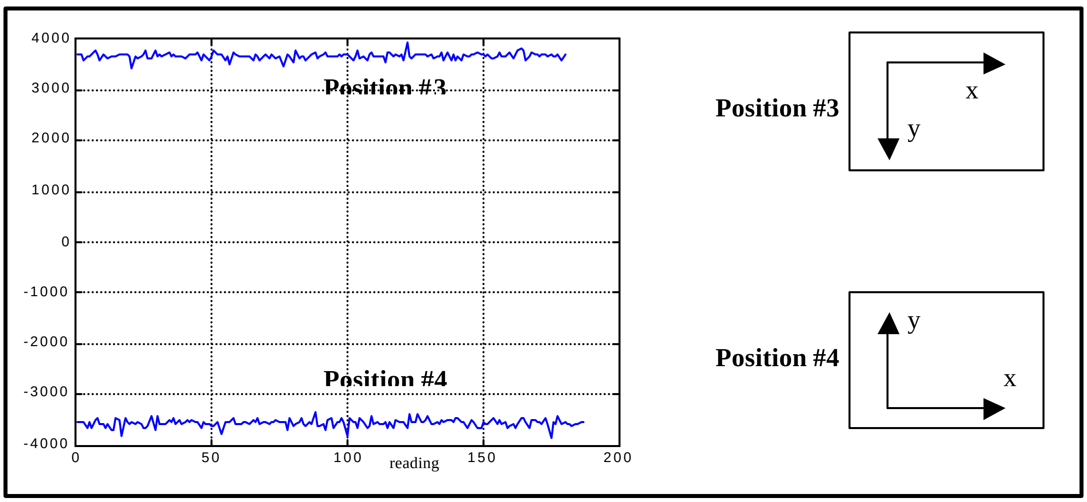


Table 1: Scaling Values

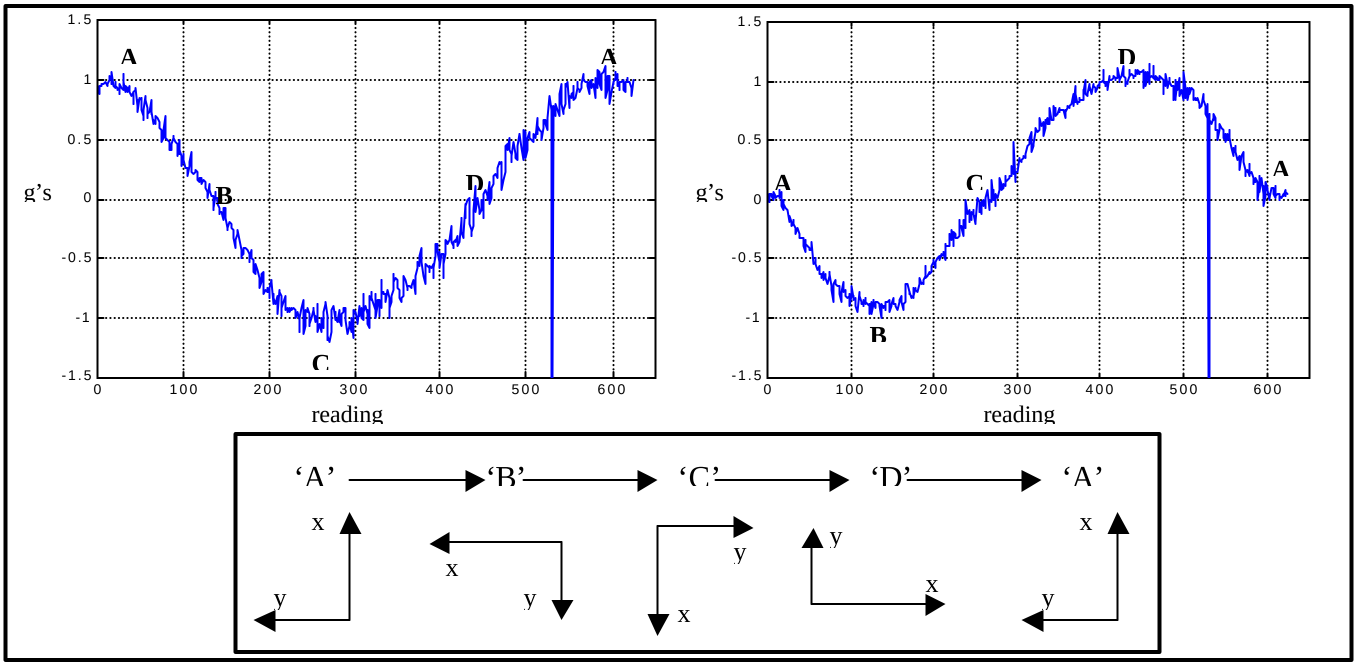
Axis	Acceleration (g's)	Magnitude
x-axis	2g	7332.6
y-axis	2g	7236.3

3.4 Putting It All Together

The filtering and scaling information was compiled into the data acquisition routine to produce a more efficient means of generating meaningful outputs. The incoming accelerometer data was first put through a fifteen point median filter, then zeroed and scaled down into measurements of acceleration. Figures 13, 14 and 15 below display acceleration outputs after filtering, zeroing, and scaling. Figure 13 displays x and y acceleration outputs, along with accelerometer positioning after a rotational experiment was applied to check for proper scaling.

Figure 14 displays the 67 degree free fall pendulum swing. Figure 15 displays the 67 degree pendulum swing with impact.

Figure 13: Filtered and Scaled Rotational Experiment (x-accel left, y-accel right)



Examination of the y-acceleration rotational curve suggested that the zeroing value was off center by approximately .1 g's. This slight offset was later eliminated by performing a minor recalibration. The new zeroing value was determined to be 42489.

Figure 14: Filtered and Scaled 67 Degree Pendulum Swing (x-accel left, y-accel right)

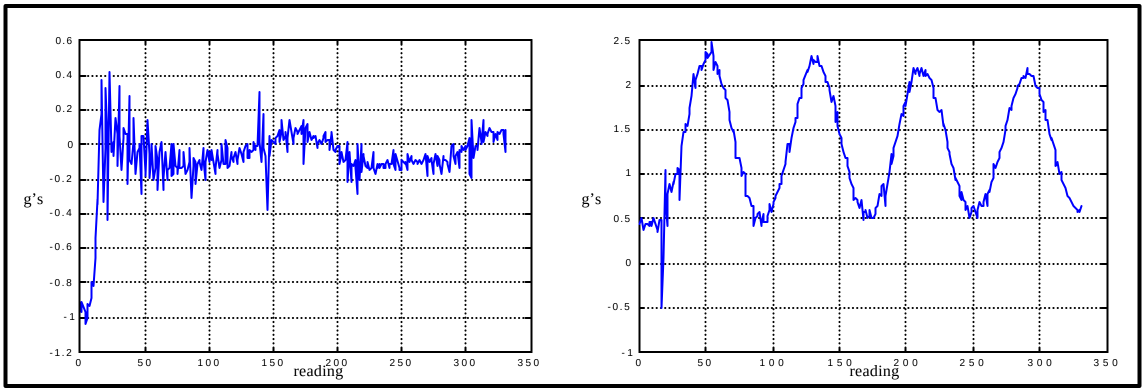
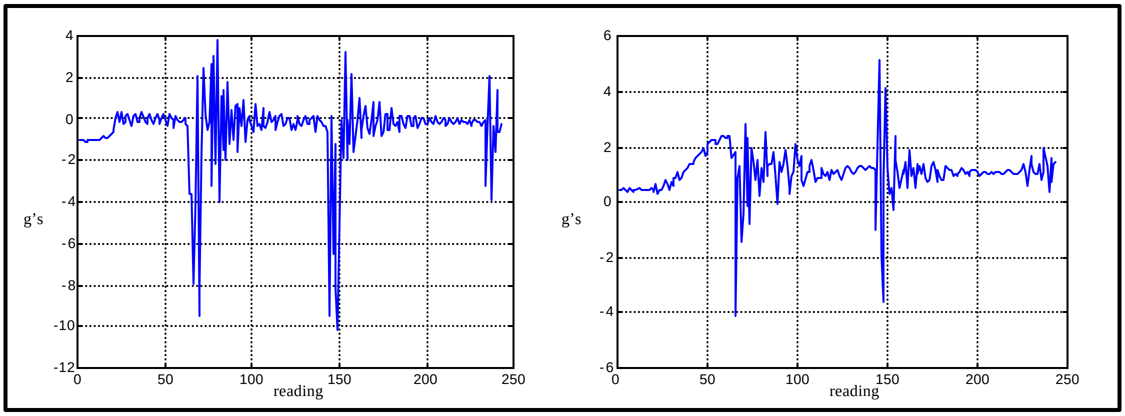
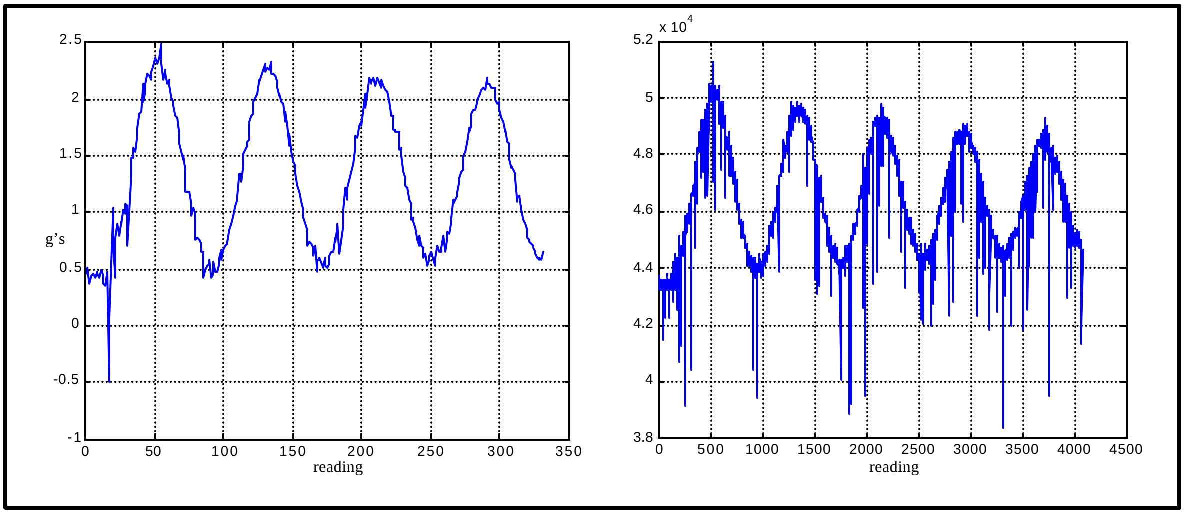


Figure 15: Filtered and Scaled 67 Degree Impact (x-accel left, y-accel right)



There is still some noise in the above outputs. Compared to the unfiltered output, however (see Figure 16 below), the noise has been significantly reduced.

Figure 16: Filtered (left) Versus Unfiltered (right) Pendulum Swing (y-accel)



4. DESIGNING A METHOD OF DISPLAY

In order to view the accelerometer output data without having to constantly read the magnitudes, I added a display mechanism to the data acquisition routine. After being filtered, zeroed and scaled, the x-acceleration and y-acceleration value from each reading were compared. Whichever value was greater in magnitude was stored and sent through a display routine which displayed the acceleration as a series of stars ('***'). Each star represented approximately one half of a 'g', or 4.9 m/s^2. A delay was also inserted into the routine in order to make the display easier to observe in real time. The completed data acquisition code can be found in Appendix (c). The display mechanism is shown in Table 2.

Table 2: Acceleration Display Mechanism

Display	**Acceleration Range (g's)**
0	0 to .25
0*	.25 to .75
0**	.75 to 1.25
"	"
0********************	9.75 to 10.25
Overload	> 10.25

5. DATA TRANSMISSION AND RECEPTION

The next crucial part of the project was designing a way to transmit and receive the acceleration measurements wirelessly. One consideration was what type of transmitter and receiver to use and at what frequency. It was also necessary to build an evaluation board for testing the data transmission. Another consideration was how to convert the data from being sent in parallel to the computer to being sent in series over the RF link. The development of these three design considerations is described in detail in the following three sections.

5.1 Choosing Transmission and Reception Devices

The transmitter and receiver pair were chosen based upon size, cost and capability. Linx Technologies manufactures several inexpensive modules from which I chose the LC series 315 MHz transmitter and receiver. The most significant reason for choosing this pair was the size. Figure 17 at right displays the transmitter and receiver, as well as a picture portraying the size of the devices. The final design requires that the transmitting portion of the device be installed in the helmets of the players. This is only reasonable if a small transmission board is constructed.

Figure 17 [16]:
Transmitter and Receiver

There were three different frequency ratings to choose from when deciding on the LC transmitter: 315 MHz, 418 MHz, and 433 MHz. The 315 MHz transmitter was chosen because although higher frequencies have more bandwidth, lower frequencies travel through objects and walls more efficiently. In a hockey or football setting, the signal must be able to efficiently travel through other players or obstacles (such as the fiberglass surrounding an ice rink).

The LC module can transmit and receive effectively at distances in excess of 300 ft, large enough for this application. The limit to the LC module is that it can only support data rates up to 5000 bits per second [17]. This is equivalent to transmitting 625 bytes every second. Since each accelerometer axis consists of two bytes, and there are currently two axes to be measured, the LC module is capable of sending approximately 156 complete sets of x and y accelerometer readings every second. When three axes are applied, this capability will be reduced to approximately 104 complete sets per second, equivalent to a sampling rate of approximately 9.55 ms. This is not an exceptionally fast rate so there is a chance that when very sharp collisions occur important magnitude readings could be lost. For this application, however, it is believed that the 9.55ms sampling rate is sufficient.

The LC transmitter/receiver module uses CPCA (Carrier-Present Carrier-Absent) modulation. This type of modulation is very simple, representing a logic low by the absence of a carrier and a logic high by the presence of a carrier, thus it is both easy to work with and cost effective.

In connection to the transmitter and receiver are the antennas that were chosen. For the receiving end a simply ¼ whip style antenna was chosen. For the transmitting end, Linx Technology's new SPLATCH Planar Antenna was chosen (see

Figure 18 [18]: SPLATCH

Figure 18). The SPLATCH is a very compact antenna, measuring 0.062” thick, 1.102” long, and 0.540” wide [19]. The dimensions made this antenna appealing since it could be easily concealed within the player’s helmet along with the transmitter.

5.2 Transmission and Reception Evaluation Board Design

The construction of the evaluation boards consisted mainly of gathering technical information on all of the devices within the system and determining the most efficient means of wiring them together. The simplicity of the devices made the resulting design very easy to understand and piece together. The transmission and reception schematics are shown in Figures 19 and 20, respectively.

Figure 19: Transmission Evaluation Board Schematic [20]

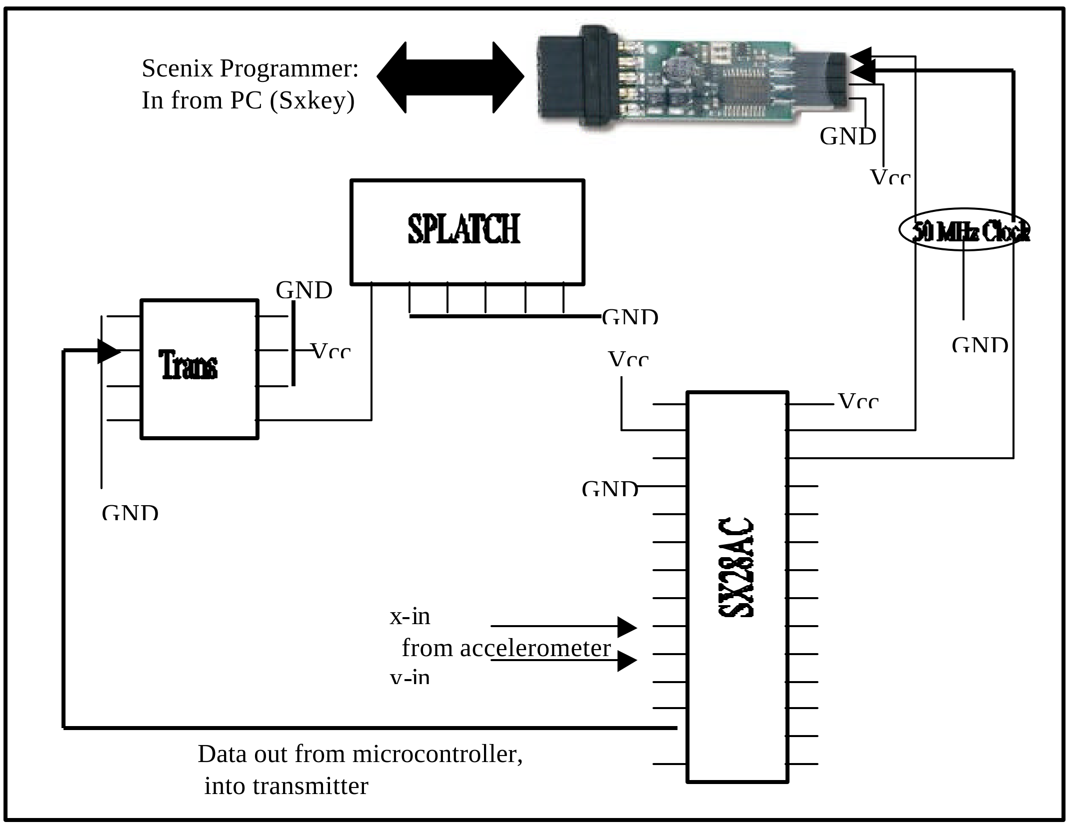
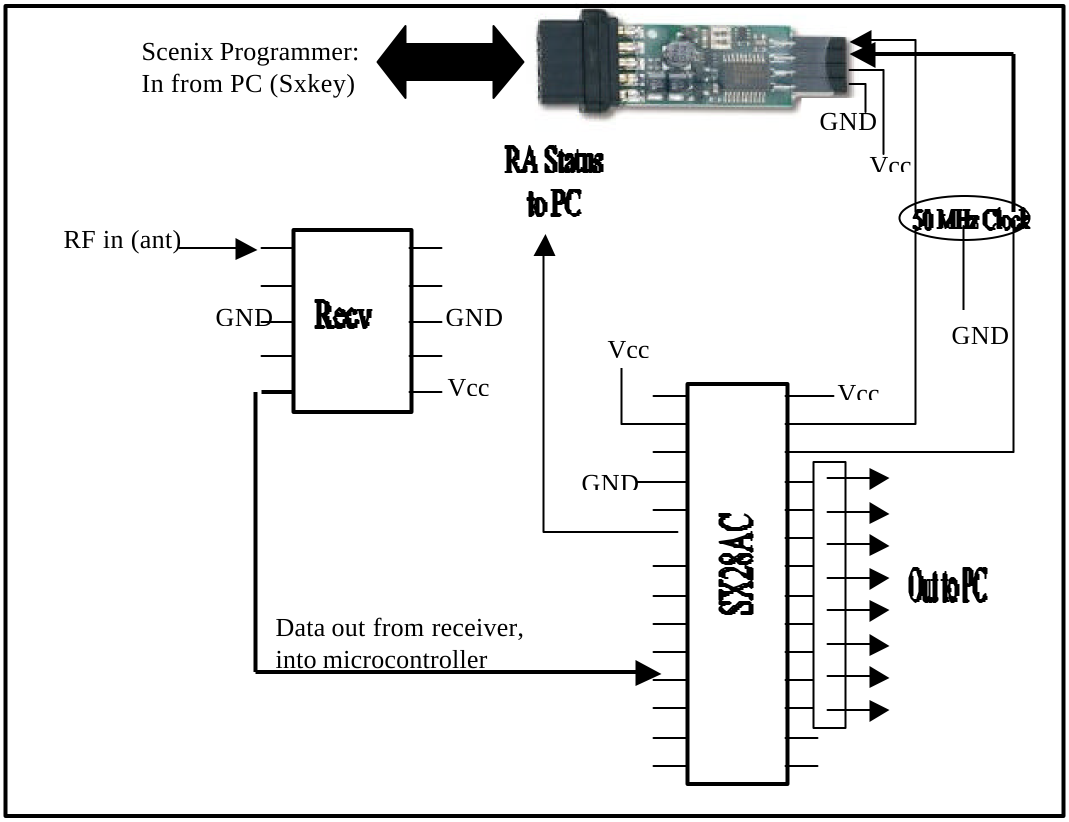


Figure 20: Reception Evaluation Board Schematic [21]



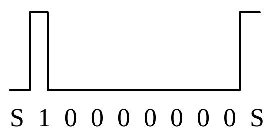
5.3 Setting Up Rs232 Communication for the Transmitter

As mentioned previously, to transmit over the RF link it is necessary to send the data serially. The main design problem I was faced with in setting up RS232 communication was programming the microcontroller. I received general RS232 assembly code from Grasp Lab graduate student Bret Victor which I studied and integrated with my data acquisition code. I considered three options when implementing the assembly code on the transmission end. One option was to create another interrupt within the acquisition code that would allow the serial transmission to occur. I found that this method was preferred among some engineers in the Grasp Lab, however, after unsuccessfully attempting to compile the assembly language several times, I decided that it would be more beneficial to try an option of less complexity.

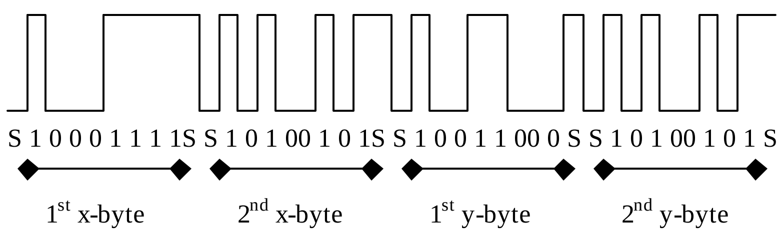
Another method involved performing the serial transmission within the main part of the code. This method was complex in that it required that an internal counter be built into the data acquisition interrupt routine to ensure that the bits were transmitted at the same rate. In other words, because interrupts occur while the main code is executing, some means must be implemented that will take account for the time that elapses while the interrupts are occurring. To avoid having to implement a counter I decided against using this method of RS232 communication.

The method I used was one less preferred among some of the engineers in Grasp Lab, however it was the most simple to implement. I decided to perform the serial transmission within the existing data acquisition interrupt. In this method the serial transmission begins immediately after an acceleration value is calculated within the data acquisition routine. The reason that this method is not the most preferred among engineers in the lab is because it requires that the microcontroller remain in the interrupt for a longer period of time than the other methods. For this application, however, the additional time spent within the interrupt is not a significant concern.

The code that I compiled performs the measurements of x and y acceleration in the same manner as before. Because the serial communication is performed directly after the calculation of an acceleration value, everything occurs within the interrupt routine, thus there is no need for a counter. The serial communication consists of the transmission of a start bit (logic low), a stop bit (logic high), and the eight bits of the byte being sent. If the number one were serially transmitted it would appear on an occilloscope as shown below:



If the x-zeroing value (42265) and the y-zeroing value (42489) are transmitted, the four byte serial output that would need to be sent over the RF link is: 1010010111111001 (x-accel) 1010010100011001 (y-accel). When coded using the above transmission method, the signal would appear as follows:

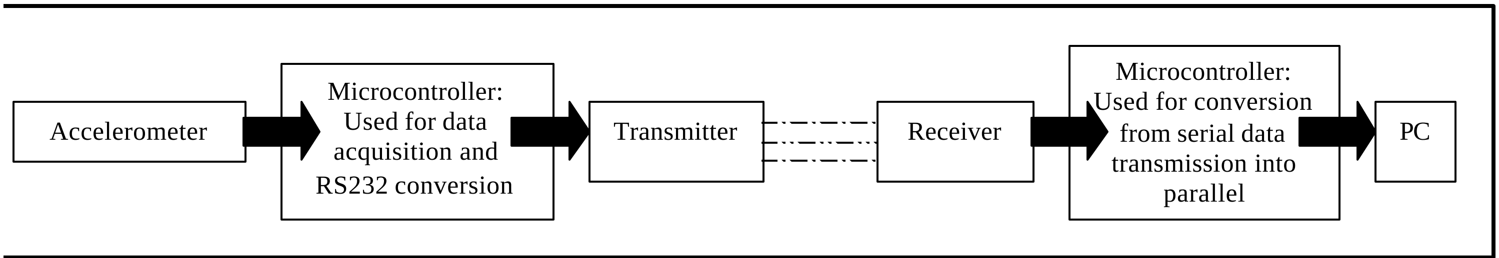


This transmission method was completed and simple tests were performed on an occsiloscope which confirmed that the output of the transmitter was as expected. Also, hooking the occsiloscope up to the receiver showed that the data was successfully being received without a significant amount of error. The transmission rate that I have implemented is approximately 1 bit per 250µs, equivalent to 4000 bits per second. This is well under the maximum bit per second rating of the transmitter and receiver modules. The completed transmission code can be found in Appendix(d).

5.4 Setting Up Rs232 Communication for the Receiver

It was also necessary to decide upon the method by which data would be interpreted at the receiving end. One possibility was to have the computer receive the data serially and interpret it in a program such as Hyper Terminal. This process was difficult because high level assembly language techniques were needed to enable the computer to output the data in decimal form. I decided to try to convert the serial data back into parallel and send it to the computer through the parallel port as before. Using this method it was possible to use the preliminary display mechanism that I created earlier in the project development. Figure 23 displays the basic serial communication setup:

Figure 21: Basic Serial Communication Setup



This portion of the data transmission scheme was not completed. The microcontroller was programmed to correctly convert the serial data into parallel data to be sent to the computer, see Appendix(e), however, it has yet to properly receive the data from the transmitter. The underlying problem with this is that the microcontroller at the receiving end has to recognize the start and stop bits and cycle through the data bytes at the proper frequency or the data will not be interpreted correctly. This portion of the accelerometer reception design must be investigated further in future work

6. CONCLUSIONS

The tri-axial accelerometer interface is in its intermediate stages of development. The filtering, zeroing, scaling and display methods are sufficient for trial experiments however, they may need to be upgraded when athletic collisions are applied to the system. The ADXL210 does not have a sufficient 'g'-range to be used in the final device, thus a different accelerometer, such as the Entran Devices' EGA series, should be experimented with. The calibration values I derived will not be correct for the new accelerometer, but the methods will most likely still apply.

The LC series transmitter and receiver modules seem to be of sufficient caliber for this application. Experimenting within the football and hockey environments will determine whether or not the transmitter can withstand the high impact forces resulting from the collision.

The transmission board is capable of sending RS232 encoded data via a Scenix microcontroller, across the RF link. The random noise associated with the transmission seems to be insignificant. The device has, however, been tested within a limited distance from the receiver, thus the noise could easily increase when the transmitter and receiver are installed within a sporting environment.

The receiver is capable of converting serial bytes into parallel bytes to be sent to the computer. It has yet to be properly linked to the transmitter, thus data can not be transmitted over the RF link. It is necessary to develop a method of receiver recognition capable of determining the presence of a start bit, a stop bit, and rotate through the information bytes at the proper sampling rate.

In summary, the accelerometer interface is project of many possibilities. It has numerous applications within the sporting environment which go beyond pure entertainment purposes. The information of this paper should be of use as further development takes place.

7. RECOMMENDATIONS

This accelerometer project is not at completion. Work should be done in the future to develop a transmission and reception scheme that properly sends the data bytes across the RF link. A trigger level should be established under which a transmission will not be sent to avoid transmitting unwanted minor impacts within the course of an event. Also, before applying this device to the athletic environment, a method must be implemented that will distinguish between the signals from different athletes. In other words, if two players are involved in a collision there must be some way to determine which impact signal came from player one's transmitter and which impact signal came from player two's transmitter. A possible solution to this problem is to have the transmissions occur at random intervals after which it could be inferred as to which player was involved in the impact. Another possibility is to implement a method of signaling to the receiver which athlete is transmitting at a given time. After the implementation has been completed in the lab it is extremely necessary to test the device in the athletic environment. There are many factors within that environment that cannot be recreated in the lab, but will most definitely play a role in the proper functioning of the device.

Accelerometer applications within impact monitoring systems are numerous. I believe the future of this device lies in the development of wireless interface system compatible with each of the applicable environments - a universal accelerometer interface requiring only minor alterations when switching from one application to the next. The system would ideally be as compact as this accelerometer design required, and it would be capable of evaluating a variety of 'g'-ranges depending on the desired application. The universal interface would have an on board calibration technique as well as a filtering mechanism sufficient for all environments.

8. ACKNOWLEDGEMENTS

I began this project because of my desire to pursue a combination of athletics and engineering in the future. Designing the tri-axial accelerometer interface provided me with the opportunity to integrate contact sports with the study of accelerometers, pulse width modulation, microcontroller assembly language, transmission, reception, and other relevant developing technologies.

I would like to thank professor Jim Ostrowski, my advisor, for his time commitment and support throughout the course of this project. He has provided many ideas that have helped me to progress in the areas of data acquisition and transmission. I would also like to thank two people involved in the University of Pennsylvania Grasp Lab: student Daniel Walker for tutoring me on the basics of the Scenix assembly language and for his help in the construction of the accelerometer evaluation board, and graduate student Bret Victor for his help with RS232 communication techniques. Lastly, I would like to thank the SUNFEST program and the NSF for providing me with the opportunity to become involved in this research and design experience.

9. REFERENCES

[1] http://www.patents.ibm.com/fcgi-bin/any2html
[2] http://www.analog.com/industry/iMEMS/mpd/products.html
[3] http://www.entran.com/egc.htm
[4] http://www.analog.com/pdf/ADXL202_10_b.pdf
[5] http://www.entran.com/egc.htm
[6] http://www.patents.ibm.com/fcgi-bin/any2html (pg 2)
[7] http://www.entran.com/pl/plausa.htm
[8] http://www.analog.com/pdf/ADXL202_10_b.pdf (pg6)
[9] http://www.analog.com/pdf/ADXL202_10_b.pdf (pg 2)
[10] http://www.analog.com/pdf/ADXL202_10_b.pdf (pg 7)
[11] http://www.analog.com/pdf/ADXL202_10_b.pdf (pg 7)
[12] http://www.analog.com/pdf/ADXL202_10_b.pdf (pg 8)
[13] http://www.analog.com/industry/iMEMS/products/Xl210eb.pdf
[14] http://www.analog.com/pdf/ADXL202_10_b.pdf
[15] http://www.analog.com/industry/iMEMS/products/Xl210eb.pdf,
http://www.scenix.com/tools/parallax.html#In system Programming tool: SX Blitz!
[16] http://www.linxtechnologies.com/f_modules.html
[17] http://www.linxtechnologies.com/f_modules.html
[18] http://www.linxtechnologies.com/f_modules.html
[19] http://www.linxtechnologies.com/f_modules.html
[20] http://www.analog.com/industry/iMEMS/products/Xl210eb.pdf,
http://www.scenix.com/tools/parallax.html#In system Programming tool: SX Blitz!
[21] http://www.analog.com/industry/iMEMS/products/Xl210eb.pdf,
http://www.scenix.com/tools/parallax.html#In system Programming tool: SX Blitz!

10. APPENDIX

Appendix (a): Initial Bi-Axial Data Acquisition Routine

```
; Dual Axis Data Aquisition Program
; Records X and Y axis data for bi-axial ADXL210 accelerometer.
; Created By Daniel Walker, edited by Heather Marandola


                        device SX28L,OSCXTMAX,TURBO,STACKX_OPTIONX,SYNC,CARRYX,BOR26

                        reset start

;DATA
                        org     $8                      ;Global Bank
pointer                 ds      1
x_time                  ds      2
y_time                  ds      2
angle                   ds      1
irangle                 ds      1
velocity        ds      1
                        org     $10                     ;Bank 0 (Int bank)
x_start                 ds      2
y_start                 ds      2
edge_sen        ds      1
clock                   ds      1
intover                 ds      1
ir                      ds      9
                        org     $30                     ;Bank 1
accconf                 ds      1
irconf                  ds      1
temp                    ds      1


INTERRUPT
                        org     0                       ;

                        mov     w,RTCC                  ; get the time from the counter
                        and     w,#$FC                  ; is the time < 3 ?
                        snz                             ; skip the next line if the above command did not return zero
                        jmp     :timer                  ; if the time is <= 3, jump to the 'timer'
                                                        ; otherwise...

                        bank    $0                      ; prep for RAM access
                        mov     w,RTCC                  ;
                        mov     intover,w               ;
                        stc                             ;
                        sub     intover,#9              ;

                        mov     w,#0
                        mov     m,#$09                  ; WKPND_B (access the wakeup pending bit register)
                                                        ;           read the status of pins, 1=edge has occured
                                                        ;           0=no valid edge has occured
                        mov     !rb,w                   ; WKPND_B reg (initializing the status to no edge)
                        mov     m,#$0A                  ; edge select : WKED_B (access wakeup edge register)
                                                        ;           selecting to detect rising (0) or falling (1) edges
                        and     w,#3                    ; which pin caused the interrupt?
                        clc                             ;
                        add     PC,w                    ; jump to the proper pin
                        reti                            ; if neither pin rb.0 or rb.1 caused the interrupt then ignore
                        jmp     :x_acc
                        jmp     :y_acc
:timer
                        inc     clock                   ; if RTCC was <=3 above, then increment the clock
                        reti                            ;
:x_acc

                        snb     edge_sen.0              ; if the rising edge has been detected then go find the falling edge
                        jmp     :x_acc_fall             ;
                        mov     w,RTCC                  ; this is when the rising edge occurred
                        mov     x_start,w               ; put it in a variable (lower 8 bits)

                        mov     w,clock                 ; going to get the upper 8 bits now
                        mov     x_start+1,w             ;
                        setb    edge_sen.0              ; we've got the rising edge, set the flag to go for the falling edge
                        mov     w,edge_sen              ;
                        mov     !rb,w                   ; change the mode for that pin to look for a falling edge
```

```
                mov     w,#250                  ; dealing with intover and the clock
                stc                             ;
                mov     w,RTCC-w                ;
                stc                             ;
                sub     intover,w               ;
                sc                              ;
                reti                            ;
                inc     clock                   ;
                reti                            ;

:x_acc_fall  mov  w,x_start                     ; go get the time of the rising edge
                stc                             ; set carry
                mov     w,#'A';RTCC-w           ; find the width of the pulse
                mov     x_time,w                ; put it in a variable (lower 8 bits)
                mov     w,x_start+1             ; going to get the upper 8 bits now
                mov     w,#'A';clock-w          ;
                mov     x_time+1,w              ;
                clrb    edge_sen.0              ; we've got the falling edge so reset the pin to its original mode
                mov     w,edge_sen              ;
                mov     !rb,w                   ;

                mov     w,#250                  ; dealing with intover and the clock again
                stc                             ;
                mov     w,RTCC-w                ;
                stc                             ;
                sub     intover,w               ;
                snc                             ;
                inc     clock                   ;


:y_acc
                snb     edge_sen.1
                jmp     :y_acc_fall
                mov     w,RTCC
                mov     y_start,w
                mov     w,clock
                mov     y_start+1,w
                setb    edge_sen.1
                mov     w,edge_sen
                mov     !rb,w

                mov     w,#250
                stc
                mov     w,RTCC-w
                stc
                sub     intover,w
                sc
                reti
                inc     clock
                reti


:y_acc_fall  mov  w,y_start
                stc                             ;set carry
                mov     w,RTCC-w
                mov     y_time,w
                mov     w,y_start+1
                mov     w,clock-w
                mov     y_time+1,w
                clrb    edge_sen.1
                mov     w,edge_sen
                mov     !rb,w

                mov     w,#250
                stc
                mov     w,RTCC-w
                stc
                sub     intover,w
                sc
                reti
                inc     clock
                reti

;MAIN PROGRAM

start
                mov     w,#%10001000            ;init
                mov     !OPTION,w
                mov     w,#$0
```

```
        mov     !rc,w
        mov     w,#$FF
        mov     !ra,w
        mov     w,#$07
        mov     !rb,w

        mov     m,#$0a
        mov     w,#0
        mov     !rb,w
        mov     w,#$9
        mov     m,w
        mov     w,#0
        mov     !rb,w
        mov     w,#%11111100
        mov     m,#$0B
        mov     !rb,w                   ;WKEN_B reg
        mov     edge_sen,#$0

        mov     rc,#0
        clrb    rb.7

main                                    ; the real deal


:byte1
        mov     w,y_time                ; go get y-time
        mov     rc,w                    ; move w into rc port

        clrb    rb.7                    ; tell computer that pic is ready to send data

        sb      ra.0                    ; data has been sent
        jmp     :byte1                  ; if it does not have the data then go back to byte1

:byte2
        mov     w,y_time+1              ; if byte is sent, move y_time+1 into w
        mov     rc,w                    ; move w to rc port

        setb    rb.7                    ;       ...etc. just like the above for the rest of the code

        snb     ra.0
        jmp     :byte2

:byte3
        mov     w,x_time
        mov     rc,w

        clrb    rb.7

        sb      ra.0
        jmp     :byte3

:byte4
        mov     w,x_time+1
        mov     rc,w

        setb    rb.7

        snb     ra.0
        jmp     :byte4

        jmp     main
```

The following is the data acquisition code on the computer terminal end:

```
/* Created by Daniel Walker, edited by Heather Marandola*/

#include <stdio.h>

main()
{
 char keep;
 unsigned short int buf;
 unsigned short int buf2;
 unsigned long x=0;
```

```
FILE *fp;

keep=inp(0x37A)&0xD0;
if ( (fp = fopen("bang67_2","w"))== NULL)
   printf ("error opening test");

while(!kbhit()) {
outp(0x37A,keep|33);
while(inp(0x379)&128==0);
buf=inp(0x378);
outp(0x37A,keep|32);
while((unsigned char)(inp(0x379)&128)!=0);
buf|=inp(0x378)<<8;
outp(0x37A,keep|33);
while(inp(0x379)&128==0);
buf2=inp(0x378);
outp(0x37A,keep|32);
while((unsigned char)(inp(0x379)&128)!=0);
buf2|=inp(0x378)<<8;

printf("%u   %u \n",buf,buf2);
fprintf(fp,"%u\t%u\n",buf,buf2);

delay(1);
}
fclose(fp);
}
```

Appendix (b): Matlab Code For Median Filter Implementation

```
% Created By Heather Marandola

load neutral                                          % Go and get the file

tx = 1:length(neutral);                               % Set up a sample number matrix
x = neutral(1:length(neutral));                       % Set up a x-acceleration matrix
                                                      % Set up a y-acceleration matrix
x_y = neutral(length(neutral)+1:length(neutral)*2);

counter = 0;                                          % Initialize the counter
j = 1;                                                % Initialize the index
k = 0;                                                % Initialize the second counter
p = 15;                                               % Select the number of points
l = floor(length(neutral)/p)*p;                       % Set up the number of times to
                                                      %       sample
for i = 1:l                                           % Median Filtering Routine…
  x(i) = x(i);
 y(i) = x_y(i);
 counter = counter + 1;
 if counter > p;
   medx(j) = median(x(1+p*k:p+p*k));
   medy(j) = median(y(1+p*k:p+p*k));
  j = j+1;
  k = k+1;
  counter = 0;
 end

end
```

Appendix (c): Data Acquisition Routine At Computer End With Filtering, Scaling and Display Mechanism Included.

```
/* Created by Heather Marandola*/
#include <stdio.h>
#include <fcntl.h>

main()
{
 char keep;
 unsigned short int buf;
 unsigned short int buf2;
 unsigned long x=0;
 unsigned short int xz;
 unsigned short int yz;
 float bnew1;
 float bnew2;
 float max;
 float data;
 float abnew1;
 float abnew2;
 int counter;
 float arrayx[15];
 float arrayy[15];
 FILE *fp;

 xz = 42300;
 yz = 41765;  /*42127;*/
 max = 0;

 keep=inp(0x37A)&0xD0;

 if ( (fp = fopen("test14","w"))== NULL)
    printf ("error opening test");
 while(!kbhit()) {

 counter = 0;

loop:
 if(counter<15){

  outp(0x37A,keep|33);
  while(inp(0x379)&128==0);
  buf=inp(0x378);
  outp(0x37A,keep|32);
  while((unsigned char)(inp(0x379)&128)!=0);
  buf|=inp(0x378)<<8;
  outp(0x37A,keep|33);
  while(inp(0x379)&128==0);
  buf2=inp(0x378);
  outp(0x37A,keep|32);
  while((unsigned char)(inp(0x379)&128)!=0);
  buf2|=inp(0x378)<<8;
  arrayx[counter] = buf;
  arrayy[counter] = buf2;
  counter = counter+1;
  goto loop;
  }
 else
  qsort(arrayx,0,15);
  qsort(arrayy,0,15);
  bnew1 = (arrayx[7] - xz)/3666;
  bnew2 = (arrayy[7] - yz)/3618;
  fprintf(fp,"%lf\t%lf\n",bnew1,bnew2);

  /* Creating a Display */

 if(bnew1<0){
 bnew1 = -bnew1;
 }
 if(bnew2<0){
 bnew2 = -bnew2;
 }
 if(bnew1>bnew2){
 data = bnew1;
 goto maximum;
 }
 else{
```

```
 data = bnew2;
 goto maximum;
 }

 /* finding a maximum over the time interval*/
maximum:
 if(data>max)
  max = data;

/*Creating a display*/

 if(data<.25){
 printf("0\n");
 goto bottom;
 }
 if(data<.75){
 printf("0*\n");
 goto bottom;
 }
 if(data<1.25){
 printf("0**\n");
 goto bottom;
 }
 if(data<1.75){
 printf("0***\n");
 goto bottom;
 }
 if(data<2.25){
 printf("0****\n");
 goto bottom;
 }
 if(data<2.75){
 printf("0*****\n");
 goto bottom;
 }
 if(data<3.25){
 printf("0******\n");
 goto bottom;
 }
 if(data<3.75){
 printf("0*******\n");
 goto bottom;
 }
 if(data<4.25){
 printf("0********\n");
 goto bottom;
 }
 if(data<4.75){
 printf("0*********\n");
 goto bottom;
 }
 if(data<5.25){
 printf("0**********\n");
 goto bottom;
 }
 if(data<5.75){
 printf("0***********\n");
 goto bottom;
 }
 if(data<6.25){
 printf("0************\n");
 goto bottom;
 }
 if(data<6.75){
 printf("0*************\n");
 goto bottom;
 }
 if(data<7.25){
 printf("0**************\n");
 goto bottom;
 }
 if(data<7.75){
 printf("0***************\n");
 goto bottom;
 }
```

```
if(data<8.25){
printf("0*****************\n");
goto bottom;
}
if(data<8.75){
printf("0******************\n");
goto bottom;
}
if(data<9.25){
printf("0*******************\n");
goto bottom;
}
if(data<9.75){
printf("0********************\n");
goto bottom;
}
if(data<10.25){
printf("0*********************\n");
goto bottom;
}
if(data>=10.25){
printf("overload\n");
goto bottom;
}
bottom:

/* printf("%lf\t%lf\n",bnew1,bnew2);*/
 delay(10);

}
fclose(fp);
printf("..............................................\n");
printf("The Maximum Acceleration In G's Was: %lf\n",max);
}
```

Appendix (d): Microcontroller Code for RS232 Transmission

```
; Dual Axis Data Aquisition Program
; Records X and Y axis data for bi-axial ADXL210 accelerometer.


                    device SX28L,OSCXTMAX,TURBO,STACKX_OPTIONX,SYNC,CARRYX,BOR26

                    reset      start
                    freq       50000000
;.........................................................................
; PORT MAPPING

serial_Tx     equu  rb.3                                    ; Transmitting Data Port
;.........................................................................

;DATA
                    org        $8                           ;Global Bank
x_time              ds         2
y_time              ds         2

                    org        $10                          ;Bank 0 (Int bank)
Bank_start =$

x_start             ds         2
y_start             ds         2
edge_sen            ds         1
clock               ds         1
intover             ds         1
ir                  ds         9

                    org        $30                          ;Bank 1
Bank_Transmit=$

stall_one           ds         1
stall_two           ds         1
stall_three         ds         1

serial_status       ds         1                            ; Serial status bits
serial_txbyte       ds         1

serial_txcount      ds         1                            ; How many bits we have left to transmit

serialTxActive      equ        serial_status.0              ; true if a byte is being transmitted


;INTERRUPT
                    org        0                            ;

                    mov        w,RTCC                       ; get the time from the counter
                    and        w,#$FC                       ; is the time < 3 ?
                    snz                                     ; skip the next line if the above command did not return zero
                    jmp        :timer                       ; if the time is <= 3, jump to the 'timer'
                                                            ; otherwise...

                    bank       Bank_Start                   ; prep for RAM access
                    mov        w,RTCC                       ;
                    mov        intover,w                    ;
                    stc                                     ;
                    sub        intover,#9                   ;

                    mov        w,#0
                    mov        m,#$09                       ; WKPND_B (access the wakeup pending bit register)
                                                            ;          read the status of pins, 1=edge has occured
                                                            ;          0=no valid edge has occured
                    mov        !rb,w                        ; WKPND_B reg (initializing the status to no edge)
                    mov        m,#$0A                       ; edge select : WKED_B (access wakeup edge register)
                                                            ;          selecting to detect rising (0) or falling (1) edges
                    and        w,#3                         ; which pin caused the interrupt?
                    clc                                     ;
                    add        PC,w                         ; jump to the proper pin
                    reti                                    ; if neither pin rb.0 or rb.1 caused the interrupt then ignore
                    jmp        :x_acc
                    jmp        :y_acc
:timer
                    inc        clock                        ; if RTCC was <=3 above, then increment the clock
                    reti                                    ;
:x_acc
```

```
                    snb     edge_sen.0              ; if the rising edge has been detected then go find the falling edge
                    jmp     :x_acc_fall             ;
                    mov     w,RTCC                  ; whis is when the rising edge occured
                    mov     x_start,w               ; put it in a variable (lower 8 bits)

                    mov     w,clock                 ; going to get the upper 8 bits now
                    mov     x_start+1,w             ;
                    setb    edge_sen.0              ; we've got the rising edge, set the flag to go for the falling edge
                    mov     w,edge_sen              ;
                    mov     !rb,w                   ; change the mode for that pin to look for a falling edge

                    mov     w,#250                  ; dealing with intover and the clock
                    stc                             ;
                    mov     w,RTCC-w                ;
                    stc                             ;
                    sub     intover,w               ;
                    sc                              ;
                    reti                            ;
                    inc     clock                   ;
                    reti                            ;

:x_acc_fall mov     w,x_start                       ; go get the time of the rising edge
                    stc                             ; set carry
                    mov     w,#1;RTCC-w             ; find the width of the pulse
                    mov     x_time,w                ; put it in a variable (lower 8 bits)
                    mov     w,x_start+1             ; going to get the upper 8 bits now
                    mov     w,#2;clock-w            ;
                    mov     x_time+1,w              ;
                    clrb    edge_sen.0              ; we've got the falling edge so reset the pin to its original mode
                    mov     w,edge_sen              ;
                    mov     !rb,w                   ;

                    mov     w,#250                  ; dealing with intover and the clock again
                    stc                             ;
                    mov     w,RTCC-w                ;
                    stc                             ;
                    sub     intover,w               ;
                    snc                             ;
                    inc     clock                   ;

            bank    Bank_Transmit

;...................................
; Sending the first x-accel byte
;...................................

:byte1                                              ; stall loop

            mov     stall_one,#25

:loop_1
            mov     stall_two,#21
:loop2_1
            mov     stall_three,#4
:loop3_1
            dec     stall_three
            sz
            jmp     :loop3_1

            dec     stall_two
            sz
            jmp     :loop2_1

            dec     stall_one
            sz
            jmp     :loop_1


            snb     serialTxActive                  ; are we already xmitting a byte?
            jmp     :sending_1                      ; if so, continue with that

:sendfirstbit_1
            mov     w,x_time;serial_one
            mov     serial_txbyte, w                ; put it into the variable
            setb    serialTxActive                  ; set the flag that says we're xmitting
            mov     w, #9                           ; nine more bits to go
            mov     serial_txcount, w
            clrb    serial_Tx                       ; put start bit on pin
            jmp     :sendend_1                      ; all done
```

```
:sending_1
		decsz	serial_txcount			; decrement bit counter
		jmp	:sendbit_1			; was that the last data bit?

:sendlastbit_1

		clrb	serialTxActive			; clear the flag
		setb	serial_Tx			; put stop bit on pin
;		call	delay
		jmp	:end_1				; we outtee

:sendbit_1
		rr	serial_txbyte			; put next bit into carry
		sc					; is it 0?
		clrb	serial_Tx			; if so, lower the pin
		snc					; is it 1?
		setb	serial_Tx			; if so, raise the pin
							; that's all for that
:sendend_1

		jmp	:byte1
:end_1
		jmp	:byte2

;........................................
; Sending the second x-accel byte
;........................................

:byte2
		mov	stall_one,#25

:loop_2
		mov	stall_two,#21
:loop2_2
		mov	stall_three,#4
:loop3_2
		dec	stall_three
		sz
		jmp	:loop3_2

		dec	stall_two
		sz
		jmp	:loop2_2

		dec	stall_one
		sz
		jmp	:loop_2


		snb	serialTxActive			; are we already xmitting a byte?
		jmp	:sending_2			; if so, continue with that

:sendfirstbit_2
		mov	w,x_time+1
		mov	serial_txbyte, w		; put it into the variable
		setb	serialTxActive			; set the flag that says we're xmitting
		mov	w, #9				; nine more bits to go
		mov	serial_txcount, w
		clrb	serial_Tx		; put start bit on pin
		jmp	:sendend_2		; all done

:sending_2
		decsz	serial_txcount		; decrement bit counter
		jmp	:sendbit_2		; was that the last data bit?

:sendlastbit_2

		clrb	serialTxActive		; clear the flag
		setb	serial_Tx		; put stop bit on pin
;		call	delay
		jmp	:end_2			; done

:sendbit_2
		rr	serial_txbyte		; put next bit into carry
		sc				; is it 0?
		clrb	serial_Tx		; if so, lower the pin
		snc				; is it 1?
		setb	serial_Tx		; if so, raise the pin
						; that's all for that
:sendend_2
```

```
        jmp     :byte2
:end_2

        reti

;...............................................
; The y-acceleration interrupt
;...............................................

:y_acc

                snb     edge_sen.1
                jmp     :y_acc_fall
                mov     w,RTCC
                mov     y_start,w
                mov     w,clock
                mov     y_start+1,w
                setb    edge_sen.1
                mov     w,edge_sen
                mov     !rb,w

                mov     w,#250
                stc
                mov     w,RTCC-w
                stc
                sub     intover,w
                sc
                reti
                inc     clock
                reti


:y_acc_fall mov w,y_start
                stc
                mov     w,#3;RTCC-w
                mov     y_time,w
                mov     w,y_start+1
                mov     w,#4;clock-w
                mov     y_time+1,w
                clrb    edge_sen.1
                mov     w,edge_sen
                mov     !rb,w


                mov     w,#250
                stc
                mov     w,RTCC-w
                stc
                sub     intover,w
                snc
                inc     clock

        bank    Bank_Transmit
;......................................
; Sending the first y-accel byte
;......................................

:byte3
        mov     stall_one,#25

:loop_3
        mov     stall_two,#21
:loop2_3
        mov     stall_three,#4
:loop3_3
        dec     stall_three
        sz
        jmp     :loop3_3

        dec     stall_two
        sz
        jmp     :loop2_3

        dec     stall_one
        sz
        jmp     :loop_3


        snb     serialTxActive                  ; are we already xmitting a byte?
        jmp     :sending_3                      ; if so, continue with that
```

```
:sendfirstbit_3
        mov     w,y_time
        mov     serial_txbyte, w                ; put it into the variable
        setb    serialTxActive                  ; set the flag that says we're xmitting
        mov     w, #9                           ; nine more bits to go
        mov     serial_txcount, w
        clrb    serial_Tx                       ; put start bit on pin
        jmp     :sendend_3                      ; all done

:sending_3
        decsz   serial_txcount                  ; decrement bit counter
        jmp     :sendbit_3                      ; was that the last data bit?

:sendlastbit_3

        clrb    serialTxActive                  ; clear the flag
        setb    serial_Tx                       ; put stop bit on pin
;       call    delay
        jmp     :end_3                          ; done

:sendbit_3
        rr      serial_txbyte                   ; put next bit into carry
        sc                                      ; is it 0?
        clrb    serial_Tx                       ; is it 1?
        snc
        setb    serial_Tx                       ; if so, raise the pin
                                                ; that's all for that
:sendend_3
        jmp     :byte3
:end_3
        jmp     :byte4

;......................................
; Sending the second y-accel byte
;......................................

:byte4
        mov     stall_one,#25

:loop_4
        mov     stall_two,#21
:loop2_4
        mov     stall_three,#4
:loop3_4
        dec     stall_three
        sz
        jmp     :loop3_4

        dec     stall_two
        sz
        jmp     :loop2_4

        dec     stall_one
        sz
        jmp     :loop_4


        snb     serialTxActive                  ; are we already xmitting a byte?
        jmp     :sending_4              ; if so, continue with that

:sendfirstbit_4
        mov     w,y_time+1
        mov     serial_txbyte, w        ; put it into the variable
        setb    serialTxActive          ; set the flag that says we're xmitting
        mov     w, #9                   ; nine more bits to go
        mov     serial_txcount, w
        clrb    serial_Tx               ; put start bit on pin
        jmp     :sendend_4              ; all done

:sending_4
        decsz   serial_txcount          ; decrement bit counter
        jmp     :sendbit_4              ; was that the last data bit?

:sendlastbit_4

        clrb    serialTxActive          ; clear the flag
        setb    serial_Tx               ; put stop bit on pin
;       call    delay
        jmp     :end_4                  ; we outtee
```

```
:sendbit_4
        rr      serial_txbyte           ; put next bit into carry
        sc                              ; is it 0?
        clrb    serial_Tx               ; if so, lower the pin
        snc                             ; is it 1?
        setb    serial_Tx               ; if so, raise the pin
                                        ; that's all for that
:sendend_4

        jmp     :byte4

:end_4
        reti


;MAIN PROGRAM

Delay
        bank    Bank_Transmit
        mov     stall_one,#5

:loop_3
        mov     stall_two,#10
:loop2_3
        mov     stall_three,#10
:loop3_3
        dec     stall_three
        sz
        jmp     :loop3_3

        dec     stall_two
        sz
        jmp     :loop2_3

        dec     stall_one
        sz
        jmp     :loop_3
        ret


start
                mov     w,#%10001000            ;init
                mov     !OPTION,w
                mov     m,#$0F
                mov     w,#$0
                mov     !rc,w
                mov     w,#$0
                mov     !ra,w
                mov     w,#%00000011
                mov     !rb,w

                mov     m,#$0a
                mov     w,#0
                mov     !rb,w
                mov     w,#$9
                mov     m,w
                mov     w,#0
                mov     !rb,w
                mov     w,#%11111000
                mov     m,#$0B
                mov     !rb,w                   ;WKEN_B reg
                mov     edge_sen,#$0
```

Appendix (e): Preliminary Microcontroller Code for RS232 Reception and Conversion into Parallel

```
                    device SX28L,OSCXTMAX,TURBO,STACKX_OPTIONX,SYNC,CARRYX,BOR26
                    reset start

                    freq          50000000

;.............
; Port Mapping
;............................................................
serial_Rx     equ          rb.1                              ; the receiving port
;............................................................


;.........
; Globals
;.........................
              org          $8

var1                ds            1
var2                ds            1
var3                ds            1
var4                ds            1
serial_status       ds            1
;.........................

;..................
; Needed Variables
;.........................
              org          $10

Bank_Var = $

stall_one     ds           1
stall_two     ds           1
stall_three   ds           1

bit_count     ds           1
serial_byte   ds           1
;.........................

;.........
; Equates
;.........
Active        equ          serial_status.0

;....................................................
;------------------------------------------------------

                    org           0
;MAIN PROGRAM

start
                    mov           w,#%10001000                       ;init
                    mov           !OPTION,w

                    setb          serial_Rx
                    mov           w,#$0F
                    mov           m,w
                    mov           w,#$0
                    mov           !rc,w
                    mov           w,#$FF
                    mov           !ra,w
                    mov           w,#%00000010
                    mov           !rb,w

                    mov           rc,#0
                    clrb          rb.7
                    clr           serial_status

main                                                                 ; the real deal

bank          bank_Var
;........................................
; Getting the bytes from the transmitter
;........................................
```

```
;......................
; Getting the 1st Byte
;......................

:begin_one
                snb     serial_Rx
                stc

                snb     Active
                jmp     :recving_one

                snb     serial_Rx
                jmp     :begin_one

                mov     w,#9
                mov     bit_count,w
                setb    Active
                call    Delay_one
                jmp     :pause_one

:recving_one
                decsz   bit_count
                jmp     :getbit_one

:recvlast_one
                clrb    Active
                mov     w,serial_byte
                mov     var1,w
                jmp     :end_one

:getbit_one
                rr      serial_byte
                call    Delay_two
:pause_one
                jmp     :begin_one
:end_one

;......................
; Getting the 2nd Byte
;......................

:begin_two
                snb     serial_Rx
                stc

                snb     Active
                jmp     :recving_two

                snb     serial_Rx
                jmp     :begin_two

                mov     w,#9
                mov     bit_count,w
                setb    Active

                call    Delay_one
                jmp     :pause_two

:recving_two
                decsz   bit_count
                jmp     :getbit_two

:recvlast_two
                clrb    Active
                mov     w,serial_byte
                mov     var2,w
                jmp     :end_two

:getbit_two
                rr      serial_byte
                call    Delay_two
:pause_two
                jmp     :begin_two
:end_two

;......................
; Getting the 3rd Byte
;......................

:begin_three
```

```
                snb     serial_Rx
                stc

                snb     Active
                jmp     :recving_three

                snb     serial_Rx
                jmp     :begin_three

                mov     w,#9
                mov     bit_count,w
                setb    Active

                call    Delay_one
                jmp     :pause_three

:recving_three
                decsz   bit_count
                jmp     :getbit_three

:recvlast_three
                clrb    Active
                mov     w,serial_byte
                mov     var3,w
                jmp     :end_three

:getbit_three
                rr      serial_byte
                call    Delay_two
:pause_three
                jmp     :begin_three
:end_three

;......................
; Getting the 4th Byte
;......................

:begin_four
                snb     serial_Rx
                stc

                snb     Active
                jmp     :recving_four

                snb     serial_Rx
                jmp     :begin_four

                mov     w,#9
                mov     bit_count,w
                setb    Active

                call    Delay_one
                jmp     :pause_four

:recving_four
                decsz   bit_count
                jmp     :getbit_four

:recvlast_four
                clrb    Active
                mov     w,serial_byte
                mov     var4,w
                jmp     :end_four

:getbit_four
                rr      serial_byte
                call    Delay_two
:pause_four
                jmp     :begin_four
:end_four

;.............................................
; Putting the bytes through the parallel port
;.............................................

:byte2
                mov     w,var2
                mov     rc,w

                setb    rb.7
```

```
                snb     ra.0
                jmp     :byte2

:byte3
                mov     w,var3
                mov     rc,w

                clrb    rb.7

                sb      ra.0
                jmp     :byte3

:byte4
                mov     w,var4
                mov     rc,w

                setb    rb.7

                snb     ra.0
                jmp     :byte4

:byte1
                mov     w,var1                  ; move value of y_time into w
                mov     rc,w                    ; move w into rc port

                clrb    rb.7                    ; tell computer that pic is ready to send data

                sb      ra.0                    ; data has been sent
                jmp     :byte1                  ; if it does not have the data then go back to byte1
                jmp     main

;................
; Delay Routines
;................

Delay_one
        mov     stall_one,#25

:loop_3
        mov     stall_two,#21
:loop2_3
        mov     stall_three,#6
:loop3_3
        dec     stall_three
        sz
        jmp     :loop3_3

        dec     stall_two
        sz
        jmp     :loop2_3

        dec     stall_one
        sz
        jmp     :loop_3
        ret


Delay_two
        mov     stall_one,#10

:loop_3
        mov     stall_two,#5
:loop2_3
        mov     stall_three,#50
:loop3_3
        dec     stall_three
        sz
        jmp     :loop3_3

        dec     stall_two
        sz
        jmp     :loop2_3

        dec     stall_one
        sz
        jmp     :loop_3
        ret

Delay_start
```

```
        mov     stall_one,#25

:loop_3
        mov     stall_two,#21
:loop2_3
        mov     stall_three,#50
:loop3_3
        dec     stall_three
        sz
        jmp     :loop3_3

        dec     stall_two
        sz
        jmp     :loop2_3

        dec     stall_one
        sz
        jmp     :loop_3
        ret
```