Development of a custom data-logger for wired and wireless collection of prosthetic socket data and its uses in volume management strategies

David Lee Swartzendruber Jr.

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Mechanical Engineering

University of Washington

2013

Committee:

Joan E. Sanders, Chair

Joseph Garbini

Nathan Sniadecki

Program Authorized to Offer Degree:
Mechanical Engineering

University of Washington

Abstract

Development of a custom data-logger for wired and wireless collection of prosthetic socket data and its uses in volume management strategies

David Lee Swartzendruber Jr.

Chair of the Supervisory Committee:
Professor of Bioengineering Joan E. Sanders
Department of Bioengineering

Transtibial amputee residual limbs shrink and swell throughout the day for a variety of reasons. If an amputee does not properly regulate their limb fit due to volume changes, then ulcers, irritation and other maladies can result. Stump socks are by far the cheapest way to manage volume. However evidence suggests that improper use of socks is one of the top reasons for socket discomfort and injury. In order to determine what constitutes proper usage of socks, the current use patterns of amputees need to be understood. This requires tracking the use of the amputee sock habits as well as their physical activity. However there is currently no way to objectively observe or track the sock usage of an amputee. A low power monitoring device capable of measuring sock usage and physical activity was developed and tested on human subjects. Between two subjects, 54% of donnings and 12% of doffings of socks were successfully detected. The future of this device should include additional development of the hardware and firmware to improve battery life and antenna effectiveness. Clinical intervention through sock recommendation should be explored after a thorough understanding of how amputees use their socks is established. This device has the potential to improve the comfort and fit of a transtibial prosthetic socket and to reduce the incidence of residual limb soft tissue injury.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

ADC: Analog to digital converter

BIT-BANG: Serial communication using GPIO pins instead of a dedicated hardware communications port

CAD: Computer aided design

CS: Chip select

DB: Decibels

DOFF: Removing a prosthetic

DON: Entering a prosthetic

FFT: Fast Fourier Transform

FSR: Force sensing resistor

GPIO: General Purpose Input/Output, refers to pin on microcontroller

HF: High frequency

HZ: Hertz

IC: Integrated circuit

IDE: Integrated development environment

KHZ: Kilohertz

LED: Light emitting diode

LF: Low frequency

LSB: Least significant bit

MEMS: Micro-electro-mechanical-system

MFCL: Medicare functional classification level

MHZ: Megahertz

MISO: Master in slave out

MOSI: Master out slave in

ODK: Open development kit

PCB: Printed circuit board

RFID: Radio frequency identification

RTV: Room temperature vulcanizing

SD: Secure digital, in reference to a memory storage card

SLS: Select laser sintering

SMCLK: Sub-master clock, in reference to the MSP430

SMD: Surface mount device

SPI: Serial peripheral interface

TRANSTIBIAL AMPUTATION: Below the knee amputation that transects the tibia and fibula

UART: Universal asychronous receive/transmit

UHF: Ultra high frequency

ULP: Ultra low power

USB: Universial serial bus

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to all the people who helped him along the way. A special thanks to Joan Sanders, John Cagle, Reid Phillips, Dave Gardner, and everyone in the lab. Also, there were a number of people in other departments that gave significant assistance including Spencer Gibbs, Alanson Sample, and Aaron Parks. The author would also like to acknowledge the funding from the Center for Translation of Rehabilitation Engineering Advances and Technology (TREAT) and Center for Sensorimotor Neural Engineering (CSNE). Last, the author thanks his friends and family who supported him throughout his two years at UW.

DEDICATION

To my family, Lesley, and all those who can benefit from this work

Chapter 1

INTRODUCTION

Transtibial amputees change socks to maintain proper fit and comfort as their residual limb changes volume in the socket throughout the day [1]. Failure by the amputee to manage the fit of their residual limb in the socket can lead to tissue breakdown or other issues such as sores, edema, contact dermatitis, and verrucose hyperplasia [2]. Beyond the medical implications of tissue damage, amputees may experience mental, social and economic hardships as a result as a loss of mobility. These factors demonstrate a need for improvement of current methods of amputee socket volume management through more informed use of socks by amputees.

An amputee uses a prosthesis to restore his or her ability to ambulate. The interface between the prosthesis and limb is the socket, and this interface must provide a secure and reliable mechanical connection. Failure to maintain this connection can lead to irritation or tissue damage and potential subsequent mental, social and economic hardships resulting from loss of mobility. Whereas the socket is a fixed volume, the amputee residual limb fluctuates in shape and volume throughout the day, changing the quality of fit. To compensate for this change, amputees currently have three technologies to choose from - stump socks, elevated vacuum systems, and variable socket geometry technology. By far the most affordable technology is stump socks at around $10 - $20 per sock compared to +$1000 for the other systems. These socks are made of various materials, in various thicknesses. As the limb shrinks, the amputee applies progressively thicker socks to fill the void that is created. This requires the amputee to not only change the socks when necessary, but to *perceive* the need as well. An amputee may not notice the socket no longer fits properly due to an insensate limb, cognitive impairment or for social/cultural reasons. Another issue with socks is unknown patient compliance with their use. If an amputee develops an injury on

their residual limb, the prosthetist or physician relies upon the amputee to self-report sock and prosthesis usage, a method which has been shown by Stepien et al. to be unreliable [5]. To eliminate this problem, a low cost monitoring system is proposed.

After receiving a definitive prosthesis, amputees are given directions from their prosthetist on when to change their socks to a specified thickness [1]. It is up to the amputee to follow these directions and maintain a consistent fit throughout the day. This self-management is a problem for otherwise healthy individuals when they begin to experience discomfort and must relay their usage and activity patterns to their prosthesis. In addition, this is also a problem when the amputee is either insensate, which is common in diabetic amputees, or is cognitively impaired [3, 4]. The amputee may not be able to accurately recall how long they wear their prosthetic, how often they change their socks, or what activities induce the greatest volume change and therefore require more frequent sock changes. Without accurate information, a prosthetist is commonly left to make a judgment on how to address the amputee's complaint. Do they prescribe a new socket, which requires significant time and money from both parties? Do they add padding to the current socket? Do they recommend wearing different socks? These questions are left to the experience of the prosthetist.

In 2005, there were 623,000 persons in the United States living with major lower limb loss [6].In 2009, there were approximately 41,000 amputations preformed as described by Ziegler-Graham et al. The estimates for total number of people living with limb loss are expected to double by 2030. This increase is largely due to a growth in dysvascluar issues related to diabetes [6]. Many diabetics suffer from neuropathy, leading to insensate limbs and injury requiring amputation. This neuropathy is progressive, and not always corrected upon amputation, leading to decreased sensitivity in the residual limb. This information is relevant considering a recent survey conducted by Dillingham et al. where 24% of respondents indicated that they experienced either skin irritation or wounds as a result of their prosthetic socket and approximately 57% were dissatisfied with the comfort of their current prosthesis [7]. It can be speculated that these numbers will continue to climb with the number of dysvascular amputees on the rise.

Before an amputee can qualify to receive a prosthesis, their activity level must be assessed using the Medicare Functional Classification Level (MFCL) scheme by their Physician. MFCL has a rating system from K0—K4, commonly referred to as K-Level, where K0 is defined as someone who is bedridden and their life would not be enhanced by a use of a prosthesis and K4 is someone who puts severe stress on their prosthesis and leads an active lifestyle. K-Level is used to justify components to insurance, as an example, a K4 amputee may qualify for a rotation adapter on their pylon while a K1 amputee would not. Figure 1.1 shows the distribution of amputees by K-Level in a study conducted by Gailey et al. [8]. The chart shows 96% of the amputees represented have a prosthesis. Using the 2005 estimate, 96% equates to 598,000 amputees with the potential to benefit from the use of a sock monitoring system [6, 8].

A proposed device dubbed the "Sock Monitor" would have the capability to record physical activity and sock usage data of a lower limb amputee. The most recent system is shown in Figure 1.2. This system uses Radio Frequency Identification (RFID) to identify the socks worn by amputees. RFID tags with unique identification numbers are placed in small fabric pouches sewn onto the amputee's socks. These tags are detected as the amputee dons and doffs their prosthesis. Physical activity is recorded by an accelerometer in the data logging device as well as by a limb-socket interface force sensor. This system will eventually be able to make recommendations for proper sock usage by alerting the amputee via a smartphone application.Additional images showing the various parts of the Sock Monitor system can be found in Appendix C.

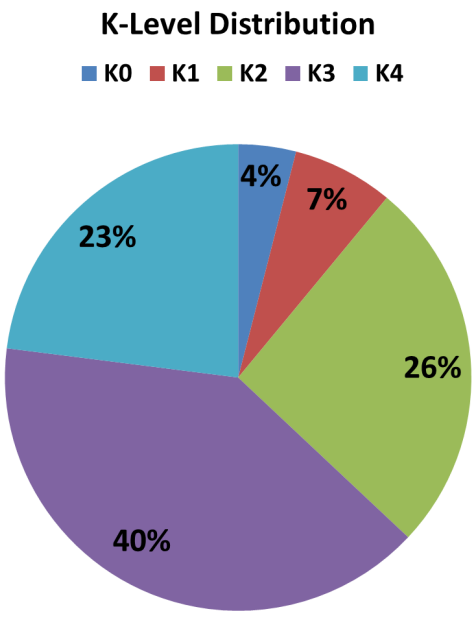


Figure 1.1: K-level distribution based on the study by Gailey et al.

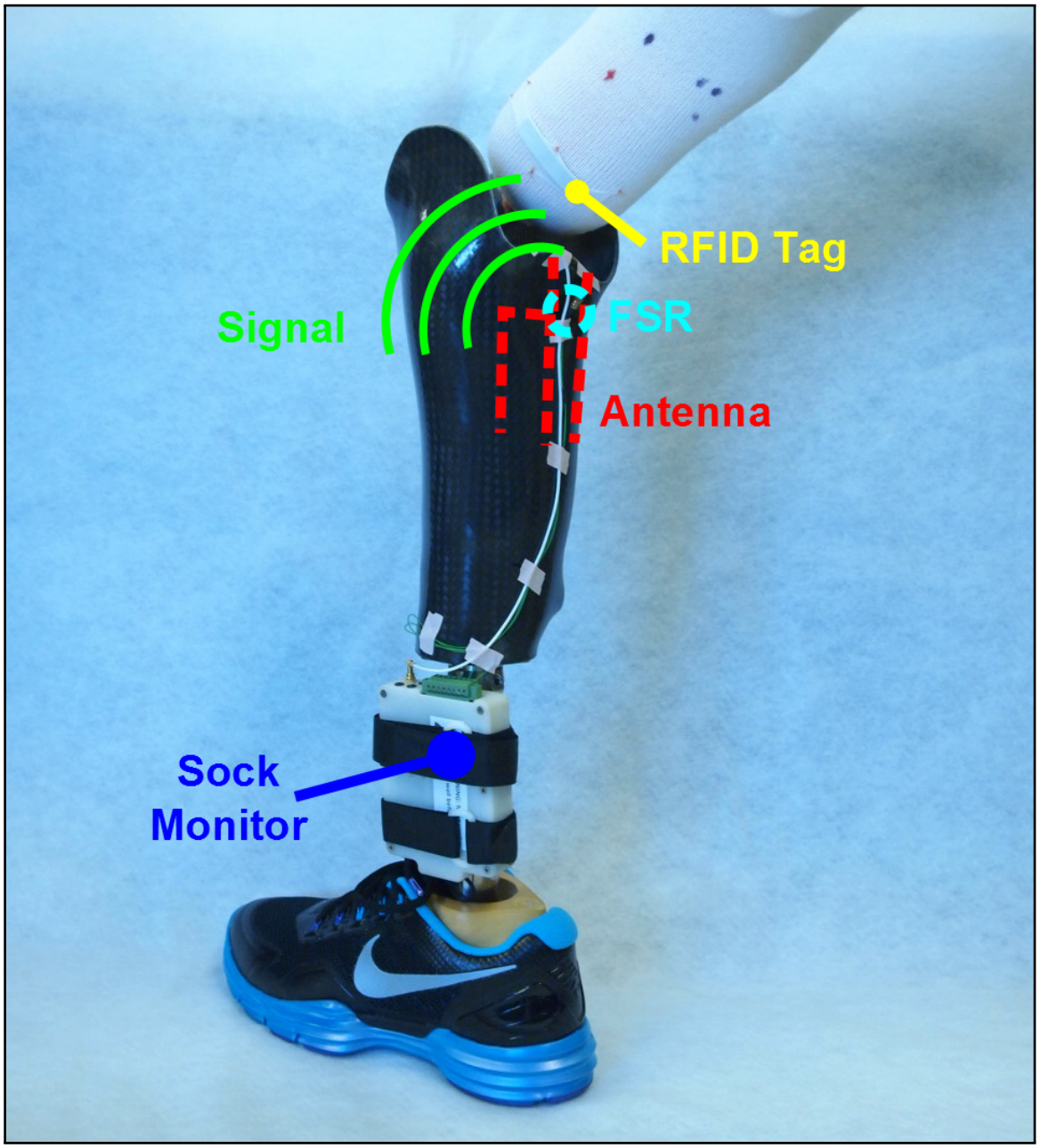


Figure 1.2: Sock Monitor system overview. The RFID tag on the sock is detected by the antenna mounted inside the socket as the limb is inserted or removed from the socket. Activity is tracked by the Force Sensing Resistor inside the socket and an accelerometer mounted inside the Sock Monitor unit that is mounted on the pylon.

Table 1.1 lists different volume management strategies as well as their strengths and limitations. Out of the possible choices, socks are the most simple and economical. Data on amputee sock usage is lacking. Anecdotal evidence suggests that amputees who have good sensation in their limbs change socks once or twice a day, however insensate amputees will go a whole day without changing socks regardless of the fit. However, there is currently no way to reliably quantify usage, necessitating the implementation of the Sock Monitor in a research study at a minimum. A study by Zachariah et al. suggests that there may be a benefit to amputee limb health and comfort if sock usage were better regulated [9]. In a study by Dudek et al. skin problems experienced by amputees that actively used a prosthesis all required several office visits to correct [10]. These visits take a large financial toll on the amputees as well as affecting their employer by requiring the amputee to take time off for treatment. By taking a proactive approach to volume management, amputees have the potential to prevent tissue damage that could escalate to a costly secondary amputation, which can range in price from $20,000 to over $90,000 [11, 12]. The information collected by the Sock Monitor would be used by prosthetists to analyze patterns of activity with the purpose of assessing health over time. Activity information coupled with sock usage would also give the prosthetist the ability to derive a better usage prescription, further enhancing amputee satisfaction. The prosthetist would benefit from amputees using the Sock Monitor by using the collected data as justification to insurance for components or changing K-level. Anecdotal evidence from various P&O websites and forums suggests that justification to insurance is a common problem experienced by all prosthetists. Overall, the Sock Monitor will help improve prosthesis fit, aid in preventing irritation and wounds, and give prosthetists a valuable tool in assessing the health of their amputees.

Table 1.1: Current volume management strategies

Method	Description	Example Products	Limitations
Socks	•Worn over residual limb to compensate for diurnal volume change •Made of various materials such as wool, cotton, nylon, and synthetics •Come in various thicknesses with a unit of measure called ply •No agreement in industry over how thick a ply actually is [13]	•Comfort, Soft Wick Stump Sock $18.16 •Knit-Rite, X-Wool Prosthetic sock $17.50 •Knit-Rite, Soft Sock $22.32	•Amputees relied upon to change socks •No reliable way to keep track of use over lifetime of sock
Elevated Vacuum Assisted Socket	•Remove air between limb and socket •Pulls tissue in total contact with socket to distribute weight •Thought to promote fluid flow in residuum and decrease volume loss	•Ottobock, Harmony $3,130.86 •Willow Wood, Limb logic •Mica Corp, Vaculink	•Limited to shorter residual limbs •Noisy •Require a sealing sleeve that is prone to developing holes •Limits knee flexion •Technically complicated
Variable Geometry Socket	•Uses gas or liquid filled pads to change socket geometry •Adapts to limb instead of adapting the limb to the socket	•Simbex SVGS [14] Not on market •Pump it up! - $4,202	•Bulky size •Air bladders can exert damaging stresses when inflated
Osseointegrated Prosthetics	•Transdermal titanium pin inserted into bone •Attaches to prosthesis •FDA has not approved the technology for now	•Not on market in US	•High risk of infection and re-amputation •Long postsurgical recovery

Chapter 2

BACKGROUND

2.1 Current Volume Management Strategies

There are a variety of volume management technologies currently available to transtibial amputees, including stump socks, elevated vacuum sockets, and inflatable bladders. These devices are intended to change the size and shape of the socket to accommodate volume change in the residual limb. These technologies are not to be confused with methods of suspension, although some in the field consider elevated vacuum sockets a method of suspension. Suspension is the way in which the prosthesis is attached to the residual limb, while volume management is how the fit is maintained. Three well-known volume management technologies are further detailed in the following sections.

2.1.1 Socks

Socks are used by amputees to serve two purposes. The first is to provide an interface between the residual limb and the socket that is washable and can absorb oils, odors, and contaminants produced by the amputee's limb [15]. The second, and most important, is to fill the voids created by limb fluid loss [15]. An amputee's limb may change in volume throughout the day for a variety of reasons. It is common for transtibial amputees to experience significant swelling overnight, then shrinkage during the day as they walk on their prosthesis [16]. When amputees experience shrinkage, they don progressively thicker socks until a comfortable fit has been attained. The comfort experienced by the amputee is directly related to the redistribution of forces on their limb.

Socks can be worn directly on the skin or over liners. There are also different length socks that are used for different purposes. One example is half-socks, which can be used in conjunction with vacuum systems. Some vacuum systems will use a special liner that

includes a seal that sits inside the socket. A half-sock can be placed over the distal end of the residual limb to account for localized shrinkage, while not compromising the integrity of the seal created by the liner. Half-socks can also be used by amputees who experience localized distal swelling and do not need the extra material on the proximal end of their residuum.

Socks are an economical way to compensate for residual limb volume changes. Typically socks can range in price from $10 - $20 for the more common socks, and around $40 - $70 for specialty socks [17]. The sock materials can include cotton, wool and synthetic fibers. These various materials are woven together into different thicknesses, referred to as ply in the industry. Typical values of ply can include but are not limited to 1, 3, and 5. Recently, a type of sock has been created with the purpose of increasing suspension for amputees who find wearing a sock under their liner comfortable. This sock is thinner than other similar socks and contains silver fibers, which protect against bacteria and odors [15]. Another new sock is the gel-infused sock. Examples of these socks include Knit-Rite's DERO® Hole-in-toe soft sock and Comfort Products' Angel Prosthetic Sock. These socks are intended to be worn under the liner on the residual limb of a transtibial amputee. The Angel Prosthetic Sock claims to provide additional moisture wicking and odor reduction through the use of special silver fibers [18]. These socks are simply another example of what is available to amputees.

2.1.2 Elevated Vacuum

Elevated vacuum systems come in two varieties, passive and active. The passive system consists of a one-way valve and a sealing mechanism, which could be a sleeve that fits over the proximal end of the prosthesis, or inside the socket with a special liner. As the amputee walks, air is driven out of the one-way valve, creating a slight vacuum inside the socket. Alternatively, an *active* vacuum system uses a mechanical pump to drive out air to a predefined pressure. Examples of an active vacuum system include WillowWood's Limb Logic system and Ottobock's Harmony® e-pulse system.

Elevated vacuum systems are thought to be beneficial because they reduce the amount of diurnal volume loss experienced by the amputee. Board et al. conducted a study evaluating the volume change, pistoning and gait change in amputees using vacuum and non-vacuum sockets. The study found the amputees wearing the vacuum sockets experienced a net volume gain over a 30 minute walk, reduced pistoning, and a more symmetrical gait [19]. In addition, 9 of 10 subjects reported feeling that the vacuum system felt more connected to their limb [19]. While vacuum systems may offer an improved fit, they can be reimbursed at upwards of $3000 - $4000 based off of the 2013 Medicare Durable Medical Equipment fee schedule (HCPCS L5781 and L5782) [20]. This cost is prohibitively expensive for many amputees who do not have sufficient health insurance.

2.1.3 Inflatable Devices

Bladders filled with either gas or liquid can be used to replace volume lost by the residual limb. These systems can either be user initiated, such as the Pump It Up!™ system, or can be automatic, such as the Active Contact System™ by Simbex. There are separate issues to consider when evaluating gas versus liquid bladder systems. Sanders et al. found that air bladder systems had a narrow dynamic range due to the compressibility of air [21]. This means that an air bladder system could not accommodate a large volume change compared to a liquid bladder system, which uses an essentially incompressible fluid.

Another issue with the bladder system involves blood occlusion due to the high pressures exerted by the bladders [21]. A final issue with inflatable devices is the cost. Similar to the vacuum systems, the inflatable devices can be in excess of $4000 [22], which as stated previously, is prohibitively expensive for many amputees with poor health insurance coverage. A confounding issue for those amputees with good coverage is the lack of reimbursement for the physical device. Reimbursement options do exist, however anecdotal evidence suggests that many clinicians will refuse to use a product that requires these alternatives due to the risk of being subjected to an audit.

Chapter 3

DEVICE DEVELOPMENT

The requirements for the Sock Monitor call for recording the amputee's sock usage habits and physical activity. The data needed to define the sock usage of an amputee requires knowledge of which socks the amputee is wearing and when. The acquisition of sock usage needs to happen automatically and without any cognitive input from the amputee. This dictated a wireless solution, which was implemented through Radio Frequency Identification.

Similarly, automated physical activity data was required to help determine if there is a correlation between physical activity and sock usage. Both interface force and limb position are important indicators of physical activity and intensity of activity. To monitor the limb-socket interface force, a sensor was needed that could work over the range of force observed at the interface and was thin enough not to interfere with the fit. The absolute pressure was not necessarily required, although it would be beneficial in certain applications. As long as the physical activity of the amputee could be extracted from the incremental change in sensor output, the sensor would satisfy its requirement. Therefore, a Force Sensing Resistor (FSR) was chosen to monitor interface forces and an accelerometer was included to monitor the amputee's limb position.

3.1 Sensors

3.1.1 Radio Frequency Identification

Radio Frequency Identification refers to a method of wireless identification of a remote system using radio waves. Generally the remote system is a very small integrated circuit (IC), commonly referred to as a tag, that passively responds with a simple identification number when queried. Although passive tags are most common, active tags are beginning to emerge [23, 24, 25]. Active tags are able to sense the world around them and then

communicate back the collected data when interrogated by a reader device. Examples of data able to be sensed by active tags include acceleration, strain, and temperature.

RFID uses radio waves at specific frequencies to communicate to a variety of tags. The frequencies are 125 kHz and 134 kHz for Low Frequency (LF), 13.56 MHz for High Frequency (HF), and 860-960 MHz and 2.4 GHz for Ultra-High Frequency (UHF) [26]. The current standard for UHF RFID is the EPCglobal Gen2 protocol, also known as ISO/IEC 19762 [24, 27]. This standard specifies the physical communication layer between the reader and tag [27].

A prior prototype of the Sock Monitor project used an HF RFID system to monitor tags [3]. This system used inductive, near-field coupling to communicate with tags mounted on the socks of amputees. There were issues with collision of these tags during the initial testing of the prototype. A collision occurs when multiple tags respond to an interrogation by a reader. There are effective protocols for mitigating the issues caused by collisions in the UHF far-field RFID systems. The current standard also provides additional security over the previous Gen1 standard. The Gen1 standard used a query tree protocol to handle collisions of UHF RFID tags [24]. When a reader using the Gen1 protocol detects a collision, it singulates a tag by sending out a bit mask defining the first bit. It then determines if it gets a unique response and repeats the process with the next bit until all tags are singulated. There are obvious eavesdropping concerns because the full powered reader antenna is broadcasting the ID of the tag it is detecting.

The same reader using the Gen2 protocol will use a query slot protocol. This protocol requires the tag to have memory that can temporarily store different states. A Gen2 tag must have a counter, inventory flag, and 16 bit random number generator. A reader will initiate an inventory by sending out an inventory request, along with a parameter Q, where Q is between 1 and 16. All tags will clear their inventory flag and the random number generator will load a 2^{Q}-1 random value into their count registers. If the tag's count value is zero, then the tag will respond back with a 16-bit random number. In turn, the reader

will reply with that same random number, after which the tag will respond with its unique ID. After the tag responds with the ID, it sets its inventory flag and goes to sleep until the next inventory cycle. A query repeat command is then issued to the remaining tags and the tags decrement their count values and the process repeats. If there is a collision, meaning that multiple tags have a count value of zero, then a query adjust command is issued with a new Q parameter and new random values are generated for their count values.

The robust query slot protocol of the Gen2 UHF RFID tags is what motivated the decision to use the CAEN R1230CB-Quark UHF RFID module in the new prototypes. In addition, the electronic interface for the module is very simple. The module only requires four lines, power, ground, data in and data out (UART). The communication protocol over UART follows a simple packet based protocol, making parsing the incoming data trivial.

One of the disadvantages to the UHF system is the need to fabricate antennas within the socket for all amputee subjects. Not only are there fabrication issues to address, but there are also design considerations as well. Certain materials, such as metals and water, can attenuate RF signals and make antenna design more complicated. Unfortunately, the prosthetics industry is using carbon fiber more and more in the construction of prosthetic sockets. While this material makes the socket stronger and lighter, it also makes constructing a usable antenna more difficult. Application specific issues also make designing an antenna for a prosthetic socket more difficult. The human body absorbs RF signals due to its high water content and the close proximity of the RFID tags on the residual limb can detune the antennas on the physical tags. These issues were addressed by other researchers working in Sanders' lab, and their work is detailed in following section of this thesis.

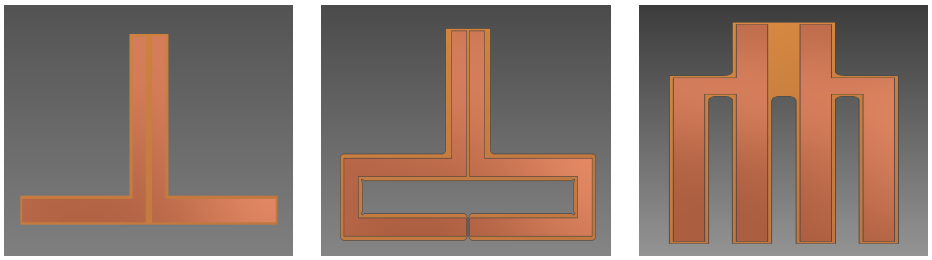
3.1.2 Antenna Design

The issues previously described with the UHF RFID antenna were addressed by preliminary research done by John Cagle and Reid Phillips, and later, David Gardner. Bench top testing was conducted to evaluate RFID tags, material effects, and antenna design. The results of the initial testing found that the Alien Technologies ALN-9629 square inlay was

able to be read from a greater distance and had preferred dimensions compared to other tags evaluated. Also, two separate antenna designs were generated, one for use on carbon fiber sockets and the other for use on polymer sockets. All antennas were made by placing copper tape onto the interior of the socket surface and soldering together the separate strips of tape. The two antenna designs can be seen in Figure 3.1a and Figure 3.1b.

Both designs provided small detection windows (i.e. the detectable area located relative to the antenna) for static tag reading. Through human subject testing, it soon became apparent that this antenna design was less than ideal. Each antenna was evaluated on a network analyzer before use to record the gain in the frequency region of 900—928 MHz. A S11 reflected power measurement was used to determine how much energy was being absorbed by the antenna. Values varied from approximately -8dB to over -30dB. Through experience, these power values had little bearing on the actual performance of the fabricated antenna. After continued subject testing, it became clear that an alternate antenna design for carbon fiber sockets was needed.

David Gardner began looking into alternate designs for antennas. With the guidance of Dr. Pavel Nikitin, a new vertical antenna was created and tested. The S11 values are, on average, around -20dB and the impedance is generally matched to 50Ω once the amputee's limb is in the socket. Figure 3.1c shows the new design used in current human subject testing.



(a) Polymer Antenna Design

(b) Original prototype antenna for carbon fiber sockets

(c) New antenna for carbon fiber sockets

Figure 3.1: Antenna designs

3.1.3 Force Sensing Resisitor

The interface between the amputee's limb and the socket is very small, and the limb can sometimes be sensitive. Therefore, to measure interface forces within the socket, a small and unobtrusive sensor must be used. To use the Sock Monitor on existing sockets, there was an additional requirement prohibiting the permanent modification of a subject's socket. Previous studies have used FSRs to unobtrusively measure interface forces with success [33]. This previous research helped contribute to the decision to use FSRs as a means of measuring interface force. Other factors contributing to the decision were the requirement to not modify the subject's socket and the lack of other sensors that could unobtrusively measure interface force inside a prosthetic socket.

Force Sensing Resistors or FSRs are devices that convert mechanical force into variable electrical resistance through piezoresistance. At rest, an FSR has a high enough resistance that it essentially appears as an open electrical circuit to the ADC. However, when a force is exerted on the sensor, the resistance decreases. This change in resistance can be used to measure force by inserting the FSR into a voltage divider and sampling at the divider.

There are currently two popular piezoresistive force sensors on the market, the Force Sensing Resistor by Interlink Electronics and the FlexiForce by Tekscan. The FSR and FlexiForce may be similar in appearance, however, they are constructed and operate in two separate ways. The FlexiForce sensor is composed of two flexible polyester substrate films with silver contacts applied to one side. These two films are used to sandwich a layer of adhesive and a proprietary "pressure sensitive" ink [28]. The Interlink Electronics FSR is constructed in a similar method. The FSR consists of two flexible substrates, with one coated in a polymer thick film that exhibits a decrease in resistance with an increase in applied force [29]. The second substrate is coated in a conductive material with an interlocking-finger design. An illustration for the FSR construction has been taken from the Interlink Electronic's FSR Integration Guide and is shown in Figure 3.2. When force is applied to the FSR, the electrodes make contact with the piezoresistive material and form a complete circuit. As the applied force on the sensor increases, the resistance decreases. The

FlexiForce works in a similar way, except that initially, the pressure sensitive ink is already in contact with the electrodes on either substrate.

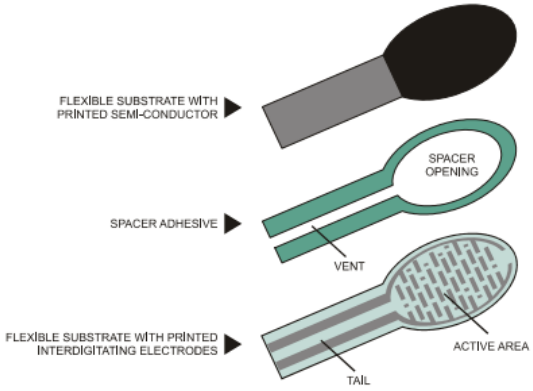


Figure 3.2: Illustration of Interlink Electronics Force Sensing Resistor [29]

Piezoresistive force sensors have issues with hysteresis, linearity, thermal drift and curvature of the load surface [30]. Ferguson-Pell et al. report that drift was 1.7 - 2.5%/logarithmic time, repeatability was 2.3 - 6.6% and linearity was 1.9 - 9.9% for the FlexiForce sensor. They also reported a significant offset error when the sensor was used on a surface with a small radius of curvature. Maalej et al. reported similar values for Interlink Electronic's FSRs. Specifically, hysteresis of 8%, repeatability was less than 7%, and temperature related drift was -0.5%/C [31]. Another issue is FSR to FSR variance. Yaniger reported that there can be as large as ±15% variance in value between different FSRs [32]. The proposed solution was to either calibrate each sensor used, or to "cherry pick" the sensor that performs closest to the desired range [32].

Given the requirements for interface force sensing, FSRs were the best option. Although it would be ideal to have the absolute value of the force exerted at the interface, it is not essential. As long as sufficient physical activity information could be captured from the incremental changes in force, the requirement of the sensor would be achieved. Also low hysteresis and dynamic response were important factors. Given the similar values between

the two devices, but more favorable dynamic response of the Interlink Electronics FSR, it was chosen as the interface force sensor for the Sock Monitor device.

3.1.4 Accelerometer

An accelerometer measures the acceleration of the object to which it is attached. In two of the three versions of the Sock Monitor, an accelerometer was chosen to monitor the limb position because the position, velocity, and acceleration can all be extracted from the data. In addition, the accelerometer chosen for the Sock Monitor is influenced by gravity, which gives a consistent frame of reference for determining the prosthesis orientation during use.

The method used to measure the acceleration of the sensor can vary from measuring the deflection of a micro-machined beam to measuring the thermal difference inside an enclosed package [34]. Godfrey et al. reported accelerometers used in the study of human motion typically rely on the spring-mass model, and include piezoelectric, piezoresistive, or differential capacitive sensing to measure deflection of the mass. Piezoelectric accelerometers utilize a mass connected to a piezoelectric material to generate a voltage differential caused by deflections of the mass during acceleration. Piezoresistive accelerometers utilize micro-machined beams and use a piezoresistive material as a very small strain gauge, this is typically connected to a Wheatstone bridge for reading the deflections. A differential capacitive accelerometer measures the capacitance of the mass between two sensing elements to quantify the value of acceleration [35].

The method of communication from the accelerometer to microprocessor can vary depending on the type of accelerometer. Pulse width modulation, analog output, and serial communication are all examples of communication methods used by commercially available accelerometers to transmit information to the target microprocessor. In a low power design, it would be ideal to have an interrupt based activation for reading the acceleration values of the sensor. This is because less power is required to service an interrupt, rather than constant polling. This feature is usually found on accelerometers with serial communication. While this method of communication may be more difficult to implement programmati-

cally, it offers faster acquisition and offloads processing from the main microprocessor. This quality was a consideration in choosing an accelerometer.

The accelerometer chosen for two of the three prototypes was the Analog Devices ADXL345. It is a 3-axes MEMS accelerometer with built in activity/inactivity, single/double tap, and free-fall detection. It has a user selectable resolution, fixed at 10 bits and up to 13 bits at the ±16g setting. The scale factor is 4mg/LSB for all ranges. This accelerometer was chosen over other designs because of the built in functions that take off the processing load from the microprocessor, as well as the selectable sensitivity.

3.2 Pilot Device

The initial pilot device was based off of a previous design by Sanders et al. that only focused on implementing an RFID tracking system for stump socks. The previous system utilized a Skyetek HF RFID system and an MSP430 microcontroller to log the data to a removable SD card [3].

The second pilot device fabricated by David Swartzendruber was made of commercially available parts. These parts included the Sparkfun Logomatic Datalogger (Sparkfun #WIG-10216), CAEN R1230CB-Quark UHF RFID module on a custom adapter board, Interlink Electronics FSR (#402), and a 2000mAh lithium polymer battery (Sparkfun #PRT-08483). These parts were housed inside of a plastic project box with the FSR and UHF antenna wires run outside the box. The FSR and antenna were mounted near the proximal posterior trimline of the socket. The components from the device can be seen in a posed setting in Figure 3.3.

The Logomatic used a simple algorithm to collect and store values at each of its 8 inputs or to automatically collect data coming into its UART port. The device could be configured by modifying a configuration file automatically created by the firmware the first time a new micro SD card is inserted into the Logomatic. The configuration file allows the user to set the collection mode (UART versus ADC), the sampling frequency of the ADC ports, the number

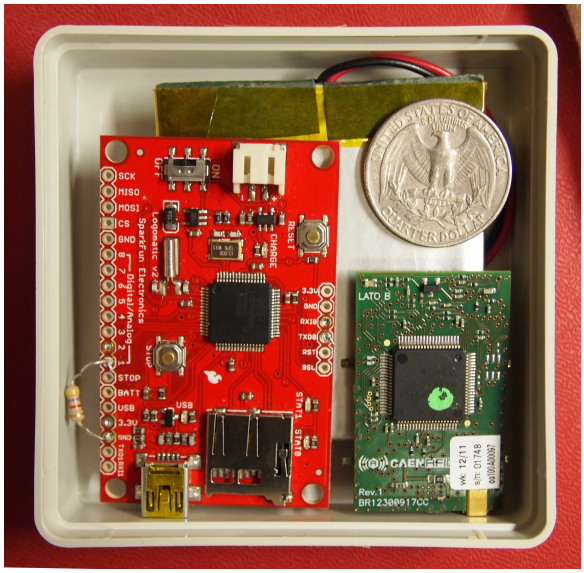


Figure 3.3: Posed pilot prototype components

of ports and format of data collected, and various aspects of the UART data collection. Unfortunately, this simple interface did not provide enough flexibility to communicate to the CAEN RFID system. The RFID system needed to be instructed to conduct an inventory on a regular schedule, meaning that multiple UART communications were necessary. This is when an alternative firmware solution was pursued. After a diligent search, an open source firmware solution was found called KWAN v1.1. This firmware was similar to the original, in that it had an auto-populated configuration file, but there were many more options that took advantage of the hardware available on the device. Some of the new features included ability to transmit a single command over UART, ability to simultaneously collect ADC and UART data, and the ability to record battery level in the output file. Again, this interface did not provide the ability to send out multiple UART transmissions and did not meet the project requirements. Consequently, alterations to the KWAN v1.1 firmware were made in an attempt to add in the desired functionality. After encountering some issues due to inexperience, the developer of the firmware, Chris Jeppesen, was contacted for assistance. He helped identify the areas that could be modified in the code to get the desired periodic polling behavior. The code was then successfully modified to send a UART transmission

located in a second configuration file once every 5 seconds. The configuration file was used to set the UART parameters and to set the sample frequency of the FSR to 25Hz. This frequency was chosen because the majority of the power observed in gait is found below 10Hz [36].

The device was then used to collect pilot data from human subjects. The details of which can be found in the *study* section of this thesis. After collecting the data, it needed to be post processed to extract the FSR and RFID tag information. This is where the modification to the firmware presented a problem. The output log file contained a predictable pattern for the timestamp and analog readings, however the tag readings were inserted into the file at random areas and were in a raw binary format. This added complexity significantly increased the amount of time to post process the data. Originally, MATLAB was used to attempt to process the data. However the computation time involved was excessive, instead, a Java program was made to process the data. This program output two data files that contained the time stamped analog and tag data that were easily read into a MATLAB analysis program.

From this device, the lessons learned were documented and were applied to the creation of a second prototype.

3.3 Second Prototype

The Sparkfun Logomatic provided a flexible platform to easily collect data. However, there were issues with it that restricted the ability to collect data long term and to conduct a small amount of on-board signal processing. Therefore a second prototype was built that incorporated these features to aid in data collection from amputee subjects. This prototype was a custom board meant to help me understand how to program, surface-mount solder, and integrate the various peripheral devices. Various line taps were added to the board to aid in using an oscilloscope and logic analyzer to troubleshoot communication between components. Due to its size and educational features, this prototype would be an education tool rather than a field-ready device.

There were several significant differences in hardware between the second prototype seen in Figure 3.4 and the Sparkfun Logomatic. Additional external communication via Bluetooth was added, the analog inputs for the FSRs were multiplexed to a single ADC, the microprocessor was changed to a MSP430F5638, a three axis accelerometer was added, 2GB of on-board flash memory was added, a transistor was added to switch the CAEN RFID unit on and off, and power was isolated via jumper pins to every component on the board. The Bluetooth module was added to the unit to allow for use in mobile phones. Eventually, the Sock Monitor was going to need to wirelessly communicate with an amputee's smartphone and Bluetooth was chosen as the most viable communication method. The analog inputs were multiplexed together and voltage dividers added to 4 of the inputs to allow more ADC pins to be open on the microprocessor. The microprocessor was changed from the Sparkfun microprocessor to reduce the power consumption of the device. An accelerometer was added to give the position of the prosthesis. The on-board flash memory eliminated the need to keep track of SD cards and to reduce write time during data collection. The transistor was added to reduce power consumption of the device due to the RFID system. Finally, the power was isolated on every device because the design was unproven and was done to prevent any overloading, shorts, or unexpected behavior.

The second prototype was used as an educational tool for learning the C programming language as well as working out any issues with the electrical design. The proprietary ExpressPCB CAD software was used to design and layout the PCB for the second prototype. During this development, other engineers at the University of Washington (UW) campus participated in hardware review meetings to verify the electrical design. After approval from the panel of engineers from the Electrical Engineering Department, the board was sent to fabrication. To conserve as much space as possible in future revisions, surface mount or SMD components were specified for the design. Therefore, a stencil was also fabricated to aid in the population of components on the PCB. After the solder paste and components were applied to the PCB, it was placed on to a hotplate. The whole board was heated to approximately 260 degrees Celsius, or until all the solder on the board melted. The hotplate was then cooled, and the remaining through-hole components populated by hand.

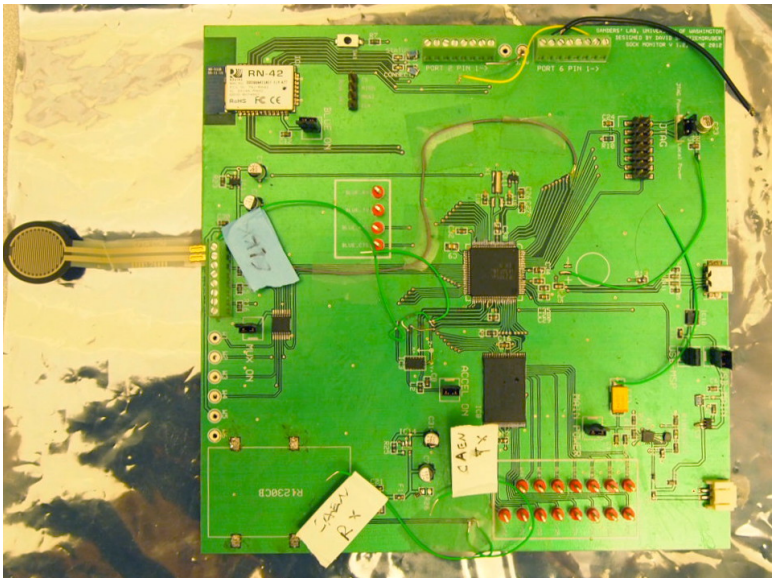


Figure 3.4: Second prototype pictured without CAEN unit installed

This prototype allowed for the gradual inclusion of the various sensors and systems on the device. The IAR Embedded Workbench Integrated Development Environment (IDE) was used to program the new device. Functionality was added to the Sock Monitor by first getting a Light Emitting Diode (LED) to blink at various frequencies. This exercise demonstrated the clock system of the microprocessor was initialized correctly. Then communication to the Bluetooth module was established, then to the accelerometer, FSR, and other systems. This prototype exposed a variety of hardware flaws that ranged from the incorrect size of resistor to wiring power to a ground pin. Each one of these issues was recorded and can be found in Appendix G. This prototype was also an introductory experience to surface mount soldering and, eventually, rework. The lessons from prototype version 1.2 were then applied to the next incarnation of the Sock Monitor.

3.4 Current Prototype

A third prototype was fabricated incorporating the lessons learned from the previous two. This prototype, also known as the Sock Monitor version 1.3, was meant to be a miniaturized version of Sock Monitor version 1.2 that could be used by human subjects. Version 1.3

incorporated additional changes including, additional power gating, backup memory, and a simple user interface.

The improvements over the previous version were mostly eliminating the exploratory power shutoffs and probe lines. However, there were some hardware changes that included the addition of two LED status indicators, one push button switch, one 16MHz crystal, one power gating IC, and the removal of a high-side switch from the RFID power rail. The addition of the LEDs and button were meant to serve as a crude user interface with the amputee as well as the researchers using the device. The 16MHz crystal was added to provide a precise high frequency clock for periods of intense computation. Also, a crystal of 4MHz or greater is required for USB operation [37]. Power gating was added to the accelerometer and FSRs to improve on the low power efficiency of the Sock Monitor design. These power rails could be disabled during periods of little to no use. The high-side switch was removed from the RFID power rail and a P-type MOSFET transistor was added.

Version 1.3 was designed to take in FSR, 3-axis accelerometer, and UHF RFID data at different sampling frequencies. The hardware supports collection of up to four FSRs and four additional analog sensors. There is also power gating to aid in the conservation of power during times when various sensors are not needed, such as when the prosthesis is not in use. Through testing of the current hardware, the power expenditure is outlined in Table 3.1. Assuming a 2000mAh battery, the unit could be expected to stay on for approximately 66 hours assuming 8.00 hours per day of no use, 15.75 hours per day of use and 0.25 hours per day of transition between donning and doffing. This time can greatly fluctuate depending on the use of the prosthesis. As of this writing the sampling frequency is 25 Hz for the FSR, 50Hz for all three axes of the accelerometer, and is variable for the RFID system. The RFID sampling algorithm is explained in detail in the firmware section of this thesis.

Table 3.1: Experimentally determined power draw as seen from the battery input

Experimental Data				
Operation	BT Connected		BT not Connected	
	Minimum (mA)	Maximum (mA)	Minimum (mA)	Maximum (mA)
Standby w/ BT & CAEN on	90	100	60	100
Standby w/ BT on & CAEN off	20	30	9	50
Standby all off except BT	20	30	8	60
Sampling (no polling)	90	100	60	110
Sampling CAEN off	20	30	9	60
Polling w/ tags detected	480	500	450	500
Polling w/o tags detected	390	400	360	420

After fabricating and testing the circuit board, several issues were discovered with the hardware. These issues ranged from a via intersecting a pad, to connecting a ground pin to power. A full listing of these errors can be found in APENDIX G. While hardware errors were one source of issues, poor electrical design was another. There were several different peripheral devices that communicated via UART, however there was only one available UART communication port. This meant that the other devices had to be bit-banged in order to communicate. Bit-banging is a term meant to describe serial communication over general input/output pins (GPIO). This is a resource intensive method of serial communication, even when driven by interrupts. Another issue is many of the peripheral devices did not have their reset pins connected to the microcontroller. This created an issue when a peripheral malfunctioned and needed to be reset. In future designs, either SPI versions of the peripherals should be substituted or the UART interface should be multiplexed. With the exception of the listed hardware inconveniences, these issues have been overcome through firmware fixes.

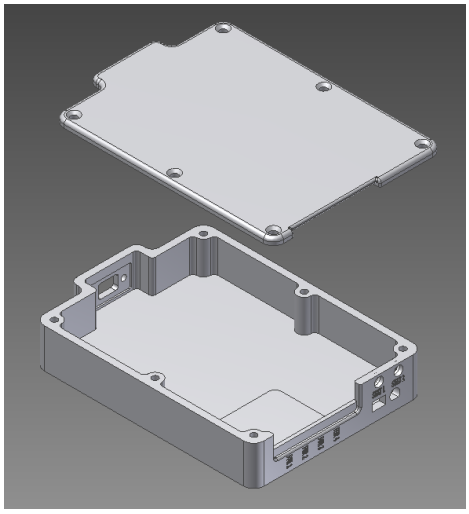
3.4.1 Enclosure Design and Fabrication

The new circuit board needed an enclosure to protect and transport all of the electronics. The new enclosure would be mounted to the pylon of the amputee's prosthesis and would have to accept up to four FSRs, a connection for an antenna, and connection for USB at a minimum. The FSR and antenna connection would be sensor inputs and the USB connection would be used for recharging, and eventually data transfer. In addition to these required connections, two LED status lights and a single push button switch would be added as a crude user interface. The purpose of the status lights would be to indicate proper functionality and error. The button was implemented to allow for future user interface expansion, such as activation of a communications system normally on standby. Given that this enclosure would be mounted on the amputee's pylon, some additional thought had to go into the design of the enclosure. Due to the proximity of the mounting location to the ground, at a minimum the enclosure would have to be water resistant. The enclosure would also have to be impact resistant due to the possibility of the amputee accidently impacting objects while walking, and be as small as possible due to the mounting location. An additional requirement was imposed on the design due to fabrication issues of the circuit board. The company used to manufacture the circuit board would not allow for large through-holes, and this meant the board could not be secured in the enclosure by mechanical fasteners. Given this large list of constraints, and time limitations for fabrication, a custom enclosure was determined as the best solution.

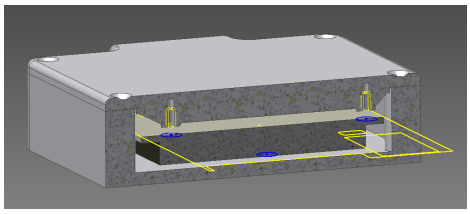
A custom enclosure was designed for fabrication using Select Laser Sintering (SLS). The SLS rapid prototyping technology was used to fabricate the enclosure for the sock monitor due to time constraints. It was decided that it would be much quicker to rapid prototype the enclosures using an outside service (Harvest Technologies), rather than obtaining bulk material and fabricating them in-house. Figure 3.5a shows a CAD rendering of the enclosure. The enclosure was made of Nylon EX, an impact resistant nylon used in SLS technology. This build material was selected due to its durability, dimensional tolerance, and surface finish. The whole electronics assembly to be enclosed was modeled in CAD and used to

design the enclosure. Because the circuit board could not be secured with mechanical fasteners, an alternative solution needed to be found. The result was to compress the board using spring plungers located in the lid of the enclosure. Figure 3.5b shows a CAD rendering of the lid pressing down on select locations on the surface of the circuit board. To maintain constant pressure on the board, sections of the interior base were raised to compensate for height differences in various components. To protect the board surface and other components, foam padding was added to the interior base of the enclosure, as well as to the areas of the circuit board where the lid would make contact. To compensate for tolerance variance in the circuit board design, threaded spring plungers with ball tips (McMaster #84805A51) were used as the force application mechanism. If too much or too little pressure was applied, the spring plunger could be retracted or extended.

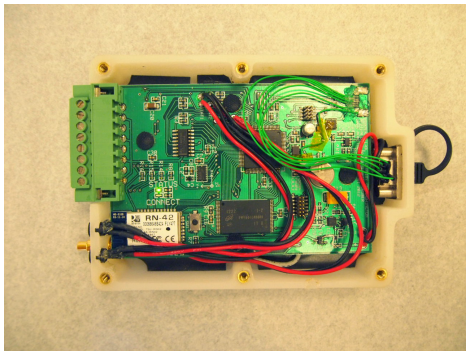
The lid of the enclosure was carefully designed for stiffness, alignment, and clearance of components. Because of the thin cross section of the lid, ribs were added to increase stiffness. These ribs also secured the bosses the held the spring plungers. A raised lip was also added to the lid to allow for an easy and repeatable alignment of the lid. Six screws were specified to secure the lid to the body of the enclosure. It was determined the lid of the enclosure would have to be repeatedly removed to access the various components for programming or service. Therefore, machine screws would be used to secure the lid to the body. This presented a problem due to the material choice of the enclosure. Repeatedly inserting and removing screws from plastic threads would quickly wear out those threads. Therefore thermal inserts (McMaster #93365A122) were inserted into the body to create a durable mechanical coupling that can be reused without fear of material failure. Care was taken to ensure that the LED, button, FSR, USB, and antenna mounts would not only fit properly, but that there would be sufficient access for tools to secure the various fittings. Figure 3.5c and Figure 3.5d show the populated completed enclosure with electronics. As final water proofing, Dow Corning RTV 3145 was added to the back of all exterior connections as both a sealant and strain relief.



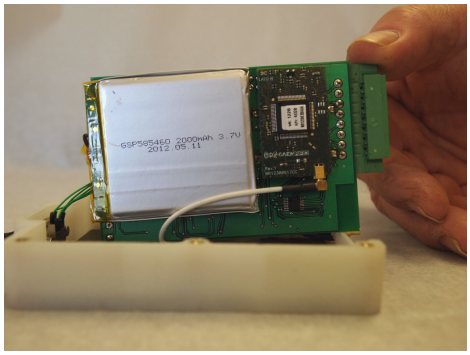
(a) CAD rendering of the lid and base of enclosure



(b) CAD rendering of cross section of lid pressing down on PCB



(c) Populated board installed in enclosure, top side



(d) Bottom side of populated board

Figure 3.5: Sock Monitor version 1.3 CAD renderings and final photos

One design feature that was added, but never fully implemented was the addition of four thermal inserts on the bottom of the enclosure. The idea was to have different mounting adapters that would be fit to different prostheses. The adapters would then be secured by four screws to the enclosure. This is still an option for future applications of the Sock Monitor.

3.4.2 Firmware Operation

The firmware refers to the programming that is installed locally on the Sock Monitor that executes commands to interface at a low level with other hardware peripherals. The firmware on the Sock Monitor uses a round-robin, interrupt driven model to schedule processing tasks. At power up, there is an initialization that occurs before entering a main loop. The main loop is designed to look for flags set during interrupts, service those flags, then return to a low power state. After an interrupt fires, the low power state is exited, and the process repeats. A high level block diagram of the firmware is seen in Figure 3.6 and a list of interrupts and their functions are seen in Table 3.2.

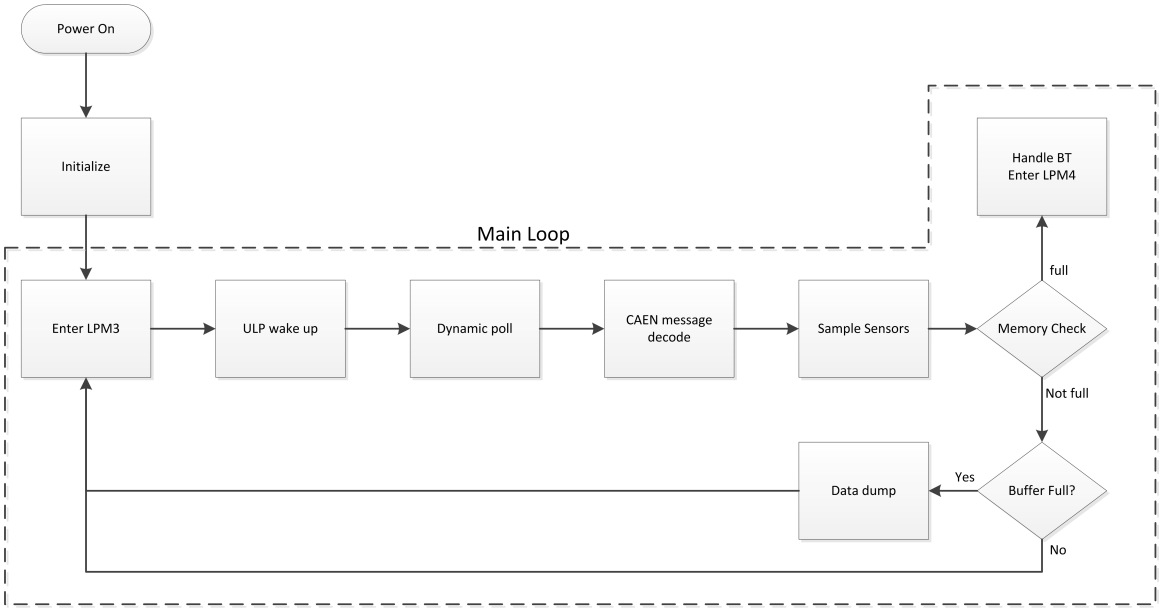


Figure 3.6: High level overview of main loop, each block is initiated by various flags

There are several different methods by which the various peripheral components on the Sock Monitor communicate. Serial Peripheral Interface (SPI), Universal Asynchronous Receive Transmit (UART), and Universal Serial Bus (USB) are the primary modes of communication between devices. In addition, the flash memory communicates through a 15 wire parallel interface. The accelerometer is the only peripheral component to connect via SPI

Table 3.2: Interrupts and descriptions

Interrupt	Description
Timer 0 - A0	Sets flag to sample data
Timer 1 - A0	Turns off STATUS 1 LED after 1/8th second of illumination due to tag read
Timer 1 - A1	Multipurpose ISR - used in dynamic polling algorithm to activate memory override and also activates ULP mode after 30 seconds
Timer 2 - A0	Error handling - handles blinking status LEDs for different errors
Timer 2 - A1	Multipurpose ISR - used in bit-bang operations with CAEN module
Timer 0 - B1	Multipurpose ISR - handles bit-bang operations
USCI - A1	Used in UART communication with Bluetooth module, determines input and sets appropriate flag
RTC	Multipurpose ISR - sets date/time variables, sets datestamp alarm flag
Port 2	Multipurpose ISR - handles button press
Port 4	Multipurpose ISR - handles bit-bang from CAEN module, handles the setting or clearing of the low battery flag, handles interrupt from accelerometer

to the microprocessor. This 4-wire interface provides a clock, Master In Slave Out (MISO), Master Out Slave In (MOSI), and Chip Select (CS) signals. The SPI clock is based off of the sub-system master clock (SMCLK) and is divided by 32 to operate at 500 kHz. This allows for up to 62,500 bytes per second to be transmitted in either direction. The UART protocol is used by two devices, the CAEN RFID module and the Roving Networks RN-42 Bluetooth (BT) module. The BT module is connected to the hardware UART port on the microprocessor, leaving no additional communication ports. Therefore, the CAEN UART connection is connected to two General Purpose Input/Output (GPIO) pins and any signal received or transmitted is bit-banged. These are all examples of serial communication, or stated differently, information transmitted a single bit at a time. The flash memory uses a parallel communication system where an entire byte is loaded onto 8 separate wires before an asynchronous strobe signal is sent to the flash Integrated Circuit (IC). In addition to the 8 data lines, there are an additional 7 control lines that latch various functions on the flash device. A large driver was developed to handle the communication between the microprocessor and the memory. Due to the use of a strobe line, writing and reading the memory

does not require the use of an interrupt. Instead any communication is handled during idle time on the microprocessor.

When a recording is initiated, a timer is started that initiates an interrupt at 50Hz. The interrupt sets a flag serviced by the data sample block in Figure 3.6 above. When the timer interrupt sets the flag to sample the data, the microprocessor exits low power mode and starts through the main loop. Figure 3.7 shows the block diagram for the sampling process. As the program progresses through the sampling function, data is written to a buffer in the output format. There are also several calculations made in this function, including limb presence via FSR value, movement via accelerometer value, and length of data in a line.

As sensors are sampled, the data is automatically formatted into the output format and stored in a buffer. As the program progresses through the main loop, the size of the buffer is checked, and if it is over a threshold size, the buffer is transferred to flash memory. All data is written to the external flash memory IC in a continuous incremental fashion, meaning that the first byte of data is written to the first position in memory and the next byte in the next position and so on. There is currently no error checking, wear leveling, or look-up tables associated with the storage of data. This is one issue that must be addressed to insure reliable and complete data retention in future firmware upgrades.

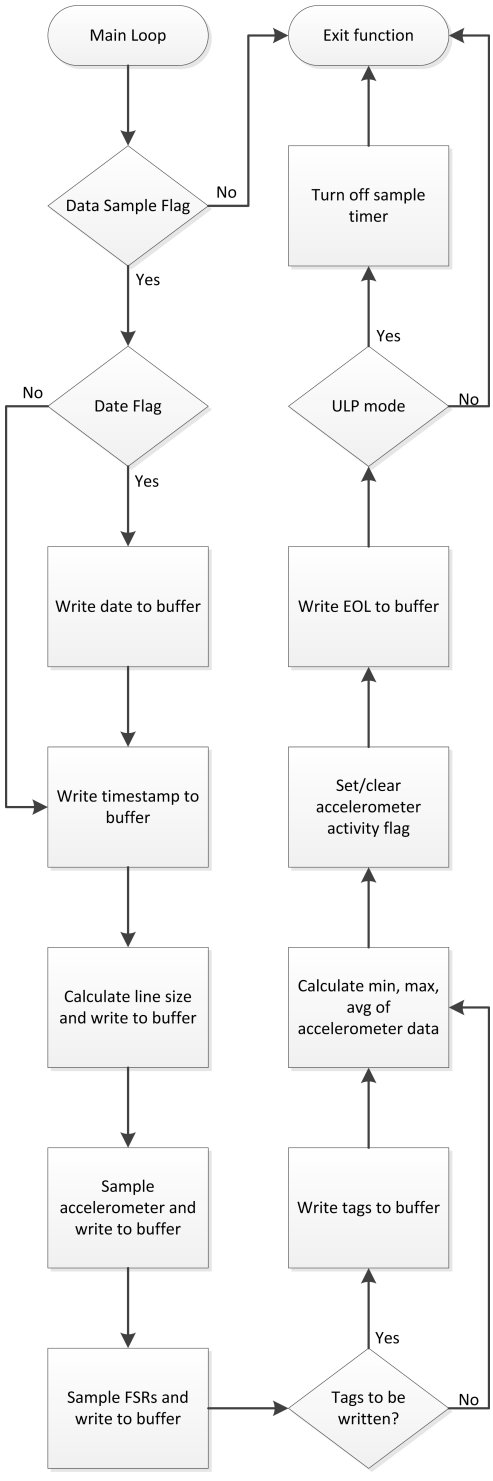


Figure 3.7: Sampling block diagram

The Sock Monitor is a mobile device meant to monitor the activities of actual amputees. Therefore the device must be independent of any external power and rely on a battery to supply energy. A complex algorithm has been developed that combines the amputee's current physical activity with power gating in the hardware to conserve as much power as possible. The algorithm is referred to as dynamic polling. The CAEN module is the largest power draw on the device, therefore restricting when it is on is very important. The only important times to read tags are during donning and doffing of the prosthesis, in order to determine what socks the amputee is currently wearing. Any time other than donning and doffing is superfluous and a waste of energy. Using the FSR and accelerometer, it is trivial to determine if the amputee is wearing their prosthesis and if they are in the process of handling/using their prosthesis.

Two separate flags are set or cleared inside the sensor sampling function. These flags are global values that are then used by the dynamic polling algorithm. When the main function enters the dynamic polling function, there are three separate states the device can be in. The first state is the amputee is wearing their prosthesis, requiring no polling. The second state is the amputee is not wearing their prosthesis but is actively handling their prosthesis, which requires polling.

The last state is the amputee is not wearing their prosthesis and is not actively handling their prosthesis, requiring no data collection. The whole process is shown graphically in Figure 3.8. When the amputee is wearing their prosthesis, the FSR will be above a threshold value that was determined experimentally to be above the static pressure exerted by a Plastazote insert, used by some amputees. This will cause the microprocessor to shut off the power to the CAEN module, turn off the tag-to-memory timer, turn off the Ultra Low Power (ULP) timer, and clear the current ULP timer value. The ULP mode flow chart can be seen in Figure 3.9. When the amputee is handling their doffed prosthesis, the CAEN module is powered on and when a tag is detected, the corresponding ID is immediately sent to the microprocessor. The tag-to-memory timer initiates a dump of unique tag IDs every two seconds during this period. If, during this period of doffed handling, the prosthesis becomes still, the ULP timer is started. At the expiration of the ULP timer after 30 seconds, the device will shut down the sampling timer, peripherals, bit-bang receive function, and configure the accelerometer to send an interrupt when activity is detected. ULP mode will be exited when the accelerometer sends an interrupt detecting activity.

The Sock Monitor collects and stores tags detected by the RFID system by storing only unique tags over a particular time window. It accomplishes this by parsing out the ID from the packet sent by the CAEN module. The firmware then compares that tag to all tags currently in the tag buffer. If the tag matches any of the tags currently in the buffer, it is discarded, otherwise it is added to the buffer. The tag buffer is written to memory after the tag-to-memory timer expires or if there is a doffed to donned transition. When the timer expires, it sets a flag that is serviced during the sensor sample function where the tags are written to the output buffer and the pointer to the last tag is reset to the first position in the tag buffer. There are two restrictions on this tag collection process. The first is only 96-bit (or 12 byte) tags can be stored. The second restriction is there can only be a maximum of 10 tag IDs read during any accumulation window. The typical accumulation window is currently set to 2 seconds.

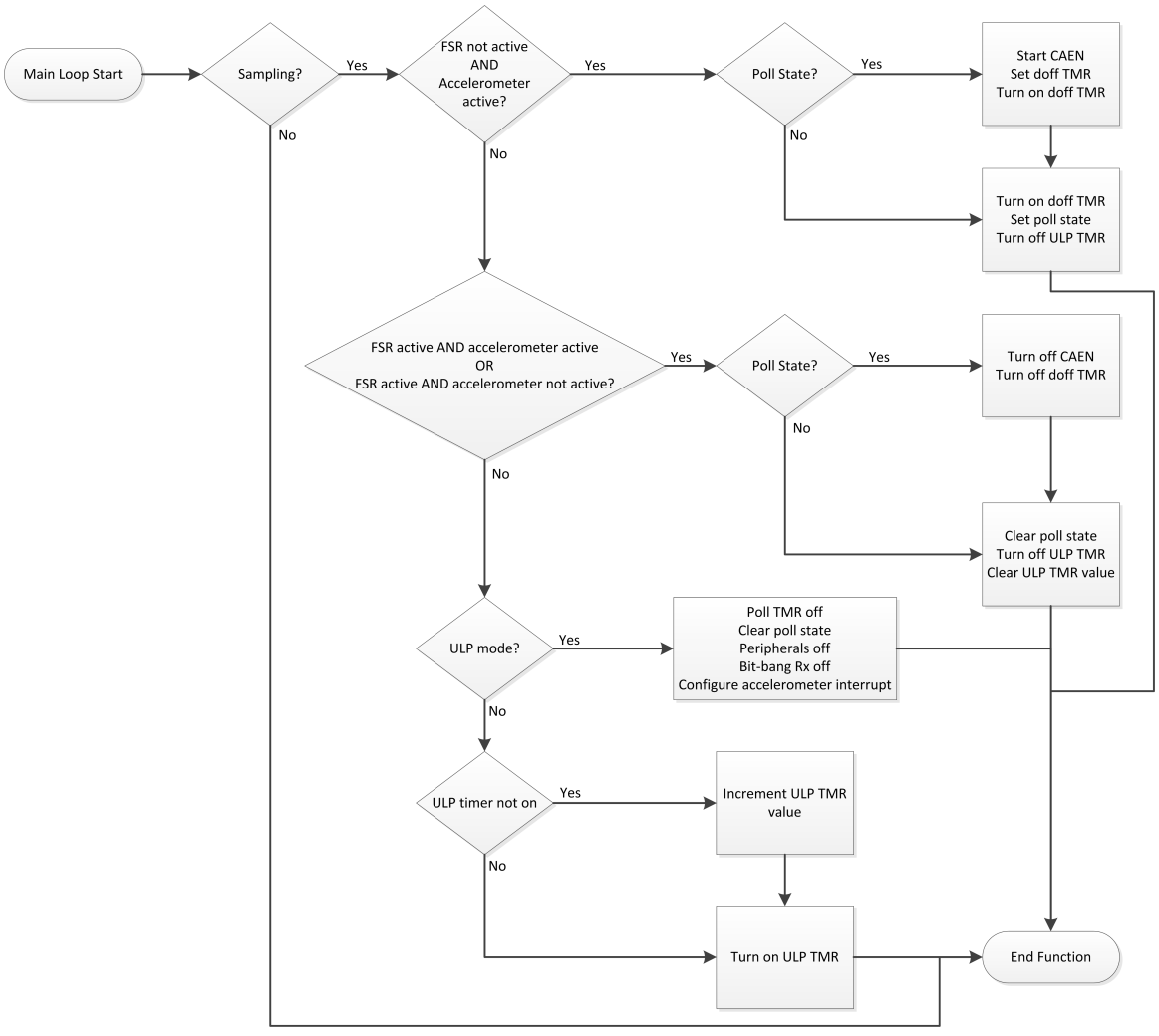


Figure 3.8: Dynamic polling flow chart

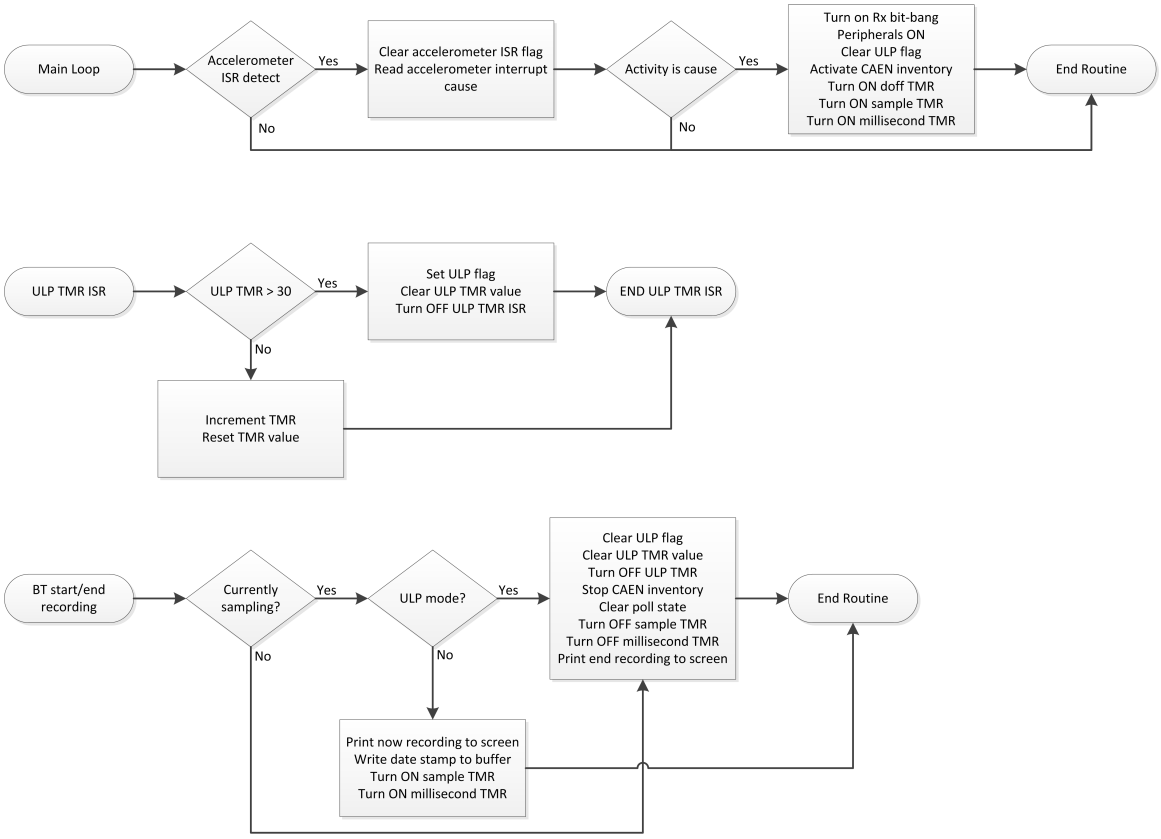


Figure 3.9: ULP mode activation and termination

Chapter 4

STUDY

4.1 Method

The various prototypes have all undergone standardized in-lab testing, as well as an unsupervised out-of-lab test. The in-lab test is meant to isolate particular physical activities carried out by amputees on a daily basis. These activities include:

- •Donning and doffing
- •Walking on a treadmill
- •Sitting
- •Standing
- •Resting the leg in an elevated position
- •Ambulating on stairs
- •Weight shifting
- •Running on a treadmill
- •Jumping

All activities were subject to the subject's health and personal discretion. These activities were proceeded and followed by periods of either standing or sitting that helped isolate the activity during post processing. In addition, the TASER video system was used to visually record the in-lab tests. The out-of-lab test was simply attaching the Sock Monitor to the amputee subject and having them go about their daily business for 24 hours to 1 week depending on the prototype and battery life. A detailed, step by step procedure is given below for the in-lab procedure with the version 1.3 Sock Monitor.

1. Install and Initialize SM (Antenna already connected)
   (a) Install FSR into socket, FSR should be as far in as possible without compromising the metal tabs with the bend radius around posterior trimline
   (b) Use single Velcro strap to secure SM to pylon
   (c) Connect FSR to FSR port 1
   (d) Strain relieve wires running from inside socket to SM with tape
   (e) Wrap SM with sports tape
   (f) Initialization complete, proceed to in-lab or out of lab test

2. Donning/Doffing

   (a) Have subject sit in chair
   (b) Subject doffs their prosthesis
   (c) Subject dons a single RFID enabled sock
   (d) Subject dons prosthesis, using whatever means necessary
   (e) Subject sits still for approximately 5 seconds
   (f) Subject doffs prosthesis
   (g) Subject sits still with prosthesis lying undisturbed for approximately 5 seconds
   (h) Repeat d g two more times, adding one additional RFID sock on the third don/doff cycle
   (i) Test ends

3. Walking

   (a) Subject dons at least 1 RFID enabled sock, but no more than 10 RFID tags at one time
   (b) Subject dons socket and walks onto treadmill
   (c) Subject stands still on treadmill in a comfortable position for approximately 5 seconds
   (d) Subject begins walking on treadmill for a period of approximately 60 seconds
   (e) Subject stands still on treadmill after a complete stop for approximately 5 seconds
   (f) Test ends

4. Sitting/Elevated leg rest/Standing

   (a) Subject sits in chair with both feet flat on ground for approximately 10 seconds
   (b) Subject raises monitored prosthesis and rests heel on chair directly across from them holding still for approximately 10 seconds.
   (c) Subject stands in front of chair with both feet on ground with equal weight distribution for approximately 10 seconds
   (d) Repeat a - c 2 more times
   (e) Test ends

5. Weight shifting

   (a) Subject stands still with both feet on ground with equal weight distribution for approximately 5 seconds
   (b) Subject oscillates at a comfortable frequency by shifting weight laterally from one leg to the other for an interval of approximately 10 seconds
   (c) Test ends.

6. Up/Down stairs (if applicable)

(a) Subject walks to top of stairs out of lab
(b) Subject stands at top of stairs for approximately 5 seconds
(c) Subject walks down to first landing, turns around to face up stairs and stands still for approximately 5 seconds
(d) Subject walks up stairs
(e) Subject reaches top of stairs and stands still for approximately 5 seconds
(f) Test ends

7. Running (if applicable)

(a) Subject walks onto treadmill
(b) Subject stands still on treadmill in a comfortable position for approximately 5 seconds
(c) Subject begins jogging/running on treadmill for a period of approximately 60 seconds
(d) Subject stands still on treadmill after a complete stop for approximately 5 seconds
(e) Test ends

8. Jumping

(a) Subject stands still for approximately 5 seconds
(b) Subject jumps at comfortable height 5 times
(c) Subject stands still for approximately 5 seconds
(d) Test ends

9. Send subject home for overnight testing (if applicable)

(a) Add 2 security stickers total to SM over mid screws and edge on each side
(b) Give subject charge wall-wart and cord, tell them to charge overnight every night until they return

10. Remove and disable Sock Monitor

(a) Ask subject to doff prosthesis
(b) Ask subject to take their prosthesis away (and indicate exactly where) to remove the SM
(c) Connect to SM and disable recording (see initial setup above for instructions)
(d) Unwrap SM
(e) Unplug FSR and Antenna wires
(f) Remove FSR
(g) Unsolder the antenna and peel out antenna
(h) Remove SM
(i) Return prosthesis to subject

4.2 Data Analysis

A custom algorithm was developed to evaluate the effectiveness of the sock detection for both prototypes of the Sock Monitor. The algorithm first applies a median filter to the FSR data to reduce small perturbations in the recorded signal. The algorithm then looks for all data below a threshold corresponding to the same threshold of the dynamic polling algorithm on the Sock Monitor version 1.3. Strings of continuous data are identified, and the beginning and ending positions of those strings are used to define the donnings and doffings of the amputee. Any doff and don that were located within two seconds of one another are eliminated to reduce the amount of false donning and doffing detected. The two second threshold was considered appropriate because removing the prosthesis, a sock, and then donning the prosthesis take more than two seconds. After each donning and doffing is identified, a window is applied to the donnings and doffings separately. The window size varies according to the prototype used and whether the amputee is donning or doffing. There is a flat ± 5 second window around each donning and doffing for the initial Sock Monitor prototype. However, in version 1.3 donnings have a window of -2/+0.5 seconds and doffings have a window of -0.5/+2 seconds. These windows were chosen due to the sample rate of polling for the initial prototype, and due to the dynamic polling algorithm in version 1.3. The dynamic polling algorithm records tags every two seconds while the prosthesis is moving and doffed, but immediately records tags when the prosthesis is donned. Also, the algorithm starts looking for tags immediately after a doffing and records the tags observed two seconds afterwards.

The pilot data was collected using different techniques than the Sock Monitor version 1.3 and consequently was evaluated using a different method as well. Two different measures were used in the evaluation of the effectiveness of the sock detection. First is the percentage of periods of activity where there is at least one sock detected. A period of activity is defined as a sustained FSR count value above 50 and is greater than 2 seconds between donning and doffing. This measure is to determine if the static detection can be used to detect socks between donnings and doffings. The next measure is false tag detections. This measure

finds all other non-zero tag detections outside of periods of activity. This information is useful in evaluating the habits of amputees, such as determining if the amputee stuffs their socks inside their empty socket.

In the evaluation of the data from the Sock Monitor version 1.3, donnings and doffings were automatically detected using a custom algorithm in MATLAB. Due to the dynamic polling algorithm, donnings and doffings were evaluated both separately and together as a whole. The measure of don, doff, and total successful detections were determined by evaluating each don/doff window to determine if there was at least one tag detected. If at least one tag was detected in the window, a counter would be incremented. The don and doff counters were then divided by the total number of dons or doffs to find the percent successful detections. Also, because the data was time stamped with absolute time, the don/doff patterns were evaluated on a daily basis as well. The daily number of dons and doffs was determined and an average value and standard deviation was recorded.

The FSR data collected by both prototypes is comparable in its form. However one notable difference is in the magnitude of ADC counts. The original prototype used a 10-bit ADC at 3.3V while version 1.3 used a 12-bit ADC at 3.0V. A qualitative evaluation of the waveforms for various modes of ambulation will be presented, and further future quantitative analysis will be discussed in the discussion section.

4.3 Subject Inclusion/Exclusion

All tests were conducted with the consent of the subject and under the supervision of a certified prosthetist. Subjects were included if they were at least 18 months post amputation, were a K-2 Medicare Functional Classification Level or higher ambulator and if their socket fit was deemed acceptable for regular use by a research prosthetist. Subjects were excluded if they had current skin breakdown.

Chapter 5

RESULTS

In the initial collection of pilot data, there were a total of 16 subjects that tested the Sock Monitor. 12 of these 16 wore the device with both sock data and FSR data collecting while the other 4 collected only FSR data. After reviewing the data from the 12 subjects recording data from both sensors, it was found that two had data containing corrupted information. Specifically one subject's data was recording without issue for about half the collection period. Then, it appears the FSR failed and the data dropped to near zero. The other subject seemed to have an issue from the beginning with the FSR that significantly skewed the magnitude of the data. Because the integrity of these two data sets was in question, they were discarded from analysis, leaving 10 of 16 subjects with both FSR and RFID data able to be evaluated.

The Sock Monitor version 1.3 was used to collect data on a total of 8 subjects. However, only two of these subjects were using the new stable firmware/hardware. Consequently, only the data sets from these two subjects were used to evaluate the effectiveness of the hardware in the Sock Monitor version 1.3. The other subject datasets were either in a different format, used a different algorithm, or were corrupted.

5.1 Subject Demographics

Table 5.1: Subject Demographics

		Age	BMI	Height (cm)	Weight (Kg)	K Level	Etiology	Gender
Pilot	Average	50.9	29.6	175.1	89.2	3.1	8 Trauma	8 Male
	Stdev	13.5	6.5	11.4	18.2	0.7	2 Disease	2 Female
Current	Average	40.0	27.0	162.5	71.8	3.5	2 Trauma	1 Male
	Stdev	14.1	6.1	3.5	19.2	0.7	0 Disease	1 Female

5.2 Sock Detection

Discussed in the following sections are results from both the original prototype and the current prototype. When reviewing these results, note the way in which RFID data was collected varies dramatically between the two devices. The pilot RFID data was collected at 0.2Hz and only a single inventory command was issued. The current system collects RFID data continuously during donning/doffing and sends the tag ID to the microcontroller as soon as it is detected.

5.2.1 Pilot Data

The pilot sock data is summarized in Table 5.2 below. There is much variability in the detection of tags during periods of activity. The average tag detection during periods of activity is 48% $\pm$ 35%. The average false positive detections, when excluding the two largest outlier values, was 1 $\pm$ 2. The excessive number of false tag readings in subject 2 can be explained by reviewing the raw data, shown in Figure 5.1. There appears to be an extended doff period that lasts about 6.5 hours where the subject is suspected to be sleeping. Subject 4 also had a large number of false positive detections which appear in periods of frequent donning and doffing in the data.

Table 5.2: Pilot Data Sock Detection Results

Subject	Test #	In/Out/Both	Tag Detection During Activity	False Positive Detections
1	1	O	0%	0
2	1	O	41%	2204
3	1	O	82%	0
4	1	O	20%	100
5	1	O	0%	0
6	1	I	100%	1
	2	O	50%	0
7	1	O	10%	0
8	1	I	60%	0
9	1	I	100%	5
10	1	O	50%	0
	2	B	57%	0

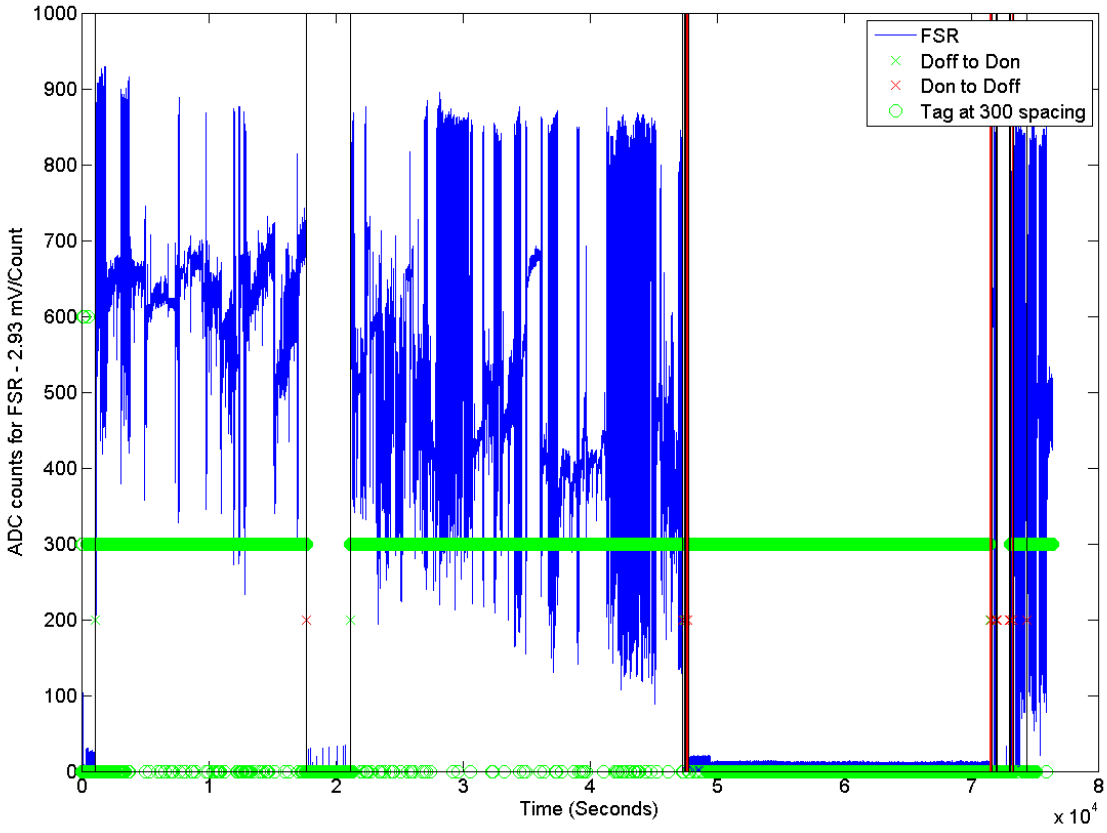


Figure 5.1: Subject 2 test summary, note the tag readings during the extended doff

5.2.2 Current Prototype

A summary of the successful detection of tags is shown in Table 5.3 below. During the 6th day of subject 3's test, there appeared to be some technical issues with the recording of tag data. When reviewing the raw data, this can clearly be seen. All other days are very consistent in picking up tags during donning and not very consistent in doffing. But on the 6h day, all tag detection stops. It is at this time the subject called the lab to report the device was no longer indicating when the subject donned and doffed their prosthesis. After trouble shooting, it was determined the connector for the coaxial cable coming from the antenna had become disconnected from the exterior antenna connector on the Sock Monitor version 1.3. This connection was fixed and is indicated by tag collection resuming.

Subject 3 collected data from only outside the lab, while subject 11 first collected outside the lab data, then upon return to the lab, completed an in-lab test protocol. Both subjects utilized the ALN-9629 UHF RFID tags, but they also were the first to test the Fujitsu silicone WT-A522 UHF RFID tag. The Fujitsu tag not only held up better over time in the socket, but was more easily detected by the RFID system. For subject 3 who wore 2 ALN tags and 1 Fujitsu tag during the test, the Fujitsu tag was detected 156 times while the ALN tags were detected 57 and 0 times. Subject 11 wore only the Fujitsu tag and it was detected 85 times. Both subjects show an additional ALN tag ending in 0x1804, this is a tag used as a standard to evaluate whether or not the system is working. At no time was this tag worn by the amputee.

Table 5.3: Current Prototype Sock Detection Results

Subject	Don Success	Doff Success	Total Success	Length of Wear
3	43%	5%	24%	6Days,1Hr,42Min
11	65%	18%	41%	1Day,23Hrs,3Min
Average	54%	12%	33%	
Stdev	16%	9%	12%	

5.3 Activity

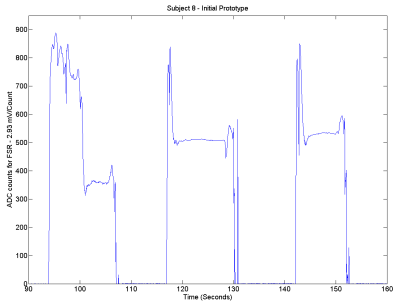
The FSR data collected from the in-lab and out-of-lab protocols was very similar and does not warrant different sections. The FSR data across all subjects showed very clear qualitative patterns for donning/doffing, walking, running, sitting, standing, and elevating the prosthesis. The patterns seen for walking up versus down stairs, as well as weight shifting were more difficult to discern.

During the out-of-lab test in both devices, the FSR data qualitatively reflected what was observed during the in-lab tests. Patterns for sitting, standing, walking, etc. were clearly observed. Three additional observations were made regarding activity outside the lab. First, the amputees seemed to be standing with some random weight shifting for various sections of time. Second, there seemed to be a small logarithmic rise in FSR data while the amputee was standing or sitting. The rise would be interrupted with some sort of movement or sudden shift in the data. The third observation was the appearance of what looked to be a pulse during periods of sitting, and in some cases standing.

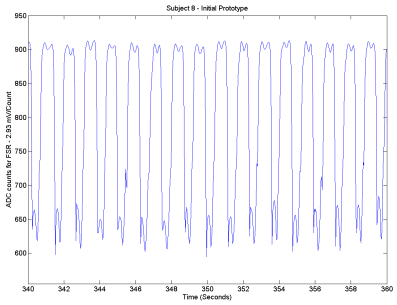
Using the custom MATLAB analysis software, the daily donning and doffing frequency of an amputee can be recorded. These results are shown for subjects 3 and 11 in Figure 5.3 and Figure 5.4. When computing the average number of dons and doffs per day for subject 3, the last day was omitted due to the excessive number of donning and doffing due to trouble shooting the Sock Monitor device. When the last day of collection is factored in, both donning and doffing go to an average of 9 ± 7 per day.

Table 5.4: Dons and Doffs per day, *indicates day dropped in computation of value

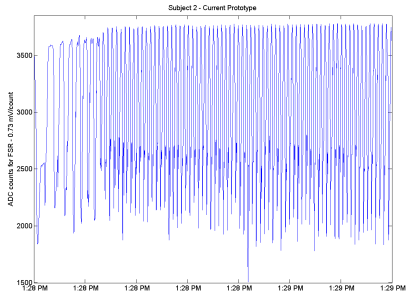
Subject	Average Dons per Day	Average Doffs per Day
3	6 ± 1	6 ± 2
11	6 ± 3*	6 ± 3*



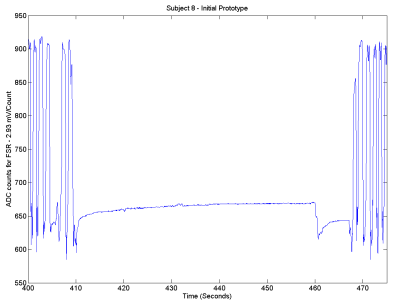
(a) Don and Doff



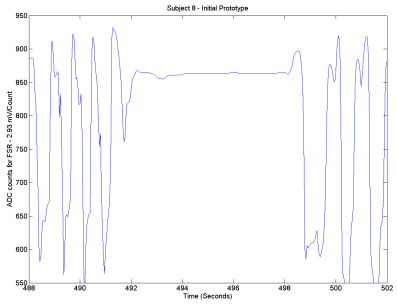
(b) Walking



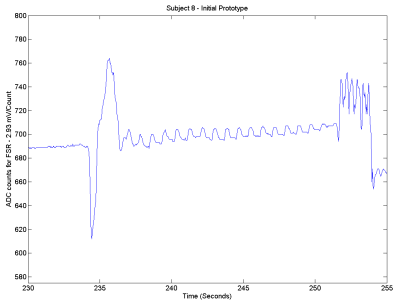
(c) Running



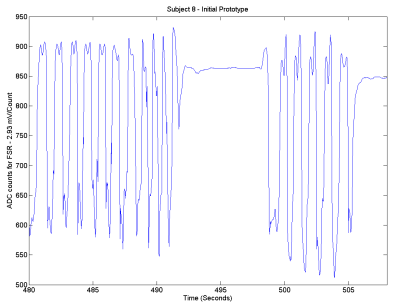
(d) Sitting



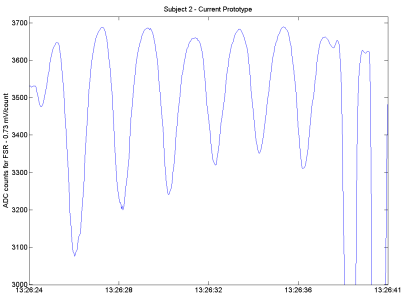
(e) Standing



(f) Elevated Prosthetic with suspected pulse



(g) Down then up stairs



(h) Weight Shifting

Figure 5.2: Examples of activities performed by amputees

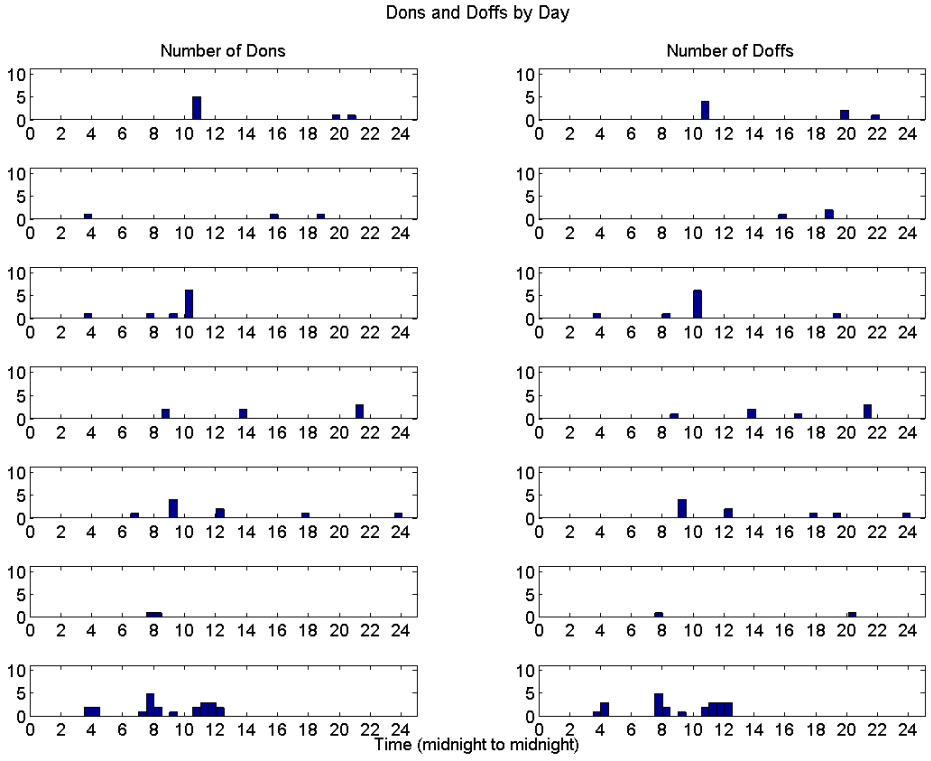


Figure 5.3: Subject 3 daily don/doff analysis

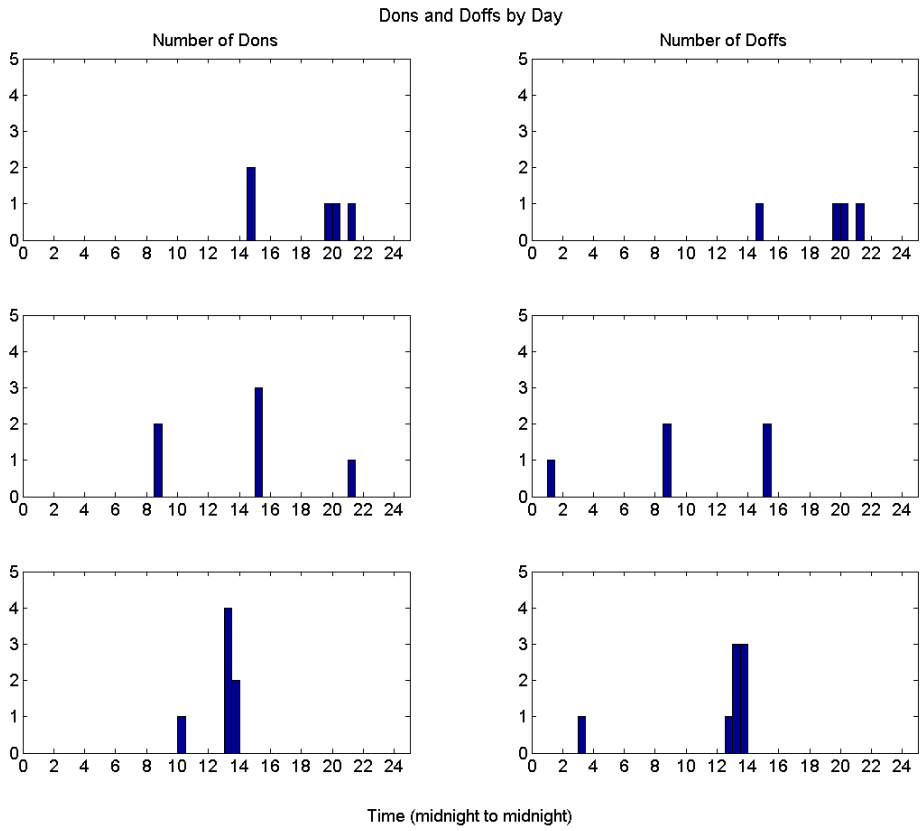


Figure 5.4: Subject 11 daily don/doff analysis

Chapter 6

DISCUSSION

6.1 Sock Detection Error

The overall detection of tags in both prototypes is poor. The metric used for the initial prototype is meant to only represent whether or not the Sock Monitor is successfully detecting tags. This value ideally should be at 100% because of the extremely low bar for success. However the average tag detection during activity was at 48% ± 35%. This extreme variability in detection was somewhat unexpected and led to further investigation into probable causes. One issue identified was the design of the in-socket RFID antenna. The antenna was found to have a very small area in which it could detect tags. Because of the small detection area, it was essential for the tags on the amputee's socks to be properly aligned with the antenna. This can be very difficult for the amputee who must rely on feel to align the tag. Also making the alignment more difficult was the amount of stretch in the socks. As the amputee would don their socks, they would stretch and change the placement of tag. Further compounding this antenna issue was the way in which the tags were being detected, which was statically. Incidental bench testing found that even though this less than ideal antenna had a small detection window for static detection, when moving tags dynamically past the antenna, the tag was almost always recorded. This observation led to the idea of capturing the tags as they enter and exit the socket. Another issue was the material the sockets were made from. Carbon fiber attenuates RF signals, creating a hostile environment inside of the socket for an antenna. Again, bench testing showed that although the antenna design and socket material were less than ideal, dynamically collecting tag data was superior to static polling that was being used on the initial prototype.

Other issues plagued the system, but were either hardware related or caused by user/investigator error. An example is subject 4, their data clearly shows a large number of tag

detections during the initial 2.5 hours of the recording. Then the subject doffs their prosthesis for about 30 minutes and after donning their prosthesis, only 3 more tags are recorded for the remaining 19.5 hours. This seems to indicate some sort of equipment failure, such as the antenna, tag, or connection to the antenna. In another subject, there were no tags detected. This was found to be a result of a shorted antenna. During the construction of the in-socket antenna, the coaxial wire attached to the copper strips was shorted across both strips of copper. This dramatically reduced the effectiveness of the antenna, preventing any tag readings for the entirety of the test.

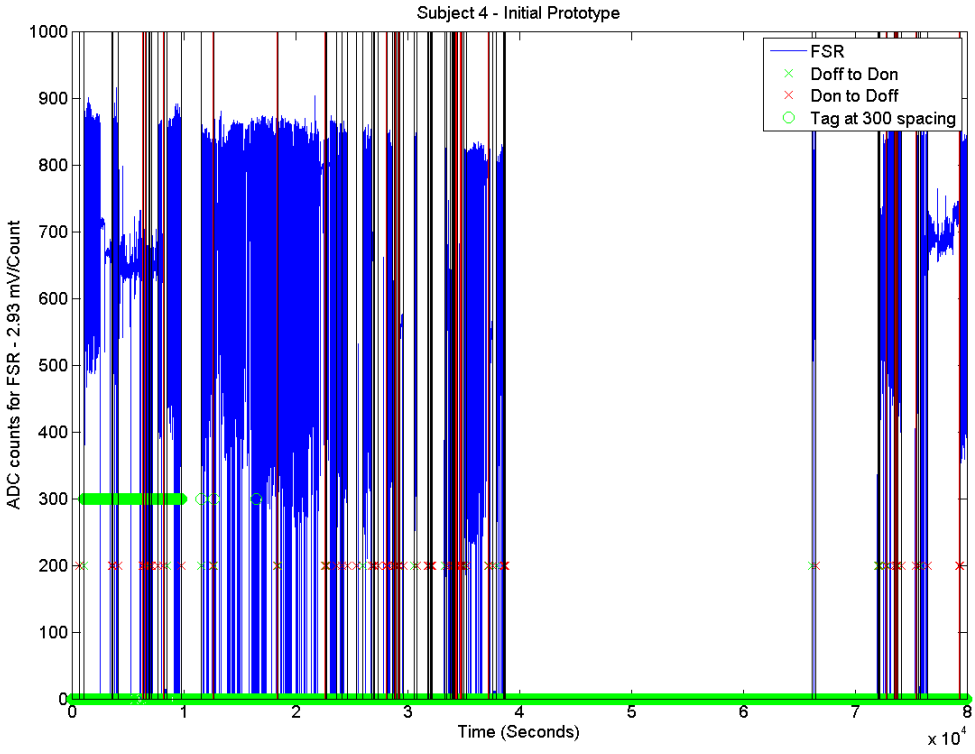


Figure 6.1: Subject 4 summary of data, note how tag detections stop after extended doff

The observations made during the initial pilot data collection were incorporated into a second prototype, version 1.3. This prototype would not look for tags on a set schedule, but instead would look for tags only during donning and doffing. This not only showed promise during bench testing but also had the added benefit of power conservation because

the RFID unit could be shut down during periods of activity or no use. The results from the only two subjects so far in this long term data collection show poor success in detecting tags during donning and very poor results during doffing, with an overall total success rate averaging at 33%. However, only one of these values needs to be around 100% in order to be considered successful. This is because the information being extracted is what socks the amputee is wearing. If an accurate count is made of what socks the amputee is wearing during donning *or* doffing, then the requirement has been satisfied.

What was encouraging from this generally poor detection was the dynamic polling algorithm is working correctly and is stable. The firmware seems to be powering down after 30 seconds of no activity and is able to quickly resume its data collection upon awakening. This is verified by plotting the raw data with and without time stamps. The data is continuous without time stamps and is discontinuous with time stamps.

In addition to the automatic detection of tags conducted by the MATLAB post processing software, a manual subjective evaluation of the FSR and tag data was conducted. Where ever the FSR data indicated a doff or the FSR magnitude changed to/from zero, the likelihood of it being a don or doff was rated on a four level scale. The four levels were a don/doff clearly occurred, a don/doff probably occurred, a don/doff probably did not occur, or a don/doff surely did not occur. Factors considered when categorizing the dons and doffs included the current activity as indicated by the FSR data from the subject, time between doff and don, and duration of activity. If at least one tag was present at the don/doff being evaluated, this was also recorded for later evaluation of tag detection effectiveness for the different levels of certainty. The results of this manual evaluation are shown in Table 6.1. Using this method, the number of tags successfully detected during periods of donning significantly improved for both subjects. Subject 3 and 11 had an increase in detection by 31% and 18% respectively. However, the number of tags successfully detected during doffing either didn't change significantly or became worse. Another observation made was how few definite or probable dons and doffs comprise the total number seen in the data. Only 26% of dons and doffs for subject 3 and 35% of dons and 33% of doffs for subject 11 were

categorized in the highest levels certainty.

Table 6.1: Subjective review of dons and doffs using the Sock Monitor v1.3 data

Subject 3				
	Instances	Tags Detected	% Detected	% of Total
Definite Don	19	14	74%	13%
Probably Don	18	10	56%	13%
Probably not Don	13	2	15%	9%
Surely not Don	92	25	27%	65%
Definite Doff	17	1	6%	12%
Probably Doff	20	1	5%	14%
Probably not Doff	13	2	15%	9%
Surely not Doff	91	7	8%	65%
Subject 11				
	Instances	Tags Detected	% Detected	% of Total
Definite Don	12	10	83%	32%
Probably Don	1	1	100%	3%
Probably not Don	3	1	33%	8%
Surely not Don	22	3	14%	58%
Definite Doff	11	1	9%	30%
Probably Doff	1	0	0%	3%
Probably not Doff	3	2	67%	8%
Surely not Doff	22	3	14%	59%

These results from the subjective evaluation show the detection rate for donnings may be significantly better than initially thought. The post-processing program used to determine the number of tags successfully detected may need to be re-evaluated to better filter out definite dons and doffs. However, a better understanding of what constitutes a definite don

and doff needs to be quantitatively defined. Another issue that needs to be better defined is the acceptable level of error in the sock detection system.

6.2 Physical Activity Patterns

As seen in the results, there are many clear qualitative patterns for various forms of activity. Sitting and standing are characterized as constant values, with standing being a significantly higher in value than sitting. Running and walking are very similar in appearance, displaying a sinusoidal pattern. The difference between the two can be found by using a Fast Fourier Transform (FFT) to determine the frequency of the stride, and if the average step length is known, the speed of the subject. Weight shifting is similar to walking or running in its output amplitude, however it is not sinusoidal in behavior. Walking up and down stairs is more difficult to distinguish using FSR data. Some subjects display a double peak around the contralateral swing phase of gate. This double peak can be used to separate walking versus walking up or down stairs. When a subject walks down stairs, the first peak is larger in magnitude than the second peak. The opposite trend is observed for walking up stairs. During in-lab testing, the peaks were approximately equal in magnitude during a treadmill walk. These observations are only made in amputees that have a clear double peak during contralateral swing, a characteristic that not all subjects have demonstrated. More investigation needs to be conducted to determine if this is a function of the amputee's gate, or if this is a function of the sensor.

Other observations made from the FSR data are the logarithmic rise in amplitude during periods of sitting and standing, as well as the suspected detection of the subject's pulse. Given the nature of the FSR to drift over time, it is suspected this is the cause in the logarithmic rise. More testing of the FSR needs to be conducted before this can be confirmed. If however the rise is not due to drift, this has potential to be anything from additional force due to swelling of the limb, to some sort of unconscious loading behavior. The possible pulse detection seems reasonable considering the proximity of the FSR to the popliteal artery in the residual limb. Additional research is needed to determine if this is indeed due to the subject's pulse, or if this is caused by electrical interference from the environment.

The distinct patterns from the FSR data can be used in determining the physical state of the amputee at various times during the day. Once the physical state of the amputee is known, this information could be used to help determine the appropriate sock prescription at that moment in time. Other applications include changing various aspects of smart prostheses, whether that be simply changing the tension in an energy harvesting device, or integrating totally into a fully robotic prosthesis. The FSR data gathered from inside the prosthetic socket is a valuable piece of data that can greatly contribute to the understanding of the physical state of the amputee.

Identifying when a subject is or is not wearing their prosthesis is made simple when using an in-socket FSR. If the sensor value is above zero, then the amputee is wearing or utilizing their prosthesis in some capacity. Identifying the time of day and frequency of donning and doffing also becomes easier with this data. Figure 5.3 and Figure 5.4 are examples of multi day data collection of don and doff frequencies by day. When combining these into a single representation for donning and doffing over the whole test period, an interesting trend is observed. The frequency of donning and doffing seem to occur at the same time of day. This result is counterintuitive to what would be expected. One would think that there would be a strong grouping in the morning for donning and a strong grouping in the evening for doffing with possibly a few other smaller groupings in between for various reasons. However, this is not the case. Upon closer inspection of the individual donnings and doffings, there were several instances of frequent don/doff periods over several seconds. These looked to be intentional actions by the amputee. This brings up the issue of possibly modifying the existing don/doff recognition algorithm to look for periods of activity greater than some time threshold. This modification could give a more accurate account of donning and doffing behavior. However, the cost would be an incorrect assessment of the effective tag capture by the Sock Monitor v1.3. Determining what the time threshold should be can be the subject of a future study.

While the FSR may be a good indication of various physical activities, there can still be some ambiguous periods in the data. However, when combining the FSR and 3-axis

accelerometer data, it is possible to clarify some of those unknown areas. One example is during periods of sitting. The FSR data will clearly show when an amputee is sitting with their feet flat on the ground. However if that amputee then puts their prosthetic limb on a piece of furniture or on something that is elevated, like a foot rest on a stool or elevated chair, then that FSR signal begins to look like the amputee is standing. This is where the accelerometer data can be used to determine the orientation of the limb and then distinguish between sitting and standing. In addition, calculating net elevation change with the accelerometer combined with the waveforms from the FSR may be able to tell the difference between stairs and inclined surfaces.

6.3 Clinical Applications

The Sock Monitor has great clinical potential to improve the quality of life for lower limb amputees. Previous to this device, the only way to track sock usage data was through self-reporting. This device now gives clinicians the opportunity to quantitatively analyze actual sock usage patterns as well as physical activity. This combination of information can help clinicians better understand how amputees are utilizing their prostheses. Clinicians will also be able to observe what activities induce volume changes in amputees and may better prescribe socks or other methods for volume accommodation.

Although it has been presented as a passive observational tool, this device has the hardware capability to be used as an active system that can communicate with the amputee user through the use of a smartphone or personal computer. The Sock Monitor hardware will be able to monitor in real time the physical activity and sock usage of an amputee. The device has the potential to constantly calculate if a sock change is necessary and suggest additions/subtractions as they are needed. The device also has the potential to be able to monitor compliance with the suggestions and report this data to the clinician. Again, this data could be used by the clinician to improve the volume management system used by the amputee.

Another side effect of using the sock monitor could be a reduced incidence of limb tissue injury due to poor volume management. As the amputee follows the prompting of the Sock Monitor, they are taking a proactive step in regulating the comfort of their limb. This proactive approach is hypothesized to prevent sores and other limb tissue issues from developing.

Insurance companies will also benefit from the development of the Sock Monitor. Clinicians will be able to use evidence-based justification for new components and/or treatments. For example, an amputee may be classified as a K-2 ambulator and may complain of an improperly fitting socket. Their insurance may deny a new socket based on their MFCL status, however, the data captured by the Sock Monitor would show high levels of activity that would redefine their MFCL status to K-3 and therefore allowing the amputee to receive a more comfortable prosthesis. Compliance to clinician instructions could also be submitted to insurance, again, providing evidence-based justification to the use of a particular component or system. Another appealing aspect of the Sock Monitor system would be the estimated reduction in healthcare costs associated with treating residual limb issues due to poor volume management. This overall savings is mutually beneficial to both the amputee and the interests of the insurance companies involved.

A recent trend in healthcare is mobile monitoring. Individuals are being given the power to monitor their own health, and in some cases, this information can also be used by their healthcare provider to remotely access their health [38]. The Sock Monitor would be a perfect example of such a device. An amputee could periodically upload their usage data to a cloud where various healthcare providers could access the information. They would be able to determine if the amputee needs additional treatment or if they appear to be healthy. This could save amputees time, by eliminating unnecessary trips to their clinician, and it would free the clinician to see more amputees that actually need attention.

Overall, the Sock Monitor device has a great potential to be utilized by many different individuals in the prosthetics community. The clinical uses for the Sock Monitor vary from

observational research tool, to an active device that reminds amputees when to change their socks. Through using this device, amputees could potentially experience a more comfortable socket and a reduction of limb soft tissue injuries.

6.4 Future Work

The Sock Monitor has shown to be a valuable tool in the research of amputee activity and sock usage. This thesis focused on developing the Sock Monitor as a tool, as well as collecting a limited pilot data set. The future work related to the Sock Monitor involves the development of clinical strategies and validation of those strategies in improving socket fit and function. Additional data needs to be collected for use in identifying various physical activities described in other sections. Once these activities are better understood and can be extracted, the relationship between sock usage and physical activity can be fully explored. A correlation between these two data sets will hopefully provide some insight into how to better manage volume in transtibial amputees. After this correlation is understood, a clinical test focused on an intervention strategy can be explored. This clinical trial could explore such questions as, is sock usage dependent on time of day? Is wearing an extra sock ply during the morning for a certain period of time help the fit for the rest of the day? Does the intervention of the Sock Monitor improve comfort and reduce limb injuries?

Before any clinical tests with intervention strategies can be conducted, a user interface is needed to communicate with the amputee. This interface must be capable of prompting the user for input. The interface must also be able to communicate to the Sock Monitor device. Finally, to reduce both the burden on the amputee, and the financial burden on the researching lab, the interface must be found on a device already owned by a majority of the amputee subjects. Smartphones naturally lend themselves to such an application. A smartphone could be integrated into the design of the Sock Monitor system to do anything ranging from basic user notification and survey prompting, to real-time signal processing. Initial steps toward this goal have already begun under this thesis. A basic application (app) for the Android platform, utilizing the Open Development Kit (ODK) developed at the University of Washington, has been started. This basic app will give the user control

of the various functions of the Sock Monitor, as well as be able to plot the physical activity data of the user in real time. This particular app will be focused on aiding researching efforts, rather than targeting the clinical setting.

Also slated for future work is improved antenna design, integration of active RFID tags, and improving the manufacturability of in-socket components. The design of the in-socket antenna must be improved for near-field detection and manufacturability. The antenna is meant to detect tags as they enter and exit the socket, but another long term goal is to communicate with active RFID tags within the socket. These active RFID tags may be able to provide additional data from sensors such as accelerometers, strain gauges, temperature, and neural based signals. These improvements will make great research tools but in order to get this device out of the lab and into the hands of those that would benefit most, improvements in the manufacturability must be made. This includes integrating the FSR and antenna into an easy to install package that is easily serviceable and prone to little error. One solution would be to contact Interlink Electronics and ask them if they could integrate one of their FSRs into a printed antenna. This flexible circuit assembly could then be placed onto the mold surface of a prosthesis while it was being manufactured, creating an integrated device.

BIBLIOGRAPHY

[1] Prosthetic Socks And Your Prosthesis.

[2] SW Levy. Skin problems of the leg amputee. *Prosthetics and Orthotics International*, 4(1):37–44, 1980.

[3] J SANDERS, B OTIS, K ALLYN, B HAFNER, and J CAGLE. LIMB VOLUME ACCOMMODATION IN PEOPLE WITH LIMB AMPUTATION, 2012.

[4] RC Eastman. *Diabetes in America*. National Institute of Diabetes and Digestive and Kidney Diseases, Bethesda, 2 edition, 1995.

[5] JM Stepien, Sally Cavenett, Leigh Taylor, and Maria Crotty. Activity levels among lower-limb amputees: self-report versus step activity monitor. *Archives of Physical Medicine and Rehabilitation*, 88(7):896–900, July 2007.

[6] Kathryn Ziegler-Graham, Ellen J MacKenzie, Patti Ephraim, Thomas Travison, and Ron Brookmeyer. Estimating the prevalence of limb loss in the United States: 2005 to 2050. *Archives of Physical Medicine and Rehabilitation*, 89(3):422–429, March 2008.

[7] TR Dillingham and LE Pezzin. Use and satisfaction with prosthetic devices among persons with trauma-related amputations: a long-term outcome study. *American journal of . . .*, 80:563–571, 2001.

[8] Robert S Gailey, Kathryn E Roach, E Brooks Applegate, Brandon Cho, Bridgid Cunniffe, Stephanie Licht, Melanie Maguire, and Mark S Nash. The amputee mobility predictor: an instrument to assess determinants of the lower-limb amputee's ability to ambulate. *Archives of Physical Medicine and Rehabilitation*, 83(5):613–627, May 2002.

[9] SG Zachariah, Rakesh Saxena, John Fergason, and Joan E Sanders. Shape and volume change in the transtibial residuum over the short term: Preliminary investigation of six subjects. *Journal Of Rehabilitation Research And Development*, 41(5):683–694, 2004.

[10] Nancy L Dudek, Meridith B Marks, and Shawn C Marshall. Skin problems in an amputee clinic. *American journal of physical medicine and rehabilitation*, 85(5):424–429, May 2006.

[11] Lower-Limb Amputation Fact Sheet, 2010.

[12] Timothy R Dillingham, Liliana E Pezzin, and Andrew D Shore. Reamputation, mortality, and health care costs among persons with dysvascular lower-limb amputations. *Archives of Physical Medicine and Rehabilitation*, 86(3):480–486, March 2005.

[13] JE Sanders, JC Cagle, DS Harrison, and A Karchin. AMPUTEE SOCKS: HOW DOES SOCK PLY RELATE TO SOCK THICKNESS? *Prosthetics and orthotics international*, 36(1):77–86, March 2012.

[14] Richard M Greenwald, Robert C Dean, and Wayne J Board. Volume Management : Smart Variable Geometry Socket. *American Academy of Orthotists and Prosthetists*, 15(3):107–112, 2003.

[15] M. Jason Highsmith and Jason T. Kahle. Prosthetic Socks: Simple , Relatively Inexpensive and Critically Important. *inMotion*, 16(2):1–3, 2006.

[16] JE Sanders, DS Harrison, Katheryn J Allyn, Timothy R Myers, Marcia Ciol, and Elaine Tsai. How do sock ply changes affect residual-limb fluid volume in people with transtibial amputation? *Journal Of Rehabilitation Research And Development*, 49(2):241–256, 2012.

[17] Prosthetic Amputee Stump Socks.

[18] Angel Gel BK Prosthetic Socks from Comfort.

[19] W. J. Board, G. M. Street, and C. Caspers. A comparison of transtibial amputee suction and vacuum socket conditions. *Prosthetics and Orthotics International*, 25(3):202–209, January 2001.

[20] DMEPOS Fee Schedule, 2013.

[21] Joan E. Sanders and Stefania Fatone. Residual limb volume change: Systematic review of measurement and management. *The Journal of Rehabilitation Research and Development*, 48(8):949–986, 2011.

[22] Pump It Up!

[23] DJ Yeager, J Holleman, JR Smith, and Brian P Otis. Neuralwisp: A wirelessly powered neural interface with 1-m range. *IEEE Transactions on Biomedical Circuits and Systems*, 3(6):379–387, 2009.

[24] Roy Want. *RFID Explained: A Primer on Radio Frequency Identification Technologies*, volume 1. January 2006.

[25] Artem Dementyev and Joshua R Smith. A Wearable UHF RFID-Based EEG System. In *Proceedings of IEEE RFID*, Orlando, 2013.

[26] Daniel Dobkin. *RF in RFID - Passive UHF RFID in Practice*. Elsevier, 2008.

[27] EPCglobal. Specification for RFID Air Interface EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz 960 MHz. Technical Report October, 2008.

[28] Sensor Construction, 2013.

[29] Force Sensing Resistor Integration Guide and Evaluation Parts Catalog, 2013.

[30] M Ferguson-Pell, S Hagisawa, and D Bain. Evaluation of a sensor for low interface pressure applications. *Medical engineering & physics*, 22(9):657–63, November 2000.

[31] N. Maalej, S. Bhat, H. Zhu, J.G. Webster, W.J. Tompkins, J.J. Wertsch, and P. Bach-y Rita. A conductive polymer pressure sensor. In *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 770–771 vol.2, New Orleans, 1988. IEEE.

[32] S.I. Yaniger. Force Sensing Resistors: A Review Of The Technology. In *Electro International, 1991*, pages 666–668, New York, NY, 1991. IEEE.

[33] P Convery and AWP Buis. Conventional patellar-tendon-bearing (PTB) socket/stump interface dynamic pressure distributions recorded during the prosthetic stance phase of gait of a transtibial. *Prosthetics and orthotics ...*, 22(3):193–198, 1998.

[34] Accelerometers and How they Work. Technical report, Texas Instruments.

[35] A Godfrey, R Conway, D Meagher, and G OLaighin. Direct measurement of human movement by accelerometry. *Medical Engineering & Physics*, 30(10):1364–1386, December 2008.

[36] EK Antonsson and RW Mann. The frequency content of gait. *Journal of biomechanics*, 18(1):39–47, 1985.

[37] MSP430x5xx/MSP430x6xx Family User's Guide, 2011.

[38] BodyGuardian Remote Monitoring System, 2013.

[39] Klaus Finkenzeller. *RFID Handbook - Fundamentals and Applications in Contactless Smart Cards and Identification (2nd Edition)*. John Wiley & Sons, 2 edition, 2003.

[40] Robert Gailey, Kerry Allen, Julie Castles, Jennifer Kucharik, and Mariah Roeder. Review of secondary physical conditions associated with lower-limb amputation and long-term prosthesis use. *Journal Of Rehabilitation Research And Development*, 45(1):15–29, December 2008.

[41] Xiaohong Jia, Ming Zhang, and WCC Lee. Load transfer mechanics between trans-tibial prosthetic socket and residual limbdynamic effects. *Journal of biomechanics*, 37(9):1371–1377, September 2004.

[42] VS Nelson, KM Flood, PR Bryant, ME Huang, PF Pasquina, and TL Roberts. Limb deficiency and prosthetic management. 1. Decision making in prosthetic prescription and management. *Archives of physical . . .*, 87(Suppl 1):S3–S9, March 2006.

[43] BM Persson and E Liedberg. A clinical standard of stump measurement and classification in lower limb amputees. *oandplibrary.org*, 7(1):17–24, 1983.

[44] Liliana E Pezzin, Timothy R Dillingham, Ellen J Mackenzie, Patti Ephraim, and Paddy Rossbach. Use and satisfaction with prosthetic limb devices and related services. *Archives of Physical Medicine and Rehabilitation*, 85(5):723–729, May 2004.

[45] S Portnoy, Z Yizhar, N Shabshin, Y Itzchak, A Kristal, Y Dotan-Marom, I Siev-Ner, and A Gefen. Internal mechanical conditions in the soft tissues of a residual limb of a trans-tibial amputee. *Journal of Biomechanics*, 41(9):1897–1909, January 2008.

[46] A Salawu, C Middleton, K Kodavali, and V Neumann. Stump ulcers and continued prosthetic limb use. *Prosthetics and orthotics international*, 30(3):279–285, December 2006.

[47] J E Sanders and D V Cassisi. Mechanical performance of inflatable inserts used in limb prosthetics. *Journal of rehabilitation research and development*, 38(4):365–374, 2001.

[48] JE Sanders, SG Zachariah, A.K. Jacobsen, and JR Fergason. in interface pressures and shear stresses over time on trans-tibial amputee subjects ambulating with prosthetic limbs: comparison of diurnal and six-month differences. *Journal of Biomechanics*, 38(8):1566–1573, August 2005.

[49] Wirelessly Powered Sensor Networks and Computational RFID. 2013.

Appendix A

SCHEMATIC

A.1 Sock Monitor Version 1.2

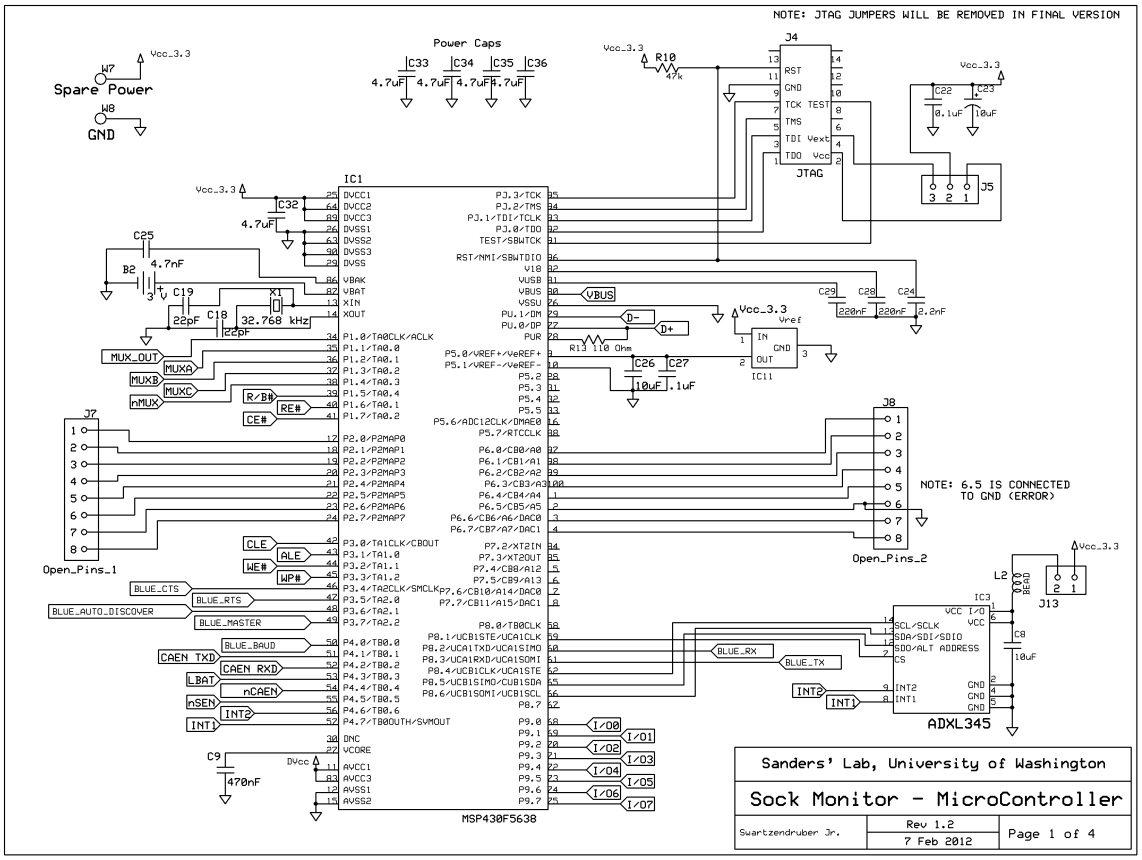


Figure A.1: Main processor with accelerometer

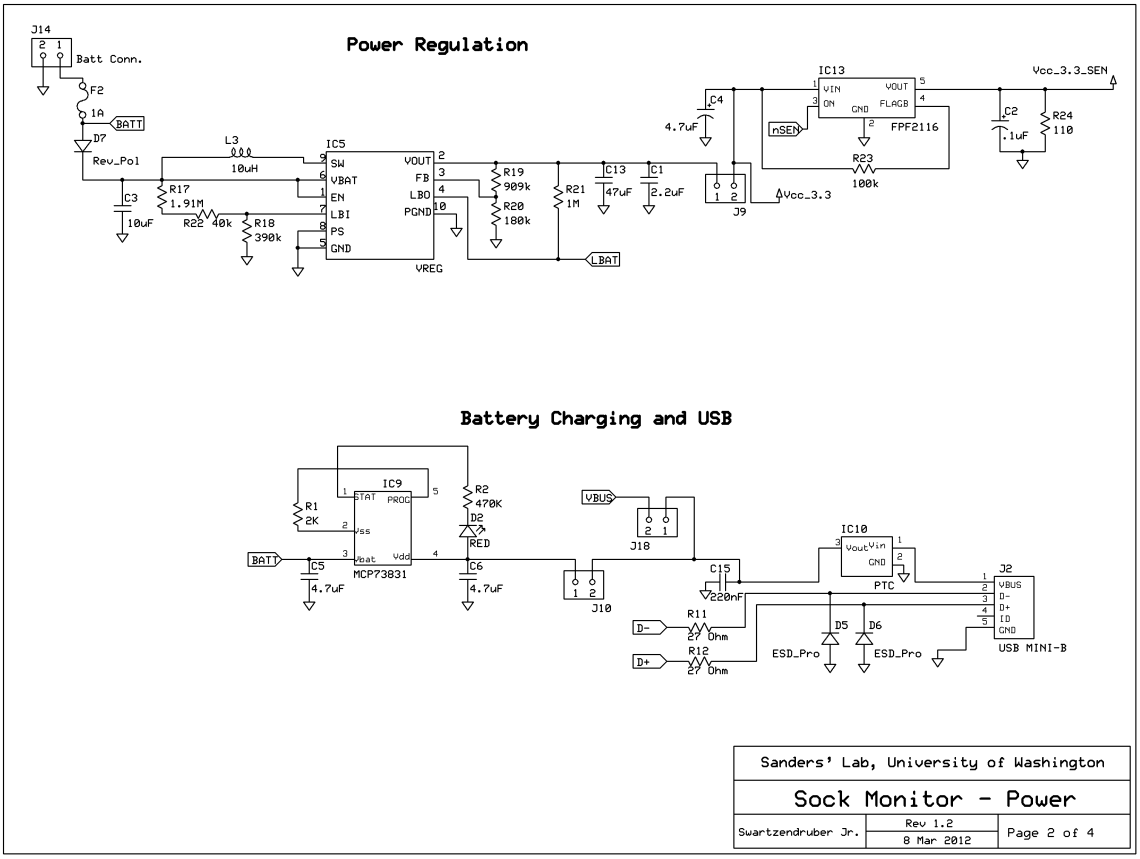


Figure A.2: Charging and regulation

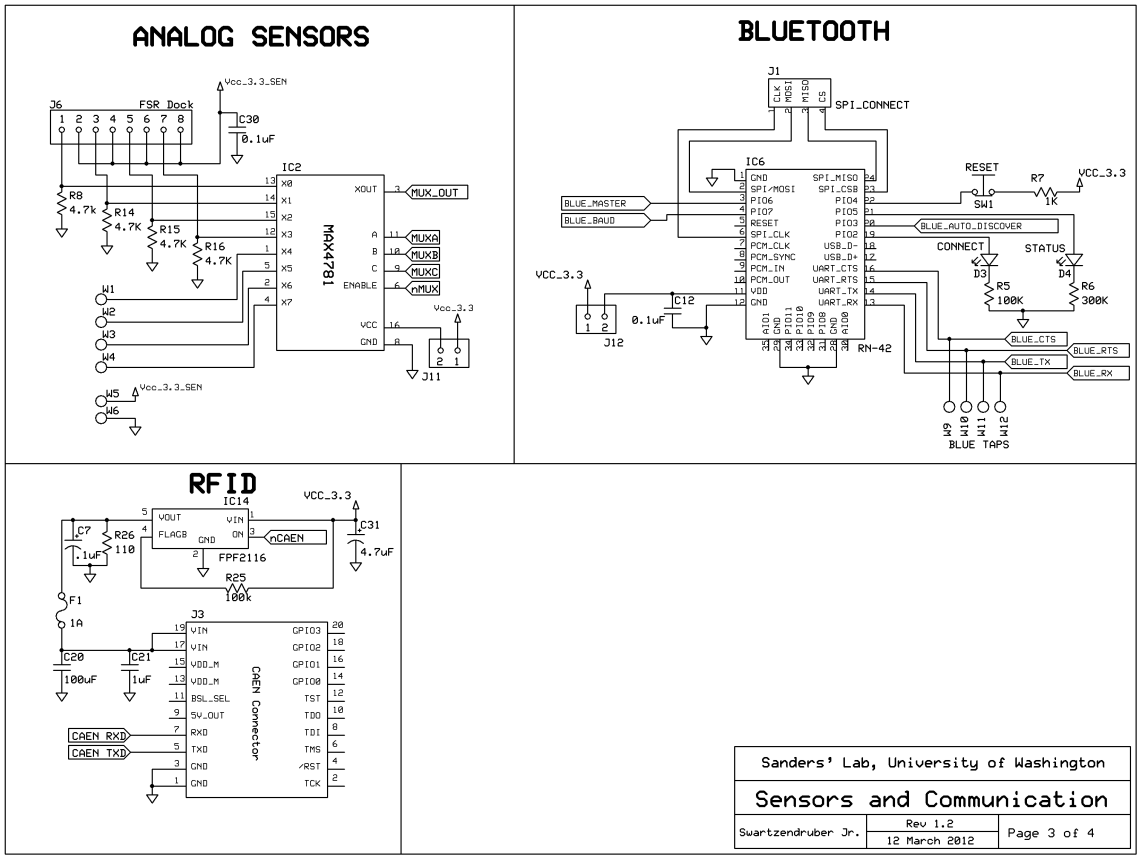


Figure A.3: Sensors

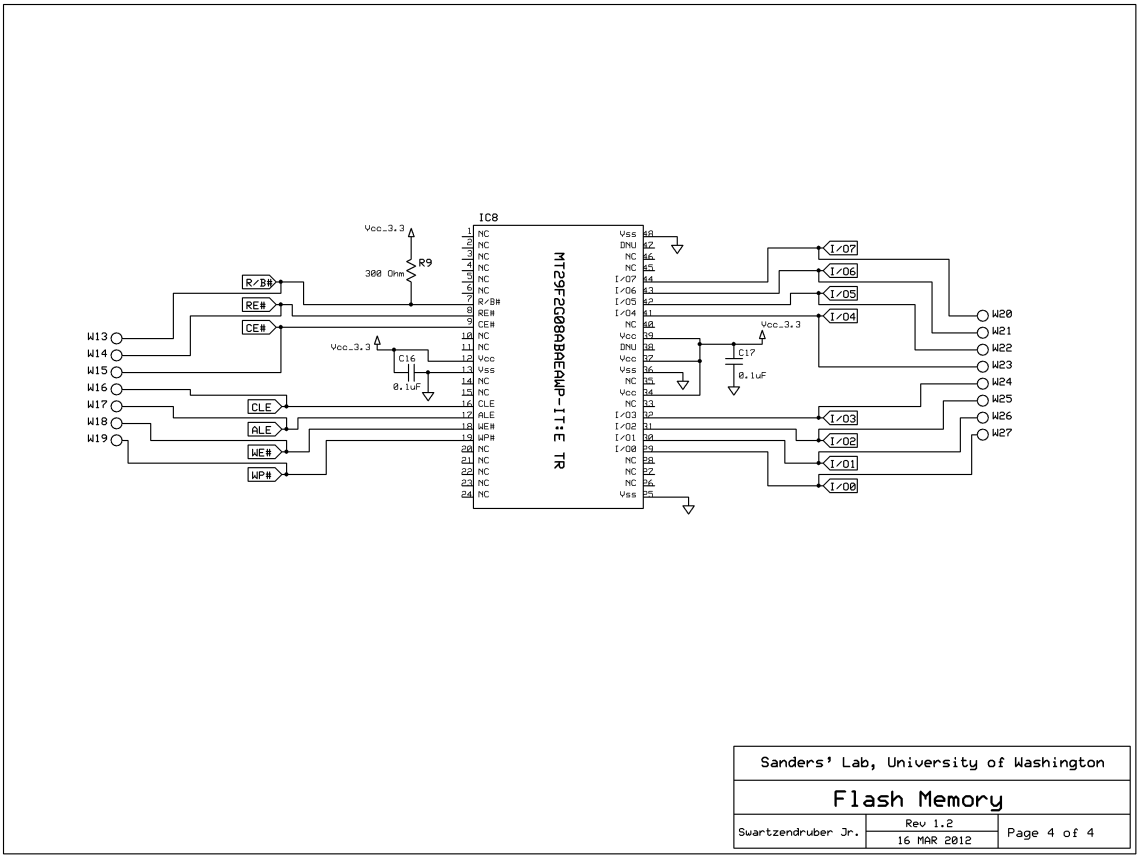


Figure A.4: Flash memory

A.2 Sock Monitor Version 1.3

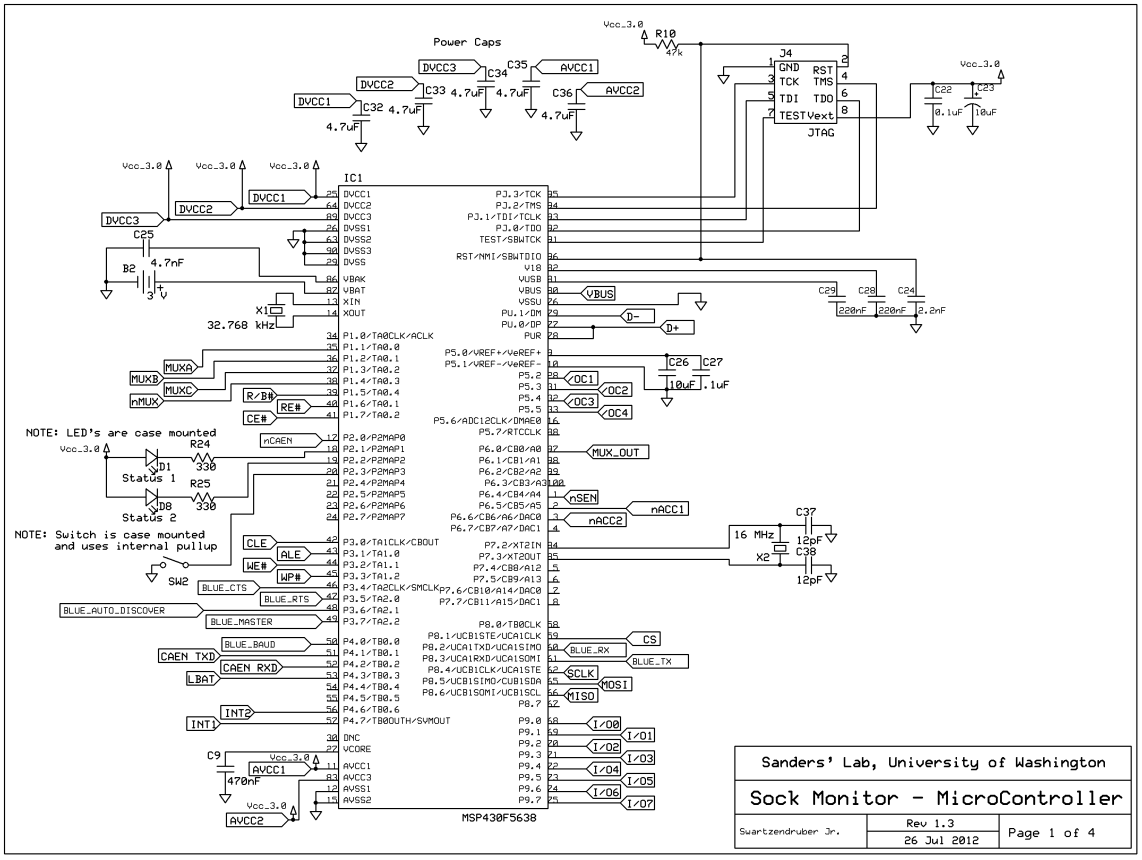


Figure A.5: Main processor with accelerometer

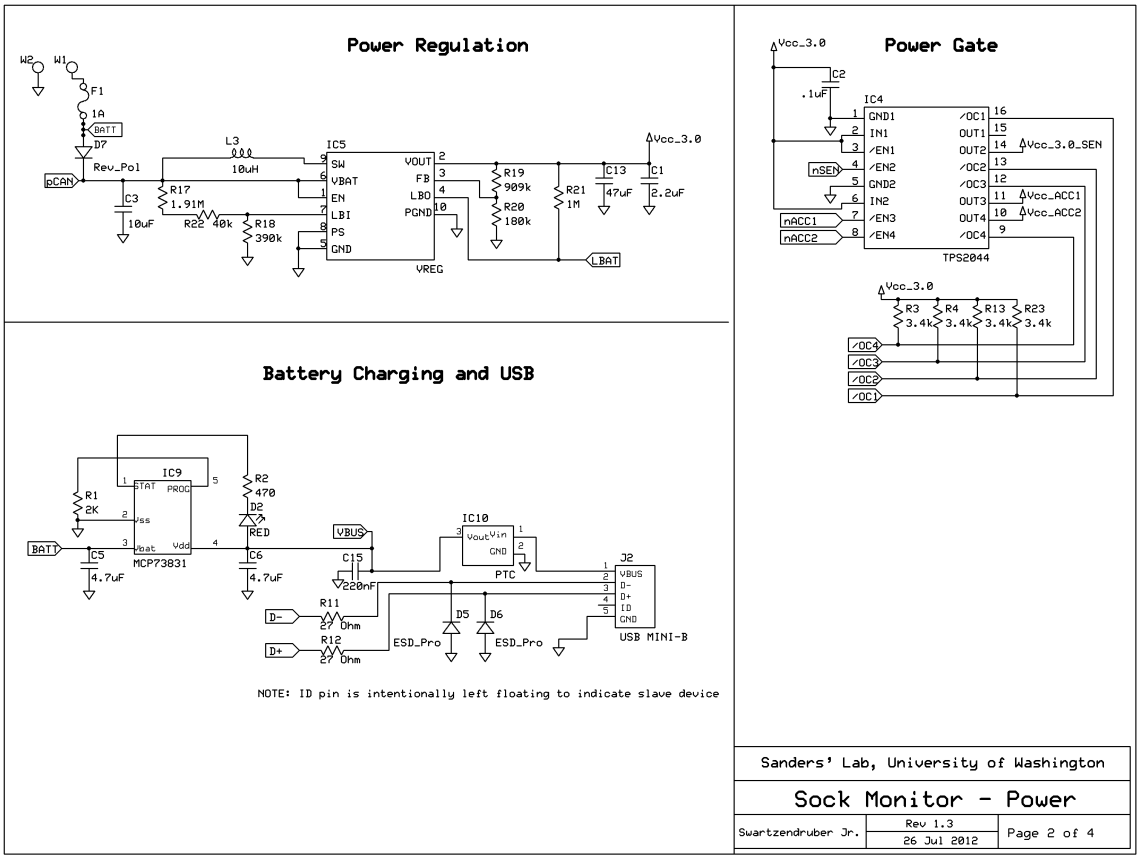


Figure A.6: Charging and regulation

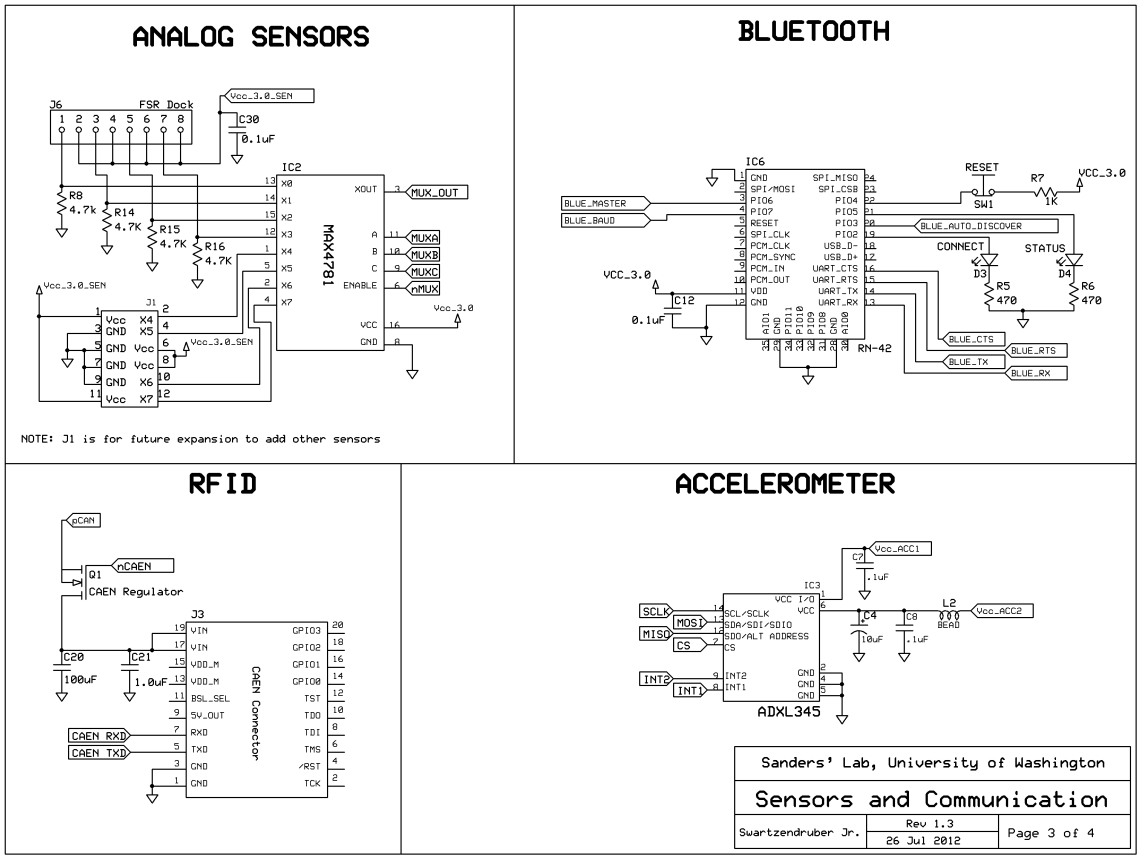


Figure A.7: Sensors

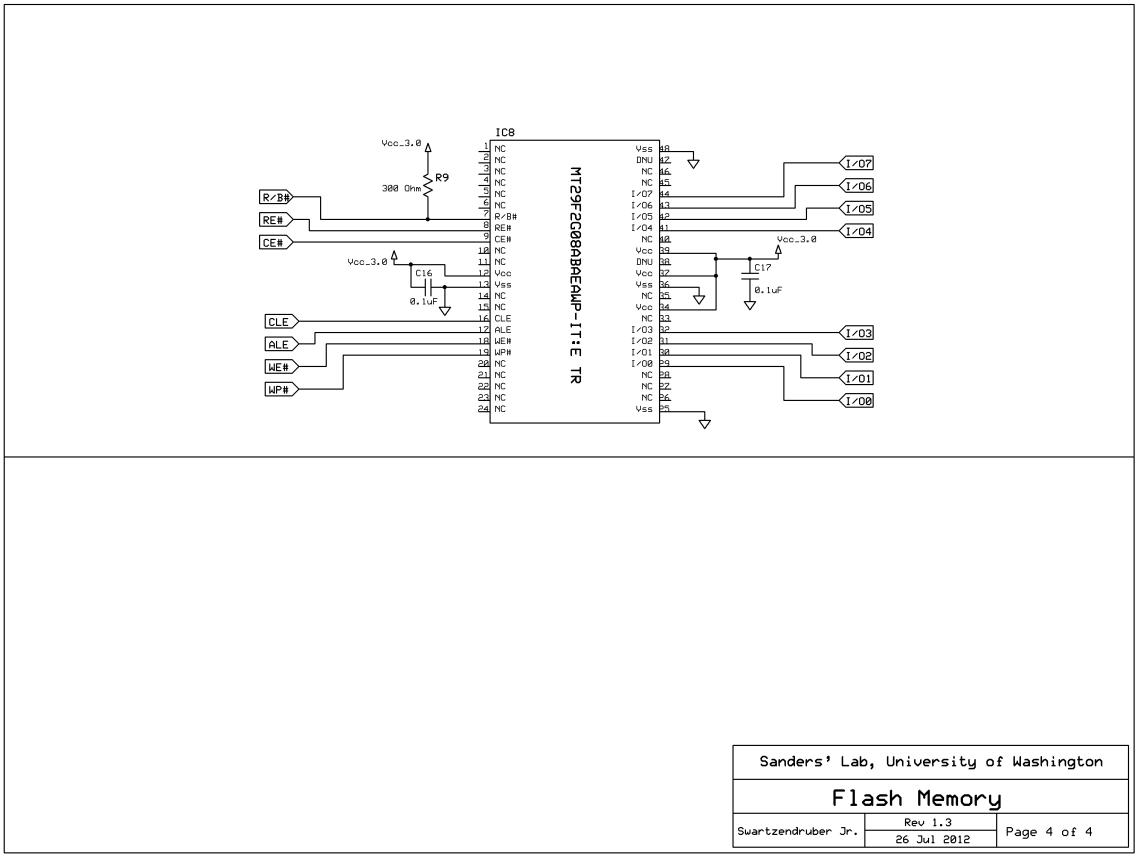


Figure A.8: Flash memory

Appendix B

LAYOUT

B.1 Sock Monitor Version 1.2

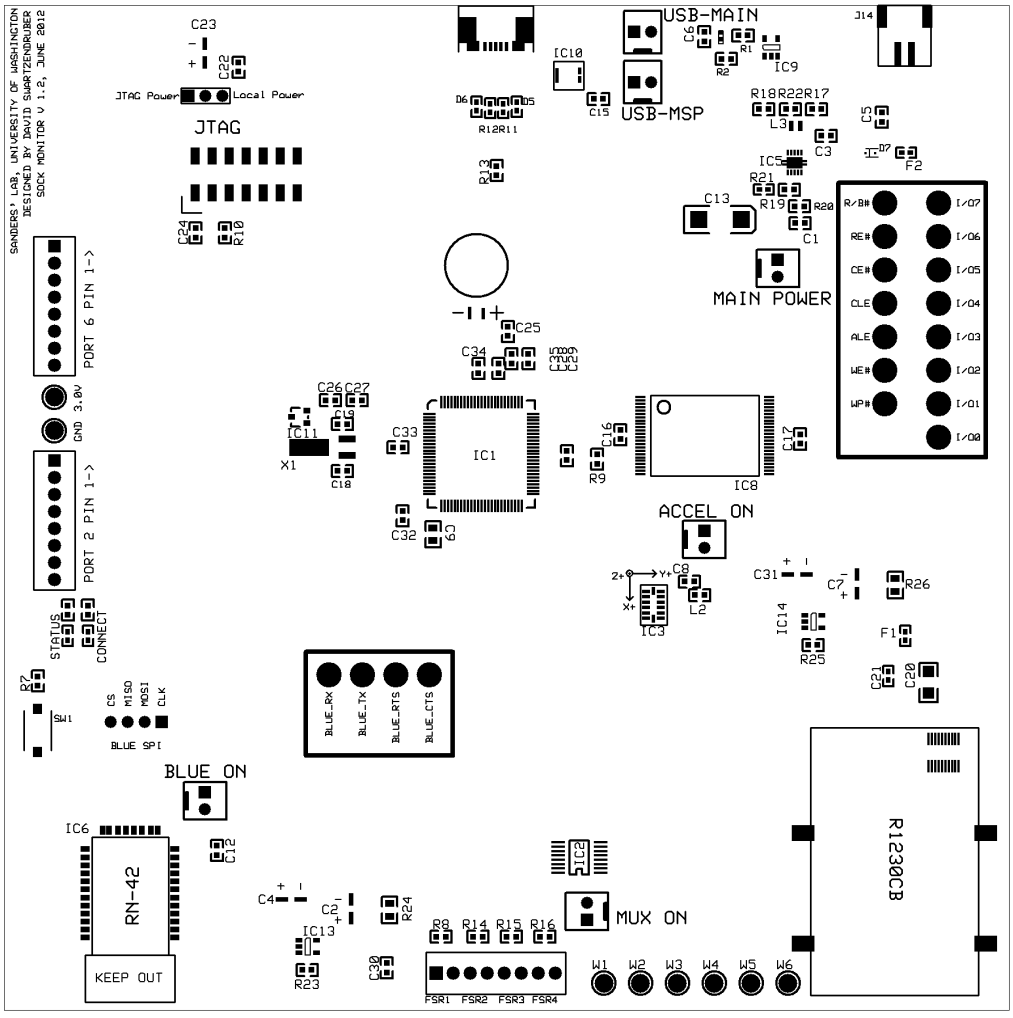


Figure B.1: Pad layout

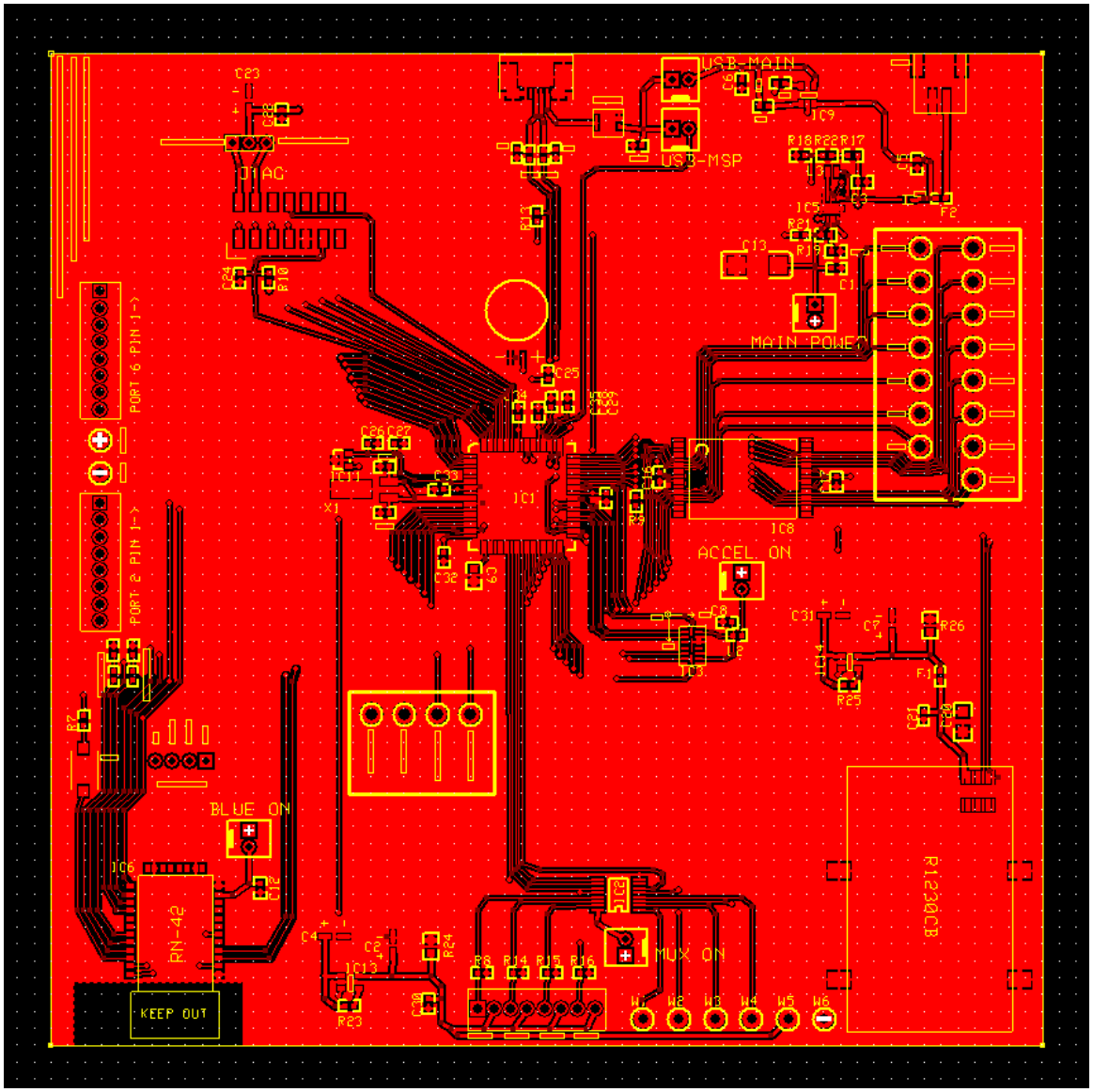


Figure B.2: Top side layout

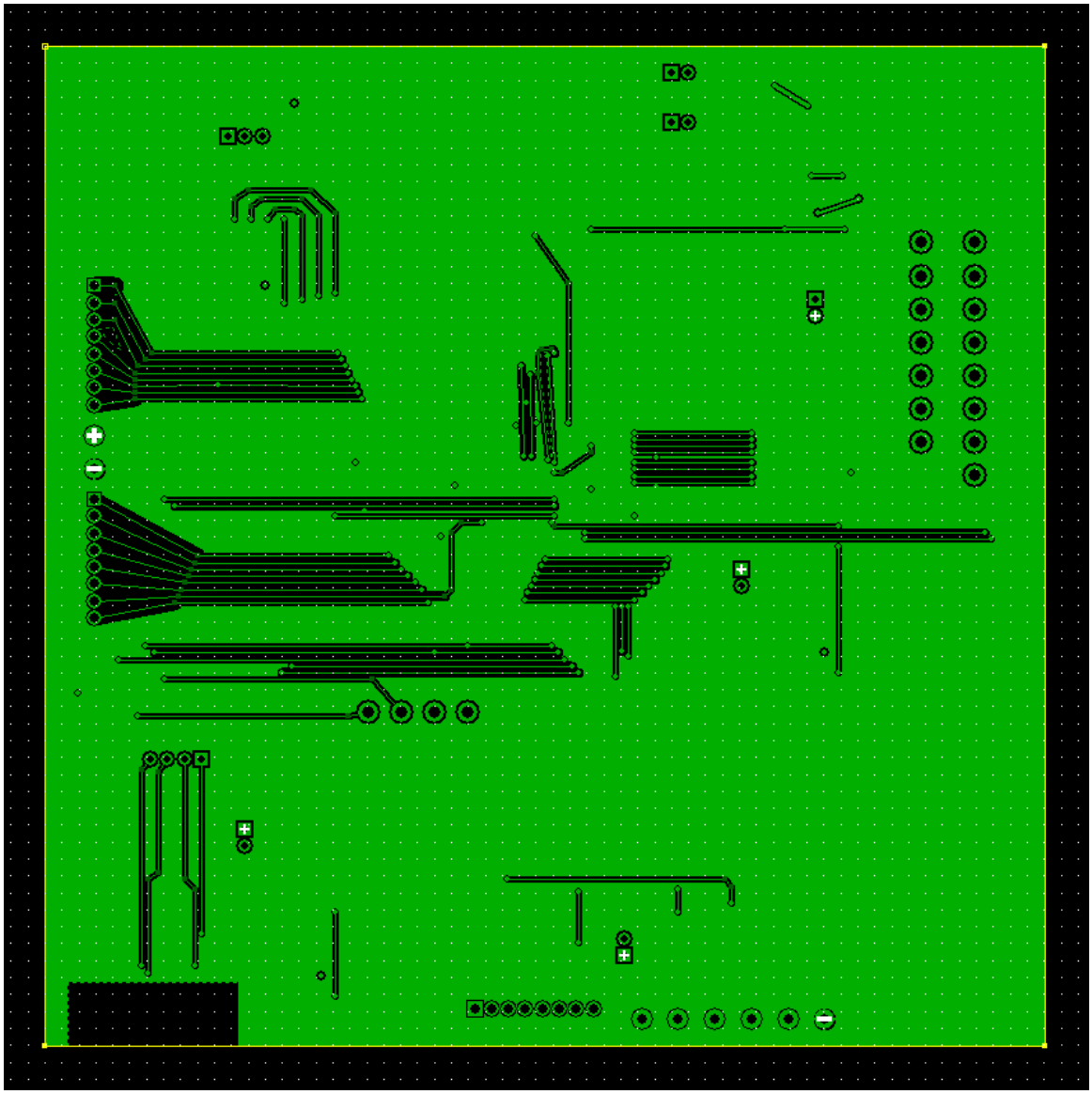


Figure B.3: Bottom side layout

B.2 Sock Monitor Version 1.3

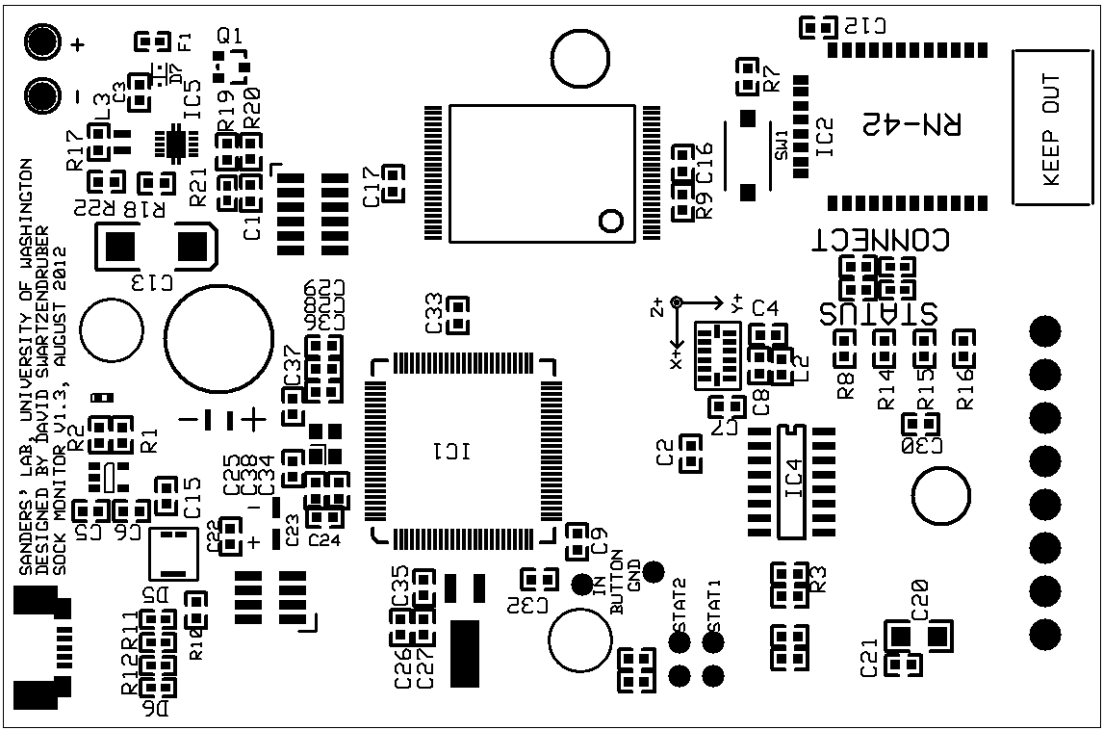


Figure B.4: Pad layout

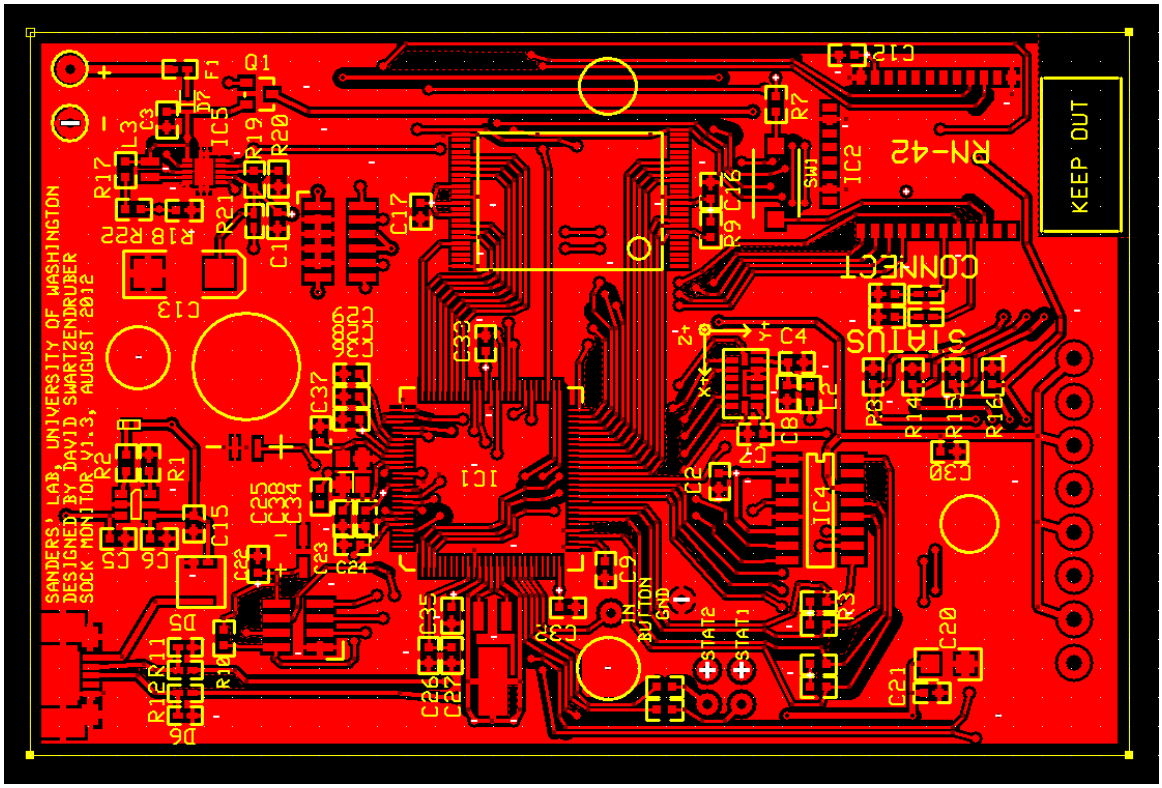


Figure B.5: Top side layout

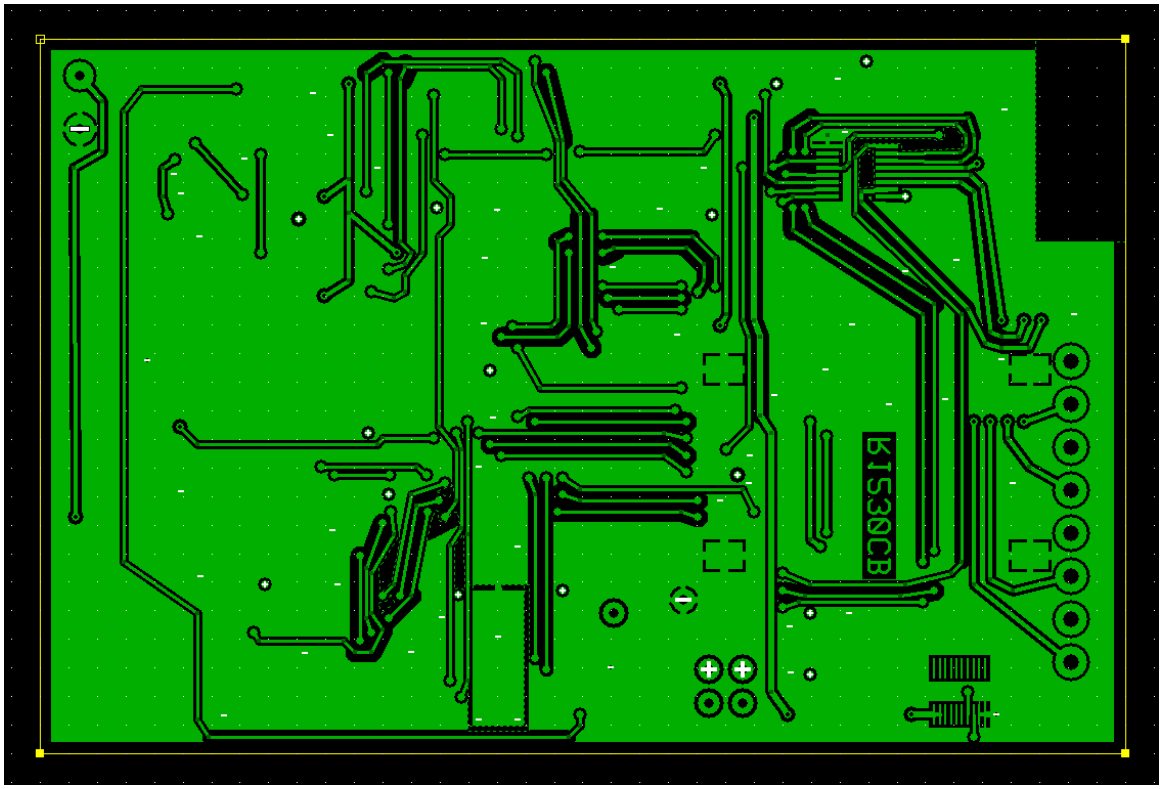


Figure B.6: Bottom side layout

Appendix C

ANNOTATED SOCK MONITOR IMAGES

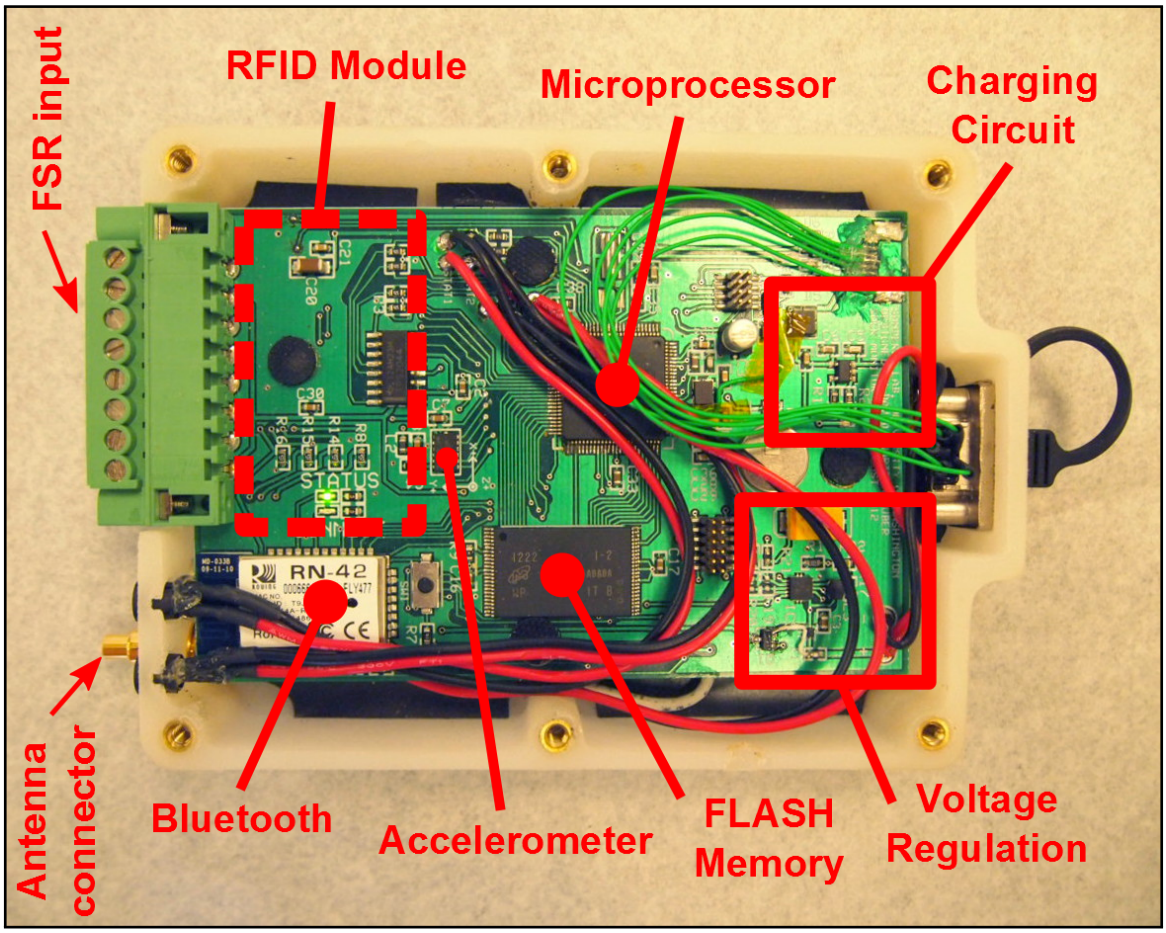


Figure C.1: Top side of v1.3 PCB

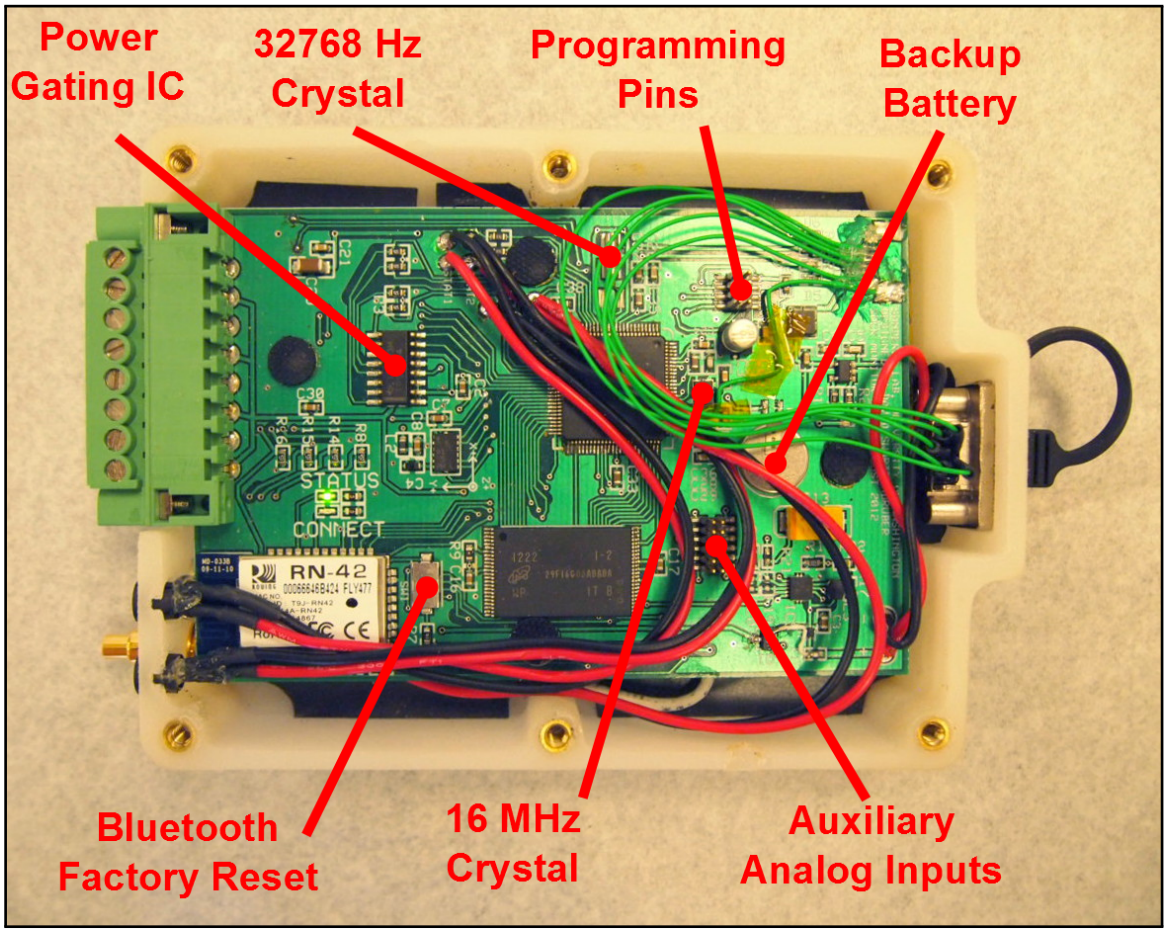


Figure C.2: Top side of v1.3 PCB

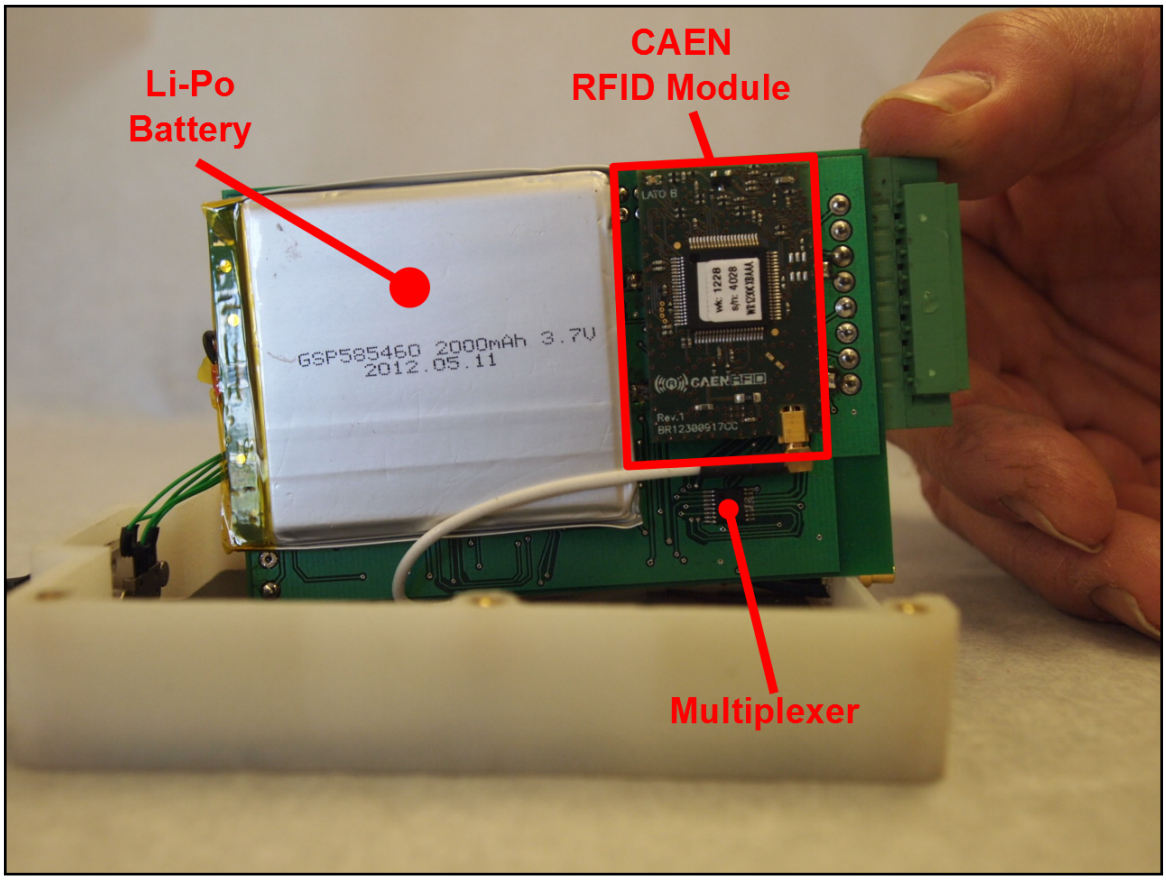


Figure C.3: Back side of v1.3 PCB

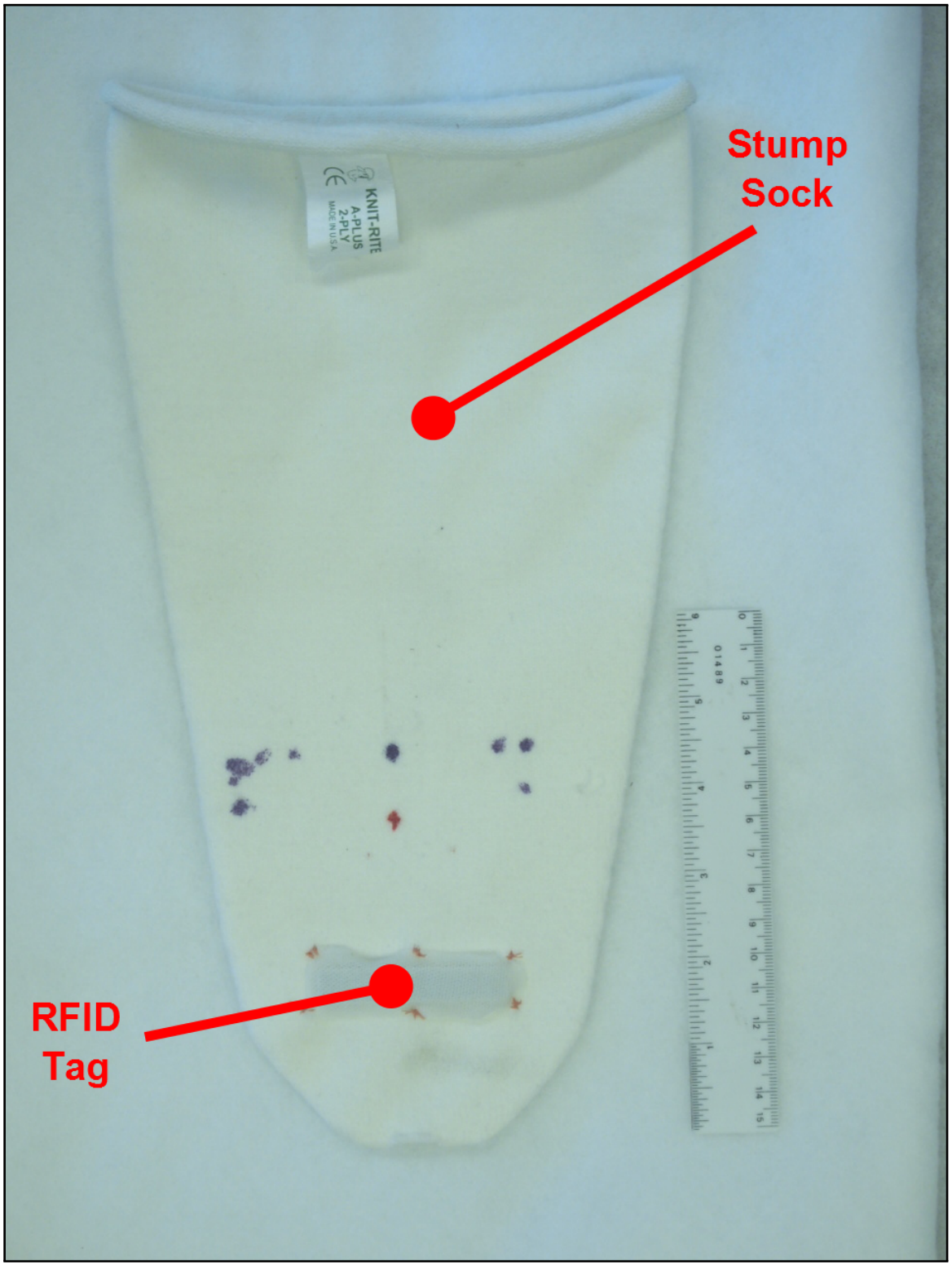


Figure C.4: Typical RFID tag setup on stump sock, bottom edge of tag approximately 4cm from doffed distal tip.

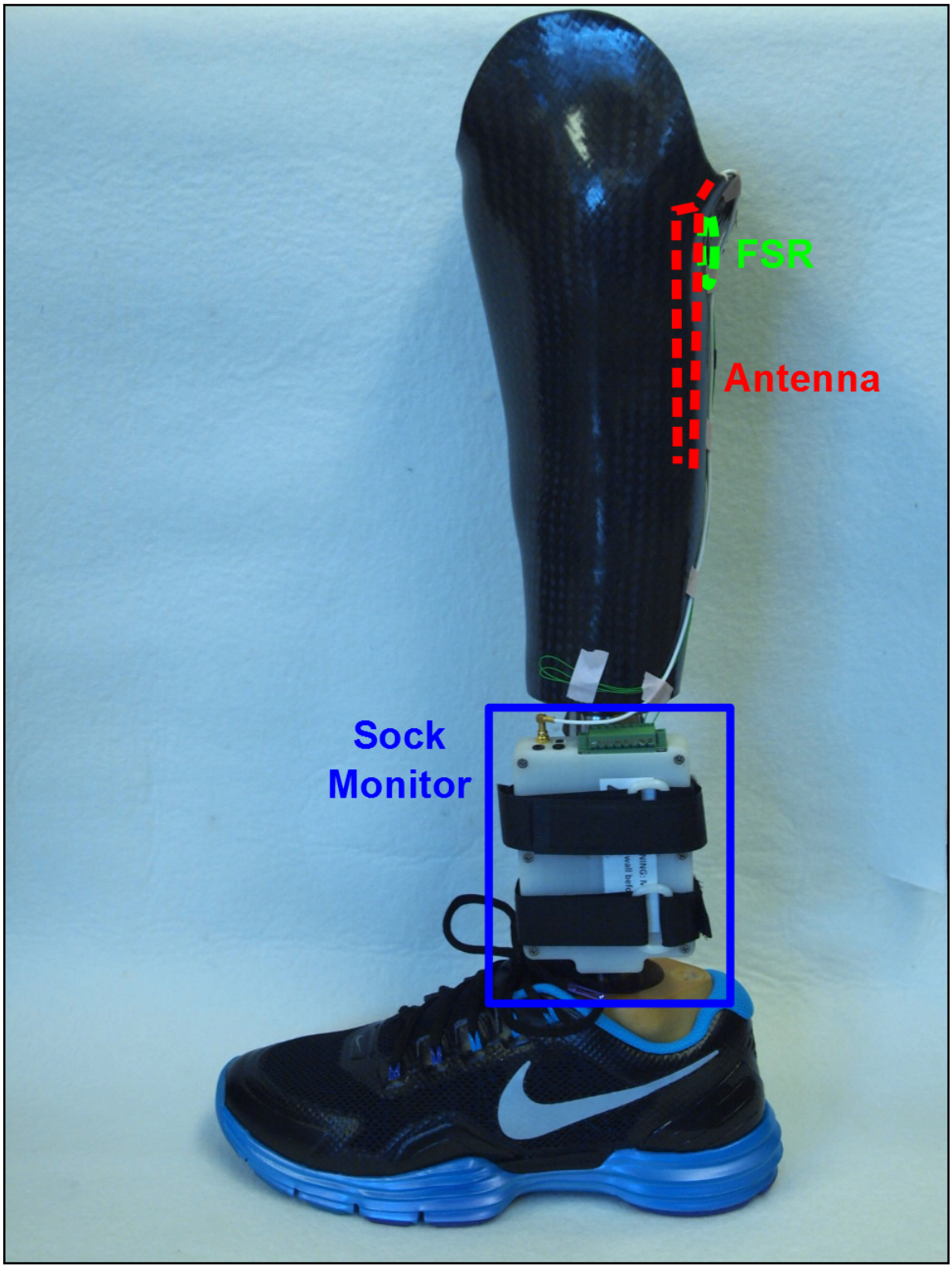


Figure C.5: Side view of Sock Monitor v1.3 on a prosthesis, note that the accelerometer is located inside the data logging unit mounted on the pylon.

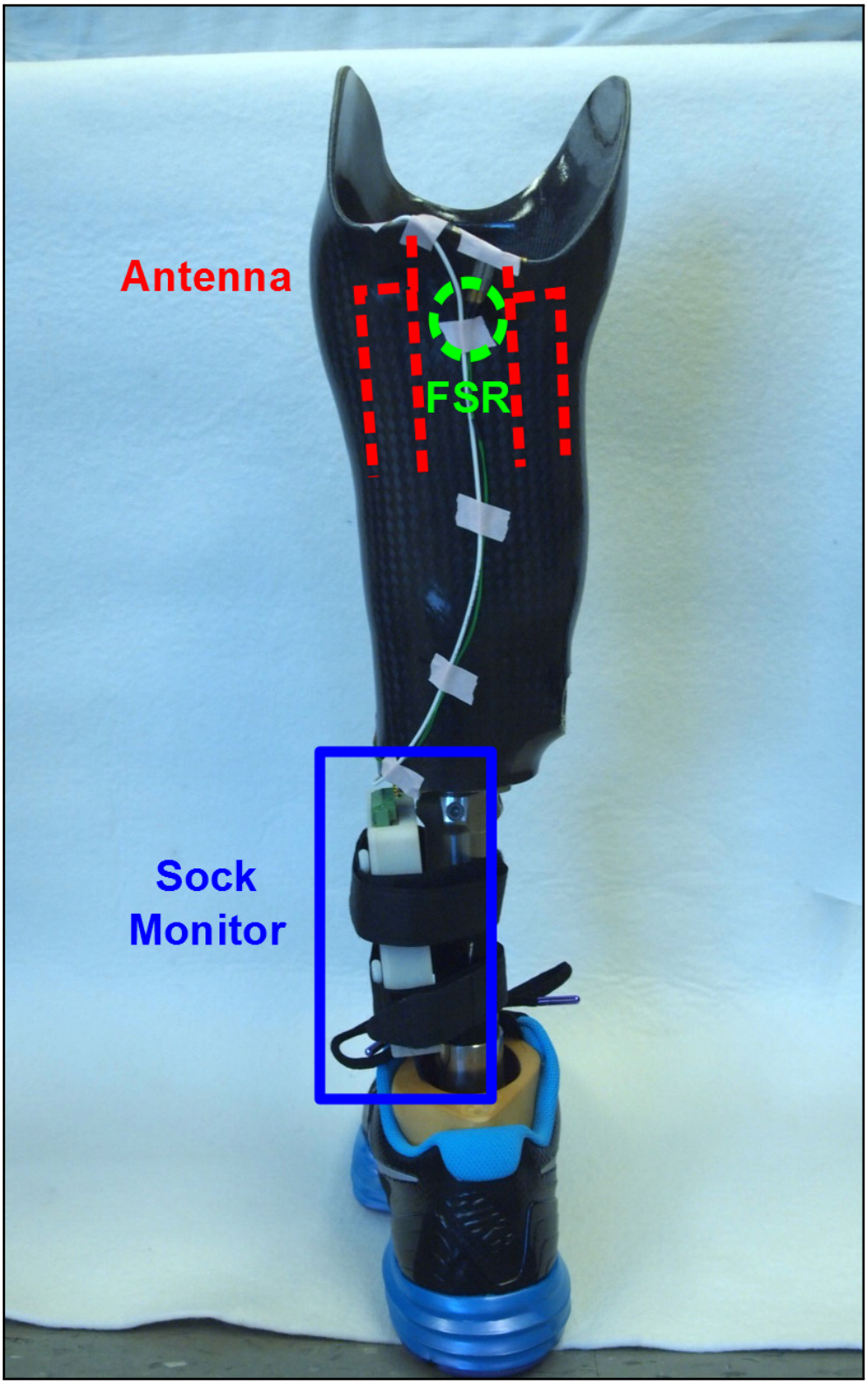


Figure C.6: Posterior view of Sock Monitor v1.3 on a prosthesis, note that the accelerometer is located inside the data logging unit mounted on the pylon.

Appendix D

SOCK MONITOR FIRMWARE

D.1 main.c

```
#include <msp430f5638.h>
#include <stdint.h>
#include <stdlib.h>
#include <intrinsics.h>
#include <spiutil.h>
#include <Initialize.h>
#include "NAND_driver.h"
#include <stdbool.h>
#include <SockMonitor.h>
#include <string.h>
#include <stdio.h>
#include "CAEN_driver.h"
#include "convertutil.h"
#include <math.h>


uint8_t ticTime = 0;                //Used in timing for sampling
uint8_t ledCt = 0;
uint8_t outBuf[BUFFSIZE];
int16_t accBuff[40];                //Ciruclar buffer for most recent Y-axis val
uint8_t accCt = 0;                  //accBuff counter;
uint16_t bufCt = 0;                 //Number of words recorded in buffer
uint32_t dataSize = 0;              //Number of words recoreded in memory
NANDReg datStat;                    //Recording register for buf -> NAND{page,block/plane,
    colAd,rowAd}
uint16_t errTmr = 0;
uint8_t doffTmr = 0;                //Doff timeout
uint8_t ulpTmr = 0;                 //Ultra-low power timer
uint8_t spiVar;                     //SPI variable in
uint8_t phonePayload[7];            //Smartphone BT payload buffer
uint8_t phoneCt = 0;                //Phone payload place counter
uint8_t oldSec = 0;                 //Previous second

//-------------------------------------------------------------------------//
//  Global Flags and Response strings
//-------------------------------------------------------------------------//
bool ersFl          = false;        //Erase flag
bool yesFl          = false;        //Yes flag
bool noFl           = false;        //No flag
bool sndFl          = false;        //Send flag
bool promptFl       = true;         //Prompt flag
bool lowBatFl       = false;        //Low battery flag
bool memFullFl      = false;        //Memory full flag
```

```
43 bool rwErrorFl        = false;       //Read/Write error
44 bool memBluFl         = false;       //Memory size request via Bluetooth
45 bool timBluFl         = false;       //Time menu request via Bluetooth
46 bool bBangRxFl        = false;       //Data received via bit-bang from CAEN
47 bool dataSampleFl     = false;       //FSR or Accelerometer record data flag
48 bool buttonFl         = false;       //Button press flag
49 bool caenInventoryFl = true;        //RFID inventory flag
50 bool gStartRecFl      = false;       //Start recording flag
51 bool gChgTimFl        = false;       //Change time flag
52 bool gDatChgFl        = false;       //Change Date flag
53 bool gInfo            = false;
54 bool timPrptFl        = true;        //Time prompt flag
55 bool pollState        = false;       //Are we currently polling?
56 bool fsrAct           = false;       //FSR activity flag, donned = T
57 bool accAct           = false;       //Accelerometer activity flag, movement = T
58 bool caenPwrFlg       = false;       //Used to toggle caen power on and off
59 bool doffOverRide     = false;       //Flag to trigger buffer to memory transfer
60 bool accPwrFlg        = false;       //Used to toggle accelerometer power
61 bool fsrPwrFlg        = false;       //Used to toggle FSR power
62 bool ulpFlg;                         //Currently in ULP mode?
63 bool btComFlg         = false;       //Bluetooth config flag
64 bool acc1Flg          = false;       //Accelerometer interrupt 1 flag
65 bool dateFlg          = false;       //Date stamp flag
66 bool phoneFlg         = false;       //Is smartphone app requesting Tx via BT?
67
68 volatile uint16_t CAENct;
69 uint16_t timeOutCt = 0;
70 unsigned char CAENin[1024];
71
72 CAENDat statReg;
73
74 /*****************************************************************************
75 * Obsolete command for tag polling, uses buffer and TX all at once
76 *****************************************************************************
77 static unsigned char tagPoll[]={0x80,0x01,0x00,0x00,0x00,0x00,0x53,0x58,0x00,
78 0x21,0x00,0x00,0x00,0x08,0x00,0x01,0x00,0x13,0x00,0x00,0x00,0x0F,0x00,0xFB,0x53,
79 0x6F,0x75,0x72,0x63,0x65,0x5F,0x30,0x00};
80 *****************************************************************************/
81 static unsigned char inventoryOn[] = {0x80,0x01,0x00,0x00,0x00,0x00,0x53,0x58,
82 0x00,0x29,0x00,0x00,0x00,0x08,0x00,0x01,0x00,0x13,0x00,0x00,0x00,0x0F,0x00,0xFB,
83 0x53,0x6F,0x75,0x72,0x63,0x65,0x5F,0x30,0x00,0x00,0x00,0x00,0x00,0x08,0x00,0x67,0x00,
84 0x0E};
85 static unsigned char inventoryOff[] = {0xAB};
86 const unsigned char welc[] = {"\033[2J\033[1;1HWelcome to the SockMonitor v1.3 Interface\r\n"
       \
87 "please select from the options below:\r\n\r\n" \
88 "d-Download Data\r\n" \
89 "e-Erase Data\r\n" \
90 "m-Memory Information\r\n" \
91 "t-Time Menu\r\n" \
92 "s-Start or end Data collection\r\n" \
93 "a-Toggle Status 1 LED\r\n" \
94 "b-Toggle Status 2 LED\r\n" \
95 "i-Current Device Information\n\r" \
```

```
"c–Toggle power to CAEN\r\n" \
"p–Toggle power to Accelerometer\n\r" \
"f–Toggle power to analog devices\n\r"};
const unsigned char sure[22] = {'A','r','e',' ','y','o','u',' ','s','u','r','e',
'?',' ','y',' ','o','r',' ','n',0x0D,0x0A};
const unsigned char goodErs[19] = {'E','r','a','s','e',' ','S','u','c','c','e',
's','s','f','u','l','!',0x0D,0x0A};
const unsigned char failErs[15] = {'E','r','a','s','e',' ','F','a','i','l','e',
'd','!',0x0D,0x0A};
const unsigned char canc[10] = {'C','a','n','c','e','l','e','d',0x0D,0x0A};
const unsigned char noDat[22] = {'N','o',' ','D','a','t','a',' ','t','o',' ','D'
,'o','w','n','l','o','a','d','!',0x0D,0x0A};
const unsigned char nowRec[] = "Now Recording...\n\r";
const unsigned char endRec[] = "Recording ending...\r\n";
const unsigned char timMen[] ={"\x1B[2J\x1B[1;1HPlease Select from the options below:\r\n" \
  "t – Change time (24 hour format)\r\n" \
  "d – Change date\r\n\r\n"};

//RTC_B variables — 24 hour clock
uint16_t millSec;
uint8_t Seconds;
uint8_t Minutes;
uint8_t Hours;
uint8_t Day;
uint8_t Month;
uint16_t Year;

/*=============================================================================
Variables used for debugging – Delete before final use
=============================================================================*/
bool testt = false;
/*=============================================================================*/

void main( void )
{
  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  Initialize();
  //Initialize dataSize


  //Verify battery is not low
  if((P4IN & BIT3) != BIT3)
  {
    lowBatFl = true;
    TA2CCR0 = TA2R + 52;
    TA2CCTL0 = CCIE;            //Turn on timer for LED blink
    P4IES &= ~BIT3;             //Switch edge direction of interrupt
  }

  while(1)
  {
    __bis_SR_register(LPM3);        //Enter low power mode 3 until interrupt
```

```
150
151     if(acc1Flg)                     //Handles waking up from ULP state
152     {
153       acc1Flg = false;
154
155       spiVar = SPI_read(0x30);   //Read cause of interrupt
156
157       if((spiVar & 0x10) != 0)
158       {
159         TB0CCTL1 |= CCIE;                         //Turn on Bit-bang FROM CAEN
160         CAEN_ON;
161         ANALOG_ON;
162         ulpFlg = false;
163         __delay_cycles(4000000);                  //Allows time for line to go high
164         BbangTx(CAENInit, sizeof(CAENInit));      //Enable continuous inventory
165         TA1CCTL1 |= CCIE;                         //Turn on doff polling timer
166         TA0CCTL0 |= CCIE;                         //Turn on sampling timer
167         TA2CCTL2 |= CCIE;                         //Turn on millSec timer
168       }
169     }
170
171     if((TA0CCTL0 & CCIE) != 0)        //Are we sampling the data?
172     {
173       DynamicPoll();                  //Determine polling state
174     }
175
176     //CAEN Action
177     if((bBangRxFl && (CAEN_STAT == C_ON)) || (bBangRxFl && ulpFlg))   //Decode CAEN message
178     {
179       CAEN_Decoder(CAENin, &statReg);
180       memset(CAENin,0,1024);
181       CAENct = 0;
182       bBangRxFl = false;
183     }
184
185     if(dataSampleFl)                  //Controls sampling of Data
186     {
187       DataSample();
188       dataSampleFl = false;
189     }
190
191     if(!MemCheck())
192     {
193       //Memory Action
194       if(bufCt > 498)                   //Push buffer to memory
195       {
196         DataPush();
197       }
198     }else
199       {
200         bhandle();                      //Added for data retrieval after full mem
201         __bis_SR_register(LPM4);
202       }
203
```

```
204     bhandle();                          //Bluetooth Handling
205   }
206 }
207
208 //-------------------------------------------------------------------------//
209 //       Name: Timer A0 ISR
210 //   Function: Polls Accelerometer every 1/50 of a second and polls FSR every
211 //             1/25 seconds
212 //     Inputs: N/A
213 //    Outputs: N/A
214 //
215 //-------------------------------------------------------------------------//
216 #pragma vector=TIMER0_A0_VECTOR
217 __interrupt void TIMER0_A0_ISR(void)
218 {
219   dataSampleFl = true;
220   __bis_SR_register(LPM3_EXIT);
221 }
222
223 //-------------------------------------------------------------------------//
224 //       Name: Timer A1 ISR
225 //   Function: This function illuminates the green Status 1 LED for 1/8 second
226 //             if there is a tag being read.
227 //     Inputs: N/A
228 //    Outputs: N/A
229 //
230 //-------------------------------------------------------------------------//
231 #pragma vector=TIMER1_A0_VECTOR
232 __interrupt void TIMER1_A0_ISR(void)
233 {
234   if(SAMP_STAT == CCIE)                 //Prevents STAT1 from being turnned off during when
        not sampling
235   {
236     TA1CCTL0 &= ~(CCIFG);                 //Clear flag
237     STAT1_OFF;                          //Turn off Status-1 LED (Green)
238   }
239
240   /*
241   if(timeOutCt > POLL_TIMEOUT)     //Disable polling after TIMEOUT seconds
242   {
243     BbangTx(inventoryOff,sizeof(inventoryOff));       //Turn off polling
244     pollState = false;
245     timeOutCt = 0;                      //Reset timer value
246   }
247   timeOutCt++;
248   */
249 }
250
251 //-------------------------------------------------------------------------//
252 //       Name: Timer A1 ISR, Channels 1 - 6
253 //   Function: Dynamic polling - doffing timeout and recording activation. When
254 //             the amputee doffs their socket, this timer is initialized. When
255 //             this ISR fires, it triggers a flag to allow the current tags to
256 //             be recorded in memory.
```

```
257 //
258 //            CCR2 is used to determine if the Sock Monitor should enter into
259 //            an ultra-low power state due to no use for at least 1 minute.
260 //
261 //     Inputs: N/A
262 //    Outputs: N/A
263 //
264 //------------------------------------------------------------------------------//
265 #pragma vector=TIMER1_A1_VECTOR
266 __interrupt void TIMER1_A1_ISR(void)
267 {
268   switch(__even_in_range(TA1IV,12))
269   {
270   case 0:  break;                   //No interrupt
271   case 2:                           //CCR 1
272     TA1CCTL1 &= ~CCIFG;             //Clear flag
273     if(doffTmr >= 3)
274     {
275       doffOverRide = true;
276       doffTmr = 0;
277       TA1CCR1 = TA1R + DOFFCONST;
278     }else
279       {
280            doffTmr++;
281            TA1CCR1 = TA1R + DOFFCONST;
282       }
283     break;
284   case 4:                           //CCR 2, used in ultra-low power mode
285     TA1CCTL2 &= ~CCIFG;             //Clear flag
286     if(ulpTmr>=30)
287     {
288       ulpFlg = true;
289       ulpTmr = 0;
290       TA1CCTL2 &= ~CCIE;
291     }else
292       {
293         ulpTmr++;
294         TA1CCR2 = TA1R + SECOND;
295       }
296
297     break;
298   case 6:  break;                   //CCR 3
299   case 8:  break;                   //CCR 4
300   case 10: break;                   //CCR 5
301   case 12: break;                   //CCR 6
302   default: break;
303   }
304 }
305
306 //------------------------------------------------------------------------------//
307 //       Name: Timer A2, channel 0 ISR
308 //   Function: Error handling, mainly for blinking of status LEDs
309 //     Inputs: N/A
310 //    Outputs: N/A
```

```
311 //
312 //-------------------------------------------------------------------------//
313 #pragma vector=TIMER2_A0_VECTOR
314 __interrupt void TIMER2_A0_ISR(void)
315 {
316   if(errTmr >= 4808) //About 1/8th of a second
317   {
318     //STAT2_TOGGLE;
319     errTmr = 0;
320   }
321   errTmr++;
322   //Add in Read/write error
323
324   //Add in Memory Full error
325 }
326
327 //-------------------------------------------------------------------------//
328 //       Name: Timer A2, channel 1 ISR
329 //   Function: Bit-bang RX end timer, activates CAEN_Decode via flag
330 //     Inputs: N/A
331 //    Outputs: N/A
332 //
333 //-------------------------------------------------------------------------//
334 #pragma vector=TIMER2_A1_VECTOR
335 __interrupt void TIMER2_A1_ISR(void)
336 {
337   switch(__even_in_range(TA2IV,12))
338   {
339   case 0:  break;                   //No interrupt
340   case 2:                           //CCR1
341     TA2CCTL1 &= ~CCIE;                      //Turn off timer
342     TA2CCTL1 &= ~CCIFG;                     //Clear interrupt flag
343     bBangRxFl = true;                       //Set decode flag
344     __bis_SR_register(LPM3_EXIT);           //Exit to allow for flag servicing
345     break;
346   case 4:                          //CCR2
347     TA2CCTL2 &= ~CCIFG;
348     TA2CCR2 += 16000;
349     if(millSec<1000)
350     {
351       millSec++;
352     }else
353       {
354         millSec = 0;
355       }
356     break;
357   case 6:  break;                   //CCR3
358   case 8:  break;                   //CCR4
359   case 10:  break;                  //CCR5
360   case 12:  break;                  //CCR6
361   default: break;                   //Error
362   }
363
364 }
```

```
365
366 //-------------------------------------------------------------------------//
367 //       Name: Bluetooth ISR
368 //   Function: Control flow based on user input
369 //     Inputs: N/A
370 //    Outputs: N/A
371 //
372 //-------------------------------------------------------------------------//
373 #pragma vector=USCIA1_VECTOR
374 __interrupt void USCI_A1 (void)
375 {
376   switch(__even_in_range(UCA1IV,4))
377   {
378   case 0: break;
379   case 2:                           //Data Received
380       switch(UCA1RXBUF)
381       {
382       case 'd':                             //Send data (download data)
383         if(timBluFl)
384         {
385           gDatChgFl = true;
386         }else
387           {
388             sndFl = true;
389           }
390         break;
391
392       case 'e':                             //Erase flash
393         ersFl = true;
394         break;
395
396       case 'y':                             //Yes
397         if(sndFl || ersFl || gStartRecFl || caenPwrFlg || accPwrFlg || fsrPwrFlg)
398         {
399           yesFl = true;
400           break;
401         }
402         break;
403
404       case 'n':                             //No
405         if(sndFl || ersFl || gStartRecFl || caenPwrFlg || accPwrFlg || fsrPwrFlg)
406         {
407           noFl = true;
408           break;
409         }
410         break;
411
412       case 'm':
413         memBluFl = true;
414         break;
415
416       case 't':
417         if(timBluFl)
418         {
```

```
419             gChgTimFl = true;
420           }
421           timBluFl = true;
422           break;
423
424         case 's':
425           gStartRecFl = true;
426           break;
427
428         case 'a':
429           STAT1_TOGGLE;
430           break;
431
432         case 'b':
433           STAT2_TOGGLE;
434           break;
435
436         case 'i':
437           gInfo = true;
438           break;
439
440         case 'c':
441           caenPwrFlg = true;
442           break;
443
444         case 'p':
445           accPwrFlg = true;
446           break;
447
448         case 'f':
449           fsrPwrFlg = true;
450           break;
451
452         case '!':
453             btComFlg = true;
454             break;
455
456         default:
457           for(int i=0;i<sizeof(welc);i++)
458           {
459             while(!(UCA1IFG&UCTXIFG)){};
460             UCA1TXBUF = welc[i];
461           }
462         }
463
464       UCA1IFG &= ~UCRXIFG;
465       __bis_SR_register(LPM3_EXIT);
466       break;
467
468     case 4:                    //Transmit Empty
469       UCA1TXBUF = phonePayload[phoneCt];
470       phoneCt++;
471       if(phoneCt>7)
472       {
```

```
473         UCA1IE &= ~UCTXIE;                        //Disable Tx ISR
474       }
475       break;
476     }
477 }
478
479
480 //-------------------------------------------------------------------------//
481 //        Name: Real Time Clock (RTC) ISR
482 //   Function: Increment RTC at appropriate intervals
483 //     Inputs: N/A
484 //    Outputs: N/A
485 //
486 //-------------------------------------------------------------------------//
487 #pragma vector=RTC_VECTOR
488 __interrupt void RTCISR (void)
489 {
490   switch(__even_in_range(RTCIV,14))
491   {
492   case  0: break;                       // Vector  0:  No interrupt
493   case  2:                              // Vector  2:  RTCRDYIFG
494     oldSec = Seconds;
495     Seconds = RTCSEC;                   // Read all associated time registers
496     Minutes = RTCMIN;
497     Hours = RTCHOUR;
498     Day = RTCDAY;
499     Month = RTCMON;
500     Year = RTCYEAR;
501
502     if(Seconds != oldSec)
503     {
504       millSec = 0;
505     }
506     break;
507   case  4: break;                       // Vector  4:  RTCEVIFG
508   case  6:                              // Vector  6:  RTCAIFG
509     if(((TA0CCTL0 & CCIE) != 0) || ulpFlg)
510     {
511       dateFlg = true;
512     }
513
514     break;
515   case  8: break;                       // Vector  8:  RT0PSIFG
516   case 10: break;                       // Vector 10:  RT1PSIFG
517   case 12: break;                       // Vector 12:  RTCOFIFG
518   case 14: break;                       // Vector 14:  Reserved
519   default: break;
520   }
521 }
522
523 //-------------------------------------------------------------------------//
524 //        Name: Port 2 ISR
525 //   Function: When button (BIT3) is pressed, perform some action
526 //     Inputs: N/A
```

```
527 //    Outputs: N/A
528 //
529 //-------------------------------------------------------------------------//
530 #pragma vector=PORT2_VECTOR
531 __interrupt void PORT2_ISR(void)
532 {
533   P2IFG=0;
534 }
535
536 //-------------------------------------------------------------------------//
537 //       Name: Port 4 ISR
538 //   Function: Bluetooth, CAEN, lowbat, and accel interrupt
539 //     Inputs: N/A
540 //    Outputs: N/A
541 //
542 //-------------------------------------------------------------------------//
543 #pragma vector=PORT4_VECTOR
544 __interrupt void PORT4_ISR(void)
545 {
546   switch(P4IV)
547   {
548   case 0x02:
549     P4IFG &= ~BIT0;
550     break;                              // Vector  0:  Port 4.0 Interrupt
551
552   case 0x04:                            // Vector  2:  FROM CAEN (Bit Bang)
553     P4IFG &= ~BIT1;
554
555     if(CAENct < 1024)
556     {
557       CAENin[CAENct] = ReadRXBUF();
558       CAENct++;
559     }
560     break;
561
562   case 0x06:
563     P4IFG &= ~BIT2;
564     break;                      // Vector  4:  Port 4.2 Interrupt
565   case 0x08:                            // Vector  6:  Low Battery Interrupt
566     if((P4IES & BIT3) == BIT3)
567     {   //If Battery is low
568       lowBatFl = true;
569       TA2CCR0 = TA2R + 52;
570       TA2CCTL0 = CCIE;            //Turn on timer for LED blink
571       P4IES &= ~BIT3;             //Switch edge direction of interrupt
572     }else
573       {   //If Battery is charged
574         lowBatFl = false;
575         TA2CCTL0 &= ~CCIE;        //Turn off timer blink
576         STAT2_OFF;                //Make sure LED is off
577         P4IES |= BIT3;            //Switch edge direction of interrupt
578       }
579
580     P4IFG &= ~BIT3;
```

```
581       break;
582
583     case 0x0A:
584       P4IFG &= ~BIT4;
585       break;                      // Vector  8:  Port 4.4 Interrupt
586
587     case 0x0C:
588       P4IFG &= ~BIT5;
589       break;                      // Vector 10:  Port 4.5 Interrupt
590
591     case 0x0E:
592       P4IFG &= ~BIT6;
593       break;                      // Vector 12:  Interrupt 2 from Accelerometer
594
595     case 0x10:                    // Vector 14:  Interrupt 1 from Accelerometer
596       P4IFG &= ~BIT7;
597       P4IE &= ~BIT7;
598       SPI_write(0x2E,0x00);
599       acc1Flg = true;
600       __bis_SR_register(LPM3_EXIT);
601       break;
602
603     default: break;
604     }
605 }
606
607 //------------------------------------------------------------------------------//
608 //       Name: Bluetooth Handling algorithm
609 //   Function: Handles I/O with BT
610 //     Inputs: N/A
611 //    Outputs: N/A
612 //
613 //------------------------------------------------------------------------------//
614 void bhandle()
615 {
616   uint8_t stat = 0;
617   bool statFl = true;
618   unsigned char bytSize[10] = {0};
619
620   if(sndFl)                                 //Get Data from device
621     {
622       if(!yesFl && !noFl && promptFl)
623       {
624         promptFl = false;
625         for(int i=0;i<22;i++)
626         {
627           while(!(UCA1IFG&UCTXIFG)){};
628           UCA1TXBUF = sure[i];
629         }
630       }
631       if(yesFl && (dataSize > 0))
632       {
633         NAND_DataDump(dataSize,&datStat);
634         yesFl = false;                  //Clear yes flag
```

```
635         sndFl = false;                  //Clear data dump flag
636         promptFl = true;                //It is TRUE that the prompt should be displayed
637       }else if(noFl)
638         {
639           noFl = false;                   //Clear no flag
640           sndFl = false;                  //Clear send flag
641           promptFl = true;                //It is TRUE to prompt the user again
642           for(int i=0;i<10;i++)           //Display Cancel message
643             {
644               while(!(UCA1IFG&UCTXIFG)){};
645             UCA1TXBUF = canc[i];
646             }
647         }else if(dataSize == 0)
648           {
649             sndFl = false;
650             yesFl = false;
651             promptFl = true;
652             for(int i=0;i<22;i++)         //Display No data message
653             {
654               while(!(UCA1IFG&UCTXIFG)){};
655             UCA1TXBUF = noDat[i];
656             }
657           }
658     }
659     //Erase Task
660     if(ersFl)
661     {
662       if(!yesFl && !noFl && promptFl)
663       {
664         promptFl = false;
665         for(int i=0;i<22;i++)
666         {
667           while(!(UCA1IFG&UCTXIFG)){};
668           UCA1TXBUF = sure[i];
669         }
670       }
671       if(yesFl)
672       {
673         __bic_SR_register(GIE);         //Disable interrupts
674         for(uint16_t k=0;k<NUMBLOCKS;k++)  //Erase all 1024 blocks over both planes
675         {
676           stat = NAND_EraseBlock((k&BLK1_MSK)<<BLK1_SFT,(k&BLK2_MSK)>>BLK2_SFT,(k&BLK3_MSK)>>
    BLK3_SFT);
677           if((stat & BIT0) != 0)      //If any erase fails, set flag to false
678           {
679             statFl = false;
680           }
681         }
682         __bis_SR_register(GIE);       //Enable interrupts
683         if(statFl)                    //If good erase...
684         {
685           for(int i=0;i<19;i++)
686           {
687             while(!(UCA1IFG&UCTXIFG)){};
```

```
688             UCA1TXBUF = goodErs[i];
689           }
690          yesFl = false;
691          ersFl = false;
692          memFullFl = false;
693          datStat.page = 0;
694          datStat.block = 0;
695          datStat.colAd = 0;
696          datStat.rowAd = 0;
697          memset(outBuf,0,1024);
698          bufCt = 0;
699          dataSize = 0;
700          promptFl = true;
701         }else                          //Else if bad erase...
702         {
703           for(int i=0;i<15;i++)
704           {
705             while(!(UCA1IFG&UCTXIFG)){};
706             UCA1TXBUF = failErs[i];
707           }
708          yesFl = false;
709          ersFl = false;
710          //Reset memory address because you aren't sure where the error was
711          datStat.page = 0;
712          datStat.block = 0;
713          datStat.colAd = 0;
714          datStat.rowAd = 0;
715          memset(outBuf,0,1024);
716          bufCt = 0;
717          dataSize = 0;
718          promptFl = true;
719         }
720       }else if(noFl)
721       {
722         noFl = false;
723         ersFl = false;
724         promptFl = true;
725         for(int i=0;i<10;i++)
726           {
727             while(!(UCA1IFG&UCTXIFG)){};
728             UCA1TXBUF = canc[i];
729           }
730       }
731     }
732
733   //Memory information
734   if(memBluFl)
735   {
736     unsigned char memBluDis[] ="Your memory consumption is ";
737     memBluFl = false;
738     for(int i=0;i<sizeof(memBluDis);i++)
739       {
740         while(!(UCA1IFG&UCTXIFG)){};
741         UCA1TXBUF = memBluDis[i];
```

```
742         }
743
744       inttochar(dataSize,bytSize,sizeof(dataSize));
745
746       for(int i=0;i<sizeof(bytSize);i++)
747         {
748           while(!(UCA1IFG&UCTXIFG)){};
749           UCA1TXBUF = bytSize[i];
750         }
751
752         while(!(UCA1IFG&UCTXIFG)){};
753         UCA1TXBUF = 0x0D;
754
755     }
756
757     //Time menu
758     if(timBluFl)
759     {
760       if(timPrptFl)
761       {
762         PrintMess(timMen,sizeof(timMen));
763         timPrptFl = false;
764       }
765
766       if(gChgTimFl)
767       {
768         ChangeTime();
769         timBluFl = false;
770         gChgTimFl = false;
771         timPrptFl = true;
772       }else if(gDatChgFl)
773         {
774           ChangeDate();
775           timBluFl = false;
776           gDatChgFl = false;
777           timPrptFl = true;
778         }
779
780     }
781
782     //Start or end recording
783     if(gStartRecFl)
784     {
785       if(!yesFl && !noFl && promptFl)
786         {
787           promptFl = false;
788           for(int i=0;i<22;i++)
789           {
790             while(!(UCA1IFG&UCTXIFG)){};
791             UCA1TXBUF = sure[i];
792           }
793         }
794       if(yesFl)
795       {
```

```
796         gStartRecFl = false;
797         yesFl = false;                  //Clear yes flag
798         promptFl = true;                //It is TRUE that the prompt should be displayed
799         if((TA0CCTL0 & CCIE) != CCIE)
800         {
801           if(ulpFlg)
802           {
803             goto ulp_override;      //If in ULP mode, then turn stop recording
804           }
805           for(int i=0;i<sizeof(nowRec);i++)
806             {
807               while(!(UCA1IFG&UCTXIFG)){};
808               UCA1TXBUF = nowRec[i];
809             }
810
811           //Add in date stamp at start
812           for(int i=0;i<sizeof(dateStp)-1;i++)
813             {
814               outBuf[bufCt] = dateStp[i];
815               bufCt++;
816             }
817             outBuf[bufCt] = Day;
818             bufCt++;
819             outBuf[bufCt] = Month;
820             bufCt++;
821             outBuf[bufCt] = Year >> 8;
822             bufCt++;
823             outBuf[bufCt] = (uint8_t)Year;
824             bufCt++;
825             outBuf[bufCt] = 0x0D;
826             bufCt++;
827             outBuf[bufCt] = 0x0A;
828             bufCt++;
829
830           TA0CCTL0 |= CCIE;             //Start timer for data collection
831           TA2CCTL2 |= CCIE;             //Turn on millSec timer
832         }else
833           {
834           ulp_override:
835             ulpFlg = false;
836             ulpTmr = 0;
837             TA1CCTL2 &= ~CCIE;                  //Turn off timer for ULP Mode
838             BbangTx(inventoryOff,sizeof(inventoryOff));        //Turn off polling
839             pollState = false;
840             TA0CCTL0 &= ~CCIE;              //End timer for data collection
841             TA2CCTL2 &= ~CCIE;              //End timer for millSec
842             for(int i=0;i<sizeof(endRec);i++)
843             {
844               while(!(UCA1IFG&UCTXIFG)){};
845               UCA1TXBUF = endRec[i];
846             }
847           }
848       }else if(noFl)
849         {
```

```
850         gStartRecFl = false;
851         noFl = false;                   //Clear no flag
852         promptFl = true;                //It is TRUE to prompt the user again
853         for(int i=0;i<10;i++)           //Display Cancel message
854           {
855             while(!(UCA1IFG&UCTXIFG)){};
856            UCA1TXBUF = canc[i];
857           }
858       }
859   }
860
861   //Display device information
862   if(gInfo)
863   {
864     GetInfo(Hours,Minutes,Seconds);
865     gInfo = false;
866   }
867
868   //Toggle CAEN power
869   if(caenPwrFlg)
870   {
871     if(!yesFl && !noFl && promptFl)
872       {
873         promptFl = false;
874         for(int i=0;i<22;i++)
875         {
876           while(!(UCA1IFG&UCTXIFG)){};
877           UCA1TXBUF = sure[i];
878         }
879       }
880     if(yesFl)
881     {
882       caenPwrFlg = false;
883       gInfo = true;
884       yesFl = false;
885       promptFl = true;
886       if((P2OUT & BIT0) == 0)               //CAEN is on and will be turned off
887       {
888         CAEN_OFF;
889       }else                                 //CAEN is off and will be turned on
890         {
891           CAEN_ON;
892           __delay_cycles(4000000);
893           BbangTx(CAENInit,sizeof(CAENInit));    //Enable continuous inventory
894           TA2CCTL1 &= ~CCIFG;               //Clear to deal with bug
895           bBangRxFl = false;                //Reset flag to deal with bug
896           gInfo = true;
897           yesFl = false;
898         }
899     }else if(noFl)
900     {
901       noFl = false;                 //Clear no flag
902       promptFl = true;              //It is TRUE to prompt the user again
903       caenPwrFlg = false;           //Clear CAEN Power flag
```

```
904         for(int i=0;i<10;i++)         //Display Cancel message
905           {
906             while(!(UCA1IFG&UCTXIFG)){};
907             UCA1TXBUF = canc[i];
908           }
909       }
910
911     }
912
913     //Toggle accelerometer power
914     if(accPwrFlg)
915     {
916       if(!yesFl && !noFl && promptFl)
917         {
918           promptFl = false;
919           for(int i=0;i<22;i++)
920           {
921             while(!(UCA1IFG&UCTXIFG)){};
922             UCA1TXBUF = sure[i];
923           }
924         }
925
926       if(yesFl)
927       {
928         accPwrFlg = false;
929         promptFl = true;
930         yesFl = false;
931         if((P6OUT & BIT6) == 0)
932         {                             //Accelerometer is on, need to turn off
933           ACCELSTANDBY;
934         }else                         //Accelerometer is off, need to turn on
935           {
936             ACCELON;
937             for(int j=0;j<ACCDEF;j++)
938               {
939                 SPI_write(accAd[j],accDa[j]);
940               }
941           }
942
943       }else if(noFl)
944         {
945           noFl = false;               //Clear no flag
946           promptFl = true;            //It is TRUE to prompt the user again
947           accPwrFlg = false;          //Clear accelerometer Power flag
948           for(int i=0;i<10;i++)       //Display Cancel message
949             {
950               while(!(UCA1IFG&UCTXIFG)){};
951               UCA1TXBUF = canc[i];
952             }
953         }
954     }
955
956     //Toggle FSR power
957     if(fsrPwrFlg)
```

```
958   {
959     if(!yesFl && !noFl && promptFl)
960       {
961         promptFl = false;
962         for(int i=0;i<22;i++)
963         {
964           while(!(UCA1IFG&UCTXIFG)){};
965           UCA1TXBUF = sure[i];
966         }
967       }
968
969     if(yesFl)
970     {
971       fsrPwrFlg = false;
972       promptFl = true;
973       yesFl = false;
974       if((P6OUT & BIT4) == 0)
975       {                                 //FSR is on, need to turn off
976         ANALOG_OFF;
977       }else                             //FSR is off, need to turn on
978         {
979           ANALOG_ON;
980         }
981
982     }else if(noFl)
983       {
984         noFl = false;                   //Clear no flag
985         promptFl = true;                //It is TRUE to prompt the user again
986         fsrPwrFlg = false;              //Clear accelerometer Power flag
987         for(int i=0;i<10;i++)           //Display Cancel message
988           {
989             while(!(UCA1IFG&UCTXIFG)){};
990             UCA1TXBUF = canc[i];
991           }
992       }
993   }
994 }
995
996 //-------------------------------------------------------------------------//
997 //       Name: MemCheck
998 //   Function: Checks if NAND Memory is full, puts system in ultra low power if
999 //             it is full.
1000 //     Inputs: N/A
1001 //    Outputs: N/A
1002 //
1003 //-------------------------------------------------------------------------//
1004 bool MemCheck()
1005 {
1006   if(dataSize > 0x77359400) //2GigaBytes
1007     {
1008       TA0CCTL0 &= ~CCIE;
1009       TA2CCTL2 &= ~CCIE;            //End timer for millSec
1010       memFullFl = true;
1011       return true;
```

```
1012      }else
1013        {
1014          return false;
1015        }
1016  }
1017
1018  //-------------------------------------------------------------------------//
1019  //       Name: DataPush
1020  //   Function: Transfers data from buffer to NAND Flash
1021  //     Inputs: N/A
1022  //    Outputs: N/A
1023  //
1024  //-------------------------------------------------------------------------//
1025  void DataPush()
1026  {
1027    NAND_PushData(bufCt,&datStat,outBuf);
1028    memset(outBuf,0x00,sizeof(outBuf));         //Set buffer to all zeros
1029
1030    dataSize += bufCt;                        //Increment memory counter, every count is 2 bytes
1031    bufCt = 0;                                  //Reset buffer counter
1032  }
1033
1034  //-------------------------------------------------------------------------//
1035  //       Name: DataSample
1036  //   Function: Records data from FSR and Accelerometer based on flag from ISR,
1037  //             stores data in global buffer.
1038  //     Inputs: N/A
1039  //    Outputs: N/A
1040  //
1041  //-------------------------------------------------------------------------//
1042  void DataSample()
1043  {
1044    uint16_t wordVar;
1045    int32_t sum = 0;                          //Counter for average
1046    int16_t avg = 0;                          //Computed average
1047    int16_t max = -4095;                      //Computed maximum acceleration
1048    int16_t min = 4096;                       //Computed minimum acceleration
1049    uint8_t fsrSize = 0;                      //Size of FSR recording in bytes
1050    uint8_t tagSize = 0;                      //Size of tag recording in bytes
1051    phoneCt = 0;
1052
1053    if(dateFlg)                               //Add date stamp at 12:01am every day
1054    {
1055      dateFlg = false;
1056      for(int i=0;i<sizeof(dateStp)-1;i++) //The "-1" accounts for the null
1057      {
1058        outBuf[bufCt] = dateStp[i];
1059        bufCt++;
1060      }
1061      outBuf[bufCt] = Day;
1062      bufCt++;
1063      outBuf[bufCt] = Month;
1064      bufCt++;
1065      outBuf[bufCt] = Year >> 8;
```

```
1066      bufCt++;
1067      outBuf[bufCt] = (uint8_t)Year;
1068      bufCt++;
1069      outBuf[bufCt] = 0x0D;
1070      bufCt++;
1071      outBuf[bufCt] = 0x0A;
1072      bufCt++;
1073    }
1074
1075    //Record Timestamp
1076    outBuf[bufCt] = Hours;
1077    phonePayload[phoneCt] = bufCt;
1078    phoneCt++;
1079    bufCt++;
1080    outBuf[bufCt] = 0x3A;
1081    bufCt++;
1082    outBuf[bufCt] = Minutes;
1083    phonePayload[phoneCt] = bufCt;
1084    phoneCt++;
1085    bufCt++;
1086    outBuf[bufCt] = 0x3A;
1087    bufCt++;
1088    outBuf[bufCt] = Seconds;
1089    phonePayload[phoneCt] = bufCt;
1090    phoneCt++;
1091    bufCt++;
1092    outBuf[bufCt] = '.';
1093    bufCt++;
1094    outBuf[bufCt] = millSec >> 8;
1095    phonePayload[phoneCt] = bufCt;
1096    phoneCt++;
1097    bufCt++;
1098    outBuf[bufCt] = (uint8_t)millSec;
1099    phonePayload[phoneCt] = bufCt;
1100    phoneCt++;
1101    bufCt++;
1102
1103    //Calculate recording size, first if is for FSR and second is for tags
1104    if(!(ticTime%2))
1105    {
1106      fsrSize = 8;
1107    }else
1108      {
1109        fsrSize = 0;
1110      }
1111    if((statReg.BUFCT > 0 && !pollState) || (statReg.BUFCT > 0 && doffOverRide))
1112    {
1113      tagSize = 12 * statReg.BUFCT;
1114    }else
1115      {
1116        tagSize = 0;
1117      }
1118
1119    outBuf[bufCt] = 0x06 + fsrSize + tagSize;
```

```
1120      bufCt++;
1121
1122    //Poll Accelerometer
1123    SPI_ON;                                 //Unlatch SPI
1124    while (!(UCB1IFG & UCTXIFG));           //Make sure that it is ok to transmit
1125    UCB1TXBUF = 0xF2;                       //BITs 7 and 6 are added for multi write and read
1126
1127    for(int j=0;j<6;j++)
1128    {
1129      while (!(UCB1IFG&UCTXIFG));           //Make sure that it is ok to transmit
1130      UCB1TXBUF = 0x00;                     //Send junk data
1131      while(UCB1STAT&UCBUSY);               //Make sure all SPI is finished
1132      if(j%2)
1133      { //Starts second
1134        outBuf[bufCt-1] = UCB1RXBUF;          //Concatenate MSB and LSB
1135        bufCt++;
1136
1137        if(j == X_AXIS)                     //Capture x-axis values
1138        {
1139          accBuff[accCt] = outBuf[bufCt-2]<<8 | outBuf[bufCt-1];
1140          if(accCt >= 39)                   //If statment makes buffer circular
1141          {
1142            accCt = 0;                      //Reset counter (pointer)
1143          }else
1144            {
1145              accCt++;                      //Increment position in buffer
1146            }
1147        }
1148      }else //Starts 1st
1149        {
1150          outBuf[bufCt+1] = UCB1RXBUF;              //Store LSB
1151          bufCt++;
1152        }
1153    }
1154    SPI_OFF;
1155
1156    //Poll FSR
1157    if(!(ticTime%2))
1158    {
1159      for(int i=0;i<4;i++)
1160      {
1161        /*********************************************************************
1162        Need to fix this area. Currently the same value is being written into 4
1163        separate areas in memory meant for FSRs 1-4 but instead is just FSR 1.
1164        *********************************************************************/
1165        //P1OUT = (P1OUT & ~14) + (analogMsk[i] << 1);
1166        ADC12CTL0 |= ADC12SC;
1167        wordVar = ADC12MEM0;
1168        //ADC12CTL0 &= ~ADC12SC;
1169        outBuf[bufCt] = wordVar>>8;
1170        bufCt++;
1171        outBuf[bufCt] = (uint8_t)wordVar;
1172        bufCt++;
1173
```

```
1174        if(i == 0)                                  //If measuring the 1st FSR
1175        {
1176          phonePayload[phoneCt] = bufCt - 2;
1177          phoneCt++;
1178          phonePayload[phoneCt] = bufCt - 1;
1179          phoneCt = 0;
1180          if(phoneFlg)
1181          {
1182            UCA1IE |= UCTXIE;                       //Enable Tx ISR
1183          }
1184
1185          if(wordVar < FSRTHRESH)
1186          {
1187            fsrAct = false;                         //Socket is doffed
1188          }else
1189            {
1190              fsrAct = true;                        //Socket is donned
1191            }
1192        }
1193      }
1194    }
1195
1196    //Add CAEN data
1197    if((statReg.BUFCT > 0 && !pollState) || (statReg.BUFCT > 0 && doffOverRide))
1198    {
1199      doffOverRide = false;                         //Disable override for doffing
1200
1201      for(int j=0;j<statReg.BUFCT;j++)              //For number of new IDs
1202      {
1203          for(int k=0;k<12;k++)                     //Read new IDs to memory
1204          {
1205            outBuf[bufCt] = statReg.TAGBUFF[j][k];
1206            bufCt++;
1207          }
1208      }
1209
1210      //Reset BUFCT and therefore TAGBUFF as well
1211      statReg.BUFCT = 0;
1212
1213    }
1214
1215    //Find average Y-axis value
1216    for(int j=0;j<40;j++)
1217    {
1218      sum += accBuff[j];
1219      //Find max
1220      if(accBuff[j] > max)
1221      {
1222        max = accBuff[j];
1223      }
1224      //Find min
1225      if(accBuff[j] < min)
1226      {
1227        min = accBuff[j];
```

```
1228      }
1229    }
1230    avg = (int16_t)sum/40;
1231
1232    //accAct logic
1233    if(abs(max - avg) > ACCTHRESH || abs(avg - min) > ACCTHRESH)
1234    {
1235      accAct = true;
1236    }else
1237      {
1238        accAct = false;
1239      }
1240
1241
1242    outBuf[bufCt] = 0x0D;
1243    bufCt++;
1244    outBuf[bufCt] = 0x0A;
1245    bufCt++;
1246    ticTime++;
1247
1248    if(ulpFlg)
1249    {
1250      TA0CCTL0 &= ~CCIE;                    //Turn off sampling of sensors if in ultra-low power
         mode
1251      TA2CCTL2 &= ~CCIE;                    //End timer for millSec
1252    }
1253  }
1254
1255  //-------------------------------------------------------------------------//
1256  //       Name: PowerReg
1257  //   Function: Sets notification LEDs to reflect battery charge state
1258  //     Inputs: state - True = Power on, False = Power off
1259  //    Outputs: N/A
1260  //
1261  //-------------------------------------------------------------------------//
1262  void PowerReg(bool state)
1263  {
1264    if(state)                           //If commanded to turn on
1265    {
1266      //Turn off CAEN
1267      //Turn off accessory (FSR) power
1268      //Put accelerometer in standby
1269    }else                               //Else turn off outside systems
1270      {
1271        //Turn on and initialize CAEN
1272        //Turn on accessory (FSR) power
1273        //Turn on and initialize accelerometer
1274      }
1275  }
1276
1277  //-------------------------------------------------------------------------//
1278  //   FUNCTION NAME: DynamicPoll
1279  //
1280  //          INPUTS: None
```

```
1281 //
1282 //         OUTPUTS: None
1283 //
1284 //   FUNCTIONALITY: turn on and off polling
1285 //-------------------------------------------------------------------------//
1286 void DynamicPoll()
1287 {
1288   if(!fsrAct && accAct)
1289   {
1290     /*************************************************************************
1291     This section activates when the socket is doffed and the socket is moving
1292     *************************************************************************/
1293     if(!pollState)
1294     {
1295       BbangTx(inventoryOn,sizeof(inventoryOn));          //Turn on polling
1296       TA1CCR1 = TA1R + DOFFCONST;               //Set Doff timeout ISR value
1297       TA1CCTL1 |= CCIE;                           //Turn on doff timer
1298     }
1299     TA1CCTL1 |= CCIE;
1300     pollState = true;                                    //We are polling
1301     TA1CCTL2 &= ~CCIE;                  //Turn off timer for ULP Mode
1302     ulpTmr = 0;                           //Reset ULP flag timer
1303   }else if((fsrAct && accAct) || (fsrAct && !accAct))
1304     {
1305       /*************************************************************************
1306       This section activates when the socket is donned
1307       *************************************************************************/
1308       if(pollState)
1309       {
1310         BbangTx(inventoryOff,sizeof(inventoryOff));       //Turn off polling
1311         TA1CCTL1 &= ~CCIE;
1312       }
1313       pollState = false;                                   //We are not polling
1314       TA1CCTL2 &= ~CCIE;                 //Turn off timer for ULP Mode
1315       ulpTmr = 0;                          //Reset ULP flag timer
1316     }else
1317       {
1318         /*************************************************************************
1319         This section is for the ultra-low power condition. This will only be
1320         activated when there is no movement and the socket is doffed. The ULP
1321         flag will only activate after 30 seconds of no movement. The ULP flag is
1322         set in the TA1CCR2 interrupt and cleared via an interrupt from the
1323         accelerometer.
1324         *************************************************************************/
1325         if(ulpFlg)
1326         {
1327           TA1CCTL1 &= ~CCIE;                       //Turn off doff polling timer
1328           pollState = false;                       //No longer polling
1329           CAEN_OFF;                                //Turn off CAEN
1330           ANALOG_OFF;                              //Turn off accessory (FSR) power
1331           STAT1_OFF;
1332           TB0CCTL1 &= ~CCIE;                       //Turn off Bit-bang FROM CAEN
1333
1334           //Configure accelerometer for activity interrupt
```

```
1335            spiVar = SPI_read(0x30);                //Read to clear any pending interrupts
1336            SPI_write(0x2E,0x10);                   //Enable interrupts
1337            P4IE |= BIT7;                           //Enable INT1 interrupt
1338
1339          }else
1340            {
1341              if((TA1CCTL2 & CCIE) != CCIE)     //Is interrupt currently activated?
1342              {
1343                TA1CCR2 = TA1R + SECOND;
1344              }
1345              TA1CCTL2 |= CCIE;                 //Turn on timer for ULP Mode
1346            }
1347        }
1348 }
```

D.2 Initialize.c

```
1  /*
2    This program initialization is for the SockMonitor V1.3
3    Written by David Swartzendruber
4  */
5  #include <msp430f5638.h>
6  #include <Initialize.h>
7  #include <stdint.h>
8  #include <spiutil.h>
9  #include <NAND_driver.h>
10 #include "SockMonitor.h"
11
12
13 void Initialize(void)
14 {
15   volatile uint16_t i;
16
17   //-------------------------------------------------------------------------//
18   //Set pin pullup/down resistor values
19   //-------------------------------------------------------------------------//
20   P1REN = 0x00;
21   P2REN = BIT3;
22   P3REN = 0x00;
23   P4REN = 0x00;
24   P5REN = 0x00;
25   P6REN = 0x00;
26   P7REN = 0x00;
27   P8REN = 0x00;
28   P9REN = 0x00;
29   PJREN = 0x00;
30
31   //-------------------------------------------------------------------------//
32   //Set direction of pullup/down resistor
33   //-------------------------------------------------------------------------//
34   P1OUT = BIT7+BIT6+BIT0;
35   P2OUT = BIT3+BIT2+BIT1+BIT0;
```

```
36  P3OUT = BIT6+BIT3+BIT2;
37  P4OUT = BIT2+BIT1;
38  P5OUT = 0x00;
39  P6OUT = 0x00;
40  P7OUT = 0x00;
41  P8OUT = 0x00;
42  P9OUT = 0x00;
43  PJOUT = BIT3+BIT2+BIT1+BIT0;
44
45  //------------------------------------------------------------------------------//
46  //Set pin Directions
47  //------------------------------------------------------------------------------//
48  P1DIR = BIT7+BIT6+BIT4+BIT3+BIT2+BIT1+BIT0;
49  P2DIR = BIT7+BIT6+BIT5+BIT4+BIT2+BIT1+BIT0;
50  P3DIR = BIT7+BIT6+BIT4+BIT3+BIT2+BIT1+BIT0;
51  P4DIR = BIT5+BIT4+BIT2+BIT0;
52  P5DIR = BIT7+BIT6;
53  P6DIR = BIT7+BIT6+BIT5+BIT4+BIT3+BIT2+BIT1;
54  P7DIR = BIT7+BIT6+BIT5+BIT4;
55  P8DIR = BIT7+BIT1+BIT0;
56  P9DIR = 0xFF;                           //These are the lines to the Flash Memory
57  PJDIR = 0x00;
58
59  //------------------------------------------------------------------------------//
60  //Set peripheral values
61  //------------------------------------------------------------------------------//
62  P1SEL = 0x00;
63  P2SEL = 0x00;
64  P3SEL = 0x00;
65  P4SEL = BIT2+BIT1; //Enable capture and output for TB0
66  P5SEL = 0x00;
67  P6SEL = BIT0;
68  P7SEL = BIT3+BIT2;
69  P8SEL = BIT6+BIT5+BIT4+BIT3+BIT2;
70  P9SEL = 0x00;
71
72
73  //------------------------------------------------------------------------------//
74  //Set clock parameters
75  //------------------------------------------------------------------------------//
76  while(BAKCTL & LOCKBAK)
77  {
78    BAKCTL &= ~(LOCKBAK);
79  }
80  UCSCTL6 = 0x400C;             //Turn on XT1 and set internal caps to 12.5pF
81
82  do
83  {
84    UCSCTL7 &= ~(XT2OFFG + XT1HFOFFG + XT1LFOFFG + DCOFFG); //Clear flags
85    SFRIFG1 &= ~OFIFG;                      //Clear general flag
86    __delay_cycles(100);
87  }while(SFRIFG1&OFIFG);                    //Test if flag reset
88
89  UCSCTL4 = (SELA_XT1CLK+SELS_XT2CLK+SELM_XT2CLK); //Set clock sources
```

```
90
91    do
92      {
93        UCSCTL7 &= ~(XT2OFFG + XT1LFOFFG + XT1HFOFFG + DCOFFG); //Clear flags
94        SFRIFG1 &= ~OFIFG;                      //Clear general flags
95      }while (SFRIFG1&OFIFG);
96
97    //-------------------------------------------------------------------------//
98    //ADC initialization
99    //-------------------------------------------------------------------------//
100   ADC12CTL0 = ADC12SHT02+ADC12ON;           //Cycles to hold and ADC on
101   ADC12CTL1 = ADC12SHP;                     //Pulse mode
102   ADC12CTL0 |= ADC12ENC;                    //Enable ADC
103
104   //-------------------------------------------------------------------------//
105   //SPI, UART and I2C initialization
106   //-------------------------------------------------------------------------//
107
108   UCB1CTL1 = UCSWRST + UCSSEL_SMCLK;          //Disable USCI, SMCLK is source
109   UCB1CTL0 = UCCKPL + UCMST + UCMSB + UCMODE0;//4-pin active low STE, Synch mode
110   UCB1BR0 = 32;                             //Clk divisor,
111   UCB1BR1 = 0;
112   UCB1CTL1 &= ~UCSWRST;                     //Enable USCI
113
114   //-------------------------------------------------------------------------//
115   //Accelerometer initialization
116   //-------------------------------------------------------------------------//
117   for(int j=0;j<ACCDEF;j++)
118   {
119     SPI_write(accAd[j],accDa[j]);
120   }
121
122   //-------------------------------------------------------------------------//
123   //External flash memory initialization
124   //-------------------------------------------------------------------------//
125   NAND_initialize();
126
127   //-------------------------------------------------------------------------//
128   //USB initialization
129   //-------------------------------------------------------------------------//
130   USBKEYPID = USBKEY;                       //Unlock USB registers
131   USBCNF &= ~PUR_IN;                        //Disable pull up resistor
132   USBKEYPID =0x0000;                        //Lock USB registers
133
134   //-------------------------------------------------------------------------//
135   //Bluetooth initialization
136   //-------------------------------------------------------------------------//
137   BAUD_115;
138   UCA1CTL1 |= UCSSEL_2 + UCSWRST;
139   UCA1BR0 = 0x8B;       //16MHz/115200=138.89
140   UCA1BR1 = 0x00;
141   //P8SEL |= BIT2 + BIT3;
142   UCA1CTL1 &= ~UCSWRST;
143
```

```
144  //------------------------------------------------------------------------//
145  //RTC initialization
146  //------------------------------------------------------------------------//
147  RTCCTL1 |= RTCHOLD_H ;          //Disable RTC
148  RTCCTL0 = RTCRDYIE+RTCAIE;
149  RTCCTL3 |= BIT0;               //512Hz output to RTCCLK pin
150  RTCSEC = 30;                    //Seconds
151  RTCMIN = 12;                   //Minutes
152  RTCHOUR = 20;                  //Hour
153  RTCDOW = 4;                    //Day of Week 0=Sunday 6=Saturday
154  RTCDAY = 18;                   //Day of Month
155  RTCMON = 4;                    //Month of year
156  RTCYEAR = 2013;                //Year
157
158  //Set alarm for 12:01am every day, this creates a date stamp
159  RTCAMIN = 0x01 | 0x80;         //The 0x80 is to enable the AE bit
160  RTCAHOUR = 0x00 | 0x80;        //The 0x80 is to enable the AE bit
161  RTCADOW = 0;
162  RTCADAY = 0;
163
164  RTCCTL1 &= ~RTCHOLD_H;
165
166  //------------------------------------------------------------------------//
167  //Initialize timers
168  //------------------------------------------------------------------------//
169  //Sample interrupt timer
170  TA0CCR0 = 655;                          //Counter Value, 32768Hz/50Hz = 655.36
171  TA0CTL = TASSEL_1 + MC_1 + TACLR;       //Source ACLK, UP mode, clear timer
172  //TA0CCTL0 = CCIE;                      //Enable TimerA0 interrupts
173
174
175  //Polling indicator
176  //TA1CCR0 = 4096;                        //Counter Value, fires 1/8 sec
177  TA1CTL = TASSEL_1 + MC_2 + TACLR;       //Source ACLK, Continous mode, clear timer
178  TA1CCTL0 = CCIE;                        //Enable TimerA1 interrupts
179
180
181  //Error interrupt timer
182  TA2CCR0 = 52;
183  TA2CCR2 = 16000;                        //Once every 1/1000 seconds
184  TA2CTL = TASSEL_2 + MC_2 + TACLR + ID_3;      //Source ACLK, UP mode, clear timer
185  //TA1CCTL0 = CCIE;                      //Enable Timer interrupt
186
187
188  /*************************************************************************
189  * Bit-bang setup moved to SockMonitor.c
190  *************************************************************************
191  //Bit-bang interrupt timer
192  TA1CCR0 = 416;                          //Counter Value
193  TA1CTL = TASSEL_2 + MC_1 + TACLR;       //Source SMCLK, UP mode, clear timer
194  //TA2CCTL0 = CCIE;                        //Enable TimerA2 interrupts
195  *************************************************************************/
196
197  //------------------------------------------------------------------------//
```

```
198     //Initialize CAEN
199     //------------------------------------------------------------------------------//
200     CAEN_ON;                                //CAEN is always on... For now...
201     UARTSetUp();                            //Initialize software UART
202
203     //------------------------------------------------------------------------------//
204     //Enable interupts
205     //------------------------------------------------------------------------------//
206     UCA1IE |= UCRXIE;                       //Bluetooth UART RX interrupt
207     P2IE = BIT3;                            //User interface button
208     P2IES = BIT3;
209     P4IE = BIT3 + BIT1;                     //Low battery, CAEN comm
210     P4IES = BIT3;
211
212     __bis_SR_register(GIE); //General interrupt enable
213
214     //------------------------------------------------------------------------------//
215     //Initialize CAEN continuous inventory
216     //------------------------------------------------------------------------------//
217     __delay_cycles(150);                    //Allows time for line to go high
218     BbangTx(CAENInit, sizeof(CAENInit));    //Enable continuous inventory
219 }
```

D.3 Initialize.h

```
1  #include <msp430f5638.h>
2
3  #ifndef INITIALIZE_H_GUARD
4  #define INITIALIZE_H_GUARD
5  void Initialize(void);
6
7
8  #define PIN1_0 BIT0
9  #define PIN1_1 BIT1
10 #define PIN1_2 BIT2
11 #define PIN1_3 BIT3
12 #define PIN1_4 BIT4
13 #define PIN1_5 BIT5
14 #define PIN1_6 BIT6
15 #define PIN1_7 BIT7
16
17 #define PIN2_0 BIT0
18 #define PIN2_1 BIT1
19 #define PIN2_2 BIT2
20 #define PIN2_3 BIT3
21 #define PIN2_4 BIT4
22 #define PIN2_5 BIT5
23 #define PIN2_6 BIT6
24 #define PIN2_7 BIT7
25
26 #define PIN3_0 BIT0
27 #define PIN3_1 BIT1
```

```
28 #define PIN3_2 BIT2
29 #define PIN3_3 BIT3
30 #define PIN3_4 BIT4
31 #define PIN3_5 BIT5
32 #define PIN3_6 BIT6
33 #define PIN3_7 BIT7
34
35 #define PIN4_0 BIT0
36 #define PIN4_1 BIT1
37 #define PIN4_2 BIT2
38 #define PIN4_3 BIT3
39 #define PIN4_4 BIT4
40 #define PIN4_5 BIT5
41 #define PIN4_6 BIT6
42 #define PIN4_7 BIT7
43
44 #define PIN5_0 BIT0
45 #define PIN5_1 BIT1
46 #define PIN5_2 BIT2
47 #define PIN5_3 BIT3
48 #define PIN5_4 BIT4
49 #define PIN5_5 BIT5
50 #define PIN5_6 BIT6
51 #define PIN5_7 BIT7
52
53 #define PIN6_0 BIT0
54 #define PIN6_1 BIT1
55 #define PIN6_2 BIT2
56 #define PIN6_3 BIT3
57 #define PIN6_4 BIT4
58 #define PIN6_5 BIT5
59 #define PIN6_6 BIT6
60 #define PIN6_7 BIT7
61
62 #define PIN7_2 BIT2
63 #define PIN7_3 BIT3
64 #define PIN7_4 BIT4
65 #define PIN7_5 BIT5
66 #define PIN7_6 BIT6
67 #define PIN7_7 BIT7
68
69 #define PIN8_0 BIT0
70 #define PIN8_1 BIT1
71 #define PIN8_2 BIT2
72 #define PIN8_3 BIT3
73 #define PIN8_4 BIT4
74 #define PIN8_5 BIT5
75 #define PIN8_6 BIT6
76 #define PIN8_7 BIT7
77
78 #define PIN9_0 BIT0
79 #define PIN9_1 BIT1
80 #define PIN9_2 BIT2
81 #define PIN9_3 BIT3
```

```
#define PIN9_4 BIT4
#define PIN9_5 BIT5
#define PIN9_6 BIT6
#define PIN9_7 BIT7

#define PJ0 BIT0
#define PJ1 BIT1
#define PJ2 BIT2
#define PJ3 BIT3

#define PU0 BIT0
#define PU1 BIT1

#define BAUD_96 (P4OUT |= BIT0)             //Baud rate set to 9600
#define BAUD_115 (P4OUT &= ~BIT0)           //Baud rate set to 115200

#endif
```

D.4 SockMonitor.c

```
#include <msp430f5638.h>
#include <stdint.h>
#include <intrinsics.h>
#include <Initialize.h>
#include <NAND_driver.h>
#include <stdbool.h>
#include <spiutil.h>
#include <SockMonitor.h>
#include "convertutil.h"

uint8_t RXBUF;              //Buffer for receive byte
uint8_t TXBUF;              //Buffer for transmit byte

//-------------------------------------------------------------------------//
//  FUNCTION NAME: ToCAEN
//
//          INPUTS: Array of
//
//         OUTPUTS: None
//
//   FUNCTIONALITY: Transmit data to CAEN via bit banging
//-------------------------------------------------------------------------//
void BbangTx(unsigned char *datIn, uint16_t datL)
{
  for(int i=0;i<datL;i++)
  {
    while((TXIFG & TXPIN) == 0){}         //Delay while TXBUF full
    WriteTXBUF(datIn[i]);
  }
}

//-------------------------------------------------------------------------//
```

```
33 //  FUNCTION NAME: PrintMess
34 //
35 //          INPUTS: toPrint = String to be printed
36 //                  sizeMess = Size of string in bytes
37 //
38 //         OUTPUTS: None
39 //
40 //   FUNCTIONALITY: Transmit string to terminal via Bluetooth
41 //-------------------------------------------------------------------------//
42 void PrintMess(const unsigned char *toPrint, uint16_t sizeMess)
43 {
44   for(int i=0;i<sizeMess;i++)        //Display Cancel message
45   {
46     while(!(UCA1IFG&UCTXIFG)){};
47     UCA1TXBUF = toPrint[i];
48   }
49 }
50
51 //-------------------------------------------------------------------------//
52 //  FUNCTION NAME: PrintMess
53 //
54 //          INPUTS: None
55 //
56 //         OUTPUTS: None
57 //
58 //   FUNCTIONALITY: Transmit string to terminal via Bluetooth
59 //-------------------------------------------------------------------------//
60
61
62
63
64
65
66 //-------------------------------------------------------------------------//
67 //  FUNCTION NAME: GetInfo
68 //
69 //          INPUTS: None
70 //
71 //         OUTPUTS: None
72 //
73 //   FUNCTIONALITY: Display the below information on the screen for the user;
74 //                   Device name
75 //                   Low Battery?
76 //                   Currently recording?
77 //                   STATUS 1 LED on?
78 //                   STATUS 2 LED on?
79 //                   Power to Accel on?
80 //                   Power to FSRs on?
81 //                   Power to CAEN on?
82 //                   Button pressed?
83 //
84 //-------------------------------------------------------------------------//
85 void GetInfo(uint8_t Hours, uint8_t Minutes, uint8_t Seconds)
86 {
```

```
87  const unsigned char dName[] = "\x1B[2J\x1B[1;1HDevice Name: SM 1\n\r";
88  const unsigned char softVer[] = "Software last updated: 18 April 2013\n\r";
89  const unsigned char sysTime[] = "System Time:          ";
90  const unsigned char bStat[] =   "Battery is:          ";
91  const unsigned char rStat[] =   "Recording?           ";
92  const unsigned char stat1[] =   "Status 1 LED on?     ";
93  const unsigned char stat2[] =   "Status 2 LED on?     ";
94  const unsigned char pwrAcc[] =  "Power to Accel on?   ";
95  const unsigned char pwrFsr[] =  "Power to FSRs on?    ";
96  const unsigned char pwrCaen[] = "Power to CAEN on?    ";
97  const unsigned char butPres[] = "Button pressed?      ";
98  const unsigned char ulpMess[] = "Ultra Low Power Mode? ";
99  const unsigned char low[] =     "Low\n\r";
100 const unsigned char chr[] =     "Charged\n\r";
101 const unsigned char ys[] =      "Yes\n\r";
102 const unsigned char neg[] =     "No\n\r";
103 const unsigned char on[] =      "On\n\r";
104 const unsigned char off[] =     "Off\n\r";
105 const unsigned char standBy[] = "Standby\n\r";
106 unsigned char time[10] = {'0','0',':','0','0',':','0','0','\n','\r'};
107 static unsigned char timeTemp[3] = {0};
108
109 //Find hours first
110 inttochar((uint32_t)Hours,timeTemp,sizeof(timeTemp));
111 if(timeTemp[1] == '\0')
112 {
113   time[1] = timeTemp[0];
114 }else
115   {
116     time[0] = timeTemp[0];
117     time[1] = timeTemp[1];
118   }
119
120 //Next find minutes
121 inttochar((uint32_t)Minutes,timeTemp,sizeof(timeTemp));
122 if(timeTemp[1] == '\0')
123 {
124   time[4] = timeTemp[0];
125 }else
126   {
127     time[3] = timeTemp[0];
128     time[4] = timeTemp[1];
129   }
130
131 //Next find seconds
132 inttochar((uint32_t)Seconds,timeTemp,sizeof(timeTemp));
133 if(timeTemp[1] == '\0')
134 {
135   time[7] = timeTemp[0];
136 }else
137   {
138     time[6] = timeTemp[0];
139     time[7] = timeTemp[1];
140   }
```

```
141
142   PrintMess(dName,sizeof(dName));
143   PrintMess(softVer,sizeof(softVer));
144   PrintMess(sysTime,sizeof(sysTime));
145   PrintMess(time,sizeof(time));
146   PrintMess(bStat,sizeof(bStat));
147
148   if((P4IN & BIT3) == BIT3) //Handle Low Batt answer
149   {
150     PrintMess(chr,sizeof(chr));
151   }else
152     {
153       PrintMess(low,sizeof(low));
154     }
155
156   PrintMess(rStat,sizeof(rStat));
157   if(((TA0CCTL0 & CCIE) == CCIE) || ulpFlg) //Handle Recording answer
158   {
159     PrintMess(ys,sizeof(ys));
160   }else
161     {
162       PrintMess(neg,sizeof(neg));
163     }
164
165   PrintMess(stat1,sizeof(stat1));         //Status 1 LED
166   if((P2OUT & STAT1) == STAT1)
167   {
168     PrintMess(off,sizeof(off));
169   }else
170     {
171       PrintMess(on,sizeof(on));
172     }
173
174   PrintMess(stat2,sizeof(stat2));         //Status 2 LED
175   if((P2OUT & STAT2) == STAT2)
176   {
177     PrintMess(off,sizeof(off));
178   }else
179     {
180       PrintMess(on,sizeof(on));
181     }
182
183   PrintMess(pwrAcc,sizeof(pwrAcc));       //Accelerometer power
184   if((P6OUT & BIT5) == 0 && (P6OUT & BIT6) == 0)
185   {
186     PrintMess(on,sizeof(on));
187   }else if((P6OUT & BIT5) == 0 && (P6OUT & BIT6) != 0)
188     {
189       PrintMess(standBy,sizeof(standBy));
190     }
191       else
192         {
193           PrintMess(off,sizeof(off));
194         }
```

```
195
196     PrintMess(pwrFsr,sizeof(pwrFsr));       //FSR Power
197     if((P6OUT & BIT4) == BIT4)
198     {
199       PrintMess(off,sizeof(off));
200     }else
201       {
202         PrintMess(on,sizeof(on));
203       }
204
205     PrintMess(pwrCaen,sizeof(pwrCaen));     //CAEN Power
206     if((P2OUT & CAENPWR) == CAENPWR)
207     {
208       PrintMess(off,sizeof(off));
209     }else
210       {
211         PrintMess(on,sizeof(on));
212       }
213
214     PrintMess(butPres,sizeof(butPres));     //Button pressed?
215     if((P2IN & BIT3) == BIT3)
216     {
217       PrintMess(neg,sizeof(neg));
218     }else
219       {
220         PrintMess(ys,sizeof(ys));
221       }
222
223     PrintMess(ulpMess,sizeof(ulpMess));
224     if(ulpFlg)
225     {
226       PrintMess(ys,sizeof(ys));
227     }else
228       {
229         PrintMess(neg,sizeof(neg));
230       }
231 }
232
233 //-------------------------------------------------------------------------//
234 //   FUNCTION NAME: ChangeTime
235 //
236 //          INPUTS: None
237 //
238 //         OUTPUTS: None
239 //
240 //   FUNCTIONALITY: Change System Time through user input
241 //-------------------------------------------------------------------------//
242 void ChangeTime()
243 {
244   const unsigned char t[] = "you pressed t\n\r";
245   PrintMess(t,sizeof(t));
246
247 }
248
```

```
249 //-------------------------------------------------------------------------//
250 //  FUNCTION NAME: ChangeDate
251 //
252 //           INPUTS: None
253 //
254 //          OUTPUTS: None
255 //
256 //    FUNCTIONALITY: Change System Date through user input
257 //-------------------------------------------------------------------------//
258 void ChangeDate()
259 {
260   const unsigned char d[] = "You pressed d\n\r";
261   PrintMess(d,sizeof(d));
262 }
263
264 //-------------------------------------------------------------------------//
265 //  FUNCTION NAME: WrongEntry
266 //
267 //           INPUTS: None
268 //
269 //          OUTPUTS: None
270 //
271 //    FUNCTIONALITY: Notify user of wrong entry
272 //-------------------------------------------------------------------------//
273 void WrongEntry()
274 {
275   const unsigned char wrg[] = "You pressed a wrong button, please try again.\n\r";
276   PrintMess(wrg,sizeof(wrg));
277 }
278
279 //-------------------------------------------------------------------------//
280 //  FUNCTION NAME: UARTSetUp
281 //
282 //           INPUTS: None
283 //
284 //          OUTPUTS: None
285 //
286 //    FUNCTIONALITY: Setup software UART to CAEN module
287 //-------------------------------------------------------------------------//
288 void UARTSetUp(void)
289 {
290   TB0CTL = TBSSEL_2 | MC_1;                    //SMCLK, Up mode
291   TB0CCTL2 = OUT | OUTMOD_0 | CCIFG;           //To CAEN; Out high, OUT set to value in OUT,
          Interrupt pending
292   TB0CCTL1 = CM_2 | CCIS_0 | SCS | CAP | OUTMOD_0 | CCIE;      //From CAEN;
293   TB0CCR1 = BITTIME;
294   RXIFG &= ~RXPIN;
295   TXIFG |= TXPIN;
296 }
297
298 //-------------------------------------------------------------------------//
299 //  FUNCTION NAME: WriteTXBUF
300 //
301 //           INPUTS: TXData
```

```
302 //
303 //          OUTPUTS: None
304 //
305 //    FUNCTIONALITY: Write byte to TXBUF, clear flag, start transmitter if needed
306 //-------------------------------------------------------------------------------//
307
308 void WriteTXBUF(const uint8_t TXData)
309 {
310   TXBUF = TXData;
311   TXIFG &= ~TXPIN;
312   TB0CCTL2 |= CCIE;
313 }
314
315 //-------------------------------------------------------------------------------//
316 //   FUNCTION NAME: ReadRXBUF
317 //
318 //           INPUTS: None
319 //
320 //          OUTPUTS: None
321 //
322 //    FUNCTIONALITY: Read byte from RXBUF, clear flag
323 //-------------------------------------------------------------------------------//
324
325 uint8_t ReadRXBUF(void)
326 {
327   RXIFG &= ~RXPIN;
328   return RXBUF;
329 }
330
331 //-------------------------------------------------------------------------------//
332 //       Name: Timer B0, channels 1 and 2 ISR
333 //   Function: Used to transmit data to and from CAEN
334 //     Inputs: N/A
335 //    Outputs: N/A
336 //
337 //-------------------------------------------------------------------------------//
338 #pragma vector=TIMER0_B1_VECTOR
339 __interrupt void TIMER0_B0_ISR(void)
340 {
341     static uint8_t TXBitCount = 0;
342     static uint16_t TXShiftReg;
343     static uint8_t RXBitCount = 0;
344     static uint16_t RXShiftReg;
345
346   switch(__even_in_range(TBIV,14))
347   {
348   case  2:                                  // Vector  2:  CCR1, RX from CAEN
349     TB0CCTL1 &= ~CCIFG;                     //Clear interrupt flag
350     if((TB0CCTL1 & CAP) != 0)               //If capture, start bit detected
351     {
352       TA2CCTL1 &= ~CCIE;                    //End b-bang end timer
353       TB0CCTL1 &= ~CAP;                     //Switch to sampling mode
354       RXBitCount = 8;
355
```

```
356       /*The if statement below is permissable because by the time you switch to
357          sampling mode, enough clock cycles have gone by to once again sample
358          and make sure you aren't getting an erroneous read*/
359       if((P4IN & RXPIN) != 0)            //Error: start bit should be low
360         {                                // abandon reception
361           TB0CCTL1 |= CAP;
362         }
363
364     }else
365       {
366         if(RXBitCount == 0)
367         {                                //Stop bit
368           if((P4IN & RXPIN) != 0)
369           {
370             RXBUF = RXShiftReg;          //Store data in buffer
371             RXIFG |= RXPIN;              //Set flag to show new data
372           }
373           TB0CCTL1 |= CAP;               //Return to capture mode
374           TA2CCR1 = TA2R + 52;           //Reset timer
375           TA2CCTL1 &= ~CCIFG;            //Clear any interrupts pending
376           TA2CCTL1 |= CCIE;              //Start b-bang end timer
377         }else
378           {
379             RXShiftReg >>= 1;
380             if((P4IN & RXPIN) != 0)
381             {
382               RXShiftReg |= BIT7;        //Set receive bit if high
383             }
384             --RXBitCount;
385           }
386       }
387     break;
388
389   case  4:                             // Vector  4:  CCR2, TX to CAEN
390     if(TXBitCount == 0)
391     {
392       if((TXIFG & TXPIN) == 0)
393       {
394         TXShiftReg = TXBUF;
395         TXShiftReg |= BIT8;            //Add stop bit after msb
396         TXIFG |= TXPIN;                //Set TXBUF empty flag
397         TXBitCount = 9;
398         TB0CCR0 = BITTIME;             //Set counter time length
399         TB0CCTL2 = OUTMOD5 | CCIE;     //Clear output on compare for start
400       }else
401         {
402           TB0CCTL2 = OUT | OUTMOD0 | CCIFG;
403         }
404     }else
405       {
406         if(TXShiftReg & BIT0)          //Send 1
407         {
408           TB0CCTL2 = OUTMOD1 | CCIE;
409         }else                          //Send 0
```

```
410             {
411               TB0CCTL2 = OUTMOD_5 | CCIE;
412             }
413           TXShiftReg >>= 1;               //Shift Byte
414           --TXBitCount;
415         }
416       break;
417
418     case  6: break;                       // Vector  6:  CCR3
419     case  8: break;                       // Vector  8:  CCR4
420     case 10: break;                       // Vector 10:  CCR5
421     case 12: break;                       // Vector 12:  CCR6
422     default: break;
423     }
424
425 }
```

D.5 SockMonitor.h

```
1  #include <msp430f5638.h>
2  #include <stdint.h>
3  #include <intrinsics.h>
4  #include <Initialize.h>
5  #include "NAND_driver.h"
6  #include "CAEN_driver.h"
7  #include <stdbool.h>
8  #include <spiutil.h>
9
10 #ifndef SOCKMONITOR_H_GUARD
11 #define SOCKMONITOR_H_GUARD
12
13 void BbangTx(unsigned char *,uint16_t);
14 bool MemCheck();
15 void DataPush();
16 void DataSample();
17 void BattCheck();
18 void bhandle();
19 void DynamicPoll();
20 void PowerReg(bool);
21 void PrintMess(const unsigned char *,uint16_t);
22 void GetInfo(uint8_t,uint8_t,uint8_t);
23 void ChangeTime();
24 void ChangeDate();
25 void WrongEntry();
26
27 #define PIN_L           (P4OUT &= ~BIT2)        //Transmit TO CAEN on this pin
28 #define PIN_H           (P4OUT |= BIT2)         //Transmit TO CAEN on this pin
29 #define CAENTX          BIT1                        //Receive FROM CAEN on this pin
30 #define BAUDDELAY       121
31 #define ACCSAMP         50
32 #define NUMSAMP         1
33 #define DELIM           0x24                            //Delimiting character
```

```
34 #define STAT1_ON            (P2OUT &= ~BIT1)
35 #define STAT1_OFF           (P2OUT |= BIT1)
36 #define STAT1_TOGGLE        (P2OUT ^= BIT1)
37 #define STAT2_ON            (P2OUT &= ~BIT2)
38 #define STAT2_OFF           (P2OUT |= BIT2)
39 #define STAT2_TOGGLE        (P2OUT ^= BIT2)
40 #define CAEN_ON             (P2OUT &= ~BIT0)
41 #define CAEN_OFF            (P2OUT |= BIT0)
42 #define ANALOG_OFF          (P6OUT |= BIT4)
43 #define ANALOG_ON           (P6OUT &= ~BIT4)
44 #define STAT1               BIT1
45 #define STAT2               BIT2
46 #define CAENPWR             BIT0
47 #define ACCEL_STANDBY       (P6OUT |= BIT6)
48 #define ACCEL_ON            (P6OUT &= ~BIT6)
49 #define C_ON                0
50 #define C_OFF               1
51 #define CAEN_STAT           (P2OUT & BIT0)
52 #define DOFFCONST           16384
53 #define SECOND              32768
54 #define HALFSECOND          16384
55 #define SAMP_STAT           (TA0CCTL0 & CCIE)
56
57 //Use when writing TO memory
58 #define DATASIZE_TO(a)  (BAKMEM0 = a&0x0000FFFF, BAKMEM1 = (a&0xFFFF0000)>>16)
59 //Use when reading FROM memory
60 #define DATASIZE_FROM ((uint32_t) ((BAKMEM1<<16)|BAKMEM0))
61
62
63 #define BUFFSIZE        1024
64 #define POLL_TIMEOUT 10                     //Time-out in seconds
65 #define FSRTHRESH      50                   //FSR activity threshold
66 #define ACCTHRESH      6                   //Accelerometer activity threshold
67 #define X_AXIS          1
68 #define Y_AXIS          3
69 #define Z_AXIS          5
70
71 extern bool ulpFlg;                        //Ultra low power flag
72
73 /*************************************************************************
74 Bit Bang Defines
75 *************************************************************************/
76 #define RXIFG    P4IFG
77 #define TXIFG    P4IFG
78 #define RXIE     P4IE
79 #define TXIE     P4IE
80 #define TXPIN    BIT2
81 #define RXPIN    BIT1
82 #define BITTIME ((16000000)/115200)        //Cycles of TACLK per bit, 16MHz
83 #define HALFTIME (BITTIME/2)               //Cycles for half a bit
84
85
86 void UARTSetUp(void);
87 void WriteTXBUF(const uint8_t TXData);
```

```
88 uint8_t ReadRXBUF(void);
89
90 /*****************************************************************************
91 Multiplexer Definitions
92 *****************************************************************************/
93
94 #define FSR1     0
95 #define FSR2     1
96 #define FSR3     2
97 #define FSR4     3
98 #define ANALOG1 4
99 #define ANALOG2 5
100 #define ANALOG3 6
101 #define ANALOG4 7
102 #define MUXON   (P1OUT &= ~BIT4)
103 #define MUXOFF  (P1OUT |= BIT4)
104
105 static uint8_t analogMsk[8] = {FSR1,FSR2,FSR3,FSR4,ANALOG1,ANALOG2,ANALOG3,ANALOG4};
106
107 /*****************************************************************************
108 Misc. strings
109 *****************************************************************************/
110 static unsigned char CAENInit[] = {0x80,0x01,0x00,0x03,0x00,0x00,0x53,0x58,0x00,
111 0x35,0x00,0x00,0x00,0x08,0x00,0x01,0x00,0x8A,0x00,0x00,0x00,0x0F,0x00,0xFB,0x53,
112 0x6F,0x75,0x72,0x63,0x65,0x5F,0x30,0x00,0x00,0x00,0x00,0x0A,0x00,0x6A,0x00,0x00,
113 0x00,0x00,0x00,0x00,0x00,0x0A,0x00,0x6B,0x00,0x00,0x00,0x00};
114
115 static unsigned char verNum[] = "Ver 1.3.3\r\n";
116
117 static unsigned char dateStp[] = "Date";
118
119 #define ACCDEF  5
120
121 //Accelerometer Addresses; DATA_FORMAT,BW_RATE,FIFO_CTL,INT_ENABLE,POWER_CTL
122 static uint8_t accAd[ACCDEF] = {0x31,0x2C,0x24,0x27,0x2D};
123 //Accelerometer Data; Full Res, 16g, 50 Hz, Activity interrupt enable, measure enable
124 static uint8_t accDa[ACCDEF] = {0x0B,0x09,0x03,0xF0,0x08};
125
126
127 #endif /*SOCKMONITOR_H_GUARD*/
```

D.6 CAEN driver.c

```
1 /*=============================================================================
2 This driver is programmed to interface between an MSP430F5638 and R1230CB-QUARK
3 on the Sock Monitor V1.3
4
5 Written by
6 David Swartzendruber Jr.
7 University of Washingon, Seattle
8 November 2012.
9 =============================================================================*/
```

```
10 #include <msp430f5638.h>
11 #include "CAEN_driver.h"
12 #include <stdbool.h>
13 #include <stdint.h>
14 #include <SockMonitor.h>
15 #include <string.h>
16
17 //uint8_t TAGBUFF [10][12] = {0};          //Buffer with tags, 96 bit tags ONLY!!!
18 //uint8_t BUFCT = 0;                       //Number of tags stored
19
20
21 void CAEN_Decoder(unsigned char *bytesIn, CAENDat* statReg)
22 {
23   uint16_t dataIn;                 //Reserve header
24   uint16_t packLen;                //Packet length
25   uint16_t packTyp;                //AVP type
26   uint8_t  packVal[64];            //AVP value
27   uint8_t newByte;
28   bool errFlg = false;             //Generic error flag
29   bool storeTag = true;           //Store new tag flag
30
31   //Extract header information
32   dataIn = (bytesIn[0] << 8) | bytesIn[1];
33
34   if(dataIn == CAEN_PAC)
35   {
36     //Read Packet length
37     packLen = (bytesIn[2] << 8) | bytesIn[3];
38     //Read packet type
39     packTyp = (bytesIn[4] << 8) | bytesIn[5];
40     //Read packet value
41
42     if(packLen <= 0)
43     {
44       errFlg = true;
45       return;
46     }
47
48     for(int i=0;i<(packLen - 6);i++)
49         {
50           packVal[i] = bytesIn[6 + i];
51         }
52
53     //Switch statement for AVP Type
54     switch(packTyp)
55     {
56     case TAGID:
57       STAT1_ON;                        //Indicate message read
58       TA1CCR0 = TA1R + 4096;                 //Reset timer
59       //Is tag new?
60       for(int k=0;k<statReg->BUFCT;k++)              //Go through all tags
61       {
62         newByte = 0;
```

```
63          for(int a=0;a<12;a++)               //Go through all bytes,Note that 12 bytes is for
        96bit tags ONLY!!!!
64          {
65            if(packVal[a] == statReg->TAGBUFF[k][a])  //Compare byte by byte
66            {
67              newByte++;
68            }
69          }
70          if(newByte < 12)
71          {
72            storeTag = true;
73          }else
74          {
75            storeTag = false;
76            break;                            //If a false value is ever reached, escape check
77          }
78        }
79
80        //If yes store tag
81        if(storeTag && statReg->BUFCT < 10)        //Prevent more than 10 tags
82        {
83          for(int j=0;j<(packLen-6);j++)
84          {
85            statReg->TAGBUFF[statReg->BUFCT][j] = packVal[j];
86          }
87          statReg->BUFCT++;                         //Increment counter
88          storeTag = false;                         //Reset flag
89        }
90        break;
91
92      case RESULT_CODE:
93        if(((packVal[0] << 8) + packVal[1]) != 0)
94        {
95          errFlg = true;
96        }
97
98        break;
99
100     default:
101       //Do nothing
102       break;
103     }
104
105   }else if(dataIn == CAEN_RX)
106     {
107       //Do nothing for now...
108     }
109 }
110
111 /*{
112   uint8_t idCount = 0;                  //Number of tags detected
113   uint16_t resultCode;
114   bool errorFl = false;                 //Error flag, error given by CAEN result code, FOR
        FUTURE USE
```

```
115  uint16_t arrayCounter = 10;
116
117  CAENHead header;
118  CAENPack packet;
119  packet.newIDs = false;                    //No new IDs
120
121  //Extract header information
122  header.messDir = (bytesIn[0] << 8) | bytesIn[1];
123
124  if(header.messDir == CAEN_RX)
125  {
126    header.messID = (bytesIn[2] << 8) | bytesIn[3];
127    header.venID = bytesIn[4];
128    for(int j=0;j<3;j++)
129    {
130      header.venID = (header.venID << 8) + bytesIn[5+j];
131    }
132    header.messLen = (bytesIn[8] << 8) + bytesIn[9];
133
134    //if(header.messDir != CAEN_RX)         //This is a TEMPORARY fix for comm issues
135    //{
136    //  errorFl = true;
137    //}
138
139    while((arrayCounter < header.messLen) && !errorFl)
140    {
141      //Begin processing packet information
142      packet.reserve = (bytesIn[arrayCounter] << 8) + bytesIn[arrayCounter + 1];
143      packet.length = (bytesIn[arrayCounter + 2] << 8) + bytesIn[arrayCounter + 3];
144      packet.avpType = (bytesIn[arrayCounter + 4] << 8) + bytesIn[arrayCounter + 5];
145      //Loop here
146      for(int j=0;j<(packet.length - 6);j++)
147      {
148        packet.avpValue[j] = bytesIn[arrayCounter + 6 + j]; //Value stored backwards
149      }
150
151      //Analyze AVP Type and value
152      switch(packet.avpType)
153      {
154      case TAG_ID:
155        for(int j=0;j<20;j++)
156        {
157          packet.tagIDs[idCount][j] = packet.avpValue[j];
158        }
159        idCount++;
160        packet.newIDs = true;
161        break;
162
163      case RESULT_CODE:
164        resultCode = (packet.avpValue[0] << 8) | packet.avpValue[1];
165        break;
166
167      case TAG_ID_LEN:
168        packet.IDlength[idCount] = (packet.avpValue[0] << 8) | packet.avpValue[1];
```

```
169           break;
170
171         default:
172             break;
173         }
174
175         //Loop escape logic
176         if((packet.avpType == RESULT_CODE) && (resultCode == 0))
177         {
178
179         }else if(packet.avpType == RESULT_CODE)
180         {
181           errorFl = true;
182         }
183
184         arrayCounter += packet.length; //Increment array counter
185       }
186     }else if(header.messDir == CAEN_MID)
187       {
188         //Extract rest of packet
189         packet.reserve = header.messDir;
190         packet.length = (bytesIn[2] << 8) | bytesIn[3];
191         packet.avpType = (bytesIn[4] << 8) | bytesIn[5];
192         for(int j=0;j<(packet.length - 6);j++)
193           {
194             packet.avpValue[j] = bytesIn[6 + j];
195           }
196
197         //Determine if the packet is indeed a new tag
198         if(packet.avpType == TAG_ID)                  //If avp type is a tag
199         {
200           bool newTag=false;
201           for(int k=0;k<idCount;k++)
202           {
203             if(strcmp((char const*)packet.avpValue,(char const*)packet.tagIDs[k]))
204             {
205               newTag = true;
206             }
207           }
208           if(newTag)
209           {
210             for(int j=0;j<20;j++)
211             {
212               packet.tagIDs[idCount][j] = packet.avpValue[j];
213             }
214             idCount++;
215             newTag = false;
216             packet.newIDs = true;
217           }
218
219         }else                                         //If avp type is not a tag
220           {
221             errorFl = true;                          //Set error,
222           }
```

```
223     }
224
225   //Put header and pack back into messRec
226   packet.numIDs = idCount;
227   messRec->headIn = header;
228   messRec->packIn = packet;
229 }*/
```

D.7 CAEN driver.h

```
1  #include <msp430f5638.h>
2  #include <stdbool.h>
3  #include <stdint.h>
4
5  #ifndef CAENH_GUARD
6  #define CAENH_GUARD
7
8  typedef struct {
9
10   uint16_t reserve;                    //Ignore first line
11   uint16_t length;                     //Size of packet in bytes
12   uint16_t avpType;                    //Information type
13   unsigned char avpValue[64];          //Information value
14   unsigned char tagIDs[10][20];        //Up to 10 tag IDs stored here
15   unsigned char IDlength[10];          //Length of ID, either 12 or 20 byte
16   uint8_t numIDs;                      //Number of new IDs
17   bool newIDs;                         //New IDs flag
18
19 }CAENPack;
20
21 typedef struct {
22
23   uint16_t messDir;                    //Message direction, sent or received?
24   uint16_t messID;                     //Unique Message ID
25   uint32_t venID;                      //Vender ID, i.e. CAEN SpA
26   uint16_t messLen;                    //Overall Message Length in bytes
27
28 }CAENHead;
29
30 typedef struct {
31   CAENPack packIn;
32   CAENHead headIn;
33 }CAENMess;
34
35 typedef struct {
36   uint8_t TAGBUFF [10][12];          //Buffer with tags, 96 bit tags ONLY!!!
37   uint8_t BUFCT;
38 }CAENDat;
39 //-------------------------------Prototypes-------------------------------------//
40 void CAEN_Decoder(unsigned char *,CAENDat*);
41
42 //-------------------------------AVP Type---------------------------------------//
```

```
#define SOURCE_NAME       0x00FB
#define COMMAND_NAME      0x0001
#define READPOINT_NAME    0x0022
#define TIME_STAMP        0x0010
#define TAG_TYPE          0x0012
#define TAGID_LEN         0x000F
#define TAGID             0x0011
#define RESULT_CODE       0x0002

//-------------------------------AVP Value-------------------------------//
#define VENDOR_ID         0x00005358
#define EPCC1G2           0x0003
#define LEN_96            0x000C
#define LEN_160           0x0014
#define ANT0              0x416E743000
#define SOURCE0           0x536F757263655F3000
#define CAEN_TX           0x8001
#define CAEN_RX           0x0001
#define CAEN_PAC          0x0000

//-------------------------------Masks-----------------------------------//
#define REMSK             0x0003
#define LENMSK            0x0006
#define AVPTMSK           0x000C
#define MESSDIRMSK        0x0003
#define MESSIDMSK         0x0006
#define VENIDMSK          0xFF00
#define MESSLENMSK        0x0018


#endif /*CAEN_H_GUARD*/
```

D.8 convertutil.c

```
/*=============================================================================//
The functions contained in this file are utilities to convert a variety of
inputs to another data type.
//=============================================================================*/

#include <msp430f5638.h>
#include <stdint.h>
#include <intrinsics.h>
#include "convertutil.h"
#include <stdbool.h>
#include <string.h>

void inttochar(uint32_t intIn, unsigned char *buf, uint8_t leng)
{
  uint8_t ct = 0;
  static const uint32_t nums[STEPS] = {1e9,1e8,1e7,1e6,1e5,1e4,1e3,1e2,1e1};
  int bufSt = 0;                //Start of buffer, used in negative numbers
  int curPt = 0;                //Current place in buffer to write
```

```
19    bool leadZero = true;
20    memset(buf,0,leng);
21
22    //Check if zero
23    if(intIn == 0)
24    {
25      buf[0] = '0';
26      return;
27    }
28
29    //Convert negative to absolute value and insert "-"
30    if((int32_t)intIn < 0)
31    {
32      buf[curPt] = '-';
33      bufSt = 1;
34      curPt++;
35      intIn = 0 - intIn;
36    }
37
38    //Converts positive number to ascii
39    for(int j=bufSt;j<(bufSt + STEPS);j++)
40    {
41      while(intIn >= nums[j])
42      {
43          ct++;
44          intIn -= nums[j];
45      }
46
47      if(leadZero)
48      {
49        if(ct != 0)
50        {
51            leadZero = false;
52        }
53      }
54
55      if(!leadZero)
56      {
57        buf[curPt] = ct + '0';
58        curPt++;
59      }
60
61      ct = 0;
62    }
63
64    buf[curPt] = intIn + '0';
65  }
```

D.9 convertutil.h

```
1  #include <msp430f5638.h>
2
```

```
#ifndef CONVERTUTIL_H_GUARD
#define CONVERTUTIL_H_GUARD

void inttochar(uint32_t, unsigned char *, uint8_t);

#define STEPS 9

#endif
```

D.10 NAND driver.c

```
/*=============================================================================
This driver is programmed to interface between an MSP430F5638 and Micron
MT29F2G08ABAEA-IT:E TR Flash memory on the Sock Monitor V1.2, and is also
compatable with version 1.3

Written by
David Swartzendruber Jr.
University of Washingon, Seattle
July 2012.

GLOSSARY:
Column Address - Location of byte within a page
Row Address    - Location of Page (Byte 1), Block (Byte 2), and LUN(Byte 3)
CAx - Column address
PAx - Page address
BAx - Block address

Cycle      I/O7    I/O6    I/O5    I/O4    I/O3    I/O2    I/O1    I/O0
-------------------------------------------------------------------------
FIRST      CA7     CA6     CA5     CA4     CA3     CA2     CA1     CA0
SECOND     LOW     LOW     LOW     LOW     CA11    CA10    CA9     CA8
THIRD      BA7     BA6     PA5     PA4     PA3     PA2     PA1     PA0
FOURTH     BA15    BA14    BA13    BA12    BA11    BA10    BA9     BA8
FIFTH      LOW     LOW     LOW     LOW     LOW     LOW     LOW     BA16

NOTE: BA6 controls plane selection
=============================================================================*/
#include <msp430f5638.h>
#include <NAND_driver.h>
#include <stdbool.h>
#include <stdint.h>
#include <SockMonitor.h>

//-----------------------------------------------------------------------------//
//  FUNCTION NAME: NAND_initialize
//
//           INPUTS: None
//
//          OUTPUTS: None
//
//   FUNCTIONALITY: Delay at power-up, then reset flash chip
```

```
42 //-------------------------------------------------------------------------//
43 void NAND_initialize(void)
44 {
45   __delay_cycles(800);                    //Wait 50uSec for power
46   while((P1IN&RB) == 0){}                 //Wait until flash is ready
47   NAND_reset();
48 }
49
50 //-------------------------------------------------------------------------//
51 //  FUNCTION NAME: NAND_reset
52 //
53 //          INPUTS: None
54 //
55 //         OUTPUTS: None
56 //
57 //   FUNCTIONALITY: Enable writing, then send reset command
58 //-------------------------------------------------------------------------//
59 void NAND_reset(void)
60 {
61   NAND_WriteCommand(0xFF,true);
62   NAND_ProgEnable();
63   NAND_EraseEnable();
64 }
65
66 //-------------------------------------------------------------------------//
67 //  FUNCTION NAME: NAND_WriteCommand
68 //
69 //          INPUTS: in - Command byte to be transmitted
70 //                  ceOn - Bool value to leave CE on or keep it off
71 //
72 //         OUTPUTS: None
73 //
74 //   FUNCTIONALITY: Transmit command byte
75 //-------------------------------------------------------------------------//
76 void NAND_WriteCommand(unsigned char in,bool ceOn)
77 {
78   P9DIR = OUTPUT;
79
80   CLE_ON;
81   CE_OFF;
82
83   P9OUT = in;
84
85   WE_OFF;
86   __delay_cycles(CYCDELAY);
87   WE_ON;
88
89   if(ceOn)
90   {
91     CE_ON;
92   }
93   CLE_OFF;
94 }
95
```

```
96  //-------------------------------------------------------------------------//
97  //  FUNCTION NAME: NAND_WriteByte
98  //
99  //          INPUTS: in - Byte to be written
100 //
101 //         OUTPUTS: None
102 //
103 //   FUNCTIONALITY: Write a single byte
104 //-------------------------------------------------------------------------//
105 void NAND_WriteByte(unsigned char in)
106 {
107   P9DIR = OUTPUT;
108
109   P9OUT = in;
110   WE_OFF;
111    __delay_cycles(CYCDELAY);
112   WE_ON;
113 }
114
115 //-------------------------------------------------------------------------//
116 //  FUNCTION NAME: NAND_WriteWord
117 //
118 //          INPUTS: in - Word to be written MSB first
119 //
120 //         OUTPUTS: None
121 //
122 //   FUNCTIONALITY: Write a single word
123 //-------------------------------------------------------------------------//
124 void NAND_WriteWord(uint16_t in)
125 {
126   uint8_t in1,in2;
127   in1 = (in & 0xFF00) >> 8;
128   in2 = in & 0xFF;
129
130   P9DIR = OUTPUT;
131
132   P9OUT = in1;
133   WE_OFF;
134    __delay_cycles(CYCDELAY);
135   WE_ON;
136
137   P9OUT = in2;
138   WE_OFF;
139    __delay_cycles(CYCDELAY);
140   WE_ON;
141 }
142
143 //-------------------------------------------------------------------------//
144 //  FUNCTION NAME: NAND_ReadBytes
145 //
146 //          INPUTS: Number of bytes to be read
147 //
148 //         OUTPUTS: out - Bytes received from flash
149 //
```

```
150 //   FUNCTIONALITY: Recover single byte transmitted by flash chip
151 //-------------------------------------------------------------------------//
152 void NAND_ReadBytes(uint16_t num,unsigned char *out)
153 {
154   P9DIR = INPUT;
155   unsigned int i;
156
157   CE_OFF;
158   for(i=0;i<num;i++)
159   {
160     RE_OFF;
161     __delay_cycles(CYCDELAY);
162     RE_ON;
163     out[i] = P9IN;
164   }
165 }
166
167 //-------------------------------------------------------------------------//
168 //  FUNCTION NAME: NAND_ReadID
169 //
170 //          INPUTS: dataOut - Pointer to empty array to be filled
171 //
172 //         OUTPUTS: None
173 //
174 //   FUNCTIONALITY: Fill array with Device ID
175 //-------------------------------------------------------------------------//
176 void NAND_ReadID(unsigned char *dataOut)
177 {
178   CLE_ON;
179   CE_OFF;
180   NAND_WriteByte(0x90);
181   CLE_OFF;
182
183   ALE_ON;
184   NAND_WriteByte(0x00);
185   ALE_OFF;
186   __delay_cycles(2);
187
188   while((P1IN&RB) == 0){}
189   NAND_ReadBytes(5,dataOut);
190   CE_ON;
191 }
192
193 //-------------------------------------------------------------------------//
194 //  FUNCTION NAME: NAND_ReadStatus
195 //
196 //          INPUTS: None
197 //
198 //         OUTPUTS: out - Current system status (single byte)
199 //
200 //   FUNCTIONALITY: Determine System status and return resulting byte
201 //-------------------------------------------------------------------------//
202 unsigned char NAND_ReadStatus(void)
203 {
```

```
204   unsigned char out[1];
205
206   NAND_WriteCommand(0x70,true);
207   NAND_ReadBytes(1,out);
208   CE_ON;
209
210   return(out[0]);
211 }
212
213 //-------------------------------------------------------------------------//
214 //  FUNCTION NAME: NAND_WriteAddress
215 //
216 //          INPUTS: col1, col2 - Column address bytes
217 //                  row1 - row3 - Row address bytes
218 //
219 //         OUTPUTS: None
220 //
221 //   FUNCTIONALITY: Write address that is passed through the variables
222 //-------------------------------------------------------------------------//
223 void NAND_WriteAddress(unsigned char col1,unsigned char col2,unsigned char row1,
224                        unsigned char row2,unsigned char row3)
225 {
226   ALE_ON;
227   NAND_WriteByte(col1);
228   NAND_WriteByte(col2);
229   NAND_WriteByte(row1);
230   NAND_WriteByte(row2);
231   NAND_WriteByte(row3);
232   ALE_OFF;
233 }
234
235 //-------------------------------------------------------------------------//
236 //  FUNCTION NAME: NAND_ReadData
237 //
238 //          INPUTS: col1, col2 - Column address bytes
239 //                  row1 - row3 - Row address bytes
240 //                  Size - Length in bytes to be read from flash
241 //                  dataOut - Empty array to be filled with retreived data
242 //
243 //         OUTPUTS: None
244 //
245 //   FUNCTIONALITY: Read data of length Size from given address
246 //-------------------------------------------------------------------------//
247 void NAND_ReadData(unsigned char col1,unsigned char col2,unsigned char row1,
248                    unsigned char row2,unsigned char row3,uint16_t Size,
249                    unsigned char *dataOut)
250 {
251   NAND_WriteCommand(0x00,false);
252   NAND_WriteAddress(col1,col2,row1,row2,row3);
253   NAND_WriteCommand(0x30,false);
254
255   while((P1IN&RB) == 0){}
256   NAND_ReadBytes(Size,dataOut);
257   CE_ON;
```

```
258 }
259
260 //-------------------------------------------------------------------------//
261 //  FUNCTION NAME: NAND_ProgData
262 //
263 //          INPUTS: col1, col2 - Column address bytes
264 //                  row1 -> row3 - Row address bytes
265 //                  Size - Length in words to be written to flash
266 //                  dataIn - Array of data to be written to flash
267 //
268 //         OUTPUTS: None
269 //
270 //   FUNCTIONALITY: Send data in dataIn array to flash at given memory address
271 //-------------------------------------------------------------------------//
272 unsigned char NAND_ProgData(unsigned char col1,unsigned char col2,unsigned char
273                             row1,unsigned char row2,unsigned char row3,uint16_t
274                               Size,uint8_t *dataIn)
275 {
276   uint16_t i;
277   unsigned char status[1];
278   uint8_t varOut;
279
280   while((P1IN&RB) == 0){}
281   NAND_WriteCommand(0x80,false);
282   NAND_WriteAddress(col1,col2,row1,row2,row3);
283   for(i=0;i<Size;i++)
284   {
285     varOut = dataIn[i];
286     NAND_WriteByte(varOut);
287   }
288   NAND_WriteCommand(0x10,false);
289   while((P1IN&RB) == 0){}
290   NAND_WriteCommand(0x70,false);
291   NAND_ReadBytes(1,status);
292   CE_ON;
293   return(status[0]);
294 }
295
296 //-------------------------------------------------------------------------//
297 //  FUNCTION NAME: NAND_EraseBlock
298 //
299 //          INPUTS: row1 - row3 - Row address bytes
300 //
301 //         OUTPUTS: Status
302 //
303 //   FUNCTIONALITY: Erase block given by past address
304 //-------------------------------------------------------------------------//
305 uint8_t NAND_EraseBlock(unsigned char row1,unsigned char row2,unsigned char row3)
306 {
307   uint8_t status[1];
308
309   NAND_WriteCommand(0x60,false);
310   ALE_ON;
311   NAND_WriteByte(row1);
```

```
312   NAND_WriteByte(row2);
313   NAND_WriteByte(row3);
314   ALE_OFF;
315   NAND_WriteCommand(0xD0,false);
316   while((P1IN&RB) == 0){}
317   NAND_WriteCommand(0x70,true);
318   NAND_ReadBytes(1,status);
319   return(status[0]);
320 }
321
322 //-------------------------------------------------------------------------//
323 //   FUNCTION NAME:  NAND_ProgEnable
324 //
325 //            INPUTS:  None
326 //
327 //           OUTPUTS:  None
328 //
329 //    FUNCTIONALITY:  Enable programming
330 //-------------------------------------------------------------------------//
331 void NAND_ProgEnable(void)
332 {
333   WP_ON;
334   NAND_WriteByte(0x80);
335   NAND_WriteByte(0x10);
336 }
337
338 //-------------------------------------------------------------------------//
339 //   FUNCTION NAME:  NAND_ProgDisable
340 //
341 //            INPUTS:  None
342 //
343 //           OUTPUTS:  None
344 //
345 //    FUNCTIONALITY:  Disable programming
346 //-------------------------------------------------------------------------//
347 void NAND_ProgDisable(void)
348 {
349   WP_OFF;
350   NAND_WriteByte(0x80);
351   NAND_WriteByte(0x10);
352 }
353
354 //-------------------------------------------------------------------------//
355 //   FUNCTION NAME:  NAND_EraseEnable
356 //
357 //            INPUTS:  None
358 //
359 //           OUTPUTS:  None
360 //
361 //    FUNCTIONALITY:  Enable erase functionality
362 //-------------------------------------------------------------------------//
363 void NAND_EraseEnable(void)
364 {
365   WP_ON;
```

```
366   NAND_WriteByte(0x60);
367   NAND_WriteByte(0xD0);
368 }
369
370 //-------------------------------------------------------------------------//
371 //  FUNCTION NAME: NAND_EraseDisable
372 //
373 //          INPUTS: None
374 //
375 //         OUTPUTS: None
376 //
377 //   FUNCTIONALITY: Disable erase functionality
378 //-------------------------------------------------------------------------//
379 void NAND_EraseDisable(void)
380 {
381   WP_OFF;
382   NAND_WriteByte(0x60);
383   NAND_WriteByte(0xD0);
384 }
385
386 //-------------------------------------------------------------------------//
387 //  FUNCTION NAME: NAND_ReadParameters
388 //
389 //          INPUTS: flashIn - Buffer to be filled with parameters
390 //
391 //         OUTPUTS: None
392 //
393 //   FUNCTIONALITY: Fills buffer with chip parameters
394 //-------------------------------------------------------------------------//
395 void NAND_ReadParameters(unsigned char *flashIn)
396 {
397   NAND_WriteCommand(0xEC, false);
398   ALE_ON;
399   NAND_WriteByte(0x00);
400   ALE_OFF;
401   while((P1IN&RB) == 0){}
402   NAND_ReadBytes(256,flashIn);
403   CE_ON;
404 }
405
406 //-------------------------------------------------------------------------//
407 //  FUNCTION NAME: NAND_DataDump
408 //
409 //          INPUTS: size - Size of file stored in memory in bytes
410 //                  datStat - Data Status structure, contains updated address
411 //
412 //         OUTPUTS: None
413 //
414 //   FUNCTIONALITY: Transmit whole contents of memory over UART to Bluetooth,
415 //                  MSB first
416 //-------------------------------------------------------------------------//
417 void NAND_DataDump(uint32_t size,NANDReg* datStat)
418 {
419   uint16_t pgSt = datStat->page;
```

```
420   uint16_t blSt = datStat->block;
421   uint16_t clAd= datStat->colAd;
422   uint8_t r1;
423   uint8_t r2;
424   uint8_t r3;
425   uint16_t pgNum;
426   uint16_t byteNum;
427
428   bool endPg = false;
429
430   if((pgSt == 0) && (blSt != 0) && (clAd == 0))  //Determine if we are starting on a new
        block
431   {
432     blSt--;
433   }
434
435   size += 14;  //Compensate for transmission of file size
436
437   //Transmit Version string
438   PrintMess(verNum, sizeof(verNum)-1);
439
440   //Transmit size of data stream, MSB first
441   while(!(UCA1IFG&UCTXIFG)){};
442  UCA1TXBUF = (size & 0xFF000000) >> 24;
443   while(!(UCA1IFG&UCTXIFG)){};
444  UCA1TXBUF = (size & 0xFF0000) >> 16;
445   while(!(UCA1IFG&UCTXIFG)){};
446  UCA1TXBUF = (size & 0xFF00) >> 8;
447   while(!(UCA1IFG&UCTXIFG)){};
448  UCA1TXBUF = (size & 0xFF);
449
450   for(uint16_t k=0;k<=blSt;k++)  //Cycles through all blocks in a target
451   {
452     r1 = (k & BLK1_MSK) << BLK1_SFT;
453     r2 = (k & BLK2_MSK) >> BLK2_SFT;
454     r3 = (k & BLK3_MSK) >> BLK3_SFT;
455
456     NAND_WriteCommand(0x00,false);
457     NAND_WriteAddress(0,0,r1,r2,r3);
458     NAND_WriteCommand(0x30,false);
459
460     while((P1IN&RB) == 0){}
461
462     if(k == blSt) //If the current block equals the final block
463     {
464       if(clAd == 0)//If there no bytes occupied in pgSt...
465       {
466         pgNum = pgSt-1;
467       }else //There are bytes occupied in pgSt
468         {
469           pgNum = pgSt;
470         }
471     }else //Else, not on final block
472       {
```

```
473           pgNum = PAGE_NUMS;
474         }
475
476       for(uint16_t j=0;j<=pgNum;j++) //Cycles through all used pages in a block
477       {
478         if((j == pgNum) && (k == blSt))
479         {
480           endPg = true;
481           byteNum = clAd;
482         }else
483           {
484             byteNum = PAGE_SIZE;
485           }
486
487         if((j == PAGE_NUMS) || endPg)
488         {
489           NAND_WriteCommand(0x3F,false);
490           endPg = false;
491         }else
492         {
493           NAND_WriteCommand(0x31,false);
494         }
495
496         while((P1IN&RB) == 0){}
497
498         P9DIR = INPUT;
499
500         CE_OFF;
501         for(uint16_t i=0x0000;i<byteNum;i++) //Collects all Bytes in a page
502         {
503           RE_OFF;
504           __delay_cycles(CYCDELAY);
505           RE_ON;
506           while(!(UCA1IFG&UCTXIFG)){};
507           UCA1TXBUF = P9IN; //Transmit Byte
508         }
509         CE_ON;
510       }
511     }
512 }
513
514 //-------------------------------------------------------------------------//
515 //  FUNCTION NAME: NAND_PushData
516 //
517 //          INPUTS: size - Length of buffer as word
518 //                  dataIn - Buffer of data that is to be recorded
519 //
520 //         OUTPUTS: None
521 //
522 //   FUNCTIONALITY: Fills NAND with data passed by buffer
523 //-------------------------------------------------------------------------//
524 void NAND_PushData(uint16_t size,NANDReg* datStat,uint8_t *dataIn)
525 {
526   uint16_t pageSt = datStat->page; //Page to start on
```

```
527   uint16_t colAd = datStat->colAd; //Column address to start on
528   uint32_t rowAd = datStat->rowAd; //Row address to start on
529   uint16_t i;                      //Counter for multi-page/block writes
530   uint8_t status[1];               //Used for write status
531   uint16_t bytEnd;
532   unsigned char c1 ;
533   unsigned char c2;
534   unsigned char r1;
535   unsigned char r2;
536   unsigned char r3;
537   enum swTyp{on_page,page_over,block_over};
538   enum swTyp writeTyp = on_page; //Set default to on page
539
540   //Start decision logic here
541   if((colAd+size)>PAGE_SIZE) //Will there be a page overflow?
542   {
543     writeTyp = page_over;
544     if(pageSt==PAGE_NUMS)  //Are we on the last page and causing a block overflow?
545     {
546       writeTyp = block_over;
547     }
548   }
549
550   c1 = colAd & 0x00FF;
551   c2 = (colAd & 0xFF00) >> 8;
552   r1 = rowAd & 0x0000FF;
553   r2 = (rowAd & 0x00FF00) >> 8;
554   r3 = (rowAd & 0xFF0000) >> 16;
555
556   //End decision logic here
557
558   switch(writeTyp)
559   {
560   case on_page:
561       status[0] = NAND_ProgData(c1,c2,r1,r2,r3,size,dataIn);
562       break;
563
564   case page_over:
565       while((P1IN&RB) == 0){}
566       NAND_WriteCommand(0x80,false);
567       NAND_WriteAddress(c1,c2,r1,r2,r3);
568       bytEnd = PAGE_SIZE-colAd;          // Find number of bytes left on page "n"
569
570       for(i=0;i < bytEnd;i++)
571       {
572         //varOut = dataIn[i]; Possibly delete this if below works
573         NAND_WriteByte(dataIn[i]);          //Write data to page n on FLASH
574       }
575
576       NAND_WriteCommand(0x15,false);
577       r1++; //Define explicitly to avoid errors
578       while((P1IN&RB) == 0){}
579       NAND_WriteCommand(0x80,false);
580       NAND_WriteAddress(0,0,r1,r2,r3);
```

```
581
582         for(i = bytEnd;i < size;i++)
583         {
584           //varOut = dataIn[i];
585           NAND_WriteByte(dataIn[i]);          //Write rest of buffer to memory
586         }
587
588         NAND_WriteCommand(0x10,false);
589         while((P1IN&RB) == 0){}
590         NAND_WriteCommand(0x70,false);
591         NAND_ReadBytes(1,status);
592         CE_ON;
593         break;
594
595     case block_over:
596         while((P1IN&RB) == 0){}
597         NAND_WriteCommand(0x80,false);
598         NAND_WriteAddress(c1,c2,r1,r2,r3);
599         bytEnd = PAGE_SIZE-colAd;
600
601         for(i=0;i < bytEnd;i++)
602         {
603           //varOut = dataIn[i];
604           NAND_WriteByte(dataIn[i]);          //Write data to end of page (and block)
605         }
606
607         NAND_WriteCommand(0x15,false);
608         rowAd++;
609         r1 = rowAd & ROW1MSK;
610         r2 = (rowAd & ROW2MSK) >> 8;
611         r3 = (rowAd & ROW3MSK) >> 16;
612         while((P1IN&RB) == 0){}
613         NAND_WriteCommand(0x80,false);
614         NAND_WriteAddress(0,0,r1,r2,r3);
615
616         for(i = bytEnd;i < size;i++)
617         {
618           //varOut = dataIn[i];
619           NAND_WriteByte(dataIn[i]);          //Write rest of buffer
620         }
621
622         NAND_WriteCommand(0x10,false);
623         while((P1IN&RB) == 0){}
624         NAND_WriteCommand(0x70,false);
625         NAND_ReadBytes(1,status);
626         CE_ON;
627         break;
628
629     default:                                  //Error, should only be seen if buffer
630       STAT2_ON;                               // size is changed or if the buffer is
631       STAT1_ON;                              // so large that it covers more than two
632       break;                                  // pages
633     }
634
```

```
635     findNew_Address(datStat, size);      //Update datStat with new start address
636 }
637
638 //-------------------------------------------------------------------------------//
639 //  FUNCTION NAME: findNew_Address
640 //
641 //          INPUTS: datStat - Data status structure that contains the new
642 //                            starting page, block, column address, and row
643 //                            address
644 //                  size - Length of data added in number of byte, value is less
645 //                         than 1 page length which is 2048 bytes
646 //
647 //         OUTPUTS: None
648 //
649 //   FUNCTIONALITY: Fills out data status array with new address information
650 //-------------------------------------------------------------------------------//
651 void findNew_Address(NANDReg* datStat, uint16_t size)
652 {
653   uint16_t pgSt = datStat->page;    //Current page start to be updated
654   uint16_t clAd = datStat->colAd;    //Current column address to be updated
655
656   if((clAd + size) > PAGE_SIZE) //Find new page start address if there is an overrun
657   {
658     datStat->colAd = (clAd + size) - PAGE_SIZE;
659
660     if(pgSt != PAGE_NUMS)  //Is the overrun into the next page? If yes...
661     {
662       datStat->page++;
663       datStat->rowAd++; //Update row address
664     }else //Overrun is into next block
665       {
666         datStat->page = 0;
667         datStat->block++; //Update Block address
668         datStat->rowAd++; //Update row address
669       }
670   }else //There is no overrun and the start byte is simply incremented
671     {
672       datStat->colAd = clAd + size;
673     }
674 }
```

D.11 NAND driver.h

```
1 #include <msp430f5638.h>
2 #include <stdbool.h>
3 #include <stdint.h>
4
5 #ifndef NAND_H_GUARD
6 #define NAND_H_GUARD
7
8 typedef struct {
9
```

```
10    uint16_t page;                    //Page to start writing on
11    uint16_t block;                   //Block/Plane to start writing on
12    uint16_t colAd;                   //Column address, bytes in page
13    uint32_t rowAd;                   //Row address, page and block/plane
14
15  }NANDReg;
16
17  void NAND_initialize(void);
18  void NAND_reset(void);
19  void NAND_WriteCommand(unsigned char,bool);
20  void NAND_WriteByte(unsigned char);
21  void NAND_WriteWord(uint16_t);
22  void NAND_ReadBytes(uint16_t,unsigned char *);
23  void NAND_ReadID(unsigned char *);
24  unsigned char NAND_ReadStatus(void);
25  void NAND_WriteAddress(unsigned char,unsigned char,unsigned char,unsigned char,
26                         unsigned char);
27  void NAND_ReadData(unsigned char,unsigned char,unsigned char,unsigned char,
28                     unsigned char ,uint16_t,unsigned char *);
29  unsigned char NAND_ProgData(unsigned char,unsigned char,unsigned char,unsigned
30                              char,unsigned char ,uint16_t,uint8_t *);
31  uint8_t NAND_EraseBlock(unsigned char,unsigned char,unsigned char);
32  void NAND_ProgEnable(void);
33  void NAND_ProgDisable(void);
34  void NAND_EraseEnable(void);
35  void NAND_EraseDisable(void);
36  void NAND_ReadParameters(unsigned char *);
37  void NAND_DataDump(uint32_t,NANDReg* );
38  void NAND_PushData(uint16_t,NANDReg*,uint8_t *);
39  void findNew_Address(NANDReg*,uint16_t);
40
41
42
43  //These definitions are specific to the Sock Monitor V1.2
44  #define RB          0x20
45  #define RE          0x40
46  #define CE          0x80
47  #define CLE         0x01
48  #define ALE         0x02
49  #define WE          0x04
50  #define WP          0x08
51  #define IO0         0x01
52  #define IO1         0x02
53  #define IO2         0x04
54  #define IO3         0x08
55  #define IO4         0x10
56  #define IO5         0x20
57  #define IO6         0x40
58  #define IO7         0x80
59
60  #define CYCDELAY  5
61  #define INPUT     0
62  #define OUTPUT    0xFF
63
```

```
64 /*
65 Obsolete page size and page numbers
66 #define PAGE_SIZE 0x800
67 #define PAGE_NUMS 0x3F
68 */
69
70 #define PAGE_SIZE 0x1000                    // Number of bytes in a page -1 (4096)
71 #define PAGE_NUMS 0x7F                      // Number of pages in a block -1 (128)
72 #define NUM_BLOCKS 4096                     // Number of blocks in a chip
73
74 #define COL1_MSK 0x00FF
75 #define COL2_MSK 0x1F00
76 #define ROW1_MSK 0x000000FF
77 #define ROW2_MSK 0x0000FF00
78 #define ROW3_MSK 0x00070000
79
80 #define BLK1_MSK 0x0001
81 #define BLK2_MSK 0x01FE
82 #define BLK3_MSK 0x0E00
83
84 #define BLK1_SFT 7
85 #define BLK2_SFT 1
86 #define BLK3_SFT 9
87
88
89 //Define Macros
90 #define CE_ON     (P1OUT |= CE)
91 #define CE_OFF    (P1OUT &= ~CE)
92
93 #define CLE_ON    (P3OUT |= CLE)
94 #define CLE_OFF   (P3OUT &= ~CLE)
95
96 #define RE_ON     (P1OUT |= RE)
97 #define RE_OFF    (P1OUT &= ~RE)
98
99 #define WE_ON     (P3OUT |= WE)
100 #define WE_OFF    (P3OUT &= ~WE)
101
102 #define WP_ON     (P3OUT |= WP)
103 #define WP_OFF    (P3OUT &= ~WP)
104
105 #define ALE_ON    (P3OUT |= ALE)
106 #define ALE_OFF   (P3OUT &= ~ALE)
107
108
109 #endif /*NAND_H_GUARD*/
```

D.12 spiutil.c

```
1 #include <msp430f5638.h>
2 #include <stdint.h>
3 #include <spiutil.h>
```

```
4
5  //Function will write one byte at a time as written
6  void SPI_write(uint8_t addr, uint8_t data)
7  {
8    while (!(UCB1IFG&UCTXIFG));
9    SPI_ON;                                  //Unlatch SPI
10   UCB1TXBUF = addr;                         //Or write bit with address
11    while (!(UCB1IFG&UCTXIFG));
12   UCB1TXBUF = data;
13    while(UCB1STAT&UCBUSY);
14   SPI_OFF;
15 }
16
17 //Function will read one or multiple byte at a time
18 int SPI_read(uint8_t addr)
19 {
20
21     while (!(UCB1IFG&UCTXIFG));
22     SPI_ON;                                  //Unlatch SPI
23     UCB1TXBUF = BIT7 | addr;
24     while (!(UCB1IFG&UCTXIFG));
25     UCB1TXBUF = 0x00;                        //Garbage for reading
26     while(UCB1STAT&UCBUSY);
27     SPI_OFF;
28
29     return UCB1RXBUF;
30
31 }
```

D.13 spiutil.h

```
1  #ifndef SPI_WRITE_H_GUARD
2  #define SPI_WRITE_H_GUARD
3  void SPI_write(uint8_t addr, uint8_t data);
4  #endif
5  #ifndef SPI_READ_H_GUARD
6  #define SPI_READ_H_GUARD
7  int SPI_read(uint8_t addr);
8  #endif
9
10
11 //-------------------------------------------------------------------------//
12 //Register definitions
13 //-------------------------------------------------------------------------//
14 #define DEVID           0x00
15 #define THRESH_TAP      0x1D
16 #define OFSX            0x1E
17 #define OFSY            0x1F
18 #define OFSZ            0x20
19 #define DUR             0x21
20 #define LATENT          0x22
21 #define WINDOW          0x23
```

```
22 #define THRESH_ACT          0x24
23 #define THRESH_INACT        0x25
24 #define TIME_INACT          0x26
25 #define ACT_INACT_CTL       0x27
26 #define THRESH_FF           0x28
27 #define TIME_FF             0x29
28 #define TAP_AXES            0x2A
29 #define ACT_TAP_STATUS      0x2B
30 #define BW_RATE             0x2C
31 #define POWER_CTL           0x2D
32 #define INT_ENABLE          0x2E
33 #define INT_MAP             0x2F
34 #define INT_SOURCE          0x30
35 #define DATA_FORMAT         0x31
36 #define DATAX0              0x32
37 #define DATAX1              0x33
38 #define DATAY0              0x34
39 #define DATAY1              0x35
40 #define DATAZ0              0x36
41 #define DATAZ1              0x37
42 #define FIFO_CTL            0x38
43 #define FIFO_STATUS         0x39
44
45 //----------------------------------------------------------------------------//
46 //Bit definitions
47 //----------------------------------------------------------------------------//
48
49 //BW_RATE
50 #define LOW_POWER           0x10
51 #define RATE_3200           15
52 #define RATE_1600           14
53 #define RATE_800            13
54 #define RATE_400            12
55 #define RATE_200            11
56 #define RATE_100            10
57 #define RATE_50             9
58 #define RATE_25             8
59 #define RATE_12             7
60 #define RATE_6              6
61 #define RATE_3              5
62 #define RATE_1              4
63 #define RATE_078            3
64 #define RATE_039            2
65 #define RATE_020            1
66 #define RATE_010            0
67
68 //POWER_CTL
69 #define LINK                0x20
70 #define AUTO_SLEEP          0x10
71 #define MEASURE             0x08
72 #define SLEEP               0x04
73 #define WAKEUP_8            0x00
74 #define WAKEUP_4            0x01
75 #define WAKEUP_2            0x02
```

```
76 #define WAKEUP_1            0x03
77
78 //INT_ENABLE, INT_MAP, INT_SOURCE
79 #define DATA_READY          0x80
80 #define SINGLE_TAP          0x40
81 #define DOUBLE_TAP          0x20
82 #define ACTIVITY            0x10
83 #define INACTIVITY          0x08
84 #define FREE_FALL           0x04
85 #define WATERMARK           0x02
86 #define OVERRUN             0x01
87
88 //DATA_FORMAT
89 #define SELF_TEST           0x80
90 #define SPI                 0x40
91 #define INT_INVERT          0x20
92 #define FULL_RES            0x08
93 #define JUSTIFY             0x04
94 #define RANGE_2             0x00
95 #define RANGE_4             0x01
96 #define RANGE_8             0x02
97 #define RANGE_16            0x03
98
99 //FIFO_CTL
100 #define BYPASS             0x00
101 #define FIFO               0x01
102 #define STREAM             0x02
103 #define TRIGGER_MODE       0x03
104 #define TRIGGER            0x20
105
106 //OTHER DEFINITIONS
107 #define SPI_ON  (P8OUT &= ~BIT1)
108 #define SPI_OFF  (P8OUT |= BIT1)
```

Appendix E

POST PROCESSOR CODE

E.1 SM Analyzer.m

```
% This script further analyzes the post processed data from the Sock
% Monitor v1.3.
%
% Written by David Swartzendruber
% University of Washington, April 2013

close all
clear all
clc

FSR_XMIN = 0;
FSR_XMAX = 10E7;
FSR_YMIN = 0;
FSR_YMAX = 4000;

ACC_XMIN = 0;
ACC_XMAX = 10E7;
ACC_YMIN = -3000;
ACC_YMAX = 3000;

MIN_CONST = .999999;
MAX_CONST = 1.000001;

FSR_THRESHOLD = 50; %Same value as in firmware
THIRTM_WIN = datenum(2000,1,1,12,30,0)-datenum(2000,1,1,12,0,0);
TWOS_WIN = datenum(2000,1,1,12,0,2)-datenum(2000,1,1,12,0,0); %Generic 2 second time value
ONES_WIN = datenum(2000,1,1,12,0,1)-datenum(2000,1,1,12,0,0);
HALFS_WIN = datenum(2000,1,1,12,0,0.5)-datenum(2000,1,1,12,0,0);
```

```
29 FIVES_WIN = 5; %For use in pilot data analysis
30
31 response = input('Would you like to import a processed file (1) or a new file (2)?\n');
32
33 switch response
34     case 1
35         response = input('Is this data from the pilot study (T)rue or (F)alse?\n','s');
36         [fname,pname] = uigetfile({'.mat'},'Select the processed Sock Monitor file');
37         cd(pname);
38         data = importdata(fname);
39         if strcmpi(response,'T')
40             commandwindow
41             stHr = input('What was the start hour?\n');
42             commandwindow
43             stMin = input('What was the start minute?\n');
44             stTime = stHr * 60 + stMin;
45             stDate = datenum(2000,1,1,0,stTime,0);
46
47             %Unpack data
48             FSR = data.analog(:,3);
49             fsrTime = data.time;
50             tagTime = [data.serial.raw{:,1}]'./1000;
51             totalTags = data.serial.count;
52             uniqueTags = data.serial.TagCounts.key';
53             uniqueTags{1,2} = data.serial.TagCounts.Data';
54             IDind = size(uniqueTags);
55             IDind(2) = [];
56
57             %Do a median filter on the FSR data from the posterior trimline FSR
58             %then find the donning and doffing points.
59             filtFSR = medfilt1(FSR(:,1),10);
60             thshInd = find(filtFSR(:,1) < FSR_THRESHOLD);
61             ddInd = find(diff(thshInd) > 25);
62             for i=1:length(ddInd)
63                 donDoffInd(i,1) = fsrTime(thshInd(ddInd(i))); %off->on
64                 donDoffInd(i,2) = fsrTime(thshInd(ddInd(i)+1)); %on->off
```

```
65              end
66
67              for i=1:length(donDoffInd(:,1))-1
68                  if (donDoffInd(i+1,1)-donDoffInd(i,2)) < 2
69                      donDoffInd(i,2) = NaN;
70                      donDoffInd(i+1,1) = NaN;
71                  end
72              end
73
74              %Get rid of NaN values
75              tempVar = reshape(donDoffInd,length(donDoffInd(:,1))*2,1);
76              tempVar = tempVar(~isnan(tempVar),1);
77              donDoffInd = reshape(tempVar,length(tempVar)/2,2);
78
79              figure
80              h1 = plot(fsrTime,FSR(:,1));
81              hold on
82              h2 = plot(donDoffInd(:,1),200*ones(length(donDoffInd(:,1))),'gx','MarkerSize',10)
    ;
83              hold on
84              h3 = plot(donDoffInd(:,2),200*ones(length(donDoffInd(:,2))),'rx','MarkerSize',10)
    ;
85              hold on
86              h4 = plot(tagTime,300*totalTags(:,1),'go','MarkerSize',10);
87              for i=1:length(donDoffInd(:,1))
88                  hold on
89                  line([donDoffInd(i,1)-FIVES_WIN,donDoffInd(i,1)-FIVES_WIN],[0,1200],'Color','
    red')
90                  hold on
91                  line([donDoffInd(i,1)+FIVES_WIN,donDoffInd(i,1)+FIVES_WIN],[0,1200],'Color','
    black')
92                  hold on
93                  line([donDoffInd(i,2)-FIVES_WIN,donDoffInd(i,2)-FIVES_WIN],[0,1200],'Color','
    red')
94                  hold on
```

```
95              line([donDoffInd(i,2)+FIVES_WIN,donDoffInd(i,2)+FIVES_WIN],[0,1200],'Color','
    black')
96          end
97          ax = axis;
98          axis([ax(1:2),0,1000]);
99          xlabel('Time (seconds)');
100         ylabel('ADC counts for FSR - 2.93 mV/count')
101         legend([h1 h2(1) h3(1) h4],'FSR','Doff to Don','Don to Doff','Tag at 300 spacing'
    )
102
103         %Extract success rate at detecting tags
104         detectSuc = zeros(length(donDoffInd(:,1)),2);
105         for i = 1:length(donDoffInd(:,1))
106             for j = 1:length(tagTime)
107                 %Handles doff->don
108                 if (tagTime(j) >= donDoffInd(i,1)) && (tagTime(j) <= donDoffInd(i,2)) &&
    (totalTags(j) >=1)
109                     detectSuc(i,1) = detectSuc(i,1) + 1;
110                 end
111             end
112         end
113
114         %Look for false positive tags
115         for i = 1:length(donDoffInd(:,1))-1
116             for j = 1:length(tagTime)
117                 %Handles doff->don
118                 if (tagTime(j) > donDoffInd(i,2)) && (tagTime(j) < donDoffInd(i+1,1)) &&
    (totalTags(j) >=1)
119                     detectSuc(i,2) = detectSuc(i,2) + 1;
120                 end
121             end
122         end
123
124         %Convert seconds to actual time
125         donDoffInd(:,1) = datenum(0,0,0,0,0,donDoffInd(:,1))+stDate;
126         donDoffInd(:,2) = datenum(0,0,0,0,0,donDoffInd(:,2))+stDate;
```

```
127
128
129             %Get independent days
130             dCt = 2;
131             prevDay = datestr(donDoffInd(1,1),'dd-mmm-yyyy');
132             dName{1,1} = prevDay;
133             dName{1,2} = [1,1];
134             for i=1:length(donDoffInd(:,1))
135                 for j=1:2
136                     curDay = datestr(donDoffInd(i,j),'dd-mmm-yyyy');
137                     if ~isequal(curDay,prevDay)
138                         dName{dCt,1} = curDay;
139                         dName{1,2}(dCt,:) = [i,j]; %Store index of day change
140
141                         dCt = dCt + 1;
142                     end
143                     prevDay = curDay;
144                 end
145             end
146
147
148             %Find number of dons and doffs per day
149             clear curDay
150             prevDay = datestr(donDoffInd(1,1),'dd-mmm-yyyy');
151             dCt = 1;
152             days = size(dName);
153             dDays = zeros(days(1),2);
154             for i=1:length(donDoffInd(:,1))
155                 for j=1:2
156                     curDay = datestr(donDoffInd(i,j),'dd-mmm-yyyy');
157                     if  isequal(curDay,prevDay) %If same
158                         dDays(dCt,j) = dDays(dCt,j) + 1;
159                     else                        %If different day
160                         dCt = dCt + 1;
161                         dDays(dCt,j) = dDays(dCt,j) + 1;
162                     end
```

```
163                 prevDay = curDay;
164             end
165         end
166
167         %Generate histogram data for plotting dons and doffs
168         histData = zeros(length(donDoffInd(:,1)),2); %Bin corresponding to every half
    hour between midnight and midnight
169         histData(:,1) = (str2num(datestr(donDoffInd(:,1),'HH'))*60 ...
170         + str2num(datestr(donDoffInd(:,1),'MM')))./60; %Dons
171         histData(:,2) = (str2num(datestr(donDoffInd(:,2),'HH'))*60 + ...
172             str2num(datestr(donDoffInd(:,2),'MM')))./60; %Doffs
173
174         %Generate histogram data for plotting dons/doffs per day
175         dNameSize = size(dName);
176         for i = 1:dNameSize(1)-1 %For number of days of data
177             if dName{1,2}(i+1,2) == 1
178                 b = dName{1,2}(i+1,1)-1;
179             else
180                 b = dName{1,2}(i+1,1);
181             end
182             histDay{1,i} = (str2num(datestr(donDoffInd(dName{1,2}(i,1):b,1),'HH'))*60 +
    str2num(datestr(donDoffInd(dName{1,2}(i,1):b,1),'MM')))./60; %Dons, time is in hours
183             histDay{2,i} = (str2num(datestr(donDoffInd(dName{1,2}(i,1):dName{1,2}(i+1,1)
    -1,2),'HH'))*60 + str2num(datestr(donDoffInd(dName{1,2}(i,1):dName{1,2}(i+1,1)-1,2),'MM'
    )))./60; %Doffs, time is in hours
184         end
185         i=dNameSize(1);
186         if dName{1,2}(end,2) == 1
187             b =  dName{1,2}(i,1);
188         else
189             b =  dName{1,2}(i,1) + 1;
190         end
191         histDay{1,i}= (str2num(datestr(donDoffInd(b:end,1),'HH'))*60 + str2num(datestr(
    donDoffInd(b:end,1),'MM')))./60; %Dons, time is in hours
192         histDay{2,i}= (str2num(datestr(donDoffInd(dName{1,2}(i,1):end,2),'HH'))*60 +
    str2num(datestr(donDoffInd(dName{1,2}(i,1):end,2),'MM')))./60; %Dons, time is in hours
```

```
193
194             %Calculate success
195             donSuc = sum(detectSuc(:,1)~=0)/length(detectSuc(:,1));
196             falsePos = sum(detectSuc(:,2));
197
198             %Plot histogram of when subject don and doffs
199             figure
200             maxVal = 0;
201             clear h;
202             subplot(2,1,1)
203             hist(histData(:,1),.25:.5:23.75)
204             h(1) = gca;
205             a = ylim;
206             if maxVal < a(2)
207                 maxVal = a(2);
208             end
209             title('Don and Doffs vs Time')
210             ylabel('Number of Dons')
211             set(gca,'XTick',[0:2:24])
212             subplot(2,1,2)
213             hist(histData(:,2),.25:.5:23.75)
214             h(2) = gca;
215             a = ylim;
216             if maxVal < a(2)
217                 maxVal = a(2);
218             end
219             xlabel('Time (midnight to midnight)')
220             ylabel('Number of Doffs')
221             set(gca,'XTick',[0:2:24])
222             set(h,'YLim',[0,maxVal + 1]);
223
224             %Plot individual don/doff plots per day
225             histDaySize = size(histDay);
226             maxVal = 0;
227             clear h;
228             figure
```

```
229             for i = 1:histDaySize(2)
230                 h(2*i-1) = subplot(histDaySize(2),2,2*i-1);
231                 hist(histDay{1,i}(:,1),.25:.5:23.75)
232                 set(gca,'XTick',[0:2:24])
233                 ylabel('Number of Dons')
234                 a = ylim;
235                 if maxVal < a(2)
236                     maxVal = a(2);
237                 end
238                 h(2*i) = subplot(histDaySize(2),2,2*i);
239                 hist(histDay{2,i}(:,1),.25:.5:23.75)
240                 ylabel('Number of Doffs')
241                 set(gca,'XTick',[0:2:24])
242                 a = ylim;
243                 if maxVal < a(2)
244                     maxVal = a(2);
245                 end
246             end
247             [ax1,h1]=suplabel('Dons and Doffs by Day','t');
248             [ax2,h2]=suplabel('Time (midnight to midnight)');
249             set(h1,'FontSize',14)
250             set(h2,'FontSize',12)
251             set(h,'YLim',[0,maxVal + 1]);
252
253             %Display results
254             clc
255             fprintf('Most tags detected at once:          %0.0f\n',...
256                 max(data.serial.count));
257             fprintf('Number of independent tags:          %0.0f\n',...
258                 length([data.serial.TagCounts.key]));
259             fprintf('\n');
260             fprintf('Tags detected\tNumber detected\n%s\t\t\t\t%i\n',...
261                     char(uniqueTags{1,1}),double(uniqueTags{1,2}(1,1)));
262             if IDind ~= 1
263                 for i = 2:(IDind)
```

```
264                     fprintf('%s\t\t\t\t%i\n',char(uniqueTags{i,1}),double(uniqueTags{1,2}(i
    ,1)));
265                 end
266             end
267             fprintf('\n');
268             fprintf('Time and date for donnings and doffings: \n');
269             fprintf('          Don                          Doff\n');
270             for i=1:length(donDoffInd(:,1))
271                 fprintf('%s          %s\n',datestr(donDoffInd(i,1)),...
272                     datestr(donDoffInd(i,2)));
273             end
274             fprintf('\n');
275             fprintf('Don detection success:                %3.0f%%\n',...
276                 donSuc*100);
277             fprintf('False positive detections:            %3.0f\n',...
278                 falsePos);
279
280
281         else
282             %Unpack data
283             out = data.raw;
284             FSR = data.activity.FSR;
285             tagIDs = data.sock.IDs;
286             tagCt = data.sock.tagCt;
287             tvec = data.time;
288             tagPlot = data.sock.tagPlot;
289             xacc = data.activity.xaccel;
290             yacc = data.activity.yaccel;
291             zacc = data.activity.zaccel;
292
293             FSRD = downsample(FSR,2);
294
295             %Pull out the various parts of the in-lab test
296
297             figure
298             plot((1:1:length(FSR))./50,FSR)
```

```
299             title('Oversampled FSR data'); xlabel('seconds'); ylabel('Counts from ADC')
300
301             [don1x,don1y] = ginput;
302
303
304             t = [downsample([don1(:,2);NaN(maxSize-length(don1(:,2)),1)]',2);
305             downsample([don2(:,2);NaN(maxSize-length(don2(:,2)),1)]',2);
306             downsample([don3(:,2);NaN(maxSize-length(don3(:,2)),1)]',2);
307             downsample([don4(:,2);NaN(maxSize-length(don4(:,2)),1)]',2)];
308
309             %Spectrograms
310             figure
311             spectrogram(data.activity.zaccel(4E4:8E4),128,100,128,50);
312             view(90,90);
313         end
314     case 2
315         [fname1,pname1] = uigetfile({'.csv'},'Select the activity CSV file');
316         [fname2,pname2] = uigetfile({'.csv'},'Select the tag CSV file');
317         cd(pname1)
318
319         saveFlg = 0;
320         saveName = strcat(fname1,'_Data');
321         try
322             cd('..')            %Back out one folder
323             cd('Analysis')      %Open Analysis folder to check for analyzed file
324         catch err
325             display('Out of normal folder structure');
326         end
327         if exist(strcat(saveName,'.mat'),'file') ~= 2
328             saveFlg = 1;
329         end
330
331         cd(pname1)
332
333         actData = csvread(fname1);
334
```

```
335         rawTag = importdata(fname2);
336
337         parfor i=1:length(rawTag)
338             tagData(i,:) = textscan(rawTag{i},...
339                 '%u8 %u8 %u16 %u8 %u8 %u8 %u16 %s %s %s %s %s %s %s %s %s %s',...
340                 'Delimiter',',');
341         end
342
343         %Extract Tag data
344         totalTags = zeros(length(tagData(:,1)),2);
345
346         for i = 1:length(tagData(:,1))
347             for j = 1:10
348                 if ~strcmp(tagData{i,7+j},'')
349                     totalTags(i,1) = totalTags(i,1) + 1;
350                 end
351             end
352         end
353
354 %% Clean and Verify Data
355         %Verify and fix time data
356         hE = 1; %Hour error counter
357         mE = 1; %Minute error counter
358         sE = 1; %Second error counter
359         ct = 1;
360         for i=1:length(actData(:,1))
361             %Check if Hour is in bounds, correct if able
362             if (actData(i,4) < 0) || (actData(i,4) > 23)
363                 if actData(i-1,4) == actData(i+1,4)
364                     actData(i,4) = actData(i-1,4);
365                 else
366                     timeErr(1,hE) = i;
367                     hE = hE + 1;
368                 end
369             end
370
```

```
371             %Check if Minute is in bounds, correct if able
372             if (actData(i,5) < 0) || (actData(i,5) > 59)
373                 if actData(i-1,5) == actData(i+1,5)
374                     actData(i,5) = actData(i-1,5);
375                 else
376                     timeErr(2,mE) = i;
377                     mE = mE + 1;
378                 end
379             end
380
381             %Check if Second is in bounds, correct if able
382             if (actData(i,6) < 0) || (actData(i,6) > 59)
383                 if actData(i-1,6) == actData(i+1,6)
384                     actData(i,6) = actData(i-1,6);
385                 else
386                     timeErr(3,sE) = i;
387                     sE = sE + 1;
388                 end
389             end
390
391             %Check Accelerometer Data, store index of bad data
392             if((actData(i,8) > 4096 || actData(i,8) < -4096) || ...
393                     (actData(i,9) > 4096 || actData(i,9) < -4096) || ...
394                     (actData(i,10) > 4096 || actData(i,10) < -4096))
395                 delList(ct) = i;
396                 ct = ct + 1;
397             end
398
399             %Check FSR Data, store index of bad data
400             if((actData(i,11) > 4096 || actData(i,11) < 0) || ...
401                     (actData(i,12) > 4096 || actData(i,12) < 0) || ...
402                     (actData(i,13) > 4096 || actData(i,13) < 0)|| ...
403                     (actData(i,14) > 4096 || actData(i,14) < 0))
404                 delList(ct) = i;
405                 ct = ct + 1;
406             end
```

```
407
408         end
409
410
411         %Erase bad accelerometer data
412         if exist('delList','var')
413             actData(delList,:) = [];
414         end
415
416         %Fix any out of place hour values, e.g. 15,15,0,15,15,15, the 0 is
417         %out of place
418         for i=2:length(actData(:,1))-1
419             if (actData(i-1,4) == actData(i+1,4)) && (actData(i,4) ~= actData(i+1,4))
420                 actData(i,4) = actData(i+1,4);
421             end
422         end
423
424         %Clean up time vectors, fix midnight to 0001(24Hr) data
425         dci = 1;
426         for i=1:length(actData(:,1))-1
427             if (actData(i+1,4) < actData(i,4)) && (actData(i+1,4) == 0)
428                 dateChangeIndex(dci) = i;
429                 dci = dci + 1;
430             end
431         end
432
433         if exist('dateChangeIndex','var')
434             for i = 1:length(dateChangeIndex) %For number of days...
435                 refDay = actData(dateChangeIndex(i),1);
436                 curDay = refDay;
437                 ctr = dateChangeIndex(i);
438                 while refDay == curDay %Cycle through until new day found
439                     if(actData(ctr,1) < actData(ctr+1,1))
440                         curDay = curDay + 1;
441                         newDayIndex = ctr + 1;
442                     else
```

```
443                         ctr = ctr + 1;
444                     end
445                 end
446
447                 actData(dateChangeIndex(i)+1:newDayIndex-1,1) = curDay;
448
449             end
450         end
451
452         dayInd = find(diff(actData(:,1)) == 1); %Index of where day changes
453         hourInd = find(diff(actData(:,4)) < 0); %Index of where hour changes
454
455         %Display error/warning message
456         if(min(dayInd == hourInd) == 0)
457             display('Date change DOES NOT match with DAY and HOUR variables!');
458         end
459
460 %% Extract time vectors
461         %Extract valid fsr date/time stamps, where valid is defined as a
462         %value that is not a NaN
463         fsrDay = actData(~isnan(actData(:,11)),1);
464         fsrMonth = actData(~isnan(actData(:,11)),2);
465         fsrYear = actData(~isnan(actData(:,11)),3);
466         fsrHour = actData(~isnan(actData(:,11)),4);
467         fsrMinute = actData(~isnan(actData(:,11)),5);
468         fsrSecond = actData(~isnan(actData(:,11)),6)+(actData(~isnan(...
469             actData(:,11)),7)/125);
470
471         %Create accelerometer date/time vector
472         accTime = datenum(actData(:,3),actData(:,2),actData(:,1),...
473             actData(:,4),actData(:,5),actData(:,6)+...
474             (actData(:,7)/125));
475
476         %Create FSR date/time vector
477         fsrTime = datenum(fsrYear,fsrMonth,fsrDay,fsrHour,fsrMinute...
478             ,fsrSecond);
```

```
479
480         %Create tag date/time vector
481         tagTime = datenum(cast([tagData{:,3}],'double'),...
482             cast([tagData{:,2}],'double'),cast([tagData{:,1}],'double'),...
483             cast([tagData{:,4}],'double'),cast([tagData{:,5}],'double'),...
484             cast([tagData{:,6}],'double')+cast([tagData{:,7}],'double')...
485             /cast(125,'double'));
486
487 %% Extract data
488         %Accelerometer date/time vector
489         xacc = actData(:,8);
490         yacc = actData(:,9);
491         zacc = actData(:,10);
492
493         %Extract FSR data
494         for i = 1:4
495            FSR(:,i) =  actData(~isnan(actData(:,11)),10+i);
496         end
497
498
499
500 %% Extract features and other important data (data mining)
501         %Extract unique tag information
502         IDind = 1; %Index for unique tag
503         uniqueTags = cell(20,2);
504         uniqueTags(:,2) = {0};
505
506         for i = 1:length(totalTags(:,1))
507             for j=1:10
508                 if ~isempty(tagData{i,j+7})
509                     if ~strcmp(tagData{i,j+7},'')  %If there is a tag
510
511                         evalTag = tagData{i,j+7};
512
513                         ind = find(cellfun(@(x) strcmp(x,evalTag), uniqueTags));
514
```

```
515                         if isempty(ind)
516                             uniqueTags{IDind,1} = evalTag;
517                             uniqueTags(IDind,2) = {1};
518                             IDind = IDind + 1;
519                         else
520                             uniqueTags(ind,2) = {uniqueTags{ind,2}+1};
521                         end
522
523                         evalTag = [];
524
525                     end
526                 end
527             end
528         end
529
530         %Do a median filter on the FSR data from the posterior trimline FSR
531         %then find the donning and doffing points.
532         filtFSR = medfilt1(FSR(:,1),10);
533         thshInd = find(filtFSR(:,1) < FSR_THRESHOLD);
534         ddInd = find(diff(thshInd) > 25);
535         for i=1:length(ddInd)
536             donDoffInd(i,1) = fsrTime(thshInd(ddInd(i))); %off->on
537             donDoffInd(i,2) = fsrTime(thshInd(ddInd(i)+1)); %on->off
538         end
539
540         for i=1:length(donDoffInd(:,1))-1
541             if (donDoffInd(i+1,1)-donDoffInd(i,2)) < TWOS_WIN
542                 donDoffInd(i,2) = NaN;
543                 donDoffInd(i+1,1) = NaN;
544             end
545         end
546
547         %Get rid of NaN values
548         tempVar = reshape(donDoffInd,length(donDoffInd(:,1))*2,1);
549         tempVar = tempVar(~isnan(tempVar),1);
550         donDoffInd = reshape(tempVar,length(tempVar)/2,2);
```

```
551
552         %Extract time donned
553         timesDon = length(donDoffInd(:,1)); %Number of independent times donned
554         onTime = donDoffInd(:,2) - donDoffInd(:,1);
555
556         avgDon = datestr(mean(onTime),'HH:MM:SS');
557         minDon = datestr(min(onTime),'HH:MM:SS');
558         maxDon = datestr(max(onTime),'HH:MM:SS');
559         stdDon = datestr(std(onTime),'HH:MM:SS');
560
561         %Extract time doffed
562         timesDoff = length(donDoffInd(:,2)); %Number of independent times donned
563         for i=1:length(donDoffInd(:,1))-1
564             offTime(i) = donDoffInd(i+1,1) - donDoffInd(i,2);
565         end
566
567         avgDoff = datestr(mean(offTime),'HH:MM:SS');
568         minDoff = datestr(min(offTime),'HH:MM:SS');
569         maxDoff = datestr(max(offTime),'HH:MM:SS');
570         stdDoff = datestr(std(offTime),'HH:MM:SS');
571
572
573         %Extract time between changes
574 %           changeInd = doffInd;
575
576         avgDiff = mean(3);
577         minDiff = min(3);
578         maxDiff = max(3);
579         stdDiff = std(3);
580
581
582         %Extract total time worn
583         totalTime = accTime(end) - accTime(1);
584         eTime = [floor(totalTime),datestr(totalTime,'HH'),...
585             datestr(totalTime,'MM'),datestr(totalTime,'SS')];
586
```

```
587         %Extract success rate at detecting tags
588         detectSuc = zeros(length(donDoffInd(:,1)),2);
589         for i = 1:length(donDoffInd(:,1))
590             for j = 1:length(tagTime)
591                 %Handles doff->don
592                 if (tagTime(j) >= donDoffInd(i,1)-TWOS_WIN) && (tagTime(j) <= donDoffInd(i,1)
    +HALFS_WIN)
593                     detectSuc(i,1) = 1;
594                 end
595 
596                 %Handles don->doff
597                 if (tagTime(j) >= donDoffInd(i,2)-HALFS_WIN) && (tagTime(j) <= donDoffInd(i
    ,2)+TWOS_WIN)
598                     detectSuc(i,2) = 1;
599                 end
600             end
601         end
602 
603         %Calculate success
604         donSuc = sum(detectSuc(:,1))/length(detectSuc(:,1));
605         doffSuc = sum(detectSuc(:,2))/length(detectSuc(:,2));
606         totalSuc = sum((detectSuc(:,1)+detectSuc(:,2)))/...
607             (length(detectSuc(:,1))*2);
608 
609         %Calculate number of missed periods of activity
610         failCt = 1;
611         for i=1:length(detectSuc(:,1))
612             if isequal(detectSuc(i,1:2),[0,0])
613                 fail(failCt,1) = i;
614                 failCt = failCt + 1;
615             end
616         end
617 
618         %Get independent days
619         dCt = 2;
620         prevDay = datestr(donDoffInd(1,1),'dd-mmm-yyyy');
```

```
621         dName{1,1} = prevDay;
622         dName{1,2} = [1,1];
623         for i=1:length(donDoffInd(:,1))
624             for j=1:2
625                 curDay = datestr(donDoffInd(i,j),'dd-mmm-yyyy');
626                 if ~isequal(curDay,prevDay)
627                     dName{dCt,1} = curDay;
628                     dName{1,2}(dCt,:) = [i,j]; %Store index of day change
629 
630                     dCt = dCt + 1;
631                 end
632                 prevDay = curDay;
633             end
634         end
635 
636 
637         %Find number of dons and doffs per day
638         clear curDay
639         prevDay = datestr(donDoffInd(1,1),'dd-mmm-yyyy');
640         dCt = 1;
641         days = size(dName);
642         dDays = zeros(days(1),2);
643         for i=1:length(donDoffInd(:,1))
644             for j=1:2
645                 curDay = datestr(donDoffInd(i,j),'dd-mmm-yyyy');
646                 if  isequal(curDay,prevDay) %If same
647                     dDays(dCt,j) = dDays(dCt,j) + 1;
648                 else                        %If different day
649                     dCt = dCt + 1;
650                     dDays(dCt,j) = dDays(dCt,j) + 1;
651                 end
652                 prevDay = curDay;
653             end
654         end
655 
656         %Generate histogram data for plotting dons and doffs
```

```
657         histData = zeros(length(donDoffInd(:,1)),2); %Bin corresponding to every half hour
    between midnight and midnight
658         histData(:,1) = (str2num(datestr(donDoffInd(:,1),'HH')).*60 ...
659         + str2num(datestr(donDoffInd(:,1),'MM'))./60; %Dons
660         histData(:,2) = (str2num(datestr(donDoffInd(:,2),'HH')).*60 + ...
661             str2num(datestr(donDoffInd(:,2),'MM'))./60; %Doffs
662
663         %Generate histogram data for plotting dons/doffs per day
664         dNameSize = size(dName);
665         for i = 1:dNameSize(1)-1
666             if dName{1,2}(i+1,2) == 1
667                 b = dName{1,2}(i+1,1)-1;
668             else
669                 b = dName{1,2}(i+1,1);
670             end
671             histDay{1,i} = (str2num(datestr(donDoffInd(dName{1,2}(i,1):b,1),'HH')).*60 +
    str2num(datestr(donDoffInd(dName{1,2}(i,1):b,1),'MM')))./60; %Dons, time is in hours
672             histDay{2,i} = (str2num(datestr(donDoffInd(dName{1,2}(i,1):dName{1,2}(i+1,1)-1,2)
    ,'HH')).*60 + str2num(datestr(donDoffInd(dName{1,2}(i,1):dName{1,2}(i+1,1)-1,2),'MM')))
    ./60; %Doffs, time is in hours
673         end
674         i=dNameSize(1);
675         if dName{1,2}(end,2) == 1
676             b =  dName{1,2}(i,1);
677         else
678             b =  dName{1,2}(i,1) + 1;
679         end
680         histDay{1,i}= (str2num(datestr(donDoffInd(b:end,1),'HH')).*60 + str2num(datestr(
    donDoffInd(b:end,1),'MM')))./60; %Dons, time is in hours
681         histDay{2,i}= (str2num(datestr(donDoffInd(dName{1,2}(i,1):end,2),'HH')).*60 + str2num
    (datestr(donDoffInd(dName{1,2}(i,1):end,2),'MM')))./60; %Dons, time is in hours
682
683 %% Display Results
684
685         %Uncomment the 19 lines below to see a plot of where the program
686         %thinks the donning and doffing points are. The red X is on->off
```

```
687         %and the green X is off->on
688         figure
689         h1 = plot(fsrTime,FSR(:,1));
690         hold on
691         h2 = plot(donDoffInd(:,1),500*ones(length(donDoffInd(:,1))),'gx','MarkerSize',10);
692         hold on
693         h3 = plot(donDoffInd(:,2),500*ones(length(donDoffInd(:,2))),'rx','MarkerSize',10);
694         hold on
695         h4 = plot(tagTime,800*totalTags(:,1),'go','MarkerSize',10);
696         for i=1:length(donDoffInd(:,1))
697             hold on
698             line([donDoffInd(i,1)-TWOS_WIN,donDoffInd(i,1)-TWOS_WIN],[0,4000],'Color','red')
699             hold on
700             line([donDoffInd(i,1)+HALFS_WIN,donDoffInd(i,1)+HALFS_WIN],[0,4000],'Color','
    black')
701             hold on
702             line([donDoffInd(i,2)-HALFS_WIN,donDoffInd(i,2)-HALFS_WIN],[0,4000],'Color','red'
    )
703             hold on
704             line([donDoffInd(i,2)+TWOS_WIN,donDoffInd(i,2)+TWOS_WIN],[0,4000],'Color','black'
    )
705         end
706         ax = axis;
707         axis([ax(1:2),0,4500]);
708         datetick('x',16,'keepticks')
709         ylabel('ADC counts for FSR - 0.73 mV/count')
710         legend([h1 h2(1) h3(1) h4],'FSR','Doff to Don','Don to Doff','Tag at 800 spacing')
711
712         %Plot accelerometer data
713         figure
714         plot(accTime,xacc,'r')
715         hold on
716         plot(accTime,yacc,'g')
717         hold on
718         plot(accTime,zacc)
719         hold off
```

```
720         axis([accTime(1)*MIN_CONST,accTime(end)*MAX_CONST,...
721             ACC_YMIN,ACC_YMAX])
722         legend('xData','yData','zData')
723         title('Accelerometer Data')
724         datetick('x',16,'keepticks')
725
726         %Plot FSR data
727         figure
728         subplot(2,2,1)
729         plot(fsrTime,FSR(:,1));
730         axis([fsrTime(1)*MIN_CONST,fsrTime(end)*MAX_CONST,...
731             FSR_YMIN,FSR_YMAX])
732         title('FSR 1 data')
733         ylabel('ADC Count')
734         datetick('x',16,'keepticks')
735         subplot(2,2,2)
736         plot(fsrTime,FSR(:,2));
737         axis([fsrTime(1)*MIN_CONST,fsrTime(end)*MAX_CONST,...
738             FSR_YMIN,FSR_YMAX])
739         title('FSR 2 data')
740         ylabel('ADC Count')
741         datetick('x',16,'keepticks')
742         subplot(2,2,3)
743         plot(fsrTime,FSR(:,3));
744         axis([fsrTime(1)*MIN_CONST,fsrTime(end)*MAX_CONST,...
745             FSR_YMIN,FSR_YMAX])
746         title('FSR 3 data')
747         ylabel('ADC Count')
748         datetick('x',16,'keepticks')
749         subplot(2,2,4)
750         plot(fsrTime,FSR(:,4));
751         axis([fsrTime(1)*MIN_CONST,fsrTime(end)*MAX_CONST,...
752             FSR_YMIN,FSR_YMAX])
753         title('FSR 4 data')
754         ylabel('ADC Count')
755         datetick('x',16,'keepticks')
```

```
756
757         %Plot FSR and tag usage
758         figure
759         [haxes,h1,h2] = plotyy(fsrTime,FSR(:,1),tagTime,totalTags(:,1),...
760             'plot','plot');
761         set(h2,'LineStyle','o','MarkerSize',8,'LineWidth',3)
762         title('FSR and Sock Data overlay')
763         axes(haxes(1))
764         ylabel('ADC counts')
765         axisLimits = axis;
766         axis(haxes(1),[axisLimits(1)-10^-1,axisLimits(2)+10^-1,FSR_YMIN,FSR_YMAX])
767         axes(haxes(2))
768         set(haxes(1),'YTick',0:200:4500)
769         ylabel('Number of Socks')
770         set(haxes(2),'YTick',0:.5:3)
771         datetick('x',16,'keepticks')
772         set(haxes(1),'XTick',[])
773         axis(haxes(2),[axisLimits(1)-10^-1,axisLimits(2)+10^-1,0,(max(totalTags(:,1))+1)])
774         xlabel('Time')
775
776         %Plot histogram of when subject don and doffs
777         figure
778         maxVal = 0;
779         clear h;
780         subplot(2,1,1)
781         hist(histData(:,1),.25:.5:23.75)
782         h(1) = gca;
783         a = ylim;
784         if maxVal < a(2)
785             maxVal = a(2);
786         end
787         title('Don and Doffs vs Time','FontSize',14)
788         ylabel('Number of Dons','FontSize',14)
789         set(gca,'XTick',[0:2:24],'FontSize',14)
790         subplot(2,1,2)
791         hist(histData(:,2),.25:.5:23.75)
```

```
792         h(2) = gca;
793         a = ylim;
794         if maxVal < a(2)
795             maxVal = a(2);
796         end
797         xlabel('Time (midnight to midnight)','FontSize',14)
798         ylabel('Number of Doffs','FontSize',14)
799         set(gca,'XTick',[0:2:24],'FontSize',14)
800         set(h,'YLim',[0,maxVal + 1]);
801
802         %Plot individual don/doff plots per day
803         histDaySize = size(histDay);
804         figure
805         clear h;
806         maxVal = 0;
807         for i = 1:histDaySize(2)
808             h(2*i-1) = subplot(histDaySize(2),2,2*i-1);
809             hist(histDay{1,i}(:,1),.25:.5:23.75)
810             set(gca,'XTick',[0:2:24],'FontSize',14)
811             ylabel(dName{i,1},'FontSize',12);
812             if i == 1
813                 title('Number of Dons','FontSize',14)
814             end
815             a = ylim;
816             if maxVal < a(2)
817                 maxVal = a(2);
818             end
819             h(2*i) = subplot(histDaySize(2),2,2*i)
820             hist(histDay{2,i}(:,1),.25:.5:23.75)
821             if i == 1
822                 title('Number of Doffs','FontSize',14)
823             end
824             a = ylim;
825             if maxVal < a(2)
826                 maxVal = a(2);
827             end
```

```
828             set(gca,'XTick',[0:2:24],'FontSize',14)
829         end
830         [ax1,h1]=suplabel('Dons and Doffs by Day','t');
831         [ax2,h2]=suplabel('Time (midnight to midnight)');
832         set(h1,'FontSize',15)
833         set(h2,'FontSize',14)
834         set(h,'YLim',[0,maxVal + 1]);
835
836
837
838         %Display extracted data
839         clc
840         fprintf('Total Time worn: %d Days, %s Hours, %s Minutes, %s Seconds\n',...
841             eTime(1),eTime(2:3),eTime(4:5),eTime(5:6));
842         fprintf('Date Start: %s\nDate End:   %s\n',datestr(accTime(1)),...
843             datestr(accTime(end)));
844         fprintf('\n\n');
845         fprintf('Number of unique tags detected: %i\n',IDind - 1);
846         fprintf('Most socks detected at once:    %i\n',max(totalTags(:,1)));
847         fprintf('\n');
848         if IDind ~= 1
849             fprintf('Tags detected                  Number detected\n%s\t%i\n',...
850                 char(uniqueTags{1,1}),double(uniqueTags{1,2}));
851             for i = 2:(IDind-1)
852                 fprintf('%s\t%i\n',char(uniqueTags{i,1}),double(uniqueTags{i,2}));
853             end
854         end
855         fprintf('\n\n');
856         fprintf('Number of Doffs Detected:                          %.0f\n',...
857             length(donDoffInd(:,2) ~= 0));
858         fprintf('Average Doff Time:                                 %s\n',...
859             avgDoff);
860         fprintf('Minimum Doff Time:                                 %s\n',...
861             minDoff);
862         fprintf('Maximum Doff Time:                                 %s\n',...
863             maxDoff);
```

```
864         fprintf('Standard Deviation in Doff:                    %s\n',...
865             stdDoff);
866         fprintf('\n');
867         fprintf('Number of Dons Detected:                       %.0f\n',...
868             length(donDoffInd(:,1) ~= 0));
869         fprintf('Average Don Time:                              %s\n',...
870             avgDon);
871         fprintf('Minimum Don Time:                              %s\n',...
872             minDon);
873         fprintf('Maximum Don Time:                              %s\n',...
874             maxDon);
875         fprintf('Standard Deviation in Don Time:                %s\n',...
876             stdDon);
877         fprintf('\n');
878         fprintf('Total activity periods missed:                 %3.0f of %0.0f\n',...
879             failCt-1,length(donDoffInd(:,1)));
880         fprintf('Total detection success:                       %3.0f%%\n',...
881             totalSuc*100);
882         fprintf('Don detection success:                         %3.0f%%\n',...
883             donSuc*100);
884         fprintf('Doff detection success:                        %3.0f%%\n',...
885             doffSuc*100);
886         fprintf('                    Dons       Doffs\n')
887         for i=1:days(1)
888             fprintf('%s          %2.0f          %2.0f                    \n',...
889                 dName{i,1},dDays(i,1),dDays(i,2))
890         end
891         fprintf('\n');
892         fprintf('Time and date for donnings and doffings: \n');
893         fprintf('          Don                              Doff\n');
894         for i=1:length(donDoffInd(:,1))
895             fprintf('%s          %s\n',datestr(donDoffInd(i,1)),datestr(...
896                 donDoffInd(i,2)));
897         end
898         fprintf('\n');
899         fprintf('Average Time Between Sock Changes:             %d\n',...
```

```
900             avgDiff);
901         fprintf('Minimum Time Between Sock Changes:                %d\n',...
902             minDiff);
903         fprintf('Maximum Time Between Sock Changes:                %d\n',...
904             maxDiff);
905         fprintf('Standard Deviation in Time Between Sock Changes: %d\n',...
906             stdDiff);
907
908
909 %% Save Data
910
911 if saveFlg
912     cd('..')            %Go up one file into the current date folder
913     cd('Analysis')      %Open Analysis folder
914
915     dataToSave.raw.activity = actData;    %Raw activity data in table format
916     dataToSave.raw.rawTag = rawTag;       %Raw tag data in table format
917     dataToSave.time.acc = accTime;        %Accelerometer time vector
918     dataToSave.time.fsr = fsrTime;        %FSR time vector
919     dataToSave.time.tag = tagTime;        %Tag time vector
920     dataToSave.ver = '0.3';               %SM decoder version
921     dataToSave.activity.FSR = FSR;        %FSR values
922     dataToSave.activity.xaccel = xacc;    %X-axis of accelerometer
923     dataToSave.activity.yaccel = yacc;    %Y-axis of accelerometer
924     dataToSave.activity.zaccel = zacc;    %Z-axis of accelerometer
925     dataToSave.sock.IDs = uniqueTags;     %The hex values of the IDs and number of instances
        during test
926     dataToSave.sock.tagCt = totalTags;    %Number of tags in each line
927     dataToSave.sock.numTags = IDind-1;    %Number of unique tags
928
929     save(saveName,'-struct','dataToSave');
930
931     %--------------------UPDATES TO VERSION----------------------%
932     % Version 0.3 fixes the fact that I was overwriting the time vectors
933     % and raw tag data
934 end
```

```
935
936
937     otherwise
938         display('Wrong choice, run program again');
939 end
```

Appendix F

SOCK MONITOR V1.3 BILL OF MATERIALS

Table F.1: Bill of materials for Sock Monitor v1.3

Component	**Value**	**Part Number**
B2	3 V	728-1058-ND
C1	2.2uF	478-4071-1-ND
C2	.1uF	311-1341-1-ND
C3	10uF	445-7486-1-ND
C4	10uF	493-2875-1-ND
C5	4.7uF	445-7482-1-ND
C6	4.7uF	445-7482-1-ND
C7	.1uF	311-1341-1-ND
C8	.1uF	311-1341-1-ND
C9	470nF	399-7339-1-ND
C12	0.1uF	311-1341-1-ND
C13	47uF	478-3325-1-ND
C15	220nF	311-1346-1-ND
C16	0.1uF	311-1341-1-ND
C17	0.1uF	311-1341-1-ND
C20	100uF	445-6007-1-ND
C21	1.0uF	311-1447-1-ND
C22	0.1uF	311-1341-1-ND
C23	10uF	338-1794-1-ND
C24	2.2nF	399-1085-1-ND

C25	4.7nF	399-1088-1-ND
C26	10uF	445-7486-1-ND
C27	.1uF	311-1341-1-ND
C28	220nF	311-1346-1-ND
C29	220nF	311-1346-1-ND
C30	0.1uF	311-1341-1-ND
C32	4.7uF	445-7482-1-ND
C33	4.7uF	445-7482-1-ND
C34	4.7uF	445-7482-1-ND
C35	4.7uF	445-7482-1-ND
C36	4.7uF	445-7482-1-ND
C37	12pF	311-1059-1-ND
C38	12pF	311-1059-1-ND
D1	Status 1	N/A
D2	RED	511-1653-1-ND
D3	CONNECT	511-1615-1-ND
D4	STATUS	511-1652-1-ND
D5	ESD_Pro	PESD0402-140CT-ND
D6	ESD_Pro	PESD0402-140CT-ND
D7	Rev_Pol	VESD12-02V-G-08CT-ND
D8	Status 2	N/A
F1	1A	0402SFF100F/24CT-ND
IC1	MSP430F5638	MSP430F5638IPZ
IC2	MAX4781	MAX4781EUE+-ND
IC3	ADXL345	ADXL345BCCZ-RLCT-ND
IC4	TPS2044	296-1959-5-ND
IC5	VREG	296-15753-1-ND
IC6	RN-42	RN-42

IC8	MT29F2G08ABAEAWP-IT:E TR	557-1488-1-ND
IC9	MCP73831	MCP73831T-2ACI/OTCT-ND
IC10	PTC	ZEN056V230A16LSCT-ND
J1	Analog expansion	609-3696-1-ND
J2	USB MINI-B	H11671CT-ND
J3	CAEN Connector	WM24007CT-ND
J4	JTAG	609-3694-1-ND
J6	FSR Dock	A98422-ND
L2	BEAD	445-6408-1-ND
L3	10uH	445-6635-1-ND
Q1	CAEN Regulator	568-6540-1-ND
R1	2K	RNCP0603FTD2K00CT-ND
R2	470	RMCF0603JT470RCT-ND
R3	3.4k	RMCF0603FT3K40CT-ND
R4	3.4k	RMCF0603FT3K40CT-ND
R5	470	RMCF0603JT470RCT-ND
R6	470	RMCF0603JT470RCT-ND
R7	1K	RMCF0603FT1K00CT-ND
R8	4.7k	311-4.70KHRCT-ND
R9	300 Ohm	RHM300DCT-ND
R10	47k	MCT0603-47K-MDCT-ND
R11	27 Ohm	RR05R27DCT-ND
R12	27 Ohm	RR05R27DCT-ND
R13	3.4k	RMCF0603FT3K40CT-ND
R14	4.7K	311-4.70KHRCT-ND
R15	4.7K	311-4.70KHRCT-ND
R16	4.7K	311-4.70KHRCT-ND
R17	1.91M	541-1.91MHCT-ND

R18	390k	311-390KHRCT-ND
R19	909k	311-909KHRCT-ND
R20	180k	311-180KGRCT-ND
R21	1M	541-1.0MYACT-ND
R22	40k	PAT40KACT-ND
R23	3.4k	RMCF0603FT3K40CT-ND
R24	330	RMCF0603JT330RCT-ND
R25	330	RMCF0603JT330RCT-ND
SW1	RESET	EG4589CT-ND
SW2	User interface button	P13589S-ND
W1	Battery positive	N/A
W2	Battery negative	N/A
X1	32.768 kHz	XC1911CT-ND
X2	16 MHz	887-1462-1-ND

Appendix G

HARDWARE ERRORS

G.1 Sock Monitor Version 1.2

1. A ground via was placed along the trace going to pin 2, P6.5
2. External capacitors were added to watch crystal, MSP has internal caps that can be used instead
3. Forgot to ground Vss of battery charger
4. Pin that has multiplexer output on it (Pin 34) does not have an ADC, need to re-route MUXOut
5. FSRs are not connected to reference voltage, when they go above 2.5V the ADC saturates
6. I did not separate the voltage supplies to the accelerometer, therefore I cannot enter standby
7. Switched MISO and MOSI lines to accelerometer
8. Resistor R2 at the charge indicator LED should be 470 not 470k
9. Need to add >4MHz clock for USB operation to XT2CLK
10. Did not properly implement USB pull-up resistor (R13)
11. Bluetooth LED resistors are too large
12. Need to current limit USB power to 500μA during suspended state
13. Voltage is too low for CAEN to operate, connected straight to battery to get it to work
14. High-side switch for CAEN cannot source enough current.
15. Need to include on-off switch
16. Physical Layout of watch crystal was bad, does not start/function properly
17. Did not implement ground ring around crystal parts

G.2 Sock Monitor Version 1.3

1. Via was laid on top of pad for Bluetooth switch

2. Pin 83 is connected to power instead of ground, incorrectly labeled on schematic

3. PUR pin is not connected to ground via 1 Mohm resistor . Note that PUR activates or pulls up D+, this is software controlled. If software control is not desired, then directly connect the pull up for D+ to VUSB (hardware control). We want hardware control!

4. R19 and R20 are not needed for the FB pin on the voltage regulator

5. The transistor for the CAEN power should be a P-type not N-type

6. The power line to the CAEN power transistor should be sourced from after the fuse but before the reverse polarity diode NOT AFTER.

7. Need to add power management for low voltage conditions, i.e. low power reset.

8. Need to connect CAEN reset pin to microcontroller, toggle low if non-responsive. Need to check the RX line from the CAEN module periodically to make sure it is high.

Appendix H

SOCK MONITOR V1.3 SPECIFICATIONS

Table H.1: Sock Monitor v1.3 Specifications

Description		**Value**
Clocks	Low Frequency	32,768 Hz
	High Frequency	16 MHz
Sensors	Acceleration	3-axis
	FSR	4 inputs
	UHF RFID	R1230CB-Quark
Communication protocols	UART	Bluetooth,RFID
	SPI	Accelerometer
	USB	Charging/Data
	Parallel	Flash memory
	UART Baud	115200
External communication	Bluetooth	RN-42
	USB	Not implemented
Power	Battery	2000mAh @ 3.7V
	System voltage	3.0V
	Max current	1.0A
	Max current draw	∼700mA
	Battery backup	Yes
	Charging	via USB
Memory	Size	2GB