

NON-INTRUSIVE FUEL CELL LOAD INTERACTION MONITORING

by

John David Lackey

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

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

April 2006

© Copyright

by

John David Lackey

2006

All Rights Reserved

APPROVAL

of a thesis submitted by

John David Lackey

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the Division of Graduate Education.

Dr. Steven R. Shaw

Approved for the Department of Electrical Engineering

Dr. James Peterson

Approved for the Division of Graduate Education

Dr. Joseph J. Fedock

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extend quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

John David Lackey

April 2006

ACKNOWLEDGEMENTS

I would like to thank Dr. Steve Shaw for all time and effort that he contributed to helping me with this project. His expert advice and brilliant ideas provided much needed insight during the most difficult challenges and helped make the project a success. I would also like to thank Dr. Shaw's group for all of their help and cooperation.

TABLE OF CONTENTS

1	INTRODUCTION	1
	Fuel Cell-Load Interactions	3
	Background on the Non-Intrusive Load Monitor	4
2	LOAD MONITORING AT ZOOT ENTERPRISES	6
	Instrumentation	8
	The Sensor Board	10
	Current Sensors	12
	The Sampling Board	13
	Revised Monitoring Hardware	15
	Results from Zoot Enterprises Installation	19
3	DATA COMPRESSION FOR FUEL CELL MONITORING	22
	Data Compression Background	23
	Commercial Compressors and Entropy Coding	26
	Gzip	27
	Bzip2	28
	PPMd	30
	Arithmetic Coding	32
	Lossless Preprocessing	35
	LPC	36
	Other Prediction Models	37
	$qd0$ Transform	38
	Lossy Compression	39
	N -bit Loss	40
	Frequency Domain Representation with Replacement	41
	Wavelet Transform Reduction	42
	Principal Component Analysis	42
4	EXPERIMENTAL PROCEDURE AND RESULTS	44
	Laboratory Results of Updated Monitoring Hardware	44
	Results from Compression Programs	46
	Simulation Results from Lossless Preprocessors	49
	Linear Predictive Coding	50
	Other Lossless Predictors	51
	$qd0$ Transform	52
	Simulation Results from Lossy Compression	53

TABLE OF CONTENTS – CONTINUED

	<i>N</i> -bit Loss	53
	Frequency Domain Representation with Replacement	54
	Wavelet Transform Reduction	56
	Principal Component Analysis	56
5	CONCLUSION	57
	REFERENCES	58
	APPENDICES	60
	APPENDIX A : PCB SCHEMATICS	61
	APPENDIX B : FIRMWARE FOR SX28AC ON THE SAMPLING/DATA TRANSFER BOARD	65
	APPENDIX C : MATLAB COMPRESSION CODE	74

LIST OF TABLES

Table		Page
1	Arithmetic Encoding for the Word “twist”	33
2	Arithmetic Decoding for the Word “twist”	34
3	Comparison of Compression Algorithms	49
4	Current and Voltage Compression	50

LIST OF FIGURES

Figure		Page
1	Conceptual diagram of a hydrogen/oxygen fuel cell.	2
2	The installation of two FuelCell Energy 250 kW DFC300A molten carbonate fuel cells at Zoot Enterprises.	7
3	The monitoring installation at Zoot Technologies.	9
4	The sensor board takes measurements, conditions and outputs the signal so that it can be sampled.	11
5	The sampling board converts the data into the digital domain and transmits that data to the computer.	14
6	The updated data collection installation in the laboratory.	16
7	The PC-104 by Advantech.	18
8	Voltage and current graphs of the zero crossing problem at Zoot Enterprises that caused UPS backups to trigger.	20
9	Data from the fuel cell at Zoot Enterprises and the distribution of that data compared to the same data and distribution with the fundamental frequency removed.	24
10	Predictor model block diagram.	35
11	Trigonometric relationship between abc variables and $qd0$ reference frame [11].	39
12	Voltage and current waveform measurements from the installation at Zoot Enterprises.	45
13	Voltage and current waveforms of the startup transient from a vacume cleaner recorded during testing of the updated monitoring installation in the laboratory.	47
14	Current waveform (blue) compared to restored compressed current waveform (red) and the difference between a larger time segment of the same two signals.	55

ABSTRACT

This thesis presents the development of a non-intrusive method for monitoring fuel cells. The instrumentation demands that only the readily accessible AC output terminals of the fuel cell be monitored, making it easy to install on an existing fuel cell. The monitoring hardware senses the current and voltage output of the fuel cell and converts the measurements into the digital domain. The original algorithm to process the data was the Non-Intrusive Load Monitor (NILM) system which detects specific events, classifies and stores them. The software that is described in this thesis is a data specific compression algorithm that is used to store a continuous stream of measurements so that it can be processed off line. This gives the user the ability to search for critical interactions between the fuel cell and loads in the system after the fact, rather than demanding that load and interaction dynamics be known before the monitor is installed.

INTRODUCTION

Fuel cells have been suggested as an alternative to conventional electrical generating and storage devices for a wide range of applications. Potential applications range from small cells designed to power mobile devices, such as cellular phones and lap top computers, to large fuel cell power plants connected to the power grid. Fuel cells are attractive because they convert chemical energy directly to electrical energy, in contrast to conventional power plants that convert chemical energy to mechanical energy generally via a heat engine, then convert that mechanical energy to electrical energy using a rotating electric generator. High temperature fuel cells have exhaust that can be used for combined cycle generation or in heating applications resulting in a very high overall efficiency. Finally, fuel cells may offer reduced emissions and quieter operation than conventional alternatives. Fuel cells also scale well, while the efficiency of rotating electromagnetic machinery favors large physical size.

Fig. 1 shows a simple schematic of a hydrogen/oxygen fuel cell. The reaction that takes place within this cell is typical of polymer exchange membrane (PEM) fuel cells. The anode side accepts a source of hydrogen which is disassociated to form protons and electrons. The protons are then conducted through the membrane to the cathode while the electrons are conducted through an external path producing electrical power. On the cathode side the protons and electrons react with oxygen to form water as the byproduct. Because the efficiency of this process is not constrained by the Carnot cycle it can potentially realize much more efficient power production than conventional generating devices.

There are many types of fuel cells, but all use the same general scheme of mixed ionic/electronic conduction. Some other examples of the most widely used fuel cell

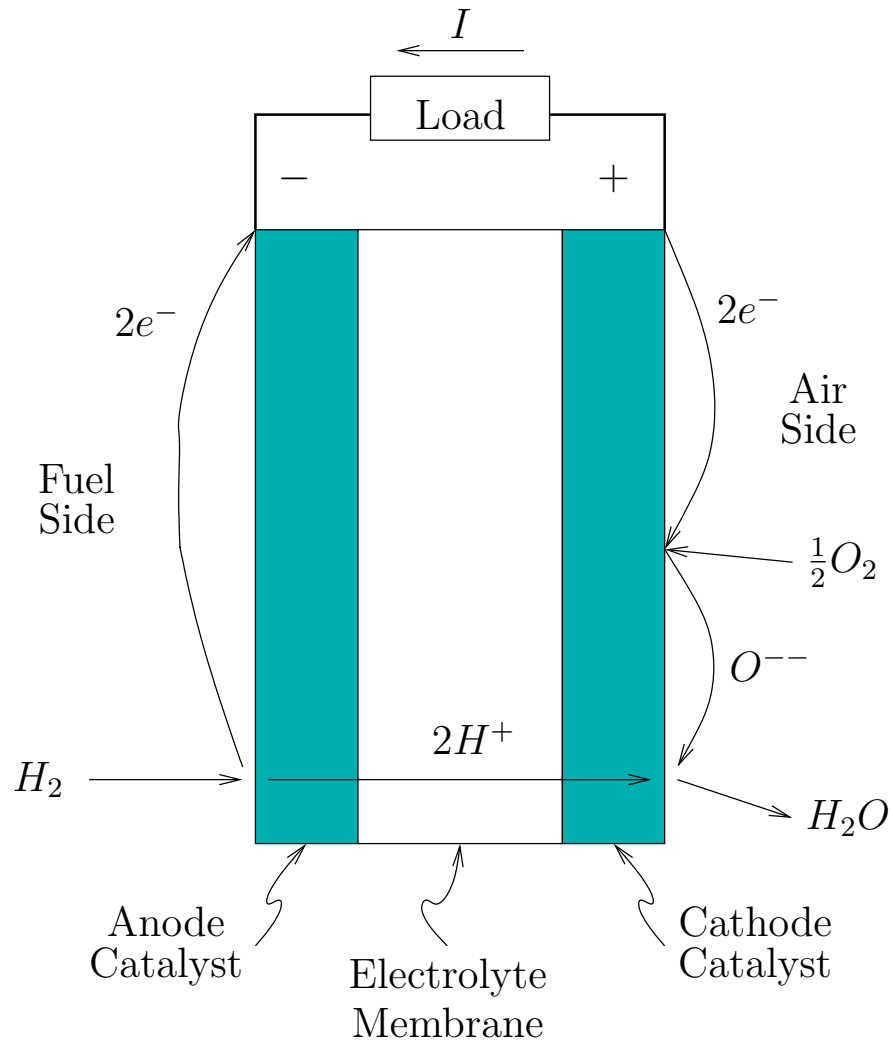


Figure 1: Conceptual diagram of a hydrogen/oxygen fuel cell.

electrolyte technologies are: solid oxide fuel cells (SOFC), molten carbonate fuel cells (MCFC), phosphoric acid fuel cells (PAFC) and alkaline fuel cells (AFC). The main variation between the different fuel cell technologies is the electrolyte and the temperature at which they operate [6]. These differences lead to different ions being conducted through the electrolyte and more importantly the possibility of using a variety of different fuels.

High temperature fuel cells are attractive because they can potentially use fuels other than hydrogen by reforming hydro-carbon fuel sources internally. The waste heat can also be reclaimed for other purposes, dramatically increasing the overall efficiency of the system.

Fuel Cell-Load Interactions

Electric power has historically been supplied by generation facilities using large rotating machinery. These facilities are usually large compared to the individual loads that they must handle. Fuel cells, however, have the advantage of scaling efficiently, so the peak loads that they are subjected to can be on the same scale as their maximum power output. While rotating machines are fairly robust to transient loads it is unclear how fuel cells might interact with loads and networks. One concern is that transient loading of fuel cells could lead to problems within the fuel cell as well as unwanted interactions with different components of the load environment.

It has been shown that unsteady loads can negatively impact the performance and lifespan of both SOFCs and PEM fuel cells [9, 1]. The tests that have been conducted are mainly interested in inverter current ripple and cover relatively short periods of time. By installing monitoring equipment on fuel cells in the field, data that spans

the lifetime of the fuel cell can be collected making long term studies on the effect of different loads possible.

Background on the Non-Intrusive Load Monitor

The Non-Intrusive Load Monitor (NILM) was developed as a multi-purpose load monitor for any system that has an accessible single service utility entry [12]. The goals of this system were to reduce the cost of monitoring power systems and provide a simple way to install the monitoring equipment while maintaining the functionality of collecting information from all of the loads in the system as well as tracking the power quality.

The NILM takes advantage of the fact that the physical work done by different electrical loads exhibit unique transient signatures. For example, all induction motors will have a similar startup transient because they all perform a similar physical task. The NILM can then match real time transient instances to ones in a stored library. Using these matches, each new transient load can be categorized and stored for future evaluation.

In principal the NILM could also be used to discover fuel cell interactions with critical loads. This could be achieved by programming the NILM with transient signatures that are likely to cause critical interactions within the fuel cell system. This thesis considers the application of NILM-style measurements to interactions between a FuelCell Energy molten carbonate fuel cell and the attached loads.

This thesis is organized as follows. The first chapter describes the physical monitoring equipment that was designed for installation on the FuelCell Energy DFC300A 250 kW molten carbonate fuel cell at Zoot Enterprises in Bozeman, MT. The chapter also includes some results from that installation and a discussion of the related

challenges. Preliminary results motivate the development of a new data compression algorithm, described in the second chapter, that replaces the NILM as the data reduction step in the monitoring process. The third chapter contains some simulation results from the data compression algorithms and the results from lab tests on the new system. The fourth chapter discusses the performance of the project and suggests future work.

LOAD MONITORING AT ZOOT ENTERPRISES

In the spring of 2003 two 250 kW molten carbonate fuel cells made by FuelCell Energy were installed at the campus of Zoot Enterprises. The intended use for the fuel cells was to provide redundant backup to the power grid so when there were power outages, the company's operations did not come to a halt. Fig. 2 shows the installation of the fuel cells.

One of the two fuel cells located at Zoot Enterprises was used to conduct initial tests and collect data. These units are FuelCell Energy DFC300A 250 kW MCFCs. This type of fuel cell operates at high temperature and in this case the byproduct heat is used to preheat water for the building's heating system. There is, therefore, no supplemental power production by means of a gas turbine running on waste heat, as is true with some similar installations [8].

The instrumentation that was installed prior to the beginning of this project to monitor the power output of the DFC300A includes three main components. The sensor board takes and conditions measurements from the external sensors. The sampling board samples and transmits those values to the computer. Finally the computer processes and stores the samples. This monitoring system was installed to collect data about how the DFC300A responds to being the sole power source for the entire campus at Zoot Enterprises as would be the case during an outage in the power grid. This mode of operation is termed "island mode" because the facility operates completely independently of the external power grid.

Some critical interactions between loads were determined from the data recorded when the fuel cell was operating in "island mode". Those findings indicated the nature of possible interactions that may be encountered by the monitoring systems



Figure 2: The installation of two FuelCell Energy 250 kW DFC300A molten carbonate fuel cells at Zoot Enterprises.

that were built during this project. There were also some inadequacies found in the data collection system itself. The buffers in the USB device were prone to overflow because the software that was used to read the stream into the computer was not able to keep up with the rate of transmission. This halted the collection of data and caused much of the important information during initial tests to be lost.

Some modifications were needed so the data collection system could take uninterrupted measurements. The largest changes involved the computer that processes and stores the data. It was replaced with a more application specific unit containing upgraded software for receiving, processing and storing the data. Also, the layout of the monitoring system was streamlined for easier installation.

Instrumentation

Two monitoring installations were built to collect data from FuelCell Energy power plants. The first was installed in the preliminary stage of this project and was intended to support NILM data processing [12, 13]. These improvements were then integrated into a second hardware package, described in this thesis, that is scheduled to be installed at Billings Deaconess Hospital in the near future.

The fuel cells at Zoot Enterprises were installed to act as a reconfigurable backup to the power grid. There was an input from the grid and an output to critical and non-critical busses for the facility. There were various configuration options as to which device powered what loads. The input from the grid to the fuel cell and the output from the fuel cells to the combined busses were the most accessible monitoring points. This is where the sensors were installed to monitor the fuel cells. The measurements from the fuel cell to the critical bus proved to be the source of the most useful information.

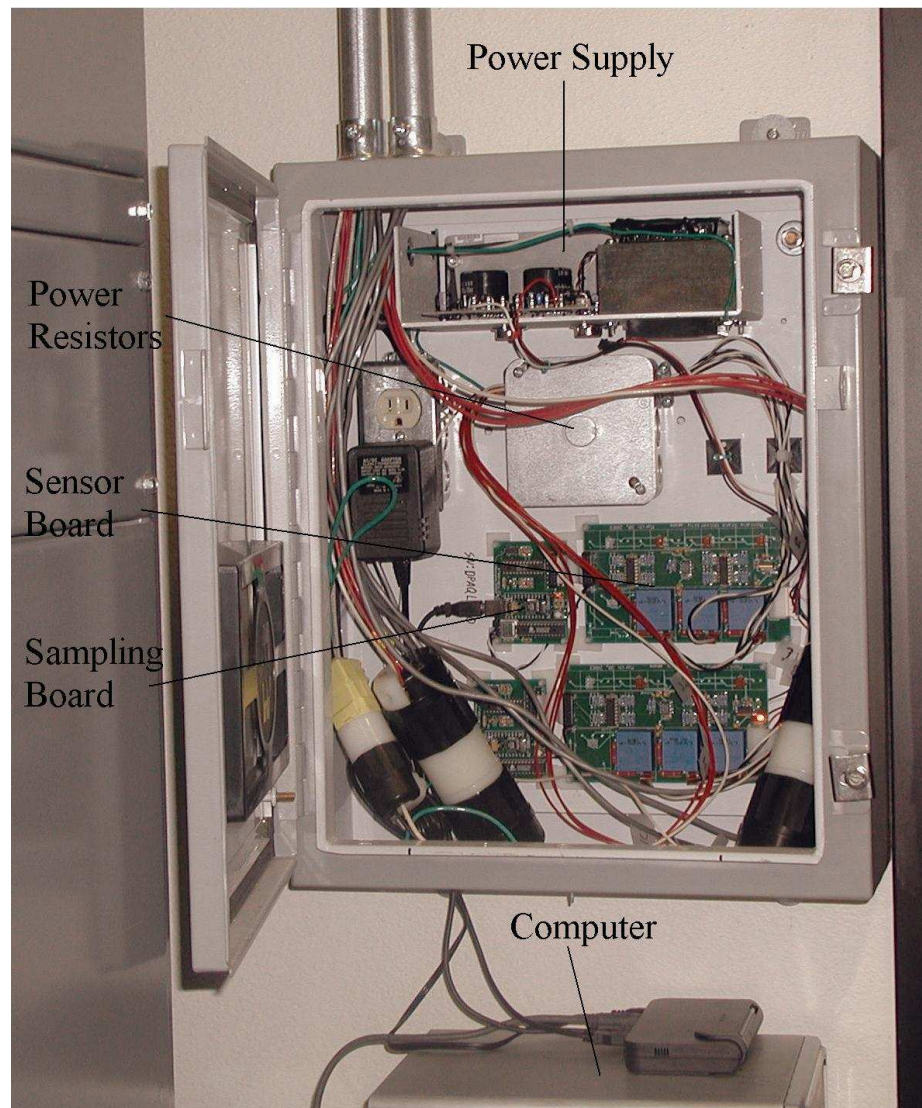


Figure 3: The monitoring installation at Zoot Technologies.

Fig. 3 shows the physical layout of the instrumentation box at Zoot Enterprises. The power supply can be seen at the top of the figure. Below that are an outlet and an enclosure for the power resistors that condition the voltage for the voltage transducers on the sensor board. The sensor boards are oriented on the right side of the box at the bottom while the sampling boards are in the lower portion of the box to the left. Schematic diagrams for all of the printed circuit boards used in both installations are included in Appendix A.

To store and process the data a standard desktop computer was used. The computer is a Shuttle XPC with a 1.15 GHz AMD Athlon processor and 512 MBytes of memory. It has a compact case that measures $7 \times 8 \times 11.5$ inches making it slightly too large to mount to the outside of the enclosure for the monitoring hardware. It has two built in USB 1.1 ports and an expansion card was added with four additional USB 2.0 ports for faster data transfer. It is also equipped with a 10/100Base-T Ethernet connection.

The Sensor Board

The sensor board is designed to accept up to four voltage inputs and four current inputs, condition them and output values between 0 and 4 volts. In the application at Zoot Enterprises only six channels were needed, one to measure each phase of the current and voltage. The input pins for the three current transducers can be seen along the bottom of the board in Fig. 4 and the corresponding voltage transducers are in a row along the top. The voltage is converted by LEM LV 25-P voltage transducers [5] with a 5:2 conversion ratio. These are essentially field canceling current transducers, but they protect the circuitry from the high power signals coming from the fuel cell. They require a positive and negative fifteen volt supply and accept a nominal input current 10 mA rms. To generate this input current, large resistors were

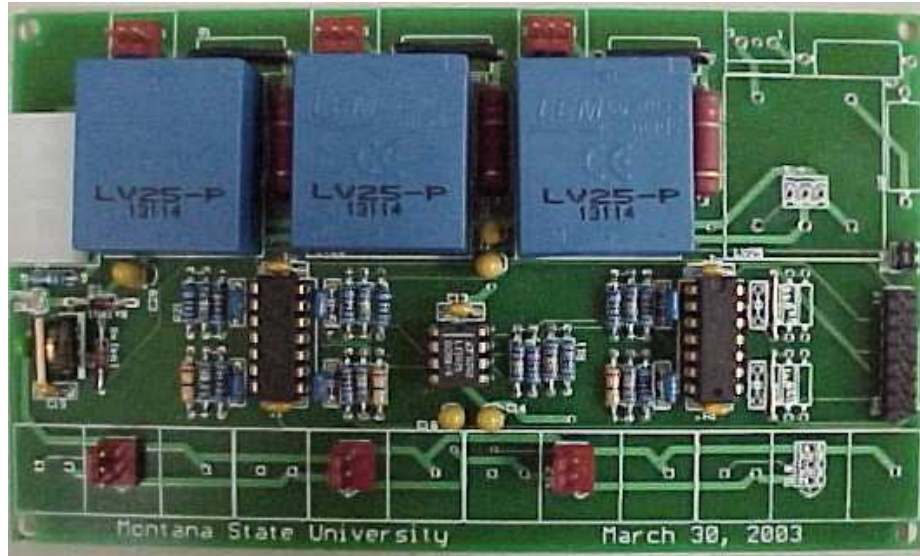


Figure 4: The sensor board takes measurements, conditions and outputs the signal so that it can be sampled.

placed from each phase of the output of the fuel cell to ground. The fuel cells produce 480 V rms on each phase so 48 k Ω resistors were chosen to achieve the nominal input current specified in the product datasheet. This configuration provides a nominal output from the sensors of 25 mA rms with the total range being -34 mA to 34 mA. This value is compared with a reference and amplified using a LT1014DN op amp to offset and gain the signal to be in the range of 0 to 4 volts.

The current is sensed using transducers that are attached to the power lines. The sensors used at Zoot Enterprises are LEM LT 505-S field canceling current transducers [4]. They have a 5000:1 turn ratio, a maximum primary current of 1200 A and have a measuring bandwidth of DC to 150 KHz. The turn ratio reduces the maximum 700 A from the fuel cell to a signal with a maximum of 140 mA as the input to the conditioning amplifiers. Like the voltage the current measurement is then converted to the range of 0 to 4 volts for the analog to digital converter (ADC).

The parts on the sensor board must be fast enough to keep up with the proposed sample rate so bandwidth concerns were taken into consideration when the parts were chosen. The current transducers are capable of measuring frequencies up to 150 KHz, so they will easily measure up to the absolute maximum sampling rate for six channels which is 33.3 kSs^{-1} per channel. The voltage transducers have a response time of $40 \mu\text{s}$ which means that they can measure frequencies up to 25 kHz, although this is less than the absolute maximum, the actual sample rate is under 10 kSs^{-1} per channel so they are also within the bandwidth constraints. To eliminate any problems that may occur in the measurements due to aliasing, a simple RC circuit is included in the amplifier design to limit the bandwidth to around 5 kHz for the 8.33 kHz sample rate.

Current Sensors

Various options are available for the external current sensors. LEM LT 505-S sensors were used at Zoot Enterprises, and are the best option for taking measurements at fuel cells that are similar to the DFC300A. Smaller current sensors from LEM can be installed onto the sensor board itself for applications where the power line that is being monitored has lower current and is more easily accessible. Using these smaller current sensors in conjunction with a high frequency low turn split core sensor can make installing current sensors on preexisting power plants much easier. This can be done without having to halt operations so the power lines can be disconnected to actually install the sensors. It also works well because the interference from the sensor is essentially zero due to the field canceling effect of the on board transducer.

The Sampling Board

The sensor board is connected to a universal serial bus (USB) ADC board. The AD7856AN ADC takes 14 bit samples with an input range of 0 to 4.096 V giving a resolution of about 50 mV for a 480 V rms input signal when the measurements are converted correctly. Two leading control bits are then added to each sample by the SX28AC/DP microcontroller and the two byte words are sent to the onboard USB controller. This device in turn transmits the data to the CPU which stores it in a file.

The physical makeup of the board displayed in Fig. 5 consists of three major components: the AD7856AN ADC on the right side of the board, the SX28AC/DP microprocessor on the far right, and the DLP-USB245M USB controller produced by DLPDesign which is protruding from the board in the middle. The I/O consists of 8 grounded analog inputs from the ADC on the top of Fig. 5, the USB port which supplies the board with power connected to the bottom of the USB controller, and some headers to pull external power for back up just to the right of the USB port. There is also a reset switch on the board that does a software reset on the microcontroller and automatically calibrates the ADC. The clock signal for the SX28AC/DP is provided by a 48 MHz crystal and for the AD7856AN this same signal is divided by eight using a 74ACT161 binary counter.

The SX28AC/DP microcontroller has 20 I/O pins, eight of these are used to transmit bytes of data to the USB controller, five are control signals for the USB controller, four are control bits for the ADC, there are two used as a serial input and output to and from the ADC and one is used to control the indicator LED. The external crystal clocks the microcontroller at 48 MHz, microcontroller software delays limit the single channel sample rate to 200 kHz.



Figure 5: The sampling board converts the data into the digital domain and transmits that data to the computer.

The USB controller board, manufactured by DLPDesign, accepts parallel bytes of data from the SX chip and transmits them to the computer using USB. All of the USB protocol is programmed into the onboard microcontroller and the data interface to the SX28AC/DP is parallel. There are eight inputs for each bit of parallel data, a receive buffer indicator output, a transmit buffer indicator output, a write input and a read input. When the receive buffer indicator is low there is data that has been received via the USB and is waiting to be read from the receive buffer. The transmit buffer indicator goes high when data is being written from the microcontroller to the transmit buffer or when the transmit buffer is full. When the write input goes from high to low the data on the I/O lines is written into the transmit buffer and sent to the computer within the transmit buffers specified timeout value. When the read line is pulled low the current byte in the receive buffer is read and put onto the I/O lines. Appendix B contains the firmware from the SX28AC microcontroller on the sensor board.

Revised Monitoring Hardware

Fig. 6 shows the physical layout of the updated monitoring hardware to be installed at another FuelCell Energy site in Billings, MT. Mounted to the outside of the monitoring hardware box is the new computer. The power supply can be seen inside the box at the bottom. The sensor boards and sampling boards are the same as in the previous installation and can be seen stacked two high on the left side with the sensor boards being oriented closer to the power supply. Finally there is a metal box that contains power resistors to precondition the voltage for the sensor boards.

The outlet that was installed in the original setup was removed because the new computer runs off the DC power supply and has enough USB ports so that an external hub is unnecessary. The sensor and sampling boards are oriented in a stacked fashion

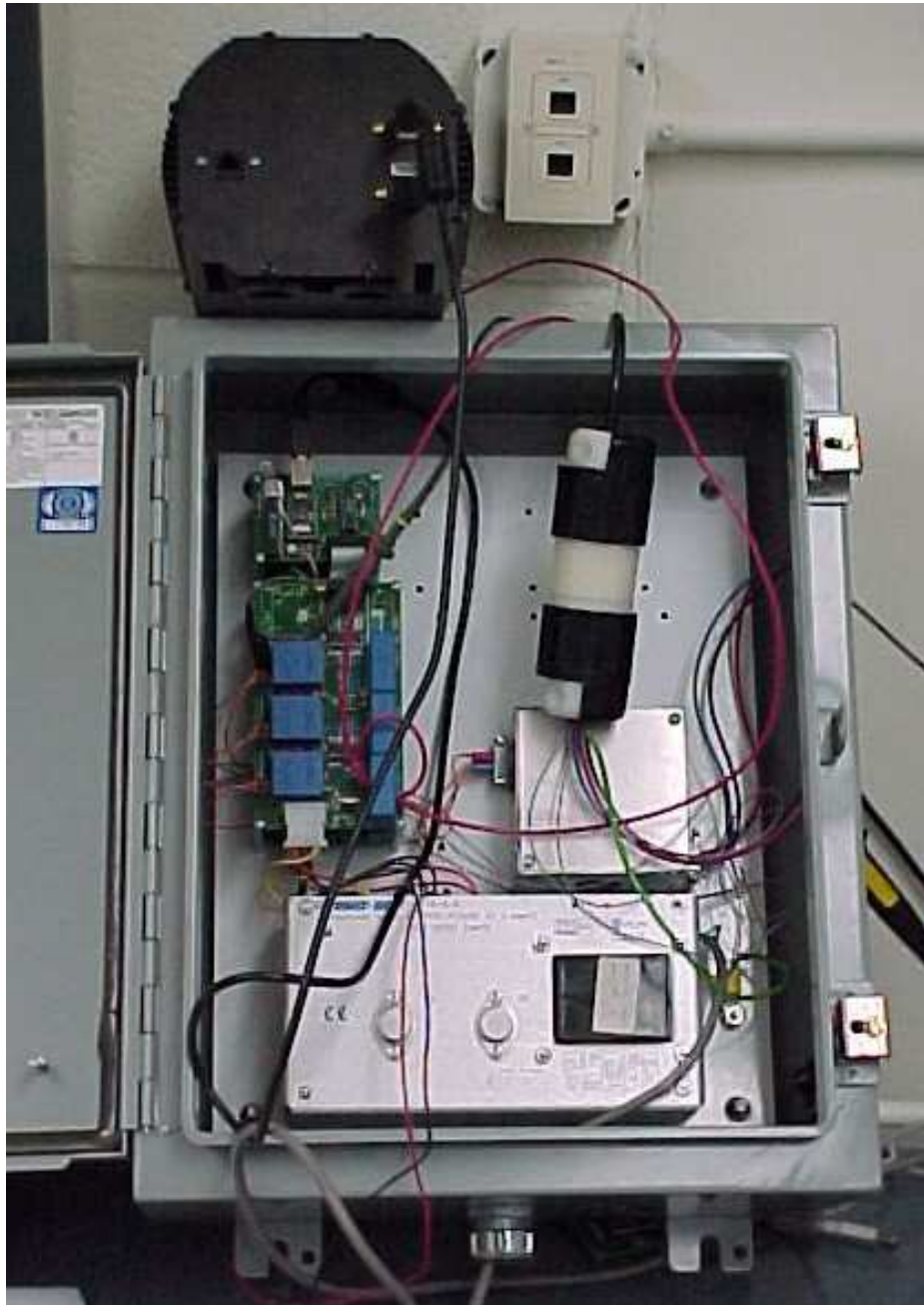


Figure 6: The updated data collection installation in the laboratory.

in the new hardware to conserve space to allow more area for resistors and connectors. Finally the computer is actually mounted on top of the enclosure to make for an all inclusive single unit.

There were physical and performance constraints that had to be met when choosing a computer to work with the hardware. The first requirement was that the physical size of the computer must be small enough to either fit into the box or easily be mounted to the outside of it. This implies that the smallest unit that could meet the performance constraints was desirable. The performance constraints started with the fact that there must be at least two USB ports to transfer data to the computer. It is also imperative that a network port be available so that it is possible to access the machine remotely as it will likely be installed in Billings, MT, and therefore would be inconvenient to physically access.

Originally the NILM ran on a Shuttle PC, which is a small but fully functional PC. The Shuttle was still slightly too large to mount easily to the side of the instrumentation box, so smaller packages had to be considered. The one that met all of the constraints was a PC104+. This is a small board with capacity for all the functionality of a standard PC using stackable expansion cards. The cards are interconnected through the PC104+ bus which is a 120 pin, 32 bit PCI local bus. The Advantech PCM-3380 was chosen because it was the only PC104+ form factor computer that addressed up to 1 Gbyte of memory. It also is equipped with a low power Intel Pentium M 1.6 GHz processor, six USB 2.0 ports and one 10/100Base-T Ethernet connection.

Various software packages have been used to collect and store data, these are all based on Linux platforms, Debian is the distribution that was chosen for this installation. The decision to run the operating system off a hard drive instead of a flash device such as a flash card or USB stick was made in order to ensure that there was enough space to get the necessary functionality out of the operating system. The

hard drive is a 60 GB, 2.5 inch IBM Travelstar laptop hard drive connected to the motherboard using an IDE connection on the CPU module. Debian easily installed on the system and all of the software that was previously being used worked well in the new configuration.

The power module for the computer is produced by Tri-M and is a PC104+ expansion board. It is specifically designed for automotive applications so it accepts a wide range of input voltages. The instrumentation power supply is a 15 V DC source which is converted to 3.3 V, 5 V and 12 V to power different components of the computer.



Figure 7: The PC-104 by Advantech.

The case that physically houses the three stacked components, shown in Fig. 7, is 5×6×7 inches and has a mounting foot on the bottom. The only I/O on the package are four USB ports and one ethernet port. Finally there is a power cord connected to the supply.

Results from Zoot Enterprises Installation

The campus at Zoot Enterprises is equipped with a highly advanced HVAC system. There are many loads that must be considered if the NILM were to be installed on the service utility entry there. It would have been extremely time consuming to enter an exhaustive list of transients into the event library, so the NILM was not used during the initial testing of the monitoring system. The monitoring equipment simply logged a continuous stream of data with no processing.

The monitoring equipment was set to collect and store all of the data during the time when the fuel cells were put into “island mode,” on the morning of August 26, 2003. There were some interesting interactions between devices within the data that contained approximately 80 minutes of information. The most notable was when the UPSs manufactured by APC switched to battery backup to power the servers. These devices are designed to monitor the power to the servers, and when they detect an outage switch the servers to backup batteries. It was observed that the loads from the building were having a direct impact on the power quality from the fuel cells. Instantaneous load variations caused perturbations in the voltage from the fuel cells. One load in particular was found to be causing the problem with the UPSs. Variable speed drives were installed in the facility with no input filters. These drives were drawing a very unsteady current from the fuel cell. The current waveform showed that at times there were large spikes in current output due to the drives. The spikes

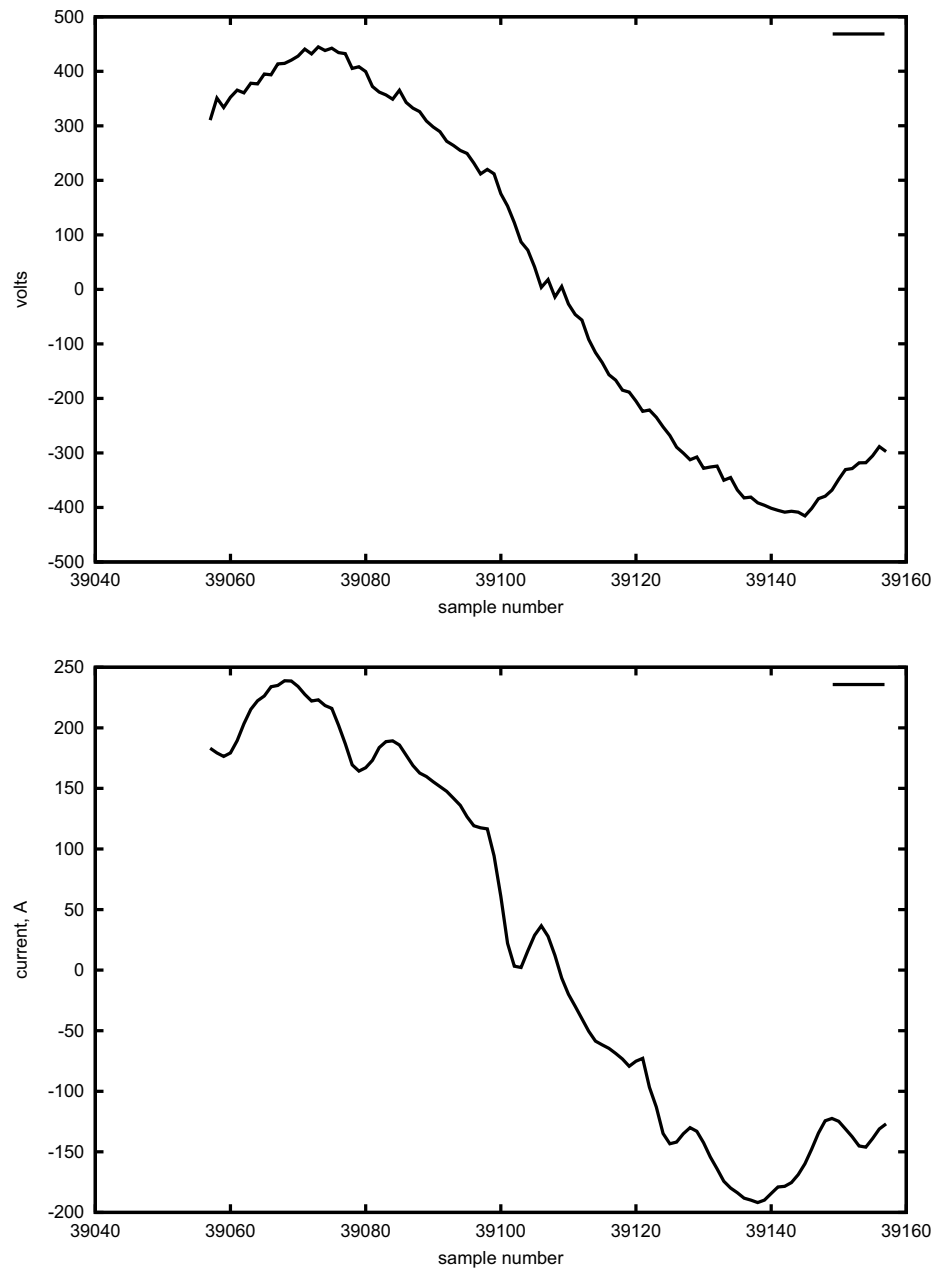


Figure 8: Voltage and current graphs of the zero crossing problem at Zoot Enterprises that caused UPS backups to trigger.

were also seen from measurements made at the drives with an oscilloscope. When these current spikes occurred, there were glitches in the voltage from the fuel cell which periodically occurred near the zero crossing of the waveform, making voltage flicker above and below zero volts twice in a cycle. The resulting power quality issue, shown in Fig. 8, in turn caused the UPS backup power supplies to the server room to determine that the power was unstable, and therefore, switch to battery backup. When the drives were operated for a long enough period of time, the batteries would die causing the computers shut down. It is difficult to determine why the fuel cell system reacted as it did to the transient loading from the limited amount of data that was collected.

When evaluating the interactions that were recorded at Zoot Enterprises, it became apparent that they are comprised of a complex network of loads rather than a single transient occurrence, making it impossible to isolate a single event as the root of the problem. While the NILM is able to collect information related to replicable transients, it would be less effective analyzing a system with complex interactions of multiple loads. This prompted a reevaluation of how the data from subsequent experiments should be reduced so that it is easily transferred and stored, yet still retains the critical interactions.

DATA COMPRESSION FOR FUEL CELL MONITORING

The NILM takes advantage of the fact that the physical work done by different electrical loads exhibit unique transient signatures upon startup. Also loads within a case, for example, induction motors, will have similar startup transients because they all perform physically similar tasks. The NILM uses easily identifiable sections of transient occurrences, called v-sections, to match incoming signals with stored examples from similar loads. A training step is required in which the NILM is presented with a comprehensive library of possibilities within the system that it is monitoring. It uses spectral envelopes to compare the v-sections of transients [13] with those in the library to in order to classify transients.

The NILM requires repeatable transient signals. For example, the NILM was used to detected when a pump system had an obstruction in the flow line [22]. In this case the transient pattern for unobstructed and obstructed flow was similar, but the diagnosis could be made using model-based system identification.

Transients may not be repeatable if they are of sufficient magnitude that they disrupt the voltage, and create complex interactions between loads that would be difficult to anticipate and store in a NILM library. Unfortunately, this is type of interaction is exactly what was observed at Zoot Enterprises.

When applied to a situation such as fuel cell monitoring, the NILM can be viewed, in part, as a lossy data compression scheme. It attempts to identify and store only information about interactions while discarding the remaining data. The NILM also identifies when and where an interaction has occurred. For fuel cells it may not be possible to do this using only one method to localize failures. It can be very difficult to anticipate failures when monitoring fuel cells, making lossless compression necessary.

In the remainder of this chapter a number of data compression schemes are considered for collecting fuel cell data in the field. It is necessary to compress data primarily because the bandwidth for NILM installations is often limited to dial-up connection speeds. When using the usual sampling rate of 8333 ksamples/sec for six channels of data, a 10:1 compression ratio is needed.

Data Compression Background

Compression algorithms can be divided into two major categories, lossless compression and lossy compression. Lossless compressors use numerical techniques to reduce the size of a data set by representing each symbol in the data set using the fewest possible amount of bits. Lossy compression is not concerned with preserving the data exactly. This type of compression can achieve much greater compression ratios by discarding information in the signal that is of the least significance according to a given metric.

The two components that are used in order to implement most of the compression schemes in this section include a preprocessor that alters the distribution of the symbols in the dataset and an entropy coder that actually performs the compression. In general, the preprocessor uses a model to redistribute the symbols in a way that can be compressed more than the original signal. This is generally accomplished by removing any underlying structure leaving the signal noise as the result. The model must then accompany the new data when it is compressed so that the original can be reconstructed later. The entropy coder can then take advantage of the statistical properties of the data to assign codes that minimize the average number of bits needed to represent each symbol.

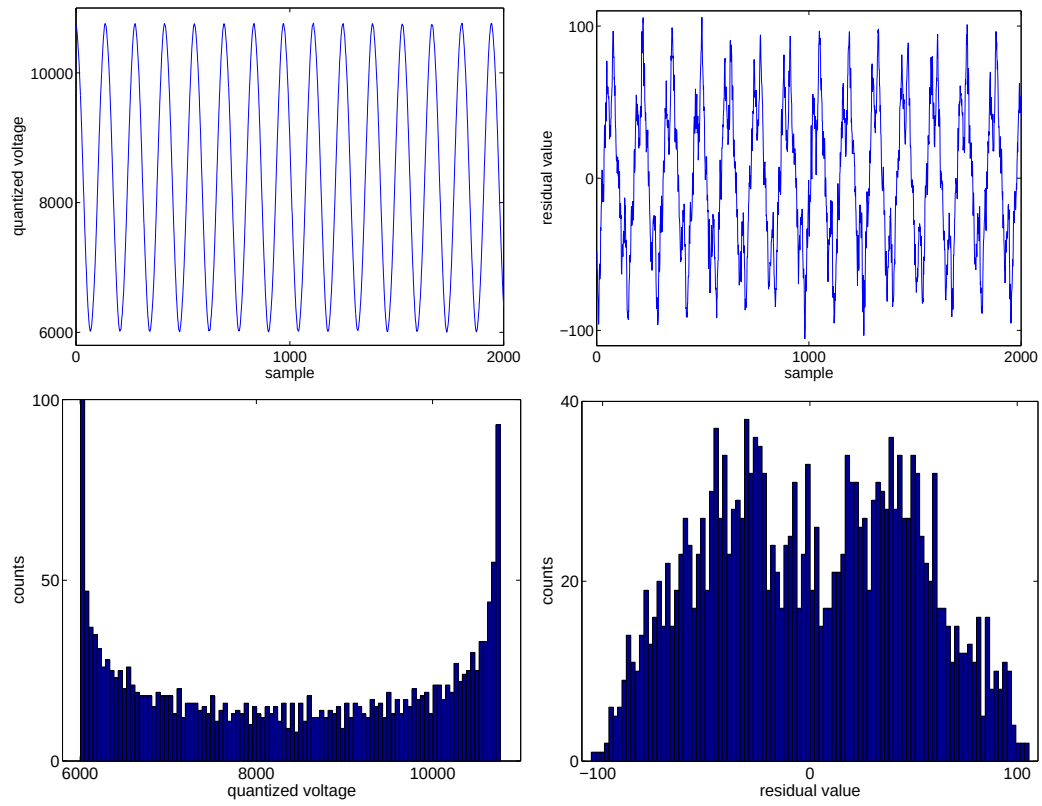


Figure 9: Data from the fuel cell at Zoot Enterprises and the distribution of that data compared to the same data and distribution with the fundamental frequency removed.

Fig. 9 shows a block of data that was collected from Zoot Enterprises and the distribution of that data compared to the same data and its distribution with the fundamental frequency removed from it. The data is plotted in quantized voltage vs. sample. The voltage is simply the value as converted by the A to D converter, so each symbol in the alphabet represents a range of approximately 50 mV, while each sample represents about 120 μ s. The probability distributions are approximated by taking a histogram of the data, which is in effect a sample probability density function. This example illustrates that even using a simple model the probability distribution function can be altered drastically. In this case a single sine wave was extracted from the signal making the number of symbols in the alphabet drop from around 5000 to around 200. It is this type of structure that can be removed from a signal that can lead to much greater compression.

Claude E. Shannon was one of the pioneers of data compression theory. In his 1948 paper, “A Mathematical Theory of Communication” [21], Shannon proposed the basis for modern data compression theory. He established that there is a fundamental limit on the compressibility of any given data when using lossless compression. He called this limit the entropy rate, denoted by the symbol H .

To determine the entropy of a given signal, the statistical properties of the data must first be understood. If the data is completely uncorrelated the only useful statistical property is the distribution of the symbols in the signal. This first order entropy rate [21] can be calculated using

$$H = - \sum_{i=1}^n p_i \log_2 p_i \quad \text{bits/character}, \quad (1)$$

where H is the entropy of a given set of data, n is the number of symbols in the alphabet and p_i are the probabilities of each member of that alphabet.

Conditional probabilities can be used to further reduce the entropy. For example it is very likely that a ‘u’ will follow a ‘q’ in the English language therefore the conditional probability of this happening is nearly 1. The conditional probability of each symbol in the alphabet given the previous N symbols can be used to assign a code to the next letter in the data set. The general entropy equation uses the correlation of all the characters in a data set to determine the entropy limit.

White noise is the natural enemy of data compression because it has no structure. When compressing white noise the entropy rate is determined purely by the first order model. Shannon calculated the entropy for continuous white noise with variance σ^2 to be

$$H(x) = \log \sqrt{2\pi e} \sigma \quad \text{bits/character.} \quad (2)$$

The minimum entropy for compressing the data from the fuel cells with no loss can be calculated using the variance of the measurement noise in (2).

Commercial Compressors and Entropy Coding

There are a number of general purpose compressors available. The three that were considered for the application of compressing the voltage and current waveforms from fuel cells were Gzip, Bzip2 and PPMd. All three of these are UNIX based compressors and are open source. Many of the commercial compression software packages use the same basic algorithms and achieve similar performance to these three. These programs are optimized for text compression, however, some proved more useful than others for compressing analog waveforms.

One more useful compression algorithm that was considered is arithmetic coding. There are no widely used open source or commercially available compression programs that use arithmetic coding to perform the entropy coding step of compression, so

built in MATLAB routines were used to test the performance of this algorithm. This section gives an overview of each of the three compression programs mentioned above, as well as arithmetic coding.

Gzip

Gzip uses the first Lempel-Ziv method (LZ77) [26] abbreviated LZ77 because it was introduced in 1977. This method makes use of a sliding window to encode the data stream. A large buffer of already encoded symbols, or search buffer, and a smaller buffer of yet to be encoded symbols, or look ahead buffer, are scanned each time a symbol must be encoded to determine if the next few symbols have been previously seen in the same arrangement. This allows for the encoder to only write a token with the data that is necessary to reconstruct the stream. The token consists of two items. First is the offset from the current symbol to be compressed to the start of a chain in the search buffer that is identical to the current symbol and the next N symbols in the look ahead buffer. Next is how long the two identical chains are, this is the number N mentioned before. After this token is recorded, the buffers are shifted forward by N symbols and the process is repeated.

Gzip uses some other techniques to optimize the algorithm [7], these also offer a tradeoff between speed and the amount of compression achieved. The match lengths and match distances are recorded in two separate Huffman trees, each stored in a separate block that must be small enough to fit in the memory of the machine. The blocks are terminated when an algorithm decides that it would be advantageous to start a new block and begin new trees. Duplicate chains are found using a hash table with all input chains of length three or greater. An index is calculated for the current chain so the hash table can be searched starting with the most recent entries, thereby taking advantage of Huffman coding. Matches that are too old are discarded as they

are outside of the 32 kbyte search buffer, however, no chains are ever discarded until the whole tree is reset.

The Gzip command line options `--best` and `--fast` determine how good of a match is acceptable before that match is recorded and the buffers are shifted. When the user enters `--best` a comprehensive search of the hash tree is performed each time a string is to be compressed to find the absolute longest match possible in the search buffer. The options for the trade off between computation speed and compression fall in the range from `-1` to `-9` entered from the command line. The fastest computationally is the `-1` option which is the same as `--fast`, while `-9` is the same as `--best` and offers the greatest compression. Any number between 1 and 9 can be used as a compromise between the two extremes.

Bzip2

The Burrows-Wheeler method (BW) [2], used by Bzip2, is a block sorting algorithm. It reads the input stream block by block and then encodes each block separately as one string. BW works well on many kinds of inputs including images and sound, however Bzip2 is optimized for text. The block size must be large to achieve the best compression ratios. The Bzip2 command line arguments `--best`, `--fast`, and `-1` through `-9` simply change the block size from 100 Kbytes to 900 Kbytes, 100 being the fastest and 900 being the best. A block size of 100 kbytes is large enough to realize the benefits of the BW method for most data so the command line options do not significantly alter the performance of Bzip2 and are offered mainly to comply with Gzip notation.

The general idea behind the BW method is to reorder the original input string, S , in a permutation that has high concentrations of a few symbols in certain regions of the rearranged string, L . This reorganization means that the new string can be

easily and efficiently compressed using techniques such as the move-to-front method or Huffman coding. Of course, it is also necessary to be able to reconstruct S from L .

A high level description of the implementation of the BW method is as follows [16]:

1. String L is created, by the encoder, as a permutation of S . Some extra information, denoted by I , is also created to be used later by the decoder.
2. The encoder compresses L and I and writes the results to the output stream. This step is achieved using Huffman coding [19] in Bzip2.
3. The decoder reads the output stream and decodes it by applying the reverse operation for Huffman coding resulting in string L and variable I .
4. Both L and I are used by the decoder to reconstruct the original string S .

The first step of the BW method is to create the string L from the original input S and understand what extra information needs to be stored for decoding. In order to do this a $n \times n$ matrix must be constructed, where n is the length of the string. The matrix contains S in the top row and each subsequent row is the one above it cyclically shifted left by one symbol. The matrix is then sorted lexicographically by row, this produces a sorted matrix where every row and column are permutations of S . The last column of the sorted matrix is the string L , this is the one selected to be compressed by the encoder. The other information that is needed to reconstruct S is the index in L of the last letter of original string [2, 16]. This information can then be compressed using Huffman coding [19, 20].

In order for this compression to work it must be possible to reconstruct S , knowing only the string L and initial index I . The first step is to perform the reverse operation

for Huffman coding to retrieve the string L and index I . Next, the string F must be created, this is just the symbols in S sorted, it is the same as the first column of the sorted $n \times n$ matrix. Keeping track of the relationship of where the symbols of L are located in string F creates another string, denoted T , of index pointers to the next symbol in the decoded stream. Now F is no longer needed and can be erased. The decoder can now use the information from T , L and I to reconstruct the string, S , symbol by symbol. By simply writing the letter that appears in index I of string L and then going to the next index as pointed to by the value in index I of string T and repeating with the new index the original string is recovered. In order to gain the benefits of the BW method, symbols in the input stream must tend to regularly follow one another so that the chosen permutation of the original string contains concentrations of symbols that are the same.

PPMd

Prediction with Partial string Matching (PPM) [3] is a compression scheme that takes advantage of previous knowledge about an input stream. It uses a variable length library of symbol strings to assign probabilities of what the next symbol may be and uses those probabilities for optimal encoding. The library of strings comes from previous observations recorded from the input data stream. Using only previous observations allows for very simple decoding, as the decoder is seeing the exact same stream as the encoder, therefore, the models can be updated in the same way. In other words the decoder is the same as the encoder except that it decodes the next character instead of encoding it.

The PPM compression scheme compiles a library of strings with lengths ranging from 0 to N . Each string in the library also contains information about the probabilities of each symbol that may follow that string. This library is used to determine the

probabilities of the next character. To encode a character in a string PPM looks at the highest order strings first to see if there is already a probability associated with the current character and if not it moves to a lower order context.

For example, if the previous three characters in the stream are “the” and the highest order instances that are recorded are $N = 3$, then probabilities can be assigned based on which letters have followed this string in the past. Supposing that “the” has been seen 27 times already. This string was followed by the letters r (11 times), s (9 times), n (6 times) and m (once), the encoder can then assign the aforementioned letters the probabilities of $11/27$, $9/27$, $6/27$ and $1/27$ respectively. If the next character in the stream is one of these letters then it can be encoded using Huffman coding and will be represented in the least amount of bits possible.

If the next letter, such as a, has not yet been seen in the third order context then the probability of its occurrence is 0 and it can not be encoded so the encoder must switch to a lower order context. Now the encoder considers the probabilities of characters following the string “he” which has been seen 54 times, followed by a (26 times), r (12 times) etc. At this point the character, a, can be encoded using the probabilities from this context. This process can go to a zero order model where the encoding probabilities are based purely on past occurrences of a character, or even a -1 order model where each letter in the alphabet is given the same probability [16].

This method for determining the probabilities of the next character may look familiar because it is a simple Markov model. An escape key must be assigned a probability just as each letter is for when the encoder must move down one context order. The escape key will be sent frequently at the start of the compression process and less frequently as the library begins to fill so assigning a probability to the escape key must use that information. There are many techniques for assigning probabilities to the escape key and this is where the many variations of PPM differ.

The PPMP variation uses Poisson distributions to assign the probabilities. PPMA, PPMB, PPMC and PPM* use simple numerical methods to assign probabilities. For example, PPMA uses a sum of ratio of the total number of times each symbol has been seen in a context to one more than the total number of times that a context has been seen to assign the return probability. PPMB is similar except it regards symbols that have only been seen once in a given context to be outliers and therefore does not consider them in the calculation.

The application PPMd (PPM by Dmitry) uses a technique that assigns probability using information inheritance from higher order nodes [23]. It is designed to be a computationally efficient technique that still realizes close to the best compression available. The PPMd read_me file specifically says that this algorithm should not work well for noisy analog data [24], such as we have. This is interesting because when PPMd is given optimal parameters it results in the best compression of any of packages discussed in this section. It is, however, slower than some of the others.

Arithmetic Coding

Arithmetic coding [25, 18] represents symbols in a fractional amount of bits giving it an advantage over other schemes such as Huffman coding, which is used in the three previously mentioned programs, that must represent each symbol as an integer number of bits. Arithmetic coding achieves near optimal entropy rates for data that has relatively small symbol sets, such as the English alphabet. In using this technique assuring that the symbol set will not grow too large is necessary because if it does the model that must be stored will counteract any improvement in the compression rate from other techniques.

The general idea behind arithmetic coding is to represent the entire input sequence using one very long code. The algorithm starts with an interval and then reads the

input stream symbol by symbol narrowing the interval as specified by each input. Let the initial interval be $[0, 1)$ and the symbol set be $[A\ B\ C\ D]$ where the probabilities of the respective symbols are $[.1\ .5\ .3\ .1]$. The initial interval is distributed by probability into four sub-intervals: $[0, 0.1)$, $[0.1, 0.6)$, $[0.6, 0.9)$, and $[0.9, 1)$ representing each of the four symbols. If the first symbol in the input is B then the coder records the new interval as $[0.1, 0.6)$ and reassigns the subintervals according to probability once again: $[0.1, 0.15)$, $[0.15, 0.40)$, $[0.40, 0.55)$, and $[0.55, 0.6)$. This narrowing of the interval is repeated until the input is exhausted resulting in a very small interval. The final code can be any number within that interval.

Table 1: Arithmetic Encoding for the Word “twist”

Char.	Bound	Calculation of New Bound	New Bounds
t	Low	$0 + (1.0 - 0) \times 0$	$= 0$
	High	$0 + (1.0 - 0) \times 0.4$	$= 0.4$
w	Low	$0 + (0.4 - 0) \times 0.4$	$= 0.16$
	High	$0 + (0.4 - 0) \times 0.6$	$= 0.24$
i	Low	$0.16 + (0.24 - 0.16) \times 0.6$	$= 0.208$
	High	$0.16 + (0.24 - 0.16) \times 0.8$	$= 0.224$
s	Low	$0.208 + (0.224 - 0.208) \times 0.8$	$= 0.2208$
	High	$0.208 + (0.224 - 0.208) \times 1.0$	$= 0.224$
t	Low	$0.2208 + (0.22208 - 0.2208) \times 0$	$= 0.2208$
	High	$0.2208 + (0.22208 - 0.2208) \times 0.4$	$= 0.22208$

Take the word “twist” for example. The symbol set is now represented as $[t\ w\ i\ s]$ with the respective probability breakdown of $[.4\ .2\ .2\ .2]$. The intervals are then found to be: $t\ [0, 0.4)$, $w\ [0.4, 0.6)$, $i\ [0.6, 0.8)$, $s\ [0.8, 1.0)$. Table 1 follows the compression algorithm as it works through the string. The final bounds indicate that word “twist” can be represented by any number in the range of $[0.2208, 0.22208)$. A good choice for this string would be 0.221 or 0.222, allowing the string to be represented by three characters.

To decode the string the frequency table must be known so the intervals can be reconstructed, this information must be stored with the compressed data making this kind of compression less effective on data with large symbol sets. Once the frequency table is known the decoder works on the string by first determining where the code falls in the symbol table then removing that symbols affect from the code. Table 2 steps through the decoding process using the same example as above to demonstrate decoding.

Table 2: Arithmetic Decoding for the Word “twist”

Char.	Current Code	Calculation of Next Symbol Code	Next Code
t	0.222	$(0.222 - 0)/0.4$	$= 0.555$
w	0.555	$(0.555 - 0.4)/0.2$	$= 0.775$
i	0.775	$(0.775 - 0.6)/0.2$	$= 0.875$
s	0.875	$(0.875 - 0.8)/0.2$	$= 0.375$
t	0.375	$(0.375 - 0)/0.4$	$= 0.9375$

Even in this simple example, some of the implementation problems with arithmetic coding become evident. The first difficulty being that the final value when decoding should be 1. This is solved by adding an eof (end of file) symbol to the end of the encoded string or recording the original number of symbols in the stream with the compressed data so the decoder knows where to stop. The next problem is that the entire input is represented as one very long number, making fixed point representation difficult. The solution for this is to store the most significant digits of the high and low limit when they become the same, because they will not change with subsequent calculations. This allows for much more efficient calculations because they are only required to operate on the least significant digits.

Lossless Preprocessing

In order to realize greater gains using the compression programs mentioned above it is helpful to preprocess the data. By preprocessing, much of the structure can be eliminated from the data leaving only the signal noise. The entropy coding step of the programs can then produce better coding for a smaller set of symbols.

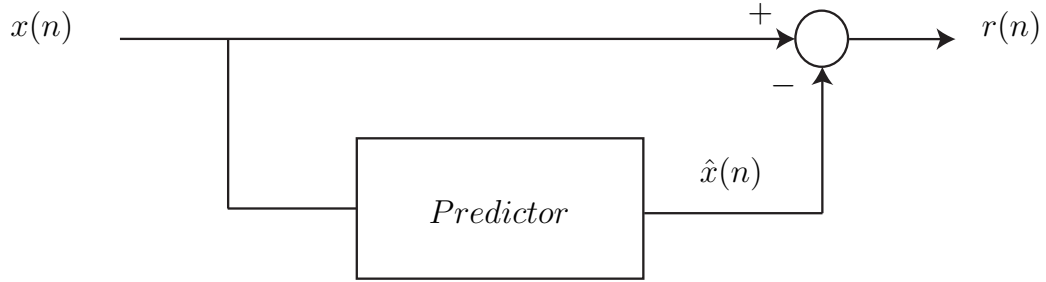


Figure 10: Predictor model block diagram.

A signal predictor model [18] represented in Fig. 10 was used to implement most of the lossless preprocessing algorithms. This model simply uses previous values of the input stream to predict the next value. It then outputs the difference between the predicted value and the actual value of the next sample, this output is called the residual sequence. The signal predictor model can be represented as

$$r(n) = x(n) - \hat{x}(n) \quad (3)$$

where $x(n)$ is the actual data, $\hat{x}(n)$ are the predicted values and $r(n)$ is the residual sequence. In order to restore the original data from the residual, the only extra information that must be stored is the model itself. If the model is good, the residual sequence should be similar to white noise and with a much smaller variance than the original data resulting in a greater compression ratio. Matlab code of the following lossless preprocessing schemes can be found in Appendix C.

LPC

Linear Predictive Coding (LPC) is one of the first methods of lossless preprocessing that was tested. The prediction model is a FIR filter of order N optimized to predict the next value in the stream with the least amount of error. The general form of a linear predictive filter is a weighted sum of the previous N samples as follows,

$$\hat{x}(n) = \sum_{i=1}^N h_i x(n-i), \quad (4)$$

where h_i represents the filter coefficients. To determine the coefficients of the filter one must solve the over determined problem,

$$\begin{aligned} A &= [z(i-1) \dots z(i-N)] \\ A * h &= z(i) \\ h &= (A^T A)^{-1} A^T z \end{aligned} \quad (5)$$

where the data is represented as the vector z with the argument denoting a shift in time. Solving for h provides filter coefficients that minimize the residual sequence in a least squares sense over the entire data set.

Solving for the filter coefficients in this manner can introduce bias in the predictions. Methods such as the Yule-Walker approach can be used to correct for this bias [14], however, the above solution was attractive for data compression applications because of its simplicity. The goal in prediction coding is not to find an unbiased estimate, but to improve the statistics of the residual.

By using the entire data set to calculate the prediction coefficients, the linear predictive filter incorporates all of the signal structure into one simple model. This approach works well when the system does not change drastically with time. The data from fuel cells does not generally see large variation with time, making this type of prediction a good candidate to model the underlying structure in the data. When

the fuel cell system does see variations in its output the residual sequence will have a greater variance, this could be used as one indication that there may be useful information contained in that window of data.

Other Prediction Models

While LPC is a general model for predicting the data in a stream, the data from the fuel cell contains some known underlying structure. For example, the fundamental frequency of the signal is 60 Hz, and the fundamental, along with the odd harmonics should contain most of the signal energy. This knowledge provided a stepping stone for some other prediction models such as a Luenberger observer and one that estimated the amplitudes and phases of the odd harmonics of the fundamental.

The Luenberger observer [17] is a full-order state observer. It uses the inputs and outputs of the plant to estimate the plant state. Using the time sequence as the input and the signal as the output the plant simply generates a sine wave of the correct frequency. Using the plant variables to create the observer variables makes it possible to predict the next internal state of the plant. Finally, inserting that predicted state into the plant results in a prediction of the next value in the sequence. This prediction can then be subtracted from the actual value to form a residual sequence.

The drawback of using a full state observer is that knowing the plant well is necessary for an accurate estimate of the state. In this case a simple sine wave was a reasonable representation of the voltage signal, which was very similar to a sine wave. The current, however, had much more structure, so the observed state could not provide an accurate prediction of the next output. This means that a general purpose plant is not available for the data and it would be very difficult and computationally expensive to determine a reasonable plant for each set of data.

The next prediction model followed from the assumption that, like many physical systems, the bulk of the signal energy is contained in the odd harmonics of the fundamental frequency. This predictor model begins with an initial guess of the frequency and amplitude of the fundamental and the third harmonic. It then uses Levenburg-Marquardt nonlinear least squares [10] to determine the parameters of the model. In this case the parameters are the DC offset of the signal, the fundamental frequency, and the amplitude and phase shift for each of N odd harmonics of the fundamental frequency.

qd0 Transform

One more potentially useful characteristic of the signal is that it is three phase. This means that the three voltage and current signals should be very similar in nature and 120° out of phase with each other. It is possible to use the relation between the waveforms to transform them into an arbitrary frame of reference. This is simply a change of variables. In changing the variables it is advantageous to make the resulting signal better suited for the desired application.

Transforming the variables from a three phase, or *abc*, sequence to a *qd0* sequence [11] simply changes the frame of reference to one that is stationary with respect to the signal. It may be helpful to consider the transformations as trigonometric relationships. Initially the variables rotate about the reference frame displaced 120° from each other. A new reference frame can be defined that rotates at the same angular velocity, ω , as the variables. By placing the variables in this new reference frame they become constant. This transformation is therefore beneficial for predicting the next value of the signal because it should not change with time. Fig. 11 shows the trigonometric relationship between the *abc* variables and the *qd0* reference frame.

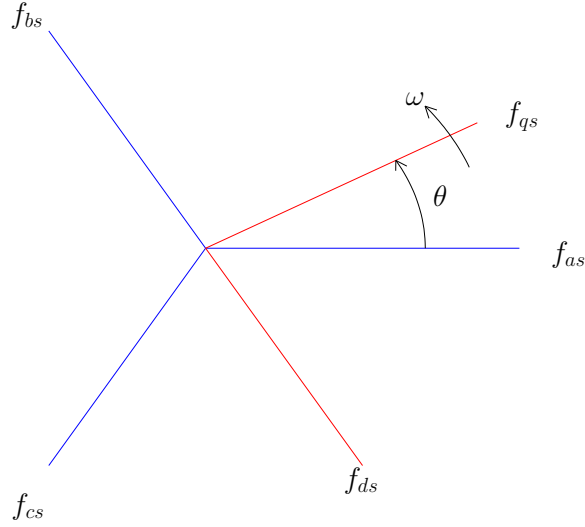


Figure 11: Trigonometric relationship between *abc* variables and *qd0* reference frame [11].

The overhead of performing a *qd0* transform is also very low. The only additional information that must be stored to transform the signal back into the *abc* reference frame is the angular velocity of the new reference frame, ω .

Lossy Compression

Lossy compression is used in applications where it is not critical that the compressed data be reconstructed exactly. Perceptual compression techniques are the used widely in the field of lossy compression. These are methods that are used on information that is output to an end user as a signal that they then must perceive. A parallel can be drawn to the NILM in that it retains only the information that is perceived to be useful for analyzing load interactions. Perceptual techniques include jpeg and mpeg compression, where the end user either sees or hears the resulting reconstructed signal as a picture or sound, not necessarily as a set of data.

Whereas perceptual compression is not concerned with the error between the reconstructed signal and the original, waveform compression is. This type of lossy compression is meant to retain as much of the original information as possible while converting the signal into a form that can realize greater compression. It aims to retain the original shape of the waveform with the total loss being minimized. This is the type of lossy compression that is more appropriate for data collected from fuel cells because it is likely to preserve more information about critical interactions which can not be anticipated.

When implementing any lossy compression it is necessary to understand how much loss is permissible. In the case of audio and images, it may be acceptable to have large errors in some parts of the signal as long as the reconstructed media can not be deciphered from the original. For the application of compressing fuel cell data it is desirable to keep the loss for any given bit under a certain threshold, so when the waveform is reconstructed, as much of the information about load interactions as possible is retained. Appendix C contains Matlab code of the following lossy algorithms.

N -bit Loss

Initially a very simple lossy compression scheme was tested. The idea is to analyze the distribution of symbols and determine which ones could be easily combined so that the resulting distribution contains fewer symbols. To do this a histogram of the data is created and for each symbol in the histogram it is determined what other symbol within N bits of the original has the greatest number of occurrences. The original symbol is then mapped to the high probability one. Using the mapping scheme described above the data is altered so that there are fewer low probability symbols. This means that the maximum loss for any given symbol is N bits, and

that the distribution contains less symbols and should, therefore, lead to greater compression.

Frequency Domain Representation with Replacement

The next scheme to compress the data follows from the idea used in the lossless predictor that modeled the signal as odd harmonics of the fundamental frequency. This scheme differs from the lossless one in that it calculates which frequencies have the highest energy by transferring the signal into the frequency domain using a Discrete Cosine Transform (DCT). In doing this a signal can be reconstructed that is very similar to the original using only a fraction of the frequency content. The drawback to this solution is that the resulting signal can have areas with large error.

An error threshold is used to find high error portions of the signal. Any point in the reconstructed dataset that is outside of this threshold is replaced with the corresponding value from the original signal. For this to be possible, the original values of the high error data points and their indices must be stored along with the frequency information. Lossless compression can then be applied to the error replacement data points to achieve even greater compression.

A probability density function can be created using the error between the signal reconstructed with a fraction of the frequency data and the original. This information can then be used to choose an error threshold that is not so high that the data continues to have excessive loss and one that is not so low that the gains in compression are nullified because many data points must be stored.

This technique provides an acceptable solution. Lowering the loss, however, would still be beneficial. The performance of other transforms was tested to see if they were capable of lowering the variance of the error signal so the initial reconstruction could be done using a lower percentage of the content of the signal.

Wavelet Transform Reduction

The previous method uses both time and frequency information to store the dataset. Wavelets [10] have been successfully used in data compression applications such as image compression and may be useful in this context because the nature of the transform itself stores signals with variations in both time and frequency. When using a wavelet transform, signals are represented with the lower frequencies spanning larger temporal sections of the signal while higher frequencies become more and more localized. The potential to find high frequency transient occurrences with this type of representation is very promising.

This method uses a Discrete Wavelet Transform (DWT) much the same as the previous one uses the DCT. The signal is first transformed using the DWT, and then the wavelet coefficients with the lowest energy are discarded so the signal can be reconstructed with less data.

The DWT uses a mother wavelet that is scaled in both time and amplitude to transform the signal. There are many mother wavelets to choose from, Daubechies wavelets are a family of wavelets that are simple, but also yield good performance. For testing the fuel cell data, Daubechies wavelets with order $M = 4$ and $M = 8$ were used to transform the signal.

Principal Component Analysis

Principal Component Analysis (PCA) [10] is a generalized transform that is optimized to a given set of data. The transform consists of the eigenvectors of the covariance matrix from the measurement vectors. In the case of the fuel cell data there are six measurement vectors so the covariance matrix will be a 6×6 matrix that characterizes the correlation between each data vector. The transform matrix,

M , is the eigenvectors the covariance matrix arranged as columns of a new 6×6 matrix. This ensures that the covariance matrix of the resulting transformed data will be a diagonal matrix, in other words the transformed data will be uncorrelated. By only using the most significant eigenvectors in M , the dimensionality of the transformed signal can be reduced. This allows the data to be represented using less information.

EXPERIMENTAL PROCEDURE AND RESULTS

This section provides some results from the best compression algorithms in the previous chapter. The results include how much compression was achieved and how lossy the reconstructed data is. The monitoring hardware is the other important part of the data collection process. It must work to collect data accurately and continuously. Lab tests show that this is the case with the updated hardware setup and the upgraded software.

The data that was collected from the FuelCell Energy DFC300A fuel cell at Zoot Enterprises provides an excellent starting point for designing algorithms that will effectively compress data from similar fuel cell installations. Fig. 12 shows a few periods of the current and voltage data that was collected at Zoot Enterprises. This represents a small time window of the data that was used in all of the compression experiments. All of the data that was collected at that site is similar in nature. The voltage is a periodic waveform which is very similar to a sine wave. This makes sense because the power plant should be producing a sinusoidal 480 V rms output. The current waveform is not as clean, this is also to be expected because all of the loads in the building demand different levels of current, some being constant, and others time variant.

Laboratory Results of Updated Monitoring Hardware

The entire updated data monitoring system was assembled in the laboratory and tested on some common appliances such as a vacuum cleaner, a light bulb and a lathe. Data acquisition from tests in the lab show that the monitoring equipment works well and that the scaling is correct for higher power applications. The 120 V rms signal

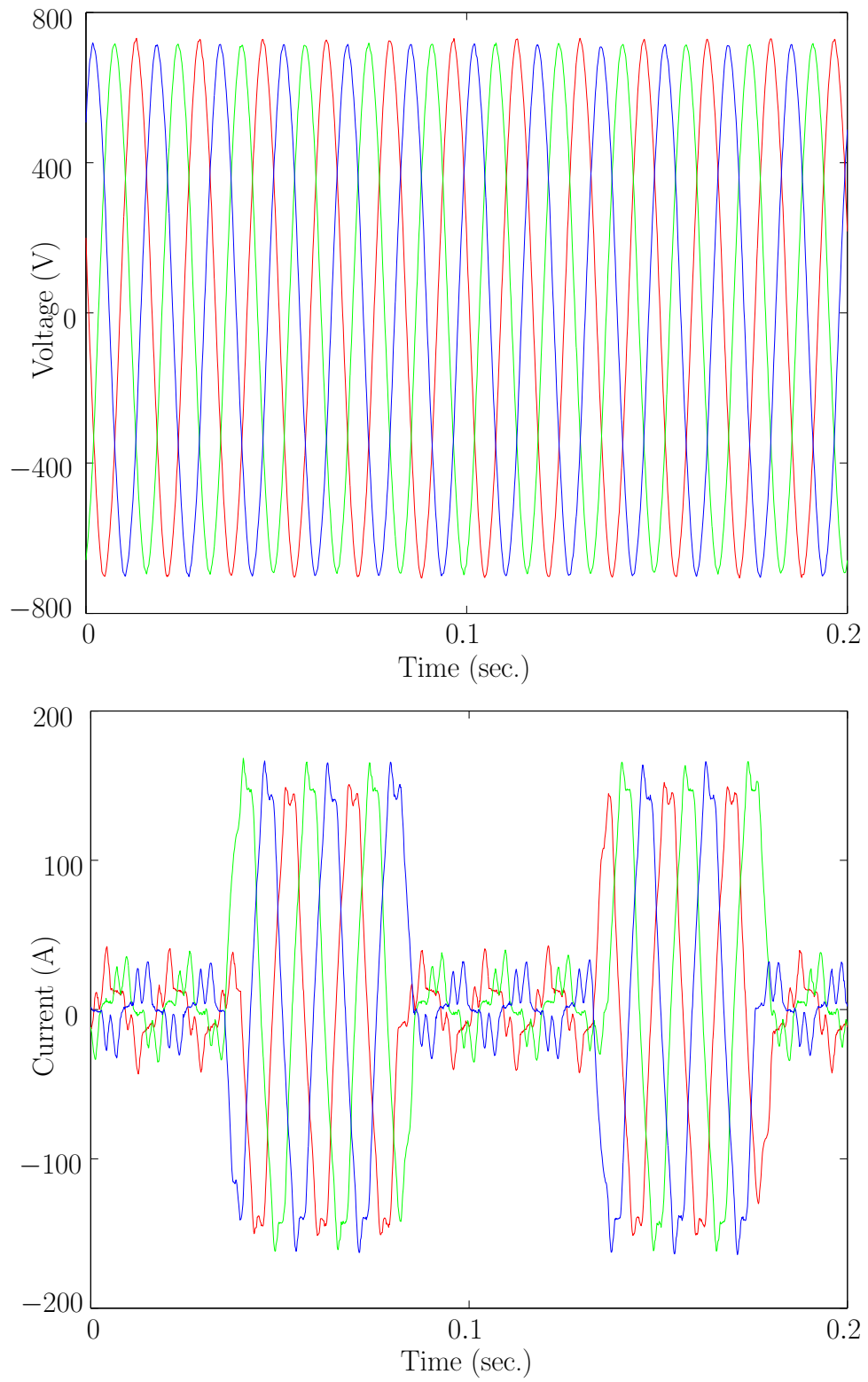


Figure 12: Voltage and current waveform measurements from the installation at Zoot Enterprises.

from the outlet occupied approximately one fifth of the available range meaning that the 480 V rms produced by the fuel cell will cover nearly all of the range of the A to D converter, leaving a small cushion to record any voltage events that may have values greater than 677 V. The current scaling was also correct with a -50 A to 50 A signal occupying approximately 1/8 of the available range, meaning that a 700 A current measurement would nearly fill the entire range of the ADC.

Fig. 13 shows the voltage and current waveforms from the startup transient of a Shop Vac vacuum cleaner. It is apparent that the initial current is much greater than that when the machine is at steady state. The voltage also sees a slight drop when the current spikes. The current from a 60 W incandescent light bulb can also be seen in the plot before the vacuum cleaner is started. When monitoring fuel cells, this type of correlation can be helpful in finding critical interactions between different loads. Both the voltage and current signals have a small amount of noise. This is the reason that it was not possible to achieve a compression ratio of 0.1 using lossless compression.

Results from Compression Programs

The three compression programs, Gzip, Bzip2, PPMd and Arithmetic coding were tested to determine which produced the best compression. Two different data sets were tested, both were from the fuel cell at Zoot Enterprises which included three phases of voltage and three phases of current measurements. The first data set appeared less noisy and was taken during normal operations at Zoot. The second was during the time when one of the large variable speed drives with no input filter was operating. Both of these data sets were compressed as raw data and also after being preprocessed using fifth order LPC.

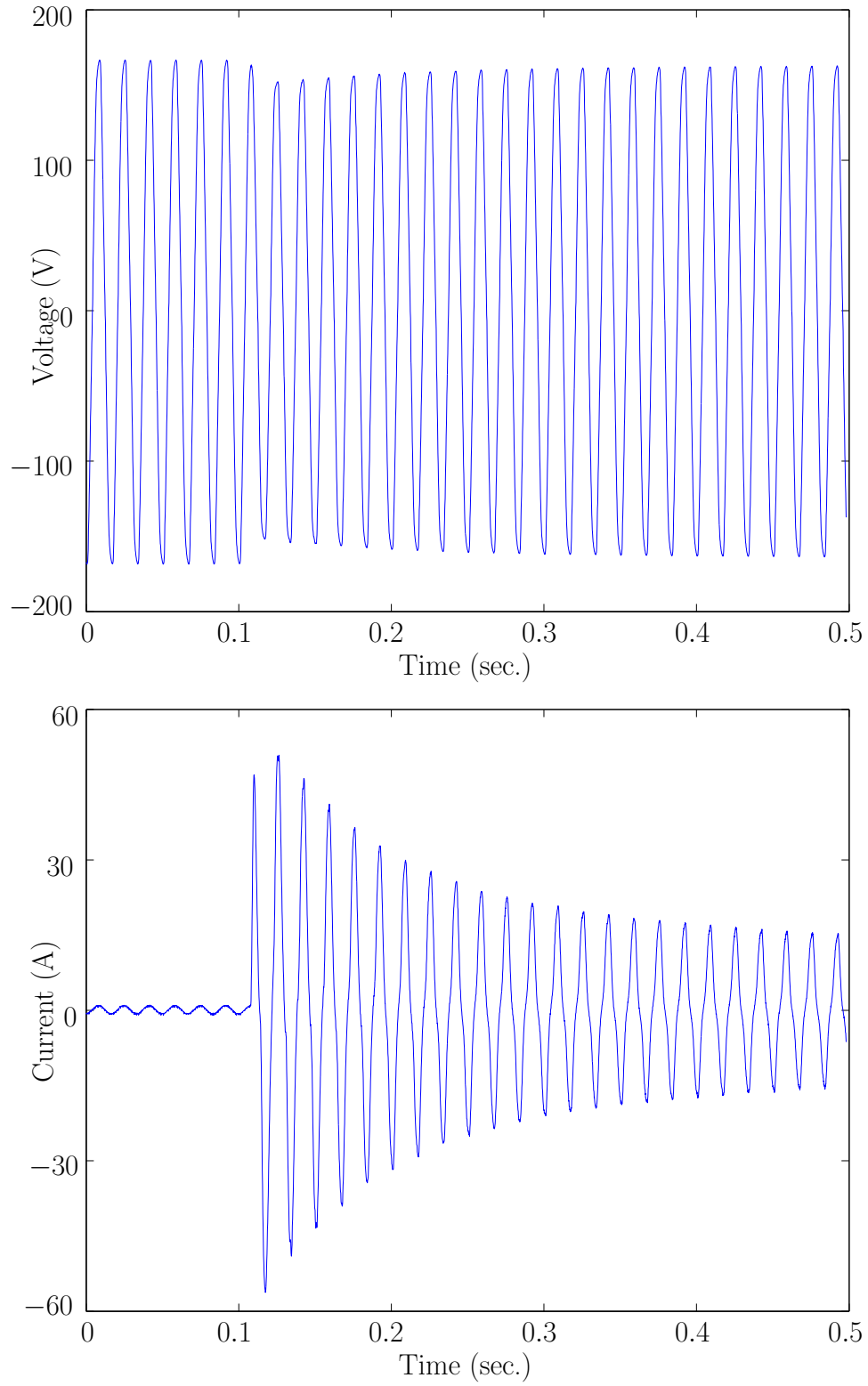


Figure 13: Voltage and current waveforms of the startup transient from a vacume cleaner recorded during testing of the updated monitoring installation in the laboratory.

Gzip was the worst of the three compressors tested, it realized a compression ratio of approximately 0.74 for raw data collected from the fuel cell and about 0.5 for data preprocessed using a fifth order linear predictor. These ratios are not close to the goal of a 0.1 compression ratio.

Bzip2 showed compression that was much improved from Gzip. It realized a compression ratio of about 0.43 for raw fuel cell data and 0.41 for the same data processed with LPC. Although this is better than Gzip it is still not acceptable for the desired application.

Although PPMd was the best of the programs tested, it still could not compress the data enough to achieve the desired size reduction. It had a compression ratio of about 0.43 for raw data from the fuel cell and 0.39 for data that had been preprocessed using a fifth order linear predictor. PPM is touted as the class of compressors with the best performance of any of the widely used algorithms [16]. Even though this particular version of PPM is optimized for text, changing the return probabilities to ones that match the data better could only improve the compression ratios slightly.

Of the four algorithms tested, arithmetic coding realized the best compression ratio for the preprocessed data. It achieved approximately a 0.69 ratio for the raw data, this is on par with the worst of the three previous examples. However it realized a 0.38 ratio for the data processed with a fifth order linear predictor. The large gain from one case to the next is a result of the algorithm's method of compression. In the first case the symbols were distributed across a wide range, therefore using arithmetic coding narrowed the code interval drastically with each step. This aggressive reduction in the range of the final interval lead to a much longer code. The second case had normally distributed variables with a much smaller variance than the first distribution. This allowed the final code interval to be narrowed much less as each

symbol was processed. The final code interval was therefore much wider leading to a shorter final code.

Table 3: Comparison of Compression Algorithms

Test data	Original size	gzip --best	bzip2 --best	PPMd e -o10 -m256	Arithmetic coding
xaa	24000000	17578243 0.732	9210115 0.384	9210774 0.384	16551888 0.690
xac	24000000	18116741 0.755	11353394 0.473	11263819 0.469	16285634 0.679
xaalp5	24000000	10880683 0.453	8976429 0.374	8412261 0.351	8205994 0.342
xaclp5	24000000	13205942 0.550	10821188 0.451	10334246 0.431	9918770 0.413

Table 3 shows a comparison of all the algorithms that were tested. The first two data sets are the raw data from the fuel cells while the next two are that same data after being preprocessed with a fifth order linear predictive coder, denoted with the lp5 suffix. Under each compressed file size is the compression ratio. It can be seen that arithmetic coding achieves the best compression on data that has been preprocessed while bzip2 and PPMd work better with the raw data.

Simulation Results from Lossless Preprocessors

When compressing data there are some metrics that can be used to determine how much a signal can be compressed using lossless compression. The first order entropy of the distribution gives a very good indication of how much the data in question can be compressed. It represents the greatest compression possible if entropy coding were the only technique used to compress the data. The distribution of symbols in the residual signal is also a good indication of how much the data set can be compressed. If nearly all of the structure has been removed from the data, the distribution should

be very similar to white noise. This means that the general entropy limit can be calculated using the variance of the residual, and that entropy coding will realize compression that is very close to that limit.

Linear Predictive Coding

When using LPC, the only variable parameter of the predictor is the order of the filter. As the order of the predictor increases for the fuel cell data the entropy of the residual decreases until the predictor order reaches 5. At this point the entropy stabilizes at about 6 bits/symbol and the residual sequence resembles white noise. Using the χ^2 test [15] to quantize the similarity between the residual sequence and white noise of the same variance provided validation that the current signals could be reduced to a residual that is statistically nearly identical to white noise with a confidence interval of 99% this means that the compression that can be realized is near optimal.

Table 4: Current and Voltage Compression

Test data	Original size	bzip2 --best	PPMd e -o10 -m256
xaa current phase 1	4000000	1456107 0.364	1360832 0.340
xaa voltage phase 1	4000000	1517726 0.379	1413610 0.353
xaa current phase 2	4000000	1530538 0.383	1441091 0.360
xaa voltage phase 2	4000000	1504167 0.376	1402132 0.351
xaa current phase 3	4000000	1466687 0.367	1377349 0.344
xaa voltage phase 3	4000000	1496976 0.374	1393880 0.348

The voltage still contains some structure after being filtered using LPC. Table 4 shows the difference between the compression ratios of each phase of the current and voltage signals. Because the current and voltage signals have the similar noise associated with them, it is safe to assume that the compression achieved for the current is comparable to the best possible compression for any of the voltage signals as well. This table illustrates that even if a perfect model for the underlying voltage signal were used as a predictor the compression gain over the LPC processed data would be minimal.

Other Lossless Predictors

Using the Luenberger observer with a plant that generated a sine wave resulted in the first order entropy being about 6.25 bits/symbol. That was slightly higher than the entropy of the LPC residual. This indicates that the compression ratio would not be as low as the LPC result, which proved true when the compression programs mentioned above were applied to the processed data. The compressed files were about 10% larger than those created using fifth order LPC.

The harmonic sum approach performed very well when the size of the data block was less than one period of the fundamental. When the block size was larger than that it became apparent that the model was not sufficient to capture differences between periods of a waveform which appeared as lower frequency content. The variance and entropy of the resulting residual from this model for single periods of the fundamental frequency was lower than that of LPC.

For each block of data that is compressed the parameters of the system must also be stored to enable reconstruction. When each block of data is a single period of the signal, or approximately 140 samples, the model must include 15 coefficients to allow for reasonable reconstruction. In this case the model represents approximately

10% of the uncompressed data. The entropy of the residual was around 5, meaning that the compression ratio of only the residual was near 3:1. This does not take into account the space that must be used to store the model.

qd0 Transform

Using the *qd0* transform in conjunction with LPC achieved the greatest compression using a lossless scheme. After converting the signals into the *qd0* reference frame then using LPC, the signal was compressed approximately 8% more than was previously possible. This is because much like the case where only LPC was used the variance of the residual sequence is statistically nearly identical to white noise, however, its variance is smaller than that of the LPC residual.

Although using the *qd0* transform in conjunction with LPC realized the greatest compression of all the lossless preprocessing schemes, the best ratio was still just under 0.3 with the residual sequence having a sample variance of about 60. Using the χ^2 and the Kolmogorov-Smirnov tests [15] to compare the residual sequence to a Gaussian distribution confirmed that the residual could not be distinguished from white noise, and that therefore nearly all of the structure had been extracted from the data.

Any truly random noise that exists in a signal can increase its entropy dramatically. Assuming that the symbols in the data set are represented in 16 bits initially (14 for the measurement and 2 control bits), the final entropy must be 1.6 bits/symbol to make the compression ratio 0.1. Calculating the noise variance [21] that will make it impossible to achieve this compression ratio can be done using (2).

$$H(x) = \log \sqrt{2\pi e \sigma}$$

$$\begin{aligned}
H(x) &= \log(\sqrt{2\pi e}) + \log(\sigma) \\
\log(\sigma) &= H(x) - \log(\sqrt{2\pi e}) \\
\sigma &= e^{H(x) - \log(\sqrt{2\pi e})}
\end{aligned} \tag{6}$$

Using (6) to find the maximum variance of Gaussian noise that can still be compressed to yield an entropy rate of 1.6 results in σ being approximately 1.2 and the variance, σ^2 , being 1.44. This variance is more than an order of magnitude smaller than even the best case residual from lossless compression which is about 60. This test indicates that it is impossible to achieve the desired compression ratio with lossless compression so lossy schemes must be considered.

Simulation Results from Lossy Compression

The lossy techniques that were tested yielded mixed results. An analysis of the trade off between loss and compression indicated how the algorithms compared against one another. Some of the programs performed poorly regarding both loss and compression, ruling them out as potential solutions, while others performed well enough to retain most of the data and still achieve a compression ratio of 0.1.

N-bit Loss

Using the simple N -bit lossy compression scheme only gained about 20% compression over its lossless counterparts, however, it did perform approximately as well as Shorten regarding compression and loss. When LPC was applied to both the original data and the mapped data, the distributions of the residuals were very similar. However, the variance of the mapped data was slightly smaller, leading the minimal increase in compression.

Frequency Domain Representation with Replacement

When using frequency domain results without replacing the high error data points to reduce the size of the signal, the first order entropy of the data recovered from only 10% of the frequency content was calculated to be 1.3 bits/symbol. This meant that the achievable compression ratio was calculated to be less than 0.1. In practice the aforementioned programs achieved even greater compression on the information by taking advantage of the repetitions within it. The compression ratio was nearly 0.05 which meant that the extra space could be used to correct for the data points in the signal with high error. For comparison with subsequent lossy schemes the variance of the error signal between the original data and reduced frequency data was on the order of 20 when using the data set from the fuel cell labeled xaa.

Adding error correction to the frequency domain representation proved to be the most effective of the lossless schemes. Because the data is almost exclusively periodic, the vast majority of signal energy is stored in a few frequencies. Using only 5% of the frequency content to reconstruct the signal and setting the error threshold to 7 provided a good compromise between loss and compression. The maximum loss was 3 bits of data out of 14 with a compression ratio very near 0.1. The average difference between the original signal and the recovered one was about 2.5 meaning that the average loss was just fewer than 2 bits.

Fig. 14 shows two periods of the original and recovered signal in the top graph and a larger time segment of the difference between the two signals below. The two signals are nearly identical as can be seen and the loss is minimal even where the original signal appears noisy. The error between the two is also relatively small. Its distribution is bell shaped except that there is a spike at zero and the tails are cut

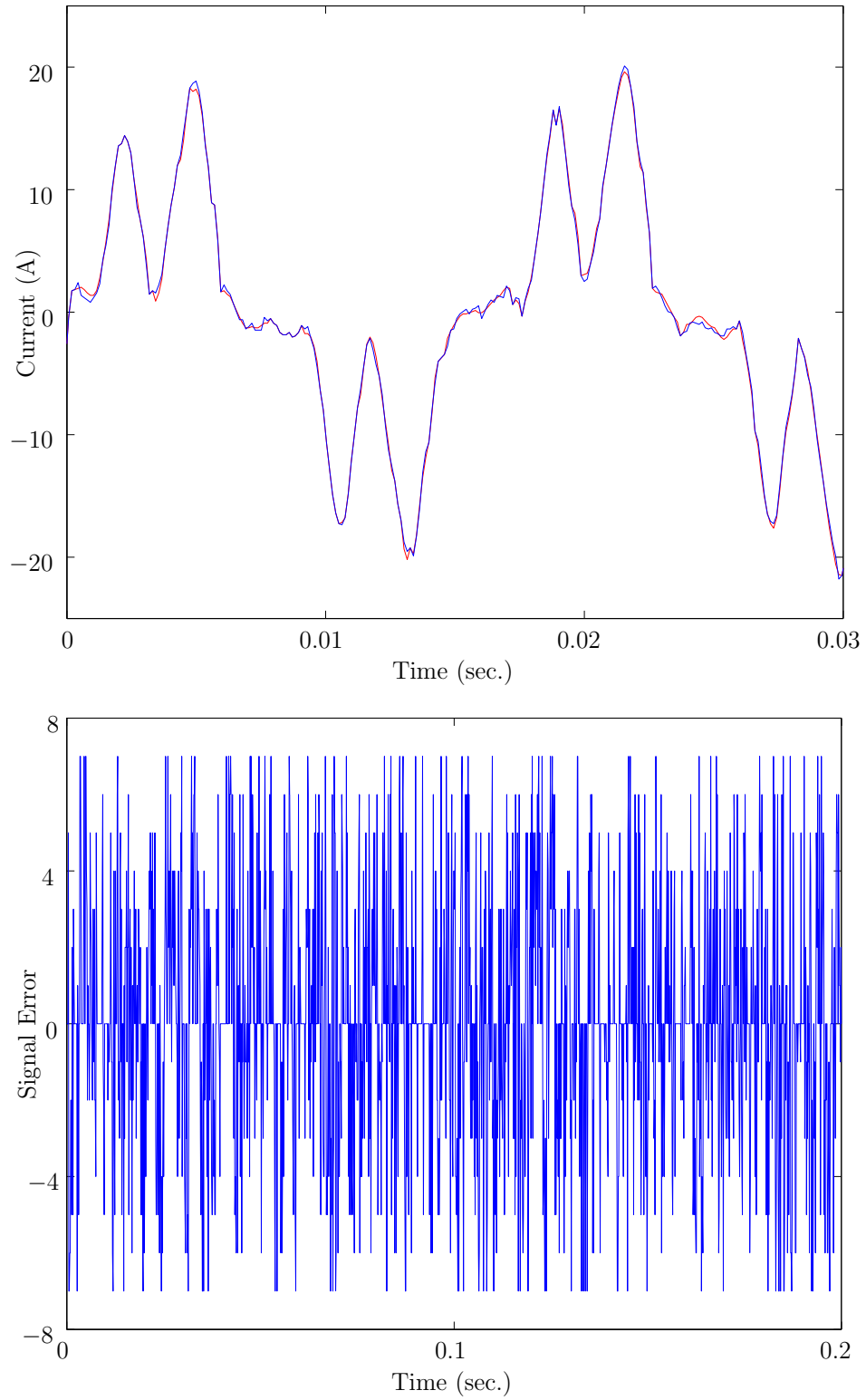


Figure 14: Current waveform (blue) compared to restored compressed current waveform (red) and the difference between a larger time segment of the same two signals.

off because all of the larger error data points have been eliminated and replaced with the correct value.

Wavelet Transform Reduction

Applying the DWT using Daubechies wavelets order $M = 8$ to the signal xaa proved the best of the two wavelet transforms tested. The signal was reconstructed using only the 10% of the wavelet coefficients with the largest amplitude, but the resulting reconstruction had more error than the corresponding DCT signal. The variance of the error was over 300 meaning that DWT represents this signal with much less efficiency than the DCT. Similar results were seen when testing on other data sets from the fuel cell. This rules out DWT reduction as a possible solution.

Principal Component Analysis

By using PCA on the fuel cell data, a reduction of one dimension for each of the three phase current and voltage signals was possible. This meant that each of the three phase signals could be stored using only two vectors that were the same length as the original three. This only reduced the data size by 0.33 while the loss from reducing the dimensionality of the signal was significant. The variance of the error ranged between 50 and 200 for the different signals. This was much worse than the DCT case with regards to the compression that was achievable and the information loss.

CONCLUSION

Installation of monitoring systems such as the one described in this thesis should be considered during the testing stage of new fuel cell technologies as well as at new and existing fuel cell sites. The information that is collected from fuel cells working in different loading environments could help indicate where fuel cells are most likely to be successfully deployed. It could also provide insight into improvements that could be made to fuel cells that could make them a viable alternative in a wider variety of applications.

A basic observation of this thesis is that the complexity of fuel cell/load interactions seen at Zoot Enterprises, in other words multiple loads interacting through the fuel cell, is not applicable to the NILM. Until a common set of fuel cell/load interactions are identified, it may be that the best approach is to store data until it can be analyzed offline when failure is known to have occurred.

Future work for this project would be to install the monitoring system at the Fuel-Cell Energy location in Billings, MT. The data collected at the Billings location may augment the understanding of fuel cell/load interactions gained at Zoot Enterprises.

REFERENCES

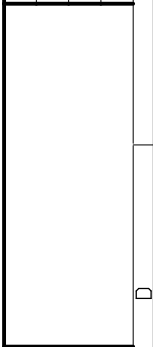
- [1] ACHARYA, K., MAZUMDER, S. K., and BURRA, P. K., “System interaction analysis of solid oxide fuel cell (sofc) power conditioning system,” in *IEEE Industry Applications Conference, 38th Annual Meeting*, pp. 2026–2032, October 2003.
- [2] BURROWS, M. and D.J. WHEELER, “A block-sorting lossless data compression algorithm,” Tech. Rep. 124, Digital Systems Research Center, Palo Alto, CA, USA, May 1994.
- [3] CLEARY, J. G. and WITTEN, I. H., “Data compression using adaptive coding and partial string matching,” *IEEE Transactions on Communications*, vol. COM-32, pp. 396–402, April 1984.
- [4] DATASHEET, “Current transducer lt 505-s.” <http://www.lemusa.com>, April 2006.
- [5] DATASHEET, “Voltage transducer lv 25-p.” <http://www.lemusa.com>, April 2006.
- [6] EG&G TECHNICAL SERVICES, INC., “Fuel cell handbook.” U. S. Dept. of Energy, November 2002.
- [7] GAILLY, J.-L., “Gzip documentation.” <http://www.gzip.org>, September 1993.
- [8] GEHEZEL-AYAGH, H., DALY, J. M., and WANG, Z.-H., “Advances in direct fuel cell/gas turbine power plants,” in *Turbo Expo 2003*, (Atlanta, GA, USA), ASME/IGTI, June 2003.
- [9] GEMMEN, R. S., “Analysis for the effect of inverter ripple current on fuel cell operating condition,” *Transactions of the ASME, Journal of Fluids Engineering*, vol. 125, pp. 576–585, May 2003.
- [10] GERSHENFELD, N., *The Nature of Mathematical Modeling*. Cambridge University Press, 1999.
- [11] KRAUSE, P. C., *Analysis of Electric Machinery*. McGraw-Hill Book Company, 1986.
- [12] LEEB, S. B. and KIRTLEY, J. L., “A multiscale transient event detector for nonintrusive load monitoring,” in *Proceedings of the 1993 IECON International Conference on Industrial Electronics, Control, and Instrumentation*, 93CH3234-2, pp. 354–359.

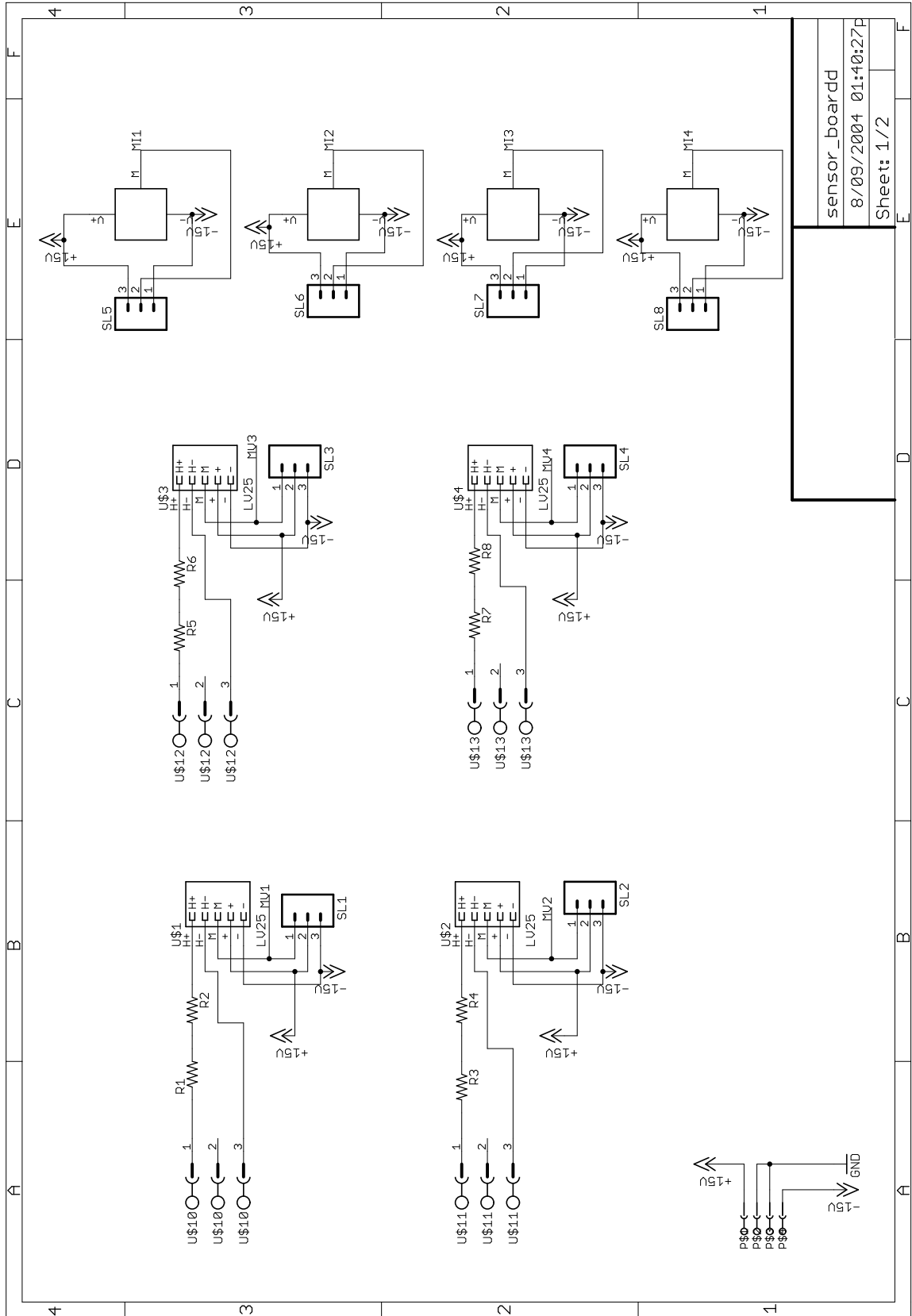
- [13] LEEB, S. B., SHAW, S. R., and KIRTLEY, J. L., “Transient event detection in spectral envelope estimates for nonintrusive load monitoring,” *IEEE Transactions on Power Delivery*, vol. 7, pp. 1200–1210, July 1995.
- [14] LJUNG, L., *System Identification: Theory for the User*. Prentice-Hall, second ed., January 1999.
- [15] PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W., and FLANNERY, B., *Numerical Recipes in C*. Cambridge University Press, second ed., 1992.
- [16] SALOMON, D., *Data Compression, the Complete Reference*. New York: Springer-Verlag New York, Inc., 3rd ed., 2004.
- [17] SANTINA, M., ALLEN STUBBERUD, and HOSTETTER, G., *Digital Control System Design*. Saunders College Publishing, 2nd ed., 1994.
- [18] SAYOOD, K., *Lossless Compression Handbook*. Academic Press, 2003.
- [19] SEWARD, J., “Bzip2 documentation.” <http://www.bzip.org>, February 2005.
- [20] SEWARD, J., “Performance of the BWT sorting algorithms,” in *Proceedings of the Data Compression Conference 2000*, pp. 173–182, March 2000.
- [21] SHANNON, C. E., “A mathematical theory of communication,” *The Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, July, October 1948.
- [22] SHAW, S. R., *System Identification Techniques and Modeling for Non-intrusive Load Diagnostics*. Ph.d., MIT, February 2000.
- [23] SHKARIN, D., “PPM: one step to practicality,” in *Proceedings of the Data Compression Conference*, April 2001.
- [24] SHKARIN, D., “PPMd documentation.” <http://packages.debian.org/unstable/utils/ppmd>, April 2002.
- [25] WITTEN, I. H., NEAL, R. M., and CLEARY, J. G., “Arithmetic coding for data compression,” *Communications of the ACM*, vol. 30, pp. 520–540, June 1987.
- [26] ZIV, J. and LEMPEL, A., “Universal algorithm for sequential data compression,” *IEEE Transactions on Information Theory*, vol. IT-23, pp. 337–343, May 1977.

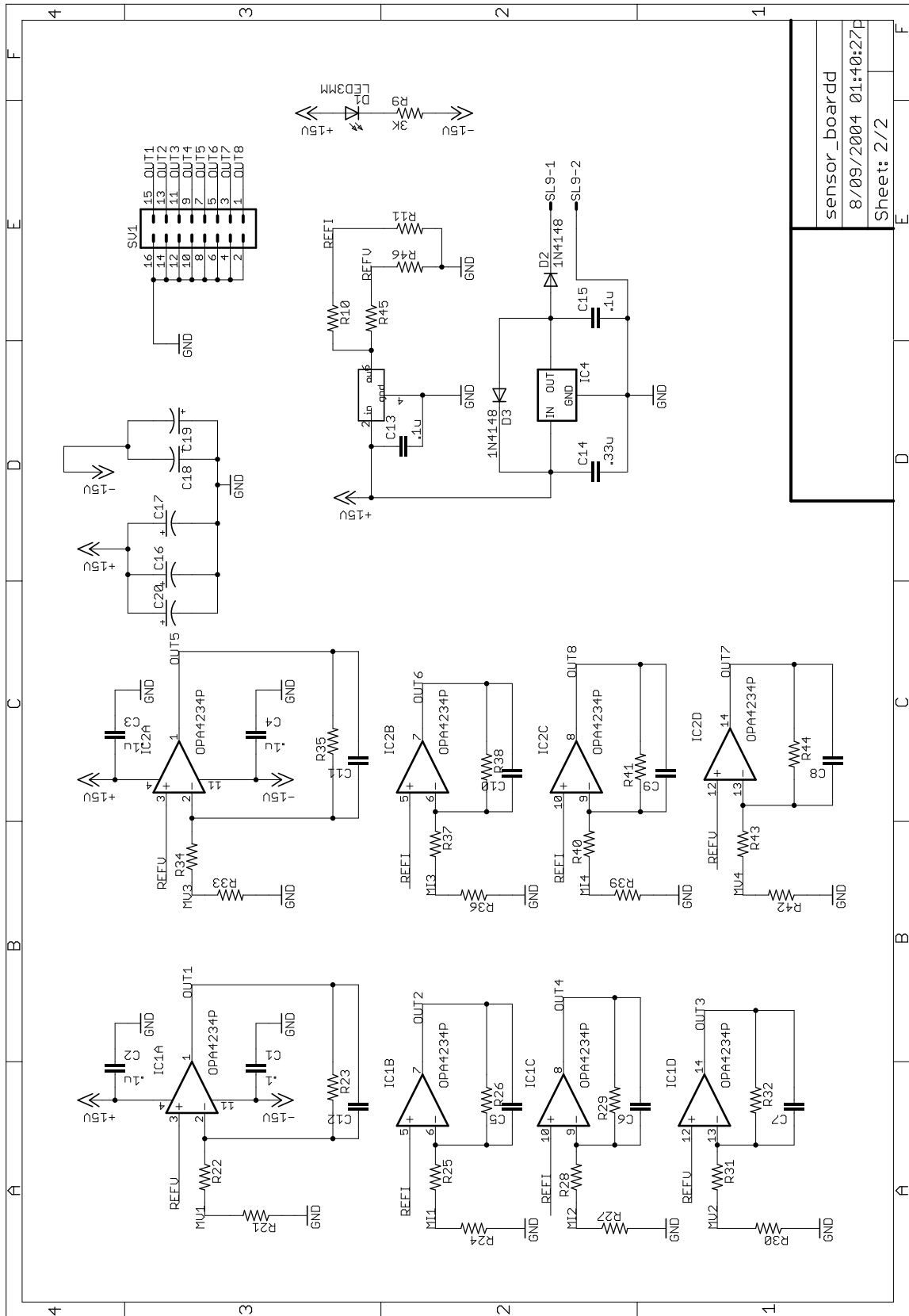
APPENDICES

APPENDIX A

PCB SCHEMATICS







sensor_boardd

8/09/2004 01:40:27P

Sheet: 2/2

APPENDIX B

FIRMWARE FOR SX28AC ON THE SAMPLING/DATA TRANSFER BOARD

```

; Code for USB interface Analog to Digital Converter
; S. R. Shaw -- sshaw@alum.mit.edu
;
; Last revised: March 22, 2003
;
; Added multiple channel capabilities.
;
;
;
device SX28L,oscxt5,turbo,stackx_optionx
id 'SX USB'
reset reset_entry
freq 48_000_000

;
; Set the number of channels and frequency here.
;
; NUM_CHANNELS is a literal between 1 and 8
;
; SAMPLE_DIV is a divisor that determines the net sampling frequency across
; all channels, i.e. the sampling frequency per channel is the
; net frequency divided by the number of channels
;
; The net sampling frequency is 200 kHz / div
;
; example: we want a 2kHz sampling frequency for 7 channels.
; SAMPLE_DIV = 14 corresponds to a net sampling frequency of 14.285 kHz
; The sampling rate per channel is therefore 2.040 kHz
;

NUM_CHANNELS = 6 ; Scan channels 1 through 6
SAMPLE_DIV = 4 ; Divisor for 8.33 kHz per channel

;
; Equates
;
usb_rxf = ra.3 ; input
usb_txe = ra.2 ; input
usb_wr = ra.1 ; output
usb_rd = ra.0 ; output
usb_data = rc ; port b 07?
usb_sleep = rb.6

adc_sclk = rb.0 ; communications clock for AD7856 (SX output)
adc_dout = rb.1 ; adc data out line (SX input)
adc_din = rb.2 ; adc data in line (SX output)
adc_sync = rb.3 ; sync for serial comm (SX output)
adc_busy = rb.4 ; adc busy status line (SX input)
adc_cnvst = rb.5 ; adc conversion start (SX output)

```

```
LED = rb.7
```

```
;
; Variables
;
org 8h
```

```
ADC_lo ds 1 ; data to/from the ADC
ADC_hi ds 1
```

```
ADC_ch ds 1 ; Channel code
```

```
byte ds 1 ; used
```

```
string ds 1 ; used
```

```
temp ds 1
```

```
irq_count ds 1 ; used by the interrupt routine
```

```
irq_stat ds 1 ; lock out.
```

```
org 0
```

```
;
;
;; org 10h ;bank0 variables
;
;;extra = $
;
; org 30h ; bank 1 variables (used as scratch by comparators)
; org 50h ;bank2 variables
;
org 0
```

```
; interrupt routine
;
; interrupt every 240 clocks, i.e. 200000 Hz
;
; This routine requires a maximum of 10 clocks, minimum of 8 clocks
;
; This is where all sample rates are set.
interrupt
djnz irq_count,end: ;2/4 execute every irq_div times
mov irq_count,#SAMPLE_DIV ;2 clocks

test irq_stat
snz
clrb adc_cnvst
```

```

inc irq_stat

end: mov w,#-240 ;1   interrupt every 240 clocks
retiw ;3 return

;
;
; Data
;
_start dw 'start',0
_stop dw 'stop',0
_cr dw 13,0
_hex dw '0123456789ABCDEF'

;
;
;*****
;* Subroutines *
;*****
;

;
; send_byte(byte), !(direction of RC)
;
; transmits the contents of byte via USB
;
send_byte
mov w,irq_stat
and w,#%11111110 ; is irq_stat bigger than 1?  If so, we've overflowed.
sz ; skip if irq_stat <= 1
clrb LED ; turn off LED indicating problems.
snb usb_txe ;
jmp send_byte

mode $0F ;change mode to write
mov !rc,#%00000000 ;port c

setb usb_wr ; hold usb_wr high
nop
mov w,byte
mov rc,w
nop
nop
clrb usb_wr
ret

get_byte

```

```
snb usb_rxf ; wait for data to be available from USB
jmp get_byte ; not yet ...
```

```
mode $0F
mov !rc, #%11111111 ; change mode to read.
```

```
nop ; 20 ns
clrb usb_rd ; set RD# low
nop
nop ; 40 ns
mov w, rc
mov byte, w
setb usb_rd ; bring RD# high
ret
```

```
; this routine scans the input to match a string
; pointed to by w
match_string
mov string, w
:here mov w, string
mov m, #0
iread
mov m, #$F
test w
snz
ret ; return with z set, string is matched
mov temp, w
inc string
call get_byte
xor temp, byte
jz :here
ret ; return w/o z set, string fails to match
```

```
; Send string at w
send_string
mov string, w ; send string at w
:loop mov w, string ; read chr at w;
mov m, #0
iread
mov m, #$F
test w ; if 0, exit
snz
ret
mov byte, w
call send_byte ; not 0, send chr
inc string ; next chr
jmp :loop
```

```

; read/write from the ADC
;
; This whole routine takes on the order of 16*280 ns = 4.5 us
;
; Control data in [ADC_hi ADC_lo] is written to the device. On
; exit, ADC data from the conversion is in these same registers.
;

adc_io
clrb adc_sync ; clear ADC sync line
nop
nop ; Must provide setup for falling edge
nop

mov temp,#$08
here: rl ADC_hi ; put MSB of control word in carry
sc ; skip if carry
clrb adc_din ; MSB is zero, clear ADC data
clrb adc_sclk ; SCLK LOW (delta = 7 cycles)
snc ; skip if not carry
setb adc_din ; MSB is one, set ADC data (*)
nop ; pad so data access is 80ns after SCLK
stc ; set carry
sb adc_dout ; if data from ADC is 1, skip
clc ; input is zero, clear carry
setb adc_sclk ; SCLK HIGH (delta = 6 cycles)
djnz temp,here: ; 280 ns loop total = 3.6 Mhz

rl ADC_hi ; roll data into ADC_hi
setb temp.3 ; put 8 in count

there: rl ADC_lo ; put MSB of control word in carry
sc ; skip if carry
clrb adc_din ; MSB is zero, clear ADC data
clrb adc_sclk ; SCLK LOW (delta = 7 cycles)
snc ; skip if not carry
setb adc_din ; MSB is one, set ADC data
nop ; pad so data access is 80ns after SCLK
stc ; set carry
sb adc_dout ; if data from ADC is 1, skip
clc ; input is zero, clear carry

setb adc_sclk ; SCLK HIGH (delta = 6 cycles)
djnz temp,there: ;

rl ADC_lo ; roll data into ax
setb adc_sync ;
ret ;

```

```

; send_hex(byte), !temp
; Send hex byte (2 digits) in arg
;
; uses:    for:
;-----
; arg      argument
; temp     internal storage
;
send_hex mov temp,byte
mov w,<>temp ;send first digit
call :digit

mov w,temp ;send second digit

:digit and w,$F ;read hex chr
mov byte,w
mov w,$_hex
clc
add w,byte
mov m,#0
iread
mov m,$F
mov byte,w
call send_byte
ret

;
; Main program
;
reset_entry

mode      $0F
mov ra, #0011 ; set USB_WR and USB_RD default high
mov !ra,#1100 ; set ra.0 and ra.1 as outputs

mov rc, #00000000 ; initial state all low
mov !rc,#11111111 ; set as inputs initially

mov rb, #00101101 ; set initial state to avoid glitch
mov !rb,#01010010 ; set inputs/outputs

clr fsr ; clear memory
loop: setb fsr.4
clr ind
ijnz fsr,loop:

setb LED ; turn LED on before calibration

:busy jb adc_busy, :busy ; wait for ADC calibration
mov ADC_hi, #11100000 ; select control reg, single-ended, ch0, no power down

```

```

mov ADC_lo, #%00000000
clr ADC_ch
call adc_io ; put the AD7856 in some reasonable initial state

clrb LED ; turn LED off after calibration
mov !option, #%10011111 ; do real-time interrupt

; wait for USB setup here.
setup: ; Wait for setup information from the host
mov w, #_start ; look for this pattern to begin emitting data
call match_string ; try to match string
jnz setup: ; if z not set, string fails to match

setb LED ; turn the light on indicating that we're good so far
clr irq_stat ; clear the pending conversion, wait for new interrupt

:sync test irq_stat
jz :sync

mov ADC_hi, #%11100000
movb ADC_hi.4, ADC_ch.0
movb ADC_hi.3, ADC_ch.2
movb ADC_hi.2, ADC_ch.1

mov ADC_lo, #%00000000

setb adc_cnvst ; bring CNVST high (minimum 100ns pulse width)

:busy jb adc_busy, :busy ; Wait for ADC to finish conversion

call adc_io ; get the data, setup for next conversion

; We've got the data now, make sure we're not overlapping.

dec irq_stat
jz :skip
clrb LED ; turn LED off to indicate overflow.
jmp reset_entry
clr irq_stat
:skip

; channel scan happens here

test ADC_ch
snz
setb ADC_hi.7 ; presently converting ch0, so data being sent
; is highest numbered channel. Mark.

```

```
inc ADC_ch ; increment

stc ; get ready for CSB
csb ADC_ch, #NUM_CHANNELS ; if below NUM_CHANNELS == 1..8, skip.
clr ADC_ch

; Send out low ADC byte first, then hi
mov byte,ADC_lo
call send_byte
mov byte,ADC_hi
call send_byte

jmp :sync

;extra code page
;ORG $400
```

APPENDIX C

MATLAB COMPRESSION CODE

Linear Predictive Coder

```

function [fy,h] = mylpc(y,n)
ind = (n+1):length(y);
ind = ind(:);
A=zeros(length(ind),n);
for i=1:n
    y1 = y(ind-i);
    y1 = y1(:);
    A(:,i)=y1;
end
y1 = y(ind);
y1=y1(:);
h=A\y1;
fy = y1-A*h;

```

Luenberger Observer

```

clear i theta X;
input = xaa(1:100000,2);
%hold off;
n = length(input);
y = input;
samplerate = 8333+1/3; %samples/sec
period = samplerate/60; %samples/period
j=(1:n)';

subplot(211);
plot(y);

% estimate values (lpc, kalman, etc...) then find residual sequence

u = 0:n-1;
Ts = 1/period;

% state space variables in observable form
A = [2*cos(2*pi*Ts) 1; -1 0];
B = [sin(2*pi*Ts)*(2*cos(2*pi*Ts)-2); 0];
C = [1 0];
D = sin(2*pi*Ts);

x0 = [0; 0];

% observer stuff
G = [2*cos(2*pi*Ts); -1];

```

```

F = A-G*C;
H = B-G*D;

z0 = [0;0];

e = zeros(n,1);
e(1) = y(1);

% estimate values using observer then find residual sequence
for i = 1:n-1;
    % predict the state at time i+1
    z1 = F*z0+G*y(i)+H*u(i);

    % predict next value from predicted state
    yhat = C*z1+D*u(i);

    % calculate residual value
    e(i+1) = y(i+1)-yhat;

    % update the observation
    z0 = z1;
end
subplot(212);
plot(e(3:length(e)));
writefile(e, 'residual');
grid = -2^15:2^15;
entropy(e,grid)

% recover transformed from residual
z0 = [0;0];
yr = zeros(n,1);
yr(1) = e(1);

for i = 1:n-1
    z1 = F*z0+G*yr(i)+H*u(i);
    yhat = C*z1+D*u(i);
    yr(i+1)= e(i+1)+yhat;
    z0 = z1;
end

```

dq0 Transfrom

```

function fqd0s = abc2qd0(fabcs)
sizefabcs = size(fabcs);
n = sizefabcs(1);
fabcs1 = [fabcs(1:n,1) fabcs(1:n,3) fabcs(1:n,5)];
fabcs2 = [fabcs(1:n,2) fabcs(1:n,4) fabcs(1:n,6)];

```

```

% this stuff could be better perhaps, now it is just hard coded maybe a
% levenburg-marquart ls fit to a sine wave or something like that to fit
% the frequency
% this is the sample rate
samplerate = 8333+1/3; %samples/sec
% this is the period
period = samplerate/60; %samples/period
j=(1:n)';
% this is the initial position of theta in the first signal
theta0 = mod(2*pi*24/period+2*pi*j/period,2*pi*ones(n,1));
theta = 2*pi*(1/period)-theta0;

fqd0s1=zeros(n,3);
fqd0s2=zeros(n,3);

% transform variables to stationary reference frame
for i = 1:n
    Ks = 2/3*[cos(theta(i)) cos(theta(i)-2*pi/3) cos(theta(i)+2*pi/3);
              sin(theta(i)) sin(theta(i)-2*pi/3) sin(theta(i)+2*pi/3);
              1/2 1/2 1/2];
    fqd0s1(i,:) = fabcs1(i,:)*Ks';
    fqd0s2(i,:) = fabcs2(i,:)*Ks';
end

fqd0s = [fqd0s1 fqd0s2];

```

qd0 Recover

```

function fabcs = qd02abc(fqd0s)
sizefqd0s = size(fqd0s);
n = sizefqd0s(1);

fqd0s1 = fqd0s(:,1:3);
fqd0s2 = fqd0s(:,4:6);

% this stuff could be better perhaps, now it is just hard coded
% this is the sample rate
samplerate = 8333+1/3; %samples/sec
% this is the period
period = samplerate/60; %samples/period
j=(1:n)';
% this is the initial position of theta in the first signal
theta0 = mod(2*pi*24/period+2*pi*j/period,2*pi*ones(n,1));

```

```

theta = 2*pi*(1/period)-theta0;

for i = 1:n
    KsT = [cos(theta(i)) sin(theta(i)) 1;
           cos(theta(i)-2*pi/3) sin(theta(i)-2*pi/3) 1;
           cos(theta(i)+2*pi/3) sin(theta(i)+2*pi/3) 1];
    fabcs1(i,1:3) = fqd0s1(i,:)*KsT';
    fabcs2(i,1:3) = fqd0s2(i,:)*KsT';
end

fabcs = [fabcs1 fabcs2];

```

N-bit Loss

```

x = xaa(1:100000,2);
bitloss = 4;
maxx = max(x);
minx = min(x);
grid = 0:2^14;
dist = hist(x,grid);
bar(dist)
figure
plot(x)
for i = 1:length(x);
    if (x(i)<=maxx-bitloss && x(i)>=minx+bitloss)
        [y,k] = max(dist(x(i)-bitloss:x(i)+bitloss));
        x(i)=x(i)-bitloss-1+k;
    end
end
grid2 = -2^14:2^14;
[fy, h]= mylpc(x,5);
entropy(fy,grid2)

```

Frequency Domain with Repalcement

```

percent = .05;
loss = 7;

% data must bs kept to 2^15 points so the storage of indexes is still 16 bits
x = xaa(1:25000*1,6);
writefile16(x,'x');

% do DCT on data
X =dct(x);

```

```

% sort freq content by magnitude
[Y,I] = sort(abs(X));
I = flipud(I);
Y = X(I);

% remove all but largest x% of frequency content
reduce = round(percent*length(X));
Y = [Y(1:reduce); zeros(length(X)-reduce,1)];
Y(I) = Y;

% Now to find the restored data that is too far from the original signal
% and save the difference for better reconstruction
writefile32(Y,'Y');
Y = readfile32('Y');
xr = round(idct(Y));
temp = x-xr;
[dif,Idif] = sort(abs(temp));
Idif = flipud(Idif);
dif = temp(Idif);
total=sum(abs(dif)>loss);
dif = dif(1:total);
Idif = Idif(1:total);

% sort temporally
[Idif,Itemp]=sort(Idif);
dif=dif(Itemp);

% do first diff on index vector
Idwrite = [Idif(1);diff(Idif)];

% Write data to file
writefile32(Y,'Y');
writefile16(dif,'dif');
writefile32(Idwrite,'Idif');

```

Frequency Domain with Replacement Recovery

```

YR = readfile32('Y');
dif = readfile16('dif');
Idread = readfile32('Idif');
for(i=2:length(Idread)) Idif(i)=Idif(i-1)+Idread(i); end;

% reconstruct signal without the smaller magnitude freq content
xr = round(idct(YR));
xr(Idif)=xr(Idif)+dif;

```

Discrete Wavelet Transform

```

function W = mydwt8(x);
if mod(length(x),2)
    x = x(1:length(x)-1);
end

C = [0.32580343 1.01094572 0.8922014 -0.03967503
     -0.26450717 0.0436163 0.0465036 -0.01498699];
B = [C(8) -C(7) C(6) -C(5) C(4) -C(3) C(2) -C(1)];

T = zeros(length(x), length(x)+6);
W = x;
ind = (1:(length(x)/2))*2-1;
% create initial transform matrix
for i = ind
    T(i,i:i+7)=C;
    T(i+1,i:i+7)=B;
end
T = T(1:length(x),1:length(x));
wrap = [C(7:8) 0 0 0 0; B(7:8) 0 0 0 0; C(5:8) 0 0;
        B(5:8) 0 0; C(3:8); B(3:8)];

ind = fliplr(2.^(3:log2(length(x))));

for i = ind
    % resize transform matrix to multiply by detail coefficients
    T = T(1:i, 1:i);
    T(i-5:i,1:6) = T(i-5:i,1:6)+wrap;
    Wx = T*W(1:i);

    % create reorder matrix to seperate parts
    xind = (1:i/2)*2-1;
    Wind = (1:i/2)*2;
    ind = [xind Wind];
    W(1:i,1) = Wx(ind);
end

```

Inverse Discrete Wavelet Transform

```

function x = myidwt8(W);
W = W(:);
if mod(length(W),2)
    W = W(1:length(W)-1);
end

```

```

C = [0.32580343 1.01094572 0.8922014 -0.03967503
      -0.26450717 0.0436163 0.0465036 -0.01498699];
B = [C(8) -C(7) C(6) -C(5) C(4) -C(3) C(2) -C(1)];

T = zeros(length(W), length(W)+6);
x = W;
ind = (1:(length(W)/2))*2-1;
% create initial transform matrix
for i = ind
    T(i,i:i+7)=C;
    T(i+1,i:i+7)=B;
end
T = T(1:length(W),1:length(W));
wrap = [C(7:8) 0 0 0 0; B(7:8) 0 0 0 0; C(5:8) 0 0;
        B(5:8) 0 0; C(3:8); B(3:8)];

ind = 2.^(3:log2(length(W)));

for i = ind
    % create reorder matrix to seperate parts
    xind = (1:i/2)*2-1;
    Wind = (1:i/2)*2;
    Rind = [xind Wind];
    Wx(Rind,1) = x(1:i);

    % resize transform matrix to multiply by detail coefficients
    T2 = T(1:i, 1:i);
    T2(i-5:i,1:6) = T2(i-5:i,1:6)+wrap;
    x(1:i) = T2^-1*Wx;
end

```

Principal Component Analysis

```

x = xaa(1:2000,:);
Cx = cov(x);
[M D] = eig(Cx);
M = M(:,3:6);
y = M'*x';

```