Hardware and Software Architecture for Non-Contact, Non-Intrusive Load Monitoring

David Lawrence
B. S., Massachusetts Institute of Technology (2014)

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

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology.

February 2016

Author _____
David M. Lawrence
Department of Electrical Engineering and Computer Science

Thesis supervisor _____
Steven B. Leeb
Professor of Electrical Engineering and Computer Science

Associate supervisor _____
John S. Donnal
Doctoral Candidate in Electrical Engineering and Computer Science

Accepted by _____
Christopher J. Terman
Chair, Masters of Engineering Thesis Committee

Hardware and Software Architecture for Non-Contact, Non-Intrusive Load Monitoring

David Lawrence
B. S., Massachusetts Institute of Technology (2014)

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

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology.

Submitted January 19, 2016.

Abstract

The non-intrusive load monitor measures voltage and current in a single power line that supplies electricity to multiple devices. Signal processing algorithms are then used to separate the summed power usage into the individual power usage of each device.

This thesis presents three new developments: a hardware architecture that measures the electromagnetic fields surrounding a multi-conductor cable, calibration methods to non-invasively determine a mapping from the sensed fields to power transmission, and a software framework that enables efficient real-time processing of data gathered by the new electromagnetic field sensors.

Thesis supervisor:
Steven B. Leeb
Professor of Electrical Engineering and Computer Science

Contents

List of Figures

Chapter 1

Introduction

A non-intrusive load monitor (NILM) is a device which measures voltage and current in a single power line that supplies electricity to multiple devices. Signal processing algorithms are applied to obtain separate estimates of the power used by each device at any instant, a process known as disaggregation. The low hardware complexity and low cost of installation make NILMs suitable for energy monitoring applications in which the cost of installing a separate voltmeter and ammeter for each device would be prohibitive.

The concept of a NILM was first proposed by Hart [1] in 1982. Hart noted that a new photovoltaic system collecting power metering data at 5 second intervals provided much richer data than the 15 minute intervals in common usage by power companies. The connected devices were modeled as finite state machines undergoing step changes in power usage and a modified Viterbi algorithm was used to perform disaggregation.

Reference [2] noted that with a sample interval of 1 second, meaningful information could be drawn not only from step changes in power usage but also from the precise shape of the transient event between the beginning and end of each step. The transient event provides richer information about the physical characteristics of the device generating the step and offers an additional ability to differentiate between devices with the same nominal power usage.

With the advent of inexpensive integrated microcontrollers, temporal resolution has increased even further. Modern NILMs sample at several kilohertz and are able to distinguish individual line cycles of AC signals. A NILM can measure not only the real and reactive components of each device's power usage, but also the higher harmonic currents drawn by nonlinear devices throughout the course of a single line cycle [3]. This additional information is useful for disaggregation and for the subsequent monitoring applications.

Rate and resolution improvements have considerably increased the bandwidth and storage requirements for load monitoring systems. High-speed data must be acquired and processed in real time, yet many years worth of data must be retained for analysis. To solve this problem, reference [4] introduces a comprehensive system for data storage and analytics in energy monitoring applications. The system, called "NilmDB", integrates a network-enabled database with a signal processing toolchain and is a designed to run on a typical GNU/Linux software distribution.

Recent research has focused on NILMs that use non-contact electromagnetic field sensors in place of a traditional voltmeter and ammeter [5]. These sensors do not require Ohmic

contact with any of the conductors, instead deducing the voltages and currents from the radiated electromagnetic fields. Non-contact sensors offer ease of installation and robust isolation in exchange for higher complexity. There is a considerable signal processing challenge in recovering the voltages and currents from the sensed fields.

1.1 Non-contact sensing

An electric current flowing through a conductor produces a magnetic field whose magnitude at any point is proportional to the current. Similarly, a voltage applied to a conductor produces an electric field whose magnitude at any point is proportional to the voltage. The voltage and current in a conductor may thus be determined by electric and magnetic field sensors placed nearby [6, 7]. The obvious appeal of this technique is that it works at a distance, i.e. it is not necessary to remove the insulation from a wire in order to measure its voltage and current [8].

When there are multiple conductors, the electric and magnetic fields from each superpose linearly. Thus a magnetic field sensor records a linear combination of the currents through each conductor and an electric field sensor records a linear combination of the voltages on each conductor. With enough sensors in different locations, the linear transformation from currents and voltages to sensor readings is invertible and the original currents and voltages may be recovered. The process of determining the transformation from sensor readings to voltages and currents is called "calibration".

Non-contact measurement of static electric potentials was first proposed by [9] in 1928. A vibrating plate is placed near an unknown potential, forming a time-varying capacitance. The voltage of the vibrating plate is adjusted until the vibrations induce no current through the plate, indicating that the plate's potential is equal to the unknown potential. The bandwidth of the sensor is limited by the vibration frequency of the plate. Reference [10] proposes a method by which the residual current through the sensor plate is integrated to determine the higher frequency components of the unknown potential.

Recent work has focused on capacitive sensors that do not vibrate. The induced current is integrated to obtain the unknown potential at all frequencies of interest. Reference [11] uses a capacitor to perform the integration. References [12] and [13] are optimized for the geometry of high voltage transmission lines. However, the gain of non-vibrating capacitive sensors is dependent upon the distance to the unknown conductor. Two sensor plates can be separately measured to compensate for this dependence [14]. Alternatively, large sensor plates can be placed close to a wire in order to enter a regime of operation in which the transfer function is not dependent on the separation distance [15].

The unique challenge of non-contact voltage sensing is reconstructing the input signal while rejecting pickup from other sources. Specifically, the currents induced by the input signal are proportional to the time derivative of the electric field and must be integrated in order to recover the input voltage. However, the noise currents induced by other sources have significant low-frequency components, which are amplified by an ideal integrator. This imposes a fundamental tradeoff between the accuracy of voltage measurements and a sensor's signal-to-noise ratio.

This thesis describes a non-contact voltage sensor based on [5]. The new sensor takes a differential measurement of two vertically stacked non-vibrating sensor plates in order to *maximize* the dependence of gain on plate-to-wire distance, so that the signal from a nearby wire is selected and the signals from more distant wires are rejected. The sensor is especially well suited for measurements that do not require the absolute scaling factor to be determined (e.g. total harmonic distortion and line regulation).

Non-contact measurement of currents is a less complicated proposition [16]. Whereas the proposed electric field sensors measure the time derivative of the electric field, this thesis uses off-the-shelf Hall effect sensors to measure the radiated magnetic fields. The signal produced by these sensors is directly proportional to the current, so integration of the signal is not necessary and disturbance rejection is less important. Hall effect sensing is a mature technology and no additional components or signal filters are required in order to acheive good results.

1.2 Thesis summary

This thesis presents hardware designs for electromagnetic field sensors with sufficient sensitivity and precision for use in non-contact power metering applications. The sensors are designed to be physically small, which allows for a wide variety of installation scenarios. A custom nylon enclosure protects the field sensors and allows them to be mounted to various types of wires with a single zip-tie. A digital integrating filter is designed for use with the electric field sensor. Compared to existing analog filters, it offers improved accuracy and superior disturbance rejection for voltage sensing applications.

A new analog-to-digital interface is designed to attach to up to four electric field sensors and four magnetic field sensors using standard ribbon cables. The interface digitizes analog inputs from the sensors and performs decimation and filtering on-board before streaming the resulting data to a computer over a buffered USB connection. The interface derives power for itself and the field sensors, as well as a reference ground voltage, from the USB connection. An injection-molded plastic enclosure protects the interface and its connections during use.

An improved mathematical model of non-contact power metering with multiple conductors is presented. For alternating signals, the model includes a spectral envelope preprocessor which extracts the Fourier coefficients of power usage over a rolling window of one line cycle. The preprocessor is shown to be equivalent to the Park transformation in the case of balanced three phase systems. An efficient real-time implementation of the non-contact preprocessor is provided. The implementation is integrated with the existing NilmDB framework.

Calibration of non-contact sensors is considered using the new mathematical framework. Algorithms are provided to calibrate non-contact sensors for direct or alternating currents with any number of conductors, whether or not a neutral conductor is present. A modified algorithm is given to perform partial calibration even when a known reference load cannot be attached to the conductors under measurement. The calibration procedure is extended to detect certain electrical faults.

A new algorithm is provided to detect transient events in NILM data (from both traditional and non-contact sensors). The algorithm uses an adaptive threshhold scheme to scale its sensitivity based on the noise floor of the signal in question. The new detector identifies

nearly all transient events without emitting false positives when the input signal is corrupted by noise.

Chapter 2

Non-contact Sensor Design

The wide variety of conductor geometries used for power transmission demand a flexible and reconfigurable system of electromagnetic field sensors. The sensors must be sensitive, because sometimes cables are designed to minimize emissions. For example, standard service entrance cable has a braided neutral conductor which is approximately coaxial with the line voltage conductors, so the magnetic fields generated by line currents are very nearly cancelled by the magnetic fields generated by return currents through the neutral conductor. The sensors must also be very small, because sometimes the space in which suitable fields exist is highly constrained. For example, when line conductors are contained within grounded metallic conduit, the radiated electric fields are confined to the interior of the conduit.

This thesis describes a system of non-contact sensors which uses a star topology to provide the required flexibility. The analog electronics for sensing electric and magnetic fields are compressed onto a 1 cm by 2 cm printed circuit board (the "analog" board). A typical installation would use four analog boards in order to measure the electric and magnetic fields at four different points. A separate circuit board containing a multi-channel analog-to-digital converter and digital signal processing hardware (the "digital" board) is attached to the analog boards using field-terminated ribbon cables. Finally, the digital board is attached to a computer (the final destination of the measurement data) using a USB connection.

2.1 Electrical design

The analog board senses magnetic fields using an off-the-shelf Hall effect IC. It senses electric fields using a new circuit topology that integrates the sensing elements into the copper layers of the printed circuit board. The sensed fields are transmitted to the digital board as voltage-mode analog signals over short runs of ribbon cable.

2.1.1 Voltage sensor

A parasitic capacitance C_p develops between a sensor plate and a nearby wire. The sensor plate is attached to AC ground by a resistance R and a capacitance C. The transfer function

from the wire voltage to the sensor plate voltage is given by

$$\frac{V_o(s)}{V_i(s)} = \frac{sRC_p}{sR(C + C_p) + 1}. \tag{2.1}$$

Conventional capacitive-divider sensors choose R to be very large. The transfer function is then approximated by

$$\frac{V_o(s)}{V_i(s)} \approx \frac{C_p}{C + C_p}.$$

If C is kept much smaller than C_p (which requires careful construction), the equation simplifies further to $V_o(s) \approx V_i(s)$. Unfortunately, this approach is not practical for the new sensor because C_p is tiny and the resistance required would impractically large.[1]

Instead, the new sensor operates in the regime where

$$|sR(C + C_p)| \ll 1$$

and so

$$\frac{V_o(s)}{V_i(s)} \approx sRC_p. \tag{2.2}$$

The sensitivity of the sensor is proportional to frequency. It is inversely proportional to the distance d between the wire and the sensor plate, because

$$C_p \propto \frac{1}{d}.$$

Note that the sensor measures the input signal v_I relative to its own ground, which must be attached (or at least AC coupled) to the input signal's ground.

As proposed by [5], improved localization is obtained by taking a differential measurement from two stacked sensor plates. This arrangement is shown in figure 2.1. Parasitic capacitance between the two plates is neglected from this model because in the differential mode it is equivalent to additional capacitance between each plate and ground.

The exact transfer function of the differential sensor is given by

$$\frac{V_o(s)}{V_i(s)} = \frac{sR(C_{p2} - C_{p1})(sRC + 1)}{(sR(C + C_{p1}) + 1)(sR(C + C_{p2}) + 1)}. \tag{2.3}$$

For frequencies satisfying $|sRC| \ll 1$, the transfer function is approximated by

$$\frac{V_o(s)}{V_i(s)} \approx sR(C_{p2} - C_{p1}) \tag{2.4}$$

which is analogous to equation 2.2 for the single-plate sensor.

[1] We are concerned with values of C_p as low as 0.1 pF and the lowest frequency of interest is 60 Hz. To place the breakpoint of the RC network at one tenth of that frequency would require a 265 GΩ resistor. Such impedances require the most exacting construction and cannot be achieved on a standard printed circuit board.

If the sensor plates are at a distance d from the wire and separated from each other by a distance $d_0 \ll d$, the differential capacitance is

$$C_{p2} - C_{p1} \propto \frac{1}{d} - \frac{1}{d + d_0} \approx \frac{d_0}{d^2}.$$

Therefore the sensitivity of the differential sensor is inversely proportional to the *square* of the distance between the wire and the sensor plates.

An alternative approximation more clearly reveals the frequency-dependent behavior of the differential sensor. When $C_{p1} \ll C$ and $C_{p2} \ll C$, the transfer function is roughly

$$\frac{V_o(s)}{V_i(s)} \approx \frac{sR(C_{p2} - C_{p1})}{sRC + 1}. \tag{2.5}$$

The input voltage is recovered by integrating the output voltage—in other words, the zero at the origin is cancelled by a new pole at the origin. At low frequencies, the remaining pole at $s = -1/RC$ has minimal effect. As the signal frequency increases, first order low-pass behavior will be observed.

Once the output is integrated, the differential capacitance $C_{p2} - C_{p1}$ must be determined in order to identify the sensor gain and recover the original input signal. If this capacitance is not known, the output will include an unknown constant scaling factor.

There are two factors which determine the sensitivity and performance of the voltage sensor: the geometry of the sensor plates, and the quality of the differential amplifier that is attached to them. Since the sensor should measure the voltage on one nearby wire without mixing in voltages from more distant wires, the sensor plates should not be made too large. The capacitance of the sensor plates then determines the maximum admissible input bias currents for the differential amplifier.

Based on the size of service entry cable and typical clearance constraints around existing wiring, the sensor plates are designed to have an area of $1\,\mathrm{cm}^2$. To minimize the cost of fabrication, the plates are built into the bottom two layers of a standard 1.6 mm four-layer printed circuit board (PCB). In a standard FR4 PCB, the bottom two layers are separated by 0.25 mm of laminate with a dielectric constant of approximately 4.5. Therefore the inter-plate capacitance is

$$C_{ip} = 4.5 \cdot \varepsilon_0 \cdot \frac{1\,\mathrm{cm}^2}{0.25\,\mathrm{mm}} = 15.9\,\mathrm{pF}.$$

With this information, the differential capacitance between the sensor plates and a nearby wire can be estimated. Suppose that the effective area of overlap between a wire and the sensor plates is $0.5\,\mathrm{cm}^2$, and the wire and the closer plate are separated by 1 mm of insulation with a dielectric constant of 2.1 (such as Teflon). The capacitance between the wire and the closer plate is

$$C_{p2} = 2.1 \cdot \varepsilon_0 \cdot 0.5\,\mathrm{cm}^2 \cdot 1\,\mathrm{mm} = 0.930\,\mathrm{pF}.$$

Then C_{p1} is given by the series combination of C_{p2} and C_{ip}, i.e.

$$\frac{1}{C_{p1}} = \frac{1}{C_{p2}} + \frac{1}{C_{ip}}$$

and the differential capacitance is

$$C_{p2} - C_{p1} = \frac{C_{p2}^2}{C_{ip} + C_{p2}} = 0.051\,\text{pF}.$$

The amplifier's input bias currents must be much smaller than the currents injected into the bias resistors by C_{p1} and C_{p2}. (This requirement is independent of the resistor values.) The limiting case is the lowest voltage of interest at the lowest frequency of interest—for design purposes, a 1 V signal at 60 Hz. The differential current produced by this signal is

$$2\pi f(C_{p2} - C_{p1})V = 2\pi \cdot 60\,\text{Hz} \cdot 0.051\,\text{pF} \cdot 1\,\text{V} = 19\,\text{pA}.$$

To avoid distorting the signal, the amplifier's input bias currents should not exceed about 1 pA. A low-cost instrumentation amplifier, such as the Texas Instruments INA332, meets this specification.

In the differential mode, the inter-plate capacitance of C_{ip} is equivalent to a capacitance between each plate and ground of

$$2C_{ip} = 31.8\,\text{pF}.$$

This capacitance reduces the bandwidth of the sensor and should be kept as small as possible. However, the amplifier is susceptible to common-mode disturbances which cause its inputs to exceed their allowable voltage range. In order to have some capacitive filtering of common mode inputs, an additional capacitance of 10 pF is provided between each sensor plate and ground. This gives a total differential mode plate-to-ground capacitance of

$$C = 41.8\,\text{pF}.$$

The last design task is to select the bias resistors attached to the sensor plates. The sensor gain is given by $sR(C_{p2} - C_{p1})$, so to maximize sensitivity R should be as large as possible. However, larger values of R increase the time constant RC and decrease the sensor bandwidth. A good balance between these requirements is achieved by $R = 1\,\text{M}\Omega$. The breakpoint of the input network is placed at

$$\frac{1}{2\pi RC} = 3.81\,\text{kHz}$$

which is significantly faster than the signals of interest. The sensor gain remains large enough to obtain usable voltage signals out of the amplifier.

Using equation 2.5, the transfer function of the specified analog sensor is

$$\frac{V_o(s)}{V_i(s)} \approx \frac{s \cdot 51\,\text{ns}}{s \cdot 42\,\mu\text{s} + 1}. \tag{2.6}$$

For sufficiently low frequencies, equation 2.4 applies and

$$\frac{V_o(s)}{V_i(s)} \approx s \cdot 51\,\text{ns}.$$

The final voltage sensor schematic is given in figure 2.2.

2.1.2 Current sensor

The magnetic field sensor analogous to the capacitive voltage sensor described above would be an inductive sensor such as a Rogowski coil. However, due to the difficulty of fabricating large inductive structures and the comparitively small magnitude of the induced voltages, this approach is not practical for low-cost sensors.

Instead, an off-the-shelf Hall effect IC is used to measure the magnetic field strength. Hall effect ICs are in common use for the measurement of currents. However, the ICs are typically relatively insensitive, requiring the conductor under test to be wound around a field-concentrating toroid in order to measure smaller currents with precision. The use of Hall effect ICs to measure radiated emissions from a multiple-conductor cable, without any modifications or additional magnetic components, is a new idea [5].

The majority of commercially available Hall effect sensors are provided in single inline packages which detect fields perpendicular to the plane of the package. However, a fabrication process developed by Melexis provides a highly sensitive Hall effect sensor in a standard surface-mount SOIC package. The sensor detects magnetic fields parallel to the pin direction of the package. which allows for much smaller clearances between an analog board and the wire under measurement. Thus the Melexis 91206 is chosen for use as the analog board's magnetic field sensor.

Reference [5] also uses an actively compensated tunneling magnetoresistive sensor obtain roughly 10 times higher sensitivity to magnetic fields than those acheivable by Hall effect sensors. Due the cost and complexity of the sensing element and compensation circuit, a Hall effect sensor is preferred. In practice, the noise floor of either system is determined by magnetic noise that is accurately sensed by both sensors, and not by additional sensor-induced noise.

2.1.3 Microcontroller

The digital board contains a microcontroller that is responsible for digitizing the analog signals from the voltage and current sensors, filtering and decimating the digitized signals, and streaming the resulting data over a USB connection to a computer. The best support for open-source toolchains is provided by ARM microcontrollers, of which the Cortex-M4 architecture is the most well suited to low-cost digital signal processing.

The Atmel SAM4S is selected as the Cortex-M4 microcontroller for the digital board. This microcontroller has a 12-bit ADC with sufficient capacity to attach four analog boards, for a total of four electric field sensors and four magnetic field sensors. The integrated memory is large enough to hold a one second transmissiion buffer, and the integrated flash is well more than large enough to store the necessary code. The microcontroller has an I2C peripheral which may be used to communicate control information with the analog boards.

The digital board is powered by a USB connection. It provides a regulated 4.5 V rail for the analog circuits and an independently regulated 3.3 V rail for the microcontroller. The board is equipped with a precision 12 MHz crystal to enable high-speed USB communication and provide an accurate clock for ADC sampling. The analog connections are terminated by 10 kΩ resistors near the microcontroller.

2.2 Hardware

A complete system implementation including printed circuit boards, enclosures, and interconnects is provided. The system implementation is suitable for installation in a wide variety of non-contact energy monitoring applications.

2.2.1 Printed circuit boards

The analog board, shown in figure 2.3, is implemented as a standard four layer printed circuit board. Because of the high impedances present on the PCB, special care must be taken to include guard traces around sensitive nodes and to clean conductive residue from the board after assembly. The PCB an integrated capacitive electric field sensor, a Hall effect-based magnetic field sensor, an EEPROM, and a connector for cabled attachment to a microcontroller. All components are surface mount. As shown in figure 2.4, the Hall effect sensor is on the bottom side and all other components are on the top side.

The digital board is implemented as a standard two layer printed circuit board. The board includes a microcontroller, a 12 MHz crystal, two low-dropout voltage regulators, a push-button switch, and several LEDs. All components are surface mount and all are located on the top side of the board. The digital board is depicted in figure 2.5.

An I2C EEPROM is located on each analog board and stores a unique board identifiifier. An I2C multiplexer on the digital board allows the microcontroller's I2C interface to reach up to four attached analog boards without address conflicts. In the future, this may allow the digital board to automatically detect which analog boards are attached and even store calibration information back to the analog boards. At the time of this writing, however, the I2C hardware is not used.

2.2.2 Interconnects

Each analog board is attached to the digital board with a six-conductor ribbon cable. Conductor number one is located at the keyed end of the Micro-Match connector. The six conductors have the following pinout:

1. Magnetic field sensor
2. Electric field sensor
3. 4.5 V
4. Ground
5. I2C clock
6. I2C data

Crosstalk between the analog conductors and the I2C conductors is minimized by this arrangement. Even so, the digital board's firmware does not perform I2C communication while analog data acquisition is taking place.

The digital boards communicate with a computer over a Universal Serial Bus connection. Standard USB cables have a braided shield which is separate from the internal ground conductor. Since the ground reference for all of the sensors is derived from the USB cable, grounding of the shield has an important effect on measurement quality. In particular, due to common impedance coupling between any circulating currents and the reference ground, *the USB shield is not grounded at the digital board.* This prevents the creation of a large ground loop. Pads are provided for an optional RC network connecting the cable shield and ground conductors. If they are used, the resistance should be kept quite large and the breakpoint of the RC network should be set higher than the highest signal frequency of interest.

2.2.3 Enclosures

The digital board is designed to fit within the Polycase KT-40, an off-the-shelf injection molded ABS enclosure. The enclosure captures a standard thickness circuit board between standoffs on either side with an appropriate gap between them. The board is shaped so that all connectors fit inside of the enclosure, which improves mechancial robustness for installations in harsh environments. A custom slot and hole are routed out of the enclosure for the ribbon cables and USB cable to exit. This assembly is shown in figure 2.6.

The analog board has more demanding casing requirements, so it is used with a custom enclosure that is fabricated out of nylon by a selective laser sintering process. Design assistance was provided by Alberto Mulero. The enclosure holds a board with retention tabs at either end and attaches to a wire with a single zip tie. The zip tie also serves as strain relief for the ribbon cable. A cover for top of the enclosure is laser cut out of sheet acrylic. Figure 2.7 shows an empty analog board enclosure and figure 2.8 shows an analog board installed in the nylon enclosure with a clear acrylic cover.

2.3 Software

The embedded software which runs on the digital board is fairly complex. It samples the microcontroller's built in ADC, applies filtering and decimation to those samples, and streams the result to a computer over a USB connection. All software discussed in this section is given in appendix E.4.

2.3.1 Decimator

The microcontroller's analog to digital converter runs at its maximum speed of approximately 1 MHz. It is triggered at 96 kHz to take one sample from each of the 8 input channels. The highest frequency of interest is less than 1 kHz, so in order to reduce the bandwidth and computational requirements of the USB connection, the microcontroller decimates each channel by 32 times to a sample rate of 3 kHz. As an added benefit, this amount of oversampling increases the effective resolution of the ADC from 12 bits to 14.5 bits.

The decimation proceeds in two stages. First, a 16-tap FIR filter applies a rough low-pass filter and reduces the sample rate by 8 times. Second, a 32-tap FIR filter applies a precise low-pass filter and reduces the sample rate by an additional 4 times. This cascaded

architecture results in an accurate frequency response with far less computational load than single stage decimation, which would require a 128-tap filter to achieve the same result.

The firmware's FIR filter implementation convolves the filter coefficients with the signal using fixed point arithmetic and then shifts the result right by 15 bits. The first filter (figure 2.9) has a zero-frequency gain of 4 after the right shift. The second filter (figure 2.10) has a zero-frequency gain of 2 after the right shift. Thus the 12 bit ADC signal results in a 15 bit output signal. The frequency response of the combined filter-decimator system is plotted in figure 2.11.

Decimation and filtering are performed within the ADC interrupt service routine (ISR) using a round-robin scheduling scheme. Define the time t to be a discrete quantity which increments by one each time the ISR runs. At time t, the first stage FIR filter processes the samples from time $8\lfloor t/8 - 2 \rfloor$ to time $8\lfloor t/8 \rfloor$ on channel number ($t \bmod 8$). If $t \bmod 4 = 0$, the second stage FIR filter processes the first stage output from time $32\lfloor t/32 - 8 \rfloor$ to time $32\lfloor t/32 \rfloor$ on channel number $((t/4) \bmod 8)$.

This decimation scheme has several advantages. The computational load is distributed evenly, so the USB stack and other time-critical tasks are never starved for CPU time. Phase shift between channels is reduced by 32 times: the sampling delay between adjacent input channels is just 1 µs, whereas an ADC that did not oversample would introduce a sampling delay of 42 µs. Although output samples are delayed by the filtering process, they are delayed by the same amount for every channel.

2.3.2 Protocol

The USB data pipeline provides an asynchronous byte-oriented transfer protocol which may occasionally drop bytes if any transmission buffers overflow. The digital board firmware provides a 32 kiB buffer for outgoing data in order to minimize the occurrence of dropped samples. Even so, some form of alignment detection is necessary for the computer to robustly ascertain the meaning of each incoming byte.

Data is transmitted in 20 byte little-endian frames with the following layout:

1. 2-byte alignment word: hexidecimal value 0x807f
2. 2-byte status word (described below)
3. 2-byte channel 0 value
4. 2-byte channel 1 value
5. 2-byte channel 2 value
6. 2-byte channel 3 value
7. 2-byte channel 4 value
8. 2-byte channel 5 value
9. 2-byte channel 6 value

10. 2-byte channel 7 value

The channel values are signed 15 bit quantites (12 bit ADC values with an additional gain of 8 from the decimating filter), which means that 0x80 and 0x7f can never appear as the high order byte of a channel value. The top two bits of the status word are reserved and always zero. Therefore the word 0x807f cannot appear anywhere in the transmitted data except as the alignment word. This permits a computer to unambiguously detect whether the incoming frames are properly aligned.

The status word is interpreted as a bit mask. Each bit denotes a different error condition. Bit i for $i < 8$ denotes ADC saturation on the ith channel. Bit 8 denotes that the on-board transmission buffer overflowed. Bits 9 through 12 are unused at the time of this writing. Bit 13 denotes that the push-button switch is pressed. Bits 14 and 15 are reserved (for frame alignment) and must never be set.

2.3.3 Toolchain

The digital board firmware is written in C and compiled using the ARM Cortex-M4 target of the GNU Compiler Collection. The firmware is a "bare metal" application, i.e. there is no operating system and the standard library is not linked in. The vendor-supplied Atmel Software Framework provides an abstraction layer[2] for usage of the microcontroller's integrated peripherals.

The software is compiled using the standard GNU Make utility. A custom makefile is provided to compile and link the firmware with the Atmel Software Framework. The makefile also supports programming and interactive debugging with a Black Magic Probe debugger using the "gdb" and "flash" targets, respectively.

The Atmel Software Framework is expected to be installed to the "asf" subdirectory of the main firmware directory. Alternatively, the "asf" subdirectory may be a symbolic link to the actual loaction of the framework. The compiler, linker, and debugger are maintained by ARM and packaged under the names "arm-none-eabi-gcc", "arm-none-eabi-binutils", and "arm-none-eabi-gdb" in most modern GNU/Linux distributions.

2.3.4 Programming

The Atmel SAM4S microcontroller on the digital board possesses integrated non-volatile memory for program storage. A program binary can be loaded into this memory in either of two ways. First, a program can be loaded using a programmer attached to the debugging header on the digital board. Second, a program can be loaded from a computer over the digital board's USB port.

The 10-pin debugging header is standardized across all ARM Cortex-M microcontrollers. It uses the Serial Wire Debug protocol for communication between the microcontroller and the programmer or debugger. The firmware given in appendix E.4 includes a Makefile target to load the program binary using a Black Magic Probe debugger, but any other ARM Cortex-M debugger will also be capable of loading a binary through the debugging header.

[2]Unfortunately, it is often more similar to an obfuscation layer.

Programming over USB uses a bootloader specific to the Atmel SAM4S. The bootloader enumerates as a communications device class modem, which most operating systems treat as a serial port. Atmel's proprietary "Sam-ba" software must be used to transfer a binary file to the flash memory. Programming over USB does not work when code other than the Atmel bootloader is running on the microcontroller.

The SAM4S has a special region of general purpose non-volatile memory (GPNVM) which stores various configuration settings. GPNVM bit 1 is particularly important, since it controls whether the microcontroller runs the Atmel bootloader (from ROM) or the program binary (from flash) on startup. GPNVM bit 1 is clear by default, so the Atmel bootloader runs on startup for unprogrammed chips. Either programming method will set GPNVM bit 1, and the Atmel bootloader will no longer run.

In order to update the program binary over USB after the board has been programmed once, it is necessary to clear GPNVM bit 1. Fortunately, the firmware has a special function to clear GPNVM bit 1. This function is activated by sending the character "g" over the USB communications device class interface which is exposed by the digital board firmware.

2.4 Voltage sensor integrating filter

The electric field sensor's output must be integrated to recover the original voltage being measured. Past implementations have used an analog integrating filter [5], but better performance is possible by performing the integration digitally. The design of the integrating filter presents a fundamental tradeoff between accuracy and disturbance rejection. Specifically, there are three design requirements:

1. The filter must faithfully reconstruct the voltage being measured.
2. The filter must reject low frequency disturbances, such as those caused by thermal drift.
3. The filter must recover quickly from impulsive disturbances.

These requirements correspond to the following three properties of a linear filter:

1. The filter's frequency response should be inversely proportional to the frequency, and introduce 90 degrees of phase lag, for every frequency present in the voltage being measured.
2. The filter's frequency response should roll off quickly below the frequencies of interest.
3. The filter's impulse response should be short.

These goals have previously been realized by a cascade of two analog filters: a high-pass filter which admits the signals of interest but blocks low frequency disturbances, followed by an integrator to recover the original voltage signal. The challenge is that a causal analog filter cannot have a sharp transition between its stop band and pass band without introducing significant phase distortion—but if the transition to the stop band is gradual, low frequency disturbances will be admitted and amplified by the integrator.

Throughout this section, ω refers to a normalized angular frequency with units of radians per sample. Suppose that there are $2N$ samples per line cycle, so that the frequency of the nth harmonic is $\pi n/N$ radians per sample. The frequency response of an ideal integrating filter is given by

$$H_i(\omega) = \frac{\pi}{j\omega N}. \tag{2.7}$$

(This filter is "ideal" only in that it integrates signals perfectly and has a unit magnitude response at line frequency. It does not satisfy the second and third filter requirements.)

If the sampled line frequency of π/N radians per sample corresponds to 60 Hz in continuous time, the frequency response of the analog filter in [5] is given by

$$H_a(\omega) = \frac{j\omega\pi/N}{(j\omega + 1/\tau_0)(j\omega + 1/\tau_1)} \tag{2.8}$$

with

$$\tau_0 = (2.2\,\mu\text{F}) \cdot (12.1\,\text{k}\Omega) \cdot (60\,\text{Hz}) \cdot 2N$$
$$\tau_1 = (2.2\,\mu\text{F}) \cdot (47\,\text{k}\Omega) \cdot (60\,\text{Hz}) \cdot 2N.$$

This analog filter is compared with two digital finite impulse response (FIR) filters. The FIR filters have antisymmetric impulse responses (such filters are known as "Type 3" FIR filters). As a consequence, they have zero group delay, introduce 90 degrees of phase lag at all frequencies, and do not pass signals at zero frequency or at the Nyquist rate.

The first FIR filter is the Type 3 filter with $2N - 1$ taps whose frequency response H_1 satisfies

$$H_1\left(\frac{\pi n}{N}\right) = \frac{1}{jn} \qquad n \in \mathbb{Z},\ 1 \le |n| < N.$$

The second FIR filter is the Type 3 filter with $4N - 1$ taps whose frequency response H_2 satisfies

$$H_2\left(\frac{\pi n}{2N}\right) = \frac{2c_n}{jn} \qquad n \in \mathbb{Z},\ 1 \le |n| < 2N$$

with

$$c_n = \begin{cases} 1/2 & |n| = 1 \\ 1 & 2 \le |n| < 2N - 1 \\ 3/4 & |n| = 2N - 1. \end{cases}$$

The filter impulse responses are computed using the inverse discrete Fourier transform:

$$h_1[t] = \frac{1}{N} \sum_{n=1}^{N-1} \left(\frac{1}{n} \cdot \sin\left(\frac{\pi n t}{N}\right)\right) \qquad |t| < N \tag{2.9}$$

$$h_2[t] = \frac{1}{N} \sum_{n=1}^{2N-1} \left(\frac{c_n}{n} \cdot \sin\left(\frac{\pi n t}{N}\right)\right) \qquad |t| < 2N \tag{2.10}$$

where t is an integer representing the discrete time. The impulse responses are plotted in figure 2.12 and figure 2.13.

One way to think of each filter is as the cascade of a special high-pass filter and an ideal integrator. In the case of $h_2[t]$, the integrator's impulse response is a unit step and the high-pass filter's impulse response is plotted in figure 2.14. This type of high-pass filter can be used on its own for the suppression of low-frequency disturbances in signals which do not need to be integrated.

By definition, $H_1(\omega) = H_2(\omega) = H_i(\omega)$ at line frequency and all of its harmonics below the Nyquist rate. h_1 is the shortest impulse response whose Fourier transform has this property, and h_2 is designed to have a smoother frequency response at the expense of being twice as long as h_1.

From the impulse responses, the discrete time Fourier transform gives the continuous frequency responses. The analytical expressions are omitted here because they provide no additional insight. Figure 2.15 shows the magnitude response of each filter and figure 2.16 shows the relative magnitude of the difference between each filter's response and the ideal response.[3]

Consider the response of each filter to the signal

$$x[t] = \sin(\pi t/N \cdot 60/50) + 30\delta[t].$$

This represents the case where the digital filters were designed for a line frequency of 50 Hz but the actual frequency is 60 Hz, and an impulsive disturbance of magnitude 30 occurs at time $t = 0$.[4] These responses are plotted in figure 2.17 and exemplify the benefits and drawbacks of each type of filter.

Lastly, to illustrate the superior disturbance rejection of the digital filters, the output of each filter is computed for the same input sequence of pink (i.e. $1/f$) noise. The results are plotted in figure 2.18. Clearly, the analog filter exhibits a greater amount of error amplification.

Although the FIR filters are non-causal, both become causal when composed with a finite time delay. It is therefore possible to implement them, with the caveat that the output will not be known in real time. In particular, the first FIR filter delays its outputs by half of a line cycle and the second FIR filter delays its outputs by one full line cycle.

[3]For a frequency response $H(\omega)$, the exact function plotted in figure 2.15 is $20\log_{10}|H(\omega)|$ and the exact function plotted in figure 2.16 is $20\log_{10}|H(\omega)/H_i(\omega) - 1|$.

[4]Such disturbances often occur when a large inductive load is disconnected, resulting in a high instantaneous rate of voltage change on the inductor. This produces a powerful electric field which causes the sensor plate voltages to briefly exceed the common-mode input range of the amplifier. The INA332 drives its output to the positive rail when this condition occurs.

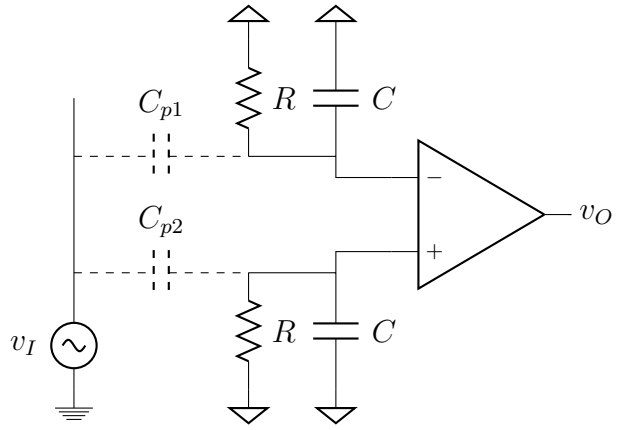


Figure 2.1: Circuit for differential non-contact sensing of an AC voltage. Equation 2.3 provides the transfer function for this circuit.

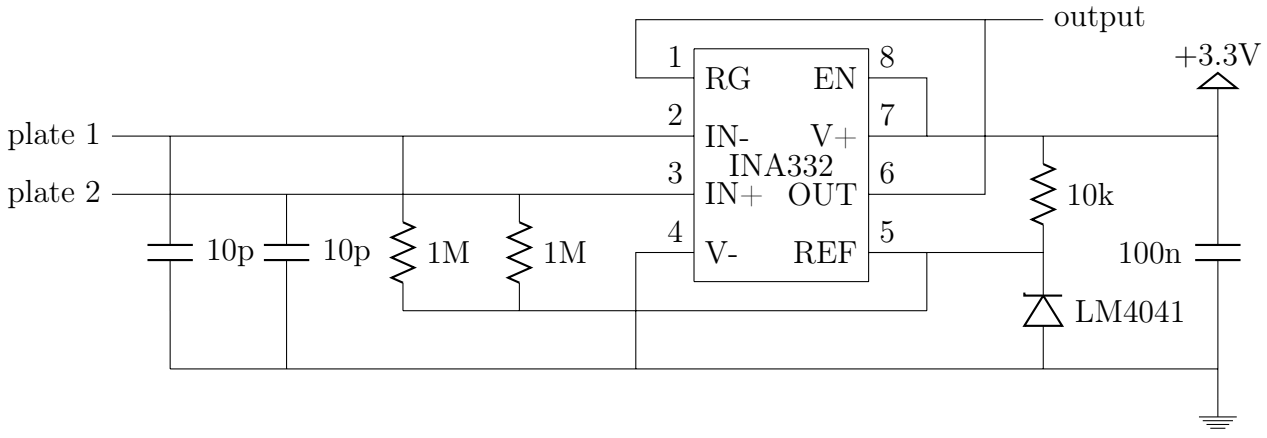


Figure 2.2: Implementation of the voltage sensor circuit using a TI INA332 instrumentation amplifier and a TI LM4041 voltage reference. The combined cost of these parts is less than $3 in quantity 1.

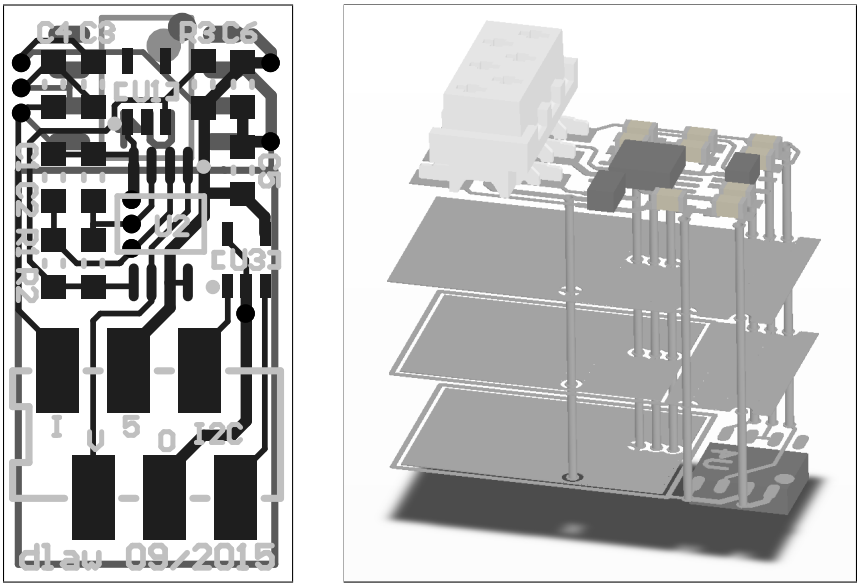


Figure 2.3: 2D and 3D exploded views of the analog board. The board dimensions are 1 cm by 2 cm. From top to bottom, layers contain: (1) connector, instrumentation amplifier, and supporting components, (2) ground plane, (3) sensor plate and ground plane, (4) sensor plate and Hall effect IC.

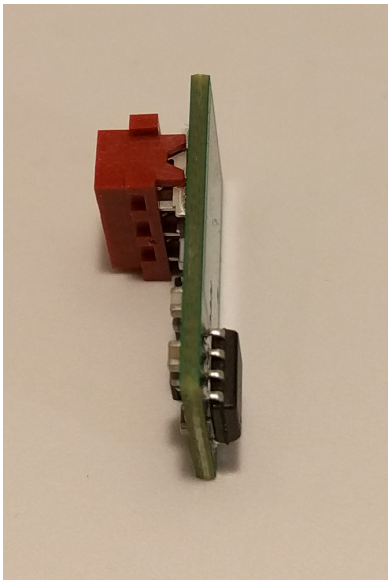


Figure 2.4: Side profile view of the analog board. The bottom side (figure right) contains the Hall effect sensor and capacitative sensing plates. The top side (figure left) contains the Micro-Match connector and all other electrical compnents.

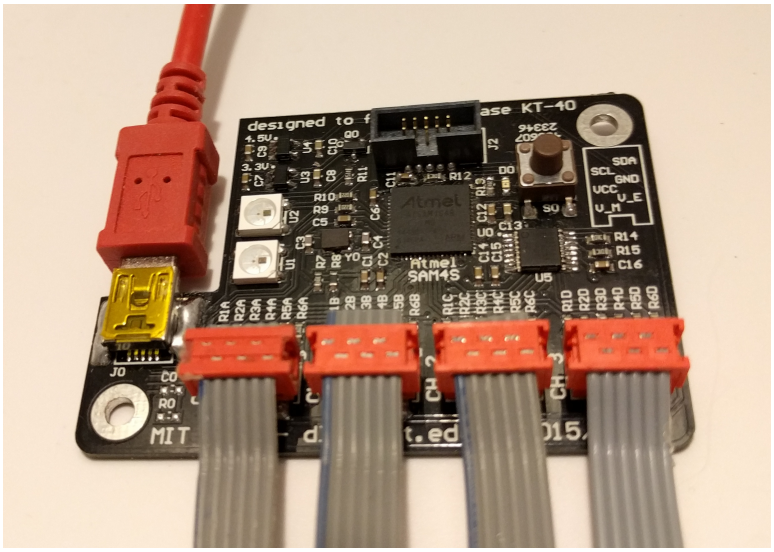


Figure 2.5: The digital board with cabled attachment to four analog boards and a computer.

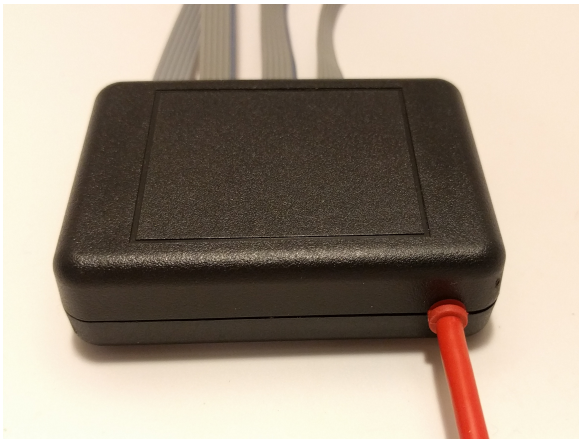


Figure 2.6: Modified Polycase KT-40 enclosure with a digital board installed. For robustness, all connectors are entirely contained within the enclosure.

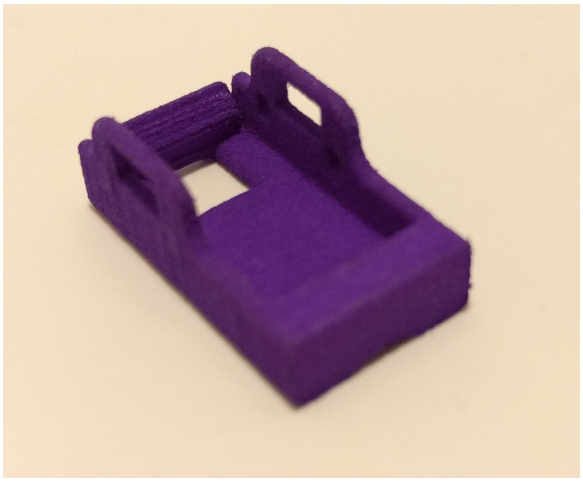


Figure 2.7: Sintered nylon enclosure for the analog board. The hole in the bottom allows the Hall effect sensor to directly contact the wire's insulation, maximizing sensitivity to magnetic fields. The slots on either side are for a zip-tie to attach the assembly to a wire.

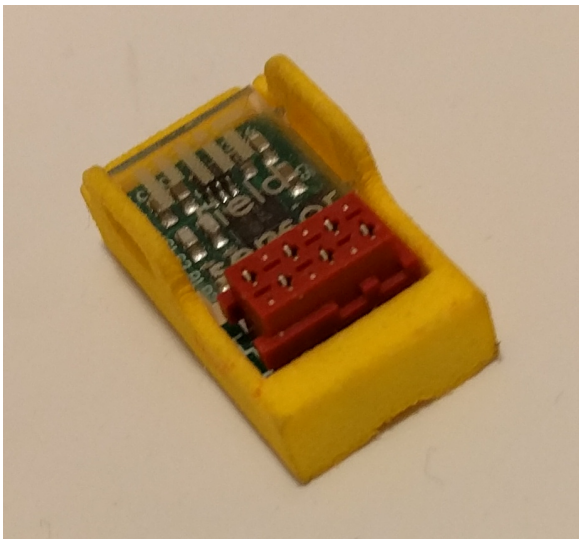


Figure 2.8: Analog board installed in the nylon enclosure with a clear acrylic cover. The assembly is 0.9 inches long by 0.5 inches wide by 0.4 inches tall, including a ribbon cable and the mating connector (not shown).

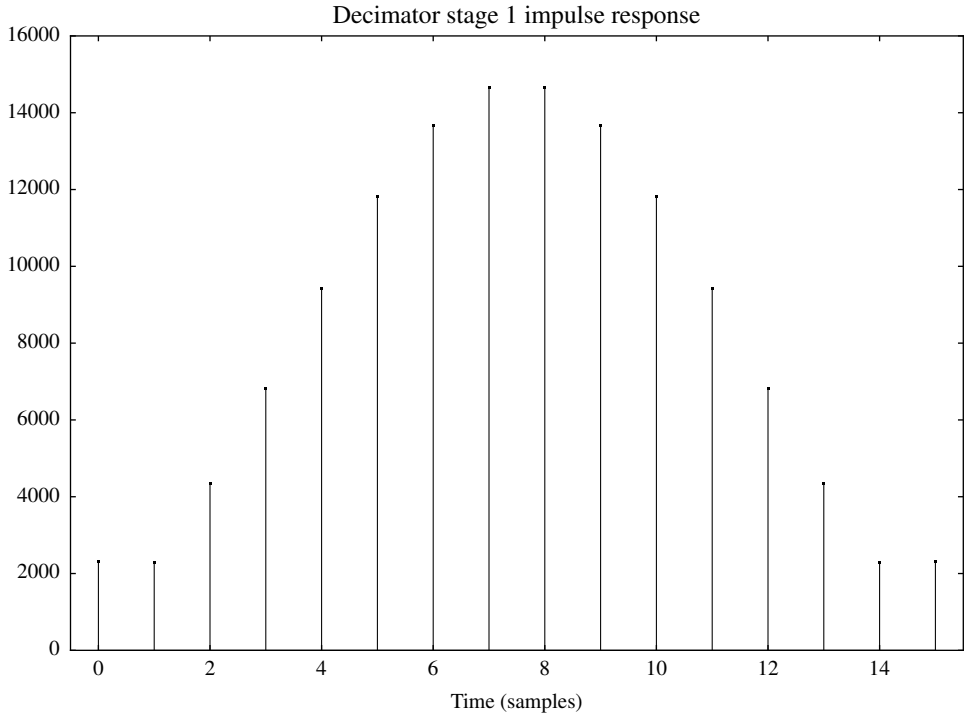


Figure 2.9: Impulse response of the 16-tap first stage FIR filter. The output from this filter is computed once every 8 samples of the input. The zero-frequency gain of the filter is 4 after a 15 bit right shift.

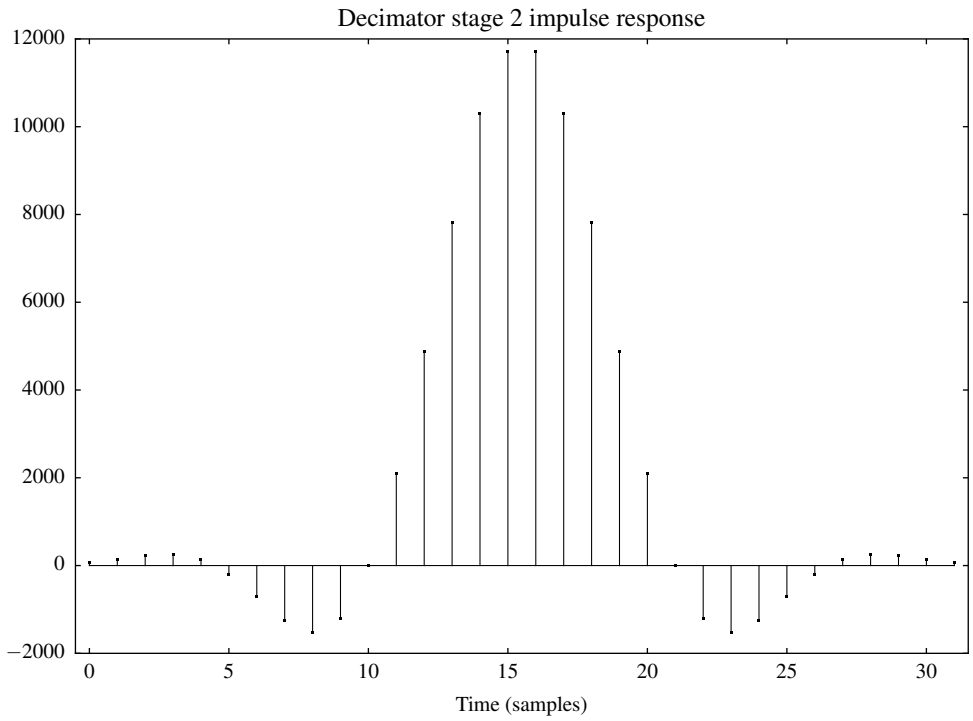


Figure 2.10: Impulse response of the 32-tap second stage FIR filter. The output from this filter is computed once every 4 samples of the input. The zero frequency gain of the filter is 2 after a 15 bit right shift.

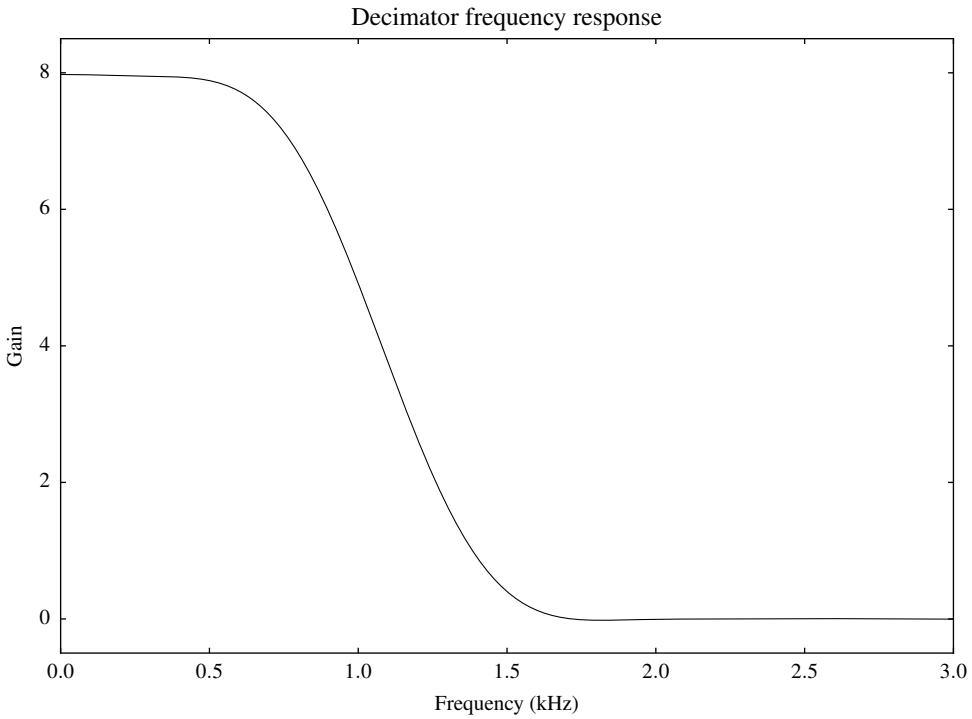


Figure 2.11: Frequency response of the combined filter-decimator system, including a 15 bit right shift after each FIR filter. The zero-frequency gain of 8 results in a 15 bit wide output signal, which is consistent with the additional resolution provided by oversampling the 12-bit ADC by 32 times.

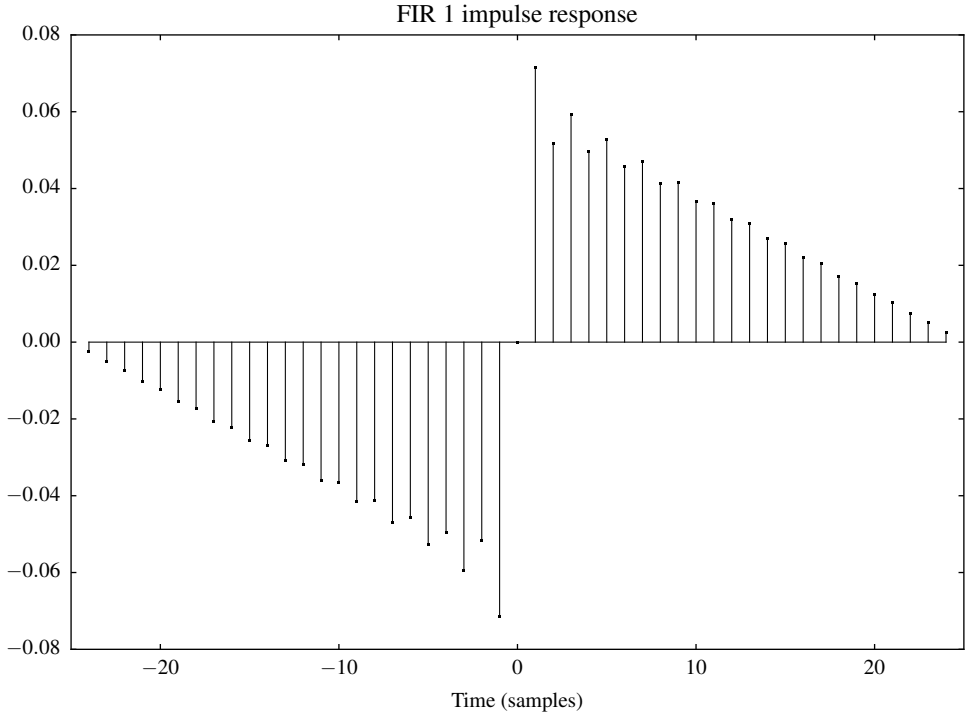


Figure 2.12: Impulse response $h_1[t]$ for $N = 25$. The impulse response is zero when $|t| \geq 25$.

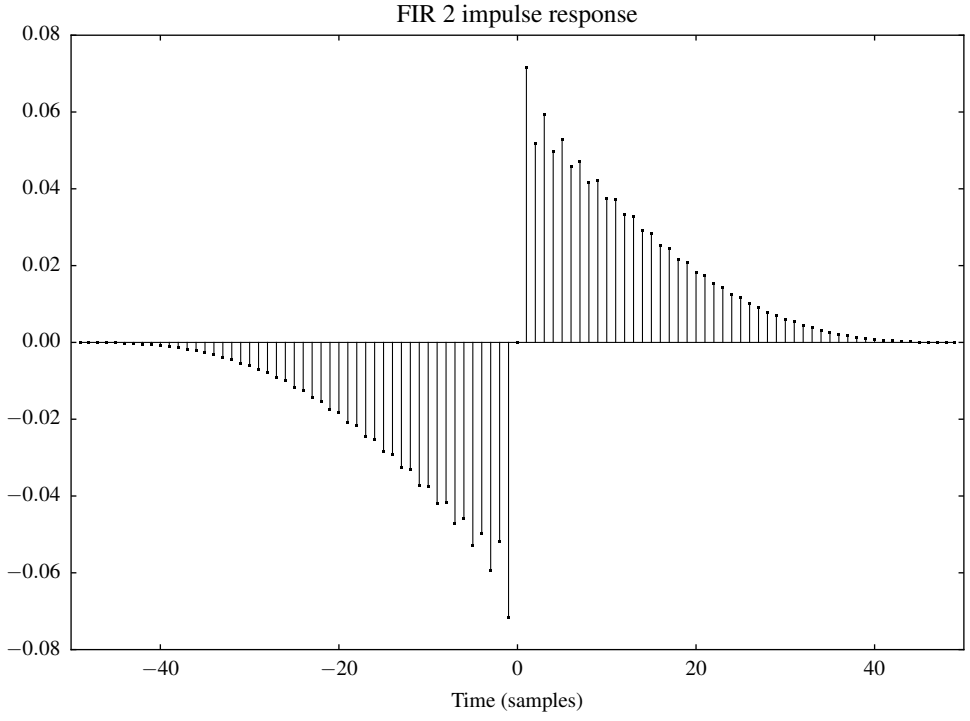


Figure 2.13: Impulse response $h_2[t]$ for $N = 25$. The impulse response is zero when $|t| \geq 50$.

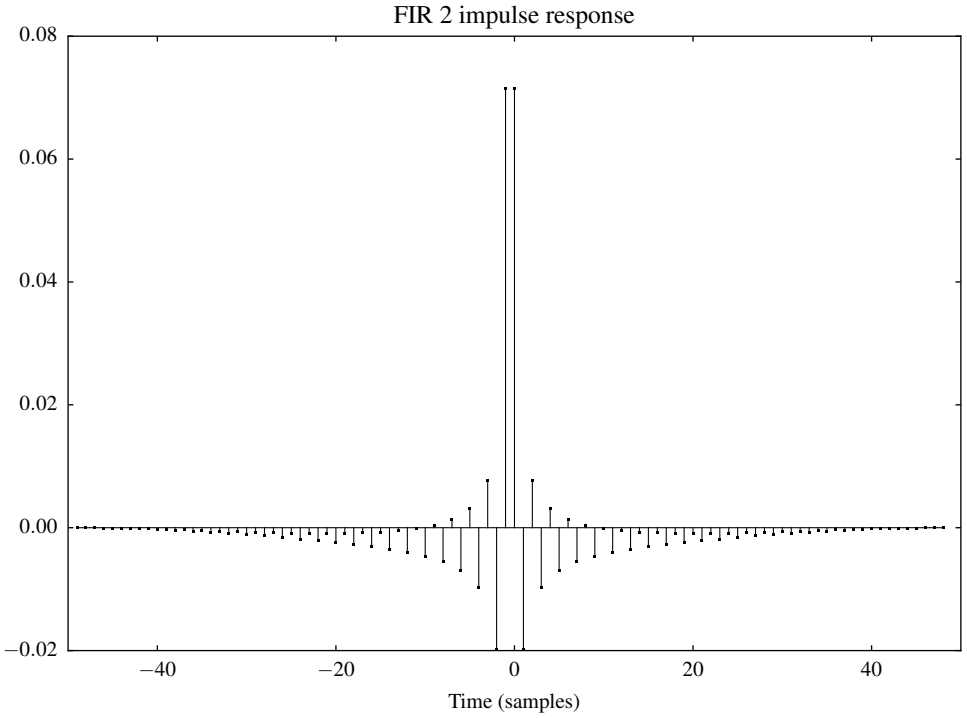


Figure 2.14: Impulse response $h_2[t]$ deconvolved with a unit step for $N = 25$. Note the half-sample delay which is introduced by this filter.

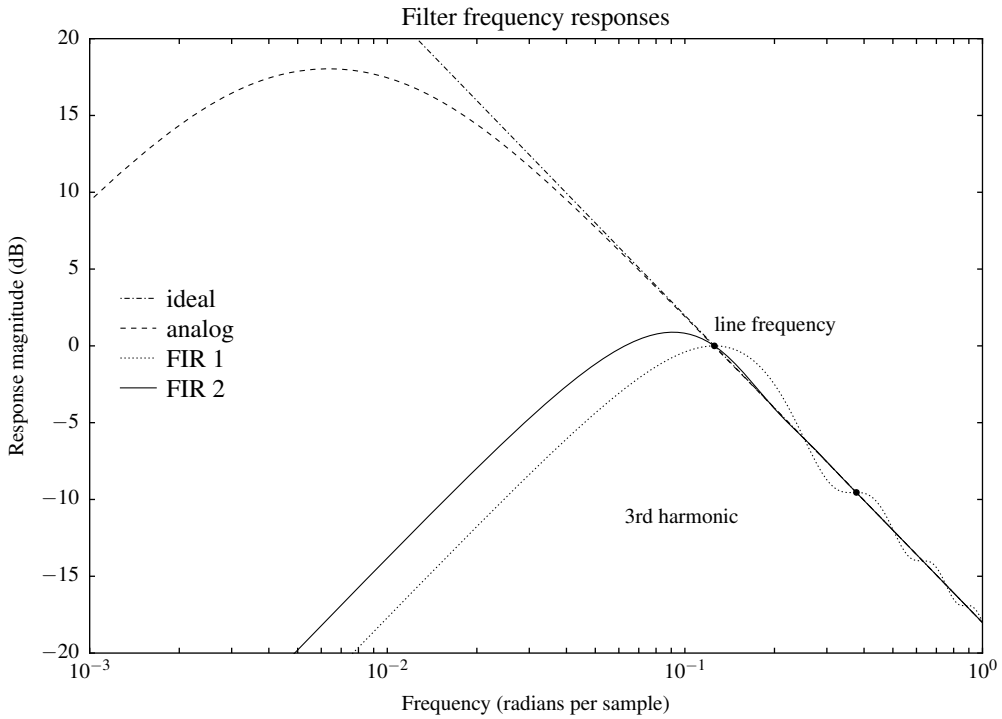


Figure 2.15: Magnitude behavior of the filters for $N = 25$. Note the logarithmic horizontal scale. The analog filter introduces phase distortion which is not depicted on this plot. Amplification of low frequency disturbances is roughly proportional to the area under the left half of the response curve.

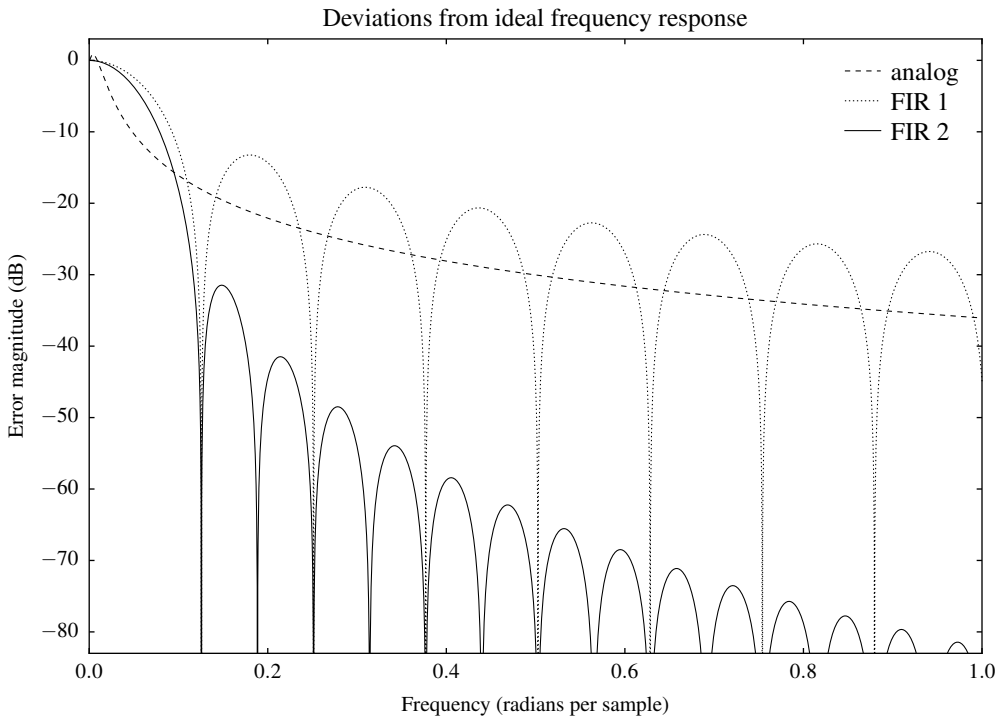


Figure 2.16: Magnitudes of the relative deviations from the ideal frequency response for $N = 25$. Both of the FIR filters have zero error at line frequency and its harmonics. Deviation from the ideal response is necessary and desirable at frequencies below line frequency.

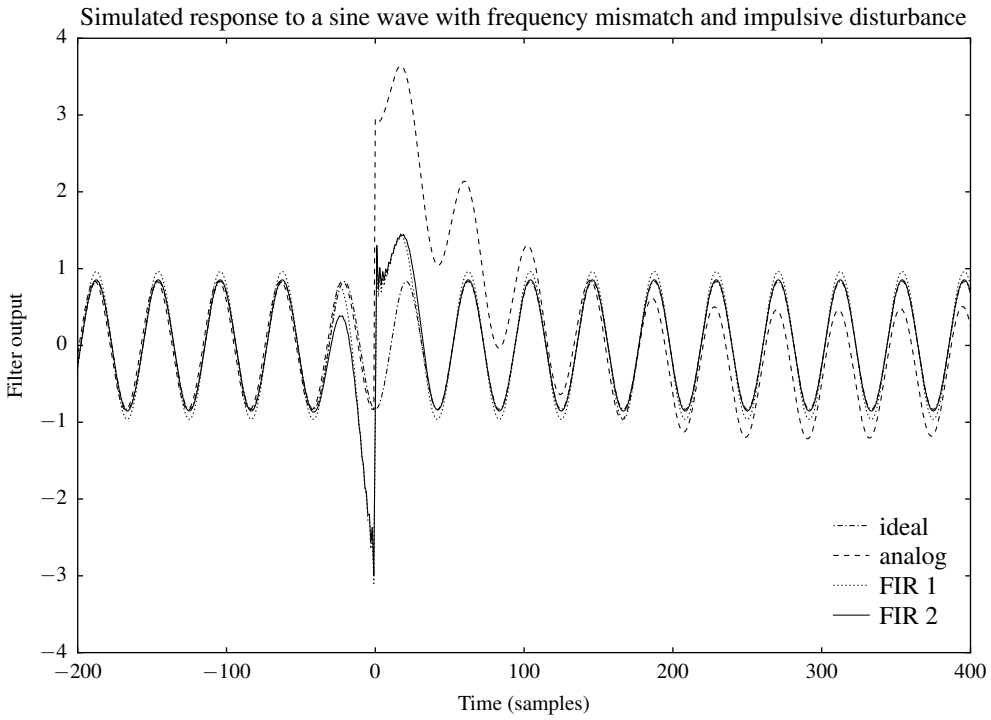


Figure 2.17: Simulated response of the filters to an impulsive disturbance with magnitude 30 at $t = 0$. The disturbance affects FIR 1 for $-25 < t < 25$ and FIR 2 for $-50 < t < 50$, but the analog filter has not yet recovered from the disturbance at $t = 400$. The filters are designed for a line frequency of 50 Hz with $N = 25$, but the input signal is provided at 60 Hz to demonstrate that the filters perform well even when line frequency is not known in advance.

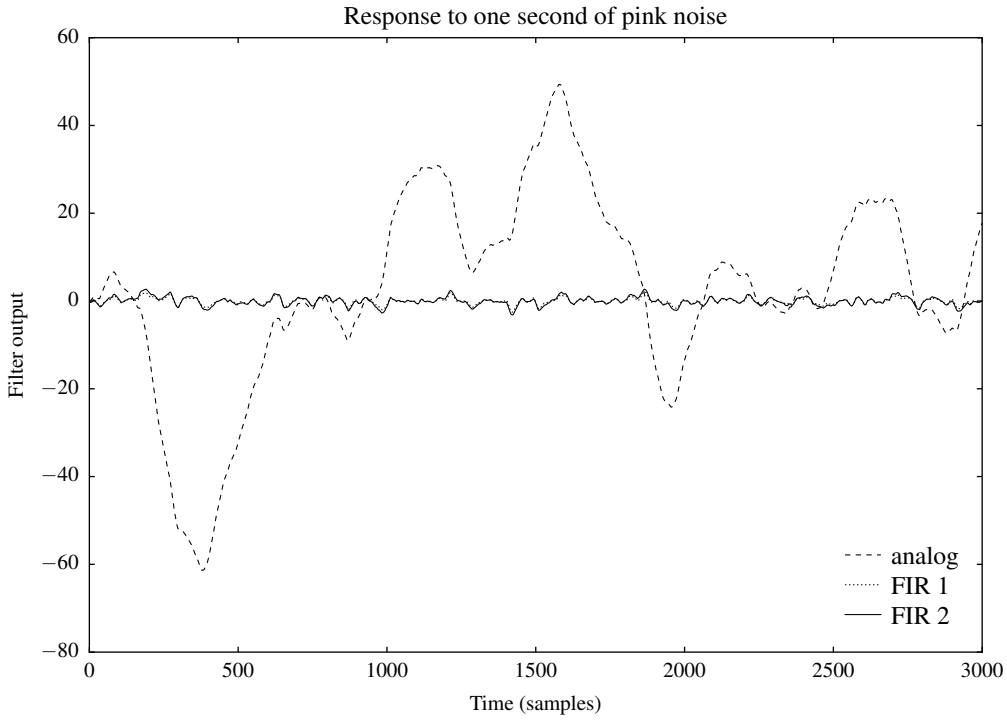


Figure 2.18: Simulated response of each filter to the same sequence of pink noise. The pink noise was generated as the cumulative sum of a sequence of numbers chosen uniformly at random between -0.5 and 0.5.

Chapter 3

Non-contact Signal Reconstruction

Each conductor in a multiple-conductor cable radiates a magnetic field proportional to its current and an electric field proportional to its voltage. Each magnetic or electric field sensor detects a superposition of the fields due to each conductor. This chapter considers the question of how to recover meaningful information from the sensed fields.

A rigorous mathematical model of non-contact sensing is developed. This model is used to express an algorithm (called the "calibration" algorithm) which determines the transformation to recover the conductor currents from the sensed fields. The calibration algorithm is initially stated for direct curents and is then extended to alternating currents and several other special cases.

Once the required transformation is determined, two separate software systems are provided to apply it to incoming data in real time. The first system adds a minimal number of new components to the existing NilmDB system [4]. The second system replaces a large swath of NilmDB components with a vertically integrated C program, resulting in a substantial speedup at the expense of interoperability with existing NILMs.

3.1 Mathematical model of non-contact sensing

Suppose that a cable containing n conductors is instrumented with k magnetic field sensors and l electric field sensors. The current and voltage on the xth conductor at time t are given by $i_x(t)$ and $v_x(t)$, respectively. Recall that our non-contact electric field sensors measure the time derivative of the electric field. The magnetic field measured by the xth magnetic sensor is $s_x(t)$ and the time derivative of the electric field measured by the xth electric sensor is $\dot{e}_x(t)$. Let $\boldsymbol{i}(t)$, $\boldsymbol{v}(t)$, $\boldsymbol{s}(t)$, and $\dot{\boldsymbol{e}}(t)$ denote the column vectors formed by stacking the corresponding scalar quantities for each of the n conductors. The symbol $\dot{\boldsymbol{v}}(t)$ denotes the elementwise derivative of $\boldsymbol{v}(t)$ with respect to t.

The electromagnetic fields from each conductor superpose linearly, so the sensor geometry and the laws of physics determine a k-by-n matrix $\boldsymbol{M}$ and an l-by-n matrix $\boldsymbol{N}$ such that

$$\boldsymbol{s}(t) = \boldsymbol{M}\boldsymbol{i}(t) \tag{3.1}$$

and

$$\dot{\boldsymbol{e}}(t) = \boldsymbol{N}\dot{\boldsymbol{v}}(t). \tag{3.2}$$

The most general goal of non-contact sensing is to recover $\boldsymbol{i}$ and $\boldsymbol{v}$ from $\boldsymbol{s}$ and $\dot{\boldsymbol{e}}$. This is accomplished by finding an n-by-k matrix $\boldsymbol{K}$ and an n-by-l matrix $\boldsymbol{L}$ such that

$$\boldsymbol{i}(t) = \boldsymbol{K}\boldsymbol{s}(t) \tag{3.3}$$

and

$$\dot{\boldsymbol{v}}(t) = \boldsymbol{L}\dot{\boldsymbol{e}}(t), \tag{3.4}$$

and then integrating $\dot{\boldsymbol{v}}$ to obtain $\boldsymbol{v}$. Section 3.2 addresses the question of how to find $\boldsymbol{K}$ and $\boldsymbol{L}$ (although it is not always possible to do so).

3.1.1 Alternating currents

In most energy monitoring applications, the voltages and currents are periodic with some period T. Thus it is useful to examine the Fourier transform of each quantity over one period. Define

$$\big(\mathcal{F}_y(f)\big)(t) = \sqrt{2}\int_0^1 f(t - T\tau)e^{2\pi y j\tau}\, d\tau, \tag{3.5}$$

so that $\mathcal{F}_y(f)(t)$ is the yth Fourier coefficient of f over the window from $t-T$ to t. The $\sqrt{2}$ normalization factor is chosen so that the magnitude of $\mathcal{F}_y(f)$ is equal to the RMS amplitude of the corresponding sinusoid.[1] This definition allows for continuous rolling windows; of course, the data rate of a signal may be dramatically reduced without significant loss of information by considering only non-overlapping windows at instants $t = zT$ for integers z.

Suppose that the first m harmonics are of interest and define the matrix-valued function

$$\big(\mathcal{F}(f)\big)(t) = \begin{bmatrix}\mathcal{F}_1(f)(t) & \cdots & \mathcal{F}_m(f)(t)\end{bmatrix}. \tag{3.6}$$

When $\mathcal{F}$ is applied to a column vector such as $\boldsymbol{i}$, the structure of its output is a matrix whose rows correspond to conductors and whose columns correspond to harmonic frequencies. This matrix is denoted by the corresponding capital letter, e.g.

$$\begin{aligned}\boldsymbol{I} &= \mathcal{F}(\boldsymbol{i})\\ \boldsymbol{S} &= \mathcal{F}(\boldsymbol{s})\\ \boldsymbol{V} &= \mathcal{F}(\boldsymbol{v})\\ \dot{\boldsymbol{V}} &= \mathcal{F}(\dot{\boldsymbol{v}})\\ \dot{\boldsymbol{E}} &= \mathcal{F}(\dot{\boldsymbol{e}}).\end{aligned}$$

For example, $\boldsymbol{I}(t)$ is an n-by-m matrix whose xyth entry is the yth harmonic current on the xth conductor: $I_{x,y}(t) = \mathcal{F}_y(i_x)(t)$.

Using the linearity of the Fourier transform, equation 3.1 implies that

$$\boldsymbol{S}(t) = \boldsymbol{M}\boldsymbol{I}(t) \tag{3.7}$$

[1] The equivalent definition for sampled signals is the discrete Fourier transform, i.e. the integral over time is simply replaced by an average over samples.

and equation 3.2 implies that

$$\dot{\boldsymbol{E}}(t) = \boldsymbol{N}\dot{\boldsymbol{V}}(t). \tag{3.8}$$

Using the time derivative property of the Fourier transform,

$$\dot{\boldsymbol{V}}(t) = \boldsymbol{V}(t) \cdot \frac{2\pi j}{T} \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & m \end{bmatrix}. \tag{3.9}$$

When working with phasors in the complex plane, the current $\boldsymbol{c}$, the power $\boldsymbol{p}$, and admittance $\boldsymbol{a}$ are all complex quantites with real and reactive components. The components of the power phasor are

$$p_x(t) = \overline{\mathcal{F}_1(i_x)(t)} \cdot \mathcal{F}_1(v_x)(t), \tag{3.10}$$

the components of the current phasor are

$$c_x(t) = \overline{\mathcal{F}_1(i_x)(t)} \cdot \frac{\mathcal{F}_1(v_x)(t)}{|\mathcal{F}_1(v_x)(t)|}, \tag{3.11}$$

and the components of the admittance phasor are

$$a_x(t) = \overline{\mathcal{F}_1(i_x)(t)} \cdot \frac{\mathcal{F}_1(v_x)(t)}{|\mathcal{F}_1(v_x)(t)|^2} = \overline{\mathcal{F}_1(i_x)(t)}/\overline{\mathcal{F}_1(v_x)(t)}. \tag{3.12}$$

The sign of the complex part, historically ambiguous, is chosen so that the power, current, and admittance phasors for a purely inductive load are positive imaginary numbers.

3.1.2 Spectral envelope preprocessor

Most power systems regulate the voltage on each conductor to be a sinusoid of period T with constant amplitude. For these systems, measurements of the power, current, or admittance provide equivalent information. In non-contact sensing applications, the preferred measurement quantity is the current, for the following reasons:

- It is easier to measure the current with non-contact sensors, as this does not require an absolute non-contact measurement of the magnitude of the voltage.

- Most power systems contain a mixture of constant-power loads (e.g. modern switching power supplies) and constant-admittance loads (e.g. linear circuit elements). If the voltage is not perfectly regulated, the power of a constant-admittance load varies quadratically with the voltage and the admittance of a constant-power load varies quadratically with the voltage. However, the currents of both sorts of load vary only linearly with changes in the voltage.

- Only the current phasor provides a clear generalization to the higher frequency harmonics (i.e. in the case of nonlinear loads that draw harmonic currents even when the voltage does not have significiant harmonics present).

In order to generalize the current phasor to higher harmonics, the phase of the voltage sinusoid is used to apply a rotation in the complex plane such that the resulting quantity is independent of the alignment of the transform window. The current phasors for higher harmonics are represented by the n-by-m matrix $\boldsymbol{C}$. This matrix is given by the elementwise definition

$$C_{x,y}(t) = \left(\frac{1}{j} \cdot \frac{V_{x,1}(t)}{|V_{x,1}(t)|}\right)^{y} \cdot \overline{\left(\frac{1}{j} \cdot I_{x,y}(t)\right)} \tag{3.13}$$

where $C_{x,y}(t)$ is the (x, y)th element of $\boldsymbol{C}$ representing the yth harmonic current phasor on the xth conductor, and $V_{x,1}(t)$ is the 1st harmonic voltage phasor on the xth conductor.

This matrix is characterized by the following properties:

- The magnitude of the current phasor is equal to the RMS amplitude of the corresponding sinusoidal current, i.e.
$$|C_{x,y}(t)| = |I_{x,y}(t)|.$$

- If the voltage on the xth conductor is given by
$$v_x(t) = \sin 2\pi t/T,$$
and current through the xth conductor is periodic given by
$$i_x(t) = \sum_{y=0}^{\infty} \big(r_y \sin(2\pi y t/T) - q_y \cos(2\pi y t/T)\big),$$
then the corresponding current phasors are independent of time and given by
$$C_{x,y}(t) = r_y + j q_y.$$
(This relationship is what necessitates the $1/j$ terms in the definition of $\boldsymbol{C}$.)

The process of computing $\boldsymbol{C}$ from $\boldsymbol{v}$ and $\boldsymbol{i}$ at a reduced sample rate is known as spectral envelope preprocessing [4]. Historically, load monitors would measure $\boldsymbol{v}$ and $\boldsymbol{i}$ directly and $\boldsymbol{C}$ would be computed using equation 3.5 followed by equation 3.13. With non-contact sensors, the "prep" algorithm must be extended with additional steps.

Matrices $\boldsymbol{K}$ and $\boldsymbol{L}$, as defined in equations 3.3 and 3.4, are determined in advance by the calibration procedure of section 3.2. The quantities $\dot{\boldsymbol{e}}(t)$ and $\boldsymbol{s}(t)$ are measured in real time by sensor hardware. The preprocessing algorithm proceeds as follows:

1. Using equations 3.3 and 3.4, compute $\dot{\boldsymbol{v}}(t)$ and $\boldsymbol{i}(t)$ from $\dot{\boldsymbol{e}}(t)$ and $\boldsymbol{s}(t)$.

2. Using equation 3.5, compute $\dot{\boldsymbol{V}}(t)$ and $\boldsymbol{I}(t)$ from the values of $\dot{\boldsymbol{v}}$ and $\boldsymbol{i}$ over the interval from $t - T$ to t.

3. Using equation 3.9, compute $\boldsymbol{V}(t)$ from $\dot{\boldsymbol{V}}(t)$.

4. Using equation 3.13, compute $\boldsymbol{C}(t)$ from $\boldsymbol{V}(t)$ and $\boldsymbol{I}(t)$.

Preprocessor implementations generally compute only the odd numbered harmonics of $\boldsymbol{C}$, because the vast majority of AC loads do not draw any current at even numbered harmonics.

Note that due to the linearity of the Fourier transform, the first two steps may be exchanged, i.e. first compute $\dot{\boldsymbol{E}}(t)$ and $\boldsymbol{S}(t)$ and then find $\dot{\boldsymbol{V}}(t)$ and $\boldsymbol{I}(t)$. Since the Fourier transform can result in a substantial bandwidth reduction, this is sometimes advantageous from a computational perspective.

For certain systems, such as variable speed drives for synchronous motors, the period T is not known in advance. If non-contact sensors are to be used with such a system, the "sinefit" algorithm of [4] must be used to determine the appropriate transform window before equation 3.5 is applied to the voltages or currents. In general, T is known in advance (for example, $T = 1/(60\,\mathrm{Hz})$ in American power distribution systems) and there is no need to apply the "sinefit" algorithm.

3.1.3 Equivalence of the Park transformation

Consider a system of balanced three-phase voltage distribution, i.e. let

$$\boldsymbol{v}(t) = \begin{bmatrix} \cos(2\pi t/T) \\ \cos(2\pi t/T - 2\pi/3) \\ \cos(2\pi t/T + 2\pi/3) \end{bmatrix}.$$

The Park transformation, also known as the DQ transformation, is the time-varying linear operator given by

$$\big(\hat{\mathcal{P}}(\boldsymbol{f})\big)(t) = \begin{bmatrix} \cos(2\pi t/T) & \sin(2\pi t/T) \\ -\sin(2\pi t/T) & \cos(2\pi t/T) \end{bmatrix} \cdot \begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{bmatrix} \cdot \boldsymbol{f}(t).$$

It is more convenient to think of the direct and quadrature terms of the Park transformation as the real and imaginary components of a single complex number, so let

$$\big(\mathcal{P}(\boldsymbol{f})\big)(t) = \begin{bmatrix} 1 & j \end{bmatrix} \cdot \big(\hat{\mathcal{P}}(\boldsymbol{f})\big)(t) = e^{-2\pi j t/T} \cdot \begin{bmatrix} 1 & e^{2\pi j/3} & e^{-2\pi j/3} \end{bmatrix} \cdot \boldsymbol{f}(t).$$

The defining characteristic of the Park transformation is that it is equal to a constant value when applied to a balanced three-phase quantity. For example,

$$\mathcal{P}(\boldsymbol{v})(t) = 3/2.$$

The Park transformation of the current vector is

$$\mathcal{P}(\boldsymbol{i})(t) = e^{-2\pi j t/T} \cdot \big(i_1(t) + e^{2\pi j/3} \cdot i_2(t) + e^{-2\pi j/3} \cdot i_3(t)\big). \tag{3.14}$$

This is the most common method for analyzing loads attached to three-phase distribution systems, so as non-contact sensing is extended to three-phase systems the question arises whether the Park transformation is related to the spectral envelope preprocessor of equation 3.13.

Consider the Fourier coefficients P_y of $\mathcal{P}(\boldsymbol{i})$, which are defined by

$$P_y(t) = \left(\frac{1}{j} \cdot \frac{V_{1,1}(t)}{|V_{1,1}(t)|}\right)^y \cdot \overline{\mathcal{F}_y(\mathcal{P}(\boldsymbol{i}))(t)} = e^{2\pi j y t/T} j^{-y} \cdot \overline{\mathcal{F}_y(\mathcal{P}(\boldsymbol{i}))(t)}. \tag{3.15}$$

Equation 3.15 is analogous to equation 3.13 in that it rotates by a power of the phase of the voltage in order to obtain a quantity which is independent of the alignment of the transform window. The signal $\mathcal{P}(\boldsymbol{i})$ over the interval from $t-T$ to t is completely characterized by its coefficients $P_y(t)$. (Since $\mathcal{P}(\boldsymbol{i})$ is a complex-valued signal with a DC component, the zero frequency and negative frequency coefficients must also be included.)

Equations 3.5, 3.13, and 3.14 are combined to obtain

$$\begin{aligned}\mathcal{F}_y(\mathcal{P}(\boldsymbol{i}))(t) &= e^{-2\pi jt/T}\cdot\left(I_{1,(y+1)}(t)+e^{2\pi j/3}I_{2,(y+1)}(t)+e^{-2\pi j/3}I_{3,(y+1)}(t)\right)\\ &= e^{2\pi jyt/T}j^{-y}\cdot\left(\overline{C_{1,(y+1)}(t)}+e^{-2\pi jy/3}\overline{C_{2,(y+1)}(t)}+e^{2\pi jy/3}\overline{C_{3,(y+1)}(t)}\right)\end{aligned}$$

which is subsituted into equation 3.15 to obtain

$$P_y(t)=C_{1,(y+1)}(t)+e^{2\pi jy/3}C_{2,(y+1)}(t)+e^{-2\pi jy/3}C_{3,(y+1)}(t). \tag{3.16}$$

This shows that the traditional analysis of three phase systems using the Park transformation provides the same information as the spectral envelope prepreprocessor applied separately to each current. Of particular interest, the DC component of the Park transformation is given by

$$P_0(t)=C_{1,1}(t)+C_{2,1}(t)+C_{3,1}(t).$$

The idea of treating the Park transformation as a complex number was first introduced in [17]. That paper also interprets the meanings of the higher harmonics.

3.2 Calibration

As specificed in equations 3.3 and 3.4, the most general goal of non-contact sensing is to find matrices $\boldsymbol{K}$ and $\boldsymbol{L}$ that will recover the currents and voltages from the sensed magnetic and electric fields. The process of determining these matrices is known as "calibration".

We begin by developing the calibration algorithm for DC systems that have an external path for return currents. The algorithm is then generalized to handle AC systems, systems without an external path for return currents, and three-phase delta-connected AC systems. Lastly, the algorithm is modified to use observation of *in situ* loads in place of a reference load.

3.2.1 DC calibration

This section considers the case of DC systems that have an unmonitored conductor to carry return currents. For example, most automobiles use 12 V DC distribution wires and return currents through the metal chassis. This section also assumes the use of a known reference load. The reference load is switched at a particular frequency and the demodulation scheme of [5] is used to distinguish it from any other loads which are present.

Recall that each sensor detects a mixture of the magnetic fields due to each current, so the sensor geometry determines a k-by-n matrix $\boldsymbol{M}$ satisfying equation 3.1. The goal of calibration is to find an n-by-k matrix $\boldsymbol{K}$ satisfying eqiatopm 3.3 using no information other than measurements of $\boldsymbol{s}$. Since there is no inherent ordering of the conductors, $\boldsymbol{M}$ is only

specified up to a permutation of its columns and $\boldsymbol{K}$ is only specified up to a permutation of its rows.

If $k < n$, such a $\boldsymbol{K}$ does not exist. This situation corresponds to an insufficient number of sensors, and is resolved by adding additional sensors. (It is also possible, although exceedingly unlikely, that $\boldsymbol{M}$ does not have rank n even when $k \geq n$. This "unlucky" case is also resolved by adding additional sensors. If $k \geq n$, matrix $\boldsymbol{K}$ is chosen to be the pseudoinverse of $\boldsymbol{M}$. This $\boldsymbol{K}$ has the smallest condition number of any left inverse of $\boldsymbol{M}$, so it minimizes the sensitivity of the unmixed currents to electromagnetic noise and physical perturbations. In general, the pseudoinverse of a matrix $\boldsymbol{M}$ will be denoted by $\boldsymbol{M}^+$.

The matrix $\boldsymbol{K} = \boldsymbol{M}^+$ is decomposed into a product $\boldsymbol{M}^+ = \boldsymbol{U}\boldsymbol{D}$ such that $\boldsymbol{U}$ is an invertible n-by-n matrix and $\boldsymbol{D}$ is an n-by-k matrix whose rows are orthonormal. Fig. 3.1 illustrates the behavior of this decomposition for the system of Fig. 4.3. To begin with, Fig. 3.1(i) depicts ten seconds of simulated sensor readings. Multiplication by $\boldsymbol{D}$ reduces the sensor readings from three dimensions to two, as shown in Fig. 3.1(iii). Finally, multiplication by $\boldsymbol{U}$ recovers the original conductor currents, as shown in Fig. 3.1(v).

Suppose that the reference load draws a current of β which is modulated at a particular frequency and duty cycle. Fig. 3.1(v) depicts a reference load with $\beta = 2\,\text{A}$ that is modulated at 2 Hz with a 75% duty cycle in the presence of background loads. When the reference load is attached to the xth conductor, it draws a modulated current of $\beta\hat{\boldsymbol{\imath}}_x$ (where $\hat{\boldsymbol{\imath}}_x$ denotes the xth basis current, i.e. the length-n vector with a 1 in the xth position and zeros everywhere else). The resulting magnetic field is $\boldsymbol{M} \cdot \beta\hat{\boldsymbol{\imath}}_x$—in other words, it is equal to β times the xth column of $\boldsymbol{M}$. Fig. 3.1(ii) depicts these magnetic field vectors for a 2 A reference load used with the system of Fig. 4.3.

In general, the demodulation algorithm of [5] is used to detect the presence of the reference load and determine the sensed magnetic fields which are due to each current that it draws. Suppose that p runs of the reference load are detected (where $p \geq n$) and that the demodulated sensor readings in the xth run are equal to $\boldsymbol{\sigma}_x$. If the reference load switches on at time t_x, then

$$\boldsymbol{\sigma}_x = \boldsymbol{s}(t_x + \epsilon) - \boldsymbol{s}(t_x - \epsilon)$$

for a sufficiently small value of ϵ. The demodulation algorithm is simply a more robust method of determining this quantity in the presence of other loads.

After the reference load has been attached to every conductor, the k-by-p matrix

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\sigma}_1 & \cdots & \boldsymbol{\sigma}_p \end{bmatrix} \tag{3.17}$$

is assembled and the eigendecomposition of the k-by-k matrix $\boldsymbol{\Sigma}\boldsymbol{\Sigma}'$ is computed. Because $\boldsymbol{\Sigma}\boldsymbol{\Sigma}'$ is Hermitian positive semidefinite, its eigenvalues are non-negative real numbers and its eigenvectors are orthonormal. Suppose that the eigendecomposition is given by

$$\boldsymbol{\Sigma}\boldsymbol{\Sigma}' = \begin{bmatrix} \boldsymbol{\rho}_1 & \cdots & \boldsymbol{\rho}_k \end{bmatrix} \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_k \end{bmatrix} \begin{bmatrix} \boldsymbol{\rho}_1' \\ \vdots \\ \boldsymbol{\rho}_k' \end{bmatrix} \tag{3.18}$$

where the $\boldsymbol{\rho}_x$ are orthonormal k-element column vectors and $\lambda_x \geq \lambda_{x+1}$.

Although the columns of $\mathbf{\Sigma}$ are k-dimensional vectors, they all correspond to valid sensor readings and so they all lie in an n-dimensional subspace defined by the image of $\boldsymbol{M}$. Therefore the rank of $\mathbf{\Sigma}$ is equal to n, and $\lambda_x = 0$ for $x > n$. In practice, any noise added to the sensor readings may increase these eigenvalues to be slightly greater then zero. The following theorem proves that the gap between the smallest nonzero eigenvalue and the largest zero eigenvalue is bounded by the signal-to-noise ratio of the sensors, so counting the nonzero eigenvalues of $\mathbf{\Sigma\Sigma'}$ is a robust method to determine n.

Theorem 1. *Given $\mathbf{\Sigma} = \boldsymbol{M}\mathbf{\Gamma} + \boldsymbol{N}\|\boldsymbol{M}\|\|\mathbf{\Gamma}\|$, where $\boldsymbol{M}$ and $\mathbf{\Gamma}$ have rank n, consider the k-by-k matrix $\mathbf{\Sigma\Sigma'}$. The n largest eigenvalues of $\mathbf{\Sigma\Sigma'}$ are at least*

$$\frac{1}{(2\|\boldsymbol{N}\| + \|\boldsymbol{N}\|^2)\kappa^2(\boldsymbol{M})\kappa^2(\mathbf{\Gamma})} - 1$$

times larger than any of the other $k - n$ eigenvalues.

$\boldsymbol{M}$ is assumed to have full rank, indicating that full reconstruction of the phase currents is possible, and $\mathbf{\Gamma}$ is assumed to have full rank, indicating that the calibration load was tested on every phase. $\boldsymbol{N}$ is a k-by-p matrix representing a small amount of measurement error. This theorem shows that counting the eigenvalues of $\mathbf{\Sigma\Sigma'}$ which are not close to zero is a robust way to determine the number of phases: when $\|\boldsymbol{N}\|$ is small, the ratio between any of the $k - n$ eigenvalues which are "close to zero" and any of the other n eigenvalues cannot exceed

$$2\|\boldsymbol{N}\|\kappa^2(\boldsymbol{M})\kappa^2(\mathbf{\Gamma}).$$

Proof. The matrix $\mathbf{\Sigma\Sigma'}$ is Hermitian positive semidefinite, so its eigenvalues $\lambda_1, \ldots, \lambda_k$ are real and non-negative. Write them in descending order, i.e.

$$\lambda_1 \geq \cdots \geq \lambda_k \geq 0.$$

The minimum ratio between any of $\lambda_1, \ldots, \lambda_n$ and any of $\lambda_{n+1}, \ldots, \lambda_k$ is exactly λ_n/λ_{n+1}.

Define

$$\boldsymbol{F} = \mathbf{\Sigma\Sigma'} - \boldsymbol{M}\mathbf{\Gamma\Gamma'}\boldsymbol{M'}$$

and let η denote its eigenvalue with the largest magnitude. Let ν denote the smallest nonzero eigenvalue of $\boldsymbol{M}\mathbf{\Gamma\Gamma'}\boldsymbol{M'}$. (This matrix is Hermitian positive semidefinite with rank n, so it has n positive eigenvalues and $k - n$ zero eigenvalues.) Weyl's inequalities on Hermitian matrices state that

$$\begin{aligned} \lambda_n &\geq \nu - |\eta| \\ \lambda_{n+1} &\leq |\eta| \end{aligned}$$

and assuming that $\nu > |\eta|$,

$$\frac{\lambda_n}{\lambda_{n+1}} \geq \frac{\nu}{|\eta|} - 1.$$

Because $\boldsymbol{M}$ and $\mathbf{\Gamma}$ have full rank,

$$\nu \geq \sigma^2_{\min}(\boldsymbol{M}) \cdot \sigma^2_{\min}(\mathbf{\Gamma}).$$

And using the properties of the matrix norm,

$$|\eta| = \|\boldsymbol{F}\| \leq (2\|\boldsymbol{N}\| + \|\boldsymbol{N}\|^2)\|\boldsymbol{S}\|^2\|\boldsymbol{\Gamma}\|^2.$$

Thus

$$\frac{\lambda_n}{\lambda_{n+1}} \geq \frac{1}{(2\|\boldsymbol{N}\| + \|\boldsymbol{N}\|^2)\kappa^2(\boldsymbol{M})\kappa^2(\boldsymbol{\Gamma})} - 1.$$

When $\|\boldsymbol{N}\|$ is small, this simplifies to

$$\frac{\lambda_{n+1}}{\lambda_n} \leq 2\|\boldsymbol{N}\|\kappa^2(\boldsymbol{M})\kappa^2(\boldsymbol{\Gamma}).$$

□

The eigendecomposition of $\boldsymbol{\Sigma}\boldsymbol{\Sigma}'$ also provides the matrix

$$\boldsymbol{D} = \begin{bmatrix}\boldsymbol{\rho}_1 & \cdots & \boldsymbol{\rho}_n\end{bmatrix}'. \tag{3.19}$$

Fig. 3.1(iv) illustrates that $\boldsymbol{D}$ projects the columns of $\boldsymbol{M}$ to an n-dimensional subspace while preserving their lengths and the angles between them. The following theorem proves that this choice of $\boldsymbol{D}$ is correct, so all that remains is to find the matrix $\boldsymbol{U}$ which completes the unmixing process.

Theorem 2. *Given $\boldsymbol{D}$ as defined in equation 3.19, there exists an n-by-n matrix $\boldsymbol{U}$ such that $\boldsymbol{M}^+ = \boldsymbol{U}\boldsymbol{D}$. Furthermore, $\boldsymbol{M}^+$ is the only left inverse of $\boldsymbol{M}$ which can be written as $\boldsymbol{U}\boldsymbol{D}$, and the choice of $\boldsymbol{U}$ is unique.*

Proof. If $\boldsymbol{U}\boldsymbol{D}$ is a left inverse of $\boldsymbol{M}$, then $\boldsymbol{U}\boldsymbol{D}\boldsymbol{M} = \boldsymbol{I}$ and

$$\boldsymbol{U} = (\boldsymbol{D}\boldsymbol{M})^{-1}.$$

$\boldsymbol{U}$ has full rank, so $\ker \boldsymbol{U}\boldsymbol{D} = \ker \boldsymbol{D}$. Recall that $\boldsymbol{\rho}_x$, the eigenvectors of $\boldsymbol{\Sigma}\boldsymbol{\Sigma}'$, are an orthonormal basis. Therefore

$$\boldsymbol{D}\boldsymbol{\rho}_x = 0 \iff x > n \iff \boldsymbol{\Sigma}\boldsymbol{\Sigma}'\boldsymbol{\rho}_i = 0$$

which implies

$$\ker \boldsymbol{D} = \ker \boldsymbol{\Sigma}\boldsymbol{\Sigma}'.$$

Matrix $\boldsymbol{M}$ has a left inverse, and $\boldsymbol{\Gamma}\boldsymbol{\Gamma}'$ is invertible, so

$$\ker \boldsymbol{\Sigma}\boldsymbol{\Sigma}' = \ker \boldsymbol{M}\boldsymbol{\Gamma}\boldsymbol{\Gamma}'\boldsymbol{M}' = \ker \boldsymbol{M}'.$$

Finally, it is a property of the pseudoinverse that

$$\ker \boldsymbol{M}' = \ker \boldsymbol{M}^+.$$

Therefore

$$\ker \boldsymbol{U}\boldsymbol{D} = \ker \boldsymbol{M}^+.$$

A left inverse of $\boldsymbol{M}$ is uniquely determined by its kernel, so

$$\boldsymbol{U}\boldsymbol{D} = (\boldsymbol{D}\boldsymbol{M})^{-1}\boldsymbol{D} = \boldsymbol{M}^+.$$

□

A spectral clustering algorithm [18] is used to group the vectors $\boldsymbol{\sigma}_x$ by conductor. The distance function d used by the clustering algorithm is the angle between the lines spanned by two reference load signatures, i.e.

$$d(\boldsymbol{\sigma}_x, \boldsymbol{\sigma}_y) = \arccos\left(\frac{\|\boldsymbol{\sigma}_x' \boldsymbol{\sigma}_y\|}{\|\boldsymbol{\sigma}_x\| \|\boldsymbol{\sigma}_y\|}\right). \tag{3.20}$$

Because $\boldsymbol{D}$ preserves the angles between reference signatures,

$$d(\boldsymbol{\sigma}_x, \boldsymbol{\sigma}_y) = d(\boldsymbol{D}\boldsymbol{\sigma}_x, \boldsymbol{D}\boldsymbol{\sigma}_y)$$

and the clustering can be performed in n-dimensional space. The elements of this space are expected to be clustered near columns of $\beta \cdot \boldsymbol{DM}$, as indicated by the dashed regions in Fig. 3.1(iv).

Suppose that the clustering algorithm partitions $\boldsymbol{D}\boldsymbol{\sigma}_1, \ldots, \boldsymbol{D}\boldsymbol{\sigma}_p$ into n clusters and selects a representative element $\boldsymbol{\delta}_x$ for the xth cluster. Because the xth cluster corresponds to the xth conductor, the reference load currents are given by

$$\beta \hat{\boldsymbol{i}}_x = \boldsymbol{U}\boldsymbol{\delta}_x.$$

This equation is solved for $\boldsymbol{U}$ to obtain

$$\boldsymbol{U} = \beta \begin{bmatrix} \boldsymbol{\delta}_1 & \cdots & \boldsymbol{\delta}_n \end{bmatrix}^{-1}. \tag{3.21}$$

Then

$$\boldsymbol{K} = \boldsymbol{U}\boldsymbol{D} \tag{3.22}$$

and calibration is finished. Fig. 3.1(vi) shows that multiplying $\beta\boldsymbol{M}$ by $\boldsymbol{UD}$ on the left recovers the original reference currents $\beta\hat{\boldsymbol{i}}_x$.

Algorithm 1 Calibration for direct currents with external path for return current.

Require: each reference load signature present in $\boldsymbol{s}$ is equal to β times a column of some matrix $\boldsymbol{M}$, and all columns of $\boldsymbol{M}$ are represented by reference load signatures.

Ensure: $\boldsymbol{K}$ is the pseudoinverse of $\boldsymbol{M}$, up to a permutation of its rows.

```
function CALIBRATE(s, β)
    σ_* ← FINDREFERENCELOADS(s)
    Σ ← [σ_1 ⋯ σ_p]
    λ_*, ρ_* ← EIGENDECOMPOSITION(ΣΣ')
    n ← COUNTNONZERO(λ_1, ..., λ_k)
    D ← [ρ_1 ⋯ ρ_n]'
    δ_* ← SPECTRALCLUSTER(Dσ_1, ..., Dσ_p)
    U ← β [δ_1 ⋯ δ_n]^{-1}
    K ← UD
    return K
end function
```

Algorithm 1 summarizes the method for determining $\boldsymbol{M}^+$ from $\boldsymbol{s}$ that was derived in this section. In summary, it is used in the following manner:

1. Attach a reference load which draws a constant current of β to each conductor of the instrumented cable in turn.

2. Call the function CALIBRATE($\boldsymbol{s}, \beta$), where $\boldsymbol{s}$ is a range of sensor data that includes all of the reference load runs. The result is the matrix $\boldsymbol{M}^+$.

3. To perform regular monitoring of currents, multiply the sensed magnetic field $\boldsymbol{s}(t)$ by $\boldsymbol{M}^+$ on the left to obtain the current $\boldsymbol{i}(t)$.

3.2.2 AC calibration

The calibration algorithm is next extended to the case of AC systems. The same algorithm that was developed for DC systems is applied to the Fourier transform of the AC sensor data. An important difference is that the Fourier transform is complex-valued and includes both magnitude and phase information. In this section, it is still assumed that an unmonitored conductor carries the return currents and that a modulated reference load is used for calibration. The AC reference load is a resistive device, i.e. when it is switched on it draws an alternating current that is in phase with the applied voltage.

The Fourier transform $\mathcal{F}$ is defined by

$$\big(\mathcal{F}_y(f)\big)(t) = \sqrt{2} \int_0^1 f(t - T\tau) e^{2\pi i j \tau} \, d\tau \tag{3.23}$$

where T denotes the period of the alternating current. In other words, $\mathcal{F}_y(f)$ is the yth Fourier coefficient of f over a sliding window with a length of one period. The normalization factor is chosen so that the magnitude of $\mathcal{F}_y(f)$ is equal to the RMS amplitude of the corresponding sinusoid. For example,

$$f(t) = \sin\left(\frac{2\pi t}{T}\right) \implies \big(\mathcal{F}_1(f)\big)(t) = \frac{j}{\sqrt{2}}.$$

For the purposes of calibration, we only consider the fundamental frequency components of the current phasor. Suppose that the phase of the voltage $v_x(t)$ on the xth conductor is θ_x, so that

$$v_x(t) = A_x \cos(2\pi t/T + \theta_x).$$

Then the current phasor $\boldsymbol{c}$ defined by equation 3.11 is also given by

$$\boldsymbol{c}(t) = e^{2\pi j t/T} \cdot \boldsymbol{\Theta} \cdot \overline{\mathcal{F}_1(\boldsymbol{i})(t)} \tag{3.24}$$

where

$$\boldsymbol{\Theta} = \begin{bmatrix} e^{j\theta_1} & & 0 \\ & \ddots & \\ 0 & & e^{j\theta_n} \end{bmatrix}.$$

It happens that $\boldsymbol{c}(t)$ is directly related to the Fourier transforms of the sensed magnetic fields. These transforms are given by

$$\boldsymbol{b}(t) = e^{2\pi j t/T} \cdot \overline{\mathcal{F}_1(\boldsymbol{s})(t)} \tag{3.25}$$

where $e^{2\pi jt/T}$ is a phase shift to compensate for the alignment of the transform window. Since $\mathcal{F}_1$ is a linear operator, equation (3.1) implies that

$$\overline{\mathcal{F}_1(\boldsymbol{s})(t)} = \boldsymbol{M} \cdot \overline{\mathcal{F}_1(\boldsymbol{i})(t)}. \tag{3.26}$$

Combine (3.24), (3.25), and (3.26) to obtain

$$\boldsymbol{b}(t) = \boldsymbol{M}\boldsymbol{\Theta}'\boldsymbol{c}(t) \tag{3.27}$$

and the inverse relation

$$\boldsymbol{c}(t) = \boldsymbol{\Theta}\boldsymbol{M}^{+}\boldsymbol{b}(t). \tag{3.28}$$

Equations (3.27) and (3.28) are analogous to (3.1) and (3.3) from the DC case.

In order to compute $\boldsymbol{b}$ from $\boldsymbol{s}$, it is necessary to deduce $e^{2\pi jt/T}$ from measurements of the conductor voltages. (This prevents the inevitable problem of clock skew between the supposed time t and the actual phases of the voltages.) Suppose that a capacitively-coupled non-contact voltage sensor [5] is used to measure an arbitrary mixture $v_m(t)$ of the conductor voltages. Since the the time t may be shifted by any constant factor, suppose without loss of generality that $v_m(t)$ is a "zero phase" signal, i.e.

$$\frac{\mathcal{F}_1(v_m)(t)}{|\mathcal{F}_1(v_m)(t)|} = e^{2\pi jt/T}. \tag{3.29}$$

Equations (3.25) and (3.29) are combined to obtain

$$\boldsymbol{b}(t) = \frac{\mathcal{F}_1(v_m)(t)}{|\mathcal{F}_1(v_m)(t)|} \cdot \overline{\mathcal{F}_1(\boldsymbol{s})(t)} \tag{3.30}$$

which allows $\boldsymbol{b}(t)$ to be determined without the need for a synchronized clock.

With this framework in place, an AC system is easily calibrated using algorithm 1:

1. Attach a reference load which draws a constant-amplitude sinusoidal current of β (in phase with the voltage) to each conductor of the instrumented cable in turn.

2. Use (3.30) to compute $\boldsymbol{b}$ over an interval of time which includes all runs of the reference load.

3. Call the function CALIBRATE($\boldsymbol{b}, \beta$). The result is the matrix $\boldsymbol{\Theta}\boldsymbol{M}^{+}$.

4. To perform regular monitoring of currents, compute $\boldsymbol{b}(t)$ from $\boldsymbol{s}(t)$ using (3.30). Then multiply by $\boldsymbol{\Theta}\boldsymbol{M}^{+}$ on the left to obtain the desired output $\boldsymbol{c}(t)$.

In other words, the DC calibration procedure seamlessly handles AC phase shifts when it is applied to complex-valued signals.

Lastly, in order to fit the result of AC calibration into the framework of the spectral envelope preprocessor, suitable matrices $\boldsymbol{K}$ and $\boldsymbol{L}$ must be determined. Suppose that the reference voltage was determined by $v_r(t) = \boldsymbol{l}\boldsymbol{e}(t)$ for some row vector $\boldsymbol{l}$. The desired matrices are then given by

$$\boldsymbol{K} = \boldsymbol{M}^{+}$$

and

$$\boldsymbol{L} = \begin{bmatrix} e^{j\theta_1} \\ \vdots \\ e^{j\theta_n} \end{bmatrix} \cdot \boldsymbol{l}.$$

As per [5], the matrix $\boldsymbol{\Theta M}^+$ produced by the calibration procedure can only be decomposed into $e^{j\theta_x}$ and $\boldsymbol{M}^+$ up to a possible sign error in each $e^{j\theta_x}$ and the corresponding row of $\boldsymbol{M}^+$. However, the potential sign errors cancel whenever an odd-numbered harmonic of the current phasor is computed.

3.2.3 Return currents

This section extends the DC and AC calibration algorithms to the common case where the currents through a multiple-conductor cable are required to sum to zero. For example, in residential AC distribution systems, any current drawn through one of the line conductors is returned through the neutral conductor. The reference load now draws a current from one conductor and returns it through a second conductor.

We begin by considering the DC case. Suppose that $i_1(t), \ldots, i_{n-1}(t)$ are the supply currents and $i_n(t)$ is the return current. The reduced-length current vector is defined by

$$\boldsymbol{i}_r(t) = \begin{bmatrix} i_1(t) \\ \vdots \\ i_{n-1}(t) \end{bmatrix}.$$

and includes the supply currents but not the return current. Using the constraint that $i_1(t) + \cdots + i_n(t) = 0$,

$$\boldsymbol{i}(t) = \boldsymbol{H}\boldsymbol{i}_r(t) \tag{3.31}$$

where

$$\boldsymbol{H} = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \\ -1 & \cdots & -1 \end{bmatrix}.$$

Combine (3.1) and (3.31) to obtain

$$\boldsymbol{s}(t) = \boldsymbol{M}\boldsymbol{H}\boldsymbol{i}_r(t) \tag{3.32}$$

and the inverse relation

$$\boldsymbol{i}_r(t) = (\boldsymbol{M}\boldsymbol{H})^+\boldsymbol{s}(t). \tag{3.33}$$

This is exactly the setup needed to apply algorithm 1:

1. Attach a reference load which draws a constant current of β to each supply conductor in turn. The return conductor always returns a current of $-\beta$.

2. Call the function CALIBRATE($\boldsymbol{s}, \beta$), where $\boldsymbol{s}$ is a range of sensor data that includes all of the reference load runs. The result is the matrix $(\boldsymbol{M}\boldsymbol{H})^+$.

3. To perform regular monitoring of currents, multiply the sensed magnetic fields $\boldsymbol{s}(t)$ by $(\boldsymbol{MH})^+$ on the left to obtain $\boldsymbol{i}_r(t)$.

The only difference when a return conductor is present is that the result of calibration is $\boldsymbol{i}_r$ instead of $\boldsymbol{i}$.

The method is similar for the AC case. Analogous to (3.24), define the reduced-length rotated currents

$$\boldsymbol{c}_r(t) = e^{2\pi jt/T} \cdot \boldsymbol{\Theta} \cdot \overline{\mathcal{F}_1(\boldsymbol{i}_r)(t)}. \tag{3.34}$$

Combine (3.25), (3.32), and (3.34) to obtain

$$\boldsymbol{b}(t) = \boldsymbol{MH\Theta}'\boldsymbol{c}_r(t) \tag{3.35}$$

and the inverse relation

$$\boldsymbol{c}_r(t) = \boldsymbol{\Theta}(\boldsymbol{MH})^+\boldsymbol{b}(t). \tag{3.36}$$

Once again, we apply algorithm 1:

1. Attach a reference load which draws a constant-amplitude current of β (in phase with the voltage) to each line conductor in turn. The neutral conductor always returns a current of amplitude β that is 180 degrees out of phase with the line voltage.

2. Use (3.30) to compute $\boldsymbol{b}$ over an interval of time which includes all runs of the reference load.

3. Call the function CALIBRATE$(\boldsymbol{b}, \beta)$. The result is the matrix $\boldsymbol{\Theta}(\boldsymbol{MH})^+$.

4. To perform regular monitoring of currents, compute $\boldsymbol{b}(t)$ from $\boldsymbol{s}(t)$ using (3.30). Then multiply by $\boldsymbol{\Theta}(\boldsymbol{MH})^+$ on the left to obtain $\boldsymbol{c}_r(t)$.

The only difference when a return conductor is present is that the result of calibration is $\boldsymbol{c}_r$ instead of $\boldsymbol{c}$.

3.2.4 AC delta-connected systems

In the special case of AC delta-connected power distribution systems, the conductor currents are required to sum to zero, but there is no designated return conductor and none of the conductors are at zero potential. A reference load must be attached between two line conductors, and draws a current that is in phase with the difference between the two voltages but out of phase with either of the individual voltages.

For example, consider a three-phase system. The voltages on all three conductors have the same amplitude, but the voltages on any pair of conductors are separated in phase by 120 degrees. Suppose that ϕ_x is the phase of the voltage signal $v_x(t)$, θ_1 is the phase of the difference $v_1(t) - v_3(t)$, and θ_2 is the phase of the difference $v_2(t) - v_3(t)$.

A reference load is first attached between conductors 1 and 3, and then between conductors 2 and 3. The previous algorithm for AC systems produces $\boldsymbol{c}_r$ according to (3.34), where

$$\boldsymbol{\Theta} = \begin{bmatrix} e^{j\theta_1} & 0 \\ 0 & e^{j\theta_2} \end{bmatrix}.$$

However, the desired result in this special case is

$$\boldsymbol{c}(t) = e^{2\pi jt/T} \cdot \boldsymbol{\Phi} \cdot \overline{\mathcal{F}_1(\boldsymbol{i})(t)} \tag{3.37}$$

where

$$\boldsymbol{\Phi} = \begin{bmatrix} e^{j\phi_1} & 0 & 0 \\ 0 & e^{j\phi_2} & 0 \\ 0 & 0 & e^{j\phi_3} \end{bmatrix}.$$

Equations (3.31), (3.34), and (3.37) are combined to obtain

$$\boldsymbol{c}(t) = \boldsymbol{\Phi}\boldsymbol{H}\boldsymbol{\Theta}' \cdot \boldsymbol{c}_r(t). \tag{3.38}$$

It follows from (3.36) and (3.38) that

$$\boldsymbol{c}(t) = \boldsymbol{\Phi}\boldsymbol{H}(\boldsymbol{M}\boldsymbol{H})^+ \cdot \boldsymbol{b}(t). \tag{3.39}$$

Therefore the special goal of three-phase delta calibration is to determine the matrix $\boldsymbol{\Phi}\boldsymbol{H}\boldsymbol{\Theta}'$, which is then multiplied by $\boldsymbol{\Theta}(\boldsymbol{M}\boldsymbol{H})^+$ (the result of the previous AC calibration algorithm) to obtain the matrix $\boldsymbol{\Phi}\boldsymbol{H}(\boldsymbol{M}\boldsymbol{H})^+$ which recovers $\boldsymbol{c}(t)$ from $\boldsymbol{b}(t)$.

In a three-phase system, the voltages on any pair of conductors have the same amplitude but are separated in phase by 120 degrees. Thus there are only two possibilities for the phase relationships between the three voltages:

$$\begin{aligned} e^{j\phi_1}/e^{j\phi_3} &= e^{\pm 2\pi j/3} \\ e^{j\phi_2}/e^{j\phi_3} &= e^{\mp 2\pi j/3} \\ e^{j\theta_1}/e^{j\phi_3} &= e^{\pm 5\pi j/6} \\ e^{j\theta_2}/e^{j\phi_3} &= e^{\mp 5\pi j/6}. \end{aligned}$$

and

$$\boldsymbol{\Phi}\boldsymbol{H}\boldsymbol{\Theta}' = \begin{bmatrix} e^{\pm \pi j/6} & 0 \\ 0 & e^{\mp \pi j/6} \\ e^{\mp \pi j/6} & e^{\pm \pi j/6} \end{bmatrix}. \tag{3.40}$$

All that remains is to determine which signs the exponents take.

Observe that $\boldsymbol{\Phi}\boldsymbol{H}(\boldsymbol{M}\boldsymbol{H})^+$ is the product of a diagonal matrix $\boldsymbol{\Phi}$ and a real-valued matrix $\boldsymbol{H}(\boldsymbol{M}\boldsymbol{H})^+$. Thus every row of $\boldsymbol{\Phi}\boldsymbol{H}(\boldsymbol{M}\boldsymbol{H})^+$ is equal to a complex scalar times a real row vector. However, if the incorrect choice of $\boldsymbol{\Phi}\boldsymbol{H}\boldsymbol{\Theta}'$ is made, the last row of the incorrect $\boldsymbol{\Phi}\boldsymbol{H}(\boldsymbol{M}\boldsymbol{H})^+$ *cannot* be expressed as a complex scalar times a real row vector. This provides a mechanism for deducing the correct value of $\boldsymbol{\Phi}\boldsymbol{H}\boldsymbol{\Theta}'$.

The following theorem shows that the function

$$r(\boldsymbol{w}) = 1 - \left| \sum_{x=1}^{k} \frac{w_x^2}{\|\boldsymbol{w}\|^2} \right|^2 \tag{3.41}$$

is equal to 0 if and only if the vector $\boldsymbol{w}$ has elements w_x which are all real multiples of a single complex number, and increases with the angle between the vector's elements in the complex plane. The function r is applied to the two candidates for the bottom row of $\boldsymbol{\Phi}\boldsymbol{H}(\boldsymbol{M}\boldsymbol{H})^+$, and whichever one is closer to zero indicates the correct choice of $\boldsymbol{\Phi}\boldsymbol{H}\boldsymbol{\Theta}'$.

Theorem 3. *The expression $r(\boldsymbol{w})$ as defined in equation 3.41 is equal to*

$$2\sum_{x=1}^{k}\sum_{y=1}^{k}\left|\text{Imag}\left(\frac{w_x}{\|\boldsymbol{w}\|}\cdot\frac{\overline{w_y}}{\|\boldsymbol{w}\|}\right)\right|^2. \tag{3.42}$$

Consequently, $r(\boldsymbol{w})$ is equal to 0 if and only if any two components w_x and w_y point in the same direction in the complex plane. Furthermore, $r(\boldsymbol{w})$ is invariant when $\boldsymbol{w}$ is multiplied by any complex scalar or orthogonal matrix.

Proof. Clearly $r(\boldsymbol{w})$ does not change when $\boldsymbol{w}$ is multiplied by a positive real scalar, so suppose without loss of generality that $\|\boldsymbol{w}\| = 1$. Equation 3.42 may be written using the Frobenius norm as

$$\begin{aligned}
r(\boldsymbol{w}) &= \frac{1}{2}\|\boldsymbol{w}'\boldsymbol{w} - \overline{\boldsymbol{w}'\boldsymbol{w}}\|_F^2 \\
&= \frac{1}{2}\text{trace}\big((\boldsymbol{w}'\boldsymbol{w} - \overline{\boldsymbol{w}'\boldsymbol{w}})(\boldsymbol{w}'\boldsymbol{w} - \overline{\boldsymbol{w}'\boldsymbol{w}})'\big) \\
&= \text{trace}(\boldsymbol{w}'\boldsymbol{w}\boldsymbol{w}'\boldsymbol{w}) - \text{trace}(\boldsymbol{w}'\boldsymbol{w}\overline{\boldsymbol{w}'\boldsymbol{w}}) \\
&= (\boldsymbol{w}\boldsymbol{w}')(\boldsymbol{w}\boldsymbol{w}') - (\boldsymbol{w}\overline{\boldsymbol{w}}')(\overline{\boldsymbol{w}}\boldsymbol{w}') \\
&= 1 - \|\boldsymbol{w}\overline{\boldsymbol{w}}'\|^2.
\end{aligned}$$

As desired, the last line is equivalent to equation 3.41.

When $\boldsymbol{w}$ is multiplied by a unit magnitude complex scalar c,

$$r(c\boldsymbol{w}) = 1 - \|c^2\boldsymbol{w}\overline{\boldsymbol{w}}'\|^2 = 1 - \|\boldsymbol{w}\overline{\boldsymbol{w}}'\|^2 = r(\boldsymbol{w}).$$

When $\boldsymbol{w}$ is multiplied by an orthognal matrix $\boldsymbol{O}$,

$$r(\boldsymbol{w}\boldsymbol{O}) = 1 - \|\boldsymbol{w}\boldsymbol{O}\overline{\boldsymbol{O}}'\overline{\boldsymbol{w}}'\|^2 = 1 - \|\boldsymbol{w}\overline{\boldsymbol{w}}'\|^2 = r(\boldsymbol{w}).$$

This derivation relied on $\overline{\boldsymbol{O}} = \boldsymbol{O}$, so r is not necessarily invariant under multiplication by a unitary matrix. □

In summary, calibrating a three-phase delta-connected system proceeds as follows:

1. Attach a reference load which draws a constant-amplitude current of β (in phase with the applied voltage) between conductors 1 and 3 and then between conductors 2 and 3.
2. Use (3.30) to compute $\boldsymbol{b}$ over an interval of time which includes all runs of the reference load.
3. Call the function CALIBRATE($\boldsymbol{b}, \beta$). The result is the matrix $\boldsymbol{\Theta}(\boldsymbol{M}\boldsymbol{H})^+$.
4. Use (3.40) and (3.41) to determine the correct value of the matrix $\boldsymbol{\Phi}\boldsymbol{H}\boldsymbol{\Theta}'$.
5. Form the product
$$\boldsymbol{\Phi}\boldsymbol{H}(\boldsymbol{M}\boldsymbol{H})^+ = \boldsymbol{\Phi}\boldsymbol{H}\boldsymbol{\Theta}' \cdot \boldsymbol{\Theta}(\boldsymbol{M}\boldsymbol{H})^+.$$
6. To perform regular monitoring of currents, compute $\boldsymbol{b}(t)$ from $\boldsymbol{s}(t)$ using (3.30). Then multiply by $\boldsymbol{\Phi}\boldsymbol{H}(\boldsymbol{M}\boldsymbol{H})^+$ on the left to obtain $\boldsymbol{c}(t)$.

If (3.40) and (3.41) do not clearly indicate which is the correct value of $\boldsymbol{\Phi}\boldsymbol{H}\boldsymbol{\Theta}'$, then the initial assumption of symmetric three-phase power distribution was incorrect.

3.2.5 Eliminating the reference load

In some cases, it is not feasible to attach a special calibration device to each conductor. In a typical energy monitoring appplication, most of the calibration process can be carried out "implicitly" using only the standard electrical devices which are already attached to the conductors. The former requirement that each $\boldsymbol{\sigma}_x$ is equal to β times a column of $\boldsymbol{M}$ is relaxed to allow each $\boldsymbol{\sigma}_x$ to be an arbitrary scalar times a column of $\boldsymbol{M}$. Thus each $\boldsymbol{\sigma}_x$ can be the change in magnetic field due to an arbitrary load switching on, rather than just the change in magnetic field due to a known reference load switching on.

However, the demodulation scheme is no longer applicable for separating the magnetic field due to a particular load from the magnetic fields due to background loads which are operating simultaneously. Instead, $\boldsymbol{s}(t)$ is passed through a high-pass filter and the local extrema of the resulting signal are adopted as the new $\boldsymbol{\sigma}_x$. Since it is very unlikely for independent loads attached to different conductors to switch on or off at exactly the same instant, these values of $\boldsymbol{\sigma}_x$ indeed represent separate columns of $\boldsymbol{M}$. The revised calibration procedure is given in algorithm 2.

Algorithm 2 Implicit calibration (without a reference load) for direct currents with external path for return current.

Require: the majority of step changes present in $\boldsymbol{s}$ are scaled columns of some matrix $\boldsymbol{M}$, and all columns of $\boldsymbol{M}$ are represented by step changes.

Ensure: $\boldsymbol{K}$ is the pseudoinverse of $\boldsymbol{M}$, up to a permutation of its rows and constant scaling factor applied to each row.

function ImplicitCalibrate($\boldsymbol{s}$, β)
  $\hat{\boldsymbol{s}} \leftarrow$ HighPassFilter($\boldsymbol{s}$)
  $\boldsymbol{\sigma}_* \leftarrow$ FindLocalExtrema($\hat{\boldsymbol{s}}$)
  $\boldsymbol{\Sigma} \leftarrow \begin{bmatrix}\boldsymbol{\sigma}_1 & \cdots & \boldsymbol{\sigma}_p\end{bmatrix}$
  $\lambda_*, \boldsymbol{\rho}_* \leftarrow$ Eigendecomposition($\boldsymbol{\Sigma}\boldsymbol{\Sigma}'$)
  $n \leftarrow$ CountNonzero($\lambda_1, \ldots, \lambda_k$)
  $\boldsymbol{D} \leftarrow \begin{bmatrix}\boldsymbol{\rho}_1 & \cdots & \boldsymbol{\rho}_n\end{bmatrix}'$
  $\boldsymbol{\delta}_* \leftarrow$ SpectralCluster($\boldsymbol{D}\boldsymbol{\sigma}_1, \ldots, \boldsymbol{D}\boldsymbol{\sigma}_p$)
  $\boldsymbol{U} \leftarrow \beta \begin{bmatrix}\boldsymbol{\delta}_1 & \cdots & \boldsymbol{\delta}_n\end{bmatrix}^{-1}$
  $\boldsymbol{K} \leftarrow \boldsymbol{U}\boldsymbol{D}$
  return $\boldsymbol{K}$
end function

Implicit calibration differs from standard calibration in that (i) p may be much larger than n, and (ii) the vectors $\boldsymbol{D}\boldsymbol{\sigma}_x$ in a cluster may have different magnitudes. The former is not important because the eigendecomposition and clustering algorithms scale well to larger datasets. The latter means that $\boldsymbol{U}$ and $\boldsymbol{K}$ can only be determined up to a constant scaling factor multiplying each row. However, in any building that is outfitted with a low-bandwidth utility-provided power meter, the non-contact sensor measurements may be compared with the utility's power measurements over a longer period of time in order to determine the unknown scaling factors.

3.3 Signal acquisition

This thesis provides two production-level software pipelines for acquiring data from the non-contact sensors and applying the spectral envelope preprocessor. The first pipeline is integrated with the existing NilmDB preprocessor [4], meaning that it is modular and flexible but also computationally demanding. The second pipeline is a separate software stack which vertically integrates all stages of data acquisition and preprocessing, so that only the final spectral envelope data is inserted into NilmDB. The second pipeline is roughly ten times faster, but it is less transparent because intermediate results are not stored to disk. A third pipeline, used primarily for debugging, is entirely separate from NilmDB.

3.3.1 Integrated acquisition pipeline

The integrated acqusition system begins with a Python script called `leebomatic.py`, which streams data from the digital interface PCB (section 2.1.3) and writes it to standard output for use with the existing `nilm-insert` command. This script is given in appendix E.2.

The raw sensor data is then processed by a new filter, called `nilm-matrix`, which is integrated into the existing nilmtools framework. The `nilm-matrix` filter is a Swiss army knife of non-contact data processing. It has the following capabilities:

- Multiply some subset of the columns of the input data by an arbitrary matrix.
- Pass the result through the linear integrating filter described in section C.2.2.

To recover the currents from the magnetic fields, only matrix multiplication is used. To recover the voltages from the derivatives of the electric fields, matrix multiplication and integration are used.

The voltages and currents output by `nilm-matrix` are then processed by the existing `nilm-sinefit` and `nilm-prep` commands as described in [4].

3.3.2 High speed aquisition pipeline

The high speed acquistion system uses a monolithic C program to stream data from the sensors and compute the Fourier transform. This data is then streamed to a Python program which applies equation 3.13 to compute the spectral envelopes, and inserts them into NilmDB.

The C program is called `capture.c` and is given in appendix E.1. It takes two command line arguments. The first is the name of the file or pipe to which its output should be written. The second is the path to the serial port from which input should be read. Input data from the sensors ($\dot{\boldsymbol{e}}$ and $\boldsymbol{s}$) is stored in a circular buffer whose length is one line cycle. The "Fastest Fourier Transform in the West" library is used to compute $\dot{\boldsymbol{E}}$ and $\boldsymbol{S}$ in real time. The output data is written to a large in-memory circular buffer so that data acquisition does not stall if the output pipe is occasionally blocked by the Python program.

The Python program is called `prep.py` and handles the remainder of the spectral envelope preprocessing algorithm. It is set up to run as a wrapper around `capture.c`, and automatically creates a pipe to stream data from the C program and executes flow control

as needed. The Python program streams in samples of $\dot{\boldsymbol{E}}$ and $\boldsymbol{S}$, computes $\boldsymbol{C}$, and inserts $\boldsymbol{C}$ into NilmDB. The unmixing matrices $\boldsymbol{K}$ and $\boldsymbol{L}$ are defined at the top of the `prep.py`. A future implementation of the calibration procedure should write $\boldsymbol{K}$ and $\boldsymbol{L}$ to a configuration file which would then be read by `prep.py`.

3.3.3 Standalone Python pipeline

There is a third pipeline which does all computation in Python and is not integrated with NilmDB at all. This pipeline is generally only useful for debugging as the Python scripts are too computationally-intensive for use on a low-powered computer. The system includes three files: `adc.py` streams raw data from the sensors, `capture.py` computes the Fourier transforms, and `scope.py` provides a live "oscilloscope" display of the incoming sensor data. These scripts are provided in appendix E.3.

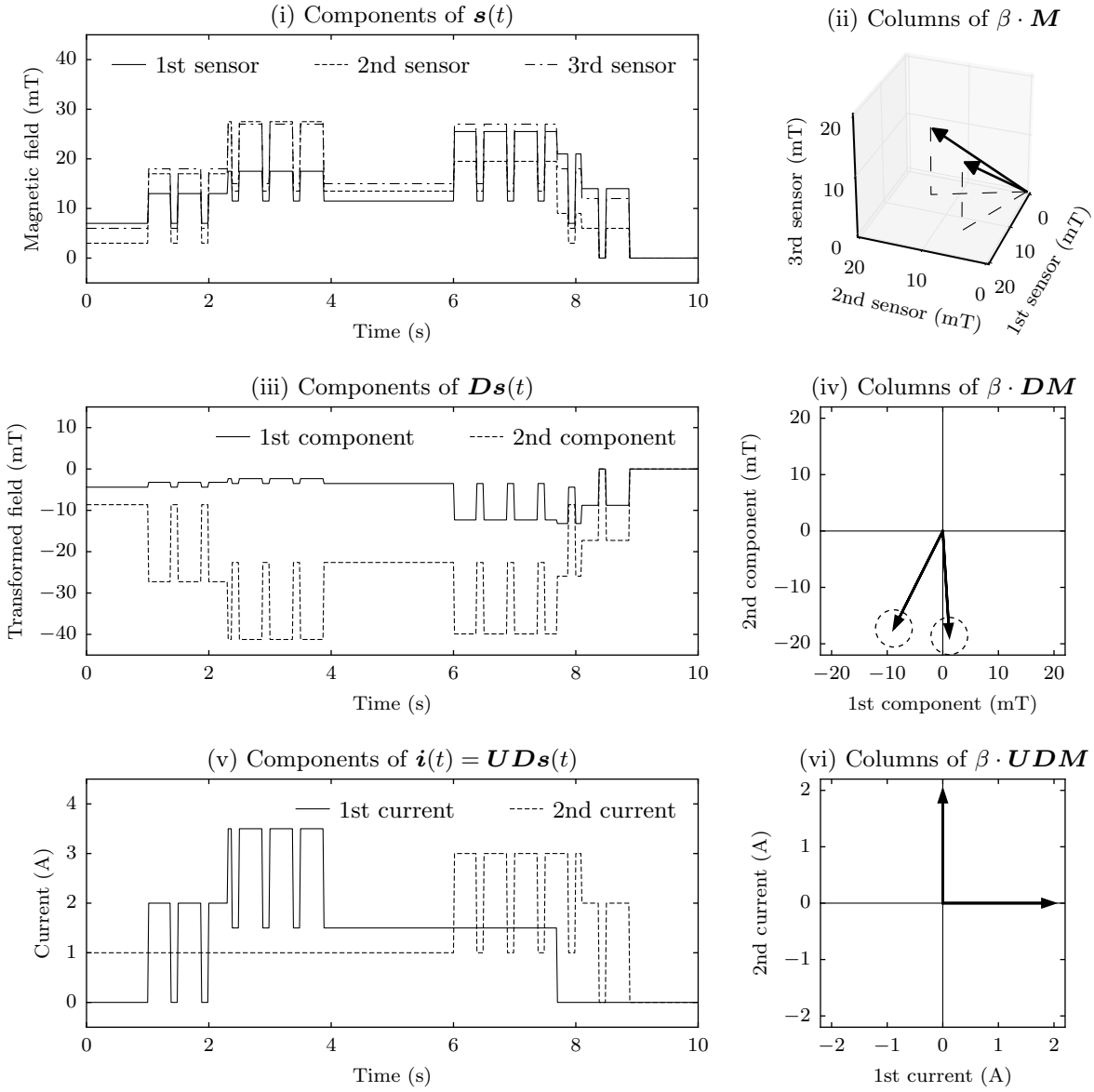


Figure 3.1: Graphical depiction of the unmixing procedure proceeding simultaneously in the time domain and in vector space. As depicted in Fig. 4.3, there are three sensors and two conductors. The reference load draws a current of $\beta = 2\,\mathrm{A}$ modulated at $2\,\mathrm{Hz}$ on the first conductor for $1 < t < 4$ and on the second conductor for $6 < t < 9$. There is an additional current of $\beta = 1.5\,\mathrm{A}$ on the first conductor for $2.3 < t < 7.7$ and an additional current of $1\,\mathrm{A}$ on the second conductor for $t < 8.1$.

Chapter 4

Load Monitoring Applications

This chapter presents an assortment of topics relevant to the practical applications of non-contact, non-intrusive load monitoring. First, we survey common building electrical faults and consider their interaction with non-contact senors. Second, a system for detecting transient events in preprocessed data is presented. Third, we share various examples and results demonstrating the usage of non-contact sensors.

4.1 Electrical faults

Various types of electrical issues are consistently detectable by non-contact sensors. In some cases, pre-existing electrical faults are detected during calibration. More typically, pre-existing faults are not detectable but a new fault which arises after calibration is immediately detected (and it may even be possible to determine its cause).

4.1.1 Non-contact issues

The first class of problems considered are not true electrical faults. Instead, they are problems related to improper installation of the non-contact sensors. These problems are revealed during calibration.

If the calibration procedure determines that $k = n$ (i.e. the number of sensors is equal to the number of conductors), then it is possible that the number of sensors was insufficient. It is usually advisable to increase the number of sensors to be strictly greater than the largest expected number of conductors, which rules out this situation completely. Otherwise, when the first stage of calibration reports that $k = n$, the clustering algorithm may be used to infer whether there were actually more than k conductors. If the detected calibration load signatures divide cleanly into k clusters, then there were k conductors. However, if the clustering algorithm obtains a better result for some number of clusters greater than k, then that number is the true n.

When the mapping from calibration load runs to the conductors under measurement is unknown (for example, in a building where it is not known which outlets are on which phase), it is possible to detect whether any phases have been missed. The standard calibration procedure and the implicit calibration procedure are both run over the same interval of

time. If the implicit calibration procedure reports a larger number of conductors, than there were conductors with loads turning on and off which were never attached to the calibration load. This situation was once observed in an apartment complex where the neighboring units injected current into a loop through neutral and earth ground, registering as an extra phase which was never calibrated.

4.1.2 Phase shifts

The next problem considered is relevant to non-contact calibration with alternating currents. Diagnostic information is obtained by studying the interaction of the standard calibration procedure with complex-valued signals. This additional information is available because the unmixing matrix is not an arbitrary complex-valued matrix. Instead, it is the product of a diagonal matrix $\boldsymbol{\Theta}$ and a real matrix $\boldsymbol{M}^+$.

Because $\boldsymbol{\Gamma}$ commutes with Θ, the matrix $\boldsymbol{\Sigma\Sigma'} = \boldsymbol{M\Theta'\Gamma\Gamma'\Theta M}$ is real, and therefore $\boldsymbol{D}$ is real. The matrix $\boldsymbol{U}$ is then expected to have rows which are decomposable as the product of a complex scalar and a real row vector. If either of these requirements is violated during calibration, then the assumption that $\boldsymbol{b}(t) = \boldsymbol{M\Theta'c}(t)$ was incorrect. This can happen if the voltages on the various conductors are not synchronous or if the length of the transform window is not the same as the period of the alternating voltages.

An electrical fault has been observed which causes the same symptoms. When an appliance in a building introduces a short circuit between the neutral conductor and the safety ground conductor, return currents from the appliance are split between neutral and safety ground. These connections typically have different complex impedances, which results in a phase shift in both of the return currents. Depending on the relative mixture of return currents detected by each non-contact sensor, the unmixing matrix can have a different phase shift applied to each column.

4.1.3 Orthogonal components

After calibration has concluded, a powerful method is capable of detecting almost anything that might go wrong. This method requires the number of sensors to exceed the number of currents and works better the more sensors there are. The basic idea is that not all combinations of sensor readings are possible—only vectors in the image of $\boldsymbol{M}$ can be explained by the sensing model. This is an n-dimensional subspace of the k-dimensional sensor space. If the sensors ever report values which are outside of this subspace, an unmodeled effect must be taking place.

A fundamental result of linear algebra states that the vector $\boldsymbol{s}(t)$ may be additively decomposed into two components $\boldsymbol{s}^{\parallel}(t)$ and $\boldsymbol{s}^{\perp}(t)$ such that $\boldsymbol{s}^{\parallel}$ is in the image of $\boldsymbol{M}$ and $\boldsymbol{s}^{\perp}$ is orthogonal to the image of $\boldsymbol{M}$, i.e. it is in the kernel of $\boldsymbol{M}^+$. These components are given by

$$\boldsymbol{s}^{\parallel}(t) = \boldsymbol{M}\boldsymbol{M}^+\boldsymbol{s}(t)$$

and

$$\boldsymbol{s}^{\perp}(t) = (\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^+)\boldsymbol{s}(t).$$

The magnitude $\|\boldsymbol{s}^{\perp}(t)\|$ is an indicator of how large the unmodeled effects are at time t. This magnitude may be computed using the `nilm-haunting` tool (a part of NilmDB), which is given in appendix E.2.

There are several reasons that $\|\boldsymbol{s}^{\perp}(t)\|$ might become nonzero. These can often be distinguished by examining its behavior over a longer period of time and its correlation with the reconstructed currents $\boldsymbol{i}(t)$, as described in the following partial list of unmodeled effects:

- Physical perturbation of the sensor locations results in $\|\boldsymbol{s}^{\perp}(t)\|$ which is proportional to the reconstructed currents, beginning at the instant at which the sensors are perturbed and continuing forever (until the system is re-calibrated).

- An exogenous magnetic field affecting all of the sensors will be completely uncorrelated with the reconstructed currents but is expected to show more than usual correlation between sensors.

- An extra conductor that was not detected during calibration but begins carrying current later on will have some correlation with the reconstructed currents but less correlation between sensors.

- An appliance leaking ground currents (i.e. a violation of the assumption that the sum of all currents is always zero) will be strongly correlated with exactly one of the reconstructed currents and uncorrelated with the rest.

In any case, when $\|\boldsymbol{s}^{\perp}(t)\|$ grows large, something is wrong and the results of the non-contact monitoring system cannot be trusted. Conversely, if the number of sensors is large and $\|\boldsymbol{s}^{\perp}(t)\|$ remains small, there is good confidence in the quality of the measurements.

4.2 Transient events

In an examination of building energy usage, the most common scenario is roughly constant power usage punctuated by step changes as devices switch on or off. These step changes are known as "transients".

The information provided by transients is often richer than that provided by the steady-state regions [2]. Transients can contain large spikes of real or reactive inrush currents which reflect the physical properties and time constants of the devices which are changing state. Transients even provide information which can be used to distinguish switching power supplies based on their control schemes, such as the presence of a "soft start" mechanism or the startup behavior of a control loop performing active power factor correction.

In order to harness any of this information, the first (and most difficult) step is to separate out the transients from the more prevalent steady state regions. This signal processing problem is complicated by a potentially low signal to noise ratio, since a low-power device may produce a turn-on transient concurrently with a high-power device emitting continuous noise back onto the power line. Due to the large amount of input data to be processed, any transient detector must have robustly correct behavior without human oversight.

4.2.1 Architecture

The transient detector uses two different algorithms to separate out the transient regions from the steady-state regions. The first algorithm accurately detects the beginnings of transient regions, but it cannot detect their ends. The second algorithm can detect whether a any instant qualifies as part of a transient, but emits occasional false positives.

Together, these two algorithms create a highly effective and accurate transient detector. The detector scans through the data forward in time using the first algorithm. When that algorithm triggers, the detector marks a "transient start". Then it switches to the second algorithm and scans forward in time. When that algorithm stops triggering, the detector marks a "transient end" and switches back to the first algorithm. In this manner, the false positives of the second algorthim have no ill effects other than slightly elongating the lengths of detected transients from time to time.

The transient detector can be applied to any real or complex scalar valued signal. In most cases it would be applied to the rotated currents $\boldsymbol{c}$ defined by equation 3.24. However, it can also be applied directly to the rotated sensor data $\boldsymbol{b}$ defined by equation 3.25. This is useful for implicit calibration (section 3.2.5) when prospective transient events must be identified by a high-pass-like filter prior to performing the calibration process.

4.2.2 Start condition detector

The beginnings of transients are signified by a sudden change in value, so a standard CUSUM change of mean algorithm [19] is used. This algorithm only detects changes of mean in a particular direction, so two change-of-mean detectors are run in tandem—one to detect increases in the mean (i.e. turn-on transients) and one to detect decreases in the mean (i.e. turn-off transients).

In order to avoid detecting slow ramps as transient events, the signal is first passed through a causal high-pass filter. A Bessel filter is used because it minimizes distortion in the time domain. The output of that filter then becomes the input to the change of mean detectors. Figue 4.5 shows a typical transient event and the output signal after it is passed through a third-order Bessel high-pass filter with a cutoff frequency of 1 Hz.

Suppose that $h[t]$ are the samples of the high-pass-filtered signal (where t is an integer). Each CUSUM change of mean detector has parameters k_1 (the "threshhold") and k_2 (the "offset"), and maintains two internal state variables $s[t]$ and $g[t]$. The state evolves according to

$$s[t] = h[t] + s[t-1] - k_2$$

and

$$g[t] = \max(h[t] + g[t-1] - k_2, 0).$$

Whenever

$$g[t] > k_1,$$

a change of mean has been detected. A changes of mean is not detected until a few samples after it took place. The time τ at which the actual change occurred is given by

$$\operatorname{argmin}(s[\tau])$$

for $\tau \leq t$.

When a change of mean is found, the algorithm marks a transient start and switches to the end condition detector.

4.2.3 End condition detector

The ends of transients are delineated by the signal maintaing a roughly constant value. In other words, when a transient event has ended, the standard deviation over an upcoming window of samples is small. Figure 4.5 shows a typical transient event and its rolling standard deviation over a window of length 0.5 s. The standard deviation clearly diminishes as the transient ends.

It is difficult to define what is meant by a "small" standard deviation, because even the steady-state regions may contain considerable noise if a large load is operating. This situation is resolved by choosing a criterion that scales with the noise floor of a signal. The standard deviation over a window of the signal is divided by the standard deviation of the rolling difference of the signal computed over the same window.

Suppose that the samples of a signal are given by $f[t]$ (where t is an integer). For a particular window length k_3, the windowed mean is defined by

$$\mu[t] = \frac{1}{k_3} \sum_{\tau=1}^{k_3} f[t+\tau]$$

and the windowed standard deviation is defined by

$$\sigma[t] = \sqrt{\sum_{\tau=1}^{k_3} (f[t+\tau] - \mu[t])^2}.$$

The rolling difference is defined by

$$f_d[t] = f[t] - f[t-1].$$

The windowed mean of the rolling difference is

$$\mu_d[t] = \sum_{\tau=1}^{k_3} f_d[t+\tau] = f[t+k_3] - f[t]$$

and the windowed standard deviation of the rolling difference is

$$\sigma_d[t] = \sqrt{\sum_{\tau=1}^{k_3} (f_d[t+\tau] - \mu_d[t])^2}.$$

By the Cauchy-Scwarz inequality, it must always be the case that

$$\sigma_d[t] \leq 2\sigma[t].$$

For a signal of uniform white noise,

$$\sigma_d[t] \approx \sqrt{2}\sigma[t].$$

And when a transient event extends within the window from time t to time $t + k_3$,

$$\sigma_d[t] \ll \sqrt{2}\sigma[t].$$

The criterion for the transient end detector is

$$\frac{\sigma_d[t]}{\sigma[t]} < k_4$$

where k_4 is a threshhold value which sets the end detector's sensitivity.

4.2.4 Implementation

The transient detector is implemented as a NILM filter using the framework provided by [4]. The code is given in appendix E.2. The detector is written for discrete-time signals provided at a constant sample rate. Since the CUSUM step detector is non-causal, a rolling look-ahead buffer is maintained while the system searches for transient starts.

There are six parameters, creatively named k_1 through k_6, which control the operation of the transient finder. They have the following functions:

k_1 controls the certainty threshhold required by the CUSUM step detector. It is specified relative to the rolling standard deviation of the signal. Higher values decrease false positives but increase false negatives. A reasonable starting value is about 30.

k_2 controls the minimum step size detectable by the CUSUM detector. It is also specified relative to the rolling standard deviation of the signal. Higher values improve noise immunity but decrease sensitivity. A reasonable starting value is about 1.

k_3 controls the window length for the standard-devation-based end of transient detector. Larger values will avoid cutting off transients early but may spuriously combine transients that are close together. An appropriate value depends on the sample rate and the expected frequency of transients. Typically, about 30 is reasonable. This quantity must be an integer.

k_4 controls the minimum ratio between the standard deviation of a window and the standard deviation of the rolling difference of the window that signifies a transient is going on. A reasonable value is about 0.8.

k_5 is the order of the causal high-pass Bessel filter. Typically second and third order filters work well. Higher order filters introduce too much group delay.

k_6 is the cutoff frequency of the high-pass filter, specified relative to the Nyquist frequency of the signal. As an approximate guideline, choose $k_6 = 1/(2 \cdot k_3)$. Higher values will weight the detector towards looking for the spikes at the beginnings of transients rather than step changes over the duration.

4.3 Results

4.3.1 Numerical example of AC calibration

Suppose that there are three sensors instrumenting a household service entrance cable with two high-voltage conductors and a neutral return, such that the sensor matrix is given by

$$\boldsymbol{M} = \begin{bmatrix} 0.5 & 0.3 & 0.7 \\ 0.3 & 0.5 & 0.7 \\ 0.4 & 0.4 & 0.8 \end{bmatrix}$$

Further suppose that the line voltages are $2\pi/3$ radians apart, and the reference phase is 0.5 radians behind the first conductor voltage, so

$$\boldsymbol{\Theta} = \begin{bmatrix} e^{-0.5j} & 0 \\ 0 & e^{2\pi j/3 - 0.5j} \end{bmatrix}.$$

The calibration load draws an RMS current of $\beta = 2.2\,\mathrm{A}$ between a line and neutral, so that

$$\boldsymbol{\sigma}_x = \beta e^{-j\theta_{\alpha_x}} \boldsymbol{M}(\hat{\boldsymbol{\imath}}_{\alpha_x} - \hat{\boldsymbol{\imath}}_3).$$

The calibration load is run four times with $\alpha_2 = \alpha_3 = 1$ and $\alpha_1 = \alpha_4 = 2$. To simulate sensor noise and other calibration errors, each simulated calibration load signature is perturbed by about 2% to obtain

$$\boldsymbol{\sigma}_1 = \begin{bmatrix} +0.01 + 0.86j \\ -0.01 + 0.43j \\ +0.01 + 0.87j \end{bmatrix} \qquad \boldsymbol{\sigma}_2 = \begin{bmatrix} -0.39 - 0.22j \\ -0.78 - 0.44j \\ -0.76 - 0.43j \end{bmatrix}$$

$$\boldsymbol{\sigma}_3 = \begin{bmatrix} -0.39 - 0.20j \\ -0.78 - 0.43j \\ -0.80 - 0.42j \end{bmatrix} \qquad \boldsymbol{\sigma}_4 = \begin{bmatrix} +0.04 + 0.90j \\ +0.02 + 0.45j \\ +0.02 + 0.87j \end{bmatrix}.$$

The goal now is to determine some matrix $\boldsymbol{K}$ which recovers the amplitude and relative phase of each line current from the Fourier transform of the sensor data, using only $\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \boldsymbol{\sigma}_3, \boldsymbol{\sigma}_4$ and without any knowledge of $\boldsymbol{M}$ or $\boldsymbol{\Theta}$.

First assemble the $\boldsymbol{\sigma}_x$ into $\boldsymbol{\Sigma}$ and compute

$$\boldsymbol{\Sigma}\boldsymbol{\Sigma}' = \begin{bmatrix} 1.94 & 1.57 - 0.02j & 2.32 - 0.02j \\ 1.57 + 0.02j & 1.98 & 2.35 + 0.02j \\ 2.32 + 0.02j & 2.35 - 0.02j & 3.09 \end{bmatrix}.$$

The eigenvalues of $\boldsymbol{\Sigma}\boldsymbol{\Sigma}'$ are 6.62, 0.40, and 0.001. Thus $n = 2$ and, from the eigendecomposition of $\boldsymbol{\Sigma}\boldsymbol{\Sigma}'$,

$$\boldsymbol{D} = \begin{bmatrix} -0.51 - 0.00j & -0.52 + 0.01j & -0.68 + 0.01j \\ +0.71 + 0.00j & -0.70 + 0.02j & -0.00 - 0.00j \end{bmatrix}.$$

As expected, these matrices are mostly real.

The clustering algorithm assigns $\boldsymbol{D\sigma}_1$ and $\boldsymbol{D\sigma}_4$ to the first cluster and $\boldsymbol{D\sigma}_2$ and $\boldsymbol{D\sigma}_3$ to the second cluster, with cluster centers

$$\boldsymbol{\delta}_1 = \begin{bmatrix} -0.03 - 1.27j \\ +0.01 + 0.32j \end{bmatrix} \qquad \boldsymbol{\delta}_2 = \begin{bmatrix} +1.14 + 0.61j \\ +0.28 + 0.15j \end{bmatrix}.$$

Therefore

$$\boldsymbol{U} = \beta \begin{bmatrix} \boldsymbol{\delta}_1 & \boldsymbol{\delta}_2 \end{bmatrix}^{-1} = \begin{bmatrix} -0.02 + 0.85j & +0.11 - 3.52j \\ +0.76 - 0.41j & +3.06 - 1.62j \end{bmatrix}$$

and

$$\boldsymbol{K} = \begin{bmatrix} +0.09 - 2.95j & -0.02 + 2.03j & -0.00 - 0.57j \\ +1.80 - 0.95j & -2.51 + 1.40j & -0.54 + 0.28j \end{bmatrix}.$$

This concludes the calibration process.

To verify that $\boldsymbol{K}$ is the correct unmixing matrix, compute

$$\boldsymbol{KMH\Theta}' = \begin{bmatrix} -0.01 - 0.00j & +1.00 + 0.01j \\ +0.98 - 0.01j & -0.01 + 0.00j \end{bmatrix}$$

which is very close to a permuted identity matrix. Deviations are caused by the noise that was added to $\boldsymbol{\sigma}_x$. (The rows are permuted because the order of clusters is determined arbitrarily, i.e. the two line conductors have no inherent ordering.) Thus the calibration procedure successfully constructs a well-conditioned left inverse to $\boldsymbol{MH\Theta}'$ *without specific knowledge of* $\boldsymbol{M}$ *or* $\boldsymbol{\Theta}$.

4.3.2 Non-contact sensing

The new non-contact voltage and current sensors were used to perform power metering as shown in Fig. 4.1. The non-contact power meter was installed in parallel with a traditional power meter so that the results could be compared. Various electrical loads were switched on and off in order to obtain the time series data depicted in Fig. 4.2. Mismatch between the traditional power meter and the non-contact power meter did not exceed 10 W over a dynamic range of 1000 W, showing that the new sensors are able to achieve 1% accuracy—sufficient for most energy monitoring applications.

In order to obtain more detailed results showing the performance of the voltage sensor's digital integrating filters, the analog board was attached to an 18-AWG computer power cable with line voltage supplied by an HP 6834B AC source. (This cable was chosen because thinner conductors produce the smallest coupling capacitance and therefore pose the most difficult sensing challenge.) This experimental setup is shown in Fig. 4.3. The sensor was attached to an Atmel SAM4S microcontroller, which sampled the sensor with its built in ADC at a sample rate of 3 kHz and processed the signal using both of the FIR filters. The analog filter of [5] was constructed using a Texas Instruments OPA4376 operational amplifier and its output was connected to a second ADC channel. The output from all three filters was streamed from the microcontroller to a computer. With a line frequency of 60 Hz, there were 50 samples per line cycle and $N = 25$.

The output voltage from each filter was measured for sinusoidal inputs at various voltages and frequencies. Equation (2.4) was solved to find that the differential capacitance was

Input V RMS	Analog % error	FIR 1 % error	FIR 2 % error	Analog phase error, degrees
30	5.7	2.3	2.3	8.92
60	3.7	1.6	1.6	9.04
90	1.1	0.0	0.0	9.07
120	0.3	0.0	0.0	9.09
150	1.0	1.2	1.2	9.11
180	−0.5	0.8	0.8	9.16
210	−2.2	−0.0	−0.0	9.21
240	−3.3	−0.1	−0.1	9.29
270	−3.8	0.3	0.3	9.34
300	−4.5	0.5	0.5	9.40

Table 4.1: Output error from each filter for various input voltages at 60 Hz.

Input Hz	Analog % error	FIR 1 % error	FIR 2 % error	Analog phase error, degrees
60	5.7	2.3	2.3	8.92
120	2.9	−1.7	−1.7	4.98
180	2.5	−2.7	−2.7	3.69
240	1.9	−3.8	−3.8	3.06
300	1.6	−4.2	−4.2	2.69
360	0.8	−5.2	−5.2	2.48
420	0.1	−6.2	−6.2	2.33
480	−1.3	−7.6	−7.6	2.22
540	−2.8	−9.1	−9.1	2.16
600	−4.8	−11.2	−11.2	2.14

Table 4.2: Output error from each filter for various input frequencies at 30 V RMS.

1.22 pF at 120 V and 60 Hz. At other voltages and frequencies, the percent magnitude error was computed for the output of each filter. The phase error of the analog filter relative to the (zero-phase) digital filters was also computed. This data is given in tables 4.1 and 4.2.

The collected data shows that the digital filters significantly outperform the analog filter with respect to phase lag and voltage linearity. As predicted by (2.5), all filters suffer from frequency-dependent gain, with a slightly more pronounced effect for the digital filters.[1] The new sensors with digital filters exhibit error less than 5% over all voltages up to 300 V and frequencies up to 300 Hz.

Finally, the disturbance rejection of each filter was tested by turning off a 100 mA fan motor at a distance of 30 cm away from the sensor. (The motor does not have a clamp circuit, so an inductive voltage spike generates a strong electric field every time it is turned off.) The response of the three filters to this situation is shown in Fig. 4.4. There is good agreement with the simulated behavior in Fig. 2.17. The digital filters are only affected by the disturbance for one or two line cycles, but the analog filter has not recovered after many line cycles. Fig. 4.4 also shows that the digital filters prevent the disturbance from affecting power metering.

4.3.3 Detection of transient events

The algorithm of section 4.2 was applied to several waveforms using the implementation given in appendix E.2. Figure 4.6 applies the transient detector to a single turn-on event for a 3 kW compressor, illustrating the expected result in a very simple case. Figure 4.7 applies the transient detector to a long duration of data gathered from Fort Devens. Figure 4.8 applies the transient detector to very noisy data gathered from Fort Polk. For all three figures, the parameters used were $k_1 = 60$, $k_2 = 1$, $k_3 = 30$, $k_4 = 0.8$, $k_5 = 3$, and $k_6 = 0.017$. These parameters err on the side of caution, occasionally missing transients or combining two transients but never reporting transients that do not exist.

4.4 Conclusion

Non-contact sensing represents a significant new direction in the development of non-intrusive load monitoring systems. The barrier to installation of a load monitoring system is significantly reduced, but non-contact sensing presents unique design and signal processing challenges which must be overcome in order to obtain reliable data.

This thesis presented a new hardware and software architecture for non-contact sensing. The new architecture is inexpensive and flexible for a wide variety of application scenarios. It integrates with existing non-intrusive load monitoring systems while fully leveraging the new capabilities of non-contact sensors. Calibration and signal processing algorithms were presented for making sense of non-contact sensor data. These algorithms run in real time with minimal computational load.

[1] If the additional frequency-dependent attenuation is carefully measured, the frequency response H_2 may be shaped to eliminate the frequency dependence at no additional computational cost by slightly altering the coefficients c_n used in its construction.

With these results, non-contact sensors are ready for deployment in non-intrusive load monitoring applications. The cost and performance penalties for using non-contact sensors are lower than ever before. The benefits, such as ease of installation and robust high-voltage isolation, will outweigh these penalties in many new applications.

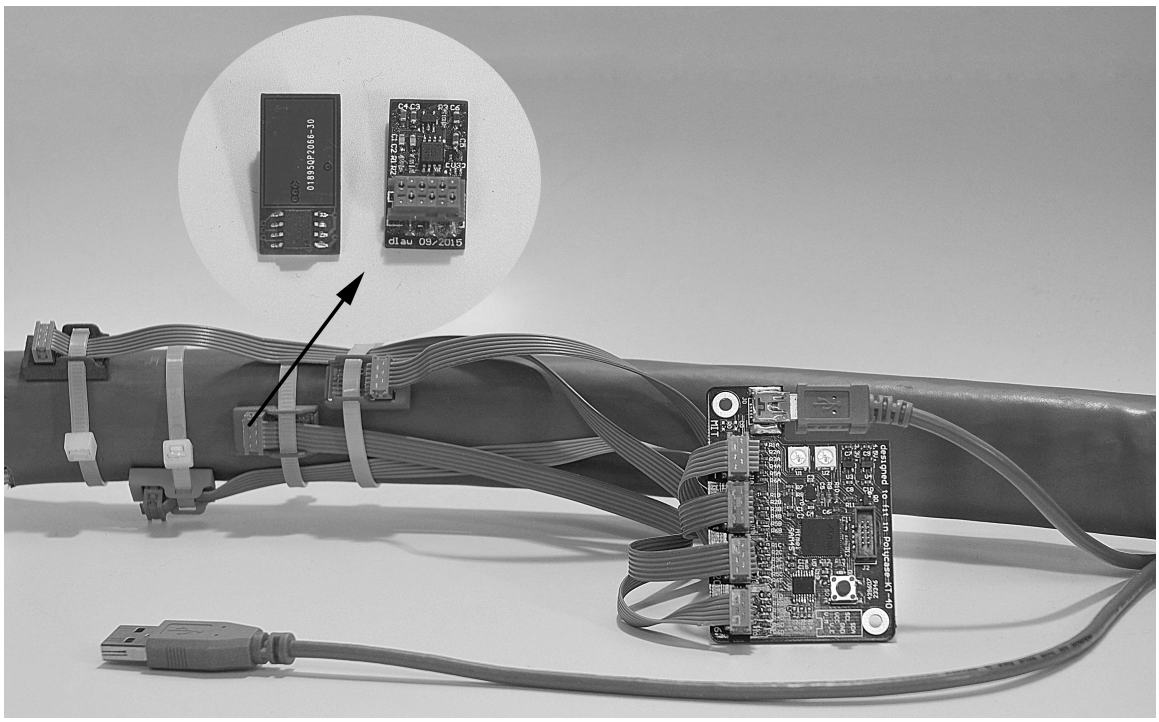


Figure 4.1: Non-contact power meter using the new analog and digital boards installed on a service entrance cable for power metering. Photo by Steven Leeb.

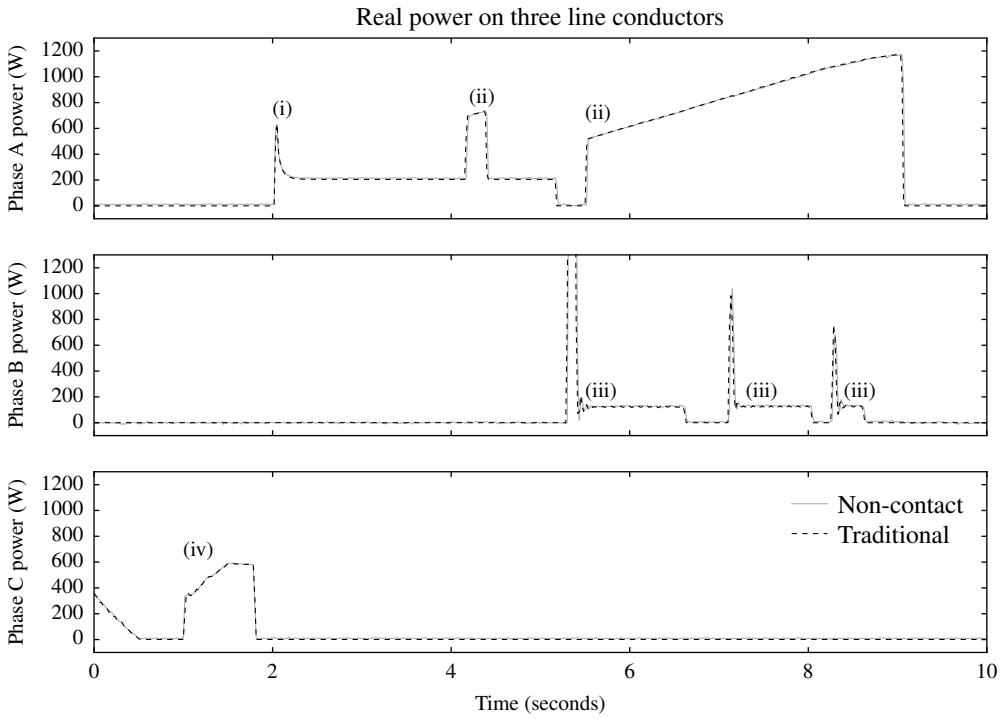


Figure 4.2: Data collected by non-contact and traditional power meters. The turn-on transients depicted are from (i) a 250 W incandescent lightbulb, (ii) a 1500 W space heater, (iii) an 0.25 hp induction motor, and (iv) a 600 W bank of dimmable incandescent lightbulbs.

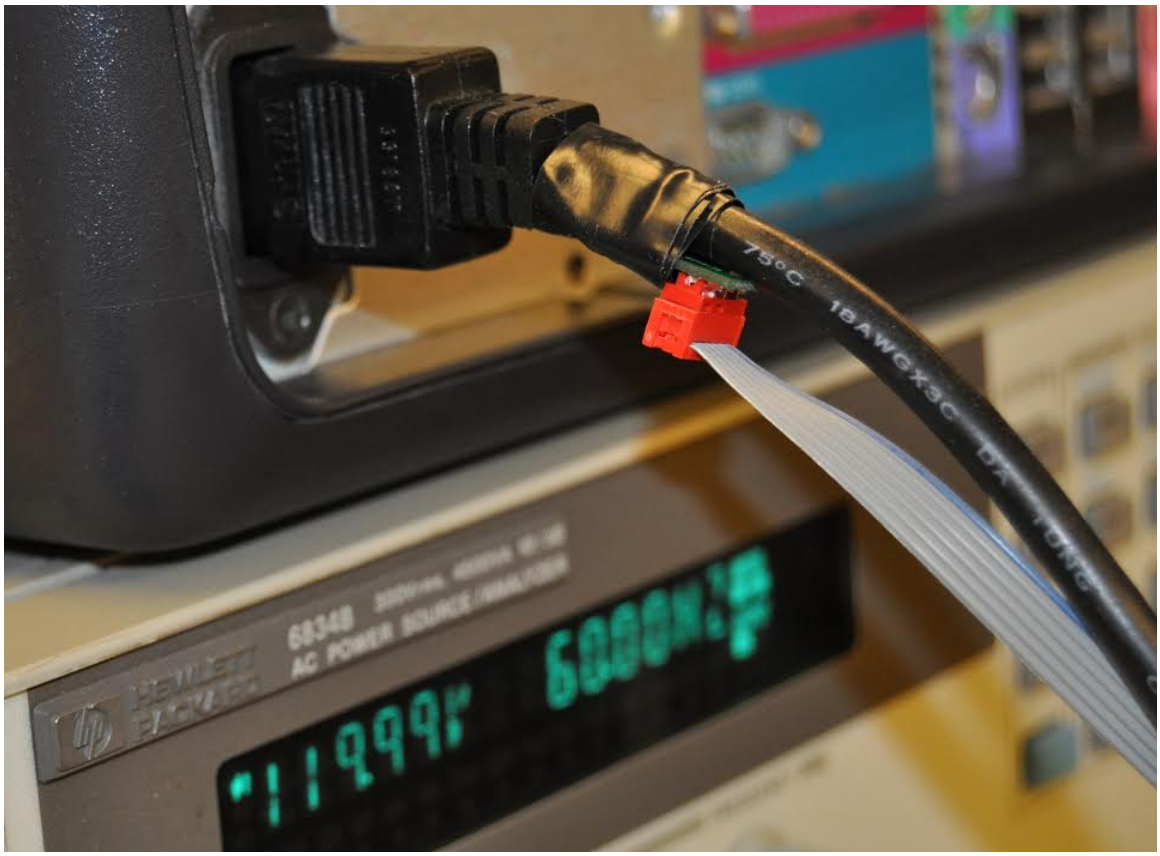


Figure 4.3: Non-contact sensor (analog board) attached to an 18-AWG computer power cable powered by an HP 6834B AC source. The ribbon cable leads to a printed circuit board which implements the analog filter and two digital FIR filters.

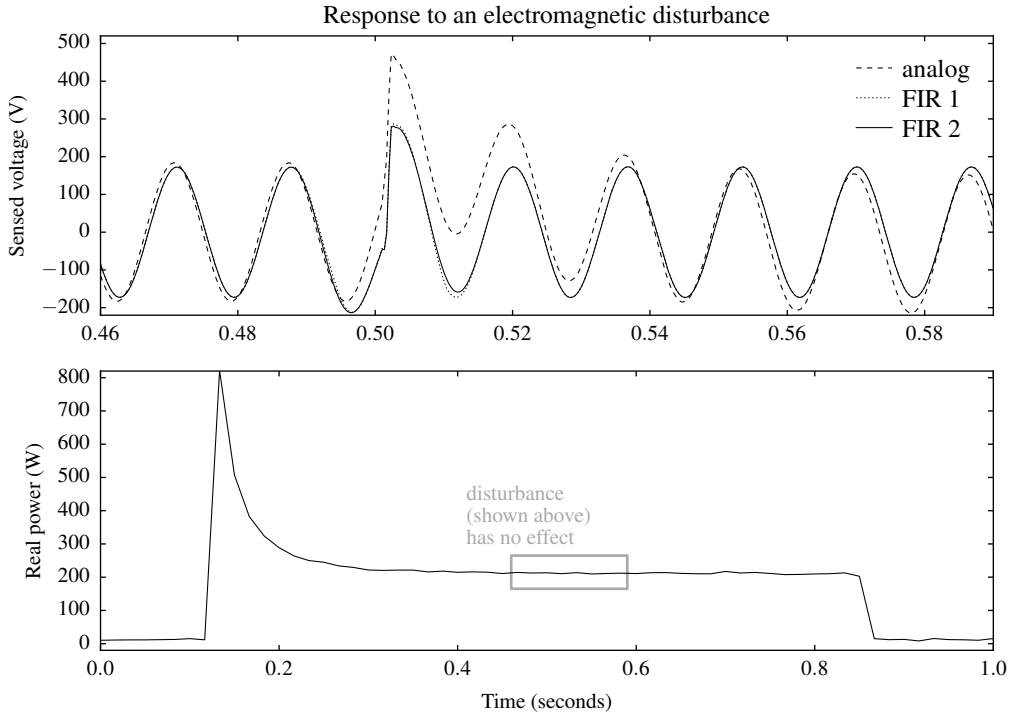


Figure 4.4: Response of the voltage sensor to a 100 mA fan motor being turned off 30 cm away from the sensor at $t \approx 0.5$. The digital filter recovers from the electromagnetic disturbance quickly, so *non-contact power metering is not affected by the disturbance*.

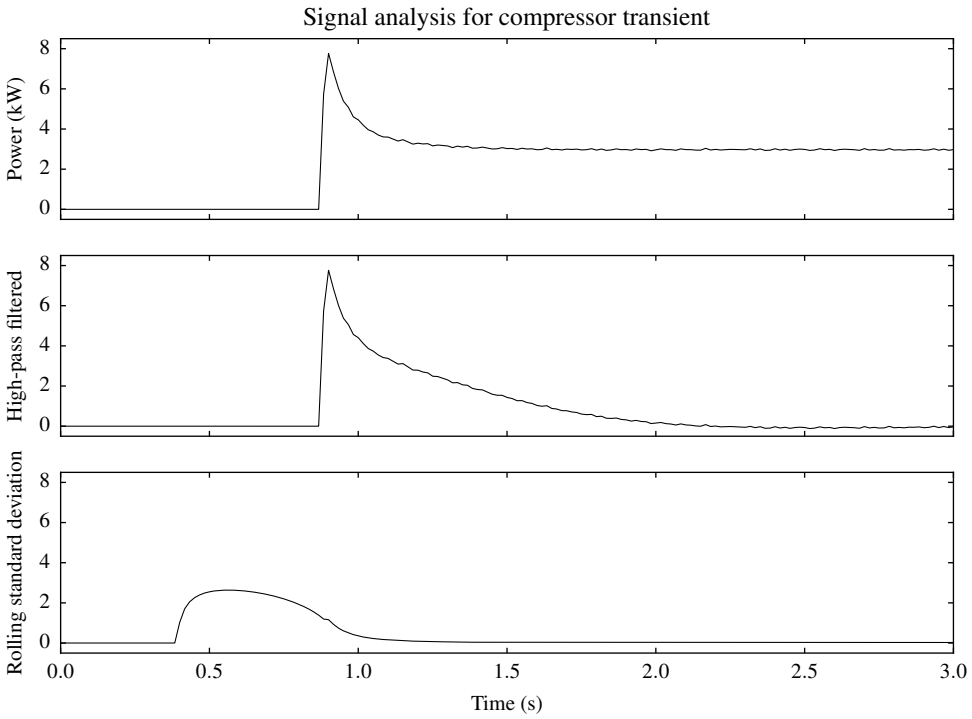


Figure 4.5: Signal processing in preparation for applying the transient detector: computation of a high-pass-filtered version of the power and computation of the rolling standard deviation of the power. The high-pass-filtered version is used to identify transient starts and the rolling standard deviation is used to identify transient ends.

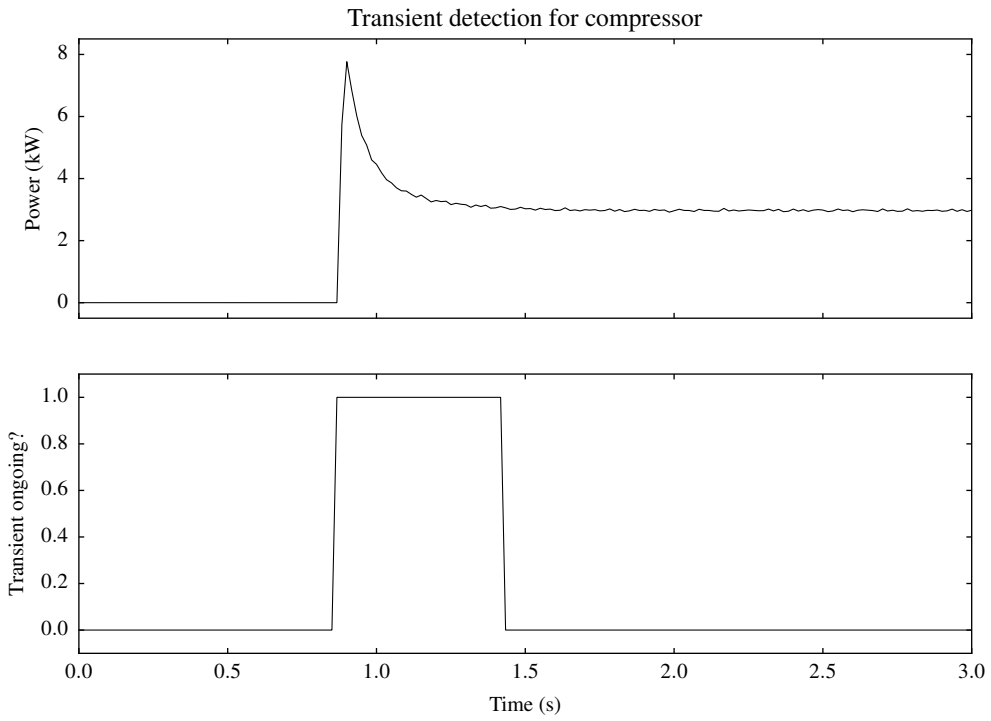


Figure 4.6: Transient detector applied to data from monitoring a 3 kW air compressor. The data represents a single turn-on of the compressor and there is minimal noise, so this is an easy case for the algorithm to handle.

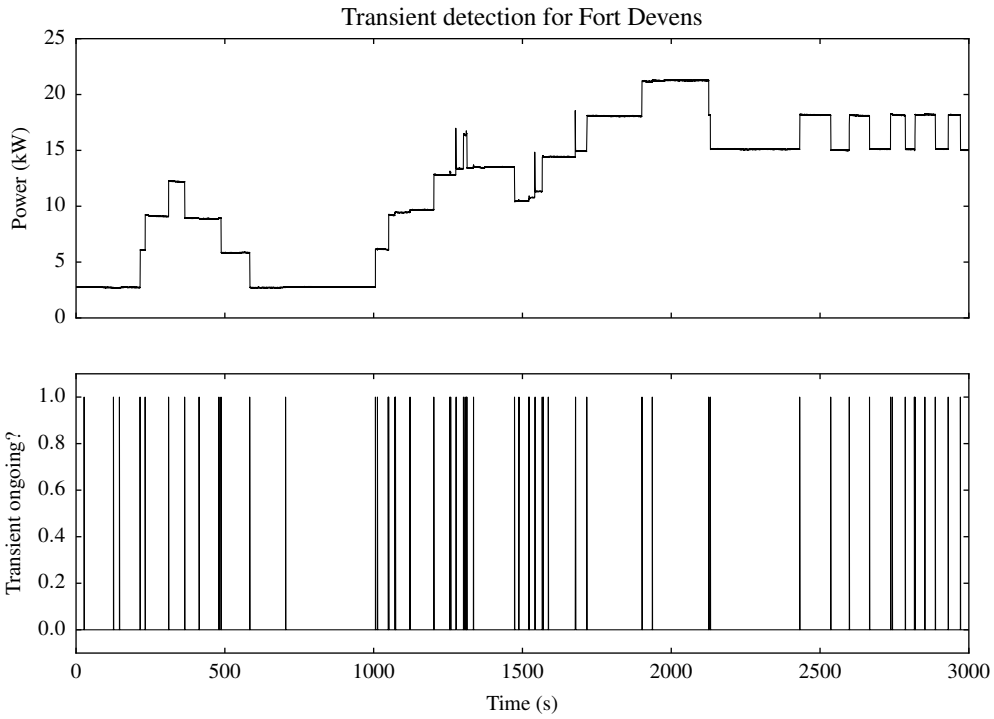


Figure 4.7: Transient detector applied to data from monitoring a large electric service at Fort Devens, MA. The transient detector operates well over a longer period of time. Because of the low noise floor, it even notices a few small transients (e.g. at $t \approx 700$) that a human would easily overlook.

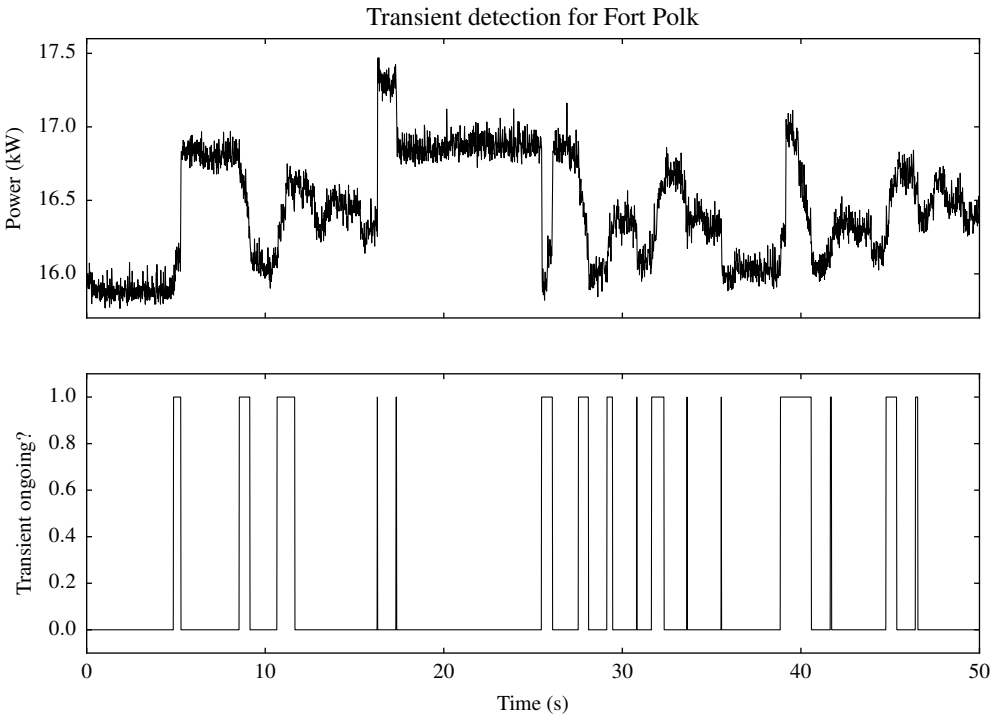


Figure 4.8: Transient detector applied to data from monitoring a large electric service at Fort Polk, LA. This is an extremely difficult case for transient detection to to the large amplitude of background noise and rapid sequencing of transient events. The detector obtains reasonably good results even in this case.

Acknowledgements

I would like to thank John Donnal and Professor Steven Leeb for their guidance, enthusiasm, and technical contributions to this thesis. Particular acknowledgement is also due to James Paris, whose 2013 PhD thesis laid the groundwork for many of these results.

I would like to thank Glen Whitney and Nancy Goroff for their generosity in helping me reach MIT. Most importantly, I offer my heartfelt thanks to my parents Cindy and Kevin for their support throughout my life.

References

[1] G. W. Hart. "Nonintrusive Applicance Load Monitoring". In: *Proceedings of the IEEE* 80 (Dec. 1992), pp. 1870–1891.

[2] L. K. Norford and S. B. Leeb. "Non-intrusive electrical load monitoring in commercial buildings based on steady-state and transient load-detection algorithms". In: *Energy and Buildings* 24 (May 1996), pp. 51–64.

[3] C. Laughman, K. Lee, et al. "Power Signature Analysis". In: *IEEE Power and Energy Magazine* 1 (Apr. 2003), pp. 56–63.

[4] J. Paris. "A Comprehensive System for Non-Intrusive Load Monitoring and Diagnostics". PhD thesis. Massachusetts Institute of Technology, Sept. 2013.

[5] J. S. Donnal and S. B. Leeb. "Noncontact Power Meter". In: *IEEE Sensors Journal* 15 (Feb. 2015), pp. 1161–1169.

[6] E. J. M. van Heesch, R. Caspers, et al. "Three phase voltage measurements with simple open air sensors". In: *Proceedings of the 7th International Symposium on High Voltage Engineering*. Aug. 1991.

[7] K. T. Selva, O. A. Forsberg, et al. "Non-contact current measurement in power transmission lines". In: *Procedia Technology* 21 (Aug. 2015), pp. 498–506.

[8] P. Pai, C. Lingyao, et al. "Non-intrusive electric power sensors for smart grid". In: *Proc. IEEE Sensors*. Oct. 2012, pp. 1–4.

[9] W. A. Zisman. "A new method of measuring contact potential differences in metals". In: *Review of Scientific Instruments* 3 (Mar. 1932), pp. 367–370.

[10] W. E. Vosteen. "A high speed electrostatic voltmeter technique". In: *Industry Applications Society Annual Meeting*. Oct. 1988, pp. 1617–1619.

[11] D. Grant, G. Hearn, et al. "An electrostatic charge meter using a microcontroller offers advanced features and easier ATEX certification". In: *Journal of Electrostatics* 67 (May 2009), pp. 473–476.

[12] L. Wu, P. Wouters, et al. "On-site voltage measurement with capacitive sensors on high voltage systems". In: *IEEE Trondheim PowerTech*. June 2011.

[13] J. Bobowski, S. Ferdous, and T. Johnson. "Calibrated Single-Contact Voltage Sensor for High-Voltage Monitoring Applications". In: *IEEE Instrumentation* 64 (Apr. 2015), pp. 923–934.

[14] K. M. Tsang and W. L. Chan. "Dual capacitive sensors for non-contact AC voltage measurement". In: *Sensors and Acturators A: Physical* 167 (June 2011), pp. 261–266.

[15] D. Balsamo, D. Porcarelli, et al. "A new non-invasive voltage measurement method for wireless analysis of electrical parameters and power quality". In: *Proc. IEEE Sensors*. Nov. 2013.

[16] J. Lenz and A. S. Edelstein. "Magnetic sensors and their applications". In: *IEEE Sensors Journal* 6 (June 2006), pp. 631–649.

[17] C. Fortescue. "Method of Symmetrical Coordinates Applied to the Solution of Polyphase Networks". In: *Transactions of the American Institute of Electrical Engineers*. Vol. 37. June 1918, pp. 1027–1140.

[18] U. von Luxburg. "A tutorial on spectral clustering". In: *Statistics and Computing* 17 (Dec. 2007), pp. 395–416.

[19] E. S. Page. "Continuous Inspection Schemes". In: *Biometrika* 41 (June 1954), pp. 100–115.

Appendix A

Mathematical Notation

The following conventions apply throughout this thesis.

- Vectors are represented by lowercase boldface letters, matrices are represented by uppercase boldface letters, and all non-bolded quantities are scalar.
- Matrices and vectors are 1-indexed (except in source code, where they are 0-indexed).
- Operators (functions whose inputs and outputs are themselves functions) are denoted by uppercase calligraphic letters. The notation $\mathcal{A}(f)(y)$ indicates that the operator $\mathcal{A}$ is applied to the function f and the result of that application is evaluated at y.
- j is the imaginary unit. The complex conjugate of a quantity x is denoted by $\overline{x}$.
- Given a vector or matrix $\boldsymbol{A}$, $\boldsymbol{A}'$ is its conjugate transpose, $\boldsymbol{A}^+$ is its Moore-Penrose pseudoinverse, $\|\boldsymbol{A}\|$ is its largest singular value, $\sigma_{\min}(\boldsymbol{A})$ is its smallest singular value, and $\kappa(\boldsymbol{A})$ is its condition number.
- The kernel of a matrix $\boldsymbol{A}$ is written as "$\ker \boldsymbol{A}$" and defined as the space of vectors $\boldsymbol{a}$ satisfying $\boldsymbol{A}\boldsymbol{a} = \boldsymbol{0}$.
- The symbol $\boldsymbol{I}$ denotes an identity matrix, whose dimension is determined by context.

Appendix B

Index of Calibration Variables

This table provides a reference for calibration variables used throughout chapter 3.

Name	Shape	Description	Properties
n	Scalar	Number of conductors	
k	Scalar	Number of current sensors	$k \geq n$
l	Scalar	Number of voltage sensors	
m	Scalar	Number of harmonics	
p	Scalar	Number of calibrations	$p \geq n$
$\boldsymbol{s}(t)$	k-element vector	Sensor data	
$\boldsymbol{i}(t)$	n-element vector	Currents	
$\boldsymbol{M}$	k-by-n matrix	Mixing matrix	$\boldsymbol{s}(t) = \boldsymbol{M}\boldsymbol{i}(t)$
$\boldsymbol{\sigma}_i$	k-element vector	Calibration load sensor signature	
$\boldsymbol{\delta}_i$	n-element vector	Cluster center (in $\boldsymbol{D}\boldsymbol{s}$ space)	
$\boldsymbol{\Sigma}$	k-by-p matrix	Calibration sensor signatures	$\boldsymbol{\Sigma} = \boldsymbol{M}\boldsymbol{\Gamma}$
$\boldsymbol{\Gamma}$	n-by-p matrix	Calibration currents	
$\boldsymbol{D}$	n-by-k matrix	Dimensionality reduction matrix	$\boldsymbol{D}^+ = \boldsymbol{D}'$
$\boldsymbol{U}$	n-by-n matrix	Unmixing matrix	$\boldsymbol{M}^+ = \boldsymbol{U}\boldsymbol{D}$
β	Scalar	Calibration load current	
α_x	Scalar	Conductor number on xth run	
$\boldsymbol{b}(t)$	k-element vector	Rotated sensor data	$\boldsymbol{b}(t) = e^{2\pi j t/T} \cdot \overline{\mathcal{F}_1(\boldsymbol{s})(t)}$
$\boldsymbol{c}(t)$	n-element vector	Rotated currents	$\boldsymbol{c}(t) = e^{2\pi j t/T} \cdot \boldsymbol{\Theta} \cdot \overline{\mathcal{F}_1(\boldsymbol{i})(t)}$
$\boldsymbol{\Theta}$	n-by-n matrix	Rotation matrix	$\boldsymbol{b}(t) = \boldsymbol{M}\boldsymbol{\Theta}'\boldsymbol{c}(t)$

Appendix C

Hardware User's Manual

This appendix is a datasheet-style summary of the non-contact monitoring hardware developed in chapter 2. It is intended for a future user of the non-contact monitoring system. Dear reader, if you are that user, then congratulations on your good taste.

C.1 System overview

There are two main components to the sensor system. The analog pickup ("A" board) is a small printed circuit board with a single electric field sensor and a single magnetic field sensor. The digital interface ("D" board) is an analog-to-digital converter which connects with up to four "A" boards (a total of eight analog channels) and streams data to a computer over a USB connection. Plastic cases are provided for both the "A" and "D" boards.

Most installations will use four (or fewer) "A" boards directly connected to one "D" board. However, in certain special cases a wye adapter ("Y" board) is attached between an "A" board and a "D" board. The "Y" board is needed for (i) installations requiring more than four magnetic field sensors, and (ii) installations requiring an electric field sensor output to be integrated using analog hardware instead of in software.

C.1.1 Sensor specifications

The "A" board has one electric field sensor and one magnetic field sensor. The magnetic field sensor is an off-the-shelf Melexis MLX91206 in the CAL-002 option, with a specified sensitivity of 380 V/T. It is sensitive to fields parallel to the pin direction of the package, such as those generated by a wire parallel to the long axis of the "A" board. The electric field sensor is a custom design described in section 2.1.1. It sensitive to voltages below the board regardless of the wire direction, and produces an output proportional to the time derivative of the electric field. Figure 4.1 illustrates the proper orientation of "A" boards on a multiple-conductor cable.

The "A" boards require a supply voltage of at least 4.5 V for the magnetic field sensor to operate properly. (All other components operate down to 3.3 V.) The connector pinout is given in section 2.2.2. The outputs from the electric and magnetic field sensors are low impedance analog signals. The approximate DC bias for the electric field output is 1.2 V

and the approximate DC bias for the magnetic field output is half of the input voltage.

C.1.2 Digital interface specifications

The valid input range of the "D" board's analog inputs extends from 0.0 V to 4.9 V. The hardware tolerates inputs up to 0.2 V outside this range on either end, but the resulting data will be meaningless. The output for each channel is signed 16-bit integer with an effective resolution of 15 bits, according to the following formula:

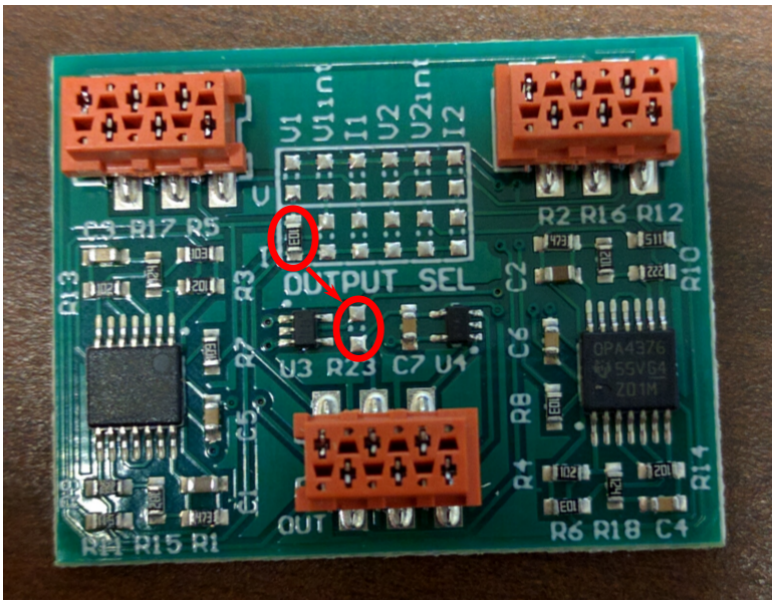
$$n_{\text{out}} = \left(\frac{2^{15}}{4.9\,\text{V}} \cdot v_{\text{in}} \right) - 2^{14}.$$

The "D" board performs some tricky signal processing on the analog input streams (see section 2.3.1), but the end result is that it streams output samples at 3 kHz per channel. Each output frame is 20 bytes long (see section 2.3.2), resulting in an output data rate of 60 kB/s. The sample rate is defined by a crystal oscillator and is accurate to within 20 ppm.

C.1.3 Wye adapter specifications

The "Y" board has two input connectors (labeled "J1" and "J2") and one output connector (labeled "OUT"). Each connector includes a conductor for the voltage (i.e. electric field) and a conductor for the current (i.e. magnetic field). The "Y" board integrates each of the two voltage input signals using the analog circuit of [5], resulting in a total of six signals derived from the inputs. A matrix of solder jumpers is provided so that each of the two output conductors can be attached to any of the six input-derived signals.

A "Y" board is pictured below. The picture illustrates an assembly error in the run of "Y" boards delivered January 7, 2016. The 10 kΩ resistor erroneously installed in the bottom-left solder jumper must be moved to the footprint labeled "R23" as indicated by the red arrow.



C.2 System assembly

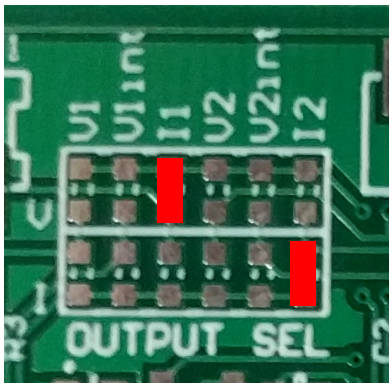
To assemble a standard monitoring system, begin with four "A" boards, four "A" board cases, and for "A" board cover plates. The "A" boards should be inserted into their cases and covered as shown in figure 2.8. First, the connector end of the "A" board is inserted under the large retaining lip. Then a flat-blade screwdriver is used to apply downward pressure just above the "C3" and "R3" silkscreen labels until the "A" board snaps in place beneath the retaining clip. Lastly, an acrylic cover is snapped into place over the top of the "A" board.

The next step is to create the ribbon cables which connect the "A" boards to a "D" board. As a matter of good practice, the "D" board should be located near the "A" boards and the ribbon cables should not be made longer than necessary. A Micro-Match connector is crimped onto each end of each ribbon cable so that the connector key lines up with the marked conductor (i.e. pin 1) and the connectors are oriented as shown in figure 4.1. The ribbon cables are then attached to the "A" boards.

Lastly, the ribbon cables and a USB cable are attached to the "D" board as shown in figure 2.5. The "D" board is inserted into a "D" board case and screwed shut with two screws. (Because all connectors are internal to the "D" board case, the connections must be made before the case is closed.) The resulting assembly is shown in figure 2.6.

C.2.1 Additional current sensors

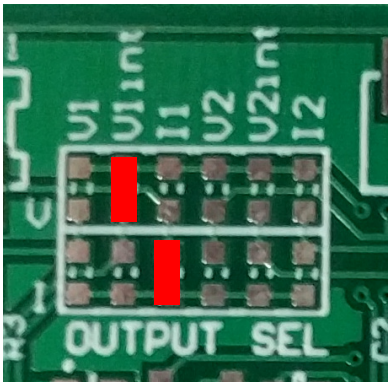
The "D" board digitizes eight channels of analog signals. Generally, these channels would be composed of four voltage sensors and four current sensors. However, some of the voltage sensors can be ignored in order to connect additional current sensors. This is accomplished by connecting two "A" boards to ports "J1" and "J2" on a "Y" board, and connecting the "Y" board's "OUT" port to a "D" board in place of one "A" board. The "Y" board solder jumpers are set as shown below. Non-contact sensing of AC signals requires at least one voltage sensor, so this technique can be used with a maximum of three "Y" boards giving a total of seven "A" boards sensing currents.



C.2.2 Hardware voltage integration

The electrical signals from the voltage sensors are typically digitized and then integrated by the software filter described in section . If it is ever desired to use an analog circuit to

perform the integration before digitization, the "Y" board can be used. An "A" board is attached to port "J1" of the "Y" board and the "OUT" port of the "Y" board is attached to a "D" board in place of the "A" board which was removed. Port "J2" of the "Y" board is left unconnected. The "Y" board solder jumpers are set as shown below.



C.2.3 Using an oscilloscope

In the lab, it is sometimes useful to see the analog sensor outputs on an oscilloscope. To accomplish this, fashion ribbon cables such that each "A" board connector is set back a few inches from the end of its ribbon cable, and separate and strip the conductors of each free end. This setup allows a "D" board and an oscilloscope to be attached simultaneously, which provides a convenient method to power the "A" boards under test.

An alternative method is to use a "Y" board and solder hookup wire to the exposed solder jumper pads. This method is more time-consuming and requires the use of an additional board, but it may result in an experimental apparatus which is slightly more permanent.

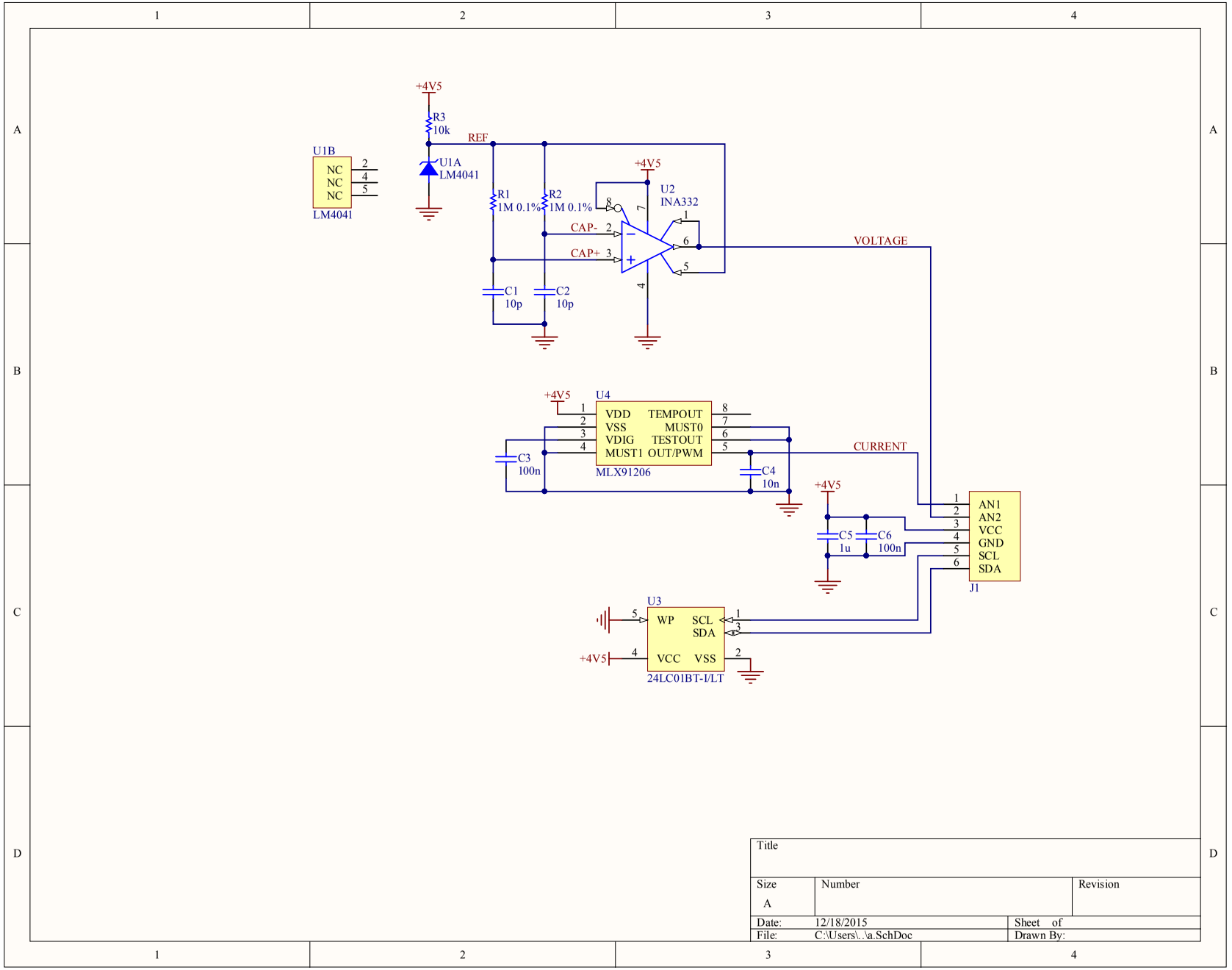
Appendix D

Printed Circuit Boards

D.1 Analog pickup

D.1.1 Bill of materials

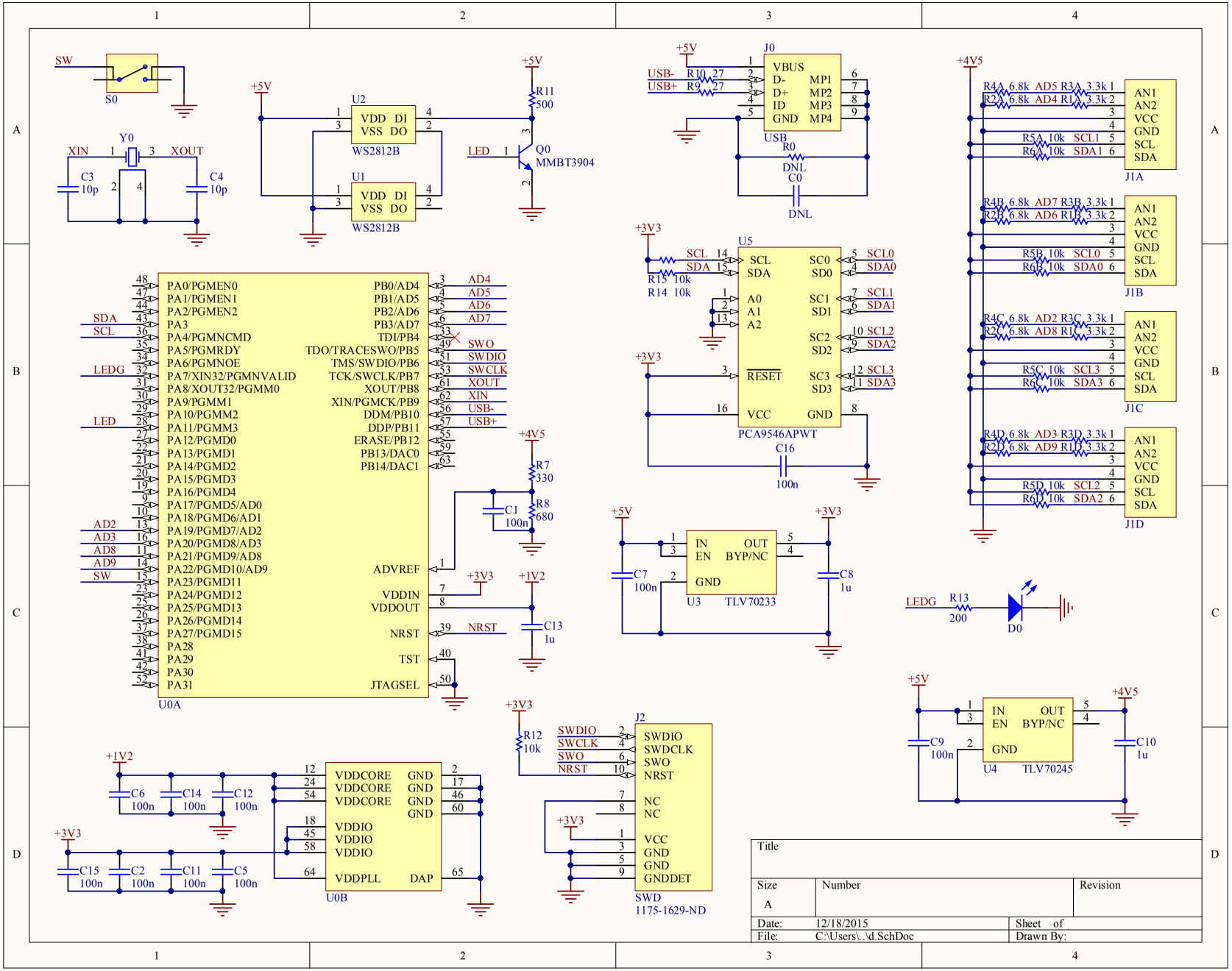
Designators	Qty	Description	Value	Footprint
R1, R2	2	Resistor	Yageo RC0603FR-071ML	0603
R3	1	Resistor	Yageo RC0603JR-0710KL	0603
C1, C2	2	Capacitor	Yageo CC0603JRNPO9BN100	0603
C3, C5	2	Capacitor	Yageo CC0603KRX7R7BB104	0603
C4	1	Capacitor	Yageo CC0603KRX7R9BB103	0603
C6	1	Capacitor	Yageo CC0603KRX5R6BB105	0603
U1	1	Shunt reference	TI LM4041DIM7-1.2	SC-70
U2	1	Instrumentation amp	TI INA332AIDGKR	VSSOP-8
U3	1	EEPROM	Microchip 24LC01BT-I/LT	SC-70
U4	1	Hall effect sensor	Melexis MLX91206LDC-CAL-002	SOIC-8
J1	1	Micro-match connector	TE Connectivity 188275-6	


+4V5
R3
10k
REF
U1B
NC
NC
NC
LM4041
U1A
LM4041
R1
1M 0.1%
R2
1M 0.1%
+4V5
U2
INA332
CAP-
CAP+
VOLTAGE
C1
10p
C2
10p
+4V5
U4
VDD
VSS
VDIG
MUST1
TEMPOUT
MUST0
TESTOUT
OUT/PWM
MLX91206
CURRENT
C3
100n
C4
10n
+4V5
C5
1u
C6
100n
AN1
AN2
VCC
GND
SCL
SDA
J1
U3
WP
SCL
SDA
VCC
VSS
+4V5
24LC01BT-I/LT
Title
Size
A
Number
Revision
Date:
12/18/2015
Sheet of
File:
C:\Users\..\a.SchDoc
Drawn By:

D.2 Digital interface

D.2.1 Bill of materials

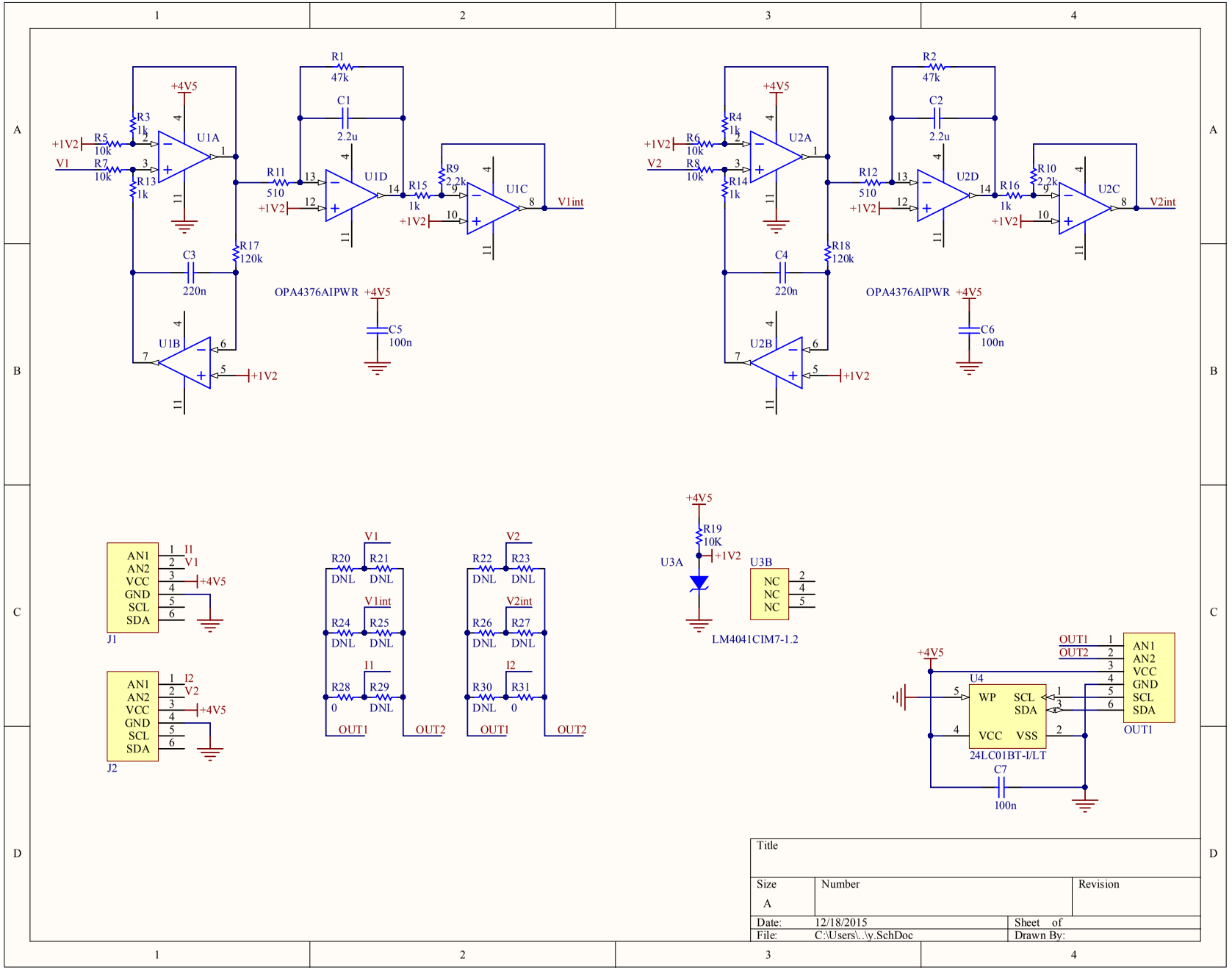
Designators	Qty	Description	Value	Footprint
R1A, R1B, R1C, R1D, R3A, R3B, R3C, R3D				
”	8	Resistor	Yageo RC0603FR-073K3L	0603
R2A, R2B, R2C, R2D, R4A, R4B, R4C, R4D				
”	8	Resistor	Yageo RC0603FR-076K8L	0603
R5A, R5B, R5C, R5D, R6A, R6B, R6C, R6D, R12, R14, R15				
”	11	Resistor	Yageo RC0603JR-0710KL	0603
R7, R11, R13	3	Resistor	Yageo RC0603FR-07330RL	0603
R8	1	Resistor	Yageo RC0603FR-07680RL	0603
R9, R10	2	Resistor	Yageo RC0603JR-0727RL	0603
C0	0	Capacitor	Do Not Load	0603
C1, C2, C5, C6, C7, C9, C11, C12, C14, C15, C16				
”	11	Capacitor	Yageo CC0603KRX7R7BB104	0603
C3, C4	2	Capacitor	Yageo CC0603JRNPO9BN100	0603
C8, C10, C13	3	Capacitor	Yageo CC0603KRX5R6BB105	0603
D0	1	LED	Lite-On LTST-C190KGKT	0603
Q0	1	N-channel MOSFET	Fairchild FDV301N	SOT23-3
Y0	1	Crystal oscillator	CTS 403C35D12M00000	
S0	1	Switch	Apem ADTSM63NVTR	
U0	1	Microcontroller	Atmel ATSAM4S4BA-MU	QFN-64
U1, U2	2	RGB LED	Worldsemi WS2812B	
U3	1	3.3V regulator	TI TLV70233DBVR	SOT23-5
U4	1	4.5V regulator	TI TLV70245DBVR	SOT23-5
U5	1	I2C multiplexer	NXP PCA9546APW	TSSOP-16
J0	1	USB connector	Molex 0675031020	
J1A, J1B, J1C, J1D				
”	4	Micro-match connector	TE Connectivity 188275-6	
J2	1	SWD header	FCI 20021521-00010T1LF	


SW
S0
Y0
XIN
XOUT
C3
10p
C4
10p
+5V
U2
VDD DI
VSS DO
WS2812B
U1
WS2812B
R11
500
LED
Q0
MMBT3904
+5V
USB-
USB+
R10 27
R9 27
J0
VBUS
D-
D+
ID
GND
MP1
MP2
MP3
MP4
USB
R0
DNL
C0
DNL
+4V5
R4A 6.8k AD5 R3A 3.3k
R2A 6.8k AD4 R1A 3.3k
R5A 10k SCL1
R6A 10k SDA1
J1A
R4B 6.8k AD7 R3B 3.3k
R2B 6.8k AD6 R1B 3.3k
R5B 10k SCL0
R6B 10k SDA0
J1B
R4C 6.8k AD2 R3C 3.3k
R2C 6.8k AD8 R1C 3.3k
R5C 10k SCL3
R6C 10k SDA3
J1C
R4D 6.8k AD3 R3D 3.3k
R2D 6.8k AD9 R1D 3.3k
R5D 10k SCL2
R6D 10k SDA2
J1D
AN1
AN2
VCC
GND
SCL
SDA
+3V3
SCL
SDA
R15 10k
R14 10k
U5
SCL
SDA
A0
A1
A2
RESET
VCC
SC0
SD0
SC1
SD1
SC2
SD2
SC3
SD3
GND
SCL0
SDA0
SCL1
SDA1
SCL2
SDA2
SCL3
SDA3
PCA9546APWT
C16
100n
PA0/PGMEN0
PA1/PGMEN1
PA2/PGMEN2
PA3
PA4/PGMNCMD
PA5/PGMRDY
PA6/PGMNOE
PA7/XIN32/PGMNVALID
PA8/XOUT32/PGMM0
PA9/PGMM1
PA10/PGMM2
PA11/PGMM3
PA12/PGMD0
PA13/PGMD1
PA14/PGMD2
PA15/PGMD3
PA16/PGMD4
PA17/PGMD5/AD0
PA18/PGMD6/AD1
PA19/PGMD7/AD2
PA20/PGMD8/AD3
PA21/PGMD9/AD8
PA22/PGMD10/AD9
PA23/PGMD11
PA24/PGMD12
PA25/PGMD13
PA26/PGMD14
PA27/PGMD15
PA28
PA29
PA30
PA31
PB0/AD4
PB1/AD5
PB2/AD6
PB3/AD7
TDI/PB4
TDO/TRACESWO/PB5
TMS/SWDIO/PB6
TCK/SWCLK/PB7
XOUT/PB8
XIN/PGMCK/PB9
DDM/PB10
DDP/PB11
ERASE/PB12
PB13/DAC0
PB14/DAC1
ADVREF
VDDIN
VDDOUT
NRST
TST
JTAGSEL
U0A
SDA
SCL
LEDG
LED
AD2
AD3
AD8
AD9
SW
AD4
AD5
AD6
AD7
SWO
SWDIO
SWCLK
XOUT
XIN
USB-
USB+
NRST
+4V5
R7
330
R8
680
C1
100n
+3V3
+1V2
C13
1u
+5V
+3V3
C7
100n
U3
TLV70233
C8
1u
LEDG
R13
200
D0
+5V
+4V5
C9
100n
U4
TLV70245
C10
1u
+1V2
C6
100n
C14
100n
C12
100n
+3V3
C15
100n
C2
100n
C11
100n
C5
100n
VDDCORE
VDDIO
VDDPLL
GND
DAP
U0B
+3V3
R12
10k
SWDIO
SWCLK
SWO
NRST
J2
SWDIO
SWDCLK
SWO
NRST
NC
NC
VCC
GND
GND
GNDDET
SWD
1175-1629-ND
Title
Size
A
Number
Revision
Date:
12/18/2015
Sheet of
File:
C:\Users\..\d.SchDoc
Drawn By:

D.3 Wye adapter

D.3.1 Bill of materials

Designators	Qty	Description	Value	Footprint
C1, C2	2	Capacitor	Yageo CC0603KRX7R6BB225	0603
C3, C4	2	Capacitor	Yageo CC0603KRX7R7BB223	0603
C5, C6, C7	3	Capacitor	Yageo CC0603KRX7R8BB103	0603
J1, J2, J3	3	Micro-match connector	TE Connectivity 188275-6	
R1, R2	2	Resistor	Yageo RC0603JR-0747KL	0603
R3, R4, R13, R14, R15, R16				
”	6	Resistor	Yageo RC0603JR-071KL	0603
R5, R6, R7, R8, R19				
”	5	Resistor	Yageo RC0603JR-0710KL	0603
R9, R10	2	Resistor	Yageo RC0603JR-072K2L	0603
R11, R12	2	Resistor	Yageo RC0603JR-07510RL	0603
R17, R18	2	Resistor	Yageo RC0603JR-07120KL	0603
U1, U2	2	Quad op-amp	TI OPA4376AIPWR	TSSOP-14
U3	1	Shunt reference	TI LM4041DIM7-1.2	SC-70
U4	1	EEPROM	Microchip 24LC01BT-I/LT	SC-70



Title		
Size A	Number	Revision
Date: 12/18/2015		Sheet of
File: C:\Users\..\y.SchDoc		Drawn By:

D.4 Connecting cables

The analog pickup, digital interface, and wye adapter are attached by standard 0.05 inch pitch ribbon cables with 6 conductors terminated by Micro-Match insulation displacement connectors. The digital interface is attached to a computer by a standard USB cable wtih a type Mini-B connector. A selection of suitable parts is given in the following table.

Item	Part number
Ribbon cable	3M 3365/06
Ribbon cable	Molex F2807S-6-050-55
Micro-Match connector	TE 7-215083-6
Micro-Match connector	Wurth 690157000672
USB mini-B cable	Qualtek 3021003-03
USB mini-B cable	Molex 0887328602
USB mini-B cable	Assmann AK672M/2-1-GR

Appendix E

Source Code

E.1 Data capture software

E.1.1 leebomatic.py

```
#!/usr/bin/python

"""Simple Python program that streams data from a USB board to stdout
in the format expected by nilm-insert.  Messages will be printed to
stderr in the event of a timeout or dropped sample.  The program may
be cleanly terminated by SIGINT or SIGTERM, and will terminate itself
if there is a USB communication error."""

serial_path = "/dev/ttyACM0"
channel_map = [0, 1, 2, 3, 4, 5, 6, 7]  # which channels to output

import numpy as np
import serial, signal, sys

def start():
    global port
    # open the serial port
    port = serial.Serial(serial_path, 3000000, timeout=0.01)
    # turn off the ADC in case it was left on
    port.write(b'o')
    # clear out old data of uncertain provenance
    while len(port.read()) == 1: pass
    # now we can turn on the ADC for real
    port.write(b's')

def print_data(raw_data):
    channels = 8   # total number of ADC channels (almost always 8)
    # confirm the correct amount of data between alignment words
    if len(raw_data) != 2 * (channels + 1):
        sys.stderr.write('Corrupted data: length {0}\n'.format(len(raw_data)))
        return
    # convert to int16 and chop off the leading status word
    real_data = np.fromstring(raw_data, dtype=np.int16)[1:]
    # convert the desired output channel data to strings
    string_outputs = ['{0:6}'.format(real_data[i]) for i in channel_map]
    # print the output
    print(' '.join(string_outputs))

def capture():
    # data capture loop
    first_time = True
    while True:
```

```
        buf = bytearray()
        # read bytes from serial until we hit an alignment sequence
        while len(buf) < 2 or buf[-2] != 0x7F or buf[-1] != 0x80:
            char = port.read()
            if len(char) == 0:  # timeout, exit the loop
                sys.stderr.write('Timeout\n')
                break
            buf.append(ord(char))
        else:  # only executed if we found alignment sequence
            # send bytes up to but not including alignment sequence
            # (except for the first row of data, which is odd length)
            if first_time:
                first_time = False
            else:
                print_data(buffer(buf[:-2]))

def stop(*args):
    try:
        # stop transmission
        port.write(b'o')
        # clear out anything left in the pipes
        while len(port.read()) == 1: pass
        # all done
        port.close()
    except:
        # if something goes wrong while exiting, give up and exit
        pass
    sys.exit(0)

# run this program
signal.signal(signal.SIGINT, stop)
signal.signal(signal.SIGTERM, stop)
try:
    start()
    capture()
finally:
    stop()
```

E.1.2 prep.py

```
#!/usr/bin/python

"""Standalone prep script for use with C capture program"""

import numpy as np
import os, signal, subprocess, time

import nilmdb.client.numpyclient
from nilmdb.utils.time import seconds_to_timestamp, now as time_now

FREQ = 3000  # Raw sample rate, in Hz
NCHANNELS = 8  # Number of sensor channels (always 8 for D boards)
NHARM = 4  # Number of odd harmonics (e.g. 4 => 1, 3, 5, 7)
STEP = 25  # Number of samples between Fourier transform outputs

CHUNK_SIZE = 120  # Number of Fourier transform outputs to be batched.

FIFO_PATH = "/tmp/serialsocket"

run_capture = True

# calibration data (TODO: load from file)
K = np.eye(4)  # current matrix
L = np.eye(4)  # voltage matrix

y = 2*np.arange(NHARM) + 1

```

```
28 def main():
29     global run_capture
30 
31     signal.signal(signal.SIGUSR1, handler)
32     signal.signal(signal.SIGINT, handler)
33     signal.signal(signal.SIGTERM, handler)
34 
35     data_ts_inc = 1e6*STEP/FREQ  # microseconds between FFT outputs
36 
37     print("Python: initializing")
38     client = nilmdb.client.numpyclient.NumpyClient("http://localhost/nilmdb")
39     streams = ['/data/prep-{}'.format(char) for char in 'abcd']
40     inserters = [client.stream_insert_numpy_context(stream).__enter__()
41                  for stream in streams]
42     try: os.unlink(FIFO_PATH)
43     except: pass
44     os.mkfifo(FIFO_PATH)
45     daemon = subprocess.Popen(['capture', FIFO_PATH], bufsize=1,
46                               stdin=subprocess.PIPE, stdout=None)
47 
48     y = 2*np.arange(NHARM) + 1)
49     buffer = np.zeros((4, CHUNK_SIZE, 1+2*NHARM))
50     index = 0
51     while run_capture:
52         print("Python: beginning interval")
53         try:
54             daemon.stdin.write('start\n')
55             fifo = open(FIFO_PATH, 'rb')
56             data_ts = time_now()
57             print("Python: FIFO open, starting capture")
58             while run_capture:
59                 data = np.fromfile(fifo, np.complex64, NHARM * NCHANNELS)
60                 if len(data) != NHARM * NCHANNELS: raise IOError("FIFO error")
61                 data.shape = (NHARM, NCHANNELS)
62                 E = data[:,0::2].T
63                 S = data[:,1::2].T
64                 I = K.dot(S)
65                 Vdot = L.dot(E)
66                 V = Vdot * y * 2 * pi * 1j * SAMPLE_RATE
67                 C = np.conj(I) * (1j)**(1-y) * (V/abs(V))[:,0:1]**y
68                 buffer[:, index, 0] = int(data_ts)
69                 buffer[:, index, 1::2] = np.real(C)
70                 buffer[:, index, 2::2] = np.imag(C)
71                 index += 1
72                 data_ts += data_ts_inc
73                 if index == CHUNK_SIZE:
74                     for ins, buf in zip(inserters, buffer):
75                         ins.insert(buf)
76                         ins.update_end(int(data_ts))
77                         ins.finalize()
78                     index = 0
79                 # TODO: make sure there isn't too much timestamp drift
80             print("Python: closing interval due to run_capture flag")
81             daemon.stdin.write('stop\n')
82         except IOError:
83             print("Python: closing interval due to IOError")
84         finally:
85             fifo.close()
86             for i in inserters: i.finalize()
87         print("Python: interval closed")
88         time.sleep(0.1)
89     print("Python: terminating")
90     daemon.terminate()
91     os.unlink(FIFO_PATH)
92 
93 def handler(signum, frame):
94     global run_capture
95     run_capture=False
```

```
96
97 if __name__=="__main__":
98     main()
```

E.1.3 capture.c

```
1  #include <fcntl.h>
2  #include <poll.h>
3  #include <signal.h>
4  #include <stdint.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <termios.h>
9  #include <unistd.h>
10
11 #include <fftw3.h>
12
13 #define DEFAULT_SERIAL "/dev/serial/by-id/usb-MIT_NILM-if00"
14
15 #define NCHANNELS 8  // number of sensor channels
16 #define NCYCLE 50    // number of samples in one line cycle
17 #define NHARM 4      // number of harmonics to keep, e.g. 2 means 1st & 3rd
18 #define STEP 25      // number of samples to read between each FFT
19                      // (STEP = NCYCLE means that FFT windows do not overlap)
20
21 // A buffer stores up to BUFSIZE outputs in case writes to the output file block.
22 // Note: the ring buffer implementation assumes that BUFSIZE is >= 2.
23 #define BUFSIZE 16384
24
25 // stdin and stdout are plain-text, line-buffered channels for control.
26 // Binary data flows in over a file descriptor for the serial port and
27 // flows out over a file descriptor for the output file or FIFO.
28
29 // Summary of functions defined below:
30 // start()
31 //   * open the output file
32 //   * open and configure the serial port
33 //   * flush old data from system serial buffers
34 //   * tell the serial device to start streaming data
35 // stop()
36 //   * close the output file
37 //   * tell the serial device to stop streaming data
38 //   * close the serial device
39 // align()
40 //   * read bytes until the alignment sequence is received
41 //   * return the number of bytes read, not including the alignment word
42 // step()
43 //   * read in a 2 byte status word
44 //   * read in a 16 bit data value for each channel
45 //   * read in the 2 byte alignment word
46 //   * convert data to floats and load into input buffer
47 //   * increment the input buffer index
48 //   * return the status word
49 // read_command(timeout)
50 //   * poll stdin for input with the specified timeout
51 //   * if input is available, read line from stdin
52 //   * for inputs "start" and "stop", return special status code
53 //   * for all other inputs, handle internally and return 0
54 // write_data()
55 //   * if the big data buffer is full, return -1
56 //   * copy data from FFT output buffer to the big data buffer
57 //   * write out as much data as possible from the big data buffer without blocking
58 // loop()
59 //   * call start()
60 //   * call align()
61 //   * call step() enough times to fully initialize all buffers
```

```
62 //   * LOOP:
63 //       - call read_command() to process stdin and check if we should stop
64 //       - call step() STEP times
65 //       - FFT the input buffer into the output buffer
66 //       - call write_data()
67 // exit_handler()
68 //   * call stop()
69 //   * exit
70 // main()
71 //   * initialize program data
72 //   * LOOP:
73 //       - call input() to process input and check if we should start
74 //       - call loop()
75 //       - call stop()
76
77 fftwf_plan fft;
78 int input_index;  // counts from 0 to NCYCLE-1
79 float in_data[NCYCLE][NCHANNELS];
80 fftwf_complex out_data[NCYCLE/2+1][NCHANNELS];
81
82 // This is a ring buffer.
83 // It is empty when next_output_byte == buffer_index * sizeof(output_sample_t).
84 typedef fftwf_complex output_sample_t[NHARM][NCHANNELS];
85 output_sample_t buffer[BUFSIZE];
86 int buffer_index;  // counts from 0 to BUFSIZE-1
87 int next_output_byte;  // counts from 0 to sizeof(buffer)-1
88
89 char *serial_path, *output_path;
90 int serial_fd, output_fd;
91
92 int start() {
93   // Reset the output buffer.
94   buffer_index = 0;
95   next_output_byte = 0;
96   // Open the output file.  In order to avoid race conditions with the
97   // data consumer when the output file is a FIFO, we must open the
98   // output file *before* checking whether the serial device is present.
99   output_fd = open(output_path, O_WRONLY | O_CREAT | O_TRUNC | O_APPEND, 0755);
100  if (output_fd < 0) {
101    printf("C: error opening output file\n");
102    return -1;
103  }
104  // Open the serial port.
105  serial_fd = open(serial_path, O_RDWR | O_NOCTTY | O_SYNC);
106  if (serial_fd < 0) {
107    printf("C: error opening serial port\n");
108    return -1;
109  }
110  // Put the serial port in non-canonical mode. This is important!
111  // Also set reads to block for up to 1 second if no data is available.
112  struct termios tty = {.c_cflag = CS8, .c_cc = {[VTIME] = 10}};
113  if (tcsetattr(serial_fd, TCSANOW, &tty) < 0) {
114    printf("C: error configuring serial port\n");
115    return -1;
116  }
117  // Turn data transmission off in case it was left on.
118  if (write(serial_fd, "o", 1) < 0) return -1;
119  // Clean out any data stuck in the pipes by reading until
120  // no serial data is available for 100 milliseconds.
121  while (poll(&(struct pollfd){serial_fd, POLLIN}, 1, 100) == 1) {
122    if (read(serial_fd, &in_data, sizeof(in_data)) < 0) return -1;
123  }
124  // Turn data transmission on again.
125  if (write(serial_fd, "s", 1) < 0) return -1;
126  // Return zero upon success.
127  return 0;
128 }
129
```

```
130 void stop() {
131   // Close the output file
132   close(output_fd);
133   // Try to turn data transmission off
134   if (poll(&(struct pollfd){serial_fd, POLLOUT}, 1, 0) == 1) {
135     write(serial_fd, "o", 1);
136   }
137   // Close the serial device
138   close(serial_fd);
139 }
140
141 int align() {
142   // Align with the data stream by watching for the 16-bit word 0x807F.
143   uint8_t c0 = 0, c1 = 0;
144   int i = -2;
145   while (c0 != 0x7F || c1 != 0x80) {
146     c0 = c1;
147     if (read(serial_fd, &c1, 1) <= 0) {
148       printf("C: error reading from serial port during align\n");
149       return -1;
150     }
151     ++i;
152     // If it's been longer than the longest possible gap, give up.
153     if (i == NCHANNELS*2 + 2) {
154       printf("C: alignment failed after %d bytes read\n", i);
155       return -1;
156     }
157   }
158   // Return the number of extra data bytes (0 for immediate alignment).
159   return i;
160 }
161
162 int step() {
163   struct {
164     uint16_t status;
165     int16_t channels[NCHANNELS];
166     uint16_t alignment_word;
167   } sample;
168   // Read in the status, one sample per channel, and the next alignment word.
169   for (int n, i = 0; i < sizeof(sample); i += n) {
170     n = read(serial_fd, ((uint8_t *) &sample) + i, sizeof(sample) - i);
171     if (n <= 0) {
172       printf("C: error reading from serial port\n");
173       return -1;
174     }
175   }
176   if (sample.alignment_word != 0x807F) {
177     printf("C: lost data alignment\n");
178     return -1;
179   }
180   // Load the channel data into the FFT input buffer.
181   for (int channel = 0; channel < NCHANNELS; ++channel) {
182     in_data[input_index][channel] = (float) sample.channels[channel];
183   }
184   // Increment the buffer index for next time.
185   if (++input_index == NCYCLE) input_index = 0;
186   // If all went well, return the status word.
187   return (int) sample.status;  // cast from uint16_t to int is never negative
188 }
189
190 // All inputs except for "start" and "stop" are handled internally.
191 #define INPUT_START 1
192 #define INPUT_STOP 2
193 int read_command(int timeout) {
194   while (poll(&(struct pollfd){STDIN_FILENO, POLLIN}, 1, timeout) == 1) {
195     char input[100];
196     if (fgets(input, sizeof(input), stdin) == NULL) return -1;
197     // TODO: add commands to e.g. set LED colors.
```

```
198     if (strcmp(input, "start\n") == 0) {
199       return INPUT_START;
200     } else if (strcmp(input, "stop\n") == 0) {
201       return INPUT_STOP;
202     } else if (strcmp(input, "exit\n") == 0) {
203       stop();
204       exit(0);
205     } else {
206       printf("Valid commands: start stop exit\n");
207     }
208   }
209   return 0;
210 }
211
212 int write_data() {
213   // Compute the buffer index which was most recently transmitted.
214   // next_output_byte == 0 is a special case because it wraps around.
215   int last_index = (next_output_byte == 0 ? BUFSIZE - 1 :
216                     (next_output_byte - 1) / sizeof(output_sample_t));
217   // If we are still on that index, then the buffer is full.
218   if (last_index == buffer_index) {
219     printf("C: output buffer full; aborting\n");
220     return -1;
221   }
222   // Now we know for sure that there is space for one more sample in the buffer.
223   for (int i = 0; i < NHARM; ++i) {
224     memcpy(buffer[buffer_index][i], out_data[2*i+1],
225            sizeof(fftwf_complex) * NCHANNELS);
226   }
227   if (++buffer_index == BUFSIZE) buffer_index = 0;
228   // Output as much as we can from the buffer without blocking.
229   while (poll(&(struct pollfd){output_fd, POLLOUT}, 1, 0) == 1) {
230     // Compute the number of bytes to be written.
231     int n = buffer_index * sizeof(output_sample_t) - next_output_byte;
232     // If we are all caught up, return.
233     if (n == 0) return 0;
234     // If we have to wrap around to the beginning, write up to the end.
235     if (n < 0) {
236       n = sizeof(buffer) - next_output_byte;
237     }
238     // Now see how many bytes we're actually able to write.
239     n = write(output_fd, ((uint8_t *) buffer) + next_output_byte, n);
240     if (n <= 0) {
241       printf("C: output write error\n");
242       return -1;
243     }
244     // Update the pointer to the next byte to be written.
245     if ((next_output_byte += n) == sizeof(buffer)) next_output_byte = 0;
246   }
247 }
248
249 int loop() {
250   if (start() < 0) return -1;
251   // Get aligned with the input stream.
252   if (align() < 0) return -1;
253   // Fill the input buffer
254   for (int i = 0; i < NCYCLE; ++i) {
255     if (step() < 0) return -1;
256   }
257   // Do the actual processing
258   for (;;) {
259     for (int i = 0; i < STEP; ++i) {
260       if (read_command(0) == INPUT_STOP) return 0;
261       int n = step();
262       if (n < 0) return -1;
263       if (n > 0) {
264         printf("C: data flags 0x%04x received\n", n);
265       }
```

```
    }
    fftwf_execute(fft);
    if (write_data() < 0) return -1;
  }
  return 0;
}

void exit_handler(int signo) {
  stop();
  exit(0);
}

int main(int argc, char** argv) {
  if (argc < 2) {
    printf("Usage: %s output_path [serial_path]\n", argv[0]);
    exit(0);
  }
  output_path = argv[1];
  serial_path = argc < 3 ? DEFAULT_SERIAL : argv[2];

  signal(SIGINT, exit_handler);
  signal(SIGTERM, exit_handler);

  // Set up the FFT plan
  fft = fftwf_plan_many_dft_r2c(1, (int[]){NCYCLE}, NCHANNELS,    // transform shape
                                in_data[0], NULL, NCHANNELS, 1,   // input layout
                                out_data[0], NULL, NCHANNELS, 1,  // output layout
                                FFTW_PRESERVE_INPUT);             // flags

  // Loop forever
  for(;;) {
    // Wait for the user to type "start"
    while (read_command(-1) != INPUT_START);
    // loop() returns upon an error condition or the user typing "stop"
    if (loop() < 0) {
      printf("C: serial communication stopped due to error\n");
    }
    stop();
  }
}
```

E.1.4 `build`

```
#!/bin/bash
gcc -O3 -std=c99 -ocapture capture.c -lfftw3f
```

E.2 NilmDB integrated components

Note: these files are integrated with the `nilmtools` package, which is distributed as part of NilmDB. They depend on other parts of NilmDB and will not operate as standalone Python scripts.

E.2.1 `matrix.py`

```
#!/usr/bin/python
import nilmtools.filter, numpy, numpy.linalg, ast, yaml

def main(argv = None):
    f = nilmtools.filter.Filter()
    parser = f.setup_parser("Non-Contact Matrix Multiply Tool")
    group = parser.add_argument_group("Matrix options (manual configuration)")
    group.add_argument("-c", "--columns", action="store",
```

```
                        help = "which columns of the input stream to use")
    # e.g. -c [1,3,5,7]
    group.add_argument("-m", "--matrix", action="store",
                        help = "the matrix by which to multiply")
    # e.g. -m [[1,0,0],[0,1,0],[0,0,1]]
    group.add_argument("-i", "--integrate", action="store", type=int, metavar="SIZE",
                        help = "integrate and filter output with this window size")
    # e.g. -i 50

    group = parser.add_argument_group("Matrix options (set from config file)")
    exc = group.add_mutually_exclusive_group()
    exc.add_argument("--yaml-voltage", action="store", metavar="PATH",
                      help = "config file to use for standard voltage setup")
    exc.add_argument("--yaml-current", action="store", metavar="PATH",
                        help = "config file to use for standard current setup")
    args = f.parse_args(argv)

    if args.yaml_voltage:  # extract voltage from raw data
        with open(args.yaml_voltage) as y:
            config = yaml.load(y)
        columns = config['setup']['e_sensors']
        matrix = numpy.array(config['voltage_matrix'])
        if config['setup']['e_integrate']:
            integrate = (config['setup']['sample_freq'] //
                         config['setup']['line_freq'])
        else:
            integrate = False
    elif args.yaml_current:  # extract current from raw data
        with open(args.yaml_current) as y:
            config = yaml.load(y)
        columns = config['setup']['m_sensors']
        matrix = numpy.array(config['current_matrix'])
        integrate = False
    else:  # manual configuration
        assert args.matrix != None, 'matrix is required'
        # matrix: numpy array of shape (num_output_columns, num_input_columns)
        matrix = numpy.array(ast.literal_eval(args.matrix))
        # columns: list, e.g. [1, 3, 5, 7]
        # default to columns 0, 1, 2, etc.
        if args.columns == None:
            columns = numpy.arange(matrix.shape[1])
        else:
            columns = ast.literal_eval(args.columns)
        integrate = args.integrate

    m = MatrixFilter(columns, matrix, integrate)
    f.process_numpy(m.process)

class MatrixFilter:
    def __init__(self, columns, matrix, integrate):
        self.columns = columns
        self.matrix = matrix
        if integrate:
            self.setup_integration(integrate)
            self.integrate = True
        else:
            self.integrate = False
    def process(self, data, interval, args, insert, final):
        timestamps, inputs = data[:,0:1], data[:,1:][:,self.columns]
        # the following is equivalent to outputs[i] = matrix.dot(inputs[i])
        outputs = inputs.dot(self.matrix.T)
        if self.integrate:
            size = len(self.filter) - 1  # assumes len(self.filter) is odd
            new_outputs = numpy.empty_like(outputs[:-size])
            for i in range(outputs.shape[1]):
                new_outputs[:,i] = numpy.convolve(self.filter,
                                                  outputs[:,i], 'valid')
            insert(numpy.hstack([timestamps[size/2:-size/2],
```

```
                                    new_outputs]))
            if final:
                return len(data)
            else:
                return len(data) - size
        else:
            insert(numpy.hstack([timestamps, outputs]))
            return len(data)
    def setup_integration(self, cycle_length):
        N = cycle_length // 2  # assumes cycle_length is even
        t = numpy.arange(1-N, N)  # -(N-1) to +(N-1) inclusive
        n = numpy.arange(1, N)    # 1 to N-1 inclusive
        self.filter = numpy.sum(numpy.sin(numpy.pi*n*t[:,None]/N)/n/N, 1)

if __name__ == "__main__":
    main()
```

E.2.2 haunting.py

```
#!/usr/bin/python
import nilmtools.filter, numpy, numpy.linalg, ast, yaml

def main(argv = None):
    f = nilmtools.filter.Filter()
    parser = f.setup_parser("Non-Contact Haunting Detector")
    group = parser.add_argument_group("Haunting options (manual configuration)")
    group.add_argument("-c", "--columns", action="store",
                       help = "which columns of the input stream to use")
    # e.g. -c [1,3,5,7]
    group.add_argument("-m", "--matrix", action="store",
                       help = "the matrix by which to multiply")
    # e.g. -m [[1,0,0],[0,1,0],[0,0,1]]
    group.add_argument("-n", "--ncycle", action="store", type=int, metavar="NUM",
                       help = "number of samples in a line cycle")

    group = parser.add_argument_group("Haunting options (set from config file)")
    group.add_argument("--yaml", action="store", metavar="PATH",
                      help = "config file to use for standard haunting setup")
    args = f.parse_args(argv)

    if args.yaml:
        with open(args.yaml) as y:
            config = yaml.load(y)
        columns = config['setup']['m_sensors']
        c_mat = numpy.array(config['current_matrix'])
        s_mat = numpy.array(config['sensor_matrix'])
        matrix = numpy.eye(len(s_mat)) - s_mat.dot(c_mat)
        ncycle = config['setup']['sample_freq'] // config['setup']['line_freq']
    else:  # manual configuration
        assert args.matrix != None, 'matrix is required'
        # matrix: numpy array of shape (num_output_columns, num_input_columns)
        matrix = numpy.array(ast.literal_eval(args.matrix))
        # columns: list, e.g. [1, 3, 5, 7]
        # default to columns 0, 1, 2, etc.
        if args.columns == None:
            columns = numpy.arange(matrix.shape[1])
        else:
            columns = ast.literal_eval(args.columns)
        # number of samples in one line cycle
        ncycle = args.ncycle

    f.process_numpy(process, args=(columns, matrix, ncycle))

def process(data, interval, args, insert, final):
    columns, matrix, ncycle = args
    timestamps, inputs = data[:,0:1], data[:,1:][:,columns]
    # the following is equivalent to outputs[i] = matrix.dot(inputs[i])
```

```
    outputs = inputs.dot(matrix.T)
    # figure out how many line cycles we have to process.
    n = len(outputs) // ncycle
    if n == 0: return 0
    # take the fourier transform to find the component at line frequency
    reshaped_outputs = outputs[:n*ncycle].reshape((n,ncycle,-1))
    ejw = numpy.exp(1j*2*numpy.pi*numpy.arange(ncycle)/ncycle)
    ft_of_outputs = numpy.sum(reshaped_outputs * ejw[:,None], axis=1)
    # output sqrt(P_1^2 + Q_1^2 + ... + P_k^2 + Q_k^2) for each row.
    norms = numpy.linalg.norm(ft_of_outputs, axis=1)[:,None]
    insert(numpy.hstack([timestamps[::ncycle][:n], norms]))
    # there is data left over iff len(data) was not divisible by ncycle.
    return n*ncycle

if __name__ == "__main__":
    main()
```

E.2.3 transient.py

```
from numpy import diff, std, zeros
from numpy.linalg import matrix_power, pinv
from scipy.signal import bessel, lfilter, tf2ss
import nilmtools.filter

"""
Transient detection parameters:

* Transient start detection
    k1 = certainty necessary to declare that a transient has started.
         larger values decrease false positives.
         smaller values decrease detection delay.
    k2 = minimum detectable step size, relative to the standard deviation.
         larger values improve noise immunity.
         smaller values improve sensitivity.
* Transient end detection
    k3 = buffer size, in samples. this is also the minimum
           "quiet length" that must separate two transients.
         larger values spuriously combine transients.
         smaller values spuriously split transients.
    k4 = transient end threshhold. condition takes the form
           std(window) / std(diff(window)) < self.k4.
         For additive white noise, 1/sqrt(2) is the optimal value.
* Moving average filter
    k5 = order of filter (integer)
    k6 = cutoff frequency divided by Nyquist frequency
"""

def main(argv = None):
    f = nilmtools.filter.Filter()
    parser = f.setup_parser("Transient Finder")
    group = parser.add_argument_group("Transient parameters")
    group.add_argument("--k1", type=float, default=33.0,
                       help="transient start certainty threshhold")
    group.add_argument("--k2", type=float, default=1.0,
                       help="minimum detectable transient step size")
    group.add_argument("--k3", type=int, default=30,
                       help="buffer size")
    group.add_argument("--k4", type=float, default=0.75,
                       help="transient end threshhold")
    group.add_argument("--k5", type=int, default=3,
                       help="order of mean filter")
    group.add_argument("--k6", type=float, default=0.017,
                       help="mean filter cutoff frequency")
    args = f.parse_args(argv)
    finder = TransientFinder(args.k1, args.k2, args.k3, args.k4, args.k5, args.k6)
    f.process_numpy(finder.process)

```

```
49
50  # Bessel low-pass filter with initialization (Chornoboy 1990)
51  class LowPass:
52      def __init__(self, init_length=30, filt_order=3, filt_cutoff=1./60):
53          self.b, self.a = bessel(filt_order, filt_cutoff)
54          assert self.a[0] == 1.0
55          # State space description of the linear system:
56          # q[i+1] = A*q[i] + B*x[i]  and  y[i] = C*q[i] + D*x[i].
57          A, B, C, D = tf2ss(self.b, self.a)
58          # convert to observer canonical form (for compatibility with lfilter)
59          A, B, C = A.T, C.T, B.T
60          # Let X = column vector x[0]...x[init_length-1] and
61          # Y = column vector y[0]...y[init_length-1].
62          # We construct F and G such that  Y - X = F*q[0] + G*X.
63          F = zeros((init_length, filt_order))
64          G = zeros((init_length, init_length))
65          for i in range(init_length):
66              F[i] = C.dot(matrix_power(A,i))
67              G[i:].flat[::init_length+1] = (C.dot(matrix_power(A,i-1)).dot(B)
68                                             if i>0 else D-1)
69          # Suppose that we want to choose q[0] = M*X for some constant matrix M.
70          # We minimize the error (F*M + G) * X by choosing M to be the projection
71          # given by -pseudoinverse(F)*G.
72          self.M = -pinv(F).dot(G)
73      def reset(self, init_data):
74          """Initialize filter state to minimize error over init_data"""
75          self.z = self.M.dot(init_data)
76          self.update(init_data)
77      def update(self, in_data):
78          """Run filter on a list of inputs, returning a list of outputs"""
79          out_data, self.z = lfilter(self.b, self.a, in_data, zi=self.z)
80          return out_data
81
82  # CUSUM step detector (Granjon 2012)
83  class StepWatcher:
84      def __init__(self, threshold, offset):
85          self.threshold, self.offset = threshold, offset
86      def reset(self):
87          self.s_argmin, self.s_min = None, 0
88          self.s = self.g = 0
89      def update(self, index, sample):
90          self.s += sample - self.offset
91          if self.s < self.s_min:
92              self.s_argmin, self.s_min = index, self.s
93          self.g = max(self.g + sample - self.offset, 0)
94          return self.s_argmin if self.g > self.threshold else None
95
96  # Now we're ready to actually find some transients
97  class TransientFinder:
98      def __init__(self, k1, k2, k3, k4, k5, k6):
99          self.up_watcher = StepWatcher(k1, k2)
100         self.down_watcher = StepWatcher(k1, k2)
101         self.mean_filter = LowPass(k3, k5, k6)
102         self.k3, self.k4 = k3, k4
103         self.reset()
104     def reset(self):
105         self.buffer = []
106         self.in_transient = False
107         self.starting = True
108         # transient_over will handle the remainder of member initialization.
109     def transient_start(self, timestamp):
110         """Update state after a transient has been detected"""
111         # Discard data until transient start is at the head of the buffer
112         while self.buffer[0][0] < timestamp:
113             self.pop()
114         self.in_transient = True
115         # Discard one sample of the transient, so that it lasts at least 1 sample
116         if self.pop()[0] > timestamp:
```

```
117             print("Warning: transient start not buffered. Input is pathological.")
118     def transient_over(self):
119         """Update state after the end of a transient"""
120         self.starting = False
121         self.in_transient = False
122         self.up_watcher.reset()
123         self.down_watcher.reset()
124         self.mean_filter.reset(self.window)
125         self.std = max(std(self.window), 0.001)  # TODO: make minimum non-arbitrary
126     def pop(self):
127         """Remove one element from the buffer and append it to this step's output"""
128         timestamp, value = self.buffer.pop(0)
129         self.result.append((timestamp, self.in_transient))
130         return (timestamp, value)
131     def update(self, timestamp, value):
132         self.result = []
133         # Pop the oldest sample if the buffer is full.
134         if len(self.buffer) >= self.k3:
135             self.pop()
136         # Add new sample to buffer.
137         self.buffer.append((timestamp, value))
138         # If we're in a transient, check whether it has ended.
139         if self.in_transient or self.starting:
140             # Only check for transient end if buffer is full.
141             if len(self.buffer) == self.k3:
142                 self.window = [b[1] for b in self.buffer]
143                 if std(self.window) <= std(diff(self.window)) * self.k4:
144                     self.transient_over()
145         # If we're not in a transient, check whether one has started.
146         else:
147             err = value - self.mean_filter.update([value])[0]
148             u_step = self.up_watcher.update(timestamp, err / self.std)
149             d_step = self.down_watcher.update(timestamp, -err / self.std)
150             if u_step is not None:
151                 self.transient_start(u_step)
152             elif d_step is not None:
153                 self.transient_start(d_step)
154         # Return the old samples that were popped during this timestep.
155         # Number of samples returned is between 0 and k3-1 inclusive.
156         return self.result
157     def process(self, data, interval, args, insert_func, final):
158         """Wrapper around update() for nilm filter interface."""
159         for timestamp, power in data[:, :2]:  # Only want T and P1
160             output = self.update(timestamp, power)
161             if output: insert_func(output)
162         if final:
163             self.reset()  # reset if we're done with an interval
164         return len(data)  # we always process all data in an interval
165 
166 # Helper function to run transient detector over a 1d list
167 def transients(arr, *args, **kwargs):
168     t = TransientFinder(*args, **kwargs)
169     result = []
170     for i, d in enumerate(arr):
171         result.extend([p[1] for p in t.update(i,d)])
172     return result
173 
174 
175 if __name__ == "__main__":
176     main()
```

E.3 Standalone Python acquisiton software

E.3.1 adc.py

```
#!/usr/bin/python

import numpy as np
import multiprocessing, serial, sys

channels = 8   # number of ADC channels
cycle = 50     # length of a line cycle, in samples
harmonics = 4  # number of harmonics to keep, e.g. 3 means 1st, 3rd, 5th

class AdcError(IOError): pass

# The AdcDaemon handles serial communiation and alignment in
# a separate process. The interface is as follows:
# daemon = AdcDaemon("/dev/ttyUSB0")
# daemon.start()        # begin streaming data from device
# daemon.read(timeout)  # read the next line of data bytes
# daemon.stop()         # stop streaming data and terminate the process
#
# Chunks of bytes coming in over the serial port are separated by
# the alignment sequence 0x7F followed by 0x80. It is the user's
# responsibility to ensure that this sequence never occurs within
# a chunk, or else the chunk will be split into two chunks.
class AdcDaemon(multiprocessing.Process):
    def __init__(self, path):
        multiprocessing.Process.__init__(self)
        self.parent_pipe, self.child_pipe = multiprocessing.Pipe()
        self.serial_path = path
    # Use AdcDaemon.start() to call AdcDaemon.run() in a separate process
    def run(self):
        try:
            self.ser = serial.Serial(self.serial_path, 3000000, timeout=0.01)
            buf = bytearray()
            # turn off the ADC in case it was left on
            self.ser.write(b'o')
            # clear out old data of uncertain provenance
            while len(self.ser.read()) == 1: pass
            # now we can turn on the ADC for real
            self.ser.write(b's')
            while True:
                # read bytes from serial until we hit an alignment sequence
                while len(buf) < 2 or buf[-2] != 0x7F or buf[-1] != 0x80:
                    char = self.ser.read()
                    if len(char) == 0: break  # timeout, exit the loop
                    buf.append(ord(char))
                else:  # only executed if we found alignment sequence
                    # send bytes up to but not including alignment sequence
                    # note hack to make this file compatible with Python 2 and 3
                    if sys.version >= '3':
                        self.child_pipe.send_bytes(buf[:-2])
                        buf.clear()
                    else:
                        self.child_pipe.send_bytes(buffer(buf[:-2]))
                        buf = bytearray()
                if self.child_pipe.poll():
                    if self.child_pipe.recv_bytes() == b'exit':
                        break
        except:
            print("ADC daemon exiting due to {}".format(sys.exc_info()[0]))
        # attempt to close ADC cleanly
        try:
            # stop transmission
            self.ser.write(b'o')
            # clear out anything left in the pipes
            while len(self.ser.read()) == 1: pass
            # all done
            self.ser.close()
        except:
            pass
```

```
    # read() and stop() are to be called by the original process.
    def read(self, timeout=1):
        if self.parent_pipe.poll(timeout):
            return self.parent_pipe.recv_bytes()
        raise AdcError("ADC disconnected")
    def stop(self):
        self.parent_pipe.send_bytes(b'exit')

# The ADC class provides a nice abstraction using the Python 'with' statement.
# 'adc' is the sole instance of class _Adc. Example usage:
# with adc:
#     for i in range(100):
#         print(adc.read_fft())
class _Adc:
    def __init__(self, path):
        self.path = path
        self.buffer = np.empty((cycle, channels))
    def __enter__(self):
        self.status = np.int16(0)
        self.buffer[:] = 0
        self.buffer_index = 0
        # start data flowing in
        self.child = AdcDaemon(self.path)
        self.child.start()
        self.child.read()  # skip the first, partial chunk
        self.read_fft(cycle)  # suppress startup transient
    def get_status(self):
        result, self.status = self.status, 0
        return result
    # read_raw() returns a single line of raw ADC data.
    def read_raw(self):
        raw = self.child.read()
        if len(raw) != 2 * (channels + 1):  # wrong number of bytes
            print("Received corrupted data")
            self.status |= 0x8000  # indicate error using reserved status bit
            return None
        raw16 = np.fromstring(raw, dtype=np.int16)
        self.status |= raw16[0]
        return raw16[1:]
    # read_fft(n) reads the next n lines of raw data, computes the
    # FFT over the most recent line cycle, and slices out only the
    # desired harmonics. When n is equal to the number of samples per
    # line cycle, there is no overlap.
    def read_fft(self, step=cycle):
        for i in range(step):
            line = self.read_raw()
            if line is not None:
                self.buffer[self.buffer_index] = line
            self.buffer_index += 1
            self.buffer_index %= cycle
        fft = np.fft.rfft(self.buffer, axis=0)
        return fft[1::2][:harmonics] / (cycle/2)
    def __exit__(self, type, value, traceback):
        self.child.stop()
adc = _Adc("/dev/ttyACM0")
```

E.3.2 capture.py

```
from adc import adc, cycle, channels, harmonics
import matplotlib.pyplot as plt
from scipy.signal import medfilt
import numpy as np

# Helper functions for working with the ADC.

# Print a representation of an array of status words.
def print_status(status):
```

```
10      if status != 0:
11          print('********')
12      if (status & (1<<15)) != 0:
13          print('Corrupted serial communication')
14      if (status & (1<<13)) != 0:
15          print('Switch pressed')
16      if (status & (1<<8)) != 0:
17          print('FIFO error')
18      for i in range(8):
19          if (status & (1<<i)) != 0:
20              print('Channel {} saturated'.format(i))
21
22  # Capture a specified number of seconds of raw data
23  # first axis = time
24  # second axis = channel number
25  def capture_raw(nlines):
26      data = np.empty((nlines, channels))
27      for line in data:
28          line[:] = adc.read_raw()
29      print_status(adc.get_status())
30      return data
31
32  # Make a nice split-screen plot of voltages and currents.
33  def plot_raw(data):
34      fig = plt.figure()
35      ax = plt.subplot(2, 1, 1)
36      plt.xlabel("Samples (50 per line cycle)")
37      plt.ylabel("Derivative of voltage")
38      vplots = plt.plot(data[:,::2])
39      plt.subplot(2, 1, 2, sharex=ax)
40      plt.ylabel("Current")
41      iplots = plt.plot(data[:,1::2])
42      return fig, vplots, iplots
43
44  def show_raw(time=1):
45      with adc:
46          data = capture_raw(time*3000)
47      plot_raw(data)
48      plt.show()
49
50  # Capture a specified number of seconds of fft data.
51  # The result is a three-dimensional array:
52  # first axis = time
53  # second axis = harmonic number
54  # third axis = channel number
55  def capture_fft(nlines=1, step=cycle):
56      data = np.empty((nlines, harmonics, channels), np.complex)
57      for fft in data:
58          fft[:] = adc.read_fft(step)
59      print_status(adc.get_status())
60      return data
61
62  # Plot prep data for all four channels.
63  def plot_prep(data, smoothing=5):
64      fig = plt.figure()
65      ax = None
66      for i in range(channels//2):
67          ax = plt.subplot(2, 2, i+1, sharex=ax)
68          plt.xlabel('Channel {}'.format(i))
69          offsets = data[:, 0, 2*i] / np.abs(data[:, 0, 2*i]) / 1j
70          for j,h in [[0,1],[1,3],[2,5]]:
71              d = data[:, j, 2*i+1] / offsets**h
72              plt.plot(medfilt(np.real(d), smoothing), label='P{}'.format(h))
73              plt.plot(medfilt(np.imag(d), smoothing), label='Q{}'.format(h))
74              plt.legend(frameon=False)
75
76  def show_prep(time=1, step=cycle, smoothing=5):
77      with adc:
```

```
78          data = capture_fft(time*3000//step, step)
79      plot_prep(data, smoothing)
80      plt.show()
81
82  # load fft data in the format stored by captured.c in the daemons repository.
83  def load_fft(path):
84      return np.fromfile(path, np.complex64).reshape((-1, harmonics, channels))
```

E.3.3 scope.py

```
1   #!/usr/bin/python
2
3   import adc, capture, multiprocessing, numpy, serial, sys
4
5   fr = numpy.hstack([0, 1/1j, 2/1j/numpy.arange(2,49), 3/2j/49])
6   h = numpy.roll(numpy.fft.irfft(fr,100), 50)[1:]
7   def integ(data):
8       return numpy.convolve(h, data, 'same')
9   def center(data):
10      return data - numpy.average(data)
11
12  q = multiprocessing.Queue()
13  def plot():
14      import matplotlib.pyplot as plt
15      data = q.get(block=True)
16      fig, vplots, iplots = capture.plot_raw(data)
17      def update():
18          data = None
19          while not q.empty(): # get everything which is clogged in the queue
20              data = q.get()
21          if data is not None:
22              for i,p in enumerate(vplots):
23                  p.set_ydata(data[:,2*i])
24              for i,p in enumerate(iplots):
25                  p.set_ydata(data[:,2*i+1])
26              fig.canvas.draw()
27      timer = fig.canvas.new_timer(interval=20)
28      timer.add_callback(update)
29      timer.start()
30      plt.show()
31  multiprocessing.Process(target=plot).start()
32  while True:
33      try:
34          with adc.adc:
35              while True:
36                  data = capture.capture_raw(800)
37                  # process the data
38                  for i in range(4):
39                      data[:,2*i] = integ(center(data[:,2*i]))
40                      data[:,2*i+1] = center(data[:,2*i+1])
41                  q.put(data)
42      except KeyboardInterrupt:
43          sys.exit()
44      except adc.AdcError:
45          pass
```

E.4 Digital interface firmware

E.4.1 Makefile

```
1   # Makefile for Atmel SAM4S using cmsis and GNU toolchain.
2
3   # The variables $(SRC), $(INC), $(LIB) are defined in path.mk.
4   include path.mk
```

```
5
6  # Object file location and linker script
7  OBJ = $(SRC:%.c=obj/%.o) $(LIB)
8  LD_SCRIPT = asf/sam/utils/linker_scripts/sam4s/sam4s4/gcc/%.ld
9
10 # Compiler and linker flags. Here be dragons.
11 CFLAGS  = -mlittle-endian -mthumb -mcpu=cortex-m4
12 CFLAGS += -g -O3 $(INC:%=-I%) -std=c99 -Wall
13 CFLAGS += -DARM_MATH_CM4 -D'__SAM4S4B__' -D'BOARD=USER_BOARD'
14 LFLAGS  = $(CFLAGS) -T$(@:bin/%.elf=$(LD_SCRIPT))
15 LFLAGS += -Wl,--entry=Reset_Handler -Wl,--gc-sections
16
17 # Targets
18 .PHONY: all clean gdb
19 .SECONDARY: $(OBJ)
20 all: bin/flash.bin bin/flash.elf
21 clean:
22         -rm -rf obj bin
23 gdb: bin/flash.elf  # attach debugger to a self-powered board
24         @arm-none-eabi-gdb -x gdb/debug
25 flash: bin/flash.elf  # load firmware to a debugger-powered board
26         @arm-none-eabi-gdb -batch -x gdb/program
27 bin/%.bin: bin/%.elf
28         arm-none-eabi-objcopy -O binary $< $@
29 bin/%.hex: bin/%.elf
30         arm-none-eabi-objcopy -O ihex $< $@
31 bin/%.elf: $(OBJ)
32         @mkdir -p $(dir $@)
33         $(info LD $@)
34         @arm-none-eabi-gcc $(LFLAGS) -o $@ $(OBJ)
35 obj/%.o: %.c
36         @mkdir -p $(dir $@)
37         $(info CC $<)
38         @arm-none-eabi-gcc $(CFLAGS) -c $< -o $@
```

E.4.2 path.mk

```
1  SRC  = $(wildcard src/*.c)
2  INC  = inc
3
4  # Subfolders to compile and include from asf/sam/drivers
5  SAM_DRIVERS = adc efc matrix pdc pio pmc pwm rstc tc udp usart wdt
6  SRC += $(wildcard $(SAM_DRIVERS:%=asf/sam/drivers/%/*.c))
7  INC += $(SAM_DRIVERS:%=asf/sam/drivers/%)
8
9  # Subfolders to compile and include from asf/common/services
10 COMMON_SERVICES  = clock gpio sleepmgr ioport
11 COMMON_SERVICES += usb usb/udc usb/class/cdc usb/class/cdc/device
12 SRC += $(wildcard $(COMMON_SERVICES:%=asf/common/services/%/sam4s/*.c))
13 SRC += $(wildcard $(COMMON_SERVICES:%=asf/common/services/%/sam/*.c))
14 SRC += $(wildcard $(COMMON_SERVICES:%=asf/common/services/%/*.c))
15 INC += $(COMMON_SERVICES:%=asf/common/services/%)
16
17 # Subfolders to compile and include from asf/sam/utils
18 SRC += asf/sam/utils/syscalls/gcc/syscalls.c
19 SRC += asf/sam/utils/cmsis/sam4s/source/templates/system_sam4s.c
20 SRC += asf/sam/utils/cmsis/sam4s/source/templates/gcc/startup_sam4s.c
21 INC += asf/sam/utils
22 INC += asf/sam/utils/header_files
23 INC += asf/sam/utils/preprocessor
24 INC += asf/sam/utils/cmsis/sam4s/include
25
26 # Subfolders to compile and include from asf/common/utils
27 SRC += $(wildcard asf/common/utils/interrupt/*.c)
28 SRC += $(wildcard asf/common/utils/stdio/*.c)
29 INC += asf/common/utils
30
```

```
# All other ASF header paths
INC += asf/common/boards
INC += asf/thirdparty/CMSIS/Include

# Specific source exclusions, because ASF is inconsistent
EXCLUDE  = asf/sam/drivers/wdt/wdt_sam4l.c
EXCLUDE += asf/sam/drivers/adc/adc2.c
EXCLUDE += asf/common/services/usb/udc/udc_dfu_small.c
SRC := $(filter-out $(EXCLUDE),$(SRC))

# Precompiled libraries
LIB = asf/thirdparty/CMSIS/Lib/GCC/libarm_cortexM4l_math.a
```

E.4.3 src/analog.c

```
#include <adc.h>
#include <arm_math.h>
#include <buffer.h>
#include <pio.h>
#include <pmc.h>
#include <sysclk.h>
#include <tc.h>

#include "analog.h"
#include "fir_filter.h"

// ADC runs at 96 kHz per channel. There are 8 channels.
// Data is low-pass filtered by a zero-phase FIR filter which
// passes frequencies below 660 Hz and rejects above 1.5 kHz.
// Result is decimated by 32x and output at 3 kHz per channel.

// Raw ADC values are between 0 and 4095 inclusive.
// We subtract 2048 and the filter applies a DC gain of 8,
// so output values are nominally between -16384 and 16383.
// The filter L-infinity norm is 1.035, so pathological inputs
// can produce outputs ever so slightly outside of this range
// (but they will still fit comfortably in signed 16 bits).

// Frequency  Filter gain (/8)
//    60 Hz     0.9967
//   180 Hz     0.9951
//   300 Hz     0.9938
//   420 Hz     0.9915
//   540 Hz     0.9801
//   660 Hz     0.9443
//  1500 Hz     0.0501

#define NUM_CHANNELS 8
static enum adc_channel_num_t channels[NUM_CHANNELS] = {4,5,6,7,8,2,9,3};

// Stage 1: decimate 8 times with a 16-tap FIR.
#define DEC1 8
#define NTAPS1 16
static const q15_t coeffs1[NTAPS1] = {  // see util/fir.py
   2305,  2296,  4354,  6819,  9420, 11818, 13661, 14663,
  14663, 13661, 11818,  9420,  6819,  4354,  2296,  2305
};
static q15_t buffer1[NUM_CHANNELS][2*NTAPS1];

// Stage 2: decimate 4 times with a 32-tap FIR.
#define DEC2 4
#define NTAPS2 32
static const q15_t coeffs2[NTAPS2] = {  // see util/fir.py
     64,   147,   234,   261,   139,  -195,  -715, -1257,
  -1529, -1196,     0,  2105,  4880,  7823, 10297, 11710,
  11710, 10297,  7823,  4880,  2105,     0, -1196, -1529,
  -1257,  -715,  -195,   139,   261,   234,   147,    64
```

```
};
static q15_t buffer2[NUM_CHANNELS][2*NTAPS2];


// The output format consists of the eight 16-bit channel values
// followed by the alignment word 0x807F and a 16-bit status word.
// (Note that it is never possible for 0x80 or 0x7F to be the
// most significant byte of the channel value or status word.)

static volatile int fifo_running;
static uint16_t status_mask;
static int fifo_write(uint16_t value) {
  int r = push(value);
  if (r != BUFFER_OK) {
    status_mask |= ERROR_FIFO;
  }
  return r;
}


void analog_init(void) {
  // Enable the switch
  pmc_enable_periph_clk(ID_PIOA);
  pio_set_input(PIOA, PIO_PA23, PIO_PULLUP);

  // Initialize the ADC
  pmc_enable_periph_clk(ID_ADC);
  adc_init(ADC, sysclk_get_cpu_hz(), 20000000, ADC_STARTUP_TIME_8);
  adc_configure_timing(ADC, 0, ADC_SETTLING_TIME_0, 1);  // "fast"

  // Set up ADC channel sequence
  adc_configure_sequence(ADC, channels, NUM_CHANNELS);
  adc_start_sequencer(ADC);
  for (int i = 0; i < NUM_CHANNELS; ++i)
    adc_enable_channel(ADC, i);  // by *sequence #*, not channel #

  // Enable end-of-conversion (EOC) interrupt for the last channel
  adc_enable_interrupt(ADC, 1 << channels[NUM_CHANNELS-1]);
  NVIC_EnableIRQ(ADC_IRQn);

  // Trigger from timer 0
  pmc_enable_periph_clk(ID_TC0);
  tc_init(TC0, 0,                     // channel 0
          TC_CMR_TCCLKS_TIMER_CLOCK1  // source clock (CLOCK1 = MCLK/2)
          | TC_CMR_CPCTRG             // up mode with automatic reset on RC match
          | TC_CMR_WAVE               // waveform mode
          | TC_CMR_ACPA_CLEAR         // RA compare effect: clear
          | TC_CMR_ACPC_SET);         // RC compare effect: set
  tc_write_ra(TC0, 0, 1);
  tc_write_rc(TC0, 0, 625);  // frequency = (120MHz/2)/625 = 96 kHz
  adc_configure_trigger(ADC, ADC_TRIG_TIO_CH_0, 0);

  // Start everything
  adc_start(ADC);
  tc_start(TC0, 0);
}

void analog_start(void) {
  flush();
  status_mask = 0;
  fifo_running = 1;
}

void analog_stop(void) {
  fifo_running = 0;
  flush();
}

```

```
121 void ADC_Handler(void) {
122   static uint32_t sample_index;  // counts up forever
123
124   for (int i = 0; i < NUM_CHANNELS; i++) {
125     q15_t adc =  adc_get_channel_value(ADC, channels[i]) - 2048;
126     // TODO: make ADC error logging PER-CHANNEL.
127     if (adc < -2000 || adc > 2000)
128       status_mask |= (1 << i);
129     buffer1[i][sample_index & (2*NTAPS1-1)] = adc;
130   }
131
132   // Update the switch status
133   if (pio_get(PIOA, PIO_INPUT, PIO_PA23) == 0)
134     status_mask |= SWITCH_PRESSED;
135
136   // We distribute the computational load of running FIR filters
137   // so that the USB interrupt never gets delayed too long.
138   // Let N = sample_index. Run stage 1 on channel N % 8.
139   // If N % 4 = 0, run stage 2 on channel (N/4) % 8.
140   // This computation is hard-coded for NUM_CHANNELS = DEC1.
141   // Quantities DEC1, DEC2, NTAPS1, NTAPS2 must be powers of 2.
142
143   // STAGE 1
144   int base = (sample_index / DEC1) * DEC1;
145   int channel = sample_index & (NUM_CHANNELS-1);
146   q15_t result = fir_filter(coeffs1, NTAPS1, buffer1[channel], base);
147   buffer2[channel][(base / DEC1) & (2*NTAPS2-1)] = result;
148
149   // STAGE 2
150   if (!(sample_index & (DEC2-1))) {
151     base = (sample_index / DEC1 / DEC2) * DEC2;
152     channel = (sample_index / DEC2) & (NUM_CHANNELS-1);
153     result = fir_filter(coeffs2, NTAPS2, buffer2[channel], base);
154
155     // Output result to the FIFO.
156     // If the status word is successfully written, clear it.
157     if (fifo_running) {
158       if (channel == 0) {
159         if (fifo_write(0x807F) == BUFFER_OK &&
160             fifo_write(status_mask) == BUFFER_OK) {
161           status_mask = 0;
162         }
163       }
164       // If the buffer is full, stop writing data to the buffer.
165       // The host will have to issue a new "start" command to resume.
166       if (fifo_write(result) != BUFFER_OK) {
167         fifo_running = 0;
168       }
169     }
170   }
171   ++sample_index;
172 }
```

E.4.4 src/buffer.c

```
1  #include "arm_math.h"
2  #include "buffer.h"
3
4  // Thread safe as long as there is only one producer and one consumer.
5  // Only the consumer is allowed to call flush().
6  // This looks simple but TRUST ME, IT ISN'T.
7
8  // head points to the oldest valid element.
9  // tail points to the element after the newest valid element.
10 // head == tail means EMPTY, not full.
11
12 #define BUFFER_SIZE 16384  // must be a power of 2
```

```
13 static volatile int16_t buffer[BUFFER_SIZE];
14 static volatile int head;
15 static volatile int tail;
16
17 static inline int next(int val) {
18   return (val + 1) & (BUFFER_SIZE - 1);
19 }
20
21 inline int push(uint16_t data) {
22   if (next(tail) == head)
23     return BUFFER_FULL;
24   buffer[tail] = data;
25   tail = next(tail);
26   return BUFFER_OK;
27 }
28
29 inline int pop(uint16_t *data) {
30   if (tail == head)
31     return BUFFER_EMPTY;
32   *data = buffer[head];
33   head = next(head);
34   return BUFFER_OK;
35 }
36
37 inline void flush() {
38   head = tail;
39 }
40
41 inline int buffer_full() {
42   return (next(tail) == head);
43 }
```

E.4.5 src/debug.c

```
1  #include "debug.h"
2
3  // When a Black Magic Probe is attached, this function prints
4  // a string to the standard output of GDB (on the host).
5  void print(const char *str) {
6  #ifdef __DEBUG__
7    int data[3] = {0x02, (int) str, 0};
8    while(*(str++)) ++(data[2]);  // count length of string
9    __asm__ volatile ("mov r0, $0x05; mov r1, %0; bkpt $0xAB" : : "r" (data) : "r0", "r1");
10 #endif
11 }
```

E.4.6 src/led.c

```
1  #include <pdc.h>
2  #include <pio.h>
3  #include <pwm.h>
4  #include <sysclk.h>
5
6  // LEDs are connected to PWM3H which is PA7.
7  // We have to run PWM3 in synchronous mode with PWM0 in order to get
8  // the duty cycle to update via DMA.
9  // PWM1 interrupts at 50 Hz to start new DMA transfers.
10 // PWM2 is not used.
11
12 // Every other element in the buffer is a dummy value for PWM channel 0
13 // and the buffer ends with two extra zeros:
14 // 2 LEDs x 24 bits per LED + terminating zero = 49
15 #define LED_BUFFER_SIZE 49
16 static uint16_t led_buffer[LED_BUFFER_SIZE];
17 static pdc_packet_t led_packet;
18
```

```
19 // The flag is used to ensure that LED colors update all at once.
20 static volatile int led_flag = 0;
21 static uint32_t led1_color, led2_color;
22
23 // Colors are in standard hex format, i.e. 0xFF0000 is red
24 void led_update(uint32_t led1, uint32_t led2) {
25   while (led_flag);  // wait for previous update to finish
26   led1_color = led1;
27   led2_color = led2;
28   led_flag = 1;
29 }
30 void led1_update(uint32_t led1) {
31   led_update(led1, led2_color);
32 }
33 void led2_update(uint32_t led2) {
34   led_update(led1_color, led2);
35 }
36
37 void led_init(void) {
38   pmc_enable_periph_clk(ID_PWM);
39   pwm_clock_t clock_setting = {
40     .ul_clka = 20000000,  // 20 MHz
41     .ul_clkb = 7500,      // 7.5 kHz
42     .ul_mck = sysclk_get_cpu_hz()
43   };
44   pwm_init(PWM, &clock_setting);
45
46   // Turn on channel 0 for high speed data output.
47   pwm_channel_t channel = {
48     .channel = 0,
49     .ul_duty = 0,
50     .ul_period = 25,  // 50 ns * 25 = 1.25 us, as required by WS2812
51     .ul_prescaler = PWM_CMR_CPRE_CLKA,
52     .polarity = PWM_LOW,  // inverted because PA11 is a high output
53     .b_sync_ch = true
54   };
55   pwm_channel_init(PWM, &channel);
56   pwm_sync_init(PWM, PWM_SYNC_UPDATE_MODE_2, 0);  // duty cycle via DMA
57   pwm_pdc_set_request_mode(PWM, PWM_PDC_UPDATE_PERIOD_ELAPSED, 0);
58   pwm_channel_enable(PWM, 0);
59
60   // Connect PA11 to PWM channel 0 output H.
61   // (This must happen AFTER the PWM channel is configured.)
62   pmc_enable_periph_clk(ID_PIOA);
63   pio_set_peripheral(PIOA, PIO_PERIPH_B, PIO_PA11);
64
65   // Turn on channel 1 to trigger DMA requests periodically.
66   pwm_channel_t trigger = {
67     .channel = 1,
68     .ul_period = 150,  // Overflow at 50 Hz
69     .ul_prescaler = PWM_CMR_CPRE_CLKB
70   };
71   pwm_channel_init(PWM, &trigger);
72   pwm_channel_enable_interrupt(PWM, 1, 1);
73   pwm_channel_enable(PWM, 1);
74   NVIC_EnableIRQ(PWM_IRQn);
75 }
76
77 void PWM_Handler(void) {
78   if (led_flag) {  // if new colors are available...
79     // Load new values into the duty cycle buffer.
80     // A 0 bit corresponds to a duty cycle of 8 (400 us on / 850 us off).
81     // A 1 bit corresponds to a duty cycle of 16 (800 us on / 450 us off).
82     // WS2812B expects G then R then B with the most significant bit first.
83     for (int i = 0; i < 8; ++i) {
84       led_buffer[i]  = 8 + 8 * (1 & (led1_color >> (15-i)));  // green 1
85       led_buffer[i+8] = 8 + 8 * (1 & (led1_color >> (23-i)));  // red 1
86       led_buffer[i+16] = 8 + 8 * (1 & (led1_color >> (7-i)));   // blue 1
```

```
87        led_buffer[i+24] = 8 + 8 * (1 & (led2_color >> (15-i)));  // green 2
88        led_buffer[i+32] = 8 + 8 * (1 & (led2_color >> (23-i)));  // red 2
89        led_buffer[i+40] = 8 + 8 * (1 & (led2_color >> (7-i)));   // blue 2
90      }
91      led_flag = 0;
92    }
93
94    // Start a new DMA transfer
95    led_packet.ul_addr = (uint32_t) led_buffer;
96    led_packet.ul_size = LED_BUFFER_SIZE;
97    pdc_tx_init(PDC_PWM, &led_packet, 0);
98    pdc_enable_transfer(PDC_PWM, PERIPH_PTCR_TXTEN);
99
100   // Acknowledge the interrupt
101   pwm_channel_get_interrupt_status(PWM);
102 }
```

E.4.7 src/main.c

```
1  #include <efc.h>
2  #include <pmc.h>
3  #include <sysclk.h>
4  #include <udi_cdc.h>
5  #include <wdt.h>
6
7  #include "analog.h"
8  #include "buffer.h"
9  #include "debug.h"
10 #include "led.h"
11 #include "usb.h"
12
13 int main(void) {
14   // Switch over to the crystal oscillator
15   sysclk_init();
16
17   // Disable the built-in watchdog timer
18   wdt_disable(WDT);
19
20   // Initialize peripherals
21   usb_init();
22   analog_init();
23   led_init();
24
25   // Green
26   led2_update(0x000F00);
27
28   for (;;) {
29     // Transmit data if we can
30     while(udi_cdc_is_tx_ready()) {
31       uint16_t data;
32       if(pop(&data) != BUFFER_OK)  // buffer empty
33         break;
34       udi_cdc_write_buf(&data, sizeof(data));
35     }
36     if (buffer_full()) {
37       led2_update(0x2F0000);
38     }
39     // Receive data in our free time
40     if (udi_cdc_is_rx_ready()) {
41       int c = udi_cdc_getc();
42       if (c < 0) {
43         print("Read error");
44       } else if (c == 's') {
45         led2_update(0x00002F);
46         analog_start();
47       } else if (c == 'o') {
48         analog_stop();
```

```
49          led2_update(0x000F00);
50        } else if (c == 'g') {
51          // Clear GPNVM 1 to boot from ROM instead of flash
52          efc_perform_command(EFC0, EFC_FCMD_CGPB, 1);
53        }
54      }
55    }
56  }
```

E.4.8 src/usb.c

```
1   // From module: USB Device CDC (Single Interface Device)
2   #include <udi_cdc.h>
3
4   // From module: USB Device Stack Core (Common API)
5   #include <udc.h>
6   #include <udd.h>
7
8   #include "usb.h"
9   #include "conf_usb.h"
10  #include "debug.h"
11
12  static volatile bool b_cdc_enable = false;
13
14  void usb_init(void){
15    print("starting usb framework\n");
16    udc_start();
17  }
18
19  //Callback hooks for the USB framework
20  void usb_resume_action(void){
21  #ifdef USB_DBG
22    print("resuming usb\n");
23  #endif
24  }
25  void usb_suspend_action(void){
26  #ifdef USB_DBG
27    print("suspending usb\n");
28  #endif
29  }
30  void usb_sof_action(void){
31    /*  if(b_cdc_enable){
32      print("sof\n");
33      }*/
34  }
35
36  //Call back hooks for the CDC framework
37  bool usb_cdc_enable(uint8_t port){
38  #ifdef USB_DBG
39    print("cdc enabled\n");
40  #endif
41    b_cdc_enable = true;
42    return true;
43  }
44
45  bool usb_cdc_disable(uint8_t port){
46  #ifdef USB_DBG
47    print("cdc disabled\n");
48  #endif
49    b_cdc_enable = false;
50    return true;
51  }
52
53  void usb_cdc_set_dtr(uint8_t port, bool b_enable){
54  #ifdef USB_DBG
55    if(b_enable)
56      print("cdc: host open\n");
```

```
57    else
58      print("cdc: host closed\n");
59  #endif
60  }
61  void usb_rx_notify(uint8_t port){
62  #ifdef USB_DBG
63    print("cdc rx notify\n");
64  #endif
65    //  udi_cdc_putc('A');
66  }
67  void usb_cdc_config(uint8_t port, usb_cdc_line_coding_t * cfg){
68  #ifdef USB_DBG
69    print("cdc config\n");
70  #endif
71  }
```

E.4.9 inc/analog.h

```
1  #ifndef __ANALOG_H__
2  #define __ANALOG_H__
3
4  void analog_init(void);
5  void analog_start(void);
6  void analog_stop(void);
7
8  // Status word
9  // Bit 15: reserved, must be 0
10 // Bit 14: reserved, must be 0
11 #define SWITCH_PRESSED 1<<13
12 // Bits 9-12: available
13 #define ERROR_FIFO 1<<8
14 // Bits 0-7: channel saturation error
15
16 #endif
```

E.4.10 inc/buffer.h

```
1  #include "arm_math.h"
2
3  #ifndef __BUFFER_H__
4  #define __BUFFER_H__
5
6  #define BUFFER_OK 0
7  #define BUFFER_FULL -1
8  #define BUFFER_EMPTY -2
9
10 int push(uint16_t data);
11 int pop(uint16_t *data);
12 void flush();
13 int buffer_full();
14
15 #endif
```

E.4.11 inc/conf_board.h

```
1  #ifndef __CONF_BOARD_H__
2  #define __CONF_BOARD_H__
3
4  /*required by usbc driver*/
5  #define CONF_BOARD_USB_PORT
6
7  #endif
```

E.4.12 inc/conf_clock.h

```
// THIS FILE IS INCLUDED BY:
// asf/common/services/clock/sysclk.h

// XTAL frequency: 12MHz
// System clock source: PLLA
// System clock prescaler: divided by 2
// PLLA source: XTAL
// PLLA output: XTAL * 20 / 1
// System clock: 12 * 20 / 1 / 2 = 120MHz
// PLLB source: XTAL
// PLLB output: XTAL * 4 / 1
// USB Clock: PLLB = 48 MHz
#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_PLLACK
#define CONFIG_SYSCLK_PRES          SYSCLK_PRES_2
#define CONFIG_PLL0_SOURCE          PLL_SRC_MAINCK_XTAL
#define CONFIG_PLL0_MUL             20
#define CONFIG_PLL0_DIV             1

#define CONFIG_PLL1_SOURCE          PLL_SRC_MAINCK_XTAL
#define CONFIG_PLL1_MUL             4
#define CONFIG_PLL1_DIV             1
#define CONFIG_USBCLK_SOURCE        USBCLK_SRC_PLL1
#define CONFIG_USBCLK_DIV           1
```

E.4.13 inc/conf_sleepmgr.h

```
// THIS FILE IS INCLUDED BY:
// asf/common/services/sleepmgr/sam/sleepmgr.h

// ASF sucks
```

E.4.14 inc/conf_usb.h

```
#ifndef _CONF_USB_H_
#define _CONF_USB_H_

#include "compiler.h"
#include "board.h"

#define  USB_DEVICE_VENDOR_ID             USB_VID_ATMEL
#define  USB_DEVICE_PRODUCT_ID            USB_PID_ATMEL_ASF_CDC
#define  USB_DEVICE_MAJOR_VERSION         1
#define  USB_DEVICE_MINOR_VERSION         0
#define  USB_DEVICE_POWER                 200  // mA
#define  USB_DEVICE_ATTR                (USB_CONFIG_ATTR_BUS_POWERED)
#define  USB_DEVICE_MANUFACTURE_NAME      "MIT"
#define  USB_DEVICE_PRODUCT_NAME          "NILM"

#define  UDC_VBUS_EVENT(b_vbus_high)
#define  UDC_SOF_EVENT()                  usb_sof_action()
#define  UDC_SUSPEND_EVENT()              usb_suspend_action()
#define  UDC_RESUME_EVENT()               usb_resume_action()
#define  UDI_CDC_PORT_NB 1
#define  UDI_CDC_ENABLE_EXT(port)         usb_cdc_enable(port)
#define  UDI_CDC_DISABLE_EXT(port)        usb_cdc_disable(port)
#define  UDI_CDC_RX_NOTIFY(port)          usb_rx_notify(port)
#define  UDI_CDC_TX_EMPTY_NOTIFY(port)
#define  UDI_CDC_SET_CODING_EXT(port,cfg) usb_cdc_config(port,cfg)
#define  UDI_CDC_SET_DTR_EXT(port,set)    usb_cdc_set_dtr(port,set)
#define  UDI_CDC_SET_RTS_EXT(port,set)
#define  UDI_CDC_DEFAULT_RATE             3000000
#define  UDI_CDC_DEFAULT_STOPBITS         CDC_STOP_BITS_1
#define  UDI_CDC_DEFAULT_PARITY           CDC_PAR_NONE
#define  UDI_CDC_DEFAULT_DATABITS         8

#include "udi_cdc_conf.h"
```

```
34 #include "usb.h"
35
36 #endif // _CONF_USB_H_
```

E.4.15 inc/debug.h

```
1  #ifndef __DEBUG_H__
2  #define __DEBUG_H__
3
4  //#define __DEBUG__
5  void print(const char *str);
6
7  #endif
```

E.4.16 inc/fir_filter.h

```
1  // The number of taps must be a power of 2.
2  // len(data) must be twice the number of taps.
3  // Returns dot product of coeffs and data[data_end-num_taps:data_end],
4  // wrapping back to the end of data if necessary.
5  static inline q15_t fir_filter(const q15_t *coeffs, int num_taps,
6                                 q15_t *data, int data_end) {
7    // SIMD: we divide all indices by two and index into 32-bit pointers.
8    int32_t *c = (int32_t *) coeffs, *d = (int32_t *) data;
9    q31_t sum = 0;
10   for (int i = 0; i < num_taps / 2; ++i) {
11     // j = [(2*i + data_end - num_taps) % len(data)] / 2
12     int j = (i + data_end / 2 - num_taps / 2) & (num_taps - 1);
13     // sum += low16(c[i]) * low16(d[j]) + high16(c[i]) * high16(d[j])
14     sum = __SMLAD(c[i], d[j], sum);
15   }
16   return (q15_t) (sum >> 15);
17 }
18
19 // Equivalent (but slower) implementation:
20 //
21 // q15_t sum = 0;
22 // for (int i = 0; i < num_taps; ++i) {
23 //   int j = (i + data_end - num_taps) & (2 * num_taps - 1);
24 //   sum += coeffs[i] * data[j];
25 // }
26 // return sum;
```

E.4.17 inc/led.h

```
1  #ifndef __LED_H__
2  #define __LED_H__
3
4  void led_init(void);
5
6  // Color format is standard hex, e.g.
7  // 0xFF0000 = bright red
8  // 0x001100 = dark green
9  // 0x000080 = medium blue
10 void led_update(uint32_t led1, uint32_t led2);
11 void led1_update(uint32_t led1);
12 void led2_update(uint32_t led2);
13
14 #endif
```

E.4.18 inc/usb.h

```
1  #ifndef __USB_H__
2  #define __USB_H__
```

```

#include <usb_protocol_cdc.h>

//enable debugging
//#define USB_DBG

// Call this function to set up USB
void usb_init(void);

// Callback hooks for the USB framework
void usb_suspend_action(void);
void usb_resume_action(void);
void usb_sof_action(void);

// Callback hooks for the CDC framework
bool usb_cdc_enable(uint8_t port);
bool usb_cdc_disable(uint8_t port);
void usb_cdc_set_dtr(uint8_t port, bool b_enable);
void usb_rx_notify(uint8_t port);
void usb_cdc_config(uint8_t port, usb_cdc_line_coding_t * cfg);


#endif
```

E.4.19 inc/user_board.h

```
// THIS FILE IS INCLUDED BY:
// asf/common/boards/board.h

#define BOARD_FREQ_SLCK_XTAL        (32768U)
#define BOARD_FREQ_SLCK_BYPASS      (32768U)
#define BOARD_FREQ_MAINCK_XTAL      (12000000U)
#define BOARD_FREQ_MAINCK_BYPASS    (12000000U)
#define BOARD_OSC_STARTUP_US        15625
```

E.4.20 gdb/debug

```
target extended-remote /dev/ttyACM0
mon swdp_scan
attach 1
file bin/flash.elf
load bin/flash.elf
```

E.4.21 gdb/program

```
target extended-remote /dev/ttyACM0
mon tpwr enable
shell sleep 0.1
mon swdp_scan
attach 1
load bin/flash.elf
mon gpnvm_set 1 1
detach
mon tpwr disable
```

E.5 README files

E.5.1 programming

```
To install the digital board firmware distribution onto this Ubuntu or
Debian computer, run the following shell command from this directory:

```

```
$ bash install.sh INSTALL_PATH

For example,

$ bash install.sh ~/Desktop/firmware/

The following instructions should be used after installation completes.

*****

The firmware distribution includes the compiled .elf and .bin files,
but should you wish to recompile them run the following commands:

$ cd INSTALL_PATH/code
$ make clean
$ make

*****

To write the firmware using a JTAG debugger, attach the Black Magic
Probe to the computer via USB and attach the Black Magic Probe to the
digital board via the 10-pin debug header. (Do NOT attach the digital
board to the computer via USB -- otherwise the Black Magic Probe may
show up as /dev/ttyACM1 instead of /dev/ttyACM0.) Then run the
following commands:

$ cd INSTALL_PATH/code
$ make flash

*****

To write the firmware over USB, attach the digital board to the
computer via USB. (The Black Magic Probe must NOT be attached to the
computer -- otherwise the digital board may show up as /dev/ttyACM1
instead of /dev/ttyACM0.) Then run the following command:

$ INSTALL_PATH/sam_ba_cdc_linux/sam-ba_64

This will launch a graphical application. Follow these steps:
1. Select the board "at91sam4s4-ek" from the drop-down.
2. Click "Connect". A new window will appear.
3. In the field labeled "Send file name", enter the path
   INSTALL_PATH/code/bin/flash.bin
4. Click "Send File". (If prompted, do not lock the flash.)
5. In the "Scripts" drop-down, choose the script
   "Boot from Flash (GPNVM1)" and click "Execute".
6. Close the application.

Unplug and reattach the digital board to start the custom firmware.

*****

USB programming only works for digital boards which are running the
Atmel factory bootloader. Once a digital board has been programmed, it
is no longer running the factory bootloader and USB programming will
not work.

The digital board firmware contains a function to revert back to the
factory bootloader. To activate it, attach the digital board to the
computer via USB. Then run the following command:

$ echo g > /dev/ttyACM0

Unplug and reattach the digital board to start the factory bootloader.
```

E.5.2 nilmdb

```
This file provides brief usage examples for the three utilities which
were added to NilmDB as part of this thesis. The files are called
transient.py, matrix.py, and haunting.py. However, when compiled by
the existing NilmDB toolchain, they are accessed by the command names
nilm-transient, nilm-matrix, and nilm-haunting.


The nilm-transient command shares the same general format as all
NilmDB filters: the flags -s and -e denote the start and end times,
followed by a source path and destination path. It also takes six
optional arguments --k1 through --k6 which allow the default values of
the transient detection to be overriden. An example invocation would be

$ nilm-transient -s min -e max /data/prep-a /data/transients-a

to find the transients present in prep-a over all time.


The nilm-matrix command also shares the same general format: the flags
-s and -e denote the start and end times, followed by a source path
and destination path. The nilm-matrix command has additional required
flags -c and -m to set the columns of the input stream and the matrix
by which they should be multiplied. There is also an optional flag -i
which applies the voltage integrating filter to its output. For
example, the invocation to integrate the first voltage sensor
(assuming 50 samples per line cycle) would be

$ nilm-matrix -c [0] -m [[1]] -i 50 /data/raw /data/voltage

and the invocation to unmix 3 current sensors into 2 currents might be

$ nilm-matrix -c [1,3,5] -m [[1,2,3],[4,5,6]] /data/raw /data/current

(clearly with a different unmixing matrix determined by calibration).
The nilm-matrix command is also integrated with John Donnal's YAML
configuration format, and the --yaml-voltage or --yaml-current flags
may be used in place of the -c, -m, and -i flags to read the
corresponding values from a YAML configuration file. The standard
command lines for a non-contact NILM's process.sh script are:

$ nilm-matrix --yaml-voltage ~/Desktop/config.yml /data/raw /data/voltage
$ nilm-matrix --yaml-current ~/Desktop/config.yml /data/raw /data/current


The nilm-haunting command accepts -c and -m arguments with exactly the
same meaning as in the nilm-matrix command. For example,

$ nilm-haunting -c [1,3,5] -m [[9,8,7],[6,5,4]] /data/raw /data/current

Note that the haunting detection matrix is not the same as the
unmixing matrix M. It is actually equal to (I - M*pinv(M)).  This
command also supports YAML configuration files. The standard command
line for a non-contact NILM's process.sh script is:

$ nilm-haunting --yaml ~/Desktop/config.yml /data/raw /data/haunting


For reference, the remainder of this file lists the entire contents of
process.sh to produce prep data on a non-contact NILM.

nilm-matrix --yaml-voltage ~/Desktop/config.yml /data/raw /data/voltage
nilm-matrix --yaml-current ~/Desktop/config.yml /data/raw /data/current
nilm-haunting --yaml ~/Desktop/config.yml /data/raw /data/haunting
nilm-sinefit --yaml ~/Desktop/config.yml -i 0 /data/voltage /data/sinefit
nilm-prep --yaml ~/Desktop/config.yml -i 0 /data/current /data/sinefit /data/prep-a
nilm-prep --yaml ~/Desktop/config.yml -i 1 /data/current /data/sinefit /data/prep-b
nilm-prep --yaml ~/Desktop/config.yml -i 2 /data/current /data/sinefit /data/prep-c
```