Demonstration System for a Low-Power Seismic Detector and Classifier

by

Elliot Richard Ranger

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2003

Author .....
Department of Electrical Engineering and Computer Science
May 9, 2003

Certified by .....
Thomas F. Knight, Jr.
Senior Research Scientist, MIT
Thesis Supervisor

Certified by .....
Kenneth M. Houston
Group Leader - Analog Systems, C.S. Draper Laboratory
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Demonstration System for a Low-Power Seismic Detector and Classifier

by

Elliot Richard Ranger

Submitted to the Department of Electrical Engineering and Computer Science
on May 9, 2003, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

Abstract

A low-power seismic detector and classifier was designed and implemented which was able to detect the footsteps of a person from as far as 35 meters away. Throughout the design an emphasis was placed on using low power circuitry and efficient algorithms. The test platform to demonstrate the concepts of the design utilizes a revolutionary low-power microcontroller and Digital Signal Processor (DSP) from Texas Instruments, Inc.. The DSP is a fixed-point processor that is underclocked to minimize power consumption and the microcontroller has idle modes which consume microamps of power. The system is designed to run on battery power, and uses solar power to continually charge the batteries during the day. Lastly, a "Commercial Off the Shelf" RF module allows multiple sensors to communicate with themselves to triangulate position, or to relay detections and commands to and from a base station.

Thesis Supervisor: Thomas F. Knight, Jr.
Title: Senior Research Scientist, MIT

Thesis Supervisor: Kenneth M. Houston
Title: Group Leader - Analog Systems, C.S. Draper Laboratory

Acknowledgment

May 9, 2003

This thesis was prepared at The Charles Stark Draper Laboratory, Inc., under Internal Company Sponsored Research Project IRD03-2-5042.

I always enjoy reading the acknowledgment section of my fellow colleagues' thesis documents. It is the one section of the report where you get to take a glimpse into the author's life and learn about the people who influenced them. Sir Isaac Newton once said, "If I have seen further [than others] it is by standing on the shoulders of giants". I prefer to take a more humble approach and say it is because I have had the shoulders of the great people who have helped me that I have seen so far.

My family has always been a source of strength and nourishment. Lots of things come and go in your life: you change careers, where you live, and what walk of life you choose to pursue. One thing that has always remained a constant in my life has been my family. They have inspired me with their continual support to always reach for the stars. I owe them a deep sense of gratitude for making education such a high priority.

Thomas Knight and Kenneth Houston, my two advisors, helped me solidify my project, come up with the resources to execute the project, and gave me the confidence that I was doing a notable job. We have had some interesting experiences together from lunches at the Cambridge Brewing Company to exhaustive days of testing in the park. Without the support of these two, this project would never have occurred or have been completed.

I owe a special recognition to the guys at Draper who helped me throughout my stay there: Jack McKenna, Mike Matranga, Jim Scholten, Dan McGaffigan, Charles Barone, Bob Menyhert, Bill Russo, Dennis Kessler and Fred Kasparian. Some of the guys I talked to about the technical aspects of the project and other guys I talked to about the stock market or politics, but I value the assistance all of them provided. One person who deserves a little more thanks is my officemate Sermed Ashkouri. He got to put up with all the daily joys of having me in the same office.

Lastly, there are numerous friends and colleagues along the way that have provided me with a spring-board to bounce ideas off of. Although there is not room to mention all of you rest assured that the thanks is still forthcoming.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recently, there has been a great emphasis on low power circuit design. This has revolutionized many areas in computers and electronics. One area specifically benefitting from the improvements in lower power consumption is sensors. Sensors can now run for weeks with the power of only a couple Lithium Ion batteries. This means they are cheaper to operate, and more importantly, they require less human intervention because the operator does not have to replace the batteries as frequently. Texas Instruments, Inc., has an Ultra-Low-Power Microcontroller which only draws 2.5 uA of power in certain operating modes [13]. New advances in programmable parts are being made as well. Combining this low power mentality in an area which has not had much research is the goal of this thesis.

When a person or animal walks along the ground they emit seismic waves as a result of the impact. These waves then propagate through the ground. A geophone is a sensor that is able to measure the amplitude of these seismic waves in the ground. geophones can be used to measure everything from a car driving by to an outright earthquake. The critical part is then establishing algorithms that allow the device to differentiate seismic waves coming from a person versus other seismic activity.

This project is an improvement to current technology in a number of regards. First, it will focus on low power components. Next, it focuses on the development of simple yet effective algorithms to detect and classify people. Numerous algorithms exist for classifying data, however, most of them are not intended for low power

applications. The algorithms utilized for this project will need to be relatively simple so they can run in a low power environment, but also still need to maintain their effectiveness in classifying people. Lastly, is the consideration of alternative forms of energy. This area is frequently overlooked when designing systems, but will no doubt become more important in future designs with our diminishing supply of natural resources. Combining all these areas in a working prototype will help advance sensor technology as well as general design principles.

The methodology behind the design of this system was to create a platform that later could be built upon and expanded. The scope of the project is quite expansive and some areas have had more attention than others. Every attempt has been made to clearly document all the aspects of the project so that in the event that someone chooses to pursue an aspect of the project at a later point the design and results will be at their disposal.

1.1 Background

The ground, like any other elastic medium, allows waves to propagate through it. The impact from a footstep hitting the ground can be distinguished from as far as 100 meters away under ideal conditions [12]. The maximum distance is directly related to the attenuation rate of the ground and the type of wave being studied. There are four types of seismic waves that propagate through the ground: compression, shear, Rayleigh, and Love. These waves have varying diminishing amplitudes as they travel through the ground. The Rayleigh wave diminishes as $1/R$ while the shear and compressional waves diminish as $1/R^2$ [11]. The Love wave, which is caused by the layering of the soil, is not really considered. In footstep detection, the most important wave is the Rayleigh wave. It is a wave which travels along the surface of the earth. Its components expand in two dimensions and diminish exponentially with depth [11]. As a result of this, the wave can be detected at much further distances than the body waves (compressional and shear). Another thing to consider is how the energy from a footstep gets partitioned into the three waves. The shear and compressional waves

which are body waves contain roughly 26% and 7% of the energy, respectively, while the Rayleigh wave contains 67% of the energy [11]. Therefore the Rayleigh wave is the critical wave for footstep detection. Not only does it propagate through the ground over greater distances, but it is also where the bulk of the energy from a footstep gets transmitted. It is also possible to extract bearing information from the Rayleigh wave using a three-axis geophone [11].

1.2 Previous Work

There are numerous algorithms for classifying data, however, most of them require massive amounts of computational processing power or memory. For instance, Succi et al, used a Levenberg-Marquardt Neural Network Classifier to track vehicle data [10]. This produced good results, but it required 6MB of dynamic memory for its matrix processing. In a low powered embedded system running a classifier of that nature would require too much power.

Kenneth Houston and Dan McGaffigan at Draper Laboratory have done a significant amount of research in the area of personnel detection using seismic sensors [3]. Most systems prior to their work was transient based. The downfall of that approach is that many real-world signals unrelated to human locomotion look like transients. A systems designed like that will have either a very high false alarm rate or else will be insensitive. They introduced the idea of using spectrum analysis on envelope-detected seismic signals. This method not only produced reasonable detection ranges but also was significantly better at discriminating footsteps from other types of seismic sources. This work is the basis for the algorithms that were utilized in the system.

1.3 Research

Continuing the work on the algorithms developed by Kenneth Houston and Daniel McGaffigan, it became desirable to examine how different geological and topological

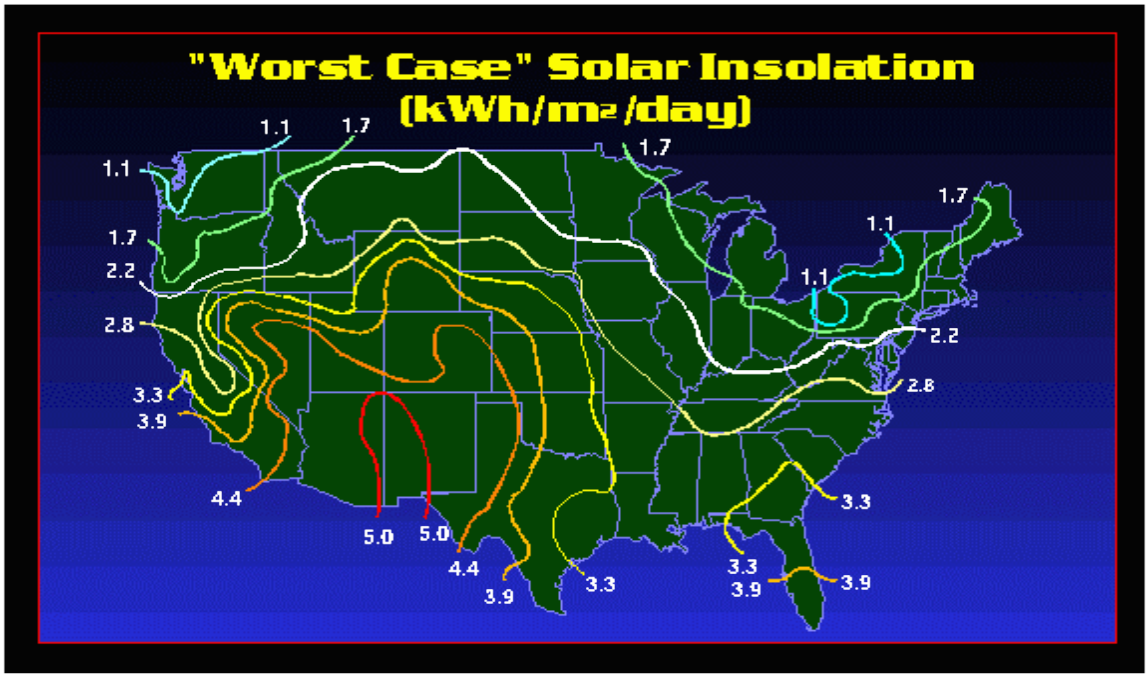
features affected the ability to detect and classify footsteps. Also, it became desirable to collect data from other ambulatory creatures, such as horses, to determine if there was noticeable features which would allow the system to discriminate from horses while still maintaining a simple algorithm.

The other area in which the project was expanded was to look at alternative forms of energy to power the system. There are many forms of energy on the planet. Wind, vibration, water, coal, oil, wood, nuclear fusion, and solar all produce energy that can be transferred into electrical energy. Considering the application of this project as a small, low-powered sensor, it is important that the energy source be able to convert the energy into electrical energy efficiently and that the mechanism are relatively small.

Energy can be readily converted from one form to another. However, most techniques do not yield enough electrical energy to sustain a system. Amirtharajah has proposed using ambient vibrational energy to power electrical devices [1]. Vibrational energy does not yield enough energy to power the prototype, but solar energy is a viable solution. Current research is around 20% efficiency which means for a 1 cm^2 solar cell 20 mW of energy can be obtained [2].

Photovoltaics are materials that when sunlight hits them an electron is released causing it to generate an electrical current. Photovoltaics are the cheapest way to produce electricity for smaller systems and often times the simplest and cleanest to operate. The problem with solar power is the sun is almost never directly overhead, except in the tropics. At a latitude of 45 degrees the solar radiation may vary from 92% (early summer) to 38% (early winter) [9]. At higher latitudes the distance from the sun to the earth becomes further and also the scattering of the sun's radiation from gases in the atmosphere becomes significant when considering the use of solar power as well. Furthermore, it is important to take into account the natural landscape such as mountains, altitude, and cloud cover. All these variables make solar power a very inconsistent source of energy. Figure 1-1 shows some of the typical solar isolation amounts in the United States.

In 1839, Edmond Becquerel was the first to discover that when sunlight is absorbed



Source: Sandia National Laboratories

Figure 1-1: Solar Isolation ($kWh/m^2/day$) in the U.S.

by certain materials it can produce an electrical current. It was not until the 1950 and the advent of solid-state devices that people were able to do something meaningful with this information. The space program was the first application for solar cells and in 1954 a 4% efficient silicon crystal was developed [9]. As early as 1958 a small array of solar cells was used to provide electrical power to a U.S. satellite.

Photovoltaic cells are created by doping a material like silicon with a substance like Boron or Phosphorus which has one less or one more electron respectively. A junction forms between the doped silicon and the undoped silicon. When a photon strikes the cell it contains enough energy to release the extra electron and allow it to move across the junction. A grid is set up to gather the current from a number of cells and different currents and voltages can be constructed depending on how the grid is arranged.

The most common photovoltaic cell used today is the single crystal silicon cell. The silicon is highly purified and sliced into wafers from single-crystal ingots, or grown as thin crystalline sheets or ribbons. Polycrystalline cells are available as well but are inherently less efficient, but they are cheaper to produce. Some of the most efficient cells are made from Gallium arsenide; however, they are very expensive. Currently, there has been a lot of focus on thin films made from amorphous silicon. Copper Indium Diselenide and Cadmium Telluride may also provide viable low-cost solutions. The thin films do not require a lot of material and have great manufacturabilty. Another area which is receiving a lot of attention is multijunction cells. This will allow the cell to use more of the spectrum from the sunlight giving higher efficiencies.

There are drawbacks with the use of solar energy. For one thing, it only works when the sun is out so batteries will be required during the evening hours or if the device is buried underground. Also, the solar cells are fragile so they will need to be protected from adverse weather conditions.

1.4 Overview

Ultimately, the goal of the project is to produce a bread-board prototype that is able to correctly detect and classify people. In order to accomplish this the device needs to incorporate many types of electronics. It will require a sensor to acquire the data, an Analog-to-Digital Converter (ADC) to convert the data to digital form, and a low power Digital Signal Processor (DSP) to process the data. In addition, the device will incorporate a radio frequency (RF) transmitter to relay data back to the operator when it has detected something. Low power devices will be featured in the design, and solar power as a means of powering the device will be explored.

Chapter 2

System Architecture

The clearest way to understand how the system processes data is to analyze it from a signal perspective. The next couple paragraphs describe the path a signal takes from when the sensor picks up vibrations all the way to the determination of whether or not the signal is classified as a person.

The geophone is an external sensor with a stake on it that penetrates the ground. It generates very small electrical voltages depending on the intensity of the propagating waves in the axis that the geophone is arranged in. The geophone is biased to fluctuate around 1.5 V, the middle of the full range input to the Analog-to-Digital Converter (ADC). This allows use of a single supply voltage for the front-end electronics and maximizes the signal voltage range. The geophone plugs into the prototype board through port J3. The port is able to support up to there seismic channels of data, however, currently the system is only utilizing a single channel. The other two channels could be used to concurrently process data from two other single-axis geophones or the data from one three-axis geophone.

When the signal arrives at the board it is initially filtered through the analog circuitry. It is filtered through two analog lowpass filters and one highpass filter. These filters bandlimit the signal to prevent aliasing. Then the signal goes through a single gain stage. Following the gain stage, the signal is sampled by a 12-bit ADC built into the microcontroller. The sampling rate is controlled by one of the internal timers on the microcontroller and is set at 1 kHz. After eight samples have been

collected by the ADC, the microcontroller directly writes the data into a certain memory location within the DSP through the Host Port Interface (HPI). Every 1200 samples, or 1.2 seconds at a 1 kHz sampling rate, the microcontroller takes the DSP out of its low-power mode and requests the DSP process the next block of data.

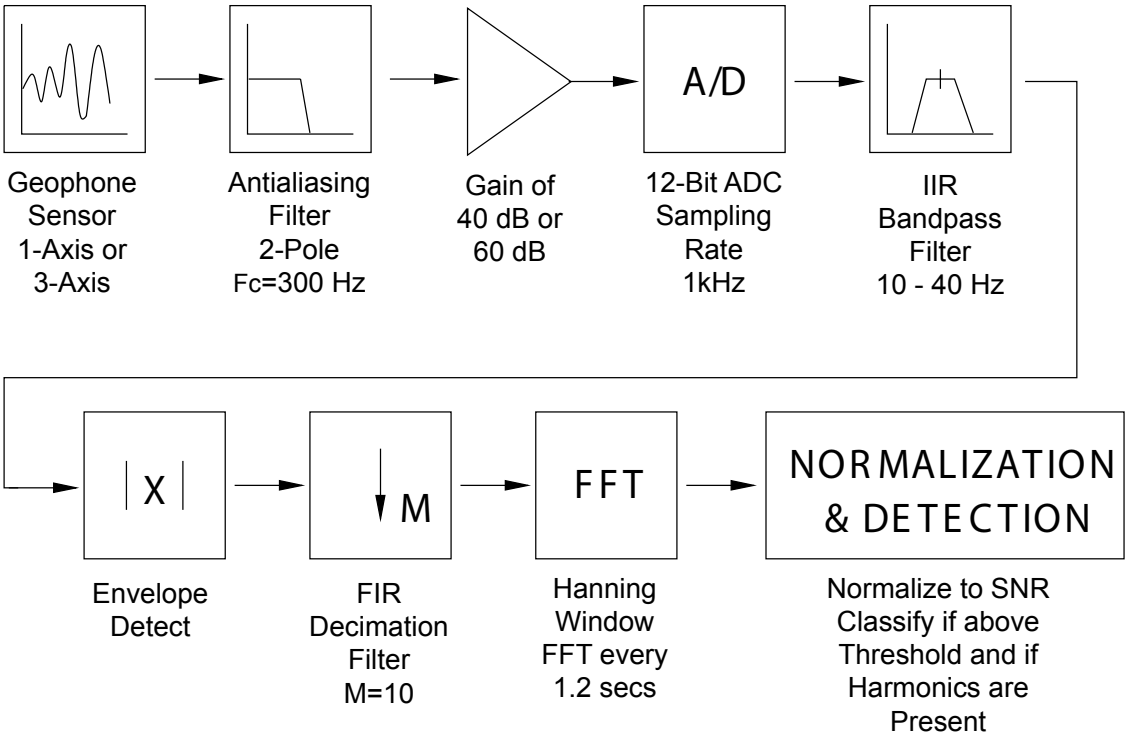


Figure 2-1: Signal Processing Path

Figure 2-1 shows the stages of the signal processing chain. The DSP begins by passing the data through a digital bandpass filter allowing signals within the range of 10 Hz to 40 Hz to pass. This bandpass range is geology dependent but was fixed for this project. The next step is critical for the detection process. The important part of detecting footsteps is the periodicity of the footsteps on the ground. The time of each distinct impact on the ground is the critical information that needs to be retained, as opposed to, the exact frequency content received at the sensor, which can be quite variable. An absolute value is performed on the whole signal to create an envelope of the received signal, which will peak for each distinct footstep. After this, the data is decimated and lowpass filtered again to prevent aliasing. The decimation

occurs in two separate steps; first by 5 and then by 2. The decimation removes high frequencies in the envelope data so as to smooth out the footstep pulses. It also allows the Fast Fourier Transform (FFT) to be smaller to get the frequency resolution that is required. A moving window technique employing a Hanning Window is then utilized because the amplitude of the data is changing as the distance from the person to the sensor changes with respect to time. Next, a 1024-point FFT is performed to convert the data into the frequency domain. The FFT returns the values of the frequency components as complex numbers. A function converts those real and imaginary parts into magnitudes, and the data is scaled to make it easier to differentiate the signal from the noise. A two-pass normalization is performed to estimate the background level. Then the background signal (converted to decibels) is subtracted from the seismic signal (also in decibels) so the detection is based on normalized signal levels. Lastly, the footstep discriminator algorithm is called to decide whether the footsteps are from a person. The details of the discriminating algorithm will be discussed later.

2.1 Digital Signal Processing

A Digital Signal Processor has specific hardware to expedite signal processing routines. It is a fixed-point processor to reduce power consumption. Using a fixed-point processor increases the complexity of the design and requires careful design to keep the signals scaled and in an appropriate range. Also, floating-point calculations take many additional clock cycles on a fixed-point processor so they should be avoided.

The way numbers are represented in the processor is an important aspect of the design stage. Using a fixed-point processor allows the programmer to pick any arbitrary number of bits to represent the integer portion of the number and the fractional component. As long as the representation remains consistent the operations will produce the correct results. Texas Instruments has provided a number of common DSP functions to assist developers using their processors. Whenever possible, these functions were utilized because they are written in assembly and attempt to maximize the efficiency of the algorithms for the hardware platform. The DSP Library functions

generally use a Q15 number representation, which means there is no integer portion, 15 bits of fractional data and one bit to represent the sign of the number. Therefore, all the numbers represented throughout the signal processing chain are scaled between -1 and 1.

2.1.1 Bandpass Filter

The Infinite Impulse Response (IIR) bandpass filter allows frequencies from within the range of 10 Hz to 40 Hz to pass. Originally, it was designed to be a 4 pole filter, 2 poles for the lowpass component and 2 poles for the highpass component, however, there were some difficulties getting the biquads to function properly. A simple 2 pole bandpass filter was resorted to, to get the system working. Within a Digital Signal Processor the easiest way to implement an IIR filter is through the difference equations. The single pole difference equations for the highpass and lowpass filter are described in the equations below.

$$Highpass : Y_i = \frac{N-1}{N} * Y_{i-1} + X_i - X_{i-1} \tag{2.1}$$

$$Lowpass : Y_i = \frac{N-1}{N} * Y_{i-1} + \frac{1}{N} * X_i \tag{2.2}$$

$$where\ N = \frac{\tau}{T},\ \tau = \frac{1}{2\pi * f_{cutoff}},\ T = sample\ period$$

To create the highpass filter $T = 1/1000$, $\tau = 1/(2\pi * 10Hz)$, and $N = 16$. The lowpass filter is realized in a similar fashion with $T = 1/1000$, $\tau = 1/(2\pi * 40Hz)$, and $N = 4$. This produces the following difference equations for the bandpass filter.

$$Highpass : Y_i = \frac{15}{16} Y_{i-1} + X_i - X_{i-1} \tag{2.3}$$

$$Lowpass : Y_i = \frac{3}{4}Y_{i-1} + \frac{1}{4}X_i \tag{2.4}$$

The fractional components are scaled as fractional components within the processor with 1 being represented by 32,767. Both of these filters are coded in assembly and can be found in Appendix F in the files `hp_filter.asm` and `lp_filter.asm`.

2.1.2 Envelope Detect

After the signal has been filtered it looks like the signal shown in Figure 2-2. Notice how the impact from the footsteps generates both positive and negative waves. The frequency of these waves is controlled by the terrain. The goal however is to treat this whole block as one impact and then determine the frequency that the impacts are occurring. In order to do this the absolute value function changes the sign of all the negative going waves and flips them about the axis so they are positive. This generates an envelope for each of the footsteps shown in Figure 2-3.

2.1.3 Decimation

Oversampling followed by decimation has several benefits. Oversampling allows the use of a less stringent analog antialiasing filter, which has a lower precision than a digital filter. In addition, a digital filter does not drift with temperature or time. The decimation stage requires a lowpass filter to prevent aliasing. The unique thing about decimating is that the function is the same as a regular Finite Infinite Response (FIR) filter except it is only necessary to do the operations on the values that will exist after decimating. It is implemented in assembly as a standard FIR filter that only runs on the samples that exist after decimating. Thus if it is decimating by two it only runs on every other sample and if it is decimating by 5 it runs on every 5th sample. For this application, the decimation operation is divided into two parts to limit the size of any filter. The other benefit of decimating is it smooths out the waveform. The same sample data after a decimation of 10 in shown in Figure 2-4.

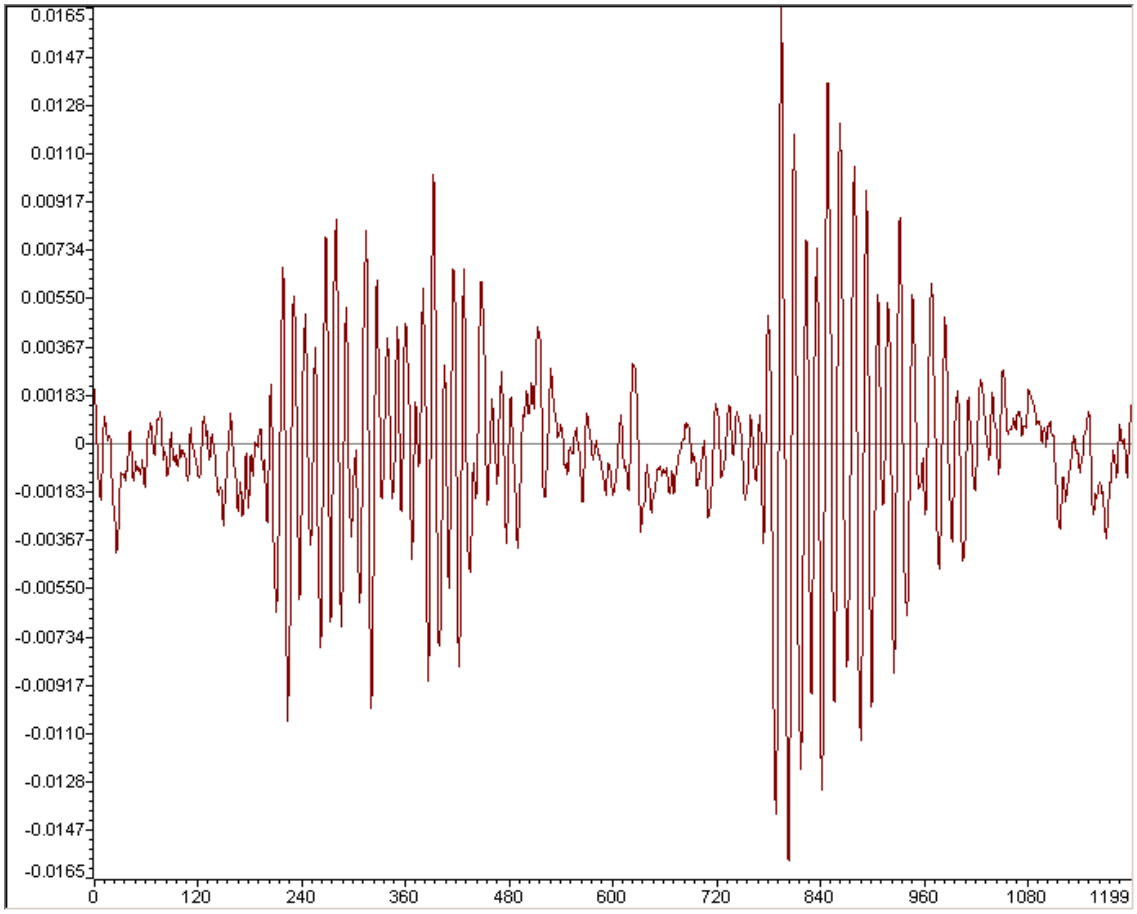


Figure 2-2: Signal After Bandpass Filter

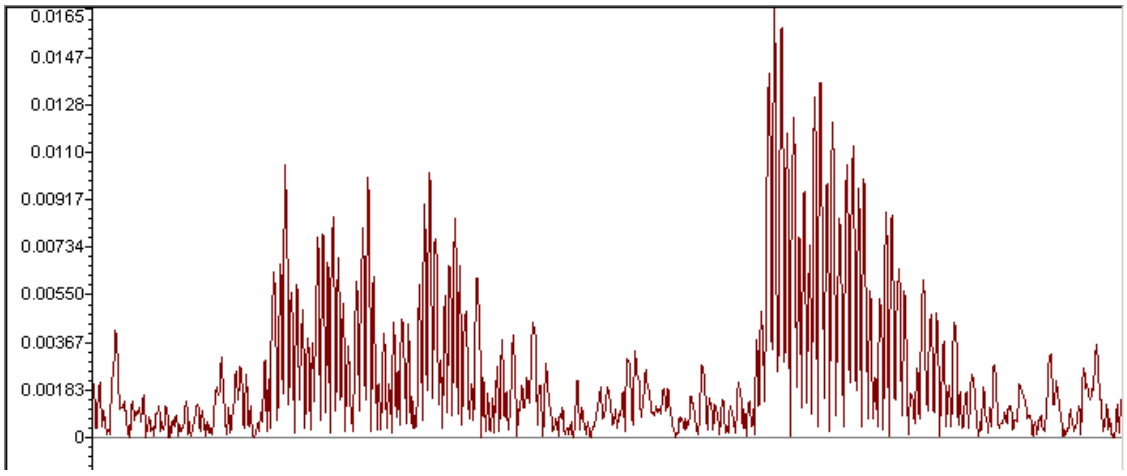


Figure 2-3: Envelope Detection

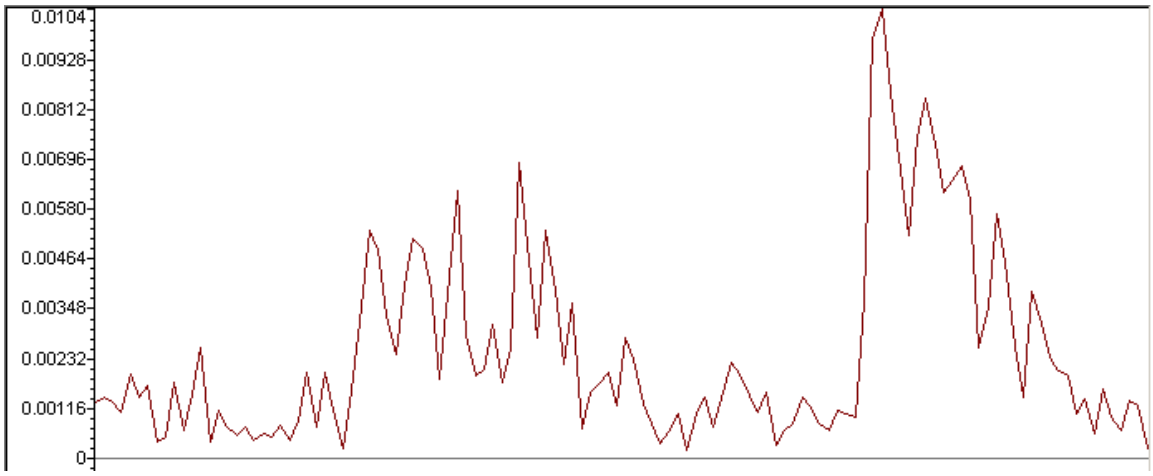


Figure 2-4: Decimation

2.1.4 Hanning Window

Before the Hanning window is applied to the data there is a window buffer function which takes advantage of the specialized hardware on the DSP to handle circular buffers. A circular buffer is able to loop through the same memory space without shifting data. It is a feature prevalent in DSPs and very handy when doing signal processing operations. The function keeps adding new data to the end of a buffer and outputs a linear array with the next set of data to run through the window. The window function simply multiplies all the values in the output by the Hanning Window coefficients. The coefficients for the Hanning Window were realized in Matlab and a plot of the window is shown in Figure 2-5. The data after it passes through the window is shown in Figure 2-7. The window contains about 8.5 seconds worth of footstep data and the sampling rate is now 100 Hz after the decimation by 10.

2.1.5 Fast Fourier Transform

The Fast Fourier Transform (FFT) routine was provided by Texas Instruments in their DSP Library. The project uses a 1024-point FFT which gives a frequency resolution of 0.098 Hz per bin. The output from the routine segments the numbers into real and imaginary components. The phase information is not useful for this application

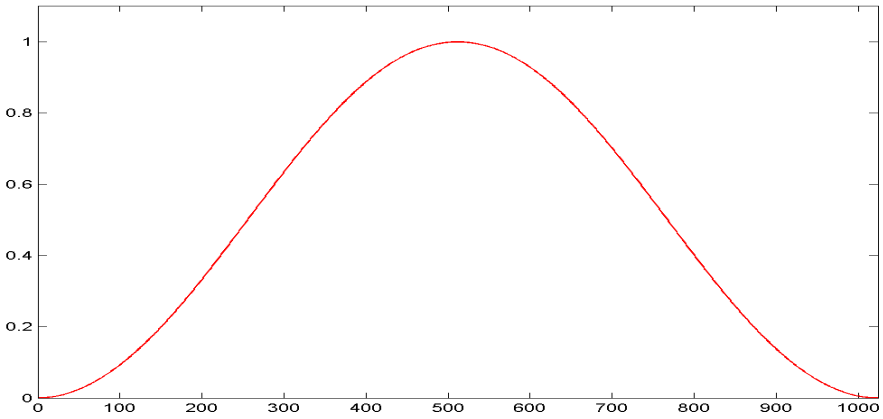


Figure 2-5: Hanning Window

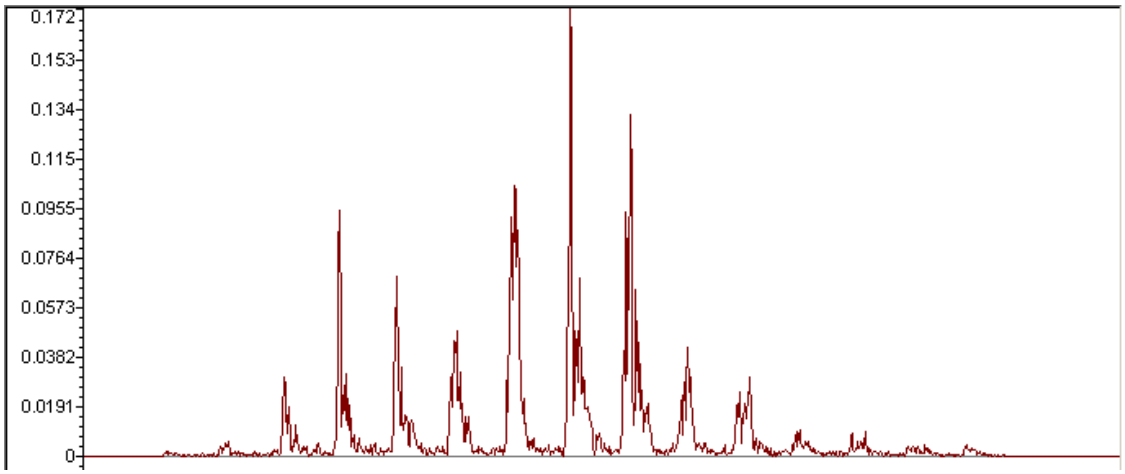


Figure 2-6: Hanning Window Applied to Footstep Data

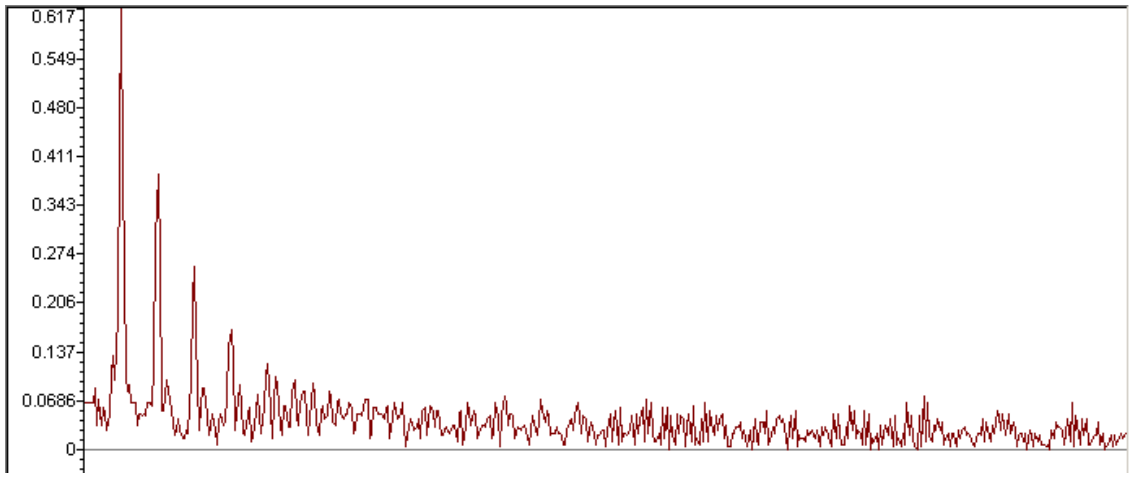


Figure 2-7: Scaled FFT Output

and can be disregarded. The `convert_to_mag function`, in Appendix F, converts the output from the FFT to a number representing the magnitude information. It computes the absolute value of the data, squares the real and imaginary component and then finds the square root of the number. The square root function was provided by Ken Turkowski [14]. The function computes the square root using only fixed-point numbers. After the data has been converted to a magnitude, the DC frequencies are removed from the spectrum and the data is scaled. The data after the FFT and scaling looks like Figure 2-7.

2.1.6 Spectral Normalization

It is very clear where the first, second and third harmonics are from the FFT output in Figure 2-7. The problem is this data are not normalized relative to the background level. In order to normalize the data a two-pass normalization is performed on the data. This calculates the average on both sides of a moving center point, and then calculates the mean within the center. Ideally, the peaks in the FFT will fit into the gap in the center. If the average of the center points divided by some threshold is above the average of the sides then the center value gets replaced by the average from the sides. It requires two passes because after the first pass a large peak will create

two smaller peaks in the output. The second pass helps to smooth out those smaller peaks so that when the operation is finished there is an array which approximates the background level of the spectrum. The width of the sides and the width of the center section can be adjusted in the `normalization.h` file. Currently, the length of the sides is 16 samples and the length of the center is 9 samples. Lastly, every point in the output of the FFT is subtracted by the background level multiplied by two.

2.2 Footstep Detection Algorithm

The footstep detection algorithm is based on the work by Kenneth Houston and Daniel McGaffigan [3]. Footstep data was collected from three days of testing in a few parks around Massachusetts. The data was recorded with a 16-channel acquisition system running into a laptop and a two channel portable DAT machine. A GPS receiver was also carried by the individual and the data was later downloaded to a computer to provide a ground truth. The test setup used to collect the data is described in further detail in Chapter 4. After collecting the data, Matlab was used to post-process the data and come up with the feature set used to distinguish footsteps.

The first criterion is that the frequency of the first harmonic for the footsteps be in the range of 1 Hz to 3 Hz. This is the frequency range of the impacts from a person's feet hitting the ground. It is remarkable how periodic the footsteps of most people are when they are walking normally. In addition to the frequency component, the peak also has to be greater than a certain threshold relative to the noise level. The next aspect of the detection algorithm is to verify that either a second or third harmonic exists and that one of them is above another threshold level. The algorithm finds the first five peaks in the range of 1 Hz to 10 Hz. Then it determines if the peaks it acquired meet the requirement to classify it as a footstep.

The digital bandpass filter at the beginning of the signal processing chain is very important in determining how well the algorithm performs detections and classifications. The environment and terrain can have a huge impact on the waveforms that are observed when a person walks on the ground. In future versions one would want

to consider an adaptive bandpass filter that automatically fine-tunes the frequency of its passband. Also, there could be intervention from the user to pick a specific land terrain for where the sensor will be operated.

Chapter 3

Design Description

The design of the footstep detector can be broken down into four major areas: analog design, digital design, RF design, and power design. The analog design includes all the electronics that are necessary to filter and amplify the small voltage signal coming from the geophone. The digital design contains the microcontroller and DSP, and all their supporting circuitry. These chips do the analog-to-digital conversion (ADC) and process the data to detect footsteps. The microcontroller also handles the user interface to control various settings. The RF design is mainly an off the shelf product which is used to allow multiple boards to communicate with each other or for the sensor to communicate with a base-station. Lastly, the power design contains all the circuity to support power from a battery, solar power or an external power supply. If the system were deployed in the field the ideal solution would be to have the solar power charge the batteries during the day and provide power for the system, and at night power the system via battery power. Also included in the power design is the power management scheme. Systems are shutdown when they are not in use and the processors are put into low-power modes when they are not actively processing data. These areas will be discussed more fully in the following sections.

3.1 Analog Design

The analog design has a separate power and ground plane from the digital area. The geophone provides a very small voltage that varies depending on the amount of seismic activity. This output needs to be amplified and filtered before it can be sampled by the ADC. A portion of the schematic which handles the analog frontend is provided in Figure 3-1 for reference. The sections will be discussed later.

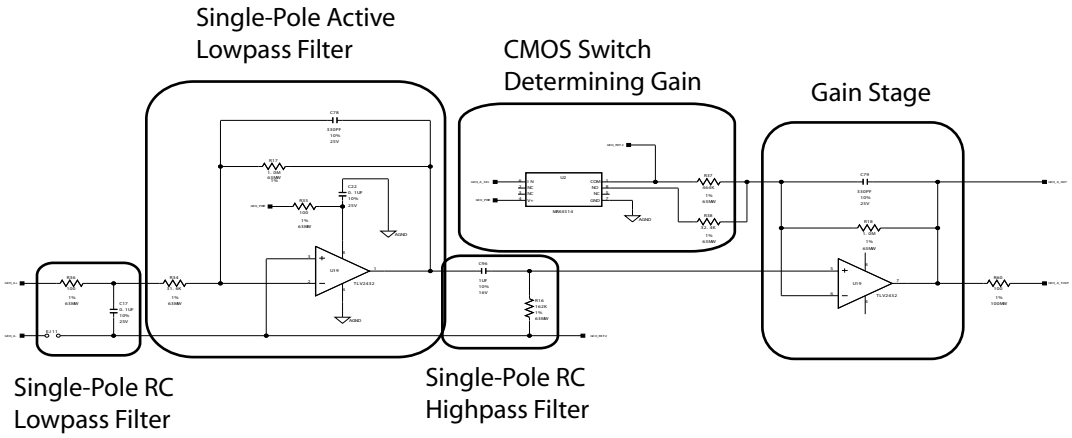


Figure 3-1: Analog Frontend Stages

3.1.1 Geophone

A geophone is a small instrument for measuring ground motion. It is part of a category of sensors called seismometers. Seismometers usually consist of a mass suspended by a spring, with the mass being either a magnet that moves within a moving coil, or a coil moving within the field of a fixed magnet. The geophone is an electromagnetic seismometer, which produces a voltage across the coil that is proportional to the velocity of the coil in the magnetic field, and thus approximately proportional to the velocity of the ground. There are two variants of geophones a one-axis version and a three-axis version. As the names imply, the one-axis version measures surface waves travelling in one axis while the three-axis version provides circuity to measure propagating waves in all three axes. For this application only a single axis geophone

is used, but the system is capable of handling a three-axis geophone.

3.1.2 Filtering

The sampling rate is set at 1 kHz. Therefore, to prevent aliasing the Nyquist frequency of 500 Hz must be observed. A lowpass filter with a cutoff of approximately 300 Hz will prevent any antialiasing. Two stages of single pole lowpass filters are provided, Figure 3-1. The first one is a simple RC filter with a cutoff of 15.92 kHz to prevent RF noise from getting into the analog inputs. The second lowpass filter is a single-pole active filter with a cutoff of 482 Hz, and is followed by another RC passive highpass filter with a cutoff of 0.98 Hz. The highpass filter prevents any DC signals from passing through and being amplified. The equations used to calculate the cutoff frequencies are shown below. Chapter 5 provides the actual measurements taken from the boards. Figure 3-2 shows the Matlab plot of the analog circuitry.

$$Lowpass\ 1 : f_{cutoff} = \frac{1}{2\pi * RC} = \frac{1}{2\pi * 100 * 0.1uF} = 15.92kHz \tag{3.1}$$

$$DCgain = 1 \tag{3.2}$$

$$Lowpass\ 2 : f_{cutoff} = \frac{1}{2\pi * R_2C} = \frac{1}{2\pi * 1Meg * 330pF} = 482Hz \tag{3.3}$$

$$DCgain = \frac{-R_2}{R_1} = -31.65 \tag{3.4}$$

$$Highpass\ 1 : f_{cutoff} = \frac{1}{2\pi * RC} = \frac{1}{2\pi * 162k * 1uF} = 0.98Hz \tag{3.5}$$

$$DCgain = 1 \tag{3.6}$$

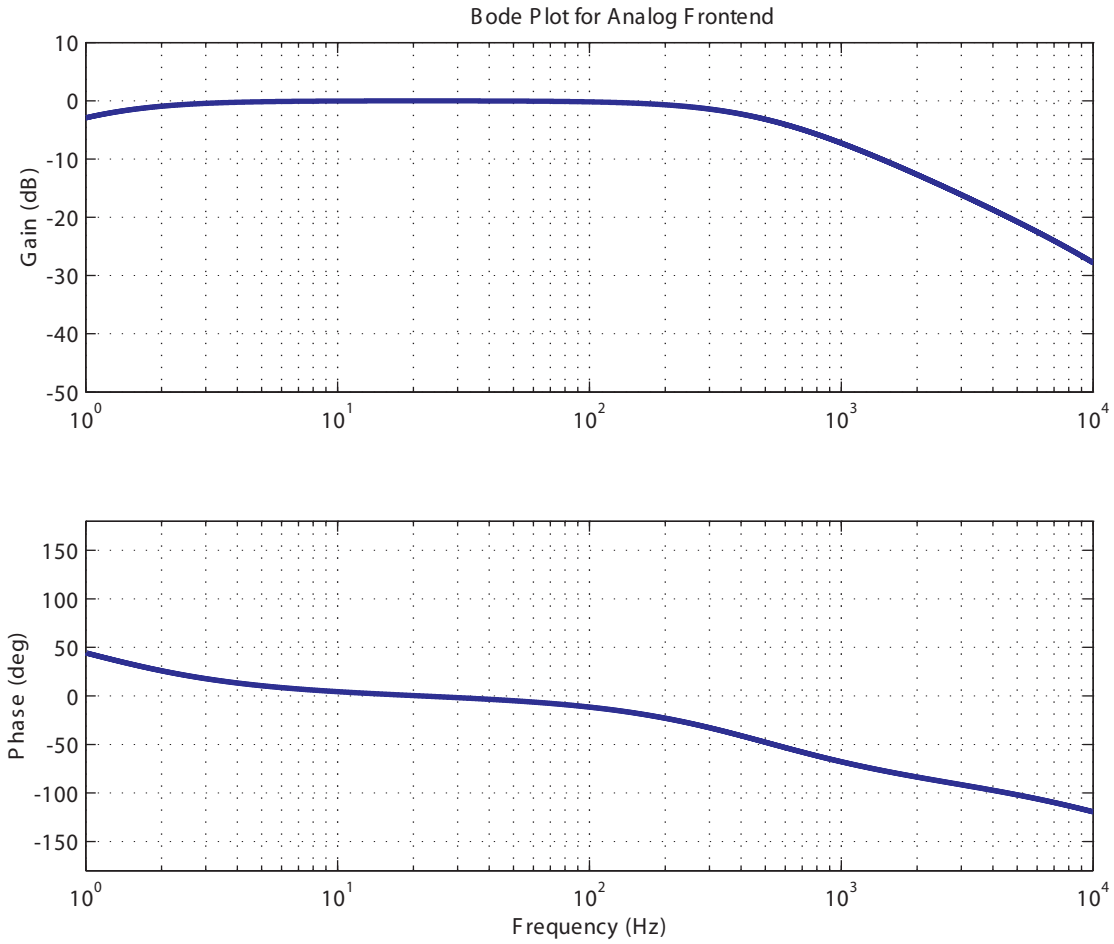


Figure 3-2: Bode Plot for Analog Frontend

3.1.3 Amplification

The amplification is through a single stage instrument amplifier, Figure 3-1. A low on-resistance CMOS analog switch is used to select whether a single 464 kΩ resistor or the 464 kΩ in parallel with a 32.4 kΩ resistor control the gain. The microcontroller is able to control the line `GEO_X_SEL` to determine which gain setting the amplifier is in. With only the 464 kΩ resistor there is 40 dB of gain, while with the two resistors in parallel the approximate resistance is the value of the lower one, 32.4 kΩ, and the gain is approximately 60 dB.

3.2 Digital Design

The digital design is comprised of the microcontroller, DSP, and all the supporting circuitry. The main functions of the microcontroller are to control the power to all the subsystems, change various settings within the subsystem, control the user interface, sample the data, and transfer the data to the memory of the DSP. The DSP processes the data and executes all the signal processing operations. After performing the detection algorithms, it reports to the microcontroller when a positive detection has been made.

3.2.1 MSP430F149 Microcontroller

The MSP430F149 (MSP) is a new generation of low-power microcontrollers developed at Texas Instruments. This microcontroller is able to run on less than a milliamp of power and has a whole host of features as well. There are numerous I/O ports on the microcontroller which allow control over all the subsystems, a built-in 12-bit ADC, and two UART ports for serial communications. In addition, the MSP has 60 KB of flash to store program code and 2 Kb of RAM.

The MSP is able to do the job that three or four ICs would perform in a traditional design. Since the number of digital I/O lines required for the design is relatively low, the MSP is able to manage the job of controlling digital lines that a separate FPGA

or CPLD would normally handle. The UART ports allow communications between the host computer and the RF unit. The flash memory is large enough to store all the MSP program code as well as all the program code for the DSP. The DSP would usually have its own flash chips to store its program code in, and external memory to process data. Having less components reduces the size of the system and more importantly the power consumption.

The MSP is able to address 64 kB memory locations. Table 3.1 shows the memory map for the MSP. The MSP program code occupies the memory address locations `0x1100-0x5FFF` about 20 kB of space, while the DSP occupies the flash memory locations `0x6000-0xC600` about 26 kB of space.

Memory Byte Address	Contents
`0x0000-0x01FF`	Registers
`0x0200-0x0FFF`	Ram
`0x1000-0x10FF`	Information Memory
`0x1100-0x5FFF`	MSP Program Code
`0x6000-0xC600`	DSP Program Code
`0xFFE0-0xFFFF`	Interrupt Vectors

Table 3.1: Microcontroller Memory Map

User Interface

The user interface allows the user to interact with the system and control various modes of operation. The commands are given over an RS232 serial link. Currently, the serial connection is set to run at 19200 bps with 8 data bits, no parity bit and one stop bit; however, those settings can easily be modified in the code.

When the system is first powered the user is presented with the main menu shown in Figure 3-3. The main menu has three standard sections. The header information for the project is shown on the top of every screen. A message area is used to display feedback to the user. The actual menu screen is in the center and allows the user to select various submenus, such as, the VC5509 menu, Geophone menu, and ADC menu. By pressing the letter associated with the menu the appropriate submenu is

brought up. These menus serve a dual purpose in the system. When the system was designed this is how the system was tested. As various subsystems were brought online they were tested by generating commands from the microcontroller to verify that they were functioning correctly. Now, the menus serve to test the subsystems again if something is not operating correctly and allow the user to make changes to the system settings.

The VC5509 menu, Figure 3-4, regulates when the DSP is booted, how code gets loaded into the DSP, and when signal processing is started. Also, this controls how the DSP code gets saved into the flash of the microcontroller. First, the code is loaded into a buffer location in the DSP memory and then it is transferred over to the flash of the microcontroller. Starting the signal processing operation turns on the geophone channel and starts the ADC on the microcontroller.

The Geophone menu, Figure 3-5, controls the geophone channels. The gain settings can be modified for any of the channels, and power to the analog subsystem can be turned on or off.

The ADC menu, Figure 3-6, allows the user to poll various channels on the ADC to find out their instantaneous values. Both the geophone and test input channels can be polled, and there is an internal temperature sensor in the microcontroller that can be checked.

The Misc menu, Figure 3-7, has a few miscellaneous features that were programmed in the system, but not really necessary for normal operation. The temperature can be checked and power supply voltage. The message area can be cleared, and the running time of the processor since it was powered on can be displayed.

Port Description

The MSP has up to 48 digital I/O lines to control various subsystems and interface with other components. Many of the ports on the MSP can be configured to perform special functions as well. For instance, the ports that are used to receive the ADC signals can either be selected as I/O ports or the inputs to the ADC. Obviously, when the ports are selected to perform special functions it reduces the number of I/O ports

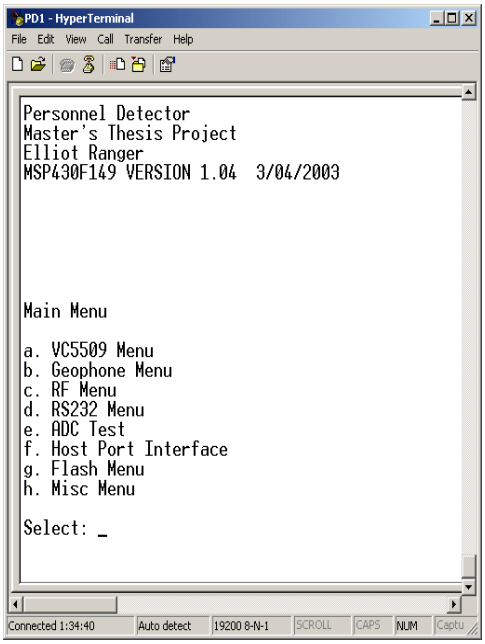


Figure 3-3: Main Menu

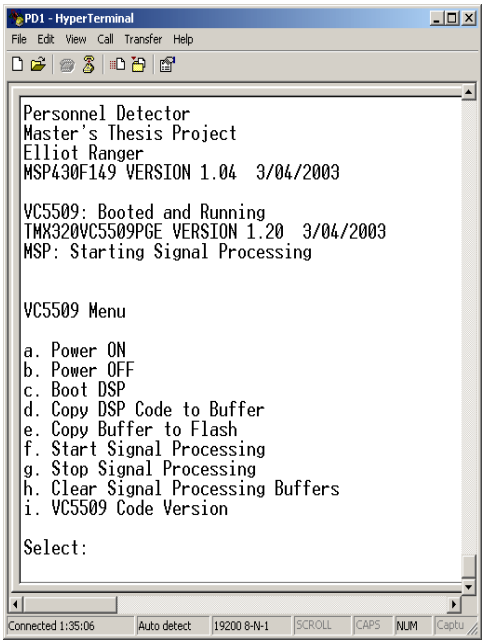


Figure 3-4: VC5509 Menu

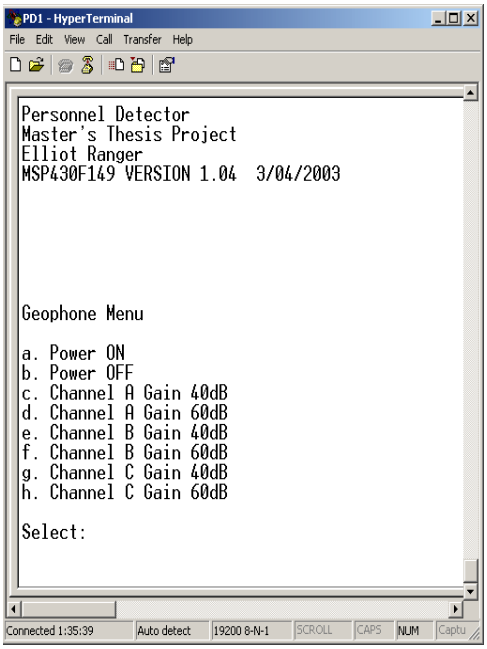


Figure 3-5: Geophone Menu

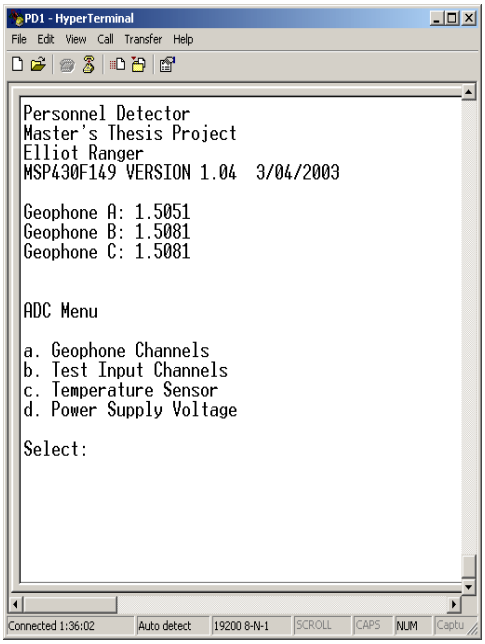


Figure 3-6: ADC Menu

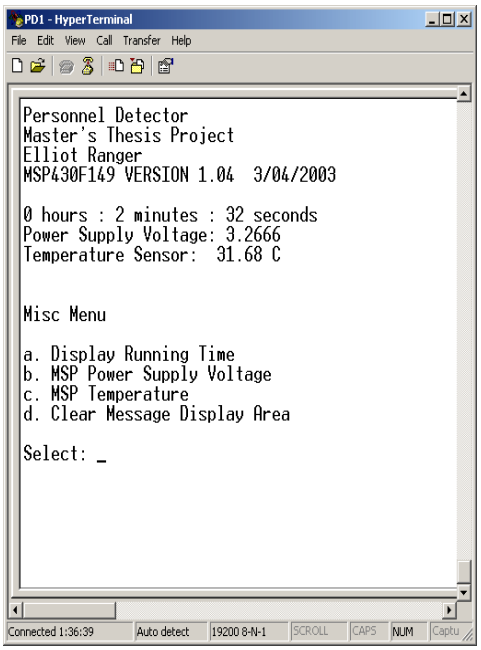


Figure 3-7: Miscellaneous Menu

available. All of the I/O ports can be configured to read or write to the port. The I/O ports are used to control the power for the DSP chip, seismic analog filtering and amplification, the RF power, and RS232 chip. Furthermore, they are used to transfer data back and forth between the DSP using the Host Port Interface (HPI).

The ports on the MSP are configured by writing to specific registers. Each port has its own register to control: whether it is a port or special function, and the direction of the port (read or write). There is a register to read the data from the port or write data to the port. In addition to all the above configurations, Port 1 and 2 on the MSP are also able to receive interrupts as well. The ports are all set to their initial values in the function `initialize_ports()` within the `initialize_system()` routine. Table 3.2 shows the initial values of all the registers associated with each of the ports. The PxSEL register determines if it is a special function (high) or the standard port (low). PxDIR sets the direction of the port as either an output (high) or input (low). PxIE enables the interrupts for the port. Interrupts can only be enabled on port 1 and port 2. PxIES determines which edge the interrupt should trigger on, the rising edge (low) or falling edge (high).

	7	6	5	4	3	2	1	0
PORT 1	HRDY	HDS1	HR/W	HCNTL1	HCNTL0	HBE1	HBE0	HINT
P1SEL	0	0	0	0	0	0	0	0
P1DIR	0	1	1	1	1	1	1	0
P1IE	0	0	0	0	0	0	0	0
P1IES	0	0	0	0	0	0	0	0
PORT 2	HP7	HP6	HP5	HP4	HP3	HP2	HP1	HP0
P2SEL	0	0	0	0	0	0	0	0
P2DIR	0	0	0	0	0	0	0	0
P2IE	0	0	0	0	0	0	0	0
P2IES	0	0	0	0	0	0	0	0
PORT 3	RF_RXD	RF_TXD	UI_RXD	UI_TXD	RF_CTS	RF_ENA	GEO_ENA	VC55_ENA
P3SEL	1	1	1	1	0	0	0	0
P3DIR	0	0	0	0	0	1	1	1
PORT 4	HP15	HP14	HP13	HP12	HP11	HP10	HP9	HP8
P4SEL	0	0	0	0	0	0	0	0
P4DIR	0	0	0	0	0	0	0	0
PORT 5	RF_PWRDN	ACLK	RS232_ENA	MCLK	NOT USED	GEO_C_SEL	GEO_B_SEL	GEO_A_SEL
P5SEL	0	1	0	1	0	0	0	0
P5DIR	1	1	1	1	1	1	1	1
PORT 6	TEST_IN3	TEST_IN2	TEST_IN1	GEO_C_OUT	GEO_B_OUT	GEO_A_OUT	NOT USED	NOT USED
P6SEL	1	1	1	1	1	1	1	1
P6DIR	0	0	0	0	0	0	0	0

Table 3.2: Initial Configuration of Port Registers

Clocks

There are three different clocks on the MSP. The master clock (MCLK) runs the CPU. The LXT2 oscillator is divided by 4 which gives a frequency of 1.23 MHz. During the low-power mode this clock gets shutdown. The subsystem clock (SMCLK) is used by the peripherals. This clock is the same frequency as the MCLK but does not get shutdown in the low-power modes. Lastly, there is the auxiliary clock (ACLK) which runs at 32,768 Hz. The ACLK is used to keep system time, and control slower events such as the sampling rate for the ADC.

Timers

There are two timers on the MSP, Timer A and Timer B. Timer A is used to keep the running time of the processor. It is connected to the ACLK which runs at 32,768 Hz and every 0x8000 ticks it generates an interrupt to add a second to the time. Timer

B is also connected to the ACLK but it is used to generate the sampling frequency for the ADC.

Analog-to-Digital Conversion

The ADC is used to convert the analog levels from the seismic sensor into digital values. The converter is a 12-bit successive-approximation converter so the digital values range between 0x0000 and 0x0FFF.

The sampling rate for the ADC is controlled by Timer B. This allows for a known sampling rate in the system. Whenever Timer B counts up to the number in its register it generates a pulse on the out signal. The ADC uses this pulse to trigger the next conversion. Timer B is connected to the ACLK which runs at 32,768 Hz therefore counting to 0x0020 gives a sampling rate of approximately 1 kHz and 0x0040 gives a sampling rate of 500 Hz.

The microcontroller has an internal temperature sensor that can be used to determine the temperature of the processor attached to one of the ADC inputs. Also, the microcontroller is able to determine the supply voltage.

Serial Interface

The serial interface communicates with the host computer through the RS232 connections and the RF module. The MSP has two built-in UARTs which facilitate serial communications. Both UARTs use the SMCLK as their source for baud rate generation. The SMCLK runs at the same frequency as the MCLK but is not shutdown during the low power mode.

Flash

The MSP has 60 KB of flash on-chip. Through control registers the MSP is able to not only read the flash memory contents but is able to erase and write to the flash memory directly as well. The voltage required to write to the flash is controlled internally and is transparent to the user. The flash memory is different from normal RAM in that a line cannot be reset high after it has had data written to it. Thus, in

order to reprogram a memory location it needs to be erased first and then it can be reprogrammed. When a memory location is erased all the bits get reset high. Also, an erase operation erases a whole segment of flash memory which is 255 bytes and not just a single word. It is important to have interrupts disabled while writing or erasing from flash.

JTAG Interface

The JTAG port is an IEEE Standard (1149.1-1990 Standard-Test-Access Port and Boundary Scan Architecture) that allows a host computer to program the MSP, set breakpoints in the code, and read the contents of registers and memory. It is crucial when debugging the software to use the debugger to get information about the MSP. The computer has a pod which connects to the parallel port of the computer to communicate with the processor.

3.2.2 TMS320VC5509 Digital Signal Processor

The DSP is the workhorse for the system. It is a fixed-point processor and performs all the signal processing algorithms. The DSP is underclocked to use less power and contains idle modes for low power consumption when it is not processing data. The microcontroller loads the data from the ADC directly into the DSP memory while it is in an idle mode. After it has finished loading 1.2 seconds worth of data it wakes up the DSP and instructs it to process the next data block. While the DSP is processing the block of data the microcontroller fills a different block of memory, and keeps alternating between the two memory locations. It takes approximately 200 ms for the DSP to process the data, and after finishing it returns to its idle mode.

A memory map for the DSP memory is shown in Figure 3.3. The DSP has Dual-Access RAM (DARAM) as well as Single-Access Ram (SARAM) built into the chip. The MSP can only access the lower 32 kB of DARAM through the HPI. All program code for the DSP is stored in SARAM beginning at memory address `010000`.

Memory Byte Address	Contents	Memory Word Address
0x0000C0-0x0000FF	Direct Access Variables	0x000060-0x00007F
0x000100-0x0001FF	Interrupt Vectors	0x000080-0x0000FF
0x000200-0x0002FF	HPI Read Area	0x000100-0x00017F
0x000300-0x0003FF	Boot Code	0x000180-0x0001FF
0x000400-0x0007FF	DARAM0 (stack)	0x000200-0x0003FF
0x000800-0x000BFF	DARAM1 (data, const)	0x000400-0x0005FF
0x000C00-0x0019FF	DARAM2 (bss)	0x000600-0x000CFF
0x001A00-0x007FFF	HPI Write Area	0x000D00-0x003FFF
0x008000-0x00BFFF	DARAM3 (filter,window data)	0x004000-0x005FFF
0x00C000-0x00DFFF	DARAM4 (delay buffers)	0x006000-0x006FFF
0x00E000-0x00FFFF	DARAM5	0x007000-0x007FFF
0x010000-0x013FFF	SARAM0 (program code)	0x008000-0x009FFF
0x014000-0x017FFF	SARAM1 (coefficients)	0x00A000-0x00BFFF
0x018000-0x01BFFF	SARAM2 (frequency data)	0x00C000-0x00DFFF

Table 3.3: DSP Memory Map

Bootloader

As stated before, the code for the DSP is actually stored in the flash memory of the microcontroller. The MSP indicates to the DSP that it should copy all the program code into the HPI Write area which the MSP can access. The HPI Write area contains the code blocks shown in Table 3.4. The General Purpose I/O pins on the DSP control which configuration the processor boots up in. It is configured to boot from the Host Port Interface in its current configuration. The ROM bootloader sets the initial state of the CPU registers and then keeps the DSP in a loop while the microcontroller loads the DSP memory with the code. After the microcontroller has finished, it sets register 0x0061 with 0x0300, the memory location for the bootloader, and register 0x0060 with 0xFF00 the sentinel that the DSP recognizes to begin executing instructions at the memory address specified by register 0x0061.

DSP Algorithms

The DSP performs all the functions that were described in the last chapter. First it shifts the data down to take out the DC offset from the ADC. This is a simple subtraction of a constant offset from the data. Next it performs a highpass and

Memory Byte Address	Contents	Memory Word Address
0x001A00-0x001AFF	Boot Code	0x000D00-0x000D7F
0x001B00-0x001BFF	Interrupt Vectors	0x000D80-0x000DFF
0x001C00-0x001DFF	DARAM1 (data, const)	0x000E00-0x000EFF
0x001E00-0x002BFF	DARAM2 (bss)	0x000F00-0x0015FF
0x002C00-0x005BFF	SARAM0 (program code)	0x001600-0x002DFF
0x005C00-0x0070FF	SARAM1 (coefficients)	0x002E00-0x00387F

Table 3.4: HPI Write Memory Map for Saving Program Code

lowpass single order IIR filter to remove the frequency content that is not needed for the algorithm. After that an envelope detect is performed. The data is decimated to focus in on only the envelopes associated with footsteps. Then the data is passed through a moving window. Next an FFT routine is performed and the output gets converted to a signed-magnitude number. A two pass normalization is performed and the background level is subtracted from the data. Lastly, the detection routine is performed.

The DSP uses different memory locations to process the data. The initial values after they are received from the MSP are stored in DARAM in the HPI Write area. All the initial routines are performed on the data in the same memory location. After the decimation, the data is added to a circular buffer. This circular buffer discards the oldest data values and adds the new data values to the end of the buffer. The buffer is stored in DARAM4 memory. The output from the window function is stored in DARAM3. The FFT converts the data to the frequency domain and then uses SARAM2 to convert the numbers to a real values for the 2-pass normalization. The detection routine is performed in this same memory location. All the coefficients for the various filters are stored in SARAM1.

JTAG Interface

The JTAG port for the DSP is very similar to the one on the MSP. It allows Code Composer, the programming suite developed by Texas Instruments: to download code to the processor, set break points, and view memory contents. The pod connects to

the JTAG port on the processor and connects to the computer through the parallel port.

3.2.3 Host Port Interface

The Host Port Interface is the set of communication procedures used to facilitate communication between the two processors. The interface utilizes 16 data lines to send and receive complete words in one transfer and 8 control lines to regulate communications between the two processors. The microcontroller is able to read and write directly to the first 32 kB of memory in the DSP. This memory is dual-access memory so two buses can read from the same location concurrently. Obviously only one bus can write to a location at a time. There are specific locations setup in the DSP memory for reading and writing instructions to the microcontroller. The microcontroller writes commands in the allocated memory location in the DSP and then initiates an interrupt when the DSP is supposed to acknowledge the next command. The DSP has a similar communication for communicating information back the MSP. The most commonly used command from the DSP is the one to print a message to the user. During a print message command, the DSP packs the characters into words in the HPI Read section and then sends the print command to the microcontroller. The beginning of the array tells how many characters are in the string.

COMMAND	CODE
PRINT_MESSAGE	0x0100
STOP_PROCESSING	0x0200
PROCESS_DATA_BLOCK_1	0x0100
PROCESS_DATA_BLOCK_2	0x0100
CLEAR_PROCESSING_BUFFERS	0x0300
GET_VC5509_VERSION	0x0900
COPY_TO_HPI_WRITE	0x0999

Table 3.5: HPI Command Codes

3.3 RF Design

The RF is controlled by the microcontroller through the second UART port. This port is programmed to run at 9600 bps, but can easily be increased in the software. The RF module is made by Maxstream and it transmits and receives in the 900 MHz band. It provides a half duplex virtual serial link with 100mW(+20 dBm) transmit power and -110 dBm receiver sensitivity. The module has its own buffering scheme and puts itself into a low power mode when it is not transmitting or receiving any characters. The microcontroller simply sends the characters to the module through an RS232 port and the Maxstream transparently transmits them to the other modules within range. If the Maxstream is not able to correct errors from the transmission the characters get discarded. It is up to the application to send or receive acknowledgements and retransmit if necessary.

Currently, the RF subsystem has only been used to echo the data that is appearing on the RS232 output. It can also read in responses from a remote terminal to modify settings within the system. In future designs the RF port could be used to network a number of sensors together. It could communicate the information in a network and multiple sensors could be used to either triangulate on a certain position or else track a person the whole way through the range of all the sensors.

3.4 Power Design

The power design is very critical for the sensor. In the field the system should run for a long time without the need for human intervention. The system uses solar power to run the system and charge the batteries during the day and automatically switches to battery power when the solar power has diminished below usable levels. The voltage coming out of the solar panels changes depending on how much sunlight is striking the panels. This voltage is regulated to 9.1 V with a buck converter and provides the power to charge the batteries and power the system. Two more buck/boost converters provide the power for the 3.3 V power plane and 5 V RF power. The system as it is

currently designed would have a lifetime dependent only on how often the batteries need replacing from the constant charging and discharging.

3.4.1 Solar Power

The solar panels incorporated in the design were manufactured by Siemens, and use thin film technology based on Copper Indium Diselenide (CIS). They are able to produce battery charging levels even in low-light environments. Additionally this is a commercially available part in a black-anodized aluminum frame with a glass covering to protect the cells. A picture of the solar panel is shown in Figure 3-8. Unfortunately, the size of the panel is rather large for a sensor application (12.9" by 8.1"). The solar panels produced 2.5 W of power even on cloudy days.

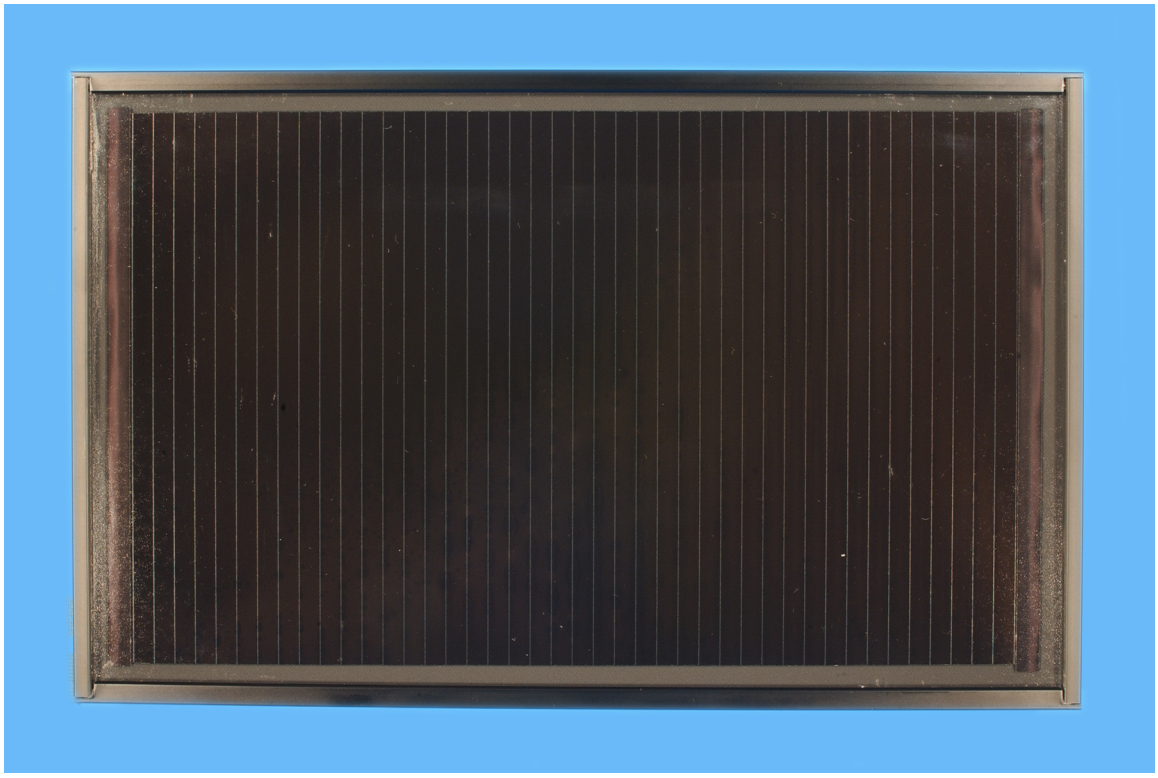


Figure 3-8: Solar Panel

3.4.2 Battery Charger

The battery charger is a Maxim part that is able to regulate the current going to the batteries. It charges up the batteries and then shuts off when it senses that the batteries are charged. It is also able to provide power to a load while it is charging. This is used to run the system while the battery charger is charging the batteries. Four AA NiMH batteries are used in the system to give a nominal voltage of 4.8 V on the power bus when it is running off battery power.

3.4.3 External Power

External Power can be supplied to the board as well. This is mainly for lab environments and during testing. The voltage for the supply should be 6V and there is diode protection circuitry to protect against reversed supply connections.

Chapter 4

Test Procedure

In order to ensure that the system was designed and working correctly, various levels of tests were performed. A bottom-up test procedure was employed to verify that each component by itself was operating correctly and then integration tests were performed to confirm that the whole system was functioning correctly.

4.1 Unit Level Testing

A complete test procedure was written to validate all the components of the system after the board was fabricated. The procedure examines all the power and ground connections to make sure there are no short circuits. Next, signals controlled by the microcontroller are tested. Last are basic functional tests to achieve communication with the subsystems and communications between the two processors through the HPI. The procedure is shown in Appendix D.

After the hardware was debugged, unit-level testing was performed on all the signal processing routines. They were each run individually with simulated test data. Then the same data was put into Matlab and the routines were verified against the results from Matlab for their accuracy.

4.2 System Testing

In addition to the testing for the signal processing routines, Matlab scripts were developed which generated simulated footstep data. These scripts were used to perform system level testing. The data from the scripts was furnished directly to the ADC of the microcontroller without any frontend analog processing. At this point the signal processing routines were debugged until they were able to process the data and give the expected results. At each stage in the signal processing chain the data could be verified against the Matlab results. When the signal processing functions were at a sufficient level then actual data, recorded on DAT, was played into the system through the sensor input. The DAT stores the time of day when the recording was made and that was used to determine the distance from the sensor via the GPS data.

4.2.1 Data Collection

Three days were spent collecting footstep data that was used to develop the algorithms and later test the system. The data was recorded on both a Sony portable DAT machine and a 16 port data acquisition system connected to a laptop. The course had 9 single-axis geophone senors and 2 three-axis geophone senors. It extended for 100 meters on each side of the sensor array. A graphic showing the configuration of the array is shown in Figure 4-1.

The test sequence involved the subject conducting two normal walks through the course, a stealthy walk, and then a jog through the course. There were waypoints along the way, to coordinate where the person was on the course. As the person crossed each waypoint they announced over the radio which waypoint they were passing. A GPS receiver was worn by each person as they went through the course and it recorded the GPS coordinates of the person every two seconds. This data later became the ground truth that was used to determine the effective ranges for detection. A background run was also performed at each location to determine the ambient background level. Lastly, for two of the locations horse data was collected to see if modifications could be made to the algorithm to discriminate a person from

a horse.

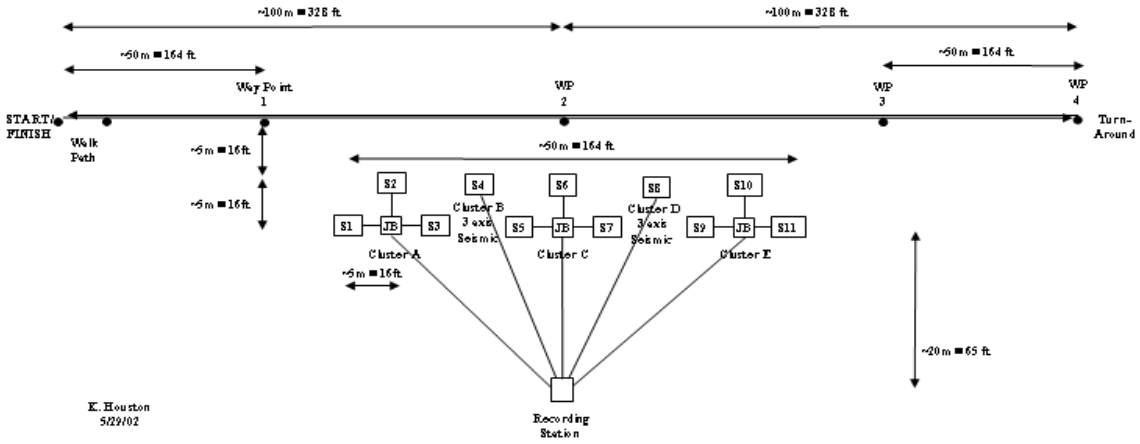


Figure 4-1: Footstep Data Collection Setup

Chapter 5

Results

Overall the system performed quite well. Considering the scope of the project, it is amazing that the system worked at all. The detection ranges ended up not being as far as originally hoped for, however, it still performed well. Typical detection ranges were on the order of 10 meters depending on the environmental conditions and the weight of the person. Having lower noise in the system would allow the system to differentiate footsteps from further distances.

5.1 Detection Performance

The detection ranges for a number of the test runs are shown in Table 5.1. These tests were performed at three different locations in Massachusetts and a couple of different types of walks were used. The first location was Great Brook Farm State Park (GB). Only one location was used at Great Brook and the terrain was an open field with a dirt path. The next location was at Harold Parker State Forest (HPSF). Two locations were used at Harold Parker: a dirt path through a wooded area, and a dense foliage area that was on a hill. Lastly, a location in Acton was used to collect horse data. Each run was analyzed three times by the system to determine detection ranges. The sensor in the middle of the array was the one used for all the tests. Also, the number of false positives was recorded as well. The subject data is recorded in Table 5.2. The environmental aspects of the area play a tremendous role

in the detection ranges of the system. For some of the runs, the system was not able to detect the person until they were right at the sensor, and in other locations the system could detect footsteps from 35 meters away. There is system lag due to the length of the FFT as well. The FFT represents 8.5 seconds of data which suggests why some detections were made after the subject passed by the sensor. Also, the sensor array is 5 meters offset from the trail. The way the person walks also plays a role in the detection ranges of the system. If the person walks in a stealthy fashion then it is very difficult to pick up the footsteps until they are right at the sensor. Unfortunately, the frequency spectrum collected from the horse data is very similar to the spectrum from people and it is very difficult to differentiate a horse from a person with the current algorithms.

Location	Track #	Subject	Type	Avg. Speed	Detection	False Positives
GB1	2	1	Normal	1.85 m/s	7.4 m	0
GB1	2	1	Normal	1.85 m/s	7.4 m	0
GB1	2	1	Normal	1.85 m/s	7.4 m	0
GB1	3	1	Normal	1.72 m/s	8.6 m	0
GB1	3	1	Normal	1.72 m/s	8.6 m	2
GB1	3	1	Normal	1.72 m/s	6.9 m	0
GB1	4	1	Normal	1.72 m/s	6.9 m	0
GB1	4	1	Normal	1.72 m/s	6.9 m	1
GB1	4	1	Normal	1.72 m/s	5.2 m	0
GB1	5	1	Stealthy	1.39 m/s	4.2 m	6
GB1	5	1	Stealthy	1.39 m/s	4.2 m	5
GB1	5	1	Stealthy	1.39 m/s	4.2 m	6
GB1	6	1	Jog	3.33 m/s	6.7 m	0
GB1	6	1	Jog	3.33 m/s	0 m	0
GB1	6	1	Jog	3.33 m/s	0 m	2
GB1	8	4	Normal	1.85 m/s	0 m	0
GB1	8	4	Normal	1.85 m/s	3.7 m	0
GB1	8	4	Normal	1.85 m/s	3.7 m	0
GB1	9	4	Stealthy	1.47 m/s	2.9 m	1
GB1	9	4	Stealthy	1.47 m/s	1.47 m	3
GB1	9	4	Stealthy	1.47 m/s	2.9 m	2
GB1	10	4	Jog	3.85 m/s	15.4 m	2
GB1	10	4	Jog	3.85 m/s	26.9 m	3
GB1	10	4	Jog	3.85 m/s	23.1 m	3
GB1	11	2	Normal	1.47 m/s	13.2 m	2
GB1	11	2	Normal	1.47 m/s	11.8 m	1
GB1	11	2	Normal	1.47 m/s	11.8 m	1
GB1	12	2	Normal	1.47 m/s	10.3 m	2
GB1	12	2	Normal	1.47 m/s	10.3 m	1
GB1	12	2	Normal	1.47 m/s	10.3 m	6
GB1	13	2	Stealthy	1.47 m/s	3.1 m	4
GB1	13	2	Stealthy	1.47 m/s	4.2 m	9
GB1	13	2	Stealthy	1.47 m/s	3.1 m	7
GB1	16	2	Jog	3.85 m/s	11.5 m	1
GB1	16	2	Jog	3.85 m/s	23.1 m	0
GB1	16	2	Jog	3.85 m/s	23.1 m	0
GB1	18	3	Normal	1.56 m/s	-1.6 m	0
GB1	18	3	Normal	1.56 m/s	-3.1 m	2
GB1	18	3	Normal	1.56 m/s	-3.1 m	0

Location	Track #	Subject	Type	Avg. Speed	Detection	False Positives
GB1	19	3	Normal	1.72 m/s	5.2 m	0
GB1	19	3	Normal	1.72 m/s	5.2 m	0
GB1	19	3	Normal	1.72 m/s	3.4 m	0
GB1	21	3	Jog	3.13 m/s	9.4 m	2
GB1	21	3	Jog	3.13 m/s	0 m	1
GB1	21	3	Jog	3.13 m/s	9.4 m	0
HPSF1	1	2	Normal	1.47 m/s	22.1 m	1
HPSF1	1	2	Normal	1.47 m/s	22.1 m	0
HPSF1	1	2	Normal	1.47 m/s	22.1 m	0
HPSF1	4	2	Normal	1.67 m/s	21.7 m	0
HPSF1	4	2	Normal	1.67 m/s	20.0 m	0
HPSF1	4	2	Normal	1.67 m/s	20.0 m	2
HPSF1	5	2	Stealthy	0.89 m/s	8.0 m	2
HPSF1	5	2	Stealthy	0.89 m/s	8.9 m	4
HPSF1	5	2	Stealthy	0.89 m/s	8.9 m	3
HPSF1	6	2	Jog	2.94 m/s	23.5 m	0
HPSF1	6	2	Jog	2.94 m/s	20.6 m	1
HPSF1	6	2	Jog	2.94 m/s	20.6 m	1
HPSF1	7	1	Normal	1.67 m/s	35.0 m	0
HPSF1	7	1	Normal	1.67 m/s	31.7 m	1
HPSF1	7	1	Normal	1.67 m/s	31.7 m	0
HPSF1	9	1	Normal	1.56 m/s	20.3 m	2
HPSF1	9	1	Normal	1.56 m/s	21.9 m	0
HPSF1	9	1	Normal	1.56 m/s	23.4 m	1
HPSF1	11	1	Jog	2.78 m/s	30.6 m	0
HPSF1	11	1	Jog	2.78 m/s	27.8 m	0
HPSF1	11	1	Jog	2.78 m/s	30.6 m	0
HPSF1	13	3	Normal	1.56 m/s	14.1 m	1
HPSF1	13	3	Normal	1.56 m/s	12.5 m	0
HPSF1	13	3	Normal	1.56 m/s	21.9 m	1
HPSF1	14	3	Normal	1.47 m/s	7.4 m	1
HPSF1	14	3	Normal	1.47 m/s	8.8 m	1
HPSF1	14	3	Normal	1.47 m/s	5.9 m	1
HPSF1	15	3	Normal	0.81 m/s	1.6 m	3
HPSF1	15	3	Normal	0.81 m/s	2.4 m	2
HPSF1	15	3	Normal	0.81 m/s	2.4 m	3
HPSF1	16	3	Jog	3.33 m/s	16.7 m	1
HPSF1	16	3	Jog	3.33 m/s	13.3 m	1
HPSF1	16	3	Jog	3.33 m/s	16.7 m	1

Location	Track #	Subject	Type	Avg. Speed	Detection	False Positives
HPSF2	1	2	Normal	1.52 m/s	3.0 m	0
HPSF2	1	2	Normal	1.52 m/s	0.0 m	0
HPSF2	1	2	Normal	1.52 m/s	0.0 m	0
HPSF2	2	2	Normal	1.52 m/s	0.0 m	0
HPSF2	2	2	Normal	1.52 m/s	1.5 m	0
HPSF2	2	2	Normal	1.52 m/s	1.5 m	0
HPSF2	5	1	Normal	1.52 m/s	-1.5 m	0
HPSF2	5	1	Normal	1.52 m/s	-3.0 m	0
HPSF2	5	1	Normal	1.52 m/s	-1.5 m	0
HPSF2	6	1	Normal	1.47 m/s	0.0 m	0
HPSF2	6	1	Normal	1.47 m/s	0.0 m	0
HPSF2	6	1	Normal	1.47 m/s	-1.5 m	0
HPSF2	9	1	Jog	2.94 m/s	0.0 m	0
HPSF2	9	1	Jog	2.94 m/s	0.0 m	0
HPSF2	9	1	Jog	2.94 m/s	0.0 m	0
HPSF2	10	3	Normal	1.56 m/s	0.0 m	0
HPSF2	10	3	Normal	1.56 m/s	-1.6 m	0
HPSF2	10	3	Normal	1.56 m/s	-1.6 m	0
HPSF2	11	3	Normal	1.56 m/s	-1.6 m	0
HPSF2	11	3	Normal	1.56 m/s	-1.6 m	0
HPSF2	11	3	Normal	1.56 m/s	-3.1 m	0
ACTON1	3	5	Normal	1.79 m/s	3.6 m	0
ACTON1	4	5	Normal	1.85 m/s	-1.9 m	0

Table 5.1: Detection Data

Number	Height (inches)	Weight (lbs.)	Type
1	70	180	Male
2	72	230	Male
3	72	180	Male
4	67	135	Male
5	N/A	N/A	Horse

Table 5.2: Subject Data

5.2 Analog Performance

A Hewlett Packard Dynamic Signal Analyzer was used to measure the frequency response, phase response, and noise floor of all the analog channels of both boards produced. Only one channel of data is included for each board, but the tests were run on all the channels. The results for the other channels are within acceptable tolerances of the first. The values are lower than 40 dB and 60 dB because the signal is decreased by the active filtering stage. Also, the gain-bandwidth product of the op-amps is only 500 kHz.

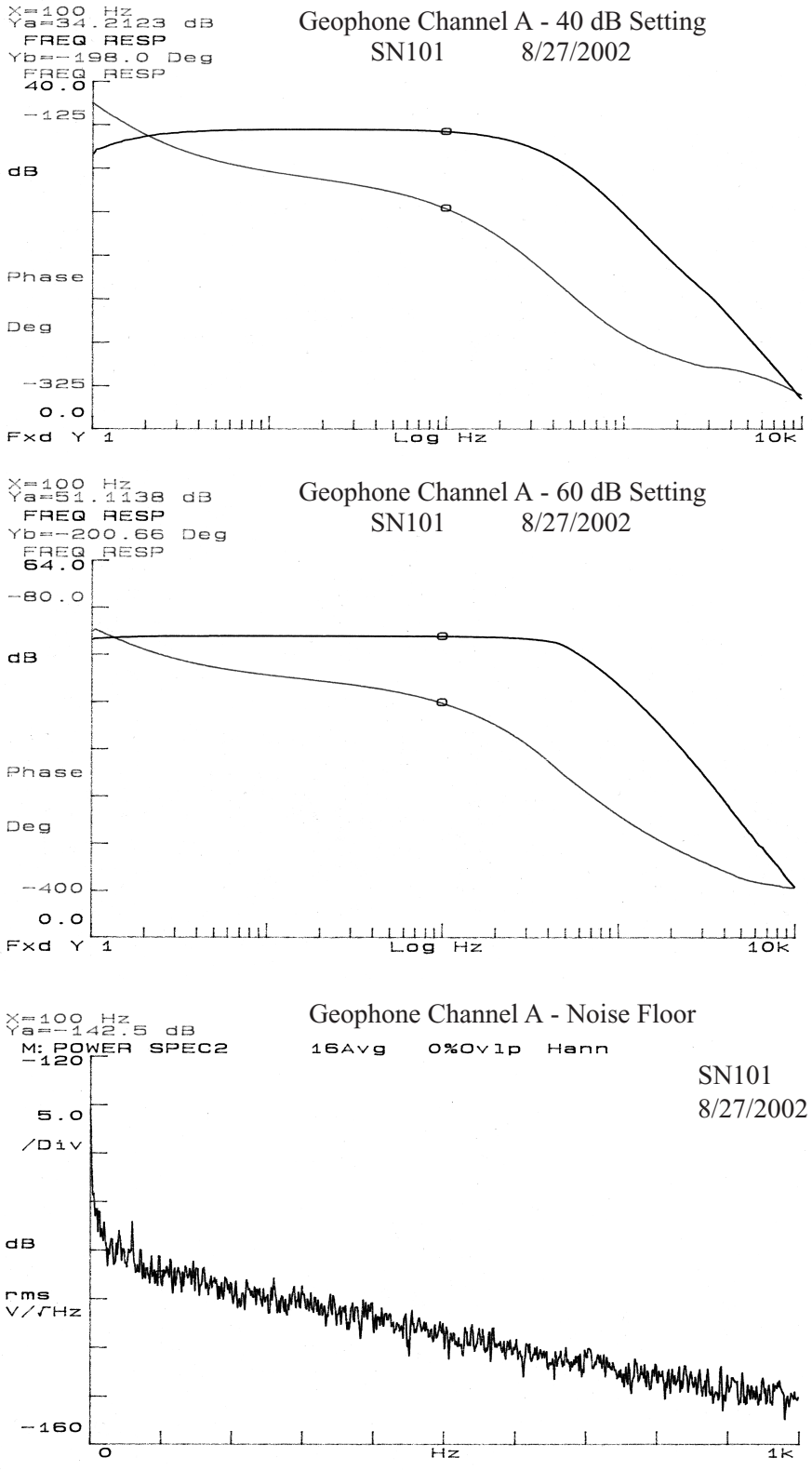


Figure 5-1: Magnitude, Phase and Noise Plots for Board SN101

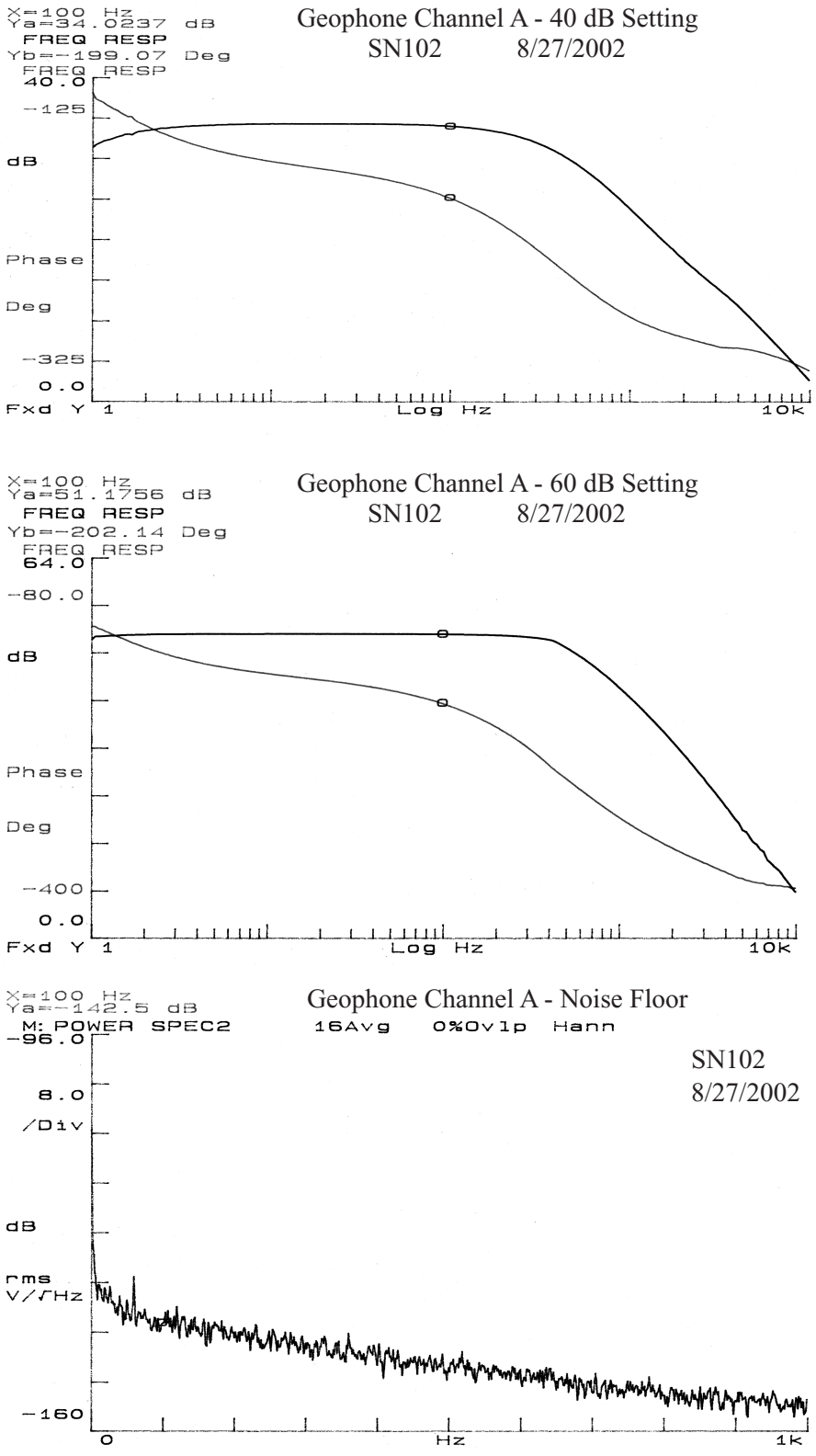


Figure 5-2: Magnitude, Phase and Noise Plots for Board SN102

5.3 Solar Power

The solar panels were taken outside and measured at various points in the day with different cloud cover. Even on cloudy days the solar panels were able to put out approximately 2.5 W of power. As the table in the power section will show, even with everything turned on and the RF running, the system only requires about 400 mW of power. This means the solar panels should be more than adequate to run the system and power the batteries at the same time even on cloudy days or in the winter time. Obviously, this data is for the Boston area and other locations would have different solar intensities.

5.4 Power Usage

The power usage is shown in Table 5.3. The microcontroller is always running because it controls the other subsystems. The RF requires a significant amount of power to transmit and receive signals. Perhaps a system where the RF unit was only transmitting and receiving at set intervals would help lower the power consumption because the RF subsystem could be shutdown when not in use.

Configuration	Current	Voltage	Power
MSP Running, DSP Off, ADC Off, RF Off	192 uA	6.0138 V	1.15 mW
MSP Running, DSP Idle, ADC Off, RF Off	4.37 mA	6.0138 V	26.28 mW
MSP Running, DSP Idle, ADC On, RF Off	5.13 mA	6.0138 V	30.85 mW
MSP Running, DSP Processing, ADC On, RF Off	9.27 mA	6.0138 V	55.75 mW
MSP Running, DSP Idle, ADC Off, RF On	51.92 mA	6.0138 V	312.24 mW
MSP Running, DSP Processing, ADC On, RF On	61.63 mA	6.0138 V	370.63 mW

Table 5.3: Power Usage

5.5 RF Range

The Maxstream 9Xtream-96 provides a 9600 bps serial link. The range was tested by setting up two transceivers and moving one of them successively further away from the other. The Maxstream does not try to resend messages that it cannot correct

the errors in, instead, it just discards the messages. Typical ranges for 100% packet transmission were on the order of 300-500 meters when the radios were placed on the ground like they would be on the sensor. Using a half-wave antenna it was possible to attain ranges of 750 meters.

Chapter 6

Conclusion

Overall this project has had very positive results. The project was a massive undertaking for one person to conduct. A picture of the final prototype board that was produced is shown in Figure 6-1. The design involved many aspects including the requirements, schematic capture, board layout, creating a test plan, debugging components, and system integration. Furthermore, the documentation aspect has been a critical component as well. Designers these days do not work in a little box. Designs get revised at later dates and clear documentation becomes necessary to allow other designers to pick up where you left off. It is my desire that if at a later point someone is interested in continuing to work on this project the time it will take to get familiar with the project will be greatly reduced as a result of this documentation.

The design contained a number of areas which had a higher chance of failure in the system. It is fun to use the cutting-edge technology, but there are some pitfalls which come with that as well. For instance, the DSP was a pre-release part. There were many functions that were not documented yet and they way they worked had to be discovered through experimentation. Also, the datasheet did not contain accurate information on some of the aspects of the chip. The lack of any extra memory to boot the DSP was another risk. If time permitted it would be better to include the memory and also work on setting up the DSP to boot from MSP. Then when the system was past the prototyping stages the memory could be removed. Fortunately, everything ended up working in the end, but in the future a more conservative approach should

probably be taken. Usually, a project of this scale requires at least two revisions. A first cut prototype version to make sure everything works as expected and then the final design. True to most projects, the funds were limited and there was not enough money to fabricate two boards.

The amount of knowledge gained from this project is indescribable. The design had to be taken all the way from the initial design stages to the final tests. Having never actually routed and laid out the parts on a board before, a lot of learning was associated with that. Also, my background is more in hardware design and most of the signal processing routines required some learning. The results achieved are very notable, but there will always be room for improvement.

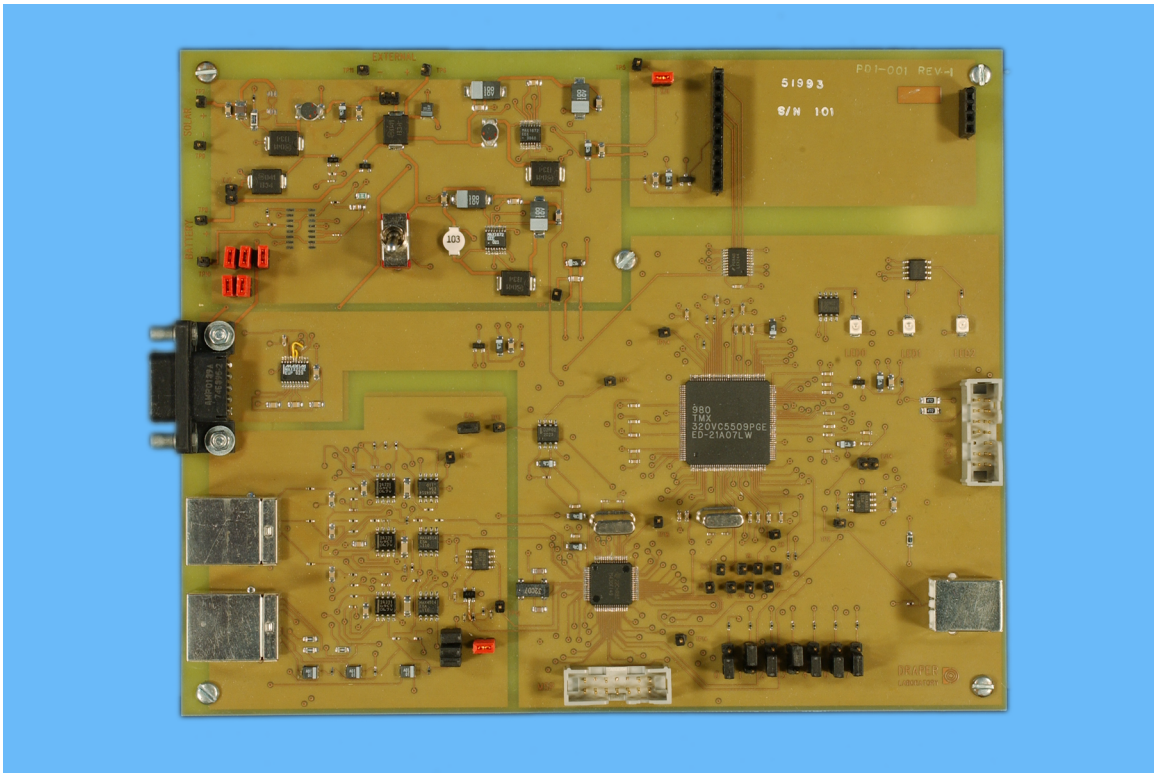


Figure 6-1: Prototype Board

6.1 Future Enhancements

There are endless possibilities for this system from intruder detection around a home to earthquake detections. Future versions will probably want to investigate a smaller package for the system. Ideally, the system should fit into a package right in the sensor.

Another area that would be interesting to explore is to see if it is possible to use the RF modules to communicate with other nearby systems and then triangulate the bearing of the person being tracked based on the information from several sensors. These could be used to follow a person through an entire array of sensors.

Hopefully, as time progresses solar panels will become more efficient and cheaper. Currently, the number of solar cells needed to power the system makes for a rather large array, but as new photovoltaic technologies are explored no doubt the arrays will become smaller and cheaper.

6.1.1 Algorithm Improvement

Obviously the area of algorithm development is something that can always be improved. As more tests are conducted with the system over a wider variety of conditions there is likely to be more enhancements to the algorithms. It would be nice to have a confidence level coming from the algorithm. This confidence level could then be used to set various warning levels or actions that should be taken.

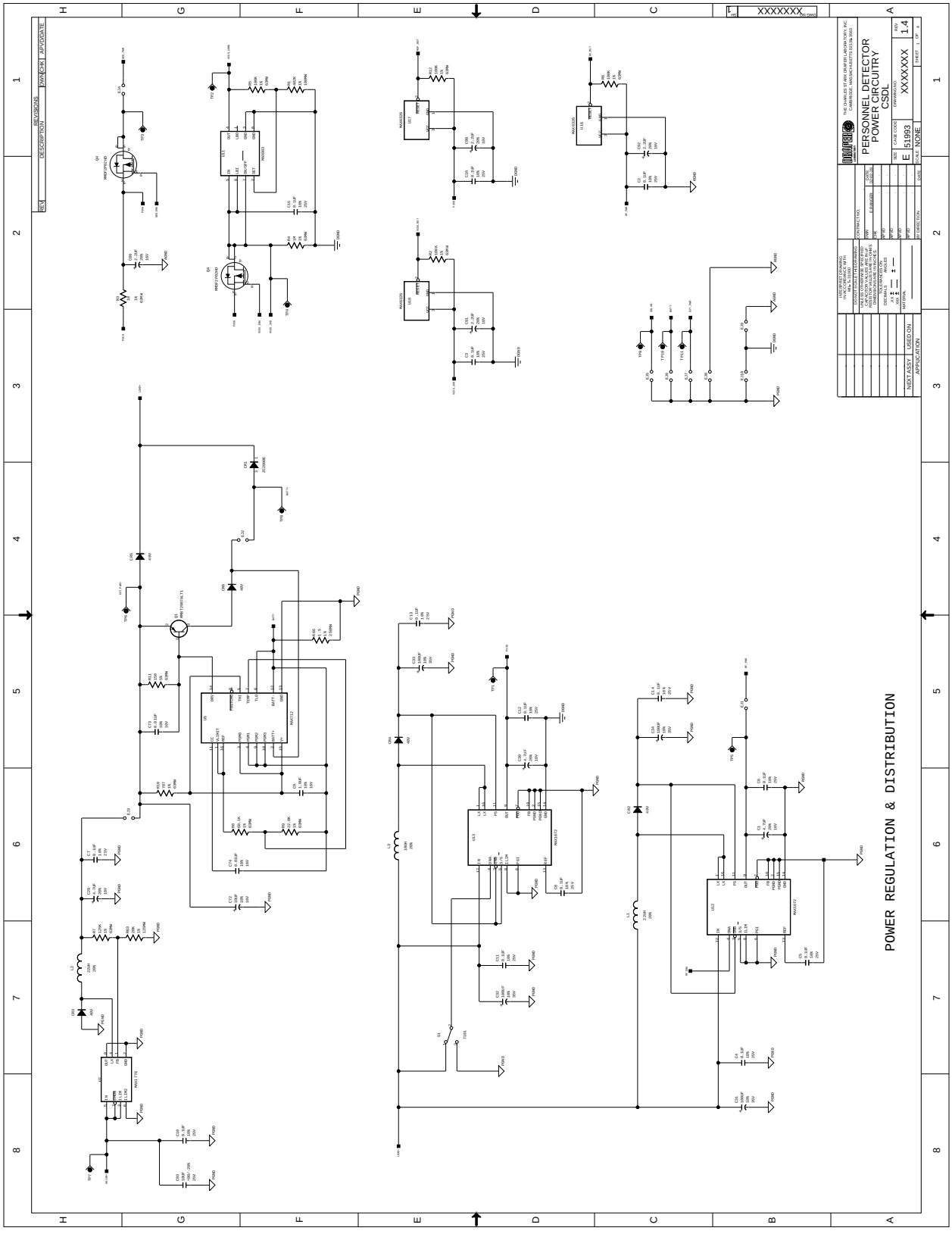
Time domain algorithms could be examined to determine if it is possible to use those methods in conjunction with frequency data to get more accurate algorithms that detect at further distances. Alternatively, the system could be easily redesigned to do math intensive tracking algorithms. More memory could be added to the DSP and the DSP can be clocked at up to 200 MHz. This opens up a whole set of possibilities for tracking algorithms.

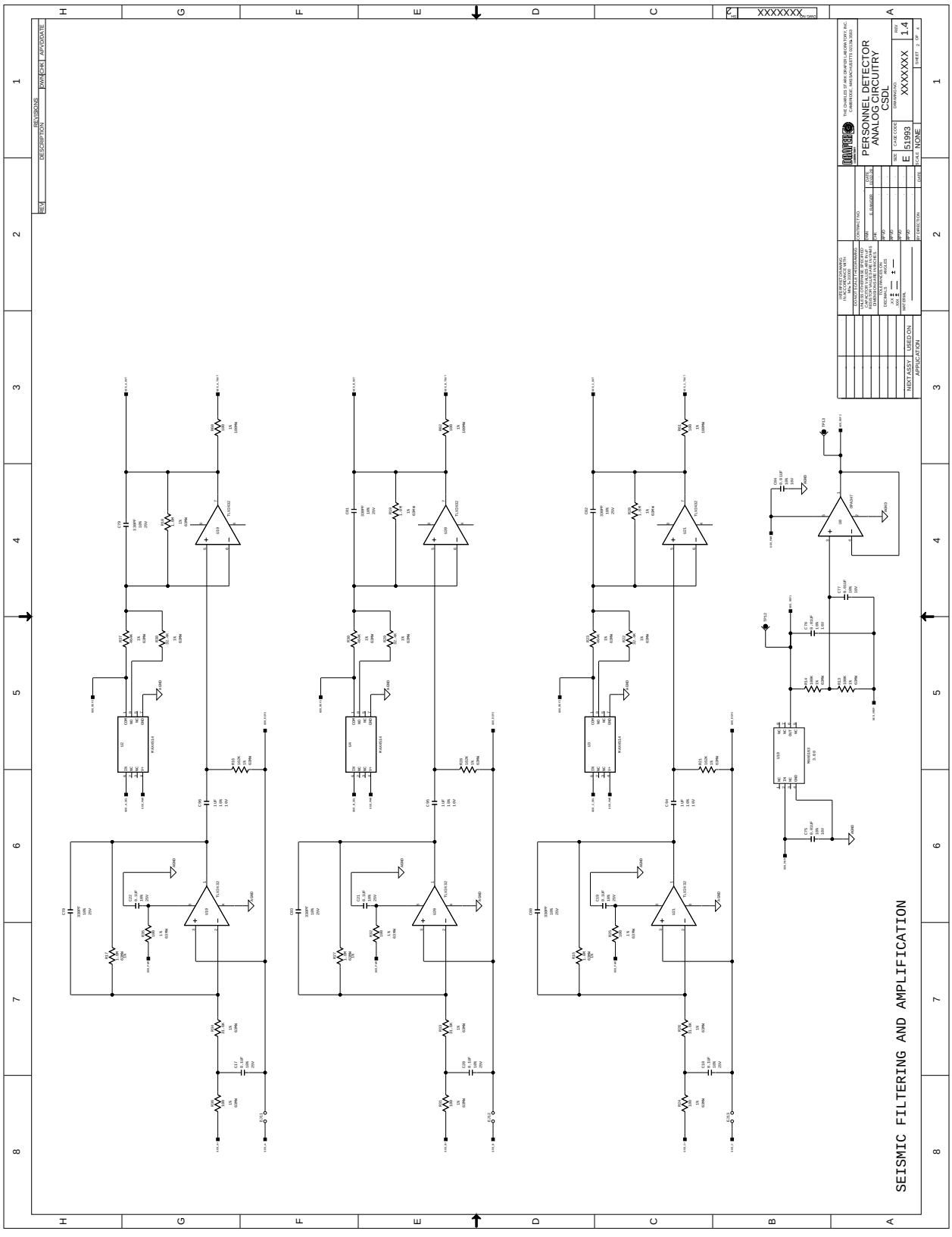
6.1.2 DSP Technology

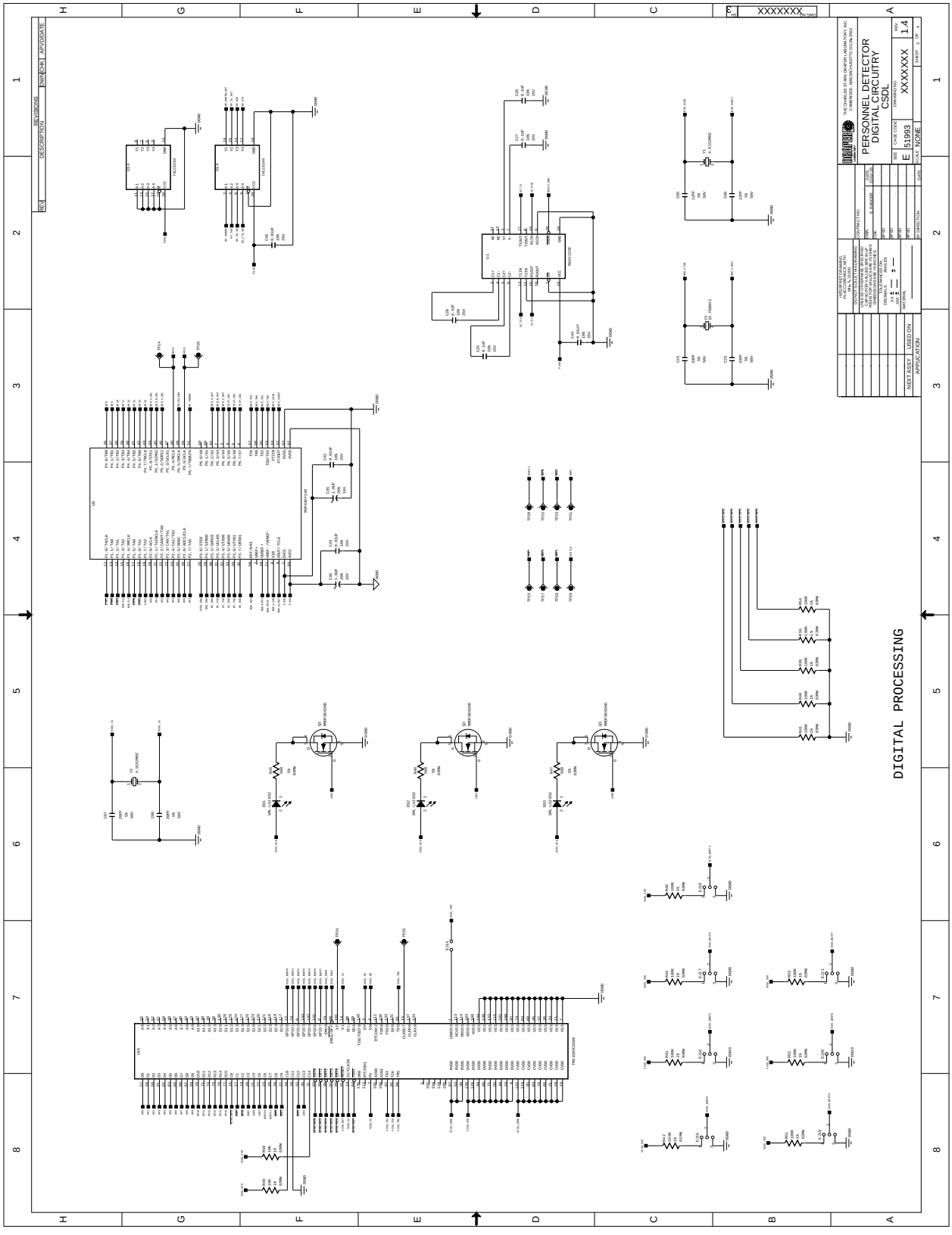
As DSP technology improves the DSP should be able to have more functionality, with less power. The TI DSP that was used in the design was a TMX part which means it had not been fully tested and not guaranteed to meet all the specifications in the datasheet. In future versions, the production level part should have better performance and power usage.

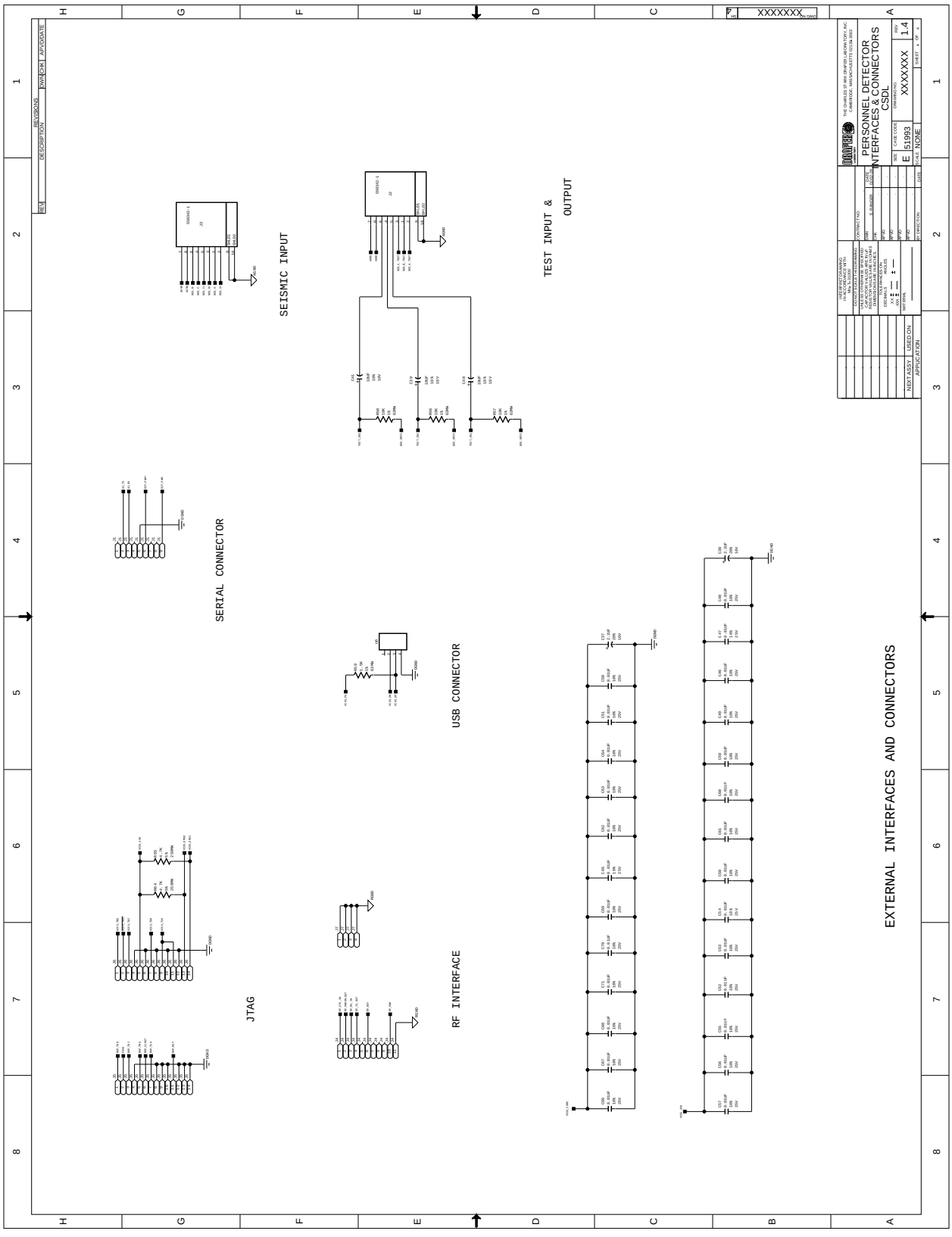
Appendix A

Schematics


POWER REGULATION & DISTRIBUTION
PERSONNEL DETECTOR
POWER CIRCUITRY
CSDL
XXXXXXX
E 51993
1.4
NONE


SEISMIC FILTERING AND AMPLIFICATION
PERSONNEL DETECTOR
ANALOG CIRCUITRY
CSDL
XXXXXXX
1.4
E 51993
NONE
REVISIONS
DESCRIPTION
DWN
APVD
DATE


DIGITAL PROCESSING
PERSONNEL DETECTOR
DIGITAL CIRCUITRY
CSDL
XXXXXXX
1.4
E 51993
NONE


SEISMIC INPUT
TEST INPUT &
OUTPUT
SERIAL CONNECTOR
USB CONNECTOR
JTAG
RF INTERFACE
EXTERNAL INTERFACES AND CONNECTORS
PERSONNEL DETECTOR
INTERFACES & CONNECTORS
CSDL
XXXXXXX
E 51993
1.4
NONE

Appendix B

Parts Listing

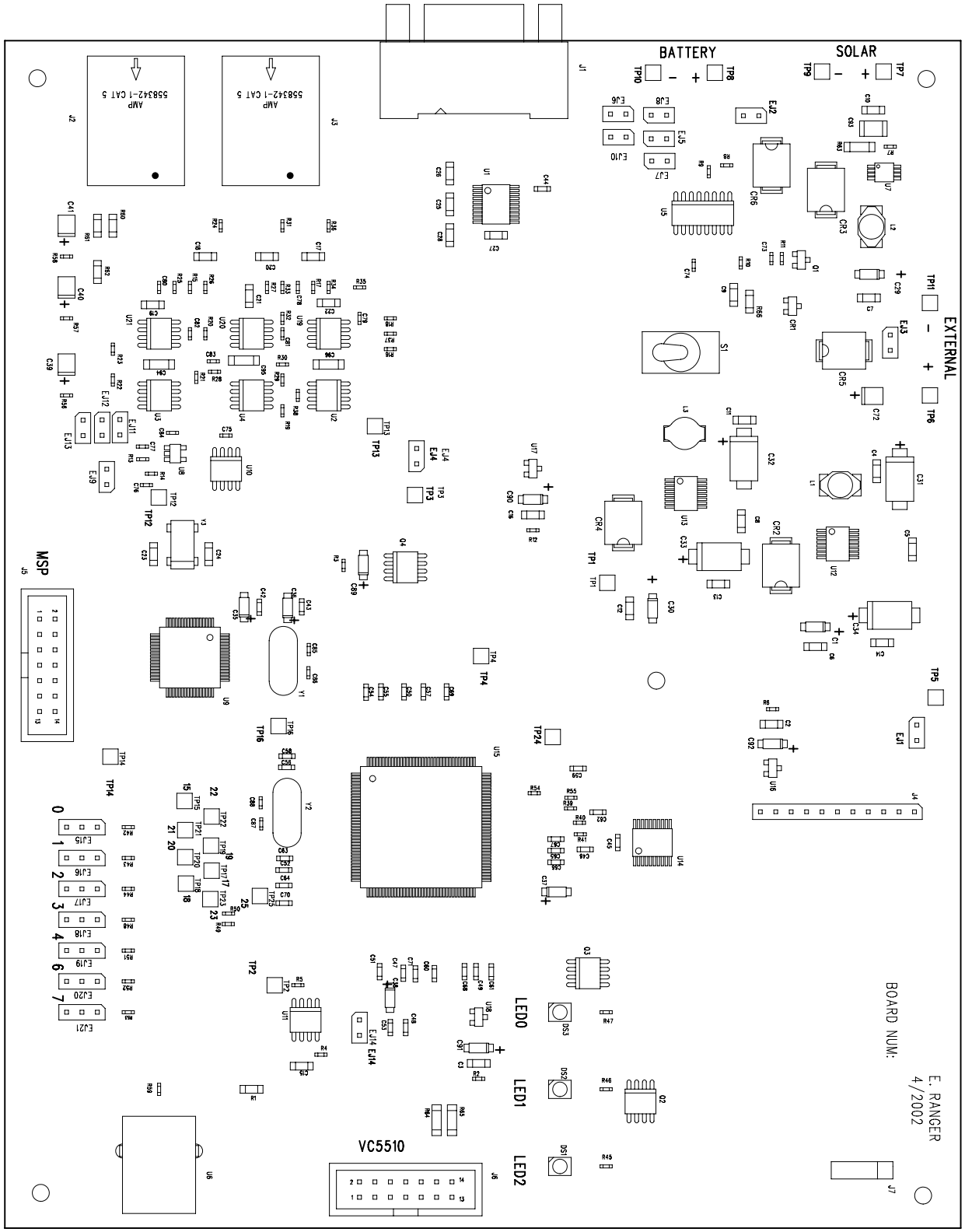
NUM	QTY	REF_DES	VALUE/COM	NAME	TOL	PWR	PART	NUMBER	DESCRIPTION	PACKAGE
1	1	C1	4.7UF	20%	16V	ECST1CY475R	CAP, FIXED, TANTALUM	CHIP	U0-EIA3216	PANASONIC
2	24	C2,C3,C4, C5,C6,C7, C8,C10,C11, C12,C13, C14,C15, C16,C17, C18,C19, C20,C21, C22,C25, C26,C27,C28	0.1UF	10%	25V	ECJ2VB1E104K	CAP, FIXED, CERAMIC, CHIP	CHIP	U0-EIA0805	PANASONIC
3	1	C9	1.0UF	10%	16V	C0805C105K4RACTU	CAP, X7R, CERAMIC, CHIP	CHIP	U0-EIA0805	KEMET
4	2	C23,C24	33PF	5%	50V	ECJ2VC1H330J	CAP, NPO, CERAMIC, CHIP	CHIP	U0-EIA0805	PANASONIC
5	2	C29,C30	4.7UF	20%	16V	ECST1CY475R	CAP, FIXED, TANTALUM	CHIP	U0-EIA3216	PANASONIC
6	4	C31,C32, C33,C34	100UF	20%	10V	ECST1AD107R	CAP, FIXED, TANTALUM	CHIP	U0-EIA7343	PANASONIC
7	2	C35,C36	1UF	20%	16V	ECST1CY105R	CAP, FIXED, TANTALUM	CHIP	U0-EIA3216	PANASONIC
8	2	C37,C38	2.2UF	20%	16V	ECST1CY225R	CAP, FIXED, TANTALUM	CHIP	U0-EIA3216	PANASONIC
9	3	C39,C40,C41	10UF	10%	10V	ECST1AX106R	CAP,FIXED,TANTALUM	CHIP, EPOXY MNT	U1-EPX3528	PANASONIC
10	30	C42,C43, C44,C45, C46,C47, C48,C49, C50,C51, C52,C53, C54,C55, C56,C57, C58,C59, C60,C61, C62,C63, C64,C65, C66,C67, C68,C69, C70,C71	0.01UF	10%	25V	ECJ1VB1E103K	CAP, FIXED, CERAMIC, CHIP	CHIP	U0-EIA0603	PANASONIC
11	1	C72	10UF	10%	16V	ECST1CX106R	CAP,FIXED,TANTALUM	CHIP, EPOXY MNT	U1-EPX3528	PANASONIC
12	6	C73,C74, C75,C76, C77,C84	0.01UF	10%	16V	ECJ0EB1C103K	CAP, X7R, CERAMIC, CHIP	CHIP	U0-EIA0402	PANASONIC
13	6	C78,C79, C80,C81, C82,C83	330PF	10%	25V	ECJ0EB1E331K	CAP, X7R, CERAMIC, CHIP	CHIP	U0-EIA0402	PANASONIC
14	2	C85,C86	12PF	5%	50V	ECJ0EC1H120J	CAP, CERAMIC, CHIP	CHIP	U0-EIA0402	PANASONIC
15	2	C87,C88	20PF	5%	50V	ECU-E1H200JCQ	CAP, CERAMIC, CHIP	CHIP	U0-EIA0402	PANASONIC
16	4	C89,C90, C91,C92	2.2UF	20%	16V	ECST1CY225R	CAP, FIXED, TANTALUM	CHIP	U1-EIA3216	PANASONIC
17	1	C93	10UF	+80/-20%	25V	ECJ-4YF1E106Z	CAP, FIXED, CERAMIC, CHIP	CHIP	U0-EIA1210	PANASONIC
18	3	C94,C95,C96	1UF	10%	16V	ECJ3YB1C105K	CAP, FIXED, CERAMIC, CHIP	CHIP	U0-EIA1206	PANASONIC
19	1	CR1				ZC2800ECT-ND	DIODE, SCHOTTKY, BARRIER	3 PIN SOT-23	U-SOT23-A	ZETEX
20	5	CR2,CR3, CR4,CR5,CR6	40V			MBRS340T3	RECTIFIER, POWER, SCHOTTKY	SMD	U1-MBRST3	MOTOROLA
21	3	DS1,DS2,DS3				SML-LX2832IC	RED LED, SMD	2 PIN LED, SMD	U1-LX2832	LUMEX
22	14	EJ1,EJ2, EJ3,EJ4, EJ5,EJ6, EJ7,EJ8, EJ9,EJ10, EJ11,EJ12, EJ13,EJ14				890-39-002-10-802	.025 SQ PIN,1X2 VERT HDR	HEADER,VERT,1X2	HDR01X02V	MILL-MAX

NUM	QTY	REF_DES	VALUE/COM	NAME	TOL	PWR	PART	NUMBER	DESCRIPTION	PACKAGE
23	7	EJ15,EJ16, EJ17,EJ18, EJ19,EJ20, EJ21				890-39-003-10-802	.025 SQ PIN,1X3 VERT HDR	HEADER,VERT,1X3	HDR01X03V	MILL-MAX
24	1	J1				745395-2	SUBMINI D, RT-ANGLE, RECEPT, 318 MNT	9 PIN PC TERMINAL	AMPHD20RA09R	AMP
25	2	J2,J3				558342-1	JACK, MODULAR, RA, CATEGORY 5	JACK, MODULAR, SMT	T-558342-1_CAT5	AMP
26	1	J4				890-39-011-10-802	.025 SQ PIN,1X11 VERT HDR	HDR,VERT,1X11	HDR01X11V	MILL-MAX
27	2	J5,J6				2514-6002UB	HEADER, 2X7, STRT, SHROUDED	2X7 PIN STRT HEADER	CON_3M2514	3M
28	1	J7				890-39-004-10-802	.025 SQ PIN,1X4 VERT HDR	HDR,V,P,1X4	T-HDR01X04V	MILL-MAX
29	1	J8							AMPHD20RA09R	
30	2	L1,L2	22UH		20%	DO1608C-223	SMT POWER INDUCTOR	SMT,BOBBIN	U0-DO1608C	COILCRAFT
31	1	L3	10UH		20%	LPT1606-103	INDUCTOR, POWER, SMT	TOROID SMT	U-LPT1606	COILCRAFT
32	1	Q1				MMBT2907ALT1	TRANSISTOR, PNP	3 PIN SOT-23	U-SOT23	ON SEMICONDUCTOR
33	2	Q2,Q3				MMDF3N02HDR2	TRANSISTOR DUAL N-CHANNEL	8 PIN SO .150 WIDE	U-MS012AA	MOTOROLA
34	1	Q4				MMDF2P02HDR2	TRANSISTOR DUAL P-CHANNEL	8 PIN SO .150 WIDE	U-MS012AA	MOTOROLA
35	1	R1	402K		1% 100MW	ERJ6ENF4023V	RES,CHIP,THK FLM	CHIP	U0-EIA0805	PANASONIC
36	18	R2,R5,R6, R12,R13, R14,R39, R40,R41, R42,R43, R44,R48, R51,R52, R53,R54,R55	100K		1% 63MW	ERJ2RKF1003X	RESISTOR, CHIP	CHIP	U0-EIA0402	PANASONIC
37	1	R3		10	1% 63MW	ERJ2RKF10R0X	RESISTOR, CHIP	CHIP	U0-EIA0402	PANASONIC
38	1	R4	1.0M		1% 63MW	ERJ2RKF1004X	RESISTOR, CHIP	CHIP	U0-EIA0402	PANASONIC
39	1	R7	125K		1% 63MW	ERJ2RKF1253V	RESISTOR, CHIP	CHIP	U0-EIA0402	PANASONIC
40	1	R8	68.1K		1% 63MW	ERJ2RKF6812V	RESISTOR, CHIP	CHIP	U0-EIA0402	PANASONIC
41	1	R9	22.0K		1% 63MW	ERJ2RKF2202V	RESISTOR, CHIP	CHIP	U0-EIA0402	PANASONIC
42	1	R10		787	1% 63MW	ERJ2RKF7870V	RESISTOR, CHIP	CHIP	U0-EIA0402	PANASONIC
43	1	R11		150	5% 63MW	ERJ2GEJ151V	RESISTOR, FIXED, THICK FILM	CHIP	U0-EIA0402	PANASONIC
44	6	R15,R17, R18,R19, R20,R27	1.0M		1% 63MW	ERJ2RKF1004X	RESISTOR, CHIP	CHIP	U0-EIA0402	PANASONIC
45	3	R16,R21,R28	162K		1% 63MW	ERJ2RKF1623X	RESISTOR, CHIP	CHIP	U0-EIA0402	PANASONIC
46	3	R22,R29,R38	32.4K		1% 63MW	ERJ2RKF3242X	RESISTOR, CHIP	CHIP	U0-EIA0402	PANASONIC
47	3	R23,R30,R37	464K		1% 63MW	ERJ2RKF4643X	RESISTOR, CHIP	CHIP	U0-EIA0402	PANASONIC
48	6	R24,R25, R31,R32, R35,R36		100	1% 63MW	ERJ2RKF1000X	RESISTOR, CHIP	CHIP	U0-EIA0402	PANASONIC
49	3	R26,R33,R34	31.6K		1% 63MW	ERJ2RKF3162X	RESISTOR, CHIP	CHIP	U0-EIA0402	PANASONIC
50	3	R45,R46,R47		560	5% 63MW	ERJ2GEJ561V	RESISTOR, FIXED, THICK FILM	CHIP	U0-EIA0402	PANASONIC
51	5	R49,R50, R56,R57,R58	10K		1% 63MW	ERJ2RKF1002V	RESISTOR, CHIP	CHIP	U0-EIA0402	PANASONIC
52	1	R59	1.5K		1% 63MW	ERJ2RKF1501V	RESISTOR, CHIP	CHIP	U0-EIA0402	PANASONIC
53	3	R60,R61,R62		100	1% 100MW	ERJ6ENF1000B	RESISTOR, CHIP	CHIP	U0-EIA0805	PANASONIC
54	1	R63	20K		1% 125MW	ERJ8ENF2002	RESISTOR, THICK FILM, CHIP	CHIP	U0-EIA1206	PANASONIC
55	2	R64,R65	4.7K		5% 250MW	ERJ8GEYJ472	RESISTOR, THICK FILM, CHIP	CHIP	U0-EIA1206	PANASONIC
56	1	R66		1.5	1% 250MW	RL1220S-1R5-F	RESISTOR, CHIP	CHIP	U0-EIA0805	SUSUMU
57	1	S1				7101SYCQE	TOGGLE SWITCH, SPDT	TOGGLE SWITCH, SPDT,	SW-7101	C&K
58	25	TP1,TP2, TP3,TP4, TP5,TP6, TP7,TP8, TP9,TP10, TP11,TP12, TP13,TP14, TP15,TP16, TP17,TP18,				890-39-001-10-802	.025 SQ PIN,1X1 VERT HDR	HDR,V,P,1X1	T-HDR01X01V	MILL-MAX

NUM	QTY	REF_DES	VALUE/COM	NAME	TOL	PWR	PART	NUMBER	DESCRIPTION	PACKAGE
		TP19,TP20, TP21,TP22, TP23,TP24, TP25								
59	1	U1				MAX3222EEUP	TRANSCEIVER, RS-232, LOW-POWER	20 PIN TSSOP	U-MO153AC	MAXIM
60	3	U2,U3,U4				MAX4514ESA	ANALOG SWITCH	8 PIN SO .150 WIDE	U-MS012AA	MAXIM
61	1	U5				MAX712ESE	CONTROLLER, FAST-CHARGE	16 PIN SO	U-MS012AC	MAXIM
62	1	U6				AU-Y1007	CONNECTOR, USB, SERIES B, 4-POS	4 PIN PC MNT	CONN_AU-Y1007	ASSMANN ELECTRONICS
63	1	U7				MAX1776EUA	CONVERTER, STEP-DOWN, 600MA, 24V	8 PIN UMAX	U-MO187AA	MAXIM
64	1	U8				OPA347NA	OPAMP, MICROPWR, RAIL-TO-RAIL	5 PIN SOT-23-5	U-SOT23-5	BURR-BROWN
65	1	U9				MSP430F149IPM	MICROCONTROLLER, MIXED SIG	64 PIN PQFP	U-MS026BCD	TI
66	1	U10			3	MAX6163BESA	VOLTAGE REF, PRECISION, 3.000V	8 PIN SO	U-MS012AA	MAXIM
67	1	U11				MAX883ESA	LINEAR REGULATOR	8 PIN SO .150 WIDE	U-MS012AA	MAXIM
68	2	U12,U13				MAX1672EEE	CONVERTER, DC-DC, UP/DOWN	16 PIN QSOP	U-QSOP16	MAXIM
69	1	U14				74LCX244MTC	BUFFER/LINE DRIVER, OCTAL NON-INV	20 PIN TSSOP	U-MO153AC	NATL SEMI
70	1	U15				TMS320VC5509PGE144	DSP, FIXED-POINT, 144MHZ	144 PIN LQFP	U-MS026BFB	TI
71	3	U16,U17,U18				MAX6326UR23-T	RESET, MICROPROCESSOR	SOT23-3	U-SOT23-3	MAXIM
72	3	U19,U20,U21				TLV2432D	DUAL OP AMP	8 PIN SO .150 WIDE	U-MS012AA	TI
73	2	Y1,Y2	4.9152MHZ			ECS-49-20-1	CRYSTAL, QUARTZ	2 PIN RADIAL LEAD	T-HC49UA	ECS
74	1	Y3	32.768KHZ			ECS-.327-12.5-17	SMD QUARTZ CRYSTAL	SF MNT 4 PIN	U-ECX306	ECS

Appendix C

Assembly Drawing


BATTERY
SOLAR
EXTERNAL
MSP
VC5510
LED0
LED1
LED2
BOARD NUM:
E. RANGER
4/2002
AMP
5583 42-1 CAT 5

Appendix D

Test Procedure

Personnel Detector
Test Procedure

Elliot Ranger
6/26/2002

TEST INFORMATION

Board Number _____

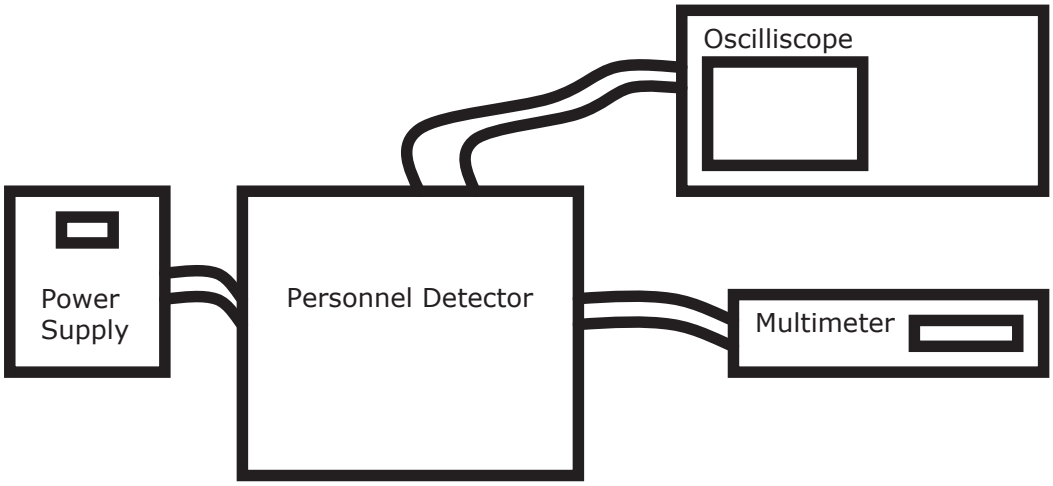
Name of Tester _____

Date _____

EQUIPMENT:

MANUFACTURER	NAME	MODEL	SERIAL NUM.	DRAPER NUM.
Lambda	Power Supply	LP-521-FM		
Fluke	Multimeter	89 IV		
Tektronix	Digital Oscilliscope	TDS 3014		
Hewlett Packard	Dynamic Signal Analyzer	3562A		

SETUP



CONTINUITY TESTS

Instructions: Make sure Jumpers EJ5,EJ6,EJ7,EJ8,EJ9, EJ10, EJ3, EJ2, EJ4, EJ2, EJ1, EJ11, EJ12, EJ13 are connected.

GROUNDS ALL CONNECTED

With one probe on TP10 check for continuity between:

EJ5 _____

EJ7 _____

EJ8 _____

EJ9 _____

EJ10 _____

POWER NOT SHORTED TO GROUND

With one probe on TP10 check to make sure there is **no** continuity between:

TP7 _____

TP6 _____

TP8 _____

TP1 _____

TP5 _____

TP3 _____

TP4 _____

TP2 _____

TP12 _____

TP13 _____

3.3V POWER TEST

Instructions: Remove all Jumpers except for EJ5,EJ6,EJ7,EJ8,EJ9, and EJ10. Use an external power supply to provide 6V of power from TP6 (EXT_PWR+) to TP11 (EXT_PWR-).

Record the voltage at TP1 with S1 on _____

Turn S1 off and on and observe at least a 100ms pulse (active low) at U17 Pin 2 _____

Sweep the power supply voltage from 3.5 V - 10 V ensure that the voltage at TP1 stays at approximately 3.3 V _____

Determine the input voltage that causes the Voltage at TP1 to drop below 3.1 V _____

MSP

CLOCK MEASUREMENTS

Instructions: Program the MSP through the JTAG port with the test program.

Record the frequency of MCLK at TP14 _____

Record the frequency of ACLK at TP16 _____

RS232 TEST

Test that the MSP is able to echo characters back to the computer _____

PORT TEST

Toggle the following signals:

VC55_ENA

Record the voltage of VC55_3VD at TP4 _____

Record the voltage of VC55_CORE at TP2 _____

Turn ON the VC55 and observe at least a 100ms pulse (active low) at U18 Pin 2 _____

GEO_ENA

Record the voltage of GEO_PWR at TP3 _____

RF_ENA

Record the voltage of RF_PWR at TP5 _____

Instructions: Connect Jumper EJ1.
Toggle RF_ENA and observe at least a
100ms pulse (active low) at U16 Pin 2 _____

RF_PWRDN (J4 PIN 2) _____

GEO_A_SEL (U2 PIN 6) _____

GEO_B_SEL (U4 PIN 6) _____

GEO_C_SEL (U3 PIN 6) _____

RS232_ENA (Characters should not be displayed
on the terminal when turned off) _____

ANALOG TEST

Instructions: Connect a Jumper across EJ4. Turn on GEO_ENA with the MSP.

SEISMIC MEASUREMENTS

Instructions: Connect a signal analyzer to the positive input and measure the output at Test out.

GEO_A:

Record the small signal gain _____

Record the high frequency -3db point _____

Record the noise floor _____

GEO_B:

Record the small signal gain _____

Record the high frequency -3db point _____

Record the noise floor _____

GEO_C:

Record the small signal gain _____

Record the high frequency -3db point _____

Record the noise floor _____

VC5509

Instructions: Turn on VC55_ENA with the MSP.

CLOCK MEASUREMENT

Instructions: Program the VC5509 through the JTAG port with the test program.

Record the frequency of CLKOUT at TP25 _____

LEDS

Observe that LED0 lights _____

Observe that LED1 lights _____

Observe that LED2 lights _____

HPI

Instructions: Select the HPI menu item.

Verify that the MSP can write a character into
The VC5509s memory and the VC5509 can read it _____

Verify that the VC5509 can write a character into
The MSPs memory and the MSP can read it _____

RF INTERFACE

Instructions: Connect a Jumper across EJ1. Turn on RF_ENA with the MSP.
Turn the RF echo ON from the MSP.

Send characters from one of the Evaluation Board
Maxstream modules to the board in question and
Observe that characters are correctly echoed back _____

SOLAR INTERFACE

Instructions: Ensure that Jumper EJ3 is removed. Use an external power supply to provide power from TP7 (SOLAR+) to TP9 (SOLAR-).

Set the voltage of the power supple to 12 V
And record the voltage at PIN 1 of EJ3 _____

Sweep the voltage of the power supply from
12 V - 21 V and ensure the voltage at PIN 1 of EJ3
Stays constant _____

Determine the input voltage that causes the
Voltage at PIN 1 of EJ3 to drop below 5 V _____

BATTERY CHARGER

Instructions: Ensure that Jumper EJ2 is removed. Use an external power supply to provide 9.1V of power from TP6 (EXT_PWR+) to TP11 (EXT_PWR-).

Record the voltage from EJ2 PIN 1 to TP10 (BATT-) _____

Place an ammeter across EJ2. Connect a battery pack
with 4 partially drained NiMH batteries in series from
TP8 (BATT+) to TP10 (BATT-) record the amount of
Current flowing through EJ2 _____

Appendix E

TMS320VC5509 Source Code

E.1 bootcode.asm

```
;
;  Personnel Detector
;  Elliot Ranger
;  Master of Science Thesis
;  Massachusetts Institute of Technology
;  in conjunction with Charles Stark Draper Laboratory
;
;
;*****************************************************************************\
; FILENAME..... bootcode.asm
; DATE CREATED.. 07/11/2002
; LAST MODIFIED. 01/15/2003
;*****************************************************************************/

 .global _hpi_save_code, _hpi_load_code
 .sect ".text"

_hpi_save_code:

; Copy bootcode to hpi write section
        AMOV #0x000180,XAR1
        AMOV #0x000D00,XAR3
        NOP
        NOP
```

```
            NOP
            NOP
            NOP
            RPT #0x007F
            MOV *AR1+,*AR3+

    ; Copy vectors to hpi write section
            AMOV #0x000080,XAR1
            AMOV #0x000D80,XAR3
            NOP
            NOP
            NOP
            NOP
            NOP
            RPT #0x007F
            MOV *AR1+,*AR3+

    ; Copy DARAM1 to hpi write section
            AMOV #0x000400,XAR1
            AMOV #0x000E00,XAR3
            NOP
            NOP
            NOP
            NOP
            NOP
            RPT #0x00FF
            MOV *AR1+,*AR3+

    ; Copy DARAM2 to hpi write section
            AMOV #0x000600,XAR1
```

```
        AMOV #0x000F00,XAR3
        NOP
        NOP
        NOP
        NOP
60      NOP
        RPT #0x06FF
        MOV *AR1+,*AR3+

    ; Copy SARAM0 to hpi write section
        AMOV #0x008000,XAR1
        AMOV #0x001600,XAR3
        NOP
        NOP
        NOP
70      NOP
        NOP
        RPT #0x17FF
        MOV *AR1+,*AR3+

    ; Copy SARAM1 to hpi write section
        AMOV #0x00A000,XAR1
        AMOV #0x002E00,XAR3
        NOP
        NOP
80      NOP
        NOP
        NOP
        RPT #0x0A7F
        MOV *AR1+,*AR3+
```

```
            RET


     .sect ".boot"
    _hpi_load_code:
90

    ; Copy hpi write section to vectors
            AMOV #0x000D00,XAR1
            AMOV #0x000080,XAR3
            NOP
            NOP
            NOP
            NOP
            NOP
            RPT #0x007F
100         MOV *AR1+,*AR3+


    ; Copy hpi write section to DARAM1
            AMOV #0x000D80,XAR1
            AMOV #0x000400,XAR3
            NOP
            NOP
            NOP
            NOP
            NOP
110         RPT #0x00FF
            MOV *AR1+,*AR3+


    ; Copy hpi write section to DARAM2
            AMOV #0x000E80,XAR1
```

```
            AMOV #0x000600,XAR3
            NOP
            NOP
            NOP
            NOP
120         NOP
            RPT #0x06FF
            MOV *AR1+,*AR3+

    ; Copy hpi write section SARAM1
            AMOV #0x001580,XAR1
            AMOV #0x008000,XAR3
            NOP
            NOP
            NOP
130         NOP
            NOP
            RPT #0x17FF
            MOV *AR1+,*AR3+

    ; Copy hpi write section SARAM2
            AMOV #0x002D80,XAR1
            AMOV #0x00A000,XAR3
            NOP
            NOP
140         NOP
            NOP
            NOP
            RPT #0x0A7F
            MOV *AR1+,*AR3+
```

```
B 0x010000
```

E.2 define.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*************************************************************************\
* FILENAME..... define.h
* DATE CREATED.. 07/11/2002
* LAST MODIFIED. 03/04/2003
\*************************************************************************/

#define TRUE 1
#define FALSE 0

#define INV_LOG2 3.321928094887
#define DC_OFFSET 16384
#define VERSION "TMX320VC5509PGE VERSION 1.20  3/04/2003"

#define MAX_BUFFER 90

/* MSP Command Codes */
#define PRINT_MESSAGE    0x0100
#define STOP_PROCESSING 0x0200

/* VC5509 Command Codes */
#define PROCESS_DATA_BLOCK_1      0x0100
#define PROCESS_DATA_BLOCK_2      0x0200
#define CLEAR_PROCESSING_BUFFERS 0x0300
#define GET_VC5509_VERSION        0x0900
#define COPY_TO_HPI_WRITE         0x0999
```

```
/* Memory locations  */
#define BLOCK1_MEM    0x1000
#define BLOCK2_MEM    0x1500

/* Detection Paramaters */
#define FIRST_THRESHOLD 300
#define SECOND_THRESHOLD 250
#define THIRD_THRESHOLD 250

#define PEAK_WIDTH 2
```

E.3 difference.asm

```
; Function Call
; ---------------
;
;   prototype: void hp_filter(DATA *x, ushort nx, DATA *y_previous, DATA *x_previous)
;
;   Entry:  arg0: AR0 - signal input pointer
;            arg1: T0 - number of samples in the input
;            arg2: AR1 - pointer to previous output y
;            arg3: AR2 - pointer to previous input x
;
;


    .def    _hp_filter
    .sect   .text

_window
    pshm  ST1_55              ; Save ST1, ST2, and ST3
    pshm  ST2_55
    pshm  ST3_55


        or    #0x340,mmap(ST1_55); Set FRCT,SXMD,SATD

    bset  SMUL                ; Set SMUL



    sub   #1,T0

    mov   T0,BRC0              ; Outer loop couter

    rptblocal lp1-1

    mov   *AR0, T2                          ; save the input value x[i] in T2
```

```
        mov    *AR1, T1              ; store y[previous] in T1
        sub    *AR2,*AR0,AC0         ; subtract x[i] - x[previous]
30      mack   T1, #30720, AC0, AC0 ; multiply y[previous] * 30720 and accumulate
        mov    hi(AC0),*AR0          ; store the new output
        mov    T2, *AR2                           ; put the previous x value in AR2
        mov    *AR0+, *AR1           ; put the previous y value in AR1
    lp1



        popm   ST3_55               ; Restore ST1, ST2, and ST3
        popm   ST2_55
        popm   ST1_55
40      ret
        .end
```

E.4 extern.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*************************************************************************\
10 * FILENAME..... extern.h
* DATE CREATED.. 07/11/2002
* LAST MODIFIED. 08/07/2002
\*************************************************************************/

extern Uint16 timer0_counter;
extern Uint16 new_command;
extern DATA* block_memloc;
```

E.5 filters.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... filters.h
* DATE CREATED.. 08/07/2002
* LAST MODIFIED. 02/19/2003
\*****************************************************************************/

#define NX_IIR 1200
#define NBIQ_IIR 2

#define NX_FIR1 1200
#define NX_FIR2 240
#define NH_FIR 31
#define D_1 5
#define D_2 2

#define NX_SIZE 120

/* delay buffers */
DATA dbuffer_bp[2*NBIQ_IIR];
DATA dbuffer_firdec1[NH_FIR+1];
DATA dbuffer_firdec2[NH_FIR+1];
ushort dbuffer_index1;
ushort dbuffer_index2;

DATA iir_hp_y;
DATA iir_hp_x;
DATA iir_lp_y;

/* coefficients - bandpass IIR filter */
```

```
DATA h_bp[5*NBIQ_IIR] ={ /* C55x: a1 a2 b2 b0 b1 ... */
−26992,
11483,
1852,
1852,
3704,
−31313,
14991,
1852,
1852,
−3704,
};

/* coefficients - FIR Decimation filter stage 1 (M=5) */
DATA h_firdec1[NH_FIR] ={
0,
39,
91,
139,
129,
0,
−271,
−609,
−832,
−696,
0,
1297,
3011,
4755,
6059,
6542,
6059,
4755,
3011,
1297,
0,
−696,
−832,
−609,
−271,
0,
129,
139,
91,
39,
```

```
0,
};

/* coefficients - FIR Decimation filter stage 2 (M=2) */
DATA h_firdec2[NH_FIR] ={
-56,
0,
96,
0,
-221,
0,
462,
0,
-878,
0,
1609,
0,
-3176,
0,
10342,
16410,
10342,
0,
-3176,
0,
1609,
0,
-878,
0,
462,
0,
-221,
0,
96,
0,
-56,
};
```

E.6 firdecimate.asm

```
;
;  Personnel Detector
```

```
;  Elliot Ranger
;  Master of Science Thesis
;  Massachusetts Institute of Technology
;  in conjunction with Charles Stark Draper Laboratory
;
;
;*************************************************************************\
; FILENAME..... firdecimate.asm
; DATE CREATED.. 09/11/2002
; LAST MODIFIED. 09/15/2002
;*************************************************************************/
;   Modified code from Real-Time Digital Signal Processing
;
;   firdecimate.asm - decimating FIR filter
;
;   prototype: ushort firdecimate(DATA *x, ushort nx, DATA *h,
;                        ushort nh, DATA *r, DATA *dbuffer, ushort dbufindex
;                                              ushort m, ushort outsize);
;
;   Entry:  arg0: AR0 - filter input buffer pointer
;            arg1: T0 - number of samples in the input buffer
;            arg2: AR1 - FIR coefficients array pointer
;            arg3: T1 - FIR filter order
;            arg4: AR2 - filter output buffer pointer
;            arg5: AR3 - signal buffer pointer
;            arg6: AR4 - signal buffer index
;                     arg7: on stack - decimation factor
;            arg8: on stack - length after decimation
;   Return: T0 = signal buffer index
;

    .def    _firdecimate
    .sect   .text

_firdecimate
    pshm  ST1_55              ; Save ST1, ST2, and ST3
    pshm  ST2_55
    pshm  ST3_55
    pshm  T2
    pshm  T3


    mov   *SP(#6), T2

    mov   *SP(#7), T3

    or    #0x340,mmap(ST1_55); Set FRCT,SXMD,SATD
```

```
        bset  SMUL                ; Set SMUL
        mov   XAR1,XCDP           ; CDP as  coefficient pointer
        mov   mmap(AR1),BSAC      ; Set up base address for CDP
50      mov   #0,CDP               ; Start from the 1st coefficient
        mov   mmap(T1),BKC        ; Set the coefficient array size
        mov   mmap(AR3),BSA23     ; Set base address for AR3
        mov   mmap(T1),BK03       ; Set signal buffer x[ ] size as L
        mov   AR4,AR3             ; AR3 signal buffer index
        or    #0x10A,mmap(ST2_55); CDP, AR1, AR3 circular pointers;
        sub   #1,T3               ;   outer repeat counter NX/D
        mov   T3,BRC0             ; Outer loop couter
        sub   #2,T2               ;   Decimation factor - 2
        sub   #3,T1,T0
60      bcc    Decim2, T2==0      ; Check to see if decimation factor =2
        rptblocal sample_loop1-1
        mov   *AR0+,*AR3          ; Put new sample to signal buffer x[n]
    ||  mov   T0,CSR              ; Inner loop counter as L-3
        mpym   *AR3+,*CDP+,AC0    ; The first operation
    ||  rpt   CSR
        macm   *AR3+,*CDP+,AC0    ; The rest MAC iterations
        macmr  *AR3,*CDP+,AC0
    ||  mov   T2,CSR
        mov   hi(AC0),*AR2+
70  ||  rpt   CSR
        mov   *AR0+,*AR3-
    sample_loop1
        b L2


    Decim2:
        rptblocal sample_loop2-1
```

```
    mov    *AR0+,*AR3          ; Put new sample to signal buffer x[n]
||  mov    T0,CSR              ; Inner loop counter as L-3
    mpym   *AR3+,*CDP+,AC0     ; The first operation
||  rpt    CSR
    macm   *AR3+,*CDP+,AC0     ; The rest MAC iterations
    macmr  *AR3,*CDP+,AC0
||  mov    T2,CSR
    mov    hi(AC0),*AR2+
    mov    *AR0+,*AR3-         ; do it only once for the case when D=2
sample_loop2


L2:
    popm   T3
    popm   T2
    popm   ST3_55              ; Restore ST1, ST2, and ST3
    popm   ST2_55
    popm   ST1_55
    mov    AR3,T0              ; Return signal buffer index
||  ret
    .end
```

E.7 hanning.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*************************************************************************\
```

```
 * FILENAME..... hanning.h
 * DATE CREATED.. 10/13/2002
 * LAST MODIFIED. 01/22/2003
 \*************************************************************************/

#define WINDOW_SIZE 1024
#define FREQ_SIZE 513

ushort windex;
DATA output[WINDOW_SIZE];
DATA wsignal[WINDOW_SIZE];

/* window coefficients */
ushort hanning[WINDOW_SIZE]={
1,
2,
6,
10,
15,
22,
30,
39,
50,
62,
74,
89,
104,
121,
138,
157,
178,
199,
222,
246,
271,
298,
325,
354,
384,
415,
448,
481,
516,
553,
590,
```

```
    628,
    668,
    709,
    751,
    795,
60  839,
    885,
    932,
    980,
    1029,
    1080,
    1132,
    1185,
    1239,
    1294,
70  1351,
    1408,
    1467,
    1527,
    1588,
    1651,
    1714,
    1779,
    1845,
    1912,
80  1980,
    2049,
    2120,
    2191,
    2264,
    2338,
    2413,
    2489,
    2567,
    2645,
90  2725,
    2806,
    2888,
    2971,
    3055,
    3140,
    3226,
    3314,
    3402,
    3492,
```

```
100  3583,
     3675,
     3768,
     3862,
     3957,
     4053,
     4150,
     4249,
     4348,
     4449,
110  4550,
     4653,
     4757,
     4861,
     4967,
     5074,
     5182,
     5291,
     5401,
     5512,
120  5624,
     5737,
     5851,
     5966,
     6082,
     6199,
     6317,
     6436,
     6556,
     6677,
130  6799,
     6922,
     7046,
     7171,
     7297,
     7424,
     7552,
     7680,
     7810,
     7941,
140  8072,
     8205,
     8338,
     8473,
     8608,
```

```
    8744,
    8881,
    9019,
    9158,
    9298,
150 9438,
    9580,
    9722,
    9865,
    10009,
    10154,
    10300,
    10447,
    10594,
    10742,
160 10892,
    11042,
    11192,
    11344,
    11496,
    11649,
    11803,
    11958,
    12114,
    12270,
170 12427,
    12585,
    12744,
    12903,
    13063,
    13224,
    13386,
    13548,
    13711,
    13875,
180 14039,
    14204,
    14370,
    14537,
    14704,
    14872,
    15041,
    15210,
    15380,
    15550,
```

```
190  15722,
     15893,
     16066,
     16239,
     16413,
     16587,
     16762,
     16938,
     17114,
     17291,
200  17468,
     17646,
     17824,
     18003,
     18183,
     18363,
     18544,
     18725,
     18907,
     19089,
210  19272,
     19455,
     19639,
     19823,
     20008,
     20193,
     20379,
     20565,
     20752,
     20939,
220  21126,
     21314,
     21503,
     21692,
     21881,
     22070,
     22260,
     22451,
     22642,
     22833,
230  23025,
     23217,
     23409,
     23602,
     23795,
```

```
    23988,
    24182,
    24376,
    24570,
    24765,
240 24959,
    25155,
    25350,
    25546,
    25742,
    25938,
    26135,
    26332,
    26529,
    26726,
250 26924,
    27121,
    27319,
    27518,
    27716,
    27914,
    28113,
    28312,
    28511,
    28710,
260 28910,
    29109,
    29309,
    29509,
    29709,
    29909,
    30109,
    30309,
    30510,
    30710,
270 30911,
    31111,
    31312,
    31512,
    31713,
    31914,
    32115,
    32316,
    32516,
    32717,
```

```
280  32918,
     33119,
     33320,
     33521,
     33721,
     33922,
     34123,
     34324,
     34524,
     34725,
290  34925,
     35126,
     35326,
     35526,
     35726,
     35926,
     36126,
     36326,
     36525,
     36725,
300  36924,
     37123,
     37322,
     37521,
     37720,
     37918,
     38117,
     38315,
     38512,
     38710,
310  38908,
     39105,
     39302,
     39498,
     39695,
     39891,
     40087,
     40283,
     40478,
     40673,
320  40868,
     41062,
     41256,
     41450,
     41644,
```

```
    41837,
    42030,
    42222,
    42414,
    42606,
330 42798,
    42989,
    43179,
    43370,
    43559,
    43749,
    43938,
    44127,
    44315,
    44503,
340 44690,
    44877,
    45063,
    45249,
    45435,
    45620,
    45804,
    45988,
    46172,
    46355,
350 46537,
    46719,
    46901,
    47082,
    47262,
    47442,
    47621,
    47800,
    47978,
    48156,
360 48333,
    48509,
    48685,
    48860,
    49035,
    49209,
    49383,
    49555,
    49728,
    49899,
```

```
370  50070,
     50240,
     50410,
     50579,
     50747,
     50915,
     51082,
     51248,
     51413,
     51578,
380  51742,
     51906,
     52068,
     52230,
     52392,
     52552,
     52712,
     52871,
     53029,
     53186,
390  53343,
     53499,
     53654,
     53809,
     53962,
     54115,
     54267,
     54418,
     54569,
     54718,
400  54867,
     55015,
     55162,
     55308,
     55453,
     55598,
     55742,
     55884,
     56026,
     56167,
410  56307,
     56447,
     56585,
     56723,
     56859,
```

```
    56995,
    57130,
    57264,
    57397,
    57529,
420 57660,
    57790,
    57919,
    58047,
    58175,
    58301,
    58426,
    58551,
    58674,
    58797,
430 58918,
    59039,
    59158,
    59277,
    59395,
    59511,
    59627,
    59741,
    59855,
    59967,
440 60079,
    60189,
    60299,
    60407,
    60514,
    60621,
    60726,
    60830,
    60933,
    61036,
450 61137,
    61237,
    61336,
    61433,
    61530,
    61626,
    61720,
    61814,
    61906,
    61998,
```

```
460  62088,
     62177,
     62265,
     62352,
     62438,
     62522,
     62606,
     62688,
     62770,
     62850,
470  62929,
     63007,
     63084,
     63159,
     63234,
     63307,
     63380,
     63451,
     63521,
     63589,
480  63657,
     63723,
     63789,
     63853,
     63916,
     63977,
     64038,
     64098,
     64156,
     64213,
490  64269,
     64323,
     64377,
     64429,
     64480,
     64530,
     64579,
     64627,
     64673,
     64718,
500  64762,
     64805,
     64847,
     64887,
     64926,
```

```
    64964,
    65001,
    65036,
    65071,
    65104,
510 65135,
    65166,
    65196,
    65224,
    65251,
    65277,
    65301,
    65325,
    65347,
    65368,
520 65387,
    65406,
    65423,
    65439,
    65454,
    65467,
    65479,
    65491,
    65500,
    65509,
530 65516,
    65523,
    65527,
    65531,
    65534,
    65535,
    65535,
    65534,
    65531,
    65527,
540 65523,
    65516,
    65509,
    65500,
    65491,
    65479,
    65467,
    65454,
    65439,
    65423,
```

```
550  65406,
     65387,
     65368,
     65347,
     65325,
     65301,
     65277,
     65251,
     65224,
     65196,
560  65166,
     65135,
     65104,
     65071,
     65036,
     65001,
     64964,
     64926,
     64887,
     64847,
570  64805,
     64762,
     64718,
     64673,
     64627,
     64579,
     64530,
     64480,
     64429,
     64377,
580  64323,
     64269,
     64213,
     64156,
     64098,
     64038,
     63977,
     63916,
     63853,
     63789,
590  63723,
     63657,
     63589,
     63521,
     63451,
```

```
    63380,
    63307,
    63234,
    63159,
    63084,
600 63007,
    62929,
    62850,
    62770,
    62688,
    62606,
    62522,
    62438,
    62352,
    62265,
610 62177,
    62088,
    61998,
    61906,
    61814,
    61720,
    61626,
    61530,
    61433,
    61336,
620 61237,
    61137,
    61036,
    60933,
    60830,
    60726,
    60621,
    60514,
    60407,
    60299,
630 60189,
    60079,
    59967,
    59855,
    59741,
    59627,
    59511,
    59395,
    59277,
    59158,
```

```
640  59039,
     58918,
     58797,
     58674,
     58551,
     58426,
     58301,
     58175,
     58047,
     57919,
650  57790,
     57660,
     57529,
     57397,
     57264,
     57130,
     56995,
     56859,
     56723,
     56585,
660  56447,
     56307,
     56167,
     56026,
     55884,
     55742,
     55598,
     55453,
     55308,
     55162,
670  55015,
     54867,
     54718,
     54569,
     54418,
     54267,
     54115,
     53962,
     53809,
     53654,
680  53499,
     53343,
     53186,
     53029,
     52871,
```

```
    52712,
    52552,
    52392,
    52230,
    52068,
690 51906,
    51742,
    51578,
    51413,
    51248,
    51082,
    50915,
    50747,
    50579,
    50410,
700 50240,
    50070,
    49899,
    49728,
    49555,
    49383,
    49209,
    49035,
    48860,
    48685,
710 48509,
    48333,
    48156,
    47978,
    47800,
    47621,
    47442,
    47262,
    47082,
    46901,
720 46719,
    46537,
    46355,
    46172,
    45988,
    45804,
    45620,
    45435,
    45249,
    45063,
```

```
730  44877,
     44690,
     44503,
     44315,
     44127,
     43938,
     43749,
     43559,
     43370,
     43179,
740  42989,
     42798,
     42606,
     42414,
     42222,
     42030,
     41837,
     41644,
     41450,
     41256,
750  41062,
     40868,
     40673,
     40478,
     40283,
     40087,
     39891,
     39695,
     39498,
     39302,
760  39105,
     38908,
     38710,
     38512,
     38315,
     38117,
     37918,
     37720,
     37521,
     37322,
770  37123,
     36924,
     36725,
     36525,
     36326,
```

```
    36126,
    35926,
    35726,
    35526,
    35326,
780 35126,
    34925,
    34725,
    34524,
    34324,
    34123,
    33922,
    33721,
    33521,
    33320,
790 33119,
    32918,
    32717,
    32516,
    32316,
    32115,
    31914,
    31713,
    31512,
    31312,
800 31111,
    30911,
    30710,
    30510,
    30309,
    30109,
    29909,
    29709,
    29509,
    29309,
810 29109,
    28910,
    28710,
    28511,
    28312,
    28113,
    27914,
    27716,
    27518,
    27319,
```

```
820  27121,
     26924,
     26726,
     26529,
     26332,
     26135,
     25938,
     25742,
     25546,
     25350,
830  25155,
     24959,
     24765,
     24570,
     24376,
     24182,
     23988,
     23795,
     23602,
     23409,
840  23217,
     23025,
     22833,
     22642,
     22451,
     22260,
     22070,
     21881,
     21692,
     21503,
850  21314,
     21126,
     20939,
     20752,
     20565,
     20379,
     20193,
     20008,
     19823,
     19639,
860  19455,
     19272,
     19089,
     18907,
     18725,
```

```
    18544,
    18363,
    18183,
    18003,
    17824,
870 17646,
    17468,
    17291,
    17114,
    16938,
    16762,
    16587,
    16413,
    16239,
    16066,
880 15893,
    15722,
    15550,
    15380,
    15210,
    15041,
    14872,
    14704,
    14537,
    14370,
890 14204,
    14039,
    13875,
    13711,
    13548,
    13386,
    13224,
    13063,
    12903,
    12744,
900 12585,
    12427,
    12270,
    12114,
    11958,
    11803,
    11649,
    11496,
    11344,
    11192,
```

```
910  11042,
     10892,
     10742,
     10594,
     10447,
     10300,
     10154,
     10009,
     9865,
     9722,
920  9580,
     9438,
     9298,
     9158,
     9019,
     8881,
     8744,
     8608,
     8473,
     8338,
930  8205,
     8072,
     7941,
     7810,
     7680,
     7552,
     7424,
     7297,
     7171,
     7046,
940  6922,
     6799,
     6677,
     6556,
     6436,
     6317,
     6199,
     6082,
     5966,
     5851,
950  5737,
     5624,
     5512,
     5401,
     5291,
```

```
    5182,
    5074,
    4967,
    4861,
    4757,
960 4653,
    4550,
    4449,
    4348,
    4249,
    4150,
    4053,
    3957,
    3862,
    3768,
970 3675,
    3583,
    3492,
    3402,
    3314,
    3226,
    3140,
    3055,
    2971,
    2888,
980 2806,
    2725,
    2645,
    2567,
    2489,
    2413,
    2338,
    2264,
    2191,
    2120,
990 2049,
    1980,
    1912,
    1845,
    1779,
    1714,
    1651,
    1588,
    1527,
    1467,
```

```
1408,
1351,
1294,
1239,
1185,
1132,
1080,
1029,
980,
932,
885,
839,
795,
751,
709,
668,
628,
590,
553,
516,
481,
448,
415,
384,
354,
325,
298,
271,
246,
222,
199,
178,
157,
138,
121,
104,
89,
74,
62,
50,
39,
30,
22,
15,
10,
```

```
6,
2,
1,
};
```

E.8 hp_filter.asm

```
;
;  Personnel Detector
;  Elliot Ranger
;  Master of Science Thesis
;  Massachusetts Institute of Technology
;  in conjunction with Charles Stark Draper Laboratory
;
;
;*****************************************************************************\
; FILENAME..... hp_filter.asm
; DATE CREATED.. 02/11/2003
; LAST MODIFIED. 02/20/2003
;*****************************************************************************/
;
;   prototype: void hp_filter(DATA *x, ushort nx, DATA *y_previous, DATA *x_previous)
;
;   Entry:  arg0: AR0 - signal input pointer
;            arg1: T0 - number of samples in the input
;            arg2: AR1 - pointer to previous output y
;            arg3: AR2 - pointer to previous input x
;
;


    .def    _hp_filter
    .sect   .text

_hp_filter
    pshm  ST1_55              ; Save ST1, ST2, and ST3
    pshm  ST2_55
    pshm  ST3_55


        or    #0x340,mmap(ST1_55); Set FRCT,SXMD,SATD

    bset  SMUL                ; Set SMUL
```

```
        sub    #1,T0
        mov    T0,BRC0                ; Outer loop couter
        rptblocal lp1-1
        mov    *AR0, T2                              ; save the input value x[i] in T2
        mov    *AR1, T1               ; store y[previous] in T1
40      sub    *AR0,*AR2,AC0          ; subtract x[i] - x[previous]
        mack   T1, #30720, AC0, AC0 ; multiply y[previous] * 30720 and accumulate
        mov    hi(AC0),*AR0           ; store the new output
        mov    T2, *AR2                              ; put the previous x value in AR2
        mov    *AR0+, *AR1             ; put the previous y value in AR1
    lp1



        popm   ST3_55                ; Restore ST1, ST2, and ST3
        popm   ST2_55
50      popm   ST1_55
        ret
        .end
```

E.9 hpi.c

```
    /*
     *  Personnel Detector
     *  Elliot Ranger
     *  Master of Science Thesis
     *  Massachusetts Institute of Technology
     *  in conjunction with Charles Stark Draper Laboratory
     *
     */
    /*************************************************************************\
10  * FILENAME..... hpi.c
    * DATE CREATED.. 07/11/2002
```

```
* LAST MODIFIED. 08/07/2002
\*****************************************************************************/

#include "include.h"

/*****************************************************************************\
*  print_message: takes a string and packs the bytes to be sent over the HPI
*              then gives the command to the MSP to print the data in the buffer
\*****************************************************************************/
void print_message(Char *message)
{
    Uint16 hpi_data_address = _MSP_DATA;
    Uint16 i=0;
    Uint16 temp_word;

    /* the first position is for the length of the string */
    hpi_data_address+=1;

    /* start at the beginning of the message */
    i=0;

    /* takes care of everything but last character */
    while ((message[i] != 0) && (i <= MAX_BUFFER))
    {
        /* load the first character of the word */
        temp_word = (Uint16) (message[i] << 8);
        i++;

        /* load the second character of the word */
        if (message[i] != 0)
        {
            temp_word |= (Uint16) (message[i]);
            i++;
        }
        else
        {
            temp_word |= 0x0000;
            /* do not increment i */
        }

        /* store the word at the address */
        (*(Uint16 *)hpi_data_address) = temp_word;
        hpi_data_address++;
    }
```

```
        /* put a null character in at the end, just in case */
        (*(Uint16 *)hpi_data_address) = 0x0000;

        /* store the length in bytes at the beginning of the buffer */
        (*(Uint16 *)_MSP_DATA) = (Uint16) i;

        /* send code to MSP to print the buffer */
        MSP_COMMAND_REGISTER = PRINT_MESSAGE;
        send_host_interrupt();
    }

    void clear_messages(void)
    {
        Uint16 hpi_data_address = _MSP_DATA;
        Uint16 i=0;

        for (i=0;i<MAX_BUFFER;i++)
        {
            (*(Uint16 *)hpi_data_address) = 0x0000;
            hpi_data_address++;
        }
    }

    /*****************************************************************************\
    *  send_host_interrupt: sends an interrupt to the MSP
    \*****************************************************************************/
    void send_host_interrupt(void)
    {
        Uint16        i;

        /* interrupts on the falling edge of HINT */
        ST3_55 &= (~MSP_HINT);

        /* delay */
        for(i=0;i<=1000;i++);

        /* release the interrupt pin */
        ST3_55 |= MSP_HINT;
    }

    /*****************************************************************************\
    *  get_command: gets the next command from the MSP
    \*****************************************************************************/
    Uint16 get_command(void)
    {
```

```
        while (new_command == FALSE)
        {
            /* Puts the processor in low power mode when it */
            /* is not processing anything                   */
            ICR |= (CACHE | CPU);
            EXECUTE_IDLE;
        }

110     /* reset new_command to prepare for next command*/
        new_command = FALSE;

        /* return command */
        return (DSP_COMMAND_REGISTER);
    }

    /*************************************************************************\
    *  send_hpi_command: sends a command to the MSP
    \*************************************************************************/
120 void send_hpi_command(Uint16 command)
    {
        /* writes a command to the register */
        MSP_COMMAND_REGISTER = command;

        /* sends an interrupt to the MSP to look at the register */
        send_host_interrupt();
    }

    /*************************************************************************\
130 *  hpi_interrupt: turns on LED1 when an interrupt is received, sets a flag
    *              to indicate there is a new command to process and takes the
    *              system out of low power mode.
    \*************************************************************************/
    interrupt void hpi_interrupt(void)
    {
        /* turn on LED1 */
        GPIO_DATA |= LED1;

        /* start timer to turn off LED1 */
140     start_timer1();

        new_command = TRUE;

        /* take out of low power mode */
        ICR &= ~(CACHE | CPU);
    }
```

E.10 include.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*************************************************************************\
* FILENAME..... include.h
* DATE CREATED.. 07/11/2002
* LAST MODIFIED. 08/07/2002
\*************************************************************************/

#include "vc5509.h"
#include "prototype.h"
#include "define.h"
#include "extern.h"
#include "math.h"

#include "dsplib.h"
```

E.11 init_sys.c

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*************************************************************************\
* FILENAME..... init_sys.c
* DATE CREATED.. 07/11/2002
* LAST MODIFIED. 08/07/2002
```

```
\*****************************************************************************/

#include "include.h"

/*****************************************************************************\
*  initialize_system: initialize all the io, interrupts, and low power mode
\*****************************************************************************/
void initialize_system(void)
{

    /* disable global interrupts while configuring */
    INTR_GLOBAL_DISABLE;

    /* set processor to '55x native mode instead of */
    /* '54x compatibility mode (reset value)        */
    ST1_55 &= (~C54CM);

    /* initialize general purpose io */
    initialize_gpio();

    /* initialize interrupts */
    initialize_intr();

    /* clear the registers used to send commands to and from the MSP */
    DSP_COMMAND_REGISTER = 0;
    MSP_COMMAND_REGISTER = 0;

        clear_messages();

    /* enable interrupts for dspint and timer0 */
    IER0 |= (DSPINT | TINT0) ;

    /* enable interrupts for timer1 */
    IER1 |= TINT1;

    /* setup the low power mode configuration */
    initialize_idle_configuration();

    /* turn off all the LEDs */
    GPIO_DATA &= (~LED0);
    GPIO_DATA &= (~LED1);
    GPIO_DATA &= (~LED2);

        /* enable global interrupts */
    INTR_GLOBAL_ENABLE;
```

```
}

/*************************************************************************\
*  initialize_gpio: setup GPIO0-7 as inputs, and enable GPIO on the ports
*              with the LEDs.
\*************************************************************************/
void initialize_gpio(void)
{
    /* setup GPIO0-GPIO7 as inputs */
    IO_DIR = 0x0000;

    /* enable LED GPIO ports */
    GPIO_ENABLE = LED2 | LED1 | LED0;

    /* setup LED GPIO ports as outputs */
    GPIO_DIR = LED2 | LED1 | LED0;

    /* turn off all the LEDs */
    GPIO_DATA &= (~(LED0 | LED1 | LED2));
}

/*************************************************************************\
*  initialize_intr: initialize interrupts and the interrupt vector table
\*************************************************************************/
void initialize_intr(void)
{
    /* disable interrupt enable register 0 */
    IER0 = 0;                                //

        /* disable interrupt enable register 1 */
        IER1 = 0;

        /* clear any pending interrupts on interrupt flag register 0 */
        IFR0 = 0xFFFF;

        /* clear any pending interrupts on interrupt flag register 1 */
        IFR1 = 0xFFFF;

        /* Locate DSP Interrupt vectors at byte address 100h */
        IVPD = 1;

        /* Locate Host Interrupt vectors at byte address 100h */
        IVPH = 1;
}
```

```
/*****************************************************************************\
*  initialize_idle_configuration: shutdown all the peripherals not in use
\*****************************************************************************/
void initialize_idle_configuration(void)
{
        /* enable the idle domain for the serial ports */
        PCR0 |= McBSP_IDLE_EN;
        PCR1 |= McBSP_IDLE_EN;
        PCR2 |= McBSP_IDLE_EN;

        /* enable the idle domain for the I2C */
        ICMDR |= I2C_IDEL_EN;

        /* enable the idle domain for the multimedia card */
        MMCFCLK |= MMC_IDEL_EN;

        /* load the idle configuration register with the peripheral */
        /* and external memory interface domains to shut them down  */
        ICR = (EMIF | PERIPH);

        /* execute the loaded idle configuration */
        EXECUTE_IDLE;

}
```

E.12 lp_filter.asm

```
;
;  Personnel Detector
;  Elliot Ranger
;  Master of Science Thesis
;  Massachusetts Institute of Technology
;  in conjunction with Charles Stark Draper Laboratory
;
;
;*****************************************************************************\
; FILENAME..... lp_filter.asm
; DATE CREATED.. 02/12/2003
; LAST MODIFIED. 02/20/2003
;*****************************************************************************/
;
;       void lp_filter(DATA *x, ushort nx, DATA *y_previous);
;
;   Entry:  arg0: AR0 - signal input pointer
```

```
;             arg1: T0 — number of samples in the input
;             arg2: AR1 — pointer to previous output y
;
;

        .def    _lp_filter
        .sect   .text

_lp_filter
        pshm  ST1_55              ; Save ST1, ST2, and ST3
        pshm  ST2_55
        pshm  ST3_55


            or    #0x340,mmap(ST1_55); Set FRCT,SXMD,SATD

        bset  SMUL                ; Set SMUL


        sub   #1,T0

        mov   T0,BRC0               ; Outer loop couter

        rptblocal lp1-1

        mov   *AR1, T1              ; store y[previous] in T1

        mov   *AR0<<#16,AC0

        mpyk  #8192,AC0,AC0       ; multiply x[i]*8192

        mack  T1, #24576, AC0, AC0 ; multiply y[previous] * 30720 and accumulate

        mov   hi(AC0),*AR0          ; store the new output

        mov   *AR0+, *AR1           ; put the previous y value in AR1

lp1


            popm  ST3_55                ; Restore ST1, ST2, and ST3

        popm  ST2_55

        popm  ST1_55

        ret

        .end
```

E.13 main.c

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... main.c
* DATE CREATED.. 07/11/2002
* LAST MODIFIED. 08/07/2002
\*****************************************************************************/

#include "include.h"
#include "variables.h"



/*****************************************************************************\
*  main: main function which initializes the system and indefinitely
*             keeps processing commands.
\*****************************************************************************/
void main(void)
{

    /* initialize the system */
    initialize_system();
        signal_proc_init();

    /* starts timer0 to flash LED0 while the processor is running */
    start_timer0();

    /* send a message to MSP */
    print_message("VC5509: Booted and Running");

    /* infinite loop */
    while(1)
    {
        process_commands();
    }
}
```

E.14 normalization.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*************************************************************************\
* FILENAME..... normalization.h
* DATE CREATED.. 12/14/2002
* LAST MODIFIED. 02/08/2003
\*************************************************************************/

#define SUMWIDTH 8
#define BLKOUT 4
#define OFFSET 12         /* SUMWIDTH + BLKOUT */
#define THRESHOLD 1
#define SPECTRUM_SIZE 512
#define CENTER_SIZE 9        /* 2*BLKOUT + 1 */
#define SIDES_SIZE 16     /* 2*SUMWIDTH */

/* used as a temporary variable for converting to magnitude */
LDATA magtemp[FREQ_SIZE];

/* extx and r should be the nx + 2*(offset) */
DATA spectemp[SPECTRUM_SIZE+(2*OFFSET)];
DATA norm_spec[SPECTRUM_SIZE+(2*OFFSET)];
```

E.15 proc_cmd.c

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
```

```
 *
 */
/*****************************************************************************\
* FILENAME..... proc_cmd.c
* DATE CREATED.. 07/11/2002
* LAST MODIFIED. 01/15/2003
\*****************************************************************************/

#include "include.h"

/*****************************************************************************\
*  process_commands: processes a command that is received through the hpi
\*****************************************************************************/
void process_commands(void)
{
     switch(get_command())
     {
          case PROCESS_DATA_BLOCK_1:
               block_memloc=(DATA*)BLOCK1_MEM;
               process_block();
               break;

                    case PROCESS_DATA_BLOCK_2:
               block_memloc=(DATA*)BLOCK2_MEM;
               process_block();
               break;

          case CLEAR_PROCESSING_BUFFERS:
               signal_proc_init();
               print_message("VC5509: Buffers Cleared");
               break;

                    case GET_VC5509_VERSION:
               print_message(VERSION);
               break;

                    case COPY_TO_HPI_WRITE:
               hpi_save_code();
               print_message("VC5509: Code Copied to Buffer");
               break;

          default: break;
     }
}
```

E.16 prototype.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... prototype.h
* DATE CREATED.. 07/11/2002
* LAST MODIFIED. 02/25/2003
\*****************************************************************************/
#include "tms320.h"

void initialize_system(void);
void initialize_gpio(void);
void initialize_intr(void);
void initialize_idle_configuration(void);

void LED0_on(void);
void LED0_off(void);
void LED1_on(void);
void LED1_off(void);
void LED2_on(void);
void LED2_off(void);
void LED_test(void);

void send_host_interrupt(void);
interrupt void hpi_interrupt(void);

void start_timer0(void);
void start_timer1(void);
interrupt void timer0_interrupt(void);
interrupt void timer1_interrupt(void);

void process_commands(void);
void print_message (Char *message);
void hpi_save_code(void);
void send_hpi_command(Uint16 command);
void clear_messages(void);

void process_block(void);
```

```
void abs_val(DATA *x, ushort nx);
void signal_proc_init(void);
ushort firdecimate(DATA *x, ushort nx, DATA *h, ushort nh, DATA *r, DATA *dbuffer, ushort dbufindex, ushort m, ushort out
void convert_to_mag(DATA *x, ushort nx, LDATA *r);
ushort Sqrt32(ulong x);
void spectral_normalization(DATA *x, ushort nx, DATA *extx, DATA *r);
void window(DATA *x, ushort nx, ushort *window, DATA *output);
ushort window_buffer(DATA *x, ushort nx, ushort nb, DATA *output, DATA *signal, ushort index);
void scale_output(DATA *x, ushort nx);
ushort detection_algorithm(DATA *x);
void shift_down(DATA *x, ushort nx, ushort shift_value);
void hp_filter(DATA *x, ushort nx, DATA *y_previous, DATA *x_previous);
void lp_filter(DATA *x, ushort nx, DATA *y_previous);
```

E.17 signal_proc.c

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*************************************************************************\
* FILENAME..... signal_proc.c
* DATE CREATED.. 07/12/2002
* LAST MODIFIED. 03/03/2003
\*************************************************************************/

#include "include.h"
#include "filters.h"
#include "hanning.h"
#include "normalization.h"

/* Memory locations */
#pragma DATA_SECTION(h_bp, "iir_coef");
#pragma DATA_SECTION(dbuffer_bp, "iir_delay");

#pragma DATA_SECTION(h_firdec1, "fir_coef");
#pragma DATA_SECTION(h_firdec2, "fir_coef");
#pragma DATA_SECTION(dbuffer_firdec1, "fir_delay");
```

```
#pragma DATA_SECTION(dbuffer_firdec2, "fir_delay");

#pragma DATA_SECTION(hanning, "window_coef");
#pragma DATA_SECTION(wsignal, "window_signal");
#pragma DATA_SECTION(output, "window_data");

#pragma DATA_SECTION(magtemp,"freq_data");
#pragma DATA_SECTION(spectemp,"freq_data");
#pragma DATA_SECTION(norm_spec,"freq_data");

/*****************************************************************************\
*  signal_proc_init: initializes all the signal processing buffers and
*              variables.
\*****************************************************************************/
void signal_proc_init()
{
     ushort i;

     /* clear buffers for bandpass filters */
     iir_hp_y=0;
     iir_hp_x=0;
     iir_lp_y=0;

     /* clear signal buffers for FIR */
     for (i=0; i<(NH_FIR+1); i++)
     {
          dbuffer_firdec1[i]=0;
          dbuffer_firdec2[i]=0;
     }

     /* clear signal buffer index for FIR */
     dbuffer_index1 =0;
     dbuffer_index2 =0;

     /* clear signal buffers for window */
     for (i=0; i<(WINDOW_SIZE); i++)
     {
          wsignal[i]=0;
          output[i]=0;
     }

          /* clear normalization buffers */
     for (i=0; i<(SPECTRUM_SIZE); i++)
     {
          norm_spec[i]=0;
```

```
            spectemp[i]=0;
        }

        /* clear index for window */
        windex=0;

        clear_messages();
    }

/*************************************************************************\
*  process_block: function called by the MSP after it has loaded a complete
*              block of data into the DSP memory.
\*************************************************************************/
    void process_block()
    {
        /* take out the DC offset from the incoming data */
        shift_down(block_memloc, NX_IIR, DC_OFFSET);

        /* IIR bandpass filter */
        hp_filter(block_memloc, NX_IIR, &iir_hp_y, &iir_hp_x);
        lp_filter(block_memloc, NX_IIR, &iir_lp_y);

        /* envelope detect */
        abs_val(block_memloc,NX_IIR);

        /* FIR decimation in two stages (5 and 2) */
        dbuffer_index1=firdecimate(block_memloc, NX_FIR1, h_firdec1, NH_FIR, block_memloc, dbuffer_firdec1, dbuffer_index1, D_
        dbuffer_index2=firdecimate(block_memloc, NX_FIR2, h_firdec2, NH_FIR, block_memloc, dbuffer_firdec2, dbuffer_index2, D_

        /* window buffer circular buffer that keeps aligning the data correctly for the FFT */
        windex=window_buffer(block_memloc, NX_SIZE, WINDOW_SIZE, output, wsignal, windex);

        /* multiply the data by a hanning window before FFT */
        window(output, WINDOW_SIZE, hanning, output);

        /* FFT routine */
        rfft(output,WINDOW_SIZE,SCALE);

        /* convert output magnitude */
            convert_to_mag(output,WINDOW_SIZE,magtemp);

        /* take out DC component and scale output */
        scale_output(output,SPECTRUM_SIZE);

        /* normalize the spectrum */
```

```
        spectral_normalization(output,SPECTRUM_SIZE, spectemp, norm_spec);

            /* detection algorithm */
120     if (detection_algorithm(norm_spec)==TRUE)
        {
            print_message("VC5509: Footsteps Detected");

            /* turn on LED2 */
            GPIO_DATA |= LED2;
        }
        else
        {
            /* turn off LED2 */
130         GPIO_DATA &= (~LED2);
        }
    }

    /*************************************************************************\
    *  shift_down: just subtracts out the DC offset from the A/D conversion
    \*************************************************************************/
    void shift_down(DATA *x, ushort nx, ushort shift_value)
    {
        /* just shift the whole signal down by DC offset */
140     ushort i;
        for (i=0;i<nx;i++) x[i] = x[i] - shift_value;
    }

    /*************************************************************************\
    *  abs_val: takes the absolute value of all the values
    \*************************************************************************/
    void abs_val(DATA *x, ushort nx)
    {
        ushort i;
150     for (i=0;i<nx;i++) x[i] = abs(x[i]);
    }

    /*************************************************************************\
    *  scale_output: removes any DC frequency components by setting all the values
    *              to the mean of the next 20 values divided by 2 to account for the
    *              fact that the data contains a number of peaks and we want it to
    *              be lower.  Then scales the whole data set so the peak value is
    *              near full scale.
    \*************************************************************************/
160 void scale_output(DATA *x, ushort nx)
    {
```

```
        ushort i;
        ushort xval;
        ulong sum=0;


    /* take out DC values by setting to the mean/2 of the first values */
    for (i=5;i<25;i++) sum = sum + x[i];
    xval=sum/40;
    for (i=0;i<5;i++) x[i] = xval;

    /* scale the data up */
    for (i=0;i<nx;i++) x[i] = x[i]<<7;
}

/*************************************************************************\
*  convert_to_mag: takes the real and imaginary data that is outputed from
*              the FFT routine and converts it to a magnitude.
\*************************************************************************/
void convert_to_mag(DATA *x, ushort nx, LDATA *r)
{
    ushort i,j;
    ushort temp;

    /* take absolute value first */
    for (i=0;i<nx;i++) x[i] = abs(x[i]);

    /* first two values are DC and Nyquist */
    temp = x[1];

    j=1;
    for (i=2;i<(nx/2-1);i=i+2)
    {
        r[j]=(ulong)x[i]*x[i]+(ulong)x[i+1]*x[i+1];
        x[j]=Sqrt32(r[j]);
        j++;
    }

    /* store Nyquist */
    x[j] = temp;
}

/*************************************************************************\
*  Sqrt32: fixed point square-root function.  This algorithm was provided
*              by Ken Turkowski Oct 1994.
\*************************************************************************/
```

```
ushort Sqrt32(unsigned long x)
{
     register unsigned long root, remHi, remLo, testDiv, count;

     /* Clear root */
     root = 0;

     /* Clear high part of partial remainder */
     remHi = 0;

     /* Get argument into low part of partial remainder */
     remLo = x;

     /* Loop couter */
     count = 15;

     do
     {
          /* get 2 bits of arg */
          remHi = (remHi<<2)|(remLo>>30); remLo <<=2;

          /* get ready for the next bit in the root */
          root <<= 1;

          /* Test radical */
          testDiv = (root <<1) + 1;
          if (remHi >= testDiv)
          {
               remHi-=testDiv;
               root++;
          }
     } while (count-- != 0);

     return(root);
}

/*****************************************************************************\
*  spectral_normalization: 2 pass normalization function which normalizes
*              the peaks to the noise floor.
\*****************************************************************************/
void spectral_normalization(DATA *x, ushort nx, DATA *extx, DATA *r)
{
     ushort i,j,lbeg,lend,rbeg,rend;
     ulong lsum=0,rsum=0,censum=0,cmpsum=0;
```

```
        /* mirror data to left side and calculate the sum */
        j=OFFSET-1;
        for (i=0;i<OFFSET;i++)
        {
             r[j]=x[i];
             if (i>=BLKOUT) lsum=lsum+x[i];
             j--;
        }
260
        /* copy over the data */
        j=OFFSET;
        for (i=0;i<nx;i++)
        {
             r[j]=x[i];
             j++;
        }

        /* mirror data to right */
270     j=nx+OFFSET;
        for (i=nx-1;i>=(nx-OFFSET);i--)
        {
             r[j]=x[i];
             j++;
        }

        /* get the sum for the right side */
        for (i=(OFFSET+BLKOUT+1);i<=(2*OFFSET);i++) rsum=rsum+r[i];

280     /* get the sum for the center */
        for (i=(OFFSET-BLKOUT);i<=(OFFSET+BLKOUT);i++) censum=censum+r[i];

        /* store value of first point */
        extx[OFFSET]=censum/CENTER_SIZE;
        cmpsum=(lsum+rsum)/SIDES_SIZE;

        /* replace with lower value */
        if ((extx[OFFSET]/THRESHOLD) > cmpsum) extx[OFFSET]=cmpsum;

290     lbeg=0;
        lend=OFFSET-BLKOUT;
        rbeg=OFFSET+BLKOUT+1;
        rend=2*OFFSET+1;

        /* first pass */
        for (i=OFFSET+1;i<(nx+OFFSET);i++)
```

```
        {
             /* move everything over by one */
             lsum=lsum-r[lbeg];
300          lsum=lsum+r[lend];
             rsum=rsum-r[rbeg];
             rsum=rsum+r[rend];

             /* get the new sum for the center */
             censum=censum-r[lend];
             censum=censum+r[rbeg];

             lbeg++;
             lend++;
310          rbeg++;
             rend++;

             /* store value of next point */
             extx[i]=censum/CENTER_SIZE;
             cmpsum=(lsum+rsum)/SIDES_SIZE;

             if ((extx[i]/THRESHOLD) > cmpsum) extx[i]=cmpsum;
        }

320     /* reset all the values */
             lsum=0;rsum=0;censum=0;cmpsum=0;

        /* mirror the data back around the sides */
        j=OFFSET-1;
        for (i=OFFSET;i<(2*OFFSET);i++)
        {
             extx[j]=extx[i];
             if (i>=(OFFSET+BLKOUT)) lsum=lsum+extx[i];
             j--;
330     }

        j=nx+OFFSET;
        for (i=nx+OFFSET-1;i>=nx;i--)
        {
             extx[j]=extx[i];
             j++;
        }

        /* get the sum for the right side */
340     for (i=(OFFSET+BLKOUT+1);i<=(2*OFFSET);i++) rsum=rsum+extx[i];
```

```
        /* get the sum for the center */
        for (i=(OFFSET-BLKOUT);i<=(OFFSET+BLKOUT);i++) censum=censum+extx[i];

        /* store value of first point */
        r[0]=censum/CENTER_SIZE;
        cmpsum=(lsum+rsum)/SIDES_SIZE;

        /* replace with lower value */
350     if ((r[0]/THRESHOLD) > cmpsum) r[0]=cmpsum;

        lbeg=0;
        lend=OFFSET-BLKOUT;
        rbeg=OFFSET+BLKOUT+1;
        rend=2*OFFSET+1;

        /* second pass */
        for (i=1;i<nx;i++)
        {
360         /* move everything over by one */
            lsum=lsum-extx[lbeg];
            lsum=lsum+extx[lend];
            rsum=rsum-extx[rbeg];
            rsum=rsum+extx[rend];

            /* get the new sum for the center */
            censum=censum-extx[lend];
            censum=censum+extx[rbeg];

370         lbeg++;
            lend++;
            rbeg++;
            rend++;

            /* store value of first point */
            r[i]=censum/CENTER_SIZE;
            cmpsum=(lsum+rsum)/SIDES_SIZE;

            if ((r[i]/THRESHOLD) > cmpsum) r[i]=cmpsum;
380     }

        /* divide the normalized spectrum by 64 otherwise the */
        /* results from the integer division are too low */
        /* want to find peaks that are twice as big as background */
        for (i=0;i<nx;i++) r[i] = r[i]<<1;
```

```
    /* all that is left is to subtract the background from */
    /* the original spectrum */
    for (i=0;i<nx;i++){
        r[i]=x[i]-r[i];
    }
}

/*****************************************************************************\
*  detection_algorithm: actual algorithm that classifies whether it is a person
*              looks at frequency and threshold of the first harmonic and if
*              there is a second or third harmonic present as well which is above
*              a certain threshold.
\*****************************************************************************/
ushort detection_algorithm2(DATA *x)
{
    ushort i;
    ushort peak1=0;
    ushort index_peak1;
    ushort second_harmonic=FALSE;
    ushort third_harmonic=FALSE;

    /* first check and see if there is a peak within the */
    /* range of 1-3 Hz above the threshold (each bin is .098 Hz) */
    for (i=10;i<=31;i++)
    {
        if (x[i] > peak1)
        {
            peak1=x[i];
            index_peak1=i;
        }
    }

    if (peak1 < FIRST_THRESHOLD) return FALSE;

    /* now check to see if there is a second or third harmonic */
    /* that is above the threshold */
    for (i=(index_peak1*2)-PEAK_WIDTH;i<=(index_peak1*2)+PEAK_WIDTH;i++)
    {
        if (x[i] > SECOND_THRESHOLD) second_harmonic=TRUE;
    }

    for (i=(index_peak1*3)-PEAK_WIDTH;i<=(index_peak1*3)+PEAK_WIDTH;i++)
    {
        if (x[i] > THIRD_THRESHOLD) third_harmonic=TRUE;
    }
```

```
    if (second_harmonic==TRUE || third_harmonic==TRUE) return TRUE;

    return FALSE;
}

ushort detection_algorithm(DATA *x)
{
    ushort i,j;
    DATA peaks[5];
    ushort index_peaks[5];

        /* clear out values */
        for (i=0;i<5;i++)
        {
                peaks[i]=0;
                index_peaks[i]=0;
        }

        /* find first 5 peaks */
        for (i=0;i<5;i++)
        {
            for (j=10;j<100;j++)
            {
                    if (x[j] > peaks[i])
            {
                peaks[i]=x[j];
                index_peaks[i]=j;
            }
            }

            /* make peak negative so it isn't included again */
        for (j=index_peaks[i]-PEAK_WIDTH;j<=index_peaks[i]+PEAK_WIDTH;j++)
        {
            /* only flip doum positive values */
            if (x[j] > 0) x[j]=-x[j];
            }
        }

    for (i=0;i<5;i++)
    {

        /* first check and see if there is a peak within the */
        /* range of 1-3 Hz above the threshold (each bin is .098 Hz) */
        if ((index_peaks[i] >= 10) &&
```

```
            (index_peaks[i] <= 31) &&
            (peaks[i] >= FIRST_THRESHOLD))
        {
480             /* this is a possible first harmonic */
                /* check and see if there is a second or third harmonic */
                /* which is above the threshold */
                for (j=0;j<5;j++)
                {
                        /* do not want to check agaist itself */
                        if (j!=i)
                        {
                            if ((index_peaks[j] >= (index_peaks[i]*2-PEAK_WIDTH)) &&
                                (index_peaks[j] <= (index_peaks[i]*2+PEAK_WIDTH)) &&
490                             (peaks[j] >= SECOND_THRESHOLD))  return TRUE;

                            if ((index_peaks[j] >= (index_peaks[i]*3-PEAK_WIDTH)) &&
                                (index_peaks[j] <= (index_peaks[i]*3+PEAK_WIDTH)) &&
                                (peaks[j] >= THIRD_THRESHOLD))  return TRUE;

                        }
                }
        }
    }
500
        return FALSE;

}
```

E.18 timer.c

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*************************************************************************\
 10 * FILENAME..... timer.c
* DATE CREATED.. 07/11/2002
* LAST MODIFIED. 08/07/2002
\*************************************************************************/
```

```
#include "include.h"

/*****************************************************************************\
*  start_timer0: timer 0 is used to flash the LED when the VC5509 is booted
*              up and running code.
\*****************************************************************************/
void start_timer0(void)
{
     /* make sure Timer 0 is stopped */
     TCR0 |= TSS;

     /* enable timer loading */
     TCR0 |= TLB;

     /* do not stop timer for breakpoints */
     TCR0 |= FREE;

     /* automatically reload the timer */
     TCR0 |= ARB;

     /* TDDR = F in PRSC0 divide the main clock by 15 */
     PRSC0 = 0x000F;

     /* period register which is loaded into TIM0 */
     PRD0 = 0xFFFF;

     /* disable further loading of register */
     TCR0 &= (~TLB);

     /* start Timer 0 */
     TCR0 &= (~TSS);

}

/*****************************************************************************\
*  start_timer1: timer 1 is used to flash the LED when the VC5509 receives
*              an hpi command from the MSP.
\*****************************************************************************/
void start_timer1(void)
{
     /* make sure Timer 1 is stopped */
     TCR1 |= TSS;

     /* enable timer loading */
     TCR1 |= TLB;
```

```
60
        /* do not stop timer for breakpoints */
        TCR1 |= FREE;

        /* TDDR = F in PRSC0 divide the main clock by 15 */
        PRSC1 = 0x000F;

        /* period register which is loaded into TIM1 */
        PRD1 = 0xFFFF;

70      /* disable further loading of register */
        TCR1 &= (~TLB);

        /* start Timer 1 */
        TCR1 &= (~TSS);

    }

    /*************************************************************************\
    *  timer0_interrupt: only turn LED0 on after the timer counts down 25 times
80  \*************************************************************************/
    interrupt void timer0_interrupt(void)
    {
        if ((GPIO_DATA & LED0) == LED0)     /* LED0 is on */
        {
            /* turn off LED0 */
            GPIO_DATA &= (~LED0);
            timer0_counter=0;
        }
        else     /* LED0 is off */
90      {
            timer0_counter++;

            if (timer0_counter >= 25 )
            {
                /* turn on LED0 */
                GPIO_DATA |= LED0;
            }
        }
    }
100

    /*************************************************************************\
    *  timer1_interrupt: turn LED1 off and stop the timer
    \*************************************************************************/
    interrupt void timer1_interrupt(void)
```

```
{
    /* turn off LED1 */
    GPIO_DATA &= (~LED1);
    TCR1 |= TSS;           // make sure Timer 0 is stopped
}
```

E.19 variables.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... variables.h
* DATE CREATED.. 07/11/2002
* LAST MODIFIED. 08/07/2002
\*****************************************************************************/

/* global variables */
Uint16 timer0_counter=0;
Uint16 new_command=0;

DATA* block_memloc;     /* stores the location of where to begin processing */
```

E.20 vc5509.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
```

```
 *
 */
/*****************************************************************************\
* FILENAME..... vc5509.h
* DATE CREATED.. 07/11/2002
* LAST MODIFIED. 08/07/2002
\*****************************************************************************/

/*************************************\
* typedef declarations
\*************************************/
typedef unsigned char   Uchar;
typedef char             Char;
typedef unsigned short  Uint16;
typedef unsigned long   Uint32;
typedef short            Int16;
typedef long             Int32;

/*************************************\
* helper functions
\*************************************/
#define PREG16(addr) (*(volatile ioport Uint16*)(addr))
#define REG16(addr) (*(volatile Uint16*)(addr))


/*************************************\
* LED declarations
\*************************************/
#define LED0               0x0002u          /* GPIO 9  */
#define LED1               0x0004u          /* GPIO 10 */
#define LED2               0x0020u          /* GPIO 13 */

/*************************************\
* I/O registers
\*************************************/
#define IO_DIR             PREG16(0x3400u)    /* GPIO0-GPIO7 Dir Register               */
#define IO_DATA            PREG16(0x3401u)    /* GPIO0-GPIO7 Data Register              */
#define GPIO_ENABLE        PREG16(0x4403u)    /* GPIO8-GPIO15 Enable Register           */
#define GPIO_DIR           PREG16(0x4404u)    /* GPIO8-GPIO15 Dir Register              */
#define GPIO_DATA          PREG16(0x4405u)    /* GPIO8-GPIO15 Data Register             */

/*************************************\
* Idle registers
\*************************************/
#define ICR                        PREG16(0x0001u)    /* Idle Configuration Register   */
```

```
#define ISRT         PREG16(0x0002u)     /* Idle Status Register           */
#define PCR0                PREG16(0x2812u)     /* Pin Control Register for McBSP0 */
#define PCR1                PREG16(0x2C12u)     /* Pin Control Register for McBSP0 */
#define PCR2                PREG16(0x3012u)     /* Pin Control Register for McBSP0 */
#define ICMDR               PREG16(0x3C09u)     /* I2C Control Register */
#define MMCFCLK             PREG16(0x4800u)     /* Multi-Media Card Clock Control */

/*************************************\
* Idle declarations
\*************************************/
#define CPU                           0x0001u                           /* for shutting down the CPU     */
#define DMA                           0x0002u                           /* for shutting down the DMA     */
#define CACHE               0x0004u                                 /* for shutting down the CACHE  */
#define PERIPH              0x0008u                                 /* for shutting down the PERIPHERALS */
#define CLKGEN              0x0010u                                 /* for shutting down the CLKGEN */
#define EMIF                0x0020u                                 /* for shutting down the External Memory Interface  */
#define McBSP_IDLE_EN 0x0040u                           /* Idle enable bit for McBSP ports */
#define I2C_IDEL_EN 0x1000u               /* Idle enable bit for I2C port */
#define MMC_IDEL_EN 0x0100u                                 /* Idle enable bit for MMC port */

/*************************************\
* Timer registers
\*************************************/
#define TIM0                PREG16(0x1000u)     /* Timer Count Register, Timer #0      */
#define PRD0         PREG16(0x1001u)     /* Period Register, Timer #0           */
#define TCR0                PREG16(0x1002u)     /* Timer Control Register, Timer #0    */
#define PRSC0        PREG16(0x1003u)     /* Timer Prescaler Register, Timer #0 */
#define TIM1                PREG16(0x2400u)     /* Timer Count Register, Timer #1      */
#define PRD1         PREG16(0x2401u)     /* Period Register, Timer #1           */
#define TCR1                PREG16(0x2402u)     /* Timer Control Register, Timer #1    */
#define PRSC1        PREG16(0x2403u)     /* Timer Prescaler Register, Timer #1 */


/*************************************\
* Timer declarations
\*************************************/
#define TSS                           0x0010u                           /* Timer Start/Stop Bit     */
#define TLB                           0x0400u                           /* Timer Load Bit           */
#define FREE                0x0100u                           /* Free running                          */
#define ARB                           0x0020u               /* Automatic Reload of Timer */

/*************************************\
* CPU registers
\*************************************/
#define ST0_55              REG16(0x0002u)              /* Status Register 0                        */
```

```
#define ST1_55              REG16(0x0003u)          /* Status Register 1                        */
#define ST3_55              REG16(0x0004u)          /* Status Register 3                        */
#define IER0                REG16(0x0000u)          /* Interrupt Enable Register 0      */
#define IFR0                REG16(0x0001u)          /* Interrupt Flag Register 0        */
#define IER1                REG16(0x0045u)          /* Interrupt Enable Register 1      */
#define IFR1                REG16(0x0046u)          /* Interrupt Flag Register 1        */
#define IVPD                REG16(0x0049u)          /* Interrupt Vector Pointer for DSP         */
#define IVPH                REG16(0x004Au)          /* Interrupt Vector Pointer for Host */

/*************************************\
* CPU declarations
\*************************************/
#define INTM                0x0800u         /* Global Interrupts */
#define MSP_HINT            0x1000u         /* HPI Interrupt bit */
#define DSPINT              0x0400u         /* DSP Interrupt bit */
#define TINT0               0x0010u         /* Timer 0 Interrupt bit */
#define TINT1               0x0040u         /* Timer 1 Interrupt bit */
#define C54CM               0x0020u         /* C54 Compatibility mode */

/*************************************\
* HPI declarations
\*************************************/
#define DSP_COMMAND_REGISTER  REG16(0x0100u)          /* Location to Read commands from MSP */
#define MSP_COMMAND_REGISTER  REG16(0x0101u)    /* Location to Write commands to the MSP */
#define _MSP_DATA                   0x0102u             /* Memory location */
#define MSP_DATA                    REG16(0x0102u)    /* Location to Write data for the MSP */
#define DSP_HPI_WRITE          REG16(0x0A00u)    /* Location to read data from the MSP */

/*****************************************************************/
/* INTR_ENABLE - enables all masked interrupts by resetting INTM  */
/*                 bit in Status Register 1                      */
/*****************************************************************/
#define INTR_GLOBAL_ENABLE \
         asm("\t RSBX\t INTM"); \
         asm("\tNOP");


/*****************************************************************/
/* INTR_DISABLE - disables all masked interrupts by setting INTM  */
/*                  bit in Status Register 1                     */
/*****************************************************************/
#define INTR_GLOBAL_DISABLE \
                 asm("\t SSBX\t INTM"); \
         asm("\tNOP");
```

```
/*****************************************************************/
/* EXECUTE_IDLE - tells the processor to execute the current idle */
/*                  configuration held in ICR                     */
/*****************************************************************/
#define EXECUTE_IDLE \
                  asm("\t IDLE\t"); \
```

E.21 vectors.asm

```
;
;  Personnel Detector
;  Elliot Ranger
;  Master of Science Thesis
;  Massachusetts Institute of Technology
;  in conjunction with Charles Stark Draper Laboratory
;
;
;*****************************************************************************\
; FILENAME..... vectors.asm
; DATE CREATED.. 07/11/2002
; LAST MODIFIED. 08/07/2002
;*****************************************************************************/

 .ref _hpi_interrupt, _timer0_interrupt, _timer1_interrupt
 .def  rsv, no_isr   ; symbols defined in this file

 .sect "vectors"                   ; interrupt vectors
rsv              .ivec no_isr
nmi              .ivec no_isr     ; non-maskable interrupt

int0     .ivec no_isr     ; External interrupt #0

int2     .ivec no_isr     ; External interrupt #2

tint0    .ivec _timer0_interrupt             ; Timer 0

rint0    .ivec no_isr     ; McBSP #0 receive

rint1    .ivec no_isr     ; McBSP #1 receive

xint1    .ivec no_isr     ; McBSP #1 transmit

usb          .ivec no_isr          ; USB interrupt
```

```
    dmac1    .ivec no_isr      ; DMA Channel #1
    dspint   .ivec _hpi_interrupt       ; Interupt from host
30  int3     .ivec no_isr     ; External interrupt #3
    rint2    .ivec no_isr     ; McBSP #2 receive
    xint2    .ivec no_isr     ; McBSP #2 transmit
    dmac4    .ivec no_isr      ; DMA Channel #4
    dmac5    .ivec no_isr      ; DMA Channel #5
    int1     .ivec no_isr     ; External interrupt #1
    xint0    .ivec no_isr     ; McBSP #0 transmit
    dmac0    .ivec no_isr      ; DMA Channel #0
    int4     .ivec no_isr     ; External interrupt #4
    dmac2    .ivec no_isr      ; DMA Channel #2
40  dmac3    .ivec no_isr      ; DMA Channel #3
    tint1    .ivec _timer1_interrupt             ; Timer 1
    int5     .ivec no_isr     ; External interrupt #5
    berr     .ivec no_isr     ; Bus error interrupt
    dlog     .ivec no_isr     ; Data log interrupt
    rtos     .ivec no_isr     ; Real-time operating system interrupt
    sint27   .ivec no_isr      ; Software interrupt #27
    sint28   .ivec no_isr      ; Software interrupt #28
    sint29   .ivec no_isr      ; Software interrupt #29
    sint30   .ivec no_isr      ; Software interrupt #30
50  sint31   .ivec no_isr      ; Software interrupt #31


    ;  Interrupt service routine:


      .text
    no_isr:  B no_isr          ; default ISR
```

E.22 wbuffer.asm

```
;
;  Personnel Detector
;  Elliot Ranger
;  Master of Science Thesis
;  Massachusetts Institute of Technology
;  in conjunction with Charles Stark Draper Laboratory
;
;
;*****************************************************************************\
; FILENAME..... wbuffer.asm
; DATE CREATED.. 08/23/2002
; LAST MODIFIED. 01/15/2003
;*****************************************************************************/
;
;   window_buffer.asm
;
;   prototype: ushort window_buffer(DATA *x, ushort nx, ushort nb,
;                           DATA *output, DATA *signal, ushort index);
;
;   Entry:  arg0: AR0 - signal input buffer pointer
;            arg1: T0 - number of samples in the input
;            arg2: T1 - buffer size
;            arg3: AR1 - output
;            arg4: AR2 - signal buffer pointer
;            arg5: AR3 - signal buffer index
;
;   Return: T0 = signal buffer index
;


      .def    _window_buffer
      .sect   .text

_window_buffer
     pshm  ST1_55              ; Save ST1, ST2, and ST3
     pshm  ST2_55
     pshm  ST3_55


          or     #0x340,mmap(ST1_55); Set FRCT,SXMD,SATD

     bset  SMUL                ; Set SMUL
     mov   mmap(AR2),BSA23     ; Set base address for AR2
     mov   mmap(T1),BK03       ; Set signal buffer size
```

```
        mov    AR3,AR2              ; AR2 signal buffer index
        or     #0x4,mmap(ST2_55)   ; AR2 circular pointer

    ; First block equal to size of input so we can save
    ; the starting point for the next we run
        mov    T0,T3
        sub    #1,T0
        mov    T0,BRC0                  ; Outer loop couter
50      rptblocal lp1-1
        mov    *AR2+,*AR1+                    ; load first part
    lp1

            mov    AR2,T0              ; save next starting point

    ; Do all other calculations which don't have the new
    ; input values
           sub   T3,T1             ; subtract beginning and ending
           sub   T3,T1             ;   samples
60      sub   #1,T1
        mov   T1,BRC0           ; Outer loop couter
        rptblocal lp2-1
           mov   *AR2+,*AR1+       ; load other values
    lp2

    ; Lastly add new data to the end of the signal buffer
    ; and do the calculations
        sub   #1,T3
        mov   T3,BRC0
70      rptblocal lp3-1
        mov   *AR0+,*AR2         ; Put new samples into signal buffer
```

```
        mov   *AR2+,*AR1+        ; create array
lp3

    popm  ST3_55           ; Restore ST1, ST2, and ST3
    popm  ST2_55
    popm  ST1_55
    ret
    .end
```

E.23 window.asm

```
;
;  Personnel Detector
;  Elliot Ranger
;  Master of Science Thesis
;  Massachusetts Institute of Technology
;  in conjunction with Charles Stark Draper Laboratory
;
;
;*****************************************************************************\
; FILENAME..... window.asm
; DATE CREATED.. 07/11/2002
; LAST MODIFIED. 01/15/2003
;*****************************************************************************/
;
;   window.asm
;
;   prototype: void window(DATA *x, ushort nx, ushort *window, DATA *output);
;
;   Entry:  arg0: AR0 - signal input buffer pointer
;            arg1: T0 - number of samples in the input buffer
;            arg2: AR1 - window array pointer
;            arg3: AR2 - output
;
;


    .def    _window
    .sect   .text
```

```
_window
30      pshm  ST1_55              ; Save ST1, ST2, and ST3
        pshm  ST2_55
        pshm  ST3_55


            or    #0x340,mmap(ST1_55); Set FRCT,SXMD,SATD

        bset  SMUL                ; Set SMUL



; just multiply the window by the input buffer and store the output

        sub   #1,T0
40      mov   T0,BRC0                   ; Outer loop couter
        rptblocal lp1-1
        mpym   uns(*AR1+),*AR0+,AC0    ; multiply the beginning values
        mov   hi(AC0),*AR2+
lp1



        popm  ST3_55              ; Restore ST1, ST2, and ST3
        popm  ST2_55
        popm  ST1_55
50      ret
        .end
```

Appendix F

MSP430F149 Source Code

F.1 boot_dsp.c

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... boot_dsp.c
* DATE CREATED.. 07/29/2002
* LAST MODIFIED. 08/06/2002
\*****************************************************************************/

#include "include.h"

/*****************************************************************************\
*  boot_dsp: copies the VC5509 boot code and program code that was stored in
*              the MSP flash to the VC5509 and then instructs the processor where
*              to start loading code from.
\*****************************************************************************/
void boot_dsp(void)
{
    Uint16 flash_data;
    Uint16 flash_memory_address;

    enable_hpi();
```

```
        /* Load Boot Code                                                  */
30      /* VC5509 memory location 0x0180 - 0x0200 (word addressing) */
        /* MSP Flash memory 0x6000 - 0x6100 (byte addressing)       */
        write_hpi_address_register(0x0180);

        for (flash_memory_address = 0x6000;
             flash_memory_address < 0x6100;
             flash_memory_address+=0x0002)
        {
              flash_data = read_flash(flash_memory_address);
              write_hpi_data_word(flash_data,TRUE);
40      }

        /* Load Program Code                                               */
        /* VC5509 memory location 0x0A00 - 0x1A80 (word addressing) */
        /* MSP Flash memory 0x6100 - 0x8200 (byte addressing)       */
        write_hpi_address_register(0x0D00);

        for (flash_memory_address = 0x6100;
              flash_memory_address <= 0xC600;
              flash_memory_address+=0x0002)
50      {
              flash_data = read_flash(flash_memory_address);
              write_hpi_data_word(flash_data,TRUE);
        }

        /* VC5509 location to start executing boot code          */
        /* The VC5509 uses byte addressing for the program code */
        write_hpi_address_register(0x0061);
        write_hpi_data_word(0x0300,FALSE);
        write_hpi_address_register(0x0060);
60      write_hpi_data_word(0xFF00,FALSE);

    }

    /*************************************************************************\
    *  copy_dsp_code_to_flash: first erases the flash memory used to store the
    *              VC5509 code.  Then copies all the code from the HPI Write section
    *              of the VC5509's memory to the MSP's flash.
    \*************************************************************************/
    void copy_dsp_code_to_flash(void)
70  {
        Uint16 flash_memory_address;
        Uint16 flash_data;
```

```
    /* Disable global interrupts */
    _DINT();

    /* Erase flash memory used to store VC5509 code */
    for (flash_memory_address = 0x6000;
         flash_memory_address <= 0xC600;
80       flash_memory_address+=0x0100)
    {
         erase_flash(flash_memory_address);
    }

    /* Save VC5509 boot code and program code to MSP flash      */
    /* VC5509 memory location 0x0D00 - 0x4000 (word addressing) */
    /* MSP Flash memory 0x6000 - 0xC600 (byte addressing)       */
    write_hpi_address_register(0x0D00);

90  for (flash_memory_address = 0x6000;
         flash_memory_address <= 0xC600;
         flash_memory_address+=0x0002)
    {

         flash_data = read_hpi_data_word(TRUE);
         write_flash(flash_memory_address, flash_data);
    }

    /* Enable global interrupts */
100 _EINT();
}

/*****************************************************************************\
*  verify_dsp_code: compares the code stored in MSP flash to the data in
*              VC5509 HPI Write section and checks for any differences.
\*****************************************************************************/
void verify_dsp_code(void)
{
    Uint16 vc55_data;
110 Uint16 flash_data;
    Uint16 flash_memory_address;
    Uchar buffer[MAX_LINE_LENGTH];
    Uint16 compare=0;

    /* Set address to beginning of HPI Write Section of VC5509 memory */
    write_hpi_address_register(0x0D00);
```

```
        /* Compare flash memory contents with VC5509 HPI write */
        /* section and increment compare if they are not equal */
120     for (flash_memory_address = 0x6000;
              flash_memory_address <= 0xC600;
              flash_memory_address+=0x0002)
        {

              vc55_data = read_hpi_data_word(TRUE);
              flash_data = read_flash(flash_memory_address);
              if (flash_data != vc55_data)
              {
                 compare++;
130           }
        }

        /* Print a message displaying the results */
        if (compare > 0)
        {
              sprintf(buffer,">>> %d Errors in compare of memory",compare);
              add_message_to_display(buffer);
        }
        else add_message_to_display("Memory verified correctly");
140

    }
```

F.2 comms.c

```
    /*
     *  Personnel Detector
     *  Elliot Ranger
     *  Master of Science Thesis
     *  Massachusetts Institute of Technology
     *  in conjunction with Charles Stark Draper Laboratory
     *
     */
    /*************************************************************************\
 10 * FILENAME..... comms.c
    * DATE CREATED.. 04/18/2002
    * LAST MODIFIED. 08/06/2002
    \*************************************************************************/

    #include "include.h"
```

```
/*****************************************************************************\
*  print_message: outputs a message to the RS232 port and the RF port if
*              RF is enabled.
\*****************************************************************************/
void print_message(Uchar *message)
{
    RF_CONTROL_TYPE rf_in;
    Uint16 i=0;
    Uint16 start_timer;

    /* turn on RS232 transmitter */
    msp_port5.byte = P5OUT;
    msp_port5.bit.rs232_enable = 1;
    P5OUT = msp_port5.byte;

    /* use timer A to make sure the transmitter is out of shutdown */
    start_timer = TAR;
    while (((TAR - start_timer) & TIMER_MASK) < 50);

    i = 0;
    while (message[i] != 0)
    {
        /* wait for TX ready on RS232 */
        while ((IFG1 & UTXIFG0) != UTXIFG0);

        /* Transmit character on RS232 */
        U0TXBUF = message[i];

        if (rf_ena == TRUE)
        {
            /* wait for TX ready on RF */
            while ((IFG2 & UTXIFG1) != UTXIFG1);

            /* wait for CTS on RF */
            rf_in.byte = P3IN;
            while (rf_in.bit.rf_cts) rf_in.byte=P3IN;

            /* Transmit character on RF */
            U1TXBUF = message[i];
        }
        i++;
    }

#if LOW_POWER_MODE==TRUE
```

```
    /* use timer A to make sure all the characters get sent out */
    start_timer = TAR;
    while (((TAR - start_timer) & TIMER_MASK) < 50);

    /* turn off RS232 transmitter */
    msp_port5.byte = P5OUT;
    msp_port5.bit.rs232_enable = 0;
    P5OUT = msp_port5.byte;
#endif
}

/*****************************************************************************\
*  rcv1_handler: interrupt handler when a character is received on RS232
*              places the character in a buffer and then takes the system out
*              of low power mode if enabled
\*****************************************************************************/
interrupt [UARTORX_VECTOR] void rcv1_handler(void)
{
    in_buffer[buffer_write_pos] = U0RXBUF;
    buffer_write_pos++;

    /* loops to beginning of buffer */
    if (buffer_write_pos >= MAX_BUFFER) buffer_write_pos=0;

#if LOW_POWER_MODE==TRUE
    clear_low_power_mode(LPM1_bits);
#endif
}

/*****************************************************************************\
*  rcv2_handler: interrupt handler when a character is received on RF
*              places the character in a buffer and then takes the system out
*              of low power mode if enabled
\*****************************************************************************/
interrupt [UART1RX_VECTOR] void rcv2_handler(void)
{
    in_buffer[buffer_write_pos] = U1RXBUF;
    buffer_write_pos++;

    /* loops to beginning of buffer */
    if (buffer_write_pos >= MAX_BUFFER) buffer_write_pos=0;

#if LOW_POWER_MODE==TRUE
    clear_low_power_mode(LPM1_bits);
#endif
```

```
}

/*************************************************************************\
*  get_char: returns the next character in the buffer.  Puts system in low power
*              mode while waiting for the next character, if enabled.
\*************************************************************************/
Uchar get_char(void)
{
    Uchar byte;

#if LOW_POWER_MODE==TRUE
    while (buffer_write_pos == buffer_read_pos) LPM1;
#else
    while (buffer_write_pos == buffer_read_pos);
#endif

    byte = in_buffer[buffer_read_pos];
    buffer_read_pos++;

    /* loops to beginning of buffer */
    if (buffer_read_pos >= MAX_BUFFER) buffer_read_pos = 0;

    return (byte);

}
```

F.3 define.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*************************************************************************\
* FILENAME..... define.h
* DATE CREATED.. 04/18/2002
* LAST MODIFIED. 08/06/2002
\*************************************************************************/

#define VERSION "MSP430F149 VERSION 1.04  3/04/2003"
```

```
#define  TRUE  1
#define  FALSE 0

#define CR  0x0D
#define LF  0x0A
#define ESC 0x1B
#define BS  0x08
#define DEL 0x7F

#define MAX_LINE_LENGTH   64
#define MAX_DISPLAY_LINES 3
#define MAX_BUFFER         64
#define TIMER_MASK         0x07FF

/* set to true to enable low power modes,  */
/* set to falsh to disable low power modes */
#define LOW_POWER_MODE TRUE

/* controls the sampling rate of ADC with timer B  */
/* 0x0021 is approx 1 KHz, 0x0042 is approx 500 Hz */
#define SAMPLING_RATE      0x0021;

/* VC5509 memory location to read commands */
#define DSP_COMMAND_REGISTER 0x0100
#define MSP_COMMAND_REGISTER 0x0101
#define DSP_HPI_WRITE_1         0x1000
#define DSP_HPI_WRITE_2         0x1500
#define DATA_BLOCK_1             0x0001
#define DATA_BLOCK_2             0x0002

/* MSP Command Codes */
#define PRINT_MESSAGE   0x0100
#define STOP_PROCESSING 0x0200

/* VC5509 Command Codes */
#define PROCESS_DATA_BLOCK_1       0x0100
#define PROCESS_DATA_BLOCK_2       0x0200
#define CLEAR_PROCESSING_BUFFERS   0x0300
#define GET_VC5509_VERSION          0x0900
#define COPY_TO_HPI_WRITE           0x0999
```

F.4 display.c

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... display.c
* DATE CREATED.. 07/31/2002
* LAST MODIFIED. 08/06/2002
\*****************************************************************************/

#include "include.h"

/*****************************************************************************\
*  add_message_to_display: adds a message to the display area of the screen
\*****************************************************************************/
void add_message_to_display(Uchar *message)
{
    Uint16 i = 0,display_pos = 0;
    Uint16 done=FALSE;

    do
    {
        switch(message[i])
        {
            case 0 : /* take care of the null by putting a new line at the end */
                    display_lines[display_current_line][display_pos] = CR;
                    display_pos++;
                    display_lines[display_current_line][display_pos] = LF;
                    display_pos++;
                    display_lines[display_current_line][display_pos] = 0;

                    /* increment current line for the next write */
                    display_current_line++;

                    /* loop around if reach maximum number of lines */
                    if (display_current_line == MAX_DISPLAY_LINES) display_current_line = 0;

                    done = TRUE;
                    break;
```

```
            case CR : /* check and see if next character is a line feed */
                      if (message[i+1] == LF )
                      {
                         /* put the line feed at the end of the line */
                         display_lines[display_current_line][display_pos] = CR;
50                       display_pos++;
                         i++;
                         display_lines[display_current_line][display_pos] = LF;
                         display_pos++;
                         i++;
                         display_lines[display_current_line][display_pos] = 0;

                         /* increment current line */
                         display_current_line++;

60                       /* loop around if reach maximum number of lines */
                         if (display_current_line == MAX_DISPLAY_LINES) display_current_line = 0;

                         /* start at the beginning of the next line */
                         display_pos = 0;
                      }
                      else
                      {
                         /* just copy the letter */
                         display_lines[display_current_line][display_pos] = message[i];
70                       display_pos++;
                         i++;
                      }
                      break;

            default: /* just copy the letter */
                      display_lines[display_current_line][display_pos] = message[i];
                      display_pos++;
                      i++;
                      break;
80       }
    } while ((done == FALSE) && (display_pos < MAX_LINE_LENGTH));

    if (display_pos == MAX_LINE_LENGTH)
    {
        display_lines[display_current_line][(MAX_LINE_LENGTH-3)] = CR;
        display_lines[display_current_line][(MAX_LINE_LENGTH-2)] = LF;
        display_lines[display_current_line][(MAX_LINE_LENGTH-1)] = 0;
    }
```

```
}

/*****************************************************************************\
*  clear_display_area: clears all the display lines and resets
*              display_current_line back to the beginning.
\*****************************************************************************/
void clear_display_area(void)
{
   Uint16 i;

   /* do for each display line */
   for (i=0;i<MAX_DISPLAY_LINES;i++)
   {
      display_lines[i][0] = CR;
      display_lines[i][1] = LF;
      display_lines[i][2] = 0;
   }

   display_current_line = 0;
}

/*****************************************************************************\
*  print_display_area: function that goes through and prints all of the
*              display lines in the correct order
\*****************************************************************************/
void print_display_area(void)
{
    Uint16 line_to_print;

    line_to_print = display_current_line;

    do
    {
        print_message(display_lines[line_to_print]);
        line_to_print++;

        /* start at the beginning when line_to_print reaches the last line */
        if (line_to_print == MAX_DISPLAY_LINES) line_to_print = 0;

    } while (line_to_print != display_current_line);

}
```

F.5 extern.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... extern.h
* DATE CREATED.. 04/18/2002
* LAST MODIFIED. 08/06/2002
\*****************************************************************************/

extern MSP_PORT3_TYPE msp_port3;
extern MSP_PORT5_TYPE msp_port5;
extern HPIC_CONTROL_TYPE hpic;
extern Uint32 time;
extern Uint16 seconds;
extern Uint16 minutes;
extern Uint16 hours;
extern Uint16 days;
extern Uchar in_buffer[MAX_BUFFER];
extern Uint16 buffer_write_pos;
extern Uint16 buffer_read_pos;
extern Uint16 rf_ena;
extern Uchar display_lines[MAX_DISPLAY_LINES][MAX_LINE_LENGTH];
extern Uint16 display_current_line;
extern Uint16 dsp_data_address;
extern Uint16 dsp_memory_block;
```

F.6 flash.c

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
```

```
/*****************************************************************************\
* FILENAME..... flash.c
* DATE CREATED.. 07/17/2002
* LAST MODIFIED. 08/06/2002
\*****************************************************************************/

#include "include.h"

/*****************************************************************************\
*  erase_flash: erases one segment of flash memory
\*****************************************************************************/
void erase_flash(Uint16 address)
{
     /* wait for busy to be reset */
     while ((FCTL3 & BUSY) == BUSY);

     /* clear lock bit */
     FCTL3 = FWKEY;

     /* enable erase from flash */
     FCTL1 = (FWKEY | ERASE);

     /* address to erase */
     (*(unsigned short *)address) = 0x0000;

     /* wait for busy to be reset */
     while ((FCTL3 & BUSY) == BUSY);

     /* reset write bit */
     FCTL1 = FWKEY;

     /* change lock bit */
     FCTL3 ^= ( FXKEY | LOCK);
}

/*****************************************************************************\
*  write_flash: writes one word of data to flash memory
\*****************************************************************************/
void write_flash(unsigned short address, unsigned short data)
{
     /* wait for busy to be reset */
     while ((FCTL3 & BUSY) == BUSY);

     /* Clear lock bit */
     FCTL3 = FWKEY;
```

```
    /* enable write to flash */
    FCTL1 = (FWKEY | WRT);

    /* data to write to flash */
    (*(Uint16 *)address) = data;

    /* wait for busy to be reset */
    while ((FCTL3 & BUSY) == BUSY);

    /* reset write bit */
    FCTL1 = FWKEY;

    /* change lock bit */
    FCTL3 ^= (FXKEY | LOCK);
}

/*****************************************************************************\
*  read_flash: returns the memory contents of the address
\*****************************************************************************/
Uint16 read_flash(Uint16 address)
{
    return(*(Uint16 *)address);
}
```

F.7 hpi.c

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... hpi.c
* DATE CREATED.. 04/18/2002
* LAST MODIFIED. 08/06/2002
\*****************************************************************************/

#include "include.h"
```

```
/*****************************************************************************\
*  enable_hpi: sets up MSP ports for HPI use, and enables host port interrupt
\*****************************************************************************/
void enable_hpi(void)
{
    HPI_CONTROL_TYPE       hpi_ctl_out;

    /*  Reset the HPIC register values */
    hpic.byte =  0;
    hpic.bit.reset = 1;
    write_hpi_control_register(hpic.byte);

    /* Enable HPI interrupt on the falling edge*/
    P1IES = 1;
    P1IFG = 0;
    P1IE  = 1;

    /* hold the host data strobe line high */
    hpi_ctl_out.byte    = 0;
    hpi_ctl_out.bit.hds = 1;
    P1OUT                = hpi_ctl_out.byte;
}

/*****************************************************************************\
*  disable_hpi: disables host port interrupt and sets hpi port as output
\*****************************************************************************/
void disable_hpi(void)
{
    HPI_CONTROL_TYPE       hpi_ctl_out;

    /* disable HPI interrupt and clear interrupt flag */
    P1IE  = 0;
    P1IFG = 0;
    P1IES = 0;

    /* Clear all lines on the control port */
    hpi_ctl_out.byte  = 0;
    P1OUT              = hpi_ctl_out.byte;

    /* reset the data lines */
    P2DIR            = 0xFF;
    P4DIR            = 0xFF;
    P2OUT            = 0x0;
    P4OUT            = 0x0;
```

```
}

/*****************************************************************************\
*  write_hpi_control_register: writes a word to the HPI control register
*              in the VC5509.
\*****************************************************************************/
void write_hpi_control_register(Uint16 data)
{
    Uint16 i;
    HPI_CONTROL_TYPE       hpi_ctl_out;
    HPI_CONTROL_TYPE    hpi_ctl_in;

    /* Configure PORT2 and PORT4 as outputs */
    P2DIR        = 0xFF;
    P4DIR         = 0xFF;

    /* Initialize hpi_ctl_out */
    hpi_ctl_out.byte  = P1OUT;

    /* Set to write mode, word addressing, HPIC register */
    hpi_ctl_out.bit.rd_wr   = 0;
    hpi_ctl_out.bit.control = 0;
    hpi_ctl_out.bit.byte_en = 0;
    hpi_ctl_out.bit.hds     = 1;
    P1OUT                      = hpi_ctl_out.byte;

    /* wait for ready from VC5509 */
    hpi_ctl_in.byte = P1IN;
    while (!hpi_ctl_in.bit.ready) hpi_ctl_in.byte = P1IN;

    /* Assert the data strobe low */
    hpi_ctl_out.bit.hds = 0;
    P1OUT                = hpi_ctl_out.byte;

    /* Write the LSB of the HPIC data register */
    P2OUT   = (char) data;

    /* Write the MSB of the HPIC data register */
    P4OUT   = (char) (data >> 8);

    /* small loop to make sure the data lines are stable */
    for(i=0;i<=2;i++);

    /* Release the data strobe */
    hpi_ctl_out.bit.hds   = 1;
```

```
    P1OUT                = hpi_ctl_out.byte;
}

/*****************************************************************************\
*  write_hpi_address_register: writes a word to the HPI address register
*              in the VC5509.
\*****************************************************************************/
void write_hpi_address_register(Uint16 data)
{
    Uint16 i;
    HPI_CONTROL_TYPE       hpi_ctl_out;
    HPI_CONTROL_TYPE       hpi_ctl_in;

    /* Configure PORT2 and PORT4 as outputs */
    P2DIR        = 0xFF;
    P4DIR         = 0xFF;

    /* Initialize hpi_ctl_out */
    hpi_ctl_out.byte  = P1OUT;

    /* Set to write mode, word addressing, HPIA register */
    hpi_ctl_out.bit.rd_wr   = 0;
    hpi_ctl_out.bit.control = 2;
    hpi_ctl_out.bit.byte_en = 0;
    hpi_ctl_out.bit.hds     = 1;
    P1OUT                     = hpi_ctl_out.byte;

    /* wait for ready from VC5509 */
    hpi_ctl_in.byte = P1IN;
    while (!hpi_ctl_in.bit.ready) hpi_ctl_in.byte = P1IN;


    /* Assert the data strobe low */
    hpi_ctl_out.bit.hds = 0;
    P1OUT                = hpi_ctl_out.byte;


    /* Write the LSB of the address */
    P2OUT   = (char) data;

    /* Write the MSB of the address */
    P4OUT   = (char) (data >> 8);

    /* small loop to make sure the data lines are stable */
    for(i=0;i<=2;i++);
```

```
    /* Release the data strobe */
    hpi_ctl_out.bit.hds = 1;
    P1OUT                = hpi_ctl_out.byte;
}

/*************************************************************************\
*  write_hpi_data_word: writes a data word to the address location stored
*              in the HPI address register.  Increments the address register
*              if auto_incr is true.
\*************************************************************************/
void write_hpi_data_word(Uint16 data, Uint16 auto_incr)
{
    Uint16 i;
    HPI_CONTROL_TYPE       hpi_ctl_out;
    HPI_CONTROL_TYPE       hpi_ctl_in;

    /* Configure PORT2 and PORT4 as outputs */
    P2DIR        = 0xFF;
    P4DIR         = 0xFF;

    /* Initialize hpi_ctl_out */
    hpi_ctl_out.byte  = P1OUT;

    /* Set to write mode, word addressing, HPID register */
    hpi_ctl_out.bit.rd_wr   = 0;
    hpi_ctl_out.bit.byte_en = 0;
    hpi_ctl_out.bit.hds     = 1;

    if (auto_incr == TRUE)
    {
        hpi_ctl_out.bit.control = 1;
    }
    else
    {
        hpi_ctl_out.bit.control = 3;
    }
    P1OUT                    = hpi_ctl_out.byte;

    /* wait for ready from VC5509*/
    hpi_ctl_in.byte = P1IN;
    while (!hpi_ctl_in.bit.ready) hpi_ctl_in.byte = P1IN;

    /* Assert the data strobe low */
    hpi_ctl_out.bit.hds = 0;
```

```
    P1OUT                = hpi_ctl_out.byte;

    /* Write the LSB of the data */
    P2OUT = (char)data;
200
    /* Write the MSB of the data */
    P4OUT = (char)(data >> 8);

    /* small loop to make sure the data lines are stable */
    for(i=0;i<=2;i++);

    /* Release the data strobe */
    hpi_ctl_out.bit.hds = 1;
    P1OUT                = hpi_ctl_out.byte;
210 }

/*****************************************************************************\
*  read_hpi_data_word: reads the data word at the address location stored
*              in the HPI address register.  Increments the address register
*              if auto_incr is true.
\*****************************************************************************/
Uint16 read_hpi_data_word(Uint16 auto_incr)
{
    Uint16 i,hpi_data;
220 HPI_CONTROL_TYPE       hpi_ctl_out;
    HPI_CONTROL_TYPE       hpi_ctl_in;

    /* Configure PORT2 and PORT4 as inputs */
    P2DIR        = 0x0;
    P4DIR         = 0x0;

    /* Initialize hpi_ctl_out */
    hpi_ctl_out.byte  = P1OUT;

230 /* Set to read mode, word addressing, HPID register */
    hpi_ctl_out.bit.rd_wr    = 1;
    hpi_ctl_out.bit.byte_en = 0;
    hpi_ctl_out.bit.hds      = 1;

    if (auto_incr == TRUE)
    {
        hpi_ctl_out.bit.control    = 1;
    }
    else
240 {
```

```
            hpi_ctl_out.bit.control    = 3;
        }
        P1OUT    = hpi_ctl_out.byte;

        /* wait for ready from VC5509 */
        hpi_ctl_in.byte = P1IN;
        while (!hpi_ctl_in.bit.ready) hpi_ctl_in.byte = P1IN;

        /* Assert the data strobe low */
        hpi_ctl_out.bit.hds = 0;
        P1OUT                 = hpi_ctl_out.byte;

        /* loop to make sure the data lines are stable */
        for (i=0;i<10;i++);

        /* Read the MSB */
        hpi_data = (((unsigned short)P4IN) << 8);

        /* Read the LSB */
        hpi_data |= ((unsigned short)(P2IN));

        /* Release the data strobe */
        hpi_ctl_out.bit.hds    = 1;
        P1OUT                    = hpi_ctl_out.byte;

        return (hpi_data);
    }

    /*************************************************************************\
    *  send_dsp_interrupt: sends an interrupt to the VC5509 by asserting the
    *              dsp interrupt bit in the control register.
    \*************************************************************************/
    void send_dsp_interrupt(void)
    {
        hpic.byte =  0;
        hpic.bit.dsp_int = 1;
        write_hpi_control_register(hpic.byte);
        hpic.bit.dsp_int = 0;
        write_hpi_control_register(hpic.byte);
    }

    /*************************************************************************\
    *  send_dsp_command: writes a command code to the VC5509's memory and then
    *              sends an interrupt to the VC5509 to look at the command.
    \*************************************************************************/
```

```
void send_dsp_command(Uint16 command)
{
   write_hpi_address_register(DSP_COMMAND_REGISTER);
   write_hpi_data_word(command,FALSE);
   send_dsp_interrupt();
}

/*****************************************************************************\
*  print_hpi_message: prints messages received from the VC5509 to the display
*              area. The characters are sent in a packed array.
\*****************************************************************************/
void print_hpi_message(void)
{
    Uint16 read_buffer, i, str_len;
    Uchar read_msg[MAX_LINE_LENGTH];

    /* print_hpi_message is always called from the hpi interrupt handler */
    /* so the next memory location in the address register is the length */
    str_len = read_hpi_data_word(TRUE);

    i=0;
    while ((i < str_len) && (i < (MAX_LINE_LENGTH-3)))
    {
       /* get next character */
       read_buffer = read_hpi_data_word(TRUE);

       /* put the MSB in first */
       read_msg[i] = (char)(read_buffer >> 8);
       i++;

       /* then the LSB */
       read_msg[i] = (char) (read_buffer);
       i++;
    }

    /* null terminate the string */
    read_msg[i] = 0;

    add_message_to_display(read_msg);
}

/*****************************************************************************\
*  hpi_handler: processes any commands that are received from the VC5509
\*****************************************************************************/
interrupt [PORT1_VECTOR] void hpi_handler(void)
```

```
{
    Uint16 read_buffer;
    Uint16 refresh=FALSE;

    /* get the command */
    write_hpi_address_register(MSP_COMMAND_REGISTER);
    read_buffer = read_hpi_data_word(TRUE);

    switch(read_buffer)
    {
        case PRINT_MESSAGE:
            print_hpi_message();
            refresh=TRUE;
            break;

        case STOP_PROCESSING:
            stop_signal_processing();
            refresh=TRUE;
            break;

        default: break;
    }

    if (refresh)
    {
        /* simulates typing a dummy character to refresh the screen */
        in_buffer[buffer_write_pos] = ']';
        buffer_write_pos++;

        /* loop back to the beginning if at the end of the buffer */
        if (buffer_write_pos >= MAX_BUFFER) buffer_write_pos = 0;

#if LOW_POWER_MODE==TRUE
            clear_hpi_low_power(LPM1_bits);
#endif
    }

    /* Clear the interrupt flag */
    P1IFG = 0;
}
```

F.8 include.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... include.h
* DATE CREATED.. 04/18/2002
* LAST MODIFIED. 08/06/2002
\*****************************************************************************/

#include <stdio.h>
#include <string.h>
#include "msp430x14x.h"
#include "typedef.h"
#include "prototype.h"
#include "define.h"

#include "extern.h"
```

F.9 init_sys.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... init_sys.c
* DATE CREATED.. 04/18/2002
* LAST MODIFIED. 08/06/2002
\*****************************************************************************/

#include "include.h"

/*****************************************************************************\
*  initialize_system: all the setting to initialize the system
```

```
\*****************************************************************************/
void initialize_system(void)
{
        disable_watch_dog_timer();
        initialize_ports();
        initialize_serial_interface();
        initialize_adc();
        initialize_timers();
        clear_display_area();

        // only temporary for debugging
        dsp_data_address = DSP_HPI_WRITE_1;
        dsp_memory_block = DATA_BLOCK_1;

        /* enable global interrupts */
        _EINT();
}

/*****************************************************************************\
*  initialize_serial_interface: sets up the serial ports for RS232 and
*             RF communications.
\*****************************************************************************/
void initialize_serial_interface(void)
{
    /* select the uarts as special function ports */
    P3SEL |= (BIT4 | BIT5 | BIT6 | BIT7);

    /* set RS232 to 8-bits per character */
    U0CTL = CHAR;

    /* select the SMCLK for the transmit and receive baud rate generator for RS232 */
    U0TCTL = SSEL1 | SSEL0;
    U0RCTL = SSEL1 | SSEL0;

    /* enable the RS232 receiver */
    U0RCTL |= URXEIE;

    /* program the RS232 baud rate to 19.2kbs */
    U0BR1 = 0;
    U0BR0 = 64;
    U0MCTL = 0;

    /* enable the RS232 transmitter and receiver */
    ME1 = UTXE0 | URXE0;
```

```
    /* enable the receive interrupt for RS232 */
    IE1 = URXIE0;

    /* set RF to 8-bits per character */
    U1CTL = CHAR;

    /* select the SMCLK for the transmit and receive baud rate generator for RF */
    U1TCTL = SSEL1 | SSEL0;
    U1RCTL = SSEL1 | SSEL0;

    /* enable the RF receiver */
    U1RCTL |= URXEIE;

    /* program the RF baud rate to 19.2kbs */
    U1BR1 = 0;
    U1BR0 = 64;
    U1MCTL = 0;
}

/*****************************************************************************\
*  initialize_adc: sets up the A/D converter
\*****************************************************************************/
void initialize_adc(void)
{
    /* Reset before changing the settings */
    ADC12CTL0 = 0x0000;
    ADC12CTL1 = 0x0000;

    /* Ref = VeRef+, VeRef-, Input = ADC2 (Geo A) */
    ADC12MCTL0 = (SREF_7 | INCH_2);

    /* Ref = VeRef+, VeRef-, Input = ADC3 (Geo B) */
    ADC12MCTL1 = (SREF_7 | INCH_3);

    /* Ref = VeRef+, VeRef-, Input = ADC4 (Geo C) */
    ADC12MCTL2 = (EOS | SREF_7 | INCH_4);

    /* Ref = VeRef+, VeRef-, Input = ADC5 (Test 1) */
    ADC12MCTL3 = (SREF_7 | INCH_5);

    /* Ref = VeRef+, VeRef-, Input = ADC6 (Test 2) */
    ADC12MCTL4 = (SREF_7 | INCH_6);

    /* Ref = VeRef+, VeRef-, Input = ADC7 (Test 3) */
    ADC12MCTL5 = (EOS | SREF_7 | INCH_7);
```

```
110     /* Ref = VREF+, AVss, Input = ADC10 (Temperature) */
        ADC12MCTL6 = (EOS | SREF_1 | INCH_10);

        /* Ref = VREF+, AVss, Input = ADC11 (Power Supply Volatge) */
        ADC12MCTL7 = (EOS | SREF_1 | INCH_11);

        /* used to buffer the conversions for signal processing */
        /* Ref = VeRef+, VeRef-, Input = ADC2 (Geo A)            */
        ADC12MCTL8  = (SREF_7 | INCH_2);
        ADC12MCTL9  = (SREF_7 | INCH_2);
120     ADC12MCTL10 = (SREF_7 | INCH_2);
        ADC12MCTL11 = (SREF_7 | INCH_2);
        ADC12MCTL12 = (SREF_7 | INCH_2);
        ADC12MCTL13 = (SREF_7 | INCH_2);
        ADC12MCTL14 = (SREF_7 | INCH_2);
        ADC12MCTL15 = (EOS | SREF_7 | INCH_2);
    }

    /*****************************************************************************\
    *  initialize_ports: sets up the I/O ports
130 \*****************************************************************************/
    void initialize_ports(void)
    {
        /* ---- Port 1 Settings ---- */
        /* use as a port */
        P1SEL = 0x00;

        /* HINT and HRDY are inputs, the rest are outputs 0x7E */
        P1DIR = (BIT6 | BIT5 | BIT4 | BIT3 | BIT2 | BIT1);

140     /* force all outputs to 0 */
        P1OUT = 0x00;

        /* disable all interrupts */
        P1IE  = 0x00;

        /* when interrupts are enabled, interrupt on a low to high transition */
        P1IES = 0x00;

        /* ---- Port 2 Settings ---- */
150     /* use as a port */
        P2SEL = 0x00;          //

        /* all bits are inputs */
```

```
        P2DIR = 0x00;          //

        /* force all output bits to 0 */
        P2OUT = 0x00;

        /* disable all interrupts */
160     P2IE  = 0x00;

        /* When interrupts are enabled, interrupt on a low to high transition */
        P2IES = 0x00;

        /* ---- Port 3 Settings ---- */
        /* use bits 0 to 3 as a port, use bits 4 to 7 as UARTs */
        P3SEL = (BIT7 | BIT6 | BIT5 | BIT4);

        /* CTS is an input, bits 0 to 2 are outputs */
170     P3DIR = (BIT2 | BIT1 | BIT0);

        /* force VC55_ENA and GEO_ENA to 1 and all other output bits to 0 */
        P3OUT = (BIT1 | BIT0);

        /* ---- Port 4 Settings ---- */
        /* Use as a port */
        P4SEL = 0x00;

        /* all bits are inputs */
180     P4DIR = 0x00;

        /* force all output bits to 0 */
        P4OUT = 0x00;

        /* ---- Port 5 Settings ---- */
        /* enable ACLK, and MCLK */
        P5SEL = (BIT6 | BIT4);

        /* all bits are outputs */
190     P5DIR = (BIT7 | BIT6 | BIT5 | BIT4 | BIT3 | BIT2 | BIT1 | BIT0);

        /* force RS232 enable high, all other outputs 0 */
        P5OUT = BIT5;

        /* ---- Port 6 Settings ---- */
        /* use as special function for ADC */
        P6SEL   = (BIT7 | BIT6 | BIT5 | BIT4 | BIT3 | BIT2 | BIT1 | BIT0);
```

```
    /* all inputs */
    P6DIR   = 0x00;
}

/*****************************************************************************\
*  initialize_timers: initializes the clock crystals and sets up the timers
\*****************************************************************************/
void initialize_timers(void)
{
    Uint16 i,j;

    /* select 1.3 MHz for the DCOC just in case */
    BCSCTL1 |= RSEL2 | RSEL0;

    /* clear the XTS bit */
    BCSCTL1 &= (~XTS);

    /* clear the OSCOFF flag in the Status Register */
    _BIC_SR(OSCOFF);

    /* turn on XT2 oscillator */
    BCSCTL1 &= (~XTOFF);

    /* test to see if it was able lock onto X2 */
    for (i=15; i > 0; i--)
    {
        IFG1 &= (~OFIFG);
        for (j=0;j<1000;j++);
        if (IFG1 && OFIFG == 0) break;
    }

    /* set MCLK and SMCLK to XT2 with a divide by 4 */
    BCSCTL2 |= SELM1 | DIVM1 | SELS | DIVS1;

    /* setup timer A to keep the running time */
    /* reset the timer */
    TACTL = TACLR;

    /* start in up mode with ACLK as source */
    TACTL = TASSEL_1 | MC_1 | ID_0;

    /* interrupt produces a toggle of OUT */
    CCTL0 = OUTMOD_4 | CCIE;

    /* set CCR0 register to 32768 ticks for one second */
```

```
        CCR0  = 0x8000;

        /* shut off CC1 and CC2 */
        CCTL1 = 0;
        CCTL2 = 0;
        CCR1  = 0x0000;
        CCR2  = 0x0000;

        /* setup timer B to keep the sampling rate for ADC */
        /* reset the timer */
        TBCTL = TBCLR;

        /* pulse OUT when it reaches the register */
        TBCCTL0 = OUTMOD_3;

        /* sets the sampling rate of the ADC */
        TBCCR0  = SAMPLING_RATE;

        /* shut off all the other CCTL registers */
        TBCCTL1 = 0;
        TBCCTL2 = 0;
        TBCCTL3 = 0;
        TBCCTL4 = 0;
        TBCCTL5 = 0;
        TBCCTL6 = 0;
        TBCCR1  = 0x0000;
        TBCCR2  = 0x0000;
        TBCCR3  = 0x0000;
        TBCCR4  = 0x0000;
        TBCCR5  = 0x0000;
        TBCCR6  = 0x0000;
    }

    /*****************************************************************************\
    *  disable_watch_dog_timer: disables the watch dog timer
    \*****************************************************************************/
    void disable_watch_dog_timer(void)
    {
            WDTCTL = WDTPW | WDTHOLD;

    }
```

F.10 low_power.s43

```
;
;  Personnel Detector
;  Elliot Ranger
;  Master of Science Thesis
;  Massachusetts Institute of Technology
;  in conjunction with Charles Stark Draper Laboratory
;
;
;*****************************************************************************\
; FILENAME..... low_power.s43
; DATE CREATED.. 04/18/2002
; LAST MODIFIED. 08/06/2002
;*****************************************************************************/

          NAME     low_power(16)
          RSEG     CODE(1)
          PUBLIC   clear_low_power_mode
          PUBLIC   clear_hpi_low_power
          EXTERN   ?CL430_1_23_L08
          RSEG     CODE
clear_low_power_mode:
          BIC.W    R12,11(SP)
          RET
clear_hpi_low_power:
          BIC.W    R12,12(SP)
          RET
          END
```

F.11 main.c

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... main.c
* DATE CREATED.. 04/18/2002
```

```
* LAST MODIFIED. 08/06/2002
\*****************************************************************************/

#include "include.h"
#include "variables.h"

/*****************************************************************************\
*  main: initializes the system and then processes commands indefinitely.
\*****************************************************************************/
void main(void)
{
     initialize_system();

     /* infinite loop */
     while (TRUE)
     {
          process_commands();
     }
}
```

F.12 menu.c

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... menu.c
* DATE CREATED.. 04/18/2002
* LAST MODIFIED. 08/06/2002
\*****************************************************************************/

#include "include.h"

/*****************************************************************************\
*  cls: ansi ESC character sequence to clear the screen.
\*****************************************************************************/
void cls(void)
{
```

```
    unsigned char buffer[10];
    sprintf(buffer,"%c[2J",ESC);
    print_message(buffer);
}

/*****************************************************************************\
*  display_header: prints out the header at the top of each menu then calls
*              print_display_area to print out the messages.
\*****************************************************************************/
void display_header(void)
{
    cls(); //    clear the screen
    print_message("Personnel Detector\r\n");
    print_message("Master's Thesis Project\r\n");
    print_message("Elliot Ranger\r\n");
    print_message(VERSION);
    print_message("\r\n\r\n");

    // call function to display messages
    print_display_area();
    print_message("\r\n\r\n");
}

/*****************************************************************************\
*  display_footer: prints out the footer at the bottom of each menu screen.
\*****************************************************************************/
void display_footer(void)
{
    print_message("\r\nSelect: ");
}

/*****************************************************************************\
*  display_main_menu: prints the main menu screen.
\*****************************************************************************/
void display_main_menu(void)
{
    display_header();
    print_message("Main Menu\r\n");
    print_message("\r\n");
    print_message("a. VC5509 Menu\r\n");
    print_message("b. Geophone Menu\r\n");
    print_message("c. RF Menu\r\n");
    print_message("d. RS232 Menu\r\n");
    print_message("e. ADC Test\r\n");
    print_message("f. Host Port Interface\r\n");
```

```
    print_message("g. Flash Menu\r\n");
    print_message("h. Misc Menu\r\n");
    display_footer();
}

/*****************************************************************************\
*  display_vc55_menu: prints the VC5509 menu screen.
\*****************************************************************************/
void display_vc55_menu(void)
{
    display_header();
    print_message("VC5509 Menu\r\n");
    print_message("\r\n");
    print_message("a. Power ON\r\n");
    print_message("b. Power OFF\r\n");
    print_message("c. Boot DSP\r\n");
    print_message("d. Copy DSP Code to Buffer\r\n");
    print_message("e. Copy Buffer to Flash\r\n");
    print_message("f. Start Signal Processing\r\n");
    print_message("g. Stop Signal Processing\r\n");
    print_message("h. Clear Signal Processing Buffers\r\n");
    print_message("i. VC5509 Code Version\r\n");
    display_footer();
}

/*****************************************************************************\
*  display_geophone_menu: prints the geophone menu screen.
\*****************************************************************************/
void display_geophone_menu(void)
{
    display_header();
    print_message("Geophone Menu\r\n");
    print_message("\r\n");
    print_message("a. Power ON\r\n");
    print_message("b. Power OFF\r\n");
    print_message("c. Channel A Gain 40dB\r\n");
    print_message("d. Channel A Gain 60dB\r\n");
    print_message("e. Channel B Gain 40dB\r\n");
    print_message("f. Channel B Gain 60dB\r\n");
    print_message("g. Channel C Gain 40dB\r\n");
    print_message("h. Channel C Gain 60dB\r\n");
    display_footer();
}

/*****************************************************************************\
```

```
*  display_rf_menu: prints the RF menu screen.
\*****************************************************************************/
void display_rf_menu(void)
{
     display_header();
     print_message("RF Menu\r\n");
     print_message("\r\n");
     print_message("a. Power ON\r\n");
     print_message("b. Power OFF\r\n");
     print_message("a. Shutdown ON\r\n");
     print_message("b. Shutdown OFF\r\n");
     display_footer();
}

/*****************************************************************************\
*  display_rs232_menu: prints the RS232 menu screen.
\*****************************************************************************/
void display_rs232_menu(void)
{
     display_header();
     print_message("RS232 Menu\r\n");
     print_message("\r\n");
     print_message("a. Shutdown ON\r\n");
     print_message("b. Shutdown OFF\r\n");
     display_footer();
}

/*****************************************************************************\
*  display_dac_menu: prints the A/D converter menu screen.
\*****************************************************************************/
void display_adc_menu(void)
{
     display_header();
     print_message("ADC Menu\r\n");
     print_message("\r\n");
     print_message("a. Geophone Channels\r\n");
     print_message("b. Test Input Channels\r\n");
     print_message("c. Temperature Sensor\r\n");
     print_message("d. Power Supply Voltage\r\n");
     display_footer();
}

/*****************************************************************************\
*  display_hpi_menu: prints the host port interface menu screen.
\*****************************************************************************/
```

```
void display_hpi_menu(void)
{
    display_header();
    print_message("Host Port Interface\r\n");
    print_message("\r\n");
    print_message("a. Enable HPI\r\n");
    print_message("b. Disable HPI\r\n");
    print_message("c. Write 0x1234 (4660 Decimal) to 0x0A00\r\n");
    print_message("d. Read Memory Location 0x0A00\r\n");
    print_message("e. Send HPI Interrupt\r\n");
    display_footer();
}

/*****************************************************************************\
*  display_flash_menu: prints the flash menu screen.
\*****************************************************************************/
void display_flash_menu(void)
{
    display_header();
    print_message("Flash Menu\r\n");
    print_message("\r\n");
    print_message("a. Write to address 0x6000\r\n");
    print_message("b. Erase address 0x6000\r\n");
    print_message("c. Read address 0x6000\r\n");
    display_footer();
}

/*****************************************************************************\
*  display_misc_menu: prints the miscellaneous menu screen.
\*****************************************************************************/
void display_misc_menu(void)
{
    display_header();
    print_message("Misc Menu\r\n");
    print_message("\r\n");
    print_message("a. Display Running Time\r\n");
    print_message("b. MSP Power Supply Voltage\r\n");
    print_message("c. MSP Temperature\r\n");
    print_message("d. Clear Message Display Area\r\n");
    display_footer();

}
```

F.13 proc_adc.c

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... proc_adc.c
* DATE CREATED.. 04/18/2002
* LAST MODIFIED. 08/06/2002
\*****************************************************************************/

#include "include.h"

/*****************************************************************************\
*  process_adc: gets a character from the process_commands function and allows
*               the user to keep sampling one of the A/D converter channels
*               until ESC is pressed.
\*****************************************************************************/
void process_adc(Char channel)
{
     Char buffer[MAX_LINE_LENGTH];
     Uint16 start_timer;

     switch(channel)
     {
          case 'a'   :     /* Geophone Channels */
          {
               /* make sure control register is cleared out */
               ADC12CTL1 = 0x0000;

               /* turn on ADC core */
               ADC12CTL0 |= ADC12ON;

               /* perform the conversion on a single channel at address 0 - Geo A */
               ADC12CTL1 = (CONSEQ_0 | ADC12SSEL_2 | SHP | CSTARTADD_0);

               /* single NOP to make sure ADC core is on */
               ;

               /* start sampling */
```

```
ADC12CTL0 |= ( ENC | ADC12SC);

/* wait for conversion to complete */
while ((ADC12CTL1 & ADC12BUSY) == ADC12BUSY);

/* turn off enable conversion */
ADC12CTL0 &= (~ENC);  //

/* perform the conversion on a single channel at address 1 - Geo B */
ADC12CTL1 = (CONSEQ_0 | ADC12SSEL_2 | SHP | CSTARTADD_1);

/* single NOP to make sure ADC core is on */
;

/* start sampling */
ADC12CTL0 |= ( ENC | ADC12SC);

/* wait for conversion to complete */
while ((ADC12CTL1 & ADC12BUSY) == ADC12BUSY);

/* turn off enable conversion */
ADC12CTL0 &= (~ENC);

/* perform the conversion on a single channel at address 2 - Geo C */
ADC12CTL1 = (CONSEQ_0 | ADC12SSEL_2 | SHP | CSTARTADD_2);

/* single NOP to make sure ADC core is on */
;

/* start sampling */
ADC12CTL0 |= ( ENC | ADC12SC);

/* wait for conversion to complete */
while ((ADC12CTL1 & ADC12BUSY) == ADC12BUSY);

/* turn off enable conversion */
ADC12CTL0 &= (~ENC);

/* turn off adc */
ADC12CTL0 &= (~ADC12ON);

/* print results to display area */
sprintf(buffer,"Geophone A: %6.4f",((double)ADC12MEM[0]/4096.0)*3.00);
add_message_to_display(buffer);
sprintf(buffer,"Geophone B: %6.4f",((double)ADC12MEM[1]/4096.0)*3.00);
```

```
            add_message_to_display(buffer);
            sprintf(buffer,"Geophone C: %6.4f",((double)ADC12MEM[2]/4096.0)*3.00);
            add_message_to_display(buffer);
        }
        break;

        case 'b'  :     /* Test Input */
        {
            /* make sure control register is cleared out */
            ADC12CTL1 = 0x0000;

            /* turn on ADC core */
            ADC12CTL0 |= ADC12ON;

            /* perform the conversion on a single channel at address 3 - Test 1 */
            ADC12CTL1 = (CONSEQ_0 | ADC12SSEL_2 | SHP | CSTARTADD_3);

            /* single NOP to make sure ADC core is on */
            ;

            /* start sampling */
            ADC12CTL0 |= (ENC | ADC12SC);

            /* wait for conversion to complete */
            while ((ADC12CTL1 & ADC12BUSY) == ADC12BUSY);

            /* turn off enable conversion */
            ADC12CTL0 &= (~ENC);

            /* perform the conversion on a single channel at address 4 - Test 2 */
            ADC12CTL1 = (CONSEQ_0 | ADC12SSEL_2 | SHP | CSTARTADD_4);

            /* single NOP to make sure ADC core is on */
            ;

            /* start sampling */
            ADC12CTL0 |= (ENC | ADC12SC);

            /* wait for conversion to complete */
            while ((ADC12CTL1 & ADC12BUSY) == ADC12BUSY);

            /* turn off enable conversion */
            ADC12CTL0 &= (~ENC);

            /* perform the conversion on a single channel at address 5 - Test 3 */
```

```
            ADC12CTL1 = (CONSEQ_0 | ADC12SSEL_2 | SHP | CSTARTADD_5);

            /* single NOP to make sure ADC core is on */
            ;

            /* start sampling */
            ADC12CTL0 |= (ENC | ADC12SC);

            /* wait for conversion to complete */
            while ((ADC12CTL1 & ADC12BUSY) == ADC12BUSY);

            /* turn off enable conversion */
            ADC12CTL0 &= (~ENC);

            /* turn off adc */
            ADC12CTL0 &= (~ADC12ON);

            /* print results to display area */
            sprintf(buffer,"Test Input 1: %6.4f",((double)ADC12MEM[3]/4096.0)*3.00);
            add_message_to_display(buffer);
            sprintf(buffer,"Test Input 2: %6.4f",((double)ADC12MEM[4]/4096.0)*3.00);
            add_message_to_display(buffer);
            sprintf(buffer,"Test Input 3: %6.4f",((double)ADC12MEM[5]/4096.0)*3.00);
            add_message_to_display(buffer);
        }
        break;

        case 'c'  :    /* Temperature Sensor */
        {
            /* make sure control register is cleared out */
            ADC12CTL1 = 0x0000;

            /* perform the conversion on a single channel at address 6 - Temperature Sensor */
            ADC12CTL1 = (CONSEQ_0 | ADC12SSEL_2 | ADC12DIV_7 | SHP | CSTARTADD_6);

            /* turn on core and reference voltage to 2.5 V */
            ADC12CTL0 |= (ADC12ON | REFON | REF2_5V);

            /* make sure reference voltages are stable */
            start_timer = TAR;
            while (((TAR - start_timer) & TIMER_MASK) < 15);

            /* start sampling */
            ADC12CTL0 |= (ENC | ADC12SC);
```

```
            /* wait for conversion to complete */
            while ((ADC12CTL1 & ADC12BUSY) == ADC12BUSY);

            /* turn off enable conversion */
            ADC12CTL0 &= (~ENC);

            /* turn off reference voltages */
            ADC12CTL0 &= (~(ADC12ON | REFON | REF2_5V));

            /* print results to display area */
            sprintf(buffer,"Temperature Sensor: %6.2f C",((((double) ADC12MEM[6])*2.50/4096.0)/3.55e-3) - 273.15);
            add_message_to_display(buffer);
        }
        break;

        case 'd'  :     /* Power Supply Voltage */
        {
            /* make sure control register is cleared out */
            ADC12CTL1 = 0x0000;

            /* perform the conversion on a single channel at address 7 - Power Supply Voltage */
            ADC12CTL1 = (CONSEQ_0 | ADC12SSEL_2 | ADC12DIV_7 | SHP | CSTARTADD_7);  // Using master clock, single channel

            /* turn on core and reference voltage to 2.5 V */
            ADC12CTL0 |= (ADC12ON | REFON | REF2_5V);

            /* make sure reference voltages are stable */
            start_timer = TAR;
            while (((TAR - start_timer) & TIMER_MASK) < 15);

            /* start sampling */
            ADC12CTL0 |= (ENC | ADC12SC);

            /* wait for conversion to complete */
            while ((ADC12CTL1 & ADC12BUSY) == ADC12BUSY);

            /* turn off enable conversion */
            ADC12CTL0 &= (~ENC);

            /* turn off reference voltages */
            ADC12CTL0 &= (~(ADC12ON | REFON | REF2_5V)); // turn off reference voltages

            /* print results to display area */
            sprintf(buffer,"Power Supply Voltage: %6.4f",((((double) ADC12MEM[7])*2.50/4096.0)/0.5));
            add_message_to_display(buffer);
```

```
            }
            break;

            default : break;
        }

230 }
```

F.14 proc_cmd.c

```
    /*
     *  Personnel Detector
     *  Elliot Ranger
     *  Master of Science Thesis
     *  Massachusetts Institute of Technology
     *  in conjunction with Charles Stark Draper Laboratory
     *
     */
    /*****************************************************************************\
 10 * FILENAME..... proc_cmd.c
    * DATE CREATED.. 04/18/2002
    * LAST MODIFIED. 08/06/2002
    \*****************************************************************************/

    #include "include.h"

    /*****************************************************************************\
    *  process_commands: main function that continually processes any key typed
    *              and executes the associated function.
 20 \*****************************************************************************/
    void process_commands(void)
    {
        Uint16 done = FALSE;
        Uchar key;
        Uint16 read_data;
        Uchar buffer[MAX_LINE_LENGTH];

        display_main_menu();
        switch(get_char())
 30     {
            case 'a' :    /* VC5509 Menu */
            {
                do
```

```
{
    done = FALSE;
    display_vc55_menu();
    switch (get_char())
    {
        case 'a' :    /* turn on VC5509 and enable hpi */
            msp_port3.byte = P3OUT;
            msp_port3.bit.vc55_enable = 0;
            P3OUT = msp_port3.byte;
            enable_hpi();
            done = TRUE;
            break;

        case 'b' :    /* disable hpi and turn off VC5509 */
            disable_hpi();
            msp_port3.byte = P3OUT;
            msp_port3.bit.vc55_enable = 1;
            P3OUT = msp_port3.byte;
            done = TRUE;
            break;

        case 'c' :    /* downloads the code from flash and boots VC5509 */
            boot_dsp();
            done = TRUE;
            break;

        case 'd' :    /* tells VC5509 to copy its program code to HPI Write */
            send_dsp_command(COPY_TO_HPI_WRITE);
            done = TRUE;
            break;

        case 'e' :    /* copy VC5509 code to flash and verify that the memory is the same */
            cls();
            print_message("\r\n\r\nCopying VC5509 Code to Flash, Please Wait...\r\n");
            copy_dsp_code_to_flash();
            verify_dsp_code();
            done = TRUE;
            break;

        case 'f' :    /* begins signal processing */
            add_message_to_display("MSP: Starting Signal Processing");

            /* turn on geophone power */
            msp_port3.byte = P3OUT;
            msp_port3.bit.geo_enable = 0;
```

```
                    P3OUT = msp_port3.byte;
                    start_signal_processing();
                    done = TRUE;
                    break;

                case 'g' :    /* stops signal processing */
                    add_message_to_display("MSP: Stopping Signal Processing");
                    stop_signal_processing();
                    done = TRUE;
                    break;

                case 'h' :    /* clear signal processing buffers*/
                    send_dsp_command(CLEAR_PROCESSING_BUFFERS);
                    done = TRUE;
                    break;

                case 'i' :    /* tells VC5509 to print out its code version */
                    send_dsp_command(GET_VC5509_VERSION);
                    done = TRUE;
                    break;

                default  :   break;
            }
        } while (!done);
    }
    break;

    case 'b' :    /* Geophone Menu */
    {
        do
        {
            done = FALSE;
            display_geophone_menu();
            switch (get_char())
            {
                case 'a' :    /* turn on geophone power */
                    msp_port3.byte = P3OUT;
                    msp_port3.bit.geo_enable = 0;
                    P3OUT = msp_port3.byte;
                    done = TRUE;
                    break;

                case 'b' :    /* turn off geophone power */
                    msp_port3.byte = P3OUT;
                    msp_port3.bit.geo_enable = 1;
```

```
            P3OUT = msp_port3.byte;
            done = TRUE;
            break;

        case 'c' :    /* set Geo A to 40 dB */
            msp_port5.byte = P5OUT;
            msp_port5.bit.geo_A_sel = 0;
            P5OUT = msp_port5.byte;
            done = TRUE;
            break;

        case 'd' :    /* set Geo A to 60 dB */
            msp_port5.byte = P5OUT;
            msp_port5.bit.geo_A_sel = 1;
            P5OUT = msp_port5.byte;
            done = TRUE;
            break;

        case 'e' :    /* set Geo B to 40 dB */
            msp_port5.byte = P5OUT;
            msp_port5.bit.geo_B_sel = 0;
            P5OUT = msp_port5.byte;
            done = TRUE;
            break;

        case 'f' :    /* set Geo B to 60 dB */
            msp_port5.byte = P5OUT;
            msp_port5.bit.geo_B_sel = 1;
            P5OUT = msp_port5.byte;
            done = TRUE;
            break;

        case 'g' :    /* set Geo C to 40 dB */
            msp_port5.byte = P5OUT;
            msp_port5.bit.geo_C_sel = 0;
            P5OUT = msp_port5.byte;
            done = TRUE;
            break;

        case 'h' :    /* set Geo C to 60 dB */
            msp_port5.byte = P5OUT;
            msp_port5.bit.geo_C_sel = 1;
            P5OUT = msp_port5.byte;
            done = TRUE;
            break;
```

```
                default   :   break;
            }
        } while (!done);
    }
    break;

    case 'c' :     /* RF Menu */
    {
        do
        {
            done = FALSE;
            display_rf_menu();
            switch (get_char())
            {
                case 'a' :     /* turn on RF power and enable interrupts */
                    msp_port3.byte = P3OUT;
                    msp_port3.bit.rf_enable = 1;
                    P3OUT = msp_port3.byte;

                    /* enable the RF transmitter and receiver */
                    ME2 = UTXE1 | URXE1;

                    /* enable RF receive interrupt */
                    IE2 = URXIE1;

                    rf_ena = TRUE;
                    done = TRUE;
                    break;

                case 'b' :     /* turn off RF power and disable interrupts */
                    msp_port3.byte = P3OUT;
                    msp_port3.bit.rf_enable = 0;
                    P3OUT = msp_port3.byte;

                    /* disable the RF transmitter and receiver */
                    ME2 = 0;              // disable the transmitter & recevier

                    /* disable RF receive interrupt */
                    IE2 = 0;

                    rf_ena = FALSE;
                    done = TRUE;
                    break;
```

```
                    case 'c' :    /* put RF in sleep mode */
                        msp_port5.byte = P5OUT;
                        msp_port5.bit.rf_shutdown = 1;
                        P5OUT = msp_port5.byte;
                        done = TRUE;
                        break;

                    case 'd' :    /* take RF out of sleep mode */
                        msp_port5.byte = P5OUT;
                        msp_port5.bit.rf_shutdown = 0;
                        P5OUT = msp_port5.byte;
                        done = TRUE;
                        break;

                    default  :   break;
                }
            } while (!done);
        }
        break;

        case 'd' :    /* RS232 Menu */
        {
            do
            {
                done = FALSE;
                display_rs232_menu();
                switch (get_char())
                {
                    case 'a' :    /* shutdown RS232 transmitter */
                        msp_port5.byte = P5OUT;
                        msp_port5.bit.rs232_enable = 0;
                        P5OUT = msp_port5.byte;
                        done = TRUE;
                        break;

                    case 'b' :    /* turn on RS232 transmitter */
                        msp_port5.byte = P5OUT;
                        msp_port5.bit.rs232_enable = 1;
                        P5OUT = msp_port5.byte;
                        done = TRUE;
                        break;

                    default  :   break;
                }
            } while (!done);
```

```
        }
260     break;


        case 'e' :    /* ADC Menu */
        {
            do
            {
                done = FALSE;
                display_adc_menu();
                switch (key=get_char())
270             {

                    case ESC :    /* wait for ESC or CR to exit menu */
                    case CR  :
                        done = TRUE;
                        break;

                    default  :
                        cls();
                        process_adc(key);
280                     break;
                }
            } while (!done);
        }
        break;

        case 'f' :    /* HPI Menu */
        {
            do
            {
290             done = FALSE;
                display_hpi_menu();
                switch (get_char())
                {
                    case 'a' :    /* enable host port interface */
                        enable_hpi();
                        done = TRUE;
                        break;

                    case 'b' :    /* disable host port interface */
300                     disable_hpi();
                        done = TRUE;
                        break;
```

```
                    case 'c' :    /* write a test word to HPI Write location */
                        write_hpi_address_register(0x0A00);
                        write_hpi_data_word(0x1234,TRUE);
                        done = TRUE;
                        break;

310                 case 'd' :    /* read first word at HPI Write location */
                        write_hpi_address_register(0x0A00);
                        read_data = read_hpi_data_word(TRUE);
                        sprintf(buffer,"VC5509 Memory Location 0x0A00: %X",read_data);
                        add_message_to_display(buffer);
                        done = TRUE;
                        break;

                    case 'e' :    /* send VC5509 host port interrupt */
                        send_dsp_interrupt();
320                     done = TRUE;
                        break;

                    default  :  break;
                }
            } while (!done);
        }
        break;

        case 'g' :    /* Flash Menu */
330     {
            do
            {
                done = FALSE;
                display_flash_menu();
                switch (get_char())
                {
                    case 'a' :    /* write test word to flash location 0x6000 */
                        write_flash(0x6000,0x1234);
                        done = TRUE;
340                     break;

                    case 'b' :    /* erase flash memory location 0x6000 */
                        erase_flash(0x6000);
                        done = TRUE;
                        break;

                    case 'c' :    /* read the memory at location 0x6000 */
                        read_data = read_flash(0x6000);
```

```
                        sprintf(buffer,"Flash Memory Location 0x6000: %X",read_data);
350                     add_message_to_display(buffer);
                        done = TRUE;
                        break;

                    default   :  break;
                }
            } while (!done);
        }
        break;

360     case 'h' :     /* Misc Menu */
        {
            do
            {
                done = FALSE;
                display_misc_menu();
                switch (get_char())
                {
                    case 'a' :     /* print out running time of processor */
                        sprintf(buffer,"\r\n%d days",days);
370                     add_message_to_display(buffer);
                        sprintf(buffer,"%d hours : %d minutes : %d seconds",hours, minutes, seconds);
                        add_message_to_display(buffer);
                        done = TRUE;
                        break;

                    case 'b' :     /* print out MSP power supply voltage */
                        process_adc('d');
                        done = TRUE;
                        break;
380
                    case 'c' :     /* print out MSP temperature */
                        process_adc('c');
                        done = TRUE;
                        break;

                    case 'd' :     /* reset display message area */
                        clear_display_area();
                        done = TRUE;
                        break;
390
                    default   :  break;
                }
            } while (!done);
```

```
            }
            break;

            default : break;
        }
    }
```

F.15 prototype.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... prototype.h
* DATE CREATED.. 04/18/2002
* LAST MODIFIED. 08/06/2002
\*****************************************************************************/

void clear_low_power_mode(Uint16 mode);
void clear_hpi_low_power(Uint16 mode);

void initialize_system(void);
void initialize_serial_interface(void);
void initialize_adc(void);
void initialize_ports(void);
void initialize_timers(void);
void disable_watch_dog_timer(void);

void print_message(Uchar *message);
Uchar get_char(void);

void process_commands(void);
void process_adc(Uchar channel);

void cls(void);
void display_header(void);
void display_footer(void);
void display_main_menu(void);
```

```
void display_vc55_menu(void);
void display_geophone_menu(void);
void display_rf_menu(void);
void display_rs232_menu(void);
void display_adc_menu(void);
void display_hpi_menu(void);
void display_flash_menu(void);
void display_misc_menu(void);

void enable_hpi(void);
void disable_hpi(void);
void write_hpi_control_register(Uint16 data);
void write_hpi_address_register(Uint16 data);
void write_hpi_data_word(Uint16, Uint16 auto_incr);
Uint16 read_hpi_data_word(Uint16 auto_incr);
void send_dsp_interrupt(void);
void send_dsp_command(Uint16 command);
void print_hpi_message(void);

void add_message_to_display(Uchar *message);
void clear_display_area(void);
void print_display_area(void);

void boot_dsp(void);
void copy_dsp_code_to_flash(void);
void verify_dsp_code(void);

void erase_flash(Uint16 address);
void write_flash(Uint16 address, Uint16 data);
Uint16 read_flash(Uint16 address);

void start_signal_processing(void);

void stop_signal_processing(void);
```

F.16 signal_proc.c

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
```

```
 */
/*****************************************************************************\
* FILENAME..... signal_proc.c
* DATE CREATED.. 08/01/2002
* LAST MODIFIED. 08/06/2002
\*****************************************************************************/

#include "include.h"

/*****************************************************************************\
*  start_signal_processing: turns on the ADC to start collecting data, sets
*              timer B as the control for the sampling rate, and enables timer B
\*****************************************************************************/
void start_signal_processing(void)
{

    /* make sure control register is cleared out */
    ADC12CTL1 = 0x0000;

    /* clear any interrupt flags and enable interrupts on channel 15 */
    ADC12IFG = 0x0000;
    ADC12IE = BITF;

    /* repeated sequence starting at address 8 using timer B */
    /* to control the sampling rate                          */
    ADC12CTL1 = (CONSEQ_3 | SHS_2 | ADC12SSEL_3 | SHP  | CSTARTADD_8);

    /* turn on ADC core */
    ADC12CTL0 |= (ADC12ON);

    /* single NOP to make sure ADC core is on */
    ;

    /* enable conversions */
    ADC12CTL0 |= ENC;

    /* start Timer B using ACLK as the source in up mode */
    TBCTL = TBSSEL_1 | MC_1 | ID_0;
}

/*****************************************************************************\
*  stop_signal_processing: resets timer B, disables all ADC interrupts and
*              turns off the ADC core.
\*****************************************************************************/
void stop_signal_processing(void)
```

```
{
    /* reset timer B */
    TBCTL = TBCLR;

    /* disable ADC12 interrupts and clear any pending interrupts */
    ADC12IE = 0x0000;
    ADC12IFG = 0x0000;

    /* turn off enable conversion */
    ADC12CTL0 &= (~ENC);

    /* turn off ADC */
    ADC12CTL0 &= (~ADC12ON);
}

/*****************************************************************************\
*  adc_handler: receives an interrupt when the ADC has done 8 conversions
*              transfers the data to the VC5509 using dsp_data_address to
*              keep tract of where the next set of conversions should go.
\*****************************************************************************/
interrupt [ADC_VECTOR] void adc_handler(void)
{
    Uint16 i;
    Uint16 scaled;

    /* transfer the data to the VC5509 */
    if ((ADC12IFG & BITF) == BITF)
    {
        /* set the correct address location to start writing data */
        write_hpi_address_register(dsp_data_address);
        for (i=8;i<=15;i++)
        {
            scaled = ADC12MEM[i] << 3;
            write_hpi_data_word(scaled,TRUE);
            dsp_data_address++;
        }

        if ((dsp_data_address >= 0x14AF) && (dsp_memory_block == DATA_BLOCK_1))
        {
            send_dsp_command(PROCESS_DATA_BLOCK_1);
            dsp_data_address = DSP_HPI_WRITE_2;
            dsp_memory_block = DATA_BLOCK_2;
        }

        if ((dsp_data_address >= 0x19AF) && (dsp_memory_block == DATA_BLOCK_2))
```

```
            {
                send_dsp_command(PROCESS_DATA_BLOCK_2);
100             dsp_data_address = DSP_HPI_WRITE_1;
                dsp_memory_block = DATA_BLOCK_1;
            }
        }

    }
```

F.17 timer.c

```
    /*
     *  Personnel Detector
     *  Elliot Ranger
     *  Master of Science Thesis
     *  Massachusetts Institute of Technology
     *  in conjunction with Charles Stark Draper Laboratory
     *
     */
    /*************************************************************************\
10  * FILENAME..... timer.c
    * DATE CREATED.. 04/18/2002
    * LAST MODIFIED. 08/06/2002
    \*************************************************************************/

    #include "include.h"

    /*************************************************************************\
    *  timer_A_handler: keeps tract of the running time of the processor
    \*************************************************************************/
20  interrupt [TIMERA0_VECTOR] void timer_A_handler(void)
    {
        time++;

        /* increment the number of seconds */
        seconds++;
        seconds%=60;
        if (seconds == 0)
        {
            /* increment the number of minutes */
30          minutes++;
            minutes%=60;
            if (minutes == 0)
            {
```

```
                /* increment the number of hours */
                hours++;
                hours%=24;
                if (hours == 0)
                {
                    /* increment the number of days */
                    days++;
                }
            }
        }
    }
```

F.18 typedef.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... typedef.h
* DATE CREATED.. 04/18/2002
* LAST MODIFIED. 08/06/2002
\*****************************************************************************/

typedef unsigned char      Uchar;
typedef char               Char;
typedef unsigned short  Uint16;
typedef unsigned long   Uint32;

typedef union
{
    struct
    {
        short vc55_enable : 1;
        short geo_enable  : 1;
        short rf_enable   : 1;
        short spare        : 5;
    } bit;
    Uchar byte;
```

```
} MSP_PORT3_TYPE;

typedef union
{
    struct
    {
        short geo_A_sel    : 1;
        short geo_B_sel    : 1;
        short geo_C_sel    : 1;
        short spare1       : 2;
        short rs232_enable : 1;
        short spare2       : 1;
        short rf_shutdown  : 1;
    } bit;
    Uchar byte;
}
MSP_PORT5_TYPE;

typedef  union
{
    struct
    {
        short spare1 :    3;
        short rf_cts :    1;
        short spare2 :    4;
    } bit;
    Uchar byte;
} RF_CONTROL_TYPE;

typedef  union
{
    struct
    {
        short hpi_int :   1;
        short byte_en :   2;
        short control :   2;
        short rd_wr   :   1;
        short hds     :   1;
        short ready   :   1;
    } bit;
    Uchar byte;
} HPI_CONTROL_TYPE;

typedef  union
```

```
{
    struct
    {
        short reset   : 1;
        short dsp_int : 1;
        short spare   : 3;
        short xadd    : 1;
        short spare1  : 10;
    } bit;
    Uint16 byte;

} HPIC_CONTROL_TYPE;
```

F.19 variables.h

```
/*
 *  Personnel Detector
 *  Elliot Ranger
 *  Master of Science Thesis
 *  Massachusetts Institute of Technology
 *  in conjunction with Charles Stark Draper Laboratory
 *
 */
/*****************************************************************************\
* FILENAME..... variables.h
* DATE CREATED.. 04/18/2002
* LAST MODIFIED. 08/06/2002
\*****************************************************************************/

/* global variables */
MSP_PORT3_TYPE msp_port3;
MSP_PORT5_TYPE msp_port5;
HPIC_CONTROL_TYPE hpic;
Uint32 time = 0;
Uint16 seconds = 0;
Uint16 minutes = 0;
Uint16 hours = 0;
Uint16 days = 0;
Uchar in_buffer[64];
Uint16 buffer_write_pos = 0;
Uint16 buffer_read_pos = 0;
Uint16 rf_ena = FALSE;
Uchar display_lines[3][64];
Uint16 display_current_line = 0;
```

```
30  Uint16 dsp_data_address = 0;
    Uint16 dsp_memory_block = 1;
```

Appendix G

Errata

Problem: Part U12 (MAX1672) pins 2, 15, 7, and 10 were tied together but were not tied to PGND.
Solution: Jumper pins 7 and 8 together. Pins 8, 6, and 5 were correctly tied to PGND on the part.

Problem: Connector J5 pin 9 should be the only connection to DGND, but 4,8,10,12,13,and 14 were tied to DGND as well.
Solution: Cut pins 4,8,and 10 to remove the connection to DGND those pins actually connect to circuitry in the FET pod. Pins 12, 13, and 14 can be left uncut because those pins are unused by the programmer.

Problem: Part U1 the RX_IN and RX_OUT are reversed.
Solution: Lift pins 15 and 16 from the circuit board, and run wires to swap the connections.

Problem: J4 and J7 are specified as pins in the parts listing, but should be sockets.
Solution: Change the part to a socket. The hole spacing on the board is the same for both parts.

Problem: GEO_VREF- was never connected to AGND.

Solution: Connect a jumper wire between the GEO_VREF- side of R13 and pin 2 of part U8.

Bibliography

[1] Amirtharajah, R. *Design of Low Power VLSI Systems Powered by Ambient Mechanical Vibration*. PhD thesis, Massachusetts Institute of Technology, May 1999.

[2] Doshi, P., et al. Modelling and characterization of high-efficiency silicon solar cells fabricated by rapid thermal processing, screen printing, and plasma-enhanced chemical vapor deposition. *IEEE Trans. on Electron Devices*, 44(9):1417–1423, September 1997.

[3] Houston, Kenneth M., and Daniel McGaffigan. Spectrum analysis techniques for personnel detection using seismic sensors. *MSS Specialty Group on Acoustic and Seismic Sensing*, September 2002.

[4] Kuo, Sen M., and Bob H. Lee. *Real-Time Digital Signal Processing*. John Wiley and Sons, LTD, 2001.

[5] Oppenheim, Alan V., and Alan S. Willsky. *Signals and Systems*. Prentice Hall, second edition, 1997.

[6] Oppenheim, Alan V., et al. *Discrete-Time Signal Processing*. Prentice Hall, second edition, 1999.

[7] Rowe, D. Demonstration system for a low-power classification processor. Master's thesis, Massachusetts Institute of Technology, February 2000.

[8] Sedra, Adel S., and Kenneth C. Smith. *Microelectronic Circuits*. Oxford University Press, fourth edition, 1998.

[9] Solar Electric Power Association. A primer on solar photovoltaics and pv systems, February 2003. http://www.solarelectricpower.org/power/fact_sheets.cfm.

[10] Succi, G., et al. Acoustic target tracking and target identification - recent results. *Proceedings of SPIE*, 3713:10–21, April 1999.

[11] Succi, G., et al. Problems in seismic detection and tracking. *Proceedings of SPIE*, 4040:165–173, April 2000.

[12] Succi, G., et al. On the design of a small passive sensor for locating vehicles, footsteps and gunshots. *Proceedings of SPIE*, 4232:367–376, April 2001.

[13] Texas Instruments, Inc. *MSP430x13x, MSP430x14x Mixed Signal Microcontroller*, February 2001.

[14] Turkowski, Ken. Fixed point square root. *Apple Technical Report 96*, October 1994.