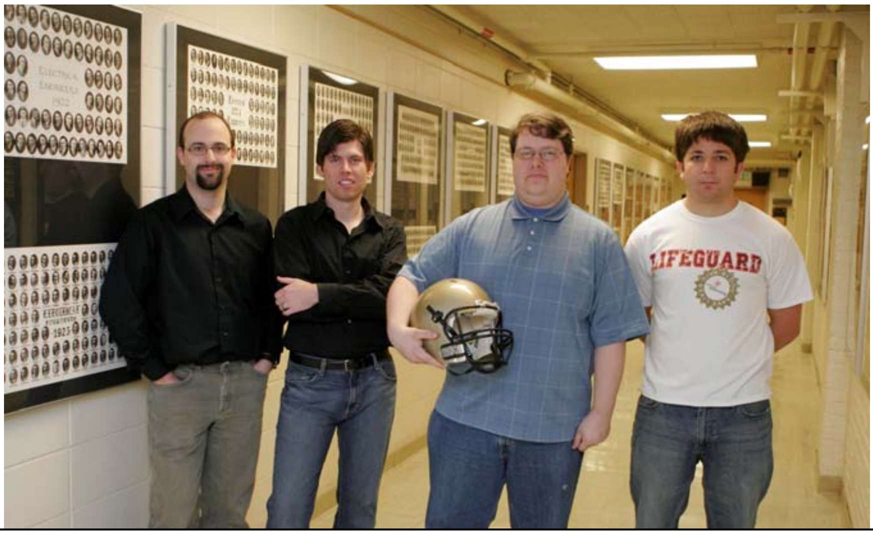
ECE 477 Final Report – Fall 2008
Team 2 – PHI-Master



Andrew Camp, Adam Boeckmann, Michael Olson, Kevin Hughes

Team Members:

#1: Adam Boeckmann **Signature: ____________________ Date: _________**

#2: Andrew Camp **Signature: ____________________ Date: _________**

#3: Kevin Hughes **Signature: ____________________ Date: _________**

#4: Michael Olson **Signature: ____________________ Date: _________**

CRITERION	SCORE	MPY	PTS
Technical content	0 1 2 3 4 5 6 7 8 9 10	3	
Design documentation	0 1 2 3 4 5 6 7 8 9 10	3	
Technical writing style	0 1 2 3 4 5 6 7 8 9 10	2	
Contributions	0 1 2 3 4 5 6 7 8 9 10	1	
Editing	0 1 2 3 4 5 6 7 8 9 10	1	
		TOTAL	

Comments:

TABLE OF CONTENTS

Abstract

The design of the PHI-Master, the instrumented football helmet designed at Purdue University, is detailed at all the design phases in the following report. The first sections will detail the research and theory behind the component selection, followed by the hardware integrations, and finally the software integration. Also one will find the specific functions that were tailored into the design. These integrations include a wireless interface, internal storage for large amounts of data, a friendly user interface in the form of a website, and the series of accelerometers that relay values for the design's use.

1.0 Project Overview and Block Diagram

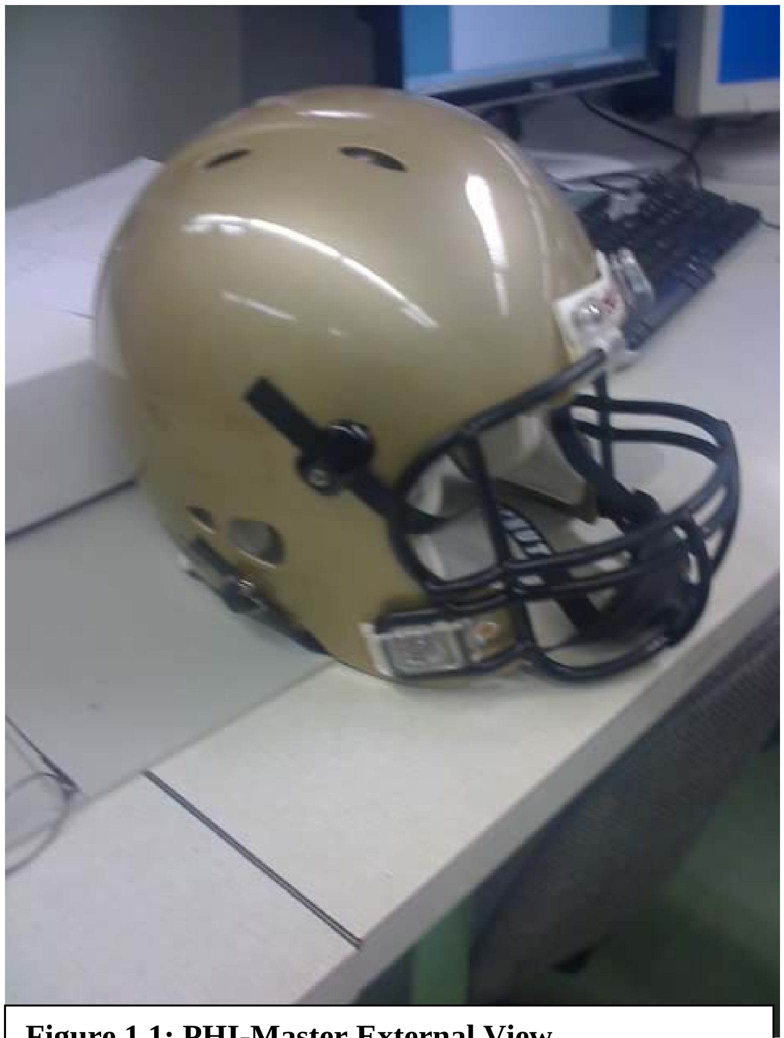


Figure 1.1: PHI-Master External View

The PHI-MASTER is a football helmet outfitted with an electronic system for detecting and reporting on collision magnitude. It is intended to be used by football players and coaching staff to record the magnitude of collisions encountered during tackles and report on collisions that could potentially cause concussions.

There are several subsystems to our product which can be observed on the PCB's mounted inside the football helmet. These subsystems include several accelerometer clusters, a WiFi board, and the main board, icro SD card holder. The WiFi system, provided by the Matchport, allows the user to remotely monitor impact levels encountered during a football game. Additionally this device allows the user to specify what magnitude impact will cause our system to record or report on the impact levels. The system also contains a micro SD card on which every impact over the specified threshold will be recorded, with a timestamp, so that the user's impact level can be observed over the course of several games. All of this functionality is provided through a built in web server on the Matchport along with a java applet for user interaction.

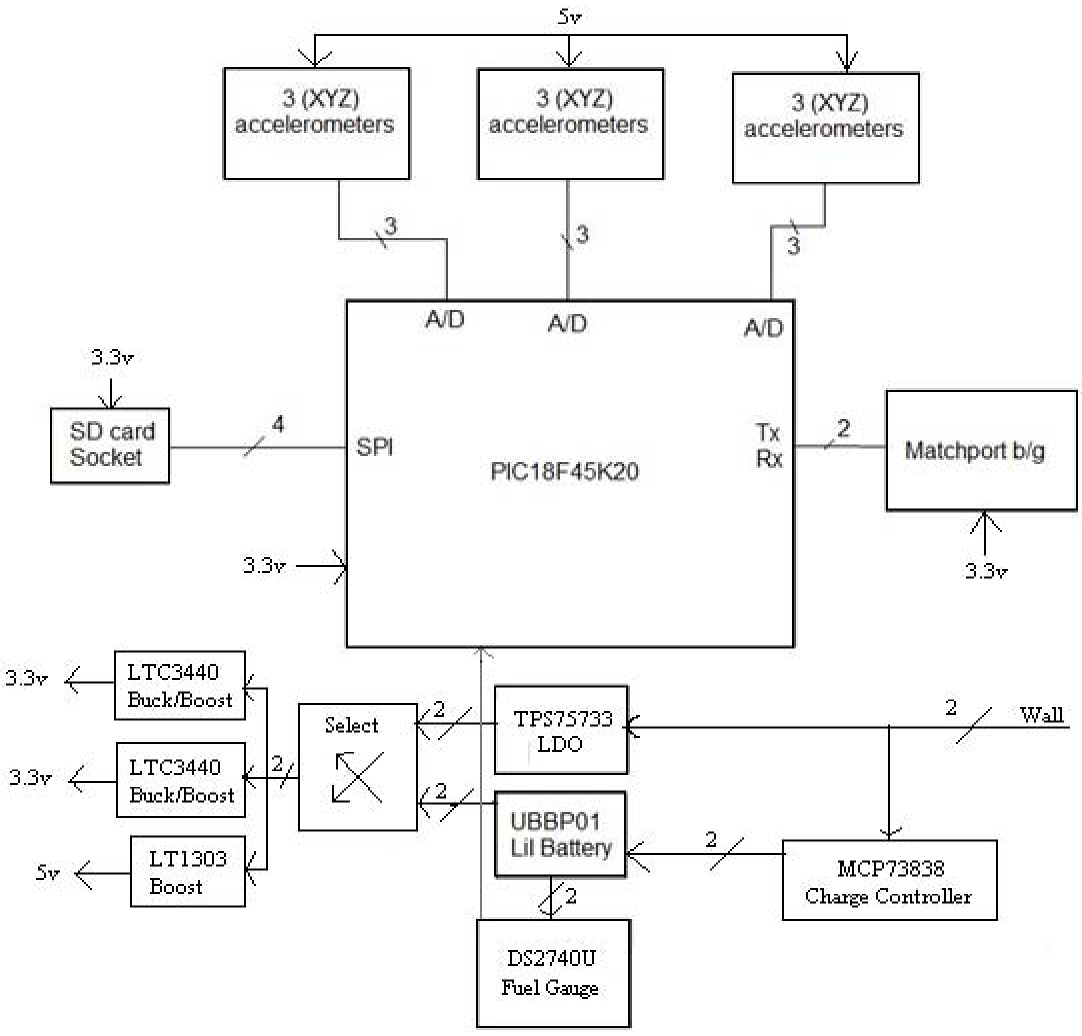


Figure 1.2: PHI-Master block diagram

2.0 Team Success Criteria and Fulfillment

1. An ability to recharge and monitor an integrated (lithium ion) battery.

   We demonstrated the recharge-ability by hooking the battery up to the Power Supply, noting the voltage/current, recharging the battery through our board for some amount of time, and then noted that the voltage/current increased/decreased respectively.

   The monitoring was achieved by keeping a running total of how much charge has been used by the device. This automatically updated in the data tab of the ui.html webpage, as well as was callable from the command "?BS".

2. An ability to sense impact data using 3-D accelerometers.

   We connected to the test.html webpage we have integrated for debugging purposes and issued the command ">DBGAC". This will activate a live feed of the current accelerometer values every 3 seconds.

3. An ability to alert sideline personel when a dangerous hit is received.

   We connected to the ui.html webpage we created for the user's use, and simulated an impact by colliding the helmet with another object. This action generated an alert to the webpage in the form of a message box displaying a potential concussion could have occurred along with the number of Gs of the impact.

4. An ability to log/timestamp impact data on a removable memory device (SD card).

   This PSSC was demonstrated by logging into the ui.html web page and simulating an impact. This generated a locally time stamped version of the impact on the website, as well as stored an internal version on the SD Card. We then issued the ">PRTSD" command and observed the output from the SD Card on the web server.

5. An ability to remotely monitor real-time impact data and tune impact detection thresholds via an embedded web server.

   We will connect to the ui.html and show that the values can be updated through the simple clicking of a button. We can also go to the data tab to observe collisions that surpass the threshold values.

1.0 Constraint Analysis and Component Selection

The problems that can come with designing an instrumented football helmet focus around the player's ability to feel comfortable and the electronic components speed and durability. One has to consider the weight and balance of the total design, the sensitivity of the components that are measuring forces, power use of all the combined electronic components, the distance and nature of the wireless device, and the computational power of the embedded device that will drive the entire system. These constraints will be highlighted throughout the rest of the document.

3.1 Design Constraint Analysis

While already briefly introduced, the design constraints are a major deciding factor in each and every component of the design of the instrumented football helmet. It is very important that the weight of the completed design be as close to the normal weight of a player's original helmet as possible. If the design weighs too much, there is a significant potential for the player's performance to decline because of the reduced reaction time with the added weight on their head or even a potential for the player to get hurt because of the added inertia from the extra components. Along with the weight, one must consider the balance of the design too. If the balance is off by just a fraction, the player could have their head rotated to one side or another while being tackled. The accelerometers must be sensitive enough to accurately measure the force of hits a player receives. A variation of +/- 10 g's could be the difference between a potentially hazardous collision and a concussion causing one. Also, the overall range of the accelerometers is important too. If a player's head gets hit with a force of 70 to 95 g's then they have a potential concussion, so the accelerometers need to be able to reliably record forces well over 95 g's [4]. In order for an instrumented helmet to be beneficial, it needs the ability to talk to the sports trainer or the coach and relay to them hits that may be dangerous. A football field is 100 yards in length, which means the helmet needs to be able to broadcast at least 60 meters distance to communicate with a receiver in the sidelines at the center of the field. The processer chosen needs to be able to handle sending and receiving data, performing various calculations, and storing data on off chip memory all at once. If the microcontroller fails during a game because the computational requirements become too great, a player's injuries may go unnoticed and then worsen because of repeated collisions. All of the components chosen need to be able to withstand impacts taken by the player to avoid device failures. Also, the combined power consumption of all the components needs to be minimal so that a smaller less powerful battery will suffice.

3.2 Computation Requirements

The microcontroller on the instrumented football helmet will be performing multiple computations, sending and receiving data, performing conversions on data, as well as operating various external peripherals. There are 9 ATD converters transforming the outputs of the accelerometer into usable data at any given time. Once the microcontroller receives the values from the ATD converters, it computes the resultant force from the values using the Pythagorean Theorem and compares the result with the user defined threshold for concussion alerting. Once the processor has determined that the concussion threshold has been met, it must access the wireless adaptor and relay the information to the sidelines and process additional data from the accelerometers to have continuous results. The microcontroller also must be able to continuously relay data to the SD card for storage. The battery charge indicator relays information about the charge remaining to the microcontroller in order to keep an up to date charge displayed on the web server for the trainers and coaches to access. In order to keep the operations running smoothly the microcontroller is running at 16 MHz. We arrived at this value because the accelerometers run at 400Hz and the SD Card can run at a maximum of 250Hz. The SD Card can only be initialized at a Clock Frequency of Master Clock / 64, which means we can't go much higher than the frequency that it bounds. Also, we want to conserve power so we don't want to run the micro processor full out.

3.3 Interface Requirements

Correct operation of the instrumented football helmet requires several external components interacting with the microcontroller. Each of these components consumes a number of I/O pins utilized by the microcontroller. The SD card requires 3.3v to operate correctly, and utilizes 4 I/O pins for data transfer. The accelerometers need 5v to function, and the microprocessor needs 3 ATD per accelerometer [5,10]. The wireless adaptor requires 3.3v to function and is going to be toggled manually using 4 I/O pins [6]. Since the design doesn't include any on helmet charge indication, the micro no longer will require 3 I/O pins to interface with LEDs in the helmet.

3.4 On-Chip Peripheral Requirements

The microcontroller utilizes specific on-chip peripheral to complete the required data gathering and analysis. There are 9 channels of 10 bit ATD. Communicating with the wireless adaptor is using its own peripheral. The nature of the chip requires the microcontroller to utilize RS-232 serial protocol to relay information [6]. The RS-232 is integrated onto the microprocessor instead of outsourced to an external chip. The SD card reader takes advantage of the SPI port for the data storage and is the only peripheral utilizing the SPI.

3.5 Off-Chip Peripheral Requirements

In general sense, it's almost impossible for one component to complete all the necessary steps expected from an embedded design. An expectation of such would require the microprocessor to be much too powerful which would mean larger batteries and embedded systems. To alleviate the requirements of the microprocessor, many dedicated tasks are outsourced to off chip components. In the instrumented football helmet, the battery management is one such task that is handled this way. The MCP73838 chip takes care of charging the battery and the DS2740U takes care of relaying the current remaining power in the battery to the microprocessor [8]. Another peripheral that is heavily relied upon is the wireless adaptor. The Matchport adaptor takes serial data from the microprocessor and converts it to the usable 802.11 b/g wireless signal, as well maintains its own internal web server [6]. The last component that we thought may be needed, depending on if the SD card needs to be readable by a computer or not, is a FAT32 conversion chip. This chip will accept serial data and then convert it to the format computers will use to read from the SD card [12]. However, we decided to go with our own system and keep the SD Card internal and only for helmet data storage. These components are all necessary to keep the amount of computations done on the microprocessor to a minimum.

3.6 Power Constraints

An important aspect of the instrumented football helmet is that it must be battery powered. This requirement provides a limitation to the power consumption of each component. If any component heats up enough for the player to notice, it will adversely affect performance, which would be a defect in the design. Currently, the power consumption of the helmet is the following:

Matchport Wireless Controller:	70/260mA [2]
MMA1212 and MMA2301 Accelerometers:	10/60mA [5]
HR1940CT SD Memory Controller:	20mA [6]
PIC18F45K20 Microprocessor:	13.5mA [10]

Running all components at maximum power we will draw 480.5mA. If the helmet was to be operational for 5 hours, the battery would need to have a capacity of at least 1587.5 mAh. One such battery is the 5169UBBP01, which has a capacity of 1800 mAh [1]. As an alternative, the design could utilize two smaller capacity batteries, the LIR14500, to attain the same battery capacity. At this point, the helmet uses just one battery because the combined weight of the two smaller batteries exceeds that of the single battery by 7g.

3.7 Packaging Constraints

Packaging an embedded system inside of a football helmet can be problematic. Each component needs to be anchored down to the helmet and be able to resist impacts of significant force. This has been accomplished by placing the PCBs into plastic coverings and then Velcroing these to the inner walls of the helmet. The size of the system needs to be able to fit inside of the helmet or on the helmet in a small compartment that still provides maximum safety to the player. The total weight of the design must be small to not affect or endanger the player. The individual component weights are:

Matchport Wireless Controller:	14g [2]
MMA1212 and MMA2301 Accelerometers:	.35g [5]
HR1940CT SD Memory Controller:	.1g [6]

PIC18F45K20 Microprocessor:	2.1g [10]
5169UBBP01 Battery:	33g [1]

The total weight of the design based solely off of the components will be 49.55g. This is approximately the weight of 10 nickels [14].

3.8 Cost Constraints

Thorough research indicates that there are few instrumented helmets on the market while there are many in use. Most designs are used for research purposes through the government and private companies. The one company, Riddell, that has introduced its instrumented helmet to the general market charges an outstanding 1000 dollars per helmet, and then charges 300 dollars for the receiver to communicate with the helmet [13]. The company has not posted any relevant technical data on their product. Battery life, receiver distance, and recharge time are pieces of information that have not been publicly advertised. The instrumented football helmet outlined in this report will only cost a fraction of that amount at 200 dollars. The accelerometers will be 38 dollars each, the wireless adaptor will be 75 dollars, the microcontroller will be 4 dollars, the SD card reader will be 1 dollar, and the SD card will be 3 dollars. The designing of the helmet might have been more expensive due to part repurchasing after mishaps, fortunately after completing the project, we haven't accumulated any extra costs.

3.9 Component Selection Rationale

Selecting components to satisfy a designs final completion goals can be a strenuous task. Breaking down the overall design into small manageable tasks, and then locating the perfect component to achieve any one of the tasks in the most cost efficient and power efficient manner, but it takes time and thoughtful consideration for each task. Knowing that the helmet would require at least 1 and up to 4 accelerometers to function and multiple communication lines to off chip memory and the wireless adaptor two microprocessors seemed to be the best fit. The PIC18F45K20 and the ATmega8 both have the required I/O and ATD [3,11]. The PIC18F45K20 has 14 10-bit ATD and 36 I/O along with an SPI and UA/SRT communication line [3,10]. The ATmega8 has 8 12-bit ATD and 23 I/O along with an SPI port [3]. After much consideration, the PIC18F45K20 was chosen because of the extra ATD and I/O at approximately the same price of under 4 dollars. In the event of a mishap there will be extra ports to spare on peripherals. In order for the data to be accurate enough to detect a potential concussion causing impact, the accelerometers must be able to record over 150 g's. Both the ADXL193 and the MMA1212, MMA2301 combo handle +/- 250 g's with +/- 3 g accuracy [2,5]. The only difference between these accelerometers is the axis count. The ADXL193 is a single axis and the MMA1212, MMA2301 combo is a simulated 3-axis accelerometer [2,5]. The MMA1212, MMA2301 combo will be the best choice because it will eliminate worrying about mounting up to 9 accelerometers to have redundancy or rotation sensitivity on the x axis, y axis, and z axis. The wireless adaptor was an easy choice. There are only two widely used adaptors and both have an embedded web server. The Matchport adaptor has a variety of antennas to choose from and costs 75 dollars, while the WiPort has only one hard wired antenna and costs 120 dollars [6,7]. On the basis of cost and choice of antenna, it was decided that the Matchport would be best. When it came to choosing a battery management chip, there were not many competitors to be found, so to aid in the search other wireless products were looked into and each one used the same company's chip. It was assumed that if other companies all chose the same chip, it had to be a good choice; which led to the selection of the MCP73838 power management chip to facilitate the recharging and charge indication of the battery [8]. In the event that we changed to use two batteries, the MCP73838 will not be able to work so the MAX1758 battery management chip will be used. The battery charge indicator to be used is the DS2740U. However, just like the battery management chip, if the two battery route is taken, the alternative chip would be the PS800X. The battery chosen, 5169UBBP01, is a lithium ion battery rated at 1800 mAh [1]. If the design will use two batteries in parallel, then the LIR14500 will be used; however, using two batteries would almost double the weight of a single battery.

3.10 Summary

Inspired by the need for additional protection in the rough sport of football, the instrumented helmet first came into concept. The purpose of the helmet is to monitor and record forces felt by the players head in an impact, and to do that an embedded design would have to be implemented. Teams from both Purdue University and Villanova University assembled to accomplish this task in the cheapest most efficient way in terms of weight, size, longevity of use, ease of use, and ability to be used from anywhere on a football field. The components that were analyzed and found fit for use in the design were:

PIC18F45K20 microprocessor

Matchport wireless adaptor

MCP73838 battery manager

DS2740U battery charge indicator

MMA1212, MMA2301 combo accelerometer

LIR18650 battery

HR1940CT SD card holder with a Sandisk SD card

These components are the best fit for the helmet that could be found after countless hours of research done by both teams. The only competition that exists out in the business world charges over 4 times as much as the design team can produce a single unit for. The helmet is usable for at least 5 hours on the field and can be adapted to any sport that would wear a helmet.

4.1 Patent Liability Analysis

The Instrumented Football Helmet (IFH) is a standard regulation football helmet that is equipped with sensors that measure, record, and analyze impacts. Upon examining similar products and existing patents, there are two areas of potential infringement. The first, is in the method of measuring and recording real time impact data. The second is the ability to notify sideline personnel upon dangerous hits. While there are patents that focus on algorithms used to calculate the linear and rotational accelerations, the IFH uses standard physics formulas to solve for linear and rotational acceleration.

4.1 Results of Patent and Product Search

After a thorough search for existing products there is only one worth discussing. The Riddell Revolution IQ HITS™ performs many of the same functions as the IFH. Both devices use accelerometers to gather impact data on a football player's head, record data into a memory device, and notify officials should there be potential injury to a player. Eight patents were found that covered the HITS™. While the majority of them govern various aspects of the physical construction of the helmet, there is one that covers

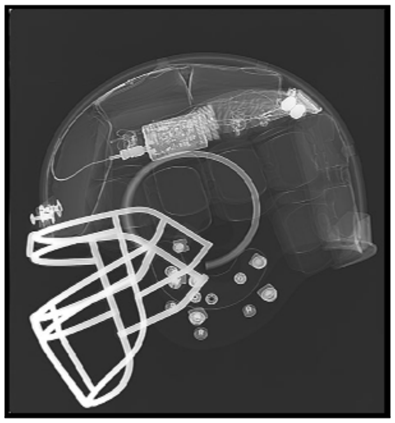


Figure 4.1: Riddell HITS™

the electronic and sensor systems [15]. It is this patent that needs to be considered.

US patent number 6826509, filed in October 10, 2001, outlines the method to collect impact data via accelerometers inside of the HITS™ helmet [18]. A summary of the patents abstract is as follows:

> A system and method for determining the magnitude of linear and rotational acceleration of and direction of impact to a body part using single-axis accelerometers proximate to the outer surface of the body part. The acceleration data sensed is collected and recorded. A hit profile is calculated.

The claims of this patent outline the functionality of the device. They each revolve around two key features. The first is at least 3 accelerometers mounted non-orthogonally into a head worn device. The second is that the device calculates the direction and magnitude of an impact. Since both of these features are of interest, the majority of the claims in this patent are of importance.

The second patent examined was US Patent number 5978972, filed in June 11, 1997, which outlines a device very similar to the Riddell HITS™ [16]:

> A system designed to measure and record in real time data relating to translational and angular acceleration of an individual's head during normal sporting activity. Data is recorded onto a memory card or other mass memory means installed in the helmet, or is transmitted to a nearby receiver for storage on a computer's hard drive. The data also allows detection of the precise motions of the head which precede the occurrence of a severe head injury.

Claim one (along with many of the claims in this patent), refer to a head worn device in which all 3 accelerometers are mounted orthogonally to each other and record data into a memory device. Like the Riddell patent, the only function that this patent covers is the means of gathering data. However, in this patent, the accelerometers are all mounted orthogonally as opposed to the Riddell patent in which they are all mounted non-orthogonally.

The last patent to be examined was US Patent number 5621922, filed December 20, 1995. This patent describes a device which consists of sensors and a signaling device [17]:

> A signaling device is installed in headwear and includes sensing devices for detecting linearly and rotationally directed impacts above a selected magnitude. The sensing devices trigger the signaling device so as to produce a perceivable signal, thereby alerting observers that a potentially injurious impact has occurred.

Claim one in this patent is the only claim of interest. In it, two criteria are laid out. First is that the device must have sensors (of any type) that can detect linear and rotational force. Secondly the device must generate a perceivable signal in response to a linear or rotational force.

4.2 Analysis of Patent Liability

As outlined above, the IFH performs many of the same functions as the Riddell HITS™, however, the only the data collection function of the HIT™ holds any patents. While the IFH mounts its accelerometers non-orthogonally to each other, it does not calculate or determine the magnitude and direction of an impact. It only calculates the magnitude of the acceleration that the head suffers as a result of the impact. Direction and impact location are not calculated. As stated under the claims in Riddell's patent, both the accelerometer mounting method and what is calculated must be met in order for there to be a violation. Therefore, no infringement is made on this patent.

The second patent (#5978972) uses orthogonally mounted accelerometers to measure movement on a head worn device and record the data onto a mass memory device. While the IFH's accelerometers are mounted non-orthogonally to each other, it can be argued that each accelerometer actually contains 3 orthogonally mounted singe-axis accelerometers. If viewed this way, then there is the potential for literal infringement on this patent.

Lastly, patent 5621922 challenges the notification function of the IFH. Fortunately under its claims it clearly states "perceivable signal" as a requirement, there is no literal infringement on any claims. However, under the doctrine of equivalence one could argue the meaning of perceivable. Since the signal given off by the IFH is Wi-Fi, it is not "perceivable" to us, but it is perceivable to a wireless adapter, the Wi-Fi could be viewed as a wireless cable and the resulting page or email could be viewed as the perceivable signal. This means that we have a potential infringement under doctrine of equivalence.

4.3 Action Recommended

Since there is no violation of Riddell's HITS™ patent, there is no action needed on its behalf. However, it would not hurt to mount each accelerometer cluster such that they are not only non-orthogonal with each other, but also non-orthogonal with the surface of the skull. This would however, pose additional challenges in packaging.

In terms of patent 5978972 infringements, the ambiguity of whether the accelerometers are orthogonally mounted to each other can be eliminated easily. By using tri-axis accelerometers instead of three single-axis accelerometers, there is no question that each sensor is non-orthogonal to each other.

Since the IFH infringes on patent 5621922 through the doctrine of equivalence, there would be two possible solutions. The first would be to pay royalty fees until the patent expires in 2015. The second would be to simply wait until 2015 to start production and marketing of the IFH.

4.4 Summary

Since many features of the IFH exist as prior art, there are only a handful of features present that would need to be addressed should this design go to market. Most likely the three single axis accelerometer clusters would need to be replaced by true multi-axis accelerometers to avoid possible infringements on patent 5978972.

5.0 Reliability and Safety Analysis

5.1 Introduction

The Instrumented Football Helmet is an impact measuring and alerting device built into a standard regulation football helmet. It is intended to be primarily used by colleges and professional football teams as a means of alerting athletic trainers on the sidelines in the case of an impact that has the potential of causing a concussion. In addition to this, the helmet will be a valuable tool in the research of head injuries as these are still not well understood. Because of the serious health impacts this helmet will have on players wearing it, the safety and reliability of the components being used in it is very important.

The safety issue most critical to this design is that of components overheating or even bursting into flame in close proximity to a player’s head. The potential loss of the sideline alerting functions of the helmet poses a large safety issue as well. Both of these possibilities may result in player injuries, either directly through harm from burning components or indirectly through non-reporting of potential concussions. Because of this, it is important to analyze the design to determine how the helmet could fail. By doing this, the helmet’s design could be further refined to ensure that when it fails, it does so in a safe and predictable way.

The greatest impact on reliability for all the components in this design is simply the environment in which they will be operating. They will be built into a football helmet and subjected to upwards of 100 g's as well as high temperatures. With the importance of safety in this design and the harsh environment the components of the design will be subjected to, it is important to conduct a thorough reliability analysis.

5.2 Reliability Analysis

Because of the importance of safety to this design, it is important to conduct a thorough reliability analysis. Normally in a thorough reliability analysis, every single component used is analyzed for failure rates and are given mean time to failure values. For this design, only those components most likely to fail will be analyzed. These include the PIC18F45K20 microcontroller, Matchport wireless adapter, the LTC3440 Buck/Boost voltage regulator, the TPS75733 LDO voltage regulator, the LT1303 Boost voltage regulator, and the 5169UBBP01 Lithium Ion battery. These components are either the most complicated or hottest operating components, and so should be analyzed for their reliability.

Unfortunately, the Matchport wireless adapter and the 5169UBBP01 lithium ion battery are not possible to analyze using the standard method. This is because the Matchport wireless adapter is essentially a black box, with its own unknown circuitry and internal components. It is not a single silicon chip and its data sheet does not include any information about internal parts. The battery is an important component of the design from a reliability standpoint given recent history of many commercial products failing due to "exploding" lithium ion batteries. Unfortunately, the military handbook used for calculating the failure rate does not include methods to determine the reliability of batteries. A reliability analysis for this lithium ion battery would likely require tests to determine failure rates, rather than simply calculating a conservative estimate using given formulas.

When conducting a reliability analysis, the failure rates of individual components in the design are calculated. These rates are calculated from the formula: $\lambda_p = (C_1 * \pi_T + C_2 * \pi_E) * \pi_Q * \pi_L$, where λ_p is the number of failures in 10^6 hours, C_1 is the die complexity failure rate, π_T is the temperature coefficient, C_2 is the package failure rate, π_E is the environment factor, π_Q is the quality factor, and π_L is the learning factor. π_E, π_Q, and π_L stay the same for each component in the design while C_1, C_2, and π_T are based on the component itself. First, the π_E is determined to be 4.0 for all components because the device is considered to be in a mobile ground environment. Second, the π_L is determined to be 1.0 for all components, signify the device has been in production for more than two years. Finally, the π_Q is determined to be 10.0 since the components used are all commercial products. All of these parameters are found using the Military Handbook for Probability Prediction of Electronic Equipment [19].

Parameter	**Description**	**Value**	**Comments**
C_1	Die Complexity Failure Rate	0.14	MOS based microprocessor
π_T	Temperature Coefficient	0.64	Digital MOS, T_J = +72 degrees C
C_2	Package failure rate	0.2144	44 pin TQFP
π_E	Environment factor	4.0	Ground Mobile
π_Q	Quality Factor	10.0	Commercial part
π_L	Learning Factor	1.0	≥ 2 years of production
λ_p	Failure rate per 10^6 hours	9.472	
MTTF	Mean Time To Failure	12.05 years	

Table 5.1 – PIC18F45K20 Microcontroller [20] Reliability

Parameter	**Description**	**Value**	**Comments**
C_1	Die Complexity Failure Rate	0.01	4 transistors, Linear MOS device
π_T	Temperature Coefficient	7.0	Linear MOS, T_J = +85 degrees C
C_2	Package failure rate	0.0043	10 pin MSOP packages
π_E	Environment factor	4.0	Ground Mobile
π_Q	Quality Factor	10.0	Commercial part
π_L	Learning Factor	1.0	≥ 2 years of production
λ_p	Failure rate per 10^6 hours	0.872	
MTTF	Mean Time To Failure	130.9 years	

Table 5.2 – LTC3440 Buck/Boost [21] Reliability

Parameter	**Description**	**Value**	**Comments**
C_1	Die Complexity Failure Rate	0.01	Linear MOS, < 100 transistors
π_T	Temperature Coefficient	58	Linear MOS, T_J = +125 degrees C
C_2	Package failure rate	0.002	5 pin TO-220 Package
π_E	Environment factor	4.0	Ground Mobile
π_Q	Quality Factor	10.0	Commercial part
π_L	Learning Factor	1.0	≥ 2 years of production
λ_p	Failure rate per 10^6 hours	5.88	
MTTF	Mean Time To Failure	19.41 years	

Table 5.3 – TPS75733 LDO [22] Reliability

Parameter	Description	Value	Comments
C_1	Die Complexity Failure Rate	0.01	Linear MOS, < 100 transistors
π_T	Temperature Coefficient	2.8	Linear MOS, T_J = +70 degrees C
C_2	Package failure rate	0.003	8 pin S8 package
π_E	Environment factor	4.0	Ground Mobile
π_Q	Quality Factor	10.0	Commercial part
π_L	Learning Factor	1.0	≥ 2 years of production
λ_p	Failure rate per 10^6 hours	0.4	
MTTF	Mean Time To Failure	285.39 years	

Table 5.4 – LT1303 Boost [23] Reliability

After conducting a reliability analysis on these four components, it can be seen that some of the parts have a rather low MTTF. While the MTTF of 12.05 years for the microcontroller is sufficient to cover the expected life of the helmet, the failure rate is much worse. The failure rate is considered to have uniform distribution over the entire 114 years, so producing a large number of these devices would see product returns because of component failure very often. One way of improving component reliability is reducing the temperature coefficients for the parts. The temperature coefficients can be reduced on many of these components by properly heat sinking them. Also, there are two potential improvements for the PICF45K20 microcontroller: the choice of a hermetically sealed package to reduce the package failure rate, and choosing a different, military grade, microcontroller. A way to radically reduce the failure rate on all the components is simply to choose different, military grade, components.

5.3 Failure Mode, Effects, and Criticality Analysis (FMECA)

In order to do a FMECA analysis, the complete design was divided up into the following blocks: Power Supply, Microcontroller Power Supply, Microcontroller, Wireless Power Supply, Wireless, and Accelerometers. The full schematics of each of these blocks can be seen in Appendix A. Also when completing a FMECA analysis, several different criticality levels needed to be defined. First, a failure with high criticality is one in which there is potential for injury to the player wearing the helmet. A failure with this level of criticality needs to have a failure rate less than 10^{-9} occurrences in every 114 years. Second, a failure with medium criticality is one in which the helmet loses the ability to send sideline alerts. This can be anything from complete failure of the entire device to the failure of just the accelerometers. Failures with medium criticality need to have a failure rate of less than 10^{-5} occurrences in every 114 years. Finally, a failure with low criticality is any other sort of failure. These failures are more of an inconvenience than something that could potentially cause an injury. These failures cause partial loss of the functions of the helmet without harming its ability to monitor and report on impacts. Low criticality failures have a failure rate of greater than 10^{-5} occurrences in every 114 years.

When conducting the FMCA analysis, it became apparent that there are very few low criticality failures. Because of the potential for injury if alerts are not sent to the sideline, most of the wireless and accelerometer failures are of medium criticality. Also, because of the way in which voltage regulators were distributed around the design, rather than just kept in the power supply, there are a large number of power supply failures. One potential way to reduce the number of medium criticality failures is to have some other way to alert to potential concussions besides the wireless adapter. A possible addition to the wireless for warnings is some sort of speaker with an audible alert. Also, the chance of an injury causing battery failure could be reduced by using a fuse to protect it. Finally, another possible minor addition to the design could be the inclusion of a power LED that is visible when the player goes to put on the helmet. This LED could simply act as a power indicator or blink to show error codes and stay solidly on to indicate the device is working correctly.

5.4 Summary

From the reliability analysis of the major components of the Instrumented Football Helmet in addition to the FMCA analysis, it can be seen that the reliability of many of the major components needs to be improved. The relatively high failure rates of many of these components will cause numerous problems when the helmet is mass produced, resulting in frequent returns and possible injury. These failure rates could be reduced by introducing heat sinks to many of the hotter components, changing component package types, and choosing different components of military rather than commercial grade. The helmet can be given a better chance to fail safely by adding a fuse to the battery connection, adding an alternative means of concussion alerting besides the wireless adapter, and including a LED to indicate proper operation or error codes. These additions would reduce the chances of the device either directly or indirectly harming a player using it.

6.0 Ethical and Environmental Impact Analysis

6.1 Introduction

Our team is designing an instrumented football helmet with the purpose of monitoring impacts encountered over the course of a football game. To achieve this, we are mounting a system to monitor acceleration at several different points of the football helmet and report when a particularly dangerous impact has occurred. A major component of our design is analyzing the ethical and environmental impact. In terms of ethical considerations, our largest concern is ensuring that our helmet is reliable and will work as expected. This can be accomplished through thorough testing and ensuring the user is well informed about possible dangers of our design. In terms of environmental impact, our largest concerns are ensuring the product's manufacturing and disposal components do not have a large impact.

6.2 Ethical Impact Analysis

One of the biggest ethical considerations in the design of our product is testing. Testing is important because our product is intended to protect its user. Our primary concern in testing is ensuring our product does not change the protection the helmet already provides. Thus it was necessary to ensure the helmet met the standards proposed by the National Operating Committee on Standards for Athletic Equipment and is recertified. Additionally, because our helmet relies heavily on radio interaction, it is necessary for us to get our helmet certified with the FCC to ensure it functions within FCC guidelines.

In addition to ensuring our product functions within specified guidelines, it is necessary to test our helmet's reliability and functioning very thoroughly. Device reliability is particularly important because any inaccurate results could lead to players not getting help when they actually should. As such, we need to do extensive testing to ensure that values we record in terms of rotational and linear accelerations are actually what they should be. In order to achieve this, a piece of hardware will be designed and built to apply measured impacts, both rotational and linear, to the helmet and record how accurately the helmet acquires them. This testing will be performed upon assembly and could also be used to test the helmet in the field to ensure continued functionality. In addition to testing the recording system, we will also do some testing to ensure the system as a whole does not produce excessive or harmful amounts of heat. Because this system is mounted inside a football helmet, we need to ensure it is not any more uncomfortable to wear than a regular football helmet.

Another aspect of testing our design is to ensure the helmet can work in a variety of conditions. This testing is necessary because our design will be used in a large variety of conditions, and we are responsible for ensuring the design will work under varied conditions. First, we need to test and ensure the helmet is moisture proof because our design will likely be used in conditions of rain and snow. Additionally, moisture could come from the sweat of a player. Also, we need to ensure the design will work over a wide range of temperatures. Football games have been played in temperatures below 0 and above 100 degrees, so our device should be able to handle this range of temperatures without breaking or recording incorrect results.

In addition to extensive testing, we need to provide the user with sufficient safety information regarding several aspects of our device. First, on the helmet we will put a warning sticker on the system that reminds the user to check for system responsiveness when starting up the helmet. In order to facilitate this process in future versions of the design, we will incorporate some sort of visual or auditory warning method to inform the user if the product is unresponsive or otherwise malfunctioning. We will also provide warning to the user in the product manual about several things. We need to warn them about the accuracy of the results. Other warnings that will be included are cautions about packaging stability and heat produced by our product. All of these warning will include information about the results that were obtained from our testing. Additionally, we will also include warnings about the battery. These warnings will include a warning not to puncture or modify the battery in any way. It should also indicate that if the battery is damaged in any way, it should be replaced immediately.

Finally there are several modifications we need add into a second iteration of our design. First, as previously mentioned, we need to add either an auditory or visual notification system to alert users if the product is not functioning fully. Secondly, we need to add an alert system to the WiFi board that would function off a watchdog timer from the microcontroller. This system would function by having the microcontroller check the status of the accelerometers at intervals and ensure they are working. If the accelerometers are not working or the microcontroller fails, the system will trigger and the Matchport will send a signal to the sidelines indicating a failure in the device has occurred. Although this would not cover failures in the Matchport, it would at least cover some problems our product may encounter. Finally, we also need to add in additional shielding for our battery unit. This extra shielding would help to prevent deformation or puncturing of the battery. Additionally, this shielding would help to protect the user if the battery were to overheat or catch on fire.

6.3 Environmental Impact Analysis

Over the life cycle of our design, there are two main times that our device may have an impact on the environment. During manufacturing, our device will impact the environment mostly through the production of its parts. The PCBs that we use contain several harmful chemicals, including lead. This means that all parts of this process will need to be specially monitored. The manufacturing of our battery may also have an environmental impact through use of unusual metals.

During the use of our product there is relatively little environmental impact. When creating our design we had a heavy focus on ensuring maximum battery life, which coincides with minimizing power consumption. As such, our device has very little demand for energy.

The other period during which our design might have an environmental impact is during the disposal phase of its lifecycle. In order to minimize our environmental impact in the disposal of our product several steps need to be taken. First of all, according to [25], many of the metals used in the battery can be recycled. For the rest of our system, it can be disposed of in an environmentally safe fashion through sources see in [26]. By encouraging our users to utilize recycling and environmentally friendly disposal companies, we can minimize the impact our design has on the environment during its disposal phase.

6.4 Summary

Because our design is meant to enhance player safety, there are many ethical considerations that were into account. However, by ensuring our device goes through thorough testing and certification and the users are well informed about possible hazards of using our product, we can ensure the ethical issues presented here are minimized. In terms of environmental impact, we also need to be aware of what goes into our product and encourage users to dispose of it in an environmentally friendly way.

7.0 Packaging Design Considerations

7.1 Introduction

When compared to most projects, the Instrumented Football Helmet is unique on the packaging front. Rather than designing a package around an embedded system, the IC was designed to fit inside an existing product. The package itself is a standard regulation football helmet.

The restrictions imposed by the helmet on the IC are numerous. The designed system must be lightweight and small as too much mass would adversely affect the performance of the athlete and anything too large wouldn’t fit well inside the helmet. Secondly the helmet must remain relatively cool. If it gets too hot, it will get uncomfortable for the athlete and again, adversely affect their performance. Thirdly all hardware must be hidden from view. All of the chips, PCBs, wires, and antennas must remain within the helmet to avoid impact or exposure to weather. Finally, all circuitry must be firmly in place and be able to withstand major shock without breaking or coming loose. These restrictions can be handled by an intelligent selection of components, an even distribution of parts inside the helmet, and proper wiring.

7.2 Commercial Product Packaging

There are two categories of products on the market that are similar to ours. The first category accomplishes the same goal and has the same target market segment. The other category uses the same technology but accomplishes something completely different and targets a separate consumer base. Since there is only one existing product on the market that is similar to ours, a product from each category will be analyzed.

7.2.1 Product #1 (Riddell Revolution IQ HITS™)

There is only one product that fell into the first category and that is the Riddell Revolution IQ HITS™. It is a very well designed product and currently sells for $1000[26]. It consists of 6 accelerometers, a Wi-Fi transmitter and rechargeable batteries. It is packaged inside of a standard regulation football helmet. Similar to our product, the HITS™ helmet monitors impact data and reports potentially dangerous impacts to sideline personnel. It also allows the player to download data after a practice or a game onto a PC where it can be analyzed, graphed, saved, or studied [26].

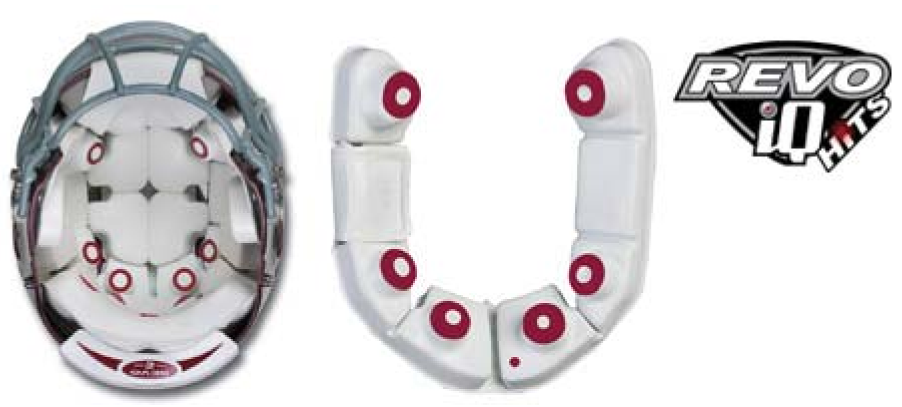


Figure 7-1. Riddell Revolution IQ HITS™

For this project to be successful, it must look and feel like a normal regulation sized football helmet. The Riddell Revolution IQ HITS™ accomplishes this goal. The antenna is placed along the roof of the helmet which gives it optimal reception and keeps it away from other parts that might cause electrical interference and transmission noise. Another packaging feature of the HITS™ helmet is that it uses multiple small PCBs instead of one large PCB. This allows an easier integration into the helmet. The distribution of parts throughout the helmet will help keep the weight even.

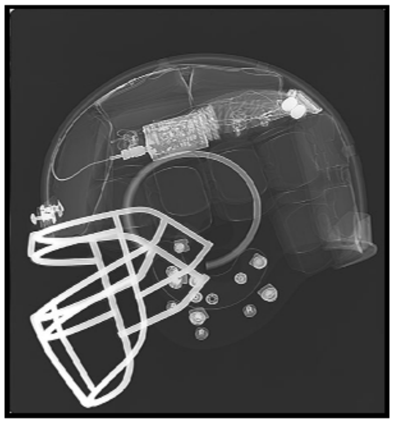


Figure 7-2. X-ray of Riddell helmet

7.2.2 Product #2 (BMW M5 Sports Sedan)

One example of a product from the second category would be a high performance sports car, such as the BMW M5. This car uses accelerometers in traction control systems, airbag deployment, and vibration sensing [27]. While the application is different than the Instrumented Football Helmet, the technology behind it is very similar.

One positive aspect in the packaging of the M5 is its seamless integration of this technology. The typical driver doesn't realize when the traction control is engaging as the accelerometers and embedded systems all work in the background. This idea must be kept in mind when building the Instrumented Football Helmet. The design of the PCB should be integrated into the helmet in such a way that the player is unaware of data being collected, analyzed, and sent to the sideline.

Figure 7-3: BMW M5

Caution must be taken that the technology does not compromise the performance of the helmet. A driver in the seat of an M5 can disable the traction control in order to get better acceleration and breaking power. However, the helmet must remain as safe and protective with the circuitry inside it as it is without it.

7.3 Project Packaging Specifications

The design is divided into 5 PCBs. The first PCB contains the MCU, the microSD and the power circuitry. The second contains the wireless adapter. The last three PCBs are accelerometer clusters. Each cluster consists of 3 single-axis accelerometers. This allows each PCB to fit comfortably inside the helmet and minimizes the chances of damage upon impact. In order to disperse the weight evenly over the helmet, the PCBs will be distributed throughout the helmet. The heaviest component is the battery, weighing 43 grams [28]. This is not restrictive and thus could be mounted anywhere. Ideally the microcontroller should be near the center of the helmet to shorten the data lines between it and the accelerometers. Ideally all circuitry should be placed inside padding to protect it from moisture and other adverse conditions. However, the helmet used in the PHI-Master was donated by the athletic department and does not have padding conducive to this type of installation. Instead, the circuitry will need to be installed in the gaps between the padding. In order to mount the PCBs securely in place and still allow easy removal, the use of industrial Velcro is used. To increase the moisture resistance of the circuitry vinyl envelopes are used.

In addition to PCB placement, twisted pair Ethernet wiring is used to reduce noise on the accelerometer data lines. This should allow appropriate distribution of the components around the helmet without adding to the cost or size of the design. Ethernet wiring is small enough to incorporate easily around the helmet padding.

Factoring in the cost of all the major components, the weight of the project is negligible and doesn’t change the weight of a standard regulation helmet, which weighs around 6-8lbs [29]. Initial estimates brought the cost of the project should come out to around $344.

7.4 PCB Footprint Layout

The PCB Footprints measure approximately 3 inches by 2.5 inches for the main board PCB and 2 inches by 2.3 inches for the Wi-Fi PCB. Each accelerometer is a cm^2 in area [30], making each cluster PCB just over 2cm by 6 cm. The accelerometer PCBs will be connected to the microcontroller PCB via Ethernet twisted pair cabling. This will help the accuracy of the helmet by reducing noise on the analog data lines. Please refer to appendix C for scaled drawings of the PCBs.

7.5 Summary

After taking into account the restrictions a football helmet imposes on packaging and analyzing existing products, it makes sense to break up the PCB into parts. This not only makes the design easier to install into the helmet, but it ensures that the weight is distributed evenly. To compensate for the increased space between the components, insulated Ethernet cabling is used to help reduce noise and ensure accurate readings.

8.0 Schematic Design Considerations

8.1 Introduction

The Instrumented Football Helmet is designed to sense impacts. The helmet will use three sets of accelerometers to accurately measure both impact strength and rotational inertia from impacts for the purpose of detecting concussion-causing hits. It will also use a micro SD card for extra data storage. The Instrumented Football Helmet will communicate to the sidelines via wireless Ethernet through a Matchport b/g. In addition to communicating through this wireless connection to sideline pagers to warn of a possible concussion, the helmet will include an embedded web server for accessing data and changing internal thresholds on the fly. Through this wireless Ethernet connection, the helmet would also be able to transfer all stored data to another network device. The Instrumented Football Helmet is intended to be used both as a safety device, warning that medical attention may be necessary on a hard hit to the head, and as an aid to research into head injuries. In order to meet these operating requirements, several different design choices must be considered.

8.2 Theory of Operation

8.2.1 Power Supply

The Instrumented Football Helmet is powered by a single cell lithium ion battery. This battery will output 3.7 volts and will have enough charge capacity to run the helmet for at least 4 hours. The battery will be recharged by a Microchip MCP73838 and a battery fuel gauge is implemented by a Maxim DS2782 chip. The MCP73838 has a thermal protection feature using a thermistor and supports both USB and standard DC power input [37]. To regulate the battery output, two chips are used.

For two regulated 3.3 volt power lines, the LTC3440 Synchronous Bock-Boost DC/DC Converter will be used. These 3.3 volt power lines will power most of the devices in the helmet, and so have a high current requirement. One LTC3440 is used to supply this voltage to the Matchport while the other is used to supply it to the PIC and the SD card. The LTC3440 is rated to supply up to 600 mA of current[35], which will be enough to support even the worst case current draw of the Matchport (360 mA). The second chip to regulate the battery is a Linear Technology LTC1751 Regulated Charge Pump DC/DC Converter. This chip is used to create a regulated 5 volt power line with a rated current of 100 mA [36]. The Matchport[4], PIC18 microcontroller[31], and micro SD card are on the two 3.3 volt power lines while only the nine single-axis accelerometers are powered by this 5 volt line so no more than 50-60 mA is needed on this line [32],[33].

Part	Current Requirement	Voltage	Number Used
Matchport	360 mA	3.3V	1
PIC18F45K20	15 mA	3.3V	1
microSD	200 mA (read/write only)	3.3V	1
MMA1212 Accelerometers	6 mA	5V	3
MMA2301 Accelerometers	6 mA	5V	6

Table 8.1 – Circuit Current Requirements

8.2.2 Microcontroller

The microcontroller that is used in the Instrumented Football Helmet is the PIC18F45K20. It is powered at 3.3V and runs at a core operating frequency of 16 MHz with an instruction frequency of 4 MHz [31]. This operating frequency allows the microcontroller to perform all necessary calculations on accelerometer data. While power consumption is important for this project, the microcontroller's power consumption is not even close to that of the more power-hungry Matchport, so reducing the microcontroller's operating frequency would have very little noticeable effect on battery life. In the helmet, the microcontroller is responsible for interfacing with the SD card, accelerometers, and Matchport. It reads and writes data from any installed SD card, captures measurement data from accelerometers, and interfaces with the Matchport for wireless communications.

8.2.3 Accelerometers

Two different accelerometers will be used in the Instrumented Football Helmet. One of these accelerometers is an x-axis accelerometer while the other is a z-axis accelerometer. By using two x-axis accelerometers and a single z-axis accelerometer, a single tri-axis accelerometer can be roughly approximated. The helmet will be using three of these tri-axis accelerometers in order to measure both standard and rotational acceleration. Each single axis accelerometer will be powered with 5 volts and returns a single signal of 0.5-4.5 volts, specifying the current acceleration from -200 to +200 g's [33],[34]. Each accelerometer also has a built in self-test which will be used in a debug mode to determine if the accelerometer is still working.

When powering on an accelerometer, it first puts out unreliable data and draws a large amount of current. Each accelerometer can be reset to properly work by first cycling the Self Test line high and then dropping it low [33],[34]. The Status line is or'd together with the Self Test line and should follow it to zero if the accelerometer is operating correctly.

8.2.4 Wireless Matchport

The Matchport will provide a wireless g Ethernet connection. This will enable in-game sideline alerting through its built in e-mail capability. The Matchport also contains an embedded web server allowing users connected wirelessly to change impact thresholds on the fly. Data stored in the SD card during the game will also be able to be downloaded to another device over the network. The Matchport needs to operate at 3.135-3.45 volts and has a maximum current draw of 360 mA [34]. It will be connected to the microcontroller via a RS-232C interface at 3.3V.

8.2.5 microSD Card

A single microSD card will be used to store accumulated measurements above a certain threshold. This card will be treated as expandable storage for the impact data recorded during the course of a game or practice. This card will be connected to the microcontroller through the processor's built-in SPI module and is powered at 3.3 volts.

8.3 Hardware Design Narrative

The PIC18F45K20 microcontroller's most important (and computationally intensive) task is to calculate the total acceleration at a given time based on all the accelerometer input data. This microcontroller also is tasked with comparing this data to thresholds and either throwing the data out, writing it to an SD card, or both writing it to the SD card and sending an alert. Its final task is to interface with the Matchport to send and receive data wirelessly.

8.3.1 SPI

The microcontroller's SPI peripheral is used exclusively to interface with the micro SD card. For connecting to the micro SD, this will use the RD0, SCK, SDI, and SDO pins. During the initialization phase of the SD card, the SCK must run at less than 400 KHz but can run up to 24 MHz after that.

8.3.2 Analog to Digital Converters

Eleven analog to digital converters are be used in this design. Nine of these are be used for data from the nine accelerometers mounted within the helmet. The final two analog to digital converters are be used as inputs for reference voltages. In these pins, 0.24 and 3.754 volts will be input as the low and high reference voltages for data retrieved from the accelerometers. These reference voltages is be generated by a simple voltage divider circuit off of the 5 volt power line. A sampling frequency of 400 Hz is be used for the analog to digital converters.

8.3.3 Serial (USART)

The microcontroller's serial interface is be used to connect to the Matchport using RS-232C. For this only two pins are needed: the Tx and Rx pins. A baud rate of 9600 is be used.

8.3.4 Timer Module

The timer module will be used to generate interrupts for sampling and other interfacing needs. An internal clock will also be kept in order to time stamp saved data for future reference.

8.4 Summary

Using these components and microcontroller peripherals, the Instrumented Football Helmet will be able to act as a warning against potential concussions and as a valuable tool for head injury research. The key piece of the entire design is the Matchport wireless adapter which will allow the helmet to send its alerts and to transfer its gathered data to another device. The power supply of the helmet was carefully considered to allow for at least four hours of continuous worst case operation, covering even overtime games and extended practices.

9.0 PCB Layout Design Considerations

In order to ensure that this system works, we have designed it to be able to minimize effect of noise on several levels. We focused on the design of the PCB for the micro in particular and the design of the power system on all PCBs. However, first we will discuss design considerations for the system as a whole.

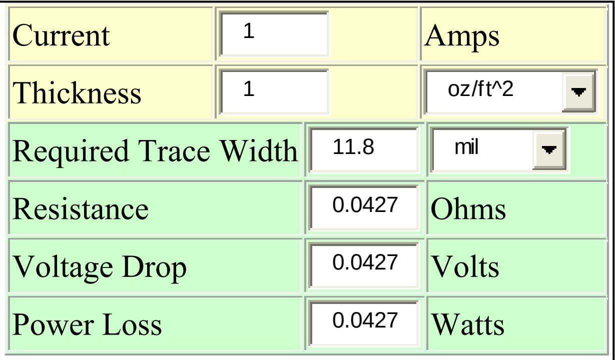
9.1 PCB Layout Design Considerations - Overall

The first aspect of our overall PCB layout that impacted many of our other layout decisions is our use of multiple PCBs. The choice to use many PCBs is a product of our packaging constraints, which have been discussed at an earlier time. In terms of breaking the PCB down, we have one PCB for power, one for the microcontroller and SD Card, one for the Matchport, and three for accelerometer sets. An initial break down of these components and proportionally how much space they will take up can be seen in Appendix D.

In order to get data around our boards, we are using an ethernet twisted pair cable. This cable was chosen for several reasons. The first reason it was chosen was that it is intended to carry data over a long distance without loss. This is very important to us especially for our accelerometer signals where an error of a couple millivolts can cause our microcontroller to be off by several g's. Another reason it was chosen is it has sufficient capacity for carrying our data. Finally, Ethernet cable is widely available and can be acquired for a very low cost.

Our PCBs are designed primarily with a focus on reducing noise on the lines by paying consideration to the recommendations in [1]. Particularly, on our 2 boards that have both analog and digital components, the Matchport and microcontroller boards, the analog and digital components are placed apart from each other. This acts to reduce the interference between these two types of components. Power components were kept off the accelerometer PCBs in particular, in order to reduce the impact on our sensitive accelerometer data.

Routing was also a major consideration in how our traces are laid out for the most critical and sensitive signals, the analog data from the accelerometers. By keeping the traces as short as possible, the inductance and impedance noise in the trace lines is kept to a minimum. Additionally, whenever these data lines cross over power or analog lines, they cross at a 90-degree angle, when possible, to keep interference between traces to a minimum. Finally, routes for the accelerometer data are also kept as close to the same length as possible to ensure that the data arriving at the microcontroller has approximately the same drop in voltage on each of the data lines.



Current	1	Amps
Thickness	1	oz/ft^2
Required Trace Width	11.8	mil
Resistance	0.0427	Ohms
Voltage Drop	0.0427	Volts
Power Loss	0.0427	Watts

Table 9-1. Feedback from [2] regarding trace width

In terms of trace sizing, we decided to use a 12 mil trace width. According to [41] (as seen in table 2.1), 12 mils is sufficient to handle the most amperage that we will see anywhere on our circuit while still being relatively thin and routable. The only exception to this width is the power lines. As suggested by [40], we are using a larger trace of 50 mils. By an increased width trace, the impedance in the power system is reduced, which in turn reduces the noise in the rest of the system.

Several other precautions are taken to reduce the amount of EMI that our PCB will emit and receive. One of these precautions was to keep the amount of 90 degree turns in our tracing on the board to an absolute minimum. This will reduce the amount of transmission reflections. Also the Matchport device will be set to run at as low a clock rate as possible. This reduces the EMI produced by the Matchport device itself.

9.2 PCB Layout Design Considerations - Microcontroller

When designing the PCB for the microcontroller, our main consideration was the minimization of noise, both caused and received by the microcontroller. The first aspect to achieving this was placing bypass capacitors as close to the inputs of the microcontroller as possible. By limiting the length on the traces to the bypass capacitors, the impedance and inductance between the capacitors and the microcontroller was kept to a minimum. Additionally, the capacitors are on the same side of the PCB as the micro. This means that the traces from the capacitors to the microcontroller don't to go through vias, which would introduce additional impedance and inductance. By ensuring that the microcontroller is well supplied with amperage, the microcontroller will introduce less noise into the other systems of the circuit.

Noise reduction was also implemented at ports of the microcontroller. One-way noise is lessened at the analog inputs for the accelerometer data, is by placing filter capacitors on the accelerometer data lines, as recommended by the accelerometer datasheets. These capacitors act to filter our minor fluctuations in voltage introduced by noise. Also, all ports on the microcontroller that are not in use will be configured to be outputs so that changes in voltage on these pins will not cause noise in the microprocessor. Noise is also reduced on our microcontroller PCB by running the microcontroller at less than the maximum clock rate. Our microcontroller, the PIC18F45K20, can run up to 64 MHz, however running at this rate will introduce much noise into the system. Thus microcontroller will run at a much lower rate, probably somewhere around 16 MHz, which will still be enough to meet our computational needs.

9.3 PCB Layout Design Considerations - Power Supply

In order to ensure that noise is kept to a minimum, bypass capacitors were placed at the power inputs of all IC components, including the Matchport device, the SD reader, and all of the accelerometers. These capacitors are placed as close the ICs as possible to reduce the noise introduced by the switching of transistors within the ICs. Additionally, bulk capacitors are located at the edge of all of the PCB's to reduce the impact of the inductance of the cables between boards.

Power and ground trace routing is also very important in handling noise in our system. In order to keep noise to a minimum, ground and power routes are as short as possible. By keeping these paths short, the impedance accumulated through routing is kept to a minimum. Additionally, on the two PCBs that contain a mix of power and digital parts, the microcontroller and Matchport PCBs, the ground is routed in a single point configuration as suggested by [40].

9.4 Summary

The system for our instrumented football helmet is broken up into several different PCBs. The system as a whole is designed to keep noise emissions to a minimum as well as minimize the impact noise within the system. The power system, the microprocessor interface, and the system as a whole are designed to withstand a reasonable level of noise. Thus the instrumented football helmet should be able to withstand the tough conditions on the football field.

10.0 Software Design Considerations

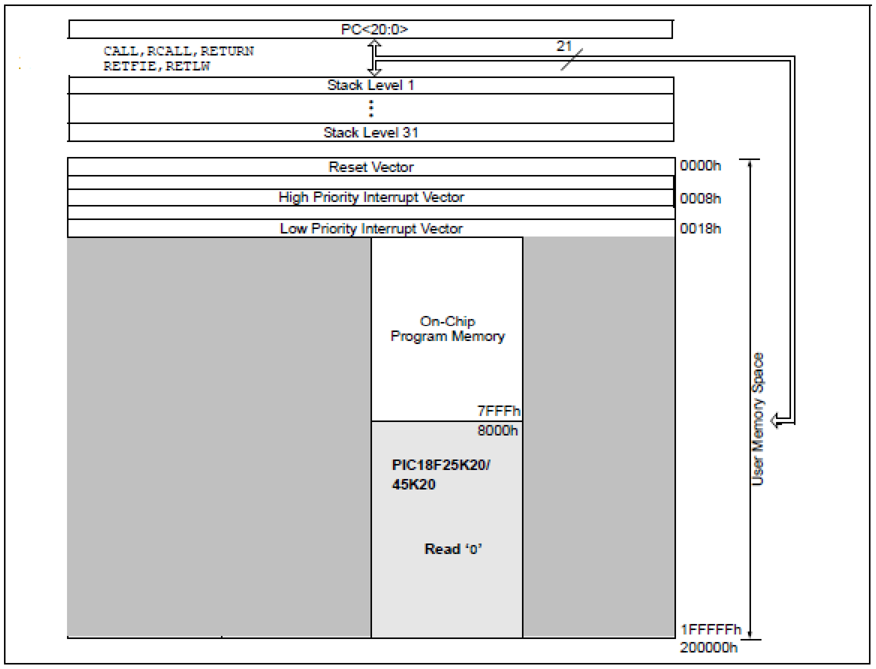
10.1 Introduction

The integration of technology with an established design is never easy. The consumer expects the product to have no drawbacks and still function, look, and feel just as it did before the integration. The design project in consideration is the Instrumented Football Helmet, the integration being attempted is adding the ability to detect and record the magnitude of impacts the player receives during a game. The software running on this design has to be fast and accurate. False alarms or missed impacts would negate any benefit added by the design. The science around how the brain responds to trauma is still evolving, especially in sports medicine, so the design must be able to be altered as more data is gathered and new conclusions about what is harmful are published.

10.2 Software Design Considerations

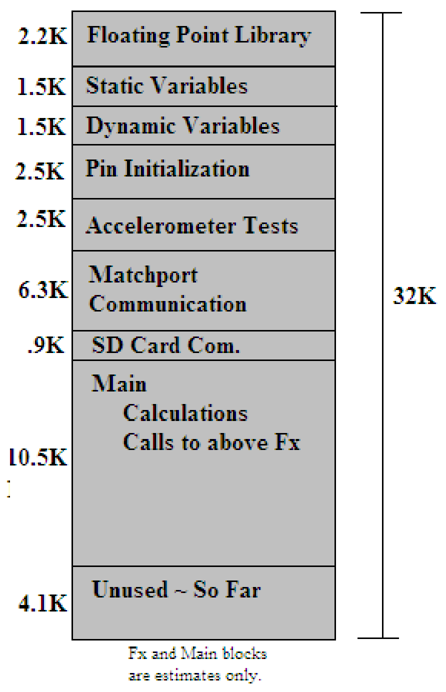
The design considerations are pretty narrow in respect that the software needs to be fast and reliable yet maintains the ability to be altered real time to adjust for changing situations. The integrated hardware has the functionality; it's just a matter of correctly implementing that functionality into a working design. The microcontroller can be clocked up to 64 MHz; however, it is being clocked at 16 MHz to conserve energy. There are three types of memory: program memory, RAM, and EEPROM [38]. The data and program memories are also located on different busses which allow for concurrent access of both memory spaces. Figure 2.1 shows how the stack is not allocated inside of program memory, as well as identifies the allocations of the size of both the flash and ram on

FIGURE 2-1: PROGRAM MEMORY MAP AND STACK FOR PIC18 DEVICE



the PIC18. Because the stack is separate from the program memory, it must be accessed using 4 registers: STKPTR, TOSU, TOSH, and TOSL. The pointer is stored in STKPTR and the bytes stored on the stack are held in the other 3 registers. Flash has the block from 0018h to 7FFFh and ram begins at 8000h. This is 32K of flash memory which allows for more than 16,000 single-word instructions [38]. The memory map for our program is relatively simple. The library for floating point with the basic mathematical functions and trig functions is only 2.2K [39]. We are using the MCC18 C compiler and its basic library's as well. These include header files for the ADC, USART, SPI, and the micro controller itself. All together they don't exceed 3K. The static and dynamic variables are not taking up very much memory; however, it was decided to leave cushion room so that if other blocks were larger than expected there would be extra space left for them. The distinction between static and dynamic in this case is that the dynamic variables are set by the data stored on the Matchport web server. The pins are discussed below. The pseudo code for the blocks Accelerometer Tests, Matchport Communication, SD Card Communication, and Main could be looked at in a ratio format which allows for code size projection based off of pre-microcontroller compilation guesses. The orientation was chosen like this to emulate the normal coding scheme of C. Also, by keeping the communication functions close to main and then the dynamic variables close to the communication, some time may be saved instead of jumping around everywhere wile executing. Once we finished the coding we found that our conservative estimates paid off. We still have available 16K of program memory and 800 bytes of data memory.

Figure 2-2 Flash Allocation



The layout of the pins is exactly as proposed in the previous sections. The nine ATD pins are associated with three registers ADCON0..2, these each have a respective bit to be set to activate the ATD and the output gets stored to ADRESH and ADRESL [38]. The SD Card pins were even easier to set; the data in and out are going to be taken care of by the SPI which runs off of seven registers:

• SSPSTA	– STATUS register
• SSPCON1	– First Control register
• SSPCON2	– Second Control register
• SSPBUF	– Transmit/Receive buffer
• SSPSR	– Shift register
• SSPADD	– Address register
• SSPMSK	– Address Mask register

The battery monitor takes advantage of the digital I/O pin RD7, its value is continually updated into the uppermost byte of PORTD for reading [1]. PORTD is controlled by the TRISD register which enables and controls the direction of PORTD. Lastly, the Matchport will use the UART for transmitting and receiving. The transmitting is governed by the TXREG and the receiving is governed by the RCREG. The baud rate of 9600 is set into BAUDCON and the status of each com register is stored into TXSTA and RCSTA respectively [38]. The configurable pins that enable the emailing and paging functions of the Matchport are stored into the register PORTD. Because of the ease of talking to the web server, it was unnecessary to implement this functionality. Like the battery monitor, the last four bits act as the storage for the data and the TRISD register configures the direction of the pins [38].

Figure 10.3 Pin Layout

Pin	Name	Use
8	AN12	x1 ATD
9	AN10	y1 ATD
10	AN8	z1 ATD
11	AN9	x2 ATD
14	AN11	y2 ATD
19	AN0	z2 ATD
20	AN1	x3 ATD
24	AN4	y3 ATD
25	AN5	z3 ATD
43	SDO	Data In
42	SDI	Data Out
36	RC2	CS
37	SCK	CLK
5	RD7	Bat Stat
1	rx	tx
44	tx	rx
38	RD0	CP1
39	RD1	CP2
40	RD2	CP3
41	RD3	CP4

The code will function as follows:

- ATD Conversions – polling loop in RTI that sets flags
- Battery Status – polling loop in RTI that sets flag
- Accelerometer Calculations – state machine approach
- SD Card and Matchport Communication – function calls with respective requests

The flow chart that illustrates these functions is in Appendix A. At this point, the only provision taken for debugging is to check the accelerometer's status pins. This turns out to be necessary to get correct operation of the accelerometers. At this time, all accelerometers are queried in sequential order and their responses are logically added together to determine the working status.

10.3 Software Design Narrative

The code for the instrumented football helmet is completed for all of the modules. The RTIs that are operating will do two different things. The first RTI, running at 800+ Hz, makes sure all the ATD conversions are complete and sets flags for the main loop to read and reset. The second RTI runs at 1Hz and counts to 300 before setting a flag for main to look at the battery status and update it in the web server. Both of the RTIs are coded and found to be working as expected. The block that tests the accelerometers does so by sending all the accelerometers a status check low, and then adds all of their outputs. If the logic is high they work fine, if not then the Matchport is sent a request by main to update the web server. This block complete and working as expected. The main block checks flags and based upon flag values, runs certain functions. The first block that main calls is Accelerometer Calculations. These calculations compile the individual data from each accelerometer into a usable resultant. The calculations are as follows:

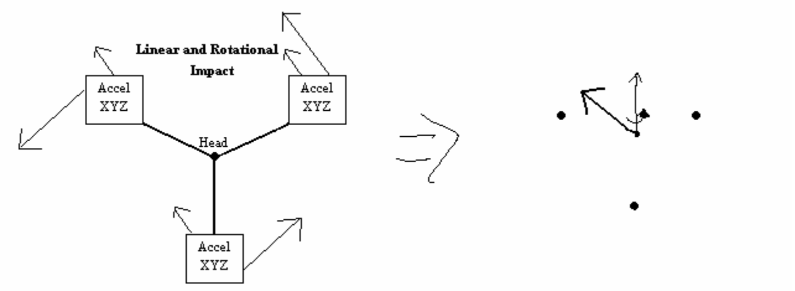


Figure 10.4 Vector Manipulation

$R_X = A_{1X} + A_{2X} + A_{3X}$

$R_Y = A_{1Y} + A_{2Y} + A_{3Y}$

$R_Z = A_{1Z} + A_{2Z} + A_{3Z}$

$R = (R_X^2 + R_Y^2 + R_Z^2) \wedge ½$

$M_X = A_{1X}*D_{1X} + A_{2X}*D_{2X} + A_{3X}*D_{3X}$

$M_Y = A_{1Y}*D_{1Y} + A_{2Y}*D_{2Y} + A_{3Y}*D_{3Y}$

$M_Z = A_{1Z}*D_{1Z} + A_{2Z}*D_{2Z} + A_{3Z}*D_{3Z}$

$M = (M_X^2 + M_Y^2 + M_Z^2) \wedge ½$

We found a way to implement the accelerometers such that they are all mounted with a common axis. However, because we are not implementing the rotational calculations we do not need to worry about calculating distances to the center of mass of the helmet. The Threshold Monitor is very simple. It takes the R determined above and compares it to the thresholds stored. If the first threshold is exceeded, then a flag is set for main to store the data and timestamp. If the second threshold is exceeded, then another flag is set for main to send the alert to the Matchport. This block is completed and working as expected. Another simple block of code is the battery monitor. It simply queries the battery monitor and reports what the monitor returned as the battery charge. This block of code had to be improvised and is not working based off of the current estimated charge. Lastly, one of the more complicated blocks, Request Fielding, handles all communications between peripherals. This block takes requests from main or the Matchport, set variables as asked, reads data to or from the SD Card, and records the battery charge. This block was anticipated to be very difficult, however turned out to be simple. It is coded and working as expected. This is split into smaller blocks that can handle each peripheral's communication rather then all of them from one large, slow block.

10.4 Summary

The Instrumented Football Helmet is finally fully coded and assembled. The completion of the design has left the engineers involved feeling very proud of their accomplishments. The considerations they evaluated are the speed of the device, the accuracy of the measurements, and the variable settings controlled by the Matchport. These considerations drove most of the software choices such as the clock speed of the overall device, and the formulation of the algorithms used. The algorithm being used was chosen because of its simplicity and lack of second or third order integration needed to compute. Our microcontroller had an abundance of flash to utilize for this project, and the time to complete all calculations is much lower than the ATD conversion cycle. We are pleased that with all of the design assembled that it works wonderfully and could be taken to market if any of the engineers wanted to learn business.

11.0 Version 2 Changes

There are several aspects to our design that should be modified in the second iteration of our design. Firstly, there are several errors in the hardware that should be changed. For instance, the wiring to the micro SD card was mixed up and the traces to this would be modified to be correct the mistake. Additionally, there are a few traces that need to be added to the design to ensure all grounds are hooked up to one spot. Finally, a few of the connections to the micro need to be changed so that all the inputs are hooked up to ports on the microchip that accept inputs.

In addition to minor hardware fixes, there are several features that should be added from an ethical point of view. First of all, an exterior casing needs to be added to shield the battery. This casing would act to absorb any pressure on the battery and reduce strain on it. Additionally, this casing would also act to shield the player should the battery malfunction. Also, an auditory or visual notification should be added to warn users if the device is currently unresponsive or malfunctioning. Finally, a system could be added to the wifi that would send a notification to the subsystems of the design become unresponsive. This could be implemented by a heartbeat signal from the microcontroller to the Matchport that tells the Matchport everything is still working.

There are also a few changes that should be made to the larger system in terms of part selection and overall system design. Firstly, a microcontroller with more data and program memory would be selected. This would allow for the addition of a Fat formatting library to interface with the SD. Currently the microcontroller is too small to fit this library. Another change would be a modification to the power system as a whole. Currently it is not possible to both run the helmet and charge the battery at the same time. If the power system was modified, the steering diodes that were in the original design could be put back in which would allow for simultaneous charging and operating of the helmet. One final system modification that would be made is the addition of a reset button that would only reset the micro and not the system as a whole. Currently a system reset, resets the Matchport which takes a while to power back on. With the addition of a separate reset system for the micro, this problem would be solved.

12.0 Summary and Conclusions

As one can see, the PHI-Master was a complete success. The end functionality was as promised. The device reads and interprets the accelerometer outputs, stores and recovers data via the internal SD Card, allows the user to change thresholds and access this data from a computer on the sidelines, keeps track of the battery status, and keeps the user safe by detecting if the impacts received during a game are capable of causing a concussion. The design team put in over 1200 hours of work to pull this off and have learned much from the multiple layers of the development process. The final product that resulted from all this work will be donated to the Purdue University sports facilities to use as they deem fit. The final helmet will need to be recertified before it can actually be used in the field, or if disassembled will still be useable.

13.0 References

[1] AllDataSheet. (2008). [Online]. Available: http://www.alldatasheet.com/datasheet-pdf/pdf/166610/AD/ADXL193.html

[2] Atmel. (2008). [Online]. Available: http://www.atmel.com/dyn/Products/product_card.asp?part_id=2004

[3] Cliff Mehrtens. (2008, Aug.). Monitoring Concussions. [Online]. http://www.charlotteobserver.com/121/story/158175.html

[4] DigiKey. (2008). [Online]. Available: http://parts.digikey.ca/1/1/97501-sensor-accelerometer-3-axis-smd-0-1003800-5.html

[5] Grid Connect. (2008). [Online]. Available: http://www.gridconnect.com/matchport.html

[6] Lantronix. (2008). [Online]. Available: http://www.lantronix.com/device-networking/embedded-device-servers/wiport.html

[7] Maxim-IC. (2008). [Online]. Available: http://www.maxim-ic.com/quick_view2.cfm/qv_pk/4882

[8] Maxim-IC. (2005). [Online]. Available: http://www.maxim-ic.com/quick_view2.cfm/qv_pk/5385

[9] Microchip. (2008). [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/30275a.pdf

[10] Microchip. (2008). [Online]. Available: http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010551

[11] Paul Stoffregen. (2005, Feb.). Understanding FAT32 Filesystems. [Online]. Available: http://www.pjrc.com/tech/8051/ide/fat32.html

[12] Riddell. (2008). [Online] Available: http://www.riddell1.com/newsite/product_info.php?cPath=104_76_241&products_id=1457

[13] WikiAnswers. (2008). [Online]. Available: http://wiki.answers.com/Q/What_weighs_50_grams

[14] UltraLife (2008), [Online]. Available: http://www.ultralifebatteries.com/documents/techsheets/UBI-5154 UBBP01.pdf

[15] Riddell.com, "Riddell: Product Detail," 2008. [Online], Available: http://www.riddell1.com/newsite/product_info.php?cPath=104_76&products_id=1457. [Accessed: November 4, 2008].

[16] Stewart, Walter, “Helmet system including at least three accelerometers and mass memory and method for recording in real-time orthogonal acceleration data of a head,” US Patent 5978972, November 9, 1999, Available: http://www.freepatentsonline.com/5978972.pdf. [Accessed: November 4, 2008].

[17] Rush III, Gus A., “Sports helmet capable of sensing linear and rotational forces,” US Patent 5621922, April 22, 1997, Available: http://www.freepatentsonline.com/5621922.pdf. [Accessed: November 4, 2008].

[18] Crisco III, Joseph J., “System and method for measuring the linear and rotational acceleration of a body part,” US Patent 6826509, November 30, 2004, Available: http://www.freepatentsonline.com/6826509.pdf. [Accessed: November 4, 2008].

[19] Department of Defense, “Military Handbook: Reliability Prediction of Electronic Equipment”, [Online Document], 1990, [accessed November 14, 2008], http://www.sqconline.com/download/MIL-HDBK-217F.pdf.

[20] Microchip, “PIC18F23K20/24K20/25K20/26K20/43K20/44K20/45K20/46K20 Data Sheet”, [Online Document], 2008, [accessed November 14, 2008], http://ww1.microchip.com/downloads/en/DeviceDoc/41303D.pdf.

[21] Linear Technology, “LTC3440 Data Sheet”, [Online Document], 2001, [accessed November 14, 2008], http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1003,C1042,C1116,P2123,D3314.

[22] Texas Instruments, “TPS75733 Data Sheet”, [Online Document], 2002, [accessed November 14, 2008], http://focus.ti.com/lit/ds/symlink/tps75733.pdf.

[23] Linear Technology, “LT1303 Data Sheet”, [Online Document], 1995, [accessed November 14, 2008], http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1003,C1042,C1031,C1061,P1035,D2445.

[24] Carl Johan Rydh, Bo Svärd (2003, Jan) Impact on global metal flows arising from the use of portable rechargeable batteries. *The Science of The Total Environment[online]. Volume 302(Issues 1-3)*, Pages 167-184 Available: http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6V78-470V02T-2&_user=10&_rdoc=1&_fmt=&_orig=search&_sort=d&view=c&_acct=C000050221&_version=1&_urlVersion=0&_userid=10&md5=4a44174f22d6008bc37a6d3badb76e33

[25] Guy Crittenden, “TCI's facility in Pell City, Alabama” [online], Available: http://www.tci-pcb.com/belle.htm

[26] Riddell.com, “Riddell: Product Detail” 2008. [Online], Available: http://www.riddell1.com/newsite/product_info.php?cPath=104_76&products_id=1457. [Accessed: October 1, 2008]

[27] R. Frampton, “Effectiveness of Electronic Stability Control Systems in Great Britain”, Mar. 15, 2007[Online], Available: http://www.dft.gov.uk/pgr/roads/vehicles/vssafety/safetyresearch/esc [Accessed: Oct 1, 2008]

[28] Mouser.com “UBBP01 Datasheet” Sept. 8, 2006 [Online], Available**:** http://www.mouser.com/Search/ProductDetail.aspx?qs=bAKSY%2fctAC6uajF7UoG97A%3d%3d [Accessed: Oct. 1, 2008]

[29] Wiki.answers.com, “What is the weight of a football helmet” 2008 [Online] Available: http://wiki.answers.com/Q/What_is_the_weight_of_a_football_helmet [Accessed: Sept. 29, 2008]

[30] freescale.com, “MMA1212 datasheet” Mar. 2006 [Online], Available: http://www.freescale.com/files/sensors/doc/data_sheet/MMA1212D.pdf [Accessed Oct. 1, 2008]

[31] Microchip, “PIC18F23K20/24K20/25K20/26K20/43K20/44K20/45K20/46K20 Data Sheet”, [Online Document], 2008, [accessed October 2, 2008], http://ww1.microchip.com/downloads/en/DeviceDoc/41303D.pdf.

[32] Freescale, “MMA1212 Data Sheet”, [Online Document], 2006, [accessed October 2, 2008], http://www.freescale.com/files/sensors/doc/data_sheet/MMA1212D.pdf.

[33] Freescale, “MMA2301 Data Sheet”, [Online Document], 2006, [accessed October 2, 2008], http://www.freescale.com/files/sensors/doc/data_sheet/MMA2301D.pdf.

[34] Lantronix, “Matchport b/g Integration Guide”, [Online Document], 2007, [accessed October 2, 2008], http://www.lantronix.com/pdf/MatchPort_IG.pdf.

[35] Linear Technology, “LTC3440 Data Sheet”, [Online Document], Unknown Year, [accessed October 2, 2008], http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1003,C1042,C1116,P2123,D3314.

[36] Linear Technology, “LTC1751 Family Data Sheet”, [Online Document], Unknown Year, [accessed October 2, 2008], http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1003,C1039,C1133,P1904,D2062.

[37] Microchip, “MCP73837/8 Data Sheet”, [Online Document], 2007, [accessed October 2, 2008], http://ww1.microchip.com/downloads/en/DeviceDoc/22071a.pdf.

[38] Freescale, “16-Bit Microcontrollers”, [Online Forum], 2005, [accessed October 31, 2008], http://forums.freescale.com/freescale/board/message?board.id=16BITCOMM&message.id=85.

[39] Microchip, “PIC18F23K20/24K20/25K20/26K20/43K20/44K20/45K20/46K20 Data Sheet”, [Online Document], 2008, [accessed October 31, 2008], http://ww1.microchip.com/downloads/en/DeviceDoc/41303D.pdf.

[40] Mark Glenewinkel.(1995) System Design and Layout Techniques for Noise Reduction in MCU-Based Systems. Motorola. Austin, Texas. [Online]. Available: https://engineering.purdue.edu/ece477/Homework/CommonRefs/AN1259.pdf

[41] http://circuitcalculator.com/wordpress/2006/01/31/pcb-trace-width-calculator

.

Appendix A: Individual Contributions

A.1 Contributions of Adam Boeckmann

As the team leader, I organized most of the meetings during the course of the semester and delegated many of the duties during the design part of the course. As the only member with a strong interest in hardware design, I played a large role in the components selection, schematic creation, and PCB layout.

During the components selection, I did the initial research and came up with a large list of possible components. Of those, only the Matchport (our Wi-Fi device) made it into our final product. I also was responsible for the selection of our final battery monitor, charge controller, power components, accelerometers, and microSD components.

I played a large role in the schematic creation process. I designed the accelerometer clusters, worked with Kevin on the Matchport schematic and worked with Kevin and Drew on the Micro/power board. I figured out the layout software and showed team members how to use it correctly. As with the schematics I laid out the accelerometer PCBs, and worked with team members on the remaining two.

Once we moved into software, Drew and I did the majority of the soldering while Kevin and Michael worked on the software. I did a lot of the later fly wiring as we found and fixed incorrect schematic connections. Once most of the soldering was finished, I worked and developed our packaging with Drew.

Aside from working in lab on our design, I made sure to communicate with Villanova on a weekly basis. I spent several hours a week reviewing their progress and giving them feedback. In addition to communicating with Villanova, I also arranged for a helmet to be donated from the athletic department.

While Michael and Kevin did the majority of the software, Drew and I took up the challenge of getting the microSD card functioning. This required over a week of research, debugging, and coding.

A.2 Contributions of Andrew Camp

Over the course of the design of our project, I largely focused my efforts on getting the hardware together and functioning.

During the initial design of our product I helped in the location and selection of many of our parts. Particularly I looked into different microcontroller possibilities which included the PIC16 and 18 families. After selecting parts, I worked to create the schematic for our microcontroller board, which included several power supply parts. This initial design was later changed when we consolidated the power and SD PCB's onto the microcontroller board. However the pin out from the microcontroller remained largely unchanged.

After finishing the schematic, I finished the initial layout of our microcontroller PCB. As mentioned earlier, the power and SD PCBs were then consolidated onto our microcontroller PCB. After Michael made this change in the schematics, I helped him in creating a finalized version of our microcontroller PCB.

Once our PCB's arrived, I soldered and populated our 3 accelerometer PCBs as well as our Wifi. I also helped solder the parts onto with a small portion of our microcontroller PCB.

After our boards were up and running a few errors in the layout of our PCBs were found. I helped with several of the fixes for these errors, including fly wiring and drilling our board. Another aspect of the hardware that I was involved with was crimping up several of our wires to run between our boards.

I also helped with a couple aspects of the software. During the initial design phase of our software I made up the flowchart for our code. Also, at the end of our product creation, I did a lot of work to get the SD interface working. This included working with Adam to figure out how we would utilize the memory on our SD and creating several of the functions that interface with the SD card.

Also, I wrote the PCB analysis paper as well as the ethical and environmental impact paper.

A.3 Contributions of Kevin Hughes

The project accomplishments were typically tackled by everyone as a team. In the beginning, everyone researched parts and participated in team discussions about which parts to go with. I compiled HW3 about our design constraints.

During the schematic design phase, I took charge of the Matchport and its power circuit. I was able to collaborate with Mike on the power circuit because he was using it as well for the SD Card and PIC18.

During the PCB design phase, I completed 3 different layouts for the Matchport' PCB, and in the end included some advice from Adam.

My major contribution came from the software sections. I wrote the software design considerations, HW9, prior to our first day of coding. I talked with two professors about how to theorize our helmet as a rigid body or not. My sources said that our helmet certainly would act as a rigid body for small impacts, but both said that our high impacts we were dealing with might need to be modeled differently. In then end we decided through testing that our design was most closely modeled as a rigid body.

I made the first steps into programming the development processor finding resources showing us how to initialize the ADC and USART. I was able to get these modules functioning and then Mike followed me and added interrupts to the ADC module and the USART module, thus improving their functionality to what we needed for our project. Mike and I worked as a team for the rest of the programming.

Once we tried to program our processor we encountered errors. I was able to identify and correct these errors which turned out to only be problems with our cable connector we designed.

Mike and I ported the development code to the processor and continued to code how things would be implemented. I then started on getting the Matchport to work. This is where I got my first experience with soldering because I fried our buck boost and had to replace it. Fortunately, I was able to connect to the Matchport from my laptop and configure it.

I was able to find a test java applet and upload it into the Matchport which allowed us to see the communications to and from it. We used this interface to debug the rest of the code. I then wrapped up the functionality of the Accelerometers and helped debug the SD Card which Adam and Drew had started on. I was able to convert the buffer to have 512 bytes; however we changed this in the end when we realized the buffer was located in read-only memory. Mike and I was able to solve the software problems with the SD Card where Drew took over and wrapped up full functionality.

Finally, I revised my homework assignments to reflect our current choice of parts and software considerations for the final paper.

A.4 Contributions of Michael Olson

During the course of the semester my contributions to the project included both hardware and software design. First, I created the preliminary power supply schematic and updated it as we changed power supply components. This happened frequently as our power supply components were changed often; sometimes for changing requirements and other times to change a component we had found to be insufficient for our design after more research. Then I created the PCB layout for the power supply. During this process I created many of the custom footprints used by the power supply for the buck/boosts, the boost, and the LDO.

The design of our project changed quite a bit after the Design Review. We decided to combine the power supply, microcontroller, and SD card schematics onto one PCB where before they were split onto individual ones. I combined the three schematics into one and then began the process of designing the PCB layout. I was responsible, along with Adam, for redesigning this PCB. I first arranged each of the individual parts of the schematic together such as the buck/boost and its passive circuitry or the battery charge controller and its passive circuitry. This allowed for the capacitors and inductors to be placed as closely as possible to the chips they were tied to. It also allowed for these individual circuits to be condensed as much as possible. I made sure to reduce the number of vias which allowed these individual parts to be placed onto the schematic on both sides of the PCB.

After the PCBs were finished, I began working on the software part of our design along with Kevin. We first began working with the PIC development board in order to test the various hardware peripherals and the methods to interface with them. My main work during the software phase was developing the algorithms and code for implementing the analog to digital converters with interrupts and the USART both with and without interrupts enabled. Along with Kevin I developed the code for recognizing commands sent to the helmet and responding accordingly. I also developed all of the printing functions for printing constant strings, character arrays, variables, and characters. I also developed the applet used for the webserver to interface with the helmet. Finally, I helped Drew debug the micro SD code when he was having troubles tracking down some of his bugs.

Appendix B: Packaging

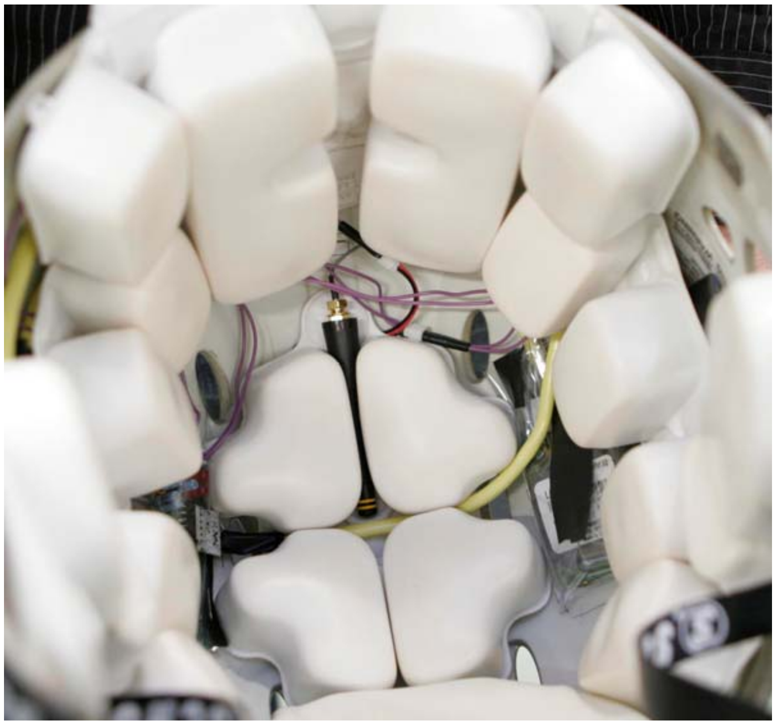


Figure B-1. Helmet internal packaging

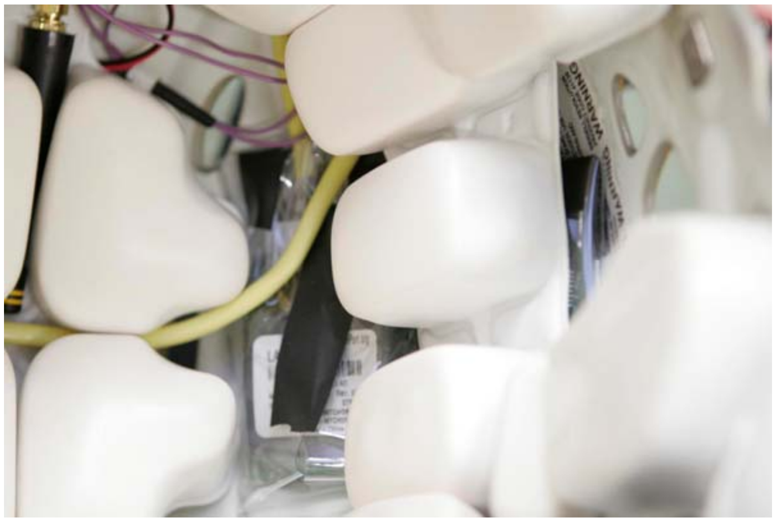


Figure B-2 Matchport placement in packaging

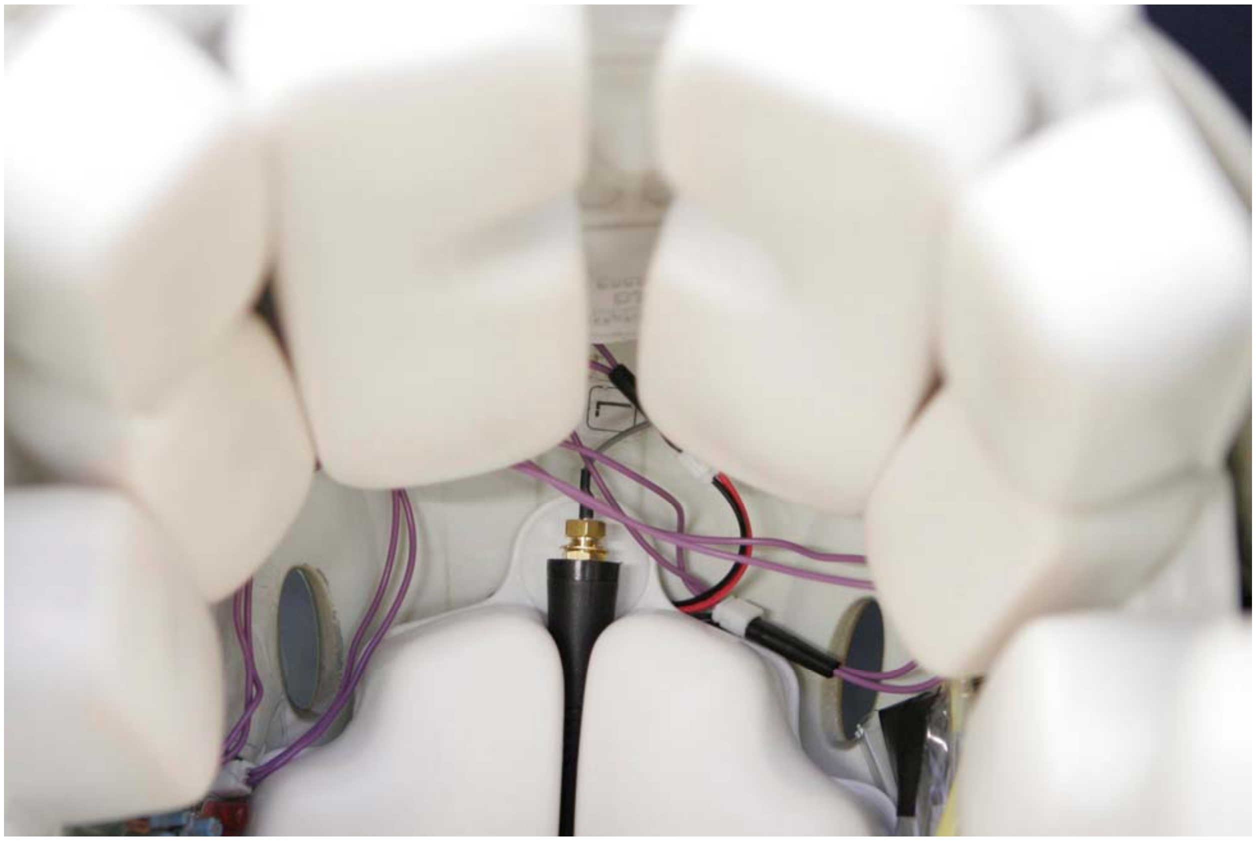


Figure B-3. Antenna placement in packaging

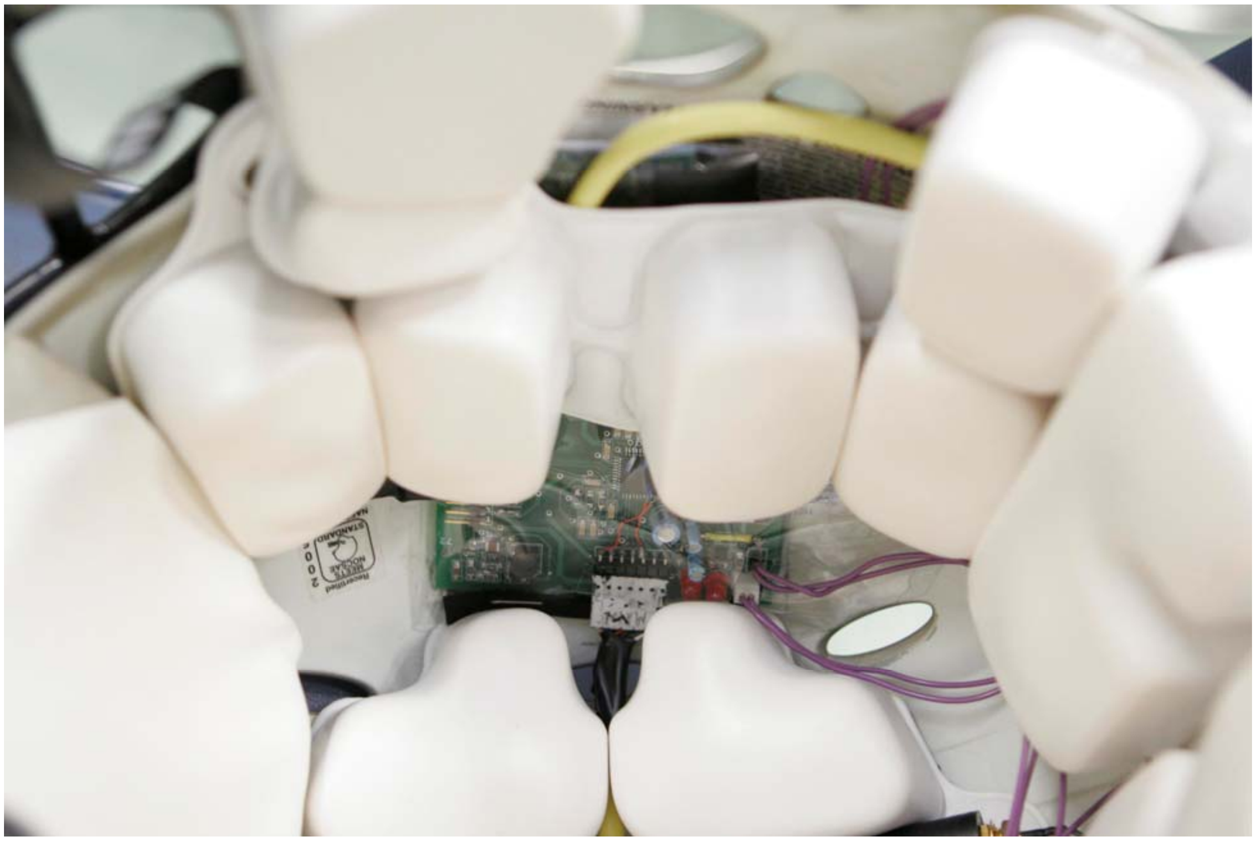


Figure B-4. Main PCB placement in packaging

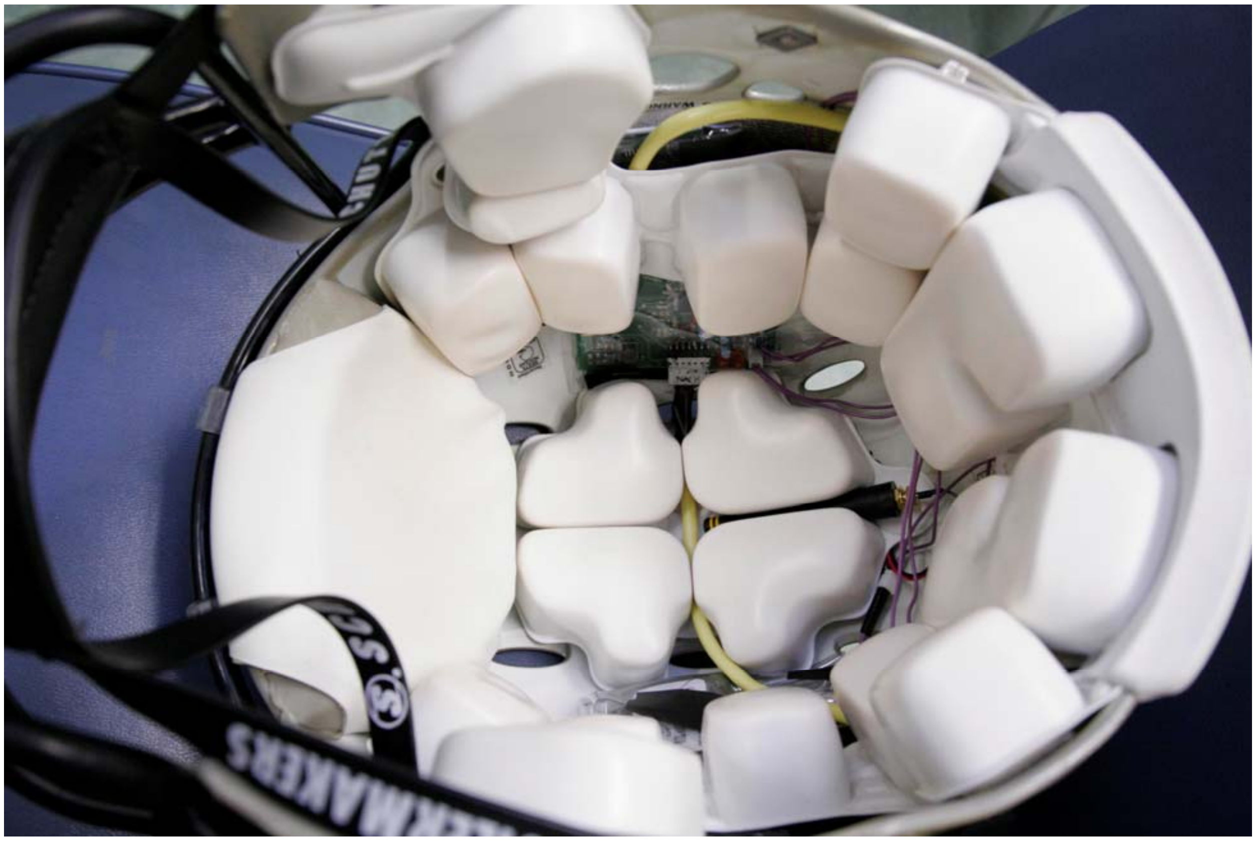


Figure B-5. Inside of helmet

Appendix C: Schematic

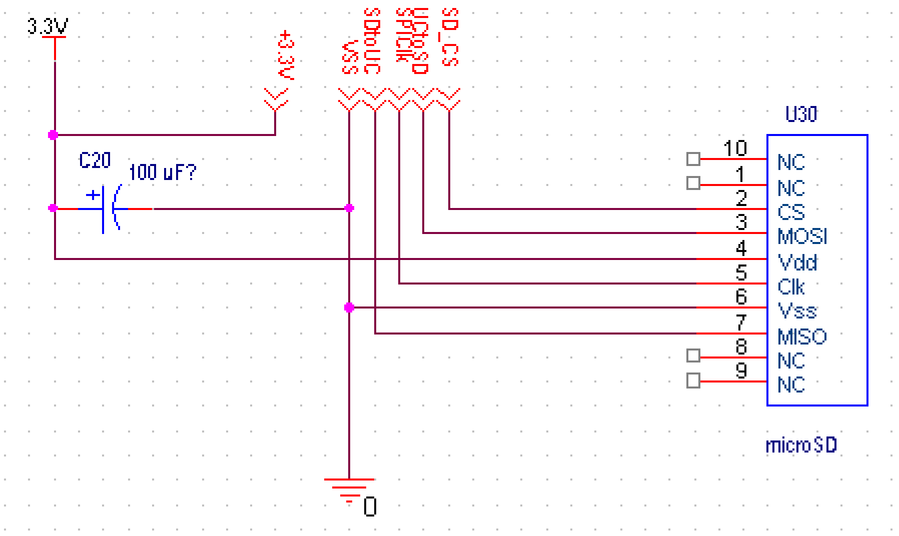


Figure C-1. microSD schematic

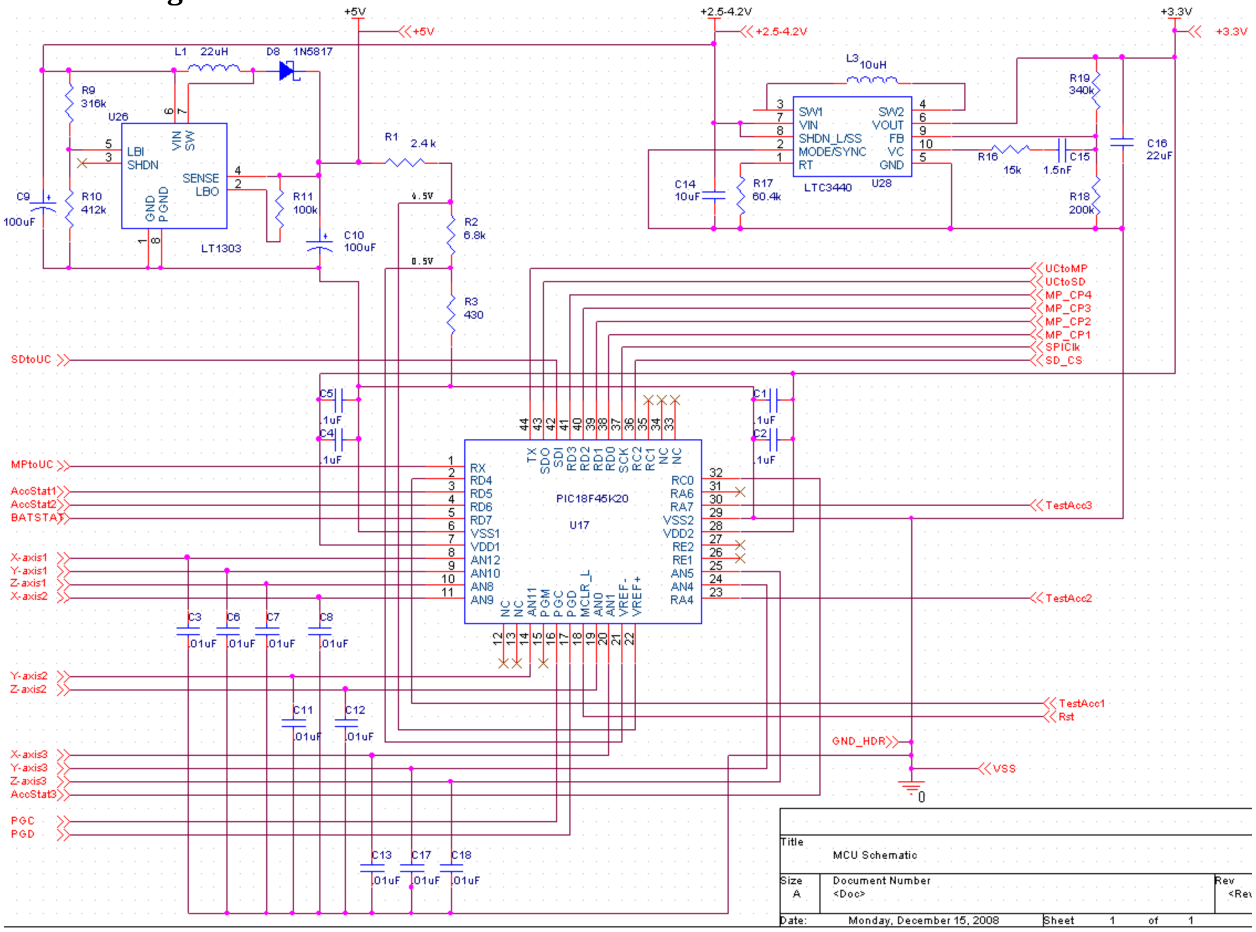


Figure C-2. Microcontroller Schematic

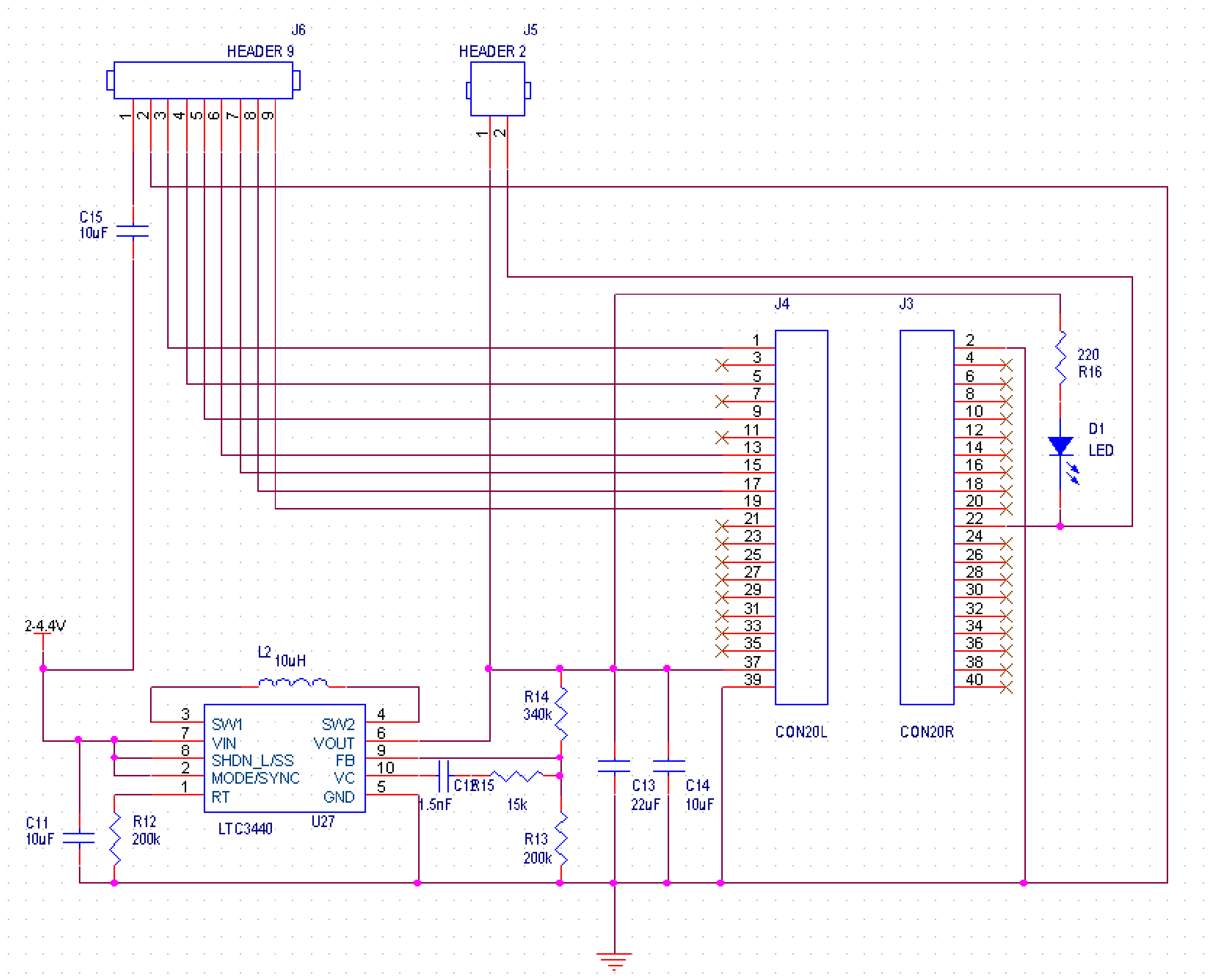


Figure C-3. Matchport Schematic

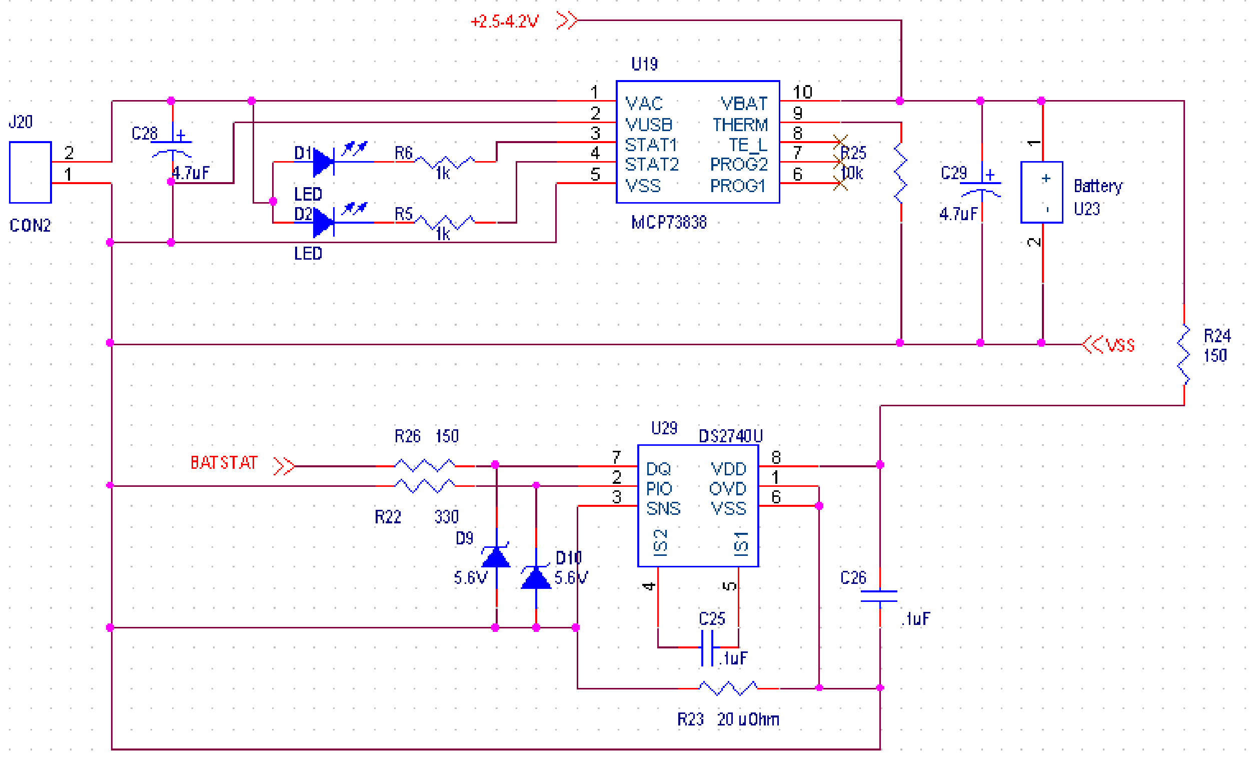


Figure C-4. Power Supply Schematic

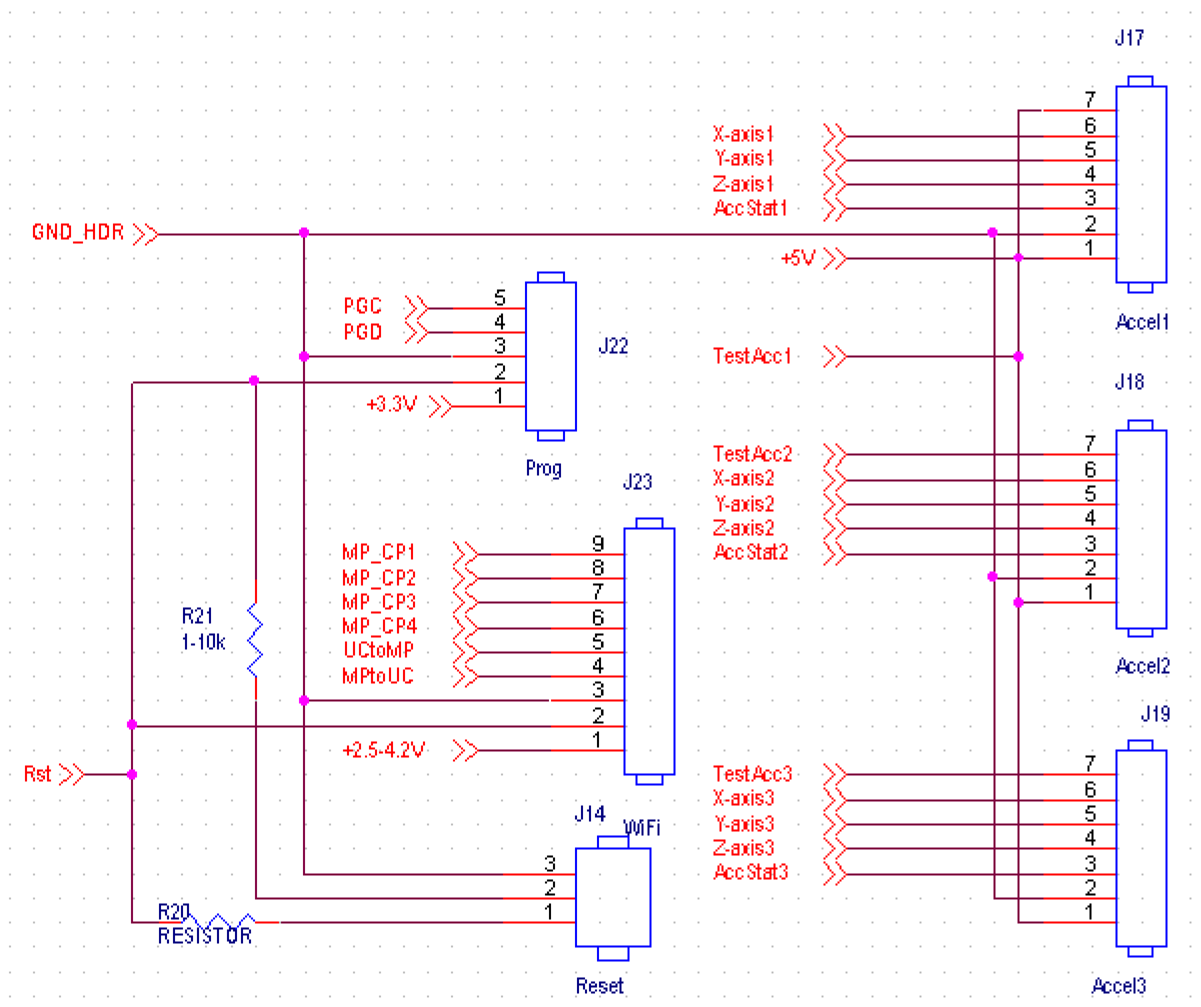


Figure C-5. Header Schematic

Appendix D: PCB Layout Top and Bottom Copper

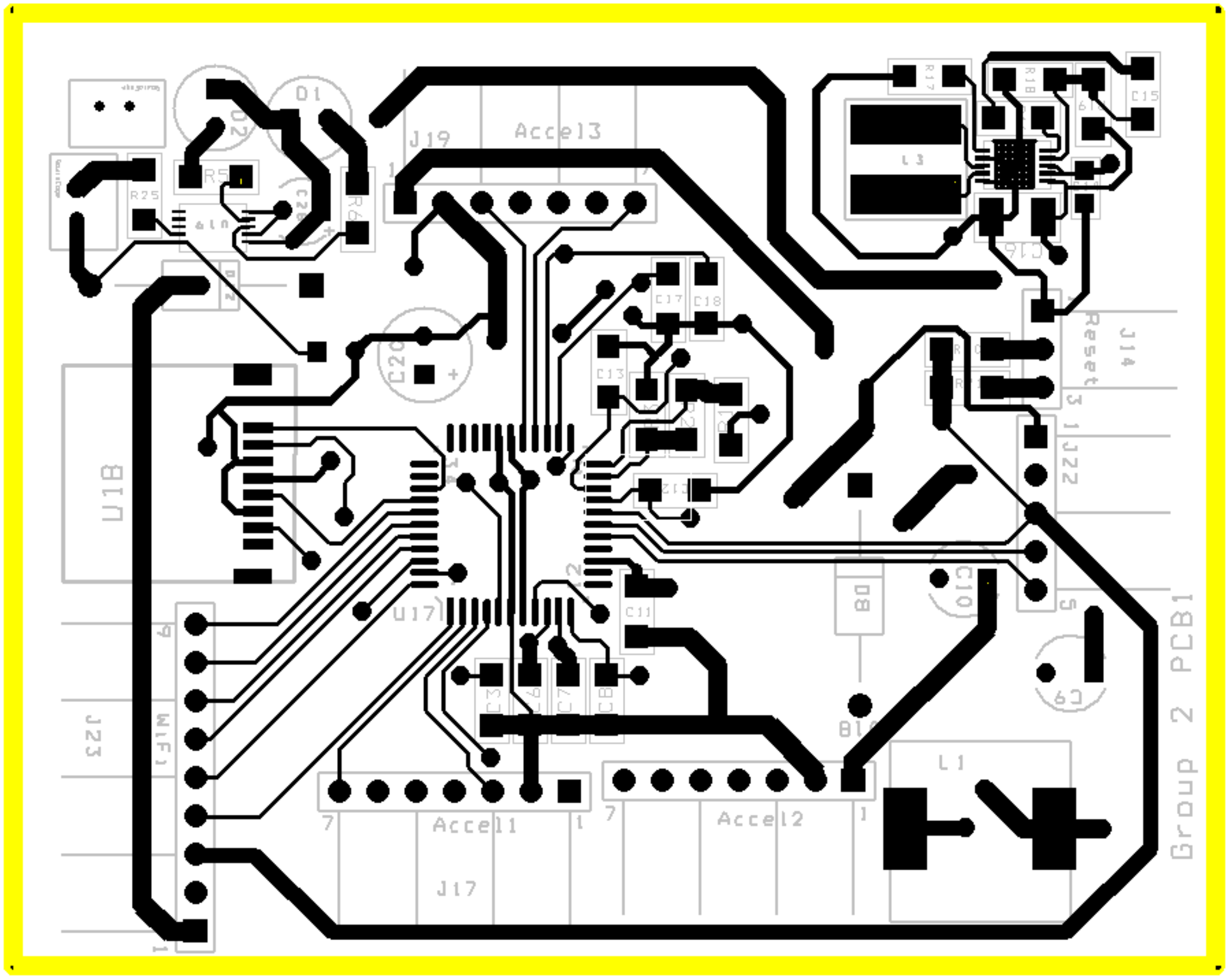


Figure D-1. Main PCB Top Layer

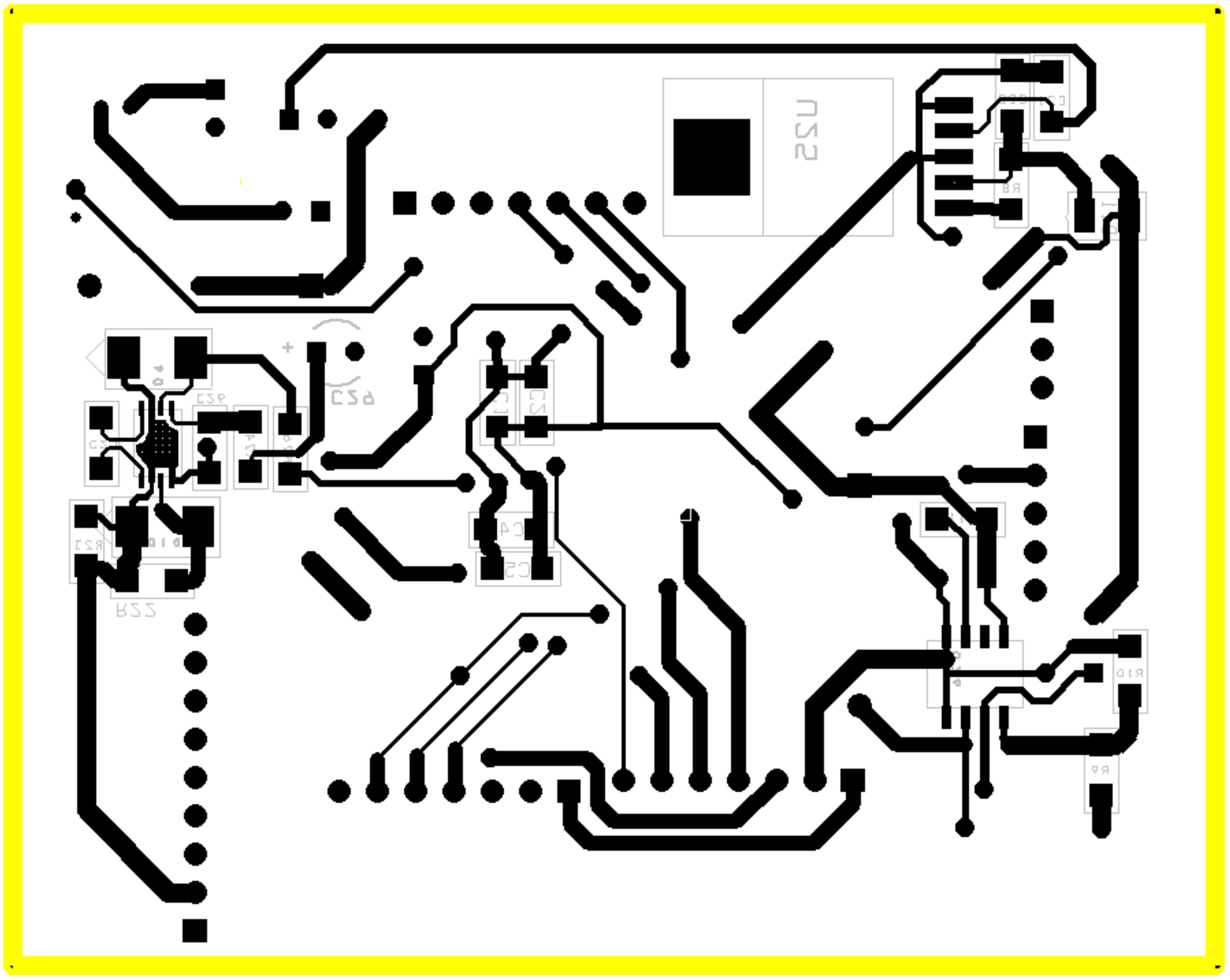


Figure D-2. Main PCB Bottom Layer

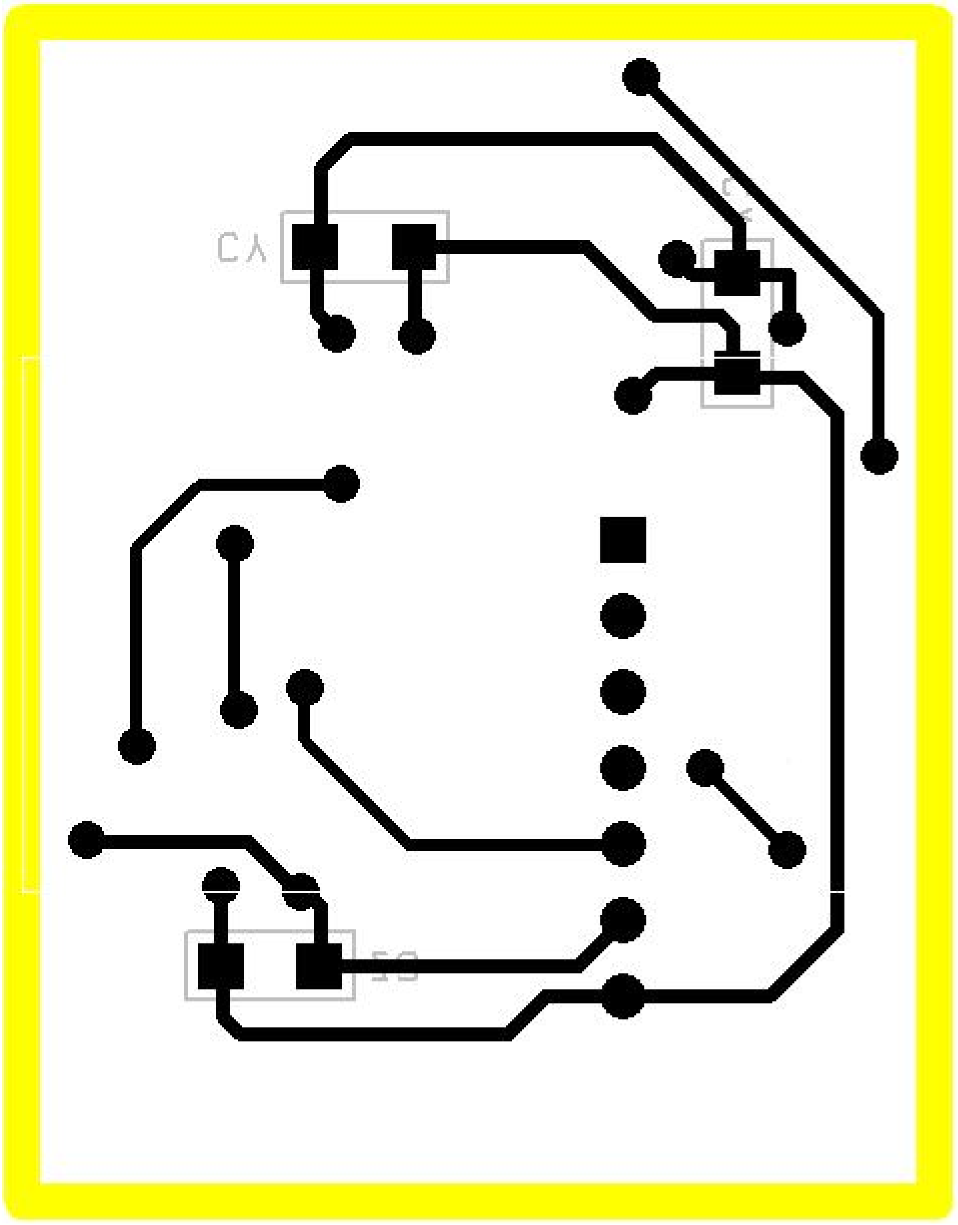


Figure D-3. Accelerometer PCB Bottom Layer

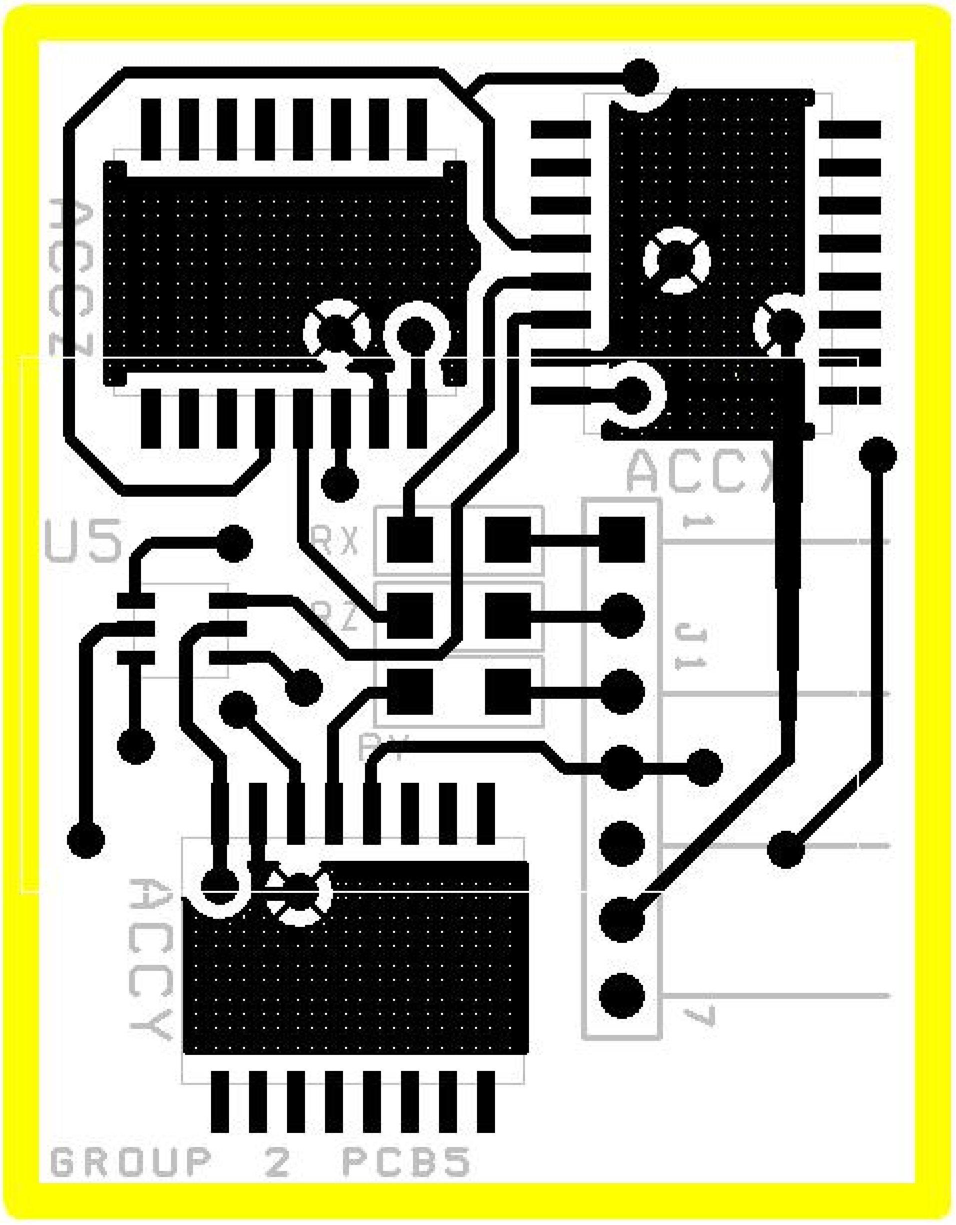


Figure D-4. Accelerometer PCB Top Layer

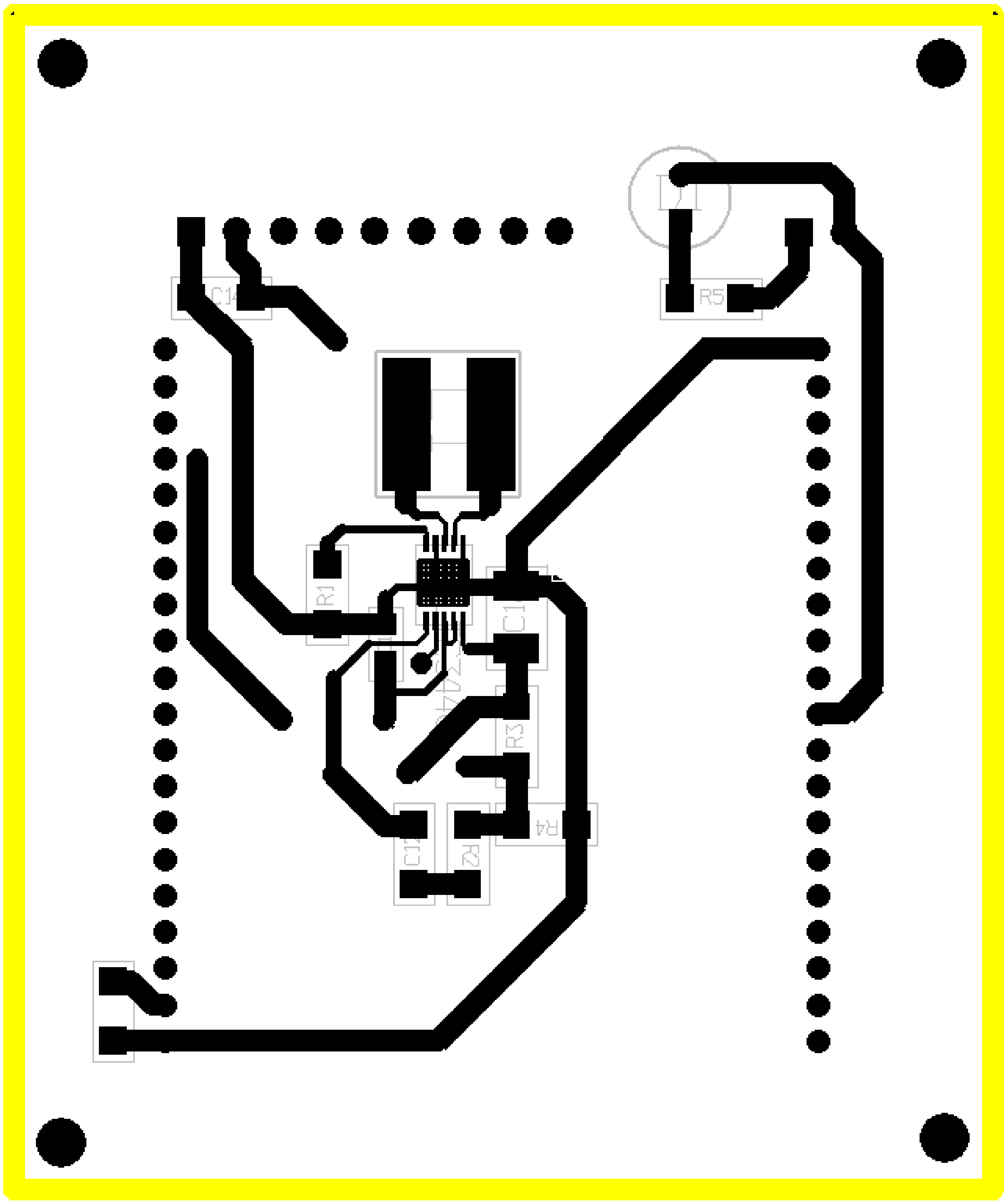


Figure D-5. Matchport PCB Bottom Layer

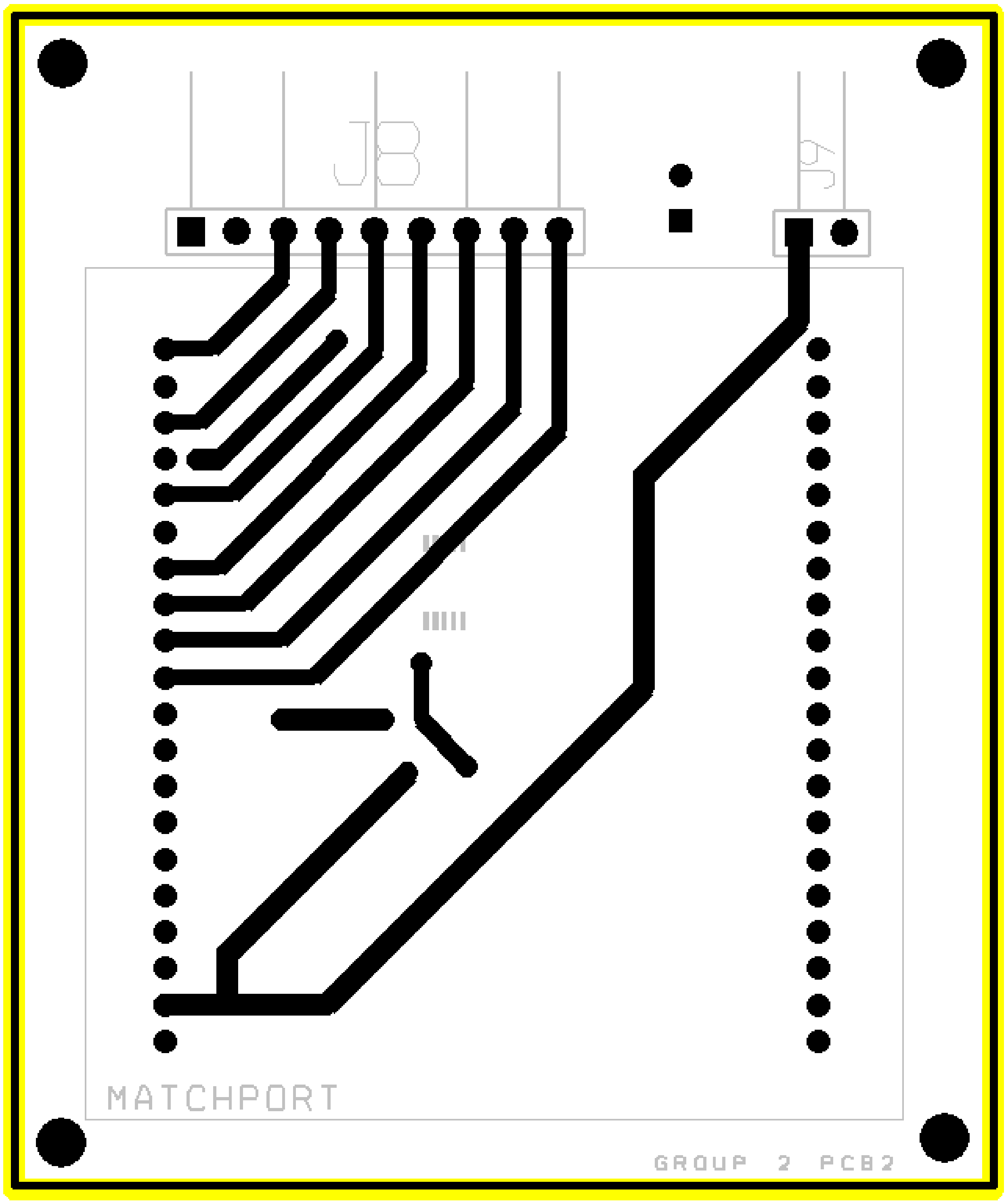


Figure D-6. Matchport PCB Top Layer

Appendix E: Parts List Spreadsheet

Part	Package	DigiKey Part No	MCU	WiFi	Accel	TOTAL
3440 inductor (10uH)	Custom	308-1541-1-ND	1	1	0	2
3440 capacitor (10uF)	805	587-1943-1-ND	1	1	0	2
3440 capacitor (10uF)	1206	PCC1894ct-ND	4	2	0	6
75733 capacitor (47uF)	1206	511-1452-1-ND	1	0	0	1
3440 capacitor (22uF)	1210	587-1384-1-ND	1	1	0	2
LT1303 inductor (22uH)	Custom	308-1445-1-ND	1	0	0	1
1N5817 schottky diode	Do41	497-4547-1-ND	1	0	0	1
battery connector		455-1127-1-ND	2	0	0	2
wall connector		455-1165-ND	1	0	0	1
capacitor (1uF)	1206	490-4452-1-ND	1	0	0	1
zener diode (5.6V)	custom	1smb5919bt3gosct-ND	2	0	0	2
capacitor (100uF)	TH	N/A	2	0	0	2
capacitor (1.5nF)	1206	N/A	1	1	0	2
capacitor (200uF)	TH	N/A	1	0	0	1
capacitor (.1uF)	1206	N/A	2	0	9	11
capacitor (4.7uF)	TH	N/A	2	0	0	2
LEDs	TH	N/A	2	1	0	3
Diode	TH	N/A	1	0	0	1
Diode	1206	N/A	1	0	0	1
PIC18F45K20			1	0	0	1
MMA1212			0	3	0	3
MMA2301			0	6	0	6
HR1940CT			1	0	0	1
DS2740U			1	0	0	1
MCP73838			1	0	0	1
LTC3440			1	0	1	2
LT1303			1	0	0	1

Appendix F: Software Listing

phi-master.h

```
/*
 * This is the main header file for the PHI-Master
 *
 * For this project, the PIC18F45K20 is used with the following peripherals:
 *
 *       ADC:
 *    AN0,1,4,5,8-12
 *    using external Vref- and Vref+
 *    Sampling at ~400 Hz
 *
 *  SPI
 *    Used for SD card interface
 *    Use SDO, SDI, and SCK
 *    Need to keep clock rate down at least during card handshaking
 *
 *  USART (RS232-C)
 *    Used for Matchport interface
 *          Tx, Rx
 *
 *       Timer0
 */

/*
 * Matchport interfacing
 *
 * using USART (RS232-C)
 * 9600 bits/sec baud rate
 * Tx, Rx pins
 *
 * Transmit:
 * use interrupts at PIR1bits.TXIF
 * enable these interrupts every time a
 * command/string is ready to be sent, and
 * disable it whenever the command has been sent
 *
 * Receive:
 * use interrupts at PIR1bits.RXIF
 * these interrupts only need to be enabled
 * during startup and then the interrupt will
 * only trigger when a byte has been received
 *
 * have to recognize certain commands/requests
 * sent from the matchport and then send a response
 *
 * Inputs to recognize:
 * Command 1: 0x01 - store the following four values received as thresholds
 *                   1: linear impact concussion alert threshold
 *                   2: linear impact recording threshold
 *                   3: rotational concussion alert threshold
 *                   4: rotational recording threshold
 *                   maybe return a bitwise or of all four imputs or just some sort of
receive flag
 * Command 2: 0x02 - send all data stored on SD card to matchport
 * Command 3: 0x03 - reformat SD card
 */

/*
 * SD card interfacing
 *
 * Upon power reset:
 * send CMD0: 40h 00h 00h 00h 00h 95h
 *
 * access considerations:
 *   Possibility 1:
 *     Whenever enough data has been stored in data or
 *     enough time has been elapsed between impacts, write
 *     that data to the SD card. Then increment the current
 *     sd write address to immediately after the last piece
 *     of data written to the memory card.
 *
 *     On a reset, after setting the card to idle the card is
 *     read until an unused memory location is found.  At this
 *     point, an EOF is written to this memory location and the
 *     current "write to" address is incremented.
 *
 *     TO DO: see if the SD card can be written to at, say 0x0000
 *                           and then written to again at 0x0002.  if this is the
 *                           case, this design will work.  Otherwise, something
 *            else will have to be thought of
 */
/*
 *       Calculation possibilities:
 *                1. Use ADC interrupts to buffer the inputs as
 *                   we are calculating the values
 *                2. Keep interrupts off on the ADC and just
 *                   read each ADC in order, then begin calculations
 *
 *       Issues: might not be able to do any calculations without at least three
 *                         inputs (x, y, z) to do resultant vector calculation
 */
#include <p18cxxx.h>
#include <delays.h>
#include <timers.h>
#include <usart.h>
#include <adc.h>
#include <spi.h>
#include <math.h>
//#include "rev2_SD.h"
#include <p18cxxx.h>

#define OK 0x00
#define PRINT_ERR 0x01
#define INIT_ERR 0x02
#define SD_ERR 0x03

#define SD_BLOCK_SIZE 128

#define LNR_THRSH_REC_ADDR 0x00
#define LNR_THRSH_WRN_ADDR 0x01
#define  RTN_THRSH_REC_ADDR 0x02
#define  RTN_THRSH_WRN_ADDR 0x03
#define  BATSTAT_ADDR 0x04
```

```
#define SECCNT1 18
#define SECCNT2 217
#define ADC_RATE 10
#define bfrsize 5
#define ADCREADS 5

void low_intr(void);
void high_intr(void);
void writeString (const rom char *, int);
void checkCmd(void);
void SetVariable(char);
void ReadVariable(char);
void WriteEEPROM(int, char);
void wait_on_wr(void);
void writeVar(unsigned int);
int TestAccOne(void);
int TestAccTwo(void);
int TestAccThree(void);
char getAdcChannel(int);
int getVar(int);
void setVar(unsigned int, int);
void printAccel(const rom char*, int);
void AccP(void);
void init(void);
void initAccels(void);
void writeVariable(char [], int);
void writeCharacter(char);
int abs(int);
void calc(void);
void printSDRecords (void);
void initFromSD(void);
void resetSD(void);
void newBlock(unsigned long loc);
void writeSDRec( unsigned char level);
//void testCurSDVals(void);

//SD Functions
char setup_SDSPI( void );
char SDReadBlock( unsigned long );
char SDWriteBlock(unsigned long );
void InitSPI( void );
char InitSD( void );
char SD_WriteCommand(char* cmd);
char SPIRead( void );
void SPIWrite(char data);
void SD_setStart( void );
void SD_SetCurrBlock(unsigned long newblock);
unsigned long SD_GetCurrBlock(void);
unsigned char SD_GetSample( int cnt );
void SD_SetSample( int cnt, unsigned char sample );
void SD_writeCurr( void );
void SD_readCurr( void );
void WriteSamples(char* buff);
```

microSD.h

```
#include <p18cxxx.h>

#define OK 0x00
#define PRINT_ERR 0x01
#define INIT_ERR 0x02
#define SD_ERR 0x03

#define SD_BLOCK_SIZE 64

char setup_SDSPI( void );
void SD_setStart( void );
char SD_GetSample( int );
void SD_SetSample( int, char );
void SD_writeCurr( void );
void SD_readCurr( void );
```

main.c

```
#include "phi_master.h"

//SD constants
#define GAME_LOC_1 0
#define GAME_LOC_0 1
#define NUMBLOCK_LOC_3 2
#define NUMBLOCK_LOC_2 3
#define NUMBLOCK_LOC_1 4
#define NUMBLOCK_LOC_0 5
#define THRESH_ROTL_LOC 6
#define THRESH_ROTH_LOC 7
#define THRESH_LINL_LOC 8
#define THRESH_LINH_LOC 9
#define BATSTAT_LOC_1 10
#define BATSTAT_LOC_0 11
#define CUR_USED_LOC 0
#define LINROT_OFFSET 1
#define GAMENUM_OFFSET_1 2
#define GAMENUM_OFFSET_0 3
#define PROCTIME_OFFSET_1 4
#define PROCTIME_OFFSET_0 5
#define MAXVAL_OFFSET 6
#define RECORD_SIZE 6

#pragma config WDTEN = OFF
#pragma config MCLRE = OFF
#pragma config FOSC = INTIO67
#pragma config LVP = OFF
#pragma config DEBUG = OFF
#pragma config BOREN = OFF

void wait_on_wr(void)
{
         while(EECON1bits.WR)
                  continue;
}
```

```
#pragma code high_vector_section=0x8
void
high_vector(void)
{
         _asm GOTO high_intr _endasm
}
#pragma code low_vector_section=0x18
void
low_vector(void)
{
         _asm GOTO low_intr _endasm
}
#pragma code

int count1 = 0;
int count2 = 0;
int count3 = 0;
int countADC = 0;
int countADC2 = 0;
int numCalc = 0;
int numAdcRead = 0;
const rom char *sendString;
int sendLength = 0;
int sendIndex = 0;
char sendDoneFlag = 0x00;
char sendVarFlag = 0x00;
char cmdInFlag = 0x00;
char txInProgressFlag = 0x00;
char testTimerFlag = 0x00;
char run_adc_flag = 0x00;
char adc_flag = 0x00;
char calc_flag = 0x00;
char debugAccelFlag = 0x00;
char DBGACFLAG = 0x00;
char A1 = 0x00;
char A2 = 0x00;
char A3 = 0x00;
char commandIn[6];
int rtnThrshRec, rtnThrshWrn, lnrThrshRec, lnrThrshWrn;
int cmdLen = 3;
int cmdIndex = 0;
char varString[7];
int X1, X2, X3, Y1, Y2, Y3, Z1, Z2, Z3;
int curAccum = 0;
int batteryStatus = 0;
int battStatusCounter = 0;
int batcnt1 = 0;
int batcnt2 = 0;
int batcnt3 = 0;
char batflg = 0xFF;

//SD Variables

//# of seconds the micro has been running since startup
unsigned int procRunTime = 0;
//unsigned char runTimeL = 0x00;
//unsigned char runTimeH = 0x00;
```

```
//the game number (used for timestamping collisions)
unsigned int gameNumber;
//# of blocks used on SD
unsigned long blocksUsed;
//# of recorded impacts
unsigned int numImpacts;


#pragma interruptlow low_intr
void low_intr (void) {
         unsigned long temp;
         PIR1bits.TMR2IF = 0;
         countADC++;
         TMR2 = 0;

         if (batflg == 0xFF){
                  batcnt1++;
                  if (batcnt1 == SECCNT1) {
                           batcnt2++;
                           batcnt1=0;
                           if (batcnt2 == SECCNT2) {
                                    batcnt3++;
                                    if (batcnt3 == 300) {//was 300
                                             batteryStatus -= 11;
                                             writeString("\n",1);
                                             printAccel("BS", batteryStatus/18);
                                             //WriteEEPROM(batteryStatus,
BATSTAT_ADDR);
                                             //temp = SD_GetCurrBlock();
                                             SD_setStart();
                                             SD_readCurr();
                                             SD_SetSample(BATSTAT_LOC_1,
(batteryStatus & 0xFF00)>>8);
                                             SD_SetSample(BATSTAT_LOC_0,
batteryStatus & 0x00FF);
                                             SD_writeCurr();
                                             SD_SetCurrBlock(1+blocksUsed);
                                             SD_readCurr();
                                             batcnt3 = 0;
                                    }
                                    batcnt2 = 0;
                           }
                  }
         }

         if (testTimerFlag == 0xFF) {
                  count1++;
                  if (count1 == SECCNT1) {
                           count2++;
                           count1=0;
                           if (count2 == SECCNT2) {
                                    writeString("a sec!\n", 7);
                                    count2 = 0;
                           }
                  }
         }

         if (DBGACFLAG == 0xFF) {
                count1++;
                if (count1 == SECCNT1) {
                        count2++;
                        count1=0;
                        if (count2 == SECCNT2) {
                                count3++;
                                if (count3 == 2){
                                        AccP();
                                        count3 = 0;
                                }
                                count2 = 0;
                        }
                }
        }

        if (countADC == 10) {
                countADC = 0;
                countADC2++;
                if (countADC2 == 400){
                        calc_flag = 0xFF;  //Do math.
                        countADC2 = 0;
                }
                adc_flag = 0xFF;
                numAdcRead = 0;
                numCalc = 0;
        }
}

#pragma interrupt high_intr
void high_intr(void) {
        char curChar;
        // this is where high interrupts are handled
        // possible interrupts: adc, usart
        if (PIR1bits.TXIF == 1) {
                PIR1bits.TXIF = 0;
                if (sendVarFlag == 0x00) {
                        if (sendLength != sendIndex) {
                                // transmit the next char
                                curChar = sendString[sendIndex];
                                WriteUSART(curChar);
                                sendIndex++;
                                if (sendIndex == sendLength) {
                                        sendDoneFlag = 0xFF;
                                        PIE1bits.TXIE = 0;
                                }
                        }
                } else {
                        if (sendLength != sendIndex) {
                                curChar = varString[sendIndex];
                                WriteUSART(curChar);
                                sendIndex++;
                                if (sendIndex == sendLength) {
                                        sendDoneFlag = 0xFF;
                                        sendVarFlag = 0x00;
                                        PIE1bits.TXIE = 0;
                                }
                        }
                }
        }
        if (PIR1bits.RCIF == 1) {
                PIR1bits.RCIF = 0;
                curChar = ReadUSART();
                if ((curChar == '>') && (cmdIndex == 0)) {
                        cmdLen = 6;
                }
                else if (cmdIndex == 0) {
                        cmdLen = 3;
                }
                if (cmdInFlag == 0x00) {
                        commandIn[cmdIndex] = curChar;
                        cmdIndex++;
                        if (cmdIndex == cmdLen) {
                                cmdInFlag = 0xFF;
                        }
                }
        }
        if (PIR1bits.ADIF == 1) {
                adc_flag = 0xFF;
                PIR1bits.ADIF = 0;
                numCalc++;
        }
}

void writeString (const rom char *str, int length) {
        sendString = str;
        sendLength = length;
        sendIndex = 0;
        sendDoneFlag = 0x00;
        PIE1bits.TXIE = 1;
        while(sendDoneFlag == 0x00);
}

void writeVar(unsigned int value) {
        int calcVal = value%1000;
        varString[0] = '>';
        varString[2] = ':';
        varString[3] = (char)((calcVal/100)+48);
        varString[4] = (char)(((calcVal%100)/10)+48);
        varString[5] = (char)(((calcVal%100)%10)+48);
        varString[6] = '\n';

        sendVarFlag = 0xFF;
        sendLength = 6;
        sendIndex = 0;
        sendDoneFlag = 0x00;
        PIE1bits.TXIE = 1;
        while (sendDoneFlag == 0x00);
}

void writeVariable(char str[], int length) {
        int i, len;
        for (i = 0; i < length; i++) {
                if (i == 7) break;
                varString[i] = str[i];
        }
        sendLength = i;
```

```
         sendVarFlag = 0xFF;
         sendIndex = 0;
         sendDoneFlag = 0x00;
         PIE1bits.TXIE = 1;
         while(sendDoneFlag == 0x00);
}

void writeCharacter(char c) {
         varString[0] = c;
         sendLength = 1;
         sendVarFlag = 0xFF;
         sendIndex = 0;
         sendDoneFlag = 0x00;
         PIE1bits.TXIE = 1;
         while(sendDoneFlag == 0x00) ;
}

void main (void)
{
         unsigned int result;
         int odd = 0;
         int three_multiple = 0;
         char c = 0x0C;
         int numsamples = 0;

         //batReset();

         init();

         while (1) {
                  if ((cmdInFlag == 0xFF) && (txInProgressFlag == 0x00)) {
                           checkCmd();
                  }
                  if ((txInProgressFlag == 0xFF) && (sendDoneFlag = 0xFF)) {
                           sendDoneFlag = 0x00;
                           txInProgressFlag = 0x00;
                           cmdInFlag = 0x00;
                           cmdIndex = 0;
                  }

                  if (adc_flag == 0xFF) {
                           // check numCalc is odd
                           odd = numCalc%2;
                           if (odd == 1) {
                                    // yes: get adc result, increment numcalc and
numAdcRead, set calc flag
                                    result = ReadADC();
                                    result &= 0xFFFC;
                                    setVar(result, numAdcRead);
                                    numCalc++;
                                    numAdcRead++;
                                    //calc_flag = 0xFF;
                           }
                           // set next adc channel, begin conversion, setADC flag to
0
                           if (numAdcRead < 9) {
                                    SetChanADC(getAdcChannel(numAdcRead));
                                    ConvertADC();
```

```
                        }
                        adc_flag = 0x00;
                }

                if (calc_flag == 0xFF) {
                        calc();
                }
        }
}

void init(void) {
        int i, j;
        int status;
        i = 0;
        j = 0;

        RCONbits.SBOREN = 0;
        varString[0] = '>';
        varString[1] = ' ';
        varString[2] = ':';
        debugAccelFlag = 0x00;

        TRISA &= 0b01101111;
        PORTA = 0b00000000;
        TRISC &= 0b10010011;
        PORTC = 0b00000000;
        TRISD &= 0b11101111;
        PORTD = 0b00000000;

        // read from EEPROM
        EECON1bits.EEPGD = 0;
        EEADR = LNR_THRSH_REC_ADDR;
        EECON1bits.RD = 1;
        lnrThrshRec = (int)EEDATA;
        if (lnrThrshRec  <= 0){
                lnrThrshRec = 50;
        }

        EECON1bits.EEPGD = 0;
        EEADR = LNR_THRSH_WRN_ADDR;
        EECON1bits.RD = 1;
        lnrThrshWrn = (int)EEDATA;
        if (lnrThrshWrn  <= 0){
                lnrThrshWrn = 50;
        }

        EECON1bits.EEPGD = 0;
        EEADR = RTN_THRSH_REC_ADDR;
        EECON1bits.RD = 1;
        rtnThrshRec = (int)EEDATA;
        if (rtnThrshRec  <= 0){
                rtnThrshRec = 50;
        }

        EECON1bits.EEPGD = 0;
        EEADR = RTN_THRSH_WRN_ADDR;
        EECON1bits.RD = 1;
        rtnThrshWrn = (int)EEDATA;
        if (rtnThrshWrn <= 0){
                rtnThrshWrn = 50;
        }

        EECON1bits.EEPGD = 0;
        EEADR = BATSTAT_ADDR;
        EECON1bits.RD = 1;
        batteryStatus = (int)EEDATA;
        if (batteryStatus  <= 0){
                batteryStatus = 1800;
        }

        // Fosc = 16MHz
        OSCCONbits.IRCF0 = 1;
        OSCCONbits.IRCF1 = 1;
        OSCCONbits.IRCF2 = 1;
        WDTCON = 0; // WDT disabled

        // enable USART
        OpenUSART( USART_TX_INT_OFF & USART_RX_INT_ON & USART_ASYNCH_MODE &
USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_LOW, 25);

        // open timer 2
        OpenTimer2 ( TIMER_INT_ON & T2_PS_1_4 & T2_POST_1_1 );
        PR2 = 0xFF;

        initAccels();

        // Enable ADC interrupts
        PIE1bits.ADIE = 1;
        PIR1bits.ADIF = 0;

        // enable interrupts and interrupt priority
        RCONbits.IPEN = 1;
        PIR1 = 0;
        PIE1bits.TMR2IE = 1;
        IPR1bits.TMR2IP = 0;
        INTCONbits.GIEH = 1;
        INTCONbits.GIEL = 1;

        //SD Card
        //InitSPI();
        setup_SDSPI();
        SD_setStart();
        initFromSD();
        writeString("OK",2);
}

void initAccels(void) {
        int i = 0;
        int j = 0;

        OpenADC( ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_0_TAD,
                            ADC_CH0 & ADC_CH1 & ADC_CH4 & ADC_CH5 & ADC_CH8 & ADC_CH9
& ADC_CH10 & ADC_CH11 & ADC_CH12 &
                            ADC_INT_ON & ADC_VREFPLUS_EXT & ADC_VREFMINUS_EXT,
                            0);
        Delay10TCYx(5);
        for (j=0;j<15000;j++){
                i++;
                i--;
        }
        if (TestAccOne()){
                i++;
                A1=0xFF;
        }
        for (j=0;j<3000;j++){
                i++;
                i--;
        }
        if (TestAccTwo()){
                i++;
                A2=0xFF;
        }
        for (j=0;j<3000;j++){
                i++;
                i--;
        }
        if (TestAccThree()){
                i++;
                A3=0xFF;
        }
        for (j=0;j<3000;j++){
                i++;
                i--;
        }
        if( i == 3 ){
                run_adc_flag = 0x00;
                while(1);
        }else if(i == 2){
                run_adc_flag = 0x00;
                while(1);
        }else{
                run_adc_flag = 0xFF;
                i=0;
        }
}

void checkCmd(void) {
        char testStr[] = "Stop it guys... I have had enough!";
        char retStr[] = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
        unsigned char tester;
        int testSD = 0;
        int numRun = 0;
        int timeWaster = 0;
        int i=0;
        unsigned char a;
        unsigned char b;


        int counter;
        txInProgressFlag = 0xFF;

        // compare individual characters of command and print a string telling type
of command received
```

```
	if (commandIn[0] == '?') {
		if (commandIn[2] == ':') {
			if (commandIn[1] == 'A') {
				ReadVariable('A');
				return;
			} else if (commandIn[1] == 'B') {
				ReadVariable('B');
				return;
			} else if (commandIn[1] == 'C') {
				ReadVariable('C');
				return;
			} else if (commandIn[1] == 'D') {
				ReadVariable('D');
				return;
			}
		}else if(commandIn[1] == 'B'){ //?BS
			if (commandIn[2] == 'S') {
				if (batcnt3 > 55){
					batteryStatus -=
(int)(11*((float)(batcnt3/300.0)));
					batcnt3 = 0;
				}
				writeString("\n",1);
				printAccel("BS", (int)(batteryStatus/18));
				return;
			}
		}
	} else if (commandIn[0] == '>') {
		if (commandIn[2] == ':') {
			if (commandIn[1] == 'A') {
				SetVariable('A');
				return;
			} else if (commandIn[1] == 'B') {
				SetVariable('B');
				return;
			} else if (commandIn[1] == 'C') {
				SetVariable('C');
				return;
			} else if (commandIn[1] == 'D') {
				SetVariable('D');
				return;
			}
		}
		else if (commandIn[1] == 'T') {
			if ((commandIn[2] == 'I') && (commandIn[3] == 'M') &&
				(commandIn[4] == 'E') && (commandIn[5] == 'R')) {
				//writeString("Easter Egg! ", 12);
				if (testTimerFlag == 0x00) {
					testTimerFlag = 0xFF;
				} else {
					testTimerFlag = 0x00;
				}
				return;
			}
		} else if (commandIn[1] == 'K') {
			if ((commandIn[2] == 'E') && (commandIn[3] == 'V') &&
				(commandIn[4] == 'I') && (commandIn[5] == 'N')) {
```

```
                                writeSDRec(67);
                                varString[1]='B';
                                writeVar(blocksUsed);
                                varString[1]='T';
                                writeVar(procRunTime/1000);
                                varString[1]='U';
                                writeVar(procRunTime);
                                return;
                        }
                } else if (commandIn[1] == 'M') {
                        if ((commandIn[2] == 'I') && (commandIn[3] == 'K') &&
                                (commandIn[4] == 'E') && (commandIn[5] == 'O')) {
                                a = 0x01;
                                b = 0xFF;
                                //i = (((int)a)<<8) | b;
                                varString[1] = 'T';
                                writeVar((unsigned int)b);
                                //writeString((rom char *)retStr,7);

                                //writeString("\n", 1);
                                return;
                        }
                } else if (commandIn[1] == 'A') {
                        if ((commandIn[2] == 'D') && (commandIn[3] == 'A') &&
                                (commandIn[4] == 'M') && (commandIn[5] == 'B')) {
                                writeString("Easter Egg! ", 12);

                                return;
                        }

                } else if (commandIn[1] == 'D') {
                        if ((commandIn[2] == 'R') && (commandIn[3] == 'E') &&
                                (commandIn[4] == 'W') && (commandIn[5] == 'C')) {
                                writeString("Where is he?\n", 13);
                                printSDRecords();
                                return;
                        }
                        if ((commandIn[2] == 'B') && (commandIn[3] == 'G') &&
                                (commandIn[4] == 'A') && (commandIn[5] == 'C')) {
                                //writeString("Easter Egg! ", 12);
                                //debugAccelFlag = 0xFF;
                                //DBGACFLAG = 0xFF;
                                AccP();
                                if (DBGACFLAG == 0x00) {
                                        DBGACFLAG = 0xFF;
                                } else {
                                        DBGACFLAG = 0x00;
                                }
                                return;
                        }
                } else if(commandIn[1] == 'R') {
                        if ((commandIn[2] == 'S') && (commandIn[3] == 'T') &&
                                (commandIn[4] == 'S') && (commandIn[5] == 'D')) {

                                resetSD();
                                return;
                        }
```

```
                  } else if(commandIn[1] == 'P') {
                           if ((commandIn[2] == 'R') && (commandIn[3] == 'T') &&
                                    (commandIn[4] == 'S') && (commandIn[5] == 'D')) {

                                    printSDRecords();
                                    return;
                           }
                  }
         }

         writeString("Invalid Command\n", 16);
}


void SetVariable(char A){
         //int Value;
         int Value;

         Value = commandIn[5] - 48;
         Value += (commandIn[4] - 48)*10;
         Value += (commandIn[3] - 48)*100;
    if (A == 'A'){
                  lnrThrshRec = Value;
//                WriteEEPROM(Value, LNR_THRSH_REC_ADDR);
                  SD_setStart();
                  SD_readCurr();
                  SD_SetSample(THRESH_LINL_LOC, lnrThrshRec & 0x00FF);
                  SD_writeCurr();
                  SD_SetCurrBlock(1+blocksUsed);
                  SD_readCurr();
         }else if(A == 'B'){
                  lnrThrshWrn = Value;
//                WriteEEPROM(Value, LNR_THRSH_WRN_ADDR);
                  SD_setStart();
                  SD_readCurr();
                  SD_SetSample(THRESH_LINH_LOC, lnrThrshWrn & 0x00FF);
                  SD_writeCurr();
                  SD_SetCurrBlock(1+blocksUsed);
                  SD_readCurr();
         }else if(A == 'C'){
                  rtnThrshRec = Value;
//                WriteEEPROM(Value, RTN_THRSH_REC_ADDR);
                  SD_setStart();
                  SD_readCurr();
                  SD_SetSample(THRESH_ROTL_LOC, rtnThrshRec & 0x00FF);
                  SD_writeCurr();
                  SD_SetCurrBlock(1+blocksUsed);
                  SD_readCurr();
         }else if(A == 'D'){
                  rtnThrshWrn = Value;
//                WriteEEPROM(Value, RTN_THRSH_WRN_ADDR);
                  SD_setStart();
                  SD_readCurr();
                  SD_SetSample(THRESH_ROTH_LOC, rtnThrshWrn & 0x00FF);
                  SD_writeCurr();
                  SD_SetCurrBlock(1+blocksUsed);
                  SD_readCurr();
         }
         writeString("**",2);
}

void ReadVariable(char A){
         if (A == 'A'){
                  varString[1] = 'A';
                  writeVar((int)lnrThrshRec);
         }else if(A == 'B'){
                  varString[1] = 'B';
                  writeVar((int)lnrThrshWrn);
         }else if(A == 'C'){
                  varString[1] = 'C';
                  writeVar((int)rtnThrshRec);
         }else if(A == 'D'){
                  varString[1] = 'D';
                  writeVar((int)rtnThrshWrn);
         }
}

void WriteEEPROM(int value, char addr) {

         // convert value to a single character
         char value_c = (char)value;

         EECON1bits.EEPGD = 0;
         EECON1bits.WREN = 1;
         EEADR = addr;
         EEDATA = value_c;

         // disable interrupts
         INTCONbits.GIEH = 0;
         INTCONbits.GIEL = 0;

         EECON2 = 0x55;
         EECON2 = 0xAA;
         EECON1bits.WR = 1;

         while (!PIR2bits.EEIF) ;
         PIR2bits.EEIF = 0;

         // re-enable interrupts
         INTCONbits.GIEH = 1;
         INTCONbits.GIEL = 1;
}

int TestAccOne(void){
int j;
         for (j=0; j < 10000; j++){
                  if (j == 0){
                  PORTD = 0b00010000;
                  }else if (j == 5000){
                           PORTD = 0b00000000;
                  }
                  if (j == 2500){
                           if (PORTDbits.RD5 == 0){
                                    writeString("Accelerometer 1 Test Fails
(First)\n", 36);
                                    return 1;
```

```
                            }
                  }
                  if (j == 7500){
                            if (PORTDbits.RD5 == 1){
                                      writeString("Accelerometer 1 Test Fails
(Second)\n", 37);
                                      return 1;
                            }
                  }
         }
return 0;
}

int TestAccTwo(void){
int j;
         for (j=0; j < 10000; j++){
                  if (j == 0){
                  PORTA = 0b00010000;
                  }else if (j == 5000){
                            PORTA = 0b00000000;
                  }
                  if (j == 2500){
                            if (PORTDbits.RD6 == 0){
                                      writeString("Accelerometer 2 Test Fails
(First)\n", 36);
                                      return 1;
                            }
                  }
                  if (j == 7500){
                            if (PORTDbits.RD6 == 1){
                                      writeString("Accelerometer 2 Test Fails
(Second)\n", 37);
                                      return 1;
                            }
                  }
         }
return 0;
}

int TestAccThree(void){
int j;
         for (j=0; j < 10000; j++){
                  if (j == 0){
                  PORTA = 0b10000000;
                  }else if (j == 5000){
                            PORTA = 0b00000000;
                  }
                  if (j == 2500){
                            if (PORTCbits.RC0 == 0){
                                      writeString("Accelerometer 3 Test Fails
(First)\n", 36);
                                      return 1;
                            }
                  }
                  if (j == 7500){
                            if (PORTCbits.RC0 == 1){
                                      writeString("Accelerometer 3 Test Fails
(Second)\n", 37);
```

```
                                        return 1;
                                }
                        }
                }
return 0;
}

char getAdcChannel(int chan) {
        // get ADC value and set it to proper variable based on numAdcRead
        char channel;
        switch(chan) {
                case 0:
                        channel = ADC_CH12;             // X1
                        break;
                case 1:
                        channel = ADC_CH8; // Y1
                        break;
                case 2:
                        channel = ADC_CH10; // Z1
                        break;
                case 3:
                        channel = ADC_CH11; // X2
                        break;
                case 4:
                        channel = ADC_CH0;  // Y2
                        break;
                case 5:
                        channel = ADC_CH9;  // Z2
                        break;
                case 6:
                        channel = ADC_CH4; // X3
                        break;
                case 7:
                        channel = ADC_CH5; // Y3
                        break;
                default:
                        channel = ADC_CH1; // Z3
                        break;
        }
        return channel;
}

void setVar(unsigned int val, int num) {
        int myVal = (int)val;
        if (myVal < 367) myVal = 367;

        myVal = myVal-695;
        myVal = (int)((float)myVal/2.87719);

        switch(num) {
                case 0:
                        if (abs(X1) < abs(myVal)){
                                X1 = -myVal;
                        }
                        break;
                case 1:
                        if (abs(Y1) < abs(myVal)){
                                Y1 = myVal;
                        }
                        break;
                case 2:
                        if (abs(Z1) < abs(myVal)){
                                Z1 = myVal;
                        }
                        break;
                case 3:
                        if (abs(X2) < abs(myVal)){
                                X2 = myVal;
                        }
                        break;
                case 4:
                        if (abs(Y2) < abs(myVal)){
                                Y2 = -myVal;
                        }
                        break;
                case 5:
                        if (abs(Z2) < abs(myVal)){
                                Z2 = -myVal;
                        }
                        break;
                case 6:
                        if (abs(X3) < abs(myVal)){
                                X3 = myVal;
                        }
                        break;
                case 7:
                        if (abs(Y3) < abs(myVal)){
                                Y3 = myVal;
                        }
                        break;
                default:
                        if (abs(Z3) < abs(myVal)){
                                Z3 = myVal;
                        }
                        break;
        }
}

int getVar(int num) {
        switch(num) {
                case 0:
                        return X1;
                case 1:
                        return Y1;
                case 2:
                        return Z1;
                case 3:
                        return X2;
                case 4:
                        return Y2;
                case 5:
                        return Z2;
                case 6:
                        return X3;
                case 7:
                        return Y3;
```

```
                  default:
                           return Z3;
         }
}

void AccP(void){
         int magnitude;

         printAccel("X1", X1);
         while (sendDoneFlag == 0x00) ;
         writeString(", ", 2);
         while (sendDoneFlag == 0x00) ;
         printAccel("Y1", Y1);
         while (sendDoneFlag == 0x00) ;
         writeString(", ", 2);
         while (sendDoneFlag == 0x00) ;
         printAccel("Z1", Z1);
         while (sendDoneFlag == 0x00) ;
         writeString(", ", 2);

         // calculate magnitude
         magnitude = X1*X1 + Y1*Y1 + Z1*Z1;
         magnitude = (int)sqrt((float)magnitude);
         while(sendDoneFlag == 0x00);
         printAccel("MG", magnitude);
         while(sendDoneFlag == 0x00);
         writeString("\n",1);

         while (sendDoneFlag == 0x00) ;
         printAccel("X2", X2);
         while (sendDoneFlag == 0x00) ;
         writeString(", ", 2);
         while (sendDoneFlag == 0x00) ;
         printAccel("Y2", Y2);
         while (sendDoneFlag == 0x00) ;
         writeString(", ", 2);
         while (sendDoneFlag == 0x00) ;
         printAccel("Z2", Z2);
         while (sendDoneFlag == 0x00) ;
         writeString(", ", 2);

         magnitude = X2*X2 + Y2*Y2 + Z2*Z2;
         magnitude = (int)sqrt((float)magnitude);
         while(sendDoneFlag == 0x00);
         printAccel("MG", magnitude);
         while(sendDoneFlag == 0x00);
         writeString("\n",1);

         while (sendDoneFlag == 0x00) ;
         printAccel("X3", X3);
         while (sendDoneFlag == 0x00) ;
         writeString(", ", 2);
         while (sendDoneFlag == 0x00) ;
         printAccel("Y3", Y3);
         while (sendDoneFlag == 0x00) ;
         writeString(", ", 2);
         while (sendDoneFlag == 0x00) ;
         printAccel("Z3", Z3);
         while (sendDoneFlag == 0x00) ;
         writeString(", ", 2);

         magnitude = X3*X3 + Y3*Y3 + Z3*Z3;
         magnitude = (int)sqrt((float)magnitude);
         while(sendDoneFlag == 0x00);
         printAccel("MG", magnitude);

         while (sendDoneFlag == 0x00) ;
         writeString("\n", 1);
         while (sendDoneFlag == 0x00) ;
}

void printAccel(const rom char *str, int accel) {
         char negative;
         int value = 0;
         if (accel < 0) {
                  value = -accel;
                  negative = 1;
         } else {
                  value = accel;
                  negative = 0;
         }

         value = value%1000;
         varString[0] = str[0];
         varString[1] = str[1];
         varString[2] = ':';
         if (negative == 0) {
                  varString[3] = (char)((value/100)+48);
                  varString[4] = (char)((value%100)/10+48);
                  varString[5] = (char)((value%100)%10+48);
                  sendLength = 6;
         } else {
                  varString[3] = '-';
                  varString[4] = (char)((value/100)+48);
                  varString[5] = (char)((value%100)/10+48);
                  varString[6] = (char)((value%100)%10+48);
                  sendLength = 7;
         }
         sendVarFlag = 0xFF;
         sendIndex = 0;
         sendDoneFlag = 0x00;
         PIE1bits.TXIE = 1;
}

int abs (int i)
{
    return i < 0 ? -i : i;
}

// calculate the accelerometer data after getting max over last 0.5-1 sec
void calc(void){
         int T_X1 = X1;
         int T_Y1 = Y1;
         int T_Z1 = Z1;
         int T_X2 = X2;
         int T_Y2 = Y2;
```

```
	int T_Z2 = Z2;
	int T_X3 = X3;
	int T_Y3 = Y3;
	int T_Z3 = Z3;
	int a1Resultant = 0;
	int a2Resultant = 0;
	int a3Resultant = 0;
	int magnitude = 0;
	X1 = 0;
	X2 = 0;
	X3 = 0;
	Y1 = 0;
	Y2 = 0;
	Y3 = 0;
	Z1 = 0;
	Z2 = 0;
	Z3 = 0;
	calc_flag = 0x00;
	if (A1 == 0xFF){
		a1Resultant = (int) sqrt((float)(T_Y1*T_Y1+T_Z1*T_Z1));
	} else if (A2 == 0xFF){
		a1Resultant = (int) sqrt((float)(T_X1*T_X1+T_Z1*T_Z1));
	} else if (A3 == 0xFF){
		a1Resultant = (int) sqrt((float)(T_Y1*T_Y1+T_X1*T_X1));
	}else{
		a1Resultant = (int) sqrt((float)(T_X1*T_X1 + T_Y1*T_Y1 +
T_Z1*T_Z1));
	}
	if (A1 == 0xFF){
		a2Resultant = (int) sqrt((float)(T_Y2*T_Y2+T_Z2*T_Z2));
	} else if (A2 == 0xFF){
		a2Resultant = (int) sqrt((float)(T_X2*T_X2+T_Z2*T_Z2));
	} else if (A3 == 0xFF){
		a2Resultant = (int) sqrt((float)(T_Y2*T_Y2+T_X2*T_X2));
	}else{
		a2Resultant = (int) sqrt((float)(T_X2*T_X2 + T_Y2*T_Y2 +
T_Z2*T_Z2));
	}
	if (A1 == 0xFF){
		a3Resultant = (int) sqrt((float)(T_Y3*T_Y3+T_Z3*T_Z3));
	} else if (A2 == 0xFF){
		a3Resultant = (int) sqrt((float)(T_X3*T_X3+T_Z3*T_Z3));
	} else if (A3 == 0xFF){
		a3Resultant = (int) sqrt((float)(T_Y3*T_Y3+T_X3*T_X3));
	}else{
		a3Resultant = (int) sqrt((float)(T_X3*T_X3 + T_Y3*T_Y3 +
T_Z3*T_Z3));
	}
	if (debugAccelFlag == 0xFF) {
		AccP();
		debugAccelFlag = 0x00;
	}

	magnitude = a1Resultant;
	if (magnitude < a2Resultant) {
		magnitude = a2Resultant;
	}
```

```
         if (magnitude < a3Resultant) {
                  magnitude = a3Resultant;
         }

         if (magnitude > lnrThrshWrn) {
                  writeSDRec((unsigned int)magnitude);
                  printAccel("!!", magnitude);
                  while(sendDoneFlag = 0x00);
         }
         else if (magnitude > lnrThrshRec) {
                  writeSDRec((unsigned int)magnitude);
                  while(sendDoneFlag = 0x00);
         }
}
//This needs to be called during initialization to load all saved values from the SD
card
void initFromSD(void){
         int temp;

         SD_setStart();
         SD_readCurr();

         gameNumber = (((unsigned int)SD_GetSample(GAME_LOC_1))<<8) |
(SD_GetSample(GAME_LOC_0));
         temp = gameNumber + 1;
         SD_SetSample(GAME_LOC_1, (temp & 0xFF00) >> 8);
         SD_SetSample(GAME_LOC_0, temp & 0x00FF);
         blocksUsed = (((unsigned long)SD_GetSample(NUMBLOCK_LOC_3))<<24) |
(((unsigned long)SD_GetSample(NUMBLOCK_LOC_2)) << 16) |
                                                (((unsigned
long)SD_GetSample(NUMBLOCK_LOC_1)) << 8) | (((unsigned
long)SD_GetSample(NUMBLOCK_LOC_0)));
         rtnThrshRec = (unsigned int) SD_GetSample(THRESH_ROTL_LOC);
         rtnThrshWrn = (unsigned int) SD_GetSample(THRESH_ROTH_LOC);
         lnrThrshRec = (unsigned int) SD_GetSample(THRESH_LINL_LOC);
         lnrThrshWrn = (unsigned int) SD_GetSample(THRESH_LINH_LOC);
         batteryStatus = (((unsigned int)SD_GetSample(BATSTAT_LOC_1))<<8) |
(SD_GetSample(BATSTAT_LOC_0));
         SD_writeCurr();
         SD_SetCurrBlock(SD_GetCurrBlock() + blocksUsed);
         SD_readCurr();

}

//Resets values in initial block of the SD to default
//Format of first block is as follows (type[# bytes]):
//(game #[2]) (# of blocks utilized on card[2]) (thresholds rot low [1])
//(threshold rot high [1]) (threshold lin low[1]) threshold lin high [1])
// (batery status [2])
void resetSD(void){
         SD_setStart();
         SD_SetSample(GAME_LOC_1, 0x00);//set game # to 0
         SD_SetSample(GAME_LOC_0, 0x01);
         SD_SetSample(NUMBLOCK_LOC_3, 0x00);//set # of blocks to 0
         SD_SetSample(NUMBLOCK_LOC_2, 0x00);
         SD_SetSample(NUMBLOCK_LOC_1, 0x00);
         SD_SetSample(NUMBLOCK_LOC_0, 0x00);
         SD_SetSample(THRESH_ROTL_LOC, 0x14);//NEED DEFAULT VALUES OF THRESHOLDS HERE
(maybe)
         SD_SetSample(THRESH_ROTH_LOC, 0x46);
         SD_SetSample(THRESH_LINL_LOC, 0x14);
         SD_SetSample(THRESH_LINH_LOC, 0x46);
         SD_SetSample(BATSTAT_LOC_1, 0x07);//Baterry Status init to 1800
         SD_SetSample(BATSTAT_LOC_0, 0x08);
         SD_writeCurr();
         newBlock(SD_GetCurrBlock() + 1);
         initFromSD();
}

//loc contains the block number of the block to be used as new
void newBlock(unsigned long loc){
         unsigned long temp;

         SD_setStart();
         SD_readCurr();
         temp = (((unsigned long)SD_GetSample(NUMBLOCK_LOC_3)) << 24) |
                           (((unsigned long)SD_GetSample(NUMBLOCK_LOC_2)) << 16)|
                           (((unsigned long)SD_GetSample(NUMBLOCK_LOC_1)) << 8) |
                           SD_GetSample(NUMBLOCK_LOC_0);
         temp++;

         blocksUsed = temp;//Increment the number of blocks that are used
         SD_SetSample(NUMBLOCK_LOC_3, (temp & 0xFF000000)>>24);
         SD_SetSample(NUMBLOCK_LOC_2, (temp & 0x00FF0000)>>16);
         SD_SetSample(NUMBLOCK_LOC_1, (temp & 0x0000FF00)>>8);
         SD_SetSample(NUMBLOCK_LOC_0,temp & 0x000000FF);
         SD_writeCurr();
         SD_SetCurrBlock(loc);
         SD_SetSample(CUR_USED_LOC, 0x00); // set the number of records in this block
to 0
         SD_writeCurr();
}

//level is the value that is to be recorded
void writeSDRec( unsigned char level){
         unsigned int curLoc;
         //63 because 1 Byte is taken up with the current count of the records in this
block
         if ( SD_GetSample(0) >= (unsigned char)((SD_BLOCK_SIZE-
1)/RECORD_SIZE)){//this block is full so we need a new one
                  newBlock(SD_GetCurrBlock() + 1);
         }
         SD_readCurr();
         curLoc = (int) SD_GetSample(CUR_USED_LOC);
         SD_SetSample(CUR_USED_LOC, (curLoc & 0x00FF )+ 1);
         curLoc = curLoc * RECORD_SIZE;
         SD_SetSample(curLoc+LINROT_OFFSET, 0x00);
         SD_SetSample(curLoc+GAMENUM_OFFSET_1, (gameNumber & 0xFF00)>> 8);
         SD_SetSample(curLoc+GAMENUM_OFFSET_0, (gameNumber & 0x00FF));
         SD_SetSample(curLoc+PROCTIME_OFFSET_1, (procRunTime & 0xFF00) >> 8);
         SD_SetSample(curLoc+PROCTIME_OFFSET_0, (procRunTime & 0x00FF));
         SD_SetSample(curLoc+MAXVAL_OFFSET, level);
         SD_writeCurr();
         return;
}
```

```
void printSDRecords (void){
         unsigned int recNum;
         unsigned int recLoc;
         unsigned long blockNum;
         unsigned long startBlock;
         unsigned int tempi;
         unsigned char tempc;
         SD_setStart();
         startBlock = SD_GetCurrBlock();
         blockNum = 1;
         while (blockNum <= blocksUsed){
                  SD_SetCurrBlock(startBlock+blockNum);
                  SD_readCurr();
                  recNum = 0;
                  while (recNum < (int)SD_GetSample(CUR_USED_LOC)){
                           recLoc = recNum * RECORD_SIZE;
                           tempc = SD_GetSample(recLoc + LINROT_OFFSET);
                           if (tempc == 0x00){
                                    varString[1] = 'L';
                           } else{
                                    varString[1] = 'R';
                           }
                           tempc = SD_GetSample(recLoc + MAXVAL_OFFSET);
                           writeVar((unsigned int) tempc);
                           tempi = (((unsigned
int)SD_GetSample(recLoc+GAMENUM_OFFSET_1)) << 8) |

         SD_GetSample(recLoc+GAMENUM_OFFSET_0);
                           varString[1] = 'G';
                           writeVar(tempi);

                           tempi = (((unsigned
int)SD_GetSample(recLoc+PROCTIME_OFFSET_1)) << 8) |

         SD_GetSample(recLoc+PROCTIME_OFFSET_0);


                           varString[1] = 'T';
                           writeVar(tempi / 1000);
                           varString[1] = 'U';
                           writeVar(tempi);
                           writeString("\n",1);
                           recNum++;
                  }
                  blockNum++;
         }
         return;
}
```

microSD.c

```
#include "phi_master.h"

//internal macros
#define HEX 16
#define READ_CMD 17
#define DUMMY 0xFF
#define START_BLOCK_TOKEN 0xFE
// R1 Response Codes (from SD Card Product Manual v1.9 section 5.2.3.1)
#define R1_IN_IDLE_STATE    (1<<0)   // The card is in idle state and running
initializing process.
#define R1_ERASE_RESET      (1<<1)   // An erase sequence was cleared before executing
because of an out of erase sequence command was received.
#define R1_ILLEGAL_COMMAND  (1<<2)  // An illegal command code was detected
#define R1_COM_CRC_ERROR    (1<<3)   // The CRC check of the last command failed.
#define R1_ERASE_SEQ_ERROR  (1<<4)  // An error in the sequence of erase commands
occured.
#define R1_ADDRESS_ERROR    (1<<5)  // A misaligned address, which did not match the
block length was used in the command.
#define R1_PARAMETER        (1<<6)  // The command's argument (e.g. address, block
length) was out of the allowed range for this card.

//ports
/*#define LED PORTCbits.RC3           //GPIO
#define LED_DIR TRISCbits.TRISC3 */
#define SD_CS      LATCbits.LATC2
#define SPI_SDI LATCbits.LATC4
#define SPI_SDO LATCbits.LATC5
#define SPI_SCK LATCbits.LATC3
#define SD_CS_DIR TRISCbits.TRISC2
#define SDI_DIR TRISCbits.TRISC4
#define SDO_DIR TRISCbits.TRISC5
#define SCK_DIR TRISCbits.TRISC3
#define SD_Enable() SD_CS = 0
#define SD_Disable() SD_CS = 1
//#define SPIFastClock() SPI1CON = 0x007F

//internal types
char buffer[SD_BLOCK_SIZE];

unsigned long curr_block;
char synced;
unsigned long total_blocks;



//functions
int currBlock = 1;
void WriteSamples(char* buff)
{
  int i = 0;
  //put in write queue
  for(i = 0; i < SD_BLOCK_SIZE; i++)
  {
    buffer[i] = *buff;
    buff++;
  }
  SDWriteBlock(currBlock);
  SDReadBlock(currBlock);
  currBlock++;
  return;
}

char setup_SDSPI( void )
{
  char res = 0;
  unsigned long start_block = 0xD6D6;

  /* Set Up UART */
  //U1BRG=216;
  //U1MODE=0x8000; /* Enable, 8data, no parity, 1 stop    */
  //U1STA =0x8400; /* Enable TX                           */
  // Configure output pins
  /*LED_DIR = 0;
  LED = 0; */

  total_blocks = 1;  //this is necessary!!

  InitSPI();
  res = InitSD();
  do
  {
    while(res)
    {
      res = InitSD();
      //LED = 0;
    }
    res = SDReadBlock(start_block);
  } while(res);

  //LED = 1;
/*
  char curr = 'M';
  int i = 0;
  for(i = 0; i < SD_BLOCK_SIZE; i++)
  {
    buffer1[i] = curr;
    if(curr++ == 'Z') curr = 'A';
  } */

  /* test */
  SDWriteBlock(0xB0);
  SDReadBlock(0xB0);
  //PrintDebug(0xB0);
  //PrintBlock();

  curr_block = 1;
  synced = 0;

  return OK;
}

void SD_setStart( void )
{
  curr_block = 1;//USED TO BE 100
  return;
}

void SD_SetCurrBlock(unsigned long newblock){
         curr_block=newblock;
         return;
}

unsigned long SD_GetCurrBlock(void){
         return curr_block;
}

unsigned char SD_GetSample( int cnt )
{
  return buffer[cnt];
}

void SD_SetSample( int cnt, unsigned char sample )
{
  buffer[cnt] = sample;
  return;
}

void SD_writeCurr( void )
{
  SDWriteBlock(curr_block);
//  curr_block++;
  return;
}

void SD_readCurr( void )
{
  SDReadBlock(curr_block);
//  curr_block++;
  return;
}

char SDReadBlock(unsigned long block)
{
  char* theData;
  char read_cmd[6];
  char status = 0x0;
  unsigned int offset = 0;
  char res = 1;
/*
  if(block >= total_blocks)
  {
    //too large for small disc
    return SD_ERR;
  } */

  while(res)
  {
    res = InitSD();
  }

  //LED = 0;
  //send the read command
  block = block * SD_BLOCK_SIZE * 8; //need to be correct offset
  read_cmd[0] = READ_CMD;
  read_cmd[1] = ((block & 0xFF000000) >> 24);
  read_cmd[2] = ((block & 0x00FF0000) >> 16);
  read_cmd[3] = ((block & 0x0000FF00) >> 8 );
  read_cmd[4] = ((block & 0x000000FF)      );
  read_cmd[5] = DUMMY;
  SD_Enable();
  status = SD_WriteCommand(read_cmd);
  if(status != 0)
  {
         //writeVar((int)status);
    return SD_ERR;
  }

  //find the start of the read
  do
  {
    status = SPIRead();
  }while(status != START_BLOCK_TOKEN);

  //read the bytes
  theData = buffer;
  for(offset = 0; offset < SD_BLOCK_SIZE; offset++)
  {
    *theData = SPIRead();
    //printbyte(*theData);
    theData++;
  }
         //writeString(buffer,512);
  SD_Disable();

  //pump for eight cycles according to spec
  SPIWrite(0xFF);

  //LED = 0;
  return OK;
}

char SDWriteBlock(unsigned long block)
{
    char* theData;
          char CMD24_WRITE_SINGLE_BLOCK[] = {24,0x00,0x00,0x00,0x00,0xFF};
          unsigned int i;
          char status;
    char res = 1;

          //writeString(buffer,512);
    while(res)
    {
      res = InitSD();
    }

    block = block * SD_BLOCK_SIZE * 8; //need to be correct offset
          CMD24_WRITE_SINGLE_BLOCK[1] = ((block & 0xFF000000) >> 24);
          CMD24_WRITE_SINGLE_BLOCK[2] = ((block & 0x00FF0000) >> 16);
          CMD24_WRITE_SINGLE_BLOCK[3] = ((block & 0x0000FF00) >> 8);
          CMD24_WRITE_SINGLE_BLOCK[4] = ((block & 0x000000FF));

    SD_Enable();

          // Send the write command
          status = SD_WriteCommand(CMD24_WRITE_SINGLE_BLOCK);

          if(status != 0)
          {
        //printbyte(status);
                    // ABORT: invalid response for write single command
                    return 1;
          }

    //write data start token
          SPIWrite(0xFE);

          //write all the bytes in the block
    theData = buffer;
          for(i = 0; i < SD_BLOCK_SIZE; ++i)
          {
                    SPIWrite(*theData);
        //printbyte(*theData);
                    theData++;
          }

    for (i=SD_BLOCK_SIZE; i < 512; ++i) {
              SPIWrite(0x00);
          }


          // Write CRC bytes
          SPIWrite(0xFF);
          SPIWrite(0xFF);

    //prints(msg);

    //wait to complete
    status = SPIRead();
    while(status != 0xFF)
    {
      //prints(msg);
      status = SPIRead();
    }

    SD_Disable();

    //wait 8 clock cycles
          SPIWrite(0xFF);

    return(0);

}

//internal functions
char InitSD( void )
{
```

```
unsigned int i = 0;
char status;
char CMD0_GO_IDLE_STATE[] = {0x00,0x00,0x00,0x00,0x00,0x95};
char CMD1_SEND_OP_COND[] = {0x01,0x00,0x00,0x00,0x00,0xFF};
char CMD55_APP_CMD[] = {55,0x00,0x00,0x00,0x00,0xFF};
char ACMD41_SD_SEND_OP_COND[] = {41,0x00,0x00,0x00,0x00,0xFF};

// Turn off SD Card
SD_Disable();
//SD_PowerOff();

// Wait for power to really go down
for(i = 0; i; i++);
for(i = 0; i; i++);
for(i = 0; i; i++);
for(i = 0; i; i++);

// Turn on SD Card
//SD_PowerOn();

// Wait for power to really come up
for(status = 0; status < 10; ++status)
{
        for(i = 0; i; i++);
        for(i = 0; i; i++);
        for(i = 0; i; i++);
        for(i = 0; i; i++);
}

// We need to give SD Card about a hundred clock cycles to boot up
for(i = 0; i < 16; ++i)
{
        SPIWrite(0xFF); // write dummy data to pump clock signal line
}

SD_Enable();


// Wait for the SD Card to go into IDLE state
i = 0;
do
{
        status = SD_WriteCommand(CMD0_GO_IDLE_STATE);

        // fail and return
        if(i++ > 50)
        {
                return 1;
        }
} while( status != 0x01 );

// Wait for SD Card to initialize

i = 0;
do
{
        status = SD_WriteCommand(CMD1_SEND_OP_COND);
        if(i++ > 50)
```

```
                {
                        return 2;
                }
        } while( (status & R1_IN_IDLE_STATE) != 0 );

    // Send CMD55, required to precede all "application specific" commands
        status = SD_WriteCommand(CMD55_APP_CMD); // Do not check response here

        // Send the ACMD41 command to initialize SD Card mode (not supported by MMC
cards)
        i = 0;
        do
        {
                status = SD_WriteCommand(ACMD41_SD_SEND_OP_COND);
                // Might return 0x04 for Invalid Command if MMC card is connected

                if(i++ > 50)
                {
                        return 3;
                }
        } while( (status & R1_IN_IDLE_STATE) != 0 );

        // Set the SPI bus to full speed now that SD Card is initialized in SPI mode
        SD_Disable();
        //SPIFastClock();

        return 0;
}

void InitSPI(void)
{
        //SD_PowerOff();
        //SD_PWR_DIR = 0;   // output
        //SD_PowerOff();

        SD_Disable();
        SD_CS_DIR = 0; // output
        SD_Disable();

        SDI_DIR = 1; // input
        SCK_DIR = 0;
        SDO_DIR = 0;

        // set SPI to slowest setting
        // master mode
        // sspen enabled
        // idle state for clock is high
        // primary prescaler of 64:1
        SSPCON1 = 0x32;
        SSPSTAT = 0x00; // bus mode of 0,0
}

char SD_WriteCommand(char* cmd)
{
        unsigned int i;
        char response;
        char savedSD_CS = SD_CS;
```

```
         // SD Card Command Format
         // (from Section 5.2.1 of SanDisk SD Card Product Manual v1.9).
         // Frame 7 = 0
         // Frame 6 = 1
         // Command (6 bits)
         // Address (32 bits)
         // Frame 0 = 1

         // Set the framing bits correctly (never change)
         cmd[0] |= (1<<6);
         cmd[0] &= ~(1<<7);
         cmd[5] |= (1<<0);

         // Send the 6 byte command
         SD_Enable();
         for(i = 0; i < 6; ++i)
         {
                  SPIWrite(*cmd);
                  cmd++;
         }

         // Wait for the response
         i = 0;
         do
         {
                  response = SPIRead();

                  if(i > 60000)  //instead of 100
                  {
                           break;
                  }
                  i++;
         } while(response == 0xFF);

         SD_Disable();

         // Following any command, the SD Card needs 8 clocks to finish up its work.
         // (from SanDisk SD Card Product Manual v1.9 section 5.1.8)
         SPIWrite(0xFF);

         SD_CS = savedSD_CS;
         return(response);
}

void SPIWrite(char data) {

//       PIR1bits.SSPIF = 0;
//       SSPBUF = data;
//
//       while(!PIR1bits.SSPIF);
//       PIR1bits.SSPIF = 0;
         WriteSPI(data);
}

char SPIRead(void) {
         char TempVar;
//       if (SSPSTATbits.BF != 0){
//                TempVar = SSPBUF;
//                  return TempVar;
//         }
         TempVar = SSPBUF;        // Clear BF
         PIR1bits.SSPIF = 0;      // Clear interrupt flag
         SSPBUF = 0xFF;                  // initiate bus cycle
         while(!PIR1bits.SSPIF);  // wait until cycle complete
         return ( SSPBUF );              // return with byte read
}
```

Appendix G: FMECA Worksheet

Failure No.	Failure Mode	Possible Causes	Failure Effects	Method of Detection	Criticality	Remarks
A1	Ouput = 0V	Failure of any component in the power supply or an external short	Rest of helmet does not function	Observation	Medium	Medium criticality since concussion alerting is lost
A2	Output > 4.2V	Failure of U25 or U19 and D12	Possible damage to rest of circuitry, loss of concussion alerting	Observation	Medium	Medium criticality since concussion alerting is lost
A3	Battery Carge Voltage out of Spec	Failure of U19	Damage to battery, possibly a fire	Observation	High	High criticality since injury to the player is possible
A4	Output out of Tolerance	U23, C21, C22	Unpredictable operation, including unpredictable alerts	Observation	Low-Medium	Low-Medium criticality since the lost functionality is variable

Table G-1 – Power Supply FEMCA

Failure No.	Failure Mode	Possible Causes	Failure Effects	Method of Detection	Criticality	Remarks
B1	5V Ouput = 0V	U26, L1, D8	Accelerometers stop functioning, no alerts sent	Observation	Medium	Medium criticality since concussion alerting is lost
B2	5V Output out of Tolerance	C9, C10, L1	Store data is inaccurate and unpredictable alarms	Observation	Medium	Medium criticality since concussion alerting is lost
B3	3.3V Ouput = 0V	U28 or external short	System does not work, no alerts sent	Observation	Medium	Medium criticality since concussion alerting is lost
B4	3.3V Output > 3.3V	U28	Damage to SD card	Observation	Low	Low criticality since no critical functionality is lost
B5	3.3V Output out of Tolerance	C14, C16, L3	Some data not store	Observation	Low	Low criticality since no critical functionality is lost
B6	0.5V, 4.5V Output out of Tolerance	R1, R2, R3	Incorrect accelerometers measurements acquired	Observation	Low-Medium	Low-Medium criticality since these are used for reference voltages of the A-D converters. It is possible that the voltages be out of spec but concussion causing impacts are still measured correctly

Table G-2 – Microcontroller Power FEMCA

Failure No.	Failure Mode	Possible Causes	Failure Effects	Method of Detection	Criticality	Remarks
C1	TestAcc1-3 continuously 1	Software, short/open –circuit in microcontroller	Accelerometers, constantly in self-test mode, no acceleration data measured	Observation	Medium	Medium criticality since concussion alerting is lost
C2	SD_CS continuously 1,0	Software, SPI peripheral fail	No data saved/read from SD card	Observation	Low	Low criticality since no critical functionality is lost
C3	MP_CP1-4 continuously 1	Software, short in micro	Matchport may continuously send false alerts	Observation	Low	Low criticality since no critical functionality is lost
C4	MP_CP1-4 continuously 0	Software, open circuit in micro	Matchport will never send alerts	Observation	Medium	Medium criticality since concussion alerting is lost

Table G-3 – Microcontroller FEMCA

Failure No.	Failure Mode	Possible Causes	Failure Effects	Method of Detection	Criticality	Remarks
D1	Output = 0V	LTC3440 or external short	Matchport stops functioning	Observation	Medium	Medium criticality since concussion alerting is lost
D2	Output > 3.3V	LTC3440	Damage to Matchport	Observation	Medium	Medium criticality since concussion alerting is lost
D3	Output out of Tolerance	C14, C16, L3	Unpredictable Matchport operation	Observation	Low-Medium	Low-Medium criticality since the Matchport may or may not work

Table G-4 – Wireless Power FEMCA

Failure No.	Failure Mode	Possible Causes	Failure Effects	Method of Detection	Criticality	Remarks
E1	No wireless connection	Damage to Matchport, improperly configured	No wireless device found for connection	Observation	Medium	Medium criticality since concussion alerting is lost
E2	No data over serial connection	Damage to Matchport, improperly setup web server, software	Data changed/requested on wireless connection is not set/sent	Observation	Low	Low criticality since no critical functionality is lost

Table G-5 – Wireless FEMCA

Failure No.	Failure Mode	Possible Causes	Failure Effects	Method of Detection	Criticality	Remarks
F1	Status output continuously high	U5 failure	Software will enter infinite loop during startup	Observation	Medium	Medium criticality since concussion alerting is lost
F2	Status output continuously low	U5 failure	Accelerometer self-test will not work	Observation	Low	Low criticality since no critical functionality is lost
F3	X,Y,Z out continuously high	AccX, AccY, AccZ failure	Warnings for high geforce impacts constantly sent	Debug mode on micro	Low	Low criticality since no critical functionality is lost
F4	X, Y, Z out continuously low	AccX, AccY, AccZ failure	Warnings for high geforce impacts constantly sent	Debug mode on micro	Low	Low criticality since no critical functionality is lost
F5	X,Y,Z out continuously middle (~2.5V)	AccX, AccY, AccZ failure	No warnings for high-geforce impacts are sent	Debug mode on micro	Medium	Medium criticality since concussion alerting is lost

Table G-6 – Accelerometers FEMCA