Activity Recognition Processing in a Self-Contained Wearable System

by

Justin B. Chong

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

Dr. Thomas L. Martin, Chair

Dr. Mark T. Jones,

Dr. Thurmon E. Lockhart

Sept 12, 2008

Blacksburg, Virginia

Keywords: E-Textiles, Activity Recognition, Singular Value Decomposition

Activity Recognition Processing in a Self-Contained Wearable System

Justin B. Chong

(ABSTRACT)

Electronic textiles provide an effective platform to contain wearable computing elements, especially components geared towards the application of activity recognition. An activity recognition system built into a wearable textile substrate can be utilized in a variety of areas including health monitoring, military applications, entertainment, and fashion. Many of the activity recognition and motion capture systems previously developed have several drawbacks and limitations with regard to their respective designs and implementations. Some such systems are often times expensive, not conducive to mass production, and may be difficult to calibrate. An effective system must also be scalable and should be deployable in a variety of environments and contexts. This thesis presents the design and implementation of a self-contained motion sensing wearable electronic textile system with an emphasis toward the application of activity recognition. The system is developed with scalability and deployability in mind, and as such, utilizes a two-tier hierarchical model combined with a network infrastructure and wireless connectivity. An example prototype system, in the form of a jumpsuit garment, is presented and is constructed from relatively inexpensive components and materials.

Acknowledgements

The work shown in this thesis would not have been accomplished without the help and support of a lot of people. For those who are not mentioned specifically, thank you.

I would like to thank my advisor Dr. Tom Martin for all of the guidance, advice, jokes, and most importantly the opportunities he has given me throughout my graduate and undergraduate career.

I would like to thank Dr. Mark Jones for his guidance, for serving on my committee, and for being the one to introduce me to the field of embedded systems.

I would also like to thank Dr. Thurmon Lockhart for serving on my committee.

I would like to thank my roommate Syed Imtiaz Haider for being an excellent sounding board for ideas and for being a great friend.

And above all, I would like to dedicate this thesis work to my parents Teri and Yong Chong. I want to thank my mother for her continual support and words of encouragement, and I want to thank my father for being my inspiration to pursue this field of work and education.

This material is based upon work supported by the National Science Foundation under Grant No. CCR-0219809, CNS-0447741, and CNS-0454195. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do

not necessarily reflect the views of the National Science Foundation (NSF).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Electronic textiles and wearable computing devices have begun to play an increasing role in the fields of pervasive and embedded computing. In these modern times virtually everyone from children, adolescents, full grown adults, to even the elderly interact with some form of sophisticated embedded technology on a daily basis. With the rising popularity and prevalent use of cellular phones, PDA's, laptops, and portable media devices, as well as the ever evolving gamut of health monitoring technologies, many people are beginning to incorporate or, in some cases, even rely upon computing elements to function in their everyday life. Whether these devices are worn on clothes, stored in pockets, held in hands, or even strapped to the user's head, electronic textiles can provide an excellent platform on which to house, connect, and conceal these computing elements. By clustering groups of embedded devices into network configurations, wearable systems targeted for supporting sophisticated applications, like realtime activity detection, become realizable.

These systems, however, are often times difficult to construct and deploy due to the large number of hardware and software components that are required to be individually developed as well as the physical limitations that are imposed within a practical implementation. Processor speed, memory requirements, communication interfaces/protocols, power consumption/distribution, and scalability are all major concerns for system designers. The primary goals of this thesis are to provide guidelines for the design of a self-contained wearable electronic textile system as well as to provide a means of evaluating system performance by identifying bottlenecks.

1.2 Contributions

This thesis presents the design and implementation of a self-contained motion sensing electronic textile with particular emphasis toward the application of activity recognition. A hierarchical design methodology is provided to serve as a framework upon which sophisticated processing applications can be built. To demonstrate the effectiveness of this approach, a prototype e-textile jumpsuit was created to monitor and analyze the movements of the wearer using a Singular Value Decomposition classification environment.

1.3 Thesis Organization

The thesis is organized in the following fashion. Chapter 2 introduces the background information needed to understand the research that was conducted in this study. Chapter 3 provides a high level overview of the wearable system as well as some design guidelines for the components required for a successful implementation. Chapter 4 discusses the specifics of the hardware utilized in the system, and Chapter 5 provides information regarding the

software components that were developed. Chapter 6 describes the communication mediums and protocols incorporated into the prototype implementation. Chapter 7 describes the results obtained through experimentation as well as an analysis of the performance of the system. Finally, Chapter 8 provides a summary of the contributions of this thesis as well as conclusions that can be drawn and possible areas where future work can be conducted.

Chapter 2

Background

This chapter presents research in related fields to the work conducted for this thesis. Several key concepts from previous academic research efforts as well as commercial products were leveraged to develop the embedded system design approach presented in this thesis. Information from these areas was also used for the development of the electronic textile prototype platform on which experimentation was performed.

2.1 Electronic Textiles and Wearable Computing

Textiles, in general, are networks of threads or yarns that are woven in intersecting patterns to create a flexible fabric substrate. Electronic textiles (e-textiles) are simply fabrics with the addition of materials with electrical characteristics. These electrical materials typically include electronics for data processing as well as an assortment of wires or films used for sensing, communication, power distribution, and actuation. A field closely related to e-textiles is wearable computing in which computing elements are attached or worn on the body for a variety of purposes such as health monitoring, communication, or even more

recently for fashion.

In e-textiles and wearable computing research conducted thus far, particular attention has been focused in the medical field as well the development of systems for military applications. Projects like Georgia Tech's wearable motherboard [1] and the US Army Soldier Systems Center's smart vest [2] are excellent examples of the current state of wearable computing. Although medical and military applications have been the major driving force thus far for e-textiles, it is believed entertainment and personal safety will be next fields to push e-textiles and wearable computing into the mainstream. This shift in fields is not because entertainment and personal safety products will feature the most cutting edge sophisticated technology, but rather because they incorporate technologies and purposes geared toward everyday activities. Products like PDD's Illuminated Cycling Jacket [3], Swany's G.Cell hands free cell phone ski and snowboard glove [4], Raymarine's LifeTag man-overboard safety communicator [5], and the abundance of iPod related clothing from companies like Zegna [6], Burton [7], and Nike [8] support this claim.

These entertainment and safety systems will need to follow a different set of criteria in order to become widely adopted and to be effective for use in everyday life. One of a kind, custom, and expensive systems will not suffice. As such, the next generation of e-textiles must feature low power consumption, low material costs, fault tolerance, and must be easily manufactured through some automated process for cost-effective mass distribution.

2.2 Previous Electronic Textile Research at Virginia Tech

The Virginia Tech E-Textiles Laboratory has conducted many research efforts in the field of electronic textiles in the past several years. Many of these projects have particular focus in wearable computing as well as large-scale sensor networks. The Virginia Tech E-Textiles Laboratory utilizes a computer controlled, automated loom to weave custom smart fabrics in-house using standard textile manufacturing techniques. Although most e-textile research efforts performed at Virginia Tech are mainly proof of concept in nature, particular emphasis is placed on design methodologies for mass-manufacturing and fault-tolerance reliability.

Some examples of e-textiles previously developed at Virginia Tech include an acoustic beamformer [9], a smart carpet [10], and an activity recognition pants system [11]. Important design methodologies can be leveraged from each one of these projects. For example, the acoustic beamformer consisted of a 30-foot long computational fabric capable of determining the location and movement of approaching vehicles such as tanks and trucks. Acoustical sensing devices as well as computing elements were distributed across the textile substrate and interconnected to form a communications network. A software simulator was also created to aid in the development of the beamformer prototype.

In contrast to the military-oriented application of the beamformer, other projects like the smart carpet serve as excellent platforms for commercial and safety applications. As part of a joint research venture between Virginia Tech and the Intel Corporation, a floor mounted carpet electronic textile was developed with various sensing and actuation technologies [10]. Using an intersecting grid of resistive wires and rows of parallel peizo-electric cables, both a software simulation model and hardware prototype system were constructed to support a footstep tracking application. The smart carpet also featured visual actuation through

the use of electro-luminescent (EL) wires. An important design feature of the smart carpet was the use of a publish-subscribe service protocol. Utilizing the concept of services allows developers to work at a higher level of abstraction and helps to facilitate the generation of more complex applications.

The activity recognition pants system [11] is perhaps the most influential and relevant effort from Virginia Tech with regard to the research presented in this thesis. The pants system essentially consisted of a wearable lower body textile substrate with several acceleration modules symmetrically distributed on the hips, knees, and ankles. Piezo-electric films were also included to capture heel strike data. Raw 2-d motion data was streamed directly off of the pants using a serial Bluetooth device connected to a backend PC. The backend PC executes an activity recognition application which processes the raw motion data. Using Singular Value Decomposition (SVD), the classification environment is able to effectively match the movements of a person wearing the pants to a corresponding set of trained motions in pseudo-realtime.

2.3 Sensor Based Activity Recognition

Sensor based activity recognition has been gaining much popularity, especially in the entertainment and consumer electronics industries. The Nintendo Wii game console utilizes a hand-held controller device containing a 3-d accelerometer along with a small infrared camera to provide user input to game applications using a pre-determined range of hand gestures and movements [12]. The Sony Playstation 3 (PS3) also features an accelerometer based tilt-sensing controller to provide input to the console for motion-based gaming activities [13]. Other more discrete consumer electronics devices like laptop computer hard drives use sensors to detect when the object is free falling with respect to gravity [14]. Many

consumer products use sensors to recognize specific activities, like the Nike iPod shoe inserts which use piezo-electric sensors to monitor running statistics [8].

Chapter 3

System Overview

A self-contained activity recognition system is comprised of many components. There are various hardware and software modules which must be set properly in place to provide an infrastructure adequate enough for data to efficiently flow and be effectively processed. These modules need to be designed with flexibility and expandability in mind, in order to ensure support for future applications. As such, a generic hardware/software infrastructure should incorporated into the textile substrate. The following sections provide general design guidelines for developing a system capable of self contained application level processing.

3.1 Design Hierarchy and Data Flow

Experience with a range of e-textile applications has shown that a two-tier hierarchy of processors is an appropriate architecture for e-textiles [10]. To achieve this hierarchy, two distinct classifications of nodes are required. The two tier model is comprised of a bottom level of several nodes defined here as type Tier 1 and an upper level of one or more nodes defined here as type Tier 2.

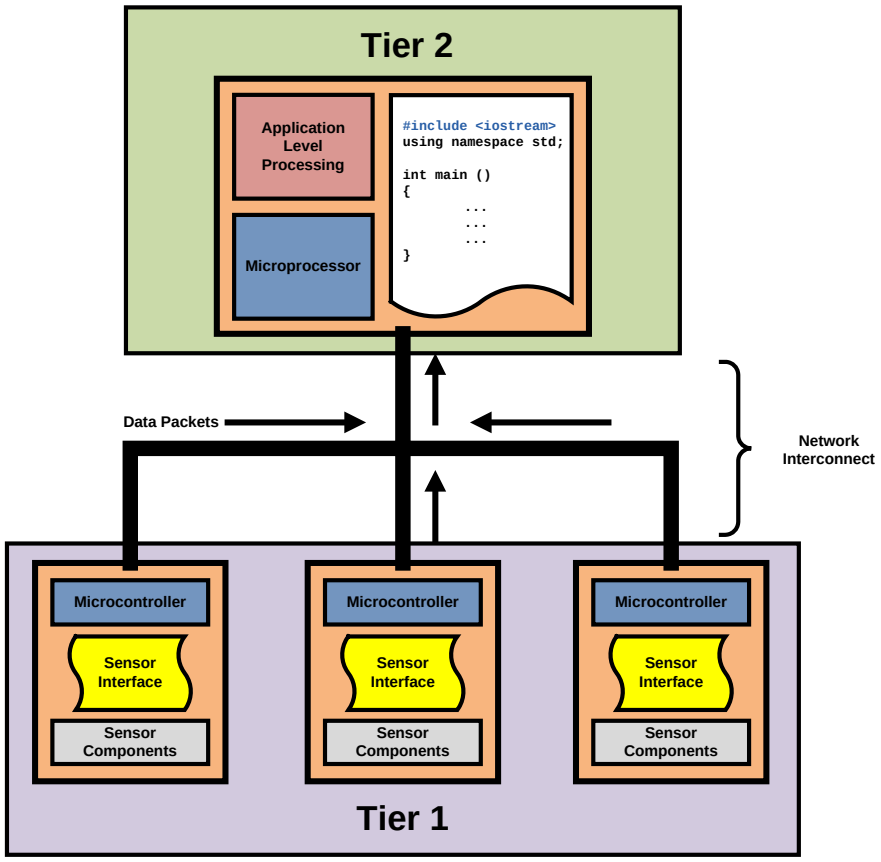

Figure 3.1: Two Tiered Model

As shown in Figure 3.1, sensor data is collected by the Tier 1 nodes. The sensor data is aggregated into packets and then streamed across the network to the Tier 2 node. The Tier 2 node buffers the data from all of the Tier 1 nodes and can then begin to perform application-level processing. It should be noted that, for the activity recognition application, the Tier 1 nodes do not communicate between one another; although the network infrastructure of the overall system does include support for inter-Tier 1 communication in the event that future applications may make use of this feature. The two-tier model is intended to serve as a scalable hierarchy for any e-textile in general, not necessarily just wearable technologies. Figure 3.2 demonstrates how the two levels of nodes may be distributed across a jumpsuit garment worn on the human body.

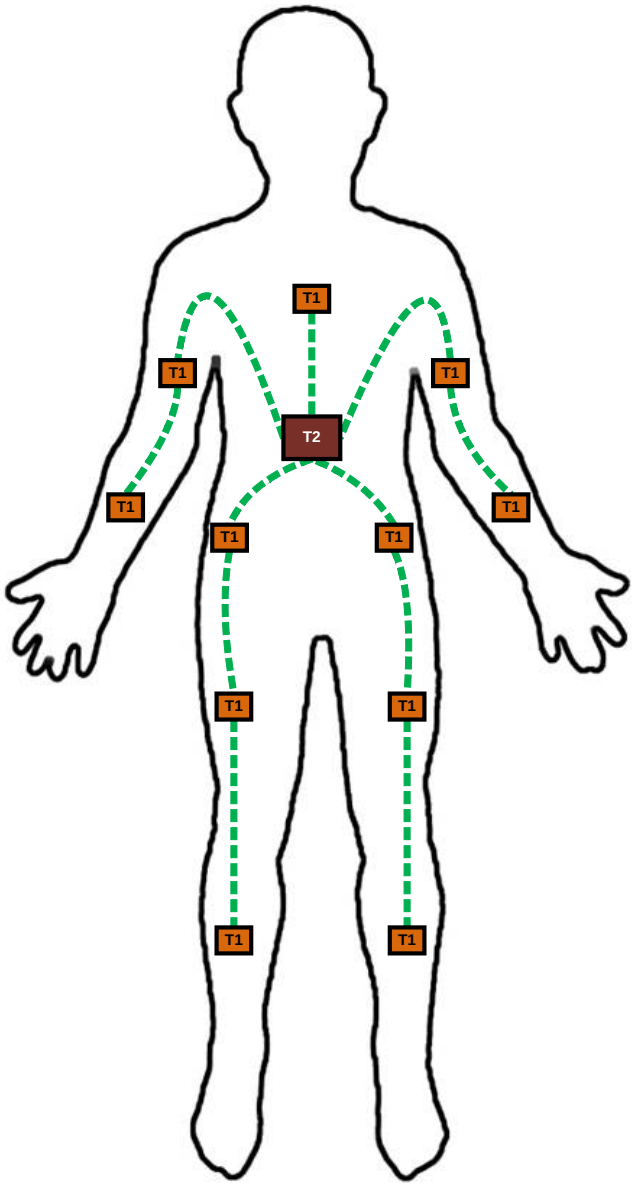

Figure 3.2: Nodes Distributed Across Human Body

3.2 Two-Tier Model

The following two sections describe the responsibilities of each class of node utilized in the two-tier model. Design guidelines are provided for each tier to serve as a starting point to aid future developers in selecting components for prototype implementations which make use of the two-tiered approach. It should be noted that these guidelines are meant to approximate the processor speed and memory space required to support an application like activity recognition within the confines of a wearable e-textile.

3.2.1 Tier 1

Tier 1 consists of an interconnected network of nodes with each node containing a relatively low-performance microprocessor. This microprocessor need not be very sophisticated since limited data processing is performed within Tier 1. These Tier 1 microprocessors are responsible only for interfacing with various sensor components, performing analog-to-digital (A/D) conversions on sensor signals, and streaming the results of those conversions across the network to the node(s) contained in Tier 2. Due to these relatively low levels of processing, the speed and memory footprint requirements for these processors are relatively low as well. Processors with speeds in the single digits or tens of megahertz range and with only a few kilobytes of memory space are sufficient to operate in Tier 1.

3.2.2 Tier 2

Tier 2 consists of at least one, more sophisticated type of node. This Tier 2 node contains a much more powerful microprocessor capable of performing application-level processing on the data streamed to it from the Tier 1. Since Tier 2 nodes should be capable of running an

operating system with potentially multiple threads of execution, a processor with moderate clock speed, in the hundreds of megahertz range, is required. Also, since a fair amount of data from several Tier 1 nodes is being received, buffered, and analyzed, the memory requirements for Tier 2 nodes are much larger. Memory space in the tens of megabytes is sufficient to contain the amount of data streamed to this tier, at least for the application described in this thesis.

Chapter 4

Hardware Development

The following sections provide example prototype implementations for the hardware components needed to perform activity recognition within the confines of a jumpsuit e-textile. In order to test the effectiveness of the two-tiered approach, a variety of hardware prototypes for the Tier 1 and Tier 2 nodes were constructed. Most of these hardware modules were re-developed from the ground up but were based upon lessons learned from previous designs from Virginia Tech [15]. Some of the more sophisticated hardware prototypes were constructed by adding custom interface modules to commercially available products.

Before development began, the limitations and problematic issues encountered in the previously developed e-textile systems were analyzed to find out where adjustments and enhancements could be made for the new jumpsuit system. Many valuable lessons were learned from the previous version of smart pants developed at Virginia Tech [16] [11] [17]. These issues include, but are in no ways limited to, how to physically attach and connect the modules to the textile substrate as well as how data must flow between nodes properly to ensure effective processing. Also, important design considerations were taken to reduce and prevent electro-magnetic-interference (EMI) noise problems stemming from analog sensor signals and

high speed digital communication lines.

4.1 Tier 1 Acceleration Modules

The development of the Tier 1 acceleration modules is outlined in the following sections. First a discussion regarding the types of acceleration sensing components incorporated in the Tier 1 hardware prototypes is provided. Next, the problems associated with using piezoelectric materials is addressed. A power regulation circuit is also presented as an upgrade to the design typically used in previous Virginia Tech E-Textiles Lab prototypes. Finally, a description of the initial breadboard Tier 1 node prototype as well as the final miniaturized printed circuit board design is provided.

4.1.1 Acceleration Sensors

Since the main focus of the jumpsuit prototype system was to perform activity recognition, a generic set of acceleration sensing modules was developed to comprise the homogeneous lower tier of the system. Each acceleration module contains both an analog accelerometer as well as an analog gyroscope sensor. The accelerometer used in the jumpsuit prototype is an MMA7260 manufactured by Freescale Semiconductor. This three axis accelerometer was selected because it features four selectable sensitivities: +/-1.5g, +/-2g, +/-4g, and +/-6g [18]. The accelerometer sensor is used to collect readings of how the module is moving with respect to gravity. The gyroscope utilized in the jumpsuit prototype is a single axis ADXL330 angular rate sensor made by Analog Devices [19]. The gyroscope is used to measures the angular acceleration imparted in the plane perpendicular to the surface of the module. Each Tier 1 acceleration module also features an 8-bit Atmel Atmega8L

microcontroller [20] used to convert the analog signals from the accelerometer and gyroscope to equivalent 10-bit digital values. The microcontroller then streams those digital values over an Inter-Integrated Circuit (I^2C) communications bus to the Tier 2 processing node.

Although conceptually similar to the design shown in [15], the acceleration sensing modules were redesigned from the ground up to include many hardware optimizations. Foremost, all of the acceleration modules used on the jumpsuit are homogeneous in that each one contains identical processor and sensor circuitry. This was not the case in the previously developed pants system since some modules had accelerometer and gyroscope sensors and some had only accelerometers.

4.1.2 Piezo-Electric Material

The previous pants also featured piezo-electric film sensors that proved to not be very useful for activity recognition. Piezo-electric films and cables [21], as shown in Figure 4.1, have proven themselves to be fairly susceptible to EMI noise in the development of the previous pants and carpet systems described in [11] and [10]. To further illustrate the problem with using these types of materials, Figure 4.2 shows the noise introduced into the signal of a piezo-electric cable caused by simply overlaying a wire carrying a high frequency AC signal in an orthogonal orientation to the piezo-electric cable. The waveforms on the left show an attempt to capture large and small heel strikes on a piezo-electric cable in a noisy environment. It is possible to extract useful information from these waveforms using digital signal processing (DSP) techniques, as shown by the application of an 8th order bandstop filter in the waveforms on the right side of Figure 4.2. However, this DSP functionality would require specialized multiply-accumulate hardware not available on the general-purpose microcontrollers utilized in this Tier 1 prototype implementation. Performing the filtering

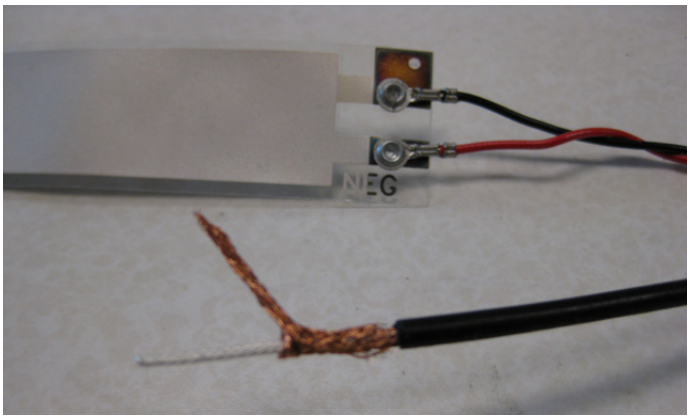

Figure 4.1: Piezo-Electric Material

techniques strictly in software is not feasible with the Atmega8L hardware selected, thus piezo-electric sensory input was not incorporated into the jumpsuit prototype.

4.1.3 Power Regulation

The power circuitry for the acceleration sensing module was also redesigned to utilize a more reliable set of 3.3V and 5V LP2292 series linear regulators manufactured by National Semiconductor [22]. These regulators provide more stable power output than the linear regulators made by Microchip that were used in [11] and [10]. These regulators are also ROHS compliant, and feature high temperature auto-cutoff and current over-draw protection functionality to ensure that sensor components on the module will not be damaged if the power circuitry malfunctions or is otherwise abused. A schematic of the revamped power circuitry is provided in Figure 4.3. Upon experimentation, it was determined that insufficient equivalent series resistance (ESR) was the leading cause of malfunction with the Microchip linear regulators. As such, tantalum capacitors could have been used instead of ceramic capacitors in the power circuitry as they are less susceptible to capacitance change due to temperature variation and typically have slightly higher ESR ratings. However, tantalum capacitors are

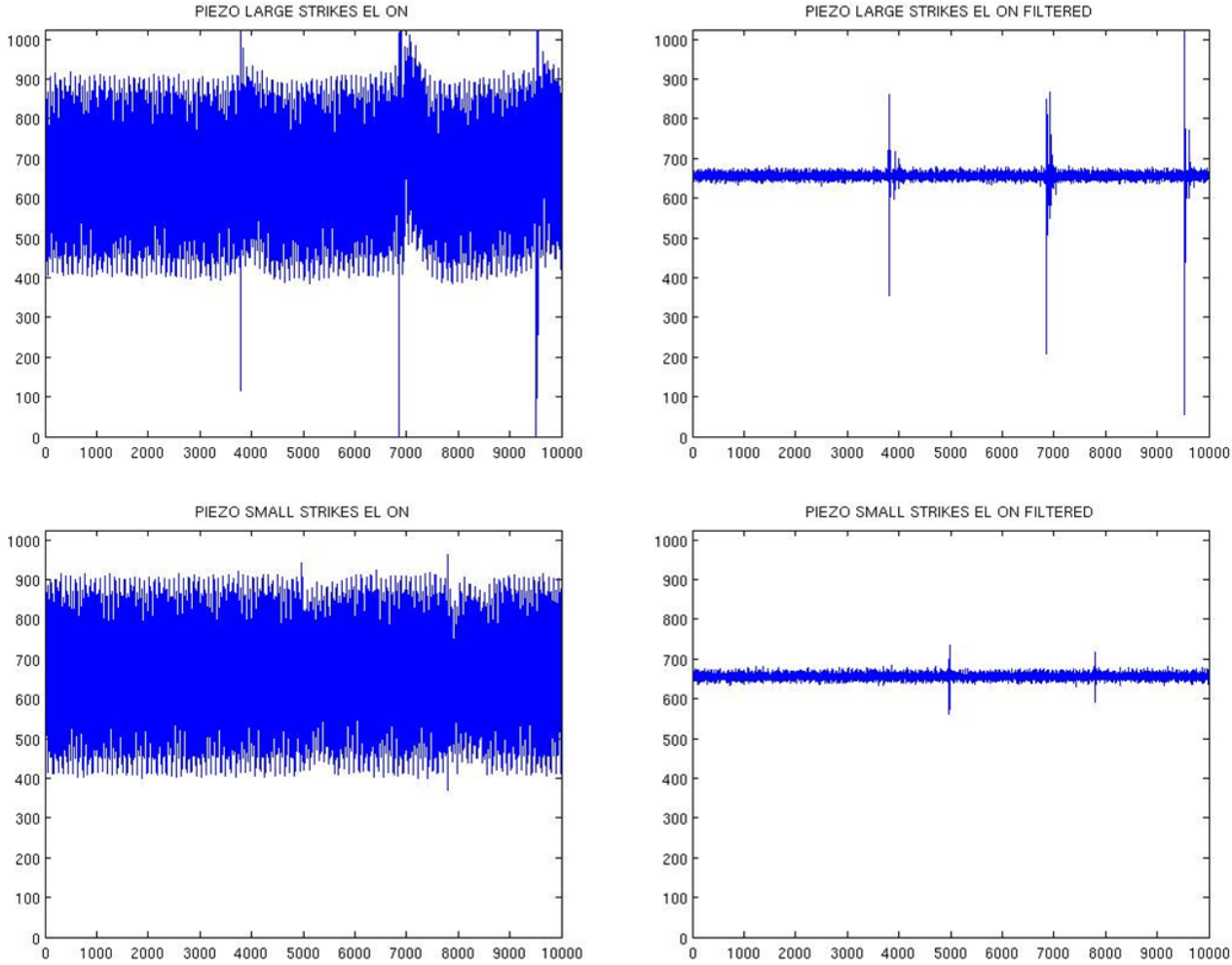

Figure 4.2: Piezo-Electric Noise

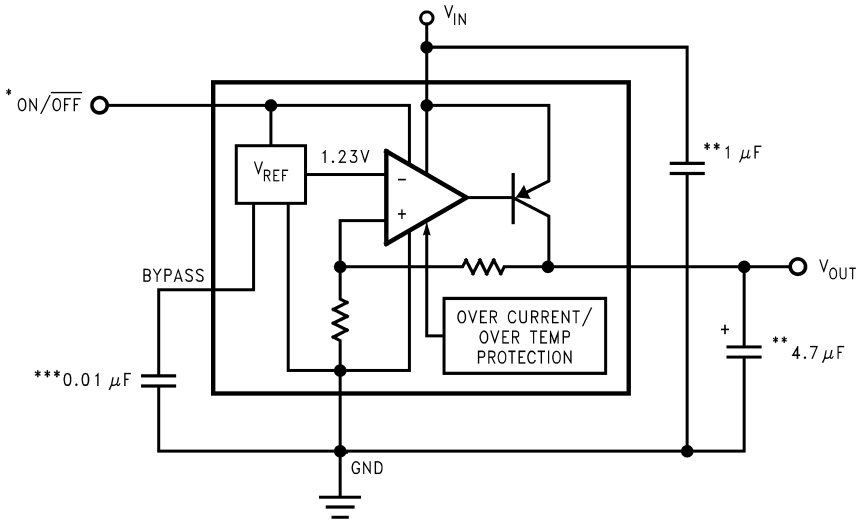

Figure 4.3: Linear Voltage Regulator Circuit

significantly more expensive to purchase, particularly for the 0402 size parts. The National Semiconductor LP2292 series linear regulator relies on low ESR ceramic capacitors to tune and filter the supply voltage and were thus an inexpensive and logical replacement [22].

4.1.4 Breadboard Prototype

Initially, a breadboard prototype of the Tier 1 acceleration module was created to verify the functionality of the design as shown in Figure 4.4. To verify the operation of the breadboard version, a rotating test platform, shown in Figure 4.5, was created out of Lego® building blocks. The breadboard was easily mounted in either a horizontal or vertical orientation and rotated through 360 degrees by either a small DC motor or cranked by hand. A small serial Bluetooth radio module was also attached to the breadboard to stream the acceleration values wirelessly to a backend PC for analysis.

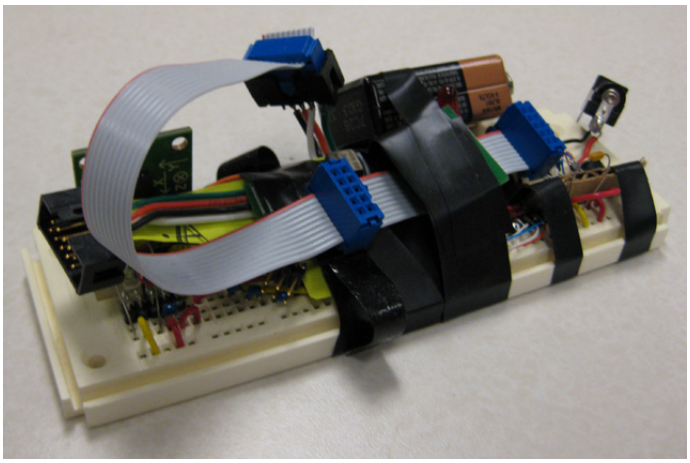

Figure 4.4: Breadboard Prototype

Figure 4.5: Gyronator Test Platform

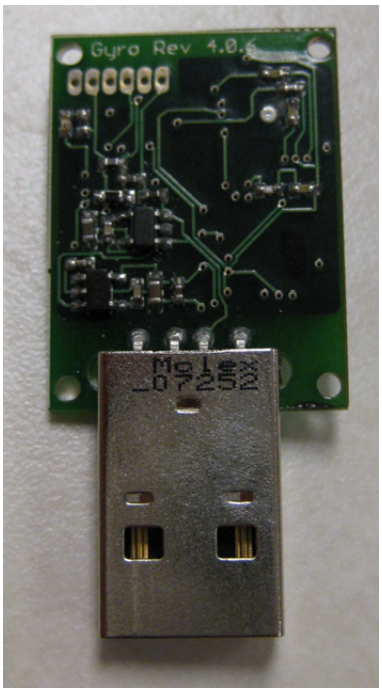

Figure 4.6: Acceleration Module (Front)

4.1.5 Miniaturized Tier 1 Prototype

After verifying the functionality of the overall Tier 1 module design on the breadboard, a miniaturized version in a wearable form-factor was developed. Since space is at a premium on an electronic textile so as not to restrict the range of mobility of the person wearing the garment, the modules themselves were built on a four layer printed circuit boards (PCB) to reduce the routing area between hardware components. The PCB also contains separate analog and digital ground planes connected via a ferrite bead. This reduces EMI introduced by the analog sensor signals being physically close to the high speed digital traces used for communication. The populated acceleration module is shown in Figures 4.6 and 4.7.

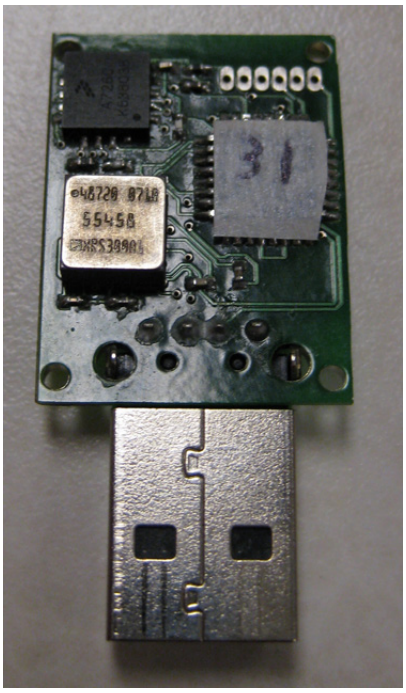

Figure 4.7: Acceleration Module (Back)

4.2 Tier 2 Processing Element

The single Tier 2 module used on the jumpsuit prototype was intended to replace the backend PC used in the previous pants system [16] [11]. It serves as the main processing element for the activity recognition application and is responsible for receiving, buffering, and analyzing the motion data streamed to it from all of the Tier 1 modules. Due to the added level of sophistication necessary for this type of processing, it was believed that finding a commercial product would be a more viable alternative for constructing a Tier 2 prototype than trying to build one from scratch. A product feature matrix, like the one shown in Figure 4.8, was constructed to determine what product most closely matches the requirements outlined for a Tier 2 module.

Product	Features								
	Processor	MMU	Operating System	Memory	Storage	UART	I2C	TCP/IP	Bluetooth
Atmel AT91 Development Board	Atmel ARM7TDMI	Yes	Ecos	256 kB	1 MB flash	2	Yes	No	No
Gumstix Verdex 400xm-bt	Marvel xScale PXA270	Yes	Open Embedded /Buildroot	64 MB	SD/MMC Card	3	Yes	Yes	Yes
ColdFire MCF235 Board	Freescale ColdFire	No	uCLinux	16 MB	2 MB flash	1	Yes	Yes	Yes

Figure 4.8: Product Feature Matrix

4.2.1 Gumstix Motherboard

After much searching, it was determined that a product from Gumstix, the Verdex 400xm-bt [23], shown in Figure 4.9, would be the best candidate due to its processing power, memory, storage, connectivity, form factor, and online community support. The Gumstix used in the prototype system features a Marvel 400 MHz xScale microprocessor, 64 MB of RAM, 16 MB of flash memory, Bluetooth, USB, and serial connectivity as well as an expansion slot for connecting an array of expansion cards. The Gumstix was also selected for its ability to run a full Linux operating system as described in the Chapter 5.

4.2.2 Network Interface Module

In order to interface the Gumstix motherboard to the I^2C network on the jumpsuit prototype, an expansion board called the Breakout-gs [24] was also purchased from Gumstix to break out the small pins of the Hirose connector on the Verdex motherboard. A custom I^2C to universal asynchronous receiver/transmitter (UART) daughter card, shown in Figure 4.10, was then created to interface the Tier 1 communications network to the Gumstix via

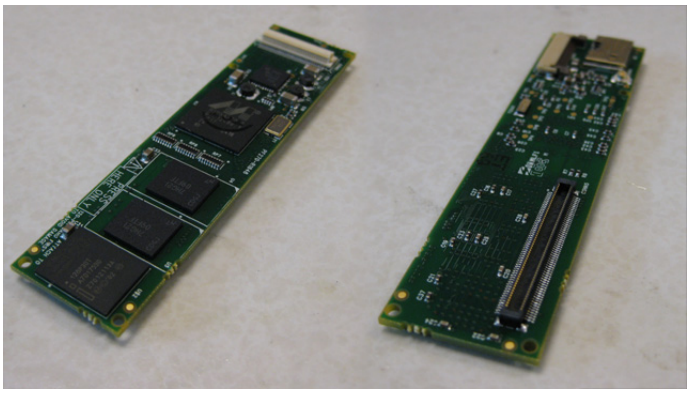

Figure 4.9: Gumstix Verdex 400xm-bt

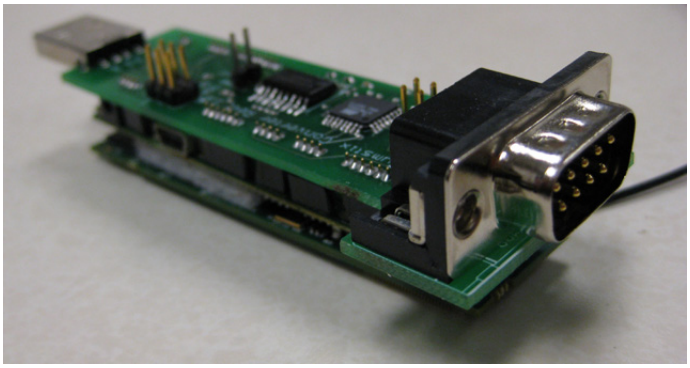

Figure 4.10: I^2C to UART Daughter Card/Gumstix Assembly

the Breakout-gs expansion board. The I^2C to UART daughter card contains an Atmega8L microcontroller and provides backwards compatibility support for the older pants system. The daughter card also includes a 9 pin D-Sub connector and MAX232 TTL/CMOS line driver/receiver to serve as a configuration port to the Gumstix for flashing the operating system image or performing serial connectivity in the event the wireless Bluetooth connection fails as outlined in the appendix. The Tier 2 Gumstix node essentially utilizes a specialized Tier 1 module as a proxy gateway to the I^2C network.

4.3 Interconnections and Textile Substrate

A major concern in all wearable computing systems is how to provide physical interconnections between the computing devices and the garment substrate itself. An effective interconnection method must adhere to the following criteria. First, the interconnections should always be polarized, that is to say that there should only be one unique way to plug the device into the system. The polarization ensures that the power, ground, and communication lines are always connected properly as to not damage the devices since often times many sensor components are not tolerant to a wide range of voltage configurations. Second, the interconnections should provide a robust mechanical fastening interface such that the computing devices will not fall off or become otherwise disconnected from the garment as the result of movement by the person wearing the system. Third, the interconnections should allow the computing devices to be easily removed so that the garment can be washed or so that the computing devices can be moved around to facilitate different geographical topologies.

The interconnections on the jumpsuit prototype were achieved using male and female USB connectors. The four contacts on each USB connector are used to connect power and ground as well the two I^2C communication lines Serial Clock (SCL) and Serial Data (SDA). For the ease of rapid prototyping, flat ribbon cables were attached to a pair of sweatpants and a long-sleeve t-shirt, as shown in Figure 4.11, to simulate a jumpsuit or coverall garment. The computing devices themselves are connected through the USB connectors to the ribbon cables via a 4-position USB to 0.1 inch dual row pin header converter board Figure 4.12. In an actual woven textile, these USB to 0.1 inch dual row pin header converter boards would be replaced by an insulation displacement connector (IDC) as shown in Figure 4.13.

The IDCs, as shown in Figure 4.14, provide an effective means of splicing into the woven wire

Figure 4.11: Ribbon Cable Prototype

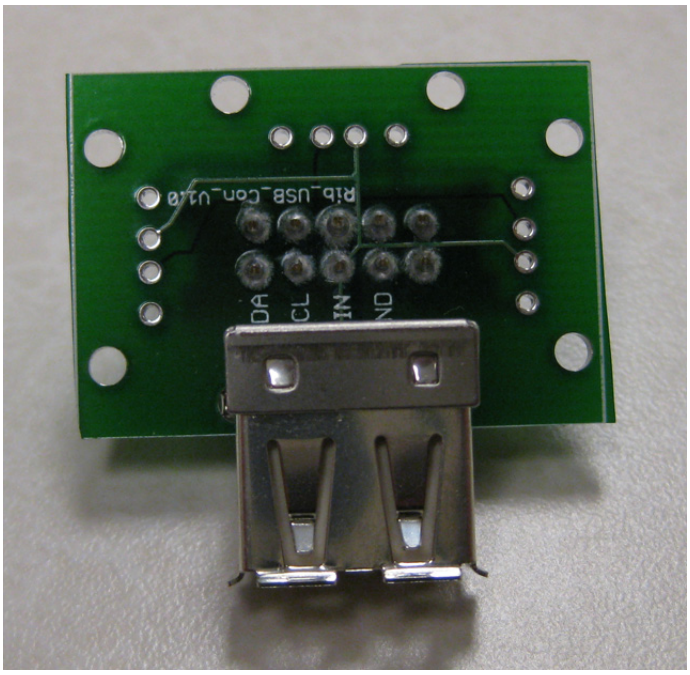

Figure 4.12: USB to Ribbon Cable Converter Board

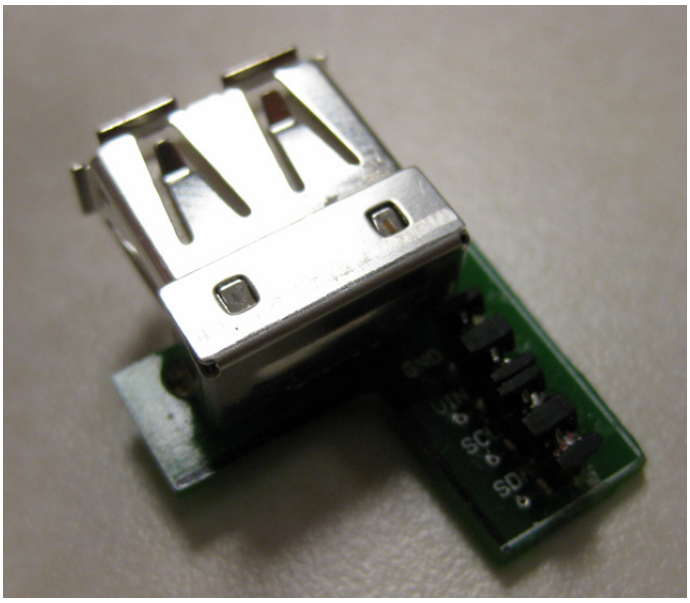

Figure 4.13: USB to IDC Converter Board

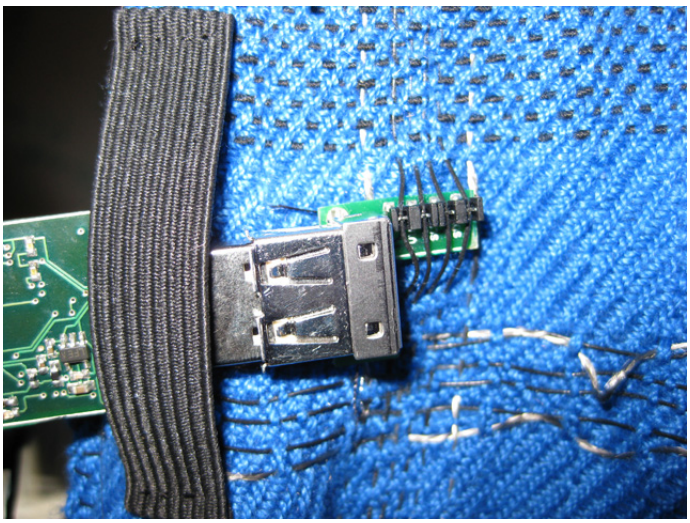

Figure 4.14: IDC Attached to Wire Bus

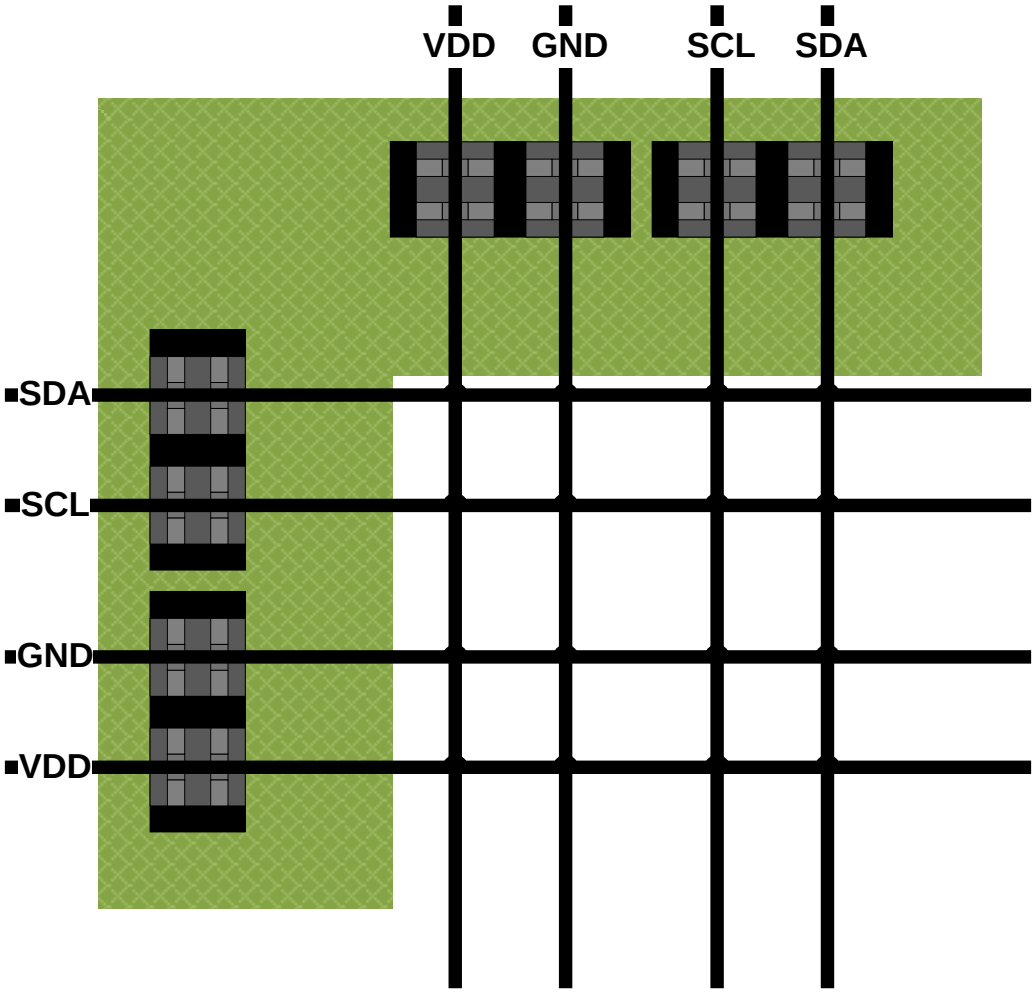

Figure 4.15: Perpendicular Bus Jumpering Via IDC Connector

busses without significantly compromising the structural integrity of the wires. Using IDCs, perpendicular wire busses can be jumpered across intersections to route power, ground, and communication signals throughout the textile as shown in Figure 4.15. Connecting parallel and perpendicular busses throughout the garment also provides a level of fault tolerance as the signals may propagate through alternative paths should an individual wire become damaged.

Chapter 5

Software Development

The following sections provide a discussion on the various software components required to effectively perform activity recognition within the confines of a jumpsuit e-textile. Two software models are presented for use on the Tier 1 nodes. The selection of an operating system to run on the Tier 2 Gumstix module is then described in detail. Finally, the modification and porting of the activity recognition application to the embedded Gumstix platform is outlined.

5.1 Tier 1 Firmware

The Tier 1 nodes are responsible for performing analog to digital conversions on sensor inputs and forwarding those conversion values to the Tier 2 processing module. There are multiple ways to implement such a system, and as such, two very different software models were developed for Tier 1. The first model was an event-driven approach intended to provide application level flexibility to the end user or developer [25]. It provides support for registering services via a publish subscribe methodology [10], easy selection of A/D operations, and

communications support for robust network configurations. The second software approach was an optimized solution intended to maximize overall system performance by increasing throughput and minimizing overhead. The following sections provide an overview of the designs of each of these software implementations.

5.1.1 Event-Driven Software Model

In order to provide a flexible baseline from which application specific Tier 1 firmware can be developed, an event-driven paradigm was selected. In this approach, the flow of the program execution is determined at runtime by sensor input, internal timer notifications, and network communication exchanges. The baseline application basically provides a generic programming stub from which the application developer can simply register the events he or she needs to support a given task. Initially, before user configuration, the stub provides only basic single timer functionality as well as I^2C network communications support. Additional support may be included for multiple timers, A/D samples and conversions on multiple channels, UART communication, serial peripheral interface (SPI) communication, and a variety of other services. The developer can then mix and match these abstracted options to suit the requirements of the application without having to understand the specifics of the underlying hardware platform. This approach was designed with portability in mind so that the stub baseline can serve as a starting point for virtually any e-textile application in general, not just wearables.

A flow diagram for the stub baseline is provided in Figure 5.1. The baseline is architected as follows: First, board specific peripherals and options are initialized based upon the hardware platform. Next, event registration begins. This is the section where the application developer would configure the system to suit his or her needs. Finally, there is the main dispatch loop

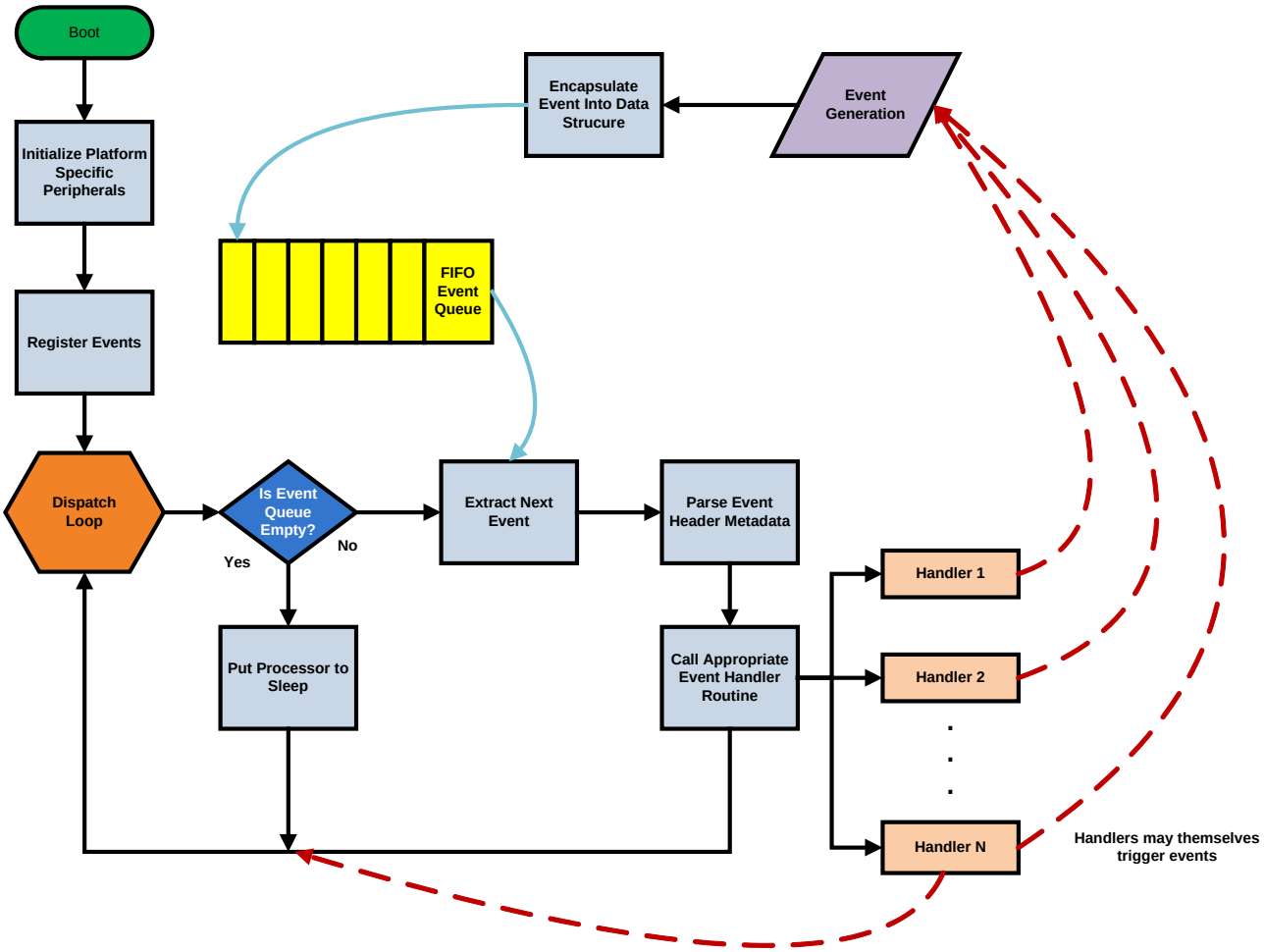

Figure 5.1: Stub Baseline Execution Flow

which is responsible for calling the appropriate event handlers. As events are generated, they are encapsulated in a generic event data structure, shown in Figure 5.2, and then stored in a first-in-first-out (FIFO) queue. During each iteration of the dispatch loop, the next event to be serviced is extracted from the event queue. The metadata of the event data structure is then parsed and the service ID of the event is used to determine which event handler function to call. If the event queue is empty, the processor is put into a sleep state until the next event is triggered.

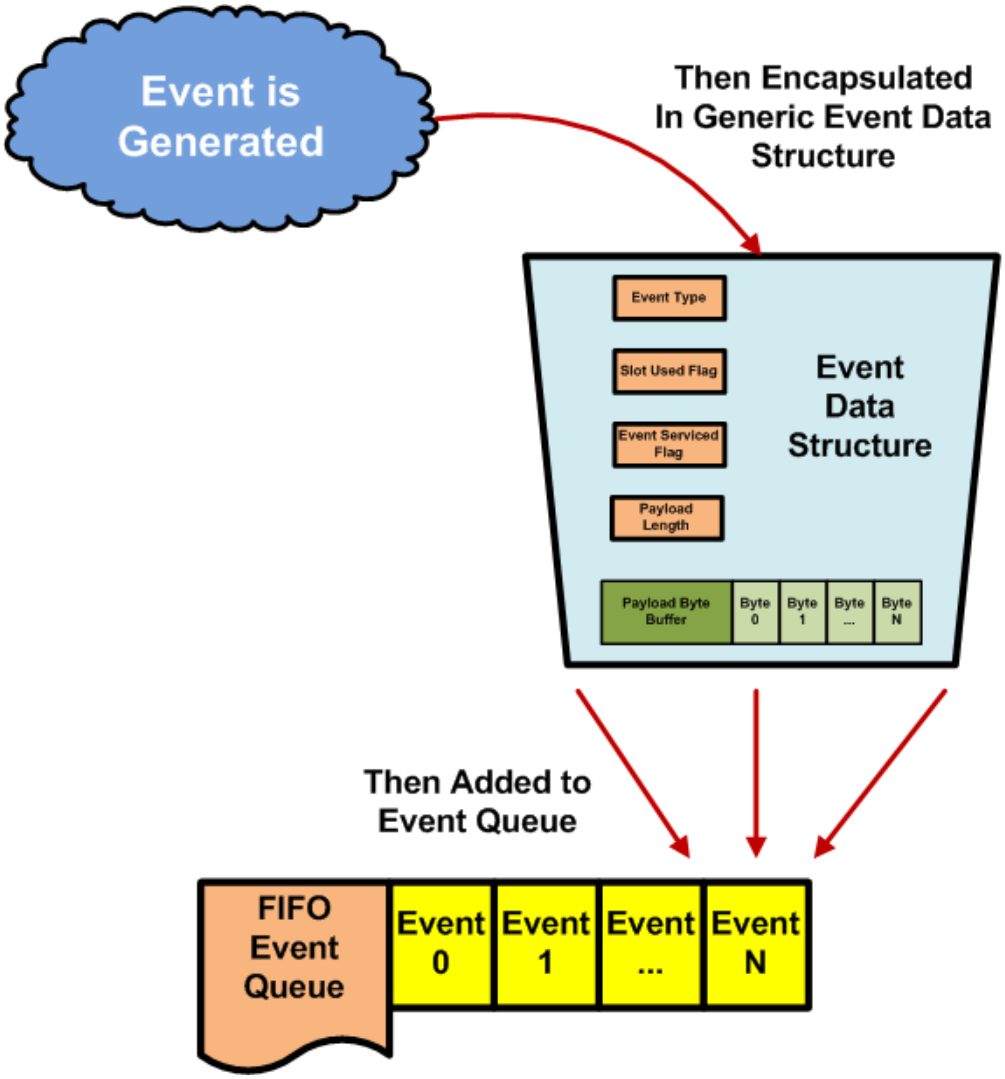

Figure 5.2: Event Data Structure

5.1.2 Optimized Software Model

The optimized software model was targeted towards the activity recognition application specifically. The focus of this approach is to collect and transfer motion data from the sensors to Tier 2 as quickly and efficiently as possible within the physical limitations of the hardware platform. This model is finely tuned for performing A/D conversions and I^2C exchanges quickly and is architected in a lock-step fashion as follows: First, board specific device drivers are initialized to configure the sensitivity on the accelerometer as well as enable the output of the accelerometer and gyroscope sensors. Next, the I^2C network driver is initialized with the address of the specific module, and a basic timer interrupt is enabled. An infinite transmission loop is then initiated wherein an A/D conversions complete flag is polled so see if motion capture data is ready to be transmitted to the Tier 1. If the conversions complete flag is set, the data is sent out over the wire and then the processor is put to sleep to conserve power until the next timer interrupt fires. Upon reaching the timer interrupt service routine, successive A/D conversions are performed, one for each of the X, Y, and Z channels on the accelerometer and the one conversion for output of the gyroscope as well. The four 10-bit conversion values are stored in a buffer and the conversions complete flag is set so that the transmission loop will know when the data in the buffer is valid. A flow diagram for the optimized solution is provided in Figure 5.3.

5.2 Tier 2 Software

In the jumpsuit prototype, only one Tier 2 node was utilized. This Tier 2 node serves as the primary computing element for the system and is responsible for the application level processing of the motion data streamed to it from the Tier 1 nodes. The following two sections describe the selection of an embedded operating system for the Tier 2 Gumstix as

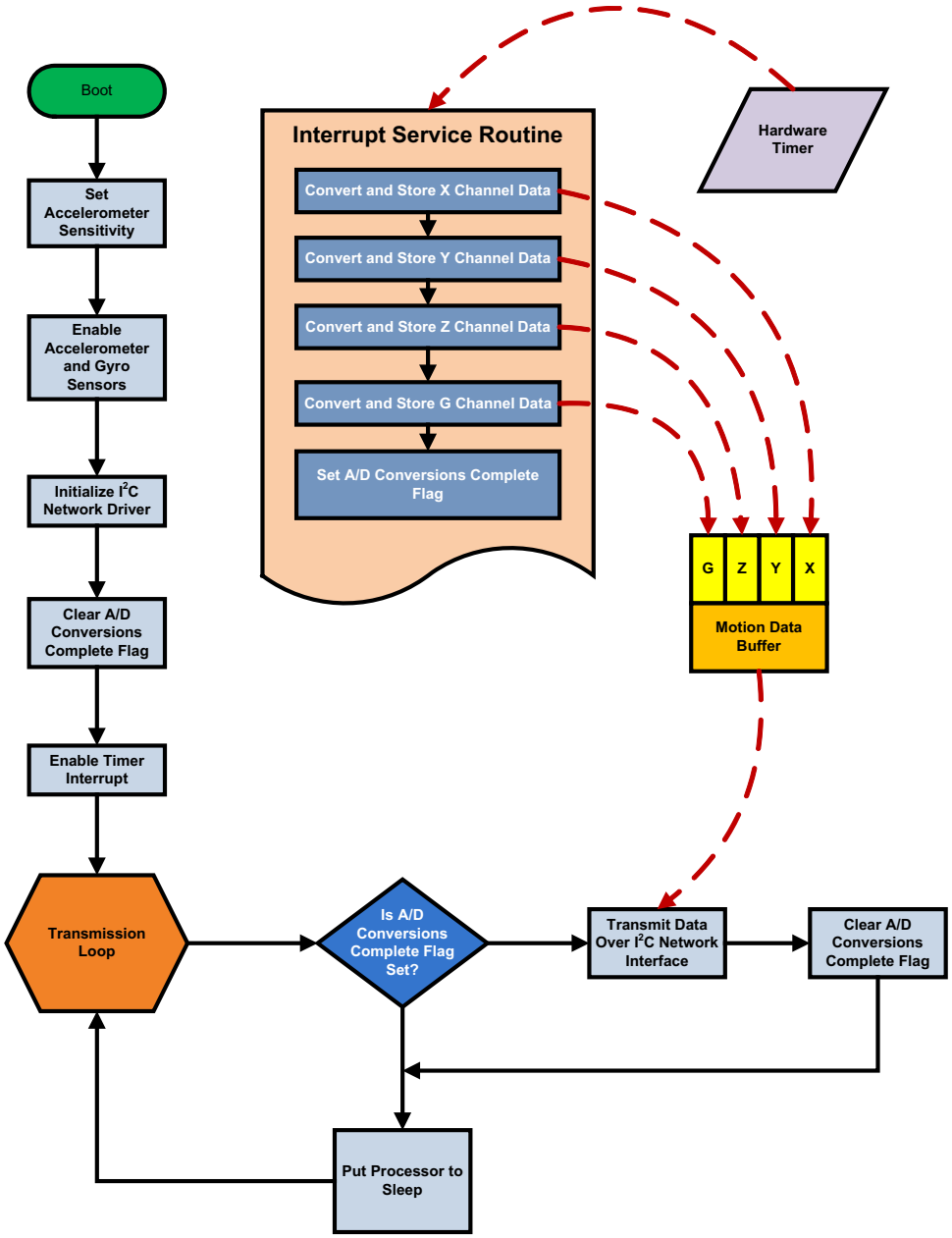

Figure 5.3: Optimized Model Execution Flow

well as how the activity recognition application was benchmarked and ported to the self-contained environment.

5.2.1 Operating System

The Marvell xScale PXA270 microprocessor [26] featured on the Gumstix Verdex motherboard is capable of running a Linux operating system since it is based off of ARMV5TE core technology. It should be noted that although Marvell now owns the rights to the xScale PXA270 product, this is actually still an Intel chipset design. Gumstix explicitly provides two main avenues for running an operating system on the xScale, Buildroot and OpenEmbedded. Both of the kernel codebases for Buildroot and OpenEmbedded provide similar useability, that is to say that virtually the same applications will run on both systems. It was determined through preliminary testing that OpenEmbedded provides more complete and more reliable Bluetooth and TCP/IP support for the specific Verdex motherboard used in the jumpsuit prototype.

OpenEmbedded is an opensource effort geared towards providing Linux support for embedded device platforms. Software development for OpenEmbedded is centered around a managed collection of BitBake recipes. The modular BitBake recipe/package management concept is similar to the ebuild process implemented in the popular Linux flavor Gentoo. Each Bitbake recipe contains information about the location of the package source code as well as specific compilation and installation options. The BitBake recipes are also used to determine and manage dependencies between packages. The BitBake utility provides a method for invoking GNU make commands inside the recipe to use the ARM GCC/G++ toolchain. This cross compiler flexibility makes it possible to port C and C++ applications developed and tested on a standard Linux PC to the embedded Gumstix platform.

The argument can be made that one should use a Real-Time Operating System (RTOS) for a sophisticated embedded platform like the e-textile jumpsuit. However, it was decided that the generic Linux support provided by OpenEmbedded was a better design choice. Although RTOS's like ECOS, QNX, and RTLinux could theoretically run on the xScale arm architecture, the advantages stemming from the ease of application code portability outweighed the potential increase of runtime performance provided by a real-time operating system. Tutorials for the installation, configuration, and use of OpenEmbedded are provided in the appendices for the convenience of peer researchers.

5.2.2 Application Code

After the OpenEmbedded Linux operating system was installed and flashed to the Tier 2 Gumstix module, the activity recognition application [11] developed by Vineet Jolly for the old pants system was ported to the embedded Gumstix platform. The original activity recognition application was intended for use on a standard backend Linux PC. As such, certain optimizations were made to increase runtime performance in the embedded environment. To identify the bottlenecks and areas for improvement within the application, the source code needed to be profiled and a baseline for the general performance of the original application on the embedded environment needed to be evaluated. An opensource C/C++ source code profiler tool called Gprof was initially used to first measure the performance of the application on the standard Linux PC. To be able to use Gprof, a special GCC/G++ compiler flag option is utilized to link the Gprof components into the application code. Gprof essentially relies on a frequently expiring timer to monitor which subroutines in the application source code utilize the most CPU time. Upon each timer expiration, information regarding the currently executing subroutine is logged to an output file until the application terminates. This information is then later parsed by the Gprof tool to develop a histogram showing the

relative execution statistics for each subroutine in the application.

Unfortunately, the subroutine granularity provided by Gprof was not fine enough to precisely measure the runtime execution of the activity detection application. The application code itself contains several nested loop sections within a **Recognize Activity** function that are used to perform the matrix multiplication operations required by the SVD algorithm [11]. These operations include query vector generation and activity matching calculations. Also the streaming nature of the application was expanded to be able to operate on raw data from an input file that was previously captured from the old pants prototype. This change was made to provide a deterministic testing platform as well as to satisfy Gprof's requirement for the application to exit normally versus just abruptly ending the process. Diagnostic information sent to the standard output as well as log file I/O were also removed from the application because the execution of those procedures tended to skew the runtime performance results. The activity recognition application was also expanded to support 3-d accelerometer data sets.

A custom C/C++ code profiler tool called VTprof was developed in order to achieve finer runtime performance granularity. Rather than relying on an expiring timer to develop execution histograms, VTprof uses a user defined delta-time per code section approach. Instead of providing a compiler flag option when using VT prof, the user must include the profiler library into their source code and add start/stop conditions around each code segment of interest. The user must also register these start/stop conditions with the profiler. At runtime, as the code segment of interest begins, a start time is captured. Once the code segment has been completed, a stop time is captured and the difference between the two times is logged to an output file. This operation is performed for every code segment of interest, and once the execution of the entire application is complete, a performance report summary is generated. This report contains information regarding the execution times and frequency of

each code segment of interest. A tutorial on the use of the VTprof tool is provided in the appendix.

Although VTprof is somewhat cumbersome to use due to the manual configuration of each code segment of interest, its flexibility and finer execution performance monitoring granularity provides much more useful information than Gprof for the activity recognition application. VTprof also provides helpful statistics such as the minimum, maximum, and mean execution times for each code segment of interest as well as the number of times each segment was executed as shown in Figure 5.3. Since VTprof was implemented using standard C++, it along with the activity recognition application were easily ported to the embedded Gumstix platform using Bitbake. It should be noted that another motivation for the creation of VTProf was the lack of GProf support within the embedded Gumstix platform.

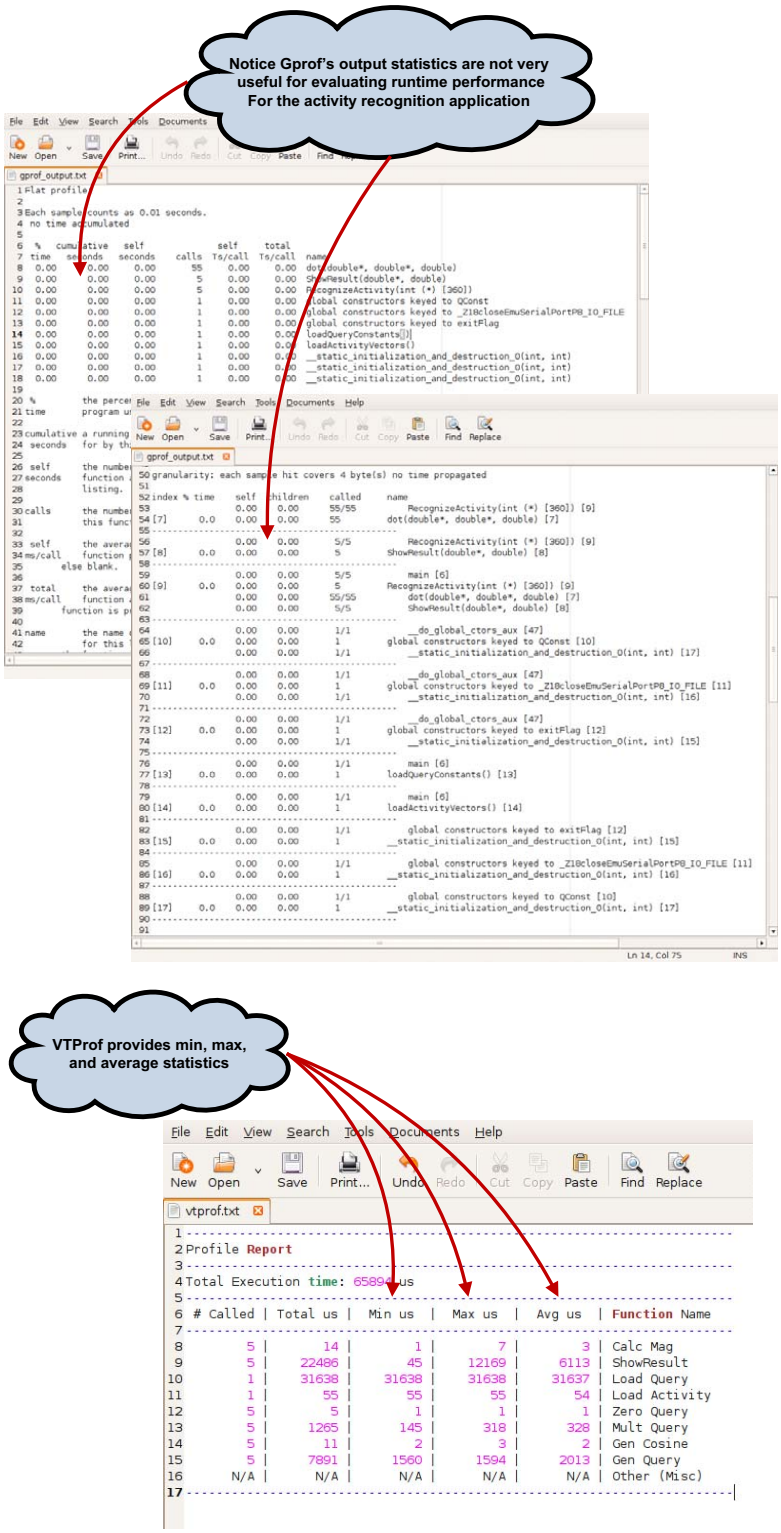

Figure 5.4: Sample Gprof and VTProf Output

Chapter 6

Communication

The jumpsuit prototype contains several mediums through which communications interfaces are provided. These interfaces include Tier 1 to Tier 2 communications as well as PC to jumpsuit communications. The following sections describe the interfaces and protocols implemented in the jumpsuit design.

6.1 I^2C

In the jumpsuit prototype, the Tier 1 nodes sample and temporarily buffer motion data from the sensor interfaces and then stream that information to the Gumstix Tier 2 node over an I^2C bus. I^2C is a serial communications bus protocol developed by Philips [27]. This protocol was selected since only two wires, SCL and SDA, are needed to implement an I^2C network. Fewer wires in the overall design of an e-textile lead to lower materials and manufacturing costs as well as help to introduce fewer points of failure since less physical wire interconnections need to be made. The I^2C protocol also provides multi-master support which allows each Tier 1 node to individually master the bus only when motion data is valid

and available for transfer. This multi-master paradigm is more efficient than having only one master Tier 2 node which successively polls each slave Tier 1 node for its motion data because the overall communications overhead for the system is reduced. Also, communication exchanges only occur when valid data is available, thus eliminating sample wait and hold time. If one node is busy performing an analog to digital conversion, another node may be able master the bus to transfer its data in the meantime.

I^2C is typically used to interface general-purpose microcontrollers to external EEPROM memories, and thus are often times only used in configurations with only one master device and one or more slaves. However, the Virginia Tech E-textiles Lab has found good success in using I^2C for multi-master inter-node communications. The I^2C protocol operates as follows: The bi-directional SCL and SDA lines are pulled high to 3.3V using pull-up resistors. These lines remain normally high until a node attempts to master the bus by issuing a start condition by successfully pulling down the SDA line. There is an arbitration process that occurs at this point to ensure that only one device can master the bus as any given time. The master then sends a unique 7-bit slave address in big-endian format while clocking each bit on the SCL line. The master then specifies a read/write bit denoting whether or not the master is expecting to send or receive data. For the multi-master configuration used in the jumpsuit prototype, this bit typically always set to 1 for sending data to the Tier 2 node. Next, if a slave matching the destination address is present on the network, the slave device will acknowledge with an ACK bit on the SDA line. The master will then begin transferring one data byte at a time with the slave acknowledging reception of the data after each byte. Once all of the data bytes are transferred and acknowledged, the master issues a stop condition and the control of the SCL and SDA lines are relinquished. Figure 6.1 shows an example I^2C transaction. It should be noted that the I^2C protocol supports three data speeds: standard (100 kbps), fast (400 kbps) and high speed (3.4 Mbps) although the

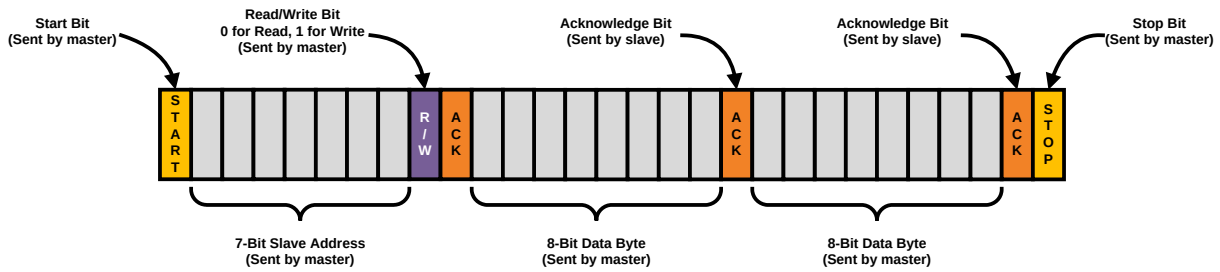

Figure 6.1: Example I^2C Transaction

Atmega8L microcontrollers only support the standard and fast speeds [28]. For the activity recognition application, the I^2C network is configured to operate at 400 kbps.

Since both an optimized and an event-driven software model were developed for the jumpsuit system, corresponding networking protocols were implemented. Each of these protocols was developed on top of I^2C to support the exchange of data between nodes on the network. The protocol used in conjunction with the optimized solution featured a raw payload model. Only the information necessary for the activity recognition application was included in each packet. As shown in Figure 6.2, each packet consists of ten bytes containing the I^2C network address of the of the sender, the current sample count, the X channel accelerometer data, the Y channel accelerometer data, the Z channel accelerometer data, and the gyro angular acceleration data. It should be noted that since the Atmega8L features a 10-bit A/D converter, each acceleration value is stored as two bytes.

The network software developed for the flexible software model provides a higher level of abstraction similar to that of TCP or UDP implemented on top of IP. The concept was to provide network protocol layered on top of I^2C that would allow for complex networking schemes for use in any e-textile network regardless of topology of the nodes. Support for packet routing, virtual circuit connections, and error checking were included. This protocol is called the Virginia Tech E-textile Data Protocol (VTEDP) and each packet contains both a header and data segment much like a UDP datagram. The header itself contains nine bytes

+	Bits 0 - 7	Bits 8 - 15
0	Source ID	Sample Count
16	X Channel Acceleration Data	
32	Y Channel Acceleration Data	
48	Z Channel Acceleration Data	
64	Gyro Angular Acceleration Data	

Figure 6.2: Raw Payload Packet

including the source node address, destination node address, virtual circuit ID, sequence number, service ID, checksum, and data length as shown in Figure 6.3. The data segment contains the message payload. For the jumpsuit prototype, the raw data payload used for the optimized model was aggregated into each VTEDP message, although for other applications this data segment could contain theoretically any kind of information and not just serialized motion data. Each VTEDP packet is then wrapped inside an I^2C frame much like the MAC layer used in ethernet and broken up into clocked byte-wise acknowledged exchanges in the same manner outlined in the I^2C protocol description above. Although, VTEDP introduces a fair amount of communications overhead, for some applications, the ability to route information through the network and support large numbers of nodes distributed across an e-textile is valuable. Since I^2C natively only supports 128 addresses within the 7-bit address space, utilizing an upper layer protocol, like VTEDP, becomes essential for dense network configurations. VTEDP provides a 16-bit address space as well as a level of transmission robustness through the use of sequence numbers and checksum values.

+	Bits 0 - 7	Bits 8 - 15
0	Source Address	
16	Destination Address	
32	Virtual Circuit ID	Sequence Num
48	Service ID	Checksum
64	Data Length	Payload Byte 0
80	Payload Byte ...	Payload Byte N

Figure 6.3: Raw VTEDP Packet

6.2 Serial

Once a packet is received by the I^2C interface daughter card on the Tier 2 Gumstix module, the data is parsed and transferred serially to the Gumstix Verdex motherboard for processing. If a raw payload packet was received, then a simple 4-bit checksum is constructed before the data is forwarded to the Gumstix. It should be noted that the upper nibble of the checksum byte is masked to indicate the end of an exchange to the Gumstix serial parsing algorithm. This serial connection is run at 230k baud with 1-8-1 framing and no flow control. In the case of a VTEDP packet, the header information is parsed, stripped, and then the encapsulated data is transferred in the same manner as the raw payload model.

6.3 Bluetooth

An interesting improvement of the jumpsuit prototype over the old pants system is in the use of Bluetooth. The previous pants system used the serial port profile provided in the Bluetooth stack to form a wireless serial connection between the pants and the back-end PC. Raw data was then simply streamed to the PC for activity recognition processing. In the new

jumpsuit prototype, TCP/IP over Bluetooth is used to form a personal area network (PAN) connection between the jumpsuit and any device supporting the PAN Bluetooth protocol. The user can simply Secure Shell (SSH) into the jumpsuit from a remote machine and execute commands wirelessly just like any other remote Linux machine. Users can also conveniently transfer files on and off the jumpsuit using Secure Copy (SCP). It should be noted that no application data is transferred over Bluetooth for the activity recognition application, unlike the earlier pants versions of this application [16] [11]. All of the processing is performed on the garment itself, and the Bluetooth connection merely serves as a convenient interface through which to interact with the jumpsuit.

Chapter 7

Results

This chapter summarizes the experimentation performed to validate the hardware and software components described in the previous chapters. First a validation of the individual acceleration modules is provided. Next, the entire jumpsuit prototype system, containing both the Tier 1 and Tier 2 nodes, is tested using the activity recognition application. Finally, the runtime performance of the I^2C network is evaluated and a discussion regarding the maximum sample rates sustainable within the jumpsuit prototype is provided.

7.1 Individual Acceleration Module Accuracy

Each Tier 1 acceleration module was individually tested to ensure the accuracy and validity of the motion data acquired from each sensor device. Initially, the motion data captured from some Tier 1 acceleration modules was found to be intermittently invalid due to the non-deterministic output of the the accelerometer sensor. The source of the problem was discovered and the design of the Tier 1 acceleration modules was modified as described in the following two sections.

7.1.1 Avoidance of Slow VDD Rise Time

After some preliminary testing, it was discovered that the MMA7260 accelerometer made by Freescale contains an undocumented design flaw that affects the output of the device. A slow VDD rise time on the power supply pin of the accelerometer can potentially cause the incorrect initialization of the trimming circuitry inside the device. This condition results in incorrect acceleration outputs on the X, Y, and Z channels of the accelerometer. The output values will typically become clamped to either 0V or 3.3V and will not respond accordingly as motion is imparted on the device. It should be noted that temperature also appears to have some effect on the likelihood of mis-initialization.

To rectify this problem, a PMOS power transistor was added to the acceleration module to facilitate the powering on of the accelerometer after all of the other components on the board have been successfully powered on and initialized. A GPIO pin was tied to the gate of the PMOS transistor to act as an on/off toggle switch for the accelerometer as shown in Figure 7.1. Applying a logic 0 to the gate of the PMOS transistor after the VDD supply has settled as well as applying a logic 1 to the enable pin allows for proper initialization of the accelerometer. Figure 7.2 shows the proper enabling of the accelerometer sensor output when the PMOS transistor is switched on and then back off.

7.1.2 Accelerometer and Gyroscope Output

Once the VDD rise time problem was averted, the rotational acceleration tests were conducted. Each board was rotated through 360 degrees to ensure a 2 g swing through the plane of orientation parallel to the force of gravity. Waveforms demonstrating the correct output of the X, Y, and Z accelerometer channels is provided in Figure 7.3. A simple X, Y, Z, and Gyro oscillation test was then performed to validate the functionality of the acceleration module by

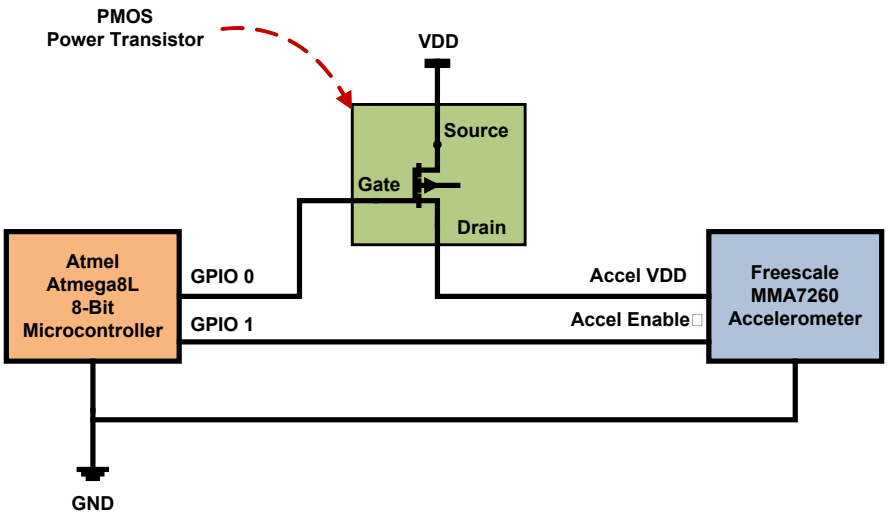

Figure 7.1: PMOS Power Transistor Configuration

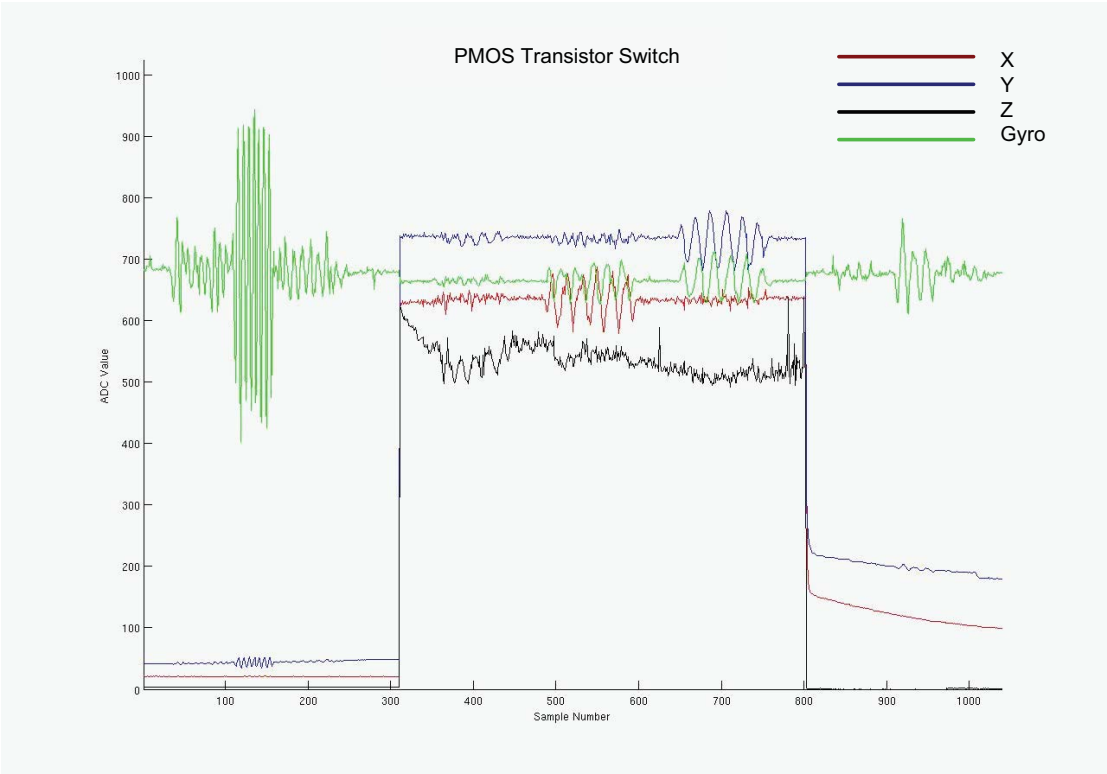

Figure 7.2: Switching Accelerometer On and Off Via PMOS Transistor

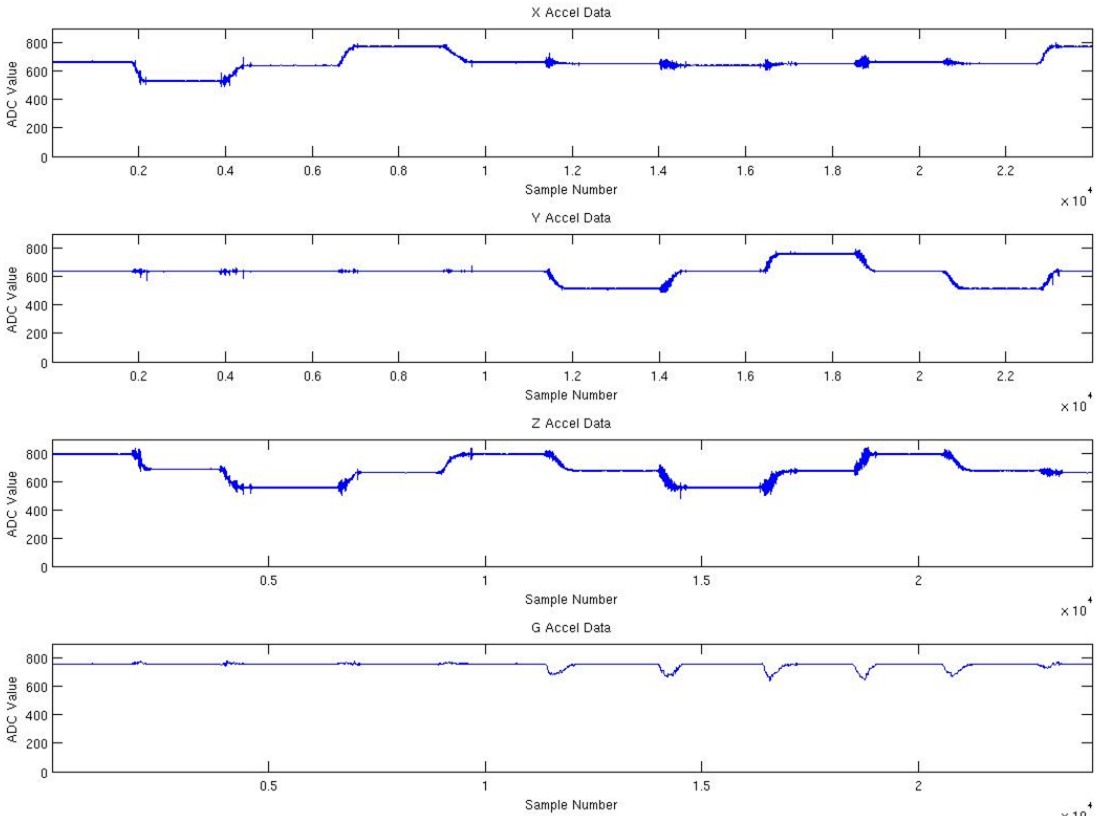

Figure 7.3: 2 g Swing Level Test

holding the board in a horizontal orientation, level with the ground, and moving the module up/down, left/right, forward/backward, and rotating the board clockwise/counter-clockwise in repeating patterns. Output waveforms for this test are provided in Figure 7.4.

7.2 Activity Recognition Application Performance

After the operation all of the individual acceleration modules were validated, the two-tier jumpsuit system as a whole was tested. Acceleration modules were placed on the mid-calf, mid-thigh, hip, mid-forearm, and mid-upper arm areas as well as one module in the middle of the chest as shown in Figure 7.5. It should be noted that the placement of these modules is

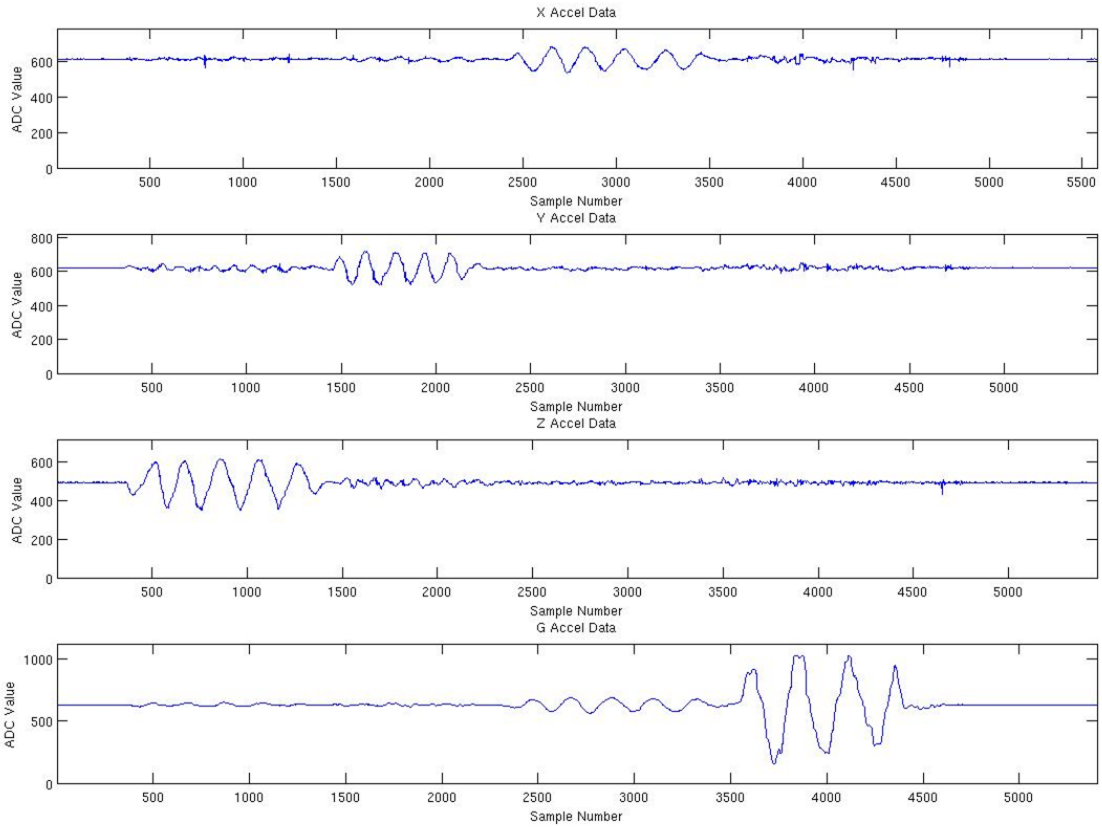

Figure 7.4: Hand Held Oscillation Test

Figure 7.5: Acceleration Module Placement

different from the joint locations used in the previous pants system. Later, custom e-textile fabric, featuring an intersecting grid of tinsel wire buses, was woven. The completed textile is shown in Figure 7.6.

7.2.1 Jumpsuit Prototype Validation

The expanded activity recognition application was then loaded to the Tier 2 Gumstix module. Training data for several rudimentary activities was then collected to calibrate the SVD

Figure 7.6: Jumpsuit Prototype

algorithm using the classifier techniques provided in [11]. Once training was complete, the prototype jumpsuit e-textile was evaluated and found to be able to successfully recognize the prescribed motions of the person wearing the garment as shown in Figure 7.7. Application level processing for activity recognition example application was shown to be adequately supported within the embedded platform. The correct detection of the motions for marching in place, referred to here as mark time, by the activity recognition application is demonstrated in Figure 7.7.

7.2.2 CPU Utilization

To further validate the runtime performance of the Gumstix xScale microprocessor, the activity recognition application was profiled using the VTProf library. By profiling strategic code segments of interest, it was determined that most of the CPU time is utilized in the

```
File  Edit  View  Terminal  Tabs  Help
% ./recognize -s -p /dev/ttyUSB0

Cosines:
0.679511 0.119958 0.842223 0.151764 0.298558
Subject is Marktime

Cosines:
0.672860 0.099523 0.848489 0.165484 0.287359
Subject is Marktime

Cosines:
0.641621 0.078822 0.870458 0.163593 0.245502
Subject is Marktime

Cosines:
0.675797 0.144372 0.842332 0.127835 0.297345
Subject is Marktime

Cosines:
0.700427 0.175054 0.819636 0.117226 0.334128
Subject is Marktime
%
```

Figure 7.7: Recognize Application Output (Marching in Place)

generation of the query vectors for the SVD algorithm. Most of the significant processing is spent in the code segments that contain nested loops which are performing matrix-matrix multiplication. The relative CPU utilization times for each of the major code segments within the **Recognize Activity** subroutine are summarized in Table 7.1.

7.3 Network Communication Performance

Typically, in most multi-node systems, network communications performance is a critical factoring component which affects the overall runtime execution. This is specially true in the case of data stream-centric applications like sensor-based activity recognition. As such, an analysis of the utilization of the I^2C network resources is provided in the following sections.

7.3.1 Theoretical Network Bandwidth Utilization

A theoretical model of the projected network bandwidth usage for varying numbers of acceleration sensing modules was developed. As a baseline, the projected estimations were

Table 7.1: Relative Percentage Utilization of CPU Within Recognize Subroutine

Code Segment	Percentage CPU Time	Description
Generating Query Vector	78.10	Populate initial query vector with buffered motion data
Multiply Query Vector and Query Constants	12.83	Perform matrix-matrix multiplication of query vector and query constants
Show Result	8.16	Match corresponding activity and display to user
Calculating Query Magnitude	0.74	Calculated the square root of the dot product of the query vector with itself
Generating Cosines	0.09	Compare query vector to activity vectors to generate cosines
Zero Out Query Vector Result	0.04	Clear the resultant query vector

calculated using information from the I^2C bus protocol specification [27] as well as some experimental data collected from benchmarks performed on the jumpsuit prototype.

The benchmarks were conducted as follows: First, a Tier 1 node was configured to repeatedly transmit a test packet at a relatively slow frequency of 4 Hz with the I^2C interface configured for fast mode (400 kHz). Two test packet sizes were utilized: 19 bytes for the event-driven model, and 10 bytes for the optimized software model. A logic analyzer was then utilized to capture the waveforms of the transmitted messages as shown in Figure 7.8 and Figure 7.9 respectively. Upon inspection, the overall delta time needed to transmit a single packet contained within an I^2C frame was found to be on average 1.192 milliseconds for the event-driven software model and 440 microseconds on average for the optimized software configuration.

The criteria for the projected bandwidth utilization model was constructed using several metrics. The number of overall bits, P, actually transmitted per I^2C frame for a 19 byte

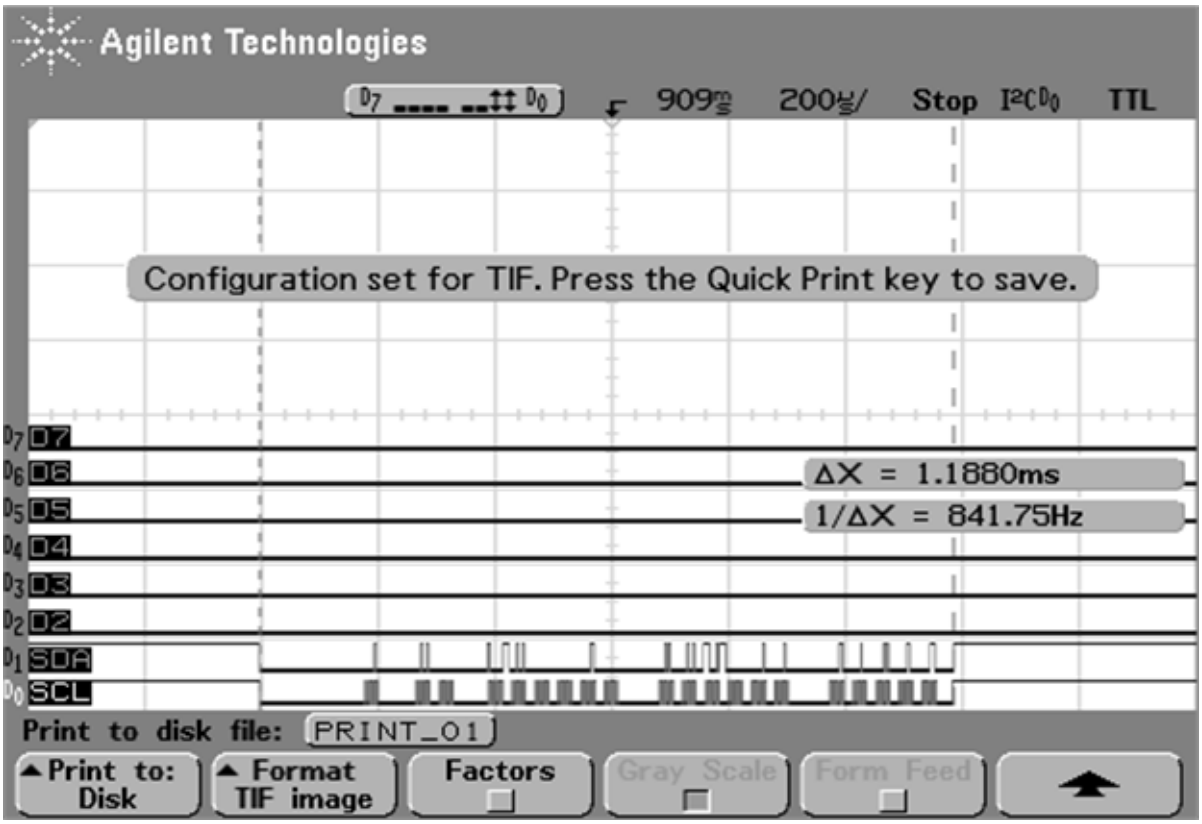

Figure 7.8: Logic Analyzer Capture Waveform for a 19 Byte Packet

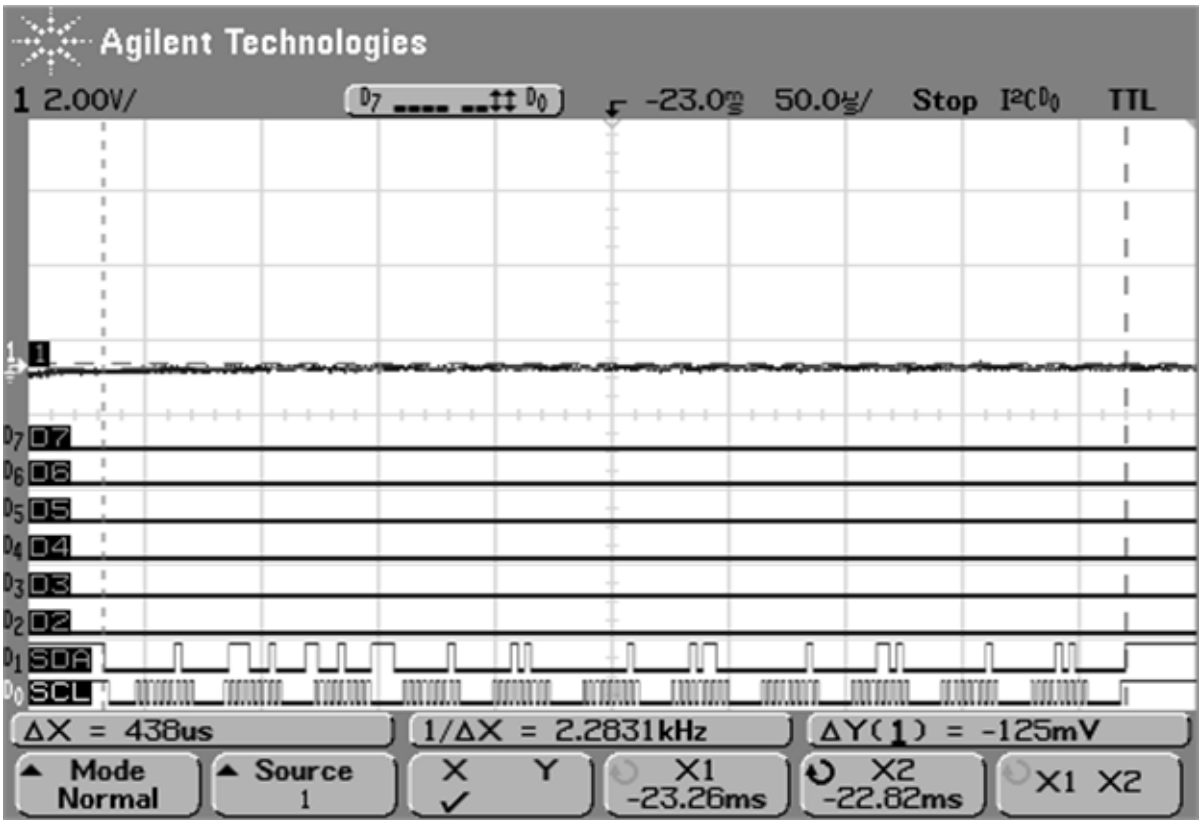

Figure 7.9: Logic Analyzer Capture Waveform for a 10 Byte Packet

message is 182 bits. For a 10 byte payload, there are 101 bits transmitted. This can be confirmed with the calculations shown in Equations 7.1 and 7.2.

$$
\begin{aligned}
P_{event\_driven} &= \textit{1 start bit + 7-bit address + 1 r/w bit + ACK bit +} \\
&\quad \textit{(19 data bytes)(8 bits/byte + 1 ACK bit/byte) + 1 stop bit} \\
&= \textit{1 start bit + (20 bytes)(8 bits/byte + 1 ACK bit/byte) +} \\
&\quad \textit{1 stop bit} \\
&= \textit{1 start bit + (20 bytes)(9 bits/byte) + 1 stop bit} \\
&= \textit{1 start bit + 180 bits + 1 stop bit} \\
&= \textit{182 bits}
\end{aligned} \tag{7.1}
$$

$$
\begin{aligned}
P_{optimized} &= \textit{1 start bit + 7-bit address + 1 r/w bit + ACK bit +} \\
&\quad \textit{(10 data bytes)(8 bits/byte + 1 ACK bit/byte) + 1 stop bit} \\
&= \textit{1 start bit + (11 bytes)(8 bits/byte + 1 ACK bit/byte) +} \\
&\quad \textit{1 stop bit} \\
&= \textit{1 start bit + (11 bytes)(9 bits/byte) + 1 stop bit} \\
&= \textit{1 start bit + 99 bits + 1 stop bit} \\
&= \textit{101 bits}
\end{aligned} \tag{7.2}
$$

Although the I^2C interface hardware in the Atmega8L microcontroller was configured using the fast mode for 400kHz SCL oscillation, the actual SCL wire twiddle rate was observed to be on average 344 kHz throughout the transmission of each acknowledged data byte. Significant relative delay was introduced between successive data byte writes due to the software processing of the I^2C network driver. Additional delay was also observed between

the transmission of start and stop conditions and the data bytes contained therein. This rather byte-pulsed transmission phenomenon was modeled as a constant transmission of bits clocked at a slower frequency. The theoretical equivalent mean bitrate μ was found to be 152.7 kHz for the event-driven model and 229.5 kHz for the optimized solution. These values were calculated by dividing the number of raw bits transferred per I^2C packet by the total amount of time $\Delta t_{transmission}$ required to transmit the packet as shown in Equation 7.3.

$$\mu = \frac{P}{\Delta t_{transmission}} \tag{7.3}$$

The projected network bandwidth utilization model was then constructed based upon the assumption that a constant stream of I^2C packets could be transmitted with one packet directly after another. It should be noted that this theoretical model incurs no arbitration or bus mastering overhead, just a predetermined amount of raw data at fixed bitrate transmitted from a varying number of devices. Since a fixed packet size is assumed, the total bandwidth consumed grows as a multiple of the number of nodes active on the network. The theoretical maximum sample rate sr_n sustainable for a varying number devices can be estimated as shown in Equation 7.4 where μ represents the equivalent mean bitrate, n is the number of nodes on the network, and P is the packet size in bits. It should be noted that this sample rate is uniform across all devices on the network and is thus referred to as the global sample rate in the remainder of this text.

$$sr_n = \frac{\mu}{(n)(P)} \tag{7.4}$$

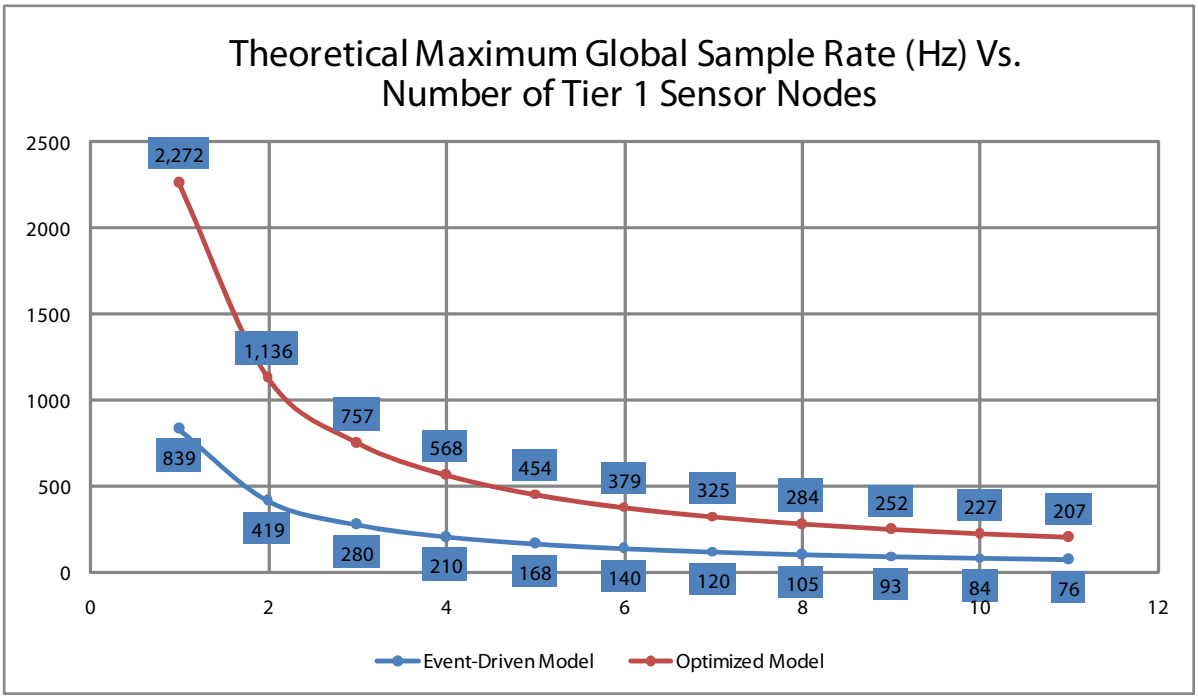

Figure 7.10: Theoretical Maximum Global Sample Rates

A plot of the maximum theoretical global sample rates for varying quantities of actively transmitting nodes on the I^2C bus is provided in Figure 7.10. As shown, an inverse relationship is implied between the global sample rate and the number of devices present on the network. As expected, the maximum theoretical sample rate sustainable for the event-driven software model is significantly less than the optimized model since more overhead is incurred from the use of the VTEDP protocol and the object-oriented event queue based approach in general as indicated by the benchmarks conducted.

7.3.2 Experimental Sample Rate Sustainability

Experimental data was collected for both the event-driven and optimized software models. This data was generated by incrementally adjusting the global sample rate upward for the specified number of nodes upward until I^2C network communication becomes unsustainable

Table 7.2: Experimental Data

Number of Nodes	Max Sample Rate Event-Driven (Hz)	Max Sample Rate Optimized (Hz)
1	340	1400
2	305	992
3	230	725
4	170	510
5	140	393
6	120	357
7	80	240
8	60	156
9	45	135
10	33	105
11	25	85

due to the exceeding of the bandwidth supported by the network infrastructure. Once the experimental limit was reached, the global sample rate was throttled back down to tune in the point at which stable network communication was observed. Table 7.2 summarizes the experimental data collected.

For the event-driven software model, a plot of the experimental global sample rate data with respect to the corresponding theoretical projections formulated by Equation 7.4 is provided in Figure 7.11. Comparing the experimental and theoretical values reveals substantial deviation, especially for network configurations containing larger numbers of nodes as well as the in case of a single node.

For the optimized software model, a plot of the experimental global sample rate data with respect to the corresponding theoretical projections formulated by Equation 7.4 is provided in Figure 7.12. Comparing the experimental and theoretical values reveals some deviation, particularly for network configurations containing larger numbers of nodes as well as in the case of a single node.

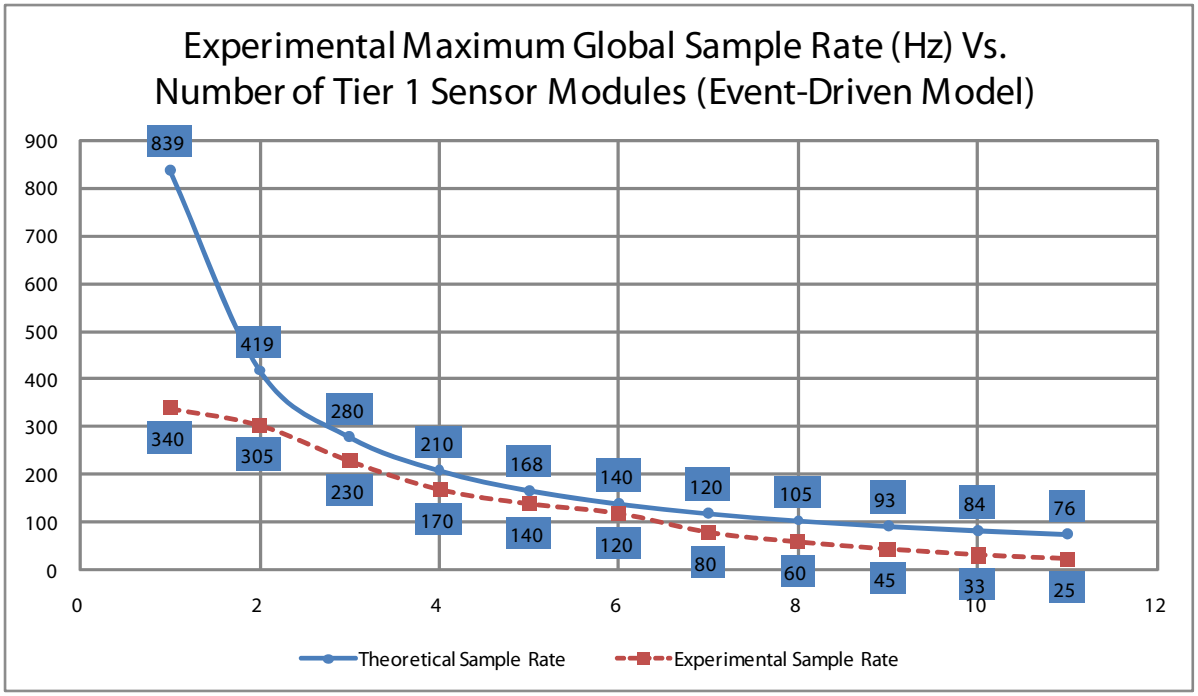

Figure 7.11: Experimental Maximum Sustainable Global Sample Rates (Event-Driven)

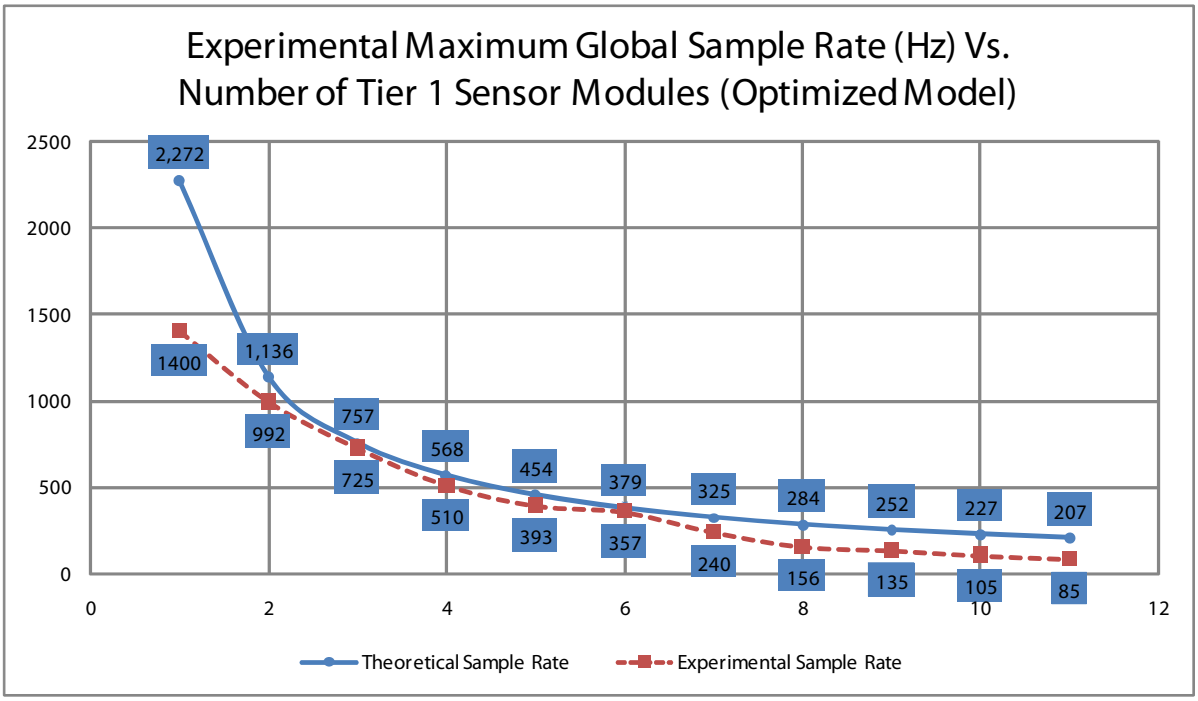

Figure 7.12: Experimental Maximum Sustainable Global Sample Rates (Optimized)

7.3.3 Deviation From Expected Results

The significant maximum sample rate deviations for the single node configurations are considered red-herring outliers because the upper limit of the sampling rates supported by a single acceleration module for this hardware platform is being reached. This limit is imposed by the hardware timer prescaling values and the respective I^2C driver used in conjunction with each software model. As such, the maximum sample rate achievable on a single node using the optimized software model is 1.4 kHz. The maximum sample rate achievable for the event-driven model, 350 Hz, is significantly lower due to the overhead inherently present in the object-oriented event-queue based approach.

The deviation of the bitrate values contained within the 7 to 12 node range can be attributed to the latency introduced by the I^2C bus arbitration process. It is widely stated, in most I^2C literature [28] [29], that arbitration is an uncommon occurrence. This statement is only partially true in that it only applies to the typical use cases of I^2C. In most applications, I^2C is almost always used in a single-master/multiple-slave configuration. Within that topology, arbitration is virtually impossible because there is only one master. However, in the case of a dense multi-master configuration, the need for arbitration becomes apparent and is actually a more common occurrence than most I^2C documentation suggests.

When multiple devices attempt to master the bus simultaneously, each node looks at the state of the SDA line and compares the current level with what it thinks the level should be using wired AND logic. If the current level of the SDA line does not match the expected level, the node assumes that it lost arbitration and is forced to wait until the bus becomes free again. It should be noted that this arbitration can go on for several bits depending on the data being sent by both nodes that are attempting to master the bus. The arbitration time is inherently longer in this particular system, since each Tier 1 node is always sending

to the same Tier 2 node slave address. This causes the first byte sent in every I^2C frame in this system to be identical for every packet transmitted regardless of the sender. Therefore, when two nodes attempt to master the bus at the same time, it will take at least one byte before one of the nodes realizes it has lost arbitration. Fortunately, by design, the I^2C protocol prevents data corruption with this arbitration process so no retransmissions of data are required by the node that won the arbitration. It should be noted though, that no random back-off procedure or alternate form of congestion or flow control is provided in this situation by the I^2C protocol. The node which loses arbitration waits until the bus becomes free and attempts a retransmission.

For the embedded jumpsuit platform, it is very difficult to accurately know when an arbitration has occurred, let alone be able to effectively measure the frequency of these occurrences. Since, the I^2C interface is the primary form of communication I/O for these devices, the options for debugging and evaluating the runtime execution are rather limited. However, a low-level rudimentary solution was discovered. In order to be able to accurately provide an indication that a node has realized it has lost an arbitration, one of the few unused general-purpose I/O (GPIO) pins on the Atmel Atmega8L microcontroller was utilized. A short, thin wire was soldered to this GPIO pin. The other end of wire was connected to the analog input of a logic analyzer. The I^2C network software driver was then modified to drive GPIO line high as soon as the Arbitration Lost condition in the I^2C interrupt service routine was reached. The GPIO pin is then driven back down low right before the I^2C interrupt service routine exits. Due to the limited resources of the testing environment, only two acceleration modules can be monitored for arbitration loss at any given time since the logic analyzer used only supports two analog input sources.

As a simple arbitration test scenario, three Tier 1 nodes were configured to send predetermined 10-byte test packets at the maximum sustainable sample rate for five nodes.

This test leaves plenty of bandwidth for each node to be able to successfully transfer its test packet before the next round of packets must be sent. As shown in Figure 7.13, it was discovered that at least one arbitration can commonly be detected while monitoring only two out of the three nodes. This effectively proves that arbitration does occur in even the minimal use case of multi-master I^2C. It was also discovered, as shown in Figure 7.13, that when two nodes attempt to master the bus, the node losing arbitration backs off and waits for the bus to become free so that it can attempt to master the bus again. Each byte of data sent by the node that won the arbitration process is acknowledged (ACK) successfully by the slave. However, an interesting phenomenon occurs once the bus becomes free and the losing node attempts to re-master the bus and transmit the 7-bit address of the slave node. The first 7-bit slave address and read/write bit comprising the first byte transferred is not acknowledged (NACK) by the slave, indicating a state where the slave device is not yet ready to accept a message. The sending node then retries sending the packet again after the failed communication attempt, and each byte the message is then successfully acknowledged by the slave device.

Upon observing many sample tests, it was demonstrated that this failed communication attempt occurs every time after an arbitration lost signal is present on the monitored GPIO line. This extra unacknowledged byte, along with its corresponding start and stop condition delays, incurs a 58us delay each time an arbitration occurs as shown in Figure 7.14. As indicated in Figure 7.15, the number of observed arbitrations increases multiplicatively to the number of nodes attempting to master the bus. The added delay caused by the transmission of the unacknowledged bytes builds up as the number of nodes on the network increases, thus leading to the deviation from the expected values since the theoretical maximum sample rate model does not factor in the overhead associated with this phenomenon.

It should be noted that the I^2C driver utilized in the jumpsuit prototype for the optimized

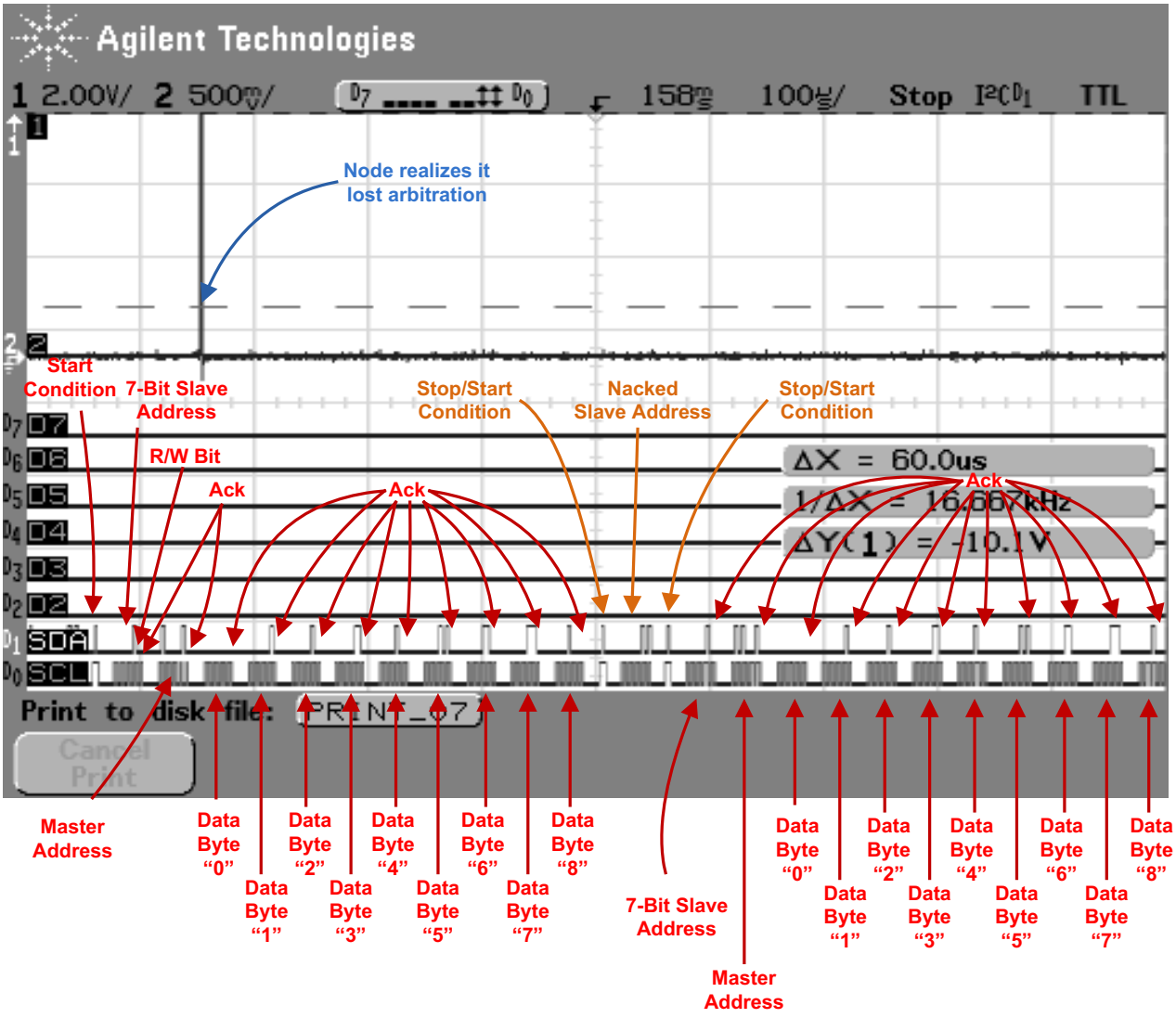

Figure 7.13: Logic Analyzer Capture of Arbitration Detection

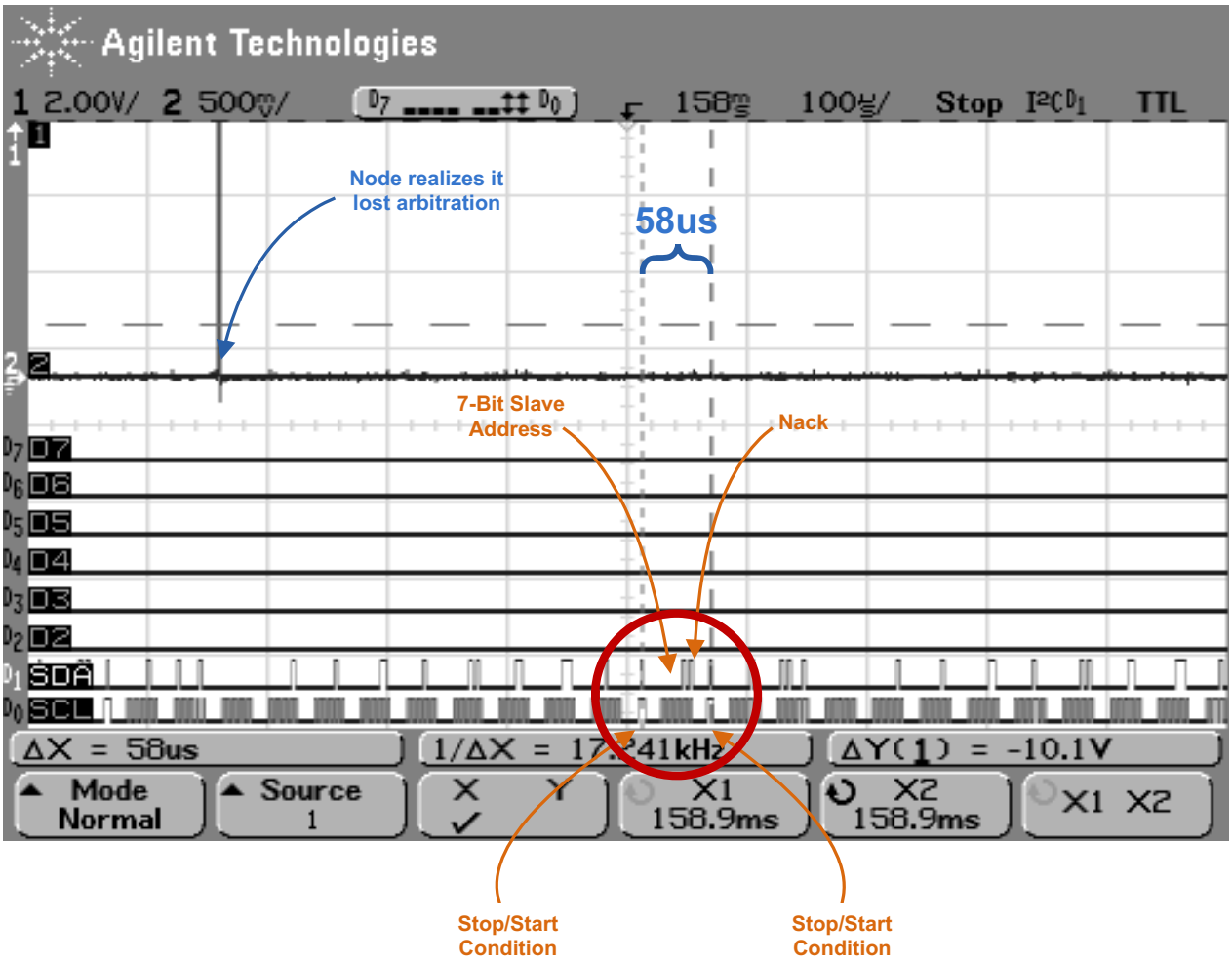

Figure 7.14: Logic Analyzer Capture of NACK

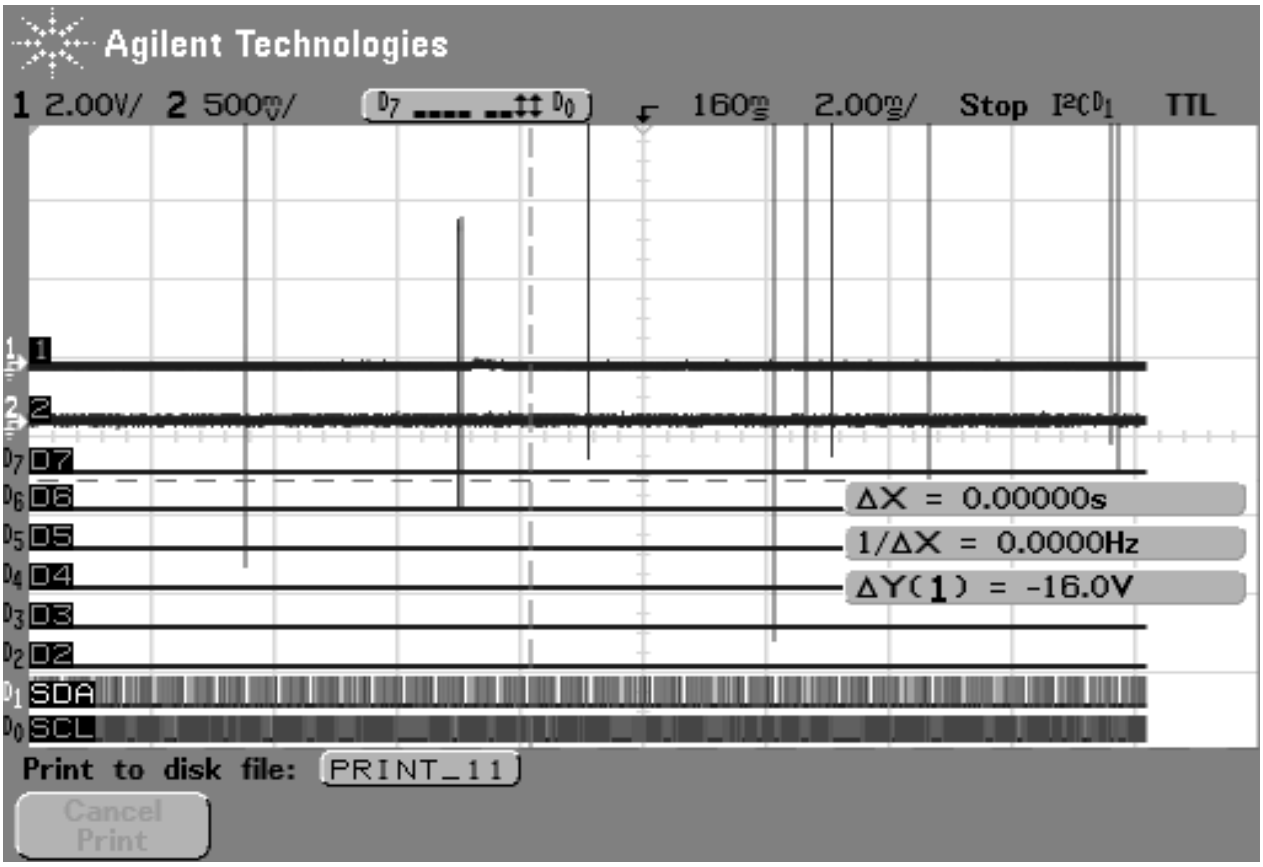

Figure 7.15: Logic Analyzer Capture of Multiple Arbitrations

software model was essentially the same driver code utilized in the previous pants system. The driver utilized in the event-driven model was based off this same design with added support for the generation of I^2C events. However, the finite state machine which controls the actual I^2C hardware interface was not modified. The added latency induced by the situation outlined earlier is considered a day-one issue and was therefore present in the previous pants system as well. This design flaw was simply highlighted by the fact that the I^2C communications network is being pushed to the limit in the experiments presented in this text. As the optimization of multi-master I^2C network communication is not the focus of this thesis, the improvement of the I^2C communication driver to resolve this issue is left as an exercise for future developers.

7.3.4 Comparison of Software Approaches

The overall results show that in general, for the event-driven software model, the experimental maximum global sample rates sustainable are significantly lower than the optimized model due to the increased packet size and the additional overhead incurred through the use of VTEDP and the event-queue based software architecture. As shown in Figure 7.16, both the event-driven and optimized software models conform to the inversely proportional relationship as expected by the theoretical approximations. As indicated in previous research [11], a sample rate between 20 and 120 Hz is suitable for the detection of most everyday activities. As such, the implementation presented in this thesis provides sustainability within that 20 to 120 Hz range for up to 11 nodes for the event-driven software model and is well above that range utilizing the optimized software model. The optimized approach supports a factor of almost three improvement when compared to the previous pants design where six acceleration modules are utilized. It should be noted that the experimental data showing the maximum sustainable global sample rate for each respective software model demonstrates

the physical limitations of this design and should not be thought of as average use case sample rate for typical applications.

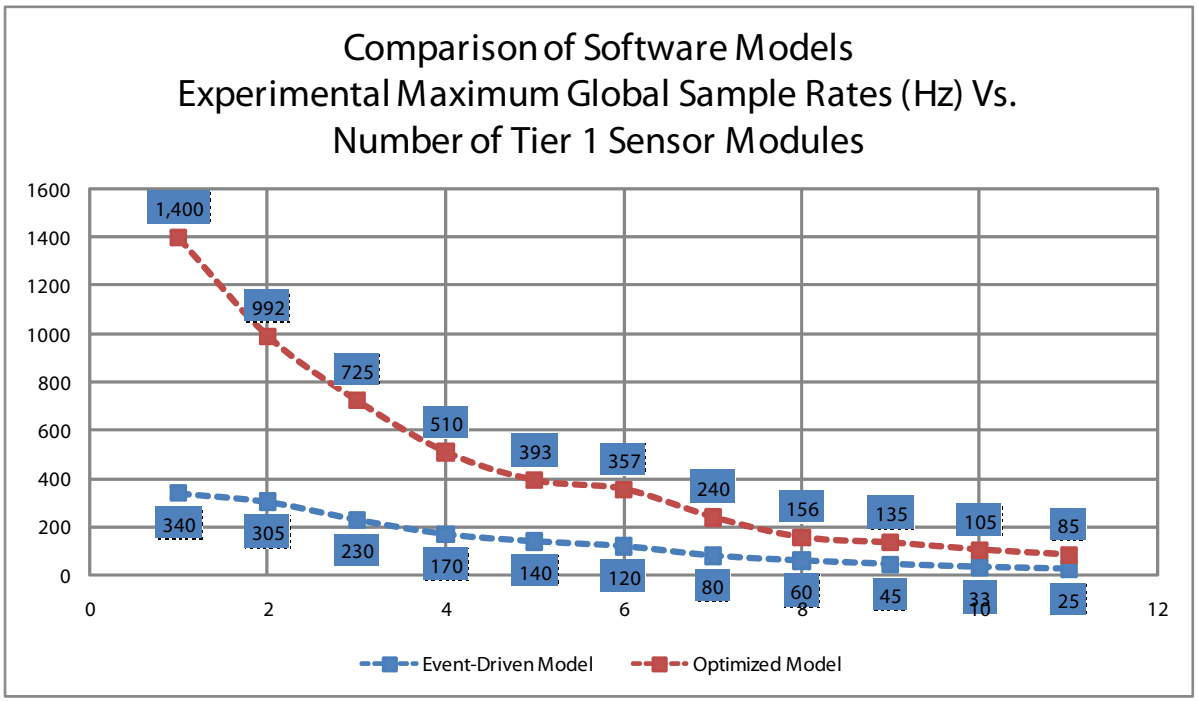

Figure 7.16: Comparison of Event-Driven and Optimized Software Models

Chapter 8

Conclusions

8.1 Contributions

In conclusion, many lessons learned can be taken from the development of the activity recognition jumpsuit prototype. Most notably, effective application-level processing, like the activity recognition application, is possible to achieve on an e-textile without the use of a back-end PC. Although the network communication appears to be the main bottleneck in the overall system, the raw data stream processing model implemented in this solution does provide sufficient performance for the 20 to 120 Hz sample rates needed to support activity recognition for up to eleven nodes using the flexible event-driven software model. The optimized solution, however, provides performance well surpassing the 20 to 120 Hz sample rate range.

The jumpsuit prototype system as a whole shows significant improvement over the previously developed pants design. The SVD based activity recognition algorithm has been shown to be successfully extended to utilize 3-d motion data collected at mid-limb locations rather than

being centered around the joints of the human body. The sample rate limitations imposed by the Bluetooth radio communication speeds in the previous pants have also been avoided since no motion data is streamed off of the textile. Computation has not only been successfully migrated to the confines of the wearable textile itself, but the generic infrastructure provided by the event-driven approach can be used to effectively realize complex applications other than just activity recognition. It should also be noted that fine grain application runtime performance within an embedded environment can be analyzed through the use of the VTProf source code profiler library.

8.2 Future Work

Depending the sample rate requirements for future applications, the raw data processing model may not be effectively scalable for dense sensor module networks. One certainly viable solution to increase the scalability of the design presented in this thesis is to separate the sections of network into subnets, and assign Tier 1 router nodes the task of managing the state of each specified region. Each router could then act as a regional manager by collecting and preprocessing the capture sensor data and then send summary statistics to one or more Tier 2 nodes. This approach would reduce the network bandwidth necessary for network as a whole and may extend the range of the number of devices physically supportable within an e-textile.

Future work for this research includes potential development in many areas. In terms of future work in the realm of hardware, a two-axis gyro should be included into the design of the acceleration modules. This would allow each acceleration module to act as a six-axis Inertial Measurement Unit (IMU) similar to those used in various land, air, and watercraft vehicles. The addition of a magnetic field sensor may also help provide some useful absolute positional

information for motion capture applications. Also, the 400 pF limitations associated with I^2C SDA and SCL line capacitance should be analyzed further as that is another factor with may limit the network performance of the system particularly with regard to dense sensor configurations. A rough estimation for capacitance added by each Tier 1 node attached to an I^2C bus is provided in the appendix. In terms of software, network software support work should be performed to increase network speed and throughput as well as to provide more complex routing algorithms.

The e-textile infrastructure provided in this research can serve as a strong starting point for future researchers to develop more complex applications tangent to the process of activity recognition. One such application is a self-aware wearable textile that contains nodes which can dynamically ascertain their individual locations on the garment. The jumpsuit infrastructure also provides application-level motion processing capabilities that would allow for future research into predictive fall technologies. With the memory space and processing power of the Gumstix platform, adaptive filter technology could be employed to monitor and analyze the gait of the person wearing the jumpsuit and potentially enact measures to reduce the severity of a fall, particularly in the case of the elderly.

Bibliography

[1] S. Park, K. Mackenzie, and S. Jayaraman, "The wearable motherboard: A framework for personalized mobile information processing (pmip)," in *Proceedings of the 39th Conference on Design Automation*, New Orleans, Louisiana, USA, June 2002.

[2] J. McHale, "Wearable computers help make individual soldiers part of the digital battlefield," *Military & Aerospace Electronics*, vol. 13, no. 7, July 2002.

[3] PPD, "Illum - cycling jacket with a twist," http://www.pdd.co.uk/en/work/consumer/illuminated_jacket/, cited Sept 2008.

[4] Swany, "Swany gloves," http://www.swanyamerica.com/, cited Sept 2008.

[5] T. Bessinger, "Lifetag system by raymarine," http://www.sailingworld.com/sailing-gear/electronics/lifetag-system-by-raymarine-51167.html, cited Sept 2008.

[6] E. Zegna, "Zegna sport," http://www.fashion.at/collections/2006/zegna7-2006.htm, cited Sept 2008.

[7] L. Ault and A. Awbrey, "Burton and apple deliver the burton amp jacket," http://www.apple.com/pr/library/2003/jan/07burtonipod.html, cited Sept 2008.

[8] Apple, "Tune your run," http://www.apple.com/ipod/nike/, cited Sept 2008.

[9] T. Martin, M. Jones, J. Edmison, T. Sheikh, and Z. Nakad, "Modeling and simulating electronic textile applications," in *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, Washington, DC, June 2004.

[10] D. Graumann, G. Raffa, M. Quirk, B. Sawyer, J. Chong, M. Jones, and T. Martin, "Large surface area electronic textiles for ubiquitous computing: A system approach," in *Fourth Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services*, Philadelphia, PA, August 2007.

[11] V. Jolly, "Activity recognition using singular value decomposition," Master's thesis, Virginia Tech, Blacksburg, VA, August 2006.

[12] P. E. Ross, "Benedetto vigna: The man behind the chip behind the wii," *IEEE Spectrum*, March 2007.

[13] A. S. Champy, "3d sensors in intuitive game design," *Analog Dialogue*, April 2007.

[14] L. V. D. Einde, W. Deng, P. Wilson, A. Elgamal, and P. Hubbard, "Neesit macbook accelerometer and video sensor platform (iseismograph) for education and research," in *38th ASEE/IEEE Frontiers in Education Conference*, Saratoga Springs, NY, 2008.

[15] C. Einsmann, "A validation of a simulation environment for motion sensing electronic textiles," Master's thesis, Virginia Tech, Blacksburg, VA, February 2006.

[16] J. Edmison, "Electronic textiles for motion analysis," Master's thesis, Virginia Tech, Blacksburg, VA, April 2004.

[17] D. Lehn, C. Neely, K. Schoonover, T. Martin, and M. Jones, "e-tags: e-textile attached gadgets," in *Communication Networks and Distributed Systems Modeling and Simulation Conference*, Philadelphia, PA, January 2004.

[18] F. Semiconductor, "+/- 1.5g - 6g three axis low-g micromachined accelerometer," http://www.freescale.com/files/sensors/doc/data_sheet/MMA7260QT.pdf, cited Sept 2008.

[19] I. Analog Devices, "+/- 300 degree/s single chip yaw rate gyro with signal conditioning," http://www.analog.com/static/imported-files/data_sheets/ADXRS300.pdf, cited Sept 2008.

[20] Atmel, "8-bit avr with 8k bytes in-system programmable flash," http://www.atmel.com/dyn/resources/prod_documents/doc2486.pdf, cited Sept 2008.

[21] I. S. Inc., "Piezoelectric film article introduction," http://www.imagesco.com/articles/piezo/piezo00.html, cited Sept 2008.

[22] N. Semiconductor, "Lp2992 micropower 250 ma low-noise ultra low-dropout regulator in sot-23 and llp packages," http://www.national.com/ds/LP/LP2992.pdf, May 2004.

[23] LinuxDevices.com, "Tiny linux sbc steps up to pxa270," http://www.linuxdevices.com/news/NS7437869444.html, cited Sept 2008.

[24] Gumstix, "breakout-gs," http://gumstix.com/store/catalog/product_info.php?products_id=140, cited Sept 2008.

[25] B. Sawyer, "Event-driven model," Private Communication, cited Sept 2008.

[26] Intel, "Intel pxa270 processor," http://www.phytec.com/pdf/datasheets/PXA270_DS.pdf, cited Sept 2008.

[27] P. Semiconductors, "The i^2c-bus specification," http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf, cited Sept 2008.

[28] T. Matouek, "i^2c bus: Inter integrated circuits bus by philips semiconductors," http://tmd.havit.cz/Papers/I2C.pdf, cited Sept 2008.

[29] NXP, "I2c vs can for multiple node networks," http://forums.semiconductors.philips.com/viewtopic.php?p=9675, cited Sept 2008.

[30] J. William R. Blood, *MECL System Design Handbook*. Motorola Semiconductor Products Inc., 1988.

[31] Technick, "Pcb impedance and capacitance calculator: Microstrip," http://www.technick.net/public/code/cp_dpage.php?aiocp_dp=util_pcb_imp_microstrip, cited Sept 2008.

[32] T. Electronics, "Idc flat ribbon cable," http://www.farnell.com/datasheets/37182.pdf, cited Sept 2008.

Appendix A

Gumstix Tutorials

A.1 How To Connect to the Gumstix

A.1.1 Connecting Via Serial Port

1. First, attach the gumstix motherboard to the I^2C converter board via the breakout-gs expansion board and connect the serial RS-232 connection from the PC to the gumstix assembly.

2. Make sure you have kermit installed. You can use Minicom if you prefer but it is more complicated. If kermit is not installed already, use

```
sudo apt-get install ckermit
```

3. Next Launch kermit

```
kermit -l /dev/<path_to_serial_port>
```

*** Note if you are using a USB to Serial RS-232 converted, the path you will most likely

use is /dev/ttyUSB0. If you plug in your USB to serial converted and it does not show up as ttyUSB0 but dmesg does acknowledge that it was attached, then you must do the following,

```
sudo apt-get remove brltty
```

This is caused by a bug in ubuntu, where a package meant to connect a serial brail teletype machine messes up the mounting of the USB to RS-232 converter to /dev/ttyUSB0.

3. Next load the correct setting into kermit and try to connect to the gumstix

```
C-Kermit> take ~/gumstix/gumstix-oe/extras/kermit-setup
C-Kermit> connect
```

4. Next connect power to the gumstix.

5. You should now see a 3 second countdown screen, and then you should see a verbose boot sequence. When prompted for a username and password, use the following,

```
user: root
pass: gumstix
```

6. To log out use,

```
ctrl + \ + c
```

7. Type "exit" to quit kermit

A.1.2 Connecting Via Bluetooth

When connecting to the gumstix wirelessly, TCP/IP over Bluetooth is used. For some reason, the gumstix always wants to be the master for the Bluetooth connection. This can

be changed after the initial connection is made. First set up the PC, then automatically bind the connection with the gumstix. Then log into the gumstix via SSH.

1. First, plug in the Bluetooth dongle into the PC and tell it to listen and accept a connection using,

```
pand --listen
```

2. Next, power on the gumstix. Upon booting, the gumstix should automatically attempt to connect to the PC and set up it's IP address on bnep0 as 192.168.1.2

3. Now setup the connection on the PC by setting its IP address using

```
sudo ifconfig bnep0 192.168.1.1
```

*** Note, if you get an error, just reboot the PC and try the process over.

4. Finally you can log into the gumstix via SSH using,

```
ssh root@192.168.1.2
```

When prompted for the password, use "gumstix".

*** Note you can also use standard SCP calls to the gumstix to transfer files back and forth.

A.2 OpenEmbedded Build Environment Setup

These instructions are for setting up an OpenEmbedded build environment using Ubuntu Linux.

1. Set up your machine with the require packages. Make sure you have the following packages installed:

```
subversion
gcc
patch
help2man
diffstat
texi2html
texinfo
libncurses5-dev
cvs
gawk
python-dev
python-pysqlite2
```

To install these packages, use

```
sudo apt-get install <name of package>
```

*** Note you can install multiple packages at once i.e.

```
sudo apt-get install subversion gcc patch help2man diffstat
     texi2html texinfo
libncurses5-dev cvs gawk python-dev python-pysqlite2
```

** Note, on ubuntu **/bin/sh** is linked to **/bin/dash**. This will cause an image that will not boot once compiled. To fix this run,

```
sudo dpkg-reconfigure dash
```

And say no to installing dash as **/bin/sh**

2. Checkout the OE source from subversion repository

```
mkdir ~/gumstix
cd ~/gumstix
svn co
     https://gumstix.svn.sourceforge.net/svnroot/gumstix/trunk
     gumstix-oe
```

** Note to check out a specific version, use the following command instead:

```
svn co -rXXX
     https://gumstix.svn.sourceforge.net/svnroot/gumstix/trunk
     gumstix-oe
```

XXX is the version number you want to check out. Currently, revision 308 is being used, which should build fine with no errors. At this point you should have the OE source in **gumstix/gumstix-oe**

3. Set up Environment. Append gumstix's bash profile file to our .bashrc using,

```
cat gumstix-oe/extras/profile >> ~/.bashrc
```

*** Note, if you did not check out the source code from subversion to **/home/–username–/gumstix/gumstix-oe** this file with not work because it relies on that explicit path to work.

4. Set up Source code caching. This part may be skipped for a single user system, but it is recommended to set up a global cache directory for the tarballs to be stored in for a multi-user setup to save both disk space and download time

```
sudo groupadd oe
sudo usermod -a -G oe your_username
```

*** Note, the last command may fail when using the machines in CCM Lab since they are using NFS and the home directories for the users are mounted over the network. Typically when a Linux machine is set up on the CCM Network, a dummy account is created and added to **/etc/passwd**. You may have to add yourself after this dummy entry to the **/etc/passwd** file as follows:

```
your_username:x:1001:1001:your_username,,,:/home/your_username
     :/bin/bash
```

Then modify the permissions on the sources directory

```
sudo mkdir /usr/share/sources
sudo chgrp oe /usr/share/sources
sudo chmod 0775 /usr/share/sources
sudo chmod ug+s /usr/share/sources
```

5. Start the build. You must log out and then log back in for the changes you made to the build environment to take effect.

*** Note this assumes you are building an image for the Verdex motherboard, if you want to build an image for the Basix motherboard, you must edit

```
/gumstix/gumstix-oe/build/conf/auto.conf
```

Comment out the verdex line and uncomment the Basix line. Start the build using bitbake as follows:

```
bitbake gumstix-basic-image
```

Bitbake should complain here if you do not have all the required packages. If you missed something, install it with apt-get and execute the command above again. Ignore any messages about **user.collection/packages/*/*.bb** as this directory is for your user developed programs which we do not have any yet so do not worry about this.

*** Note that although bitbake also suggests you to install the psyco JIT compile for better performance, this in not required. To install it, use

```
sudo apt-get install python-psyco
```

This supposedly speeds up the build time by acting as a just-in-time compile like Java but little performance gain by using this.

6. Wait a long long long time ... You may as well go do something else why performing this initial build as it takes hours to complete. This is mainly because the first time you do the build it has to download a lot of tarballs from an ftp server which is usually really slow.

Once the build completes, you should have the image in

```
~/gumstix/gumstix-oe/tmp/deploy/glibc/images/gumstix-custom-verdex
```

If you do,

```
$ ls -l ~/gumstix/gumstix-oe/tmp/deploy/glibc/images/gumstix-
     custom-verdex/
```

You should see something like,

```
Angstrom-gumstix-basic-image-glibc-ipk-2007.9-test-20071101-
     gumstix-custom-verdex.rootfs.jffs2
gumstix-basic-image-gumstix-custom-verdex.jffs2
modules-2.6.22-r1-gumstix-custom-verdex.tgz
uImage-2.6.22-r1-gumstix-custom-verdex.bin
```

*** Note that there is really only one .jffs2 file and that's the one with the long name. The **gumstix-basic-image-gumstix-custom-verdex.jffs2** is the symbolic link to that file with a shorter name for convenience for typing when flashing the file to the gumstix.

And now you should have a build environment setup and ready to go.

A.3 Flashing the OpenEmbedded Filesystem Image to the Gumstix

Note that for the setup we have, it is advised to flash the filesystem image using a serial connection to the gumstix. The I^2C to serial conversion board has a 9-pin d-sub connector which breaks out the FFUART on the gumstix to the PC. Simply connect the gumstix motherboard to the I^2C converter board via the breakout-gs expansion board and connect the serial RS-232 connection from the PC to the gumstix assembly.

First setup a serial connection.

1. Make sure you have kermit installed. You can use Minicom if you prefer but its is more complicated. If kermit is not installed already, use

```
sudo apt-get install ckermit
```

2. Next Launch kermit,

```
kermit -l /dev/<path_to_serial_port>
```

*** Note if you are using a USB to Serial RS-232 converted the path you will most likely use is **/dev/ttyUSB0**. If you plug in your USB to serial converted and it does not show up as ttyUSB0 but dmesg does acknowledge that it was attached then, you must do the following:

```
sudo apt-get remove brltty
```

This is because of a bug in ubuntu where a package meant to connect a serial brail teletype machine messes up the mounting of the converter to /dev/ttyUSB0.

3. Next load the correct setting into kermit and try to connect to the gumstix.

```
C-Kermit> take ~/gumstix/gumstix-oe/extras/kermit-setup
C-Kermit> connect
```

4. Next connect power to the gumstix. You should see a U-Boot screen with a 3-second countdown. Press any key to stop the gumstix from booting into Linux and put it into the programming mode. You should see a terminal that says

```
GUM>
```

5. Next, tell U-Boot to start receiving a file at location a2000000 in RAM.

```
GUM> loadb a2000000
```

6. Break the serial connection and return to the kermit prompt by using,

```
------------------------------------------------------------
C-Kermit 8.0.211, 10 Apr 2004, localhost



Current Directory: /home/jchong/gumstix/gumstix-oe/tmp/deploy/glibc/images/
Communication Device: /dev/ttyUSB0
Communication Speed: 115200
Parity: none
RTT/Timeout: 01 / 02
SENDING:  => GUMSTIX-BASIC-IMAGE-GUMSTIX-CUSTOM-VERDEX.JFFS2
File Type: BINARY
File Size: 7491192
Percent Done: 2   /
...10...20...30...40...50...60...70...80...90..100
Estimated Time Left: 00:13:32
Transfer Rate, CPS: 8963
Window Slots: 1 of 1
Packet Type: D
Packet Count: 68
Packet Length: 4096
Error Count: 0
Last Error:
Last Message:


X to cancel file, Z to cancel group,  to resend last packet,

E to send Error packet, ^C to quit immediately, ^L to refresh screen.
------------------------------------------------------------
```

Figure A.1: Download Monitor Screen

```
ctrl-\-c
```

7. Send the filesystem image using,

```
C-Kermit> cd ~/gumstix/gumstix-
     oe/tmp/deploy/glibc/images/gumstix-custom-verdex/
C-Kermit> send gumstix-basic-image-gumstix-custom-
     verdex.jffs2
```

You should see a download monitor screen which appears similar to Figure A.1.

The download should take around 12 to 15 minutes to complete depending on what packages you selected in the filesystem build.

8. Once the download completes, the filesystem is loaded into RAM. You still need to write the contents of the RAM to flash. To do this type,

```
C-Kermit> connect
```

9. Erase the current contents of flash using,

```
GUM> protect on 1:0-1
GUM> erase all
```

*** Note, this is the most important part. The 1:0-1 protects the bootloader from being erased. Omitting this line will brick your gumstix although it can be fixed by sending back to gumstix to be factory re-flashed for a small fee.

10. Then write the contents of RAM to flash using,

```
GUM> cp.b a2000000 40000 ${filesize}
```

11. Now we have to load the kernel image using the same steps as above.

```
GUM> loadb a2000000
C-Kermit> send uImage-2.6.21-r1-gumstix-custom-verdex.bin
C-Kermit> connect
GUM> katinstall 100000
GUM> katload 100000
GUM> bootm
```

12. Your gumstix should now boot into Linux. When prompted for a username and password, use the following:

```
User: root
Password: gumstix
```

A.4 Developing Applications for Gumstix OpenEmbedded

1. Set up the build environment using the instructions found in the OpenEmbedded Build Environment Setup appendix.

2. To Build a Sample C Hello World program use instructions here:

```
http://www.gumstix.net/Software/view/Build-system-
     overview/Hello-world-tutorial/111.html
```

** Note this is for C++ for a new gumstix install – fresh out of box

In order to run C++ programs you need to download

```
libstdc++6_4.1.2-r10_armv5te.ipk
```

from here:

```
http://gumstix.net/feeds/current-old/glibc/ipk/armv5te/
```

Then,

```
scp ~/gumstix/libstdc++6_4.1.2-r10_armv5te.ipk
     root@192.168.1.2:/home/root
```

While SSHed into the gumstix,

```
cd /home/root
ipkg install helloworld2_1.0.0-r2_armv5te.ipk
```

3. **/gumstix/gumstix-oe/user.collection/packages** is where user programs source code goes.

4. hellofunction is the template for a multi-source-file C++ program using bitbake with make. See step 2 for more details.

5. To compile, use

```
bitbake <name_of_program>
```

*** Note, to trigger a recompile, you need to either change the revision number in the bitbake recipe, or you can use the rebuild flag (there is no make clean).

```
bitbake -c rebuild <name_of_program>
```

6. Then copy the package over to the gumstix via scp, (package not built in source dir).

```
scp ~/gumstix/<path_to_package> root@192.168.1.2:/home/root
```

While SSHed into the gumstix,

```
cd /home/root
ipkg install <name_of_package>_1.0.0-r2_armv5te.ipk
```

7. To remove an installed package, use

```
ipkg remove <name_of_package>_1.0.0-r2_armv5te.ipk
```

A.5 Setting up Bluetooth on the Verdex 400xm-bt Gumstix

After flashing the Gumstix with a fresh kernel image, you must tweak some setting on the Gumstix to get Bluetooth configured properly. First, set up a serial connection:

1. Make sure you have kermit installed. You can use Minicom if you prefer but it is more complicated. If kermit is not installed already, use

```
sudo apt-get install ckermit
```

2. Next Launch kermit,

```
kermit -l /dev/<path_to_serial_port>
```

*** Note if you are using a USB to Serial RS-232 converted the path you will most likely use is **/dev/ttyUSB0**. If you plug in your USB to serial converted and it does not show up as ttyUSB0 but dmesg does acknowledge that it was attached then, you must do the following:

```
sudo apt-get remove brltty
```

This is because of a bug in ubuntu where a package meant to connect a serial brail teletype machine messes up the mounting of the converter to **/dev/ttyUSB0**.

3. Next load the correct setting into kermit and try to connect to the gumstix.

```
C-Kermit> take ~/gumstix/gumstix-oe/extras/kermit-setup
C-Kermit> connect
```

4. Next connect power to the gumstix.

5. You should now see a 3 second countdown screen and then you should see a verbose boot sequence. When prompted for a username and password use the following,

```
user: root
pass: gumstix
```

6. Once logged in, edit **/etc/init.d/bluetooth** as follows:

```
vi /etc/init.d/bluetooth
```

Make sure that the startup script uses the ttyS1 serial port, this is sometimes set incorrectly in some OpenEmbedded builds. If it is set correctly you should see this line,

```
HCIATTACH_TTY=ttyS1
```

Next make sure the following lines appear in the start stanza after the **/sbin/modprobe proc-gpio** line:

```
echo "AF3 out" > /proc/gpio/GPIO9
echo "AF1 in"  > /proc/gpio/GPIO42
echo "AF2 out" > /proc/gpio/GPIO43
echo "AF1 in"  > /proc/gpio/GPIO44
echo "AF2 out" > /proc/gpio/GPIO45
```

7. Next edit /etc/network/interfaces to set up the Bluetooth interface with a static IP address as follows:

```
vi /etc/network/interfaces
```

Comment out the line

```
#iface bnep0 inet dhcp
```

And replace it with,

```
iface bnep0 inet static
address 192.168.1.2
netmask 255.255.255.0
network 192.168.1.0
gateway 192.168.1.1
broadcast 192.168.1.255
```

8. Finally, edit /etc/default/bluetooth to configure pand as follows:

```
vi /etc/default/bluetooth
```

Change the line,

```
PAND_ENABLE=false
```

to,

```
PAND_ENABLE=true
```

Then replace the line,

```
PAND_OPTIONS="--listen --role NAP"
```

with,

```
PAND_OPTIONS="--role PANU --connect 00:11:67:24:05:BD"
```

where **00:11:67:24:05:BD** is the MAC address of the Bluetooth dongle on the PC you are going to connect to. This address can be found by executing the following command on the PC,

```
hcitool dev
```

9. Now the Gumstix should be properly configured to automatically try to pair with the Bluetooth dongle on the PC and set up its own IP address to **192.168.1.2**

Appendix B

Atmel Tutorials

B.1 Programming an Atmel Microcontroller

To program an Atmel Microcontroller, you must have the programmer connected to the PC and the correct build environment set up.

1. To set up the build environment, you must have the gcc, avrdude, avr-libc, binutils-avr, and gcc-avr packages installed. To do this, use

```
sudo apt-get install  gcc avrdude avr-libc binutils-avr gcc-avr
```

2. Next, many of the applications rely on the libraries found in **/jumpsuit/lib/**. Inside that directory you will find avr-2.0 and avr1.1. You may use either set of libraries for your application depending upon your specific needs.

3. Next make sure your directory is **/jumpsuit/build** as this contains some scripts the programmer uses to flash the Atmel.

4. Develop your source code in directories parallel to the lib and build directories. You may

```
~$ lsusb
Bus 005 Device 006: ID 1131:1001 Integrated System Solution Corp. KY-BT100
       Bluetooth Adapter
Bus 005 Device 005: ID 05e3:1205 Genesys Logic, Inc. Afilias Optical Mouse H3003
Bus 005 Device 002: ID 0409:0058 NEC Corp. HighSpeed Hub
Bus 005 Device 001: ID 0000:0000
Bus 004 Device 002: ID 03eb:2104 Atmel Corp.
Bus 004 Device 001: ID 0000:0000
Bus 002 Device 003: ID 0403:6001 Future Technology Devices International, Ltd 8-bit
       FIFO
Bus 002 Device 001: ID 0000:0000
Bus 003 Device 001: ID 0000:0000
Bus 001 Device 001: ID 0000:0000
```

Figure B.1: Permissions Setup Output

change this if you like, but then you must edit the top source directory path in the Makefile for your applications as follows:

Find the line,

```
top_srcdir = ..
```

Change it to,

```
top_srcdir = <path_to_your_source_files>
```

5. Next attach the avrmkII programmer to the PC. Most likely when you first connect the programmer, it will not have the correct permissions to allow you to flash the Atmel. To setup the permissions properly do the following:

```
lsusb
```

You should see output similar to Figure B.1.

Now edit the permission for the programmer by specifying the bus number and device ID number.

```
sudo chmod a+rw /dev/bus/usb/004/002
```

***Note, you will need to use your specific numbers found from lsusb

6. Next connect the programmer to the board containing the Atmel. A six pin programming header is used, and the connector must face the proper orientation to work. Fortunately, plugging in the programmer backwards does not damage the Atmel. If it is connected properly you should see a green light on the programmer. If the light is orange, then you have it backwards. The is red when the programmer is disconnected or when it is connected and the Atmel is not powered on.

7. Finally in your source code directory execute the following:

```
make clean
make
make usb_prog
```

This will delete any old hex files, create the new hex files, and flash the Atmel with your program.

B.2 Fixing the slow or skewed clock on the Atmel Atmega8l

If when you flash the Atmel with your program code and it appears to be running, but much slower than you anticipated for an 8Mhz clock rate, it may be the case that the fuses have been set incorrectly to use the internal oscillator.

To fix this problem, connect the Atmel programmer to the board and issue the following

command to reset the fuses on the Atmel Atmega8l microcontroller:

```
make usb_init
```

This should reset the fuses to use the 8Mhz internal oscillator.

Appendix C

VTProf Tutorial

C.1 Using VTprof to Profile your C/C++ Source Code

VTprof is a profiling library used for time specific sections of C and C++ code. Unlike gprof, VTprof can be used for more than just monitoring the execution time of functions. It can be used to time loop structures or even individual instructions with microsecond precision. This flexibility does come as a cost. However, you must include timing code segments explicitly in your source in order to use VTprof.

1. Include profile.h in your source files, and make sure you have profile.cpp in the same source directory.

2. VTprof relies on using extern struct pointer variables to collect the timing information. You must declare a function counter pointer for each function or code segment you want to monitor in profile.h. You must also edit the corresponding declarations in profile.cpp.

3. Next, declare two struct timeval's in profile.h for each code segment you want to monitor. Two timevals are required because one is needed for capturing the time the segment starts

and the other is for capturing the time the code segment ends. Again you must edit the corresponding declarations in profile.cpp.

4. Next, you must edit your main subroutine to initialize these function counter variables and place gettimeofday calls before and after each code segment you want to monitor. For more information about the gettimeofday call, see the Linux manpages.

```
man gettimeofday
```

To, illustrate how to edit the main subroutine, the following example code is provided in Figure C.1.

5. Finally, compile your source code normally using G++. When you run your program, the function counters will monitor your code and a profile report will be written to a text file profile.txt. The report will look similar to Figure C.2.

```
////////////////////////////////////////////////////////////////////////////////

// At the top of your code be sure to include the profile header file,
#include "profile.h"

// In your main routine,

      ////////////////////////////////////////////////////////////////////////
      // Add this to the beginning of your program to set up an outputfile, and
      // the profiler
      FILE *outputFile;
      outputFile = fopen("profile.txt", "w");
      init_profiler( &myProfiler );
      ////////////////////////////////////////////////////////////////////////

      ////////////////////////////////////////////////////////////////////////
      // Register each function you want to monitor with the profiler
      ////////////////////////////////////////////////////////////////////////
      counter_test_function1 = register_function( "test_function1", &myProfiler );
      counter_test_function2 = register_function( "test_function2", &myProfiler );
      counter_test_function3 = register_function( "test_function3", &myProfiler );
      ////////////////////////////////////////////////////////////////////////

      ////////////////////////////////////////////////////////////////////////
      // Add this before the work really begins to save the start time for overall
      // program
      gettimeofday (&prog_start, NULL);
      ////////////////////////////////////////////////////////////////////////

*******************************************************************************
// Actual program starts here
*******************************************************************************
      // Your code goes here

      ////////////////////////////////////////////////////////////////////////
      // Add this right before each function call you want to monitor
      gettimeofday (&f1_start, NULL);
      ////////////////////////////////////////////////////////////////////////
            <**** Your function call goes here ****>
      ////////////////////////////////////////////////////////////////////////
      // Add this right after each function call you want monitor
      gettimeofday (&f1_current, NULL);
      end_func_call(counter_test_function1, &t_start, &t_current );
      ////////////////////////////////////////////////////////////////////////

      // Your code goes here

*******************************************************************************
// Actual program ends here
*******************************************************************************

      ////////////////////////////////////////////////////////////////////////
      // Add this to the end of the code to record the stop time for the overall
      // program and print the profile report
      gettimeofday (&prog_stop, NULL);
      print_profile_report( outputFile, &myProfiler, &prog_start, &prog_stop );
      ////////////////////////////////////////////////////////////////////////
```

Figure C.1: Example Code

```
-------------------------------------------------------------------------------
Profile Report
-------------------------------------------------------------------------------
Total Execution time: 64844 us
-------------------------------------------------------------------------------
 % Time  | # Called | Total us |  Min us  |  Max us  |  Avg us  | Function Name
-------------------------------------------------------------------------------
    92.8 |    60190 |       20 |     2315 |     4427 |     3169 | test_function1
     7.0 |     4510 |       50 |       71 |      118 |       92 | test_function2
     0.1 |       81 |       30 |        2 |        3 |        2 | test_function3
     0.0 |       31 |       20 |        1 |        3 |        1 | test_function4
     0.1 |       40 |       30 |        1 |        2 |        1 | test_function5
    -0.0 |      N/A |      N/A |      N/A |      N/A |      N/A | Other (Misc)
-------------------------------------------------------------------------------
```

Figure C.2: Profile Report

Appendix D

Miscellaneous

D.1 Soldering Tips and Tricks

D.1.1 General Soldering Tips

Before reading any further or attempting to solder anything on your own, be sure to check out the following link. These tutorials from sparkfun contain excellent information on soldering various types of components.

```
http://www.sparkfun.com/commerce/present.php?p=BEE-6-
    SolderingBasics
```

In general soldering is not that complicated an operation to perform, but it does take a fair amount of practice to become proficient. It is advised that you start by getting comfortable soldering through-hole components first, then move on to surface mound devices. If you do not solder on a regular basis or feel that you are rusty on the hand-eye coordination skills needed, it is advised that you warm up on a test piece before trying to solder anything too

important.

D.1.2 SMD Soldering Techniques

Once you become proficient using the soldering iron for through-hole parts, you will find that surface mount devices (SMD's) are the next logical progression. Typically for any given design, you will have only maybe 2 or 3 difficult parts to put on like a microcontroller, accelerometer, gyro, and etc. Everything else will be easy SMD's like resistors, capacitors, inductors, and voltage regulators. Most of the resistors, capacitors, and inductors used come in the same size packages, typically 0805, 0603, or 0402. These numbers refer to the widths and lengths of the actual device. It is advised that you start out practicing on soldering 0805 packages since those are the biggest of the sizes typically used. You'll find that once you can put these size components on proficiently, soldering smaller packages like 0402 is just as easy.

To solder one of these rectangular SMD packages, start by putting a small amount of solder on one of the pads. Next, using tweezers, carefully pick up the part and move into position on top of the pads. Then press the tip of the iron onto the small bit of solder previously put on the pad. This should secure the device in place so that you can apply solder to the other side of the part without having to hold it with the tweezers. Often times, several of these types of components need to be put on the board, so just solder one side of all the parts first to get them in place then go back and solder the other sides of all the parts at once at the end. Here are some good links to tips on SMD soldering

`http://www.sparkfun.com/commerce/present.php?p=BEE-7-SMDSoldering`
`http://www.sparkfun.com/commerce/present.php?p=SMD-HowTo-2`
`http://www.sparkfun.com/commerce/present.php?p=Crazy%20Soldering`

http://www.youtube.com/watch?v=EWeAOemTY_E&feature=related

D.1.3 SMD With Leads Soldering Techniques

Small Outline Packages (SOP) like TSOP, MSOP, SOIC, and even Thin Quad Flat Pack (TQFP) packages that have short gull-wing leads on 2 or 4 sides are relatively easy to attach as well. You can use a technique similar to the regular SMD parts where you can put solder on one of the pads, typically on a corner, and then align the part tacking down the corner. Then, I solder the opposite corner, and fill in the rest of the pins in between. As I solder each individual pin, I place the tip of the soldering iron onto the edge of the lead where lead meets the pad. Then simply heat up that pin and touch the solder to the lower half of the pin. Usually the solder will wick down the remaining length of the pin and meet the pad since solder flows toward the heat source naturally. Alternatively you can just hold the soldering iron horizontally across groups of pins and allow the solder flow across them all effectively shorting them all together. Then you can use a solder wick copper braid to remove the excess solder thus removing the shorts. The following links demonstrate this technique.

http://www.youtube.com/watch?v=e5qYG95bbz8&feature=related
http://www.youtube.com/watch?v=AcbezX8TrOU&feature=related
http://www.youtube.com/watch?v=0EUAEtri3h0&feature=related

D.1.4 BGA Soldering Techniques

Soldering Ball Grid Array (BGA) parts may look like a daunting task at first, but in general, they are fairly easy to put on if you use a stencil. Since BGA parts already have the solder balls mounted on the underside of the device, no external solder is needed. Also the stencil

used is a one-time-use stencil that gets permanently sandwiched between the PCB and the part. The stencil is typically made of vinyl with an adhesive backing on it. Simply peel the stencil off of its paper or plastic backing and align the holes in the stencil with the pads on the PCB. If you mess up and do not get the stencil aligned properly on the first try, do not worry, just carefully peel the stencil off the board, realign it, and reapply it. This lab usually get the stencils from a company called Stencils Unlimited. It usually take about a week to get a stencil from them, and if it is a custom stencil like the one used for our gyro parts, you will have to select the custom stencil option from their website and enter in the part number. Typically, you will have to order in multiples of ten stencils, with each set of ten costing around 75 dollars. The part number for the gyro stencil is BT03208007007050. Below is a link to the Stencils Unlimited webpage:

`http://www.stencilsunlimited.com/`

Now, in order to actually attach the BGA part to the PCB you will need a hot plate. Some people have had success using a toaster oven or a skillet to mount these devices but it has been found that the using a hot plate directly works quite well. You can monitor the temperature of the hot plate using an infrared thermometer. Through testing, it was found that around 285 degrees Celsius is the ideal temperature for reliably mounting these parts. If you cannot pin down 285 degrees Celsius exactly on the hot plate, do not worry, anything between 270-295 will work (this corresponds to setting the knob on the hot plate to a little above the markings for the low setting). This may seem too hot for the part to handle as per the datasheets, but you have to remember that the actual part is not touching the hot plate or even the PCB for that matter since it is essentially sitting on top of the grid of solder balls. Once you have the hot plate set to the right temperature, carefully put the BGA part on top of the stencil, aligning the solder balls to the holes. Next, place the assembly on top of the hot plate roughly 1/3 of the way from the outer edge. This is an area you want to

be monitoring with the infrared thermometer. Let it sit there for around 1-1/2 minutes to 2 minutes, then carefully remove the assembly from the hot plate using tweezers or needle-nose pliers. Let the assembly cool for a minute or two, and you should be good to go. Note, never push down on top of the BGA at any point when the solder balls are hot, just let gravity do the work. On another side note, if you are simply prototyping a board that requires the use of a BGA part, you may want to look into using a schmart board. These generic perf-boards are specially designed with vias which align perfectly to the pitch of the ball grid array. Here are links to this product and how to use them.

`http://www.schmartboard.com/index.asp?page=products_bga&id=109`
`http://www.youtube.com/watch?v=D3PTpaB4kro`

Here is a link to a side profile view of how the BGA works,

`http://www.necel.com/pkg/en/mount/4/4_2/index.html`

D.1.5 QFN Soldering Techniques

Parts with Quad Flat No leads (QFN) packages are quite possibly the most difficult types of components to put on. This is because there are no leads sticking out, and the only way to connect the part is by soldering the pads on the bottom of the component to the pads on PCB. This can not be accomplished using a soldering iron especially for pad located in the middle of part. There are two quite reliable methods for attaching these QFN components but much care must be taken when performing these techniques as to not damage the components or create unwanted shorts between pads.

The first technique is to use solder paste. Solder paste is basically a cream composed of small solder particle suspended in a flux based substrate liquid. It can be stored at room

temperature, but it is recommended to store it in a refrigerated environment to prolong the life of the product since there is an expiration date associated with it. With refrigeration, the manufacturers of most solder paste products say the paste can last up to 9 months or so. Through experimentation, it was found that the solder paste is still good past a year and a half with virtually no problems. If you do refrigerate the paste, be sure to allow it to warm up to room temperature before using it. Also, mix it thoroughly with a small stirring stick before use, as typically the solder particles will settle in the fluid slightly.

The concept of this solder paste method is simple. Apply the paste to the pads on the PCB surface using a reusable stencil, line up the part above the pads in the proper orientation, and carefully mate the component to the pasted surface. It may be possible to create a jig system to ease the process of placing the QFN components onto the pastes pads, but from experience it is just better to practice doing it by hand a lot until you become proficient. If you place the part crooked or incorrectly or some how smudge the paste, you can simply remove the component at this time, clean both the part and PCB surface with rubbing alcohol, and attempt another placement. Once the part is placed correctly, steps similar to the BGA approach can be used to permanently attach the component. The solder paste will "flash" at around 265 degrees Celsius. The color of the paste will change from grey to silver, and the component will be electrically connected once it cools. Be sure to allow adequate cooling time before moving the assembly as the solder pasted components may shift while the solder is still hot. The following links illustrate the solder pasting technique.

`http://www.sparkfun.com/commerce/present.php?p=Reflow%20Toaster`
`http://www.sparkfun.com/commerce/present.php?p=Reflow%20Skillet`

The second method found to work quite well for soldering QFN parts involves essentially turning the QFN part into a BGA component by hand and applying heat with hot air re-flow

gun. This is the preferred method as it reliable creates the electrical connection between the component and the board and also typically takes less time to perform the operation compared to the hot plate solder paste method.

First, clean both the surface of the PCB as well as the part with rubbing alcohol. Next, tin the pads on the PCB surface with solder by hand. Try to distribute the solder as evenly as possible to minimize having some solder bumps higher than others. Minor variations in the size of the bumps is acceptable. Next, use a flux pen to apply a think layer of flux to the bottom of the QFN part. Apply a small amount of solder to each pad on the part creating essentially solder balls similar to a BGA part. Next, apply a fairly generous layer of flux paste to the bottom of the component. The flux paste helps keep the component down in place during alignment as well allow the solder to make a better connection between the part and the PCB once heat is applied. Next, align the part over the tinned pads on the PCB. Using the hot air reflow gun, use a circular motion around the perimeter of the part to heat up the entire area evenly. Once the solder reaches its "flashing" point, you will see the part drop down into position, and slightly realign itself. The conduction of heat from the paste flux as well as the viscous nature of solder in its liquid state helps to make the process almost self aligning. Once the part looks like it is finished dropping down and moving, turn off the hot air reflow gun and wait for the assembly to cool down. If for some reason, the part is not aligned properly, you can remove it again with the hot air reflow gun in a similar processes as you attached it. You can then clean off all the solder from the PCB and QFN component using copper braid and attempt the entire process over again. The following link demonstrates a similar technique.

http://www.curiousinventor.com/guides/Surface_Mount_Soldering/QFN

D.2 How to Tile Boards Into a Merged Panel Using Gerbermerge

1. First design the boards you wish to build using eagle. Once the board layout has been completed, use one of the following CAM process for 2 or 4 layer boards to generate gerber files for each individual board:

```
/jumpsuit/gerber/SFE-Special-Border_2.cam
/jumpsuit/gerber/SFE-Special-Border_4.cam
```

2. Next, using layout.cfg as a template add the paths to each board directory to the end of a configuration file as follows:

```
[name_of_board]
Prefix=<path_to_board_files>
*TopLayer=%(prefix)s.cmp
*BottomLayer=%(prefix)s.sol
*TopSilkscreen=%(prefix)s.plc
*BottomSilkscreen=%(prefix)s.pls
*TopSoldermask=%(prefix)s.stc
*BottomSoldermask=%(prefix)s.sts
Drills=%(prefix)s.drd
BoardOutline=%(prefix)s.bor
Repeat=4
```

*** Note for 4 layer board you must include paths to the inner layer files as follows,

```
[name_of_board]
```

```
Prefix=<path_to_board_fil
*TopLayer=%(prefix)s.cmp
*Layer2=%(prefix)s.lay2
*Layer3=%(prefix)s.lay3
*BottomLayer=%(prefix)s.sol
*TopSilkscreen=%(prefix)s.plc
*BottomSilkscreen=%(prefix)s.pls
*TopSoldermask=%(prefix)s.stc
*BottomSoldermask=%(prefix)s.sts
Drills=%(prefix)s.drd
BoardOutline=%(prefix)s.bor
Repeat=9
```

3. Next, edit the sources line in the gerber Makefile to use the configuration file created in step 2.

```
SRCS=<name_of_your_configuration_file>.cfg
```

4. To run gerbermerge and merge all of the individual gerbered boards into one panel, type

```
make
```

You should see a screen that appears as in Figure D.1.

5. When you are satisfied with the percentage of board area utilization, type

```
ctrl-c
```

```
---------------------------------------------------------------------------
      echo y | gerbmerge gumstix_converter.cfg

      ***************************************************
      *           R E A D    C A R E F U L L Y          *
      *                                                 *
      * This program comes with no warranty. You use    *
      * this program at your own risk. Do not submit    *
      * board files for manufacture until you have      *
      * thoroughly inspected the output of this program *
      * using a previewing program such as:             *
      *                                                 *
      * Windows:                                        *
      *          - GC-Prevue <http://www.graphicode.com> *
      *          - ViewMate  <http://www.pentalogix.com> *
      *                                                 *
      * Linux:                                          *
      *          - gerbv <http://gerbv.sourceforge.net>  *
      *                                                 *
      * By using this program you agree to take full    *
      * responsibility for the correctness of the data  *
      * that is generated by this program.              *
      ***************************************************

      To agree to the above terms, press 'y' then Enter.
      Any other key will exit the program.


      Reading data from gumstix_converter ...
      Job gumstix_converter: (4 instances)
        Extents: (793,1590)-(35163,14586)
        Size: 3.437000" x 1.299600"

      Trimming Excellon data to board outlines ...
      Trimming Gerber data to board outlines ...
      Performing layout ...
      ======================================================================
      Starting random placement trials. You must press Ctrl-C to
      stop the process and use the best placement so far.
      Estimated maximum possible utilization is 93.3%.
        1411 placements / Smallest area: 19.1 sq. in. / Best utilization: 93.7%
---------------------------------------------------------------------------
```

Figure D.1: Screen After Gerbermerge

```
-------------------------------------------------------------------------------
Interrupted.
Computed 1411 placements in 5 seconds / 277.0 placements/second
=====================================================================
Writing merged output files ...
--------------------------------------------------
     Job Size : 2.724200" x 6.999000"
     Job Area : 19.07 sq. in.
   Area Usage : 93.7%
   Drill hits : 424
Drill density : 22.2 hits/sq.in.

Tool List:
  T01 0.0984"     8 hits
  T02 0.0433"     8 hits
  T03 0.0400"   104 hits
  T04 0.0362"    16 hits
  T05 0.0240"    12 hits
  T06 0.0236"    92 hits
  T07 0.0197"   176 hits
  T08 0.1300"     8 hits

Output Files :
   merged.placement.txt
   merged.bottomsilkscreen.ger
   merged.topsilkscreen.ger
   merged.bottomsoldermask.ger
   merged.toplayer.ger
   merged.topsoldermask.ger
   merged.boardoutline.ger
   merged.bottomlayer.ger
   merged.drills.xln
   merged.toollist.drl
-------------------------------------------------------------------------------
```

Figure D.2: Screen of Best Permutation

To output the current best permutation to gerber files. You should see message similar to Figure D.2.

6. To view the merged panels, use

```
make gerbv1 -- for 2 layer boards
```

or,

```
make gerbv2 -- for 4 layer boards
```

to load the current panel into the gerber viewer.

7. Finally, zip up these output files, and upload them to Advanced Circuits freedom checker to make sure there are no potential layout problems. The link to the dfm checker is here:

`http://www.freedfm.com/!freedfmstep1.asp`

D.3 Capacitance Estimation

The capacitance for a microstrip PCB can be estimated [30] [31] as shown in Figure D.3 using Equation D.1, where C is the capacitance in pF per foot, e_r is the relative dielectic constant, w is the width, h is the height, and t is the thickness.

$$C = \frac{7.76 \times 10^{-12}(e_r + 1.41)}{\ln[(5.98h)/(0.8w + t)]} \tag{D.1}$$

For flat ribbon cable, capacitance can be estimated at 46 pF per meter [32].

For the I/O pin of each Tier 1 node, the capacitance is 10 pF per node [20].

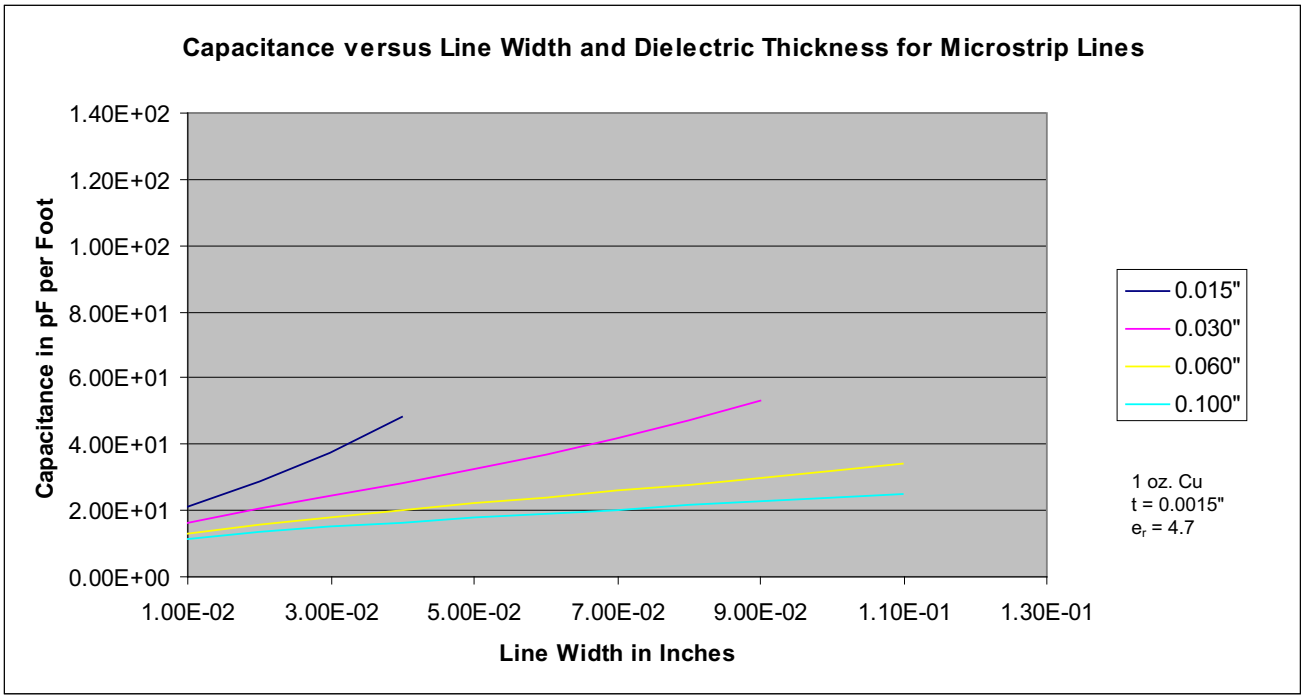

Figure D.3: Microstrip Capacitance of PCB