FLIGHT DATA RECORDER FOR THE AMERICAN FOOTBALL

By

Chris J. Nowak

A project submitted to the
Faculty of the Graduate School of
State University of New York at Buffalo in partial
fulfillment of the requirements for the degree of
Master of Engineering

May 25th, 2003

Table of Contents:

Abstract:

American Football is a game enjoyed both by participants and spectators, young and old. The game is played on a field 100 yards long by 30 yards wide, with an inflatable ball that is somewhat oval in shape with a point on each end. When properly thrown, the ball spins as it travels, not unlike a bullet fired from a rifled gun barrel. As the ball flies along its trajectory its spin begins to degrade and it exhibits a wobble. It's also been observed that the ball follows a curved path to its target as opposed to a simple ballistic trajectory.

The goal of this project is to develop an electronic data collection system that can be mounted inside a football to measure, store, and download several seconds of acceleration data that are sampled while the ball is "in play". The data recorders that I will design will each be a small self contained unit with 2 dual axis accelerometers, microprocessor, and 4 memory devices. The entire project will require 2 of these devices to monitor accelerations in different areas within the football. The electronics are all mounted on professionally printed circuit boards, powered by a battery, and containing enough electronic memory to store the data. A total of seven channels of acceleration data will be recorded at a rate of 200 samples per second. Data will be continually stored for at least 10 seconds from the point that the recording is initiated. The "ball" used for this project will have the familiar shape and appearance of a football, but will be constructed of foam rubber (a "Nerf" football) and a zipper for ease of access to the electronics for service, repair, battery replacement, or future system upgrades. Efforts will be made to keep the weight of the unit within 2 ounces of regulation, and that the ball will be approximately balanced.

Background:

This project follows the research done in 2001 for SUNY at Buffalo by David Francis and Narayana Sundaram about mounting an accelerometer inside a football. They proceeded to mount a singe tri-axial accelerometer at the center of the ball, and broadcast data from this accelerometer while in flight to a receiver connected to a desktop PC. [1]

Using accelerometers to measure and record data about some sort of event in a moving body is not new, crash test dummies can have 30 or more accelerometers in them to measure forces on the different body parts in an automotive crash test. There are products on the market available to race car owners that will monitor vehicle performance based on accelerations in each of the 3 principle axes. Recent breakthroughs in accelerometer design have allowed accelerometers, which were once fairly large, high precision, expensive instruments, to become much smaller and less expensive. The accelerometers chosen for this project are actually micro machined semiconductors, in an integrated circuit package, that can be soldered directly onto a printed circuit board. Some accelerometers also contain multiple axes of sensitivity, as well as a broad range of acceleration sensing ability, with their maximum range being as little as 1 or 2 G's, up to the thousands of G's.

The accelerometers chosen for this project will be 50 G's in sensitivity. They are based on micromachining a differential capacitor into the actual die of the chip. Figure 1 at the top of the next page shows the basic theory behind the accelerometer technology. Micromachining, or the manufacture of tiny moving mechanical structures, is considered to be one of the most significant new technologies since genetic engineering. Funded by governments, universities, and corporations worldwide, researchers have actually built motors smaller than the size of a pinhead. They have developed complex manipulators capable of grasping a single red blood cell and demonstrated prototype micromachines, ranging from gear trains to microscopic steam engines - notions once reserved for science fiction writers. [2]

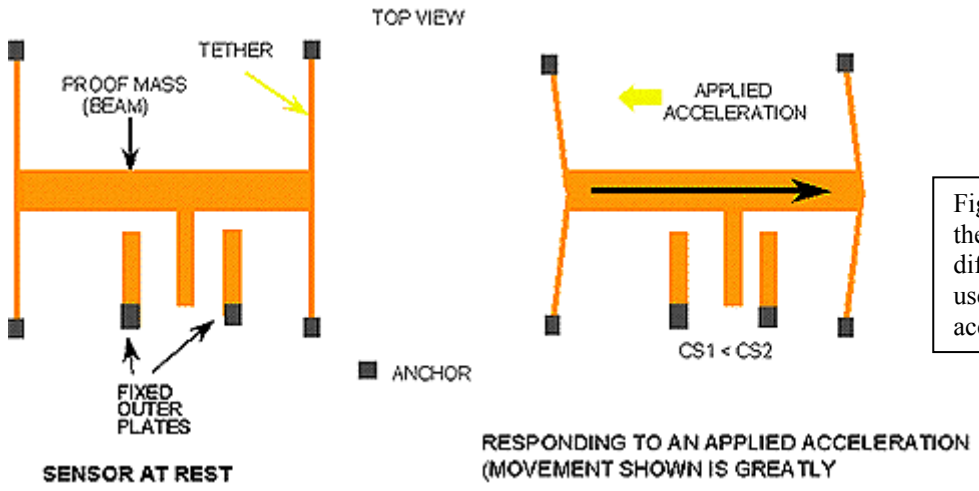


Figure 1 – The basic theory on how a differential capacitor is used to measure acceleration [3]

Surface micromachining builds layers of material on top of a silicon wafer and then selectively etches material away to make sensor structures, as opposed to machining material out of the wafer itself (Figure 2). [4]

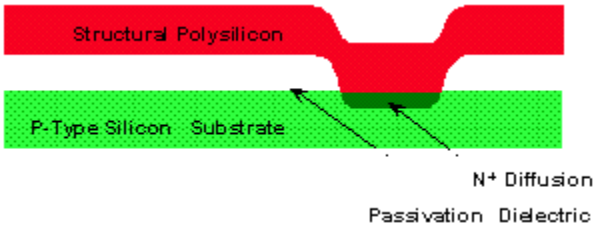


Figure 2 – Side view of iMEMS surface micromachined structure. [5]

Because it is easy to electrically isolate the different structural components, most implementations use a capacitive sensing technique to measure the movement of the mass due to acceleration. This is a significant advantage as the capacitors drift little over temperature and therefore temperature compensation circuitry is minimal. [6]

Analog Devices uses an integrated approach that combines the sensor and the electronics on a single chip. They call this process is called iMEMS, or integrated-MEMS to highlight the more integrated nature of this approach. The advantages of the surface approach are versatility in design, the ability to build a two axis sensor on a single chip, and a built-in capacitive sensing technique that is superior to the piezoresistive approach. The disadvantage is that the sensor structures are much smaller, and have lower mass. Deflections of the structures are minute, generating capacitive changes at the attofarad

level; the ability of the electronics to sense these small changes sets the resolution limit of the device. [7]

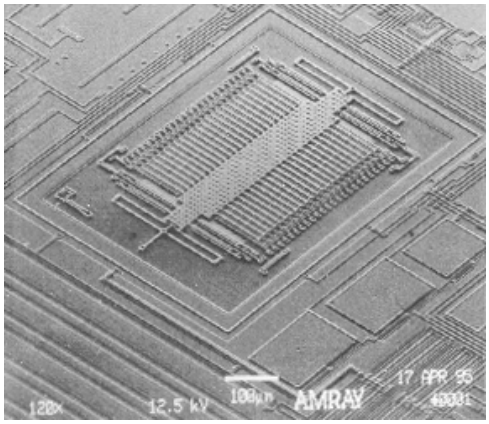


Figure 3 – iMEMS Accelerometer showing structural beam and surrounding electronics [8]

Figure 3 (above) shows a photograph taken of an actual sensing element on an accelerometer, and Figure 4 (below) shows the sensing elements in place on the die of the ADXL250. [9]

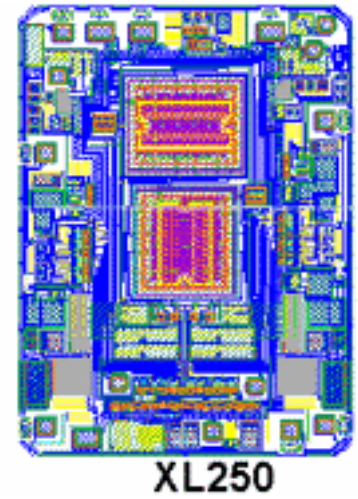


Figure 4 – Sensing Elements (pink areas) in place on the die of the ADXL250 [10]

The Calspan Corporation, now part of Veridian, has used these small inexpensive accelerometers in systems that monitor driving behavior in automobiles, as well as in systems designed to automatically detect if the vehicle has been in an accident and assistance is required. Other uses for these small accelerometers include vibration measurement, tilt sensing, and also as a replacement for a shock switch, which is a sensor that can determine if a package was handled roughly during shipping. [11]

Technical Background:

Plan:

The plan for the design of the project began with the idea to expand and improve upon the project design of David Francis and Narayana Sundaram. Their idea of mounting a single tri-axial accelerometer in the center of a football was a good one, however there were flaws in the design mostly revolving around technology available and selection of components. My first idea was to do away with the accelerometer at the centroid of the ball, and instead mount accelerometers a radially around the circumference of the ball, as well as at the points. The electronics would be laid out not on a circuit board per se, but instead on a "flex circuit" of Kynar film with copper traces. This would then be held in place between the ball's bladder and outer shell. In researching available accelerometers, it was decided early on that an integrated circuit or "chip" style package was desirable, for purposes of both cost and of ease of integration. Because it could be mounted directly on a printed circuit board, or flex circuit, it eliminated excess cables, connectors, and bracketry that would otherwise be required. Analog Devices manufactured the least expensive accelerometers I found that would suit my needs. They were available in single and dual axis configurations and in varying sensitivities. The orientation for the sensing elements for these was in the X, or X-Y axis, on the plane of the chip. Motorola had similar form factor accelerometers, and also made a single axis accelerometer that was sensitive normal to the chip. Details of how these accelerometers function was presented in an earlier section of this report.

This idea was abandoned for a number of reasons, such as expense of the manufacture of the flex circuit, selection of accelerometers available, cost of other electrical components, and also of course, my own ability to accomplish this task. It was also decided that although this would be a very packageable solution, the lack of sensors at the centroid would make the calculation of the ball's position with respect to the ground much more difficult, as well as the need for at least some of the accelerometers to be sensitive in the normal direction. It was decided that if accelerometers would be needed at the centroid, then they could all be attached to 2 PCB's (printed circuit boards) and somehow suspended within the ball. After consulting with an expert in electrical design, who had experience with the Analog Devices products, and who volunteered to assist me in answering technical questions and offering advice, the configuration of four bi-axial accelerometers

was chosen. Two accelerometers would be placed at the centroid, to measure accelerations of the ball regardless of spin and wobble; one at a point of the ball to capture and end over end rotations, and one a distance off to the side of the ball to measure roll velocity. The raw acceleration data would be processed by a low pass Butterworth multiple-feedback (MFB) filter for the ADXL250, seen below in Figure 5. The filtered data will be read by one of two Microchip PIC16F876 microprocessors, which has 5 channels of built in A/D Conversion, each of which is responsible for 4 channels of acceleration (2 accelerometer chips). Each channel of acceleration data will then be written to a 64k EEPROM (electronically erasable programmable read only memory) chip for storage.

To extract the data form the ball after the event, a serial cable is plugged into the ball, and the data is continuously looped and can be captured by a communications program such as Microsoft's HyperTerminal. The data can be imported into a spreadsheet and analyzed further.

The plan in the previously described paragraphs is the third in a series of designs that was attempted for this project, and the first one to actually succeed. Plan A was a more elegant and ambitious design that involved a single PCB and a larger single microprocessor and conventional SRAM (as found in a PC), plan B involved 4 small self contained data acquisition units, each of which was able to record 2 channels of acceleration to EEPROMS . Details of the design and implementation problems of those designs will be addressed in a later section of this report.

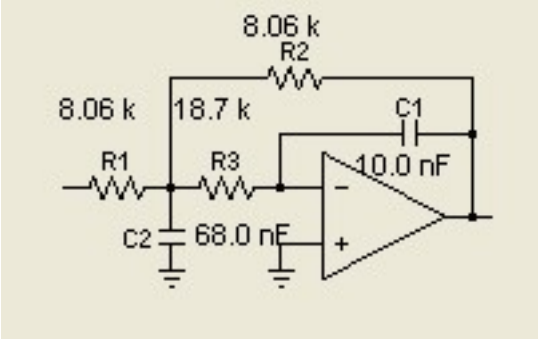


Figure 5 – Schematic of a Multiple Feedback Filter

Implementation:

The electronics for Plan A can be seen on the following pages in Figures 6-9 with the major components labeled. This plan was abandoned because I was never able to successfully read from, and I'm assuming also to write to, the SRAM device. The schematic, which can be seen in Appendix C, called for port D of the microprocessor to have 3 separate functions;

1. To send the high byte of the memory address (a 17 bit number) to one of the flip flop chips
2. To send the low byte of the memory address to the other flip flop chip
3. To send/receive the data to/from the SRAM chip

A "flip flop" is actually a bistable multivibrator, which is also a type of memory device. In this case it's used to remember what either the high or low byte of the memory address is. It had 8 input and 8 output pins, as well as a clock pin. When the clock pin is manipulated, it will latch the output pins to match the input pins. Each of these functions was controlled in software by enabling a clock pin on each of the chips (flip flop 1, flip flop 2, and SRAM) at the right time to determine which chip was receiving the data across the microprocessor's Port D. This process was further complicated by having to toggle the pins between being output pins for generating the address and writing to memory, to input pins for reading from memory. I was never able to identify cause of the problem in either hardware of software. The decision was made to abandon this approach and attempt a more brute force approach to solve the problem. The factors influencing that decision are as follows:

- This was my first ever attempt at designing a PCB, and no less than 12 mistakes were identified in the board layout including missing traces, crossed traces, and misplaced traces. All these were fixed after the fact with a razor blade and fine wire. I couldn't find any more mistakes on the memory area of the PCB, but that didn't convince me that there wasn't one.
- The code required to manipulate Port D was complicated
- Because surface mount components were chosen for this design, the questionable part of the circuit could not be taken off the PCB and tested on a breadboard to aid diagnosis

- My lack of experience reading data sheets resulted in my running the clock pins to the flip flop chips and the SRAM chip in reverse, possibly damaging the chips. (The clock pin was "active low, which means that it is held high normally, and then pulled low to make something happen. I had been driving it as if it were active high)

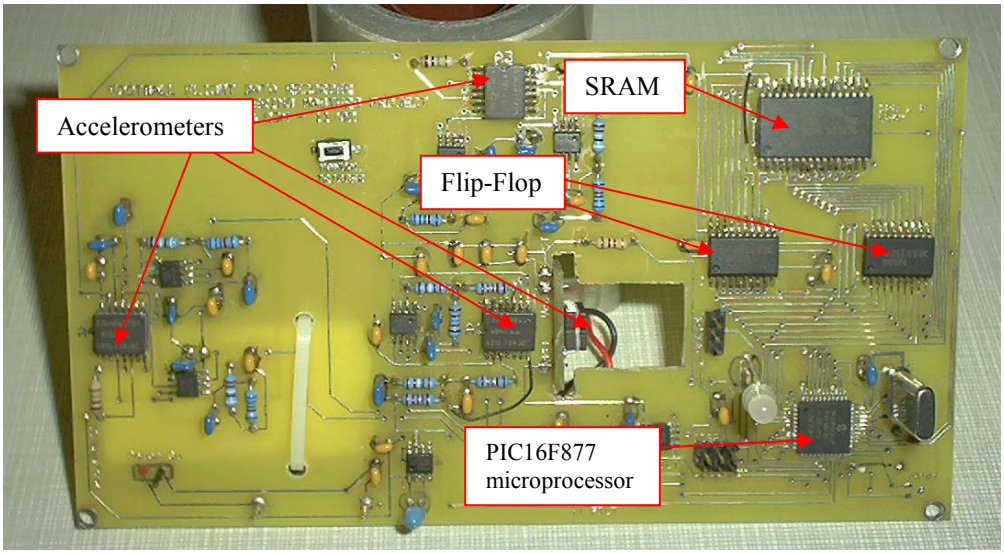


Figure 6 – Front of PCB for Plan "A" showing major components

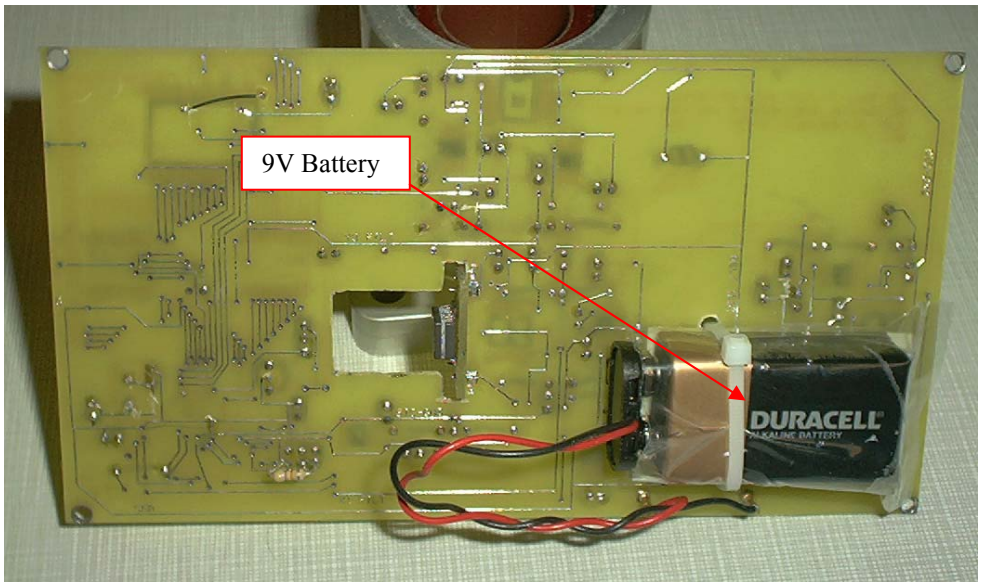


Figure 7 – Rear of PCB for Plan "A" showing major components

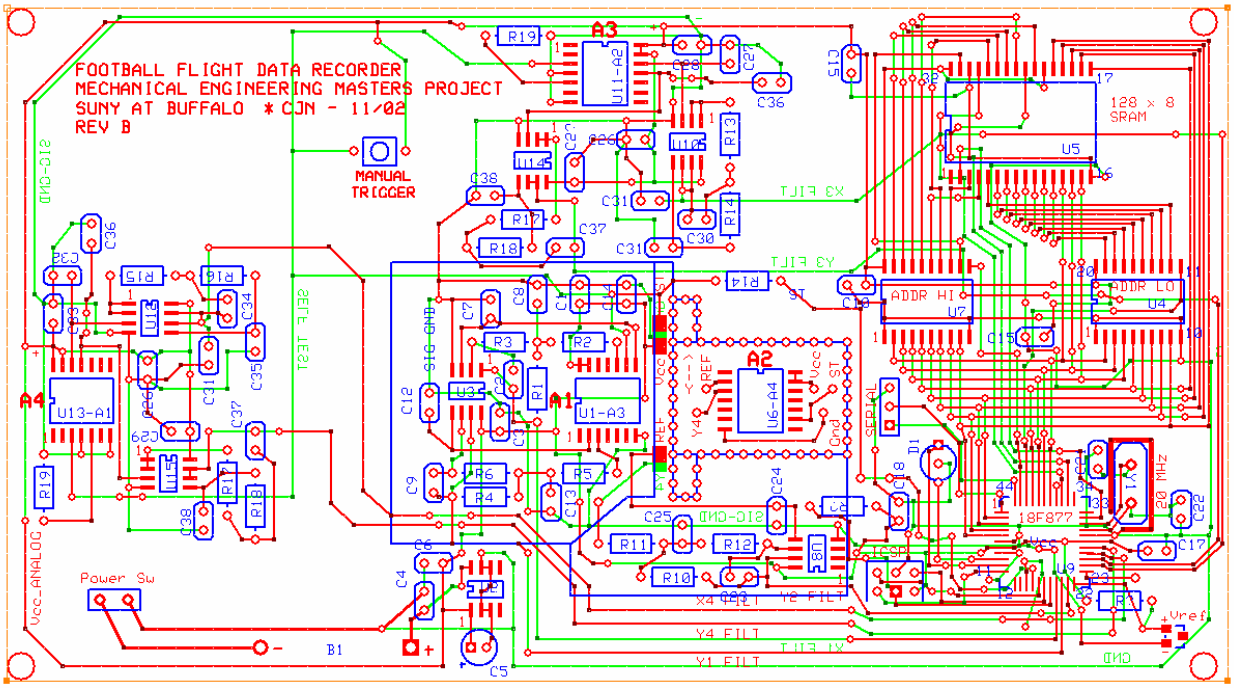


Figure 8 – PCB Layout. This design was created with software downloaded from ExpressPCB.com, who was the manufacturer of the PCB. This software both allows you to lay out the traces and components on the PCB, and also automatically transfers the information over the internet to the manufacturing facility. The printed bare boards arrived 3 business days later.

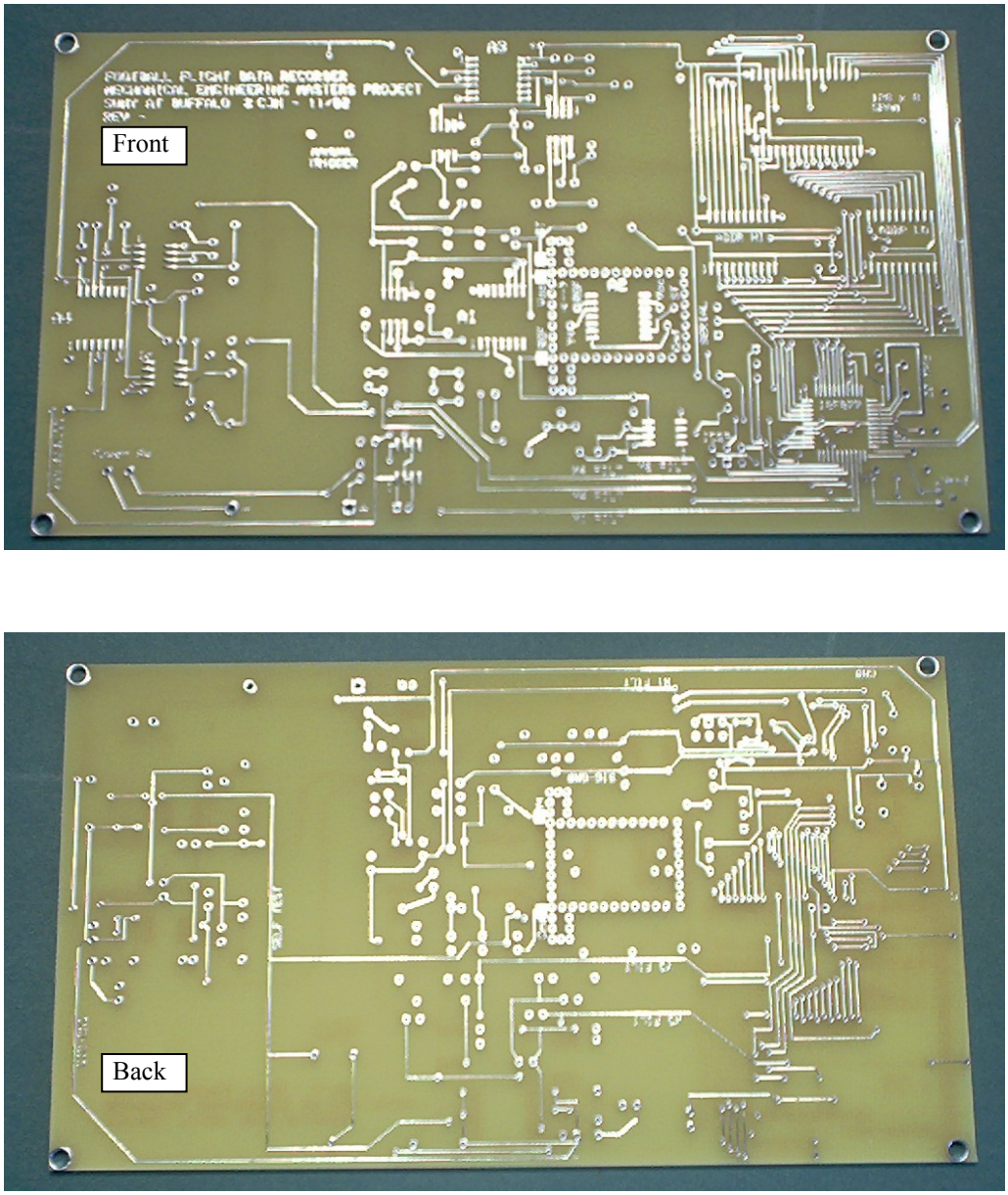


Figure 9 – Front and back of PCB as received from ExpressPCB.com

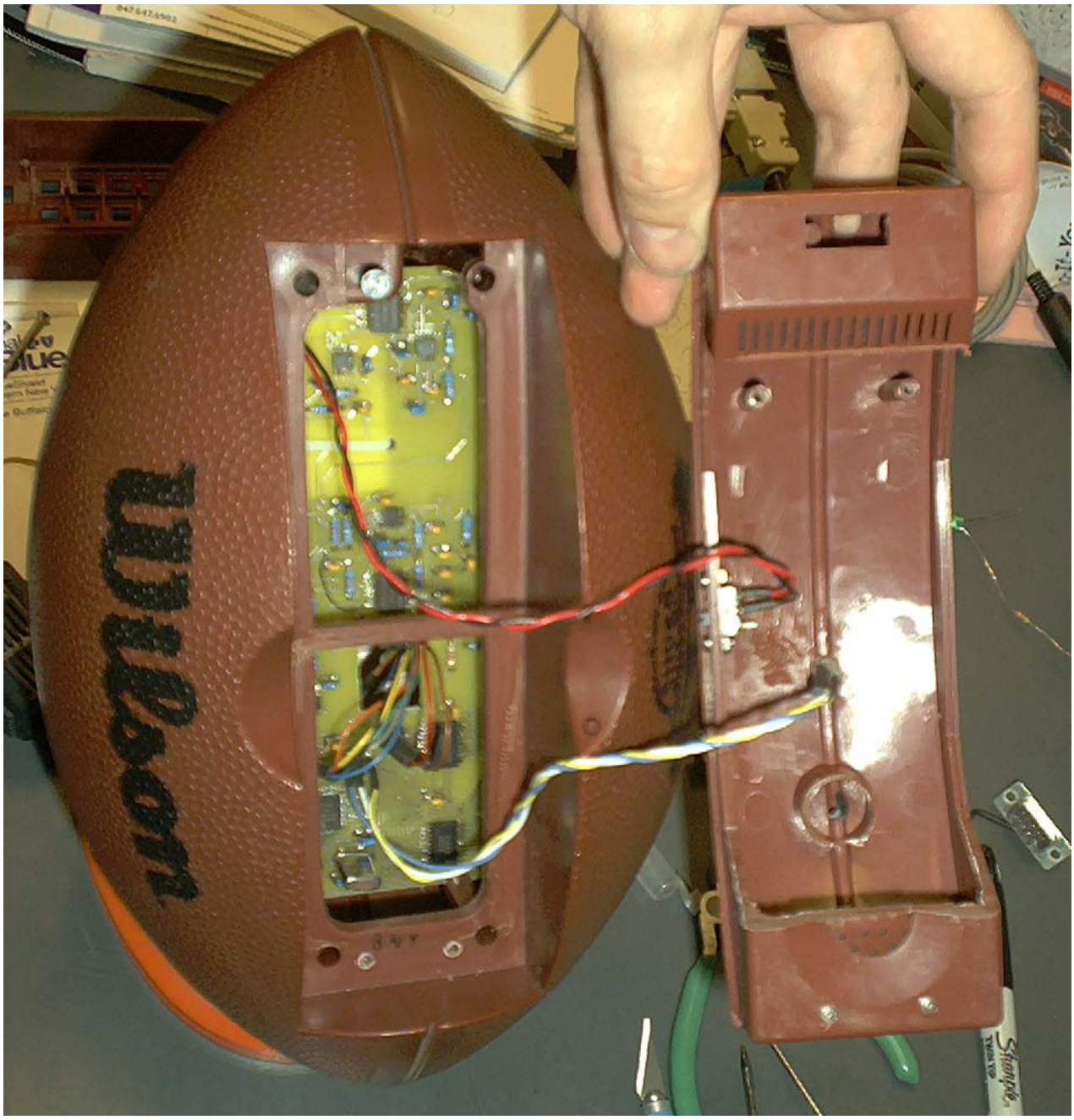


Figure 10 – Assembled electronics shown in football shaped enclosure

The design cycle for Plan B was much shorter than that of Plan A, since there were fewer learning curves involved. A smaller microprocessor was chosen, as well as EEPROMs for the memory unit instead of the SRAM. The schematic for the Plan B design can be found in Appendix C. In a trade off for speed, as Plan A was expected to record 1000 samples per second, per acceleration channel, the EEPROMs could only record 200, but the advantage being in the simplicity of the interface. The data is written to and read from them via a SPI serial interface, which is a much less complex procedure, from both hardware and software standpoints. The board layout for Plan B can be seen below in Figure 11. A photo of the assembled 2 channel board can be seen on the following page in Figure 12.

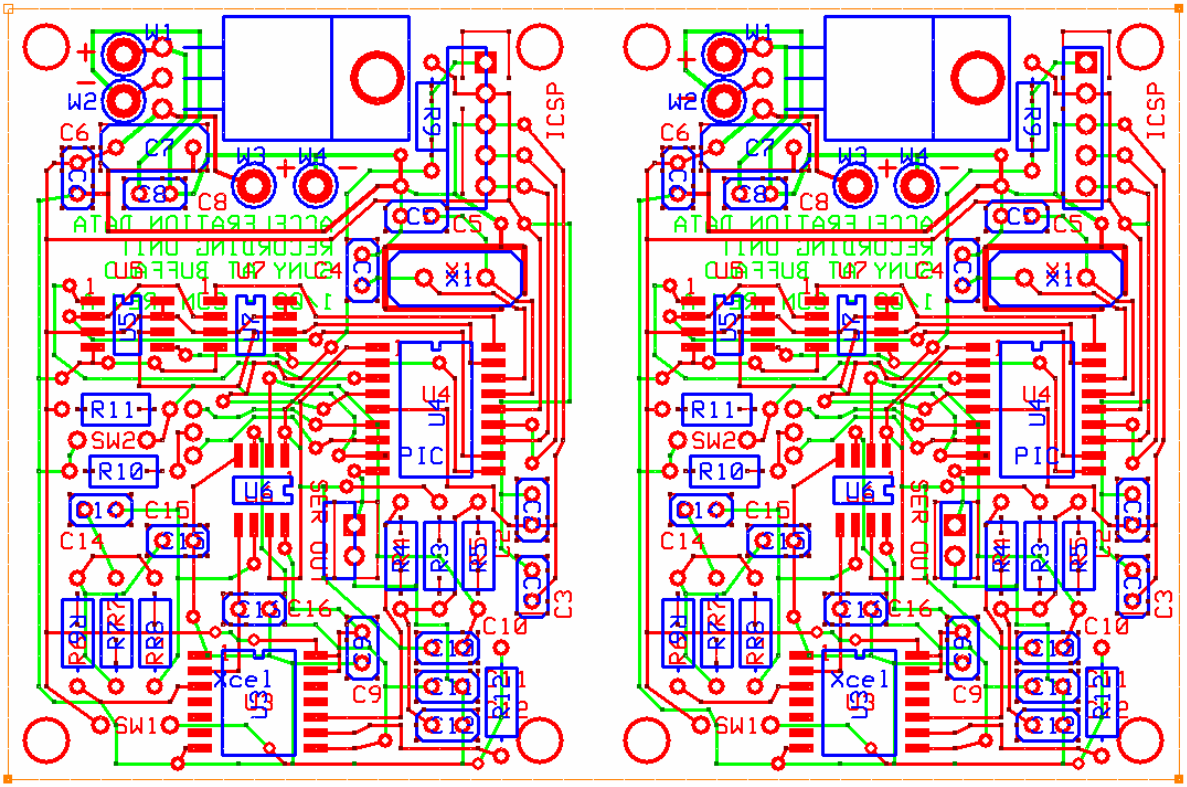


Figure 11 – PCB Layout for Plan B. The design is seen twice here, the board was ordered from a stock size, and the layout fit on the available space 2 times. After the finished boards were received, they were cut in half.

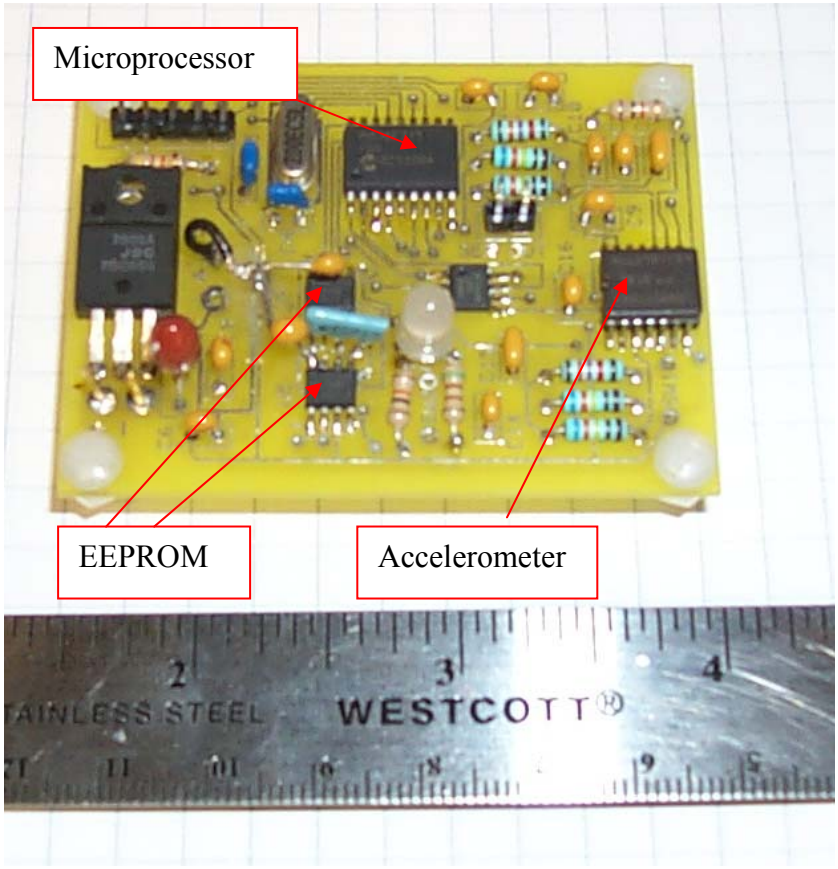


Figure12 – Plan B assembled PCB

The flaw in the design of Plan B that ultimately led to the decision to move on to Plan C was in the choice of the microprocessor. The Microchip PIC16C717 was chosen based on its features, most notably, 5 built in A/D converters, a direct SPI interface, and an 18 pin surface mount package. The fact that was overlooked was that it was not a "flash programmable" microprocessor, but was instead a "write once" microprocessor. After programming, if any problems were discovered in the code, and the software needed revising, the microprocessor had to be replaced with a new one. If I had realized this when I chose it, I would have not soldered it directly to the board, but instead would have had a socket placed there so the microprocessor could be easily changed. I also only had a few of them on hand that I received as samples, and felt that it would not be practical to de-solder and re-solder the microprocessor as many times as I'd need to, to make the device work. I attempted to locate an alternate microprocessor that had the same pin layout, which was flash programmable, but failed to find an exact match. However, I found one that was close, but the programmer and compiler I was using was not compatible, and an upgrade for those was prohibitively expensive.

Plan C was devised as a combination of the better design aspects of both Plan A and Plan B. It was decided that rather than needing 4 separate self contained units, a more integrated approach was attractive, similar to Plan A. The design details from Plan B that were retained were the EEPROM memory, with a separate chip for each channel of acceleration data, and the idea of multiple microprocessors on multiple, less complex, identical PCBs. Even Plan A was not truly a "one PCB" design as there was a small area of the main board that was cut out, and turned 90º to the plane of the main PCB and reattached. This was done so that an accelerometer that is only sensitive in the plane of the chip could be turned on end, so that the chips Y axis would be the Z axis of the main board. The can be seen below detailed in Figure 13.

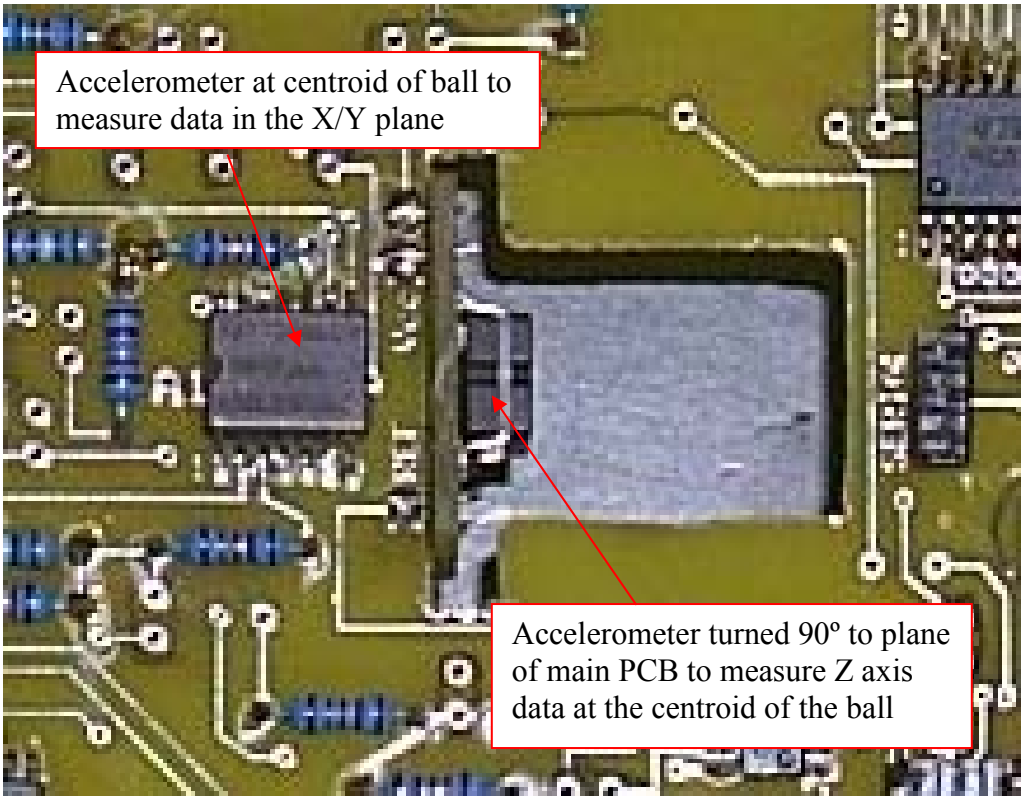


Figure13 –Detail of modification to Plan A board to allow triaxial acceleration data to be captured at centroid of the ball

To maintain the ability to capture triaxial data from the ball's centroid, but to allow all the PCBs to identical, and therefore requiring only a single board layout design, a slotted board was designed that allowed 2 boards to "nest" together, similar to a divider in a packing container. The board layout can be seen below in Figure 13, and photos of the boards both individually and also nested together can be seen in figures 15-18 on the following pages.

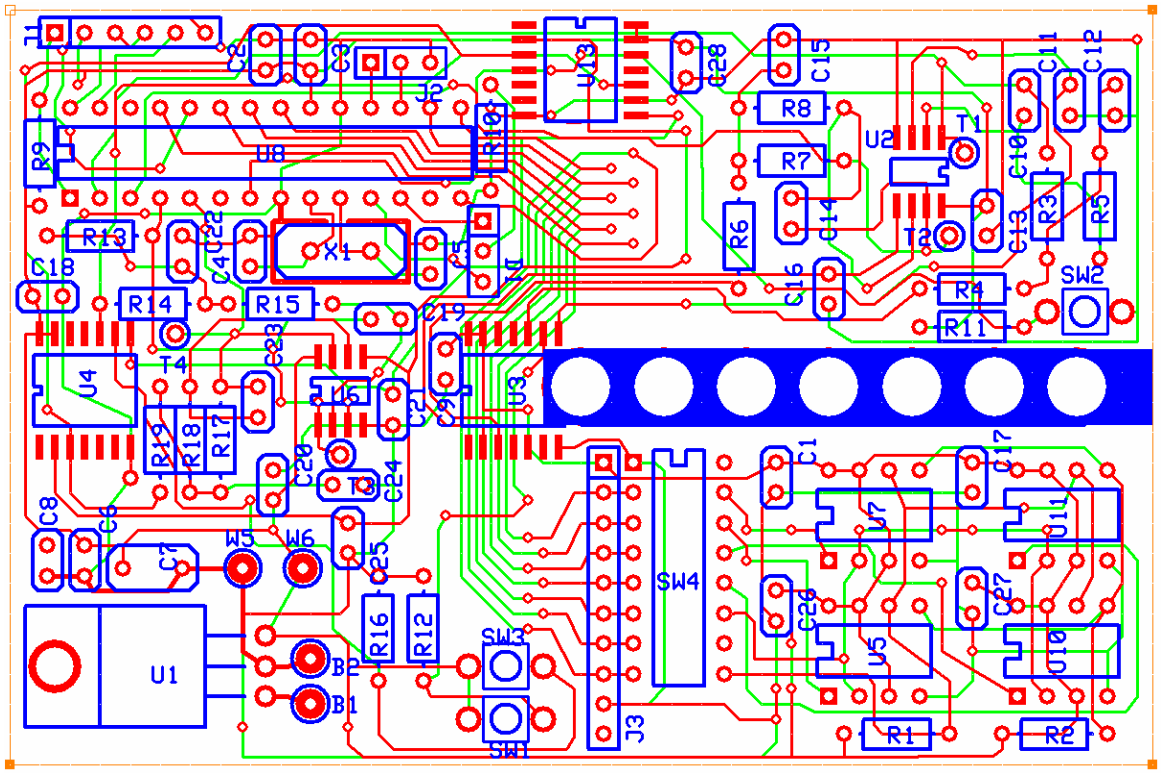


Figure14 – PCB Layout for Plan C. The Blue area with the holes in the middle right side of the board is material that would need to be removed after the boards were received

In the notation in Figure 15 a vacant area is shown for an accelerometer. The circuit is designed so that an accelerometer is placed in either the location denoted as vacant (U13), or in the location at the middle of the left side of the board (U4), but not both. The data output pins for both locations are wired together and share common low pass filters for the X and Y accelerations before the signal is sent to the microprocessor for A/D conversion and then on to the EEPROMS for storage. These low pass filters were present in Plans A, B, and C, and will be detailed out in a later section. Accelerometers at these locations are

used to measure the centripetal accelerations from the spinning and/or tumbling ball to better formulate a complete picture of the ball's motion.

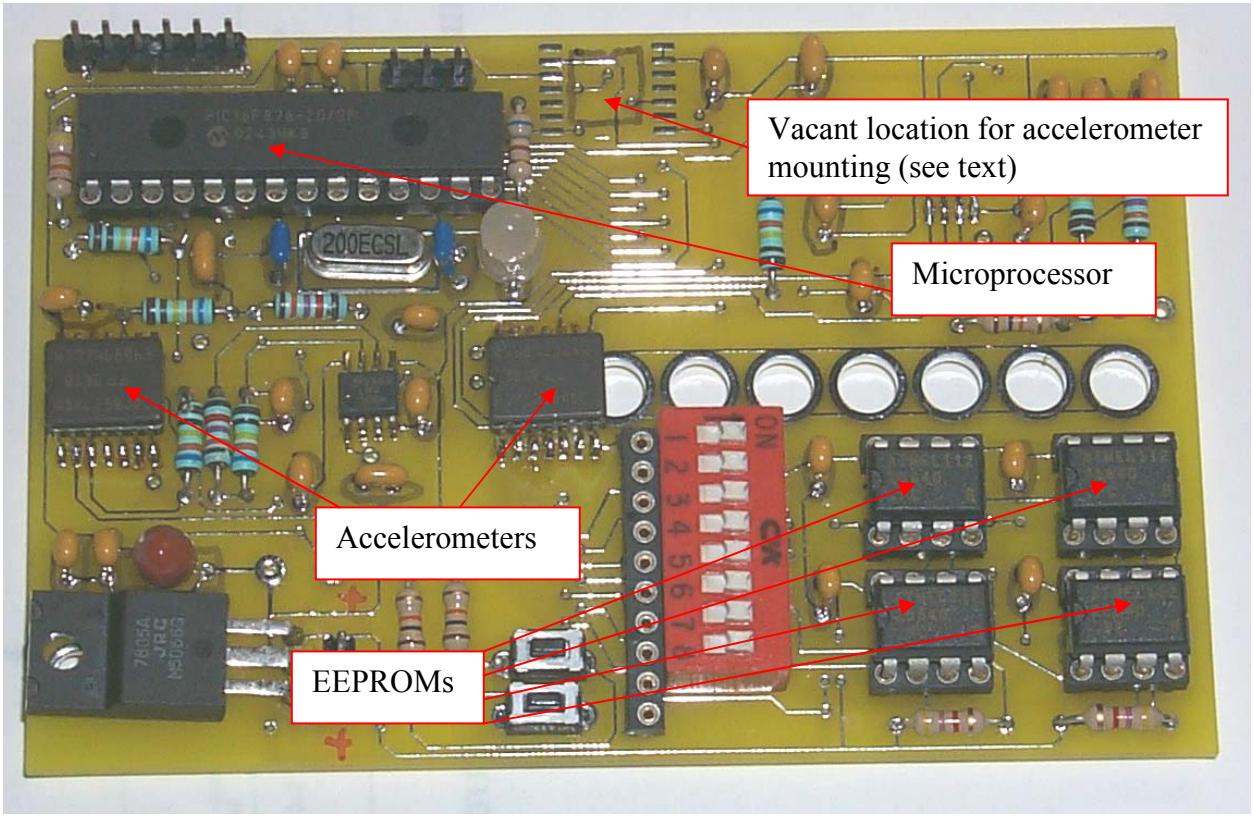


Figure15 – Assembled PCB. Two (almost) identical units are required to record all the data inside the football.

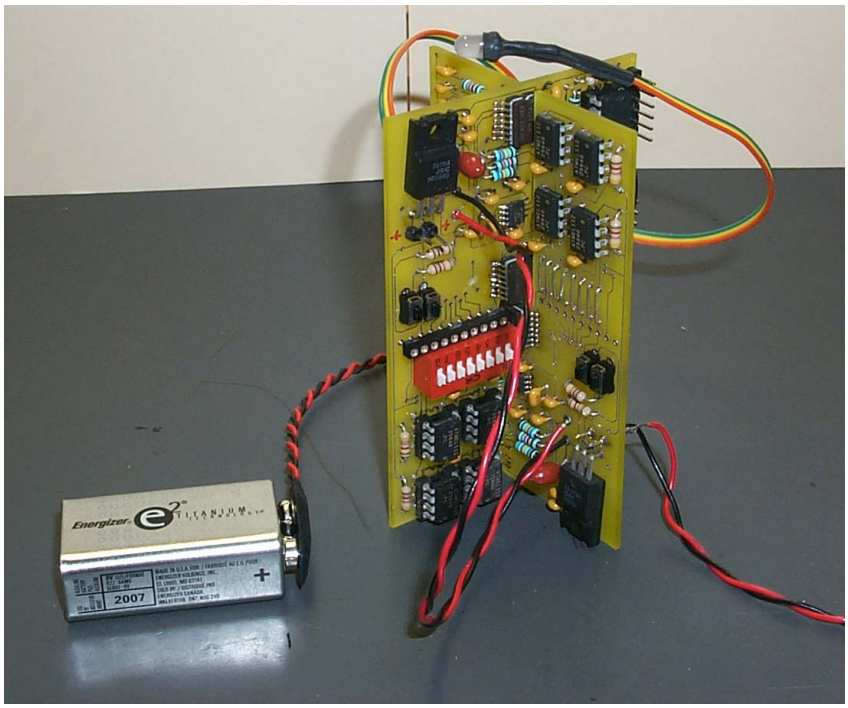


Figure16 – Two assembled PCBs nested together. On Board 1 the accelerometer is placed in the U4 location, which is nearest the "point" of the football, and on board 2 the accelerometer is placed in the U13 location

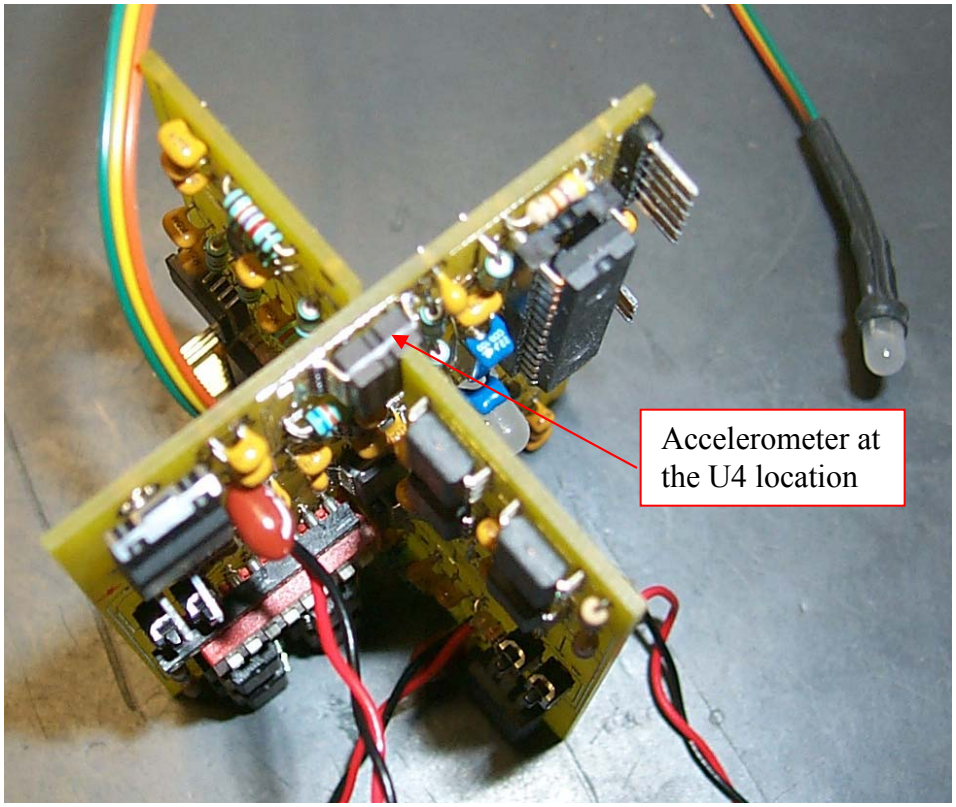


Figure17 – Alternate view of Board 1 and Board 2 nested together

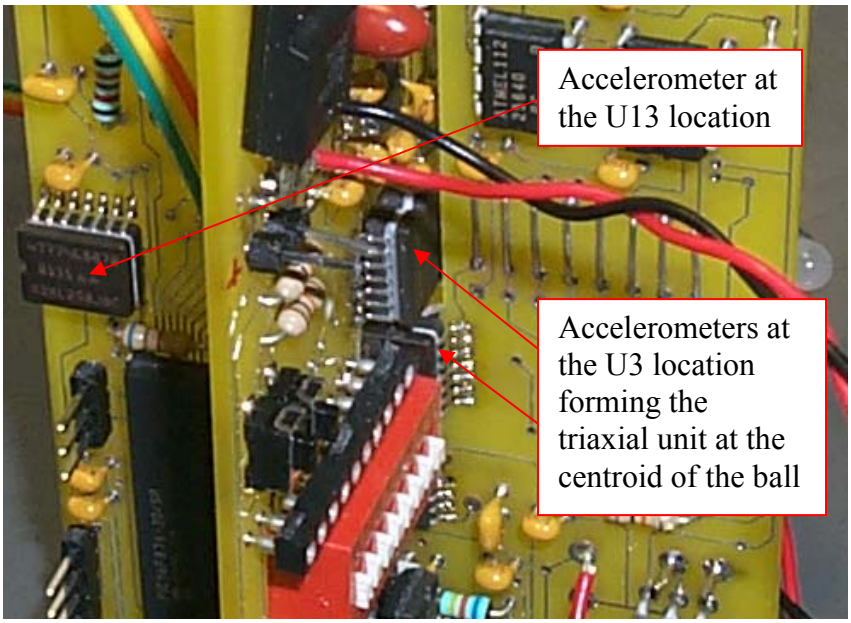


Figure18 – Two assembled PCBs nested together. The accelerometers at the U3 position of each PCB are 90º opposed to each other at the centroid of the ball. The X axis of the accelerometer on board 2 is recorded as the Z axis acceleration at the centroid.

Each channel of acceleration has a Multiple Feedback Butterworth style of filter in place to remove the high frequency components of the acceleration data. A freeware program called *Filter Pro* which is a low pass filter design program was downloaded from www.ti.com, the Texas Instruments website. A screen capture of the program can be seen on the next page in Figure 19.

This software allows you to easily select the sizes of the resistors and capacitors needed to accomplish the desired result. The Multiple Feedback (MFB) or Infinite Gain topology places two feedback paths around an op amp. This Filter is less sensitive to component values that the Sallen-Key topology, which uses an op amp in a noninverting gain mode. The Sallen-Key filter provides excellent passband gain accuracy. The Butterworth filters have the flattest possible passband response and a smooth transition into the stopband. The MFB Butterworth filter was recommended to me by a colleague that has past experience working with the ADXL250 and a Pic processor with built in A/D conversion.

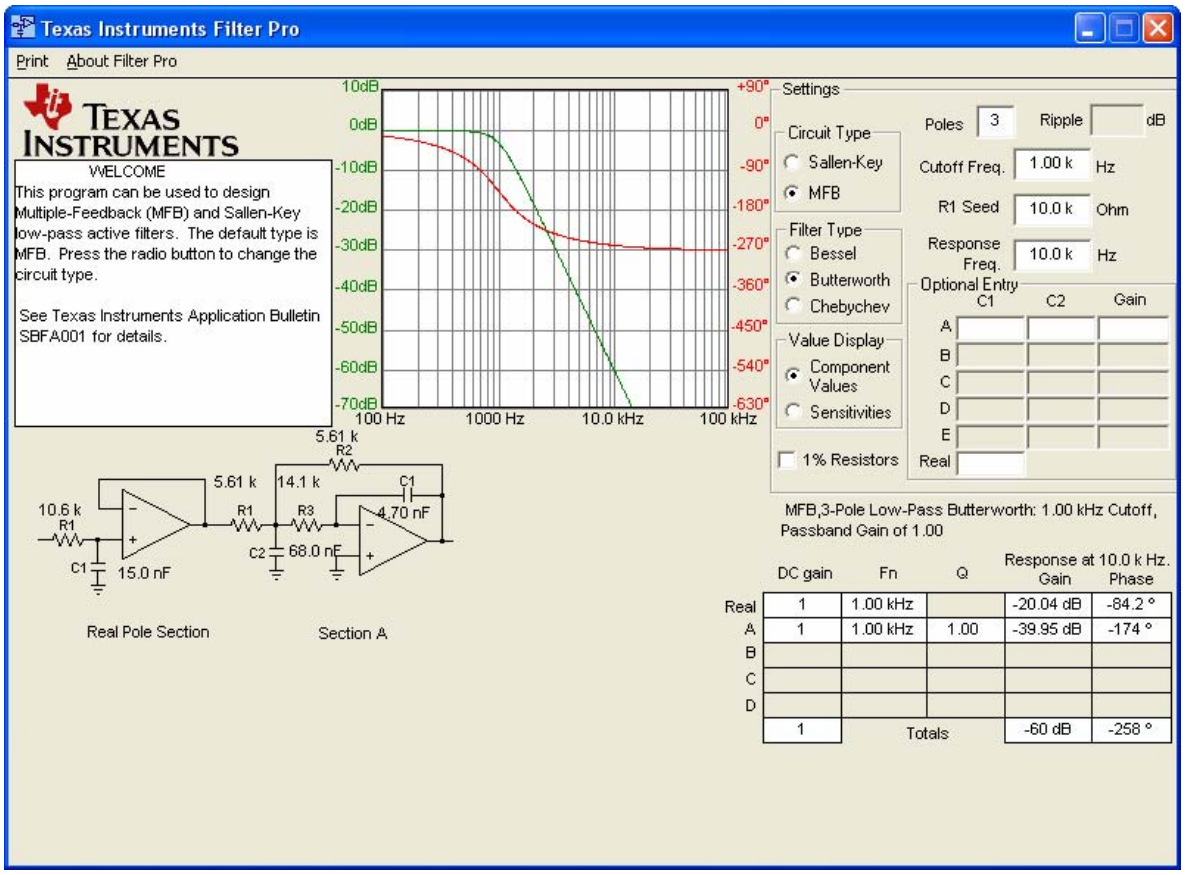


Figure19 – Screen capture of *Filter Pro*, a freeware program to aid in the design of low-pass filters

I had chosen 200 samples per second per channel to be my target data acquisition rate, and as such selected 200 Hz as the cutoff frequency. Filter design is a very complex area of electrical engineering, and as such I was relying heavily on advice from people that are much better versed in it than I. At some point I had a misunderstanding of exactly how to enter the criteria into *Filter Pro* to achieve the result that I wanted. After the unit was built and testing had begun, it was learned that what I actually should have had was a filter with a cutoff frequency in the range of 75-100 Hz. This cutoff frequency has to do with the sampling theorem which states that to reconstruct the frequency content of a measured signal accurately, the sample rate must be more than twice the highest frequency contained in the measured signal. If the filter was set at a 200 Hz cutoff, the sampling rate would have to be increased to 400+ samples per second to meet these criteria. The 2 nested PCBs of the football unit are joined together by hot melt glue that had encapsulated some of the

resistors and capacitors that would need to be replaced in order to change the filter's cutoff frequency, and because of time and expense, rebuilding the entire unit, or trying to remove the glue was not an option. The EEPROM chosen as the storage device was limited to performing a read or write action with a minimum of 5ms between each action. This limits the EEPROM to recording 200 samples per second. An additional EEPROM could be designed in and have the data sent to alternating EEPROMs to get 400+ data samples per second, but again, extensive hardware redesign was not an option. What was decided on as a solution was to "oversample" the accelerometer, or to take more samples than you record, since the microprocessor was very capable of working faster, and to average 2 consecutive samples from the accelerometer at 400 samples per second thus recording 200 samples per second. The algorithm for this process was as follows:

1. do the A/D conversion on x1, y1, x2, and y2
2. save y1 and y2 each to a dummy variable
3. do the A/D conversion on x1, y1, x2, and y2
4. y1 = (y1 + stored y1)/2 **and** y2 = (y2 + stored y2)/2
5. save y1 and y2 to memory
6. save x1 and x2 each to a dummy variable
7. do the A/D conversion on x1, y1, x2, and y2
8. x1 = (x1 + stored x1)/2 **and** x2 = (x2 + stored x2)/2
9. save x1 and x2 to memory
10. go to step 2 and repeat for the duration of the data recording time envelope

The code for each of the two boards in the football is virtually identical, and the exact same code could be used. The only differences between the two versions are in the interface with HyperTerminal. This stems from the fact that although the 4 channels of acceleration data available on the secondary board, only three are useful. When the primary board downloads, it streams out via a serial cable the following information:

1. Start of data recording
2. End of data recording
3. Beginning of data dump
4. Data in tab delimited columns;
    a. Memory Address
    b. Acceleration X1

c. Acceleration Y1
d. Acceleration X2
e. Acceleration Y2

5. End of data dump
6. go to step 3 and loop indefinitely

The secondary board works the same except for step 4, which downloads the following

a. Memory Address
b. Acceleration Z1 (actually the X channel of accelerometer U3)
c. Acceleration X3
d. Acceleration Y3

The code for the football was written in *PicBasic Pro*, which is a compiler based on the BASIC computer language, and very similar to the code used to program the Parallax family of Basic Stamps. The user's manual and disk can be seen to the left in Figure 20. The final code for each of the 2 PCBs (as well as for some of prior design iterations) in the football can be found in Appendix E. The actual coding is done in a program called *CodeDesigner Lite*, or *CD Lite*. A screenshot of *CD Lite* can be seen below in Figure 21.

Figure 20 – PicBasic Pro [13]

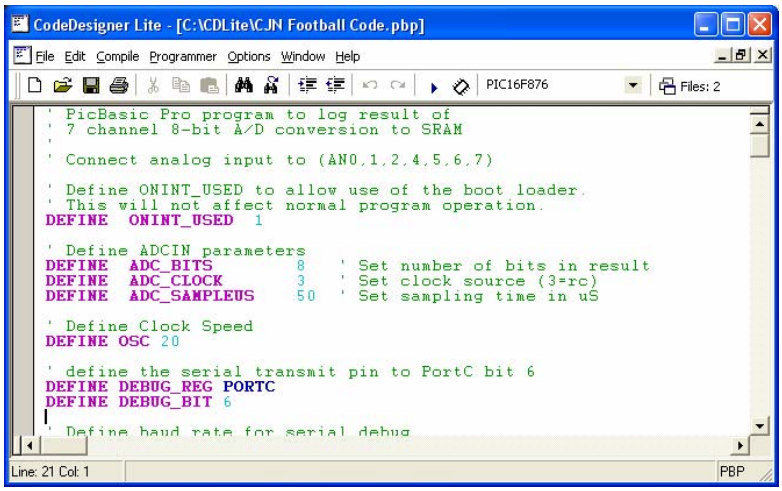


Figure 21 – Screen capture of *CodeDesigner Lite*, a programming environment used with PicBasic Pro.

Upon writing the code in PicBasic the Code was compiled into .hex and .asm files that could be used by a programmer to install the software on to the Pic Microcontroller. The programmer that was used was the EPIC Programmer (see below in Figure 22) by microEngineering Labs Inc.

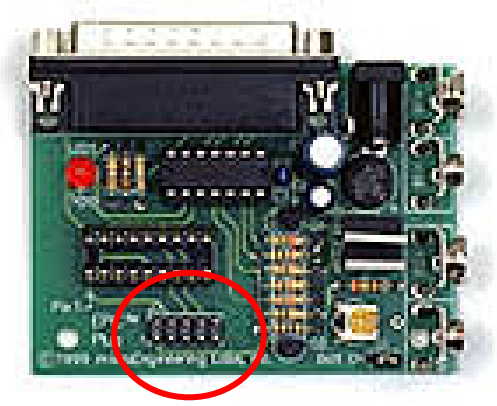


Figure 22 – The pocket-sized EPIC Plus Programmer quickly and easily programs most PICmicro microcontrollers. The basic programmer includes an 18-pin socket for programming 8-, 14- and 18-pin PICmicro MCUs. The connection for ICSP is circled in red. [14]

The programmer runs off two 9-volt batteries or an AC adapter (16VDC, 500ma). It plugs into the PC parallel printer port using a 25-pin male to 25-pin female parallel printer extension cable. It connects to the ICSP (in circuit serial programming) header on each of the football PCBs with a custom built cable. The design of a cable such as this is detailed in the microEngineering Lab website, and in the schematics seen in the appendix of this document.

After a period of software debugging and experimental code writing, the code for each of the two boards was finalized and the microprocessors were flash programmed a final time. This code can be found in Appendix E at the end of this document. It was at this point that data collection began.

The final step before collecting real data was to install the electronics into a football, or in this case a "football like" enclosure. As seen in Figure 10 (page 8) Plan "A" utilized a hard plastic football shaped enclosure, which was actually the carcass of a football phone. It was the same size as a real football, and with the system installed weighed only a couple ounces more. Both of those features were attractive, but it also had serious drawbacks. Its primary drawbacks were that it was rather fragile, and more

importantly, poorly balanced. For Plan "C" it was decided that something a little more robust might be required. A Nerf football was settled upon as a reasonable compromise. After a failed search for an "official size" Nerf football, a smaller football was chosen. The football was sectioned into halves along the plane in the middle of the ball, so that it would split into a "laces half, and a "non-laces" half. Additional material was then removed to create voids that then would accept the electronics, batteries, wires, etc. To return the ball into a single unit, a variety of mechanisms were considered, ranging from tape, to Velcro, to snaps. The solution finally chosen was a zipper. Since the current design mandated that the electronics be exposed for downloading data to a PC, the zipper provided an ideal access control device. The zipper was secured to the ball with cloth gaffer's tape and adhesive glue. A LED from one of the PCB is poked through the ball to tell the user the status of the data recorder. The electronics can be seen in the ball below in Figure 23 -24.

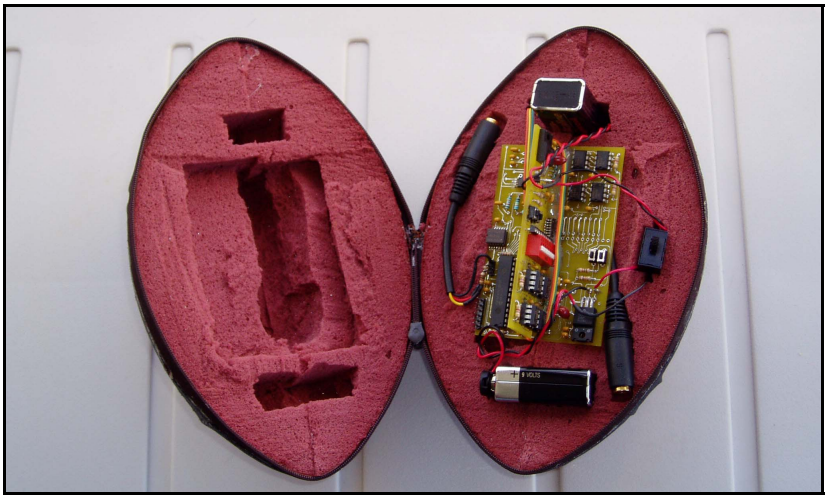


Figure 23 – Electronics installed into the Nerf football

Figure 24 – Zipper closure. The green arrow points out the status LED

Data:

Data collection began with some bench testing of the PCBs individually. The first test performed involved holding the boards and just shaking and wiggling them to induce some random accelerations. A sample plot of the data can be seen below in Figure 25. Test data was also collected for the other PCB, which looked similarly random. At this time the programming on the 2 boards was identical, and as such both boards had channels X1, Y1, X2, and Y2.

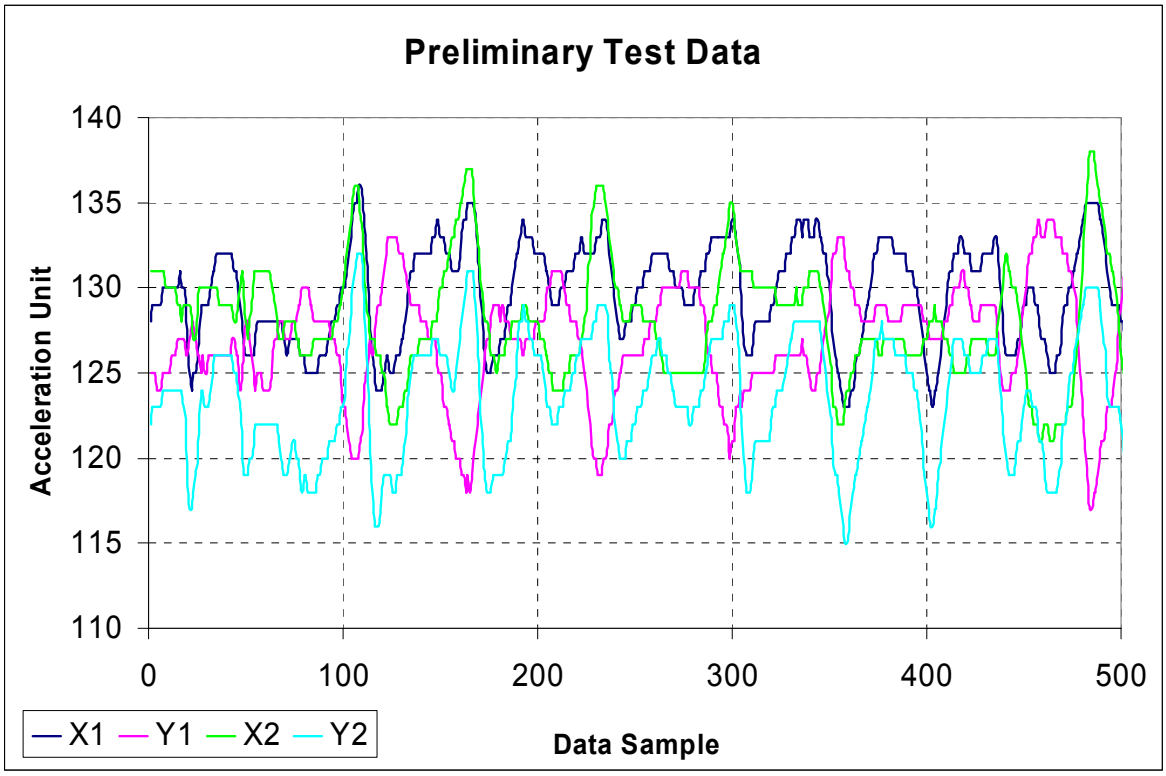


Figure 25 – Preliminary Test Data

The coordinate system for the football was defined as seen on the following page in Figure 26. After reprogramming the boards to work together, the seven channels of data to be collected were labeled X1, Y1, X2, and Y2 for the 4 channels available on the board in the X-Y Plane of the ball, and Z1, X3, and Y3 for the board in the Y-Z Plane of the ball. X1, Y1, and Z1 are nearest the centroid of the ball (which will be dubbed Accelerometer 1 from now on), X2 and Y2 are at the "forward point" of the ball (which will be dubbed Accelerometer 2) , and X3 and Y3 are offset from the centroid of the ball along Yb (which will be dubbed Accelerometer 3).

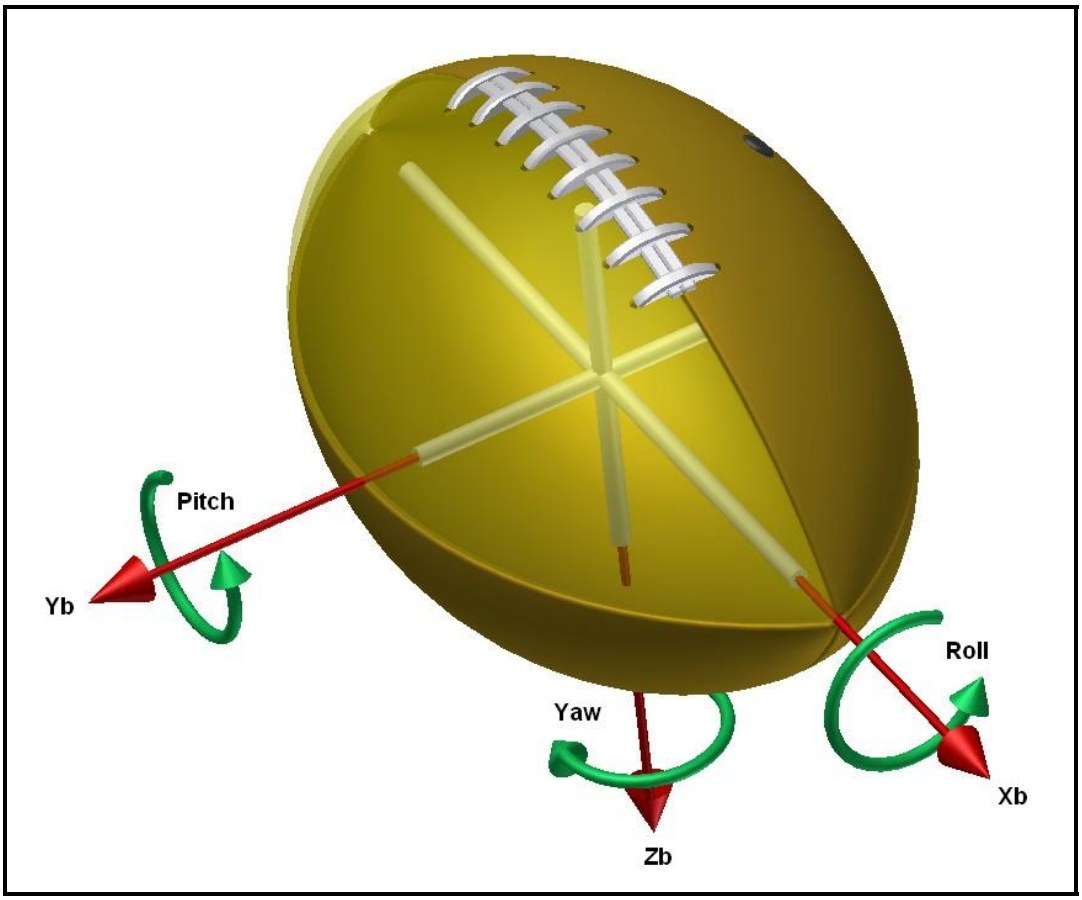


Figure 26 – Axis Definitions

The next step in data collection was to take some baseline data to determine calibration points for the various accelerometers. Nominally, at 0g, each accelerometer would register 128 in what I'm calling "acceleration units" but is actually the raw value stored as part of the 8-bit analog to digital conversion of the voltage output of the ADXL250 accelerometer, which is equivalent to 2.5 volts. There is some variation between the 4 accelerometers, and thus the test data seen in the following figures shows the accelerometers either under -1, 0, or +1 g's of acceleration depending on the orientation of the ball. Figure 27 is a graph showing all seven channels of recorded data, and 28-30 is the same data shown per accelerometer. The data received for channel Z1 fluctuates rapidly between 128 and 129 when sitting at 0g. Once the accelerometer has been exposed to accelerations under a power on, this appears to subside. The other channel of the physical accelerometer chip that Z1 is a part of is not relevant, as it is redundant with Y1, but it does not exhibit this behavior.

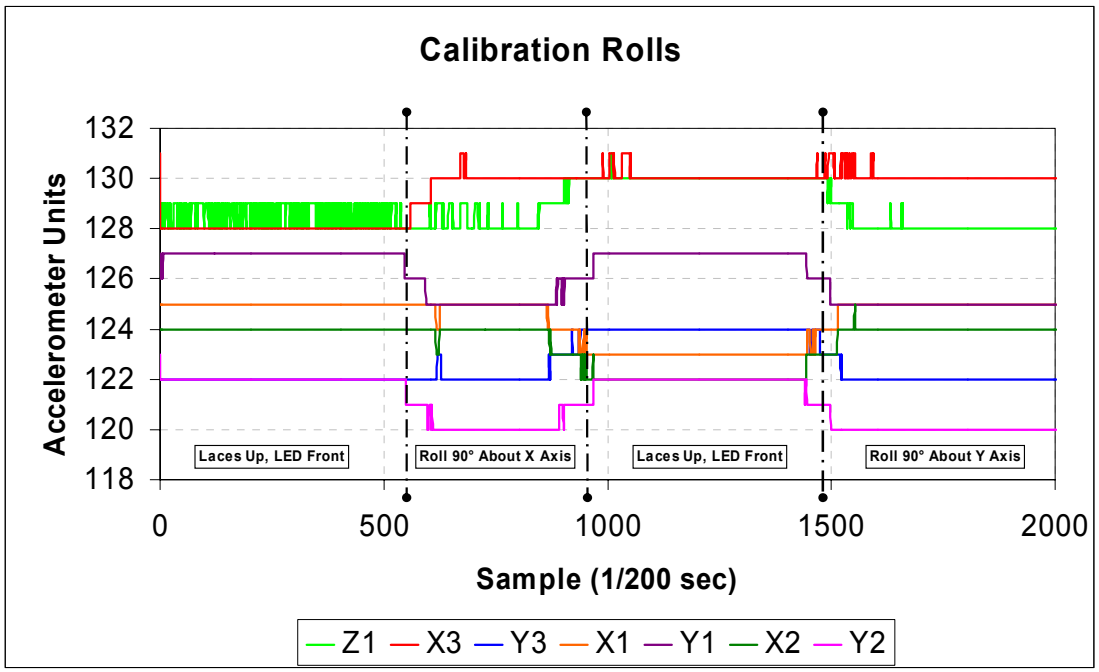


Figure 27 – Baseline Data For All Accelerometers

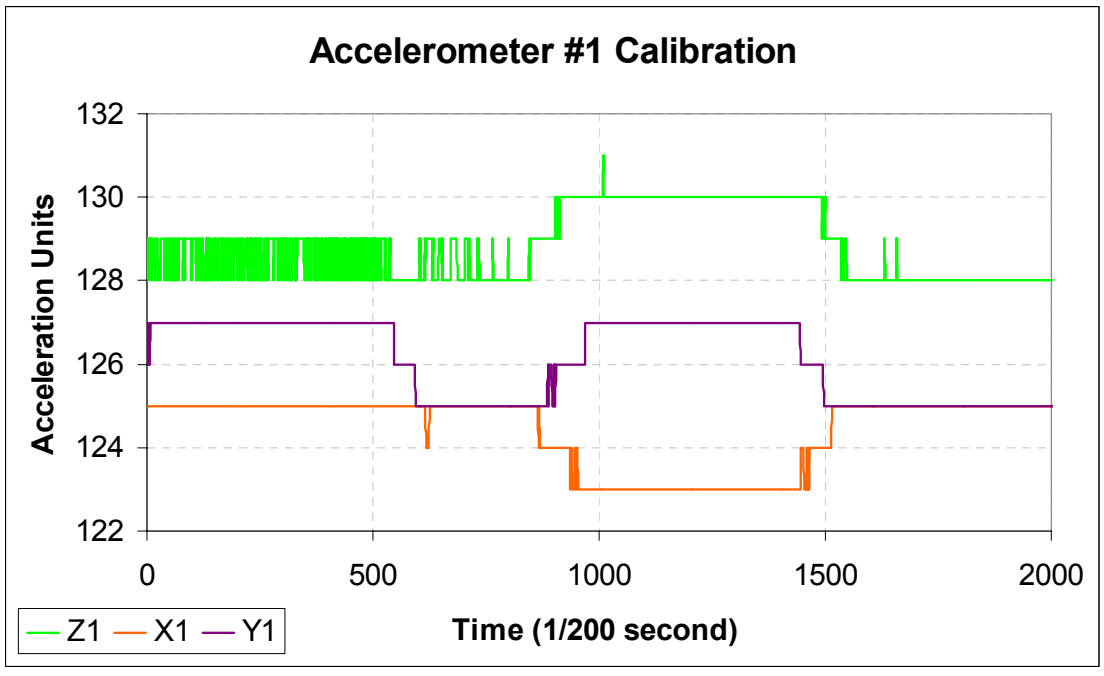


Figure 28 – Baseline Data For Accelerometer #1

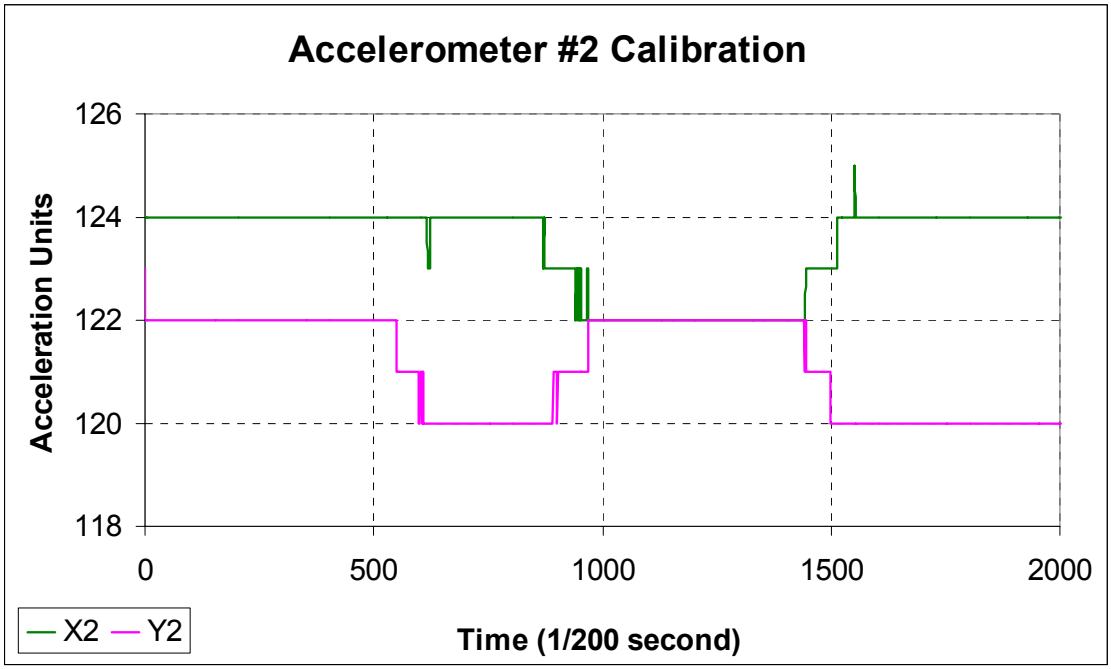


Figure 29 – Baseline Data For Accelerometer #2

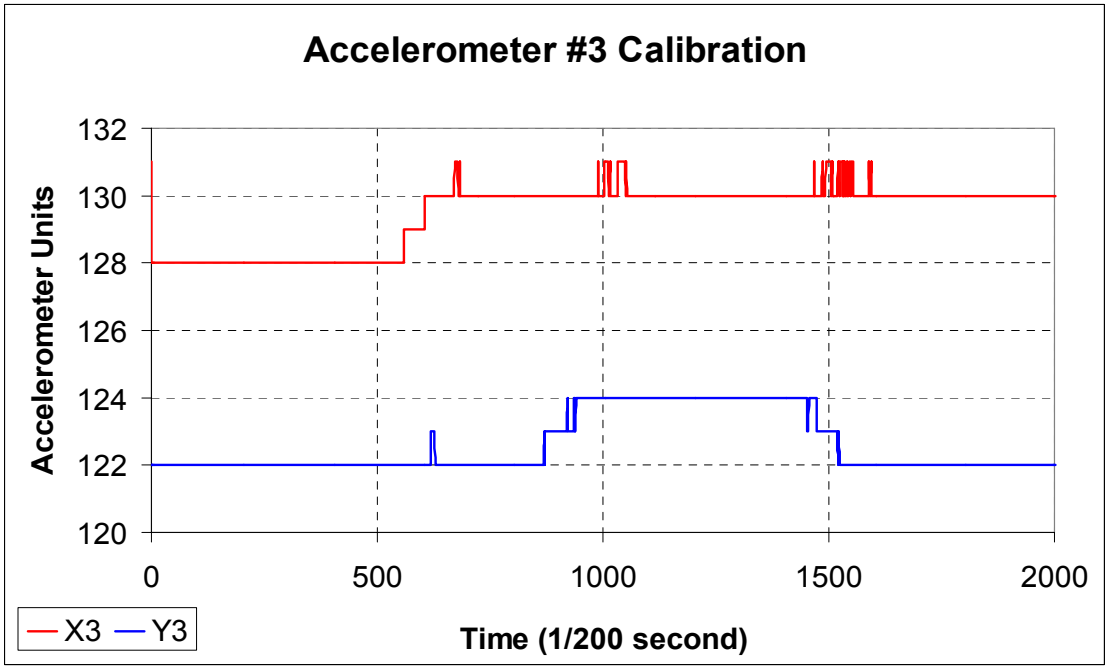


Figure 30 – Baseline Data For Accelerometer #3

After acquiring baseline data, I began to simulate actual conditions present in the motion of the ball as best I could to determine if the data I was going to receive from an actual throw could be successfully interpreted. The graph seen below in Figure 31 is data collected from an induced "spin" on the ball, which by the previously defined coordinate system would be classified as a "roll condition". I held the ball in a laces up orientation, induced the roll, with very little horizontal displacement of the ball, and some limited vertical displacement, and then caught the ball again before it struck the ground. The data from Accelerometer #1 was of little interest and was omitted from the graph to aid in clarity.

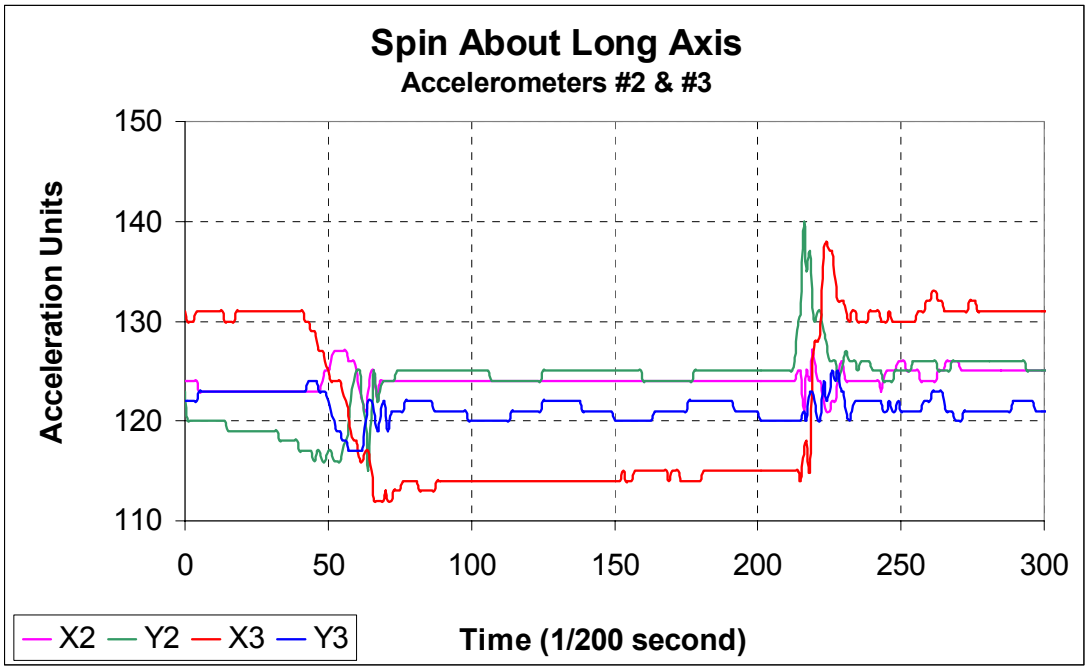


Figure 31 – Roll about Xb with minimal horizontal displacement

The most noteworthy data in the above figure is the data from channel "X3." The roll condition is introduced after about 2.5 seconds of sampling had begun (50/200ths of a second) as can be seen by the activity across all 4 displayed channels. The X3 value shifts abruptly from ~131 to 112, and then gradually up to about 116, before the ball was caught, which is the event at about 225 on the time scale. This is the centripetal acceleration induced by the rotation. At ~0.4g per "acceleration unit" this equated to an induced acceleration of ~7.6 G's, which degraded to ~6 G's before the roll was stopped. The offset from the axis of rotation is 1 inch. The rpm's of the ball can be calculated from this as follows:

A = 7.6 * 32.2 ft/sec^2 = 244.72 ft/sec^2

N is the revolutions per second of the ball

ω is the angular velocity

A = v^2/r and therefore v = sqrt(A*r) so that v = 15.64 in/sec

Since r = 1, ω = v

N = $\omega/2\pi$ therefore N = 15.64/6.28 = 2.49 RPS

The ball rolled at 149 rpm

Another noteworthy bit of data from the graph is the "Y3" channel is the sinusoidal oscillation present. That is an acceleration that is induced by the "wobble" of the ball as it spins, which is an observed phenomenon present in actual football flight.

The next test to be preformed was an actual "forward pass" of the football. The conditions were a pass of ~25 yards, thrown to the best of my quarterbacking abilities. The ball was not caught and instead impacted the ground. The data from Accelerometer #3 can be seen below in Figure 32.

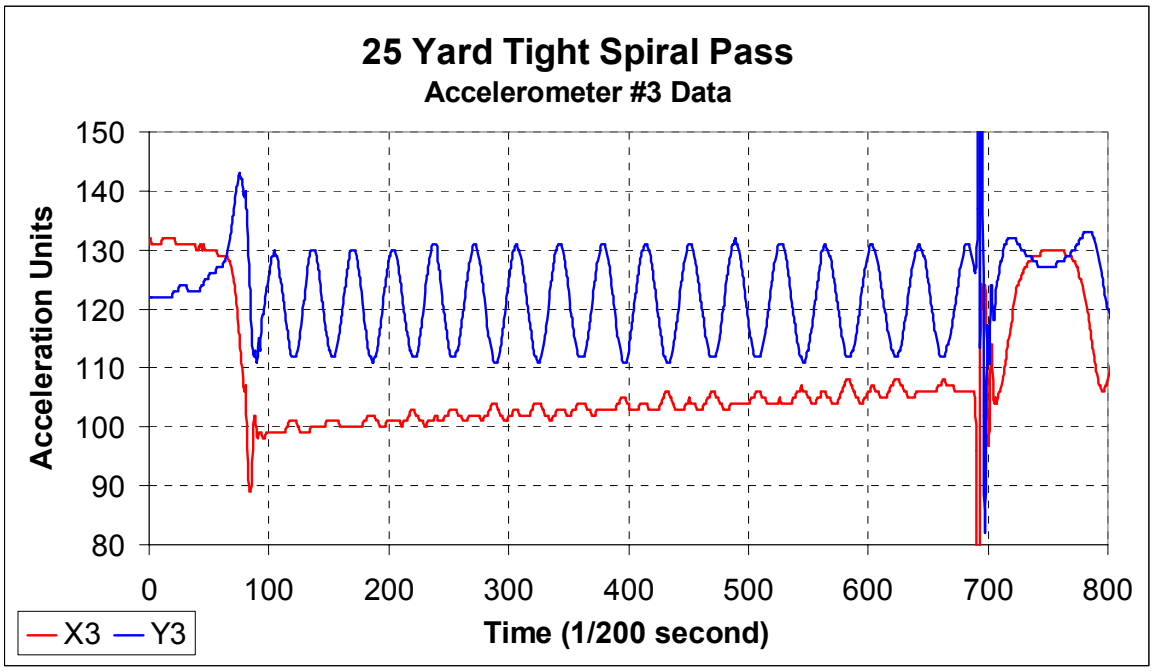


Figure 32 – 25 Yard "Dropped" Forward Pass

Similar to the data seen in the previously discussed roll condition test, X3 Exhibits predictable behavior. A sudden and abrupt change of the acceleration of X3, that gradually over time drifts back closer to it's quiescent value, followed by a chaotic event and then some unpredictable oscillations. After some interpretation, what is visible is that at the point in time the ball is released a roll condition is initiated. The angular velocity of the roll decreases over time as the ball flies to the target. The oscillation in Y3 is again characteristic of the wobble of the pass, as the desirable "spiral" condition of the forward pass was not ideal. It can be seen in the Y3 data that the wobble changes over time as the roll speed is reduced. The frequency is gradually decreasing, and the amplitude is slightly increasing. A longer pass thrown by a stronger individual ought to show continued loss "spiral integrity" as the ball fights aerodynamic forces.

The final test preformed was to see if undesirable pass characteristics could be discerned and differentiated from good characteristics, the good characteristics being a fast tight spiral with little wobble. Seen below in Figure 33 is what is usually referred to as an "end-over-end" or "tumbling" pass, which is a result of little if any roll, and excessive yawing and/or pitching.

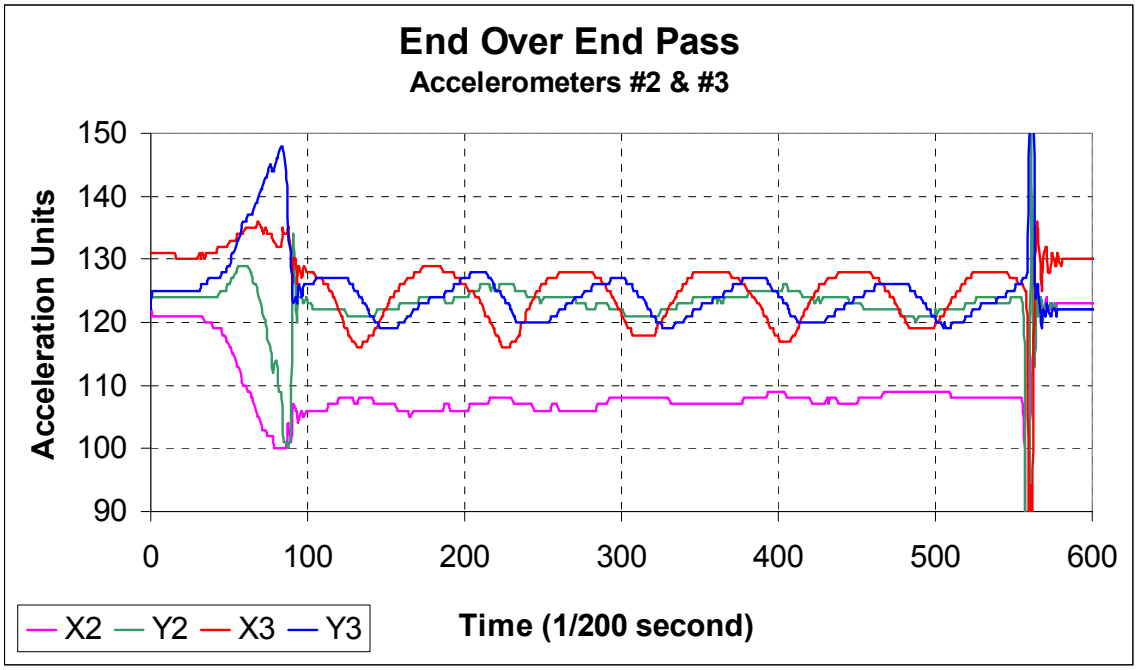


Figure 33 – 20 Yard "Dropped" Tumbling Pass

This pass initiated be holding the ball at its “point” and flipping it high into the air with an underhand toss. An effort was made to impart as little roll on the ball as possible. It can be seen that Y2, X3, and Y3 have and oscillatory behavior, but of small magnitude. The data of X2 however shows that same sort of characteristics that was seen in X3 in the previous examples, that being that large change in acceleration, gradually decaying back nearing to the quiescent value.

Discussion:

The data presented in the previous section focused primarily on the recordings of the accelerometers located at the periphery of the ball, and paid little attention to the accelerometer at the centroid. The reasons for this are two-fold;

1. The accelerations at the centroid are relatively small when compared to the values of the other 2 accelerometers
2. The PCB that records X1, Y1, X2, and Y2 seems to have an intermittent problem that I'm unable to fully diagnose.

To elaborate on the second point, the electronics seem to function perfectly while sitting on a workbench, or collecting random data such as violently shaking the ball, the impact between the ball and the ground, some good data like the "spin" test data that was presented in Figure 28 (page 30), the calibration roll test, etc. Where the problem seems to manifest itself is during an actual forward pass. As you can see below in Figure 34, the data seems to collect without variation, and then suddenly begins to respond. Fortunately for the sake of this project the data that is invalid is of the least interest for the test performed, but it is a problem that must be solved before the football flight data recorder can be declared complete. My primary theory for the problem is that a soldered connection is losing contact somewhere when subjected to the sustained high G's imparted by a fast roll condition of the ball.

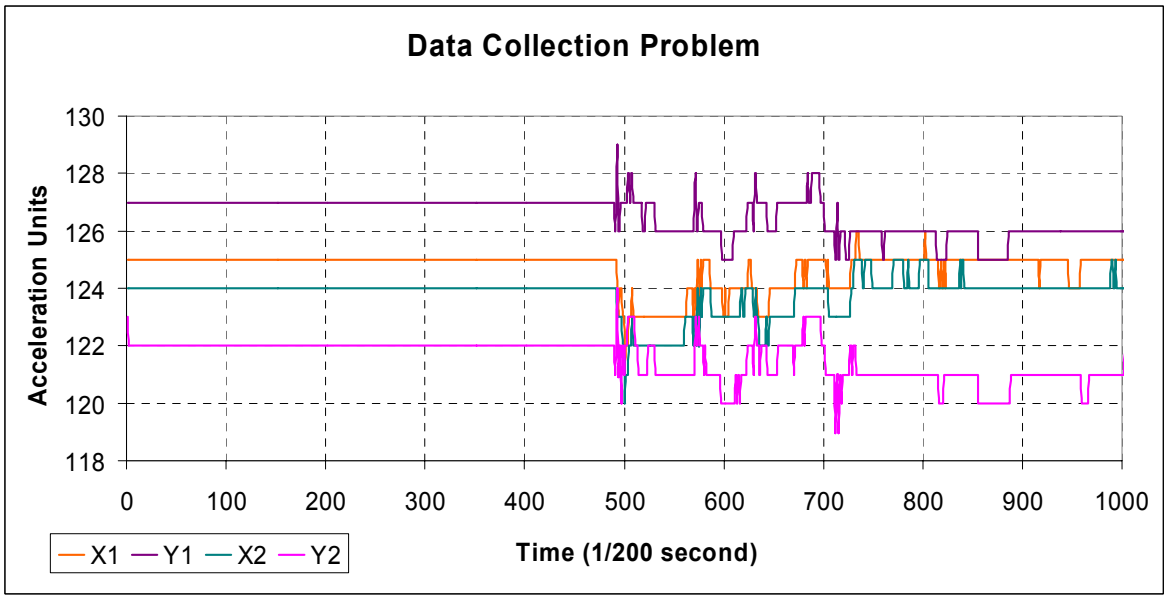


Figure 34 – Data Recording Problem

Conclusions and Recommendations:

This project proved to be a much more substantial learning experience that I originally thought it would be. The data recorder that I designed served its purpose, however it if far from being optimized. If I were to continue development, the next iteration would see the following improvements:

- More optimized power source, the 9v batteries work well, but they are big and heavy
- Instead of through hole resistors and capacitors, I'd switch to surface mountable chip versions
- Surface mount IC's for the Pic and EEPROMS
- An additional EEPROM per channel, which would allow 400 samples per second to be recorded
- Wireless serial communication for transmitting the data from memory to a PC for analysis, either with an infrared link or RF communication.
- Better integration into the ball itself, with the switches and download ports (if necessary) available on the exterior of the ball.
- More development of a trigger algorithm that would allow for an indefinite window for data capture
- A custom downloading application that will have 2 way communication with the ball to start the download only when it is ready to receive the data
- Scale the accelerometers to increase resolution
- Diagnose and repair the intermittent data collection problem

The design being what it was, I feel that it was a good solid foundation on which to build a football analysis/simulation system. The goal of this project was to capture 3 axis of acceleration at the centroid of the ball, as well as attempt to measure spin and wobble that are important contributors to a balls in flight characteristics. I believe that all those goals were met. This project taught me a great deal about integrating an electronic measurement system to capture real world events, electronics in general, microprocessors in particular, and also, and possibly most importantly, that it is possible to look forward to and enjoy working on engineering problems.

Endnotes:

1. **Accelerometer for Football Aerodynamics Study;** *Francis, David and Narayana Sundaram*, SUNY at Buffalo, 2001
2. **Analog Devices,** www.analog.com
3. ibid
4. ibid
5. ibid
6. ibid
7. ibid
8. ibid
9. ibid
10. ibid
11. ibid
12. **Filter Pro**; *Texas Instruments,* www.ti.com
13. **MicroEngineering Labs**; www.melabs.com
14. ibid

References:

Accelerometer for Football Aerodynamics Study; *Francis, David and Narayana Sundaram*, SUNY at Buffalo, 2001

The Art of Electronics 2nd Edition; *Horowicz, Paul and Winfield Hill*, Cambridge University Press, 1989, Reprinted 2001

http://www.analog.com

http://www.digikey.com

http://www.expresspcb.com

http://http://www.melabs.com

http://www.ti.com

PicBasic Pro Compiler; *microEngineering Labs, Inc*., 2002

Programming and Customizing PICmicro Microcontrollers 2nd Edition; *Predko, Mike*, McGraw-Hill, 2001

Acknowledgements:

The completion of this project would have been impossible without the input and guidance of the following individuals and/or institutions (in alphabetical order):

- Mr. Daniel P. Fuglewicz
- Mr. James Hussar
- Dr. Venkat Krovi
- Dr. William Rae
- Mr. David G. Schabel
- Veridian Engineering

The all have my utmost thanks and appreciation for their advice, moral support, understanding, and any other burdens which I bestowed upon them.

Appendix A - Parts Lists:

Parts List

Plan A

Item	Qty Rqd	Part Description	package	unit price	sub total
1	22	0.1 uF Capacitor, Ceramic	.1" leads	$0.16	$3.52
2	2	22 pF Capacitor, Ceramic	.1" leads	$0.07	$0.14
3	8	0.022 uF Capacitor, Ceramic	.1" leads	$0.18	$1.44
4	8	0.033 uF Capacitor, Ceramic	.1" leads	$0.18	$1.44
5	1	47 uF Capacitor, Electrolytic	.1" leads	$2.85	$2.85
6	2	ADXL250JQC Accelerometer	14 pin	$17.00	$34.00
7	1	LM7805 Voltage Regulator	TO-220	$0.48	$0.48
8	1	LM358 Op-Amp	8 pin SOIC	$0.50	$0.50
9	1	20 MHz Crystal	HC-49U	$0.64	$0.64
10	2	ADXL210JQC Accelerometer	14 pin	$17.00	$34.00
11	1	PIC16C877 Microprocessor	44 pin QFP	$18.99	$18.99
12	1	Bi-Color LED	5mm	$0.90	$0.90
13	1	Pushbutton Switch	NO	$1.00	$1.00
14	1	128k SRAM		$3.49	$3.49
15	12	4.7k Resistor 5%	1/4 watt	$0.06	$0.67
16	12	6.27k Resistor 1%	1/4 watt	$0.19	$2.28
17	4	13.9k Resistor 1%	1/4 watt	$0.19	$0.76
18	6	10k Resistor 5%	1/4 watt	$0.06	$0.34
19	1	500 ohm Resistor 5%	1/4 watt	$0.06	$0.06
20	2	Flip Flop		$2.47	$4.94
21	1	Custom PCB		$152.00	$152.00
	90			**Total Cost**	**$218.46**

Parts List

Plan B

Item	Qty Rqd	Part Description	package	unit price	sub total
1	12	0.1 uF Capacitor, Ceramic	.1" leads	$0.16	$1.92
2	2	22 pF Capacitor, Ceramic	.1" leads	$0.07	$0.14
3	2	0.022 uF Capacitor, Ceramic	.1" leads	$0.18	$0.36
4	2	0.033 uF Capacitor, Ceramic	.1" leads	$0.18	$0.36
5	1	47 uF Capacitor, Electrolytic	.1" leads	$2.85	$2.85
6	1	ADXL250JQC Accelerometer	14 pin	$17.00	$17.00
7	1	LM7805 Voltage Regulator	TO-220	$0.48	$0.48
8	1	LM358 Op-Amp	8 pin SOIC	$0.50	$0.50
9	1	20 MHz Crystal	HC-49U	$0.64	$0.64
10	2	AT25320 Serial EEPROM	8 pin SOIC	$3.21	$6.42
11	1	PIC16C717 Microprocessor	18 pin SOIC	$2.50	$2.50
12	1	Bi-Color LED	5mm	$0.90	$0.90
13	1	Pushbutton Switch	NO	$1.00	$1.00
14	1	Pushbutton Switch	NC	$1.00	$1.00
15	3	4.7k Resistor 5%	1/4 watt	$0.06	$0.17
16	4	6.27k Resistor 1%	1/4 watt	$0.19	$0.76
17	2	13.9k Resistor 1%	1/4 watt	$0.19	$0.38
18	2	10k Resistor 5%	1/4 watt	$0.06	$0.11
19	1	500 ohm Resistor 5%	1/4 watt	$0.06	$0.06
20	1	Custom PCB		$15.50	$15.50
	42			**Total Cost**	**$46.72**

Parts List

Plan C

Item	Qty Rqd	Part Description	package	unit price	sub total
1	20	0.1 uF Capacitor, Ceramic	.1" leads	$0.16	$3.20
2	2	22 pF Capacitor, Ceramic	.1" leads	$0.07	$0.14
3	4	0.022 uF Capacitor, Ceramic	.1" leads	$0.18	$0.72
4	4	0.033 uF Capacitor, Ceramic	.1" leads	$0.18	$0.72
5	1	47 uF Capacitor, Electrolytic	.1" leads	$2.85	$2.85
6	2	ADXL250JQC Accelerometer	14 pin	$17.00	$34.00
7	1	LM7805 Voltage Regulator	TO-220	$0.48	$0.48
8	2	LM358 Op-Amp	8 pin SOIC	$0.50	$1.00
9	1	20 MHz Crystal	HC-49U	$0.64	$0.64
10	4	AT25640 Serial EEPROM	8 pin DIP	$3.21	$12.84
11	1	PIC16C876 Microprocessor	28 pin DIP	$14.98	$14.98
12	1	Bi-Color LED	5mm	$0.90	$0.90
13	1	Pushbutton Switch	NO	$1.00	$1.00
14	1	Pushbutton Switch	NC	$1.00	$1.00
15	6	4.7k Resistor 5%	1/4 watt	$0.06	$0.34
16	8	6.27k Resistor 1%	1/4 watt	$0.19	$1.52
17	4	13.9k Resistor 1%	1/4 watt	$0.19	$0.76
18	2	10k Resistor 5%	1/4 watt	$0.06	$0.11
19	1	500 ohm Resistor 5%	1/4 watt	$0.06	$0.06
20	2	Custom PCB		$20.67	$20.67
21	1	10 position dip switch		$0.79	$0.79
22	1	SPDT slide switch		$1.15	$1.15
23	2	9v battery		$3.65	$7.30
24	2	1/8" phono jack female cable		$2.58	$5.16
	74			**Total Cost**	**$72.54**

Appendix B - Data Sheets:

Microchip:

16F876/16F877:
http://www.microchip.com/download/lit/pline/picmicro/families/16f87x/30292c.pdf

16C717
http://www.microchip.com/download/lit/pline/picmicro/families/16c71x/41120b.pdf

Atmel

AT25640 EEPROM
http://www.atmel.com/dyn/resources/prod_documents/doc0675.pdf

Cypress

SRAM
http://rocky.digikey.com/WebLib/Cypress/Web%20Data/CY62128BLL.pdf

Texas Instruments

Flip Flop
http://www-s.ti.com/sc/psheets/scls148e/scls148e.pdf

Analog Devices

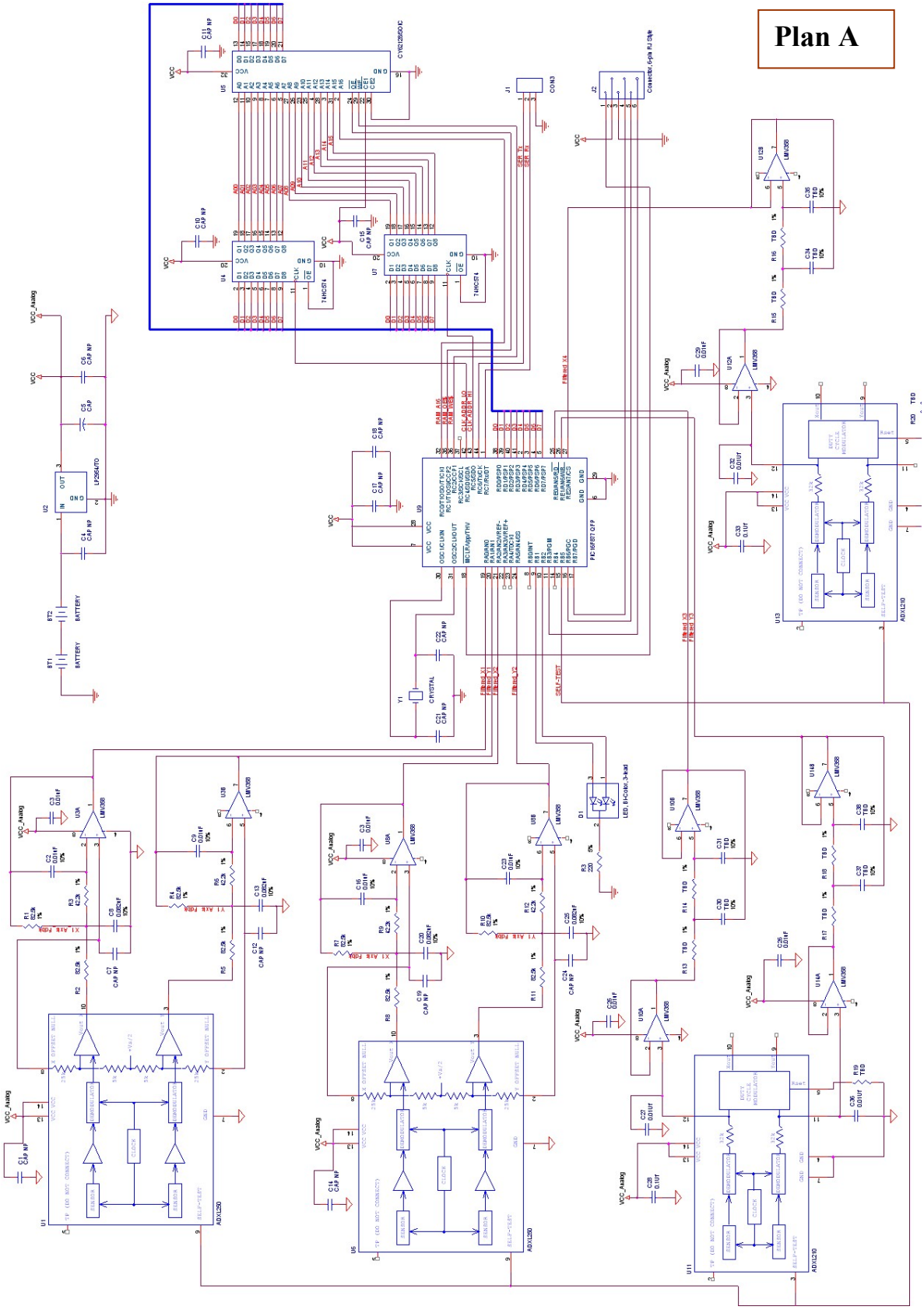
Accelerometer ADXL202/210
http://www.analog.com/UploadedFiles/Datasheets/70885338ADXL202_10_b.pdf

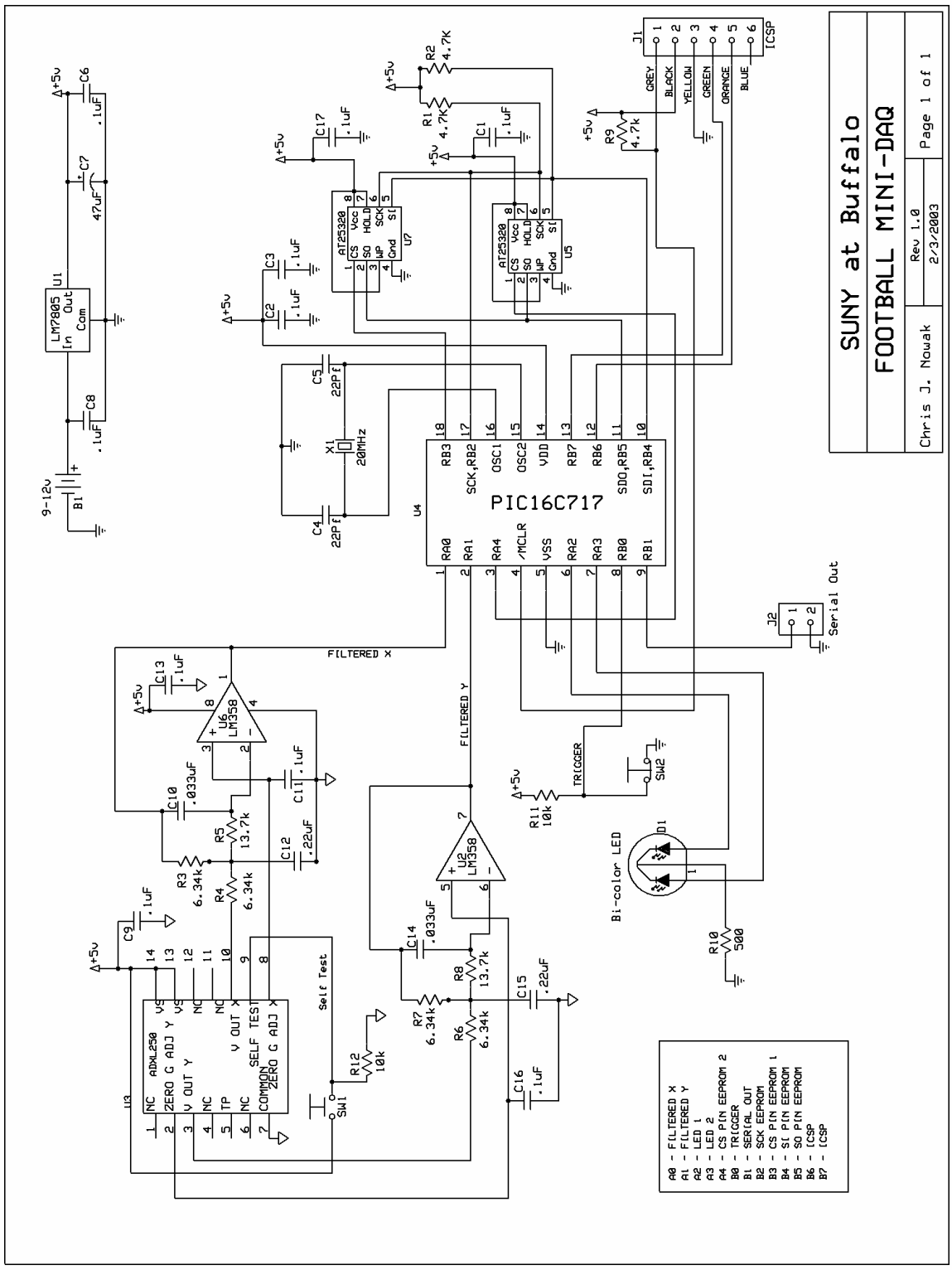
Accelerometer ADXL250
http://www.analog.com/UploadedFiles/Datasheets/573918736ADXL150_250_0.pdf

National Semiconductor
Dual Op-Amp
http://www.national.com/ds/LM/LM158.pdf

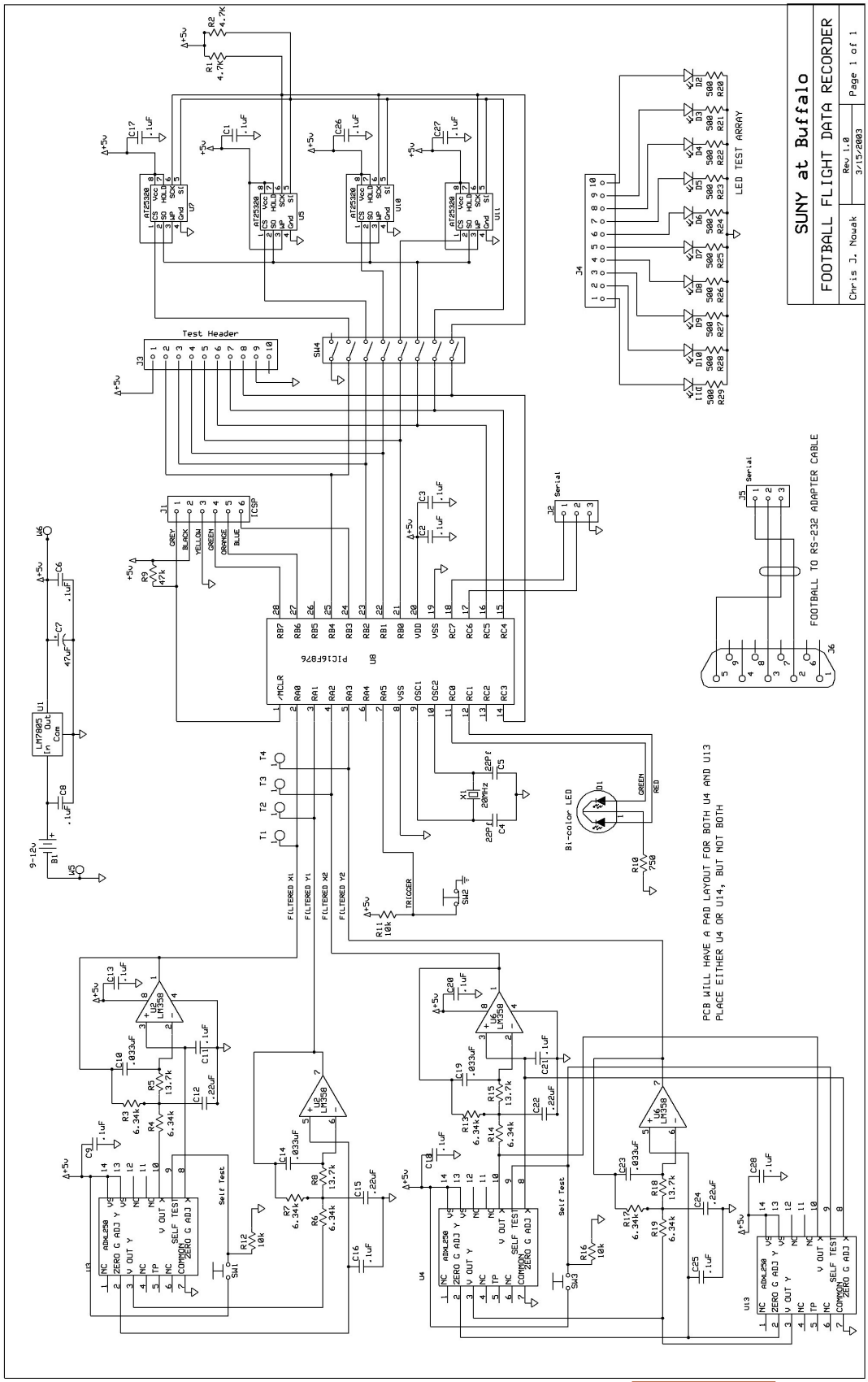
Voltage Regulator
http://www.national.com/ds/LM/LM7512C.pdf

Appendix C – Schematics:




SUNY at Buffalo
FOOTBALL MINI-DAQ
Chris J. Nowak
Rev 1.0
2/3/2003
Page 1 of 1
PIC16C717
ADXL250
LM358
LM7805
AT25320
FILTERED X
FILTERED Y
TRIGGER
Serial Out
Bi-color LED
Self Test
ICSP
A0 - FILTERED X
A1 - FILTERED Y
A2 - LED 1
A3 - LED 2
A4 - CS PIN EEPROM 2
B0 - TRIGGER
B1 - SERIAL OUT
B2 - SCK EEPROM
B3 - CS PIN EEPROM 1
B4 - SI PIN EEPROM
B5 - SO PIN EEPROM
B6 - ICSP
B7 - ICSP

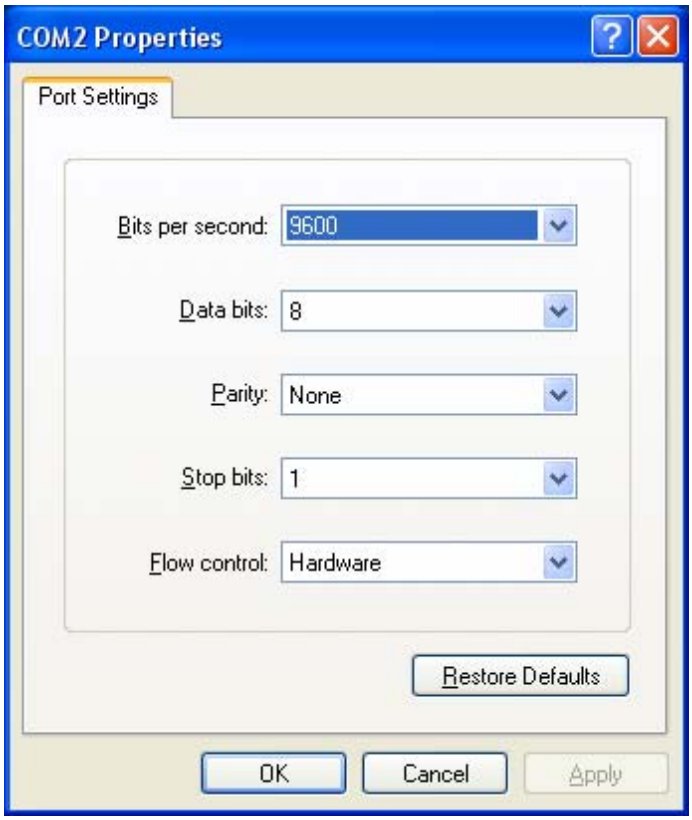
Plan B


SUNY at Buffalo
FOOTBALL FLIGHT DATA RECORDER
Chris J. Nowak
Rev 1.0
3/15/2003
Page 1 of 1
Test Header
LED TEST ARRAY
FOOTBALL TO RS-232 ADAPTER CABLE
PCB WILL HAVE A PAD LAYOUT FOR BOTH U4 AND U13
PLACE EITHER U4 OR U14, BUT NOT BOTH
Bi-color LED
Self Test

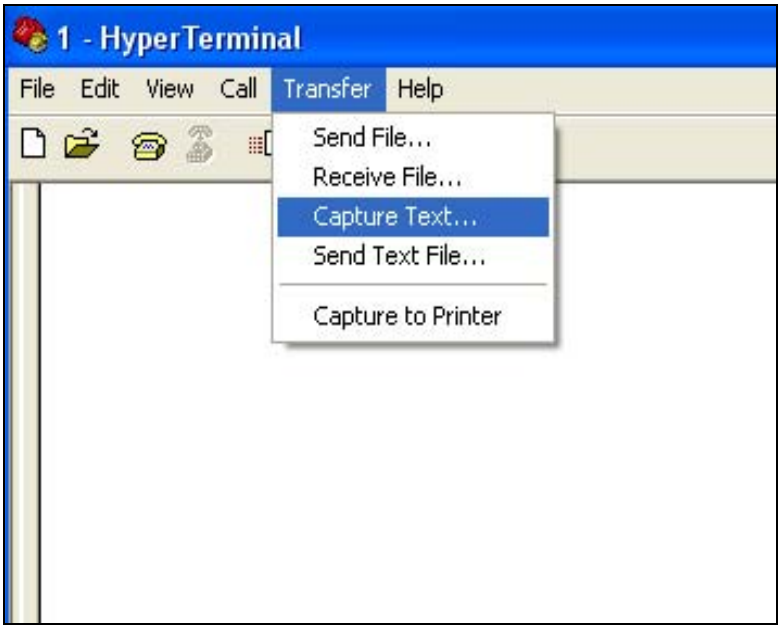
Plan C

Appendix D - Operating Instructions:

1. Unzip football
2. Slide power switch to ON (I) position
3. LEDs will cycle red, then amber then green
4. While LED is green, data is collecting
5. LED will hold steady red at end of data collection
6. LED will Flash red to signal that data is available for download
7. on a PC with Microsoft Windows 95 or better, start HyperTerminal
8. Unzip and open the football
9. Connect the Data Cable to Connector #1 inside the football
10. Set up a connection to Com x (whatever port the cable is connected to)
11. Setting are as shown below



12. In the “Transfer” pull down, select capture text



13. Enter a file name

14. When all data is captured, close HyperTerminal

15. Open the Data File in a Spreadsheet and view and/or graph data.

Appendix E - Microcontroller Code:

Plan A

```
' PicBasic Pro program to log result of
' 7 channel 8-bit A/D conversion to SRAM
'
' Connect analog input to (AN0,1,2,4,5,6,7)

' Define ONINT_USED to allow use of the boot loader.
' This will not affect normal program operation.
DEFINE      ONINT_USED      1

' Define ADCIN parameters
DEFINE      ADC_BITS        8       ' Set number of bits in result
DEFINE      ADC_CLOCK               3       ' Set clock source (3=rc)
DEFINE      ADC_SAMPLEUS    50      ' Set sampling time in uS

' Define Clock Speed
DEFINE OSC 20

' define the serial transmit pin to PortC bit 6
DEFINE DEBUG_REG PORTC
DEFINE DEBUG_BIT 6

' Define baud rate for serial debug
DEFINE DEBUG_BAUD 9600

' Define serial debug mode for inverted
DEFINE DEBUG_MODE 1

advalX1             VAR   WORD              ' Create adval to store result of AN0
advalY1             VAR   WORD              ' Create adval to store result of AN1
advalY2             VAR   WORD              ' Create adval to store result of AN2
advalX3             VAR   WORD              ' Create adval to store result of AN4
advalY3             VAR   WORD              ' Create adval to store result of AN5
advalX4             VAR   WORD              ' Create adval to store result of AN6
advalY4             VAR   WORD              ' Create adval to store result of AN7

accdata         VAR BYTE            'create accdata to count which accel channel to write
to memory
addlow          VAR   BYTE          'create addlow to generate low byte of sram address
addhigh             VAR BYTE            'create addhigh to generate high byte of sram
address
add16           VAR BIT                 'create addlow to generate A16 of sram
address

triglow         VAR BYTE            'create triglow to store low byte of trigger address
trighigh        VAR   BYTE          'create trighigh to store high byte of trigger address
```

```
trig16          VAR BIT                 'create trig16 to store A16 value of trigger
address

triggeron       VAR BIT                 'create triggeron to indicate trigger status

scratch         VAR BYTE            'dummy variable for misc

ClkLowReg   VAR PORTC.4             'rename clock pin on low register
ClkHiReg    VAR   PORTC.5           'rename clock pin on high register

GreenLED    VAR PORTB.2             'rename pine to control green led
RedLED              VAR PORTB.1             'rename pin to control red led
                                            'LED is amber when red and green are
both on
WriteRAM    VAR   PORTC.2           'write enable for SRAM
ReadRAM             VAR PORTC.1             'object enable for SRAM

'*************************************************************
TrigValu    CON   245               '****  set trigger here for Y3  ****

'*************************************************************


        TRISA = %11111111 ' Set PORTA to all input
        TRISE = %11111111  ' Set PORTE to all input
        TRISD = %00000000 ' Set PORTD to all output


        '?????????????????????????????????????????????????????


        ADCON1 = 0  ' Set PORTA and PORTE analog and right justify result



        '?????????????????????????????????????????????????????

        accdata = 0                 ' Set initial state of accdata



        Low RedLED                  ' turn off red LED on PORTB.1
        High GreenLED       ' Turn on Green LED connected to PORTB.2
        Low PORTC.0                 ' set ram a16 to zero
        triggeron = 0


Readloop:       For Add16 = 0 TO 1
                        For AddHigh = 0 TO 255
```

```
For AddLow = 0 TO 255
IF addlow = 191 AND addhigh = 99 AND add16 = 1
                                    'turns LED amber. this
                                    'the entireevent require
IF addlow = 255 AND addhigh = 255 AND add16 =
                                    'turns LED Green
IF accdata = 9 Then accdata = 0
                                    'tracks which
Low PORTC.0                         'set A16 of
Poke PORTD, AddLow  'Set Port D to low byte of
Low ClkHiReg                        'Send nothing to add hi
High ClkLowReg                      'enable low register to
PauseUs 5                           'DELAY FOR 5 uS
Low ClkLowReg                       'lock address low byte
Poke PORTD, AddHigh                 'Set Port D to high byte
High ClkHiReg                       'enable high register to
PauseUs 5                           'DELAY FOR 5 uS
Low ClkHiReg                        'lock address in high
IF Add16 = 1 Then High PORTC.0
                                    'set ram A16 to
PauseUs 5                           'DELAY FOR 5 uS
```

Then High RedLED

and the next line ensure that

no wraparound in the buffer

1 Then Low RedLED

channel to write to memory

SRAM low

address

register

recieve

in low register

of address

recieve

byte of register

1

```
                                        High WriteRAM           'enable writing to
memory address

                                        'GoSub AccelData
                                        'IF triggeron = 1 Then GoTo skiptrigger
                                        'GoSub AccelTrigger

                SkipTrigger:


                                        Poke PORTD, addlow      'write mem address low
to memory address
                                        'IF accdata = 1 Then Poke PORTD, advalY1
        'write accel y1 to memory address
                                        'IF accdata = 2 Then Poke PORTD, advalY2
        'write accel y2 to memory address
                                        'IF accdata = 3 Then Poke PORTD, advalX3
        'write accel x3 to memory address
                                        'IF accdata = 4 Then Poke PORTD, advalY3
        'write accel y3 to memory address
                                        'IF accdata = 5 Then Poke PORTD, advalX4
        'write accel x4 to memory address
                                        'IF accdata = 6 Then Poke PORTD, advalY4
        'write accel y4 to memory address
                                        'IF accdata = 7 Then Poke PORTD, %00000000
        'write zeros to memory address

                                        'Debug DEC advalx1,9, DEC advaly1,9, DEC
advaly2,9,_
                                         'DEC advalx3,9, DEC advaly3,9, DEC advalx4,9,
DEC advaly4,9, DEC addhigh,9, DEC addlow,10,13

                                'send accelerometer data to hyperterminal


                                        PauseUs 5               'DELAY FOR 5 uS

                                        'accdata = accdata + 1


                                        Low WriteRAM            'lock in data to memory
address

                                        PauseUs 5

                                        TRISD = %11111111 ' Set PORTD to all input

                                        Peek PORTD, scratch
```

```
                                        Debug "Ram16 = ",add16,9, "RAM HI =
",addhigh,9, "RAM LOW = ",addlow,9,"DATA = ",scratch,10,13


                                        PauseUs 5

                                        TRISD = %00000000 ' Set PORTD to all output

                                        Next AddLow

                                Next AddHigh

                        Next Add16


                'IF triggeron = 1 Then WriteLoop

                'GoTo ReadLoop                                  'endless read loop if no
trigger detected


WriteLoop:      'Debug "TRIGGER CONDITION WAS MET",10,13
                        'Debug "DATA SAVED TO RAM",10,13
                        'Debug "TRIGGER MEMORY ADDRESS IS ", BIN trig16,9, BIN
trighigh,9, BIN triglow,10,13

Blinker:        For scratch = 1 TO 20
                        Low GreenLED                            'temporary blinker to
indicate program finished
                        High RedLED
                        Pause 500
                        Low RedLED
                        Pause 500
                        Next Scratch


End




'Subroutines

'AccelData:             ADCIN 0, advalX1     ' Read channel 0 to adval
                        '       ADCIN 1, advalY1     ' Read channel 1 to adval
```

```
                    '       ADCIN 2, advalY2    ' Read channel 2 to adval
                    '       ADCIN 4, advalX3    ' Read channel 4 to adval
                    '       ADCIN 5, advalY3    ' Read channel 5 to adval
                    '       ADCIN 6, advalX4    ' Read channel 6 to adval
                    '       ADCIN 7, advalY4    ' Read channel 7 to adval

                    '       Return

'AccelTrigger: IF advalY3 > trigValu Then triggeron = 1
'                           triglow = addlow
'                           trighigh = addhigh
'                           trig16 = add16
'
'                           Return
```

Plan B

Code was never written

Plan C

Board 1

```
' PicBasic Pro program to log result of
' 4 channel 8-bit A/D conversion to SRAM
'
' Connect analog input to (AN0,1,2,3)

' Define ONINT_USED to allow use of the boot loader.
' This will not affect normal program operation.
DEFINE      ONINT_USED      1

DEFINE OSC 20

' define the serial transmit pin to PortC bit 6
DEFINE DEBUG_REG PORTC
DEFINE DEBUG_BIT 6
' Define baud rate for serial debug
DEFINE DEBUG_BAUD 9600
' Define serial debug mode for inverted
DEFINE DEBUG_MODE 1

' Define ADCIN parameters
DEFINE      ADC_BITS        8       ' Set number of bits in result
DEFINE      ADC_CLOCK               3       ' Set clock source (3=rc)
DEFINE      ADC_SAMPLEUS    50      ' Set sampling time in uS



    INCLUDE "modedefs.bas"

CS1         VAR         PORTB.4     ' Chip select pin
CS2         VAR         PORTB.2     ' Chip select pin
CS3         VAR         PORTB.1     ' Chip select pin
CS4         VAR         PORTB.0     ' Chip select pin

SCK         VAR   PORTC.3       ' Clock pin
SI          VAR         PORTC.4     ' Data in pin
SO          VAR         PORTC.5     ' Data out pin

addr        VAR         WORD        ' Address
B0          VAR         BYTE        ' Data
B1          VAR         BYTE        ' Data
B2          VAR         BYTE        ' Data
B3          VAR         BYTE        ' Data

adX1        VAR         BYTE        ' Create adval to store result of AN0
adY1        VAR         BYTE        ' Create adval to store result of AN1
adX2        VAR         BYTE        ' Create adval to store result of AN2
```

```
adY2        VAR         BYTE            ' Create adval to store result of AN3

dumX1       VAR         BYTE            ' Create adval to store adX1
dumY1       VAR         BYTE            ' Create adval to store adY1
dumX2       VAR         BYTE            ' Create adval to store adX2
dumY2       VAR         BYTE            ' Create adval to store adY2

REDLED      VAR         PORTC.0         ' Red LED pin
GREENLED    VAR         PORTC.1         ' Green LED pin

memloop     VAR         WORD            ' create memloop to store # of address'


        '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
        '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~

                memloop = 2000              ' Set memloop to number of address'

        '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
        '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~

    TRISB.0 = 0         ' Set CS4 to output
    TRISB.1 = 0         ' Set CS3 to output
    TRISB.2 = 0         ' Set CS2 to output
    TRISB.4 = 0         ' Set CS1 to output

    ADCON1 = %00000010          ' Set PORTA analog

    High REDLED                 ' led turns red

    Pause 1000          ' Wait 1 second

    High GREENLED               ' led turns amber

    Pause 1000                  ' Wait 1 second

    Low REDLED                  ' led turns green

    Pause 1000                  ' Wait 1 second

    Debug "DATA COLLECTION BEGINNING",10,13

    GoSub AccelData

    GoSub OvrSmplX
```

```
        GoSub eewrite1
        GoSub eewrite3

        dumY1 = adY1
        dumY2 = adY2

        For addr = 1 TO memloop         ' Loop "memloops" times

                Pause 2

                GoSub AccelData

                GoSub OvrSmplY

            GoSub eewrite2          ' Write to SEEPROMs
            GoSub eewrite4              ' Write to SEEPROMs

            dumX1 = adX1
            dumX2 = adX2

            Pause 2                                     ' Delay 2ms

            GoSub AccelData                 ' Read Accelerometers

            GoSub OvrSmplX

            GoSub eewrite1              ' Write to SEEPROMs
            GoSub eewrite3              ' Write to SEEPROMs

            dumY1 = adY1
            dumY2 = adY2

        Next addr

        Low GREENLED

        High REDLED

        Debug "DATA COLLECTION OVER",10,13

        Pause 1000                                  'Wait 1 second

        Debug "DATA DUMP BEGINNING",10,13


          '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
loop:   Debug "Address",9, "X1",9, "Y1",9, "X2",9, "Y2",13
```

```
            For addr = 0 TO memloop             ' Loop "memloop" times

          GoSub eeread1         ' Read from SEEPROM #1
          GoSub eeread2         ' Read from SEEPROM #2
          GoSub eeread3         ' Read from SEEPROM #3
          GoSub eeread4         ' Read from SEEPROM #4

          Debug DEC addr,9, DEC B0,9, DEC B1,9, DEC B2,9, DEC B3,13

          Low REDLED
          Pause 50
          High REDLED

        Next addr

        High GREENLED

        Debug "DATA DUMP OVER",10,13

        Pause 5000

        Low GreenLED

      GoTo loop

' Subroutines to read data from addr in serial EEPROM

eeread1:        CS1 = 0                         ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$03, addr.byte1, addr.byte0]
                                                                ' Send read
command and address
        ShiftIn SO, SCK, MSBPRE, [B0]     ' Read data
        CS1 = 1                         ' Disable
        Return

eeread2:        CS2 = 0                         ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$03, addr.byte1, addr.byte0]
                                                                ' Send read
command and address
        ShiftIn SO, SCK, MSBPRE, [B1]     ' Read data
        CS2 = 1                         ' Disable
        Return

eeread3:        CS3 = 0                         ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$03, addr.byte1, addr.byte0]
                                                                ' Send read
command and address
        ShiftIn SO, SCK, MSBPRE, [B2]     ' Read data
        CS3 = 1                         ' Disable
```

```
        Return

eeread4:        CS4 = 0                         ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$03, addr.byte1, addr.byte0]
                                                                        ' Send read
command and address
        ShiftIn SO, SCK, MSBPRE, [B3]     ' Read data
        CS4 = 1                         ' Disable
        Return

' Subroutine to write data at addr in serial EEPROM

eewrite1:       CS1 = 0                                 ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$06]         ' Send write enable command
                CS1 = 1                   ' Disable to execute command
        CS1 = 0                         ' Enable
        ShiftOut SI, SCK, MSBFIRST, [$02, addr.byte1, addr.byte0, adx1]
                                                                        ' Send address
and data
                        CS1 = 1                         ' Disable

eewrite2:       CS2 = 0                                 ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$06]         ' Send write enable command
        CS2 = 1                   ' Disable to execute command
        CS2 = 0                         ' Enable
        ShiftOut SI, SCK, MSBFIRST, [$02, addr.byte1, addr.byte0, ady1]
                                                                        ' Send address
and data
        CS2 = 1                         ' Disable

eewrite3:       CS3 = 0                                 ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$06]         ' Send write enable command
        CS3 = 1                   ' Disable to execute command
        CS3 = 0                         ' Enable
        ShiftOut SI, SCK, MSBFIRST, [$02, addr.byte1, addr.byte0, adx2]
                                                                        ' Send address
and data
        CS3 = 1                         ' Disable

eewrite4:       CS4 = 0                                 ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$06]         ' Send write enable command
        CS4 = 1                   ' Disable to execute command
        CS4 = 0                         ' Enable
        ShiftOut SI, SCK, MSBFIRST, [$02, addr.byte1, addr.byte0, ady2]
                                                                        ' Send address
and data
        CS4 = 1                         ' Disable
        Return
```

```
'  Subroutine to read accelerometer data into microprocessor

AccelData:      ADCIN 0, adX1                           ' Read channel 0 to
adx1
                ADCIN 1, adY1                           ' Read channel 1
to ady1
                ADCIN 2, adX2                           ' Read channel 2
to adx2
                ADCIN 3, adY2                           ' Read channel 3
to ady2
                Return

OvrSmplX:       adX1 = (adX1 + dumX1) / 2
                adX2 = (adX2 + dumX2) / 2
                Return

OvrSmplY:       adY1 = (adY1 + dumY1) / 2
                adY2 = (adY2 + dumY2) / 2
                Return

    End
```

Plan C
Board 2

```
' PicBasic Pro program to log result of
' 4 channel 8-bit A/D conversion to SRAM
'
' Connect analog input to (AN0,1,2,3)

' Define ONINT_USED to allow use of the boot loader.
' This will not affect normal program operation.
DEFINE      ONINT_USED      1

DEFINE OSC 20

' define the serial transmit pin to PortC bit 6
DEFINE DEBUG_REG PORTC
DEFINE DEBUG_BIT 6
' Define baud rate for serial debug
DEFINE DEBUG_BAUD 9600
' Define serial debug mode for inverted
DEFINE DEBUG_MODE 1

' Define ADCIN parameters
DEFINE      ADC_BITS        8       ' Set number of bits in result
DEFINE      ADC_CLOCK               3       ' Set clock source (3=rc)
DEFINE      ADC_SAMPLEUS    50      ' Set sampling time in uS



    INCLUDE "modedefs.bas"

CS1         VAR         PORTB.4     ' Chip select pin
CS2         VAR         PORTB.2     ' Chip select pin
CS3         VAR         PORTB.1     ' Chip select pin
CS4         VAR         PORTB.0     ' Chip select pin

SCK         VAR         PORTC.3     ' Clock pin
SI          VAR         PORTC.4     ' Data in pin
SO          VAR         PORTC.5     ' Data out pin

addr        VAR         WORD        ' Address
B0          VAR         BYTE        ' Data
B1          VAR         BYTE        ' Data
B2          VAR         BYTE        ' Data
B3          VAR         BYTE        ' Data

adX1        VAR         BYTE        ' Create adval to store result of AN0
adY1        VAR         BYTE        ' Create adval to store result of AN1
adX2        VAR         BYTE        ' Create adval to store result of AN2
```

```
adY2        VAR         BYTE            ' Create adval to store result of AN3

dumX1       VAR         BYTE            ' Create adval to store adX1
dumY1       VAR         BYTE            ' Create adval to store adY1
dumX2       VAR         BYTE            ' Create adval to store adX2
dumY2       VAR         BYTE            ' Create adval to store adY2

REDLED      VAR         PORTC.0         ' Red LED pin
GREENLED    VAR         PORTC.1         ' Green LED pin

memloop     VAR         WORD            ' create memloop to store # of address'


        '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
        '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~

                memloop = 2000              ' Set memloop to number of address'

        '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
        '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~

    TRISB.0 = 0         ' Set CS4 to output
    TRISB.1 = 0         ' Set CS3 to output
    TRISB.2 = 0         ' Set CS2 to output
    TRISB.4 = 0         ' Set CS1 to output

    ADCON1 = %00000010          ' Set PORTA analog

    High REDLED                 ' led turns red

    Pause 1000          ' Wait 1 second

    High GREENLED               ' led turns amber

    Pause 1000                  ' Wait 1 second

    Low REDLED                  ' led turns green

    Pause 1000                  ' Wait 1 second

    Debug "DATA COLLECTION BEGINNING",10,13

    GoSub AccelData

    GoSub OvrSmplX
```

```
        GoSub eewrite1
        GoSub eewrite3

        dumY1 = adY1
        dumY2 = adY2

        For addr = 1 TO memloop         ' Loop "memloops" times

                Pause 2

                GoSub AccelData

                GoSub OvrSmplY

            GoSub eewrite2          ' Write to SEEPROMs
            GoSub eewrite4              ' Write to SEEPROMs

            dumX1 = adX1
            dumX2 = adX2

            Pause 2                                     ' Delay 2ms

            GoSub AccelData                 ' Read Accelerometers

            GoSub OvrSmplX

            GoSub eewrite1              ' Write to SEEPROMs
            GoSub eewrite3              ' Write to SEEPROMs

            dumY1 = adY1
            dumY2 = adY2

        Next addr

        Low GREENLED

        High REDLED

        Debug "DATA COLLECTION OVER",10,13

        Pause 1000                              'Wait 1 second

        Debug "DATA DUMP BEGINNING",10,13


          '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
loop:   Debug "Address",9, "Z1",9, "X2",9, "Y2",13
```

```
            For addr = 0 TO memloop             ' Loop "memloop" times

          GoSub eeread1         ' Read from SEEPROM #1
          GoSub eeread2         ' Read from SEEPROM #2
          GoSub eeread3         ' Read from SEEPROM #3
          GoSub eeread4         ' Read from SEEPROM #4

          Debug DEC addr,9, DEC B0,9, DEC B2,9, DEC B3,13

          Low REDLED
          Pause 50
          High REDLED

        Next addr

        High GREENLED

        Debug "DATA DUMP OVER",10,13

        Pause 5000

        Low GreenLED

      GoTo loop

' Subroutines to read data from addr in serial EEPROM

eeread1:        CS1 = 0                         ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$03, addr.byte1, addr.byte0]
                                                                ' Send read
command and address
        ShiftIn SO, SCK, MSBPRE, [B0]     ' Read data
        CS1 = 1                         ' Disable
        Return

eeread2:        CS2 = 0                         ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$03, addr.byte1, addr.byte0]
                                                                ' Send read
command and address
        ShiftIn SO, SCK, MSBPRE, [B1]     ' Read data
        CS2 = 1                         ' Disable
        Return

eeread3:        CS3 = 0                         ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$03, addr.byte1, addr.byte0]
                                                                ' Send read
command and address
        ShiftIn SO, SCK, MSBPRE, [B2]     ' Read data
        CS3 = 1                         ' Disable
```

```
        Return

eeread4:        CS4 = 0                         ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$03, addr.byte1, addr.byte0]
                                                                        ' Send read
command and address
        ShiftIn SO, SCK, MSBPRE, [B3]     ' Read data
        CS4 = 1                         ' Disable
        Return

' Subroutine to write data at addr in serial EEPROM

eewrite1:       CS1 = 0                                 ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$06]         ' Send write enable command
                CS1 = 1                   ' Disable to execute command
        CS1 = 0                         ' Enable
        ShiftOut SI, SCK, MSBFIRST, [$02, addr.byte1, addr.byte0, adx1]
                                                                        ' Send address
and data
                        CS1 = 1                         ' Disable

eewrite2:       CS2 = 0                                 ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$06]         ' Send write enable command
        CS2 = 1                     ' Disable to execute command
        CS2 = 0                         ' Enable
        ShiftOut SI, SCK, MSBFIRST, [$02, addr.byte1, addr.byte0, ady1]
                                                                        ' Send address
and data
        CS2 = 1                         ' Disable

eewrite3:       CS3 = 0                                 ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$06]         ' Send write enable command
        CS3 = 1                     ' Disable to execute command
        CS3 = 0                         ' Enable
        ShiftOut SI, SCK, MSBFIRST, [$02, addr.byte1, addr.byte0, adx2]
                                                                        ' Send address
and data
        CS3 = 1                         ' Disable

eewrite4:       CS4 = 0                                 ' Enable serial EEPROM
        ShiftOut SI, SCK, MSBFIRST, [$06]         ' Send write enable command
        CS4 = 1                     ' Disable to execute command
        CS4 = 0                         ' Enable
        ShiftOut SI, SCK, MSBFIRST, [$02, addr.byte1, addr.byte0, ady2]
                                                                        ' Send address
and data
        CS4 = 1                         ' Disable
        Return
```

```
'  Subroutine to read accelerometer data into microprocessor

AccelData:      ADCIN 0, adX1                                   ' Read channel 0 to
adx1
                ADCIN 1, adY1                                   ' Read channel 1
to ady1
                ADCIN 2, adX2                                   ' Read channel 2
to adx2
                ADCIN 3, adY2                                   ' Read channel 3
to ady2
                Return

OvrSmplX:       adX1 = (adX1 + dumX1) / 2
                adX2 = (adX2 + dumX2) / 2
                Return

OvrSmplY:       adY1 = (adY1 + dumY1) / 2
                adY2 = (adY2 + dumY2) / 2
                Return

    End
```