

An analog and digital data acquisition system for  
Non-Intrusive Load Monitoring

by  
Zachary Alan Clifford

B.S., Massachusetts Institute of Technology (2008)

Submitted to the Department of Electrical Engineering and Computer  
Science

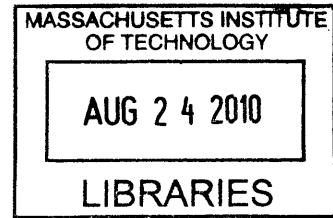
in partial fulfillment of the requirements for the degree of  
Master of Engineering in Electrical Science and Engineering

at the

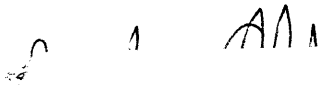
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

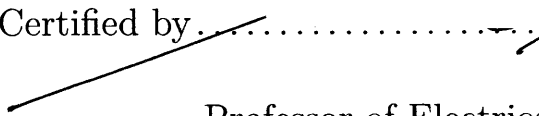
September 2009


© Massachusetts Institute of Technology 2009. All rights reserved.

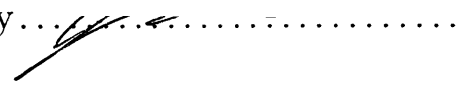


**ARCHIVES**

Author  .....  
Department of Electrical Engineering and Computer Science  
August 31, 2009

Certified by  .....  
Steven B. Leeb  
Professor of Electrical Engineering and Computer Science  
Thesis Supervisor

Certified by  .....  
John Cooley  
Doctoral Candidate  
Thesis Supervisor

Certified by  .....  
James Paris  
Doctoral Candidate  
Thesis Supervisor

Accepted by .....  
Dr. Christopher J. Terman  
Chairman, Department Committee on Graduate Theses



**An analog and digital data acquisition system for  
Non-Intrusive Load Monitoring**

by

Zachary Alan Clifford

Submitted to the Department of Electrical Engineering and Computer Science  
on August 31, 2009, in partial fulfillment of the  
requirements for the degree of  
Master of Engineering in Electrical Science and Engineering

**Abstract**

Non-Intrusive Load Monitoring (NILM) is a method for characterizing and monitoring discrete loads connected to a power distribution system. This can include a ship, a car, or a utility distribution system. The entire concept is predicated on having access to digital samples of the current and voltage signals at the distribution point. This thesis presents an analog to digital converter for this task and a new low-power inductive current sensor for deployment in a standard circuit breaker box. The current sensor uses discrete JFET devices to passively transmit data inductively through the steel door of the circuit breaker.

Thesis Supervisor: Steven B. Leeb

Title: Professor of Electrical Engineering and Computer Science

Thesis Supervisor: John Cooley

Title: Doctoral Candidate

Thesis Supervisor: James Paris

Title: Doctoral Candidate





## Acknowledgments

I would like to thank Professor Leeb for his guidance and support with this work. I would also like to acknowledge and thank Jim Paris, John Cooley, and Al-Thaddeus Avestruz for their oversight and assistance with this work. I also appreciate the invaluable mechanical support provided by Chris Schantz in building the experiments. Finally, I would like to acknowledge my fiancée, Bronwyn Edwards for supporting me in finishing my degree.



# Contents

<b>1</b>	<b>Introduction</b>	<b>17</b>
1.1	Overview . . . . .	17
1.2	NILM Background . . . . .	18
<b>2</b>	<b>NerdJack Analog-to-Digital Frontend</b>	<b>21</b>
2.1	Part selection . . . . .	23
2.2	Software selection . . . . .	24
2.3	Hardware development . . . . .	25
2.4	Device-side application . . . . .	26
2.4.1	Overview . . . . .	26
2.4.2	Interrupts . . . . .	28
2.4.3	Task Overview . . . . .	29
2.4.4	WDTtask . . . . .	29
2.4.5	TCP/IP and Ethernet . . . . .	30
2.4.6	Samplemanager . . . . .	30
2.4.7	Copytask . . . . .	30
2.4.8	DSTRM, CMD, and AUTOD . . . . .	31
2.4.9	Serial . . . . .	31
2.5	PC side application . . . . .	32
2.6	Testing and Results . . . . .	34
2.6.1	Methods . . . . .	34
2.6.2	Results . . . . .	34

<b>3</b>	<b>Inductively powered current sensor</b>	<b>37</b>
3.1	Introduction and Motivation . . . . .	37
3.2	System Overview . . . . .	38
3.3	Breaker Pickup . . . . .	39
3.4	JFET Mixer . . . . .	45
3.5	Through-door Inductive Link . . . . .	49
3.6	Sense and Demodulation Circuit . . . . .	54
3.6.1	Power front-end . . . . .	54
3.6.2	Analog filter chain . . . . .	55
3.6.3	DSP operation . . . . .	56
3.6.4	I/Q Demodulation Overview . . . . .	56
3.7	Test setup and results . . . . .	59
3.7.1	Coil design procedure . . . . .	63
3.7.2	Results . . . . .	64
3.8	Future Work . . . . .	70
3.8.1	Inductive Link Improvements . . . . .	70
3.8.2	Demodulation Board Improvements . . . . .	71
3.8.3	DSP Software Improvements . . . . .	74
<b>4</b>	<b>Conclusions</b>	<b>75</b>
<b>A</b>	<b>Microcontroller-based educational tool</b>	<b>77</b>
A.1	BurnIt theory of operation . . . . .	77
A.2	Programming the AT89C2051 . . . . .	79
A.3	Programming the PIC16F628 . . . . .	79
A.4	Programming the GAL22V10 . . . . .	79
<b>B</b>	<b>Data Acquisition Device Manual</b>	<b>81</b>
B.1	Theory of Operation . . . . .	81
B.2	Installing software to use a NerdJack . . . . .	81
B.2.1	Windows . . . . .	82

B.2.2	Mac OS X and Linux . . . . .	82
B.3	Using the NerdJack . . . . .	83
B.4	Installing software to program a NerdJack . . . . .	83
B.4.1	Windows . . . . .	84
B.4.2	Mac OS X and Linux . . . . .	84
B.5	Programming a NerdJack . . . . .	85
B.6	Building a NerdJack . . . . .	86
B.7	Pinouts for NerdJack . . . . .	86
B.8	Device Overview . . . . .	87
B.9	Updating the Firmware . . . . .	89
B.10	Building the development environment . . . . .	90
B.11	Remaking the Windows installer . . . . .	90
B.12	Known Issues . . . . .	91
B.13	Customizations to the stock Framework . . . . .	92
B.14	Software Overview . . . . .	93
B.14.1	FreeRTOS . . . . .	93
B.14.2	lwIP . . . . .	93
B.14.3	General Program Structure . . . . .	93
<b>C</b>	<b>NerdJack Analog-to-Digital Converter Schematics and Layout</b>	<b>97</b>
C.1	Schematic . . . . .	97
C.2	Layout . . . . .	104
<b>D</b>	<b>BurnIt Schematics and Layout</b>	<b>109</b>
D.1	Schematic . . . . .	109
D.2	Layout . . . . .	111
<b>E</b>	<b>IQ Demodulator Schematics and Layout</b>	<b>115</b>
E.1	Schematic . . . . .	115
E.2	Layout . . . . .	119

<b>F</b>	<b>NerdJack Source Code Listing</b>	<b>127</b>
F.1	Firmware Source Code . . . . .	127
F.1.1	FreeRTOSConfig.h . . . . .	127
F.1.2	conf_eth.h . . . . .	129
F.1.3	conf_lwip_threads.h . . . . .	132
F.1.4	externalmem.h . . . . .	134
F.1.5	lwipopts.h . . . . .	134
F.1.6	DataStream.h . . . . .	142
F.1.7	DataStream.c . . . . .	144
F.1.8	InitBoard.h . . . . .	158
F.1.9	InitBoard.c . . . . .	159
F.1.10	ethernet.h . . . . .	173
F.1.11	ethernet.c . . . . .	175
F.1.12	samplemanager.h . . . . .	180
F.1.13	samplemanager.c . . . . .	180
F.1.14	serialport.h . . . . .	182
F.1.15	serialport.c . . . . .	183
F.1.16	wdtreset.h . . . . .	187
F.1.17	wdtreset.c . . . . .	187
F.1.18	main.c . . . . .	190
F.1.19	version.h . . . . .	194
F.2	Ethstream Source . . . . .	194
F.2.1	ethstream.h . . . . .	194
F.2.2	ethstream.c . . . . .	195
F.2.3	nerdjack.h . . . . .	209
F.2.4	nerdjack.c . . . . .	210
F.3	Nerdconfig Source . . . . .	224
F.3.1	configData.py . . . . .	225
F.3.2	nerdconfig.py . . . . .	233

<b>G</b>	<b>BurnIt Source Code Listing</b>	<b>241</b>
G.1	ATMEGA Firmware . . . . .	241
G.1.1	2051.h . . . . .	241
G.1.2	2051.c . . . . .	242
G.1.3	avrutils.h . . . . .	247
G.1.4	avrutils.c . . . . .	248
G.1.5	burnitall.c . . . . .	249
G.1.6	gal.h . . . . .	258
G.1.7	gal.c . . . . .	260
G.1.8	pic.h . . . . .	275
G.1.9	pic.c . . . . .	277
<b>H</b>	<b>IQ Demodulator DSP Source Code Listing</b>	<b>291</b>
H.1	Programming the IQ Demodulator DSP . . . . .	291
H.2	DSP Firmware . . . . .	291
H.2.1	main.c . . . . .	291
H.2.2	adcDrv2.h . . . . .	297
H.2.3	adcDrv2.c . . . . .	298
H.2.4	funcs.h . . . . .	304
H.2.5	funcs.c . . . . .	305
H.2.6	i2cdac.h . . . . .	308
H.2.7	i2cdac.c . . . . .	310
H.2.8	ocmodules.h . . . . .	318
H.2.9	ocmodules.c . . . . .	320
H.2.10	traps.c . . . . .	322





# List of Figures

2-1	Block diagram for NerdJack . . . . .	23
2-2	Top surface of data acquisition device . . . . .	26
2-3	Bottom surface of data acquisition device . . . . .	27
3-1	Current Sensor System Overview . . . . .	38
3-2	Maxwell 3D model of breaker . . . . .	39
3-3	FEMM Breaker Pickup . . . . .	41
3-4	Breaker pickup model . . . . .	41
3-5	Breaker pickup photograph. . . . .	44
3-6	JFET Modulator Circuit . . . . .	45
3-7	JFET Mixer small signal model . . . . .	48
3-8	Reluctance model of through door transmission . . . . .	50
3-9	Top view of transmission coil configuration . . . . .	51
3-10	Through Door Link Transformer Model . . . . .	52
3-11	Analog filter block diagram . . . . .	55
3-12	Experimental setup photo . . . . .	59
3-13	Open door photo . . . . .	60
3-14	Demodulation board photo . . . . .	62
3-15	Carrier frequency compared to secondary coil resonance . . . . .	64
3-16	60 and 180 Hz 5 A Results . . . . .	66
3-17	Low Current Experimental Results . . . . .	67
3-18	70 Hz Experimental Results . . . . .	68
3-19	Noise floors . . . . .	69

A-1	BurnIt block diagram . . . . .	78
C-1	The main microprocessor . . . . .	98
C-2	Ethernet PHY . . . . .	99
C-3	External connectors . . . . .	100
C-4	Memory, Power and USB . . . . .	101
C-5	First ADC . . . . .	102
C-6	Second ADC . . . . .	103
C-7	Top copper layer without ground plane filled . . . . .	105
C-8	Bottom copper layer without ground plane filled . . . . .	106
C-9	Top silk layer . . . . .	107
C-10	Bottom silk layer . . . . .	108
D-1	BurnIt Schematic . . . . .	110
D-2	BurnIt Top Copper . . . . .	112
D-3	BurnIt Bottom Copper . . . . .	113
D-4	BurnIt Silkscreen . . . . .	114
E-1	The analog filter stages . . . . .	116
E-2	DSP and supporting hardware . . . . .	117
E-3	Power, modulation generators, and 60 Hz notch filter (currently not populated) . . . . .	118
E-4	Top copper layer . . . . .	120
E-5	Bottom copper layer . . . . .	121
E-6	Top silk layer . . . . .	122
E-7	Bottom silk layer . . . . .	123
E-8	Copper layer 2 with ground plane not filled . . . . .	124
E-9	Copper layer 3 with power planes not filled . . . . .	125

# List of Tables

2.1	Priority levels for NerdJack . . . . .	29
B.1	Command line arguments to Ethstream . . . . .	84
B.2	DB15 table pinout . . . . .	87
B.3	DB37 connector pinout . . . . .	88



# Chapter 1

## Introduction

### 1.1 Overview

In many industrial and home applications it is useful to monitor an electrical system for both faults and for energy consumption. One common approach involves attaching specialized instrumentation to each device to be monitored. However, previous work with Non-Intrusive Load Monitoring (NILM) has shown that this problem can be addressed much more simply by adding instrumentation to the power distribution system rather than each device. A NILM system identifies and monitors individual loads by measuring the frequency content of transient events in the power distribution system from a centralized location. Work with this technology was demonstrated in [13, 3, 5] for shipboard systems. Further experiments with this technology were done in [7, 11, 6, 14, 9, 5, 3, 10, 2, 8, 12].

The NILM concept is predicated on having access to digital samples of the voltage and current waveforms at a power distribution center of the system to be monitored. Chapter 2 presents an analog to digital conversion front-end to be used with NILM. This device, the “NerdJack”, takes as input properly conditioned voltage signals and outputs the digitized version of these over Ethernet to a personal computer for processing. This device is meant to be installed in a custom analog front-end to NILM.

The current signal for the NILM system is typically measured using a magnetic

field sensor wrapped around the utility feed for the subsystem to be monitored. The previously mentioned converter is meant to interface with such a device. However, such a sensor may be impractical for some retrofit applications especially in the home where skilled labor would be required to separate Line and Neutral. This would be required to deploy a wrap-around magnetic field sensor because such a sensor would measure no net current if Line and Neutral were not separated. The sensor presented in Chapter 3 is an alternative to the wrap-around magnetic field sensor. It measures the current in the utility feed by sensing the resulting magnetic field at the face of the main circuit breaker in a standard breaker panel, where the Line and Neutral are already separated. A major challenge that is overcome by the system presented here is that of communication through the steel breaker panel door, which must be closed to comply with safety regulations.

Work with the microcontroller used in the analog to digital front-end led to “BurnIt” presented in Appendix A. MIT’s Microcontroller Laboratory class educates students on the development and usage of microcontroller-based digital systems. Many microcontrollers can be purchased cheaply or sampled for free for use in personal projects, but the programming tools for these devices are usually relatively expensive. BurnIt is an inexpensive multiprogrammer created using publicly available algorithms. It is designed to be assembled by the students as part of the class and used both during the class for lab work and after the class for personal projects. The design and overall functionality of BurnIt will be detailed in Appendix A.

## 1.2 NILM Background

The NILM system is generic and can be used to monitor a variety of systems, including a home, a ship, a car, or any other system with a power bus and multiple connected devices. This monitoring is done by examining transients on the electrical line. Previous work has established a collection of “fingerprints” for a variety of devices, and training the system with new loads is straightforward. NILM explores transients in the frequency domain using spectral analysis to determine the strength

of the various harmonics of the 60 Hz power delivery fundamental. It also uses phase relationships to determine whether a load is predominantly capacitive, inductive, or resistive. It can also understand more complicated loads, such as desktop computers. With access to the aggregate current and voltage signals, the contribution of each different load can be separated. This method of diagnostics requires only one sensor to monitor the health of multiple loads rather than requiring multiple specialized sensors at each load.

Early experiments using NILM required a system power down and skilled labor to install the current and voltage sensors. The methods presented in this thesis will simplify NILM installation and make it more applicable in a wide variety of systems.





## Chapter 2

# NerdJack Analog-to-Digital Frontend

The NILM requires a relatively inexpensive data acquisition system that can provide high quality samples of a voltage or current signal. The current prototypes require up to six analog to digital converters to sample the voltage and current waveforms of a three phase power system. Fewer channels are required for a home split-phase system. The system monitors a 60 Hz fundamental power delivery waveform and its odd harmonics to perform its analysis. In order to capture this information, the data acquisition system needs to sample its data at approximately 6 kHz.

Previous NILM prototypes used a commercial product, the LabJack UE9, to accomplish this task. The LabJack is a printed circuit board meant to be installed into another system through an edge connector running along its edge. The device carries a large number of analog to digital converters, digital input/output, and other peripherals. It is meant to communicate with a host PC via Ethernet for configuration of its peripherals and for delivery of the sampled data. This device served the needs of a prototype NILM system, but a device optimized to NILM's needs would deliver better results and be cheaper to deploy.

The new device presented here, the “NerdJack”, replaces the LabJack in the NILM prototypes. This device had numerous functional requirements to meet to ensure that it was suitable for replacing the LabJack. The most important requirement was that

it be compatible with the LabJack so that it could fit in the current NILM prototypes with little to no modification to either the software or the hardware. With that need met, the device needed to provide either either superior analog to digital conversion capabilities, lower cost or both.

Defining superior analog-to-digital conversion capabilities helped narrow the device choices. Sixteen-bit analog-to-digital converters were required to help NILM see small transient events. Simultaneous sampling would also simplify the NILM algorithms and post processing. Additional channels would allow NILM system expansion and permit it to monitor more devices or have different filter front-ends. The new device needed at least 8 channels. Finally a sampling rate of at least 8 kHz was desired to determine if any additional higher frequency information was beneficial.

The current NILM prototypes are meant to be installed once and forgotten until they sound an alarm. Therefore, the system as a whole needed to be highly reliable in the event of network or hardware failure. The device needed to survive network disconnections of up to 30 seconds without losing data. This requirement demanded large on-board memory buffers to store data until the network could be restored.

In addition to the basic functional requirements there were substantial practical limitations on the new device. First, the components should all be hand solderable. This would simplify prototyping and lead to less expensive production costs in the future. The device should fit on a two-layer printed circuit board. In addition to being cheaper than a four layer board, it would allow integration of the functionality of the design into other two-layer boards without requiring a connection interface or rerouting the design. This would permit future iterations of NILM to include the NerdJack functionality without requiring a plug-in card.

The tools and supporting software should be free or inexpensive. This lowers the cost of the device and limits licensing problems in the future. Using software licensed under the GNU Public License (GPL) or similar allows the use of many libraries and support software to speed development. The GPL requires that the source code to software using it be freely available. The GPL source code distribution requirements do not pose a problem to this device because it is academic in nature.

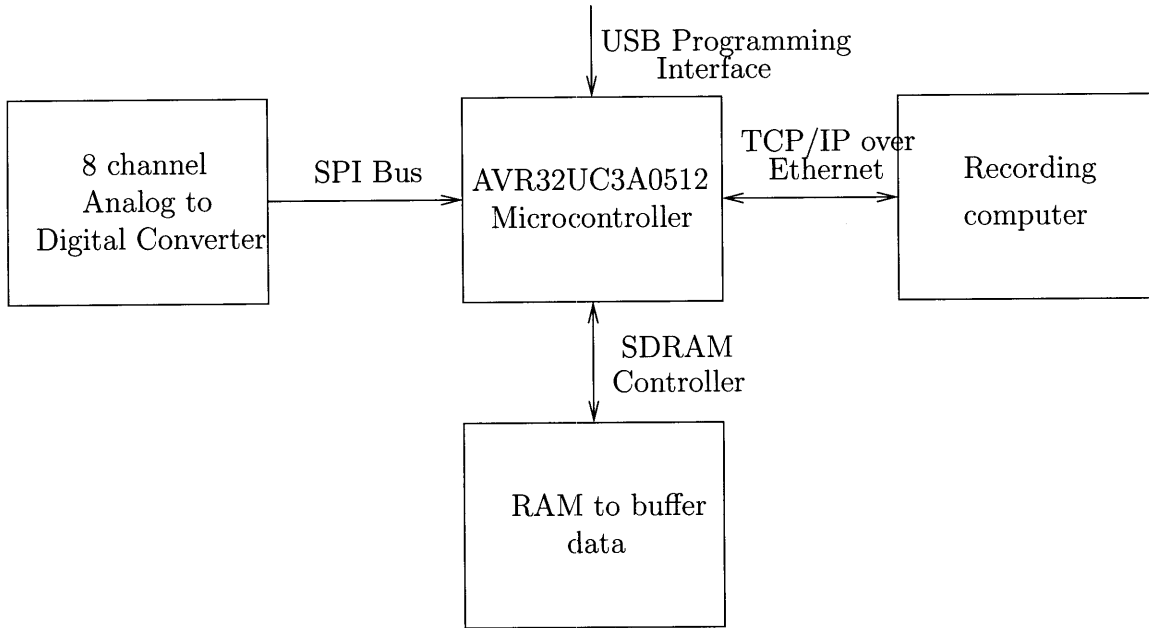


Figure 2-1: Block diagram for NerdJack

The block diagram of the device meeting these functional requirements is given in Figure 2-1. The basic flow of information follows the marked arrows. The analog-to-digital converter is connected to the NILM system to be measured. This data is accepted by the microcontroller and buffered in external RAM. As the network becomes available, the data is streamed to a waiting personal computer or custom hardware for processing. The firmware on the microcontroller should be easily installed using a USB cable for easy deployment.

## 2.1 Part selection

The aforementioned requirements led to fairly limited choices for the new device. The device needed to be a microcontroller-based single board computer system, so the first step was to pick the microcontroller. Ideal microcontrollers for this application would have peripherals to interface with external memory as well as communicate over Ethernet. This led to some of the high-end ARM processors and some of the Freescale ColdFire line of processors. Unfortunately, the devices that had both features tended

to have large pin counts in BGA packaging, making them unsuitable for this project.

The final choice was the new Atmel AVR32 line of processors. These devices were in pre-release at the time, but they had the needed features in a flat package that could be soldered by hand. These chips could interface with external memory and had an Ethernet MAC module in addition to the standard microcontroller communication features. Best of all, the free GNU C Compiler Collection tools could be used for code development.

The next step was to select an analog-to-digital converter. The AVR32 microcontroller has an on-board analog-to-digital converter, but it is of very limited resolution. This led to the selection of an external converter with a digital interface to the microcontroller. This converter needed to be able to capture channels at 16 bits of accuracy, with either unipolar or bipolar inputs. The Analog Devices AD7656 was chosen because it could sample six channels on a single chip and communicate with a microcontroller using a standard SPI bus. A serial interface was important because the external data bus of the AVR32 used most of its input/output pins. A parallel interface would have used too many microcontroller pins. In addition, the AD7656 allows multiple AD7656 chips to be connected in a daisy-chain configuration. With this selection two chips would provide twelve channels of sampling using only one SPI port on the microcontroller. This solution would still give the needed sampling rate because the SPI bus can operate at a high clock speed.

## 2.2 Software selection

The device firmware must manage a variety of different tasks, including operating the network, communicating with the analog-to-digital converters, and buffering the data into SDRAM. This led to the selection of a real time operating system and a networking stack.

After examining a few of the operating systems already ported to AVR32, FreeRTOS version 5.0.4 was the best match for the device requirements. It is free, lightweight, and much simpler than Linux. As configured, FreeRTOS manages a group of paral-

lel prioritized tasks. Much like a full-featured operating system, it ensures that the highest priority ready task is running at all times. It also manages primitives like mailboxes and semaphores to allow inter-task communication and cooperation.

The other major library selection is lwIP, the Lightweight Internet Protocol stack version 1.3.0. This library provides a simple abstraction to the Ethernet interface using TCP/IP, the standard protocol of the Internet. It implements enough of the TCP/IP protocol to interface with standard network hardware without requiring too much processor overhead.

## 2.3 Hardware development

The EVK1100 evaluation kit from Atmel was used as a starting point for the design. This small board interfaced SDRAM and Ethernet to the microcontroller as well as many other peripherals. The AD7656 was then connected to this core module to begin testing and software development. The final design used the basic EVK1100 schematic with unnecessary peripherals removed and analog-to-digital converters added. The design was then rerouted on a two layer board with the same pin-out interface as the LabJack.

CadSoft's EAGLE layout editor was used to create the custom printed circuit board using the EVK1100 bill of materials and provided schematic as a starting point. Careful attention to the power supply decoupling and analog front-end were required for proper functionality. Additionally, many of the data and address lines from the SDRAM had to be properly separated from the clock signals on the board.

The final board is pictured in Figures 2-2 and 2-3, while the final schematics are included in Appendix C. The analog-to-digital converters are connected to both the screwtab terminals and the DB connectors along the top edge. The Ethernet and USB connections for data acquisition and firmware programming respectively lie opposite to the screwtab terminals. The DIP switches set network parameters for the device, and the LEDs report its status. The device has power regulators that accept bipolar +/- 12 Volt, common, and +5 Volt power supplies. These regulated supplies are

delivered to the DB connectors to help expand the device. The DB connectors also hold some digital input/output pins and a UART serial port.

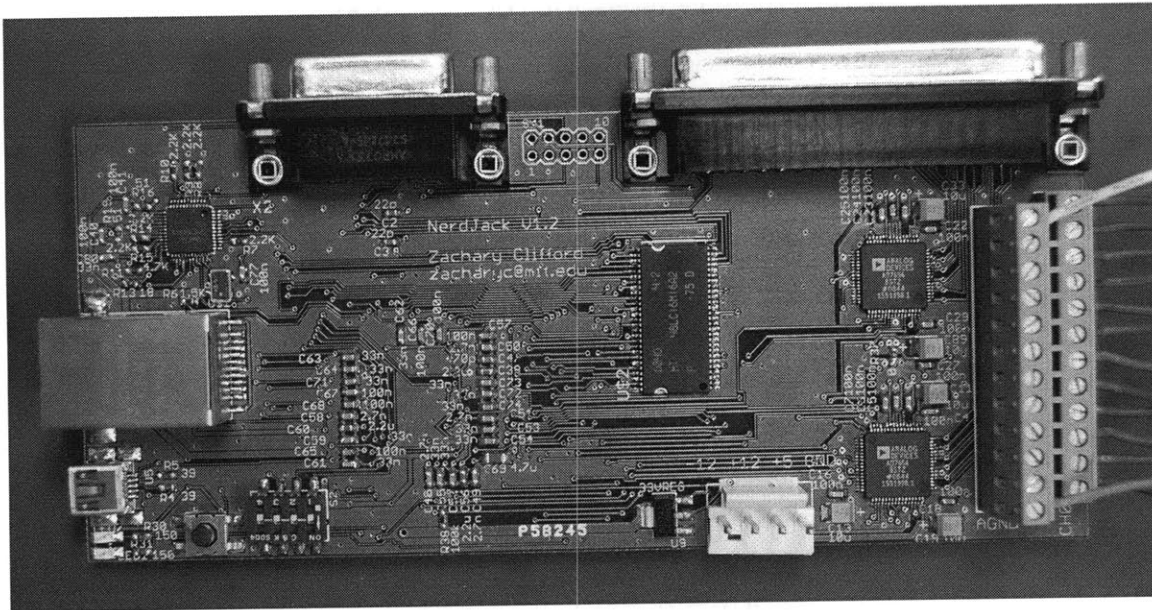


Figure 2-2: Top surface of data acquisition device

## 2.4 Device-side application

### 2.4.1 Overview

Most of the complexity of the device is in its onboard firmware. The microcontroller is programmed via USB with custom firmware once the device is built. The firmware application on the device is organized into a collection of interacting tasks that coordinate the sampling and transmission of data from the analog-to-digital converters. All of these tasks are coordinated by FreeRTOS. Each one is a non-terminating routine with a private stack and priority level from zero to seven. Tasks can ready each other or be readied by interrupt service routines tied to external events. FreeRTOS ensures the the highest priority ready task is executing. If multiple tasks of the same priority are ready, it switches between them in a round-robin style.

Both FreeRTOS and lwIP are included in Atmel's software framework for the

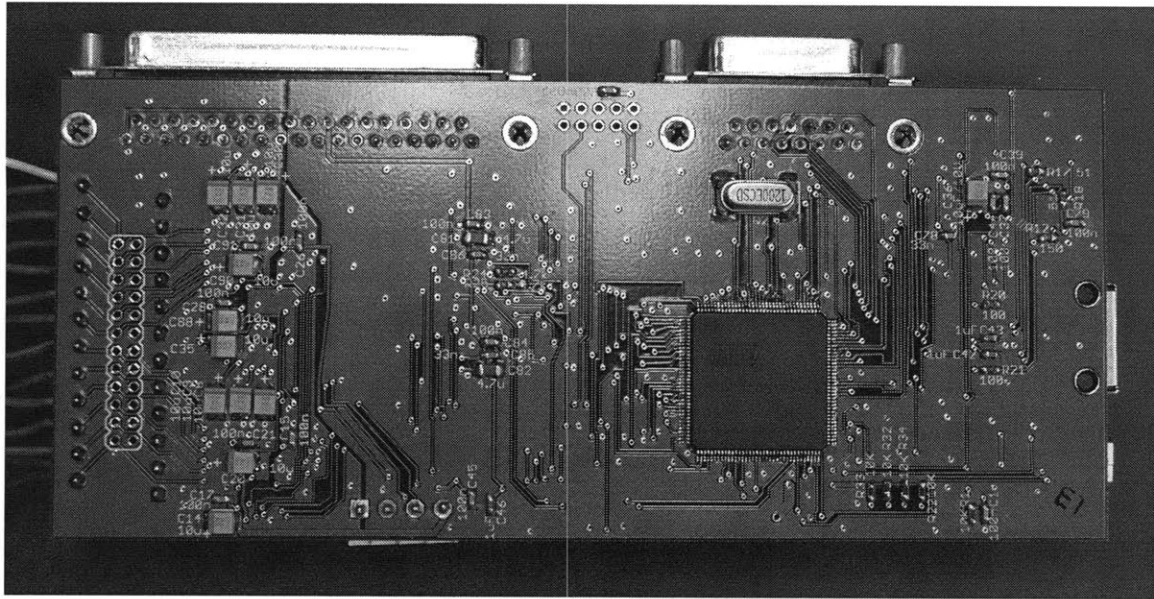


Figure 2-3: Bottom surface of data acquisition device

AVR32. The basic web server demo served as a baseline for development. The firmware has the same basic structure as the framework, but a new version of FreeRTOS version 5.0.4 was added, and numerous changes were made to the driver level to fix bugs and add needed features. The framework drivers and operating system configure the resources of the device and provide an API for interacting with it. After setup, the framework hands control of the device to the application.

After powering the device on, the application initializes the hardware resources needed for interacting with the ADCs and the Ethernet module and waits for sampling commands from the PC host. A sampling command tells the device which channels to sample, the sampling rate, and the range required. This allows the device to trade dynamic range for accuracy if desired. Sampling continues until the PC requests a sampling stop or the device depletes its buffer space. Data is delivered to the PC in real-time, so buffering is only necessary in the event of network congestion. Software mechanisms are in place to resume interrupted transfers without losing samples using both the network stack and the features of FreeRTOS.

## 2.4.2 Interrupts

The scheme for sampling the analog-to-digital converters is a complex interaction of a few different interrupt service routines. The analog-to-digital converters are connected to a pulse width modulation generator on the AVR32. This is set to produce a sampling pulse at the chosen sampling frequency. Because of the daisy-chain nature of the ADCs, all channels less than the highest desired channel must be sampled. This sampling pulse causes a BUSY line to become active on the ADCs for the duration of the sampling. When this line falls, it signals an interrupt handler on the AVR32 that data is ready for reading.

This interrupt handler loads one of the Direct Memory Access (DMA) modules of the microcontroller. This module simply requires a source peripheral, a destination memory address, and the number of bytes to copy. Without CPU interaction it will copy this information and can use an interrupt to signal completion. The DMA module in this case transfers dummy data to the ADCs. Because of the nature of an SPI bus, the “slave” (the ADC) cannot initiate a transfer. When the “master” writes data to the slave, the slave will simultaneously send data back to the master. Therefore, the master must write dummy data to the slave to read its data. This DMA module performs this task every time the ADCs indicate that they have data ready through the BUSY line. This handler is of the highest supported priority because it is imperative that reading data begin as soon as data is ready. If there is delay, the data might not be read before the next sampling pulse arrives. In that event, the remaining unread data would be lost. Any unpredictability in microcontroller response time would limit the speed of the device.

There is another interrupt handler tied to a different DMA module that reads information from the ADCs. The aforementioned module writes data to the ADCs. This one reads large amounts of data from the ADCs into onboard SRAM in the AVR32. When it completes a full packet, this interrupt handler reloads the DMA module and signals the application that a packet is ready. Other tasks in the application then copy the packet into external SDRAM for buffering and eventual transmission.



<b>Task Name</b>	<b>Priority</b>
WDTtask	7
TCP/IP	6
Ethernet	6
samplemanager	5
copytask	4
DSTRM	3
CMD	2
AUTOD	1
Serial	1

Table 2.1: Priority levels for NerdJack

The interrupt service routines are very fast and cannot tightly interact with other parts of the application. For the second DMA interrupt a custom semaphore implementation was required for speed. Invoking the FreeRTOS semaphore abstraction led to very slow preemptive task switching. The “light” semaphore implementation involves a shared global variable between an application task and the interrupt service routine. Interrupts are disabled before accessing the variable to guard against race conditions.

### 2.4.3 Task Overview

FreeRTOS manages a collection of tasks that cooperate to implement the application. The task priority levels are numbered 0 through 7 with higher numbers having higher priority. The tasks in priority order are given in Table 2.1. Each one has a private stack and a specific purpose in the application. An overview of the various cooperating tasks should clarify application operation.

### 2.4.4 WDTtask

The WDTtask manages the hardware Watchdog timer (WDT). The watchdog in the AVR32 is a simple timer that will reset the microcontroller if the timer is allowed to expire. This serves as a failsafe that should reset the device in the event of a software failure without requiring manual power cycling. At regular intervals the other tasks

in the application send messages to check in with the watchdog task. The watchdog task resets the hardware watchdog timer when all other threads it monitors have checked in. Should all threads not check in for the allotted time, the hardware timer will reset the device.

### **2.4.5 TCP/IP and Ethernet**

These are necessary for proper functioning of the Ethernet interface and lwIP TCP/IP stack. The Ethernet monitoring task simply waits to be readied by an interrupt tied to the Ethernet peripheral “receive” event. It then reads the incoming packet and passes it upwards to the lwIP stack. The TCP/IP task is responsible for sending and receiving TCP/IP packets on the Ethernet interface. It handles retransmission of lost packets and all necessary memory management. When the application calls an lwIP API function, this routine takes over the processing of that request.

### **2.4.6 Samplemanager**

This coordinates sampling of the ADC channels. When lower levels of the application wish to start sampling with certain parameters or stop it, a message is sent to this task. This ensures that race conditions do not develop with multiple parts of the application trying to alter the ADC parameters simultaneously. Because this task is the only one able to alter the converters, its internal state variables actually reflect the sampling parameters. This was necessary because sampling could be altered by both user requests and network failure. A single interface was required to prevent race conditions between those two command sources.

### **2.4.7 Copytask**

Copytask was necessary to solve latency problems with external memory. The application is supposed to sample data into external SDRAM and then transmit it to a waiting PC for analysis. Most of this copying from the ADCs is performed using on-chip DMA hardware described above. The copying task was necessary because

the on-chip DMA hardware has no buffer and data is sampled simultaneously. This means that all channels of the ADC hardware become ready in bursts, but the DMA hardware is not designed for burst operation. If multiple channels are selected, a new 16-bit word of data arrives every 16 clock cycles. The SDRAM latency is long because it is engineered for burst operation, but it becomes unpredictable when the Ethernet task is taken into account. In the worst case scenario the TCP/IP stack is actively reading past data from the SDRAM while the DMA controller is trying to write a burst of data into a different part of SDRAM. The lack of any sort of buffering in the DMA hardware leads to data loss as these two modules contend for access to the SDRAM. This problem was solved by using on-chip SRAM as a buffer for the DMA hardware. The copy task is responsible for moving data from this internal buffer into external SDRAM. Although this scheme requires more total data movement, this relieves congestion because this copying can take place between data bursts from the ADC hardware without tight real-time constraints.

#### **2.4.8 DSTRM, CMD, and AUTOD**

These three tasks are the network ones that listen on three different ports. The Command task listens on TCP and accepts single commands to start, stop, and resume sampling before closing the connection. The Autodetection task listens on a UDP port and simply replies to datagrams sent on that port. This allows a PC to use UDP broadcasts to detect the IP address of the device. It can look for replies and then use the source address of the reply to make this determination. The final task is the Datastream task. It waits for a TCP connection and then sends data through that connection as data becomes available.

#### **2.4.9 Serial**

Serial uses the USART serial port on the device. This task listens on the standard Telnet port and simply echoes characters received on that port to the serial port. Any received characters are similarly echoed to a connected PC. A small amount of

buffer space keeps the device running smoothly. This was added to make interfacing the system to other digital devices more straightforward.

The lowest priority task is the FreeRTOS Idle task. It performs required maintenance routines for the operating system kernel, but it is otherwise not important to the functioning of the application.

## 2.5 PC side application

Utility software for programming the device and reading data from it were developed to interface with the device. Both of these were developed in parallel with the hardware to make a full system ready for deployment outside of the laboratory. The device programmer is a mixture of C and Python, while the data reading program is purely C. Both were written to be as portable as possible between different PC operating systems and potentially other embedded systems.

The programming software utilizes fully open source software to perform this task. The device is equipped with a mini-USB port that allows it to be programmed using commodity USB cables. AVR32 devices are shipped pre-loaded with a bootloader that permits in system programming via USB. When the right conditions are met (in this case a switch on the device is flipped at bootup), the application is not loaded. Instead, the USB port is activated and ready to communicate using an Atmel variant of the USB Device Firmware Update protocol. Part of this project involved contributions to an open-source C utility called “dfu-programmer” to help implement this protocol in a cross-platform way. Extending the existing programming utility was much more straightforward than starting over. This utility is better than the Atmel-provided batchisp program because it can run on Windows, Macintosh, and other Unix derivatives like Linux. At its core it uses an open source program called libUSB. This library permits communication with the PC USB port with a very simple interface that is portable between different operating systems. Because of this dependence, dfu-programmer can operate on any system that provides a libusb-like interface. This utility takes as input AVR32 firmware and burns it to the AVR32.

It can also read the firmware image and interpret the settings programmed into the device.

Around this utility is a simple Python command line package called “nerdconfig”. This package exposes a command-line interface to dfu-programmer that makes programming simple for the end-user. This utility burns stock firmware onto the device and programs it with a unique Ethernet address and serial number. Finally, it configures the IP address the device will assume after initialization. It can determine the firmware version running on the device and upgrade if necessary.

The serial number, MAC address, and IP settings are recorded to a special section of memory in the device known as the “User Page”. The User Page is a section of Flash 512 bytes in size that is not erased by the Flash Erase command on the AVR32. This makes it suitable for configuration and serial number data.

The other PC software component is “ethstream”. This command line program is a modification of an earlier “ljstream” written by Jim Paris for interfacing with the LabJack. Ethstream interfaces with both devices so that both old and new NILM hardware will work with the same PC-side software. This program accepts as input the channels to sample and the rate requested. It then outputs space separated samples to STDOUT. This allows the output to be easily piped to other programs for processing or redirected to a file for storage. It has other modes for testing the device, detecting its IP address, and determining its firmware revision. This software can also operate on Windows, Macintosh, and other Unix derivatives like Linux.

A manual for the entire system was written and is attached as Appendix B. This details the usage and construction of the device. The source code to both PC-side applications and the device firmware is attached as Appendix F.

## 2.6 Testing and Results

### 2.6.1 Methods

Testing was an important part of the development process that helped ensure that the device met specifications.

The intended use of the device places it in a difficult place to physically access. Software crashes might not be noticed for some time, and manually resetting the device might not be possible. The device needed to be able to successfully start acquisition every time it was asked to do so. In any normal failure condition it needed to reset without human intervention.

To evaluate it a test fixture was created that started ten thousand acquisitions with random sampling parameters. The device was connected to a resistor ladder to assert known voltages on each channel. Easy automated tests programmatically confirmed that the proper channels were being sampled for the correct amount of time.

Another test involved running the device for a full day with a sine wave on one channel and the resistor ladder on the others. The sine wave was programmatically tested to ensure that points were not lost or mixed up in any way with the known voltages asserted by the resistor ladder.

The device sampled at a variety of speeds under different network conditions to evaluate its actual sampling rate limitations.

The final test involved using a Matlab code called ADCTest to evaluate the accuracy of the device. This code takes a few periods of a perfect sine wave captured by the device and analyzes it. It computes the effective bit accuracy of the device by measuring how closely it follows the reference sine wave.

### 2.6.2 Results

The device passed both functional tests with the latest revision of firmware by starting every time in capturing the correct data. It did not crash or mix up channels in

multiple 24 hour tests. It also kept accurate time during that period.

The effective accuracy of the device appears to be 11 bits according to the previously mentioned Matlab package, but there is reason to believe that it is better than that. A Wavetek Function Generator Model 182A calibrated on 8/21/92 generated the reference sine wave. It is unclear whether the observed bit rate is limited by the resolution of the sensing device or the signal generator. The apparent accuracy changes with the amplitude and frequency of the sine wave. However, the measured accuracy in this test was superior to the LabJack used as a reference. Because the board was designed to the specifications of the ADC manufacturer, it should match the performance specified by Analog Devices. A new signal generator was not purchased for testing because the new device is already demonstrably better than the reference device. The new device passes this test.

The sampling rate of the device is constrained by the SPI bus of the microcontroller connection to the analog-to-digital converters. The network had to be highly congested for it to become a performance bottleneck. As such, the sampling speed is determined by how much data needs to be copied from the analog-to-digital converters. Data is read out from the converters serially starting with channel zero. Since each 16 bit sample requires sixteen more periods of the 18 MHz SPI clock, every channel sampled increases the amount of time required to empty the converter of data. There is no mechanism for sampling high numbered channels without first reading out lower channels, so the constraint applies to the highest channel sampled. These limitations are built in to the PC client to NerdJack to provide warnings to the user about potential data corruption. These tests showed that the NerdJack could sample 8 channels at 8 kHz comfortably and can sample much faster with fewer channels.





# Chapter 3

## Inductively powered current sensor

### 3.1 Introduction and Motivation

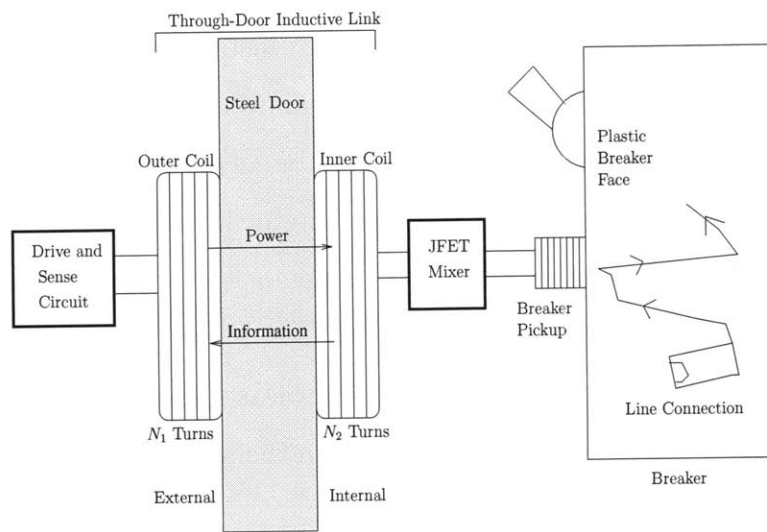
For NILM to be deployable in wider settings, it would need to be installed without the help of a trained electrician, and it would need to attach to standard equipment. Electrical current is difficult to probe because any sensor needs to be located “upstream” of any loads to be measured. Directly measuring the incoming utility connection is difficult because the connection is inaccessible or at least non-standard among homes or other systems. Additionally, utility connection bundles usually consist of both the power and return path connections. This makes inductively measuring the current difficult because the net current flowing in the bundle is zero in the absence of energy storage in the home or ground faults.

A small sensor described in this chapter placed inside a standard circuit breaker panel allows for this measurement. This configuration is advantageous because a circuit breaker panel and circuit breaker are both very standard devices that are all constructed similarly. Each breaker also carries line current without the neutral connection nearby. The sensor consists of two parts both magnetically attached to the steel door of the breaker panel hereafter referred to as the “inner” component and the “outer” component. The inner component rests up against the main breaker in the circuit breaker box to inductively sense the current flowing in the breaker. This device then modulates the sensed waveform onto a carrier waveform via a novel low-

power JFET mixer. This data is then transmitted through the door to be received by the external part of the sensor. The outer component inductively powers the internal device and receives signal from it. All of this is accomplished without drilling any holes in the steel door.

### 3.2 System Overview

The sensor shown in Figure 3-1(a) consists of three main parts: an inductive pickup for sensing current from the breaker face (Breaker Pickup), an inductive link designed to transmit power through the steel breaker panel door (Through-door Inductive Link), and a balanced JFET modulator circuit for transmitting information through that inductive link (JFET Mixer).



(a) Sensor block diagram



(b) Circuit Breaker Cross-section

Figure 3-1: The current sensor measures magnetic fields at the face of the circuit breaker and modulates a high frequency carrier signal to transmit that information through the panel door.

The outer coil in Figure 3-1(a) is driven with a high-frequency sinusoidal carrier voltage. That voltage couples to the inner coil through the inductive link in Figure 3-1(a) and drives the JFET mixer. The JFET mixer controls the amount of current drawn from the inner coil according to the low-frequency (60 Hz) current signal mea-

sured by the breaker pickup. The result is a modulation between the high frequency carrier signal and the low-frequency (60 Hz) signal measured at the breaker face. The external sense circuit in Figure 3-1(a) monitors the current drawn through the inductive link to extract the resulting modulated signal. The internal device is fully powered by the applied carrier, and the entire system works without modification to the breaker panel or the circuit breaker itself. With the modulated signal available to the sense circuit external to the door, the current through the main breaker can be analyzed with the NILM system described above for load identification and power monitoring [13, 3, 5, 7, 11, 6, 14, 9, 10, 2, 12]. The sense circuit consists of both a power supply stage to drive the coils and an I/Q demodulation stage to recover the measured current.

### 3.3 Breaker Pickup

The current path inside a typical circuit breaker passes by the lower face of the circuit breaker as illustrated by Figures 3-1(a) and 3-1(b). Therefore the breaker pickup was designed to focus and measure the magnetic field resulting from current flowing inside the lower breaker face.

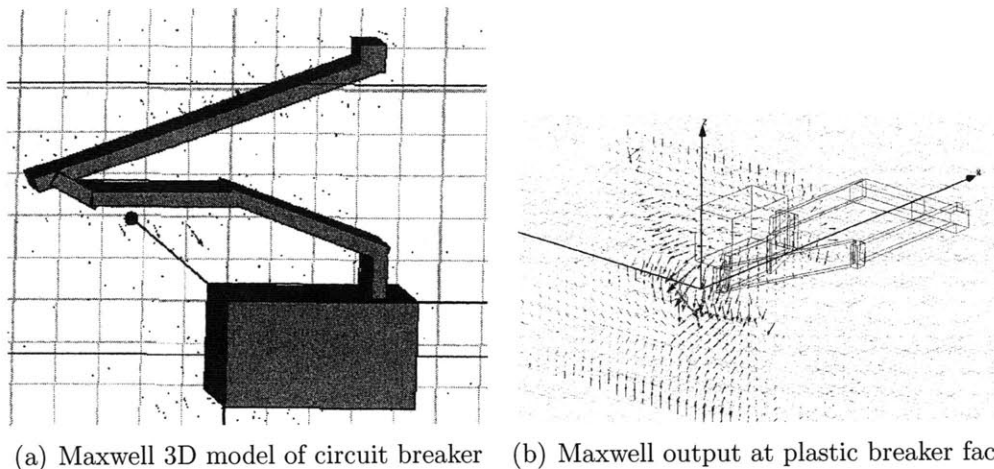


Figure 3-2: Maxwell 3D model of breaker with field lines drawn at the breaker face. One figure is from above and the other is angled to show field lines. The extra bar at the back of the view was added to complete the current path and has no effect on the field shown.

Ansoft's Maxwell 3D was used to model the current carrying member to identify the appropriate location of the breaker pickup. The output from this software is shown in Figures 3-2(a) and 3-2(b). The plane of magnetic field vectors is the plane of the plastic breaker face. The predominant flow of current through the circuit breaker is from top to bottom, so the magnetic fields were expected to wrap around the current carrying member across the breaker face. From Figure 3-1(b), this corresponds to a flux into the page just to the left of the breaker. Maxwell 3D confirmed that this was the appropriate model of the breaker and that the best position for the pickup was on top of the breaker where the metal is closest to the breaker face. At that point the magnetic field is most uniform and strongest.

The free software FEMM (Finite Element Method Magnetics) in its two dimensional mode was used to design the pickup. This software was written by David Meeker and can be downloaded from <http://www.femm.info/wiki/HomePage>. It can model the fields generated by sources into and out of a two dimensional plane. The Maxwell 3D model showed that the field at the surface strongly resembles the field produced by a point current source at the depth of the current carrying member. In FEMM, the plastic breaker face was modeled as air, and an analysis of the geometry was conducted. Various pickup shapes were considered to yield the strongest concentration of magnetic flux in the pickup core. An ideal pickup would focus as much magnetic flux into itself as possible, so it needed to roughly follow the magnetic field. FEMM showed that a half toroid of high permeability material placed on the breaker face was suitable in that it would result in significant flux-focusing in the material. The output of this program is shown in Figure 3-3. This is a representation of the breaker and pickup down its long axis.

This flux needed to be efficiently converted to an electrical signal. One possibility involved a Hall Effect sensor. This was not ideal because such a sensor would require voltage rails on the inside of the door where power is highly constrained. A coil was chosen for the sensor because the magnetic flux would directly produce a voltage signal that could be immediately used in the circuit.

Assuming the pickup coil has  $N$  turns of wire, the proposed model of the pickup

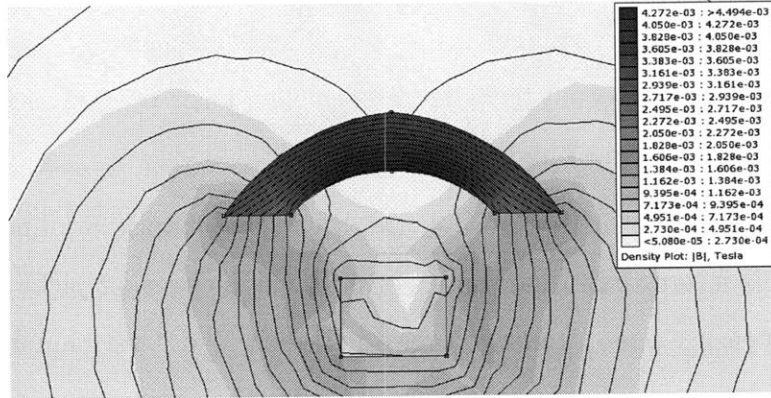


Figure 3-3: Finite Element Magnetic Model (FEMM) of magnetic flux through the breaker. The plastic breaker is ignored because it is not conductive or permeable to the magnetic flux.

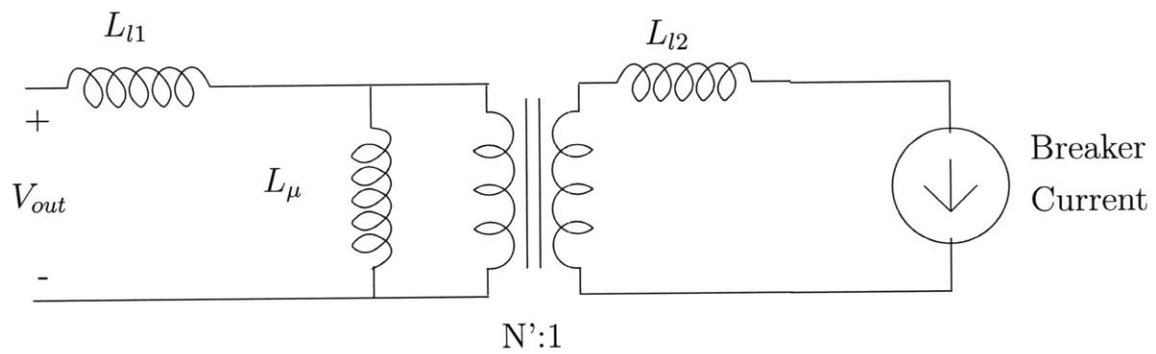


Figure 3-4: Breaker pickup model. The breaker pickup looks like a virtual transformer around the current to be measured.

is a  $1:N'$  step up transformer with poor coupling shown in Figure 3-4 where  $N' < N$  may be used to model the poor coupling from the breaker current member to the  $N$  turn windings. The current source inside the breaker face is modeled as driving the single turn of a  $1:N'$  turn virtual transformer. The leakage inductances  $L_{l1}$  and  $L_{l2}$  of the transformer are expected to be large because a large portion of the magnetic path consists of air instead of high  $\mu$  material.

Looking at the model in Figure 3-4, the breaker pickup is nonlinear across frequency. NILM is interested in harmonics of 60 Hz. The breaker pickup is connected to stages that measure open circuit voltage and do not load it very much. This means, according to the proposed model in Figure 3-4, that the output is approximately the voltage that the current source generates across the magnetizing inductance  $L_\mu$  given in Equation (3.1). In Equation (3.1),  $L_\mu$  is the magnetizing inductance,  $L_{l2}$  is the secondary leakage inductance,  $\omega$  is the breaker current frequency and  $I_{in}$  is the breaker current magnitude.

$$V_{out} \propto 2\pi j\omega L_\mu I_{in} \quad (3.1)$$

From this Equation it is clear that  $V_{out}$  is a function of both  $I_{in}$  and  $\omega$ . This nonlinearity applies gain to higher harmonics that must be characterized in the full system. It is clear that a higher  $L_\mu$  will lead to higher output voltage. From the Equation it is also clear that different frequencies will leave

The geometry of the coil fixes the level of leakage in this transformer. Using a high  $\mu$  material for the toroid and matching the shape of the toroid to the natural magnetic field lines make  $N'$  approach the true  $N$ . Increasing the cross sectional area of the toroid increases the flux it can capture and thereby improves coupling in the same manner.

Turns on the pickup are related to the turns ratio of the virtual transformer and the inductance looking into the pickup coil. The coupling is fixed by the geometry of the pickup and breaker. The easiest way to increase magnetizing inductance is to simply add more turns. Having a high magnetizing inductance is critical to proper

functioning of the circuit, otherwise, it tends to short out the signal to be measured. Unfortunately, the desired 60 Hz signal needs a high inductance for the impedance at that frequency to become appreciable i.e.,  $Z_\mu = \omega L_\mu$  will be small when  $\omega = 2\pi * 60$ .

Wire selection for the pickup was simplified by later stages of the JFET Mixer. At 60 Hz skin effect is negligible, and the pickup is not significantly loaded. This means that a very fine wire gauge can be used because its series resistance will not impact a voltage measurement of the coil.

Using the proposed virtual transformer model in Figure 3-4 as a design guide, a Ferroxcube part TX25/15/10-3E6 was used with 1200 turns of 34 AWG magnet wire. The toroid has a 25 mm outer diameter, a 15 mm inner diameter, and a 10 mm thickness. This toroid was cut in half on a diamond band saw, and the two halves were glued together side by side to increase cross sectional area. This toroid has a very high relative permeability of approximately  $10000\mu_0$  [4]. A photograph of this toroid affixed to the breaker is shown as Figure 3-5. The exposed current carrying member is partially obscured by electrical tape, but the wound toroid is clearly visible.

The voltage signal from this pickup still is too small to drive the JFET mixer at small current signal levels. A 1:14.1 step-up audio transformer module was added between the JFET mixer and the pickup. The audio transformer works at low frequencies and has enough turns that its own magnetizing inductance does not significantly load down the pickup coils at 60 Hz.

An additional resonant capacitor could be added to the audio step up transformer to boost signal levels at the cost of distortion. Normally, resonating with a small inductor at low frequency would require a large capacitor. In this case the inductor is reflected across the transformer, increasing its apparent impedance by the square of the turns ratio. This allows for a more reasonable choice of capacitor when added to the transformer secondary. The capacitor would need to be the appropriate value to resonate with the parallel combination of the magnetizing inductance of the audio transformer and the reflected inductance of the pickup. This method provides more voltage gain at the desired 60 Hz signal frequency while attenuating the higher harmonics that are important to NILM function and introducing a 180 degree phase

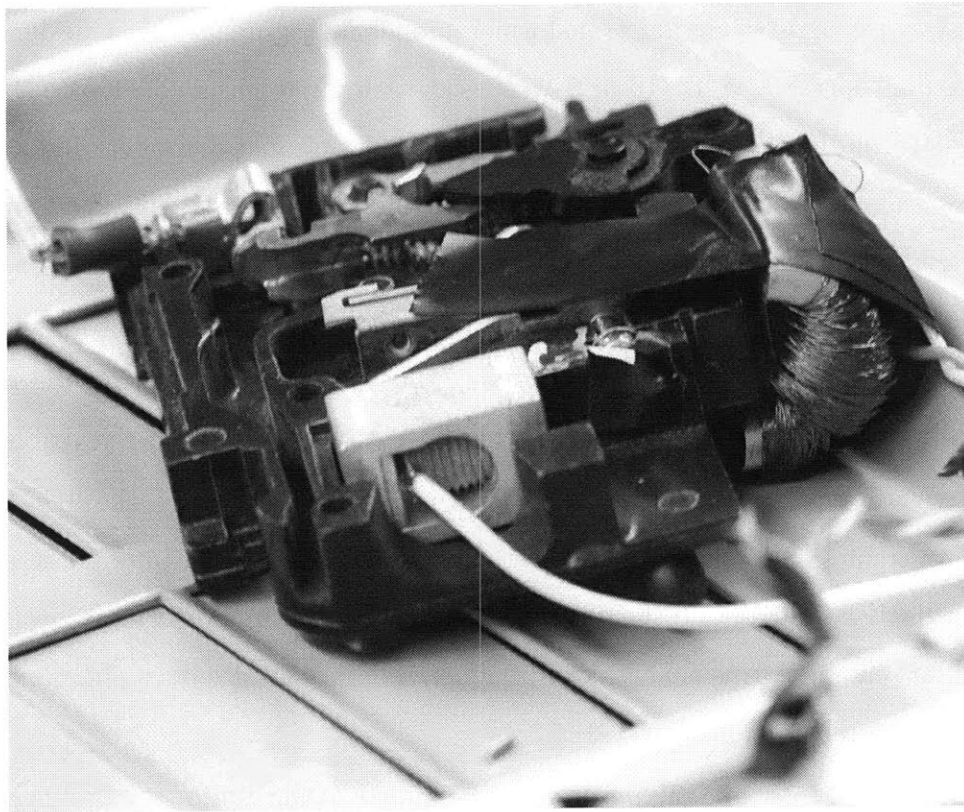


Figure 3-5: Breaker pickup photograph.



shift. This was not used in the experimental setup because gain was sufficient without it. In experiments two  $.47 \mu\text{F}$  capacitors were placed in parallel with the secondary of the transformer, but they were removed as they became unnecessary.

### 3.4 JFET Mixer

The four-quadrant balanced JFET modulator (mixer), shown in Figure 3-6, was designed to transmit information from the breaker pickup through the inductive link and out of the breaker panel. This circuit consists of two JFET devices for modulation control and two resistors to improve linearity, but it does not require a DC power supply.

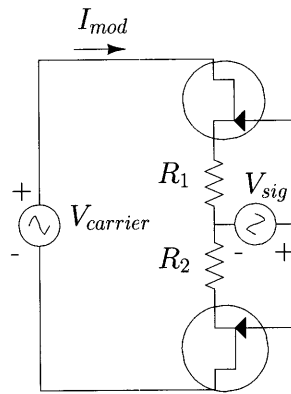


Figure 3-6: Adaptive Referencing Balanced two-JFET Modulator circuit enables simultaneous powering and modulation with no DC bus required

The JFET mixer can be modeled as a time-varying load on the carrier voltage source,  $V_{carrier}$  in Figure 3-6. In the circuit breaker system of Figure 3-1(a) the carrier voltage source that drives the JFET mixer is the voltage on the inner coil which couples from the outer coil. The load presented to  $V_{carrier}$  in Figure 3-6 varies with the control signal  $V_{sig}$  applied to the JFET gates and leads to a corresponding modulation of the current  $I_{mod}$ . The two-JFET mixer circuit is particularly advantageous for this application because it requires a minimal amount of circuitry inside the breaker door and lends itself to a low-cost solution.

An overview of the mode of operation of this circuit is instructive. The JFET is a

normally-on device that requires a negative gate to source voltage  $V_{gs}$  to turn it off. It is a symmetric device, meaning that the drain and source are interchangeable. By convention the source is the side of the JFET at a lower potential than the drain.

This circuit is best examined in half cycles of the carrier supply. On positive half cycles the lower JFET is fully turned on because the from the breaker pickup,  $V_{sig}$  signal drives its  $V_{gs}$  positive making its gate voltage much higher than both its drain and source. This JFET becomes a small on state resistance. Since the currents are low, the two resistors shown in Figure 3-6 drop little voltage. Most of the carrier voltage is impressed across the top JFET device. The top JFET then has a negative voltage applied to its gate due to  $V_{sig}$  that causes its current to be controlled by  $V_{sig}$ .

$V_{sig}$  from the breaker pickup is a bipolar signal, but this is controlled because the circuit is self-referencing. The voltage drop across the top resistor  $R$  acts to bias  $V_{sig}$  because  $V_{gs}$  is equal to  $V_{sig} - V_R$  with  $V_R$  the drop across the top resistor. As long as  $R$  is large enough,  $V_R$  will always be less than  $V_{sig}$  to keep the JFET device controlled. Approximating  $V_{gs}$  of the JFET as constant, the current through it then reduces to  $\frac{V_R}{R} = \frac{V_{sig} - V_{gs}}{R}$ .

On negative half-cycles of the carrier supply, the situation is reversed. In this way the symmetric nature of the JFET stack permits control during both half cycles of the applied sinusoid. Square wave multiplication of the two input signals is achieved because the sign of the mixer current matches the sign of the carrier while the amplitude is determined by the breaker pickup. This causes the envelope of the carrier to track the breaker pickup signal, yielding simple amplitude modulation. Because there is always some steady current in the system, the carrier is not suppressed. The signal looks like Equation (3.2) with  $\omega_c$  the carrier frequency,  $A$  the remaining carrier amplitude, and  $M$  the modulation amplitude. Some higher frequency terms appear after simplification of this expression, but with appropriate demodulation, the higher harmonics can be filtered away.

$$R(t) = \cos(\omega_c t)(A + M\cos(\omega_m t)) \quad (3.2)$$

The two resistors have been inserted to improve the performance of the mixer. They limit the steady state current of the mixer and improve the linearity of the response to the control voltage. This comes at the cost of gain, so an optimum choice balances those two concerns.

A small signal analysis of the circuit details its gain and design strategy. The symmetry of the device means that an examination of the circuit during one half cycle can be applied to its operation on the other one. In this analysis it is assumed that the voltage applied is well above the saturation voltage of the JFET devices and that it makes quick zero crossings. Ideally, the applied source should be a square wave, but a large enough sine wave is sufficient for proper operation.

Assuming a valid steady state voltage  $V_{ds}$  is impressed over the JFET and the  $V_{gs}$  is higher than its pinch-off voltage  $V_p$ , it will admit a current  $I_d$ . These are set by the JFET parameters, the two resistors, and the source impedance of the carrier supply. In a small signal sense, the top JFET device looks like a resistor  $r_o$  in parallel with a current source with magnitude  $g_m V_{gs}$ . The transconductance  $g_m$  is given in Equation (3.3) in terms of JFET parameters  $\beta$  and  $I_d$ . with  $\beta$  a JFET gain parameter. The output resistance  $r_o$  as in Figure 3-7 is  $\frac{V_{ds}+1/\lambda}{I_d}$  with  $\lambda$  the channel length modulation parameter of the JFET. The bottom JFET device looks like a small  $R_{on}$  resistor.  $R_{on}$  is assumed to be small compared to  $R_1$  and  $R_2$ .

$$g_m = \sqrt{\beta I_d} \quad (3.3)$$

Under the assumption that the sum of  $R_1$ ,  $R_2$ , and  $R_{on}$  is much smaller than  $r_o$ , simple expressions for the gain of the circuit can be found. The top JFET device is modeled as a transconductance source in parallel with a resistor  $r_o$  as shown in Figure 3-7. Looking at the small signal model in Figure 3-7, the transconductance source of the top JFET device has a value of  $g_m V_{gs} = g_m (V_{sig} - V_{R_1})$ .  $V_{R_1}$  is simply the current delivered through the current source,  $I_{in}$  if current through  $r_o$  is negligible. Rearranging terms to solve for  $\frac{I_{in}}{V_{sig}}$ , the gain  $G$  of the circuit, yields the gain from control voltage input to current output given in Equation (3.4). This needs to be

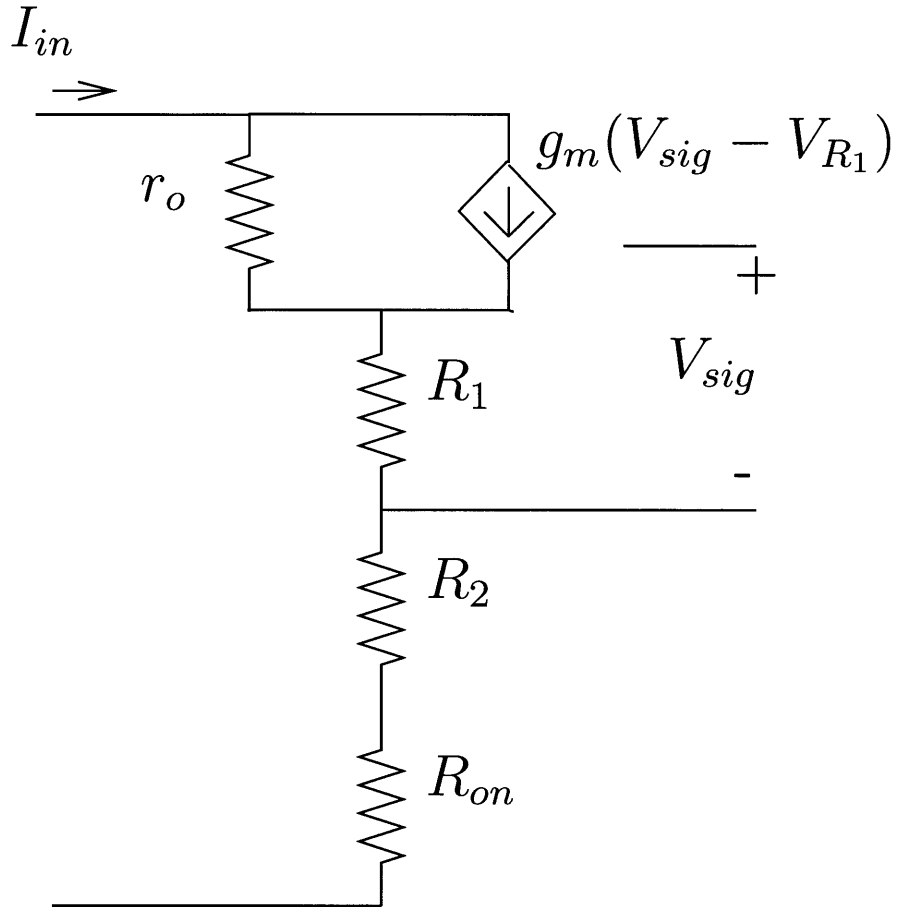


Figure 3-7: Small signal model for JFET Mixer

maximized within the power constraints of the door and the linearity provided by having a bigger  $R_1$ .

$$G = \frac{g_m}{1 + g_m R_1} \quad (3.4)$$

With this small signal understanding of the JFET mixer, actual JFET devices can be selected. The two most important device parameters are the  $\beta$  and the  $V_p$  of the device. Bigger  $\beta$  increases  $g_m$ . It also increases the zero input  $I_d$  because the JFET gate is further away from pinchoff.  $V_p$  is negative in JFET devices. However, bigger  $V_p$  values require larger  $V_{ds}$  values to maintain saturation. Operating under the assumption that  $g_m * R_1 \ll 1$  means that the gain is proportional to  $g_m$ . With some manipulation and removal of higher order terms,  $g_m$  is proportional to  $-V_p \beta / R_1$ .

Further assuming that  $R_{on}$  is smaller than  $R_1$  and  $R_1 = R_2$ , we find that  $V_{in} > -V_p(\beta + 1)$ . As such, a good JFET will have small  $V_p$ , and large  $\beta$ . The power available is essentially set by the applied carrier from the through-door inductive link, so the best solution maximally uses this available power with the highest  $I_D$  possible while maintaining saturation voltage across the JFETs.

The current setup uses PN4117A JFET devices from Fairchild semiconductor, and 1.2 k $\Omega$  resistors to improve linearity. These JFET devices have very small  $V_p$ , and modest  $\beta$ . These devices have a  $V_p$  between -.6 and -1.8 and a common source forward transconductance of 70 to 210 mmhos. Modulation behavior was confirmed experimentally.

### 3.5 Through-door Inductive Link

The inductive link across the steel door shown in Figure 3-1(a) consists of two resonant coils wrapped around samarium cobalt magnets with a  $N_1:N_2$  turns ratio. These two coils and the steel door form a magnetic circuit linking the two coils with the steel as a core material. A transformer model of the system will be developed from a reluctance model of this magnetic circuit.

The reluctance model of the core was created to better understand the effects of high leakage and core eddy current losses on the eventual transformer model of the system. To a first approximation, the reluctance model looks like Figure 3-8. Reluctance is given in Equation (3.5) where  $l$  is the magnetic path length and  $A$  is the magnetic area. The MMF generators  $N_1I_p$  and  $N_2I_2$  model the two coils. The reluctances  $R_{core}$  capture the reluctance of the magnets on each side of the door.  $R_{air}$  models the flux path through air around the two coils.  $R_{steel}$  captures the steel path between the coils. Note that the mutual path linking the two coils requires passing through both  $R_{core}$  and  $R_{air}$ , two reluctances that are relatively large. This path is in parallel with a shunt path consisting of low reluctance steel  $R_{steel}$ . Most of the flux will flow across this low reluctance parallel path.

$$R = \frac{l}{\mu A} \quad (3.5)$$

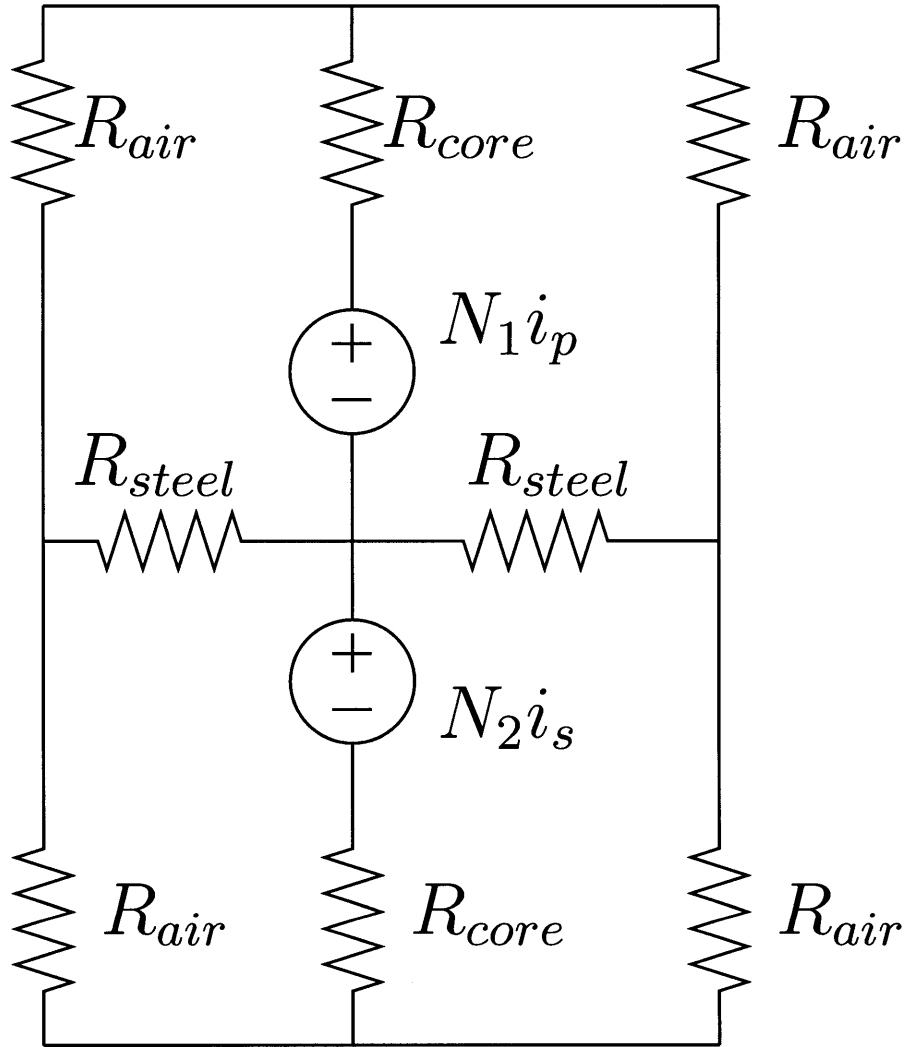


Figure 3-8: Reluctance model of through door transmission

In [1] it was shown that the steel becomes less permeable at higher frequencies. This means that  $R_{steel}$  is strongly frequency dependent. At low frequencies it is a reluctance in Figure 3-8, but it increases with frequency as the  $\mu$  of steel decreases. At higher frequencies the eddy currents of the steel have become significant. At some high frequency the flux is effectively rejected from the steel by eddy currents. The  $R_{steel}$  reluctances will become  $R_{air}$  at that time, and  $R_{core}$  will increase as the magnetic core eddy currents reject the flux. There is likely an optimum frequency

where the shielding due to high-frequency eddy currents and low-frequency magnetic permeability is minimized. At that point the mutual path linking the two coils will be the most favorable.

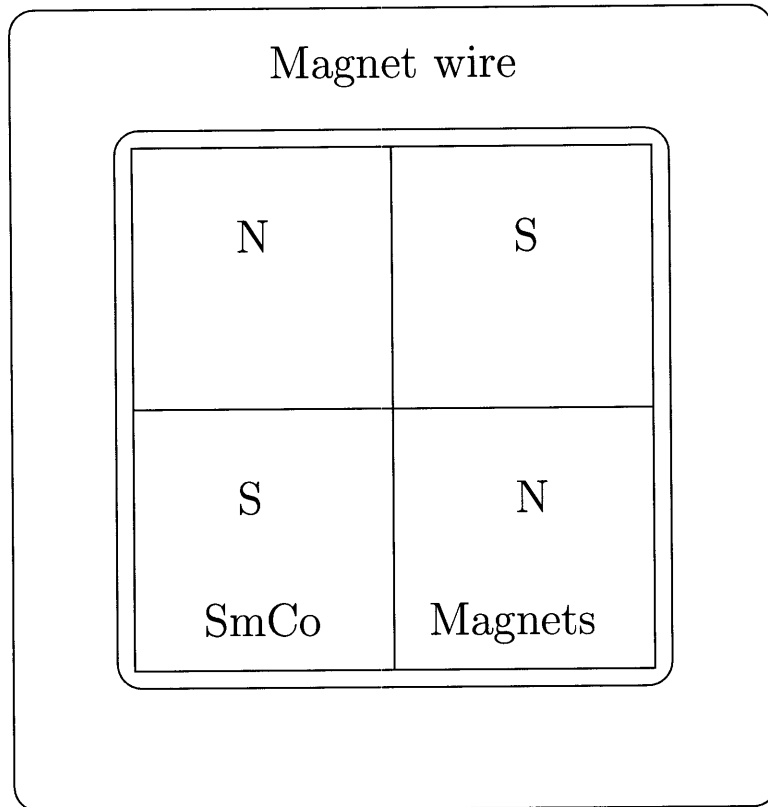


Figure 3-9: Top view of transmission coil configuration

One innovation in the coil design involves using powerful magnets to alter the steel door permeability by partially saturating it. Both coils of the inductive link are wound around high field strength samarium cobalt magnets shown from above in Figure 3-9. In the reluctance model of 3-8, this increases  $R_{steel}$  by making a return path along the steel door have higher reluctance. This also increases  $R_{core}$ , but the thickness of the steel door relative to the radius of the coil is very small.

Neodymium was considered for the core because of its high field strength, but it did not work because the nickel coating of the magnets led to unacceptably high losses. The lower conductivity of samarium cobalt and the lower price make it better as a core material.

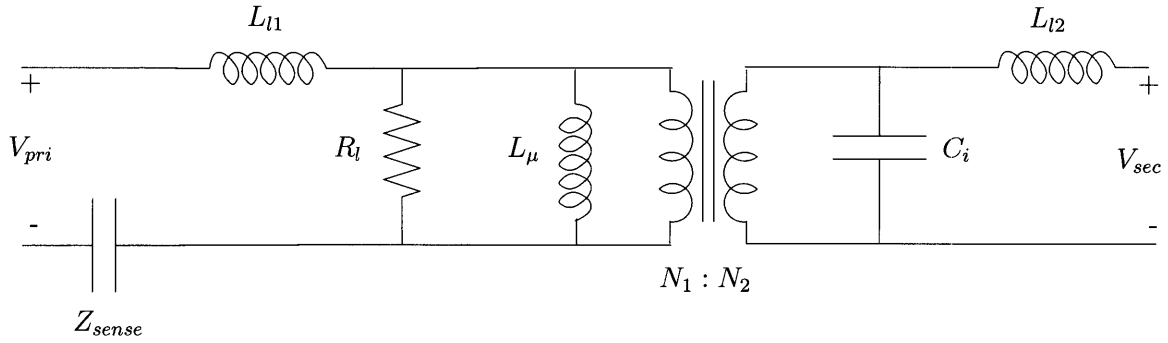


Figure 3-10: Transformer model of the through door inductive link.

The magnets provide a convenient means of securing the device to the steel door, and they can further decrease the permeability of the steel as mentioned above. This tends to shift the permeability curve downward allowing a lower operating frequency for a given value of steel  $\mu$ . A lower frequency is desirable because it leads to less eddy current loss. From Figure 3-8, the magnets directly increase  $R_{steel}$  for a value of eddy current loss fixed by the operating frequency.

The dimensions of the coil are important to the reluctance model. A coil with a larger radius should work better because it will increase the inductance directly. As mentioned before, saturating the steel works because the radius of the coil is large compared to the thickness of the door. Making the coil radius big makes that statement even more true. A flatter coil is also better because it makes the mutual path linking the two coils shorter. Therefore, the coil should be as broad and flat as is practical.

This reluctance model of the door-coil system was used to motivate a transformer model of the coil system depicted in Figure 3-10. The loose coupling of the coil system is captured by having very large leakage inductance terms ( $L_{l1}$  and  $L_{l2}$ ) and a very small magnetizing inductance ( $L_{\mu}$ ) because of the significant leakage from the reluctance model. Core eddy current loss in the door is modeled as a shunt resistor ( $R_l$ ) in parallel with the already small magnetizing inductance. This means that at high frequencies most of the voltage applied is dropped across the leakage inductance and is not transferred through the magnetizing inductance. In addition, there is



an interwinding capacitance component  $C_i$  that makes the system self-resonant at a certain frequency. In this electrical transformer model, the ideal frequency discussed above occurs when the loss resistor impedance matches the magnetizing impedance at the carrier frequency and is as large as possible.

Finding this operating frequency is a non-trivial matter complicated by interrelationships between the door and the coil geometry. The ideal frequency is a function of the steel material properties, the dimensions of the coils, and the thickness of the door. Computer modeling of this system to find the ideal frequency is difficult because the frequency relies on the presence of strong magnets. Saturation of magnetic core materials is poorly modeled, and the frequency dependence of steel electrical properties are poorly understood. FEMM and Maxwell 2D were used to gain an intuitive understanding of the problem, but experimentation was required to find an empirical solution.

The design and operation of the through-door inductive link must take into account its intended use. The signal of interest to the sense circuitry of Figure 3-1(a) is the current drawn by the JFET mixer on the inside of the door. That current is the high-frequency carrier modulated with the low-frequency (60 Hz with harmonics) signal sensed from the breaker face. A large turns ratio,  $\frac{N_2}{N_1}$  from Figure 3-1(a), yields a large voltage gain to the inner coil to develop the necessary saturation voltage on the JFET mixer. Meanwhile a large turns ratio amplifies the current drawn by the JFET mixer to the outer coil. The turns on the inner coil should consist of as many turns of wire as can fit of a fine gauge of magnet wire. There is little power on the inside of the door, so thick wire is unnecessary. The number of turns on the outer coil is lower-bounded by the current drive capability of the voltage source. It should be constructed of thick enough wire and wound with enough turns to match the output impedance of the voltage source driving it.

Because the system only need operate at the carrier frequency, the impedance problems of driving the coil can be solved through resonance. A high number of turns inside the door is advantageous for the functioning of the circuit, but it also leads to high interwinding capacitance parasitics shown in Figure 3-10 as  $C_i$ . This parasitic

can be used advantageously by recognizing that it provides a parallel self resonance on the inside of the door. Driving the system at approximately that frequency yields more voltage gain on the secondary that helps establish operating voltage on the JFETs. The addition of a series capacitance on the primary matches the two coils at the operating frequency for maximum signal transfer. In this way, the high leakage of the system can be mitigated from each side of the transformer. Additionally, the capacitor provides a useful sense impedance for measuring the current drawn by the coils. The resonant capacitor impedance should match the winding impedance for best measured signal. This is coincident with resonance.

## **3.6 Sense and Demodulation Circuit**

The sense circuit is a DSP-based I/Q demodulation circuit on a printed circuit board. The schematic and layout for this system are given in Appendix E. It consists of a power front-end responsible for driving the coils, an analog filter chain responsible for demodulating the signal, and a DSP for performing post processing and filtering of the final signal. The signal is then outputted using a DAC for viewing on an oscilloscope or spectrum analyzer.

### **3.6.1 Power front-end**

The power front-end is a push-pull driver composed of two BJT devices capable of standing off high voltage and delivering reasonable current. The bases of these BJT devices are driven using a high voltage decompensated operational amplifier in a high gain configuration. A square wave at the carrier frequency is AC coupled to the noninverting input of the amplifier. The operational amplifier then increases the voltage to a level suitable for driving the push-pull.

The push-pull driver is connected to the series combination of the coil and two sense impedances. These sense impedances can either be resonant capacitors or sense resistors depending on signal requirements and willingness to accurately match the inner coil.

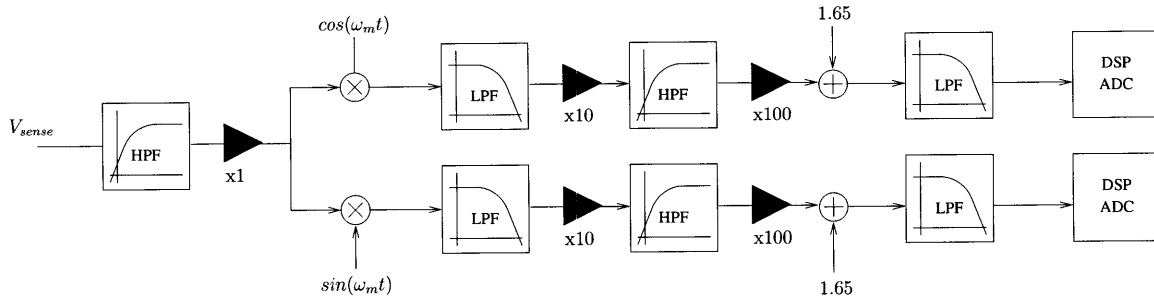


Figure 3-11: Block diagram of analog filter chain

The voltage between the two sense impedances is taken as the input to the analog filter chain. The total sense impedance is matched to the coil system at the carrier frequency, but the ratio of the two impedances is chosen to deliver acceptable voltage levels to the analog filters.

### 3.6.2 Analog filter chain

A block diagram of the operations performed by the chain is given in Figure 3-11. The analog circuitry requires as input the voltage across the coil sense impedance ( $V_{sense}$ ) and two square waves at the carrier frequency 90 degrees apart in phase. Its output is the in phase and quadrature component of the input signal (I and Q channels, or  $\cos(\omega t)$  and  $\sin(\omega t)$ ). All stages of the filter chain operate on  $\pm 15$  voltage rails, and the final output of the chain is limited to the 0 to 3.3 Volt range of the DSP analog-to-digital converter.

The input voltage is first high-passed to remove any DC components present in the sense voltage and center it in the filter chain input stage. This will not damage the signal because at this stage it is still modulated with the carrier. This signal is then gained to ensure that it occupies the  $\pm 15$  volt range of the filter chain.

The next stage is I/Q demodulation. Two analog switch bridges are independently operated with the I and Q waveforms to provide a differential output signal. This multiplication brings the desired breaker current signal down to baseband, but the high frequency components are still present. At this point, the remaining stages of the filter chain are exactly parallel for the I and Q channels.

The differential signal is passed through a third order lowpass RC ladder filter with a cutoff of 1 kHz, marked as LPF in Figure 3-11, to attenuate high frequency content. This signal then passes through an instrumentation amplifier with a gain of 10 (x10 in Figure 3-11) to reference it to the system ground while applying some gain. This reduces the signal to a DC offset with the 60 Hz data on top of it. The DC offset is the nonsuppressed carrier demodulated to DC.

The low modulation depth of this system means that gain in the signal chain is required for suitable SNR at the ADC input. At this point in the filter chain the carrier has been demodulated to DC, leading to an offset. That offset must be removed with a highpass filter before gaining the signal (HPF in Figure 3-11). Otherwise, the offset is gained with the signal and saturates the filter chain. The high pass filter has a cutoff of 10 Hz, and the DC offset is suppressed. This signal is then passed through a gain stage with a gain of 100 (x100 in Figure 3-11). At this point, the DC signal level is wrong for the single-ended analog-to-digital converter. The final step is a level shifter to add 1.65 volts to the signal (+1.65 in Figure 3-11). This centers the signal in a 0 to 3.3 Volt range for use in the analog-to-digital converter. A final LPF is used as an anti-aliasing filter before the input to the ADC (LPF in Figure 3-11).

### 3.6.3 DSP operation

The device uses a dsPIC33 device to perform digital processing and filtering of the incoming signals. It is responsible for sampling both the I and Q channels and combining them into a demodulated signal. A brief overview of I/Q demodulation of an amplitude modulated signal should clarify the operation of the DSP.

### 3.6.4 I/Q Demodulation Overview

An AM signal looks like Equation (3.6) with  $\omega_c$  the carrier frequency,  $A$  the carrier amplitude,  $M$  the modulation amplitude,  $\omega_m$  the modulation frequency, and  $\phi$  an arbitrary phase offset.  $M$  and  $\omega_m$  are desired. In this device the phase term is unknown or time varying, so I/Q demodulation is used to detect the signal and cancel

out the phase term. A Phase Locked Loop (PLL) would also have demodulated the signal by locating the unknown phase and permitting synchronous detection.

$$R(t) = A\cos(\omega_c t + \phi) + \frac{M}{2}(\cos((\omega_c + \omega_m)t + \phi) + \cos((\omega_c - \omega_m)t + \phi)) \quad (3.6)$$

For the I channel, the signal in Equation (3.6) is multiplied by  $\cos(\omega_c t)$ , while the Q channel is multiplied by  $\sin(\omega_c t)$ . Each channel is then low pass filtered to eliminate high frequency signal components. After using some trigonometric identities and assuming high frequency terms have been perfectly eliminated by low-pass filtering, the final signals are given in Equations (3.7) and (3.8).

$$I = \cos(-\phi)\left(\frac{M}{2}\cos(\omega_m t) + \frac{A}{2}\right) \quad (3.7)$$

$$Q = \sin(-\phi)\left(\frac{M}{2}\cos(\omega_m t) + \frac{A}{2}\right) \quad (3.8)$$

These two expressions are the output of the multiplier and lowpass circuit. Before being sampled by the ADC, these are highpass filtered to remove the  $\frac{A}{2}$  offset. One note here is that assuming  $\frac{A}{2}$  is larger than  $\frac{M}{2}$ , these quantities are strictly positive or negative as determined by  $\phi$ . This is the case for the door system because the modulation depth is very small.

Squaring these two channels after a high pass filter, adding them, and taking the square root removes the  $\phi$  terms and recovers the desired signal. However, the cosine signal has lost sign information in the operation that must be recovered. Both I and Q will follow the modulated wave and be multiplied by an unknown constant determined by  $\phi$ . There are two ways of preserving this sign information. One method involves choosing either the I or Q channel to be the sign reference. The other method involves adding an offset to both channels to ensure that they are both positive entering the square root.

The first method is the cleanest method mathematically, but it leads to a problem when the magnitude of the sign reference is small. In that case, there is uncertainty

about the sign of the output, and noise can cause the signal to rapidly cross this threshold. It also requires some prior knowledge to know which channel, I or Q, is the correct sign reference. Having access to the raw I and Q channels before high pass filtering is sufficient for making this determination, but it requires more ADC channels or an analog comparator that can provide digital input to the DSP.

The algorithm for using this information involves reasoning about the sign of  $-\phi$ . I and Q will be either strictly positive or strictly negative because of the large carrier that has been demodulated to DC. Suppose I and Q are both strictly positive. This implies that both  $\cos(-\phi)$  and  $\sin(-\phi)$  are positive, meaning that  $-\phi$  is between 0 and 90 degrees in the first quadrant. After removing the DC offset, the I and Q channels will be in phase. I will be larger in magnitude if  $-\phi$  is closer to 0 and Q will be larger if it is closer to 90. Either channel is suitable for a sign reference, so the larger one should be used because it crosses through zero faster than the other channel. A similar argument could be applied if both channels are negative and in agreement.

The problem becomes interesting if one is positive and one is negative. Suppose I is positive and Q is negative, placing  $-\phi$  in the fourth quadrant. This implies that  $\cos(-\phi)$  is positive and that I is the “true” reference because its sign tracks the true sign. However, this also implies that Q is of the opposite sign. Again, the larger channel can be used and the sign flipped if that is the Q channel.

The other method of adding an offset is currently used in the system because it requires less analog hardware. An overview of the impact of this offset is important to understand the tradeoffs it involves.

Assume that the I and Q channels have a DC offset added that is different in each channel. Since this offset is unwanted arbitrary error in the filter chain, it could be called  $O_I \frac{M}{2} \cos(-\phi)$  and  $O_Q \frac{M}{2} \sin(-\phi)$  with  $O_I$  and  $O_Q$  chosen appropriately. The channels become distorted as in Equations (3.10) and (3.10).

$$I = \frac{M}{2} \cos(-\phi) (\cos(\omega_m t) + O_I) \tag{3.9}$$

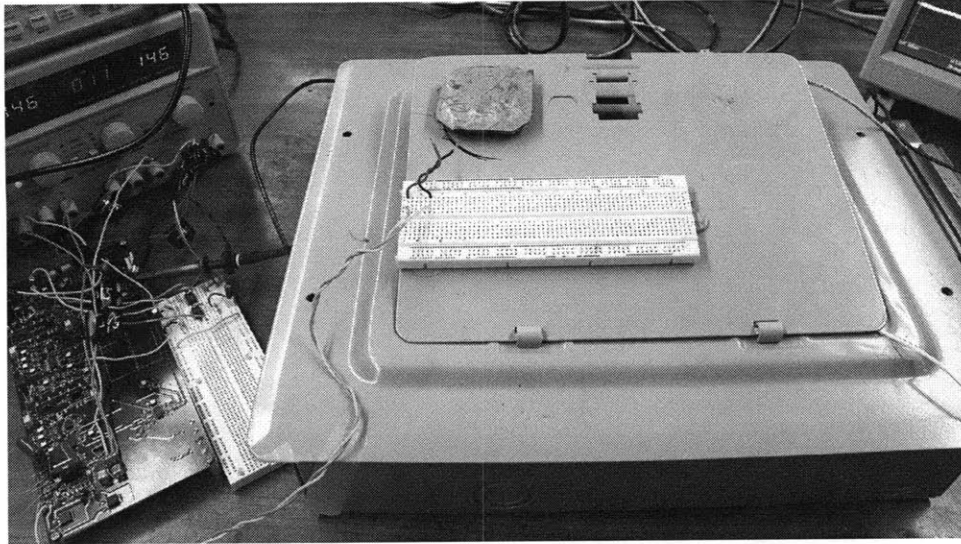


Figure 3-12: Entire experimental setup with circuit breaker door closed. The outer through door coil and demodulation circuitry are visible.

$$Q = \frac{M}{2} \sin(-\phi) (\cos(\omega_m t) + O_Q) \quad (3.10)$$

When the expressions in these equations are squared and summed, the offsets generate cross terms that cause phase-dependent distortion. For this reason, the DSP should subtract out any offsets prior to squaring the incoming channels. This is done by median filtering the signal and subtracting out the median from all samples.

The previously mentioned method of adding an offset does lead to unwanted noise and distortion. It also limits the range of the system to only half of the full bipolar range. In effect, this method trades high noise at the zero crossings for lower noise distributed across the entire signal. For now this was done to make cleaner waveforms, but better software and filter chains should make the sign reference method a better choice.

### 3.7 Test setup and results

The experimental setup is an implementation of the system described here. The entire system is shown in Figure 3-12. A small circuit breaker panel was used with a circuit breaker resting inside the panel rather than installed properly. This was done

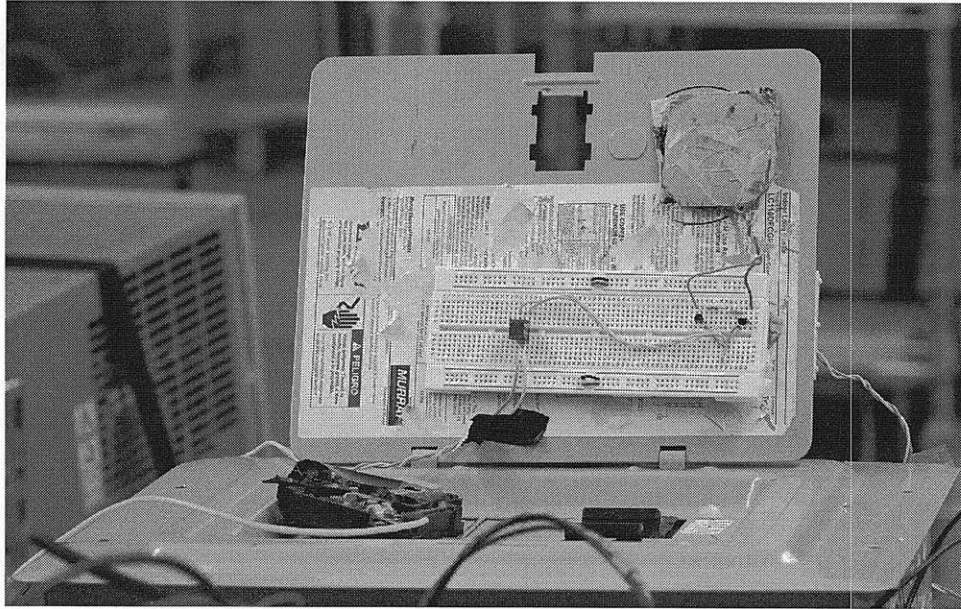


Figure 3-13: Experimental setup with circuit breaker door open. The JFET Mixer, test circuit breaker, and inner coil are visible.

to allow easy access to the circuit breaker for testing. This breaker was left in the ON position and connected to a  $2\ \Omega$  resistor and an HP 6834B AC power source. The source can provide variable frequency current waveforms up to 5 Amperes through the breaker.

The current pickup was secured to this breaker face with electrical tape with leads running to a solderless breadboard attached to the inside surface of the steel door. This was connected to the 14.4:1 step up audio transformer, the Tamura MET-01, before being connected to the JFET devices. This portion of the setup is shown in Figure 3-13.

On the same breadboard were two PN4117A JFET devices and two  $1.2\ \text{k}\Omega$  resistors in the configuration described earlier. These were connected to the inside coil.

The inside coil consists of four samarium cobalt magnets of dimension  $1/2$  inch by  $1/2$  inch by  $1/4$  inch grade 26 MGOe arranged as shown in Figure 3-9. Around this were 1000 turns of 34 AWG magnet wire and epoxy to hold the structure in place. The coil was constructed with cardboard on both sides to ease winding, but cardboard on the side closest to the door was carefully removed after the epoxy cured.



This ensured that the system would be as flat as possible against the door.

A similar coil was made to connect to the other side of the steel door. It was arranged so that the N and S poles of each magnet were on top of each other. This coil only has 24 active windings. During development, 1000 turns were placed on the core, but these are not being used now. This coil electrically connects to the printed circuit board described above.

The solderless breadboard, sideways breaker, and inner coils are all too thick to allow the steel door to fully close. It is likely that doing so will lower signal levels, but a new coil design and surface mount components for the breadboard would be required.

The demodulation board pictured in Figure 3-14 was used to drive the coils and demodulate the signal. However, the current board did not have a crystal oscillator installed and is instead operating from an internal RC oscillator. The frequency delivered from this oscillator could not produce a stable carrier for use in the power stage or demodulation stage. Instead, the present setup uses an HP 33120A signal generator to provide double the operating frequency. A dual flip-flop and inverter provide in phase and quadrature square waves suitable for the demodulator as shown in the schematics in Appendix E.

The stability of the carrier is of critical importance in this system because of the low frequency nature of the desired signal. Variations in carrier frequency from DSP clock jitter overwhelm the desired signal easily. This is why the signal generator was used instead.

Other changes were made to the schematic in Appendix E. These included:

- The addition of level shifting circuitry
- The adjustment of the high pass filter. The high pass filter was implemented with a  $.1 \mu\text{F}$  capacitor and a  $150\text{k} \Omega$  resistor. These were changed to  $1 \mu\text{F}$  and  $15 \text{k} \Omega$  respectively. This improved offset at the input to the x100 gain stage in Figure 3-11.

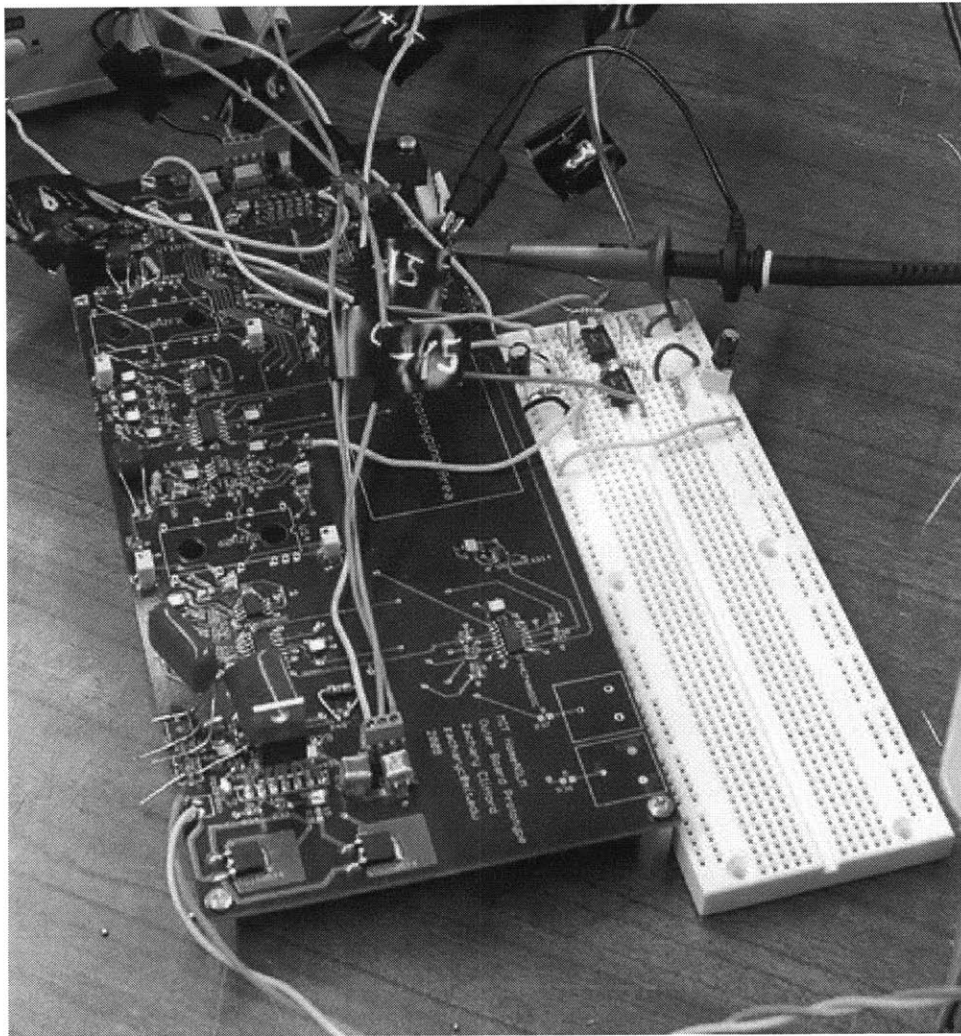


Figure 3-14: Demodulation board attached to power supply.

### 3.7.1 Coil design procedure

The most complicated part of this system is the through-door inductive link design. In the current setup, the design was very empirically motivated.

Since an optimum frequency for a given door is unknown and difficult to measure without a coil, the first step is to wind a coil set with many turns on one coil and few turns on the other. The next step is to locate the resonance of the coil system by sweeping the operating frequency. Connecting the primary coil to a signal generator and sweeping it from approximately 10 kHz to 300 kHz while monitoring the secondary coil with an oscilloscope worked well. However, adding a ground connection on the secondary coil by using an oscilloscope changes the circuit model of the through-door link significantly because the signal generator driving the outer winding is also ground referenced. This might lead to error in finding the resonance because the real system has no such direct connection through the door. However, this measurement was taken to provide an approximate resonant point for the secondary coil. The final step is to find the impedance of the primary coil at that frequency where the secondary coil is approximately resonant. This allows for the selection of a series impedance to match the primary coil for the purposes of measuring the current for demodulation. This impedance may be resistive or capacitive if resonance is desired. Iteration may be required to maximize signal output in the true system.

Modifications to this procedure might produce a better coil setup. If the coil can be connected to the sense circuitry mentioned in Figure 3-1(a), the demodulated output can be monitored while the frequency is swept. This implies that the DSP of the demodulation circuit could perform resonance location at startup. The problem is that the sense impedance must be variable as well to attain good matching between the coils and that sense impedance. It is an unclear tradeoff whether operating in an unmatched state or operating away from the secondary resonance is better. To be sure, the sense impedance and frequency would both need to be swept simultaneously whilst always ensuring an impedance match at the signal frequency.

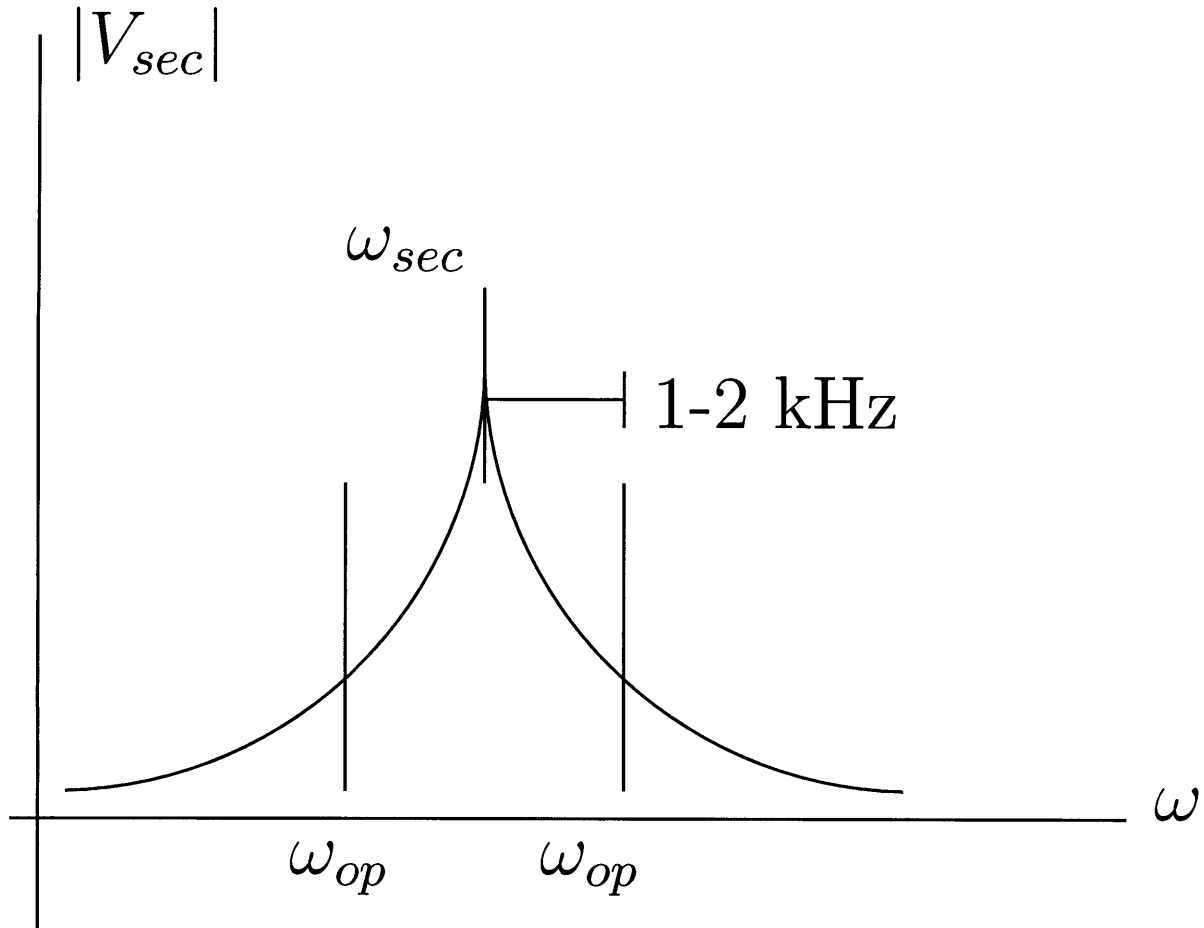


Figure 3-15: Operating point compared to secondary coil resonance. Ideally, the carrier frequency should be 1-2 kHz away from the resonance of the secondary coil.

### 3.7.2 Results

The experimental setup described in Section 3.7 was used to determine signal levels and evaluate the present setup.

Experiments showed that a resonant secondary is sufficient for adequate functioning of the system and that the primary need not be resonant. The resonant capacitor can be replaced with a small current sense resistor. This reduces signal levels, but it removes the requirement that the two coils be matched. Should the capacitor not be exactly matched, the two nearby resonant points will distort the signal.

Empirical results from this coil system showed that the coils work best approximately 1 to 2 kiloHertz off the resonance of the secondary coil on either side of the

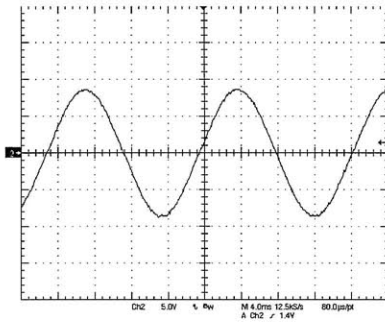
resonance as shown in Figure 3-15. In the Figure,  $\omega_{sec}$  is the resonant frequency of the secondary coil, and  $\omega_{op}$  are desirable operating frequencies for the carrier. This configuration is advantageous because the resonance is variable because it is a complicated function of the door properties and the immediate surroundings. Moving a hand closer than a foot or two to the door panel affects received signals considerably. This may be due to moving of the resonant point by changing the capacitance of the door with the present coils. When off resonance, the gain is still high, but small changes in the resonant point do not strongly affect the signal. Working on one side of the resonance also ensures that the rapid phase shift at a resonant point does not perturb the signal.

Figure 3-16 shows some results from this device at high current levels of 5 Amperes. From the spectrum plots it is clear that some higher harmonic distortion is present, but the time domain waveforms appear relatively clean.

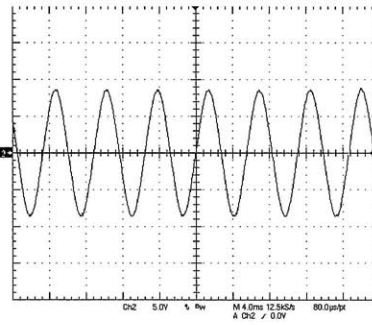
Figure 3-17 shows the performance of the system at low current levels of .2 and .1 Amperes. The noise present on these signals is predominantly quantization noise from the analog-to-digital converters in the block diagram of Figure 3-11 integrated into the DSP on the schematic in Appendix E. Moreover, due to signal chain problems in the current board, the final signal covers less than a quarter of the input dynamic range of the analog-to-digital converter. This means that the small signals are dominated by quantization error rather than instrumentation noise. This problem could be fixed by applying more gain to utilize the full dynamic range of the ADCs. This was not done because of high offset error in the high pass filter stage of Figure 3-11. Gain applied at that stage would push the desired signal above the input range of the ADCs.

Of particular interest is the magnitude plot at the carrier frequency. The signal of interest is below the noise floor of the spectrum analyzer, but the I/Q demodulator is capable of recovering the signal anyway. With further improvements to the demodulator it is likely that even smaller signals will be observable.

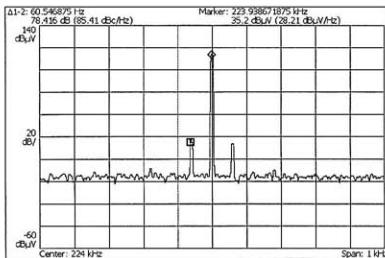
Figure 3-18 shows demodulation and processing of a 70 Hz signal. Since 70 Hz is not among the harmonics of 60 Hz, this shows that the signal being measured is actually from the current source and not unwanted pickup from the air. The relative



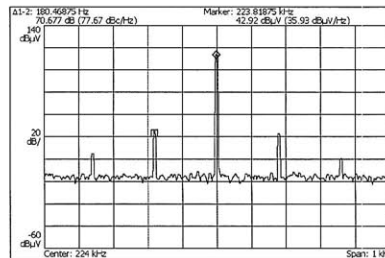
(a) 60 Hz sine wave input



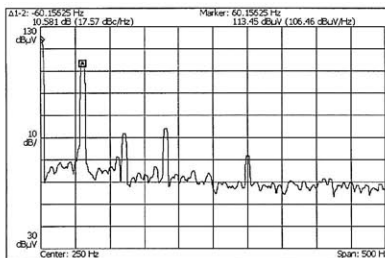
(b) 180 Hz sine wave input



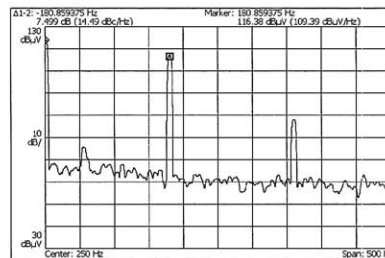
(c) 60 Hz sine wave magnitude at carrier frequency



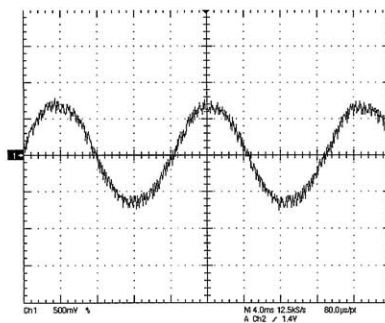
(d) 180 Hz sine wave magnitude at carrier frequency



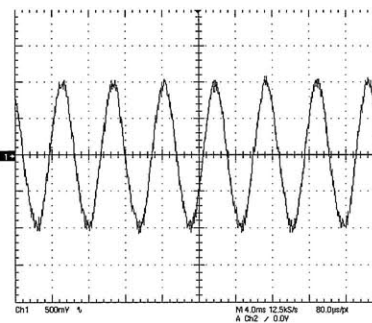
(e) 60 Hz sine wave magnitude at baseband



(f) 180 Hz sine wave magnitude at baseband

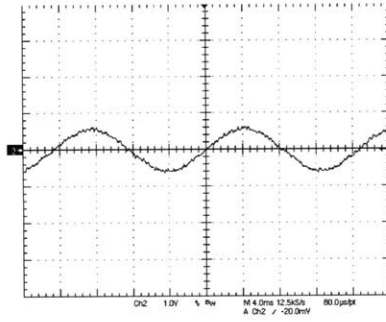


(g) 60 Hz sine wave demodulated output

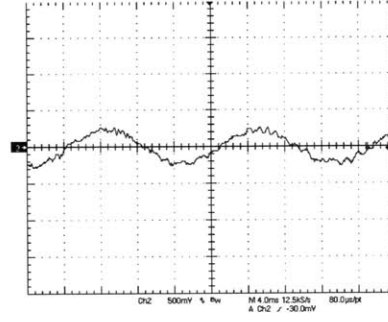


(h) 180 Hz sine wave demodulated output

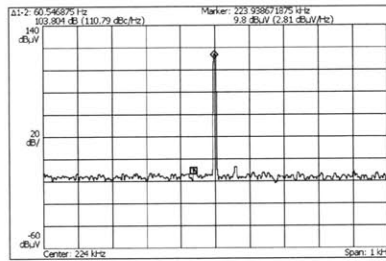
Figure 3-16: Experimental results. The system is capable of measuring current signals and reconstructing them outside the door. The 60 Hz and 180 Hz input signals were 5 Amps in amplitude. The input signal is the voltage across a  $2 \Omega$  resistor and captures the input current.



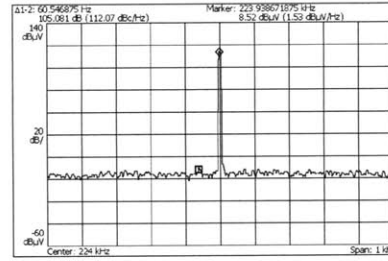
(a) 60 Hz .2 A sine wave input



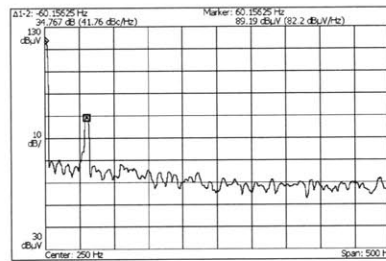
(b) 60 Hz .1 A sine wave input



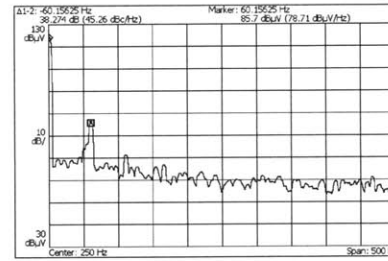
(c) 60 Hz .2 A sine wave magnitude at carrier frequency



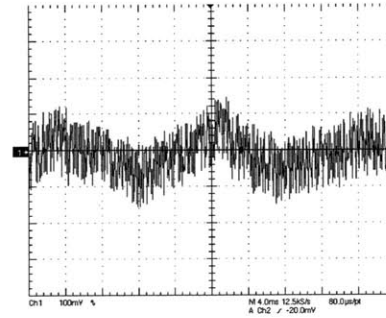
(d) 60 Hz .1 A sine wave magnitude at carrier frequency



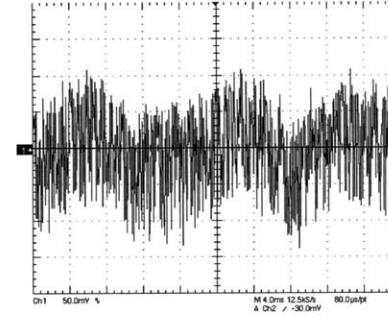
(e) 60 Hz .2 A sine wave magnitude at baseband



(f) 60 Hz .1 A sine wave magnitude at baseband

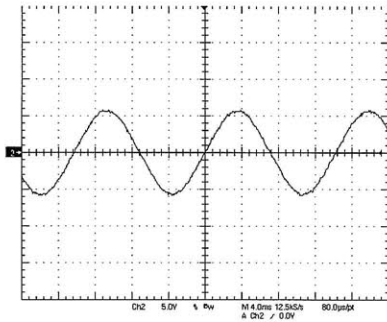


(g) 60 Hz .2 A sine wave demodulated output

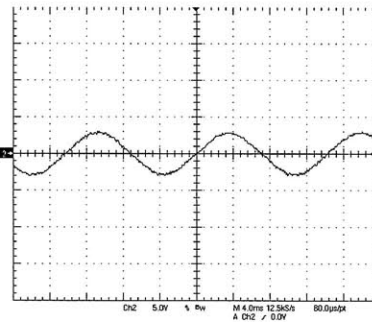


(h) 60 Hz .1 A sine wave demodulated output

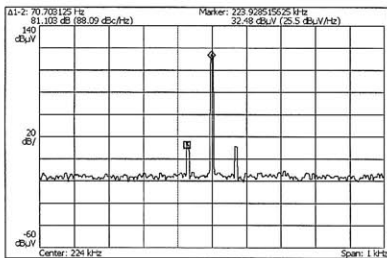
Figure 3-17: Experimental results at low current levels. The system is capable of measuring current signals and reconstructing them outside the door. Even signals as small as 100 mA can be resolved. The input signal is the voltage across a 2  $\Omega$  resistor and captures the input current.



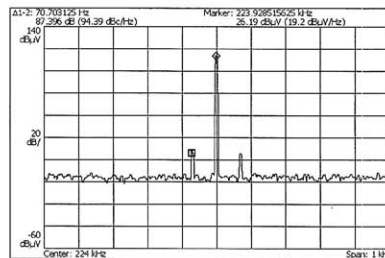
(a) 70 Hz 2 A sine wave input



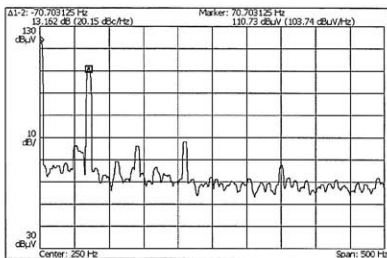
(b) 70 Hz 1 A sine wave input



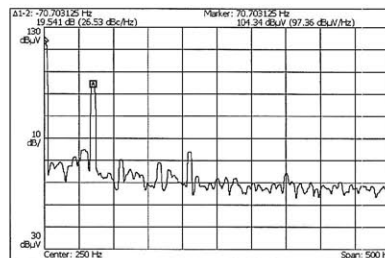
(c) 70 Hz 2 A sine wave output frequency magnitude



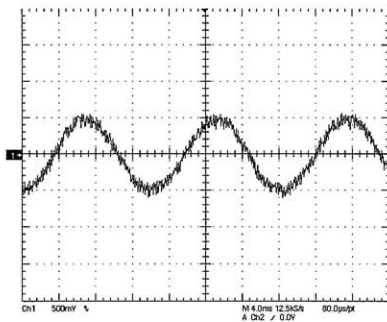
(d) 70 Hz 1 A sine wave output frequency magnitude



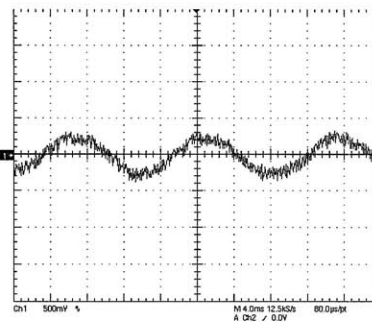
(e) 70 Hz 2 A sine wave output frequency magnitude



(f) 70 Hz 1 A sine wave output frequency magnitude



(g) 70 Hz 2 A sine wave demodulated output



(h) 70 Hz 1 A sine wave demodulated output

Figure 3-18: 70 Hz experimental results. The system is capable of measuring current signals and reconstructing them outside the door. Both 2 A and 1 A 70 Hz signals can be resolved. These show that the measured signal is the one being generated and not pickup from the air. The input signal is the voltage across a 2  $\Omega$  resistor and captures the input current.



absence of 60 Hz pickup from air implies that the 60 Hz signals measured before have very small pickup from air.

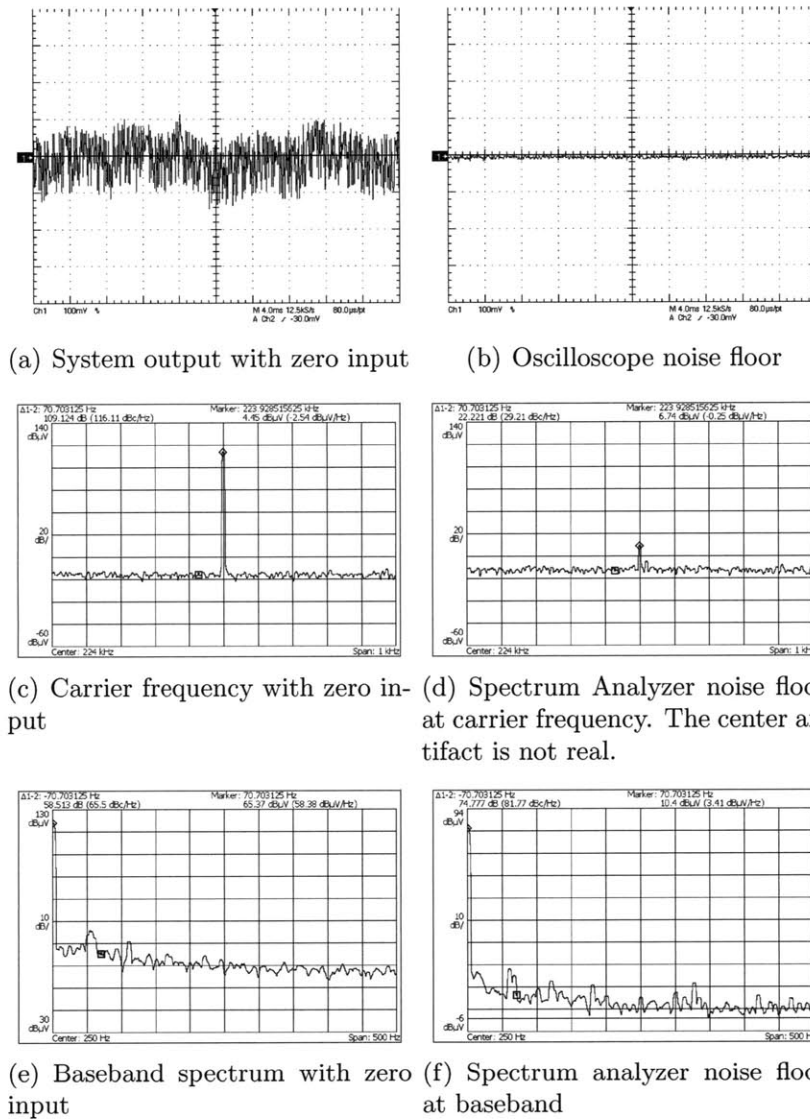


Figure 3-19: Noise floor of instrumentation. The left column are system outputs with no current signal to illustrate system noise levels. The right column are shots with the input ports of the test equipment shorted together to illustrate instrumentation noise floor.

Figure 3-19 shows the noise floor measured at the output of the DAC for instrumentation and demodulation circuitry shown in the schematic in Appendix E. The noise floor of the oscilloscope is substantially lower than the quantization noise of the demodulation circuitry. However, the noise floor of the spectrum analyzer is well

above the noise floor of the demodulation circuitry at the carrier frequency.

## 3.8 Future Work

Although the current experimental setup shows promise, there are some areas that could be altered to improve the operation and noise level of the device. These can be grouped into a few main categories, including demodulation board improvements, DSP software changes, and inductive link improvements.

### 3.8.1 Inductive Link Improvements

More work needs to be done to understand the steel door frequency dependent properties. As mentioned before, the door relationship between magnetic permeability and conductivity is complicated and the current operating frequency is likely not the optimum one.

One experiment might involve making a reference coil setup with a very high resonant point. Parallel capacitors could be added to the inner coil to shift the resonant point down. At each resonant point the outer coil could be matched with an appropriate sense impedance and connected to the demodulator. Signal levels could be measured at each point and compared to find the optimum door frequency.

There are multiple related problems with this approach. First, lower frequencies will result in lower coil impedance. This results in higher current requirements. This increases the power demand of the system and might overdrive the thin magnet wire used in the coils. Operating at a lower frequency requires placing more windings on the primary coil to increase its impedance and lower the current draw. This in turn reduces the turns ratio discussed previously and might harm overall functioning of the system.

Another more complex experiment could be created to find the appropriate natural door frequency. If an ideal wideband Hall Effect magnetic field sensor were available, it could directly detect magnetic field strength inside the door. A coil could be placed on the outside and swept across frequency to find the maximum field inside the door.

Next, an inner coil could be designed to self-resonate at this frequency. Finally an outer coil could be designed to present an impedance at this frequency that is suitable for driving the system. A matched  $Z_{sense}$  could then be placed in series with it.

The inductive link also needs to be rebuilt to be mechanically more stable and physically thin enough to fit inside the door. In addition, it could be made broader to improve coupling as discussed earlier.

If these changes drastically alter the coils as they appear to the JFET mixer, different JFET devices with higher  $\beta$  and higher current might be beneficial.

### 3.8.2 Demodulation Board Improvements

The analog processing chain of the demodulation board works, but there are numerous ways that it could be improved. The current board is a first revision, so an exhaustive list of problems might include:

- Add level shifting circuitry. This is currently present on a solderless breadboard and should be copied to the printed circuit board. Both I and Q channels need this module between the gain stage and the anti-aliasing filter at the input to the DSP ADCs.
- Substitute OP97 for LT1028 op amp for lower input bias current.
- Change  $10\ \Omega$  R48 to  $200\ \Omega$  to limit shoot-through current on analog switches. Without these resistors feedthrough at the carrier frequency is injected every time the switches change state.
- Add a similar resistor to the input to the other analog switch so each channel is balanced.
- Add a similar resistor to the ground connection of each analog switch to balance them. When balanced the operational amplifier driver does not see discontinuous impedance even during switching. This should completely suppress the feedthrough.

- Change the highpass filter at C39 and R42 to  $1\ \mu\text{F}$  and  $15\text{k}\Omega$  to reduce input bias current effects. The LT1028 has relatively large bias current that is currently drawn across the  $150\text{k}$  resistor. This leads to substantial offset voltage that is then gained.
- Ensure that the other input to that op amp is matched in impedance. This is less important, but it ensures that input bias current effects are canceled because the bias current on each channel is drawn across the same impedance values.
- Make similar changes to C45 and R43.
- Change the resistors at R44, R45, R46, R47 to have gain of 1000. This is at the main gain stage of the filter chain after the carrier has been filtered away and the modulated signal needs to be recovered. This should fix the ADC input range properly so that the full dynamic range is utilized.
- Correct the footprints for the BNC connectors, U2 and C6. The ground holes are too small.
- Add traces from the raw I and Q channel output to the DSP before highpass filtering. As described earlier, this permits the software to find a sign reference. This signal would need buffering to ensure that the ADCs on the DSP do not disturb the carrier. It will need level shifting. Another possibility is to locate an analog comparator and set one reference to ground. Assuming its output is digital, it could be connected directly to the DSP and not require more analog-to-digital conversion.
- Separate the ground plane into three parts, a high power part for coil drive current, an analog part, and a digital part.
- Consider a larger input range. The current system is powered from bipolar 15 Volt power supplies, but the incoming signal is only a Volt peak to peak. Changing the signal path to tolerate this range would be advantageous, and it

would likely only require changes to the AD620 instrumentation amplifier gains. Since this is set by a single resistor, a few different ones could be ordered and experimented with, though a full scale input signal should only require a gain of 1 at that point. In the current system that gain was increased to 10 because of the small amplitude input signal.

- Physically separate the coil drive circuitry from the signal chain input.
- Use a crystal oscillator to drive the DSP and determine if it is stable enough for usage as a carrier. The current board has space for an oscillator to the DSP, but it is not installed. The DSP on board RC oscillator and PLL have enough frequency jitter that the DSP cannot directly provide a drive frequency for the analog filter chain and coil drive circuitry. A crystal oscillator might fix this. The other option might be to purchase a powered oscillator to drive the coils at fixed frequency directly.
- Add silkscreen indicators showing the pinouts of the various connectors. Some of them are missing.
- Add silkscreen indicators for the polarity of the polarized capacitors and diodes. This makes assembly easier.
- The “5k” resistors marked on the silkscreen are actually 6.28k. The silkscreen should be changed because 5k is not a standard size.
- Shrink the board by a few square inches to qualify for 4PCB’s specials.

Another possibility to investigate is demodulation to an intermediate frequency with analog circuitry followed by digital demodulation to baseband. This is difficult and requires the use of a notch filter capable of removing the carrier while leaving 60 Hz data undisturbed. However, this method removes the need for a highpass filter. This is beneficial because of superior settling time of notch filters compared to highpass ones. Space was left on the board for this idea, but it should be removed if the idea is not pursued.

### 3.8.3 DSP Software Improvements

Currently there is no filtering done in the digital domain on the DSP. Experiments have shown that this system distorts incoming waveforms. Higher harmonics of 60 Hz pass through the system with a larger amplitude and different phase. This would make composite waveforms with more than one frequency look very different. The DSP could be improved to add a filter stage to help invert this characteristic. First the properties of the door system would be measured experimentally to aid in this effort. If the filter is complicated, DSP assembly language could be used to increase speed.

An important change involves the sign uncertainty of this system. The current method of arbitrarily choosing the I channel as a sign reference works, but it is not a general solution, and a more complex software scheme could reduce sign uncertainty. The idea is to have access to the raw I and Q channels after multiplication and lowpassing but before highpassing.

# Chapter 4

## Conclusions

This thesis presented multiple improvements to the experimental NILM system. With these improvements NILM should be able to realize higher resolution leading to better load discrimination. The new current sensor will also allow for simple installation in any system that has a standard circuit breaker.

The NerdJack device has been presented to be of lower cost and of higher accuracy than the LabJack upon which it is based. Some of these devices have been built and deployed in experimental NILM installations. Other researchers are working to understand the features that can be resolved with the new device.

Because NerdJack's channels are simultaneously sampled, acquiring more channels with it does not inherently reduce the sampling rate and alignment of other channels. The increased number of channels available in this way have led to additional NILM developments. The system will soon be outfitted with channels that have 60 Hz notch filters. This allows the primary to be suppressed so that the signal can be gained. This allows the higher harmonics of 60 Hz to be visible to the instrumentation and the NILM algorithms. The implications of this additional information are still being explored.

This thesis has also presented a novel through-door current sensor. This device can successfully measure 100 mA signals inside the breaker and deliver this information through a solid steel plate using approximately 7.5 Watts of power. Possible improvements have been presented that should increase the sensitivity of the device,

decrease the power requirements, or both.

Energy monitoring by device is an important means of reducing energy usage. Earlier methods required sensors at each device to be monitored. This device shows promise as a single device that is easily installed and can monitor an entire power distribution system. Further work should make the system smaller and cheaper while giving it a more polished appearance. A final system should ideally incorporate NILM technology with the sensor and an appropriate interface for power monitoring. The system could be outfitted with WiFi to deliver power usage data in realtime to a specialized display or to a general purpose personal computer program.



# Appendix A

## Microcontroller-based educational tool

One spin-off project of the digital data acquisition device was an educational tool called “BurnIt”. One of the popular laboratory classes at MIT helps students build microcontroller-based digital and analog systems using a loaned laboratory kit and single board computer. Unfortunately, the equipment required to program microcontroller systems has a substantial up-front cost. This erects an unnecessary barrier to experimentation both during and after taking the class. Using a similar chip to the one used in the data acquisition system, a programming device inexpensive enough to give away to graduates of the class was built. This device allows students to continue learning about microcontroller systems after the class by actually building them.

### A.1 BurnIt theory of operation

BurnIt is meant to program the various microcontrollers used in MIT’s 6.115 microcontroller laboratory. These include the Lattice GAL22V10, the Microchip PIC16F627-8 series, and the Atmel AT89C2051. The class mostly uses an Intel 8051 based single board computer, so the 2051 was chosen to be familiar to the students. The class also includes a module on the language C. The PIC microcontroller is programmed in that language and is included for that purpose. The class briefly covers programmable

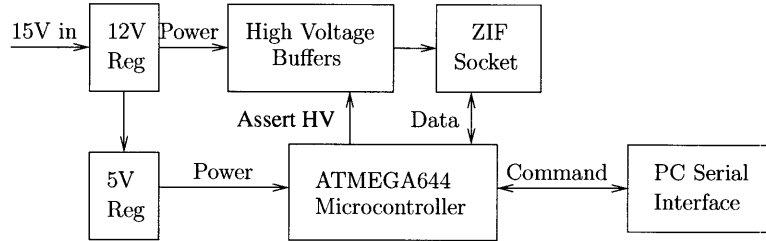


Figure A-1: BurnIt block diagram

logic devices, so the 22V10 serves as an introduction to that. These chips were chosen because the compiling tools are freely available, they serve a purpose in the class, and the programming algorithms are either published by the company or otherwise widely known.

The main components of BurnIt are an Atmel Atmega644 microcontroller, a serial port, and a zero insertion force (ZIF) socket. This AVR-based device is pre-programmed with firmware to interact with both the serial port and a chip inserted in the ZIF socket. The chip to be programmed is placed into the ZIF socket. When powered on, the Atmega firmware checks the state of a DP3T three position switch to decide which chip is present in the ZIF socket. The Atmega then uses the serial port to interact with a computer running a terminal emulator program. The user can then interact with the device through terminal emulation to download code to BurnIt or read code from the inserted chip. BurnIt presents a simple menu to the user tailored to the chip in the ZIF socket. The user issues single letter commands to the device to control it. The object files for all three chips are plaintext suitable for transmission over a serial port using standard file sending mechanisms.

“BurnIt” needed to be easy to build, easy to use, and useful at the same time. It also needed to be robust against student error. This led to a design involving a handful of components soldered to a small two layer printed circuit board. All parts were through-hole parts to make them easy to install for first time solderers. The device can be powered using an inexpensive generic power supply because it includes its own power regulators. The remaining pieces of BurnIt allow it to assert the high voltages required for programming various chips. They also do the proper voltage

level shifting to interact with a PC serial port.

The schematics and layout for BurnIt are included in Appendix D. The firmware for BurnIt is present in Appendix G.

## **A.2 Programming the AT89C2051**

The most important chip programmable by BurnIt is the 2051 because of its usefulness to students. This chip is a 5 Volt part, but it requires 12 Volts to be asserted to a specific pin to start its programming mode. At some points in the programming algorithm this pin also needs to have 5V asserted on it. Because the Atmega is incapable of supplying this high voltage, some extra circuitry was required to assert both 12V, 5V, and 0V while still protecting the Atmega I/O pins. The final solution was an open collector buffer with two diodes and a pullup resistor.

The 2051 requires parallel programming, so most of the Atmega digital I/O lines must be connected to the lines of the 2051. Effectively, all 8 bits of every byte are asserted on every clock cycle to the device and bytes are transmitted sequentially.

## **A.3 Programming the PIC16F628**

This chip is substantially easier to program because it was designed for in-system programming. Only four wires are actually required to program the device, and data is clocked into it serially. There is no requirement for high voltage to start programming mode. This section of the code was borrowed heavily from jimpic by Jim Paris. Jimpic was the original method for programming the chips available to students in the class.

## **A.4 Programming the GAL22V10**

The programming algorithm for the GAL22V10 was not available from the manufacturer, but other hobbyists had reverse engineered the algorithm. BurnIt uses the

ideas from GALBlast to program the chip. This chip is difficult to program because it requires variable timing and high voltage.

# Appendix B

## Data Acquisition Device Manual

### B.1 Theory of Operation

The NerdJack device is an Atmel AVR32UC3A0512-based microcontroller system integrating Ethernet capabilities with Analog Devices AD7656 analog to digital converters. It is responsible for digitizing analog signals and transmitting them to a waiting computer system for analysis and processing. It is capable of delivering 16 bit samples of bipolar  $\pm 10$  or  $\pm 5$  volt signals. Channels number from 0-11, and each half (0-5 and 6-11) can have their range individually configured. The data can be sampled up to 100 kHz, with the maximum throughput depending on the number of channels sampled and the network conditions. There are software warnings in place if too high of a rate is attempted.

### B.2 Installing software to use a NerdJack

The software requirements for NerdJack are very simple. Software for programming the device is somewhat more involved, but it should not be necessary for basic Nerd-Jack usage.

All software for NerdJack is available in the LEES Subversion repository at:

<https://bucket.mit.edu/svn/nilm/acquisition/nerdjack>

Throughout this document files will be referenced based on where they are located

in the repository directory structure. However, a checkout of the repository from 8-27-09 is present in `/home/zacharyc/nerdjack/` on `bucket.mit.edu`. This directory is accessible to any user of bucket.

### B.2.1 Windows

There is a file called “`ethstream.exe`” located under `./installer/utils`. This file should be copied to the Windows machine. A decent location might be

```
C:\Program Files\NerdJack\
```

Another decent place might be the same directory the older `ljstream` was installed under. The file should then be added to the `PATH` of the machine. From the Windows Control Panel, open `System Properties`. Select the `Advanced` tab and click the `Environment Variables` button. In the lower box of `System Environment Variables`, edit `PATH`. Add the path to your `ethstream.exe` to the beginning of the `PATH` followed by a semicolon like so:

```
C:\Program Files\NerdJack;rest of path
```

Now you can open a command prompt by selecting `Run...` from the Start menu. Type “`cmd`” in the box and press `OK`. Now you can type “`ethstream`” to connect to the NerdJack.

An alternative to the above instructions involves running the installer from:

```
./installer/installNerdJack.exe
```

This file installs programming tools and `ethstream` automatically.

### B.2.2 Mac OS X and Linux

Binaries are available for these platforms, but you are likely to have more luck compiling `Ethstream` from source. Linux generally has `gcc` and `make` installed already. Under OS X you should install the developer tools from the installation DVD.

The code is located in `./ethstream` in the repository. It should be copied to the computer.

Type the following command in the main directory of `ethstream`:

```
make && sudo make install
```

This will install the program to the system path in `/usr/local/bin`

### B.3 Using the NerdJack

After software installation, all necessary utilities should be present on the system. Turn on the NerdJack with DIP switch 4 in the OFF position. The other three switches select between the seven configured IP settings and DHCP. They can be interpreted as binary numbers (i.e. configuration 0 is all switches off, configuration 4 is just switch 3 ON, etc.). Position 7 is DHCP. This DHCP implementation seems to work in LEES, but it may be missing some features.

After the unit is powered on and connected to the network, simply type “`ethstream -N`” followed by your desired command line options to retrieve data. Typing “`ethstream --help`” explains the options. Typing “`ethstream -X`” gives example usage. The output from `ethstream` can be piped to other programs using standard stream redirections. The data is emitted as space separated numbers with each sample on a separate line.

`Ethstream` is almost fully compatible with the older `ljstream` that communicated with the LabJack. The most important options are summarized in Table B.1.

### B.4 Installing software to program a NerdJack

If the user wishes to change the serial number, the TCP/IP settings, or the Ethernet MAC address of the NerdJack, programming software must be installed. This is also necessary to upgrade the firmware on NerdJack in the future.

Short Option	Long Option	Description
-a	-address string	host/address of UE9 (192.168.1.209)
-n	-numchannels n	sample the first N ADC channels (2)
-d	-detect	Detect NerdJack IP address
-R	-range a,b	Set range on NerdJack for channels 0-5,6-11 to either 5 or 10 (10,10)
-C	-channels a,b,c	sample channels a, b, and c
-r	-rate hz	sample each channel at this rate (8000.0)
-c	-convert	convert output to volts
-H	-converthex	convert output to hex
-l	-lines num	if set, output this many lines and quit

Table B.1: Command line arguments to Ethstream

### B.4.1 Windows

There is a Windows installer in the subversion repository to install NerdJack programming tools located at `./installer/installNerdJack.exe`

Run this installer, and it will automatically install the binary image, the programming tools, and the USB drivers.

Now “nerdconfig” should be in your system path and usable from the command line.

### B.4.2 Mac OS X and Linux

Because of the wide variety of Unix systems, it will probably be necessary to install the software from source, though some binary packages can be made available. In order to use the tools, both Python 2.6 and libusb 0.1.12 must be installed.

Download those packages and follow the appropriate installation instructions. They can probably be installed from the package manager of the Linux distribution. After those libraries are installed, `dfu-programmer` must be installed. The latest release from the repository should be used. It is located in `./dfu-programmer/`. Newer releases from upstream might fix problems. The current version works with almost all firmware installs, but an off-by-one error in it is unable to program certain FLASH locations according to a list of fixes applied to the code upstream. The newer version



claims to fix this but has been untested for lack of a good failing test case on the current version.

Copy the source distribution to the local machine. Simply navigate to its directory and type, `./configure && make && make install`

Next the `nerdconfig` tools should be installed. It is a standard Python distutils package in `./nerdconfig/`

It should be installed by typing `sudo python setup.py install` from the package directory.

Assuming your system paths are configured appropriately, `nerdconfig.py` should be in your path. Occasionally the Python `scripts` path is not included in the system path, so it should be added if necessary.

## B.5 Programming a NerdJack

Programming a NerdJack requires a power supply and a USB cable. It can be any standard USB to mini-USB cable for the purposes of programming. A cable from a digital camera works well. The fourth DIP switch on the NerdJack should be flipped to its programming position (ON).

Connect the NerdJack to the PC and power it on. The operating system should detect the device and may give some notification. On Windows open a command prompt and type `nerdconfig`. Unix users should use `nerdconfig.py`. This utility will detect the NerdJack and print out its current TCP/IP settings, serial number, and MAC address. If the NerdJack is blank, it will configure the NerdJack with standard IP settings, a random MAC address, and the stock NerdJack firmware. Typing `nerdconfig --help` should print a list of options that `nerdconfig` understands. `nerdconfig -R` should print the firmware revision on the attached NerdJack.

After programming the device or changing its settings, be sure to flip the programming DIP switch back to the OFF position before use.

## B.6 Building a NerdJack

In the subversion repository there is a section for NerdJack schematics in:

```
./schematics/NerdJackv4/
```

These have been processed and configured for use with 4PCB's board manufacturing service. The board is a small two-layer PCB. In the same directory there is a bill of materials with Digi-Key part numbers for all parts that can be bought there. Some pieces (in particular the Ethernet jack), are available from Mouser. The silkscreen on the board and the part designators in the BOM should be sufficient for building the device. A letter is enclosed in `./documents/` suitable for delivering to Proxy Manufacturing.

## B.7 Pinouts for NerdJack

The NerdJack device has a few connectors with important pinouts. The main one is the Molex power connector. The silkscreen labels the four pins as `-12 Volts`, `+12 Volts`, `+5 Volts`, and `GND`. This is the pin order from left to right.

On one edge of the board is the prototyping connector for connecting the analog channels. This section can be populated either with header or two-level screw terminals. If populated with screw tabs, the upper level tabs are all shorted together and connected to the analog ground plane. The lower 12 tabs are connected to the analog channels 0 to 11 on the NerdJack. They are sequential starting from the side marked `CH0`. If populated with header, the back pins are connected to `AGND`, and the front pins are connected to the channels in sequence from `CH0`. In effect, they are oriented identically to the screw tab connector.

On the top edge of the board there is a DB15 and a DB37 connector. The pins are numbered in the standard way for DB connectors with Pin 1 for each connector closest to the left side of the board toward the Ethernet connector. On DB connectors the pins are numbered sequentially from there on the wider part of the connector. Numbering then continues from the same side as pin one. For example, the two

Pin	Label	AVR32 pin
1	3.3 Volts Output	N/A
4	CS1	PA14
5	MOSI1	PA16
6	TC_A1	PB25
8	GND	N/A
11	GND	N/A
12	SCK1	PA15
13	MISO1	PA17
14	TC_B1	PB26

Table B.2: DB15 table pinout

leftmost pins on the DB15 connector are 1 and 8, while the rightmost ones are 8 and 15.

Unmentioned pins are unconnected. The DB15 connector breaks out SPI port 1 from the AVR32 and the Timer Counter pins A1 and B1. The table mentions the function and the AVR32 name of each pin. The pinout for the DB15 connector is given in B.7.

The DB37 connector breaks out a USART serial port and the analog channels. Its pinout is given in Table B.7.

The last connector is the JTAG connector between the two DB connectors. This is meant to mate with the JTAG ICE MKII programming device from Atmel. The pin numbering is labeled on the board and does match the programming tool. Because of the USB bootloader, this should not be required unless debugging operations are needed in the future.

## B.8 Device Overview

The NerdJack device consists of both hardware and firmware components. From a hardware perspective, it has two AD7656 analog to digital converters connected to the SPI bus of the AT32UC3A0512 microcontroller. These devices are controlled through various control signals supplied by the microcontroller, and they deliver data via their SPI port. The microcontroller has a 256 Mbit SDRAM peripheral to allow it to buffer

Pin	Label	AVR32 pin
1	GND	N/A
2	USART_TXD	PA1
8	GND	N/A
10	GND	N/A
13	CH11	N/A
14	CH9	N/A
15	CH7	N/A
16	CH5	N/A
17	CH3	N/A
18	CH1	N/A
19	GND	N/A
20	USART_RXD	PA0
27	3.3 Volts Out	N/A
30	Analog GND	N/A
32	CH10	N/A
33	CH8	N/A
34	CH6	N/A
35	CH4	N/A
36	CH2	N/A
37	CH0	N/A

Table B.3: DB37 connector pinout

samples in the event of network congestion. Whenever it is unable to send data, it will continue sampling into its buffer until the connection is formally terminated or the buffer is overflowed. The device also has an Ethernet subsystem to deliver data. This port can be connected to any LAN or directly to a computer using standard TCP/IP. It's IP settings are configurable using DIP switches on the board, and the meaning of those DIP switches can be configured during firmware programming. Finally, it exposes a JTAG port and USB port for firmware installation.

The firmware uses a custom combination of lwIP 1.3, a lightweight TCP/IP stack, and FreeRTOS 5.0.4, a free realtime operating system. The application itself makes use of interrupts to sample the ADCs, pull data off of them, and deliver data to the Ethernet port. FreeRTOS helps ensure that the system meets realtime constraints. The software itself is built on Atmel's Software Framework 1.4.0 and relies on it for drivers that access the hardware. The entire package is compiled using Atmel's customized version of the GNU GCC toolchain and is written in C using Newlib.

The computer software component of the NerdJack consists of the utilities to retrieve data from the NerdJack and install its firmware. The primary method for installing this firmware is through the USB port. There is a Python 2.6 program "nerdconfig" that is responsible for programming the device and altering its IP configuration. Internally it uses `libusb` in conjunction with `dfu-programmer`. This utility is available for any platform supported by `libusb`. Another program called "ethstream" is responsible for communicating with the device for data acquisition. It can configure which channels to sample and the sampling parameters. It will then display the sampled data to `STDOUT` so that it can be embedded in larger data processing programs.

## B.9 Updating the Firmware

For day-to-day usage, this should not be necessary, but in the future a firmware change may be useful. To do this, a development environment must be set up.

## B.10 Building the development environment

First the necessary tools should be installed. From Atmel's home page, the GNU GCC Toolchain for AVR32 should be downloaded. They have binary distributions for a few platforms and a source distribution available. I have had trouble making some of their code compile on an unsupported platform, but the problems may be fixed in the future. Atmel provides documentation for installing those pieces. Since Linux is not used on the NerdJack, the compiler should be set up for standalone operation with Newlib as the C library. These are also available from `bucket.mit.edu` in zacharyc's home directory under `nerdjacksupport`.

A newer version of the Atmel Software Framework for the UC3A should also be downloaded if required. This will allow for newer drivers in case they are required or useful. For the most part, the framework files can be imported directly into this project assuming no API changes were made. All of the application code is under MAIN and depends on the `config.mk` file in the project root. Some fixes were applied to the MACB driver and to the SDRAM controller driver. In particular, the Framework's version of FreeRTOS is incompatible with the current version, so some changes will be necessary should a driver update be desired. The Framework also did a poor job implementing the portable layer for lwIP, so that should not be changed without good reason. The standard Makefile was also changed to allow the firmware to be versioned.

From there, a standard Makefile manages the configuration of the project, and `nerdconfig.py` can be used to program the device. `Nerdconfig.py` depends on an external Python package "intelhex," but this is included in the distribution.

## B.11 Remaking the Windows installer

This is harder than it should be. Windows is required for making the Windows installer, though it is not too difficult.

The Windows machine will need Python 2.6, NSIS Installer, and py2exe for

Python 2.6 installed.

The first step after installing the prerequisites is to have binary copies of the various executables. You need `dfu-programmer.exe`, and `ethstream.exe`. These can be cross-compiled using MinGW from a Unix platform or done in Windows with the MinGW environment. See the README files in those directories for cross-compilation suggestions.

These need to be placed in the appropriate place before beginning. The source code in subversion already has a binary copy of `libusb-win32` included. The two executables need to be placed in `utils` under the `nerdconfig` tree. The hex file of the NerdJack firmware should be placed under `extras`, and any desired changes to default IP settings should be altered in `default.csv`.

Now, navigate to the `nerdconfig` directory and run `python setup.py py2exe`. This will generate the executable and support files under `dist`. Copy this folder into the `installer` directory. Now just right click the `nerdjack.nsi` script and compile it using NSIS. This will produce a bundled installer in the directory for your usage.

## B.12 Known Issues

There are some problems with the NerdJack that need to be recognized.

First, autodetection is overly simplistic. In general, UDP broadcasts do not propagate through routers. As such it might not be delivered if the NerdJack is in a different subnet. If there are multiple NerdJacks on the local network, the first one to respond to probing will be accessed.

Next, throughput is much degraded when high channels are used. When high channels are used, the lower ones must be read first. This means that sampling channel 11 is much slower than sampling channel 0. For highest throughput, sample a continuous block of the lowest channels.

The NerdJack cannot sample at arbitrary frequencies. It uses a 66 MHz system clock. It can only sample at divisions of this clock. This means that there may be

desired frequencies that it cannot do. A warning will be displayed and NerdJack will sample the valid frequency closest to the requested one.

## B.13 Customizations to the stock Framework

A few changes were made to the stock Framework.

First, FreeRTOS was manually updated beyond the old framework version. This was done because some of the V5.0 API was very useful and required no changes to the portable layer. The version in the framework was part of the older 4.7 FreeRTOS. The main change was that some of the functions called from ISRs changed their type signatures. These have been fixed in the MACB driver, and they may need repair in the future if the framework is updated. Also, the portable layer's `portENTER_CRITICAL` macro was changed. The old implementation disabled all interrupts. However, the Atmel MACB driver sits in a “critical” region for the entire time it is sending a packet. The new implementation only disables interrupts zero to one. The MACB interrupt was demoted in priority to one. This was done because the interrupt handler responsible for initiating transfers from the ADCs is an incredibly simple routine that must be done even in a critical region. This is safe because that interrupt has no direct interaction with FreeRTOS or any running task.

Second, lwIP's portable layer was repaired. The Framework layer had a problem with a race condition that Atmel has not yet accepted upstream. When Atmel updated lwIP in the framework from 1.2 to 1.3, they did not implement the API changes to their drivers.

Finally, the stock ethernet interface driver was moved into MAIN and altered. A new version could be integrated from a newer framework. This was necessary because the framework version hardcodes a MAC address and TCP/IP settings. The new version examines the Flash User page to determine these values. It also uses the DIP switches during bootup for configuration.



## B.14 Software Overview

### B.14.1 FreeRTOS

This is a free real time operating system kernel used in the NerdJack. It allows multiple simultaneous tasks to share a processor. It provides a scheduler and priority system to ensure that the highest priority “ready” task is always running. In addition, there are queue and semaphore primitives to manage intertask communication. Because of this, adding new functionality to the NerdJack should be relatively easy as long as the priority structure of the NerdJack is not perturbed too much.

### B.14.2 lwIP

This is the LightWeight IP stack used in the NerdJack. It has a full-featured custom API as well as an interface to the standard BSD Sockets API. It can allow so-called “raw” connections, but these are not used in the NerdJack. The stack itself is highly configurable through “lwipopts.h” to control which features are compiled in as well as how its memory should be used. It is presently configured to have a lot of buffer space for sending packets.

The NerdJack program can retransmit packets independent of the lwIP stack, but it uses the error handling of the stack when possible.

### B.14.3 General Program Structure

The program has an interrupt-driven structure that can make following its flow difficult. It relies on a few different Tasks that are readied by each other and external interrupts.

During bootup a few tasks are started, including a “datastream” server, an “autodetect” server, a “command” server, a “sample manager”, a watchdog timer resetter, and the underlying lwIP stack. Interrupts are attached to the PDCA controller and the analog to digital converter ready signals. A brief description of the tasks should be helpful.

The Autodetect server is available for autodetection of the NerdJack. It listens on a specific UDP port and simply responds to any packets it receives on that port. This permits the computer software to perform a UDP broadcast on that port to find the IP address of the NerdJack. The watchdog timer resetter simply resets the watchdog timer on a regular basis. If the NerdJack crashes, this should reboot it.

The Datastream server is responsible for delivering data in tandem with the Command server. When a TCP connection is opened on the appropriate port to the command server, it expects a command word. This should tell it the range, the desired channels, and the sampling rate. It then uses this to configure the ADCs and start sampling. The command server can also rewind sampling to retransmit old packets that might have been lost. The samplemanager task is responsible for actually configuring the ADCs. This was done to centralize access to the ADCs so that different tasks could easily cooperate on them.

The ADCs are internally grouped into channel pairs that are each connected to PWM generators in the AVR32 microcontroller. These generators are configured to the desired sampling frequency by the Sample Manager task, and the desired channel groups are activated. Because these generators are independent of the CPU, they should be as accurate as the crystal clock on the NerdJack board.

The ADCs signal the end of a conversion by lowering their BUSY line. This is connected to an interrupt on the AVR32 that is connected to the DMA controller. This high priority interrupt simply starts the DMA transfer for data from the SPI port to the next available spot in RAM. DMA simply copies data from one part of memory to another without CPU intervention required. It can signal the CPU when the transfer is complete.

When the DMA controller is finished with its transfer, another interrupt is fired. This interrupt reloads the DMA channels and determines whether a full Ethernet packet of data is ready. If it is, it uses a semaphore to wake the packet maker task.

The packet maker task reads data from the scratchpad used by the DMA controller and assembles it into a packet suitable for Ethernet transmission. The semaphore used here is not a standard FreeRTOS semaphore. Instead it is a crude implementation

of a semaphore involving enabling and disabling interrupts before accessing a shared variable with the interrupt handler. This was done so that the DMA interrupt could be of higher priority than FreeRTOS. Because the Ethernet driver starves the OS of processing time during heavy load, data was being missed. This way the interrupt can be of higher priority without using the FreeRTOS mechanism for readying the packet maker task. It was also necessary to have the DMA dump to internal SRAM. The packet maker task copies data to the external SDRAM. Finally, the Datastream task copies it back. This excessive data movement was done because the external SDRAM is too slow to meet the realtime constraints of incoming data. It is fast enough for the application, but its latency is too long.

Because the channels are sampled in groups of two, it might be the case that unwanted channels were sampled. The personal computer must receive and discard unwanted data. This turned out to be more reliable than having the NerdJack selectively send data. If only sequential channels starting from zero are sampled, there is no wasted bandwidth because the data is written directly into packet form from the DMA controller.

When the datastream task receives the semaphore from the packetmaker task, it delivers the assembled packet to lwIP for transmission. This task proceeds indefinitely until the TCP connection is closed.

Another function is the serial server. The NerdJack can accept Telnet connections and transmit characters out its serial port. It can also echo received characters to the Telnet terminal. This is not presently used for anything other than demonstration purposes.



# Appendix C

## NerdJack Analog-to-Digital Converter Schematics and Layout

### C.1 Schematic

The next six pages contain the schematics for this device produced from CadSoft's EAGLE layout program.





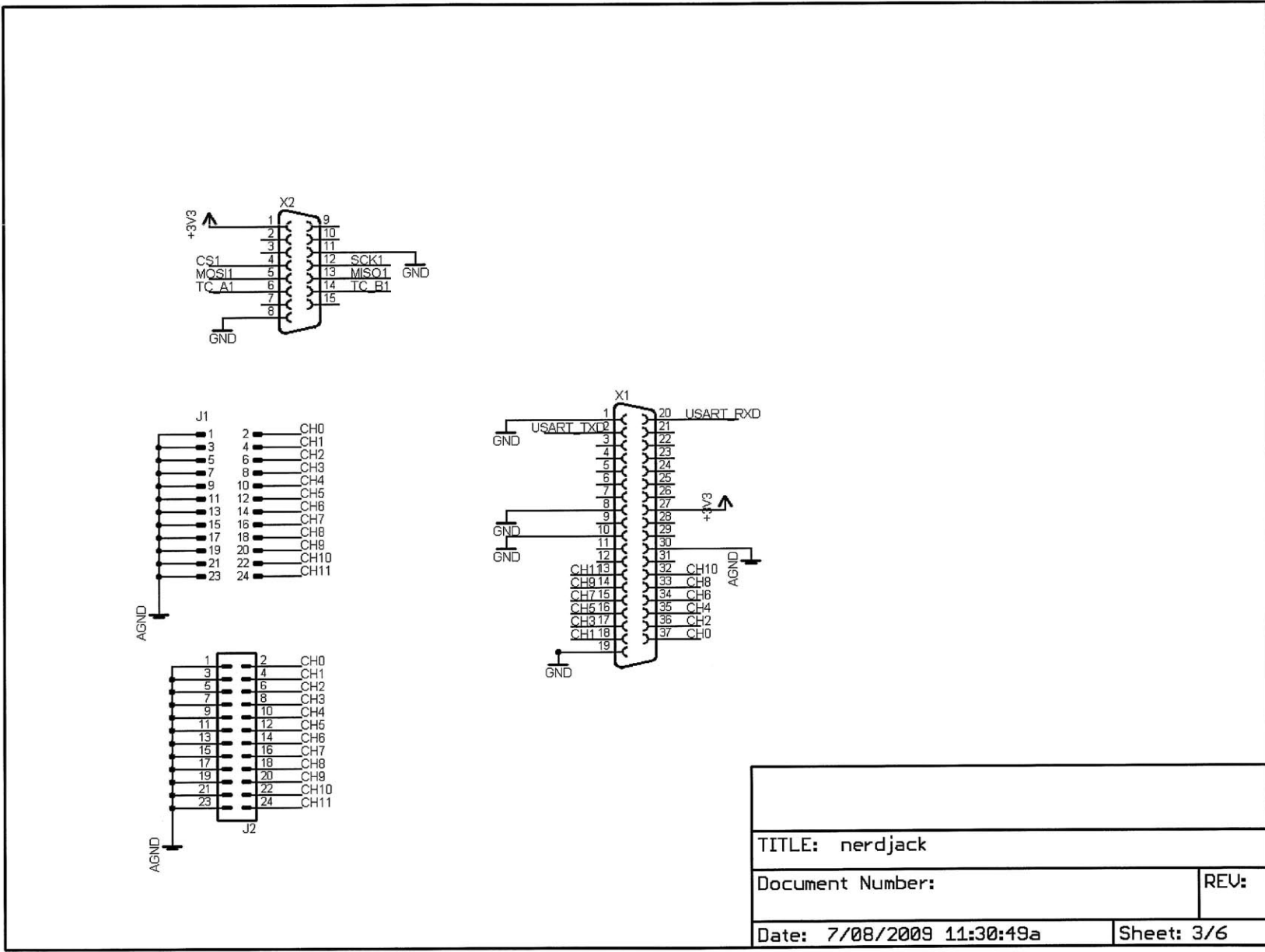


Figure C-3: External connectors





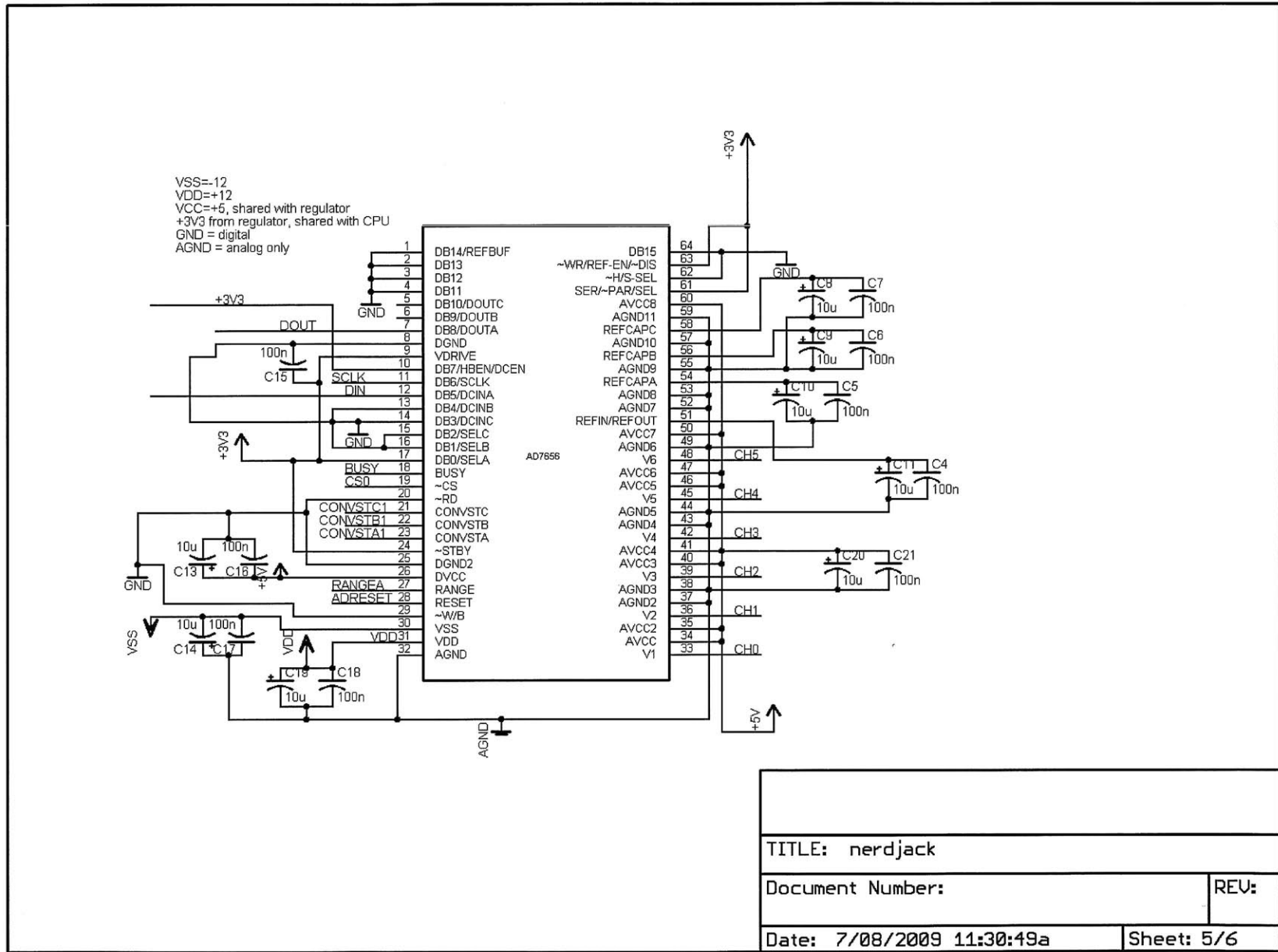
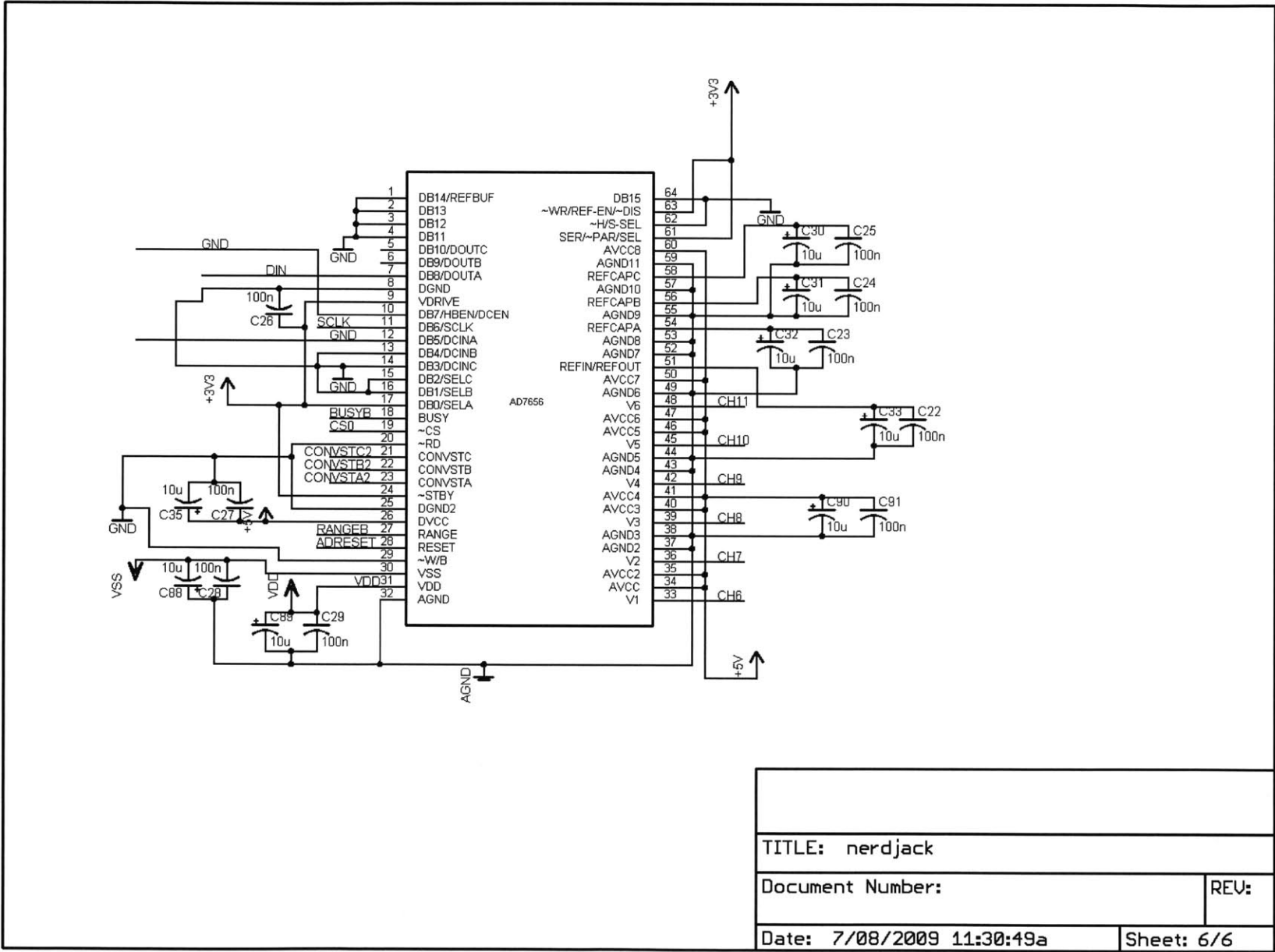


Figure C-5: First ADC



TITLE: nerdjack	
Document Number:	REV:
Date: 7/08/2009 11:30:49a	Sheet: 6/6

Figure C-6: Second ADC

## C.2 Layout

The next four pages contain images of the two layer circuit board for the data acquisition device. The copper layers omit the ground plane for clarity, and all vias are shown on all images.

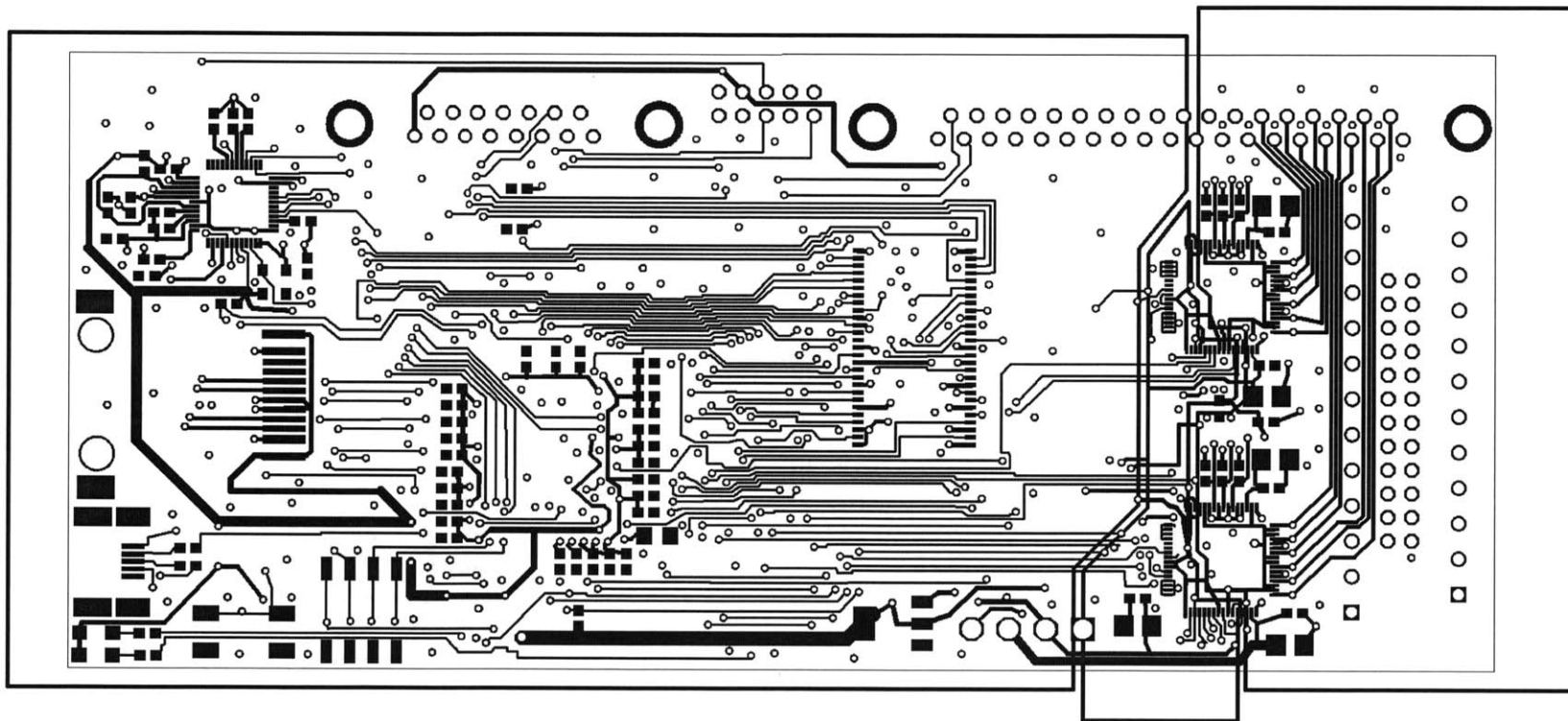


Figure C-7: Top copper layer without ground plane filled

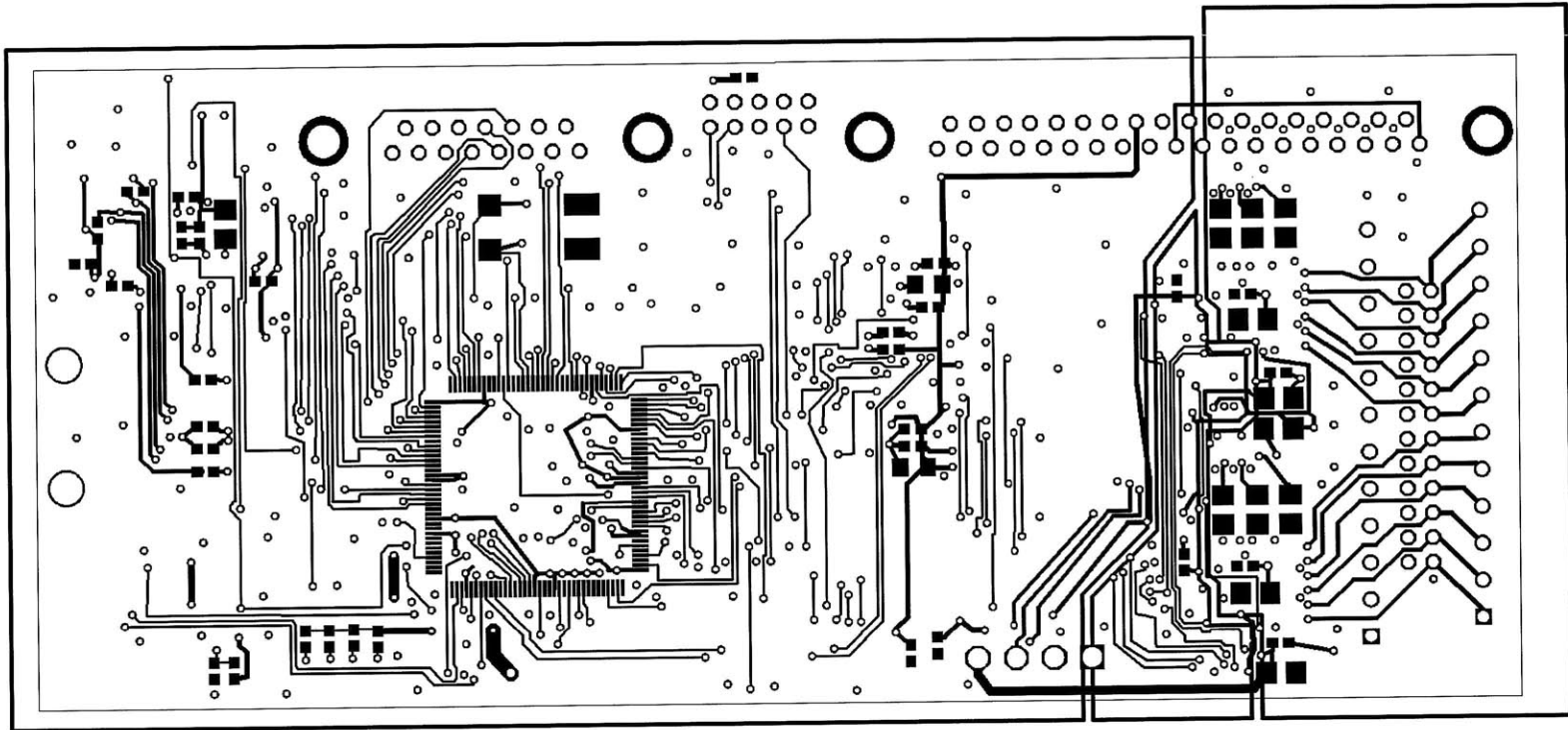


Figure C-8: Bottom copper layer without ground plane filled

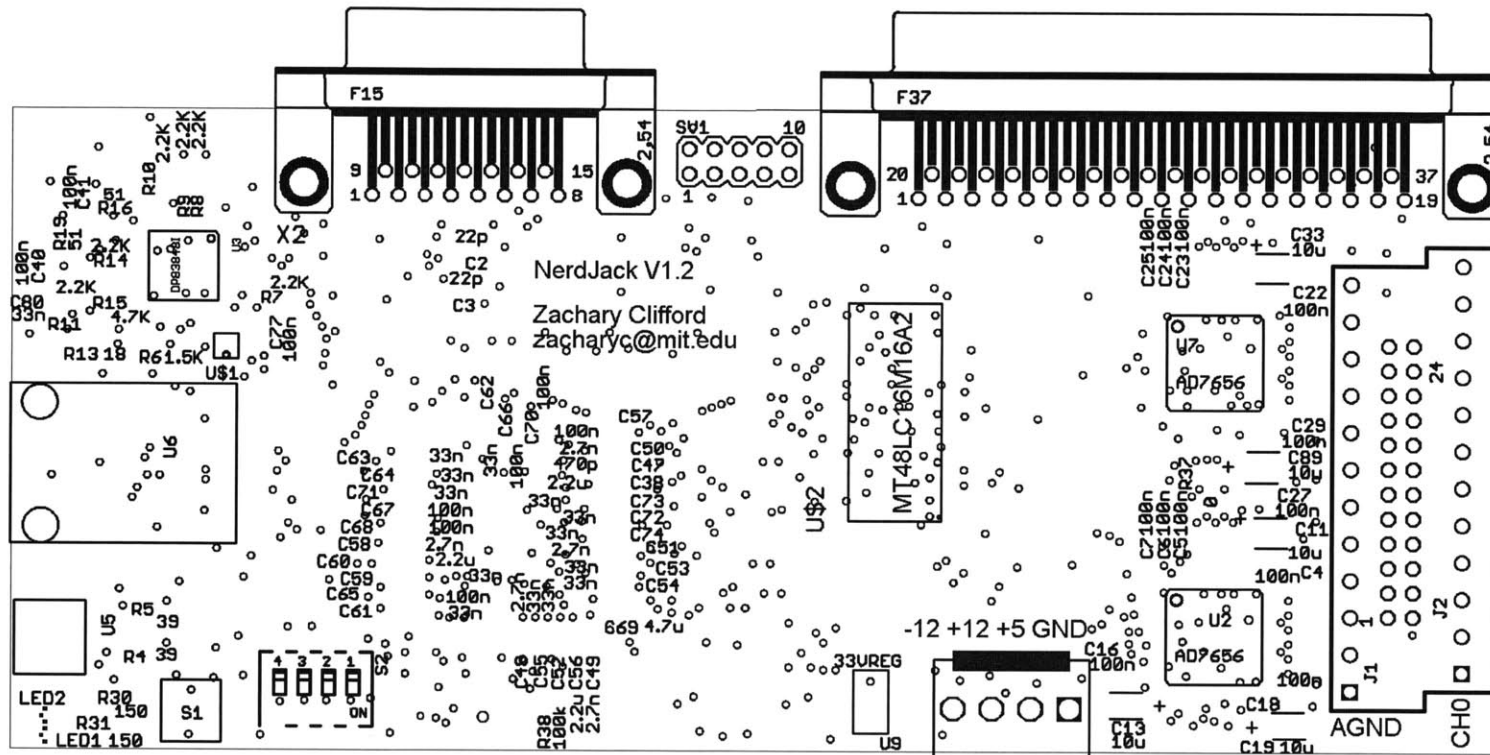


Figure C-9: Top silk layer



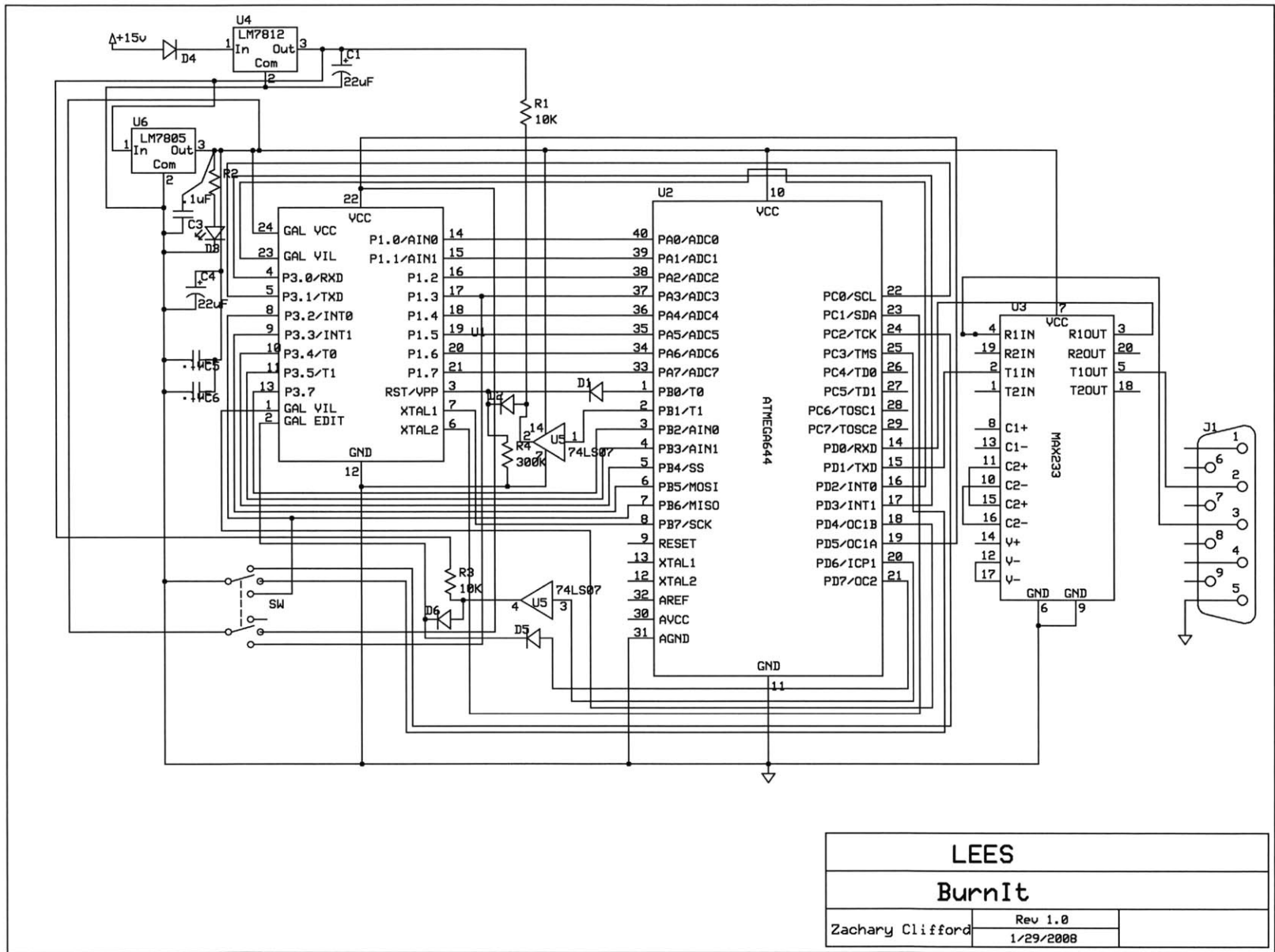


# Appendix D

## BurnIt Schematics and Layout

### D.1 Schematic

The following page shows the BurnIt schematic.

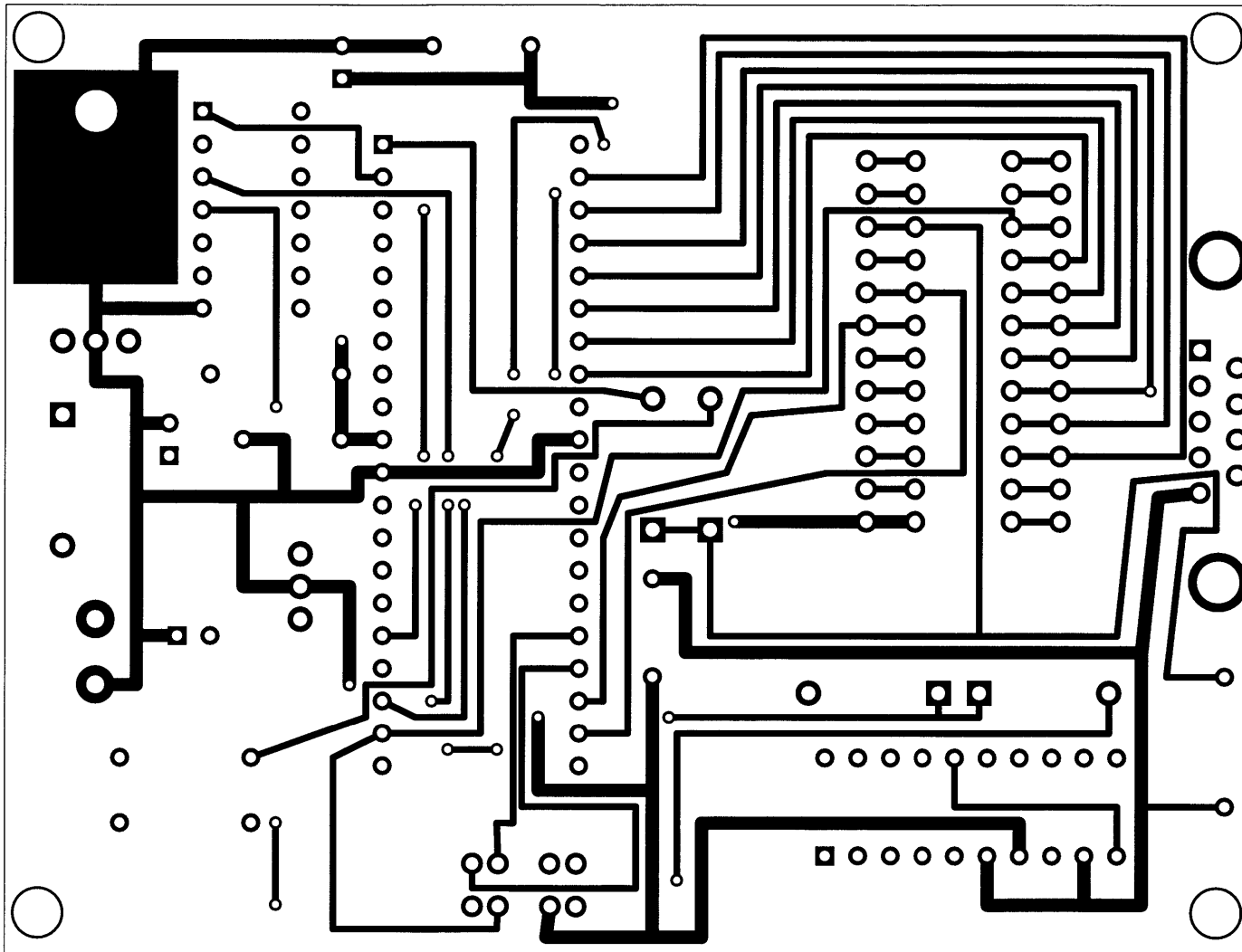


<b>LEES</b>	
<b>BurnIt</b>	
Zachary Clifford	Rev 1.0 1/29/2008

Figure D-1: BurnIt Schematic

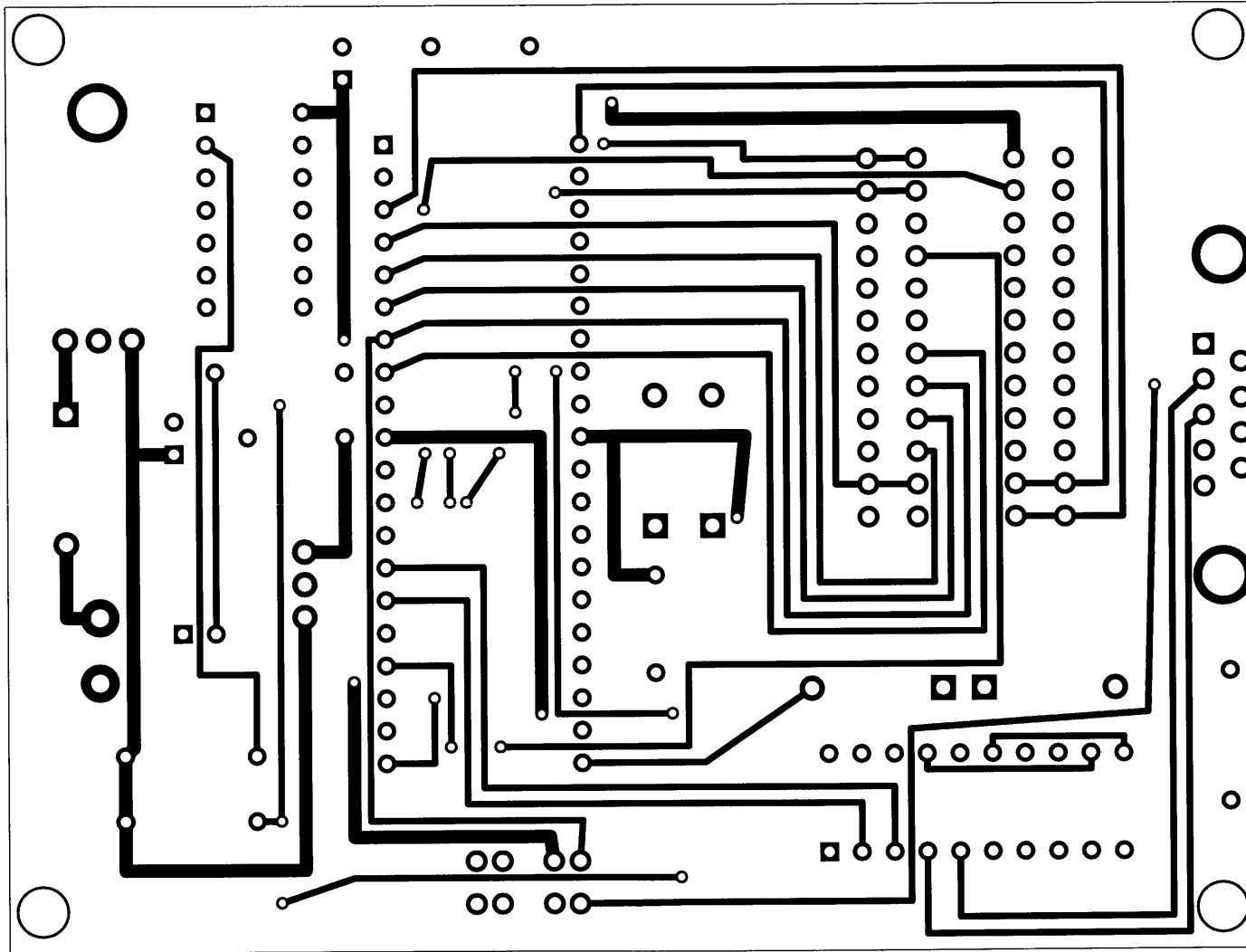
## D.2 Layout

The following pages show the BurnIt PCB layout.



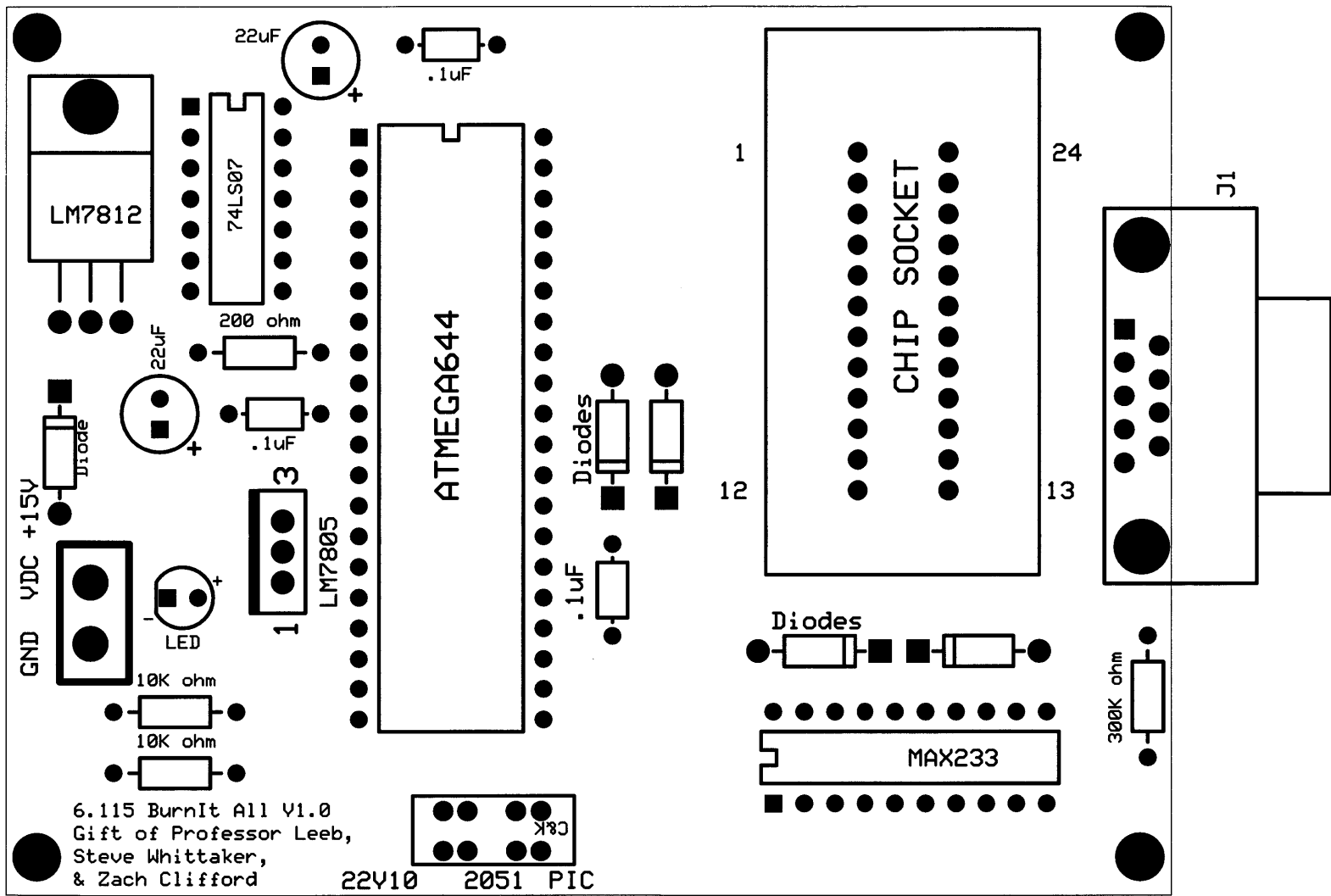
C:\Documents and Settings\Zach\Desktop\burnit.pcb (Top layer)

Figure D-2: BurnIt Top Copper



C:\Documents and Settings\Zach\Desktop\burnit.pcb (Bottom layer)

Figure D-3: BurnIt Bottom Copper



C:\Documents and Settings\Zach\Desktop\burnit.pcb (Silkscreen & pads)

Figure D-4: BurnIt Silkscreen

# Appendix E

## **IQ Demodulator Schematics and Layout**

### **E.1 Schematic**

The next three pages contain the schematics for the I/Q Demodulation circuit and power driving circuitry produced from CadSoft's EAGLE layout program.

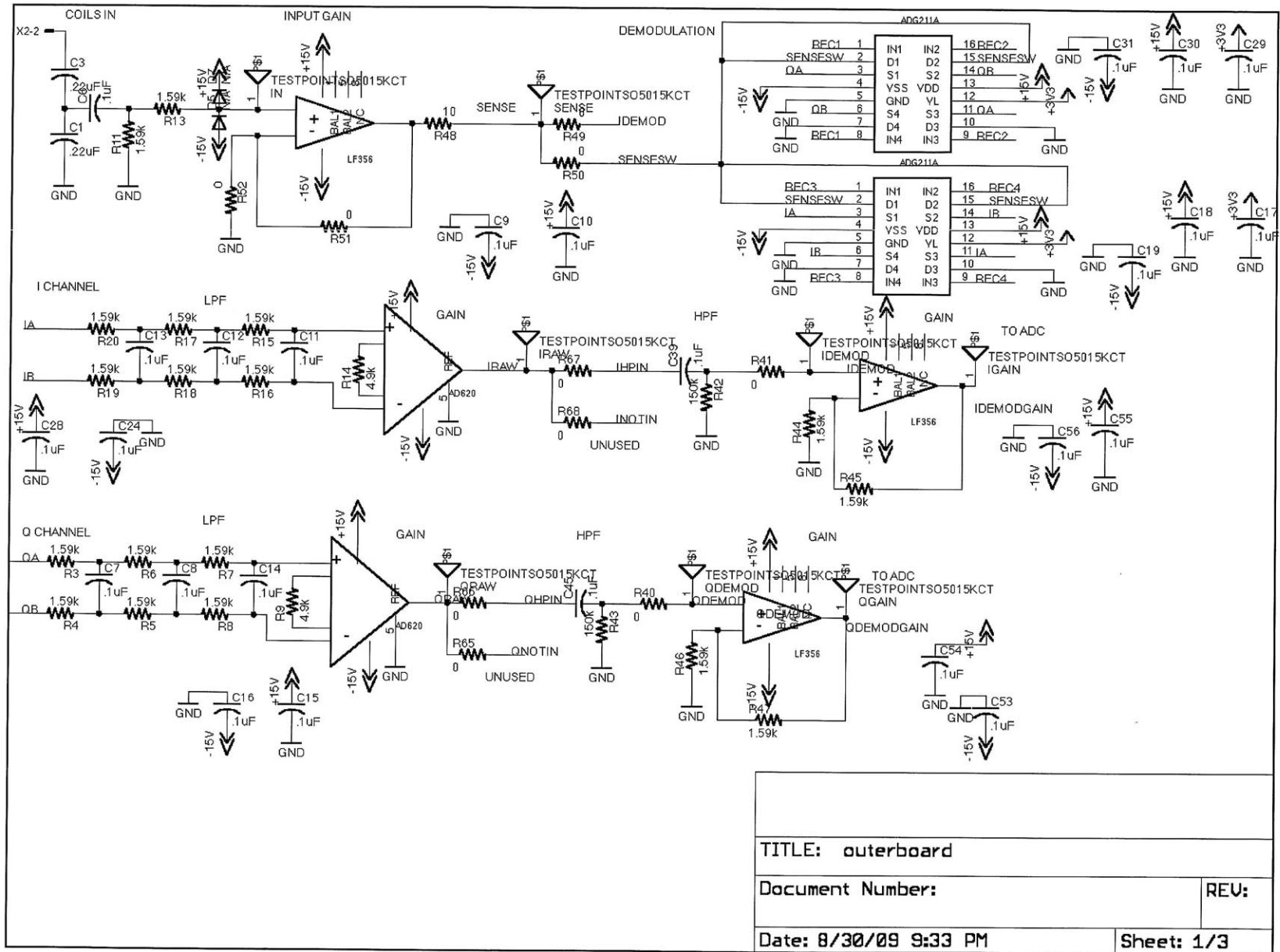


Figure E-1: The analog filter stages



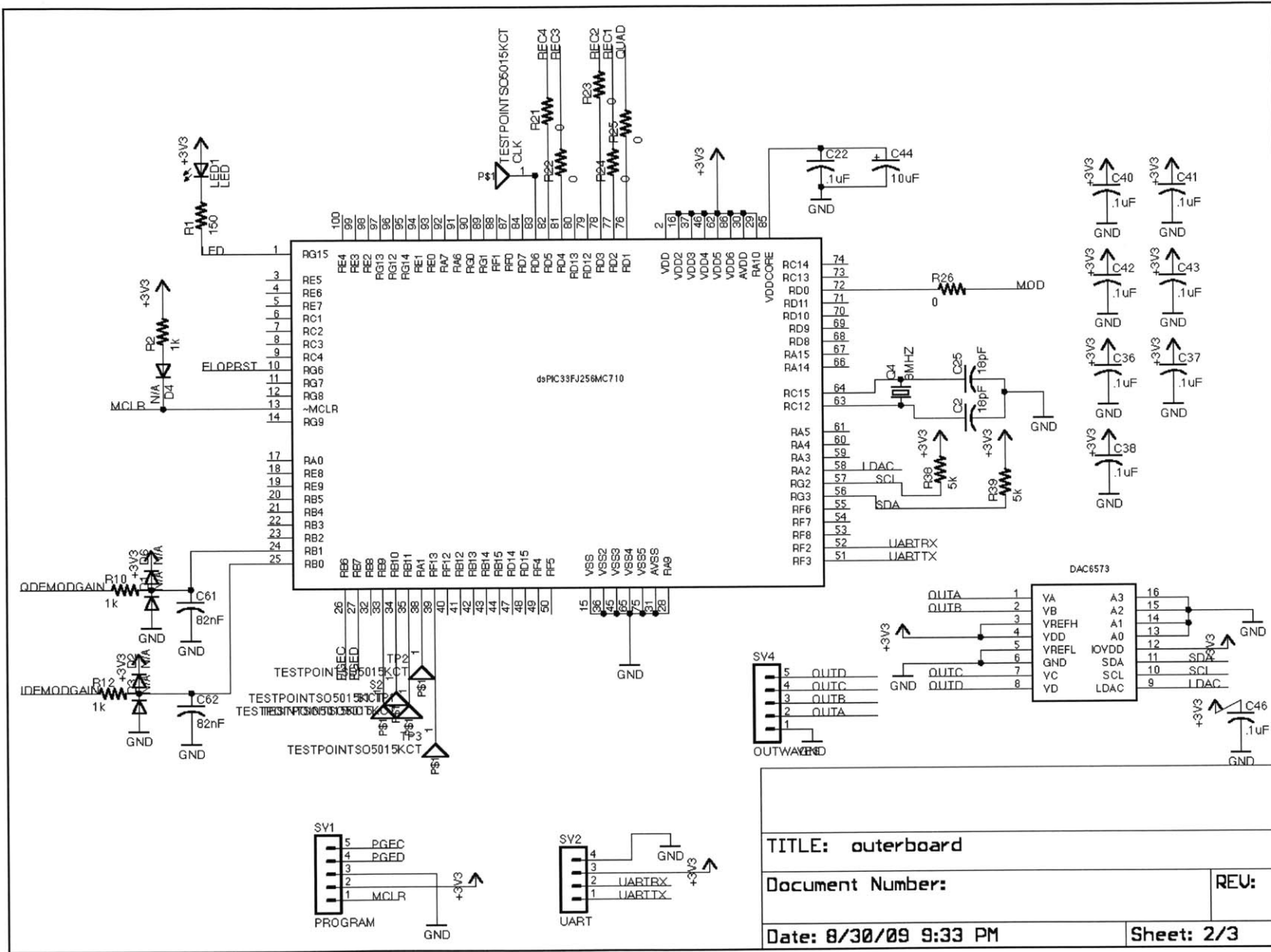


Figure E-2: DSP and supporting hardware



## E.2 Layout

The next six pages contain images of the four layer circuit board for the demodulation and drive circuit board. The copper layers omit the filled planes for clarity, and all vias are shown on all images.

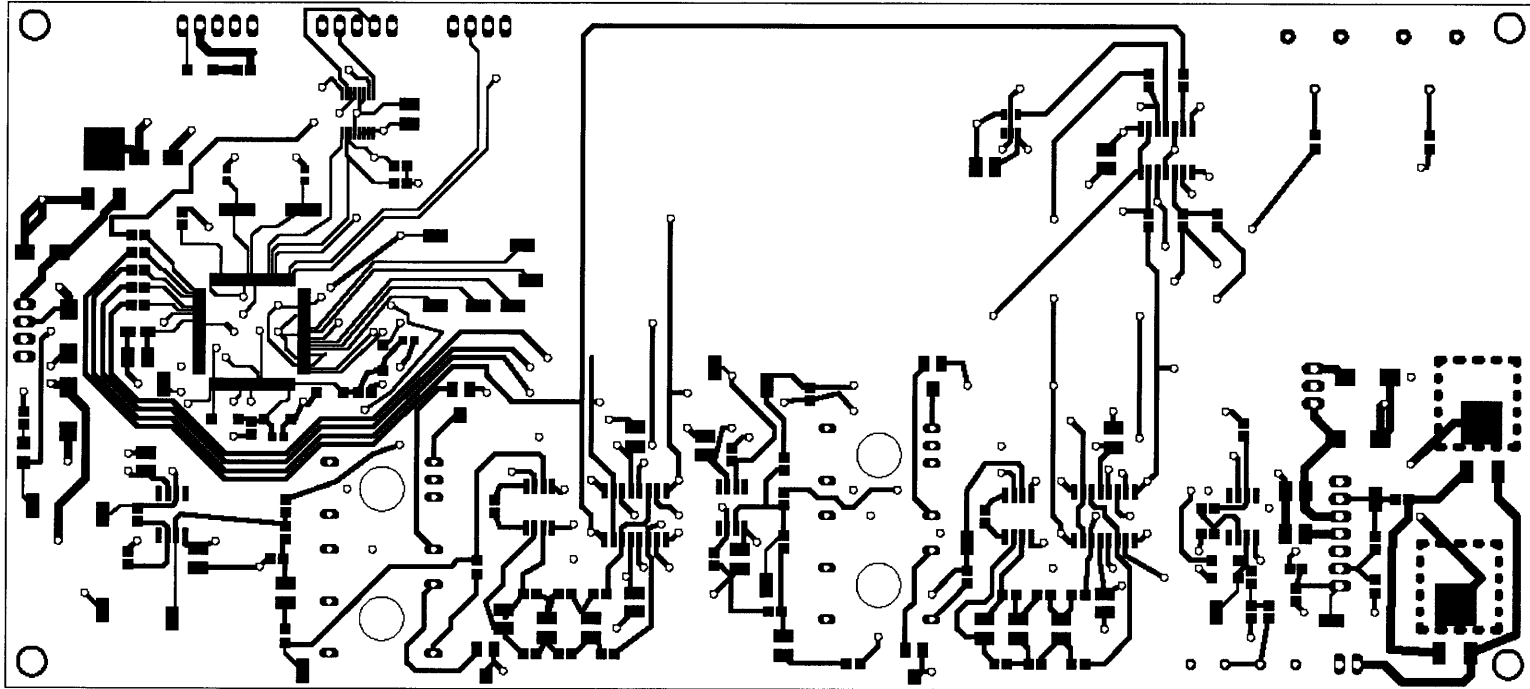


Figure E-4: Top copper layer

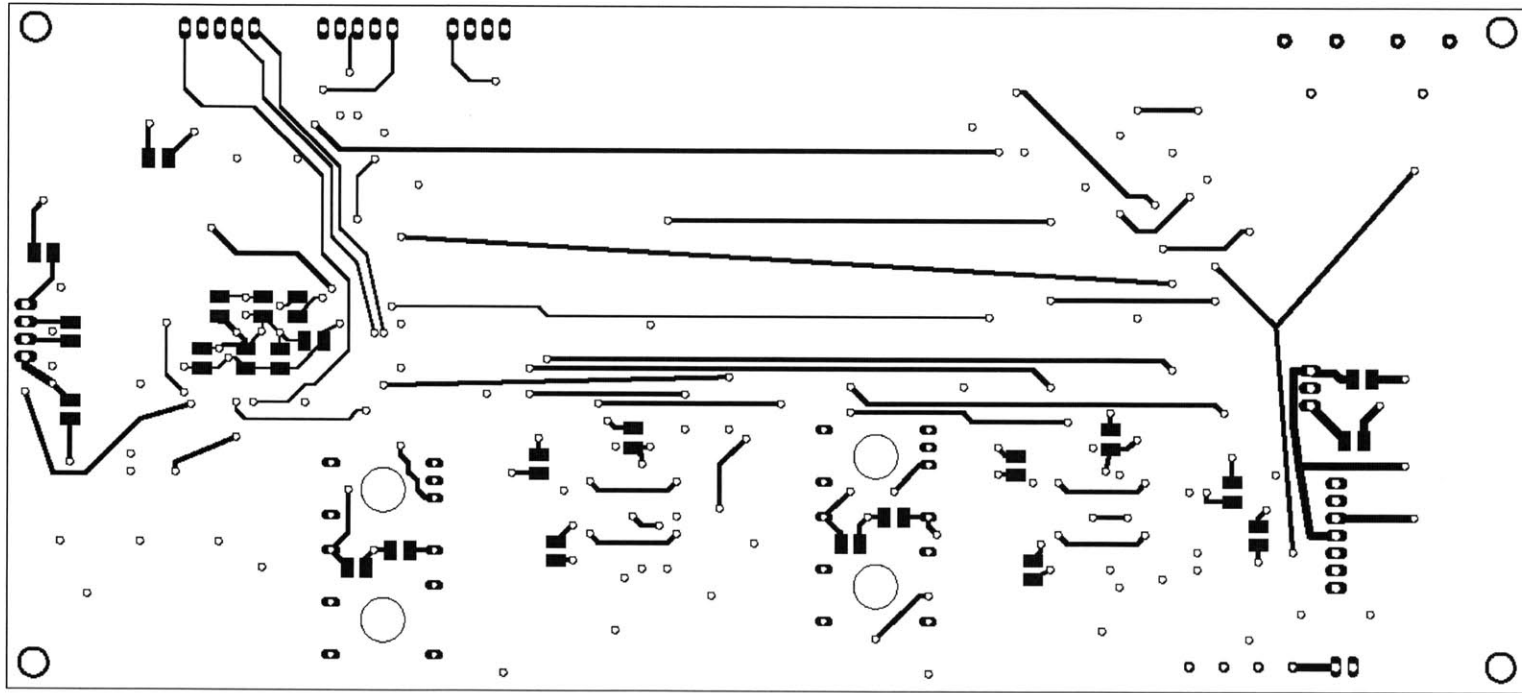


Figure E-5: Bottom copper layer

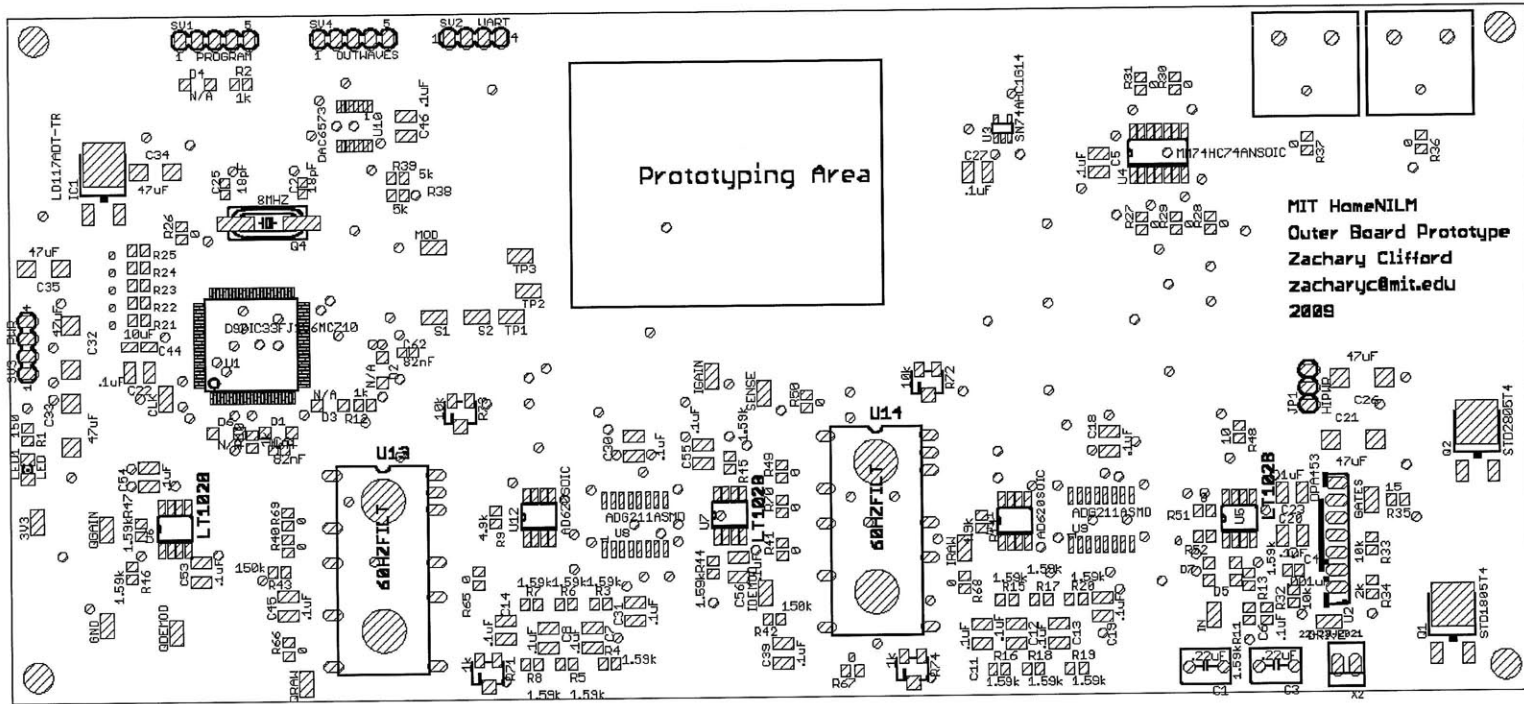


Figure E-6: Top silk layer

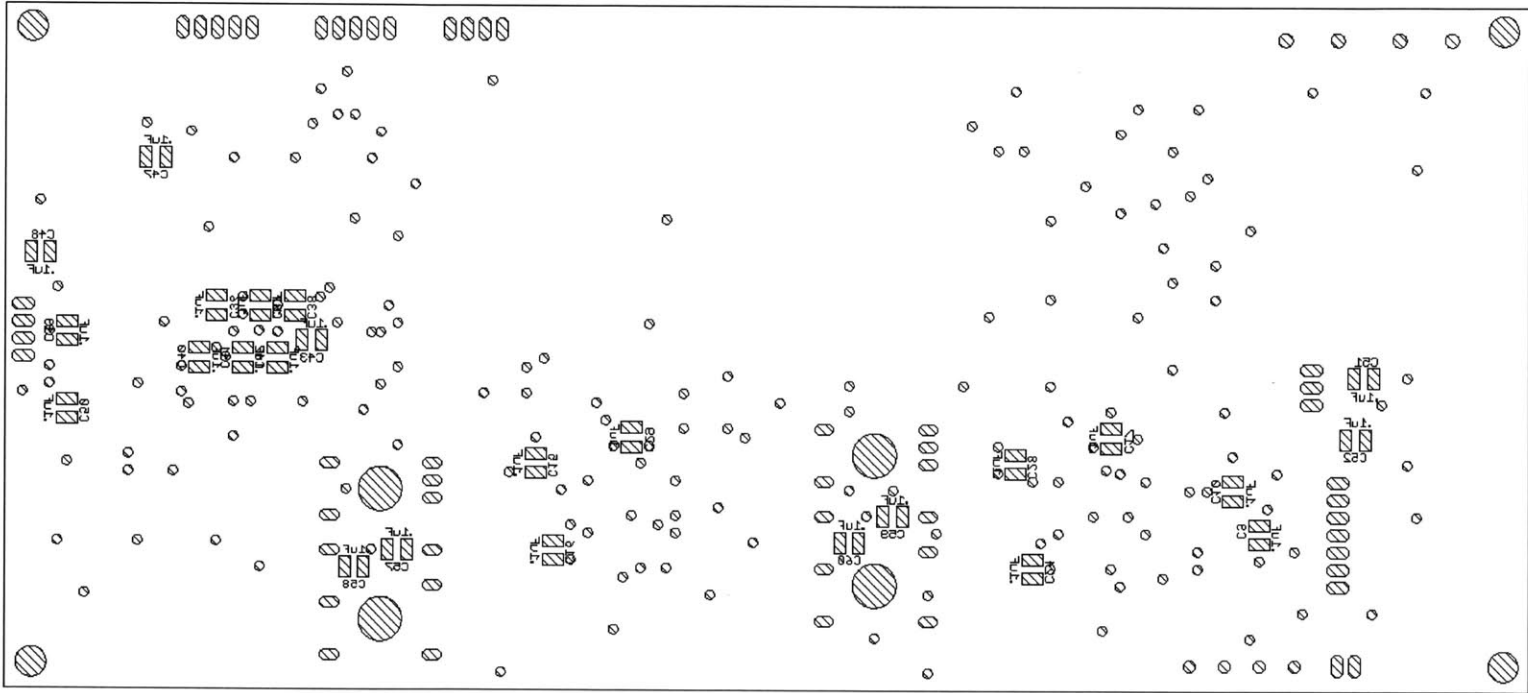


Figure E-7: Bottom silk layer

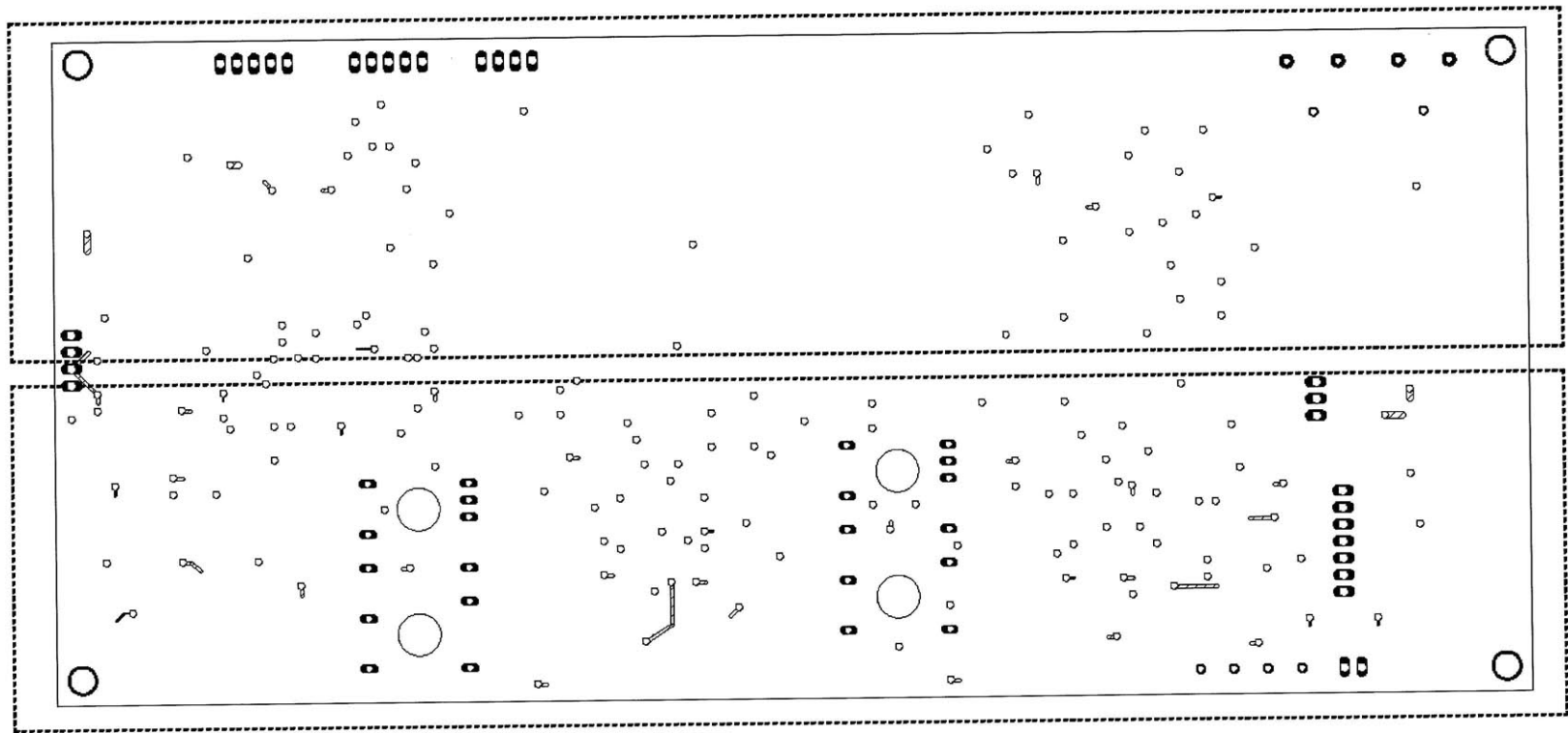


Figure E-8: Copper layer 2 with ground plane not filled



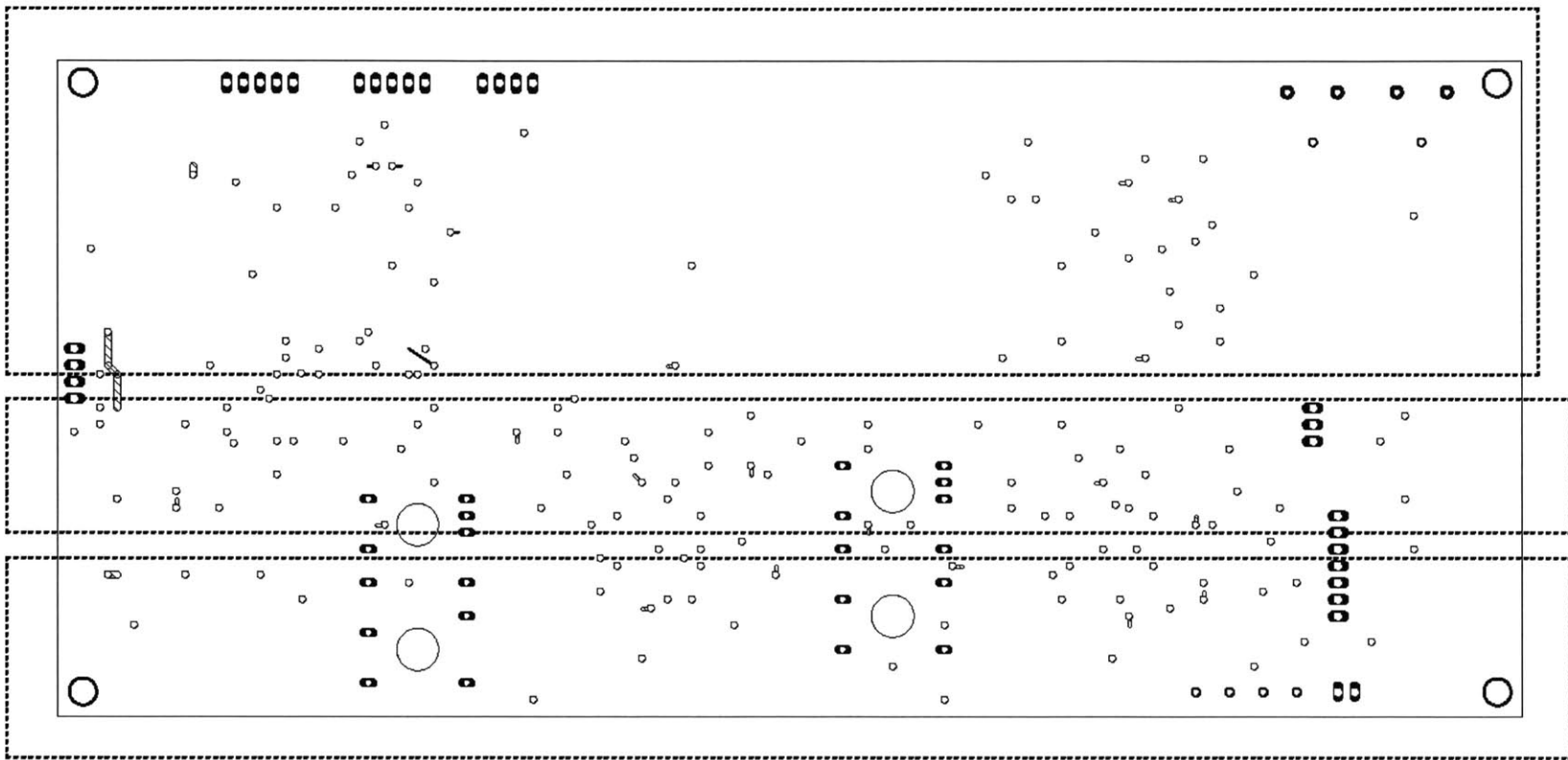


Figure E-9: Copper layer 3 with power planes not filled



# Appendix F

## NerdJack Source Code Listing

### F.1 Firmware Source Code

This section contains the source code for the parts of the Data Acquisition Device that were modified from Atmel's Software Framework. Operating System files and drivers are not included in order to save printing space. These can be downloaded from Atmel's website. The included program uses the Software Framework version 1.4.0 and compiles using Atmel's modified GCC toolchain version 2.2.1. The framework was updated to have FreeRTOS 5.0.4 and lwIP 1.3.0. Minor changes were made to the driver layer to fix bugs and to match the new API of FreeRTOS 5.

#### F.1.1 FreeRTOSConfig.h

---

```
/* This header file is part of the ATMEL AVR32-SoftwareFramework-1.2.1ES-AT32UC3A  
Release */
```

```
/*This file is prepared for Doxygen automatic documentation generation.*/
```

```
/*! \file *****
```

```
*
```

```
* \brief FreeRTOS and lwIP example for AVR32 UC3.
```

```
*
```

```
* - Compiler: IAR EWAVR32 and GNU GCC for AVR32
```

```
* - Supported devices: All AVR32 devices can be used.
```

```
* - AppNote:
```

```
*
```

```
* \author Atmel Corporation: http://www.atmel.com \n
```

```
* Support and FAQ: http://support.atmel.no/
```

```
*
```

```
*****/
```

```
/* Copyright (c) 2007, Atmel Corporation All rights reserved.
```

```
*
```

```
* Redistribution and use in source and binary forms, with or without
```

\* modification, are permitted provided that the following conditions are met:  
 \*  
 \* 1. Redistributions of source code must retain the above copyright notice,  
 \* this list of conditions and the following disclaimer.  
 \*  
 \* 2. Redistributions in binary form must reproduce the above copyright notice,  
 \* this list of conditions and the following disclaimer in the documentation  
 \* and/or other materials provided with the distribution .  
 \*  
 \* 3. The name of ATMEL may not be used to endorse or promote products derived  
 \* from this software without specific prior written permission.  
 \*  
 \* THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR  
 \* IMPLIED  
 \* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES  
 \* OF  
 \* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY  
 \* AND  
 \* SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY  
 \* DIRECT,  
 \* INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES  
 \* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR  
 \* SERVICES;  
 \* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER  
 \* CAUSED AND  
 \* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,  
 \* OR TORT  
 \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE  
 \* USE OF  
 \* THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  
 \*/

```
#ifndef FREERTOS_CONFIG_H
#define FREERTOS_CONFIG_H
```

```
#include "board.h"
```

```
/*-----  

  * Application specific definitions .  

  *  

  * These definitions should be adjusted for your particular hardware and  

  * application requirements.  

  *  

  * THESE PARAMETERS ARE DESCRIBED WITHIN THE 'CONFIGURATION' SECTION  

  * OF THE  

  * FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS.org WEB SITE.  

  *-----*/
```

```
#define configUSE_PREEMPTION 1
#define configUSE_IDLE_HOOK 0
#define configUSE_TICK_HOOK 0
#define configCPU_CLOCK_HZ ( 66000000 ) /* Hz clk gen */
```

```

#define configPBA_CLOCK_HZ ( 66000000 )
#define configTICK_RATE_HZ ( ( portTickType ) 1000 )
#define configMAX_PRIORITIES ( ( unsigned portBASE_TYPE ) 8 )
#define configMINIMAL_STACK_SIZE ( ( unsigned portSHORT ) 256 )
/* configTOTAL_HEAP_SIZE is not used when heap_3.c is used. */
#define configTOTAL_HEAP_SIZE ( ( size_t ) ( 1024*25 ) )
#define configMAX_TASK_NAME_LEN ( 20 )
#define configUSE_TRACE_FACILITY 0
#define configUSE_16_BIT_TICKS 0
#define configIDLE_SHOULD_YIELD 1

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES 0
#define configMAX_CO_ROUTINE_PRIORITIES ( 0 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet 1
#define INCLUDE_uxTaskPriorityGet 1
#define INCLUDE_vTaskDelete 1
#define INCLUDE_vTaskCleanUpResources 0
#define INCLUDE_vTaskSuspend 1
#define INCLUDE_vTaskDelayUntil 1
#define INCLUDE_vTaskDelay 1
#define INCLUDE_xTaskGetCurrentTaskHandle 1
#define INCLUDE_xTaskGetSchedulerState 0

/* configTICK_USE_TC is a boolean indicating whether to use a Timer Counter
for the tick generation. Timer Counter will generate an accurate Tick;
otherwise the CPU will generate a tick but with time drift .
configTICK_TC_CHANNEL is the TC channel. */
#define configTICK_USE_TC 0

/* configHEAP_INIT is a boolean indicating whether to initialize the heap with
0xA5 in order to be able to determine the maximal heap consumption. */
#define configHEAP_INIT 0
#define configUSE_COUNTING_SEMAPHORES 1
#define configUSE_MUTEXES 1

#endif /* FREERTOS_CONFIG_H */

```

---

## F.1.2 conf\_eth.h

---

```

/* This header file is part of the ATMEL AVR32-SoftwareFramework-AT32UC3A-1.4.0
Release */

/* This file is prepared for Doxygen automatic documentation generation.*/
/*! \file *****
*
* \brief Ethernet module configuration file .

```

```

*
* This file contains the possible external configuration of the Ethernet module.
*
* - Compiler: IAR EWAVR32 and GNU GCC for AVR32
* - Supported devices: All AVR32 devices can be used.
* - AppNote:
*
* \author Atmel Corporation: http://www.atmel.com \n
* Support and FAQ: http://support.atmel.no/
*
*****/

/* Copyright (C) 2006–2008, Atmel Corporation All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* 1. Redistributions of source code must retain the above copyright notice,
* this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution .
*
* 3. The name of ATMEL may not be used to endorse or promote products derived
* from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY ATMEL “AS IS” AND ANY EXPRESS OR
* IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
* OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY
* AND
* SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY
* DIRECT,
* INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
* SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
* CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
* USE OF
* THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#ifndef _CONF_ETH_H_
#define _CONF_ETH_H_

#include "arch/cc.h"

```

```

/*! Phy Address (set through strap options) */
#define ETHERNET_CONF_PHY_ADDR 0x01
#define ETHERNET_CONF_PHY_ID    0x20005C90

/*! Number of receive buffers. Max ethernet frame size is 1526. A Rx buffer is
128 Bytes long. So 12 Rx buffers are necessary to store one max sized frame.
Multiply that by 2 for performance. */
#define ETHERNET_CONF_NB_RX_BUFFERS 24

/*! USE_RMII_INTERFACE must be defined as 1 to use an RMII interface, or 0
to use an MII interface. */
#define ETHERNET_CONF_USE_RMIIINTERFACE 1

/*! Number of Transmit buffers */
#define ETHERNET_CONF_NB_TX_BUFFERS 10

/*! Size of each Transmit buffer. */
#define ETHERNET_CONF_TX_BUFFER_SIZE 512

/*! Clock definition - PBB clock */
#define ETHERNET_CONF_SYSTEM_CLOCK 66000000
/*! Use Auto Negotiation to get speed and duplex */
#define ETHERNET_CONF_AN_ENABLE    1

/*! Do not use auto cross capability. Unused because not supported by the DP83848
phy on the EVK1100. */
#define ETHERNET_CONF_AUTO_CROSS_ENABLE 0
/*! use direct cable */
#define ETHERNET_CONF_CROSSED_LINK    0

/*! Base address of the flash user page */
#define USERPAGE_BASE_ADDR            0x80800000

/*! This is the structure of data in the FLASH user page
* It will line up with the data there.
*/
typedef struct __attribute__((__packed__))
{
    u32_t ipaddr [7];
    u32_t netmask[7];
    u32_t gateway[7];
    u8_t mac[3];
    u8_t serialnum [6];
} userpagedata;

/* ethernet default parameters */
/*! \brief MAC address definition.
* The MAC address must be unique on the network.
* The lower three are set by the DIP switch settings from the User Page.
*/
#define ETHERNET_CONF_ETHADDR0        0x00
#define ETHERNET_CONF_ETHADDR1        0x04

```

```
#define ETHERNET_CONF_ETHADDR2          0x25

#endif
```

---

### F.1.3 conf\_lwip\_threads.h

---

```
/* This header file is part of the ATMEL AVR32-SoftwareFramework-AT32UC3A-1.4.0
   Release */

/* This file is prepared for Doxygen automatic documentation generation.*/
/*! \file *****
 *
 * \brief lwIP core & application threads configuration file .
 *
 * This file contains the possible external configuration of the Ethernet module.
 *
 * - Compiler:          IAR EWAVR32 and GNU GCC for AVR32
 * - Supported devices: All AVR32 devices can be used.
 * - AppNote:
 *
 * \author             Atmel Corporation: http://www.atmel.com \n
 *                    Support and FAQ: http://support.atmel.no/
 *
 *****/

/* Copyright (C) 2006-2008, Atmel Corporation All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 * this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution .
 *
 * 3. The name of ATMEL may not be used to endorse or promote products derived
 * from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR
 * IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY
 * AND
 * SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY
 * DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES;
```



```

* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
  CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
  USE OF
* THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

```

```

#ifndef _CONF_LWIP_THREADS_H_
#define _CONF_LWIP_THREADS_H_

```

```

/* define stack size for DataStream server task */
#define ethDATASTREAM_SERVER_STACK_SIZE configMINIMAL_STACK_SIZE

```

```

// #define lwipBASIC_WEB_SERVER_STACK_SIZE configMINIMAL_STACK_SIZE

```

```

/* define stack size for lwIP task */
#define lwipINTERFACE_STACK_SIZE 512

```

```

/* define stack size for netif task */
#define netifINTERFACE_TASK_STACK_SIZE 256

```

```

#define ethWDT_TASK_STACK_SIZE configMINIMAL_STACK_SIZE

```

```

#define packSTACK_SIZE configMINIMAL_STACK_SIZE

```

```

#define COMMAND_STACK_SIZE configMINIMAL_STACK_SIZE

```

```

#define AUTOD_STACK_SIZE configMINIMAL_STACK_SIZE

```

```

#define packPriority 4

```

```

#define ethWDT_TASK_PRIORITY ( configMAX_PRIORITIES - 1 )
#define COMMAND_PRIORITY ( tskIDLE_PRIORITY + 2 )
#define SAMPLEMANAGER_PRIORITY ( tskIDLE_PRIORITY + 5 )
#define AUTOD_PRIORITY ( tskIDLE_PRIORITY + 1 )

```

```

/* define DataStream server priority */
#define ethDATASTREAMSERVER_PRIORITY ( tskIDLE_PRIORITY + 3 )

```

```

// #define lwipBASIC_WEB_SERVER_PRIORITY ( tskIDLE_PRIORITY + 1 )

```

```

/* define lwIP task priority */
#define lwipINTERFACE_TASK_PRIORITY ( configMAX_PRIORITIES - 2 )

```

```

/* define netif task priority */
#define netifINTERFACE_TASK_PRIORITY ( configMAX_PRIORITIES - 2 )

```

```

/* Number of threads that can be started with sys_thread_new() in lwip */
#define SYS_THREAD_MAX 8

```

```
#endif // #ifndef _CONF_LWIP_THREADS_H
```

---

## F.1.4 externalmem.h

---

```
/*!\ file externalmem.h */
#ifndef EXTERNALMEM_H
#define EXTERNALMEM_H

/*!Location in memory where packets are stored */
#define ADC_SAMPLE_START 0xD000000
#define ADC_SAMPLE_END 0xD1FFFFFF

/*!Location of the SDRAM in the memory space */
#define SDRAM_START 0xD000000
#define SDRAM_END 0xD1FFFFFF

#endif /*EXTERNALMEM_H_*/
```

---

## F.1.5 lwipopts.h

---

```
/* This header file is part of the ATMEL AVR32-SoftwareFramework-AT32UC3A-1.4.0
   Release */

/*This file has been prepared for Doxygen automatic documentation generation.*/
/*!\ file *****/
*
* \brief lwIP configuration for AVR32 UC3.
*
* - Compiler: GNU GCC for AVR32
* - Supported devices: All AVR32 devices can be used.
* - AppNote:
*
* \author Atmel Corporation: http://www.atmel.com \n
* Support and FAQ: http://support.atmel.no/
*
*****/

/* Copyright (C) 2006-2008, Atmel Corporation All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* 1. Redistributions of source code must retain the above copyright notice,
* this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution .
*
* 3. The name of ATMEL may not be used to endorse or promote products derived
```

```

* from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR
  IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
  OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY
  AND
* SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY
  DIRECT,
* INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
  SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
  CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
  USE OF
* THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

```

```

#ifndef _LWIPOPTS_H_
#define _LWIPOPTS_H_

```

```

/* Include user defined options first */
#include "conf_lwip_threads.h"

```

```

/* Turn off debugging mode */
#define LWIP_NOASSERT 1

```

```

/* These two control is reclaimer functions should be compiled
   in. Should always be turned on (1). */
#define MEM_RECLAIM 1
#define MEMP_RECLAIM 1

```

```

/* Platform specific locking */

```

```

/*
 * enable SYS_LIGHTWEIGHT_PROT in lwipopts.h if you want inter-task protection
 * for certain critical regions during buffer allocation, deallocation and memory
 * allocation and deallocation.
 */
#define SYS_LIGHTWEIGHT_PROT 1

```

```

/* ----- Memory options ----- */

```

```

/* MEM_ALIGNMENT: should be set to the alignment of the CPU for which
  lwIP is compiled. 4 byte alignment -> define MEM_ALIGNMENT to 4, 2
  byte alignment -> define MEM_ALIGNMENT to 2. */

```

```

#define MEM_ALIGNMENT 4

/* MEM_SIZE: the size of the heap memory. If the application will send
a lot of data that needs to be copied, this should be set high. */
#define MEM_SIZE 12 * 1024

/* MEMP_NUM_PBUF: the number of memp struct pbufs. If the application
sends a lot of data out of ROM (or other static memory), this
should be set high. */
#define MEMP_NUM_PBUF 6

#define LWIP_RAW 0
/* Number of raw connection PCBs */
#define MEMP_NUM_RAW_PCB 0

/* ----- UDP options ----- */
#define LWIP_UDP 1
#define UDP_TTL 255
/* MEMP_NUM_UDP_PCB: the number of UDP protocol control blocks. One
per active UDP "connection". */

#define MEMP_NUM_UDP_PCB 2

/* MEMP_NUM_TCP_PCB: the number of simultaneously active TCP connections. */
#define MEMP_NUM_TCP_PCB 4
/* MEMP_NUM_TCP_PCB_LISTEN: the number of listening TCP connections. */
#define MEMP_NUM_TCP_PCB_LISTEN 4
/* MEMP_NUM_TCP_SEG: the number of simultaneously queued TCP segments. */
#define MEMP_NUM_TCP_SEG 24
/* MEMP_NUM_SYS_TIMEOUT: the number of simultaneously active timeouts. */
#define MEMP_NUM_SYS_TIMEOUT 9

/* The following four are used only with the sequential/sockets API and can be
set to 0 if the application only will use the raw API. */
/* MEMP_NUM_NETBUF: the number of struct netbufs. */
#define MEMP_NUM_NETBUF 3
/* MEMP_NUM_NETCONN: the number of struct netconns. */
#define MEMP_NUM_NETCONN 6

/* ----- Pbuf options ----- */
/* PBUF_POOL_SIZE: the number of buffers in the pbuf pool. This is for data
* reception, not data sending */

#define PBUF_POOL_SIZE 3

/* PBUF_POOL_BUFSIZE: the size of each pbuf in the pbuf pool. */

#define PBUF_POOL_BUFSIZE 500

/* PBUF_LINK_HLEN: the number of bytes that should be allocated for a
link level header. */
#define PBUF_LINK_HLEN 16

```

```

/* ----- TCP options ----- */
#define LWIP_TCP          1
#define TCP_TTL          255
/* TCP receive window. */
#define TCP_WND          1460
/* Controls if TCP should queue segments that arrive out of
   order. Define to 0 if your device is low on memory. */
#define TCP_QUEUE_OOSEQ 1

/* TCP Maximum segment size. */
#define TCP_MSS          1460

/* TCP sender buffer space (bytes). */
#define TCP_SND_BUF      7*1460

/* This defines when the buffer is "low" */
#define TCP_SNDLOWAT     6*1460

/* TCP sender buffer space (pbufs). This must be at least = 2 *
   TCP_SND_BUF/TCP_MSS for things to work. */
#define TCP_SND_QUEUELEN 2 * TCP_SND_BUF/TCP_MSS

/* Maximum number of retransmissions of data segments. */
#define TCP_MAXRTX       12

/* Maximum number of retransmissions of SYN segments. */
#define TCP_SYNMAXRTX    4

/* Limiting retransmits and making the timers
   * faster allows dead connections to die quickly */
// #define TCP_TMR_INTERVAL 100

/* Enable receive timeout processing so that we can have nonblocking
   * receive calls
   */
// #define LWIP_SO_RCVTIMEO 1

/**
   * DEFAULT_RAW_RECVMBOX_SIZE: The mailbox size for the incoming packets on a
   * NETCONN_RAW. The queue size value itself is platform-dependent, but is passed
   * to sys_mbox_new() when the recvmbox is created.
   */
#define DEFAULT_RAW_RECVMBOX_SIZE 6

/**
   * DEFAULT_UDP_RECVMBOX_SIZE: The mailbox size for the incoming packets on a
   * NETCONN_UDP. The queue size value itself is platform-dependent, but is passed
   * to sys_mbox_new() when the recvmbox is created.
   */
#define DEFAULT_UDP_RECVMBOX_SIZE 6

/**

```

```

* DEFAULT_TCP_RECVMBOX_SIZE: The mailbox size for the incoming packets on a
* NETCONN_TCP. The queue size value itself is platform-dependent, but is passed
* to sys_mbox_new() when the recvmbox is created.
*/
#define DEFAULT_TCP_RECVMBOX_SIZE 6

/**
* DEFAULT_ACCEPTMBOX_SIZE: The mailbox size for the incoming connections.
* The queue size value itself is platform-dependent, but is passed to
* sys_mbox_new() when the acceptmbox is created.
*/
#define DEFAULT_ACCEPTMBOX_SIZE 6

/* ----- ARP options ----- */
#define ARP_TABLE_SIZE 10
#define ARP_QUEUEING 0

/* ----- IP options ----- */
/* Define IP_FORWARD to 1 if you wish to have the ability to forward
IP packets across network interfaces. If you are going to run lwIP
on a device with only one network interface, define this to 0. */
#define IP_FORWARD 0

/* If defined to 1, IP options are allowed (but not parsed). If
defined to 0, all packets with IP options are dropped. */
#define IP_OPTIONS 1

/* ----- ICMP options ----- */
#define ICMP_TTL 255

/* ----- DHCP options ----- */
/* Define LWIP_DHCP to 1 if you want DHCP configuration of
interfaces. DHCP is not implemented in lwIP 0.5.1, however, so
turning this on does currently not work. */
#define LWIP_DHCP 1

/* 1 if you want to do an ARP check on the offered address
(recommended). */
#define DHCP_DOES_ARP_CHECK 1

/*
-----
----- Thread options -----
-----
*/
/**
* TCPIP_THREAD_NAME: The name assigned to the main tcpip thread.
*/
#define TCPIP_THREAD_NAME "TCP/IP"

/**
* TCPIP_THREAD_STACKSIZE: The stack size used by the main tcpip thread.

```

```

* The stack size value itself is platform-dependent, but is passed to
* sys_thread_new() when the thread is created.
*/
#define TCPIP_THREAD_STACKSIZE lwipINTERFACE_STACK_SIZE

/**
* TCPIP_THREAD_PRIO: The priority assigned to the main tcpip thread.
* The priority value itself is platform-dependent, but is passed to
* sys_thread_new() when the thread is created.
*/
#define TCPIP_THREAD_PRIO      lwipINTERFACE_TASK_PRIORITY

/**
* TCPIP_MBOX_SIZE: The mailbox size for the tcpip thread messages
* The queue size value itself is platform-dependent, but is passed to
* sys_mbox_new() when tcpip_init is called.
*/
#define TCPIP_MBOX_SIZE        6

/**
* SLIPIF_THREAD_NAME: The name assigned to the slipif_loop thread.
*/
#define SLIPIF_THREAD_NAME     "slipif"

/**
* SLIP_THREAD_STACKSIZE: The stack size used by the slipif_loop thread.
* The stack size value itself is platform-dependent, but is passed to
* sys_thread_new() when the thread is created.
*/
#define SLIPIF_THREAD_STACKSIZE configMINIMAL_STACK_SIZE

/**
* SLIPIF_THREAD_PRIO: The priority assigned to the slipif_loop thread.
* The priority value itself is platform-dependent, but is passed to
* sys_thread_new() when the thread is created.
*/
#define SLIPIF_THREAD_PRIO     1

/**
* PPP_THREAD_NAME: The name assigned to the pppMain thread.
*/
#define PPP_THREAD_NAME        "pppMain"

/**
* PPP_THREAD_STACKSIZE: The stack size used by the pppMain thread.
* The stack size value itself is platform-dependent, but is passed to
* sys_thread_new() when the thread is created.
*/
#define PPP_THREAD_STACKSIZE   configMINIMAL_STACK_SIZE

/**
* PPP_THREAD_PRIO: The priority assigned to the pppMain thread.
* The priority value itself is platform-dependent, but is passed to
* sys_thread_new() when the thread is created.

```

```

*/
#define PPP_THREAD_PRIO      1

/**
 * DEFAULT_THREAD_NAME: The name assigned to any other lwIP thread.
 */
#define DEFAULT_THREAD_NAME "lwIP"

/**
 * DEFAULT_THREAD_STACKSIZE: The stack size used by any other lwIP thread.
 * The stack size value itself is platform-dependent, but is passed to
 * sys_thread_new() when the thread is created.
 */
#define DEFAULT_THREAD_STACKSIZE configMINIMAL_STACK_SIZE

/**
 * DEFAULT_THREAD_PRIO: The priority assigned to any other lwIP thread.
 * The priority value itself is platform-dependent, but is passed to
 * sys_thread_new() when the thread is created.
 */
#define DEFAULT_THREAD_PRIO  1

/*! Use the thread-safe NETIF API for controlling the interface */
#define LWIP_NETIF_API      1

/* ----- Statistics options ----- */
// #define LWIP_STATS 0

// #define LWIP_STATS_DISPLAY 0

#if LWIP_STATS
#define LINK_STATS 1
#define IP_STATS 1
#define ICMP_STATS 1
#define UDP_STATS 1
#define TCP_STATS 1
#define MEM_STATS 1
#define MEMP_STATS 1
#define PBUF_STATS 1
#define SYS_STATS 1
#endif /* STATS */

/* ----- Lwip Debug options ----- */

/* Disable debugging */

#undef LWIP_DEBUG

#define DBG_TYPES_ON      0xff

```



```

#define ETHARP_DEBUG          DBG_OFF
#define NETIF_DEBUG          DBG_OFF
#define PBUF_DEBUG           DBG_ON
#define API_LIB_DEBUG        DBG_OFF
#define API_MSG_DEBUG        DBG_ON
#define SOCKETS_DEBUG        DBG_OFF
#define ICMP_DEBUG           DBG_OFF
#define INET_DEBUG           DBG_OFF
#define IP_DEBUG             DBG_OFF
#define IP_REASS_DEBUG        DBG_OFF
#define RAW_DEBUG            DBG_OFF
#define MEM_DEBUG            DBG_OFF
#define MEMP_DEBUG           DBG_OFF
#define SYS_DEBUG            DBG_OFF
#define TCP_DEBUG            DBG_ON
#define TCP_INPUT_DEBUG      DBG_OFF
#define TCP_FR_DEBUG         DBG_OFF
#define TCP_RTO_DEBUG        DBG_OFF
#define TCP_CWND_DEBUG       DBG_OFF
#define TCP_WND_DEBUG        DBG_OFF
#define TCP_OUTPUT_DEBUG     DBG_OFF
#define TCP_RST_DEBUG        DBG_OFF
#define TCP_QLEN_DEBUG       DBG_OFF
#define UDP_DEBUG            DBG_OFF
#define TCPIP_DEBUG          DBG_OFF
#define DBG_MIN_LEVEL        LWIP_DBG_LEVEL_SEVERE

#endif /* __LWIPOPTS_H__ */

```

---

## F.1.6 DataStream.h

---

```
/*This file has been prepared for Doxygen automatic documentation generation.*/
/*! \file *****
*
* \brief Basic WEB Server for AVR32 UC3.
*
* - Compiler:          GNU GCC for AVR32
* - Supported devices: All AVR32 devices can be used.
* - AppNote:
*
* \author             Atmel Corporation: http://www.atmel.com \n
*                    Support and FAQ: http://support.atmel.no/
*
*****/

/* Copyright (c) 2007, Atmel Corporation All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* 1. Redistributions of source code must retain the above copyright notice,
* this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution .
*
* 3. The name of ATMEL may not be used to endorse or promote products derived
* from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR
* IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
* OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY
* AND
* SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY
* DIRECT,
* INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
* SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
* CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
* USE OF
* THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
```

```

#ifndef DATASTREAM_H
#define DATASTREAM_H

#include "portmacro.h"

/*! The maximum number of samples in a packet. There might be less if the
 * channels sampled does not evenly divide into it
 */
#define NUM_SAMPLES_PER_PACKET 726

/*! \brief The structure of an ethernet packet
 *
 * Note that an Ethernet MTU is 1500. With TCP/IP headers, I can fit 1460
 * bytes of data. This packet is being designed to fit in one unfragmented
 * TCP/IP segment.
 * The header is 12 bytes long. There will be 724 samples per packet. The first
 * sample should be channel 1. Any left over will just be junk to be ignored
 */
typedef struct __attribute__((__packed__))
{
    unsigned char headerone;
    unsigned char headertwo;
    unsigned short packetNumber;
    unsigned short adcsused;
    unsigned short packetsready;
    signed short data[NUM_SAMPLES_PER_PACKET];
} dataPacket;

#define SIZE_OF_PACKET    1460
#define HEADER_LENGTH    8

/*! The address of the base packet. They can be thought of as an array spanning
 * the external SDRAM
 */
#define basePacket ((dataPacket *) ADC_SAMPLE_START)

#define NUM_PACKETS      (ADC_SAMPLE_END - ADC_SAMPLE_START) / (
    sizeof(dataPacket) )

extern xSemaphoreHandle PacketReadySemaphore;

extern volatile int ADCReadySemaphore;

extern dataPacket PacketStore[8];

/*! \brief Reset the Datastream task
 *
 * This should be called only when DataStream is stopped
 */
void resetDataStream (void);

void StartCopyTask (void);

```

```

/*! \brief Datastream server main task
*
* \param pvParameters Input. Not Used.
*
*/
portTASK_FUNCTION_PROTO (vDataStreamServer, pvParameters);

/*! \brief Command server main task
*/
portTASK_FUNCTION_PROTO (vCommandServer, pvParameters);

/*! \brief Autodetection server main task
*/
portTASK_FUNCTION_PROTO (vAutodetectServer, pvParameters);

portTASK_FUNCTION_PROTO (copyDataTask, pvParameters);

#endif

```

---

## F.1.7 DataStream.c

```

/*This file has been prepared for Doxygen automatic documentation generation.*/
/*! \file *****
*
* \brief Basic WEB Server for AVR32 UC3.
*
* - Compiler:          GNU GCC for AVR32
* - Supported devices: All AVR32 devices can be used.
* - AppNote:
*
* \author              Atmel Corporation: http://www.atmel.com \n
*                      Support and FAQ: http://support.atmel.no/
*
*****/

/* Copyright (c) 2007, Atmel Corporation All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* 1. Redistributions of source code must retain the above copyright notice,
* this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution .
*
* 3. The name of ATMEL may not be used to endorse or promote products derived
* from this software without specific prior written permission.
*
*

```

\* THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR  
 IMPLIED  
 \* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES  
 OF  
 \* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY  
 AND  
 \* SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY  
 DIRECT,  
 \* INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES  
 \* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR  
 SERVICES;  
 \* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER  
 CAUSED AND  
 \* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,  
 OR TORT  
 \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE  
 USE OF  
 \* THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  
 \*/

/\*  
 \* This code was taken from BasicWEB sources and modified by Zachary Clifford  
 \* <zacharyc@mit.edu> for use in the ethernet data acquisition hardware at LEES.  
 \*  
 \* This file implements a server for starting data capture and sending it  
 \* to a client. The client sends "GET" followed by three hex digits representing  
 \* the channels to sample to the command port, followed by a 0, 1, 2, or 3 to indicate  
 \* what the range setting should be for each ADC. Data can be captured from the data  
 \* port. Autodetection is also provided, as well as other commands.  
 \*  
 \* Data stops sending when there is an error. Usually this is from the client  
 \* resetting the TCP/IP connection.  
 \*  
 \* \*/

/\* Standard includes. \*/  
 #include <stdio.h>  
 #include <string.h>  
 #include <stdint.h>

#include "conf\_eth.h"

/\* Scheduler includes. \*/  
 #include "FreeRTOS.h"  
 #include "task.h"  
 #include "semphr.h"

/\* Demo includes. \*/  
 #include "portmacro.h"

/\* lwIP includes. \*/  
 #include "lwipopts.h"  
 #include "lwip/api.h"  
 #include "lwip/tcpip.h"

```

#include "lwip/memp.h"
#include "lwip/stats.h"
#include "lwip/opt.h"
#include "lwip/arch.h"
#include "lwip/sys.h"
#include "netif/loopif.h"
#include "lwip/sockets.h"

/* ethernet includes */
#include "ethernet.h"

#include "externalmem.h"
#include "samplemanager.h"
#include "InitBoard.h"
#include "gpio.h"
#include "wdtreset.h"
#include "DataStream.h"
#include "usart.h"
#include "version.h"

/*! The port on which we listen for datastream. */
#define DataStream_PORT ( 49155 )

/*! The Autodetection port */
#define Autodetect_PORT ( 49156 )

/*! The command port */
#define Command_PORT ( 49157 )

/*! Function to process the current connection */
static void parseCommand (int lSocket);

dataPacket PacketStore[8];

/*! Semaphore indicates when we're streaming */
static xSemaphoreHandle streamActive;

/*! Queue that signals the Datastream task to stop streaming
 * Its messages are never examined
 */

static xQueueHandle stopstream;

/*! How many fully assembled packets awaiting transmission */
xSemaphoreHandle PacketReadySemaphore;

volatile int ADCReadySemaphore = 0;

/*! The packet that we are currently reading to the Ethernet buffer */
static unsigned short tcpipPacketRead = 0;

/*! The 16 bit packet count to append to the current packet */

```

```

static unsigned short currentCount = 0;

static int packetStoreIndex = 0;
static int copyPacket = 0;

/* The structure of a GET request to start sampling */
typedef struct __attribute__((__packed__))
{
    unsigned long period;
    unsigned short channelbit;
    unsigned char precision;
    unsigned char prescaler;
} getPacket;

/* \brief Resets the Datastream task.
 *
 * Should be called only when datastream is not running. Use the streamactive
 * and stopstream members for this
 */
void
resetDataStream (void)
{
    currentCount = 0;

    tcpipPacketRead = 0;

    copyPacket = 0;

    packetStoreIndex = 0;

    while (xSemaphoreTake (PacketReadySemaphore, 1) == pdTRUE);

    //Interrupts are always disabled when this function is called
    ADCReadySemaphore = 0;
}

/*
 * \brief Autodetection server task.
 *
 * Listens for incoming UDP packets on the appropriate port and responds.
 * This allows the device to be identified by broadcasts.
 */
portTASK_FUNCTION (vAutodetectServer, pvParameters)
{
    int lSocket;
    int lDataLen, lRecvLen, lFromLen;
    struct sockaddr_in sLocalAddr, sFromAddr;
    char Data[5] = "HERE"; ///< Buffer of data to send
    char incomingData[10];

    // Set up port

```

```

// Network order in info; host order in server:

for (;;)
{
    // Create socket
    lSocket = socket (AF_INET, SOCK_DGRAM, 0);
    if (lSocket < 0)
    {
        return;
    }
    int opt = 1;
    setsockopt (lSocket, SOL_SOCKET, SO_BROADCAST, &opt, sizeof (int));

    memset ((char *) &sLocalAddr, 0, sizeof (sLocalAddr));
    sLocalAddr.sin_family = AF_INET;
    sLocalAddr.sin_len = sizeof (sLocalAddr);
    sLocalAddr.sin_addr.s_addr = INADDR_ANY;
    sLocalAddr.sin_port = DataStream_PORT;

    if (bind (lSocket, (struct sockaddr *) &sLocalAddr, sizeof (sLocalAddr))
        < 0)
    {
        // Problem setting up my end
        close (lSocket);
        return;
    }

    lRecvLen = sizeof (incomingData);
    lFromLen = sizeof (sFromAddr);

    lDataLen = recvfrom (lSocket, incomingData, lRecvLen, 0,
                        (struct sockaddr *) &sFromAddr,
                        (socklen_t *) &lFromLen);

    if (lDataLen < 0)
    {
        //Problem receiving data. Do nothing
    }
    else
    {
        sFromAddr.sin_port = Autodetect_PORT;
        sendto (lSocket, Data, 5, 0, (struct sockaddr *) &sFromAddr,
              sizeof (struct sockaddr));
    }
    close (lSocket);
}

void
StartCopyTask (void)
{

```

```

xTaskCreate (copyDataTask, (const signed portCHAR * const) "SAMP",

```



```

        packSTACK_SIZE, NULL, packPriority, (xTaskHandle *) NULL);
    }

    /*! \brief DataStream server main task
    *      check for incoming connection and process it
    *
    *      \param pvParameters Input. Not Used.
    *
    */
    portTASK_FUNCTION (vDataStreamServer, pvParameters)
    {
        int bytesSent;
        int lSocket, lconnection;
        int lFromLen;
        struct sockaddr_in sFromAddr, sLocalAddr;
        //char message = DSTREAMOK; //!< Message to WDT reset task
        char stopmessage;

        PacketReadySemaphore = xSemaphoreCreateCounting (NUM_PACKETS, 0);

        /* Create a new tcp connection handle */
        lSocket = socket (PF_INET, SOCK_STREAM, 0);

        //struct timeval tv;

        //tv.tv_sec = 1;          /* 1 Secs Timeout */

        // setsockopt (lSocket, SOL_SOCKET, SO_RCVTIMEO, (struct timeval *) &tv,
        //              sizeof (struct timeval));

        memset ((char *) &sLocalAddr, 0, sizeof (sLocalAddr));
        sLocalAddr.sin_family = PF_INET;
        sLocalAddr.sin_len = sizeof (sLocalAddr);
        sLocalAddr.sin_addr.s_addr = INADDR_ANY;
        sLocalAddr.sin_port = DataStream_PORT;

        if (bind (lSocket, (struct sockaddr *) &sLocalAddr, sizeof (sLocalAddr)) <
            0)
        {
            // Problem setting up my end
            close (lSocket);
            return;
        }

        if (listen (lSocket, 0) < 0)
        {
            // Problem setting up my end
            close (lSocket);
            return;
        }

        streamActive = xSemaphoreCreateMutex ();
        stopstream = xQueueCreate (5, 1);
    }

```

```

gpio_enable_gpio_pin (LED1_PIN);

char gotstop = 0;

/* Loop forever */
for (;;)
{
    //Make sure the LED is off going into the accept call
    gpio_set_gpio_pin (LED1_PIN);

    /* Wait for a first connection. */
    lconnection =
        accept (lSocket, (struct sockaddr *) &sFromAddr,
                (socklen_t *) &lFromLen);

    //Tell WDT that the TCPIP task is OK
    //xQueueSend (watchdogMbox, &message, 0);

    //Since we're using a timeout, we might not actually have a connection
    if (lconnection > 0)
    {

        //Clear the stop queue in case it has junk in it
        while (xQueueReceive (stopstream, &stopmessage, 1) == pdTRUE);

        //Take the semaphore to notify that we're starting a stream
        //This mutex lets other tasks know if we're streaming. It also stops
        //streaming from happening should configuration be happening
        xSemaphoreTake (streamActive, portMAX_DELAY);

        //Turn on the LED now that we have taken the connection
        gpio_clr_gpio_pin (LED1_PIN);
        while (1)
        {
            //Go to sleep until Packet is ready.
            //Give it a 1 second timeout to reset the watchdog timer.
            gotstop = 0;
            while (pdTRUE !=
                    xSemaphoreTake (PacketReadySemaphore,
                                    portTICK_RATE_MS * 1000))
                //100))
            {
                //Tell WDT that the TCPIP task is OK
                //xQueueSend (watchdogMbox, &message, 0);

                //Take this time to check for a stop message
                if (xQueueReceive (stopstream, &stopmessage, 1) == pdTRUE)
                {
                    //We got a stop message
                    gotstop = 1;
                    break;
                }
            }
        }
    }
}

```

```

if (gotstop)
{
    gotstop = 0;
    break;
}

//Tell WDT that the TCPIP task is OK again
//xQueueSend (watchdogMbox, &message, 0);

//Fill in header data
basePacket[tcpipPacketRead].headerone = 0xF0;
basePacket[tcpipPacketRead].headertwo = 0xAA;

//Don't put currentCount in here. It's already done in the ISR

basePacket[tcpipPacketRead].adcused = 0;
//!< This is leftover from earlier implementation
basePacket[tcpipPacketRead].packetsready =
    htons (uxQueueMessagesWaiting (PacketReadySemaphore));

currentCount++;

bytesSent =
    send (lconnection, &basePacket[tcpipPacketRead],
        SIZE_OF_PACKET, 0);

//We want to increment this regardless of the success of the transmit
//Buffers need to stay consistent if we're to resend data successfully later
tcpipPacketRead++;
if (tcpipPacketRead == NUM_PACKETS)
{
    tcpipPacketRead = 0;
}

if (bytesSent != SIZE_OF_PACKET)
{
    //There was some error, so break out and close the connection
    //Usually this means the PC closed the connection on us
    //Clear the stop queue in case it has junk in it
    while (xQueueReceive (stopstream, &stopmessage, 1) ==
        pdTRUE);
    break;
}

//Take this time to check for a stop message
if (xQueueReceive (stopstream, &stopmessage, 1) == pdTRUE)
{
    //We got a stop message
    break;
}

}
close (lconnection);
xSemaphoreGive (streamActive);

```

```

        }                                /* end if new connection */

    }                                /* end infinite loop */
close (lSocket);
}

/*! \brief Starts the Command server.
 *
 * This listens on a port to accept commands to start, stop, or rewind
 * sampling. It is higher priority than other sampling tasks so that
 * they can be controlled properly
 */
portTASK_FUNCTION (vCommandServer, pvParameters)
{

    int lSocket, lconnection;
    int lFromLen;
    struct sockaddr_in sFromAddr, sLocalAddr;
    //char message = CMDOK;          //!< Message to WDT reset task

    /* Create a new tcp connection handle */
    lSocket = socket (PF_INET, SOCK_STREAM, 0);
    memset ((char *) &sLocalAddr, 0, sizeof (sLocalAddr));
    sLocalAddr.sin_family = PF_INET;
    sLocalAddr.sin_len = sizeof (sLocalAddr);
    sLocalAddr.sin_addr.s_addr = INADDR_ANY;
    sLocalAddr.sin_port = Command_PORT;

    //struct timeval tv;

    //tv.tv_sec = 1;                /* 1 Secs Timeout */

    //setsockopt (lSocket, SOL_SOCKET, SO_RCVTIMEO, (struct timeval *) &tv,
    //           sizeof (struct timeval));

    if (bind (lSocket, (struct sockaddr *) &sLocalAddr, sizeof (sLocalAddr)) <
        0)
    {
        // Problem setting up my end
        close (lSocket);
        return;
    }

    if (listen (lSocket, 0) < 0)
    {
        // Problem setting up my end
        close (lSocket);
        return;
    }

    /* Loop forever */
    for (;;)
    {

```

```

    /* Wait for a connection. */
    lconnection =
        accept (lSocket, (struct sockaddr *) &sFromAddr,
                (socklen_t *) &lFromLen);

    //Tell WDT that the CommandServer task is OK
    //xQueueSend (watchdogMbox, &message, 0);

    if (lconnection > 0)
    {
        parseCommand (lconnection);
    }
    /* end if new connection */

}
/* end infinite loop */
close (lSocket);
}

/**
 * \brief Copy data task
 *
 * Task to copy data from fast SRAM on chip to off-chip SDRAM
 */
portTASK_FUNCTION (copyDataTask, pvParameters)
{

    int gotflag = 0;

    while (1)
    {
        gotflag = 0;
        while (1)
        {
            //We have to yield a bit to give other tasks a chance to run
            taskYIELD();

            //This is a basic semaphore implementation
            //Using FreeRTOS semaphores was not fast enough
            portDISABLE_INTERRUPTS ();
            if (ADCReadySemaphore > 8)
            {
                {
                    gotflag = 2;
                }
            }
            else if (ADCReadySemaphore > 0)
            {
                {
                    ADCReadySemaphore = ADCReadySemaphore - 1;
                    gotflag = 1;
                }
            }
            portENABLE_INTERRUPTS ();
            if (gotflag == 1)
            {
                {
                    break;
                }
            }
            if (gotflag == 2)
            {

```

```

        SendStopMessage ();
        gotflag = 0;
    }
    //Yield here if no flag
    vTaskDelay (1);
}

memcpy (basePacket + copyPacket, PacketStore + packetStoreIndex,
        SIZE_OF_PACKET);

//Now signal the transmitter.
if (xSemaphoreGive (PacketReadySemaphore) != pdTRUE)
{
    //We can't give any more because the pipeline is stalled
    //Terminate sampling
    SendStopMessage ();
}

//Now update internal counters

packetStoreIndex++;
copyPacket++;

if (copyPacket == NUM_PACKETS)
{
    copyPacket = 0;
}

if (packetStoreIndex == 8)
{
    packetStoreIndex = 0;
}
}
}

/*\brief Converts ASCII to hex
 *
 * Utility function to convert an ASCII 0-9, A-F or a-f into
 * a hex representation. Returns 255 on invalid input
 */
static unsigned char
atohex (unsigned char input)
{
    if (input < 0x40 && input >= 0x30)
    {
        return input - 0x30;
    }
    if (input <= 0x46 && input > 0x40)
    {
        return input - 0x37;
    }
    if (input <= 0x66 && input > 0x60)
    {
        return input - 0x47;
    }
}

```

```

    }
    return 255;
}

/*! \brief parse the incoming request
 *      Take appropriate action based on it
 *
 * \param lSocket Input. The socket to use to send and receive data.
 *
 */

static void
parseCommand (int lSocket)
{
    int lDataLen;
    char pcRxString[9];
    getPacket thisGetPacket;

    /* We expect to immediately get a command data. */
    lDataLen = recv (lSocket, pcRxString, 4, 0);

    if (lDataLen > 0)
    {
        if (!strcmp (pcRxString, "TEST", 4))
        {
            /*It was a Test. Just reply with "WORKING"
            send (lSocket, "WORKING", 7, 0);
            }

        if (!strcmp (pcRxString, "VERS", 4))
        {
            /*Give our version string without the NERD: tag
            send (lSocket, versionstr+6,strlen (versionstr)-6,0);
            uint32_t config0_reg; // Config0 register
            uint8_t procId; // Processor ID
            uint8_t procRev; // Processor revision
            uint8_t archRev; // Architecture revision

            uint32_t did_reg; // Device ID register
            uint8_t revNum; // Revision number
            uint16_t prodNum; // Product number
            uint16_t manId; // Manufacturer ID
            char str [100];

            config0_reg = Get_system_register(AVR32_CONFIG0);
            procId = (config0_reg & AVR32_CONFIG0_PROCESSORID_MASK) >>
                AVR32_CONFIG0_PROCESSORID_OFFSET;
            procRev = (config0_reg & AVR32_CONFIG0_PROCESSORREVISION_MASK) >>
                AVR32_CONFIG0_PROCESSORREVISION_OFFSET;
            archRev = (config0_reg & AVR32_CONFIG0_AR_MASK) >>
                AVR32_CONFIG0_AR_OFFSET;
            sprintf (str, "Processor_ID=%d, Processor_Rev=%d, Architecture_Rev=%d\r\n", procId, procRev, archRev);

```

```

send (lSocket, str, strlen(str), 0);

did_reg = Get_debug_register(AVR32_DID);
revNum = (did_reg & AVR32_DID_RN_MASK) >> AVR32_DID_RN_OFFSET;
prodNum = (did_reg & AVR32_DID_PN_MASK) >> AVR32_DID_PN_OFFSET;
manId = (did_reg & AVR32_DID_MID_MASK) >> AVR32_DID_MID_OFFSET;
sprintf(str, "Revision_Number=%d, Product_Number=%x, Manufacturer_ID=%x\n", revNum, prodNum, manId);
send (lSocket, str, strlen(str), 0);
}
if (!strncmp (pcRxString, "SETC", 4))
{
    //We are trying to resend lost packets
    //Next two bytes tell what packet we want

    //First ensure that Datastream is stopped
    if (xSemaphoreTake (streamActive, 1) == pdFALSE)
    {
        char message = 0;
        xQueueSend (stopstream, &message, portMAX_DELAY);
        xSemaphoreTake (streamActive, portMAX_DELAY);
    }

    lDataLen = recv (lSocket, pcRxString, 5, 0);
    unsigned short desiredCount = atohex (pcRxString[0]) * 10000 +
        atohex (pcRxString[1]) * 1000 +
        atohex (pcRxString[2]) * 100 +
        atohex (pcRxString[3]) * 10 + atohex (pcRxString[4]);

    //We are using the status of the sampling LED to tell if we're still sampling right
    now
    if (gpio.get_pin_value (LED2_PIN) == 1)
    {
        //The LED is off, so we're not sampling If we're reset, there is no data
        //If we stopped because of a full buffer, the earlier data is gone.
        //Ignore the request and tell the PC to deal with it. It can decide how to
        proceed.
        send (lSocket, "NO", 3, 0);
        close (lSocket);
        xSemaphoreGive (streamActive);
        return;
    }

    //currentCount is the next count to be sent
    //It is the count in the current tcpipPacketRead pointer
    //because that pointer also points to the next packet to send
    signed long change = 0;
    int i;

    change = currentCount - desiredCount;
    //If it's negative, we need to wrap around modulo UINT16_MAX
    if (change < 0)
    {
        change = change + UINT16_MAX + 1;
    }
}

```



```

    }

    //Now change holds how many positions to rewind tcpipPacketRead
    signed long temptcpipPacketRead;

    temptcpipPacketRead = tcpipPacketRead - change;

    //If this is negative, wrap it around modulo NUM_PACKETS
    if (temptcpipPacketRead < 0)
    {
        temptcpipPacketRead = temptcpipPacketRead + NUM_PACKETS;
    }

    //We needed the +1 above because packet numbers range from 0 to UINT16_MAX
    //We do not need it here because packet indexes range from 0 to NUM_PACKETS - 1

    if (basePacket[temptcpipPacketRead].packetNumber != desiredCount)
    {
        //We already lost the data. Tell the PC and do nothing
        send (lSocket, "NO", 3, 0);
        close (lSocket);
        xSemaphoreGive (streamActive);
        return;
    }

    //We've still got the data. Update counters and proceed
    tcpipPacketRead = temptcpipPacketRead;

    //Fix the current count so that the DataStream packet header will be right
    currentCount = desiredCount;

    for (i = 0; i < change; i++)
    {
        xSemaphoreGive (PacketReadySemaphore);
    }
    //Allow datastream to start up again
    xSemaphoreGive (streamActive);
}

//GET command
if (!strcmp (pcRxString, "GETD", 4))
{
    lDataLen =
        recv (lSocket, &thisGetPacket, sizeof (thisGetPacket), 0);

    //After GET the next four chars are a bitmask of the channel pairs to send
    unsigned short channels = ntohs (thisGetPacket.channelbit);
    unsigned char precision = thisGetPacket.precision;
    unsigned long period = ntohl (thisGetPacket.period);

    //Ensure the ADCs and Datastream are really stopped
    if (xSemaphoreTake (streamActive, 1) == pdFALSE)
    {

```

```

        char message = 0;
        xQueueSend (stopstream, &message, portMAX_DELAY);
        xSemaphoreTake (streamActive, portMAX_DELAY);
    }

    SendStopMessage ();
    tcpipPacketRead = 0;

    //Configure the ADC with this information and start sampling
    //This is passed off to the SampleManager task for thread safety.
    //Because the samplemanager is of a higher priority, sending the
    //message will block this task until the ADCs are ready
    SendStartMessage (channels, precision, period);

    //Now permit the Datastream to run again
    xSemaphoreGive (streamActive);

}

//STOP command
if (!strncmp (pcRxString, "STOP", 4))
{
    //Stop datastream first

    if (xSemaphoreTake (streamActive, 1) == pdFALSE)
    {
        char message = 0;
        xQueueSend (stopstream, &message, portMAX_DELAY);
        xSemaphoreTake (streamActive, portMAX_DELAY);
    }
    SendStopMessage ();
    tcpipPacketRead = 0;
    xSemaphoreGive (streamActive);
}

    send (lSocket, "OK", 3, 0);
}
close (lSocket);
}

```

---

## F.1.8 InitBoard.h

---

```

/*!\ file Initboard.h */
#ifndef INITBOARD_H
#define INITBOARD_H

#define LED1_PIN AVR32_PIN_PB30
#define LED2_PIN AVR32_PIN_PB31

#ifdef INCLUDE_POST
void POST_LEDs (void);

```

```

int POST_SDRAM (void);
#endif

void SetupADCSPI (void);

void SetupADCTimer (unsigned short channels, unsigned char precision,
                    unsigned long period);

void StopADC (void);

extern volatile int transfer_not_finish ;

#endif /*INITBOARD.H */

```

---

## F.1.9 InitBoard.c

---

```

/** \file InitBoard.c
 * \brief Initializes board peripherals
 *
 * It helps set up the timer interrupt for sampling
 * ADCs.
 *
 */

/* Environment include files. */
#include <stdlib.h>
#include <string.h>
#include <avr32/io.h>
#include <stdint.h>

/* Scheduler include files . */
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"

#include "gpio.h"

#include "pwm.h"

#include "pdca.h"
#include "pm.h"
#include "eic.h"
#include "InitBoard.h"
#include "spi.h"
#include "usart.h"

#include "externalmem.h"
#include "samplemanager.h"
#include "DataStream.h"

#include "board.h"
#include "sdramc.h"

```

```

#include "rtc.h"

#include "debug.h"

/*! \brief The address in memory currently being filled
 *
 * Starts at 1 because the PDCA reload counter is at 1
 */
static unsigned long fillingPacket = 1;
static unsigned long thisPacket = 0;

/*! The actual number of channels we sampled */
static unsigned char numchannelsSampled = 0;

/*! Number of data points in a packet. Might be different if
 * number of channels does not divide evenly into a packet
 */
static unsigned short num_data_per_packet = 726;

static unsigned short packet_number = 0;

/*! \brief Tests for PWM silicon bug
 *
 * The current AVR32 PWM hardware resets to 0x0001 instead of 0x0000
 * In case it is fixed in the future, this routine tests the module.
 * It returns 1 if the bug is present and 0 if not
 */
static int testPWMReset (void);

/*! Holds output of testPWMReset for later usage */
static int buggyPWM = 0;

/*! \brief Setup information for ADC DMA transfer to ADC
 *
 * Needs size information to be filled in before use
 */
static pdca_channel_options_t PDCA_OPTIONS_SPI_TX = {
    .addr = (unsigned int) 0x00000000, // memory address. It's a dummy
    .pid = AVR32_PDCA_PID_SPI0_TX, //Transmit to ADC
    .r_addr = 0, // next memory address
    .r_size = 0, // next transfer counter
    .transfer_size = PDCA_TRANSFER_SIZE_HALF_WORD, // select size of the transfer
};

/*! \brief Setup information for ADC DMA transfer from ADC
 *
 * Needs size information to be filled in before use
 */
static pdca_channel_options_t PDCA_OPTIONS = {
    .addr = PacketStore[0].data,
    .pid = AVR32_PDCA_PID_SPI0_RX, //Incoming data from SPI
    .r_addr = PacketStore[1].data,
    .transfer_size = PDCA_TRANSFER_SIZE_HALF_WORD, // select size of the transfer
    // because each transfer is 2 bytes)
};

```

```

};

/*! \brief The PDCA interrupt handler.
 *
 * The handler to reload the PDCA settings after each time the buffer fills
 * This corresponds to a whole Ethernet packet
 */
#if __GNUC__
__attribute__((__interrupt__))
#elif __ICCAVR32__
#pragma handler = AVR32_PDCA_IRQ_GROUP, 0
__interrupt
#endif
static void
pdca_int_handler (void)
{
    //Stamp the ID number on this packet before calling the transmitter
    //thisPacket is one less than packetNumber, but it's a separate variable so that
    //we can avoid the modulus mess with fillingPacket
    PacketStore[thisPacket].packetNumber = packet_number;
    packet_number++;

    thisPacket++;
    if (thisPacket == 8)
    {
        thisPacket = 0;
    }

    fillingPacket++;
    if (fillingPacket == 8)
    {
        fillingPacket = 0;
    }

    //This is my makeshift semaphore. The FreeRTOS one was too slow.
    ADCReadySemaphore = ADCReadySemaphore + 1;

    pdca_reload_channel (0, PacketStore[fillingPacket].data,
                        num_data_per_packet);
}

/*! \brief The ADC "BUSY" interrupt handler.
 *
 * This starts the SPI transfer to get data from the ADC. It is the highest
 * priority, and it is essential that it be serviced fast.
 */
#if __GNUC__
__attribute__((__interrupt__))
#endif
static void adc_busy_handler (void)
{
    eic_clear_interrupt_line (&AVR32_EIC, EXT_INT7);
}

```

```

//We need to detect if we're out of sync with packets
//if the amount remaining in pdca modulo the number of channels sampled
//is not 0, we have trouble. Grab and dump the remaining channels to get back
//in sync.
if (pdca_get_load_size (0) % numchannelsSampled == 0)
{
pdca_load_channel (1, (void *) 0x00000000, numchannelsSampled);
}
else
{
//Trouble
pdca_load_channel (1, (void *) 0x000000,
pdca_get_load_size (0) % numchannelsSampled);
}

//Re-initialize channel 1 to send dummy data to the ADC
pdca_enable (1);
}

#if (INCLUDE_POST == 1)
/*! \brief Blink the LEDs in a distinctive pattern.
*
* This gives a quick visual verification that something is working.
* It delays bootup so is usually disabled
*/
void
POST_LEDs (void)
{
unsigned long volatile currentRTC;

//Turn on each LED for one second
gpio_enable_gpio_pin (LED1_PIN);
gpio_enable_gpio_pin (LED2_PIN);

gpio_clr_gpio_pin (LED1_PIN);
gpio_set_gpio_pin (LED2_PIN);

//Now use the RTC to delay for a bit.

currentRTC = rtc_get_value (&AVR32_RTC);
while (currentRTC + 1 >= rtc_get_value (&AVR32_RTC));

gpio_clr_gpio_pin (LED2_PIN);

currentRTC = rtc_get_value (&AVR32_RTC);
while (currentRTC + 1 >= rtc_get_value (&AVR32_RTC));
gpio_set_gpio_pin (LED1_PIN);

currentRTC = rtc_get_value (&AVR32_RTC);
while (currentRTC + 1 >= rtc_get_value (&AVR32_RTC));

```

```

    gpio_set_gpio_pin (LED2_PIN);

    currentRTC = rtc_get_value (&AVR32_RTC);
    while (currentRTC + 1 >= rtc_get_value (&AVR32_RTC));
}

typedef uint16_t datum;      /* Set the data bus width to 32 bits. */

/*! \brief Perform memory test on the data bus
*/
datum
memTestDataBus (volatile datum * address)
{
    datum pattern;

    /*
     * Perform a walking 1's test at the given address.
     */

    for (pattern = 1; pattern != 0; pattern <<= 1)
    {
        /*
         * Write the test pattern.
         */
        *address = pattern;
        /*
         * Read it back (immediately is okay for this test).
         */
        if (*address != pattern)
        {
            return (pattern);
        }
    }
    return (0);
} /* memTestDataBus() */

/*! \brief perform memory test on address bus
*/
datum *
memTestAddressBus (volatile datum * baseAddress, unsigned long nBytes)
{
    unsigned long addressMask = ((nBytes / sizeof (datum)) - 1);
    unsigned long offset;
    unsigned long testOffset;
    datum pattern = (datum) 0xAAAAAAAA;
    datum antipattern = (datum) 0x55555555;

    /*
     * Write the default pattern at each of the power-of-two offsets.
     */
    for (offset = 1; (offset & addressMask) != 0; offset <<= 1)
    {
        baseAddress[offset] = pattern;
    }
}

```

```

    }

    /*
     * Check for address bits stuck high.
     */

    testOffset = 0;
    baseAddress[testOffset] = antipattern;

    for ( offset = 1; ( offset & addressMask) != 0; offset <<= 1)
    {
        if (baseAddress[offset] != pattern)
        {
            return ((datum *) & baseAddress[offset]);
        }
    }

    baseAddress[testOffset] = pattern;

    /*
     * Check for address bits stuck low or shorted.
     */

    for ( testOffset = 1; ( testOffset & addressMask) != 0; testOffset <<= 1)
    {

        baseAddress[testOffset] = antipattern;
        for ( offset = sizeof (datum); (offset & addressMask) != 0; offset <<= 1)
        {
            if ((baseAddress[offset] != pattern) && (offset != testOffset))
            {
                return ((datum *) & baseAddress[testOffset]);
            }
        }
        baseAddress[testOffset] = pattern;
    }

    return (NULL);

}                                     /* memTestAddressBus() */

/*! \brief Test the entire SDRAM device
 */
datum *
memTestDevice (datum volatile *baseAddress, unsigned long nBytes)
{
    unsigned long offset;
    unsigned long nWords = nBytes / sizeof (datum);
    datum pattern;
    datum antipattern;

    /*
     * Fill memory with a known pattern.
     */

```



```

for (pattern = 1, offset = 0; offset < nWords; pattern++, offset++)
{
    baseAddress[offset] = pattern;
}

/*
 * Check each location and invert it for the second pass.
 */

for (pattern = 1, offset = 0; offset < nWords; pattern++, offset++)
{
    if (baseAddress[offset] != pattern)
    {
        return ((datum *) & baseAddress[offset]);
    }
    antipattern = ~pattern;
    baseAddress[offset] = antipattern;
}

/*
 * Check each location for the inverted pattern and zero it.
 */

for (pattern = 1, offset = 0; offset < nWords; pattern++, offset++)
{
    antipattern = ~pattern;
    if (baseAddress[offset] != antipattern)
    {
        return ((datum *) & baseAddress[offset]);
    }
    baseAddress[offset] = 0;
}

return (NULL);

}                                     /* memTestDevice() */

/*! \brief Run all memory tests in sequence.
 *
 * Illuminate LEDs to indicate problems
 */
int
POST_SDRAM (void)
{
    gpio_set_gpio_pin (LED1_PIN);
#define BASE_ADDRESS (volatile datum *) SDRAM_START
#define NUM_BYTES (32 * 1024 * 1024)

    datum *errorLocation;
    int busBitError;
    int i = 0;

```

```

busBitError = memTestDataBus (BASE_ADDRESS);

if (busBitError != 0)
{
    for (i = 1; i < busBitError; i <<= 1)
    {
        //If necessary to debug a data bit, uncomment this line
        //POST_LEDS();
    }
    gpio_clr_gpio_pin (LED2_PIN);
    gpio_set_gpio_pin (LED1_PIN);
    return (-1);
}

errorLocation = memTestAddressBus (BASE_ADDRESS, NUM_BYTES);

if (errorLocation != NULL)
{
    gpio_clr_gpio_pin (LED1_PIN);
    gpio_set_gpio_pin (LED2_PIN);
    return (-1);
}

errorLocation = memTestDevice (BASE_ADDRESS, NUM_BYTES);
if (errorLocation != NULL)
{
    gpio_clr_gpio_pin (LED1_PIN);
    gpio_clr_gpio_pin (LED2_PIN);
    return (-1);
}

return (0);
}
#endif

/*! \brief Configures the SPI bus to talk to the ADC.
 *
 * Initializes the interrupts and SPI bus for use. Should be called once
 * before any sampling occurs.
 */
void
SetupADCSPI (void)
{
    static const spi_options_t spiOptions = {
        /*! The SPI channel to set up.
        .reg = 0,
        /*! Preferred baudrate for the SPI.
        /*Internally will hit this or round up to next bit of PBA
        /*The ADCs can do 18 MHz, but PBA is running at 66 MHz.
        /*This means that the baudrate can be 16.5 MHz.
        /*I limit it to a lower level to increase reliability
        .baudrate = 16500000,
        /*! Number of bits in each character (8 to 16).

```

```

    .bits = 16,
    ///! Delay before first clock pulse after selecting slave
    ///! (in PBA periods, or 32 x TPBA with FDIV set).
    .spck_delay = 4,          ///! Delay between each transfer/character
    ///! (in PBA periods, or 32 x TPBA with FDIV set).
    .trans_delay = 4,
    ///! Sets this chip to stay active after last transfer to it.
    .stay_act = 0,
    ///! Which SPI mode to use when transmitting.
    .spi_mode = 2,
    ///! Disables the mode fault detection.
    ///! With this bit cleared, the SPI master mode will disable itself if another
    ///! master tries to address it.
    .modfdis = 0
};

///Map to assign SPI pins to SPI controller
static const gpio_map_t ADC_SPI_GPIO_MAP = {
    {AVR32_SPI0_SCK_0_0_PIN, AVR32_SPI0_SCK_0_0_FUNCTION}, /// SPI Clock.
    {AVR32_SPI0_MISO_0_0_PIN, AVR32_SPI0_MISO_0_0_FUNCTION}, /// MISO.
    {AVR32_SPI0_MOSI_0_0_PIN, AVR32_SPI0_MOSI_0_0_FUNCTION}, /// MOSI.
    {AVR32_SPI0_NPCS_0_0_PIN, AVR32_SPI0_NPCS_0_0_FUNCTION} /// Chip Select NPCS.
};

/// Assign I/Os to SPI
gpio_enable_module (ADC_SPI_GPIO_MAP,
                   sizeof (ADC_SPI_GPIO_MAP) /
                   sizeof (ADC_SPI_GPIO_MAP[0]));

/// Initialize as master
spi_initMaster (&AVR32_SPI0, &spiOptions);

/// Set selection mode: variable_ps, pcs.decode, delay
spi_selectionMode (&AVR32_SPI0, 0, 0, 5);

/// setup chip registers
spi_setupChipReg (&AVR32_SPI0, &spiOptions, 66000000); ///66 MHz going to this device
from PBA
/// Enable SPI
spi_enable (&AVR32_SPI0);

/// Select the ADC. Since it is the only peripheral, it can just stay selected.
spi_selectChip (&AVR32_SPI0, 0);

///Options for External Interrupt Controller
static const eic_options_t eic_options = {
    /// Enable edge-triggered interrupt.
    .eic_mode = EIC_MODE_EDGE_TRIGGERED,
    /// Interrupt will trigger on falling edge.
    .eic_edge = EIC_EDGE_FALLING_EDGE,
    /// Initialize in synchronous mode : interrupt is synchronized to the clock
    .eic_async = EIC_SYNCH_MODE,
    /// Set the interrupt line number.
    .eic_line = EXT_INT7,
};

```

```

    . eic_filter = EIC_FILTER_ENABLED,
};

//Give pin to External Interrupt Controller
gpio_enable_module_pin (AVR32_EIC_EXTINT_7_PIN,
                       AVR32_EIC_EXTINT_7_FUNCTION);

//The following is OK because interrupts are disabled

//Note: FreeRTOS tick is at INT0 priority.
//My interrupts must be that priority to use FreeRTOS
//API calls without special handling.

INTC_register_interrupt (( _int_handler ) & pdca_int_handler,
                        AVR32_PDCA_IRQ_0, AVR32_INTC_INT2);

//The ADC sample interrupt is extremely important to do on schedule.
//Since the interrupt
//just modifies a peripheral, it can fire even in so-called "critical "
//regions. This is because
//it will never cause a context change and has no effect on FreeRTOS.
//portENTER_CRITICAL has been modified so this interrupt is NEVER masked.
INTC_register_interrupt (( _int_handler ) & adc_busy_handler,
                        AVR32_EIC_IRQ_7, AVR32_INTC_INT3);

eic_init (&AVR32_EIC, &eic_options, 1);

eic_enable_line (&AVR32_EIC, eic_options.eic_line);
eic_enable_interrupt_line (&AVR32_EIC, eic_options.eic_line);

//Finally test the PWM module for bugginess
buggyPWM = testPWMReset ();
}

/*! \brief Test for buggy PWM silicon
 *
 * Determines whether this chip suffers from the reset to 0x0001 instead of
 * 0x0000 bug.
 */
static int
testPWMReset (void)
{
    int retval;

    pwm_opt_t pwm_opt;          // PWM option config.
    avr32_pwm_channel_t pwm_channel; // One channel config.

    // PWM controller configuration.
    pwm_opt.diva = AVR32_PWM_DIVA_CLK_OFF;
    pwm_opt.divb = AVR32_PWM_DIVB_CLK_OFF;
    pwm_opt.prea = AVR32_PWM_PREA_MCK;
    pwm_opt.preb = AVR32_PWM_PREB_MCK;

```

```

pwm_init (&pwm_opt);

pwm_channel.CMR.calg = PWM_MODE_LEFT_ALIGNED; // Channel mode.
pwm_channel.CMR.cpol = PWM_POLARITY_LOW; // Channel polarity.
pwm_channel.CMR.cpd = PWM_UPDATE_DUTY; // Not used the first time.
pwm_channel.CMR.cpre = AVR32_PWM_CMR_CPRE_MCK_DIV_16; // Channel prescaler.
pwm_channel.cdy = 1; // Channel duty cycle, should be < CPRD.
pwm_channel.cprd = 5; // Channel period.
pwm_channel.cupd = 0; // Channel update is not used here.

pwm_channel_init (1, &pwm_channel);

volatile avr32_pwm_t *pwm = &AVR32_PWM;

pwm_start_channels (0x02); // Start appropriate channels (ch 1)

//Now we monitor for the rollover problem

//Wait until the counter has passed the 0 and 1 points
while (pwm->channel[1].ccnt == 0x0000);

while (pwm->channel[1].ccnt == 0x0001);

//Now we wait for the rollover
while (1)
{
    if (pwm->channel[1].ccnt == 0x0000)
    {
        //No bug.
        retval = 0;
        break;
    }
    if (pwm->channel[1].ccnt == 0x0001)
    {
        //Bug present
        retval = 1;
        break;
    }
}

pwm_stop_channels (0x002);

return retval;
}

/*! \brief Starts ADC sampling
*
* This starts the ADC sample timer and begins gathering samples. It needs
* information on which channels to sample and how fast.
*/
void
SetupADCTimer (unsigned short channels, unsigned char precision,
               unsigned long period)

```

```

{
unsigned short sampledchannelmask = 0;
unsigned short desiredchannelmask = 0;
unsigned short num_groups_per_packet = 726;
taskDISABLE_INTERRUPTS ();
desiredchannelmask = channels;

//Enable the RESET to the ADC
gpio_enable_gpio_pin (AVR32_PIN_PB29);

//Enable the RANGE to the ADC
gpio_enable_gpio_pin (AVR32_PIN_PB17);
gpio_enable_gpio_pin (AVR32_PIN_PB23);

//Pull RESET HIGH for ADC
gpio_set_gpio_pin (AVR32_PIN_PB29);

//Now set the RANGE pin
if (precision & 0x1)
{
    gpio_set_gpio_pin (AVR32_PIN_PB17); //Lower it for maximum range
}
else
{
    gpio_clr_gpio_pin (AVR32_PIN_PB17);
}
if (precision & 0x2)
{
    gpio_set_gpio_pin (AVR32_PIN_PB23);
}
else
{
    gpio_clr_gpio_pin (AVR32_PIN_PB23);
}

//All channels lower than the highest channel get sampled.
sampledchannelmask = (desiredchannelmask & 0x01 ? 0x1 : 0) |
    (desiredchannelmask & 0x02 ? 0x03 : 0) |
    (desiredchannelmask & 0x04 ? 0x07 : 0) |
    (desiredchannelmask & 0x08 ? 0x0F : 0) |
    (desiredchannelmask & 0x10 ? 0x1F : 0) |
    (desiredchannelmask & 0x20 ? 0x3F : 0) |
    (desiredchannelmask & 0x40 ? 0x7F : 0) |
    (desiredchannelmask & 0x80 ? 0xFF : 0) |
    (desiredchannelmask & 0x100 ? 0x1FF : 0) |
    (desiredchannelmask & 0x200 ? 0x3FF : 0) |
    (desiredchannelmask & 0x400 ? 0x7FF : 0) |
    (desiredchannelmask & 0x800 ? 0xFFF : 0);

numchannelsSampled = (sampledchannelmask & 0x01 ? 1 : 0) +
    (sampledchannelmask & 0x02 ? 1 : 0) +
    (sampledchannelmask & 0x04 ? 1 : 0) +
    (sampledchannelmask & 0x08 ? 1 : 0) +

```

```

(sampledchannelmask & 0x10 ? 1 : 0) +
(sampledchannelmask & 0x20 ? 1 : 0) +
(sampledchannelmask & 0x40 ? 1 : 0) +
(sampledchannelmask & 0x80 ? 1 : 0) +
(sampledchannelmask & 0x100 ? 1 : 0) +
(sampledchannelmask & 0x200 ? 1 : 0) +
(sampledchannelmask & 0x400 ? 1 : 0) +
(sampledchannelmask & 0x800 ? 1 : 0);

num_groups_per_packet = NUM_SAMPLES_PER_PACKET / numchannelsSampled;
num_data_per_packet = num_groups_per_packet * numchannelsSampled;

resetDataStream ();

PDCA_OPTIONS_SPI_TX.size = numchannelsSampled;
pdca_init_channel (1, &PDCA_OPTIONS_SPI_TX); //Init Channel 1

PDCA_OPTIONS.size = num_data_per_packet;
PDCA_OPTIONS.r.size = num_data_per_packet;
PDCA_OPTIONS.addr = PacketStore[0].data;
PDCA_OPTIONS.r.addr = PacketStore[1].data;
pdca_init_channel (0, &PDCA_OPTIONS); // init PDCA channel with options.

pdca_enable_interrupt_reload_counter_zero (0);

// Enable the transfer (but will not do anything without timer to hit
//sample lines)
pdca_enable (0);

//Configure the PWM module
//Give PWM pins to module from GPIO

gpio_enable_module_pin (AVR32_PWM_0_PIN, AVR32_PWM_0_FUNCTION);

gpio_enable_module_pin (AVR32_PWM_2_PIN, AVR32_PWM_2_FUNCTION);

gpio_enable_module_pin (AVR32_PWM_3_PIN, AVR32_PWM_3_FUNCTION);

gpio_enable_module_pin (AVR32_PWM_4_1_PIN, AVR32_PWM_4_1_FUNCTION);

gpio_enable_module_pin (AVR32_PWM_5_1_PIN, AVR32_PWM_5_1_FUNCTION);

gpio_enable_module_pin (AVR32_PWM_6_PIN, AVR32_PWM_6_FUNCTION);

//The PWM is connected to Peripheral Bus A. This bus has a clock speed
//of 66 MHz looking at main.c and its power manager configuration.
//I want the waves to go low briefly at either the beginning or end of the
//wave.
//For now, no division of the clock. I could add the ability to sample slower

pwm_opt.t pwm_opt; // PWM option config.
avr32_pwm_channel.t pwm_channel; // One channel config.

```

```

// PWM controller configuration.
pwm_opt.diva = AVR32_PWM_DIVA_CLK_OFF;
pwm_opt.divb = AVR32_PWM_DIVB_CLK_OFF;
pwm_opt.prea = AVR32_PWM_PREA_MCK;
pwm_opt.preb = AVR32_PWM_PREB_MCK;

pwm_init (&pwm_opt);

pwm_channel.CMR.calg = PWM_MODE_LEFT_ALIGNED; // Channel mode.
pwm_channel.CMR.cpol = PWM_POLARITY_LOW; // Channel polarity.
pwm_channel.CMR.cpd = PWM_UPDATE_DUTY; // Not used the first time.
pwm_channel.CMR.cpre = AVR32_PWM_CMR_CPRE_MCK; // Channel prescaler.
pwm_channel.cdy = 5; // Channel duty cycle, should be < CPRD.
pwm_channel.cprd = period + buggyPWM; // Channel period.
pwm_channel.cupd = 0; // Channel update is not used here.

pwm_channel_init (0, &pwm_channel); // Set channel configuration to channel 0.
pwm_channel_init (2, &pwm_channel); // Set channel configuration to channel 2.
pwm_channel_init (3, &pwm_channel); // Set channel configuration to channel 3.
pwm_channel_init (4, &pwm_channel); // Set channel configuration to channel 4.
pwm_channel_init (5, &pwm_channel); // Set channel configuration to channel 5.
pwm_channel_init (6, &pwm_channel); // Set channel configuration to channel 6.

gpio_clr_gpio_pin (AVR32_PIN_PB29); //De-assert ADC RESET line

pwm_start_channels (0x7D);
taskENABLE_INTERRUPTS ();
}

/* \brief Halt ADC conversions and reset the device.
*/
void
StopADC (void)
{
taskDISABLE_INTERRUPTS ();

pwm_stop_channels (0x7D); // Stop all channels (this function takes a bitmask).

//Pull RESET HIGH for ADC
gpio_set_gpio_pin (AVR32_PIN_PB29);

//Shut off SPI transfer to ADC
pdca_disable (1);

//Shut off SPI transfers from ADC
pdca_disable (0);

//Reset counter
fillingPacket = 1;
thisPacket = 0;
packet_number = 0;

resetDataStream ();

```



```
taskENABLE_INTERRUPTS ();  
}
```

---

## F.1.10 ethernet.h

---

```
/* This header file is part of the ATMEL AVR32-SoftwareFramework-AT32UC3A-1.4.0  
Release */
```

```
/* This file has been prepared for Doxygen automatic documentation generation.*/
```

```
/*! \file *****
```

```
*
```

```
* \brief ethernet headers for AVR32 UC3.
```

```
*
```

```
* - Compiler: IAR EWAVR32 and GNU GCC for AVR32
```

```
* - Supported devices: All AVR32 devices can be used.
```

```
* - AppNote:
```

```
*
```

```
* \author Atmel Corporation: http://www.atmel.com \n
```

```
*
```

```
Support and FAQ: http://support.atmel.no/
```

```
*
```

```
*****/
```

```
/* Copyright (C) 2006-2008, Atmel Corporation All rights reserved.
```

```
*
```

```
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:
```

```
*
```

```
* 1. Redistributions of source code must retain the above copyright notice,  
* this list of conditions and the following disclaimer.
```

```
*
```

```
* 2. Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution .
```

```
*
```

```
* 3. The name of ATMEL may not be used to endorse or promote products derived  
* from this software without specific prior written permission.
```

```
*
```

```
* THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR  
* IMPLIED
```

```
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES  
* OF
```

```
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY  
* AND
```

```
* SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY  
* DIRECT,
```

```
* INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES  
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR  
* SERVICES;
```

```
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER  
* CAUSED AND
```

```

* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
  USE OF
* THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

```

```

#ifndef ETHERNET_H
#define ETHERNET_H

```

```

#include "arch/cc.h"
#include "lwip/ip_addr.h"

```

```

/*!
* Struct definition for holding configuration information for the network
* interface
*/

```

```

typedef struct
{
    char dhcpenable;
    char addr0;
    char addr1;
    char addr2;
    char addr3;
    char netmask0;
    char netmask1;
    char netmask2;
    char netmask3;
    char gateway0;
    char gateway1;
    char gateway2;
    char gateway3;
    struct ip_addr ipaddr;
    struct ip_addr netmask;
    struct ip_addr gateway;
    u8_t mac[3];
} MACinterfaceparams;

```

```

/*! \brief Create the vStartEthernetTask task.
*
* \param uxPriority Input; priority of the task to create.
*
*/

```

```

void vStartEthernetTaskLauncher (unsigned portBASE_TYPE uxPriority);

```

```

/*! \brief create ethernet task, for ethernet management.
*
* \param pvParameters Input; not used.
*
*/

```

```

portTASK_FUNCTION (vStartEthernetTask, pvParameters);

```

#endif

---

## F.1.11 ethernet.c

---

*/\* This source file is part of the ATMEL AVR32-SoftwareFramework-AT32UC3A-1.4.0  
Release \*/*

*/\*This file has been prepared for Doxygen automatic documentation generation.\*/*

*/\*! \file \*\*\*\*\**

*\**

*\* \brief ethernet management for AVR32 UC3.*

*\**

*\* - Compiler: IAR EWAVR32 and GNU GCC for AVR32*

*\* - Supported devices: All AVR32 devices can be used.*

*\* - AppNote:*

*\**

*\* \author Atmel Corporation: <http://www.atmel.com> \n*

*\* Support and FAQ: <http://support.atmel.no/>*

*\**

*\*\*\*\*\*/*

*/\* Copyright (C) 2006-2008, Atmel Corporation All rights reserved.*

*\**

*\* Redistribution and use in source and binary forms, with or without  
\* modification, are permitted provided that the following conditions are met:*

*\**

*\* 1. Redistributions of source code must retain the above copyright notice,  
\* this list of conditions and the following disclaimer.*

*\**

*\* 2. Redistributions in binary form must reproduce the above copyright notice,  
\* this list of conditions and the following disclaimer in the documentation  
\* and/or other materials provided with the distribution .*

*\**

*\* 3. The name of ATMEL may not be used to endorse or promote products derived  
\* from this software without specific prior written permission.*

*\**

*\* THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR  
\* IMPLIED*

*\* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES  
\* OF*

*\* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY  
\* AND*

*\* SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY  
\* DIRECT,*

*\* INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES  
\* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR  
\* SERVICES;*

*\* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER  
\* CAUSED AND*

```

* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
  USE OF
* THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

```

```
#include <string.h>
```

```
#include "gpio.h"          // Have to include gpio.h before FreeRTOS.h as long as
  FreeRTOS                // redefines the inline keyword to empty.
```

```
/* Scheduler include files . */
#include "FreeRTOS.h"
#include "task.h"
```

```
/* Demo program include files. */
#include "conf_lwip_threads.h"
```

```
/* ethernet includes */
#include "ethernet.h"
#include "conf_eth.h"
#include "macb.h"
```

```
#include "DataStream.h"
```

```
#ifdef ENABLE_SERIAL
#include "serialport.h"
#endif
```

```
/* lwIP includes */
#include "lwip/sys.h"
#include "lwip/api.h"
#include "lwip/tcpip.h"
#include "lwip/memp.h"
#include "lwip/stats.h"
#include "netif/loopif.h"
```

```
//----- M A C R O S -----
```

```
//----- D E F I N I T I O N S -----
```

```
/* global variable containing MAC Config (hw addr, IP, GW, ...) */
struct netif MACB_if;
```

```
MACinterfaceparams ethernetconfig;
```

```
//----- D E C L A R A T I O N S -----
```

```

/* Initialisation required by lwIP. */
static void prvlwIPInit (void);

/* Initialisation of ethernet interfaces by reading config file */
static void prvEthernetConfigureInterface (void *param);

/*! \brief Small task to launch lwIP
 *
 * Responsible for spawning the various servers after lwIP is initialized
 *
 * \param uxPriority sets priority of the launcher task
 */
void
vStartEthernetTaskLauncher (unsigned portBASE_TYPE uxPriority)
{
    /* Spawn the Sentinel task. */
    xTaskCreate (vStartEthernetTask, (const signed portCHAR *) "ETHLAUNCH",
                configMINIMAL_STACK_SIZE, NULL, uxPriority,
                (xTaskHandle *) NULL);
}

/*! \brief Ethernet task, for ethernet management.
 *
 */
portTASK_FUNCTION (vStartEthernetTask, pvParameters)
{
    static const gpio_map_t MACB_GPIO_MAP = {
        {AVR32_MACB_MDC_0_PIN, AVR32_MACB_MDC_0_FUNCTION},
        {AVR32_MACB_MDIO_0_PIN, AVR32_MACB_MDIO_0_FUNCTION},
        {AVR32_MACB_RXD_0_PIN, AVR32_MACB_RXD_0_FUNCTION},
        {AVR32_MACB_TXD_0_PIN, AVR32_MACB_TXD_0_FUNCTION},
        {AVR32_MACB_RXD_1_PIN, AVR32_MACB_RXD_1_FUNCTION},
        {AVR32_MACB_TXD_1_PIN, AVR32_MACB_TXD_1_FUNCTION},
        {AVR32_MACB_TX_EN_0_PIN, AVR32_MACB_TX_EN_0_FUNCTION},
        {AVR32_MACB_RX_ER_0_PIN, AVR32_MACB_RX_ER_0_FUNCTION},
        {AVR32_MACB_RX_DV_0_PIN, AVR32_MACB_RX_DV_0_FUNCTION},
        {AVR32_MACB_TX_CLK_0_PIN, AVR32_MACB_TX_CLK_0_FUNCTION}
    };

    // Assign GPIO to MACB
    gpio_enable_module (MACB_GPIO_MAP,
                       sizeof (MACB_GPIO_MAP) / sizeof (MACB_GPIO_MAP[0]));

    /* Setup lwIP. */
    prvlwIPInit ();

    sys_thread_new ("DSTRM", vDataStreamServer, (void *) NULL,
                    ethDATASTREAM_SERVER_STACK_SIZE,
                    ethDATASTREAMSERVER_PRIORITY);
    sys_thread_new ("AUTOD", vAutodetectServer, (void *) NULL,
                    AUTOD_STACK_SIZE, AUTOD_PRIORITY);
    sys_thread_new ("CMD", vCommandServer, (void *) NULL,
                    COMMAND_STACK_SIZE, COMMAND_PRIORITY);
}

```

```

#ifndef ENABLE_SERIAL
    sys_thread_new ("SERIAL", SerialServer, (void *) NULL,
                    COMMAND_STACK_SIZE, 1);
#endif

    // Kill this launcher task.
    vTaskDelete (NULL);
}

//! Callback executed when the TCP/IP init is done.
static void
tcpip_init_done (void *arg)
{
    sys_sem_t *sem;
    sem = (sys_sem_t *) arg;
    sys_sem_signal (*sem);    // Signal the waiting thread that the TCP/IP init is done.
}

/*!
 * \brief start lwIP layer.
 */
static void
prvlwIPInit (void)
{
    sys_sem_t sem;
    int uswvalue;

    sem = sys_sem_new (0);    // Create a new semaphore.
    tcpip_init (tcpip_init_done, &sem);
    sys_sem_wait (sem);    // Block until the lwIP stack is initialized .
    sys_sem_free (sem);    // Free the semaphore.

    //Now Init the GPIO pins for the DIP switches
    gpio_enable_gpio_pin (AVR32_PIN_PX16);    //USW1
    gpio_enable_gpio_pin (AVR32_PIN_PX19);    //USW2
    gpio_enable_gpio_pin (AVR32_PIN_PX22);    //USW3

    uswvalue = (gpio_get_pin_value (AVR32_PIN_PX16) ? 0 : 1) +
        (2 * (gpio_get_pin_value (AVR32_PIN_PX19) ? 0 : 1)) +
        (4 * (gpio_get_pin_value (AVR32_PIN_PX22) ? 0 : 1));

    userpagedata *configdata = (userpagedata *) USERPAGE_BASE_ADDR;

    //These are bytes 4-6 of the MAC
    ethernetconfig.mac[0] = configdata->mac[0];
    ethernetconfig.mac[1] = configdata->mac[1];
    ethernetconfig.mac[2] = configdata->mac[2];

    //Use the switch value to select a configuration
    if (uswvalue != 7)
    {
        ethernetconfig.dhcpenable = 0;
    }
}

```

```

        ethernetconfig.ipaddr.addr = configdata->ipaddr[uswvalue];
        ethernetconfig.gateway.addr = configdata->gateway[uswvalue];
        ethernetconfig.netmask.addr = configdata->netmask[uswvalue];
    }
else
    {
        ethernetconfig.dhcpenable = 1;
    }

    /* Set hw and IP parameters, initialize MACB too */
    prvEthernetConfigureInterface (&ethernetconfig);
}

/*!
 * \brief set ethernet config
 */
static void
prvEthernetConfigureInterface (void *param)
{
    extern err_t ethernetif_init (struct netif *netif);
    unsigned portCHAR MacAddress[6];
    MACinterfaceparams *ethernetconfig = param;

    /* Default MAC addr. */
    MacAddress[0] = ETHERNET_CONF_ETHADDR0;
    MacAddress[1] = ETHERNET_CONF_ETHADDR1;
    MacAddress[2] = ETHERNET_CONF_ETHADDR2;
    MacAddress[3] = ethernetconfig->mac[0];
    MacAddress[4] = ethernetconfig->mac[1];
    MacAddress[5] = ethernetconfig->mac[2];

    /* pass the MAC address to MACB module */
    vMACBSetMACAddress (MacAddress);

    netifapi_netif_add (&MACB_if, &ethernetconfig->ipaddr,
                       &ethernetconfig->netmask, &ethernetconfig->gateway,
                       NULL, ethernetif_init, tcpip_input);

    /* make it the default interface */
    netifapi_netif_set_default (&MACB_if);

    if (ethernetconfig->dhcpenable)
    {
        netifapi_dhcp_start (&MACB_if);
    }
else
    {
        /* bring it up */
        netifapi_netif_set_up (&MACB_if);
    }
}

```

---

## F.1.12 samplemanager.h

---

```
/*! \file samplemanager.h */
#ifndef SAMPLEMANAGER_H_
#define SAMPLEMANAGER_H_

/*! The structure of a message to the samplemanager */
typedef struct
{
    char command;
    unsigned short channels;
    unsigned char precision;
    unsigned long period;
} managerMessage;

//Possible commands
#define STARTSAMPLING 0x00
#define STOPSAMPLING 0x01

extern xQueueHandle managerMbox;

void SendStopMessage (void);

void SendStartMessage (unsigned short channels, unsigned char precision,
                       unsigned long period);

void vStartSampleManagerTask (void);

#endif /*SAMPLEMANAGER_H_ */
```

---

## F.1.13 samplemanager.c

---

```
/*! \file samplemanager.c */

/* Environment include files. */
#include <stdlib.h>
#include <string.h>

#include <avr32/io.h>
#include "compiler.h"
#include "preprocessor.h"

#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"
#include "queue.h"

#include "samplemanager.h"
#include "InitBoard.h"
#include "gpio.h"
#include "conf_lwip_threads.h"
```



```

static portTASK_FUNCTION_PROTO (vSampleManagerTask, pvParameters);

/*! Mailbox for the sample manager */
xQueueHandle managerMbox;

/*! Start the sample manager */
void
vStartSampleManagerTask (void)
{
    //Mailbox will have 5 slots each holding an object with
    //enough space to hold a managerMessage
    managerMbox = xQueueCreate (5, sizeof (managerMessage));
    gpio_enable_gpio_pin (LED2_PIN);

    xTaskCreate (vSampleManagerTask, (const signed portCHAR * const) "SAMP",
                configMINIMAL_STACK_SIZE, NULL, SAMPLEMANAGER_PRIORITY,
                (xTaskHandle *) NULL);
}

/*! \brief Send a stop message
 *
 * This sends a message to the sample manager to halt sampling and reset the
 * ADCs.
 */
void
SendStopMessage (void)
{
    managerMessage message;
    message.command = STOPSAMPLING;
    xQueueSend (managerMbox, &message, portMAX_DELAY);
    return;
}

/*! \brief Start sampling
 *
 * Internally this calls SetupADCTimer, but it ensures that it is done
 * in a safe way with other tasks not bothering it. It also manages
 * the sampling LED properly
 */
void
SendStartMessage (unsigned short channels, unsigned char precision,
                  unsigned long period)
{
    managerMessage message;
    message.command = STARTSAMPLING;
    message.channels = channels;
    message.precision = precision;
    message.period = period;
    xQueueSend (managerMbox, &message, portMAX_DELAY);
    return;
}

/*!

```

```

* \brief Sample manager task
*
* The sample manager exists to start and stop sampling in one central way
* by passing messages to it. It is high priority so that it will take priority
* over other tasks doing sampling and packet processing. It's main purpose
* is to allow interrupt routines to stop sampling if they detect a buffer
* overflow.
*/
static
portTASK_FUNCTION (vSampleManagerTask, pvParameters)
{
    managerMessage messagebuffer;

    while (1)
    {

        //Pend forever on messages
        xQueueReceive (managerMbox, &messagebuffer, portMAX_DELAY);

        //Process received message
        switch (messagebuffer.command)
        {
            case STARTSAMPLING:
                StopADC ();
                //Configure the ADC with this information and start sampling
                SetupADCTimer (messagebuffer.channels, messagebuffer.precision,
                    messagebuffer.period);
                gpio_clr_gpio_pin (LED2_PIN);
                break;
            case STOPSAMPLING:
                StopADC ();
                gpio_set_gpio_pin (LED2_PIN);
                break;
            default:
                break;
        }
    }
}

```

---

## F.1.14 serialport.h

---

```

/*! \file serialport.h */
#ifndef SERIALPORT_H_
#define SERIALPORT_H_

extern xQueueHandle serialChars;

portTASK_FUNCTION_PROTO (SerialServer, pvParameters);

#endif /*SERIALPORT_H_ */

```

---

## F.1.15 serialport.c

---

```
#ifndef ENABLE_SERIAL

/* Standard includes. */
#include <stdio.h>
#include <string.h>
#include <stdint.h>

#include "conf.eth.h"

/* Scheduler includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"

/* Demo includes. */
#include "portmacro.h"

/* lwIP includes. */
#include "lwipopts.h"
#include "lwip/api.h"
#include "lwip/tcpip.h"
#include "lwip/memp.h"
#include "lwip/stats.h"
#include "lwip/opt.h"
#include "lwip/arch.h"
#include "lwip/sys.h"
#include "netif/loopif.h"
#include "lwip/sockets.h"

/* ethernet includes */
#include "ethernet.h"

#include "intc.h"
#include "usart.h"

xQueueHandle serialChars;

#if __GNUC__
    __attribute__((__noinline__))
#elif __ICCAVR32__
    #pragma optimize = no_inline
#endif /*
    */
static long
usart_nonNakedBehavior (void)
{

portBASE_TYPE xhigherTaskWoken = FALSE;
```

```

int receivedchar;

usart_read_char (&AVR32_USART0, &receivedchar);

char truncated = receivedchar;

xQueueSendFromISR (serialChars, &truncated, &xhigherTaskWoken);

return xhigherTaskWoken;
}

#ifdef __GNUC__
__attribute__((__naked__))
#endif /*
*/
static void
usart_int_handler (void)
{

portENTER_SWITCHING_ISR ();

usart_nonNakedBehavior ();

portEXIT_SWITCHING_ISR ();

}

void
initSerialPort (void)
{

    //Init the serial port 57600 baud 8 bit, 1 stop, no parity, no flow
    init_dbg_rs232 (66000000); //66 MHz from PBA
    print_dbg ("Debug_Port_started\n");

INTC_register_interrupt (( _int_handler ) & usart_int_handler ,
AVR32_USART0_IRQ, AVR32_INTC_INT0);

}

/*! \brief The Serial server.
*
* This uses Telnet to talk to the serial port
*/

```

```

portTASK_FUNCTION (SerialServer, pvParameters)
{
    int lSocket;

    struct sockaddr_in sLocalAddr;

    volatile avr32_usart_t *usart = &AVR32_USART0;

    serialChars = xQueueCreate (25, 1);

    lSocket = lwip_socket (AF_INET, SOCK_STREAM, 0);

    if (lSocket < 0)
        return;

    memset ((char *) &sLocalAddr, 0, sizeof (sLocalAddr));

    sLocalAddr.sin_family = AF_INET;

    sLocalAddr.sin_len = sizeof (sLocalAddr);

    sLocalAddr.sin_addr.s_addr = htonl (INADDR_ANY);

    sLocalAddr.sin_port = 23;

    if (lwip_bind
        (lSocket, (struct sockaddr *) &sLocalAddr, sizeof (sLocalAddr)) < 0)
    {
        lwip_close (lSocket);

        return;
    }

    if (lwip_listen (lSocket, 20) != 0)
    {
        lwip_close (lSocket);

        return;
    }

    while (1)
    {

```

```

int clientfd ;

struct sockaddr_in client_addr ;

int addrlen = sizeof (client_addr);

char buffer [10];

int nbytes;

int i;

char serialbuffer ;

clientfd =
    lwip_accept (lSocket, (struct sockaddr *) &client_addr,
                (socklen_t *) & addrlen);

if ( clientfd > 0)
    {

        //Enable the USART RX interrupt
        usart->ier = AVR32_USART_IER_RXRDY_MASK;

        do
        {

while (xQueueReceive (serialChars, &serialbuffer, 0) == pdTRUE)
            {

lwip_send ( clientfd , &serialbuffer , 1, 0);

}

nbytes =
            lwip_recv ( clientfd , buffer , sizeof (buffer), MSG_DONTWAIT);

if (nbytes == -2)
            {

                //delay a few ticks go give another routine a chance to run
                vTaskDelay (10);

            }

if (nbytes > 0)
            {

for (i = 0; i < nbytes; i++)
            {

```

```

print_dbg_char (buffer [i]);
}
}
}
    while (nbytes > 0 || nbytes == -2);

    //Turn off the RX interrupt
    usart->idr = AVR32_USART_IDR_RXRDY_MASK;

lwip_close (clientfd);
}
}

lwip_close (lSocket);
}

#endif /*
*/

```

---

## F.1.16 wdtreset.h

---

```

#ifndef WDTRESET_H_
#define WDTRESET_H_

extern xQueueHandle watchdogMbox;

#define DSTREAMOK 0x00
#define TCPIPOK 0x01
#define CMDOK 0x02

void vStartWDTTask (void);
void wdtResetTimeout (void *arg);

#endif /*WDTRESET_H_ */

```

---

## F.1.17 wdtreset.c

---

```

/*! \file wdtreset.c */

/* Environment include files. */
#include <stdlib.h>

#include <avr32/io.h>

```

```

#include "compiler.h"
#include "preprocessor.h"

#include "FreeRTOS.h"
#include "conf_lwip_threads.h"
#include "task.h"

#include "wdt.h"
#include "queue.h"

/* lwIP includes. */
#include "lwipopts.h"
#include "lwip/api.h"
#include "lwip/tcpip.h"
#include "lwip/memp.h"
#include "lwip/stats.h"
#include "lwip/opt.h"
#include "lwip/arch.h"
#include "lwip/sys.h"
#include "netif/loopif.h"
#include "lwip/sockets.h"

#include "wdtreset.h"

/*! Mailbox for the watchdog */
xQueueHandle watchdogMbox;

static portTASK_FUNCTION_PROTO (vWDTTask, pvParameters);

void
vStartWDTTask (void)
{
    //Set WDT to 5,000,000 microseconds, or 5 seconds
    //We will hit it much more often than this.

    watchdogMbox = xQueueCreate (5, 1);

    wdt_enable (5000000);

    xTaskCreate (vWDTTask, (const signed portCHAR * const) "WDT",
                ethWDT_TASK_STACK_SIZE, NULL, ethWDT_TASK_PRIORITY,
                (xTaskHandle *) NULL);
}

/*! \brief Notify WDT that lwIP is OK
 *
 * Fired from within lwIP to ensure it is functional
 */
void
wdtResetTimeout (void *arg)
{

```



```

char message = TCPIPOK;

//Tell WDT that the TCPIP task is OK
xQueueSend (watchdogMbox, &message, 10);

//Reset the timer
sys_timeout (500, wdtResetTimeout, NULL);
}

/*! \brief the Watchdog Timer reset task
*
* Waits for notification from both Datastream and lwIP
* before resetting the watchdog timer. If it gets no notification,
* the timer expires and resets the chip.
*/
static
portTASK_FUNCTION (vWDTTask, pvParameters)
{
    char tcpipok = 0;
    char dstreamok = 1;
    char cmdok = 1;

    char messagebuffer;

    while (1)
    {

        //Pend forever on messages
        xQueueReceive (watchdogMbox, &messagebuffer, portMAX_DELAY);

        switch (messagebuffer)
        {
            //case DSTREAMOK:
            // dstreamok = 1;
            // break;
            case TCPIPOK:
                tcpipok = 1;
                break;
            //case CMDOK:
            // cmdok = 1;
            // break;
            default:
                break;
        }

        if (tcpipok && dstreamok && cmdok)
        {
            //Hit the WDT
            wdt_clear ();
            //cmdok = 0;
            //dstreamok = 0;
            tcpipok = 0;
        }
    }
}

```

}

---

## F.1.18 main.c

---

```
/*This file has been prepared for Doxygen automatic documentation generation.*/
/*! \file *****
*
* \brief FreeRTOS and lwIP example for AVR32 UC3.
*
* - Compiler:      GNU GCC for AVR32
* - Supported devices: All AVR32 devices can be used.
* - AppNote:
*
* \author          Atmel Corporation: http://www.atmel.com \n
*                  Support and FAQ: http://support.atmel.no/
*
*****/

/* Copyright (c) 2007, Atmel Corporation All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* 1. Redistributions of source code must retain the above copyright notice,
* this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution .
*
* 3. The name of ATMEL may not be used to endorse or promote products derived
* from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR
* IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
* OF
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY
* AND
* SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY
* DIRECT,
* INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
* SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
* CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
* USE OF
* THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
```

```

/*
 * This program was written by Zachary Clifford from modified Atmel software
 * framework files. It implements the firmware necessary for the LEES data
 * acquisition board to function properly
 */

/* Environment include files. */
#include <stdlib.h>
#include <string.h>
#include "pm.h"
#include "flashc.h"
#include "sdramc.h"

#include <avr32/io.h>
#include "compiler.h"
#include "preprocessor.h"

/* Scheduler include files . */
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"

/* Demo file headers. */
#include "ethernet.h"
#include "netif/etharp.h"

/* Custom headers */
#include "InitBoard.h"
#include "externalmem.h"
#include "wdtreset.h"
#include "samplemanager.h"
#include "gpio.h"
#include "rtc.h"
#include "wdt.h"
#include "pwm.h"
#include "spi.h"

#include "DataStream.h"
#include "serialport.h"

#include "print_funcs.h"

#if _GNUC_
# include "nlao_cpu.h"
#endif

/* Priority definitions for most of the tasks in the demo application. */
#define mainETH_TASK_PRIORITY ( tskIDLE_PRIORITY + 1 )

#include "debug.h"

//!

```

```

//! \fn main
//! \brief start the software here
//! 1) Initialize the microcontroller and the shared hardware resources
//! of the board.
//! 2) Launch the IP modules.
//! 3) Start FreeRTOS.
//! \return 42, which should never occur.
//! \note
//!

int
main (void)
{
    wdt_disable ();
    Disable_global_interrupt ();

    //Enable the RESET to the ADC
    gpio_enable_gpio_pin (AVR32_PIN_PB29);
    //Pull RESET HIGH for ADC
    gpio_set_gpio_pin (AVR32_PIN_PB29);

    pwm_stop_channels (0x7D); // Stop all channels (this function takes a bitmask).

    spi_disable (&AVR32_SPI0);
    volatile avr32_pm_t *pm = &AVR32_PM;

    rtc_init (&AVR32_RTC, RTC_OSC_RC, RTC_PSEL_RC_1.76HZ); //RTC ticks in about 1Hz
    rtc_enable (&AVR32_RTC);

#if (INCLUDE_POST == 1)
    //First check the LEDs before trying to init the external clock
    POST_LEDs ();
#endif

    /* 1) Initialize the microcontroller and the shared hardware resources of the board. */
    /* Switch to external oscillator 0 */
    pm_switch_to_osc0 (pm, FOSC0, OSC0_STARTUP);

    //Setup PLL0 on OSC0, mul+1=11, divisor by 1, lockcount=16, ie. 12Mhzx11/1 = 132MHz
    output for VCO.
    pm_pll_setup (pm, /* volatile avr32_pm_t* pm */
        0, /* unsigned int pll */
        10, /* unsigned int mul */
        1, /* unsigned int div */
        0, /* unsigned int osc */
        16); /* unsigned int lockcount */

    //After this line, the PLL will be divided by 2
    pm_pll_set_option (pm, 0, // pll0
        1, // Choose the range 160–240MHz with 0 and 80–180 with 1
        (VCO is at 132MHz).
        1, // div2

```

```

        0);          // wbdisable

//Now PLL0 is configured to output 66 MHz

#if __GNUC__
    set_cpu_hz (66000000); //This is weird, but copied from pm_conf_clocks.c //Look out
        because of bugs in the NEWLIB addons.
#endif

/* Enable PLL0 */
pm_pll.enable (pm, 0);

/* Wait for PLL0 locked */
pm_wait_for_pll0_locked (pm);

/* switch to clock */
pm_cksel (pm, 0,          /* PBA clock divisor enable */
          0,             /* PBA select */
          0,             /* PBB clock divisor enable */
          0,             /* PBB Select */
          0,             /* HSB divisor enable (CPU clock = HSB clock) */
          0,             /* HSB select (CPU clock = HSB clock) */
          );

/* Now PBA is at 66 MHz, HSB is 66, and PBB is 66 */
flashc_set_wait_state (1); //Need to set wait state because operating
//above 33 MHz

//Switch the main power manager to using the PLL0 clock instead of osc0.
pm_switch_to_clock (pm, AVR32_PM_MCCTRL_MCSEL_PLL0);

#if (INCLUDE_POST == 1)
    //Now try again after engaging the external oscillator
    POST_LEDs ();
#endif

// Initialize the SDRAM Controller and the external SDRAM chip.
sdramc_init (66000000); //66 MHz to SDRAM Controller because it uses system clock
//Now the SDRAM lives from 0xD0000000 to 0xD2000000
//It will be used to hold samples from the ADC
//It has 256 Mbits. The linker does not know about it, so it will be
//referenced through pointers. See externalmem.h for info about
//how this memory is used.

#if (INCLUDE_POST == 1)
    //This takes forever, but is useful for initial testing
    if (POST_SDRAM () == 0)
    {
        POST_LEDs ();
    }
#endif

//Setup the ADC's SPI bus. It is the only device there.
SetupADCSPI ();

```

```

#ifdef ENABLE_SERIAL
    //This initializes the serial port for the telnet feature
    initSerialPort ();
#endif

    Enable_global_interrupt ();

    vStartSampleManagerTask ();

    /* Start the Ethernet tasks launcher. */
    vStartEthernetTaskLauncher (configMAX_PRIORITIES);

    //Start the packet processing task
    StartCopyTask ();

    //Start the Watchdog Timer and its resetting task
    vStartWDTTask ();

    /* 3) Start FreeRTOS. */
    vTaskStartScheduler ();

    /* Will only reach here if there was insufficient memory to create the idle task. */

    return 0;
}

/*-----*/

```

---

### F.1.19 version.h

```

/* This file was automatically generated. */
char * versionstr = "NERD:_Version_1.1_(2009-04-25)\n";

```

---

## F.2 Ethstream Source

This section contains the source for the ethstream utility for acquiring data from the NerdJack device. Ethstream is a modified version of LJStream written by Jim Paris. Ethstream added the ability to communicate with a NerdJack to LJStream. The parts relevant to NerdJack are included, but the original LabJack protocol and network drivers are omitted for brevity and clarity.

### F.2.1 ethstream.h

---

```

#ifdef ETHSTREAM_H
#define ETHSTREAM_H

#define CONVERT_DEC 0

```

```
#define CONVERT_VOLTS 1
#define CONVERT_HEX 2

#endif
```

---

## F.2.2 ethstream.c

---

```
/*
 * Labjack Tools
 * Copyright (c) 2003–2007 Jim Paris <jim@jtan.com>
 *
 * This is free software; you can redistribute it and/or modify it and
 * it is provided under the terms of version 2 of the GNU General Public
 * License as published by the Free Software Foundation; see COPYING.
 */

/* ljstream: Stream data from the first N (1–14) analog inputs.
   Resolution is set to 12-bit and all channels are in bipolar (–5 to
   +5V) mode.
 */

#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/time.h>
#include <time.h>
#include <sys/stat.h>
#include <signal.h>
#include <unistd.h>
#include "debug.h"
#include "ue9.h"
#include "ue9error.h"
#include "nerdjack.h"
#include "opt.h"
#include "version.h"
#include "compat.h"
#include "ethstream.h"

#include "example.h"

#define DEFAULT_HOST "192.168.1.209"
#define UE9_COMMAND_PORT 52360
#define UE9_DATA_PORT 52361

struct callbackInfo
{
    struct ue9Calibration calib;
    int convert;
    int maxlines;
};
```

```

};

struct options opt[] = {
    {'a', "address", "string", "host/address_of_device_(192.168.1.209)"},
    {'n', "numchannels", "n", "sample_the_first_N_ADC_channels_(2)"},
    {'N', "nerdjack", NULL, "Force_NerdJack_device"},
    {'L', "labjack", NULL, "Force_LabJack_device"},
    {'d', "detect", NULL, "Detect_NerdJack_IP_address"},
    {'R', "range", "a,b",
     "Set_range_on_NerdJack_for_channels_0-5,6-11_to_either_5_or_10_(10,10)"},
    {'C', "channels", "a,b,c", "sample_channels_a,b,and_c"},
    {'r', "rate", "hz", "sample_each_channel_at_this_rate_(8000.0)"},
    {'o', "oneshot", NULL, "don't_retry_in_case_of_errors"},
    {'f', "forceretry", NULL, "retry_no_matter_what_happens"},
    {'c', "convert", NULL, "convert_output_to_volts"},
    {'H', "converthex", NULL, "convert_output_to_hex"},
    {'m', "showmem", NULL, "output_memory_stats_with_data_(NJ_only)"},
    {'l', "lines", "num", "if_set,_output_this_many_lines_and_quit"},
    {'h', "help", NULL, "this_help"},
    {'v', "verbose", NULL, "be_verbose"},
    {'V', "version", NULL, "show_version_number_and_exit"},
    {'i', "info", NULL, "get_info_from_device_(NJ_only)"},
    {'X', "examples", NULL, "show_ethstream_examples_and_exit"},
    {0, NULL, NULL, NULL}
};

int doStream (const char *address, uint8_t scanconfig, uint16_t scaninterval,
              int *channel_list, int channel_count, int convert,
              int maxlines);
int nerdDoStream (const char *address, int *channel_list, int channel_count,
                  int precision, unsigned long period, int convert, int lines,
                  int showmem);
int data_callback (int channels, uint16_t * data, void *context);

int columns_left = 0;
void
handle_sig (int sig)
{
    while (columns_left--)
        {
            printf ("_0");
        }
    fflush (stdout);
    exit (0);
}

int
main (int argc, char *argv[])
{
    int optind;
    char *optarg, *endp;
    char c;
    int tmp, i;
    FILE *help = stderr;

```



```

char *address = strdup (DEFAULT_HOST);
double desired_rate = 8000.0;
int lines = 0;
double actual_rate;
int oneshot = 0;
int forceretry = 0;
int convert = CONVERT_DEC;
int showmem = 0;
int inform = 0;
uint8_t scanconfig;
uint16_t scaninterval;
#if UE9_CHANNELS > NERDJACK_CHANNELS
int channel_list [UE9_CHANNELS];
#else
int channel_list [NERDJACK_CHANNELS];
#endif
int channel_count = 0;
int nerdjack = 0;
int labjack = 0;
int detect = 0;
int precision = 0;
int addressSpecified = 0;
int donerdjack = 0;
unsigned long period = NERDJACK_CLOCK_RATE / desired_rate;

/* Parse arguments */
opt_init (&optind);
while ((c = opt_parse (argc, argv, &optind, &optarg, opt)) != 0)
{
    switch (c)
    {
        case 'a':
            free (address);
            address = strdup (optarg);
            addressSpecified = 1;
            break;
        case 'n':
            channel_count = 0;
            tmp = strtol (optarg, &endp, 0);
            if (*endp || tmp < 1 || tmp > UE9_CHANNELS)
            {
                info ("bad_number_of_channels:_%s\n", optarg);
                goto printhelp;
            }
            for (i = 0; i < tmp; i++)
                channel_list [channel_count++] = i;
            break;
        case 'C':
            channel_count = 0;
            do
            {
                tmp = strtol (optarg, &endp, 0);
                if (*endp != '\0' && *endp != ',')
                    {

```

```

        info ("bad_channel_number:_%s\n", optarg);
        goto printhelp;
    }
    //We do not want to overflow channel_list, so we need the check here
    //The rest of the sanity checking can come later after we know
    //whether this is a
    //LabJack or a NerdJack
#if UE9_CHANNELS > NERDJACK_CHANNELS
    if (channel_count >= UE9_CHANNELS)
    {
#else
    if (channel_count >= NERDJACK_CHANNELS)
    {
#endif
        info ("error:_too_many_channels_specified\n");
        goto printhelp;
    }
    channel_list [channel_count++] = tmp;
    optarg = endp + 1;
}
while (*endp);
break;
case 'r':
    desired_rate = strtod (optarg, &endp);
    if (*endp || desired_rate <= 0)
    {
        info ("bad_rate:_%s\n", optarg);
        goto printhelp;
    }
    break;
case 'l':
    lines = strtol (optarg, &endp, 0);
    if (*endp || lines <= 0)
    {
        info ("bad_number_of_lines:_%s\n", optarg);
        goto printhelp;
    }
    break;
case 'R':
    tmp = strtol (optarg, &endp, 0);
    if (*endp != ',')
    {
        info ("bad_range_number:_%s\n", optarg);
        goto printhelp;
    }
}
if(tmp != 5 && tmp != 10) {
    info("valid_choices_for_range_are_5_or_10\n");
    goto printhelp;
}
if(tmp == 5) precision = precision + 1;

    optarg = endp + 1;
    if (*endp == '\0') {

```

```

        info("Range_needs_two_numbers,_one_for_channels_0-5_and_another_for_
            6-11\n");
        goto printhelp;
    }
    tmp = strtol (optarg, &endp, 0);
    if (*endp != '\0') {
        info("Range_needs_only_two_numbers,_one_for_channels_0-5_and_another_for_
            6-11\n");
        goto printhelp;
    }
    if(tmp != 5 && tmp != 10) {
        info(" valid_choices_for_range_are_5_or_10\n");
        goto printhelp;
    }
    if(tmp == 5) precision = precision + 2;
    break;
case 'N':
    nerdjack++;
    break;
case 'L':
    labjack++;
    break;
case 'd':
    detect++;
    break;
case 'o':
    oneshot++;
    break;
case 'f':
    forceretry++;
    break;
case 'c':
    if (convert != 0)
    {
        info ("specify_only_one_conversion_type\n");
        goto printhelp;
    }
    convert = CONVERT_VOLTS;
    break;
case 'H':
    if (convert != 0)
    {
        info ("specify_only_one_conversion_type\n");
        goto printhelp;
    }
    convert = CONVERT_HEX;
    break;
case 'm':
    showmem++;
case 'v':
    verb_count++;
    break;
case 'X':
    printf("%s",examplestring);

```

```

return 0;
break;
case 'V':
    printf ("etherstream_ VERSION "\n");
    printf ("Written_by_Jim_Paris_<jim@jtan.com>\n");
    printf ("and_Zachary_Clifford_<zacharyc@mit.edu>\n");
    printf ("This_program_comes_with_no_warranty_and_is_
            "provided_under_the_GPLv2.\n");
    return 0;
    break;
case 'i':
    inform++;
    break;
case 'h':
    help = stdout;
default:
    printhelp:
        fprintf (help, "Usage:_%s_[options]\n", *argv);
        opt_help (opt, help);
        fprintf (help, "Read_data_from_the_specified_Labjack_UE9"
                "via_Ethernet_...See_README_for_details.\n");
    return (help == stdout) ? 0 : 1;
}
}

if (detect && labjack) {
    info("The_LabJack_does_not_support_autodetection\n");
    goto printhelp;
}

if (detect && !nerdjack) {
    info("Only_the_NerdJack_supports_autodetection_-_assuming_-N_option\n");
    nerdjack = 1;
}

if (detect && addressSpecified) {
    info("Autodetection_and_specifying_address_are_mutually_exclusive\n");
    goto printhelp;
}

if (nerdjack && labjack) {
    info("Nerdjack_and_Labjack_options_are_mutually_exclusive\n");
    goto printhelp;
}

donerdjack = nerdjack;

//First if no options were supplied try the Nerdjack
//The second time through, donerdjack will be true and this will not fire
if (!nerdjack && !labjack) {
    info("No_device_specified ... Defaulting_to_Nerdjack\n");
    donerdjack = 1;
}

```

doneparse:

```
if(inform) {
    //We just want information from NerdJack
    if(!detect) {
        if(nerd_get_version(address) < 0) {
            info("Could_not_find_NerdJack_at_specified_address\n");
        } else {
            return 0;
        }
    }
    info ("Autodetecting_NerdJack_address\n");
    free (address);
    if (nerdjack_detect (address) < 0)
    {
        info ("Error_with_autodetection\n");
        goto printhelp;
    }
    else
    {
        info ("Found_NerdJack_at_address:_%s\n", address);
        if(nerd_get_version(address) < 0) {
            info("Error_getting_NerdJack_version\n");
            goto printhelp;
        }
        return 0;
    }
}
```

```
if (donerdjack)
{
    if (channel_count > NERDJACK_CHANNELS)
    {
        info ("Too_many_channels_for_NerdJack\n");
        goto printhelp;
    }
    for (i = 0; i < channel_count; i++)
    {
        if (channel_list [i] >= NERDJACK_CHANNELS)
        {
            info ("Channel_is_out_of_NerdJack_range:_%d\n",
                channel_list [i]);
            goto printhelp;
        }
    }
}
else
{
    if (channel_count > UE9_CHANNELS)
    {
        info ("Too_many_channels_for_LabJack\n");
        goto printhelp;
    }
}
```

```

    for (i = 0; i < channel_count; i++)
    {
        if (channel_list[i] >= UE9_CHANNELS)
        {
            info ("Channel is out of LabJack range: %d\n", channel_list[i]);
            goto printhelp;
        }
    }
}

if (optind < argc)
{
    info ("error: too many arguments (%s)\n\n", argv[optind]);
    goto printhelp;
}

if (forceretry && oneshot)
{
    info ("forceretry and oneshot options are mutually exclusive\n");
    goto printhelp;
}

/* Two channels if none specified */
if (channel_count == 0)
{
    channel_list[channel_count++] = 0;
    channel_list[channel_count++] = 1;
}

if (verb_count)
{
    info ("Scanning channels:");
    for (i = 0; i < channel_count; i++)
        info ("AIN%d", channel_list[i]);
    info ("\n");
}

/* Figure out actual rate. */
if (donerdjack)
{
    if (nerdjack_choose_scan (desired_rate, &actual_rate, &period) < 0)
    {
        info ("error: can't achieve requested scan rate (%lf.Hz)\n",
            desired_rate);
    }
}
else
{
    if (ue9_choose_scan (desired_rate, &actual_rate,
        &scanconfig, &scaninterval) < 0)
    {
        info ("error: can't achieve requested scan rate (%lf.Hz)\n",

```

```

        desired_rate);
    }
}

if ((desired_rate != actual_rate) || verb_count)
{
    info ("Actual_scanrate_is_%lf_Hz\n", actual_rate);
    info ("Period_is_%ld\n", period);
}

if (verb_count && lines)
{
    info ("Stopping_capture_after_%d_lines\n", lines);
}

signal (SIGINT, handle_sig);
signal (SIGTERM, handle_sig);

if (detect)
{
    info ("Autodetecting_NerdJack_address\n");
    free (address);
    if (nerdjack_detect (address) < 0)
    {
        info ("Error_with_autodetection\n");
        goto printhelp;
    }
    else
    {
        info ("Found_NerdJack_at_address:%s\n", address);
    }
}

for (;;)
{
    int ret;
    if (donerdjack)
    {
        ret =
            nerdDoStream (address, channel_list, channel_count, precision,
                          period, convert, lines, showmem);
        verb ("nerdDoStream_returned_%d\n", ret);
    }
    else
    {
        ret = doStream (address, scanconfig, scaninterval,
                       channel_list, channel_count, convert, lines);
        verb ("doStream_returned_%d\n", ret);
    }
    if (oneshot)

```

```

    break;

    if (ret == 0)
        break;

//Neither options specified at command line and first time through.
//Try LabJack
    if (ret == -ENOTCONN && donerdjack && !labjack && !nerdjack)
    {
        info ("Could_not_connect_NerdJack...Trying_LabJack\n");
        donerdjack = 0;
        goto doneparse;
    }

//Neither option supplied, no address, and second time through.
//Try autodetection
    if (ret == -ENOTCONN && !donerdjack && !labjack && !nerdjack &&
        !addressSpecified) {
        info ("Could_not_connect_LabJack...Trying_to_autodetect_Nerdjack\n");
        detect = 1;
        donerdjack = 1;
        goto doneparse;
    }

    if (ret == -ENOTCONN && nerdjack && !detect && !addressSpecified) {
        info ("Could_not_reach_NerdJack...Trying_to_autodetect\n");
        detect = 1;
        goto doneparse;
    }

    if (ret == -ENOTCONN && !forceretry)
    {
        info ("Initial_connection_failed,_giving_up\n");
        break;
    }

    if (ret == -EAGAIN || ret == -ENOTCONN)
    {
        /* Some transient error. Wait a tiny bit, then retry */
        info ("Retrying_in_5_secs.\n");
        sleep (5);
    }
    else
    {
        info ("Retrying_now.\n");
    }
}

debug ("Done_loop\n");

return 0;
}

int

```



```

nerdDoStream (const char *address, int *channel_list, int channel_count,
              int precision, unsigned long period, int convert, int lines,
              int showmem)
{
    int retval = -EAGAIN;
    int fd_data;
    static int first_call = 1;
    static int started = 0;
    static int wasreset = 0;
    getPacket command;
    static unsigned short currentcount = 0;
tryagain:

    //If this is the first time, set up acquisition
    //Otherwise try to resume the previous one
    if (started == 0)
    {
        if (nerd_generate_command
            (&command, channel_list, channel_count, precision, period) < 0)
        {
            info ("Failed_to_create_configuration_command\n");
            goto out;
        }

        if (nerd_send_command (address, "STOP", 4) < 0)
        {
            if ( first_call ) {
                retval = -ENOTCONN;
                if(verb_count) info("Failed_to_send_STOP_command\n");
            } else {
                info ("Failed_to_send_STOP_command\n");
            }
            goto out;
        }

        if (nerd_send_command (address, &command, sizeof (command)) < 0)
        {
            info ("Failed_to_send_GET_command\n");
            goto out;
        }
    }
else
    {
        //If we had a transmission in progress, send a command to resume from there
        char cmdbuf[10];
        sprintf (cmdbuf, "SETC%05hd", currentcount);
        retval = nerd_send_command (address, cmdbuf, strlen (cmdbuf));
        if (retval == -4)
        {
            info ("NerdJack_was_reset\n");
            //Assume we have not started yet, reset on this side.
            //If this routine is retried, start over
            printf ("#_NerdJack_was_reset_here\n");
        }
    }
}

```

```

        currentcount = 0;
        started = 0;
        wasreset = 1;
        goto tryagain;
    }
    else if (retval < 0)
    {
        info ("Failed_to_send_SETC_command\n");
        goto out;
    }
}

//The transmission has begun
started = 1;

/* Open connection */
fd_data = nerd_open (address, NERDJACK_DATA_PORT);
if (fd_data < 0)
{
    info ("Connect_failed:_%s:%d\n", address, NERDJACK_DATA_PORT);
    goto out;
}

retval = nerd_data_stream
    (fd_data, channel_count, channel_list, precision, convert, lines,
    showmem, &currentcount, period, wasreset);
wasreset = 0;
if (retval == -3)
{
    retval = 0;
}
if (retval < 0)
{
    info ("Failed_to_open_data_stream\n");
    goto out1;
}

info ("Stream_finished\n");
retval = 0;

out1:
    nerd_close_conn (fd_data);
out:
    //We've tried communicating, so this is not the first call anymore
    first_call = 0;
    return retval;
}

int
doStream (const char *address, uint8_t scanconfig, uint16_t scaninterval,
    int *channel_list, int channel_count, int convert, int lines)
{
    int retval = -EAGAIN;
    int fd_cmd, fd_data;

```

```

int ret;
static int first_call = 1;
struct callbackInfo ci = {
    .convert = convert,
    .maxlines = lines,
};

/* Open command connection. If this fails, and this is the
   first attempt, return a different error code so we give up. */
fd_cmd = ue9_open (address, UE9_COMMAND_PORT);
if (fd_cmd < 0)
{
    info ("Connect_failed:_%s:%d\n", address, UE9_COMMAND_PORT);
    if ( first_call )
        retval = -ENOTCONN;
    goto out;
}
first_call = 0;

/* Make sure nothing is left over from a previous stream */
if (ue9_stream_stop (fd_cmd) == 0)
    verb ("Stopped_previous_stream.\n");
ue9_buffer_flush (fd_cmd);

/* Open data connection */
fd_data = ue9_open (address, UE9_DATA_PORT);
if (fd_data < 0)
{
    info ("Connect_failed:_%s:%d\n", address, UE9_DATA_PORT);
    goto out1;
}

/* Get calibration */
if (ue9_get_calibration (fd_cmd, &ci.calib) < 0)
{
    info ("Failed_to_get_device_calibration\n");
    goto out2;
}

/* Set stream configuration */
if (ue9_streamconfig_simple (fd_cmd, channel_list, channel_count,
                             scanconfig, scaninterval,
                             UE9_BIPOLAR_GAIN1) < 0)
{
    info ("Failed_to_set_stream_configuration\n");
    goto out2;
}

/* Start stream */
if (ue9_stream_start (fd_cmd) < 0)
{
    info ("Failed_to_start_stream\n");
    goto out2;
}

```

```

/* Stream data */
ret = ue9_stream_data (fd_data, channel_count, data_callback, (void *) &ci);
if (ret < 0)
{
    info ("Data_stream_failed_with_error_%d\n", ret);
    goto out3;
}

info ("Stream_finished\n");
retval = 0;

out3:
    /* Stop stream and clean up */
    ue9_stream_stop (fd_cmd);
    ue9_buffer_flush (fd_cmd);
out2:
    ue9_close (fd_data);
out1:
    ue9_close (fd_cmd);
out:
    return retval;
}

int
data_callback (int channels, uint16_t * data, void *context)
{
    int i;
    struct callbackInfo *ci = (struct callbackInfo *) context;
    static int lines = 0;

    columns_left = channels;
    for (i = 0; i < channels; i++)
    {
        switch (ci->convert)
        {
            case CONVERT_VOLTS:
                if (printf
                    ("%lf",
                     ue9_binary_to_analog (&ci->calib, UE9_BIPOLAR_GAIN1, 12,
                                             data[i])) < 0)
                    goto bad;
                break;
            case CONVERT_HEX:
                if (printf ("%04X", data[i]) < 0)
                    goto bad;
                break;
            default:
                case CONVERT_DEC:
                    if (printf ("%d", data[i]) < 0)
                        goto bad;
                    break;
        }
        columns_left--;
    }
}

```

```

    if (i < (channels - 1))
    {
        if (ci->convert != CONVERT_HEX && putchar ('.') < 0)
            goto bad;
    }
    else
    {
        if (putchar ('\n') < 0)
            goto bad;
        lines++;
        if (ci->maxlines && lines >= ci->maxlines)
            return -1;
    }
}

return 0;

bad:
    info ("Output_error_(disk_full?)\n");
    return -3;
}

```

---

### F.2.3 nerdjack.h

---

```

/*
 * Labjack Tools
 * Copyright (c) 2003-2007 Jim Paris <jim@jtan.com>
 *
 * This is free software; you can redistribute it and/or modify it and
 * it is provided under the terms of version 2 of the GNU General Public
 * License as published by the Free Software Foundation; see COPYING.
 */

#ifndef NERDJACK_H
#define NERDJACK_H

#include <stdint.h>
#include <stdlib.h>

#include "netutil.h"

#define NERDJACK_CHANNELS 12
#define NERDJACK_CLOCK_RATE 66000000
#define NERDJACK_DATA_PORT 49155
#define NERDJACK_UDP_RECEIVE_PORT 49156
#define NERDJACK_COMMAND_PORT 49157

#define NERDJACK_PACKET_SIZE 1460
#define NERDJACK_NUM_SAMPLES 726

/* Packet structure used in message to start sampling on NerdJack */
typedef struct __attribute__((__packed__))

```

```

{
    char word[4];
    unsigned long period;
    unsigned short channelbit;
    unsigned char precision;
    unsigned char prescaler;
} getPacket;

/* Open/close TCP/IP connection to the NerdJack */
int nerd_open (const char *address, int port);
int nerd_close_conn (int data_fd);

/* Generate the command word for the NerdJack */
int nerd_generate_command (getPacket * command, int *channel_list,
                          int channel_count, int precision,
                          unsigned long period);

/* Send given command to NerdJack */
int nerd_send_command (const char *address, void *command, int length);

/* Get the version string from NerdJack */
int nerd_get_version (const char *address);

/* Stream data out of the NerdJack */
int nerd_data_stream (int data_fd, int numChannels, int *channel_list,
                    int precision, int convert, int lines, int showmem,
                    unsigned short *currentcount, unsigned int period,
                    int wasreset);

/* Detect the IP Address of the NerdJack and return in ipAddress */
int nerdjack_detect (char *ipAddress);

/* Choose the best ScanConfig and ScanInterval parameters for the
   desired scanrate. Returns -1 if no valid config found */
int nerdjack_choose_scan (double desired_rate, double *actual_rate,
                        unsigned long *period);

#endif

```

---

## F.2.4 nerdjack.c

---

```

/*
 * Labjack Tools
 * Copyright (c) 2003–2007 Jim Paris <jim@jtan.com>
 *
 * This is free software; you can redistribute it and/or modify it and
 * it is provided under the terms of version 2 of the GNU General Public
 * License as published by the Free Software Foundation; see COPYING.
 */

#include <errno.h>
#include <stdint.h>

```

```

#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

#include <math.h>

#include "netutil.h"
#include "compat.h"

#include "debug.h"
#include "nerdjack.h"
#include "util.h"
#include "netutil.h"
#include "ethstream.h"

#define NERDJACK_TIMEOUT 5 /* Timeout for connect/send/recv, in seconds */

#define NERD_HEADER_SIZE 8
#define MAX_SOCKETS 32

typedef struct __attribute__((__packed__))
{
    unsigned char headerone;
    unsigned char headertwo;
    unsigned short packetNumber;
    unsigned short adused;
    unsigned short packetsready;
    signed short data[NERDJACK_NUM_SAMPLES];
} dataPacket;

struct discovered_socket {
    int sock;
    uint32_t local_ip;
    uint32_t subnet_mask;
};

struct discover_t {
    struct discovered_socket socks[MAX_SOCKETS];
    unsigned int sock_count;
};

/* Choose the best ScanConfig and ScanInterval parameters for the
   desired scanrate. Returns -1 if no valid config found */
int
nerdjack_choose_scan (double desired_rate, double *actual_rate,
                     unsigned long *period)
{
    //The fffe is because of a silicon bug. The last bit is unusable in all
    //devices so far. It is worked around on the chip, but giving it exactly
    //0xffff would cause the workaround code to roll over.
    *period = floor ((double) NERDJACK_CLOCK_RATE / desired_rate);
    if (*period > 0xffff)
    {

```

```

    info ("Cannot_sample_that_slowly\n");
    *actual_rate = (double) NERDJACK_CLOCK_RATE / (double) 0x0ffffe;
    *period = 0x0ffffe;
    return -1;
}
//Period holds the period register for the NerdJack, so it needs to be right
*actual_rate = (double) NERDJACK_CLOCK_RATE / (double) *period;
if (*actual_rate != desired_rate)
{
    return -1;
}
return 0;
}

/**
 * Create a discovered socket and add it to the socket list structure.
 * All sockets in the structure should be created, bound, and ready for broadcasting
 */
static int discovered_sock_create(struct discover_t *ds, uint32_t local_ip, uint32_t
    subnet_mask)
{
    if (ds->sock_count >= MAX_SOCKETS) {
        return 0;
    }

    /* Create socket. */
    int sock = (int)socket(AF_INET, SOCK_DGRAM, 0);
    if (sock == -1) {
        return 0;
    }

    /* Allow broadcast. */
    int sock_opt = 1;
    setsockopt(sock, SOL_SOCKET, SO_BROADCAST, (char *)&sock_opt,
        sizeof(sock_opt));

    /* Set nonblocking */
    if (soblock (sock, 0) < 0)
    {
        verb ("can't_set_nonblocking\n");
        return 0;
    }

    /* Bind socket. */
    struct sockaddr_in sock_addr;
    memset(&sock_addr, 0, sizeof(sock_addr));
    sock_addr.sin_family = AF_INET;
    sock_addr.sin_addr.s_addr = htonl(local_ip);
    sock_addr.sin_port = htons(0);
    if (bind(sock, (struct sockaddr *)&sock_addr, sizeof(sock_addr)) != 0) {
        close (sock);
        return 0;
    }
}

```



```

    /* Write sock entry. */
    struct discovered_socket *dss = &ds->socks[ds->sock_count++];
    dss->sock = sock;
    dss->local_ip = local_ip;
    dss->subnet_mask = subnet_mask;

    return 1;
}
/**
 * Enumerate all interfaces we can find and open sockets on each
 */
#ifdef USE_IPHLPAPI
static void enumerate_interfaces(struct discover_t *ds)
{
    PIP_ADAPTER_INFO pAdapterInfo = (IP_ADAPTER_INFO
        *)malloc(sizeof(IP_ADAPTER_INFO));
    ULONG ulOutBufLen = sizeof(IP_ADAPTER_INFO);

    DWORD Ret = GetAdaptersInfo(pAdapterInfo, &ulOutBufLen);
    if (Ret != NO_ERROR) {
        free(pAdapterInfo);
        if (Ret != ERROR_BUFFER_OVERFLOW) {
            return;
        }
        pAdapterInfo = (IP_ADAPTER_INFO *)malloc(ulOutBufLen);
        Ret = GetAdaptersInfo(pAdapterInfo, &ulOutBufLen);
        if (Ret != NO_ERROR) {
            free(pAdapterInfo);
            return;
        }
    }

    PIP_ADAPTER_INFO pAdapter = pAdapterInfo;
    while (pAdapter) {
        IP_ADDR_STRING *pIPAddr = &pAdapter->IpAddressList;
        while (pIPAddr) {
            uint32_t local_ip = ntohl(inet_addr(pIPAddr->IpAddress.String));
            uint32_t mask = ntohl(inet_addr(pIPAddr->IpMask.String));

            if (local_ip == 0) {
                pIPAddr = pIPAddr->Next;
                continue;
            }

            discovered_sock_create(ds, local_ip, mask);
            pIPAddr = pIPAddr->Next;
        }

        pAdapter = pAdapter->Next;
    }

    free(pAdapterInfo);
}

```

```

#else
static void enumerate_interfaces(struct discover_t *ds) {
    int fd = socket(AF_INET, SOCK_DGRAM, 0);
    if (fd == -1) {
        return;
    }

    struct ifconf ifc ;
    uint8_t buf[8192];
    ifc.ifc_len = sizeof(buf);
    ifc.ifc_buf = (char *)buf;

    memset(buf, 0, sizeof(buf));

    if (ioctl(fd, SIOCGIFCONF, &ifc) != 0) {
        close(fd);
        return;
    }

    uint8_t *ptr = (uint8_t *)ifc.ifc_req ;
    uint8_t *end = (uint8_t *)&ifc.ifc_buf[ifc.ifc_len ];

    while (ptr <= end) {
        struct ifreq *ifr = (struct ifreq *)ptr;
        ptr += _SIZEOF_ADDR_IFREQ(*ifr);

        if (ioctl(fd, SIOCGIFADDR, ifr) != 0) {
            continue;
        }
        struct sockaddr_in *addr_in = (struct sockaddr_in *)&(ifr->ifr_addr);
        uint32_t local_ip = ntohl(addr_in->sin_addr.s_addr);
        if (local_ip == 0) {
            continue;
        }

        if (ioctl(fd, SIOCGIFNETMASK, ifr) != 0) {
            continue;
        }

        struct sockaddr_in *mask_in = (struct sockaddr_in *)&(ifr->ifr_addr);
        uint32_t mask = ntohl(mask_in->sin_addr.s_addr);

        discovered_sock_create(ds, local_ip, mask);
    }
}
#endif
/**
 * Close all sockets previously enumerated and free the struct
 */
static void destroy_socks(struct discover_t *ds)
{
    unsigned int i;
    for (i = 0; i < ds->sock_count; i++) {
        struct discovered_socket *dss = &ds->socks[i];

```

```

        close(dss->sock);
    }

    free(ds);
}

/* Perform autodetection. Returns 0 on success, -1 on error
 * Sets ipAddress to the detected address
 */
int
nerdjack_detect (char *ipAddress)
{
    int32_t receivesock;
    struct sockaddr_in sa, receiveaddr, sFromAddr;
    int buffer_length;
    char buffer[200];
    char incomingData[10];
    unsigned int lFromLen;

    sprintf (buffer, "TEST");
    buffer_length = strlen (buffer) + 1;

    net_init ();

    receivesock = socket (PF_INET, SOCK_DGRAM, IPPROTO_UDP);

    /* Set nonblocking */
    if (soblock (receivesock, 0) < 0)
    {
        verb ("can't set nonblocking\n");
        return -1;
    }

    if (-1 == receivesock) /* if socket failed to initialize , exit */
    {
        verb ("Error Creating Socket\n");
        return -1;
    }

    //Setup family for both sockets
    sa.sin_family = PF_INET;
    receiveaddr.sin_family = PF_INET;

    //Setup ports to send on DATA and receive on RECEIVE
    receiveaddr.sin_port = htons (NERDJACK_UDP_RECEIVE_PORT);
    sa.sin_port = htons (NERDJACK_DATA_PORT);

    //Receive from any IP address
    receiveaddr.sin_addr.s_addr = INADDR_ANY;

    bind (receivesock, (struct sockaddr *) &receiveaddr,
        sizeof (struct sockaddr_in));
}

```

```

struct discover_t *ds = (struct discover_t *)calloc(1, sizeof(struct discover_t));
    if (!ds) {
        return -1;
    }

    /* Create a routable broadcast socket. */
    if (!discovered_sock_create(ds, 0, 0)) {
        free(ds);
        return -1;
    }

    /* Detect & create local sockets. */
    enumerate_interfaces(ds);

    /*
     * Send subnet broadcast using each local ip socket.
     * This will work with multiple separate 169.254.x.x interfaces.
     */
    unsigned int i;
    for (i = 0; i < ds->sock_count; i++) {
        struct discovered_socket *dss = &ds->socks[i];
        uint32_t target_ip = dss->local_ip | ~dss->subnet_mask;
        sa.sin_addr.s_addr = htonl(target_ip);
        sendto (dss->sock, buffer, buffer_length, 0, (struct sockaddr *) &sa,
                sizeof (struct sockaddr_in));
    }

    destroy_socks(ds);

lFromLen = sizeof (sFromAddr);

if (0 >
    recvfrom_timeout (receivesock, incomingData, sizeof (incomingData), 0,
        (struct sockaddr *) &sFromAddr, &lFromLen,
        &(struct timeval)
        {
            .tv_sec = NERDJACK_TIMEOUT}))
    {
        close(receivesock);
        return -1;
    }

ipAddress = malloc (INET_ADDRSTRLEN);

    /*It isn't ipv6 friendly, but inet_ntop isn't on Windows...
    strcpy (ipAddress, inet_ntoa (sFromAddr.sin_addr));

    close (receivesock);
    return 0;
}

/*
 * Get the NerdJack version string and print it
 */

```

```

int
nerd_get_version (const char *address)
{
    int ret, fd_command;
    char buf[200];
    fd_command = nerd_open (address, NERDJACK_COMMAND_PORT);
    if (fd_command < 0)
        {
            info ("Connect_failed:_%s:%d\n", address, NERDJACK_COMMAND_PORT);
            return -2;
        }

    /* Send request */
    ret = send_all_timeout (fd_command, "VERS", 4, 0, &(struct timeval)
        {
            .tv_sec = NERDJACK_TIMEOUT});

    if (ret < 0)
        {
            verb ("short_send_%d\n", (int) ret);
            return -1;
        }

    ret = recv_all_timeout (fd_command, buf, 200, 0, &(struct timeval)
        {
            .tv_sec = NERDJACK_TIMEOUT});

    nerd_close_conn (fd_command);

    if (ret < 0)
        {
            verb ("Error_receiving_command\n");
            return -1;
        }

    //Slice off the "OK" from the string
    buf[strlen(buf)-2] = '\0';

    printf ("%s\n",buf);

    return 0;
}

/* Send the given command to address. The command should be something
 * of the specified length. This expects the NerdJack to reply with OK
 * or NO
 */
int
nerd_send_command (const char *address, void *command, int length)
{
    int ret, fd_command;
    char buf[3];
    fd_command = nerd_open (address, NERDJACK_COMMAND_PORT);
    if (fd_command < 0)

```

```

    {
        info ("Connect_failed:_%s:%d\n", address, NERDJACK_COMMAND_PORT);
        return -2;
    }

    /* Send request */
    ret = send_all_timeout (fd_command, command, length, 0, &(struct timeval)
        {
            .tv_sec = NERDJACK_TIMEOUT});
    if (ret < 0 || ret != length)
    {
        verb ("short_send_%d\n", (int) ret);
        return -1;
    }

    ret = recv_all_timeout (fd_command, buf, 3, 0, &(struct timeval)
        {
            .tv_sec = NERDJACK_TIMEOUT});

    nerd_close_conn (fd_command);

    if (ret < 0 || ret != 3)
    {
        verb ("Error_receiving_OK_for_command\n");
        return -1;
    }

    if (0 != strcmp ("OK", buf))
    {
        verb ("Did_not_receive_OK...Received_%s\n", buf);
        return -4;
    }

    return 0;
}

int
nerd_data_stream (int data_fd, int numChannels, int *channel_list,
                 int precision, int convert, int lines, int showmem,
                 unsigned short *currentcount, unsigned int period,
                 int wasreset)
{
    //Variables that should persist across retries
    static dataPacket buf;
    static int linesleft = 0;
    static int linesdumped = 0;

    //Variables essential to packet processing
    signed short datapoint = 0;
    int i;

    int numChannelsSampled = channel_list[0] + 1;

```

```

//The number sampled will be the highest channel requested plus 1
//i.e. channel 0 requested means 1 sampled)
for (i = 0; i < numChannels; i++)
{
    if (channel_list [i] + 1 > numChannelsSampled)
        numChannelsSampled = channel_list[i] + 1;
}

double voltline[numChannels];

unsigned short dataline[numChannels];

unsigned short packetsready = 0;
unsigned short adcused = 0;
unsigned short tempshort = 0;
int charsread = 0;

int numgroupsProcessed = 0;
double volts;

//The timeout should be the expected time plus 60 seconds
//This permits slower speeds to work properly
unsigned int expectedtimeout =
    (period * NERDJACK_NUM_SAMPLES / NERDJACK_CLOCK_RATE) + 60;

//Check to see if we're trying to resume
//Don't blow away linesleft in that case
if (lines != 0 && linesleft == 0)
{
    linesleft = lines;
}

//If there was a reset, we still need to dump a line because of faulty PDCA start
if (wasreset)
{
    linesdumped = 0;
}

//If this is the first time called, warn the user if we're too fast
if (linesdumped == 0)
{
    if (period < (numChannelsSampled * 200 + 600))
    {
        info ("You_are_sampling_close_to_the_limit_of_NerdJack\n");
        info ("Sample_fewer_channels_or_sample_slower\n");
    }
}

//Now destination structure array is set as well as numDuplicates.

int totalGroups = NERDJACK_NUM_SAMPLES / numChannelsSampled;

```

```

//Loop forever to grab data
while ((charsread =
    recv_all_timeout (data_fd, &buf, NERDJACK_PACKET_SIZE, 0,
                    &(struct timeval)
                    {
                        .tv_sec = expectedtimeout})))
{
    if (charsread != NERDJACK_PACKET_SIZE)
    {
        //There was a problem getting data. Probably a closed
        //connection.
        info ("Packet_timed_out_or_was_too_short\n");
        return -2;
    }

    //First check the header info
    if (buf.headerone != 0xF0 || buf.headertwo != 0xAA)
    {
        info ("No_Header_info\n");
        return -1;
    }

    //Check counter info to make sure not out of order
    tempshort = ntohs (buf.packetNumber);
    if (tempshort != *currentcount)
    {
        info ("Count_wrong_Expected_%hd_but_got_%hd\n", *currentcount,
            tempshort);
        return -1;
    }

    //Increment number of packets received
    *currentcount = *currentcount + 1;

    adcused = ntohs (buf.adcused);
    packetsready = ntohs (buf.packetsready);
    numgroupsProcessed = 0;

    if (showmem)
    {
        printf ("%hd_%hd\n", adcused, packetsready);
        continue;
    }

    //While there is still more data in the packet, process it
    while (numgroupsProcessed < totalGroups)
    {
        //Poison the data structure
        switch (convert)
        {
            case CONVERT_VOLTS:
                memset (voltline, 0, numChannels * sizeof (double));
                break;

```



```

default:
case CONVERT_HEX:
case CONVERT_DEC:
    memset (dataline, 0, numChannels * sizeof (unsigned char));
}

//Read in each group
for (i = 0; i < numChannels; i++)
{
    //Get the datapoint associated with the desired channel
    datapoint =
        ntohs (buf.
            data[channel_list [i] +
                numgroupsProcessed * numChannelsSampled]);

    //Place it into the line
    switch (convert)
    {
        case CONVERT_VOLTS:
            if (channel_list [i] <= 5)
            {
                volts =
                    (double) (datapoint / 32767.0) *
                    ((precision & 0x01) ? 5.0 : 10.0);
            }
            else
            {
                volts =
                    (double) (datapoint / 32767.0) *
                    ((precision & 0x02) ? 5.0 : 10.0);
            }
            voltline [i] = volts;
            break;
        default:
        case CONVERT_HEX:
        case CONVERT_DEC:
            dataline[i] = (unsigned short) (datapoint - INT16_MIN);
            break;
    }
}

//We want to dump the first line because it's usually spurious
if (linesdumped != 0)
{
    //Now print the group
    switch (convert)
    {
        case CONVERT_VOLTS:
            for (i = 0; i < numChannels; i++)
            {
                if (printf ("%lf.", voltline [i]) < 0)
                    goto bad;
            }
            break;
        case CONVERT_HEX:

```

```

        for (i = 0; i < numChannels; i++)
        {
            if (printf ("%04hX", dataline[i]) < 0)
                goto bad;
        }
        break;
    default:
    case CONVERT_DEC:
        for (i = 0; i < numChannels; i++)
        {
            if (printf ("%hu_", dataline[i]) < 0)
                goto bad;
        }
        break;
    }
    if (printf ("\n") < 0)
        goto bad;

    //If we're counting lines, decrement them
    if (lines != 0)
    {
        linesleft --;
        if (linesleft == 0)
        {
            return 0;
        }
    }

}
else
{
    linesdumped = linesdumped + 1;
}

//We've processed this group, so advance the counter
numgroupsProcessed++;

}
}

return 0;

bad:
    info ("Output_error_(disk_full?)\n");
    return -3;

}

/* Open a connection to the NerdJack */
int
nerd_open (const char *address, int port)
{

    struct hostent *he;

```

```

net_init ();

int32_t i32SocketFD = socket (PF_INET, SOCK_STREAM, 0);

if (-1 == i32SocketFD)
{
    verb ("cannot_create_socket");
    return -1;
}

/* Set nonblocking */
if (soblock (i32SocketFD, 0) < 0)
{
    verb ("can't_set_nonblocking\n");
    return -1;
}

struct sockaddr_in stSockAddr;
memset (&stSockAddr, 0, sizeof (stSockAddr));

stSockAddr.sin_family = AF_INET;
stSockAddr.sin_port = htons (port);

he = gethostbyname (address);
if (he == NULL)
{
    verb ("gethostbyname(\"%s\")_failed\n", address);
    return -1;
}
stSockAddr.sin_addr = *((struct in_addr *) he->h_addr);

debug ("Resolved_%s->%s\n", address, inet_ntoa (stSockAddr.sin_addr));

/* Connect */
if (connect_timeout
    (i32SocketFD, (struct sockaddr *) &stSockAddr, sizeof (stSockAddr),
    &(struct timeval)
    {
        .tv_sec = 3}) < 0)
{
    verb ("connection_to_%s:%d_failed:_%s\n",
        inet_ntoa (stSockAddr.sin_addr), port, compat_strerror (errno));
    return -1;
}

return i32SocketFD;
}

//Generate an appropriate sample initiation command
int
nerd_generate_command (getPacket * command, int *channel_list,
    int channel_count, int precision, unsigned long period)
{

```

```

short channelbit = 0;
int i;
int highestchannel = 0;

for (i = 0; i < channel_count; i++)
{
    if (channel_list[i] > highestchannel)
    {
        highestchannel = channel_list[i];
    }
    //channelbit = channelbit | (0x1 << channel_list[i]);
}

for (i = 0; i <= highestchannel; i++)
{
    channelbit = channelbit | (0x01 << i);
}

command->word[0] = 'G';
command->word[1] = 'E';
command->word[2] = 'T';
command->word[3] = 'D';
command->channelbit = htons(channelbit);
command->precision = precision;
command->period = htonl(period);
command->prescaler = 0;

return 0;
}

int
nerd_close_conn (int data_fd)
{
    shutdown (data_fd, 2);
    close (data_fd);
    return 0;
}

```

---

## F.3 Nerdconfig Source

This section contains the source for the nerdconfig utility for programming the Nerd-Jack device. It depends on a Python packaged called intelhex to parse the object files for NerdJack, but this code is not included. It can be freely downloaded from the Python package repository.

It also depends on dfu-programmer and libUSB, which can both be downloaded from SourceForge.

### F.3.1 configData.py

---

```
from intelhex import IntelHex
from intelhex import hex2bin
from subprocess import *
import sys
import StringIO
import csv
import tempfile
import os
import pkgutil
import re

class DFUException(Exception):
    def __init__( self , value ):
        self .value = value
    def __str__( self ):
        return repr(self.value)

def which(program):
    """Finds if a file exists in the path
    Don't forget about the extension
    """
    def is_exe(fpath):
        return os.path.exists(fpath) and os.access(fpath, os.X_OK)

    fpath, fname = os.path.split(program)
    if fpath:
        if is_exe(program):
            return program
    else:
        for path in os.environ["PATH"].split(os.pathsep):
            exe_file = os.path.join(path, program)
            if is_exe( exe_file ):
                return exe_file

    return None

class ConfigData:
    """Holds the data encoded in the User Page of the NerdJack device.
    """
    def __init__( self , chipInit=False):
        """ Initializes the internal representation of the User Page
        """
        if which('dfu-programmer') is None and which('dfu-programmer.exe') is None:
            print ""
            print "dfu-programmer is not installed and available in the path"
            print "Please make sure it is installed and try this program again"
            print ""
            sys.exit(-1)
        if (chipInit):
            self.readChip()
        else:
```

```

        self.hexfile = IntelHex()
        self.initbootloader()
        self.serial = "FF-FF-FF-FF-FF-FF"
        self.mac = "FF:FF:FF"
        for i in range(7):
            self.config[i].ipaddress = '255.255.255.255'
            self.config[i].gateway = '255.255.255.255'
            self.config[i].netmask = '255.255.255.255'

def __getattr__( self ,name):

    return {'serial': self.readserial ,
            'config': self.readconfig,
            'mac':self.readmac}[name]()

def __setattr__( self ,name,value):

    try:
        testdict = {'serial': self.writeserial ,
                    'mac':self.writemac}[name](value)
    except KeyError:
        self.__dict__[name] = value

def createCSV(self):
    outputbuffer = StringIO.StringIO()
    csvwriter = csv.writer(outputbuffer)
    for i in range(7):
        csvwriter.writerow([ self.config[i].ipaddress, \
                             self.config[i].netmask, \
                             self.config[i].gateway])

    output = outputbuffer.getvalue()
    outputbuffer.close()
    return output

def createTable(self):
    outputstring = "\nCurrent Configuration\n" + \
        "-----\n" + \
        "Serial_Number:" + self.serial + "\n" + \
        "MAC_Address:_00:04:25:" + self.mac + "\n" + \
        "Config_number".center(16) + "|" + \
        "IP_Address".center(16) + "|" + \
        "Netmask".center(16) + "|" + \
        "Gateway".center(16) + "\n"
    for i in range(7):
        outputstring = outputstring + str(i).center(16)+ "|" + \
            self.config[i].ipaddress.center(16)+ "|" + \
            self.config[i].netmask.center(16)+ "|" + \
            self.config[i].gateway.center(16)+ "\n"
    return outputstring

def shiftCodeDown(self,hexfile):
    """Strips_off_the_higher_address_information_by_shifting_this_data_downward

```

```

_____Required_if_using_batchisp_under_Windows.
_____
"""
for index in range(512*1024):
    #if self.hexfile[0x80800+index] != 0xff:
        #print "Copying byte from location: "+hex(0x80800+index)
        hexfile[index] = hexfile[0x8000000+index]
    #del self.hexfile[0x8080000+index]
    #else:
        # pass
        #print "No copy from address: "+hex(0x8080000+index)
    try:
        del hexfile[0x8000000+index]
    except KeyError:
        pass

def shiftAddressesDown(self):
    """Strips off the higher address information by shifting this data downward
_____Required_if_using_batchisp_under_Windows.
_____
"""
for index in range(512):
    #if self.hexfile[0x80800+index] != 0xff:
        #print "Copying byte from location: "+hex(0x80800+index)
        self.hexfile[index] = self.hexfile[0x80800000+index]
    #del self.hexfile[0x8080000+index]
    #else:
        # pass
        #print "No copy from address: "+hex(0x8080000+index)
    del self.hexfile[0x8080000+index]

def shiftAddressesUp(self):
    """Shifts data up in address to make batchisp happy
_____
"""
for index in range(512):
    self.hexfile[0x8080000+index]=self.hexfile[index]

del self.hexfile[:512]

def readISPVersion(self):
    """Returns the ISP version of the bootloader
_____
"""
version = StringIO.StringIO()
dfu_programmer = Popen(('dfu-programmer', 'at32uc3a0512'
    , 'get', 'bootloader-version'), stdout=PIPE)
for line in dfu_programmer.stdout:
    version.write(line)
dfu_programmer.wait()
dfu_programmer.stdout.close()
versionstring = version.getvalue()
version.close()
if dfu_programmer.returncode is not 0:
    raise DFUException("dfu-programmer_error:_" +str(dfu_programmer.returncode))
return versionstring

def readCode(self):

```

```

        """ Returns a string of the bin data coming out of the code memory
        """
        binfile = StringIO.StringIO()
        dfu_programmer = Popen(('dfu-programmer', 'at32uc3a0512', 'dump'), stdout=PIPE)
        for line in dfu_programmer.stdout:
            binfile.write(line)
        dfu_programmer.wait()
        dfu_programmer.stdout.close()
        binstring = binfile.getvalue()
        binfile.close()
        if dfu_programmer.returncode is not 0:
            raise DFUException("dfu-programmer_error:_" + str(dfu_programmer.returncode))
        return binstring

    def readChip(self):
        """ Initializes the instance from inputted chip data.
        First try dfu-programmer, then fall back to batchisp
        """
        if which('dfu-programmer') is not None or which('dfu-programmer.exe') is not None:
            # We found dfu-programmer. Use it
            dfu_programmer = Popen(('dfu-programmer', 'at32uc3a0512',
                                   'dump-user'), stdout=PIPE)
            self.hexfile = IntelHex()
            self.hexfile.loadfile(dfu_programmer.stdout, 'bin')
            dfu_programmer.wait()
            if dfu_programmer.returncode is not 0:
                raise DFUException("dfu-programmer_error:_"
                                   + str(dfu_programmer.returncode))
        else:
            # Fall back on batchisp
            try:
                (handle, filename) = tempfile.mkstemp(suffix='.hex')
                os.close(handle)
                batchisp =
                    Popen(('batchisp', '-device', 'at32uc3a0512', '-hardware', 'usb', '-operation', 'memory', 'USER', 're
                    # batchisp =
                    Popen(('batchisp', '-device', 'at32uc3a0512', '-hardware', 'usb', '-operation', 'memory', 'USER', 're

                batchisp.wait()
                if batchisp.returncode is not 0:
                    raise DFUException("Batchisp_error:_" + str(batchisp.returncode))
                self.hexfile = IntelHex()
                self.hexfile.loadfile(filename, 'hex')
                # self.hexfile.loadfile('read.hex', 'hex')
                self.shiftAddressesDown()
                # self.hexfile.dump()
            finally:
                os.remove(filename)
                pass
            # self.hexfile = IntelHex('test.hex')
            # self.hexfile.readfile()

    def initbootloader(self):
        """ Initializes the bootloader configuration section of the User Page

```



```

        """This must not be damaged or the bootloader will not work.
        """

        self.hexfile[508] = 0x92
        self.hexfile[509] = 0x9E
        self.hexfile[510] = 0x14
        self.hexfile[511] = 0x24

    def writemac(self, lowerin):
        """Writes the given MAC address. lowerthree should be a list of the
        last three bytes of the MAC address, MSB first
        """
        if isinstance(lowerin, str):
            macsplit = lowerin.split(':')
            lowerthree = []
            for byte in macsplit:
                lowerthree.append(int(byte,16))
        else:
            lowerthree = lowerin
        self.hexfile[84] = lowerthree[0]
        self.hexfile[85] = lowerthree[1]
        self.hexfile[86] = lowerthree[2]

    def readmac(self):
        """Reads the MAC address from the current image. Returns a tuple
        of the last 3 bytes, MSB first.
        """
        return hex(self.hexfile[84][2:] + ':' + \
            hex(self.hexfile[85][2:] + ':' + \
            hex(self.hexfile[86][2:]

    def writeaddress(self, addressnum, addrin):
        """Writes the address in network byte order
        This happens because we are manually setting the bytes of the list
        """
        if ((addressnum > 20) or (addressnum < 0)):
            raise Exception("Address number invalid")
        if isinstance(addrin, str):
            addrsplit = addrin.split('.')
            addresses = []
            for byte in addrsplit:
                addresses.append(int(byte,10))
        else:
            addresses = addrin
        self.hexfile[addressnum*4] = addresses[0]
        self.hexfile[1+addressnum*4] = addresses[1]
        self.hexfile[2+addressnum*4] = addresses[2]
        self.hexfile[3+addressnum*4] = addresses[3]

    def readaddress(self, addressnum):
        """Reads the requested address and returns a tuple in network
        byte order.
        """
        return str(self.hexfile[addressnum*4]) + '.' + \

```

```

        str ( self . hexfile [1+addressnum*4] ) + '.' + \
        str ( self . hexfile [2+addressnum*4] ) + '.' + \
        str ( self . hexfile [3+addressnum*4] )

def writeserial ( self , serialin ) :
    """ Records_the_given_serial_number_to_the_User_Page
    """
    if isinstance ( serialin , str ) :
        serialsplit = serialin . split ( '-' )
        serialnum = []
        for byte in serialsplit :
            serialnum.append(int(byte,16))
    else :
        serialnum = serialin

    self . hexfile [87] = serialnum[0]
    self . hexfile [88] = serialnum[1]
    self . hexfile [89] = serialnum[2]
    self . hexfile [90] = serialnum[3]
    self . hexfile [91] = serialnum[4]
    self . hexfile [92] = serialnum[5]

def readserial ( self ) :
    """ Reads_the_current_serialnumber.
    """
    return hex(self.hexfile [87] [2:]+ '-' + \
        hex(self . hexfile [88] [2:]+ '-' + \
        hex(self . hexfile [89] [2:]+ '-' + \
        hex(self . hexfile [90] [2:]+ '-' + \
        hex(self . hexfile [91] [2:]+ '-' + \
        hex(self . hexfile [92] [2:]

def readconfig( self ) :
    """ Reads_out_config_information_from_the_User_Page.
    Returns_a_list_of_all_of_the_config_setups
    """
    configlist = []

    for confignum in range(7):
        configlist .append(IPConfig( confignum, self.hexfile ))

    return configlist

def writeout( self , file ) :
    self . hexfile . tofile ( file , 'hex' )

def exitDFU(self):
    if which('dfu-programmer') is not None or which('dfu-programmer.exe') is not None:
        dfu_programmer = Popen(('dfu-programmer', 'at32uc3a0512',
            'setfuse', 'ISP_FORCE', '0'))
        dfu_programmer.wait()
        if dfu_programmer.returncode is not 0:

```

```

        raise DFUException("DFU-Programmer_Setfuse_error:_"
                            +str(dfu_programmer.returncode))
    else:
        #Fall back on batchisp

        batchisp =
            Popen(('batchisp', '-device', 'at32uc3a0512', '-hardware', 'usb', '-operation', 'start ', 'reset ', '0'))
        batchisp.wait()
        if batchisp.returncode is not 0:
            raise DFUException("Batchisp_error:_" +str(batchisp.returncode))

def programChip(self):
    if which('dfu-programmer') is not None or which('dfu-programmer.exe') is not None:
        dfu_programmer = Popen(('dfu-programmer', 'at32uc3a0512', 'flash-user',
                                'STDIN'),stdin=PIPE)
        self.writeout(dfu_programmer.stdin)
        dfu_programmer.wait()
        if dfu_programmer.returncode is not 0:
            raise DFUException("DFU-Programmer_error:_"
                                +str(dfu_programmer.returncode))
    else:
        #Fall back on batchisp
        try:
            (handle, filename) = tempfile.mkstemp(suffix='.hex')
            f = os.fdopen(handle,"w")
            #f = open('test.hex', 'w')
            self.shiftAddressesUp()
            self.writeout(f)
            f.close()
            self.shiftAddressesDown()
            batchisp =
                Popen(('batchisp', '-device', 'at32uc3a0512', '-hardware', 'usb', '-operation', 'memory', 'USER', 'load'))
            batchisp.wait()
            if batchisp.returncode is not 0:
                raise DFUException("Batchisp_error:_" +str(batchisp.returncode))
        finally:
            os.remove(filename)

def versionString( self , file == 'default'):
    if file == 'default':
        code = pkgutil.get_data('nerdjack', 'data/code.hex')
    elif file == 'chip':
        binstring = self.readCode()
        versre = re.search('NERD:_(.*?)\n',binstring)
        return versre.group(1)
    else:
        code = open(file).read()

codefile = StringIO.StringIO(code)

hexfile = IntelHex()
hexfile.loadfile ( codefile , 'hex')

```

```

self .shiftCodeDown(hexfile)
binstring = hexfile .tobinstr ()

versre = re.search('NERD:_(.*?)\n',binstring)
return versre.group(1)

def programCode(self,file='default'):
    if which('dfu-programmer') is not None or which('dfu-programmer.exe') is not None:
        dfu_programmer = Popen(('dfu-programmer', 'at32uc3a0512', 'erase'))
        dfu_programmer.wait()
        if dfu_programmer.returncode is not 0:
            raise DFUException("DFU-Programmer.Erase_error:_
                               "+str(dfu_programmer.returncode))

        if file == 'default':
            code = pkgutil.get_data('nerdjack', 'data/code.hex')
        else:
            code = open(file).read()
        dfu_programmer = Popen(('dfu-programmer', 'at32uc3a0512', 'flash', 'STDIN',
                               '--suppress-bootloader-mem'),stdin=PIPE)
        dfu_programmer.stdin.write(code)
        dfu_programmer.wait()

        if dfu_programmer.returncode is not 0:
            raise DFUException("DFU-Programmer.Flash_error:_
                               "+str(dfu_programmer.returncode))

    else:
        #Fall back on batchisp
        try:
            (handle, filename) = tempfile.mkstemp(suffix='.hex')
            f = os.fdopen(handle,"w")
            code = pkgutil.get_data('nerdjack', 'data/code.hex')
            f.write(code)
            self .writeout(f)
            f.close ()
            batchisp =
                Popen(('batchisp', '-device', 'at32uc3a0512', '-hardware', 'usb', '-operation', 'erase', 'f', 'memory
                       ' flash ', 'blankcheck', 'loadbuffer', filename, 'program', 'verify'))
            batchisp.wait()
            if batchisp.returncode is not 0:
                raise DFUException("Batchisp_error:_"+str(batchisp.returncode))
        finally:
            os.remove(filename)

class IPConfig(ConfigData):
    """ Holds the netmask, gateway, and IP address of a configuration record
    """
    def __init__ ( self ,confignum, hexfile):
        """ Initializes the number of the config
        Must be between 0 and 6
        """

```

```

    if (confignum < 0 or confignum > 6):
        raise Exception("Config_number_invalid")

    self.confignum = confignum
    self.hexfile = hexfile

def __getattr__( self , name):
    return { 'ipaddress': self.readip,
            'netmask':self.readnetmask,
            'gateway':self.readgateway}[name]()

def __setattr__( self , name,value):
    try:
        { 'ipaddress': self.writeip,
          'netmask':self.writenetmask,
          'gateway':self.writegateway}[name](value)
    except KeyError:
        self.__dict__[ name] = value

def readip( self ):
    """Reads_the_specified_ipaddress_as_a_tuple
    """
    return self.readaddress( self.confignum)

def readnetmask(self):
    """Reads_the_netmask_as_a_tuple
    """
    return self.readaddress( self.confignum+7)

def readgateway(self):
    """Reads_the_gateway_as_a_tuple
    """
    return self.readaddress( self.confignum+14)

def writeip( self , address):
    """Writes_the_ipaddress
    """
    self.writeaddress( self.confignum,address)

def writenetmask(self,address):
    """Writes_the_netmask
    """
    self.writeaddress( self.confignum+7,address)

def writegateway(self,address):
    """Writes_the_gateway
    """
    self.writeaddress( self.confignum+14,address)

```

---

### F.3.2 nerdconfig.py

```
#!/opt/local/bin/python
```

```

from nerdjack import ConfigData
from nerdjack import DFUException
from optparse import OptionParser
import csv
import sys
import os.path
import subprocess
import pkgutil
import StringIO
import random

versionNumber = "1.0"

def readChip():
    chip = ConfigData(chipInit=True)

def getDataFile(mode):
    #First check to see if the data directory exists
    if (not os.path.exists('./serialdata')):
        #We were not in a working copy so check one out
        retval = subprocess.call(["svn","co","https://bucket.mit.edu/svn/"+\
            "nilm/acquisition/customboard/deviceprogrammer/serialdata","serialdata"])
        if retval is not 0:
            raise Exception("Error_with_subversion_checkout")

    #Check status to see if data is a working copy and if there are any
    #local changes to trash
    subversion = subprocess.Popen(['svn','status',' serialdata '], stdout=subprocess.PIPE)
    statusout = subversion.stdout.readline()
    subversion.wait()
    if subversion.returncode is not 0:
        raise Exception("Error_with_subversion_status")

    if "not_a_working_copy" in statusout:
        raise Exception("serialdata_exists_but_is_not_a_working_copy.\n"+ \
            "Please_delete_it_and_try_again.")

    #We know data exists and is a working copy. Try to update it
    subup = subprocess.Popen(['svn','up',' serialdata '])
    subup.wait()
    if subup.returncode is not 0:
        raise Exception("Error_with_subversion_update")

    #Now we have updated. Make sure that the status is clear
    subversion = subprocess.Popen(['svn','status',' serialdata '], stdout=subprocess.PIPE)
    statusout = subversion.stdout.readline()
    subversion.wait()
    if subversion.returncode is not 0:
        raise Exception("Error_with_second_subversion_status")

    if len(statusout) is not 0:
        raise Exception("Local_changes_exist_in_serialdata._Please_correct.")

```

```

#The data is definitely up to date. Return a file object
return open('./serialdata/nerdjack.csv',mode)

def flushtoSVN():
    #We assume that the subversion changes are ours and that we are working
    #with a valid working copy. This should be satisfied if getDataFile
    #was used
    retcode = subprocess.call(['svn', 'commit', 'serialdata'])
    if retcode is not 0:
        raise Exception("Error with subversion commit")

def getSerials():
    datafile = getDataFile('rb')
    stringserials = csv.reader(datafile)

    serials = []
    allmac = []

    for row in stringserials :
        #Convert to list from string
        serials.append(map(lambda x: int(x,16), row[0].split('-')))
        allmac.append(map(lambda x: int(x,16), row[1].split(':')))

    datafile.close()
    return (serials,allmac)

def getHole(serials):
    hole = [0,0,0,0,0,0]
    for j in reversed(range(6)):
        for i in range(256):
            hole[j] = i
            if hole not in serials :
                break
    return hole

def CSVtoChip(filename,chip,type='filename'):
    if isinstance(filename, str):
        if type == 'filename':
            defaultsettings = csv.reader(open(filename,'rb'))
        elif type == 'data':
            defaultsettings = csv.reader(StringIO.StringIO(filename))
        else:
            raise Exception("Unrecognized data type to CSVtoChip")
    else:
        defaultsettings = csv.reader(filename)
    confnumber = 0
    for line in defaultsettings :
        chip.config[confnumber].ipaddress = line[0]
        chip.config[confnumber].netmask = line[1]
        chip.config[confnumber].gateway = line[2]
        confnumber = confnumber + 1

def initializeChip (hole, chip, mac):

```

```

    #This is a new chip that needs a serial and MAC address
    chip.serial = hole
    chip.mac = mac
    chip.programChip()

def printDirections():
    print "The changes requested have been made."
    print "Switch the programming DIP switch (Switch 4) to the"
    print "OFF position and reset the NerdJack."
    print ""

#Find if the read serial number is legitimate
#if chip.serial not in sortedserials:
#    print "Chip serial number unrecognized!"

if __name__ == '__main__':
    parser = OptionParser()
    parser.add_option("-f", "--file", dest="filename",
        help="read configuration from FILE", metavar="FILE")
    parser.add_option("-p", "--pipe", dest="pipe", default=False,
        help="read configuration from stdin", action="store_true")
    parser.add_option("-c", "--csv", dest="csv", default=False,
        help="output configuration info in CSV", action="store_true")
    parser.add_option("-m", "--mac", dest="mac",
        help="Set last three bytes of MAC address to MAC", metavar="MAC")
    parser.add_option("-d", "--display-serial", dest="display", default=False,
        action="store_true", help="Print MAC address and serial number")
    parser.add_option("-s", "--standard-config", dest="standard", default=False,
        help="Use standard configuration settings", action="store_true")
    parser.add_option("-b", "--burn", dest="code", default=False,
        help="Program NerdJack code into flash", action="store_true")
    parser.add_option("-i", "--install", dest="installfile",
        help="Install HEXFILE on NerdJack", metavar="HEXFILE")
    parser.add_option("-S", "--use-subversion", dest="subversion", default=False,
        help="Use Subversion to manage serialization", action="store_true")
    parser.add_option("-r", "--reburn-serial", dest='changeserial',
        default=False,
        help="Generate and burn another random serial number",
        action="store_true")
    parser.add_option("-V", "--version", dest="version", default=False,
        help="Print version of Nerdconfig and bundled firmware then exit",
        action="store_true")
    parser.add_option("-R", "--read-version", dest="readversion", default=False,
        help="Read firmware revision from attached NerdJack",
        action="store_true")

    (options, args) = parser.parse_args()

    if(options.version):
        chip = ConfigData(chipInit=False)
        print "Nerdconfig version_" + versionNumber
        print "Reading version from bundled firmware..."
        print "Bundled NerdJack Firmware is_" + chip.versionString()

```



```

sys.exit(0)

if(options.pipe and options.filename is not None):
    parser.error("Cannot take input from file and STDIN")

if(options.standard and options.pipe):
    parser.error("Cannot use standard config and STDIN")

if(options.standard and options.filename is not None):
    parser.error("Cannot use standard config and input file")

if(options.code and options.installfile is not None):
    parser.error("Cannot use standard NerdJack and custom code")

try:
    chip = ConfigData(chipInit=True)

    if((chip.serial == 'ff-ff-ff-ff-ff-ff') and (options.filename is None) and (not
        options.pipe) and (not options.standard) and (not options.changeserial) and (not
        options.code) and (options.installfile is None) and (options.mac is None)):
        #This should fire if it's a blank chip with no options given
        print "\nThis is a blank chip."
        print ""
        print "Assuming -s and -b options to burn with standard"
        print "settings and stock NerdJack firmware"
        options.standard = True
        options.changeserial = True
        options.code = True

    performProgram = options.filename is not None or options.pipe or \
        options.mac is not None or options.standard or options.changeserial

    if options.filename is not None:
        CSVtoChip(options.filename,chip)

    if options.pipe:
        CSVtoChip(sys.stdin,chip)

    if options.standard:
        CSVtoChip(pkgutil.get_data('nerdjack','data/default.csv'),chip,type='data')

    if options.mac is not None:
        chip.mac = options.mac

    if(options.changeserial):
        if(options.subversion):
            (serial,allmac) = getSerials()
            hole = getHole(serial)
        else:
            hole = random.sample(range(256),6)
        if not options.mac:
            #Change the MAC with the serial unless told otherwise
            mac = (hole[3],hole[4],hole[5])

```

```

print ""
print " Initializing _chip_with_new_serial_number"
print " and_MAC_address..."
print ""
initializeChip (hole,chip,mac)
if(options.subversion):
    datafile = getDataFile('ab')
    datawriter = csv.writer( datafile )
    datawriter.writerow([chip. serial ,chip.mac])
    datafile .close ()
    flushtoSVN()
else:
    #This is only needed if we didn't change the serial number
    if performProgram:
        print "Programming_specified_settings_into_NerdJack"
        chip.programChip()

if(options.code):
    print "Reading_version_information_from_bundled_firmware..."
    print "Burning_NerdJack_Firmware_" + chip.versionString()
    chip.programCode()

if(options.installfile is not None):
    print "Burning_specified_firmware"
    chip.programCode(options.installfile)

if(options.readversion):
    print "Reading_firmware_version..."
    print "Installed_Firmware_is_" + chip.versionString('chip')
    print chip.readISPVersion()

if(options.display):
    print chip. serial +",_" +chip.mac

if(options.csv):
    print chip.createCSV()
else:
    if not options.display:
        print chip.createTable()

chip.exitDFU()

if(options.code or performProgram or options.installfile):
    printDirections ()
    sys.exit(0)

except DFUException:
    print ""
    print "There_was_an_error_communicating_with_the_NerdJack_over_USB."
    print "It_might_need_to_be_reset_after_you_last_ran_NerdConfig."
    print ""
    print "If_that_does_not_work,"

```

```
print "Ensure that drivers are installed, the NerdJack is connected,"  
print "and that the programming DIP switch (switch 4) is in the ON"  
print "position."  
sys.exit(-1)
```

---



# Appendix G

## BurnIt Source Code Listing

### G.1 ATMEGA Firmware

This section contains the C code for BurnIt firmware. The programming algorithms for the PIC16F628, GAL22V10, and AT89C2051 are implemented here for the BurnIt hardware. It can be compiled using the AVR port of the GNU Compiler Collection toolchain. It is meant to be programmed to the BurnIt ATMEGA644 prior to installation in the BurnIt PCB.

Portions of this code relevant for programming the AT89C2051 were originally authored by Chris Whittaker and modified for the current revision of BurnIt.

The PIC code was borrowed and modified from `jimpic` written by Jim Paris.

The GAL programming algorithm and portions of code were modified from `GAL-Blast` written by Manfred Winterhoff.

#### G.1.1 2051.h

---

```
/*
 * Standard port definitions for a MEGA644,
 * This should be all you need to change for a different micro.
 */

/* the port that data is transferred to and from the device with*/
#define DATA_PORT PORTA
#define DATA_DDR DDRA
#define DATA_PIN PINA

/* the port that controls the device (program codes, etc.)*/
#define CONTROL_PORT PORTB
#define CONTROL_DDR DDRB
#define CONTROL_PIN PINB

#define RSTDDR DDRB
#define RSTPORT PORTB
```

```

#define SENSE2051PIN PINC
#define SENSE2051 PC3

#define RST5 PB0
#define RST12 PB1
#define XTAL PB7
#define P32 PB6
#define P33 PB5
#define P34 PB4
#define P35 PB3
#define P37 PB2

#define FEEDBACK_PORT PORTC
#define FEEDBACK_DDR DDRC
#define FEEDBACK_PIN PINC

#define P31 PC0

//static void print_ihex(void);
void atmel_pulse_xtal(void);
void atmel_write_chip(void);
void atmel_clear_chip(void);
int atmel_signature(void);
void atmel_view(void);
void atmel_verify(void);
void atmel_program(void);

```

---

## G.1.2 2051.c

---

```

/*
 * Routines for interacting specifically with an Atmel 2051/4051
 */

#include <stdio.h> /* used for string manipulation procedures*/
#include <avr/io.h> /* direct access to the AVR's IO ports/SFRs */
#include <avr/pgmspace.h> /* for accessing strings in program memory */

#include <ctype.h> /* to convert characters from upper to lowercase */

#include "avrutils.h"
#include "2051.h"

/* Iterate through the BurnIt chip memory and display on the
 * serial port, in ascii-coded hex
 *
 */

//static void print_ihex() {

```

```

//  printstr_p (PSTR("Displaying code in RAM\n"));
//  printstr_p (PSTR("ADDR : \tDATA\n"));
//  /*go thru all of memory*/
//  int16_t ix;
//  for(ix = 0; ix < CHIPMEMSIZE; ix++) {
//      if((ix % 32) == 0) {          /*line numbers and newlines */
//          printstr_p (PSTR("\n"));
//          char line_num[10];
//          sprintf(line_num, "%04X : \t", ix);
//          printstr (line_num);
//
//      }
//
//      uint8_t curr_mem = chipMemory[ix];
//      printhex(curr_mem);
//
//  }
// }
// }
/*
 *   Pulse XTAL1 on the ATMEL 2051/4051 to advance the programming
 *   memory location.
 */
void atmel_pulse_xtal(void) {
    CONTROL_PORT |= _BV(XTAL);
    _delay_us(1);
    CONTROL_PORT &= ~_BV(XTAL);
    _delay_us(1);
}

/*
 *   Write the data currently contained in memory to an Atmel 2051/4051
 *
 */
void atmel_write_chip(void) {
    printstr_p (PSTR("\nWriting chip..."));
    CONTROL_PORT = 0x00;
    DATA_PORT = 0x00;
    _delay_us(10);
    CONTROL_PORT |= _BV(RST5) | _BV(P32);    //5v on RST, P3.2 high

    CONTROL_PORT &= ~( _BV(P33)); // L H H H set mode bits.
    CONTROL_PORT |= _BV(P34) | _BV(P35) | _BV(P37);
    _delay_us(2);

    CONTROL_PORT |= _BV(RST12); // Raise up to 12V (Enable programming)
    _delay_us(30);

    int16_t ix;
    for(ix = 0; ix < CHIPMEMSIZE; ix++) {
        if((ix % 128) == 0) {          /*progress indicator*/
            printstr_p (PSTR("."));
        }
    }
}

```

```

    uint8_t curr_mem = chipMemory[ix];
    DATA_PORT = curr_mem;
    _delay_us(2);
    CONTROL_PORT &= ~(_BV(P32));
    _delay_us(10);
    CONTROL_PORT |= _BV(P32);

    loop_until_bit_is_clear (FEEDBACK_PIN, P31);
    loop_until_bit_is_set (FEEDBACK_PIN, P31);

    atmel_pulse_xtal();
    _delay_us(2);

}
_delay_ms(10);
CONTROL_PORT &= ~(_BV(RST12));
_delay_ms(10);
CONTROL_PORT = 0x00;
DATA_PORT = 0x00;
printstr_p (PSTR("_OK.."));
}
/*
 * Erase an atmel 2051/4051. Sets all bytes to 0xFF.
 */
void atmel_clear_chip(void) {
    printstr_p (PSTR("\nClearing chip..."));
    CONTROL_PORT = 0x00;
    CONTROL_PORT |= _BV(RST5) | _BV(P32);

    CONTROL_PORT |= _BV(P33); //H L L L
    CONTROL_PORT &= ~(_BV(P34)|_BV(P35)|_BV(P37));
    _delay_us(10);
    CONTROL_PORT |= _BV(RST12); // Raise up to 12V
    _delay_us(12);
    CONTROL_PORT &= ~(_BV(P32));
    _delay_ms(10);
    CONTROL_PORT |= _BV(P32);
    _delay_ms(10);
    CONTROL_PORT &= ~(_BV(RST12));
    _delay_ms(10);
    CONTROL_PORT = 0x00;
    printstr_p (PSTR("OK."));
}
/*
 * Request a signature bytes from the AT2051/4051.
 * Returns 0 if a known signature is given
 * 1 if invalid signature is given
 */

```



```

int atmel_signature(void) {
    printstr_p (PSTR("\nDetermining_chip_type..."));
    DATA_DDR = 0x00;
    CONTROL_PORT = 0x00;
    _delay_us(10);
    CONTROL_PORT |= _BV(RST5) | _BV(P32);

    CONTROL_PORT &= ~(_BV(P33)|_BV(P34)|_BV(P35)|_BV(P37)); // L L L L

    printstr_p (PSTR("0x"));
    _delay_us(2);    //time until data is valid

    uint8_t manufact_byte;
    manufact_byte = DATA_PIN;
    printhex(manufact_byte);

    printstr_p (PSTR("_0x"));
    atmel_pulse_xtal();
    _delay_us(2);

    uint8_t part_byte;
    part_byte = DATA_PIN;
    printhex(part_byte);

    CONTROL_PORT = 0x00;
    DATA_DDR = 0xFF;
    DATA_PORT = 0x00;

    if(manufact_byte == 0x1e && part_byte == 0x21) {
        printstr_p (PSTR("...Atmel_2051..."));
        return 0;
    } else if (    manufact_byte == 0x1e && part_byte == 0x41) {
        printstr_p (PSTR("...Atmel_4051..."));
        return 0;
    } else {
        printstr_p (PSTR("\nUnknown_Part...Check_Connections"));
        return 1;
    }
}

/*
 *   View the Data currently on the chip
 */
void atmel_view(void) {

    DATA_DDR = 0x00;
    CONTROL_PORT = 0x00;
    _delay_us(10);
    CONTROL_PORT |= _BV(RST5) | _BV(P32);

    CONTROL_PORT &= ~(_BV(P33)|_BV(P34)); // L L H H
    CONTROL_PORT |= _BV(P35) | _BV(P37);

```

```

_delay_us(2);

int16_t ix;
for(ix = 0; ix < CHIPMEMSIZE; ix++) {
    if((ix % 32) == 0) {          /*line numbers and newlines */
        printstr_p(PSTR("\n"));
        char line_num[10];
        sprintf(line_num, "%04X:\t", ix);
        printstr(line_num);

    }

    uint8_t curr_mem = DATA_PIN;
    printheex(curr_mem);
    atmel_pulse_xtal();
    _delay_us(2);

}
CONTROL_PORT = 0x00;
DATA_DDR = 0xFF;
DATA_PORT = 0x00;

}
/*
 * Checks to see if the data in the internal representation of
 * the chip's memory matches the data on the currently inserted chip
 */
void atmel_verify(void) {
    printstr_p(PSTR("\nVerifying_program_data..."));
    DATA_DDR = 0x00;
    CONTROL_PORT = 0x00;
    _delay_us(10);
    CONTROL_PORT |= _BV(RST5) | _BV(P32);

    CONTROL_PORT &= ~(_BV(P33) | _BV(P34)); // L L H H
    CONTROL_PORT |= _BV(P35) | _BV(P37);
    _delay_us(2);

    int16_t ix;
    for(ix = 0; ix < CHIPMEMSIZE; ix++) {
        if((ix % 128) == 0) {      /*progress indicator*/
            printstr_p(PSTR("."));
        }
        uint8_t curr_mem = DATA_PIN;
        if(curr_mem != chipMemory[ix]) {
            printstr_p(PSTR("Verification_Failed!"));
            CONTROL_PORT = 0x00;
            DATA_DDR = 0xFF;
            DATA_PORT = 0x00;
            return;
        }
    }
    atmel_pulse_xtal();
    _delay_us(2);
}

```

```

    }
    CONTROL_PORT = 0x00;
    DATA_DDR = 0xFF;
    DATA_PORT = 0x00;
    printstr_p(PSTR("_OK."));
}
/*
 *   Go through the steps of programming:
 *       check the signature, if invalid then do nothing
 *       if valid, clear the chip, write the chip and then verify it
 */
void atmel_program(void) {
    printstr_p(PSTR("\nPreparing to program Atmel 2051/4051..."));
    if(atmel_signature() == 0) {
        atmel_clear_chip();
        atmel_write_chip();
        atmel_verify();
    }
    //verify chip
}

```

---

### G.1.3 avrutils.h

---

```

/*
 * Some simple UART IO functions.
 * modified from AVR Freaks
 */

#define F_CPU 8000000UL          /* CPU clock in Hertz (8 MHz) for internal RC
    Oscillator*/
#include <util/delay.h>        /* for precision delays using the clock freq */

/* The representation in RAM of the memory to be burned */
#define CHIPMEMSIZE 2048
extern uint8_t chipMemory[CHIPMEMSIZE];

/*
 * Send character c down the UART Tx, wait until tx holding register
 * is empty.
 */
void putchar(char c);

/* Get a character from the UART Rx, wait until the rx holding register
 * is set
 */
char getch(void);

/*
 * Send a C (NUL-terminated) string down the UART Tx.
 */
void printstr(const char *s);

```

```

/*
 * Same as above, but the string is located in program memory,
 * so "lpm" instructions are needed to fetch it.
 */
void printstr_p(const char *s);

/*
 * returns a char given a received ascii character
 */
unsigned char ascii_to_bin( char data );

void printhex(uint8_t print_byte);

```

---

## G.1.4 avrutils.c

---

```

/*
 * Some simple UART IO functions.
 * modified from AVRfreaks
 */

#include <stdio.h> /* used for string manipulation procedures*/
#include <avr/io.h> /* direct access to the AVR's IO ports/SFRs */
#include <avr/pgmspace.h> /* for accessing strings in program memory */
#include <ctype.h> /* to convert characters from upper to lowercase
 */
#include "avrutils.h"
uint8_t chipMemory[CHIPMEMSIZE];
/*
 * Send character c down the UART Tx, wait until tx holding register
 * is empty.
 */
void putchar(char c)
{
    loop_until_bit_is_set (UCSR0A, UDRE0);
    UDR0 = c;
}

/* Get a character from the UART Rx, wait until the rx holding register
 * is set
 */
char getchr(void) {
    loop_until_bit_is_set (UCSR0A, RXC0);
    return UDR0;
}

/*
 * Send a C (NUL-terminated) string down the UART Tx.
 */
void printstr(const char *s)
{
    while (*s)

```

```

    {
        if (*s == '\n')
            putchar('\r');
        putchar(*s++);
    }
}

/*
 * Same as above, but the string is located in program memory,
 * so "lpm" instructions are needed to fetch it.
 */
void printstr_p(const char *s)
{
    char c;

    for (c = pgm_read_byte(s); c; ++s, c = pgm_read_byte(s))
    {
        if (c == '\n')
            putchar('\r');
        putchar(c);
    }
}

/*
 * returns a char given a received ascii character
 */
unsigned char ascii_to_bin( char data )
{
    if( data < 'A' )
    {
        return( data - '0' );
    }
    else
    {
        return( data - 55 );
    }
}

/*end simple IO functions */

void printhex(uint8_t print_byte) {
    char string_byte [3];
    sprintf (string_byte, "%02X", print_byte);
    printstr (string_byte);
}

```

---

## G.1.5 burnitall.c

```

/* BURNIT ALL
 * This simple program allows an ATMEL 2051, a PIC 16F628, or a GAL22V10 part to be
 * burned
 * using an interface similar to the MINMON application used by MIT's
 * Microprocessor Project Laboratory (6.115).
 *

```

```

*   Original 2051 work by Steve Whittaker with PIC and 22V10 additions by Zachary Clifford
*
*   Steve Whittaker, April–May 2007
*   Zachary Clifford Jan–Aug 2008
*/

#include <stdio.h>      /* used for string manipulation procedures*/
#include <avr/io.h>     /* direct access to the AVR's IO ports/SFRs */
#include <avr/pgmspace.h> /* for accessing strings in program memory */

#include <ctype.h>     /* to convert characters from upper to lowercase */

#include "avrutils.h"
#include "pic.h"       /* Constants for PIC programming*/
#include "gal.h"
#include "2051.h"

//0 is ATMEL mode, 1 is PIC mode, 2 is 22V10 mode
static uint8_t currentMode = 0;

/*
* Set up the UART using the cpu clock defined above.
* 9600 Baud / 8-bit / 1-stop bit
*/
static void ioinit(void) {
    //baud rate calculation
    UBRRH = 0UL;
    UBRRL = ((F_CPU / (16 * 9600UL)) - 1); //9600 Bd (calculated using F_CPU,
    // defined above)

    UCSRB = _BV(TXEN)|_BV(RXEN); // tx/rx enable
    UCSRC = _BV(UCSZ00)|_BV(UCSZ01); // 8 bit bit
    UCSRC &= ~(_BV(USBS0)); // 1 stop bit
}

/*
* Reads in a Intel HEX file over the serial port and
* stores the data in memory
* Also understands if PIC is being used.
*
* Some hex processing hints taken from AVRFreaks.net forums.
*/
static uint8_t load_ihex() {
    printstr_p(PSTR("\n>")); //the programming prompt
    uint8_t data_pairs, address_hi, address_lo, temp_byte, i, checksum, type;

    while(1) {

        while (getchr() != ':') // go forward until we get a colon
        {
            ;
        }

        /*get the number of ascii character pairs on this line*/

```

```

data_pairs = ascii_to_bin( getch() ) << 4;
data_pairs |= ascii_to_bin( getch() );

/*address to write to*/
address_hi = ascii_to_bin( getch() ) << 4;
address_hi |= ascii_to_bin( getch() );

address_lo = ascii_to_bin( getch() ) << 4;
address_lo |= ascii_to_bin( getch() );

/* get the data type */
type = ascii_to_bin( getch() ) << 4;
type |= ascii_to_bin( getch() );

checksum = (type + address_lo + address_hi + data_pairs) & 0xFF;

if(type != 0 && type != 1) {
    //Something is wrong
    //Borrowed error handling from JimPic by Jim Paris
    if(type==0x04 && type==0x02) {
        /* Ignore this one silently; not sure why
        so many programs include it. */
    } else {
        printstr_p(PSTR("\nWarning: ignoring_HEX_record"));
        char s[20];
        sprintf(s, "type_%02x\n",type);
        printstr(s);
    }
    continue;
}

for( i = 0; i < data_pairs; i++)
{
    temp_byte = ascii_to_bin( getch() ) << 4;
    temp_byte |= ascii_to_bin( getch() );
    checksum += temp_byte;
    if((address_lo + 256*address_hi + i <= 0x400F) &&
        (address_lo + 256*address_hi + i >= 0x4000))
    {
        //It's PIC configuration bytes
        picConfigMemory[address_lo + i] = temp_byte;
    } else if((address_lo + 256*address_hi + i >= 0x4200) &&
        (address_lo + 256*address_hi + i <= 0x4280))
    {
        //It's PIC Data. That usually lives in 0x2100 to 0x2180
        //Doubled to 0x4200 to 0x4280
        picDataMemory[address_lo + i] = temp_byte;
    } else if(address_lo + 256*address_hi + i >= CHIPMEMSIZE) {
        //It is bigger than normal data but not any of the
        //other special cases
        printstr_p(PSTR("\nERROR: This_hex_file_has_data_in_high_memory"
            "that_does_not_correspond\n"));
        printstr_p(PSTR("to_Data_or_Config_memory...It_might_also_be"
            "too_big_for_this_chip."));
    }
}

```

```

        return -1;
    } else {
        //It's plain old data.
        if((currentMode == 1) &&
            ((PICtype == PIC16F627) || (PICtype == PIC16F627A)) &&
            ((uint16_t)(address_lo + 256*address_hi + i) > 1024))
        {
            //It is a small PIC and we're writing outside
            //its implemented memory
            printstr_p(PSTR("\nERROR:_This_hex_file_has_more_data"
                "_than_the_PIC16F627_can_hold."));
            printstr_p(PSTR("\nCheck_your_compiler_settings_or_get_a_16F628."));
            return -1;
        }
        chipMemory[address_lo + 256*address_hi + i] = temp_byte;
    }
}

/*and the checksum*/
temp_byte = ascii_to_bin( getch() ) << 4;
temp_byte |= ascii_to_bin( getch() );

if(((checksum + temp_byte) & 0xFF) != 0) {
    printstr_p(PSTR("\nERROR:_Checksum_invalid_in_hex_file."));
}

//if it's the end of record and no data, terminate
if(type == 1 && data_pairs == 0) {
    getch();
    break;
}

printstr_p(PSTR("."));          /*display progress*/
}

return 0;
}

/*
 * initPins - Called at the beginning of BurnIt. Because we might have a GAL, pull-ups
 *            should be active on
 * all outputs. Shuts off all output pins
 * and ensures that high voltage pins are inactive
 */
void initPins(void) {

    //Ensure pullups are on
    MCUCR &= ~(_BV(PUD));

    //Set all pins to inputs except for the high voltage buffers at PB1 and PD6
    //Keep those low to keep from applying the high voltage. Pull others high with weak
    pullups.

```



```

        //Setting the pins in this way should not interfere with the serial port because of
        //internal override signals
DDRA = 0x00;
RSTDDR = _BV(RST12);
    DDRC = 0x00;
    EDITDDR = _BV(EDIT);

    RSTPORT = ~(_BV(RST12));
PORTA = 0xFF;
    PORTC = 0xFF;
    EDITPORT = ~(_BV(EDIT));
}

/* Check for which mode we're in.
 * The DP3T switch applies ground to different pins.
 * BURNIT senses the GND pins of the DP3T switch. ATMEGA pins will have
 * weak pullups to VCC so they are 1 if in that position and 0 otherwise.
 * If switch is "right"
 * we're in PIC mode and the PIC VSS will be grounded. PIC VCC will be
 * applied as well. If switch is "center" VCC will be applied for 2051 and
 * other part of the switch will be grounded.
 * If switch is "left", no power is applied. The GND side pulls down on a pin
 * for sensing purposes.
 */

int getInsertedChip(void) {
    if((GALSENSEPIN & _BV(GALSENSE)) == 0) {
        //GALSENSE was pulled low, so it's a GAL
        return 2;
    }
    if((PICSENSEPIN & _BV(PICSENSE)) == 0) {
        //It's a PIC
        return 1;
    }
    if((SENSE2051PIN & _BV(SENSE2051)) == 0) {
        //It's a 2051
        return 0;
    }
    //We don't know for sure. However, BurnIt and PICBurnIt do not have a
    SENSE2051PIN
    //So that those boards work, we will default to 2051.
    return 0;
}

/*
 * The BurnIT main program loop
 */

int main(void) {

```

```

//      short location;

//For all chips, ensure that drivers are off and that high voltage is
//not active.
initPins();
ioinit();          //set up the serial port

getchr();          //wait for first keypress.
printstr_p(PSTR("\nWelcome to BURNIT!\nPress 'H' for help.));

//Now detect the chip and properly initialize for that chip
currentMode = getInsertedChip();

if(currentMode == 1) {
    printstr_p(PSTR("\nPIC mode enabled"));

    //Set up port pins for PIC mode
    DDRB = PORTB_PIC_DDR;
    DDRA = PORTA_PIC_DDR;

    pic_get_revision();

    if(PICtype == 0) {
        printstr_p(PSTR("\nMaybe the chip isn't inserted or
            "the switch is in the wrong position.));
        printstr_p(PSTR("\nPlease correct and reset BurnIt\n"));
        while(1);
    }
} else if (currentMode == 0) {
    printstr_p(PSTR("\nAtmel 2051 mode enabled"));

    //standard setup, data is output, everything zeroed.
    DATA_DDR = 0xFF; //Data is output
    CONTROL_DDR = 0xFF; //Control Port is all output.
    FEEDBACK_DDR = 0x00; //Feedback is inputs

    CONTROL_PORT = 0x00;
    DATA_PORT = 0x00;

    atmel_pulse_xtal();
} else if (currentMode == 2) {
    printstr_p(PSTR("\nGAL22V10 mode enabled"));
    //Leave the pins alone for now
    //Because we do not know how the GAL is presently programmed, we do not
    //know what are inputs and outputs
    //We cannot activate the drivers until EDIT is asserted
} else {
    printstr_p(PSTR("\nUnrecognized chip, please ensure it is inserted and try
        again"));
    while (1);
}
}

```

```

/* initialize memory to 0x00 if necessary for chip type */
int ix;
if(currentMode == 0) {
    for(ix = 0; ix < CHIPMEMSIZE; ix++) {
        chipMemory[ix] = 0x00;
    }
} else if(currentMode == 1) {
    //We're in PIC mode. Initialize to 0xFF3F repeating
    //That way we know which words need programming.
    for(ix = 0; ix < CHIPMEMSIZE / 2; ix++) {
        chipMemory[2*ix] = 0xFF;
        chipMemory[2*ix+1] = 0x3F;
    }
    for(ix = 0; ix < 128; ix++) {
        picDataMemory[ix] = 0xFF;
    }
    for(ix = 0; ix < 8; ix++) {
        picConfigMemory[2*ix] = 0xFF;
        picConfigMemory[2*ix+1] = 0x3F;
    }
} else if(currentMode == 2) {
    //We're in GAL mode, so no init is necessary
    //The JEDEC file determines whether the default is zeros or ones
}

/*Main Loop*/
while(1) {
    printstr_p(PSTR("\nBURNIT>")); /*display prompt*/
    char rxbyte = toupper(getchr()); /*get character from serial, convert to upper*/
    putchar(rxbyte);
    if(currentMode == 1) {
        switch(rxbyte) {
            case 'C':
                pic_bulk_erase();
                break;
            case 'D':
                load_ihex();
                break;

            case 'A':
                load_ihex(); //Fallthrough intentional
            case 'P':

                pic_bulk_erase();

                if(pic_write_program()<0) break;

                if(pic_write_data()<0) break;

                pic_write_configuration();

                break;
            case 'V':
                pic_view();

```

```

        break;
    case 'U':
        pic_totalErase ();
        break;
    case 'H':
        printstr_p (PSTR("\nBURNIT_Help"));
        printstr_p (PSTR("\nP_____Program_a_PIC."));
        printstr_p (PSTR("\nC_____Clear_a_PIC."));
        printstr_p (PSTR("\nD_____Download_an_Intel_Hex_file_to_BURNIT."));
        printstr_p (PSTR("\nA_____Auto_Download_and_Program_PIC."));
        printstr_p (PSTR("\nH_____Print_this_help_document."));
        printstr_p (PSTR("\nV_____View_the_code_on_the_currently_inserted_PIC."));
        printstr_p (PSTR("\nU_____Unlock_PIC_if_code_protection_enabled."));
        break;
    case '\r': //FALLTHROUGH intentional
    case '\n':
        break;
    default:
        printstr_p (PSTR("\ninvalid_command"));
    }

} else if (currentMode == 2) {
    //Show GAL menu
switch(rxbyte) {
    case 'C':
        //ReadPES();
        ParsePES(0);
        printstr_p (PSTR("\nErasing_GAL..."));
        EraseGAL();
        printstr_p (PSTR("\nGAL_Erased"));

        break;
    case 'D':
        if(readJEDEC()) {
            printstr_p (PSTR("\nJEDEC_Download_Complete"));
        } else {
            printstr_p (PSTR("\nERROR_in_downloading_JEDEC"));
        }
        break;

    case 'A':
        if(readJEDEC()) {
            printstr_p (PSTR("\nJEDEC_Download_Complete"));
        } else {
            printstr_p (PSTR("\nERROR_in_downloading_
                JEDEC...Autoprogramming_will_not_continue"));
            break;
        } //Fallthrough intentional
    case 'P':
        //ReadPES(); //Get the programmer's signature
        ParsePES(0); //Parse it
        printstr_p (PSTR("\nErasing_GAL..."));
        EraseGAL();
        printstr_p (PSTR("\nGAL_Erased"));
        printstr_p (PSTR("\nProgramming_GAL..."));

```

```

//WriteGAL(1);
//EraseGAL();
WriteGAL(0);
printstr_p (PSTR("\nGAL_Programmed...Verifying..."));
//location = ReadGAL(1);
if(ReadGAL(1)) {
    printstr_p (PSTR("\nVerification_Error"));
//char s[10];
//sprintf(s,"%hd",location);
//printstr(s);
} else {
    printstr_p (PSTR("\nVerification_OK.));
}
break;
case 'V':
//ReadPES(); //Get the programmer's signature
ParsePES(0); //Parse it
ReadGAL(0);
PrintFuses();
break;
case 'T':
ReadPES();
ParsePES(1);
break;
case 'H':
printstr_p (PSTR("\nBURNIT_Help"));
printstr_p (PSTR("\nP_____Program_a_GAL.));
printstr_p (PSTR("\nC_____Clear_a_GAL.));
printstr_p (PSTR("\nD_____Download_a_JEDEC_file_to_BURNIT.));
printstr_p (PSTR("\nA_____Auto_Download_and_Program_a_GAL.));
printstr_p (PSTR("\nV_____View_Fusemap_of_inserted_GAL.));
//printstr_p (PSTR("\nT Test view the PES.));
printstr_p (PSTR("\nH_____Print_this_help_document.));
break;
case '\r': //FALLTHROUGH intentional
case '\n':
break;
default:
printstr_p (PSTR("\ninvalid_command"));
}
} else{
switch(rxbyte) {
case 'P':
atmel_program();
break;
case 'V':
atmel_view();
break;
case 'C':
atmel_clear_chip ();
break;
case 'A':

```



```

//Calling GAL PA0 – 5 GALPA0 – 5 to avoid name conflicts

#define GALPA0 PD3
#define GALPA0PORT PORTD

#define GALPA1 PC0
#define GALPA1PORT PORTC

#define GALPA2 PC1
#define GALPA2PORT PORTC

#define GALPA3 PB7
#define GALPA3PORT PORTB

#define GALPA4 PB6
#define GALPA4PORT PORTB

#define GALPA5 PB5
#define GALPA5PORT PORTB

/*
#define GALPA0 PA0
#define GALPA1 PA1
#define GALPA2 PA2
#define GALPA3 PA3
#define GALPA4 PA4
#define GALPA5 PA5

#define GALADPORT PORTA
#define GALADDDR DDRA
*/

//Port A has input on 0, but all others are VIL, so need to be asserted as outputs
#define PORTA_GAL_DDR ~_BV(0)

//Port B has outputs on 5, 6, 7
//PB0 is output, and PB1 is undesirable high, so it stays that way. All others are data pins
#define PORTB_GAL_DDR 0xFF

//0 and 1 are outputs
#define PORTC_GAL_DDR 0x03

//Port D has outputs on 2, 3, 4, 6, 7
#define PORTD_GAL_DDR 0xDC

typedef enum { UNKNOWN,GAL22V10} GALTYPE;

typedef struct
{
    GALTYPE type;
    unsigned char id0,id1;
}

```

```

    char *name;
    int fuses;
    int pins;
    int rows;
    int bits;
    int uesrow;
    int uesfuse;
    int uesbytes;
    int eraserow;
    int eraseallrow;
    int pesrow;
    int pesbytes;
    int cfgrow;
    int *cfg;
    int cfgbits;
} GALINFO;

#define LATTICE 0xA1
#define NATIONAL 0x8F
#define SGSTHOMSON 0x20

#define READGAL 0
#define VERIFYGAL 1
#define READPES 2
#define SCLKTEST 3
#define WRITEGAL 4
#define ERASEGAL 5
#define ERASEALL 6
#define BURNSECURITY 7
#define WRITEPES 8
#define VPPTEST 9

int readJEDEC(void);
void EraseGAL(void);
void ReadPES(void);
void ParsePES(int DisplayPES);
void WriteGAL(int ones);
int ReadGAL(char verify);
void PrintFuses(void);

```

---

## G.1.7 gal.c

---

```

//JEDEC and GAL algorithms
//Taken from GALBlast and ported to AVR without interface

#include <stdio.h>           /* used for string manipulation procedures*/
#include <avr/io.h>          /* direct access to the AVR's IO ports/SFRs */
#include <avr/pgmspace.h>    /* for accessing strings in program memory */
#include <ctype.h>           /* to convert from upper to lowercase */
#include <string.h>

```



```

#include "avrutils.h"
#include "gal.h"

//Programming Pulse time. Learned from PES
static int pulse=0;

//Erase Pulse time. Learned from PES
static int erase=0;

//Programming Voltage from PES.
static int vpp= 0;

//Array mapping PES duration values to milliseconds
static int duration [16]={1,2,5,10,20,30,40,50,60,70,80,90,100,200,0,0};

static unsigned char fusemap[737];

unsigned char pes[12];

static int cfg22V10[]=
{
    5809,5808,
    5811,5810,
    5813,5812,
    5815,5814,
    5817,5816,
    5819,5818,
    5821,5820,
    5823,5822,
    5825,5824,
    5827,5826
};

GALINFO galinfo[]=
{
    {UNKNOWN, 0x00,0x00,"unknown", 0, 0, 0, 0, 0,
    0,0, 0, 0, 0, 8, 0,0,0},
    {GAL22V10, 0x48,0x49,"GAL22V10", 5892,24,44,
    132,44,5828,8,61,60,58,10,16, cfg22V10,sizeof(cfg22V10)/sizeof(int)},
};

/*
 * SetAddr(uint8_t addr)
 * Sets the address lines of the GAL22V10 to the specified value
 */
static void SetAddr(uint8_t addr)
{
    //Unfortunately, these address lines are scattered across different ports, so each bit
    has to be
    //done individually.
    if(addr & _BV(0)) {
        GALPA0PORT |= _BV(GALPA0);
    } else {
        GALPA0PORT &= ~_BV(GALPA0);
    }
}

```

```

    }
    if(addr & _BV(1)) {
        GALPA1PORT |= _BV(GALPA1);
    } else {
        GALPA1PORT &= ~_BV(GALPA1);
    }
    if(addr & _BV(2)) {
        GALPA2PORT |= _BV(GALPA2);
    } else {
        GALPA2PORT &= ~_BV(GALPA2);
    }
    if(addr & _BV(3)) {
        GALPA3PORT |= _BV(GALPA3);
    } else {
        GALPA3PORT &= ~_BV(GALPA5);
    }
    if(addr & _BV(4)) {
        GALPA4PORT |= _BV(GALPA4);
    } else {
        GALPA4PORT &= ~_BV(GALPA4);
    }
    if(addr & _BV(5)) {
        GALPA5PORT |= _BV(GALPA5);
    } else {
        GALPA5PORT &= ~_BV(GALPA5);
    }
}

/*
 * SetEDIT()
 * sets the value of the EDIT line on the GAL
 */
static void SetEDIT(int setEdit) {
    if(setEdit) {
        EDITPORT |= _BV(EDIT);
    } else {
        EDITPORT &= ~(_BV(EDIT));
    }
}

/*
 * Sets the drivers to be either on or off
 */
static void SetDrivers(int setDrivers)
{
    if(setDrivers) {
        //Drive everything low except for EDIT itself and STB on PortB
        PORTA = 0x00;
        PORTB = _BV(STB);
        PORTC = 0x00;
        EDITPORT &= _BV(EDIT); //Leave EDIT alone, but put everything else low
        DDRA = PORTA_GAL_DDR;
        DDRB = PORTB_GAL_DDR;
        DDRC = PORTC_GAL_DDR;
        DDRD = PORTD_GAL_DDR;
    }
}

```

```

} else {
    //Convert all back to inputs except the high voltage bits. They keep driving
    DDRA = 0x00;
    DDRB = _BV(PB1); //This is to keep the 2051 high voltage driver from being
        silly
    DDRC = 0x00;
    EDITDDR = _BV(EDIT);

    //Now enable pullups except on ultra high voltages
    PORTA = 0xFF;
    PORTB = ~(_BV(PB1));
    PORTC = 0xFF;
    //Enable all pullups except on EDIT. Just leave it alone
    PORTD |= ~(_BV(EDIT)) ;
}
}

static void SetPV(int setPV) {
    if(setPV) {
        GALCTRLPORT |= _BV(PV);
    } else {
        GALCTRLPORT &= ~(_BV(PV));
    }
}

static void SetSCLK(int setSCLK) {
    if(setSCLK) {
        GALCTRLPORT |= _BV(SCLK);
    } else {
        GALCTRLPORT &= ~(_BV(SCLK));
    }
}

//Enters programming mode. This will take care of initializing output pins
//Mode determines whether we are doing a READ or some kind of write. The programming
    voltage is different
static char TurnOn(int mode)
{
    char writeorerase;

    //First decide if this turn on will involve writing.
    if(mode==WRITEGAL||mode==ERASEGAL||mode==ERASEALL||
        mode==BURNSECURITY||mode==WRITEPES||mode==VPPTEST)
    {
        writeorerase=1;
    } else {
        writeorerase=0;
    }
    //Pins should not be driving, but make sure
    SetDrivers(0);

    _delay_us(20);

    //Turn on programming/reading voltage only

```

```

SetEDIT(1);

_delay_ms(100);

_delay_ms(10);

//Now all pins in programming mode, so drivers can come on
//This puts all VIL LOW and all GAL INPUTS to LOW except STB
SetDrivers(1);

_delay_ms(20);

if(writeorerase)
{
    SetPV(1);
    _delay_ms(10);
} else {
    SetPV(0);
    _delay_ms(10);
}
return 1;
}

//Turns off the programming mode
static void TurnOff(void)
{
    _delay_us(200);

    //Disable the drivers.
    SetDrivers(0);

    _delay_ms(10);

    //Turn off edit now
    SetEDIT(0);

    _delay_us(20);
}

void SendBit(int bit)
{
    if(bit) {
        GALCTRLPORT |= _BV(SDIN);
    } else {
        GALCTRLPORT &= ~(_BV(SDIN));
    }

    _delay_us(2);
    SetSCLK(1);
    _delay_us(2);
    SetSCLK(0);
    _delay_us(2);
}

```

```

void SendBits(int n,int bit)
{
    while(n-->0) SendBit(bit);
}

void Strobe(int msec)
{
    //_delay_ms(3);
    _delay_ms(1);
    GALCTRLPORT &= ~(_BV(STB));
    _delay_ms(msec);
    GALCTRLPORT |= _BV(STB);
    //_delay_ms(3);
    _delay_ms(1);
}

void SendAddress(int n,int row)
{
    while(n-->0)
    {
        SendBit(row&1);
        row>>=1;
    }
}

//Selects the row we want to talk to
void StrobeRow(int row)
{
    SetAddr(0);
    SendBits(132,0); //Sends 132 zeroes (because 132 bits per row in GAL22V10)
    SendAddress(6,row); //Now send address of the row we want
                        //The 6 is for how many bits are in the row

    Strobe(1); //Strobe the pin
}

unsigned char ReceiveBit(void)
{
    unsigned char bit;

    bit= (SDOUTPIN & _BV(SDOUT) ? 1 : 0);
    _delay_us(2);
    SetSCLK(1);
    _delay_us(2);
    SetSCLK(0);
    _delay_us(2);
    return bit;
}

//Parses the PES information retrieved from the GAL
//This contains information on programming voltage, programming time, and other things
void ParsePES(int displayPES)
{
    int algo=pes[1]&0x0F;
}

```

```

if(algo==5) { //Algorithm is specified
    erase=(25<<(((pes[4]>>2)&7))/2;
    pulse=duration[(((unsigned)pes[5]<<8)|pes[4]>>5)&15];
    vpp=2*((pes[5]>>1)&31)+20;
} else {
    erase = (pes[3]==NATIONAL?50:100);
    switch(algo) {
    case 0:
        vpp=66; // 16.5V
        pulse=10;
        break;
    case 1:
        vpp=63; // 15.75V
        pulse=100;
        break;
    case 2:
        vpp=pes[3]==NATIONAL?60:58; // 15/14.5V
        pulse=40;
        break;
    case 3:
        vpp=56; // 14V
        pulse=100;
        break;
    }
}

//I am hardwiring this to be a long time so that the chip will work even if the PES is
    toasted.
pulse = 100;
erase = 200;

//Note: I do not want to compile in software floating point arithmetic
//for this. Since I am just dividing by two, I can do it with bit shifts
if(displayPES) {
    char s[100];
    int realvpp = vpp / 4;
    char isQuarter = vpp & 0x01; //See if dividing by two truncates
    char isHalf = vpp & 0x02 ? 1 : 0;
    char fraction = isQuarter * 25 + isHalf * 50;
    sprintf(s, "\nPES_says_Vpp=_%d.%d,Pulse_"
        "time=_%d_ms", realvpp, fraction, pulse);
    printstr(s);
    sprintf(s, "\nPES:_%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d", pes[0], pes[1],
        pes[2], pes[3], pes[4], pes[5], pes[6], pes[7], pes[8],
        pes[9], pes[10], pes[11]);
    printstr(s);
}
return;
}

//Reads the Programmer's Electronic Signature to make sure
//this is a GAL22V10 and we know the programming
//algorithm
void ReadPES()

```

```

{
    int bitmask,byte;

    TurnOn(READPES);

    StrobeRow(58); //Select row 58, the PES on GAL22V10
    for(byte=0;byte<10;byte++) { //The PES is 10 bytes long. Get them
        pes[byte]=0;
        for(bitmask=0x1;bitmask<=0x80;bitmask<<=1) { //Get each bit and
                                                    //assemble into PES
            if(ReceiveBit()) pes[byte]|=bitmask;
        }
    }

    TurnOff();
}

int ReadGAL(char verify)
{
    int row,bit;
    short fuseindex;
    int fusebit;

    char s[7];
    int errorfound = 0;

    char tempfusemap = 0;

    TurnOn(READGAL);

    for(row=0;row<galinfo[GAL22V10].rows;row++) {
        StrobeRow(row);
        for(bit=0;bit<galinfo[GAL22V10].bits;bit++){
            fuseindex = (galinfo[GAL22V10].rows*bit+row) / 8;
            fusebit = (galinfo[GAL22V10].rows*bit+row) % 8;
            if(verify) {
                tempfusemap = ReceiveBit() ?
                tempfusemap | _BV(fusebit) :
                tempfusemap & ~(_BV(fusebit));
                if((tempfusemap & _BV(fusebit)) !=
                    (fusemap[fuseindex] & _BV(fusebit))){
                    sprintf(s, "\n%hd",fuseindex*8 + fusebit);
                    printstr(s);
                    errorfound = 1;
                }
            } else {
                fusemap[fuseindex] = ReceiveBit() ?
                fusemap[fuseindex] | _BV(fusebit) :
                fusemap[fuseindex] & ~(_BV(fusebit));
            }
        }
    }
}

// UES

```

```

StrobeRow(galinfo[GAL22V10].uesrow);
for(bit=0;bit<galinfo[GAL22V10].uesbytes*8;bit++) {
    fuseindex = (galinfo[GAL22V10].uesfuse+bit) / 8;
    fusebit = (galinfo[GAL22V10].uesfuse+bit) % 8;
    if(verify) {
        tempfusemap = ReceiveBit() ?
        tempfusemap | _BV(fusebit) :
        tempfusemap & ~(_BV(fusebit));
        if(((tempfusemap & _BV(fusebit)) !=
            (fusemap[fuseindex] & _BV(fusebit)))){
            sprintf(s, "\n%hd", fuseindex*8 + fusebit);
            printstr(s);
            errorfound = 1;
        }
    } else {
        fusemap[fuseindex] = ReceiveBit() ?
        fusemap[fuseindex] | _BV(fusebit) :
        fusemap[fuseindex] & ~(_BV(fusebit));
    }
}

// CFG
SetAddr(galinfo[GAL22V10].cfgrow);
Strobe(2);
for(bit=0;bit<galinfo[GAL22V10].cfgbits;bit++) {
    fuseindex = (galinfo[GAL22V10].cfg[bit]) / 8;
    fusebit = (galinfo[GAL22V10].cfg[bit]) % 8;
    if(verify) {
        tempfusemap = ReceiveBit() ?
        tempfusemap | _BV(fusebit) :
        tempfusemap & ~(_BV(fusebit));
        if(((tempfusemap & _BV(fusebit)) !=
            (fusemap[fuseindex] & _BV(fusebit)))){
            sprintf(s, "\n%hd", fuseindex*8 + fusebit);
            printstr(s);
            errorfound = 1;
        }
    } else {
        fusemap[fuseindex] = ReceiveBit() ?
        fusemap[fuseindex] | _BV(fusebit) :
        fusemap[fuseindex] & ~(_BV(fusebit));
    }
}

TurnOff();
return errorfound;
}

void PrintFuses(void)
{
    int i;
    char s[8];
    for(i = 0; i < sizeof(fusemap); i++) {
        if(i % 32 == 0){

```



```

        printstr_p(PSTR("\n"));
    }
    sprintf(s,"%X_",fusemap[i]);
    printstr(s);
}
}

void EraseGAL(void)
{
    if(TurnOn(ERASEGAL)) {
        SetAddr(61);
        Strobe(erase);
        TurnOff();
    }
}

//Write the current fusemap to the GAL in the socket using data from PES
void WriteGAL(int ones)
{
    int row,bit;
    short fuseindex;
    int fusebit;

    if(TurnOn(WRITEGAL)) {
        SetAddr(0);
        for(row=0;row<galinfo[GAL22V10].rows;row++) {
            for(bit=0;bit<galinfo[GAL22V10].bits;bit++) {
                fuseindex = (galinfo[GAL22V10].rows*bit + row) / 8;
                fusebit = (galinfo[GAL22V10].rows*bit + row) % 8;
                if(ones) {
                    SendBit(1);
                } else {
                    SendBit(fusemap[fuseindex] & _BV(fusebit) ? 1 : 0);
                }
            }
            SendAddress(6,row);
            GALCTRLPORT &= ~(_BV(SDIN));
            Strobe(pulse);
        }
        // UES
        for(bit=0;bit<galinfo[GAL22V10].uesbytes*8;bit++) {
            fuseindex = (galinfo[GAL22V10].uesfuse+bit) / 8;
            fusebit = (galinfo[GAL22V10].uesfuse+bit) % 8;
            if(ones) {
                SendBit(1);
            } else {
                SendBit(fusemap[fuseindex] & _BV(fusebit) ? 1 : 0);
            }
        }
        if(galinfo[GAL22V10].uesbytes*8<galinfo[GAL22V10].bits) {
            SendBits(galinfo[GAL22V10].bits-galinfo[GAL22V10].uesbytes*8,0);
        }
        SendAddress(6,galinfo[GAL22V10].uesrow);
        GALCTRLPORT &= ~(_BV(SDIN));
    }
}

```

```

    Strobe(pulse);
    // CFG
    SetAddr(galinfo[GAL22V10].cfgrow);
    for(bit=0;bit<galinfo[GAL22V10].cfgbits;bit++) {
        fuseindex = (galinfo[GAL22V10].cfg[bit]) / 8;
        fusebit = (galinfo[GAL22V10].cfg[bit]) % 8;
        if(ones) {
            SendBit(1);
        } else {
            SendBit(fusemap[fuseindex] & _BV(fusebit) ? 1 : 0);
        }
    }
    GALCTRLPORT &= ~(_BV(SDIN));
    Strobe(pulse);
}
TurnOff();
}

```

*//Computes the checksum of the fusemap  
//argument is the size of the fusemap. For GAL22V10, should be  
//5892, but this is checked before CheckSum is called*

```

static unsigned short CheckSum(int n)
{
    unsigned short c,e;
    long a;
    int i;

    unsigned short fusemapindex=0;
    unsigned char bitindex = 0;

    c=e=0;
    a=0;
    for(i=0;i<n;i++)
    {
        e++;
        if(e==9)
        {
            e=1;
            a+=c;
            c=0;
        }
        c>>=1;
        fusemapindex = i / 8;
        bitindex = i % 8;
        if(fusemap[fusemapindex] & _BV(bitindex)) c+=0x80;
    }
    return (unsigned short)((c>>(8-e))+a);
}

```

*//Parses a JEDEC file being transmitted over the serial port  
//Uses a messy state machine to do it. Returns 1 if successful and 0 if an error*

```

int readJEDEC()

```

```

{
//int i , n,type ,checksumpos,address,pins, lastfuse , state ;

// 0=outside JEDEC, 1=skipping comment or unknown, 2=read command
// Other states were undocumented.
//Looking at the JEDEC standard, a transmission starts with hex 0x02
//and ends with hex 0x03
//Every useful line starts with a '*' character, so the
//state machine goes to 2

short address = 0;

//Used because the fusemap must be packed for space reasons.
short fuseaddress = 0;
char bitlocation = 0;

//This is the 16-bit sum of the entire transmission
unsigned short xmitchecksum = 0;

char state=0;
char security=0;
short checksum=0;
char pins=0;
short lastfuse=0;

unsigned char receivedChar;

printstr_p(PSTR("\n>"));

//Wait until we get START byte
while(getchr() != 0x02){
;
}
xmitchecksum = 0x02;
//We got it. Now process the file
while((receivedChar = getchr())) {
xmitchecksum = (xmitchecksum + receivedChar) & 0xFFFF;
//First check for end
if (receivedChar == 0x03) {
//We're done
break;
} else if (receivedChar == '*'){
printstr_p(PSTR("."));
state=2;
} else switch(state) {
case 2: //This char defines what the line is
if (!isspace(receivedChar)) {
switch(receivedChar) {
case 'L': //Fuse List
address=0;
state=3;
break;
case 'F': //Default Fuse State
state=5;

```

```

        break;
    case 'G': //Security Fuse
        state=13;
        break;
    case 'Q': //Specifies features of GAL
        state=7;
        break;
    case 'C': //Fuse Checksum
        state=14;
        break;
    default:
        state=1;
    }
}
break;
case 3://Getting first digit of Fuse List
    if(! isdigit (receivedChar)) return 0;
    address=receivedChar-'0';
    state=4;
    break;
case 4: //Getting remaining digits of Fuse List until Space received
    if(isspace(receivedChar)){
        state=6;
    } else if( isdigit (receivedChar)) {
        address=10*address+(receivedChar-'0');
    } else {
        return 0;
    }
    break;
case 5: //Default Fuse state command "format" the fusemap
    if(isspace(receivedChar)) break; // ignored
    if(receivedChar=='0' || receivedChar=='1'){
        memset(fusemap,receivedChar-'0',sizeof(fusemap));
    } else {
        return 0;
    }
    state=1;
    break;
case 6: //Reading in fuses from Fuse List entry until next '*'
    if(isspace(receivedChar)) break; // ignore spaces
    if(receivedChar=='0' || receivedChar=='1') {

        //Divide by 8 to get index into packed fusemap array
        fuseaddress = address / 8;
        //Use modulo to get which bit in that cell to read
        bitlocation = address % 8;

        //Finally, write the bit into that cell
        if(receivedChar == '0') {
            fusemap[fuseaddress] &= ~(_BV(bitlocation));
        } else {
            fusemap[fuseaddress] |= _BV(bitlocation);
        }
        address++;
    }

```

```

    } else {
        return 0;
    }
    break;
case 7: //Get configuration information. Currently ignored
    if(isspace(receivedChar)) break; // ignored
    if(receivedChar=='P') {
        pins=0;
        state=8;
    } else if(receivedChar=='F') {
        lastfuse=0;
        state=9;
    } else state=2;
    break;
case 8: //Setting expected number of pins
    if(isspace(receivedChar)) break; // ignored
    if(!isdigit(receivedChar)) return 0;
    pins=receivedChar-'0';
    state=10;
    break;
case 9: //Setting expected number of fuses
    if(isspace(receivedChar)) break; // ignored
    if(!isdigit(receivedChar)) return 0;
    lastfuse=receivedChar-'0';
    state=11;
    break;
case 10: //Getting remaining digits of number of pins
    if( isdigit(receivedChar)) {
        pins=10*pins+(receivedChar-'0');
    } else if(isspace(receivedChar)) {
        state=12;
    } else return 0;
    break;
case 11: //Getting remaining digits of number of fuses
    if( isdigit(receivedChar)) {
        lastfuse=10*lastfuse+(receivedChar-'0');
    } else if(isspace(receivedChar)) {
        state=12;
    } else return 0;
    break;
case 12: //Ensuring that there is whitespace after setting config?
    if(!isspace(receivedChar)) return 0;
    break;
case 13: //Security Fuse setting
    if(isspace(receivedChar)) break; // ignored
    if(receivedChar=='0'||receivedChar=='1') {
        security=receivedChar-'0';
    } else {
        return 0;
    }
    state=1;
    break;
case 14: //Get the checksum first digit
    if(isspace(receivedChar)) break; // ignored

```

```

        if( isdigit (receivedChar)) {
            checksum=receivedChar-'0';
        } else if(toupper(receivedChar)>='A'&&
            toupper(receivedChar)<='F') {
            checksum=toupper(receivedChar)-'A'+10;
        } else return 0;
        state=15;
        break;
    case 15: //Get the remaining digits until a space
        if( isdigit (receivedChar)) {
            checksum=16*checksum+receivedChar-'0';
        } else if(toupper(receivedChar)>='A'&&
            toupper(receivedChar)<='F') {
            checksum=16*checksum+toupper(receivedChar)-'A'+10;
        } else if(isspace(receivedChar)) {
            state=2;
        } else return 0;
        break;
    }
} //Ends for loop

```

```

//Now get the transmit checksum
unsigned short tempxmitcheck = 0;
tempxmitcheck = ascii_to_bin(getchr());
tempxmitcheck = (tempxmitcheck << 4) | ascii_to_bin(getchr());
tempxmitcheck = (tempxmitcheck << 4) | ascii_to_bin(getchr());
tempxmitcheck = (tempxmitcheck << 4) | ascii_to_bin(getchr());

```

```

/*
 * Note we are not checking the transmission checksum.
 * It is broken if the JED file
 * moves between Unix and Windows or if it is mangled in Hyperterminal because it
 * includes
 * line feeds and carriage returns as meaningful elements.
 * If the file is moved as a text file , line endings get
 * mangled and invalidate the checksum.
 * Besides, there is already a fuse checksum, so this feels superfluous.
 */

```

```

//Make sure that a GAL22V10 was used
if( lastfuse != 5892) {
    printstr_p(PSTR("\nThis JEDEC has the wrong number of fuses or"
        "_no_fuse_declaration...Is it for a GAL22V10?"));
    return 0;
}
if(pins != 24) {
    printstr_p(PSTR("\nThis JEDEC has the wrong number of pins or no"
        "_pin_declaration...Is it for a GAL22V10?"));
    return 0;
}

```

```

//Now compute fuse checksum using the known fuse size
//I will demand that a checksum be present
if(checksum == 0) {

```

```

    printstr_p (PSTR("\nNo_Fuse_Checksum_present_in_JEDEC_file_"
                    "Check_compiler_settings"));
    return 0;
}
if(checksum != CheckSum(lastfuse)) {
    printstr_p (PSTR("\nChecksum_Wrong"));
    return 0;
}
if (security == 1) {
    printstr_p (PSTR("\nYou_have_chosen_to_program_the_Security_Fuse_in_this_
                    JEDEC."));
    printstr_p (PSTR("\nBecause_this_will_prevent_reading_out_the_fuse_map,this_
                    GAL"));
    printstr_p (PSTR("\nwill_give_a_verification_error_after_programming."));
    printstr_p (PSTR("\nSimply_clear_or_reprogram_the_GAL_to_clear_the_security_
                    fuse"));
}
return 1;
}

```

---

## G.1.8 pic.h

---

```

#define PIC16F628 0x3E
#define PIC16F628A 0x83

#define PIC16F627 0x3D
#define PIC16F627A 0x82

#define PIC16F648A 0x88

#define LOAD_CONFIG_COMMAND 0x00
#define LOAD_PROGRAM_COMMAND 0x02
#define LOAD_DATA_COMMAND 0x03
#define INCREMENT_ADDRESS 0x06
#define READ_PROGRAM_COMMAND 0x04
#define READ_DATA_COMMAND 0x05

//The following command is for no A
//A version calls this Program Only
#define BEGIN_ERASE_PROGRAM 0x08
#define A_PROGRAM_ONLY 0x08

//Just for old chip
#define BEGIN_PROGRAM_ONLY 0x18

#define BULK_ERASE_PROGRAM 0x09
#define BULK_ERASE_DATA 0x0B

//Just for old chip
//Used to hose code protection bits
#define BULK_ERASE_SETUP_ONE 0x01
#define BULK_ERASE_SETUP_TWO 0x07

```

```

//This is the definition for the PIC burning pins
#define VSS    PB6
#define VDD    PA3
#define VPP    PB7
#define PGM    PB2
#define CLOCK  PA1
#define DATA  PA2

#define PICSENSE PB6
#define PICSENSEPIN PINB

//PB2,PB7 are outputs. PB6 input
//PA1, output, PA2 is both (calling an input in default
#define PORTB_PIC_DDR 0x84
#define PORTA_PIC_DDR 0x02
#define VSSDDR  DDB6
#define VSSPIN  PINB6
#define CLOCK_PORT PORTA
#define DATA_PORT PORTA
#define PGM_PORT PORTB
#define VPP_PORT PORTB
#define VSS_PORT PORTB
#define VSS_DDR DDRB
#define VSS_PIN  PINB
#define DATADDR DDA2
#define DATA_DDR DDRA
#define DATA_PIN PINA
#define DATAIN PINA2

//sets type of PIC
extern uint8_t PICtype;

//Data memory for the PIC to be burned
extern uint8_t picDataMemory[128];

//Configuration memory for the PIC to be burned
extern uint8_t picConfigMemory[16];

//void pic_enter_programming(void);
//void pic_exit_programming(void);
//void pic_pulse_clock(void);
//void pic_send_command(uint8_t comm);
//uint16_t pic_receive_data(void);
//void pic_send_data(uint16_t comm);
void pic_bulk_erase(void);
int pic_write_program(void);
int pic_write_data(void);
int pic_write_configuration(void);
uint8_t pic_get_revision(void);
void pic_view(void);
void pic_total_erase(void);

```

---



## G.1.9 pic.c

---

```
#include <stdio.h>      /* used for string manipulation procedures */
#include <avr/io.h>     /* direct access to the AVR's IO ports/SFRs */
#include <avr/pgmspace.h> /* for accessing strings in program memory */

#include <ctype.h>     /* to convert characters from upper to lowercase */

#include "avrutils.h"
#include "pic.h"       /* Constants for PIC programming */

//sets type of PIC
uint8_t PICtype = 0;

//Data memory for the PIC to be burned
uint8_t picDataMemory[128];

//Configuration memory for the PIC to be burned
uint8_t picConfigMemory[16];

/*
 * The following are low level routines to help portability to other platforms
 */
static void pic_set_clock(int setClock) {
    if(setClock) {
        CLOCK_PORT |= _BV(CLOCK);
    } else {
        CLOCK_PORT &= ~(_BV(CLOCK));
    }
}

static void pic_set_pgm(int setPGM) {
    if(setPGM) {
        PGM_PORT |= _BV(PGM);
    } else {
        PGM_PORT &= ~(_BV(PGM));
    }
}

static void pic_set_vpp(int setVPP) {
    if(setVPP) {
        VPP_PORT |= _BV(VPP);
    } else {
        VPP_PORT &= ~(_BV(VPP));
    }
}

/*
 * pic_enter_programming
 * Enter PIC programming mode
 *
 */
static void pic_enter_programming(void) {
```

```

    //Ensure everything is low
    pic_set_clock (0);
    pic_set_pgm(0);
    pic_set_vpp(0);

    //Wait delay between power on and PGM rise
    _delay_us(5);

    //Now raise lines
    pic_set_pgm(1);
    _delay_us(5);

    pic_set_vpp(1);
    _delay_us(5);

    //Now we are ready for command
}

/*
 * pic_exit_programming
 * Exit PIC programming mode
 */
static void pic_exit_programming(void) {
    pic_set_vpp(0);
    pic_set_pgm(0);
}

/*
 * pic_pulse_clock
 * Pulses the clock to the PIC
 */
static void pic_pulse_clock(void) {
    //Clock needs setup and hold of 100ns each. Both should be met
    //without explicit delays.

    _delay_us(2);
    pic_set_clock (0);

    _delay_us(2);

    pic_set_clock (1);

    _delay_us(2);
}

/*
 * pic_send_command
 * Sends the specified 6 bit command to the PIC
 * Assumes already in programming mode
 */
static void pic_send_command(uint8_t comm) {

    //Make DATA an output

```

```

DATA_DDR |= _BV(DATADDR);

//Raise the clock line
pic_set_clock (1);

DATA_PORT = comm & 0x1 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x2 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x4 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x8 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x10 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x20 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));

//Don't pulse last clock because it needs to remain low for a while.
pic_set_clock (0);

//Shut off the data port before the PIC makes an attempt to drive it.
//Make an input.
DATA_DDR &= ~(_BV(DATADDR));

//Delay after command sent
_delay_us(1);
}

/*
 * pic_receive_data
 * Receives 16 bits of data and returns 14 bits actually sent.
 */
static uint16_t pic_receive_data (void) {

    uint16_t data = 0;
    //Make DATA an input
    DATA_DDR &= ~(_BV(DATADDR));

    //Raise the clock line
    pic_set_clock (1);

    //Pulse clock to receive leading zero
    pic_pulse_clock ();

    //Now get the data
    data |= ((DATA_PIN & _BV(DATAIN)) ? 0x1 : 0);
    pic_pulse_clock ();
    data |= ((DATA_PIN & _BV(DATAIN)) ? 0x2 : 0);

```

```

pic_pulse_clock ();
data |= ((DATA_PIN & _BV(DATAIN)) ? 0x4 : 0);
pic_pulse_clock ();
data |= ((DATA_PIN & _BV(DATAIN)) ? 0x8 : 0);
pic_pulse_clock ();
data |= ((DATA_PIN & _BV(DATAIN)) ? 0x10 : 0);
pic_pulse_clock ();
data |= ((DATA_PIN & _BV(DATAIN)) ? 0x20 : 0);
pic_pulse_clock ();
data |= ((DATA_PIN & _BV(DATAIN)) ? 0x40 : 0);
pic_pulse_clock ();
data |= ((DATA_PIN & _BV(DATAIN)) ? 0x80 : 0);
pic_pulse_clock ();
data |= ((DATA_PIN & _BV(DATAIN)) ? 0x100 : 0);
pic_pulse_clock ();
data |= ((DATA_PIN & _BV(DATAIN)) ? 0x200 : 0);
pic_pulse_clock ();
data |= ((DATA_PIN & _BV(DATAIN)) ? 0x400 : 0);
pic_pulse_clock ();
data |= ((DATA_PIN & _BV(DATAIN)) ? 0x800 : 0);
pic_pulse_clock ();
data |= ((DATA_PIN & _BV(DATAIN)) ? 0x1000 : 0);
pic_pulse_clock ();
data |= ((DATA_PIN & _BV(DATAIN)) ? 0x2000 : 0);
pic_pulse_clock ();

//Now have the data. Just lower clock one more time to clear lagging zero
pic_set_clock (0);

_delay_us(1);

return data;
}

/*
 * pic_send_data
 * Sends the specified 14 bit command (will be 0 padded to 16) to the PIC
 * Assumes already in programming mode and command already sent
 */
static void pic_send_data(uint16_t comm) {
    //Make DATA an output
    DATA_DDR |= _BV(DATADDR);

    //Raise the clock line
    pic_set_clock (1);

    //Send 0

    DATA_PORT = DATA_PORT & ~(_BV(DATA));
    pic_pulse_clock ();

    DATA_PORT = comm & 0x1 ?
        DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
    pic_pulse_clock ();
}

```

```

DATA_PORT = comm & 0x2 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x4 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x8 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x10 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x20 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x40 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x80 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x100 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x200 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x400 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x800 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x1000 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();
DATA_PORT = comm & 0x2000 ?
    DATA_PORT | _BV(DATA) : DATA_PORT & ~(_BV(DATA));
pic_pulse_clock ();

//Send trailing zero
DATA_PORT = DATA_PORT & ~(_BV(DATA));
//Don't pulse last clock because it needs to remain low for a while.
pic_set_clock (0);

//Shut off the data port
//Make an input.
DATA_DDR &= ~(_BV(DATADDR));

//Delay after data sent
_delay_us(1);
}

/*

```

```

* pic_bulk_erase
* Erase the PIC's memory
*/
void pic_bulk_erase(void) {
    pic_enter_programming();
    printstr_p(PSTR("\nErasing PIC program memory..."));

    //First load data for program memory with data set to all ones.
    pic_send_command(LOAD_PROGRAM_COMMAND);

    pic_send_data(0x3FFF);

    pic_send_command(BULK_ERASE_PROGRAM);

    if((PICtype == PIC16F628) || (PICtype == PIC16F627)) {
        pic_send_command(BEGIN_PROGRAM_ONLY);
    }

    _delay_ms(6);

    printstr_p(PSTR("\nErasing PIC data memory..."));

    pic_send_command(LOAD_PROGRAM_COMMAND);

    pic_send_data(0x3FFF);

    pic_send_command(BULK_ERASE_DATA);

    if((PICtype == PIC16F628) || (PICtype == PIC16F627) ) {
        pic_send_command(BEGIN_PROGRAM_ONLY);
    }

    _delay_ms(6);
    pic_exit_programming();
}

/*
* pic_write_program
* Program PIC memory
*/
int pic_write_program(void) {
    uint16_t i = 0;
    uint16_t tempdata;
    uint16_t sendingdata;

    pic_enter_programming();
    printstr_p(PSTR("\nProgramming PIC Program memory..."));

    for( i = 0; i < CHIPMEMSIZE / 2; i++) {
        if(i % 32 == 0) {
            printstr_p(PSTR("."));
        }

        sendingdata = (((chipMemory[2*i+1] << 8) & 0x3F00) |

```

```

        chipMemory[2*i] & 0x3FFF;

//Don't bother sending data if it's blank.
if((sendingdata == 0x3FFF) {
    pic_send_command(INCREMENT_ADDRESS);
    continue;
}

//Load data for program memory
pic_send_command(LOAD_PROGRAM_COMMAND);
//Send the data
pic_send_data((((chipMemory[2*i+1] << 8) & 0x3F00) |
               chipMemory[2*i] & 0x3FFF);
//Begin programming only cycle
if((PICtype == PIC16F628) || (PICtype == PIC16F627)) {
    pic_send_command(BEGIN_PROGRAM_ONLY);
    _delay_ms(8);
} else {
    pic_send_command(A_PROGRAM_ONLY);
    _delay_ms(4);
}

//Now verify that it was written
//Read from Program
pic_send_command(READ_PROGRAM_COMMAND);
tempdata = pic_receive_data();
if((tempdata & 0x3FFF) != (((chipMemory[2*i+1] << 8) & 0x3F00) |
                           chipMemory[2*i] & 0x3FFF)) {
    printstr_p(PSTR("\nVerification_error_during_programming."));
    printstr_p(PSTR("\nCode_protection_may_be_enabled."));
    pic_exit_programming();
    return -1;
}
//Increment address
pic_send_command(INCREMENT_ADDRESS);
}
pic_exit_programming();
return 0;
}

/*
 * pic_write_data
 * Program PIC memory
 */
int pic_write_data(void) {
    uint8_t i = 0;
    uint16_t tempdata;
    pic_enter_programming();
    printstr_p(PSTR("\nProgramming_PIC_Data_memory..."));

    for( i = 0; i < 128; i++) {
        if(picDataMemory[i] == 0xFF) {
            pic_send_command(INCREMENT_ADDRESS);
            continue;

```

```

    }

    //Load data for program memory
    pic_send_command(LOAD_DATA_COMMAND);
    //Send the data
    pic_send_data(picDataMemory[i] & 0x00FF);
    //Begin programming only cycle
    if((PICtype == PIC16F628) || (PICtype == PIC16F627)) {
        pic_send_command(BEGIN_PROGRAM_ONLY);
        _delay_ms(8);
    } else {
        pic_send_command(A_PROGRAM_ONLY);
        _delay_ms(4);
    }

    //Now verify that it was written
    //Read from Data
    pic_send_command(READ_DATA_COMMAND);
    tempdata = pic_receive_data();
    if((tempdata & 0x00FF) != (picDataMemory[i] & 0x00FF)) {
        printstr_p (PSTR("\nVerification_error_during_Data_programming."));
        pic_exit_programming();
        return -1;
    }
    //Increment address
    pic_send_command(INCREMENT_ADDRESS);
}
pic_exit_programming();
return 0;
}

/*
 * pic_write_configuration
 * Program PIC config memory
 */
int pic_write_configuration (void) {
    uint8_t i = 0;
    uint16_t tempdata;
    uint16_t sendingdata;
    printstr_p (PSTR("\nProgramming_PIC_Configuration_Word..."));
    pic_enter_programming();
    pic_send_command(LOAD_CONFIG_COMMAND);
    pic_send_data(0x3FFF);

    for( i = 0; i < 8; i++) {
        if(i > 3 && i < 7) {
            pic_send_command(INCREMENT_ADDRESS);
            continue;
        }
        sendingdata = (((picConfigMemory[2*i+1] << 8) & 0x3F00) |
            picConfigMemory[2*i]) & 0x3FFF;

        //Don't bother sending data if it's blank.
        if((sendingdata == 0x3FFF)) {

```



```

        pic_send_command(INCREMENT_ADDRESS);
        continue;
    }

    //Load data for program memory
    pic_send_command(LOAD_PROGRAM_COMMAND);
    //Send the data
    pic_send_data((((picConfigMemory[2*i+1] << 8) & 0x3F00) |
        picConfigMemory[2*i]) & 0x3FFF);
    //Begin programming only cycle
    if((PICtype == PIC16F628) || (PICtype == PIC16F627)) {
        //Using Erase/Program because the bulk erase apparently doesn't
        //clear this memory space.
        pic_send_command(BEGIN_ERASE_PROGRAM);
        _delay_ms(13);
    } else {
        pic_send_command(A_PROGRAM_ONLY);
        _delay_ms(4);
    }

    //Now verify that it was written
    //Read from Program
    pic_send_command(READ_PROGRAM_COMMAND);
    tempdata = pic_receive_data();
    if((tempdata & 0x3FFF) != (((picConfigMemory[2*i+1] << 8) & 0x3F00) |
        picConfigMemory[2*i]) & 0x3FFF) {
        printstr_p(PSTR("\nVerification_error_during_Configuration"
            "_programming."));
        printstr_p(PSTR("\n_Sent:_"));
        printhex(picConfigMemory[2*i+1]);
        printhex(picConfigMemory[2*i]);
        printstr_p(PSTR("\n_Read:_"));
        printhex((tempdata >> 8) & 0x00FF);
        printhex(tempdata & 0x00FF);

        printstr_p(PSTR("\nNOTE:_This_programmer_cannot_disable_Low_"
            "Voltage_Programming\n"));
        printstr_p(PSTR("If_that_was_the_only_problem,_do_not_worry."));
        pic_exit_programming();
        return -1;
    }

    //Increment address
    pic_send_command(INCREMENT_ADDRESS);

}
pic_exit_programming();
return 0;
}

/*
 * pic_get_revision
 * Gets the revision of this PIC and prints it for debugging right now.
 */

```

```

uint8_t pic_get_revision (void) {
    uint16_t deviceid;
    pic_enter_programming();

    //Load configuration
    pic_send_command(0x00);
    pic_send_data(0x0000);

    pic_send_command(0x06);
    pic_send_command(0x06);
    pic_send_command(0x06);
    pic_send_command(0x06);
    pic_send_command(0x06);
    pic_send_command(0x06);

    //Should be advanced to 0x2006 by now
    //Get data
    pic_send_command(0x04);

    deviceid = pic_receive_data();

    //Mask off the lower 5 bits because they are just revision.
    deviceid = (deviceid >> 5) & 0xFF;

    switch(deviceid){
        case PIC16F648A:
            printstr_p (PSTR("\nPIC16F648A_detected"));
            printstr_p (PSTR("\nNote:_This_programmer_can_only_program_"
                "the_first_2K_of_memory_and_not_the_whole"));
            printstr_p (PSTR("\n4K_of_memory_on_this_device."));
            printstr_p (PSTR("\nIt_can_also_only_program_the_first_128_bytes"
                "_of_Data_and_not_all_256_bytes."));
            PICtype = PIC16F648A;
            return PIC16F648A;
            break;
        case PIC16F627A:
            printstr_p (PSTR("\nPIC16F627A_detected"));
            PICtype = PIC16F627A;
            return PIC16F627A;
            break;
        case 0x83:
            printstr_p (PSTR("\nPIC16F628A_detected"));
            PICtype = PIC16F628A;
            return PIC16F628A;
            break;
        case PIC16F627:
            printstr_p (PSTR("\nPIC16F627_detected"));
            PICtype = PIC16F627;
            return PIC16F627;
            break;
        case 0x3E:
            printstr_p (PSTR("\nPIC16F628_detected"));
            PICtype = PIC16F628;
            return PIC16F628;
    }
}

```

```

        break;
    default:
        printstr_p(PSTR("\nUnknown_PIC_Config_word_was:"));
        printhex((deviceid >> 8) & 0xFF);
        printhex(deviceid & 0xFF);
        return deviceid;
        break;
    }

    pic_exit_programming();
}

/*
 * pic-view
 * view the code on the PIC
 */
void pic_view(void) {
    pic_enter_programming();

    pic_send_command(Load_program_command);
    pic_send_data(0x0000);

    int16_t ix;
    for(ix = 0; ix < CHIPMEMSIZE / 2; ix++) {
        if((ix % 16) == 0) { /*line numbers and newlines */
            printstr_p(PSTR("\n"));
            char line_num[10];
            sprintf(line_num, "%04X:\t", (2*ix));
            printstr(line_num);

        }
        pic_send_command(Read_program_command);
        uint16_t curr_mem = pic_receive_data();
        printhex(curr_mem & 0xFF);
        printhex((curr_mem >> 8) & 0xFF);
        pic_send_command(Increment_address);
    }

    pic_exit_programming();
    pic_enter_programming();

    printstr_p(PSTR("\n_DATA:"));
    for(ix = 0; ix < 128; ix++){
        if((ix % 32) == 0) { /*line numbers and newlines */
            printstr_p(PSTR("\n"));
            char line_num[10];
            sprintf(line_num, "%04X:\t", (ix));
            printstr(line_num);
        }
        pic_send_command(Read_data_command);

        uint16_t curr_mem = pic_receive_data();
        printhex(curr_mem & 0x00FF);
    }
}

```

```

    pic_send_command(INCREMENT_ADDRESS);
}

pic_exit_programming();
pic_enter_programming();

pic_send_command(LOAD_CONFIG_COMMAND);
pic_send_data(0x3FFF);

printstr_p (PSTR("\n_USER_ID_:"));

for(ix = 0; ix < 4 ; ix++) {

    pic_send_command(READ_PROGRAM_COMMAND);
    uint16_t curr_mem = pic_receive_data();
    printhex(curr_mem & 0xFF);
    printhex((curr_mem >> 8) & 0xFF);
    pic_send_command(INCREMENT_ADDRESS);

}

pic_send_command(INCREMENT_ADDRESS);
pic_send_command(INCREMENT_ADDRESS);
pic_send_command(INCREMENT_ADDRESS);

printstr_p (PSTR("\n_CONFIG_WORD_:"));

pic_send_command(READ_PROGRAM_COMMAND);
uint16_t curr_mem = pic_receive_data();
printhex(curr_mem & 0xFF);
printhex((curr_mem >> 8) & 0xFF);

pic_exit_programming();
}

/*
 * pic_total_erase
 * Perform total PIC bulk erase. Necessary to clear data protection bits.
 */
void pic_total_erase (void) {

    printstr_p (PSTR("\nPerforming_total_bulk_erase_to_"
                    "clear_code_protection ..."));
    pic_enter_programming();

    if((PICtype == PIC16F628) || (PICtype == PIC16F627)) {
        pic_send_command(LOAD_CONFIG_COMMAND);
        pic_send_data(0x3FFF);

        pic_send_command(INCREMENT_ADDRESS);
        pic_send_command(INCREMENT_ADDRESS);
        pic_send_command(INCREMENT_ADDRESS);
        pic_send_command(INCREMENT_ADDRESS);
        pic_send_command(INCREMENT_ADDRESS);
    }
}

```

```
pic_send_command(INCREMENT_ADDRESS);
pic_send_command(INCREMENT_ADDRESS);

pic_send_command(BULK_ERASE_SETUP_ONE);
pic_send_command(BULK_ERASE_SETUP_TWO);

pic_send_command(BEGIN_ERASE_PROGRAM);

_delay_ms(13);

pic_send_command(BULK_ERASE_SETUP_ONE);
pic_send_command(BULK_ERASE_SETUP_TWO);
} else {

    pic_send_command(LOAD_CONFIG_COMMAND);

    pic_send_data(0x3FFF);

    pic_send_command(BULK_ERASE_PROGRAM);

    _delay_ms(6);
}

pic_exit_programming();
}
```

---



# Appendix H

## IQ Demodulator DSP Source Code Listing

### H.1 Programming the IQ Demodulator DSP

The demodulation dsPIC, the dsPIC33FJ256MC710, can be programmed using Microchip's MPLAB tools. The code can be compiled using the MPLAB IDE Version 8.33 with the MPLAB C compiler for dsPIC version 3.12. The PIC was burned using the MPLAB ICD 2 programming device. A README.txt file can be found with the source code in `./homenilm/firmware` in the bucket repository. The board is powered with  $\pm 15$  Volt rails.

### H.2 DSP Firmware

The code included here takes the magnitude of I and Q and combines them to accomplish AM demodulation.

The square root algorithm was taken from Al-Thaddeus Avestruz and the Quick Select median algorithm was taken from Nicolas Devillard.

#### H.2.1 `main.c`

---

```
/*
*****
*   2005 Microchip Technology Inc.
*
*   FileName:        main.c
*   Dependencies:    Header (.h) files if applicable, see below
*   Processor:       dsPIC33Fxxx/PIC24Hxxx
*   Compiler:        MPLAB C30 v3.00 or higher
*   Tested On:       dsPIC33FJ256GP710
*
*   SOFTWARE LICENSE AGREEMENT:
*   Microchip Technology Incorporated ("Microchip") retains all ownership and
*   intellectual property rights in the code accompanying this message and in all
*   derivatives hereto. You may use this code, and any derivatives created by
```

\* any person or entity by or on your behalf, exclusively with Microchip's  
 \* proprietary products. Your acceptance and/or use of this code constitutes  
 \* agreement to the terms and conditions of this notice.  
 \*  
 \* CODE ACCOMPANYING THIS MESSAGE IS SUPPLIED BY MICROCHIP "AS IS". NO  
 \* WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT  
 \* NOT LIMITED  
 \* TO, IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND  
 \* FITNESS FOR A  
 \* PARTICULAR PURPOSE APPLY TO THIS CODE, ITS INTERACTION WITH  
 \* MICROCHIP'S  
 \* PRODUCTS, COMBINATION WITH ANY OTHER PRODUCTS, OR USE IN ANY  
 \* APPLICATION.  
 \*  
 \* YOU ACKNOWLEDGE AND AGREE THAT, IN NO EVENT, SHALL MICROCHIP BE  
 \* LIABLE, WHETHER  
 \* IN CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE OR BREACH OF  
 \* STATUTORY DUTY),  
 \* STRICT LIABILITY, INDEMNITY, CONTRIBUTION, OR OTHERWISE, FOR ANY  
 \* INDIRECT, SPECIAL,  
 \* PUNITIVE, EXEMPLARY, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, FOR  
 \* COST OR EXPENSE OF  
 \* ANY KIND WHATSOEVER RELATED TO THE CODE, HOWSOEVER CAUSED, EVEN IF  
 \* MICROCHIP HAS BEEN  
 \* ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE  
 \* FULLEST EXTENT  
 \* ALLOWABLE BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY  
 \* WAY RELATED TO  
 \* THIS CODE, SHALL NOT EXCEED THE PRICE YOU PAID DIRECTLY TO MICROCHIP  
 \* SPECIFICALLY TO  
 \* HAVE THIS CODE DEVELOPED.  
 \*  
 \* You agree that you are solely responsible for testing the code and  
 \* determining its suitability. Microchip has no obligation to modify, test,  
 \* certify, or support the code.

\* REVISION HISTORY:

-----

<i>Author</i>	<i>Date</i>	<i>Comments on this revision</i>
<i>Settu D.</i>	<i>07/09/06</i>	<i>First release of source file</i>

-----

\* ADDITIONAL NOTES:

\* This code is tested on Explorer 16 board with dsPIC33FJ256GP710 controller  
 \*  
 \* The Processor starts with the Internal oscillator without PLL enabled and then the Clock is  
 \* switched to PLL Mode.

\*\*\*\*\*/

```
#if defined(__dsPIC33F__)
#include "p33Fxxxx.h"
#elif defined(__PIC24H__)
#include "p24Hxxxx.h"
```



**#endif**

**#include** "i2cdac.h"  
**#include** "adcDrv2.h"  
**#include** "ocmodules.h"

*// Internal FRC Oscillator*

*\_FOSCSEL(FNOSC\_FRC);*

*\_FOSC(FCKSM\_CSECMD & OSCIOFNC\_OFF & POSCMD\_XT);*

*//\_FOSC(FCKSM\_CSECMD & OSCIOFNC\_OFF & POSCMD\_OFF);*

*// FRC Oscillator*

*//*  
*Clock*  
*Switch*  
*is*  
*enabled*  
*and*  
*Fail*  
*Safe*  
*Clock*  
*Monitor*  
*is*  
*disabled*  
*//*  
*OSC2*  
*Pin*  
*Function*  
*OSC2*  
*is*  
*Clock*  
*Output*  
*//*  
*Primary*  
*Oscillator*  
*Mode:*  
*Disabled*

```

_FWDT(FWDTEN_OFF); // Watchdog Timer
    Enabled/disabled by user software

//
// (LPRC
//
// can
//
// be
//
// disable
//
// by
//
// clearin
//
// SWDI
//
// bit
//
// in
//
// RCON
//
// registe:

_FPOR(FPWRT_PWR1); // Turn off the
    power-up timers.
_FGS(GCP_OFF); // Disable Code
    Protection

// Instantiate Drive and Data objects
I2CDAC_DRV i2cdac= I2CSDAC_DRV_DEFAULTS;
I2CDAC_DATA wData;
I2CDAC_DATA rData;

//unsigned int wBuff[10],rBuff[10];
unsigned int enable;

int main(void)
{
//int i=0;

// Configure Oscillator to operate the device at 40Mhz
// Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
// Fosc= 8M*40/(2*2)=80Mhz for 8M input clock
// Only 73.7 for Internal FRC Oscillator, so Fcy = 36.86 MHz
    PLLFBD=38; // M=40 (M = 2 + PLLDIV)
    CLKDIVbits.PLLPOST=0; // N1=2 (N1 = 2 + PLLPOST)
    CLKDIVbits.PLLPRE=0; // N2=2 More complicated
    OSCTUN=0; // Tune FRC oscillator, if FRC is used
    CLKDIVbits.FRCDIV = 0;

```

```

// Disable Watch Dog Timer
    RCONbits.SWDTEN=0;

// Make all the ANx pins as digital pins
    AD1PCFGL=0xFFFF;
    AD1PCFGH=0xFFFF;

// Clock switch to incorporate PLL
//Clock not installed ... oops
//_builtin_write_OSCCONH(0x03);           // Initiate Clock Switch to Primary
    _builtin_write_OSCCONH(0x01);

// Oscillator
// with
// PLL
// (NOSC=0b011

    _builtin_write_OSCCONL(0x01);           // Start clock switching
    //while (OSCCONbits.COSC != 0b011); // Wait for Clock switch to occur
while (OSCCONbits.COSC != 0b001); // Wait for Clock switch to occur

// Wait for PLL to lock
    while(OSCCONbits.LOCK!=1) {};

// Initialise I2C peripheral and Driver
    i2cdac.init(&i2cdac);

// Initialise Data to be written to serial EEPROM
    //for(i=0;i<10;i++)
    // wBuff[i]=i;

// Initialise I2C Data object for Write operation
    //wData.buff=wBuff;
    //wData.n=10;
    //wData.addr=0x00;
    //wData.csel=0x00;
    wData.addr = 0x00;
    wData.sample = 0x00;
    wData.channel = 0x00;
    wData.mode = DAC_UPDATE;
    wData.burstmode = DAC_BURST;

// Initialise I2C Data Object for Read operation
    //rData.buff=rBuff;
    //rData.n=10;
    //rData.addr=0x00;
    //rData.csel=0x00;

// Enable data write to I2C serial EEPROM

```

```

    //enable=1;

initOCModules();

//Need a nice long delay to wait for power rails to stabilize
//current board has offset problems that need the modulation
//source to be active before measuring.

//wait 3 seconds using Timer 6/7
//Fcy is 36.86 MHz, period is 27.1 ns
//With 256:1 prescaler, each tick is 6.95 us
//3 sec / 6.96 us = 431953

T6CONbits.T32 = 1;
T6CONbits.TCKPS = 0b11; //256 to 1 prescaler
T6CONbits.TCS = 0; //Use Tcy
PR6 = (431953 & 0xFFFF);
PR7 = 0x06; //High word of the count
TMR6 = 0;
TMR7 = 0;
IFS3bits.T7IF = 0;
T6CONbits.TON = 1;

while(!IFS3bits.T7IF);

T6CONbits.TON = 0;

initAdc1(); // Initialize the A/D converter to convert Channel 5
initDma0(); // Initialise the DMA controller to
            //buffer ADC data in conversion order

while(!performCalibration()); // Wait for offset calibration

while(1)
{
    // Write Data
    i2cdac.oData=&wData;
    i2cdac.cmd = I2C_WRITE;

    while(i2cdac.cmd!=I2C_BURSTOK )
    {
        i2cdac.tick(&i2cdac);
    }
    wData.sample = pendLatestValue();

    //wData.sample = (unsigned int) (ADC1BUF0 & 0x3FF);

    //wData.sample = 0x5555;

    /*if(wData.sample == 0x0000)
    {
        wData.sample = 0xFFFF;
    }

```

```

    } else {
        wData.sample = 0x0000;
    }
    */

        // Read Data
        //i2cmem.oData=&rData;
        //i2cmem.cmd = I2C_READ;
        //while(i2cmem.cmd!=I2C_IDLE)

    //{
        //    i2cmem.tick(&i2cmem);
        //}

    //}
};
}

```

---

## H.2.2 adcDrv2.h

---

```

/*****
*   2005 Microchip Technology Inc.
*
*   FileName:        adcDrv2.h
*   Dependencies:    Other (.h) files if applicable, see below
*   Processor:       dsPIC33Fxxx/PIC24Hxxx
*   Compiler:        MPLAB C30 v3.00 or higher
*
*   SOFTWARE LICENSE AGREEMENT:
*   Microchip Technology Incorporated ("Microchip") retains all ownership and
*   intellectual property rights in the code accompanying this message and in all
*   derivatives hereto. You may use this code, and any derivatives created by
*   any person or entity by or on your behalf, exclusively with Microchip's
*   proprietary products. Your acceptance and/or use of this code constitutes
*   agreement to the terms and conditions of this notice.
*
*   CODE ACCOMPANYING THIS MESSAGE IS SUPPLIED BY MICROCHIP "AS IS". NO
*   WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
*   NOT LIMITED
*   TO, IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND
*   FITNESS FOR A
*   PARTICULAR PURPOSE APPLY TO THIS CODE, ITS INTERACTION WITH
*   MICROCHIP'S
*   PRODUCTS, COMBINATION WITH ANY OTHER PRODUCTS, OR USE IN ANY
*   APPLICATION.
*
*   YOU ACKNOWLEDGE AND AGREE THAT, IN NO EVENT, SHALL MICROCHIP BE
*   LIABLE, WHETHER
*   IN CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE OR BREACH OF
*   STATUTORY DUTY),
*   STRICT LIABILITY, INDEMNITY, CONTRIBUTION, OR OTHERWISE, FOR ANY
*   INDIRECT, SPECIAL,

```

\* PUNITIVE, EXEMPLARY, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, FOR COST OR EXPENSE OF  
 \* ANY KIND WHATSOEVER RELATED TO THE CODE, HOWSOEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT  
 \* ALLOWABLE BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS CODE, SHALL NOT EXCEED THE PRICE YOU PAID DIRECTLY TO MICROCHIP SPECIFICALLY TO HAVE THIS CODE DEVELOPED.

\* You agree that you are solely responsible for testing the code and determining its suitability. Microchip has no obligation to modify, test, certify, or support the code.

\* REVISION HISTORY:

```

  * ~~~~~
  * Author          Date      Comments on this revision
  * ~~~~~
  * Settu D         03/09/06  First release of source file
  * ~~~~~
  
```

\* ADDITIONAL NOTES:

\* 1. This file contains definitions commonly used in this project.

```

  *****/
  #ifndef __ADCDRV2_H__
  #define __ADCDRV2_H__
  
```

```

  // External Functions
  extern void initAdc1(void);
  extern void initDma0(void);
  extern void __attribute__(( _interrupt_ )) _DMA0Interrupt(void);
  extern void __attribute__(( _interrupt_ )) _DMA1Interrupt(void);
  extern int  pendLatestValue(void);
  extern int  performCalibration(void);
  
```

#endif

## H.2.3 adcDrv2.c

```

  /*****
  * 2005 Microchip Technology Inc.
  *
  * FileName:      adcDrv2.c
  * Dependencies:  Header (.h) files if applicable, see below
  * Processor:     dsPIC33Fxxx/PIC24Hxxx
  * Compiler:      MPLAB C30 v3.00 or higher
  *
  * SOFTWARE LICENSE AGREEMENT:
  * Microchip Technology Incorporated ("Microchip") retains all ownership and
  
```

\* intellectual property rights in the code accompanying this message and in all derivatives hereto. You may use this code, and any derivatives created by any person or entity by or on your behalf, exclusively with Microchip's proprietary products. Your acceptance and/or use of this code constitutes agreement to the terms and conditions of this notice.

\*  
 \* CODE ACCOMPANYING THIS MESSAGE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS CODE, ITS INTERACTION WITH MICROCHIP'S PRODUCTS, COMBINATION WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

\*  
 \* YOU ACKNOWLEDGE AND AGREE THAT, IN NO EVENT, SHALL MICROCHIP BE LIABLE, WHETHER IN CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE OR BREACH OF STATUTORY DUTY), STRICT LIABILITY, INDEMNITY, CONTRIBUTION, OR OTHERWISE, FOR ANY INDIRECT, SPECIAL, PUNITIVE, EXEMPLARY, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, FOR COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE CODE, HOWSOEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWABLE BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS CODE, SHALL NOT EXCEED THE PRICE YOU PAID DIRECTLY TO MICROCHIP SPECIFICALLY TO HAVE THIS CODE DEVELOPED.

\*  
 \* You agree that you are solely responsible for testing the code and determining its suitability. Microchip has no obligation to modify, test, certify, or support the code.

\* REVISION HISTORY:

```

  ~~~~~
  * Author          Date          Comments on this revision
  ~~~~~
  * Settu D          03/09/06 First release of source file
  ~~~~~**
  
```

\* ADDITIONAL NOTES:

\* This file contains two functions – `initAdc1()`, `initDma0` and `_DMA0Interrupt()`.

\*\*\*\*\*/

```

#if defined(_dsPIC33F_)
#include "p33Fxxxx.h"
#elif defined(_PIC24H_)
#include "p24Hxxxx.h"

```

```

#endif

#include "adcDrv2.h"
#include "funcs.h"
//#include "tglPin.h"

#define NUMSAMP 1
signed int BufferA1[NUMSAMP] __attribute__((space(dma)));
signed int BufferB1[NUMSAMP] __attribute__((space(dma)));

signed int BufferA2[NUMSAMP] __attribute__((space(dma)));
signed int BufferB2[NUMSAMP] __attribute__((space(dma)));

int latestValue,ready, calibrated, oldvalue;

signed int chan1off, chan2off;

signed int calibrate1 [100];
signed int calibrate2 [100];

void ProcessADCSamples(signed int * AdcBuffer1, signed int * AdcBuffer2);
//void ProcessADC2Samples(int * AdcBuffer);

//Functions:
//initAdc1() is used to configure A/D to convert AIN0 using CH0 and CH1 sample/hold in
//sequential mode
//at 1.1MHz throughput rate. ADC clock is configured at 13.3Mhz or Tad=75ns
void initAdc1(void)
{
    calibrated = 0;
    //DAC can run at 22.22 kSPS on "Fast" mode. Sample about that level.

    AD1CON1bits.FORM = 1;    // Data Output Format: Signed Integer
    AD1CON1bits.SSRC = 7;    // Internal Counter (SAMC) ends sampling and
//                               starts conversion
    AD1CON1bits.ASAM = 1;    // ADC Sample Control: Sampling begins
//                               immediately after conversion
    AD1CON1bits.AD12B = 1;   // 12-bit ADC operation

    AD2CON1bits.FORM = 1;    // Data Output Format: Signed Integer
    AD2CON1bits.SSRC = 7;    // Internal Counter (SAMC) ends sampling and
//                               starts conversion
    AD2CON1bits.ASAM = 1;    // ADC Sample Control: Sampling begins
//                               immediately after conversion
    AD2CON1bits.AD12B = 1;   // 12-bit ADC operation

    //AD1CON2bits.CHPS = 1;           // Converts CH0/CH1

    //AD2CON2bits.CHPS = 1;           // Converts CH0/CH1

```



```

AD1CON3bits.ADRC=0;           // ADC Clock is derived from Systems
                               Clock
AD1CON3bits.SAMC=0;          // Auto Sample Time = 0*Tad
AD1CON3bits.ADCS=63;         // ADC Conversion Clock
                                $Tad = Tcy * (ADCS + 1) = (1/40M) * 64 = 1.6\mu s (625kHz)$ 
                               // ADC Conversion
                               Time for 12-bit
                                $Tc = 14 * Tab =$ 
                               22.4us (44.6kHz)

AD2CON3bits.ADRC=0;           // ADC Clock is derived from Systems
                               Clock
AD2CON3bits.SAMC=0;          // Auto Sample Time = 0*Tad
AD2CON3bits.ADCS=63;         // ADC Conversion Clock
                                $Tad = Tcy * (ADCS + 1) = (1/40M) * 64 = 1.6\mu s (625kHz)$ 
                               // ADC Conversion
                               Time for 12-bit
                                $Tc = 14 * Tab =$ 
                               22.4us (44.6kHz)

AD1CON1bits.ADDMABM = 1; // DMA buffers are built in conversion order
                               mode
AD1CON2bits.SMPI = 0;        // SMPI must be 0 – increment DMA address
                               every time

AD2CON1bits.ADDMABM = 1; // DMA buffers are built in conversion order
                               mode
AD2CON2bits.SMPI = 0;        // SMPI must be 0 – increment DMA address
                               every time

//AD1CHS0/AD1CHS123: A/D Input Select Register
AD1CHS0bits.CH0SA=0;         // MUXA +ve input selection (AIN0) for CH0
AD1CHS0bits.CH0NA=0;         // MUXA -ve input selection (Vref-) for CH0

AD2CHS0bits.CH0SA=1;         // MUXA +ve input selection (AIN1) for ADC2
AD2CHS0bits.CH0NA=0;         // MUXA -ve input selection (Vref-) for ADC2
                               CH0

//AD1CHS123bits.CH123SA=0; // MUXA +ve input selection (AIN0) for CH1
//AD1CHS123bits.CH123NA=0; // MUXA -ve input selection (Vref-) for CH1

//AD1PCFGH/AD1PCFGL: Port Configuration Register
AD1PCFGL=0xFFFF;
AD1PCFGH=0xFFFF;
AD1PCFGLbits.PCFG0 = 0;     // AN0 as Analog Input

AD2PCFGL=0xFFFF;
//There is no AD2PCFGH register...
AD2PCFGLbits.PCFG1 = 0;     // AN1 as Analog Input

IFS0bits.AD1IF = 0;         // Clear the A/D interrupt flag bit

```

```

IEC0bits.AD1IE = 0;           // Do Not Enable A/D interrupt

IFS1bits.AD2IF = 0;         // Clear the A/D interrupt flag bit
IEC1bits.AD2IE = 0;         // Do Not Enable A/D interrupt

AD2CON1bits.ADON = 1;      // Turn on the A/D converter
AD1CON1bits.ADON = 1;      // Turn on the A/D converter

ready = 0;
//tglPinInit();
}

// DMA0 configuration
// Direction: Read from peripheral address 0-x300 (ADC1BUF0) and write to DMA RAM
// AMODE: Register indirect with post increment
// MODE: Continuous, Ping-Pong Mode
// IRQ: ADC Interrupt
// ADC stores results stored alternatively between DMA_BASE[0]/DMA_BASE[16] on every
// 16th DMA request

void initDma0(void)
{
    DMA0CONbits.AMODE = 0;    // Configure DMA for Register indirect with post
        increment
    DMA0CONbits.MODE = 2;    // Configure DMA for Continuous Ping-Pong
        mode

    DMA1CONbits.AMODE = 0;    // Configure DMA for Register indirect with post
        increment
    DMA1CONbits.MODE = 2;    // Configure DMA for Continuous Ping-Pong
        mode

    DMA0PAD=(int)&ADC1BUF0;
    DMA0CNT=(NUMSAMP-1);

    DMA1PAD=(int)&ADC2BUF0;
    DMA1CNT=(NUMSAMP-1);

    DMA0REQ=13;

DMA1REQ=21;

    DMA0STA = _builtin_dmaoffset(BufferA1);
    DMA0STB = _builtin_dmaoffset(BufferB1);

    DMA1STA = _builtin_dmaoffset(BufferA2);
    DMA1STB = _builtin_dmaoffset(BufferB2);

    IFS0bits.DMA0IF = 0;     //Clear the DMA interrupt flag bit
    IEC0bits.DMA0IE = 1;     //Set the DMA interrupt enable bit

    IFS0bits.DMA1IF = 0;     //Clear the DMA interrupt flag bit

```

```

IEC0bits.DMA1IE = 1;                //Set the DMA interrupt enable bit

DMA0CONbits.CHEN=1;
DMA1CONbits.CHEN=1;

}

int pendLatestValue(void) {
    while(ready == 0);
    return latestValue;
}

/*=====
_DMA0Interrupt(): ISR name is chosen from the device linker script.
=====
unsigned int Dma0Buffer = 0;

void __attribute__((interrupt, no_auto_psv)) _DMA0Interrupt(void)
{
    if(Dma0Buffer == 0)
    {
        ProcessADCSamples(BufferA1,BufferA2);
    }
    else
    {
        ProcessADCSamples(BufferB1,BufferB2);
    }

    Dma0Buffer ^= 1;

    //tglPin();                // Toggle RA6
    IFS0bits.DMA0IF = 0;      //Clear the DMA0 Interrupt Flag
}

/*=====
_DMA1Interrupt(): ISR name is chosen from the device linker script.
=====
unsigned int Dma1Buffer = 0;

void __attribute__((interrupt, no_auto_psv)) _DMA1Interrupt(void)
{
    /*
        if(Dma1Buffer == 0)
        {
            ProcessADC2Samples(BufferA2);
        }

```

```

        else
        {
            ProcessADC2Samples(BufferB2);
        }
    */
    Dma1Buffer ^= 1;

    //tglPin(); // Toggle RA6
    IFS0bits.DMA1IF = 0; //Clear the DMA0 Interrupt Flag
}

void ProcessADCSamples(signed int * AdcBuffer1,signed int * AdcBuffer2)
{
    /* Do something with ADC Samples */
    if(calibrated < 100){
        calibrate1 [ calibrated ] = *AdcBuffer1;
        calibrate2 [ calibrated ] = *AdcBuffer2;
        calibrated++;
    } else{
        typeCartesian currentSample = {*AdcBuffer1,*AdcBuffer2,0};
        latestValue = Magnitude(currentSample,chan1off,chan2off, oldvalue);
        oldvalue = latestValue;
        ready = 1;
    }
    //latestValue = *AdcBuffer1 >> 2;
}

int performCalibration(void) {
    if(calibrated < 100) return 0;
    else {
        chan1off = quick_select( calibrate1 ,100);
        chan2off = quick_select( calibrate2 ,100);
        oldvalue = 0;
        return 1;
    }
}
/*
void ProcessADC2Samples(int * AdcBuffer)
{
    // Do something with ADC Samples
    //ready = 1;
    //latestValue = *AdcBuffer;
}
*/

```

---

## H.2.4 funcs.h

```

/*
Header for DSP functions from Al-Thaddeus Avestruz
*/

```

```

#ifndef FUNCS_H
#define FUNCS_H

//Integer Data Types
//In lcc short int=int=2 bytes, long=4 bytes
typedef signed short int _Int16;
typedef unsigned short int _UInt16;
typedef signed long int _Int32;
typedef unsigned long int _UInt32;

typedef struct
{
    _Int16 d;
    _Int16 q;
    _UInt16 ovf;//overflow
} typeCartesian;

inline _UInt16 Lsqrt(_UInt32 x);

_UInt16 Magnitude(typeCartesian C1, signed int offset1, signed int offset2, int oldvalue);

signed int quick_select(signed int arr [], int n);

#endif

```

---

## H.2.5 funcs.c

---

```

/*
DSP functions borrowed from Al-Thaddeus Avestruz's Aardvark
project
*/

#include "funcs.h"

inline _UInt16 Lsqrt(_UInt32 x)
{ //Tested 4/18/08
    unsigned long y=0;
    _UInt32 y2=0;
    _UInt16 yp=0;
    _UInt16 yshift=0;
    _UInt16 i=0;

    yshift = 0x8000;
    yp = 0x8000;
    for (i=1; i<17; i++)
    {
        //yp = y + yp;
        yp = y + yshift;
        y2 = (_UInt32)yp * (_UInt32)yp;
        if (y2<x) y=yp;
        yshift = yshift>>1;
        //yp = 0x8000>>1;
    }
}

```

```

    }

    return(y);
}

#define MAGTHRESHOLD 20
#define SIGNTHRESHOLD 3
#define HYSTTHRESH 2

_UInt16 Magnitude(typeCartesian C1, signed int offset1, signed int offset2, int oldvalue)
{
    _UInt32 lprod1=0;
    _UInt32 lprod2=0;
    _UInt32 sum=0;
    _UInt16 x=0;
    _Int16 basicsum=0;

    //This would be nice to do on the DSP hardware, but currently it is not
    //needed

    //First subtract off the measured offset. Being able to change this while
    //running would be nice.
    //The constant is used to center the wave in the lower half of the range.
    //This eliminates zero crossings at the expense of more noise throughout the
    //wave. Zero crossings introduce digital noise, while this offset introduces
    //analog noise. When the gain stage is fixed, this hack should not be needed.
    C1.d = C1.d - offset1 - 270;
    C1.q = C1.q - offset2 - 270;

    if(C1.d >= 0) basicsum = 1; else basicsum = -1;

    //Take the absolute value of the args
    if (C1.d<0) C1.d = -C1.d;
    if (C1.q<0) C1.q = -C1.q;

    //Square them. Note the typecasting to prevent overflowing
    lprod1 = (_UInt32)C1.d * (_UInt32)C1.d;
    lprod2 = (_UInt32)C1.q * (_UInt32)C1.q;

    //Now add them to an *unsigned* sum
    sum = lprod1 + lprod2;
    sum = sum << 8;
    x = l_sqrt(sum);
    x = x >> 4;

    //Stop overflows before they happen
    if(x > 511) x = 511;

    //Take I as the sign reference.
    if(basicsum >= 0) x = x + 512;
    else
    x = 512 - x;
}

```

```

//Double the wave. This goes hand in hand with the earlier offset hack. This
//digitally gets the output right, but more signal input would be better.
x = x << 1;
if(x > 1023) x = 1023;

return(x);
}

/*
 * This Quickselect routine is based on the algorithm described in
 * "Numerical recipes in C", Second Edition,
 * Cambridge University Press, 1992, Section 8.5, ISBN 0-521-43108-5
 * This code by Nicolas Devillard - 1998. Public domain.
 */

#define ELEM_SWAP(a,b) { register int t=(a);(a)=(b);(b)=t; }

signed int quick_select(signed int arr [], int n)
{
    signed int low, high ;
    signed int median;
    signed int middle, ll, hh;

    low = 0 ; high = n-1 ; median = (low + high) / 2;
    for (;;) {
        if (high <= low) /* One element only */
            return arr[median] ;

        if (high == low + 1) { /* Two elements only */
            if (arr[low] > arr[high])
                ELEM_SWAP(arr[low], arr[high]) ;
            return arr[median] ;
        }

        /* Find median of low, middle and high items; swap into position low */
        middle = (low + high) / 2;
        if (arr[middle] > arr[high]) ELEM_SWAP(arr[middle], arr[high]) ;
        if (arr[low] > arr[high]) ELEM_SWAP(arr[low], arr[high]) ;
        if (arr[middle] > arr[low]) ELEM_SWAP(arr[middle], arr[low]) ;

        /* Swap low item (now in position middle) into position (low+1) */
        ELEM_SWAP(arr[middle], arr[low+1]) ;

        /* Nibble from each end towards middle, swapping items when stuck */
        ll = low + 1;
        hh = high;
        for (;;) {
            do ll++; while (arr[low] > arr[ll]) ;
            do hh--; while (arr[hh] > arr[low]) ;

            if (hh < ll)
                break;

```

```

    ELEM_SWAP(arr[ll], arr[hh]) ;
}

/* Swap middle item (in position low) back into correct position */
ELEM_SWAP(arr[low], arr[hh]) ;

/* Re-set active partition */
if (hh <= median)
    low = ll;
    if (hh >= median)
        high = hh - 1;
}
}

#undef ELEM_SWAP

```

---

## H.2.6 i2cdac.h

---

```

/*****
* 2005 Microchip Technology Inc.
*
* FileName: i2cEmem.h
* Dependencies: Other (.h) files if applicable, see below
* Processor: dsPIC33Fxxxx/PIC24Hxxxx
* Compiler: MPLAB C30 v3.00 or higher
*
* SOFTWARE LICENSE AGREEMENT:
* Microchip Technology Incorporated ("Microchip") retains all ownership and
* intellectual property rights in the code accompanying this message and in all
* derivatives hereto. You may use this code, and any derivatives created by
* any person or entity by or on your behalf, exclusively with Microchip's
* proprietary products. Your acceptance and/or use of this code constitutes
* agreement to the terms and conditions of this notice.
*
* CODE ACCOMPANYING THIS MESSAGE IS SUPPLIED BY MICROCHIP "AS IS". NO
* WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
* NOT LIMITED
* TO, IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND
* FITNESS FOR A
* PARTICULAR PURPOSE APPLY TO THIS CODE, ITS INTERACTION WITH
* MICROCHIP'S
* PRODUCTS, COMBINATION WITH ANY OTHER PRODUCTS, OR USE IN ANY
* APPLICATION.
*
* YOU ACKNOWLEDGE AND AGREE THAT, IN NO EVENT, SHALL MICROCHIP BE
* LIABLE, WHETHER
* IN CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE OR BREACH OF
* STATUTORY DUTY),
* STRICT LIABILITY, INDEMNITY, CONTRIBUTION, OR OTHERWISE, FOR ANY
* INDIRECT, SPECIAL,
* PUNITIVE, EXEMPLARY, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, FOR
* COST OR EXPENSE OF

```



\* ANY KIND WHATSOEVER RELATED TO THE CODE, HOWSOEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWABLE BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS CODE, SHALL NOT EXCEED THE PRICE YOU PAID DIRECTLY TO MICROCHIP SPECIFICALLY TO HAVE THIS CODE DEVELOPED.

\* You agree that you are solely responsible for testing the code and determining its suitability. Microchip has no obligation to modify, test, certify, or support the code.

\* REVISION HISTORY:

```

-----
* Author          Date          Comments on this revision
-----
* Settu D.        07/09/06 First release of source file
-----

```

\* ADDITIONAL NOTES:

```

*****/
#ifndef _I2CDAC_H_
#define _I2CDAC_H_

```

```

#define MAX_RETRY 1000
#define ONE_BYTE 1
#define TWO_BYTE 2

```

```

// DAC ADDRESS SIZE
#define ADDRWIDTH ONE_BYTE

```

```

// DAC DRIVER COMMAND DEFINITION
#define I2C_IDLE 0
#define I2C_WRITE 1
#define I2C_READ 2
#define I2C_BURSTOK 3
#define I2C_ERR 0xFFFF

```

```

// DAC BURST OPTIONS
#define DAC_NOBURST 0
#define DAC_BURST 1

```

```

// DAC MODE OPTIONS
#define DAC_STORE 0
#define DAC_UPDATE 1
#define DAC_SYNC 2
#define DAC_BROADCAST 3

```

```

// DAC DATA OBJECT
//typedef struct {
//    unsigned int *buff;

```

```

//      unsigned int n;
//      unsigned int addr;
//      unsigned int csel;
//}I2CDAC_DATA;

// DAC DATA OBJECT
typedef struct {
    unsigned int addr;
    unsigned int sample;
    unsigned int channel;
    unsigned int mode;
    unsigned int burstmode;
}I2CDAC_DATA;

// DAC DRIVER OBJECT
typedef struct {
    unsigned int  cmd;
    I2CDAC_DATA *oData;
    void (*init)(void *);
    void (*tick)(void *);
}I2CDAC_DRV;

#define I2CSDAC_DRV_DEFAULTS { 0,\
    (I2CDAC_DATA *)0,\
    (void (*)(void *))I2CDACinit,\
    (void (*)(void *))I2CDACdrv}

void I2CDACinit(I2CDAC_DRV *);

void I2CDACdrv(I2CDAC_DRV *);

#endif

```

---

## H.2.7 i2cdac.c

---

```

/*****
*   2005 Microchip Technology Inc.
*
*   FileName:       i2cdac.c
*   Dependencies:   Header (.h) files if applicable, see below
*   Processor:      dsPIC33Fxxx/PIC24Hxxx
*   Compiler:       MPLAB C30 v3.00 or higher
*   Tested On:      dsPIC33FJ256GP710
*
*   SOFTWARE LICENSE AGREEMENT:
*   Microchip Technology Incorporated ("Microchip") retains all ownership and
*   intellectual property rights in the code accompanying this message and in all
*   derivatives hereto. You may use this code, and any derivatives created by
*   any person or entity by or on your behalf, exclusively with Microchip's

```

\* *proprietary products. Your acceptance and/or use of this code constitutes agreement to the terms and conditions of this notice.*

\*  
 \* *CODE ACCOMPANYING THIS MESSAGE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS CODE, ITS INTERACTION WITH MICROCHIP'S PRODUCTS, COMBINATION WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.*

\*  
 \* *YOU ACKNOWLEDGE AND AGREE THAT, IN NO EVENT, SHALL MICROCHIP BE LIABLE, WHETHER IN CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE OR BREACH OF STATUTORY DUTY), STRICT LIABILITY, INDEMNITY, CONTRIBUTION, OR OTHERWISE, FOR ANY INDIRECT, SPECIAL, PUNITIVE, EXEMPLARY, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, FOR COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE CODE, HOWSOEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWABLE BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS CODE, SHALL NOT EXCEED THE PRICE YOU PAID DIRECTLY TO MICROCHIP SPECIFICALLY TO HAVE THIS CODE DEVELOPED.*

\*  
 \* *You agree that you are solely responsible for testing the code and determining its suitability. Microchip has no obligation to modify, test, certify, or support the code.*

\* *REVISION HISTORY:*

<i>Author</i>	<i>Date</i>	<i>Comments on this revision</i>
<i>Settu D.</i>	<i>07/09/06</i>	<i>First release of source file</i>

\*\*\*\*\*/

```
#if defined(_dsPIC33F_)
#include "p33Fxxxx.h"
#elif defined(_PIC24H_)
#include "p24Hxxxx.h"
#endif
```

```
#include "i2cdac.h"
```

```
unsigned int jDone;
```

/\*=====

*I2C Master Interrupt Service Routine*

```
=====
void __attribute__((interrupt, no_auto_psv)) _MI2C1Interrupt(void)
{
    jDone=1;
    IFS1bits.MI2C1IF = 0;      //Clear the DMA0 Interrupt Flag;
}

```

*I2C Slave Interrupt Service Routine*

```
=====
void __attribute__((interrupt, no_auto_psv)) _SI2C1Interrupt(void)
{
    IFS1bits.SI2C1IF = 0;      //Clear the DMA0 Interrupt Flag
}

```

*I2C Peripheral Initialisation*

```
=====
void I2CDACinit(I2CDAC_DRV *i2cDac)
{
    i2cDac->cmd=0;
    i2cDac->oData=0;

    // Configure SCA/SDA pin as open-drain
    ODCGbits.ODCG2=1;
    ODCGbits.ODCG3=1;

    I2C1CONbits.A10M=0; //7 bit addressing
    I2C1CONbits.SCLREL=1; //Release SCL (only matters if slave
    //I2C1BRG=300; //Baud rate generator...leaving alone for now
    I2C1BRG = 80; //We want Fast standard...400 kHz..Actually is 381

    I2C1ADD=0; //Set slave address to 0
    I2C1MSK=0; //No bit masking...all must match

    I2C1CONbits.I2CEN=1; //Enable I2C
    IEC1bits.MI2C1IE = 1; //Enable I2C Master events interrupt
    IFS1bits.MI2C1IF = 0; //Clear I2C Master events flag
}

```

*I2C DAC, STATE-MACHINE BASED DRIVER*

```
=====
void I2CDACdrv(I2CDAC_DRV *i2cDac)
{

```

```

static int state=0, cntr=0, rtrycntr=0;

switch(state)
{
case 0:
    if( (i2cDac->cmd == I2C_WRITE) || (i2cDac->cmd == I2C_READ) )
        state=1;

    break;

    /*=====*/
    /* Control/Address Phase */
    /*=====*/
case 1:
    // Start Condition
    I2C1CONbits.SEN=1;
    state=state+1;
    break;

case 2:
    // Start Byte with device select id
    if(jDone==1) {
        jDone=0;
        state=state+1;
        //The following assumes we will always write to the DAC
        I2C1TRN=(0x0098)|(((i2cDac->oData->addr)&0x3)<<1); //Assert DAC
        //hardwired address ANDed with address selected
        //I2C1TRN=(0x00A0)|(((i2cDac->oData->csel)&0x7)<<1);
    }
    break;

case 3:
    // Send control byte, if ack is received. Else Retry
    if(jDone==1) {
        jDone=0;

        if(I2C1STATbits.ACKSTAT==1) { // Ack Not received, Retry

            if(rtrycntr < MAX_RETRY)
                state=18;
            else
                state=16; // Flag
                error and exit

        } else {

            rtrycntr=0;

            //Powerdown not implemented
            I2C1TRN=((i2cDac->oData->addr)&0x000C) << 4|
                (((i2cDac->oData->mode)&0x0003) << 4)|
                (((i2cDac->oData->channel)&0x0003) << 1);
        }
    }
}

```

```

state = state + 2;
/*
    #if ADDRWIDTH==TWO_BYTE
    I2C1TRN=((i2cDac->oData->addr)&0xFF00)>>8;
        state=state+1;
    #endif

    #if ADDRWIDTH==ONE_BYTE
    I2C1TRN=((i2cDac->oData->addr));
    state=state+2;
    #endif
*/
    }
    }
    break;

/*
case 4:
    // Send address byte 2, if ack is received. Else Flag error and exit
    if(jDone==1) {
        jDone=0;

        if(I2C1STATbits.ACKSTAT==1) { // Ack Not received, Flag error
            and exit
            state=16;

        } else {

            #if ADDRWIDTH==TWO_BYTE
            I2C1TRN=((i2cMem->oData->addr)&0x00FF);
            #endif
            state=state+1;
        }
    }
    break;
*/

case 5:
    // Read or Write
    if(jDone==1) {
        jDone=0;

        if(I2C1STATbits.ACKSTAT==1) { // Ack Not received, Flag error
            and exit
            state=16;

        } else {

            if(i2cDac->cmd == I2C_WRITE)
                state=state+1;

            if(i2cDac->cmd == I2C_READ)
                state=8;
        }
    }
*/

```

```

        }

    }
    break;

    /*=====*/
    /* Write Data Phase                               */
    /*=====*/

case 6:
    // Send first data chunk
    i2cDac->cmd = I2C_WRITE;
    I2C1TRN=(i2cDac->oData->sample & 0x03FC) >> 2;
    state=state+1;
    cntr=cntr+1;
    break;

case 7:
    //Send second data chunk
    if(jDone==1) {
        jDone=0;
        state = state+1;

        if(I2C1STATbits.ACKSTAT==1) { // Ack Not received, Flag error and exit
            state=16;
        } else {
            I2C1TRN=(i2cDac->oData->sample & 0x0002) << 6;
        }
    }
    break;

case 8:
    //Set completed flag and hold selected
    if(jDone==1) {
        jDone=0;

        if(i2cDac->oData->burstmode == DAC_BURST) {
            state = 6;
            i2cDac->cmd = I2C_BURSTOK;
        } else {
            state = 14;
        }

        if(I2C1STATbits.ACKSTAT==1) { // Ack Not received, Flag error and exit
            state=16;
        }
    }
    break;
/*
case 7:
    // Look for end of data or no Ack
    if(jDone==1) {
        jDone=0;
        state=state-1;

```

```

        if (I2C1STATbits.ACKSTAT==1) { // Ack Not received, Flag error
            and exit
            state=16;
        } else {

            if (cntr== i2cMem->oData->n) // Close the
                state=14; // Close the
                Frame
            }
        }
    }
    break;
*/
/*=====*/
/* Read Data Phase */
/*=====*/
/* case 8:
    // Repeat Start
    I2C1CONbits.RSEN=1;
    state=state+1;
    break;

case 9:
    // Re-send control byte with W/R=R
    if (jDone==1) {
        jDone=0;
        state=state+1;
        I2C1TRN=(0x00A1)|(((i2cMem->oData->tsel)&0x7)<<1);
    }
    break;

case 10:
    // Check, if control byte went ok
    if (jDone==1) {
        jDone=0;
        state=state+1;

        if (I2C1STATbits.ACKSTAT==1) // Ack Not received, Flag error
            and exit
            state=16;

    }
    break;

case 11:
    // Receive Enable
    I2C1CONbits.RCEN=1;
    state++;
    break;

case 12:
    // Receive data
    if (jDone==1) {
        jDone=0;

```



```

        state=state+1;

        *(i2cMem->oData->buff+cntr)=I2C1RCV;
        cntr++;

        if (cntr== i2cMem->oData->n) {
            I2C1CONbits.ACKDT=1;        // No ACK
        } else {
            I2C1CONbits.ACKDT=0;        // ACK
        }

        I2C1CONbits.ACKEN=1;

    }
    break;

case 13:
    if (jDone==1)    {
        jDone=0;
        if (cntr== i2cMem->oData->n)
            state=state+1;
        else
            state=state-2;
    }
    break;
*/
/*=====*/
/* Stop Sequence                               */
/*=====*/
case 14:
    I2C1CONbits.PEN=1;
    state++;
    break;

case 15:
    if (jDone==1)    {
        jDone=0;
        state=0;
        cntr=0;
        i2cDac->cmd=0;
    }
    break;

/*=====*/
/* Set Error                               */
/*=====*/
case 16:
    I2C1CONbits.PEN=1;
    state++;
    break;

case 17:
    if (jDone==1)    {
        jDone=0;

```

```

        state=0;
        rtrycntr=0;
        cntr=0;
        i2cDac->cmd=0xFFFF;
    }
    break;

    /*=====*/
    /* Retry                                     */
    /*=====*/
case 18:
    I2C1CONbits.PEN=1;
    state++;
    rtrycntr++;
    break;

case 19:
    if(jDone==1) {
        jDone=0;
        state=0;
        cntr=0;
    }
    break;

}
}

```

---

## H.2.8 ocmodes.h

---

```

/*****
*   2005 Microchip Technology Inc.
*
*   FileName:        adcDrv2.h
*   Dependencies:    Other (.h) files if applicable, see below
*   Processor:       dsPIC33Fxxx/PIC24Hxxx
*   Compiler:        MPLAB C30 v3.00 or higher
*
*   SOFTWARE LICENSE AGREEMENT:
*   Microchip Technology Incorporated ("Microchip") retains all ownership and
*   intellectual property rights in the code accompanying this message and in all
*   derivatives hereto. You may use this code, and any derivatives created by
*   any person or entity by or on your behalf, exclusively with Microchip's
*   proprietary products. Your acceptance and/or use of this code constitutes
*   agreement to the terms and conditions of this notice.
*
*   CODE ACCOMPANYING THIS MESSAGE IS SUPPLIED BY MICROCHIP "AS IS". NO
*   WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
*   NOT LIMITED
*   TO, IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND
*   FITNESS FOR A

```

\* PARTICULAR PURPOSE APPLY TO THIS CODE, ITS INTERACTION WITH MICROCHIP'S PRODUCTS, COMBINATION WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

\* YOU ACKNOWLEDGE AND AGREE THAT, IN NO EVENT, SHALL MICROCHIP BE LIABLE, WHETHER IN CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE OR BREACH OF STATUTORY DUTY), STRICT LIABILITY, INDEMNITY, CONTRIBUTION, OR OTHERWISE, FOR ANY INDIRECT, SPECIAL, PUNITIVE, EXEMPLARY, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, FOR COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE CODE, HOWSOEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWABLE BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS CODE, SHALL NOT EXCEED THE PRICE YOU PAID DIRECTLY TO MICROCHIP SPECIFICALLY TO HAVE THIS CODE DEVELOPED.

\* You agree that you are solely responsible for testing the code and determining its suitability. Microchip has no obligation to modify, test, certify, or support the code.

\* REVISION HISTORY:

```

* ~~~~~
* Author          Date      Comments on this revision
* ~~~~~
* Settu D          03/09/06 First release of source file
* ~~~~~*

```

\* ADDITIONAL NOTES:

\* 1. This file contains definitions commonly used in this project.

\*\*\*\*\*/

```

#ifndef _OCMODULES_H_
#define _OCMODULES_H_

#define MODULATION_CHAN 1
#define QUADRATURE_CHAN 2
#define MANUAL_1 3
#define MANUAL_2 4
#define MANUAL_3 5
#define MANUAL_4 6
#define FILTER_CLK 7

```

```

// External Functions
extern void initOCModules(void);

```

```

#endif

```

## H.2.9 ocmodes.c

```

/*****
*   2005 Microchip Technology Inc.
*
*   FileName:      adcDrv2.c
*   Dependencies:  Header (.h) files if applicable, see below
*   Processor:     dsPIC33Fxxx/PIC24Hxxx
*   Compiler:      MPLAB C30 v3.00 or higher
*
*   SOFTWARE LICENSE AGREEMENT:
*   Microchip Technology Incorporated ("Microchip") retains all ownership and
*   intellectual property rights in the code accompanying this message and in all
*   derivatives hereto. You may use this code, and any derivatives created by
*   any person or entity by or on your behalf, exclusively with Microchip's
*   proprietary products. Your acceptance and/or use of this code constitutes
*   agreement to the terms and conditions of this notice.
*
*   CODE ACCOMPANYING THIS MESSAGE IS SUPPLIED BY MICROCHIP "AS IS". NO
*   WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
*   NOT LIMITED
*   TO, IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND
*   FITNESS FOR A
*   PARTICULAR PURPOSE APPLY TO THIS CODE, ITS INTERACTION WITH
*   MICROCHIP'S
*   PRODUCTS, COMBINATION WITH ANY OTHER PRODUCTS, OR USE IN ANY
*   APPLICATION.
*
*   YOU ACKNOWLEDGE AND AGREE THAT, IN NO EVENT, SHALL MICROCHIP BE
*   LIABLE, WHETHER
*   IN CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE OR BREACH OF
*   STATUTORY DUTY),
*   STRICT LIABILITY, INDEMNITY, CONTRIBUTION, OR OTHERWISE, FOR ANY
*   INDIRECT, SPECIAL,
*   PUNITIVE, EXEMPLARY, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, FOR
*   COST OR EXPENSE OF
*   ANY KIND WHATSOEVER RELATED TO THE CODE, HOWSOEVER CAUSED, EVEN IF
*   MICROCHIP HAS BEEN
*   ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE
*   FULLEST EXTENT
*   ALLOWABLE BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY
*   WAY RELATED TO
*   THIS CODE, SHALL NOT EXCEED THE PRICE YOU PAID DIRECTLY TO MICROCHIP
*   SPECIFICALLY TO
*   HAVE THIS CODE DEVELOPED.
*
*   You agree that you are solely responsible for testing the code and
*   determining its suitability. Microchip has no obligation to modify, test,
*   certify, or support the code.
*
*   REVISION HISTORY:
*   ~~~~~
*   Author      Date      Comments on this revision
*   ~~~~~

```

```

* Settu D 03/09/06 First release of source file
* -----**
*
* ADDITIONAL NOTES:
* This file contains two functions – initAdc1(), initDma0 and _DMA0Interrupt().
*
* *****/

#if defined(_dsPIC33F_)
#include "p33Fxxxx.h"
#elif defined(_PIC24H_)
#include "p24Hxxxx.h"
#endif

#include "ocmodules.h"

void initOCModules(void){

//First drive the FLOPRST line high
TRISGbits.TRISG6 = 0;
PORTGbits.RG6 = 1;

OC1CONbits.OCTSEL = 1; // Use Timer3 for clock source
OC2CONbits.OCTSEL = 0; // Use Timer2 for clock source

OC2R = 20; //Trigger on 0
OC1R = 0x00;

OC2CONbits.OCM = 0b011; //Set Modulation to toggle mode
OC1CONbits.OCM = 0b011; //Set Quadrature to toggle mode

//Now set up Timer 2 to go at 448 kHz
//Fcy = 40 MHz is assumed
//Fcy = 36.85 MHz for now
//Closest period is 82 ticks. No prescaling or 32 bit timer.
//This gives a frequency of: 449.4 kHz.

T2CONbits.T32 = 0; //Two 16 bit timers
T2CONbits.TCS = 0; //Clock source is Fcy
T2CONbits.TCKPS = 0b00; //No prescaler
T2CONbits.TGATE = 0; //Gating accumulation disabled

T3CONbits.TCS = 0;
T3CONbits.TCKPS = 0b00;
T3CONbits.TGATE = 0;

TMR2 = 0x00;
TMR3 = 0x00;
//PR2 = 82; //Set timer period to 89
//Must be miscomputing clock or hitting a prescaler. Halve it.
PR2 = 41;
//PR3 = 83;
PR3 = 82; //Intentionally off to send signal for loopback
T2CONbits.TON = 1; //Activate timer

```

T3CONbits.TON = 1;

}

---

## H.2.10 traps.c

---

```
/******  
* 2005 Microchip Technology Inc.  
*  
* FileName: traps.c  
* Dependencies: Header (.h) files if applicable, see below  
* Processor: dsPIC33Fxxx/PIC24Hxxx  
* Compiler: MPLAB C30 v3.00 or higher  
*  
* SOFTWARE LICENSE AGREEMENT:  
* Microchip Technology Incorporated ("Microchip") retains all ownership and  
* intellectual property rights in the code accompanying this message and in all  
* derivatives hereto. You may use this code, and any derivatives created by  
* any person or entity by or on your behalf, exclusively with Microchip's  
* proprietary products. Your acceptance and/or use of this code constitutes  
* agreement to the terms and conditions of this notice.  
*  
* CODE ACCOMPANYING THIS MESSAGE IS SUPPLIED BY MICROCHIP "AS IS". NO  
* WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT  
* NOT LIMITED  
* TO, IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND  
* FITNESS FOR A  
* PARTICULAR PURPOSE APPLY TO THIS CODE, ITS INTERACTION WITH  
* MICROCHIP'S  
* PRODUCTS, COMBINATION WITH ANY OTHER PRODUCTS, OR USE IN ANY  
* APPLICATION.  
*  
* YOU ACKNOWLEDGE AND AGREE THAT, IN NO EVENT, SHALL MICROCHIP BE  
* LIABLE, WHETHER  
* IN CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE OR BREACH OF  
* STATUTORY DUTY),  
* STRICT LIABILITY, INDEMNITY, CONTRIBUTION, OR OTHERWISE, FOR ANY  
* INDIRECT, SPECIAL,  
* PUNITIVE, EXEMPLARY, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, FOR  
* COST OR EXPENSE OF  
* ANY KIND WHATSOEVER RELATED TO THE CODE, HOWSOEVER CAUSED, EVEN IF  
* MICROCHIP HAS BEEN  
* ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE  
* FULLEST EXTENT  
* ALLOWABLE BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY  
* WAY RELATED TO  
* THIS CODE, SHALL NOT EXCEED THE PRICE YOU PAID DIRECTLY TO MICROCHIP  
* SPECIFICALLY TO  
* HAVE THIS CODE DEVELOPED.  
*  
* You agree that you are solely responsible for testing the code and  
* determining its suitability. Microchip has no obligation to modify, test,
```

```

* certify , or support the code.
*
* REVISION HISTORY:
* -----
* Author          Date      Comments on this revision
* -----
* Settu D          07/09/06 First release of source file
* -----
*
* ADDITIONAL NOTES:
* 1. This file contains trap service routines (handlers) for hardware
*    exceptions generated by the dsPIC33F device.
* 2. All trap service routines in this file simply ensure that device
*    continuously executes code within the trap service routine. Users
*    may modify the basic framework provided here to suit to the needs
*    of their application.
*
*
*****/

#if defined(_dsPIC33F_)
#include "p33fxxxx.h"
#elif defined(_PIC24H_)
#include "p24hxxxx.h"
#endif

void __attribute__((__interrupt__)) _OscillatorFail (void);
void __attribute__((__interrupt__)) _AddressError(void);
void __attribute__((__interrupt__)) _StackError(void);
void __attribute__((__interrupt__)) _MathError(void);
void __attribute__((__interrupt__)) _DMACError(void);

void __attribute__((__interrupt__)) _AltOscillatorFail (void);
void __attribute__((__interrupt__)) _AltAddressError(void);
void __attribute__((__interrupt__)) _AltStackError(void);
void __attribute__((__interrupt__)) _AltMathError(void);
void __attribute__((__interrupt__)) _AltDMACError(void);

/*
Primary Exception Vector handlers:
These routines are used if INTCON2bits.ALTIPT = 0.
All trap service routines in this file simply ensure that device
continuously executes code within the trap service routine. Users
may modify the basic framework provided here to suit to the needs
of their application.
*/
void __attribute__((interrupt, no_auto_psv)) _OscillatorFail (void)
{
    INTCON1bits.OSCFAIL = 0; //Clear the trap flag
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _AddressError(void)
{
    INTCON1bits.ADDRERR = 0; //Clear the trap flag

```

```

        while (1);
    }
void __attribute__((interrupt, no_auto_psv)) _StackError(void)
{
    INTCON1bits.STKERR = 0; //Clear the trap flag
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _MathError(void)
{
    INTCON1bits.MATHERR = 0; //Clear the trap flag
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _DMACError(void)
{
    INTCON1bits.DMACERR = 0; //Clear the trap flag
    while (1);
}

/*
Alternate Exception Vector handlers:
These routines are used if INTCON2bits.ALTIVT = 1.
All trap service routines in this file simply ensure that device
continuously executes code within the trap service routine. Users
may modify the basic framework provided here to suit to the needs
of their application.
*/

void __attribute__((interrupt, no_auto_psv)) _AltOscillatorFail (void)
{
    INTCON1bits.OSCFAIL = 0;
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _AltAddressError(void)
{
    INTCON1bits.ADDRERR = 0;
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _AltStackError(void)
{
    INTCON1bits.STKERR = 0;
    while (1);
}

void __attribute__((interrupt, no_auto_psv)) _AltMathError(void)
{
    INTCON1bits.MATHERR = 0;

```



```
    while (1);  
}  
  
void __attribute__((interrupt, no_auto_psv)) _AltDMACError(void)  
{  
    INTCON1bits.DMACERR = 0; //Clear the trap flag  
    while (1);  
}
```

---



# Bibliography

- [1] N. Bowler. Frequency-dependence of relative permeability in steel. *Review of Quantitative Nondestructive Evaluation*, 25:1269–1276, 2006.
- [2] R. W. Cox, P. Bennett, D. McKay, J. Paris, and S. B. Leeb. Using the non-intrusive load monitor for shipboard supervisory control. In *IEEE Electric Ship Technologies Symposium*, Arlington, VA, May 2007.
- [3] T. DeNucci, R. Cox, S. B. Leeb, J. Paris, T. J. McCoy, C. Laughman, and W. Greene. Diagnostic indicators for shipboard systems using non-intrusive load monitoring. In *IEEE Electric Ship Technologies Symposium*, Philadelphia, Pennsylvania, July 2005.
- [4] Inc. Ferroxcube. Tx25/15/10-3e6 datasheet. Available <http://www.ferroxcube.com/prod/assets/tx251510.pdf>.
- [5] W. Greene, J. S. Ramsey, S. B. Leeb, T. DeNucci, J. Paris, M. Obar, R. Cox, C. Laughman, and T. J. McCoy. Non-intrusive monitoring for condition-based maintenance. In *American Society of Naval Engineers Reconfigurability and Survivability Symposium*, Atlantic Beach, Florida, February 2005.
- [6] U. A. Khan, S. B. Leeb, and M. C. Lee. A multiprocessor for transient event detection. *IEEE Transactions on Power Delivery*, 12(1):51–60, 1997.
- [7] S. B. Leeb, S. R. Shaw, and Jr. J. L. Kirtley. Transient event detection in spectral envelope estimates for nonintrusive load monitoring. *IEEE Transactions on Power Delivery*, 10(3):1200–1210, July 1995.
- [8] G. Mitchell, R. W. Cox, M. Piber, P. Bennett, J. Paris, W. Wichakool, and S. B. Leeb. Shipboard fluid system diagnostic indicators using nonintrusive load monitoring. In *American Society for Naval Engineers Day 2007*, Arlington, VA, June 2007.
- [9] G. R. Mitchell, R. W. Cox, J. Paris, and S. B. Leeb. Shipboard fluid system diagnostic indicators using non-intrusive load. *Naval Engineers Journal*, 119(1), November 2007.
- [10] J. P. Mosman, R. W. Cox, D. McKay, S. B. Leeb, and T. McCoy. Diagnostic indicators for shipboard cycling systems using non-intrusive load monitoring. In *American Society for Naval Engineers Day 2006*, Arlington, VA, June 2006.

- [11] L. K. Norford and S. B. Leeb. Non-intrusive electrical load monitoring in commercial buildings based on steady state and transient load-detection algorithms. *Energy and Buildings*, 24:51–64, 1996.
- [12] E. Proper, R. W. Cox, S. B. Leeb, K. Douglas, J. Paris, W. Wichakool, L. Foulks, R. Jones, P. Branch, A. Fuller, J. Leghorn, and G. Elkins. Field demonstration of a real-time non-intrusive monitoring system for condition-based maintenance. In *Electric Ship Design Symposium*, National Harbor, Maryland, February 2009.
- [13] J. S. Ramsey, S. B. Leeb, T. DeNucci, J. Paris, M. Obar, R. Cox, C. Laughman, and T. J. McCoy. Shipboard applications of non-intrusive load monitoring. In *American Society of Naval Engineers Reconfigurability and Survivability Symposium*, Atlantic Beach, Florida, February 2005.
- [14] S. R. Shaw, S. B. Leeb, L. K. Norford, and R. W. Cox. Nonintrusive load monitoring and diagnostics in power systems. *IEEE Transactions on Instrumentation and Measurement*, 57(7):1445–1454, July 2008.