Power **I**nformation **C**ollection **A**rchitecture

Final Design Report

Engineering 340: Senior Design 2010-2011

May 2011

Team 01: Amy Ball, Nathan Jen, Avery Sterk, Kendrick Wiersma

Abstract

The PICA system addresses several problems with power metering through the use of two metering systems, the E-meter and Smart Breakers, and one data collection center, the base station. The E-meter monitors the line voltage and current, computer power and accumulating the energy used over time. The E-Meter design centers on an MSP430 from Texas Instruments. The Smart Breakers measure circuit-by-circuit information, providing a map of power usage throughout the home or business. Additionaly solid-state relays, part of the Smart Breakers, function as a circuit interruptor in place of the traditional electromechanical breakers. The ADE776 metering chip and ATmega328 microprocessor provide measurement and control respectively for the Smart Breakers. Team PICA chose to prototype the base station using a PC application running on a Linux desktop computer. For a production model the base station would use a LEON3 soft-processor programmed onto a Spartan V FPGA running a custom Linux distribution. The team also designed a switching mode power supply to power the E-Meter and the Smart Breakers.

Team PICA developed requirements, test plans, and designed a prototype for each of the four subsystems that would become part of the team's potential production unit. During each of these stages the design team sought to provide the consumer with easy to understand data regarding power usage. The entire design process for Team PICA took place over the course of their senior year at Calvin College in Grand Rapids, MI.

Contents

List of Figures

List of Tables

Listings

1 Acronyms

AC Alternating Current . . . 26
ADC Analog-to-Digital Converter . . . 32
AES Advanced Encryption Standard . . . 25
ANSI American National Standards Institute . . . 23
CAD Computer Aided Design . . . 7
CCS4 Code Composer Studio 4 . . . 35
CF Compact Flash . . . 82
CPLD Complex Programmable Logic Device . . . 82
DC Direct Current . . . 19
DHCP Dynamic Host Configuration Protocol . . . 16
DIP Dual Inline Package . . . 42
DMM Digital Multi-Meter . . . 52
DTR Data Transmit Ready . . . 43
EM electromagnetic . . . 20
FCC Federal Communications Commission . . . 20
FPGA Field Programable Gate Array . . . 3
FPU Floating Point Unit . . . 78
GCC GNU Compiler Collection . . . 78
GNU GNU's not Unix
GPL GNU General Public License . . . 75
HTTP Hypertext Transfer Protocol . . . 17
IC Integrated Circuit . . . 42
IEC International Electrotechnical Commision . . . 111
IEEE International Electrical and Electronics Engineers . . . 47
ISR Interrupt Service Routine . . . 47
JCI Johnson Controls, Inc. . . . 50
JTAG Joint Test Area Group . . . 7
LAN Local Area Network . . . 14
LCD Liquid Crystal Display . . . 23
LED Light Emitting Diode . . . 15

2 Introduction

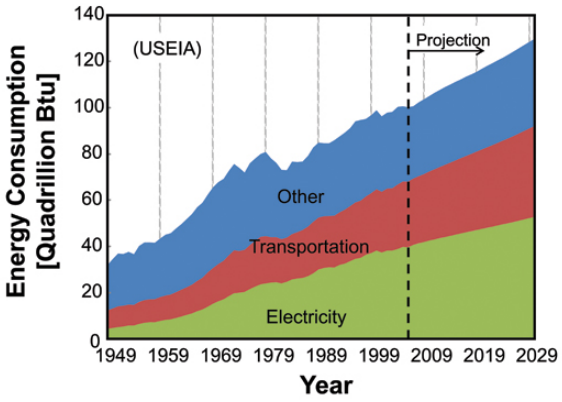
This report summarizes the work of Calvin College's senior design team 01: Team PICA during the academic year 2010-2011. Each section further breaks down a component of our system, beginning at the high-level requirements until reaching component selection. Each part of our final report relates a discussion of concepts relating to the business case for the project and the engineering design that went into producing a working prototype.

2.1 Problem Definition

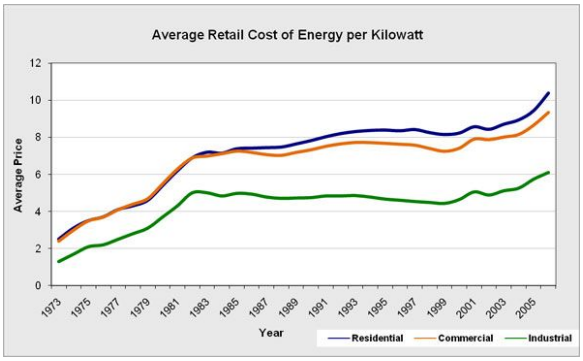
Figure 1: Standard electric meter; photo copyright Beige Alert (Creative Commons).

Standard electric meters were developed decades ago and are still used today, despite many technological advances in the last several years. Additionally, Americans have become accustomed to having access to large amounts of data, but due to the nature of standard electric metering, data regarding the usage of power is severely limited for both the consumer and electric company.

Both usage and cost of electricity continue to increase, see figures 2a and 2b respectively, causing consumers and providers to look for ways to become more efficient. With more data about electric usage available, both parties can find inefficencies and take action as appropriate. Currently, few simple or inexpensive solutions exist, and most only address part of the whole problem, giving some information to



(a) MIT Power usage data.[3]



(b) Energy cost projections.[4]

Figure 2: Power usage and cost metrics.

the consumer and none to the power company or vice-versa.

2.2 Customer Base

The target market for the entire PICA system comprises both electricity producers and electricity consumers, as set forth by the nature of the subsystems. As the power companies supply and own the electricity meters attached to the buildings to which they supply power, the PICA E-meter appeals only to the market of electricity-producing companies. The other two subsystems, the solid-state breakers and base station, target the power-consuming audience, as the devices will assist in monitoring power flow inside the building, where the power company has no presence. As these two markets are essentially exclusive in both membership and interest in the PICA subsystems, the E-meter will be able to function independently of the other consumer-targeted subsystems, and vice versa.

2.2.1 Power Companies

As power companies currently distribute the whole-building metering hardware that determines how much energy their customer used, the E-meter clearly targets power companies. In fact, the power companies own the power-measuring hardware external to the buildings to while they provide power, so only they may replace or upgrade those devices. At present, power companies send trained meter-readers to read the data from most traditional power meters under their control. The PICA E-meter subsystem aims to improve on this process by automatically sending the measurements to the power company using a means and protocol selected by the particular company. While this will require some hardware customization for each

company, the volume of company-specific production should allow the cost to develop the design to spread into a small per-unit cost.

The PICA E-meter subsystem also provides numerous more measures of power than the simple spinning-dial meters. For example, the E-meter will measure the frequency and the RMS voltage of the incoming supply lines, which help indicate the overall quality of power delivered to the customers in the area. This information may also help diagnose any observed issues with power delivery without dispatching a worker to take measurements by hand. In this way, power companies using the PICA E-meter can improve the quality of the service they provide and can save on the labor costs associated with making a site visit.

2.2.2 Power Consumers

Although the power company's customers cannot modify the metering panel installed by the power company, they are free to modify the other power distribution components inside their own buildings. The solid-state breakers fall into this category, and provide previously unavailable measurements regarding power consumption and its location within the building. However, as these breakers will replace the pre-existing breakers inside the building, the consumer must be convinced that using the PICA system is worth the trouble and cost of replacing the mechanical circuit breakers with the more feature-rich PICA breakers. To this effect, the most receptive market for the solid-state breakers includes homeowners and building managers who are curious or concerned about power usage inside their building. That is, the people for whom this information can inspire a meaningful change in practice will likely become the first adopters of the subsystem.

The product may also gain a following as an alternative to mechanical breakers in during the construction of a new home or building. This would likely require that the product already have a proven history of reliability and safety, so the previous group of cost- or environmentally-concerned individuals might have to adopt the produce first. If the PICA solid-state breakers become an alternative during construction, the net cost to the user will be lessened, as the building-to-be will not have any pre-existing breakers to discard or replace.

The base station may apply to either of these two consumer groups, as its primary purpose is to manage and interface with the other systems. It does not specifically require the solid-state breakers or the E-meter, but provides little value in a building without any installed PICA systems. The base station exists solely to

manage and collect data from other PICA subsystems, as well as format and display these measurements, so its target audience consists of power consumers whose buildings use the E-meter, the solid-state breakers, or both.

2.3 About Us

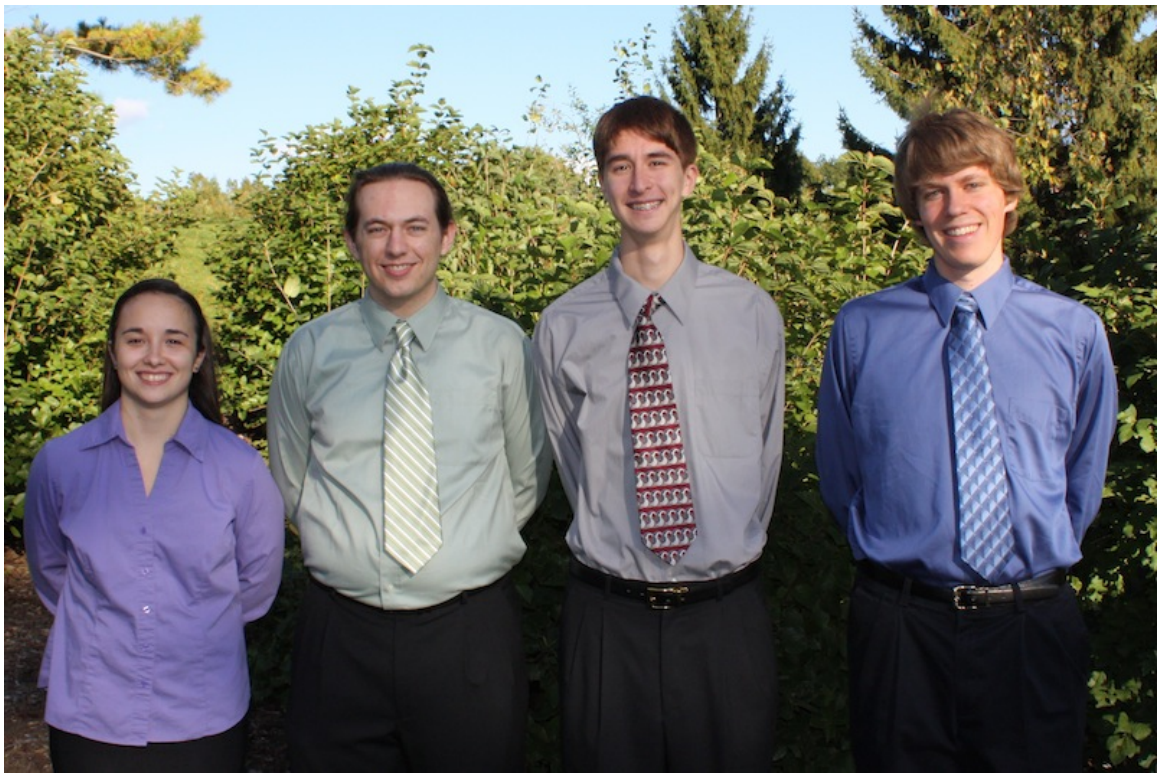


Figure 3: Team PICA left to right: Amy Ball, Kendrick Wiersma, Nathan Jen, and Avery Sterk.

Amy Ball: Amy, from Wayland, Michigan, works as an intern at Johnson Controls as part of the Systems Engineering Team. She brings good communication skills, circuit-building experience, and presentation skills to the project. Originally her section was the solid-state breakers, but the team decided to use solid-state relays for the breaker purposes. After which her new task was to build all of the power supplies for the project. Additionally, she was assigned to identify a method to match outlets to breakers.

Kendrick Wiersma: Kendrick works as an intern at Raytheon in the Electronics Center, where he performs embedded system design and verification in VHSIC Hardware Description Language (VHDL). Kendrick hails from Tucson, Arizona where he was born and raised. He brings real-world project experience and experience working with embedded hardware and software to the team. Kendrick leads the development of the E-meter, which primarily measures total power consumption, reporting data to the

power company and the PICA base station.

Nathan Jen: Nathan, from Kentwood, Michigan, has worked at Amway on the production floor and has gained involvement with club leadership at Calvin College. He will be graduating with an Engineering degree, Electrical and Computer Concentration. He brings leadership experience and a good understanding of how smaller elements of a system fit together as a whole. After graduation, Nate has accepted a position with American Electric Power, working at the D.C. Nuclear Plant in Bridgeman, Michigan. His section of the project is the monitoring of individual circuits and some of the control logic for the breakers.

Avery Sterk: Avery comes from Santa Clara, California where he worked as an intern at the SLAC National Accelerator Laboratory doing CAD design. He will be graduating with a degree in Engineering Electrical and Computer Concentration and a Math minor. He brings varied experience with software design and implementation to the project. Avery is the technical lead on the base station, especially providing the primary user interface and designing embedded software. Additionally Avery became the software lead on the E-Meter after changing the focus of the base station component of the project.

2.4 About the Course

Taken from the Calvin College course catalogue:

> This course takes place over two semesters in the senior design project sequence. For the first semester course (Engineering 339), emphasis is placed on design team formation, project identification, and production of a feasibility study. Students focus on the development of task specifications in light of the norms for design and preliminary validation of the design by means of basic analysis and appropriate prototyping. Lectures focus on integration of the design process with a reformed Christian worldview, team building, and state-of-the-art technical aspects of design. Interdisciplinary projects are encouraged. Prerequisites: Concurrent registration in the seventh semester of the model program for a particular concentration or permission of the instructors; developing a Christian mind and philosophical foundations.
>
> For the second semester in the senior design project sequence (Engineering 340), emphasis is placed on the completion of a major design project initiated in Engineering 339. This project should entail task specifications in light of the norms for design by means of engineering analysis and an appropriate prototype focused on primary functionality. A final presentation is

given at the May Senior Design Night Banquet. Lectures continue to focus on integration of the design process with a Christian reformed world-view, team activity, and state-of-the art technical aspects of design.

3 Scheduling

The team initially created a Gantt chart to establish a timeline for the project and display the critical path for the tasks in the project. As the team decided to first focus on the Smart Breakers, they elaborated the different elements of that subsystem first. Additionally, the team included the known class deadlines for assignments into the Gantt chart to visualize how much time to allot to those assignments. These charts are included in appendix A.

In practice, however, the team's usage of the Gantt chart dimished soon after its introduction. As such, the chart remains essentially the same now as it was in late October, with many tasks and sub-tasks yet to be filled and assigned. While the attention paid to the Gantt chart waned, the emphasis of schedules in team meetings increased. The team would use its meetings to review the upcoming duedates and assign them to individuals, much in the same way that the Gantt chart would be used.

Additionally, the team created their own website on Google Sites to provide a flexible scheduling system independent of Microsoft Project, which is limited to Engineering computers. After each weekly status update meeting, during which individuals acquired tasks, Nathan would update the website's task list to reflect the new assignments and their due dates. While this system does not allow for the critical-path linkages that Gantt charts feature, this customizable task list can include any information desired, such as priority, status, and reviewing information. In some instances, the team mistakenly assigned a small task for the duration of the week, leading to wasted time and letting the project fall behind.

The class deadlines posted on Moodle kept the team focused on some of the long-term goals. Focusing on the larger time scale helped the team allocate more proportionally equal amounts of time for each component of a system. This allowed the team to design by choosing components to fit the system, instead of modifying the system to fit the components.

4 System Overview

4.1 System Block Diagram

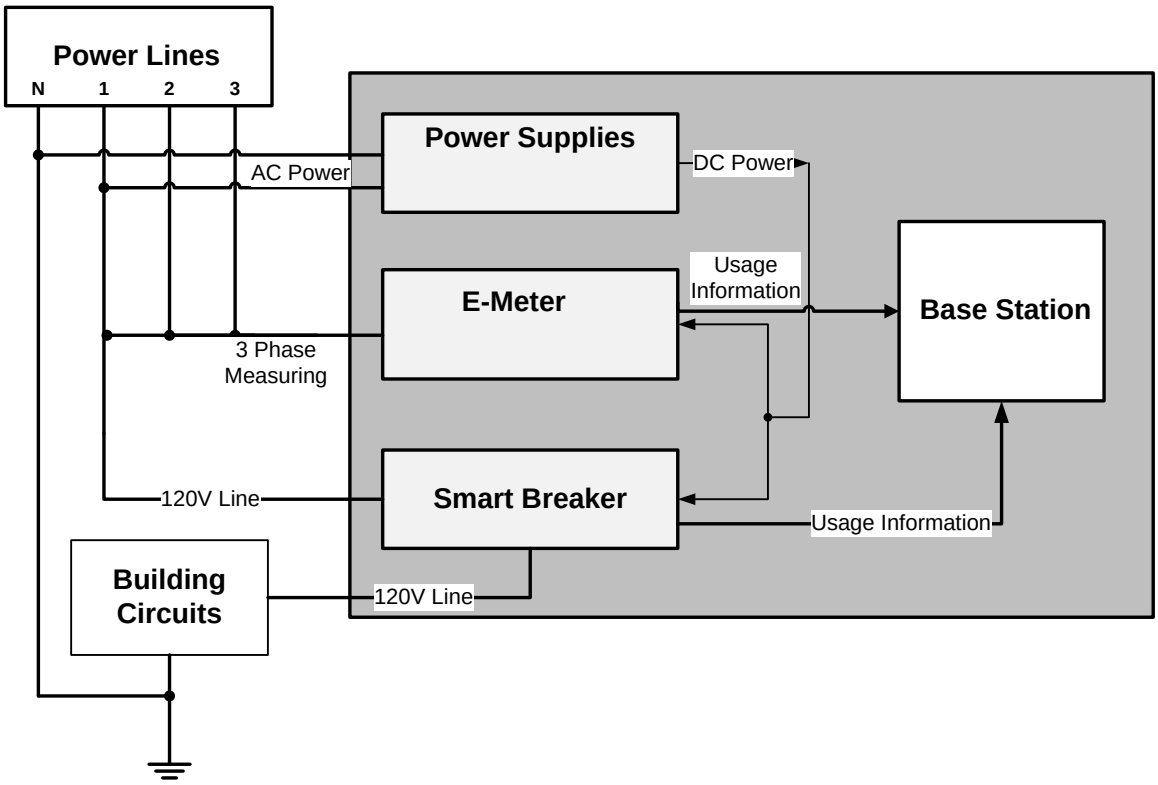


Figure 4: System block diagram

4.2 System Explanation

The full PICA system consists of the e meter, base station, smart breakers and power supply. The system was designed to minimize dependencies between subsystems, allowing the user to choose which information they want without paying for more than that. The fact that the power company and homeowner both have an interest in the PICA system drove much of this decision. As two customers with different requirements, neither party would want to pay to provide the other with information that isn't useful to the first. The team wanted both parties to have information if they wanted it without forcing unwanted costs on

one party or the other. The team later decided that designing for the power company was not feasible. However, keeping the subsystems as independent as possible allows for continued development with the power company's interests in mind should another team work on the project later.

The e meter measures power for the entire building and provides information to the power company. The built in display allows the e meter to operate and provide information completely independently of the base station, although the Xbee connection allows viewing of the information through the base station if desired. The e meter is dependent on the power supply.

The base station is targeted at the homeowner, providing a simple and easy way of viewing information collected by other parts of the system. While the e meter and breakers are not required for the base station to run, it would have no information to provide, and would therefore be useless. Because the base station provides no controls for the other subsystems, it is not required for the other subsystems to operate properly. The base station is not dependent on the power supply. After a lot of work on the base station, the team decided to focus on the e meter and smart breakers and use a standard computer in place of the dedicated base station for prototyping and testing.

The smart breakers provide circuit breaker level information to the homeowner, in addition to providing the same functionality as standard air-gap breakers. The smart breaker's only means of displaying the information is through the base station, so to provide useful data the smart breakers are dependent on the base station. However, the smart breakers still function as breakers without the base station. This is important so that a problem with the base station doesn't shut off power to all of the smart breaker circuits. The smart breakers are also dependent on the power supply.

The power supply provides power to the E-Meter and Smart Breakers and is placed so that it is still able to power everything when all of the breakers are shut off. This is important because the smart breakers' default state is off and won't turn on until they have power, so if the smart breaker came before the power supply, the system would never turn on.

4.3 Testing

Once each subsystem passes its respective verification plan, the three PICA systems can begin integration testing. The system level integration testing verifies that all systems are capable of talking to one another under normal operating conditions.

4.3.1 Simple Talk Test

This test verifies that all systems are capable of talking to one another. Begin by placing both the smart meter and the breaker in a situation where they are guarenteed to talk, such as monitoring an active load. Attach a computer to each device, verifying that in fact they are correctly sending data. Once each system has been verified to be operational, attach Xbee radio cards to the prototype boards and a computer console to the base station. Verify that data transmitted from each device is received by the base station computer. The team expects that data received at the base station will be identical to data seen leaving each device.

4.3.2 Simple Command Test

This test verifies that the base station is capable of sending various commands to the individual subsystems. After all systems have been powered on and placed in an operation state, send the ‘q’ command out across the Xbee network. This should result in both systems halting in an expected fashion. The E-Meter should cease taking measurements and blank the screen while the Smart Breaker changes to the Off state. At this time, both should stop talking.

Continue this test by trying this command an additional time, but specifying which subsystem should respond to the command. Test the E-Meter and Smart Breaker individually, each time sending the ‘q’ command. The response should be the same, but only the device specified should halt.

4.3.3 Complex Command Test

Repeat the previous test, using more complex commands asking for specific information. Commands can be seen in the table 1. As a result of sending any one of these commands, either all or the specified system should return the data requested.

4.3.4 Web Interface Test

Verify that the system is properly reporting data to the web interface and rerun all tests through the web interface command line. Results should match those seen in previous tests.

Command	Behavior
q	Kills the system
b#	Specifies a breaker to command
m	Specifies the command is for the E-Meter
v [#]	Specifies that voltage should be reported. An optional number is allowed for sending to the meter to select which phase.
i [#]	Identical to v except returns current.
p [#]	Identical to v except returns power.
wh	Returns the accumulated watt-hours.
s	Used in conjunction with a breaker to return its status.

Table 1: PICA System commands.

5 Requirements

5.1 Base Station Requirements

All requirements under this heading are to be assumed to be of the base station ("the system") unless explicitly stated otherwise.

5.1.1 Functional Requirements

1. Shall be capable of upgrading its software and firmware upon administrator requests.
2. Shall be capable of connecting with other PICA sub-systems over a pre-defined and pre-arranged communications protocol.
    (a) Shall receive and store power usage measurements from connected PICA systems.
        i. These measurements will be taken at a frequency that does not exceed the established bandwidth of the chosen protocol.
        ii. These measurements will be summarized or discarded (at the user's selection) when the storage space of the system nears its capacity.
    (b) Shall receive and store events and alerts from connected PICA systems.
        i. The nature of these events shall be determined by each individual system, but shall be communicated to the base station in a standardized format.
        ii. The system shall organize these events internally and display them to the user.
    (c) Shall function as a Network Time Protocol (NTP) server for connected PICA systems.
    (d) Shall receive and record event log information from connected PICA systems.
3. Shall be capable of connecting to a Local Area Network (LAN).
4. Shall be capable of using settings that the user selects.
    (a) Shall be capable of recognizing an invalid setting.
    (b) Shall be capable of reverting to a default setting when the user setting is not valid.
5. Shall be capable of displaying the most recent measurements from the connected PICA systems.

6. Shall be capable of displaying the status of the connected PICA systems.
    (a) Status shall include whether the connected system is in an error state.
    (b) Status shall include the time since the last observed error state, if applicable.
    (c) Status shall include the nature of the last error, if communicated.
7. Shall be capable of giving an authenticated base-station adminstrator similarly-privileged administrative access to connected PICA systems.
    (a) The extent of this access shall be deteremined by the individual PICA systems.
    (b) The base station shall inform the administrator when the remote administrative access is rejected or unavailable.
8. Shall be capable of distributing system updates for connected PICA systems.
    (a) Shall be capable of sending update data and images to the appropriate subsystems.
    (b) Shall identify whether or not the connected PICA systems require updating.
9. Shall be capable of giving debugging and troubleshooting output.
    (a) Shall display simple status indication using Light Emitting Diodes (LEDs). (See 5.1.5)
    (b) Shall present more detailed debugging and troubleshooting information over a dedicated RS-232 connection.
10. Shall be capable of actively notifying the power-company and consumer.

5.1.2 Behavioral Requirements

1. Shall store user-defined configuration in non-volatile media.
    (a) Shall initialize this configuration using a pre-defined default configuration.
    (b) Shall load the default configuration if the user-defined configuration is unavailable.
        i. This includes the user-defined configuration being absent.
        ii. This includes the user-defined configuration containing invalid data.
2. Shall include a backup firmware in the event of a failed firmware upgrade.

   (a) Firmware upgrade success or failure shall be determined by comparing checksums of the firmware to be written and the firmware present after writing.

   (b) The backup firmware shall be engaged if the system fails to boot from the first firmware.

3. Shall store critical event logs from connected PICA systems in a non-volatile media.
4. Shall host a webpage to display system information when browsed over the LAN connection.
5. Shall store its software in non-volatile media.
6. Shall run an operating system to manage hardware, device drivers, and connections to connected PICA systems.
7. Shall use a defined communication protocol to communicate with connected PICA systems.

5.1.3 Software Requirements

1. Shall include and run an upgradable Operating System (OS).
   (a) The OS shall include the drivers necessary to operate the system hardware.

   (b) The OS shall include the protocols necessary to connect to PICA sub-systems.

2. The OS shall have an administrator-privileged user who may change the configuration of the system and of connected PICA systems.
3. The OS shall give the system owner access this administrator-privileged user account.
   (a) The OS is not required to give such access to connected systems that are not owned by the owner of the base station system (such as those owned by the power company).

   (b) The OS shall authenticate that the user requesting administrator access is authorized to do so using a means configured during system installation.

4. The OS shall include the protocols necessary to connect to an existing LAN.
   (a) The OS shall include a Dynamic Host Configuration Protocol (DHCP) client.

   (b) The OS shall support manual LAN connection configuration.

5. The OS shall control the system connectivity hardware to prevent unwanted devices, such as those owned by other customers, from being connected to the system.

(a) The OS shall distinguish between wanted and unwanted systems as defined by administrator selection.

i. By default, ZigBee connection requests will be rejected until authorized by the administrator.

ii. By default, LAN connection requests will be trusted unless disallows by the administrator.

(b) The OS shall record connection requests from unwanted devices and display them to the administrator.

6. Shall include and run Hypertext Transfer Protocol (HTTP) server software to provide the web interface.

7. Shall include the necessary tools to download and apply software and firmware updates.

5.1.4 Hardware Requirements

1. Shall include a power supply to power the system from line voltage during regular usage conditions.

2. The power supply shall tolerate moderate expected fluctuations in input voltage from a standard power outlet.

(a) The power supply is not required to tolerate incorrect voltage input (240V instead of 120V, etc.)

(b) The power supply is not required to tolerate voltage spikes as may be associated with electrical storms.

3. Shall include an internal power storage device to allow the system to perform necessary data storage immediately after a power failure.

4. Shall have adequate Random-Access Memory (RAM) to contain system software, cache data from other PICA devices, and cache data to send to other PICA devices without requiring a cache flush more than once per minute.

5. Shall have a processor to execute the system software and process data from connected PICA devices.

(a) The processor must have sufficient processing speed to process the incoming data from other PICA devices at a rate greater than the rate of data incoming.

   (b) The processor must be able to perform all necessary mathematical operations without a co-processor.

6. Shall have an Ethernet controller for connecting to a LAN.
7. Shall have an RS-232 controller for debugging and troubleshooting the system.
8. Shall have ZigBee connectivity hardware for communication with other PICA systems.
9. Shall have non-volatile storage dedicated to storing system firmware.
   (a) Shall have a re-writable non-volatile storage device to contain boot firmware.
   (b) Shall have a separate non-volatile storage device to contain backup firmware to use in the event of boot failure.
10. Shall have non-volatile storage sufficient to store system software.
11. Shall have non-volatile storage sufficient to store recorded events and short-term consumption history for up to a period of 3 years.

5.1.5 User Interface Requirements

1. Shall provide a web interface for viewing collected data over the LAN.
2. Shall provide a web interface for viewing current and predicted pricing information as provided by the power company.
3. Shall provide a web interface for system administration to an authenticated administrator.
   (a) Shall include an interface for managing updates to the system's firmware and software
      i. Shall include an interface for displaying the version numbers or codes of the firmware and software currently installed on the base station.
      ii. Shall include an interface for indicating the availability of system updates for the base station.
      iii. Shall include an interface by which the administrator may request that the base station apply its available updates and be informed on whether or not the base station must be rebooted after the update is installed.

iv. Shall include an interface for indicating the progress of update installation.

(b) Shall include an interface for managing connections to other PICA systems.

i. Shall include an interface for displaying current connections to other PICA systems.

ii. Shall include an interface for adding, removing, or connections made to other PICA systems.

(c) Shall include an interface for administration of connected PICA systems.

i. Shall include interfaces for managing configurations of connected PICA systems.

ii. Shall include an interface for deploying software/firmware upgrades to connected PICA systems.

4. Shall provide a debugging interface over an RS-232 serial connection.

5. Shall display status indication lights.

(a) System is connected to power.

(b) System is connected to other PICA subsystems.

(c) System is connected to the home network.

(d) System has encountered an error.

i. Error light shall trigger if a connection to another PICA system is lost and cannot be recovered within 30 seconds.

ii. Error light shall trigger when the storage medium contains less than 5% free space.

iii. Error light shall trigger when the storage medium's file system cannot be mounted.

A. This includes the storage medium being physically absent.

B. This includes the storage medium being corrupt.

5.1.6 Power Requirements

1. Shall be powered from a standard 120V wall outlet.

2. Shall have a Direct Current (DC) power supply to power internal components.

3. Shall have a backup power supply to enable the system to save all measurements residing in memory to the non-volatile storage medium.
4. Shall require less than 10W to operate.

5.1.7 Codes and Compliances

1. Shall have a polarized electrical plug if the power supply features an an off-on control switch.
2. Shall restrict electromagnetic (EM) radiation to comply with Federal Communications Commission (FCC) Title 47 Part 15.

5.2 Smart Breaker Requirements

All requirements are to be assumed to be of the solid state breakers (“the system”) unless explicitly stated otherwise.

The breaker subsystem must be capable of two major functions; it must protect the connected circuit when a specified threshold is exceeded, and it must pass information to and from the user interface, either directly or through another subsystem.

5.2.1 Functional Requirements

1. Shall be capable of completely disconnecting, either physically or virtually, the power delivered to the connected circuit.
2. Shall provide two-way communications to the base station through the Master Control Unit (MCU).
   (a) Shall provide power usage information, including at a minimum, instantaneous voltage and current of the connected circuit.
   (b) Shall be capable of providing on/off status of the breaker.
   (c) Shall be capable of turning breakers on and off when requested by the base station.
3. Shall be capable of detecting power sags, brownout conditions and blackouts.

(a) The National Power Corporation (NPC) defines sag as ”80% to 85% below normal [voltage] for a short period of time,” [5]. For project uses, this is below 100V. The team defines “a short period of time” to be for three cycles to ten cycles, based on information given in [6]. The NPC defines a brownout as “a steady lower voltage state,” [5] Blackouts are defined by the NPC as “as zero-voltage condition that lasts for more than two cycles,”[5].

4. Shall store up to a minute of gathered information for at least five minutes in on-chip memory for transmission to the MCU.
5. Shall package the stored information for transmission to MCU.
6. Shall be capable of turning off circuits individually.
7. Shall stop current flow to a circuit when it exceeds a specified threshold.
8. Shall be capable of being reset when stopped, without requiring new parts such as fuses.

5.2.2 Behavioral Requirements

1. Shall initialize all components by writing from non-volatile storage to all necessary registers when power is restored to the circuit.
2. Shall report all system events that are not part of standard operation to the critical event log.
3. Shall measure voltage levels in the connected circuit.
4. Shall measure current flow in the connected circuit.

5.2.3 Hardware Requirements

1. Shall use non-volatile storage to store data when the system is without power.
2. Shall be capable of managing its own data and functions.

5.2.4 User Interface Requirements

1. Shall have an external interface that is understandable by the average consumer.
2. Shall provide a way to control the breakers independent from the base station.

3. Shall encase all circuitry except user interface controls (buttons, switches) so that they cannot be tampered with without breaking the casing.

5.2.5 Power Requirements

1. Shall be powered by line-voltage.
2. Shall be powered such that interrupting the circuit does not cause the breaker control circuitry to lose power.

5.2.6 Physical Requirements

1. Shall have dimensions less than or equal to 1in x 3in x 4in for a single breaker or 2in x 3in x 4in for a breaker pair.
   (a) Dimensions are determined by the standard size of a mechanical breaker, so that smart breakers may be interchangeable with conventional residential or commercial mechanical breakers.
2. Shall not weigh more than one pound.
   (a) Weight is determined by the average weight of a standard mechanical breaker [7], so that smart breakers may be interchangeable with conventional residential or commercial mechanical breakers.
3. Shall accommodate wire sizes from 14-4 Cu to 12-8 Al.
   (a) Wire size is based on the average size wire used with standard mechanical breakers [7], so that smart breakers may be interchangeable with conventional residential or commercial mechanical breakers.

5.2.7 Safety Requirements

1. Shall protect everything connected to the circuit from current exceeding a specified threshold by providing circuit interruption.
2. Shall have safety hazards clearly marked and visible from outside the system.
3. Shall safely isolate high-voltage areas so that they provide no more threat than a standard wall outlet.

5.2.8 Codes and Compliances

1. Shall be compliant with American National Standards Institute (ANSI) C12.19 [8].
2. Shall be compliant with ANSI C12.21 [9].
3. Shall be compliant with FCC Title 47 Part 15 [10].

5.3 E Meter Requirements

All requirements are to be assumed to be of the electric panel meter (“the system”) unless explicitly stated otherwise.

5.3.1 Functional Requirements

1. Shall continuously monitor the power used from either a single-phase or a multi-phase installation.
2. Shall store power usage data locally, to be transmitted back to the base station at regular intervals.
3. Shall by default display instantaneous and historical power usage data on an Liquid Crystal Display (LCD) module integrated into the electric panel.
4. Shall provide two-way communication with the MCU to report usage data.
5. Shall be capable of detecting a brownout condition and storing critical data before shutting down.
6. Shall be capable of restarting and restoring stored data after a brownout condition.
7. Shall be capable of detecting any tampering, such as opening the sealed metering unit, and transmitting a tamper message to both the power-company and the consumer.
8. Shall monitor current flow through the main service lines for automated meter reading.
9. Shall monitor voltage levels on the main service lines for automated meter reading.
10. Shall control the service shutoff switch by receiving and validating a service shutoff message from the power-company.
11. Shall provide a method for controlling the service shutoff switch from a local interface.

12. Shall provide an interface for 3rd party meters, such as gas, water, or other utility meters to report data over the PICA network.
13. Shall support on-demand reports from the power company via the Zigbee network or the user via the PICA web interface of power usage, energy consumption, demand, power quality and system status.
14. Shall support bi-directional metering and calculation of net power usage to support alternative energy generation systems.
15. Shall support automatic meter reads.
16. Shall analyze the voltage flicker, logging a warning when the flicker exceeds 20% of the stated $117V_{RMS}$.
17. Shall meter reactive power consumption, logging data for billing purposes.

5.3.2 Behavioral Requirements

1. Shall, in the event of wireless link loss; attempt to re-establish the wireless link.
2. Shall, in the event of a wireless link loss, revert to stand-alone mode, storing data internally until internal storage is full, at which point the system will begin overwriting the oldest data with the newest data.
3. Shall, in the event of a wireless link loss, notify the user via the LCD display.
4. Shall perform a built-in self-test upon system boot up to verify onboard storage integrity and to verify proper operating software.
5. Shall, in the event of a brownout, save all volatile information to non-volatile storage space.
6. Shall be capable of detecting corrupted data, via parity bits, when brought out of a brownout condition.
7. Shall log all events processed into the following four categories:
   (a) critical: messages requiring immediate attention.
   (b) error: messages requiring attention and may affect system functionality.
   (c) warning: messages that require attention but do no impact system functionality.

(d) note: messages that require no attention but provide verification of proper operation.

8. Shall report all events to the PICA base station.
9. Shall report to the power-company as specified by event criticality.
10. Shall have dedicated non-volatile storage for all critical settings and configuration data.
11. Shall compute the total power used in kilowatt-hours.
12. Shall be capable of receiving messages from the power-company, providing the user with the current cost of a kilowatt-hour.
13. Shall use 128-bit Advanced Encryption Standard (AES) encryption for all messages transmitted outside of the device.
14. Shall report the total amount of outage time to both the power company and the PICA base station.
15. Shall date-stamp all detected outages with the date, time, and duration of the outage.

5.3.3 Software Requirements

1. Shall verify system firmware on boot up.
2. Shall periodically, minimum once per day, perform a system check to verify the health and status of the system.
3. Shall perform an on-demand system health and status check as demanded by the PICA base station or the power-company.
4. Shall contain sufficient non-volatile storage for all system configuration settings.
5. Shall be updateable through the power-company wireless interface.
6. Shall be capable of properly recovering from a failed software update.
7. Shall give authorized access to components of the system configuration as appropriate to the power-company and consumer.
8. Shall notify the power-company and the PICA base station once service has been restored containing the time of restoration and a voltage measurement.
9. Shall have a unique IPv6 address for the power-company mesh network.

10. Shall have a unique IPv4 or IPv6 network address for the local home-area-network.
11. Shall receive an NTP message from the PICA base station to set the hardware clock.
12. Shall synchronize the hardware clock with the base station time once per day.
13. Shall support on-demand hardware clock synchronization via the PICA base station web interface.

5.3.4 Hardware Requirements

1. Shall be completely enclosed in a weatherproof case, tolerant of extreme temperature differences.
2. Shall be completely Alternating Current (AC) coupled against transient AC voltages.
3. Shall be mounted in the same location as a standard power meter.
4. Shall provide non-volatile storage.
5. Shall be grounded.
6. Shall provide a hardware system clock, set by the software and synchronized with the PICA base station.

5.3.5 User Interface Requirements

1. Shall have a 160-segment LCD display module, viewable from outside the electric panel.
2. Shall be capable of interfacing with a web-based application for stand-alone configuration.
3. Shall provide push-buttons for changing the viewing contents on the display module between metering and system status views.

5.3.6 Power Requirements

1. Shall be capable of operating from line-voltage.
2. Shall be powered from before the master breaker, preventing the meter from losing power when the master breaker is switched off.

5.3.7 Safety Requirements

1. Shall meet or exceed safety requirements for devices inside an electric panel.
2. Shall provide a grounding point to ground the system when installed.
3. Shall be protected against the elements. 4. Shall safely isolate high-voltage areas.

5.3.8 Codes and Compliances

1. Shall be compliant with ANSI C12.19.
2. Shall be compliant with ANSI C12.21.
3. Shall be compliant with FCC Title 47 Part 15.

5.4 Power Supply Requirements

All requirements under this heading are to be assumed to be of the power supply ("the system") unless explicitly stated otherwise.

1. Shall be 78% efficient.
   (a) This efficiency was chosen, because it is what SMPS were generally.
2. Shall power 2 ADE devices.
3. Shall provide 5VDC +/- 5%.
4. Shall provide 2mA, 3mA (total 5mA) plus a Safety factor.
   (a) A safety factor can be efficient as 1.0001x.
5. Shall also power an FPGA that will control these chips in a production-scale design at 5VDC $\pm$5%.
   (a) While the exact requirement for the FPGA are unknown the current was chosen much higher than probably needed in order to make sure the power supply would not need to be changed to provide more power.
6. Shall have a ripple voltage of less than 1%.
7. Shall be isolated from mains supply by a high frequency transformer $60Hz$.

8. Shall be small and not exceed $4in^3$.

9. Shall not be hot enough to where heat sinks are required.

6 Design Goals

6.1 Provide a physical system that accurately monitors power usage

Inaccurate information provides no benefit to either the power company or the consumer, despite any added features. Therefore, the system should be as accurate as or more accurate than monitors currently used.

6.2 Provide manuals for maintenance and general use

The design team recognizes that no system is perfect and will eventually need maintenance and that most consumers do not have extensive knowledge of electrical systems or components. Providing manuals will assist the consumer in understanding their system and getting the most benefit from it. The design team will look to create a manual that will include an overview of how the system works, how to install and configure the system, and how to maintain the system during the course of normal use. This manual will also contain a troubleshooting guide, support information, liabilities, safety information, and any other pertinent information.

6.3 Design the system to be modular

Providing information to both the power company and to the consumer is the main goal of the project, but it is possible that an installation will not include all the subsystems. For example, a consumer may want the consumer-oriented part of the system, while the power company does not want the power-company-oriented subsystem, or vice versa. The modularity goal aims to satisfy all situations without forcing extra costs on any party. To do this, the design team will design the system so that the subsystems providing information to the power company and the subsystems providing information to the consumer do not depend on or require each other.

6.4 Present power usage information in a way that is understandable to an average consumer

The goal of the design team is to present the information in an understandable format for the consumer. Because the average consumer does not have an engineering background, the design team would like to

display the power usage information in dollars per minute, per hour, per day, per week, per month and per yearly. The system will report these costs in addition to more technical information including voltage, current, power factor and more.

6.5 Minimize on-site maintenance as much as possible

In many cases, the consumer calls the power company to fix something as simple as a tripped breaker, and the power companies waste a lot of time just driving to and from the site. The ability to do work remotely allows the power company to minimize this cost. Remotely controlling or monitoring different aspects of the meter also lets the power company quickly assess if a problem still needs attention. Aggravated customers may also become violent towards power-company technicians, so by having remote access, the power company's employees are safer from these threats.

7 E-Meter

The E-Meter, short for electricity meter, fulfills the role of monitoring all power consumed for a given customer. The PICA E-Meter exists as a replacement for the standard electricity meter, seen in figure 1, found outside most homes and businesses. As the PICA E-Meter will replace an already existing piece of technology, the PICA E-Meter must fulfill all the same roles as a traditional power meter. In addition, team PICA seeks to provide extended capabilities, such as wireless meter reading and tamper detection.

Since the average consumer does not actually own the physical metering hardware, this portion of our senior design project targets the electric companies. We hope that by providing more accurate readings, more reliable meters, and more convenience the PICA E-Meter could entice some, if not all of the electric utility providers who still rely on traditional electric meters. The PICA E-Meter should cost less than $200 each fully assembled. Team PICA currently cannot verify how this price compares to that of a traditional power meter, as they are not available on the open market.

The PICA E-Meter design functions for either 3-phase of 1-phase power, usually found in commercial or residential applications respectively. Both devices share the majority of the hardware, with only a different input board the main hardware functions identically for both applications.

7.1 Original Ideas

Early in the project team PICA decided to use a Texas Instruments (TI) MSP430 low-power microprocessor. The family of MSP430 devices includes several devices that could perform the necessary tasks, however, the MSP430F471xx family of devices are tailored for metrology. After some investigation the team decided to use an MSP430F47197 microprocessor as the core of the E-Meter. A summary of the specifications for this micrprocssor follows in table 2.

Table 2: Specifications for the MSP430F47197[2]

Specification	MSP430F47197
Frequency	16 MHz
Flash	120 kB
RAM	4 kB
GPIO	68

Continued on next page

Table 2: Continued from previous page

Specification	MSP430F47197
Pin/Package	100LQFP
LCD Segments	160
ADC	7 16-bit Sigma Delta
Other Integrated Peripherals	Comparator, DMA, Hardware Multiply, SVS
End Equipment Optimization	Energy Meter
Interfaces	2 USCI_A, 2 USCI_B
Timers	1 Watchdog, 2 16-bit, 2 8-bit

With its included flash, RAM, and onboard peripherals the MSP430 provides a good all-in-one solution on which to base the E-Meter. Along with these features the MSP430 boasted the lowest power consumption of any solution the team examined. A full discussion of one of the alternatives can be found in Team PICA's Project Proposal and Feasibility Study (PPFS) (pages 34-35)[11]. Additionally, the team considered using an Arduino for the E-Meter MCU, which would then need to interface with an external Analog-to-Digital Converter (ADC). However, having an integrated solution, where data can simply be read from a register rather than programming and debugging a communications protocol eventually won, taking the Arduino out of the possibilities. The final decision to stick with the MSP430 came when TI approved to sample Team PICA a development board, programming pod, and extra processor free of charge. This donation allowed the team to allocate $150 to another part of the project.

In order to be compliant with ANSI codes C12.19 [8] and C12.21 [9] the PICA E-Meter must display the instantaneous power usage on the local metering unit. For this reason, the E-Meter should have an LCD screen displaying the necessary data. After some research, the team found 2 possible display units the SCLCD from SynchroSystems Embedded Computer Design [12] and the Softbaugh SBLCDA4 from SoftBaugh Inc. [13]. Each of these displays would use the MSP430's internal 160-segment LCD driver. As both display units are comparably priced the team chose to use the display from SynchroSystems because they provided a driver written in embedded C along with a backlight kit to make the display more readable. The implementation of this display will be discussed in more detail in the next section.

As a second alternative, the team investigated using a graphic display LCD from Newhaven Display International. This particular display, the NHD-C160100DiZ-FSW-FBW, interfaces with the MCU over

I2C and cost approximately $20 after shipping. The display featured a 160 x 100 pixel display that could operate on a 3V supply. Using a display that operates on I2C would only require two I/O pins, rather than the 44 pins required by the SCLCD discussed previously, freeing up 42 pins for more åMCU I/O devices (such as buttons, data-links, etc.). Because the Newhaven display is a graphic display, the team could draw much more complex graphics, making a much more aesthetically pleasing interface. However, the team would be required to write the entire display driver from scratch, which could be a very lengthy process. Ultimately the team chose to use the SCLCD from SynchroSystems for its shorter development time, and because I/O usage was not a concern.

7.2 Hardware Design

In order to break down the E-Meter into several subsystems Team PICA began with the block diagram seen in figure 5. This block diagram describes 5 major subsections of the E-Meter, the microcontroller, the LCD

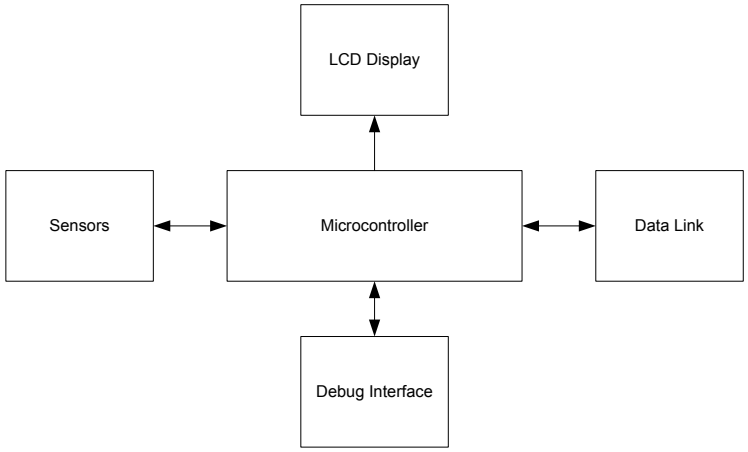


Figure 5: A simple hardware block diagram of the PICA E-Meter.

screen, the sensors, the data-link, and the debug interface. In order to construct a fully working E-Meter each of these components required some individual proving and then integration into the final design.

7.2.1 Microprocessor Integration

Before beginning work on any of the other components, Team PICA needed to prove that code could be compiled and loaded onto the MSP430 platform. In order to accomplish these two tasks several options

exist:

1. IAR Workbench
2. Code Composer Studio
3. MSPGCC 4.x

The first two options, both provided by TI, are proprietary development environments tailored for compiling and debugging embedded C. However, both of these tools are provided for Microsoft Windows only. That being the case, the team wanted to try and work with the open-source MSPGCC4 project as the compiler and debugger could run on any computing platform. At the time this seemed like a good idea as the two team-members working on the E-Meter did not use Microsoft Windows as their primary computing platform.

After some configuration, the team successfully compiled and installed the MSPGCC compiler. A full guide to repeating this process can be found in appendix B.3. The first test, after setting up the toolchain, consisted of determining if the computer could detect and properly configure the MSP430 programming pod. Figure 6 shows the dmesg output showing successful pod detection and configuration. At this point,

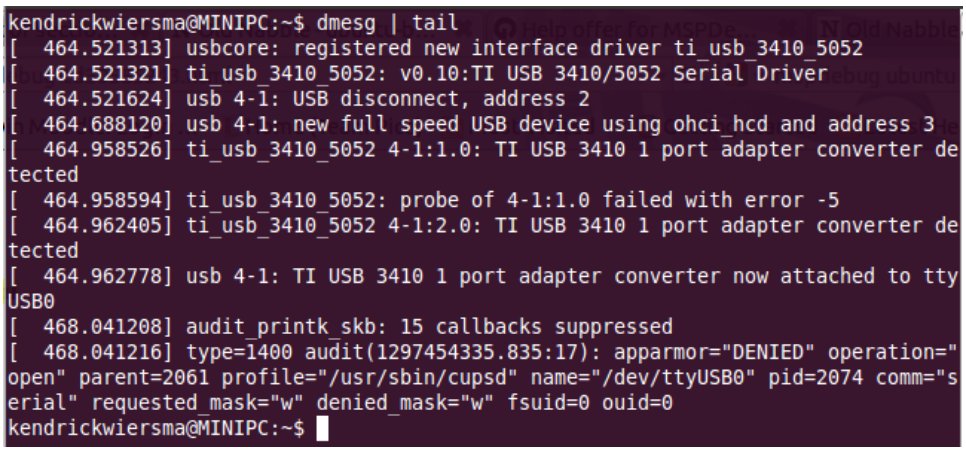


Figure 6: Dmesg output showing Ubuntu Linux correctly detecting and configuring the MSP430 programming pod.

the programming pod is recognized as a valid USB device and configured, thus the pod can configure the MSP430 device. However, at this point the team learned that while the MSPGCC4 toolchain supported their device, the debug utility, mspdebug did not support loading code through the team's programming pod. Thus, while MSPGCC4 could compile code, the team would need another tool to load compiled code onto the microprocessor.

Upon learning that MSPGCC4 would not work, Team PICA turned to the suggested tool for developing MSP430 software, IAR Workbench. However, after several trial runs, the team readily switched to Code Composer Studio 4 (CCS4) for its advanced debugging abilities and familiar interface (Code Composer Studio is based on the Eclipse platform). The version of IAR workbench provided with the MSP430 development kit allowed the user to step through either C code or the translated Assembly instructions, but did not allow the user to view that status of the internal registers.

Using CCS4 team PICA wrote a "Hello World" program for the MSP430, capable of blinking one of the onboard LEDs. This provided conclusive evidence that the team could successfully write, compile, load, execute, and debug code for the MSP430. See appendix B.2 for a listing of our "Hello World" program.

7.2.2 LCD Integration

Team PICA decided to begin integration of external peripherals with the LCD as SynchroSystems had already provided driver software compatible with the MSP430 family of devices. After some minor tweaking to tailor their provided driver and demo to run on the MSP430F47197, the design team could successfully compile code to work with the LCD screen. Primarily this tailoring consisted of porting their code to use the LCD_A controller (an integrated peripheral on the MSP430F47197) as opposed to the LCD Controller found on some of the other MSP430 devices. However, simply porting the driver did not address the physical hardware connection of adding a screen to the MSP-TS430PZ100A development board donated by TI.

In order to make the physical connection between the SynchroSystems SCLCD and the MSP-TS430PZ100A (hereafter refered to as MSP430 development board) Team PICA designed a breakout board using the freely available EAGLE schematic and Printed Circuit Board (PCB) layout software. See figure 7 for a picture of the CAD drawing of the breaout board. After correspondance with Chuck Cox, owner of SynchroSystems, SynchroSystems provided an EAGLE library containing the CAD drawings of their part, allowing the breakout board to be produced quicker. At the time of this writting this part library is not available for public release.

During the LCD integration testing, Team PICA discovered that the MSP430 peripherals did not operate in the same clock domain as the microprocessor core. Thus, while the core worked properly via the JTAG pod, the peripherals did not receive any clock signal. After reading through the user's manual [14], the

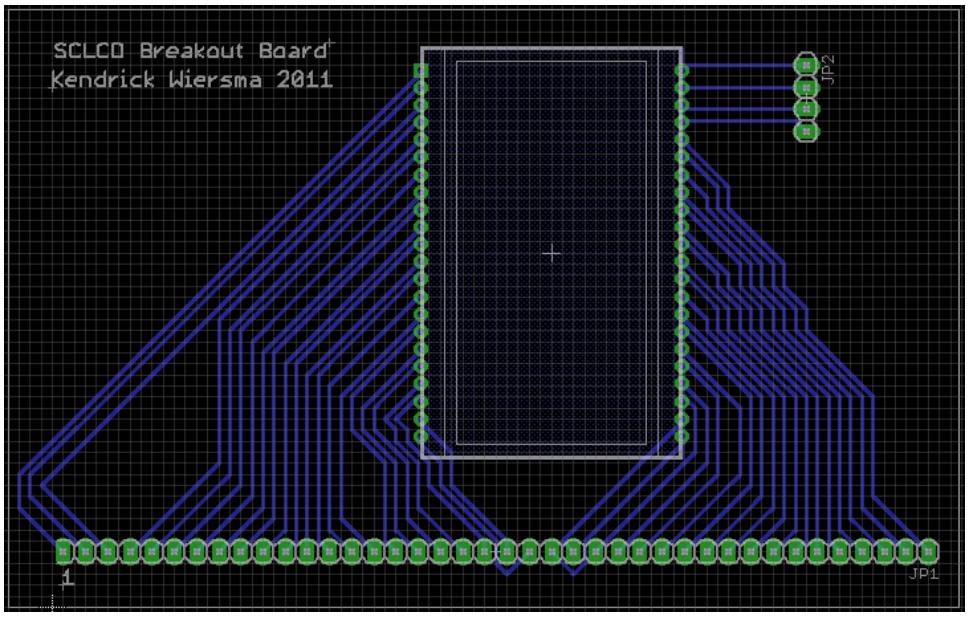


Figure 7: EAGLE CAD drawing of the SCLCD Breakout Board.

team learned that the MSP430F47197 microprocessor required an additional clock circuit to drive the peripherals, which operate at a different frequency than the main processor. The clock circuit can be seen in figure 8. This clock operates at 32.768kHz, and from it the MSP430 derives the signal known as ACLK.

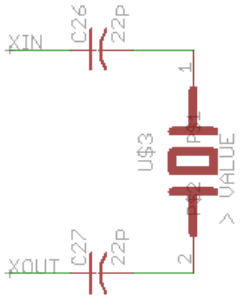


Figure 8: 32.768MHz clock circuit for MSP430 development board.

With the addition of this clocking circuit, and a 22uF capacitor on the LCDREF pin, the LCD screen immediately began displaying characters and the modified demo from SynchroSystems ran properly. A video of the LCD screen working can be seen at `http://youtu.be/9oDglYHUhxo?hd=1`.

7.2.3 Current Sense Integration

After Team PICA had the ability to display data on a local LCD, the next logical step becomes to collect data from our sensors. As previously noted, the MSP430F47197 contains seven 16-bit Sigma-Delta (abbreviated SD16) ADCs used to convert a voltage into a digital measurement. For either our 3-phase or our 1-phase configuration, each SD16 converter used requires an input network to ensure that the differential voltage applied to the terminals of the ADC does not exceed 500mV. As the team decided to demonstrate 3-phase metering in its prototype, that configuration will be used as an example.

In order to verify that the current sense input network would function as specified, the team ran simulations in LTSpice IV. Figure 9 shows the LTSpice schematic for the current sense network, while figure 10 shows the final test output. This particular input board design covers 10A circuits, as such, in the worst possible

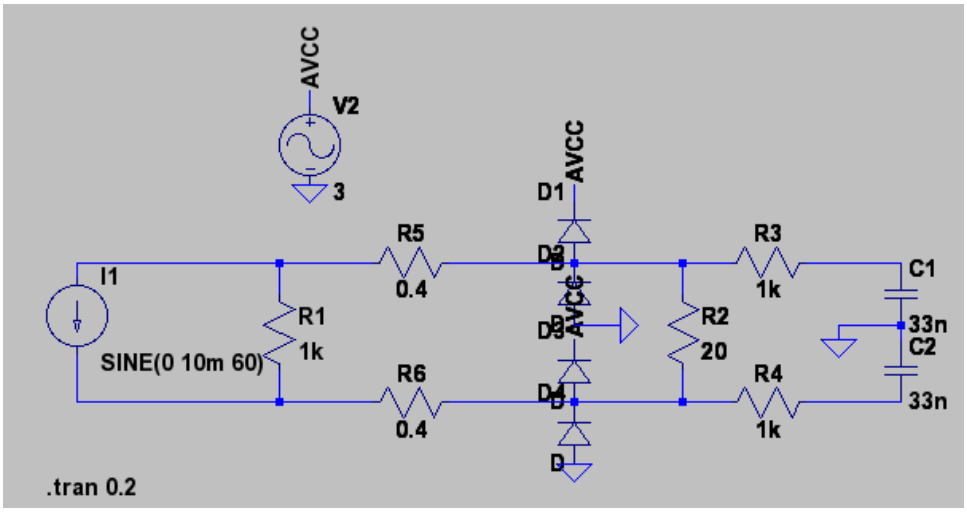


Figure 9: Current sense input network verification using LTSpice.

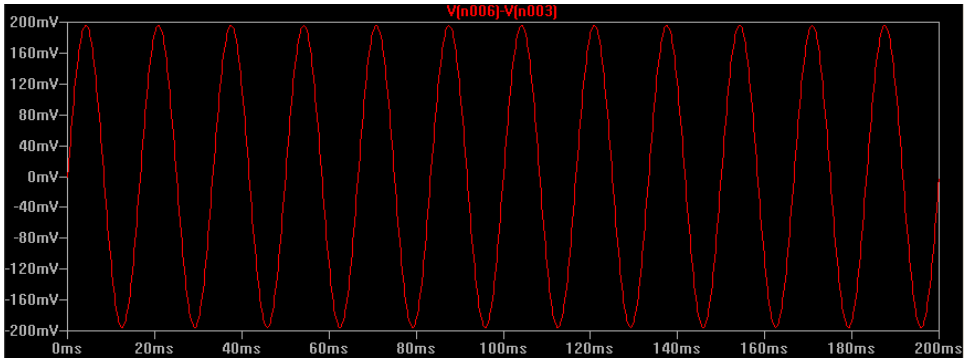


Figure 10: Results from the LTSpice verification of the current sense input network.

scenario the current transformer can produce 10mA when 10A conducts on the main lines. A current

source in figure 9 represents the current transformer output. The results then show that when measuing maximum line current flow, the differential output of the current sense network remains at approximately 400mV, as seen in figure 10. This test ignores the varistor, as the LTSpice model behaves poorly and it exists for extreme cases. Diodes D1 through D2, in figure 9, provide clamping, should some part of the input network fail, and exceed the maximum rating of the circuit.

Of the seven available SD16 converters, two are dedicated to each phase of the power. The first SD16 for each phase measures the current, using an input network seen in figure 11. The positive and negative

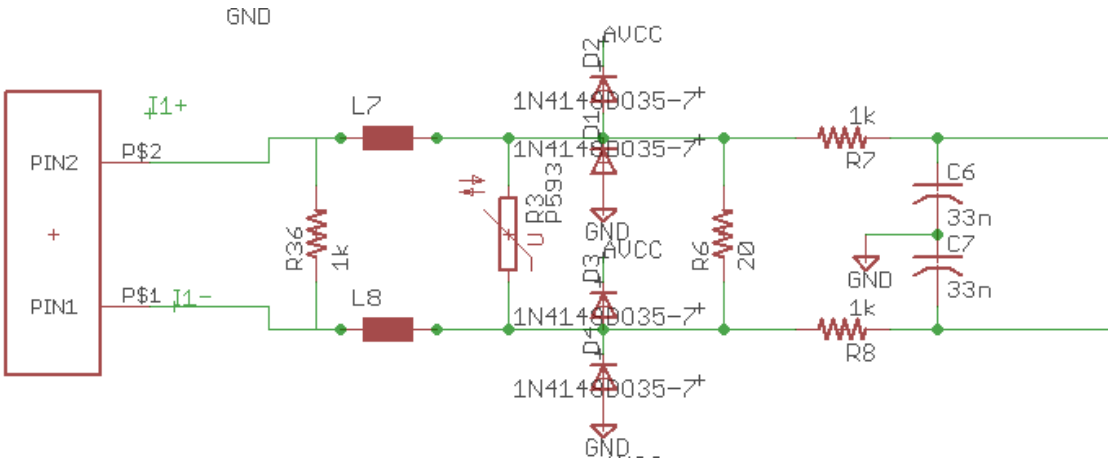


Figure 11: SD16 current sense input network.

terminals of a current transformer attach to points PIN1 and PIN (as seen in figure 11) in the SD16 input network. A current transformer provides a convenient method for measuring large currents by keeping the wires carrying large currents physically detached from the circuit. Current transformers rely on a process known as induction, where a wire carrying current is passed through a loop, or many loops, of wire. Those loops of wire then begin to conduct current in response to the EM field produced by the current carrying wire. The input network can sense the wire-loop's induced current as a differential voltage, which the SD16 can then convert into a digital measurement. The inductors, L1 and L2 provide protection against any electro-magnetic interference while the varistor R3 and diodes D1-D4 provide protection against large transient spikes should the current transformer induce too large a current. Resistor R36 is known as the burden resistor for the current transformer, which provides a low impedance path for induced current to flow, and further limiting the amount of induced current in the current transformer, keeping the circuit operating in a stable and predictable manner. Additionally, the burden resistor translates the induced current to a voltage for the reaminder of the circuit. The current transformer datasheet [15] contains a table specifying the burden resistor to produce a desired voltage. The datasheet recommended a 500Ω burden

resistor to produce 0.4V and 2kΩ burdon resistor to produce 0.6V. As the SD16 input differential constaint is 0.5V, we chose a value between those specified on the datasheet, 1kΩ for the burden resistor. The remaining resistors, R7 and R8, along with capacitors C6 and C7, server to further limit and scale the generated differential voltage from the burdon resistor before being read by the SD16 converter. This portion of the circuit comes from a TI application note regarding the SD16 input network [16].

The design team chose to use a relatively small turns ratio current transformer to allow for more headroom on the input network. The CST-1020 used in the prototype has a turns ratio of 1:1000, meaning there are 1000 turns for every one turn of wire passing through the center. For the prototype, where our meter would not be reading more than 20A, the limit of these current transformers, this would provide us with a maximum of 20mA of induced current. For a production design, the current transformers would be required to read currents of up to 200A or more, requiring that the current transformer be resized to 1:10000 or greater. However, as large turn ratio current transformers tend to be costly, the team chose to use the more cost effective, lower turns ratio current transformers for the proof-of-concept E-Meter prototype.

Once the design team added the serial data port, discussed in the next section, the upper four bits of the data could be transmitted back to the workstation Personal Computer (PC) for processing in MATLAB. While the SD16 converter measures 16-bit voltages, the serial port can only transmit 8-bit characters, only some of which are printing characters. By truncating the measurements to four bits, the values could be added to a reference character (uppercase 'A') to map the values 0 through 15 into the characters 'A' through 'P'. As the intent of this test was to verify that readings from the SD16 were periodic in nature, four bits of low-resolution could still satisfy the goal. By showing that the SD16 converter could properly capture and read a periodic waveform the team proved that the SD16 converter functions as expected. See figure 12 for the test results. The results really need very little explanation, figure 12 shows that the output of the SD16 converter, once normalized to zero, can properly read a sinusoid. Given that only the top 4 bits of the result were captured for this test, the resolution of these readings dropped significatly: the extreme values of the sinusoids are not represented, as this four-bit system does not provide enough resolution to distinguish them from the neighboring values. Restoring more of the 16-bit resolution by keeping more bits would help relieve this issue. See appendix B.4 for the code used in this test.

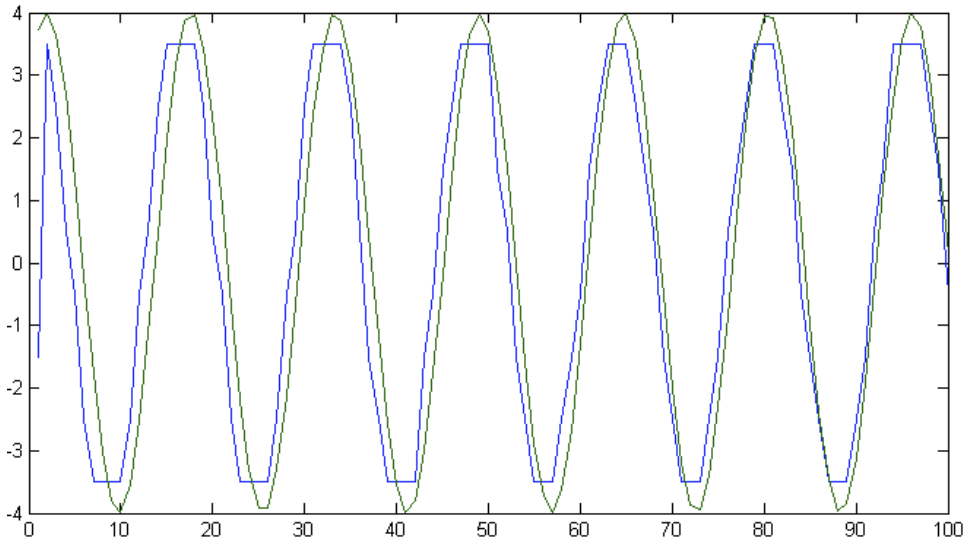


Figure 12: Data captured from the SD16 converter over RS232 (blue). Ideal sinusoid (green).

7.2.4 Voltage Shunt Integration

In addition to measuring current, the PICA E-Meter must also collect data about the line voltage. While three of the 7 Signal-Delta ADCs are used for current measurements, the next three are used for voltage measurements when in the 3-phase configuration. Again, the SD16 converters only allow a 500mV differential input, so the team designed an input network to step down the 120VAC to a safe level to input into the converters. This circuit comes almost entirely from a TI application note regarding using the MSP430 for 3-phase power monitoring [16] and can be seen in figure 13.

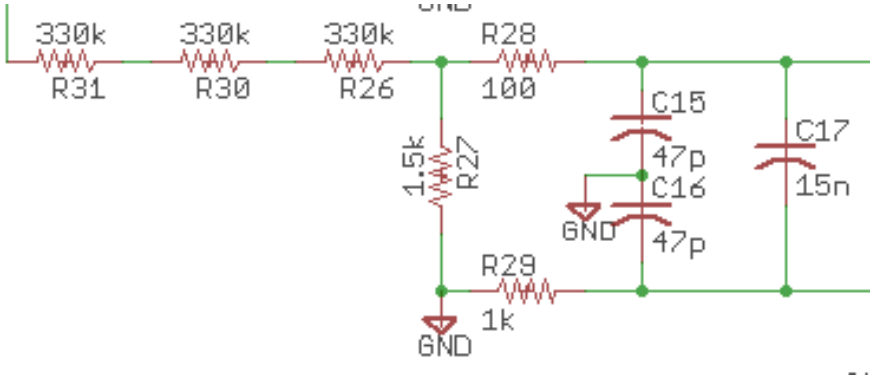


Figure 13: Voltage shunt input network.

In order to verify that the circuit worked as expected, LTSpice IV was used to simulate a 120VAC signal entering the network. The schematic for this simulation can be seen in figure 14 and the results in figure 15.

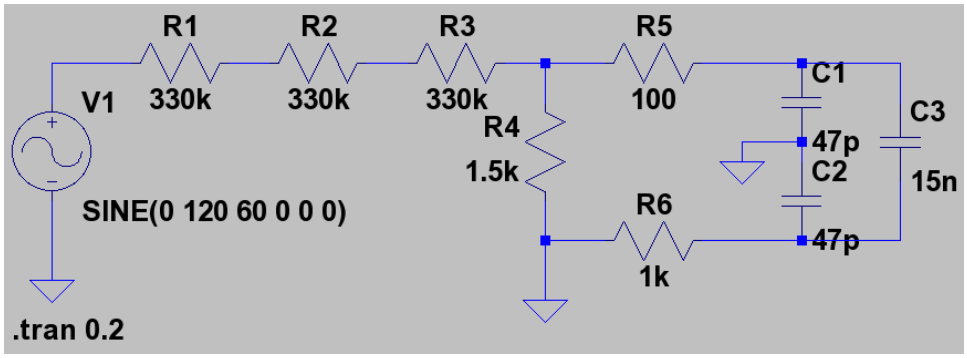


Figure 14: Voltage shunt input verification using LTSpice.

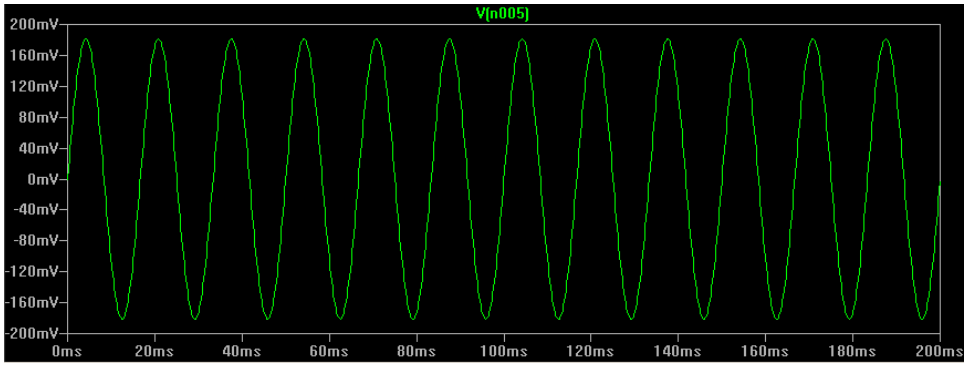


Figure 15: Results from LTSpice verification of voltage shunt input network.

Figure 15 shows a peak-to-peak measurement of approximately 330mV, which is within the required range for the SD16 converter. Measurements taken of the realized circuit indicate approximately a 355mV differential voltage which is within the component tolerance of 5%.

$$330\text{mV} \times 0.05 = 16.5\text{mV} \tag{1}$$

$$16.5\text{mV} \times 2 = 33\text{mV peak} - \text{to} - \text{peak} \tag{2}$$

$$330\text{mV} + 33\text{mV} = 363\text{mV} < 500\text{mV} \tag{3}$$

Like the previous sensor integration test, upon first attaching the voltage shunt network, the top four bits of the converted result were transmitted back over the Recommended Standard 232 (RS232) uart. This data could then be processed by MATLAB and plotted. The results are seen in figure 16. The data captured looks very corse and lacks the precision necessary for an E-Meter to properly calculate the power used. The team expects that by finding a method to transmit all 16 bits back, as opposed to only the truncated top four bits, the precision of the results will increase.

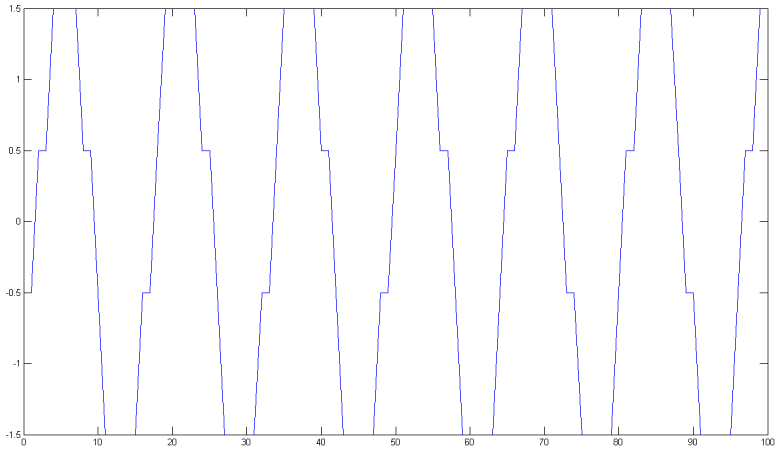


Figure 16: Voltage data captured from the SD16 converter over RS232.

7.2.5 Serial Data Link Integration

The second to last block of the block diagram in figure 5, data link, describes the method for communicating the measured data to the rest of the world. Ultimately, Team PICA desires to implement an Xbee radio communication system, however, before using the Xbee radio a simple RS232 Universal Asynchronous Receiver/Transmitter (UART) fulfills the data transmission requirement. By first implementing a simply prototype, RS232, the team could begin development with some basic functionality before moving onto integrating the more sophisticated wireless Xbee. Additionally, if serial communications functions properly, making the change to Xbee is trivial, as the Xbee radio uses a serial RS232 based interface.

In order to implement RS232 communication the design team chose to use the MAX233A Integrated Circuit (IC) which comes in a 20-pin Dual Inline Package (DIP). The MAX233A provides line-level conversion from the 3.0V output of the MSP430 up to the 5.5V required for RS232 communication. Conversely, the chip also converts the receive line from RS232 5.5V down to 3.0V for the MSP430 UART. Other methods exist for performing this line-level conversion, however the MAX233A provides a simple, 1-chip solution without requiring large amounts of board space and without adding significant cost to the final product. See figure 17 for the serial data-link circuit used in the PICA E-Meter. Any number of the MAX23X devices could have been used for this purpose, however the MAX233A required the least number of external components. Without an integrated solution, the team could have designed a circuit using optocouplers and transistors to mimick the functionality (stepping up and down the RS232 voltages),

however this method would require 6 discrete components, totaling a space larger than the MAX233A IC.

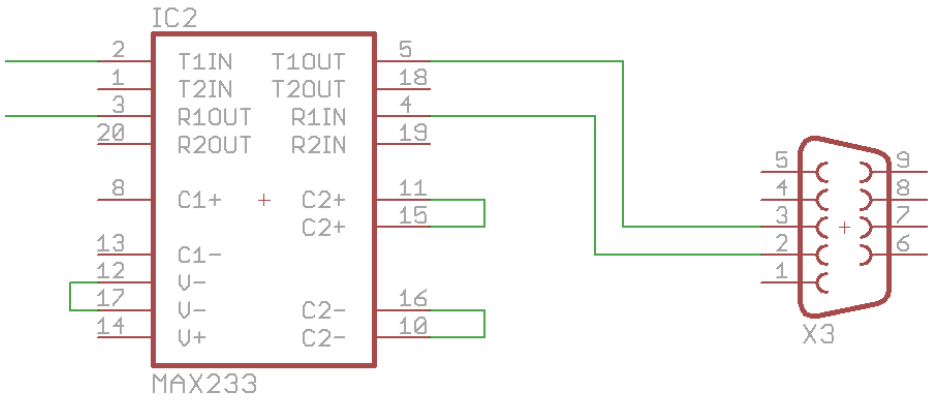
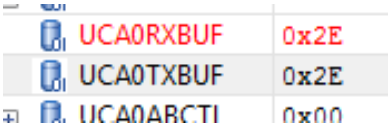
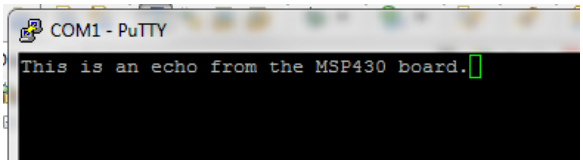


Figure 17: MAX233A RS232 serial data link circuit.

In order to verify the proper operation of the serial data link, the team performed an echo test. Code running on the MSP430 receives a character from the receive buffer and simply moves it to the transmit buffer on a receive interrupt. Characters typed into a serial terminal on the team's workstation PC would then be transmitted from the computer, to the MSP430 before being transmitted back to the PC and displayed in the terminal window. This operation succeeded on the very first attempt with the results seen in figure 18a and 18b. Even though serial communication provides basic data transmission, the team



(a) Serial UART registers in the MSP430



(b) Serial UART terminal on the workstation PC

Figure 18: RS232 Echo Test

desired to implement a wireless interface to the E-Meter, allowing for greater distance between the unit and the data recipient. The original target, Xbee radio, uses a serial interface, essentially making it a wireless RS232 cable. Each Xbee device holds non-volatile configuration data describing the devices function, either an end-device or a router. The Xbee devices, however, require an additional signal, Data Transmit Ready (DTR), to alert the device when a data packet is ready for transmission. The MSP430 software will

need to generate this signal when it wants to transmit data.

During testing with the Xbee radio the team added support for flow control to the RS232 port to better control the transmission timing of the Xbee devices. This simply involves adding two additional signals between the D-Sub 9 connector and the MAX233A chip, along with two additional lines to the MSP430. As of right now, software support for flow control does not exist. See figure 19 for a full schematic for the RS232 connection.

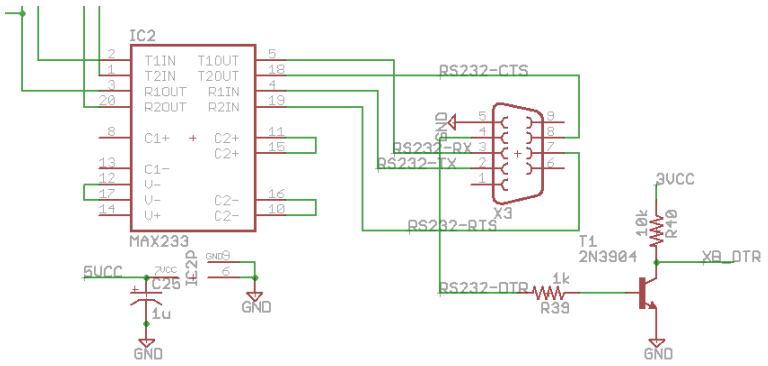


Figure 19: Flow control and DTR lines for RS232 communications

7.2.6 Debug Interface Integration

In order to properly load software onto the MSP430, the team needed to implement a JTAG port in the final design of the prototype. The JTAG standard defines the implementation of the communications protocol as well as circuitry needed to interface a JTAG programmer and the JTAG port on a device. For the E-Meter, see figure 20 for the JTAG interface circuit. TI has chosen to use a 14-pin interface for its JTAG devices. Additionally figure 20 shows the pin header (SV1) that allows the source of the 3V supply to change between an external power supply and the internal JTAG pod power. In a final production version the JTAG interface must remain on the PCB for the initial programming of the microprocessor and allowing for software reset. However, the 14-pin header should not be populated on the PCB making it more difficult for the end-user to reprogram the meter, or read out proprietary firmware.

In order for JTAG to properly configure a device, four signals must be present, with an optional fifth signal:

1. TDI: Test Data In
2. TDO: Test Data Out

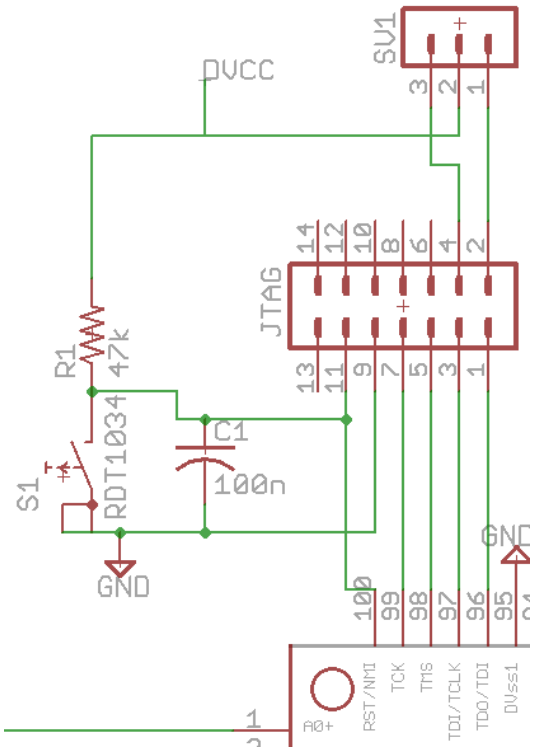


Figure 20: JTAG circuit for the MSP430F47197.

3. TCK: Test Clock
4. TMS: Test Mode Select
5. TRST: Test Reset (optional)

The MSPP430 processor has dedicated ports for each of these signals which arrive through the TI programming pod. In order to add this functionality to the E-Meter PCB, a 14-pin header provides the interface between the programming pod and the microprocessor. The signals can either be hardwired into the microprocessor, or as seen in figure 20 with minimal extra circuitry, allow for a push-button reset switch. Effectively, this switch drops the VCC to ground, killing power to the microprocessor, and pulsing the Test Reset (TRST) line, which halts activity. Because the programming pod provides capability to power the microprocessor, and additional header allows for switching between the programmer power and the board power. Capacitor C1 in figure 20 simply provides debouncing on the incoming signal. This prevents cases where the user depresses the buttons incompletely, potentially causing several partial resets.

7.3 Software Design

The typical programming paradigm for the MS430 is to maximize the amount of time the processor spends in the sleep state. To do this, many of the peripherals on the MS430 chip can be configured to generate interrupts that wake the processor and execute code relevant to the triggering events. The software designed for the E-meter follows this paradigm of sleeping and interrupts, as pictured in 21.

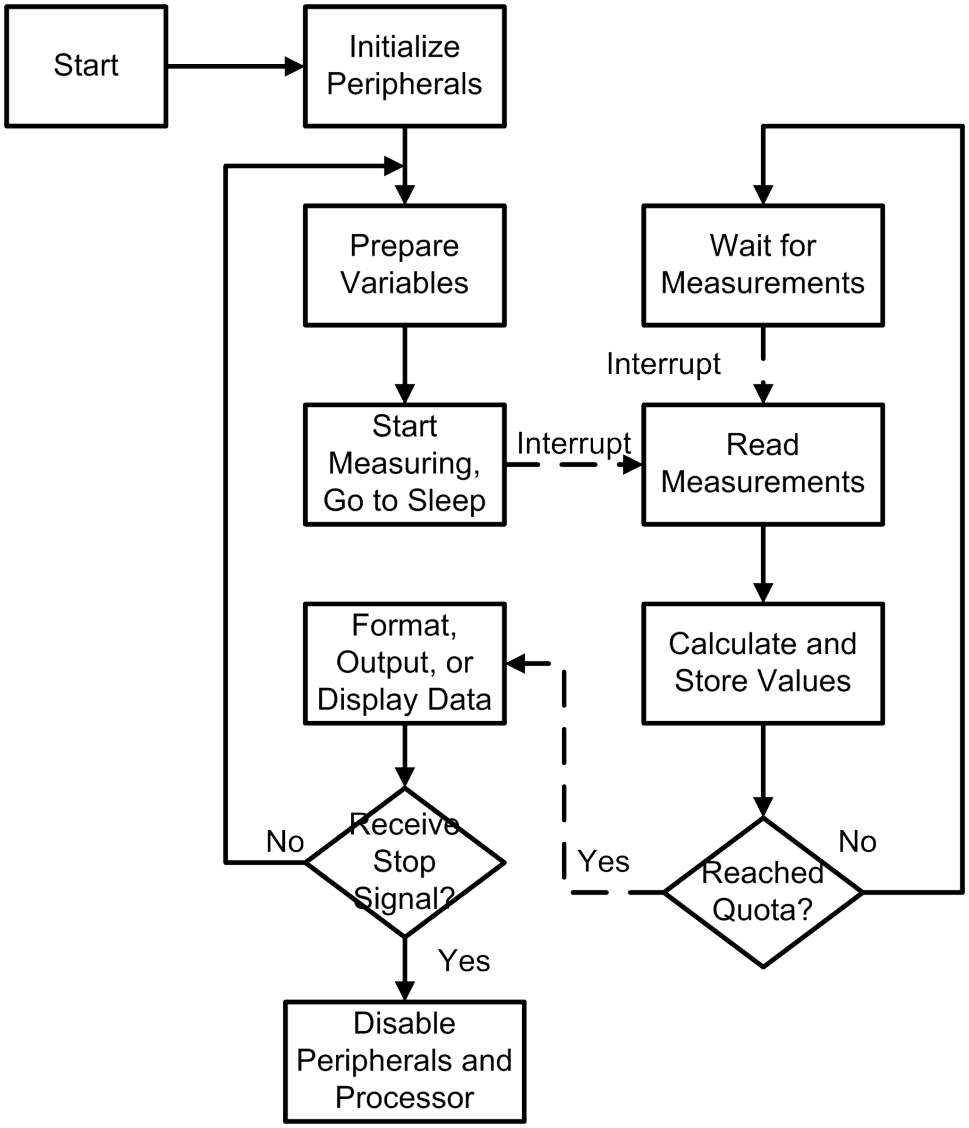


Figure 21: E-Meter Software Flow Diagram

7.3.1 Design

The primary focus of the E-meter is measuring the current load and line voltage of each phase of electricity entering the building. The input networks for the E-meter convert these different elements into voltages that the MSP430 can read on its dedicated ADC hardware. The E-meter software must allow time for these ADCs to resolve the voltages, but must also output summaries of these measurements to its built-in LCD screen and to the base station. To do this, the software running in the E-meter first loads pre-defined values into the device's configuration registers, then measures the voltages on the ADCs, outputs a summary of the information collected, then repeats the collection and output sections. For simplicity, the conversion factors to map these values ranging from $-0x8000$ to $0x7999$ into their real-world values (such as $\pm 120\sqrt{2}$V) are hard-coded into the program at compile time using several `#define` statements.

The initialization step loads values into the MSP430's configuration registers that enable the desired features of the E-meter. The list of possible values can be found in the MSP430 User Manual[17], which design team referenced frequently in determining the values for a suitable configuration. In the current version of the software, the first six ADC channels are "grouped" together while the seventh is disabled. Grouping the channels ensures that all six will measure at the same time, and will only generate one interrupt that signals when all six have finished measuring. The ADCs measure voltages between -0.6V and 0.6V and returns 16-bit two's-complement integers that represent that range. The International Electrical and Electronics Engineers (IEEE) 754 standard requires a 32-bit floating point number to have at least 23 binary places of precision, which could store several of these readings without roundoff error. Additionally, the setup stage selects the clocking frequency for the ADCs and for the serial UART, selects which of the ADCs to measure, and which pins are used for the LCD display.

As the final task in this preparation stage, the program enters a loop where the main calculations take place. At the beginning of this loop, the processor enables the ADCs and puts the main process to sleep.

When the ADCs have measured the voltage from their attached circuits, the "group leader" may raises an interrupt to signal the completion of the measurement. This interrupt triggers the ADC Interrupt Service Routine (ISR), which checks that the ISR was indeed caused by one of the active ADCs. After doing so, the ISR turns off the ADCs, preventing the ADCs from interrupting again before the original interrupt can be cleared. the ISR then reads the volage and current for the first phase. It uses the hardware multiplier to square the 16-bit-integer current and voltage measurements into 32-bit square measurements, and also computes the instantaneous power by multiplying the current by the voltage. Each of these 32-bit integers

are then summed with a corresponding 32-bit floating-point number, a process which takes a considerable amount of computation but allows for much larger numbers to be represented. This is essential for computing RMS: in order to find the mean of the squares, the squares must be summed together without exceeding the limits on their representation. 32-bit IEEE floating-point numbers can express numbers up to 2^{127}, which is many orders greater than the 2^{31} limit on integers; hence, using floating-point numbers for storing sums prevents overflow when a limited number of sums are to be stored. The program defines a quota for the number of samples to take before computing the mean and outputting data. This limit means that a known number of items will be summed, which not only prevents overflow but also ensures that the output from the E-meter will occur at regular intervals. While a timer could provide the desired regularity, connecting the output rate to the measurement rate gives an implicit indication of the measurement rate without requiring special debugging equipment.

In contrast with resetting the processor, waking the main process resumes execution at the same location where it went to sleep. With the newly-measured sum-of-squares for current and voltage for all three channels, as well as the sum of instantaneous power during the sampling period, the main process decides what it needs to calculate for output. In the current version of the software, the E-meter transmits power of all three phases and total power for each run through the main loop; unless the LCD screen displays current or voltage, their Root Mean Square (RMS) values are unneeded. The main loop always calculates power and energy consumed for output and accumulation, respectively.

If necessary, the loop continues the computation of RMS voltage and current by first finding the mean of the squares by dividing the sum of the squares by the number of measurements, then calculates the square root of these mean squares, thereby giving the RMS measurement; multiplying by the callibration factors gives the real-world voltage or current. These values are then output to the LCD screen, as detailed in the

As the power is not measured by squaring, the power sum is simply divided by the number of measurements to obtain the mean real power; the energy total is incremented by the product of the power and the time taken to perform the measurements and calculations, which one of the timers tracks. By applying the callibrated conversion factors, the software computes power in watts and the acculated power in watt-hours.

7.3.2 User Interface

In lieu of a working XBee network, the primary interface to the E-meter uses the single touch-button and the LCD screen. The LCD display uses the four digit places in the upper-right corner to display the current uptime in hours and minutes. The center dial displays the seconds elapsed: every five seconds, another of its twelve regions fills, until it resets at the end of each minute. The upper-left characters indicate which of power, energy, current, or voltage is displayed. The actual value is displayed in the six primary characters.

The six primary characters show the real-time reading of the selected value. The first character only indicates the sign of the number: it is empty for positive values and shows a negative sign when the value is negative. The next three characters display the value in one of two forms: either a number from 10 to 999, or a decimal number less than 10 with two following decimal places. The software automatically determines in which International System of Units (SI) range to place the value to fit these forms. The fifth character displays this range: at the moment, this will be one of “k”, “m”, or “u” for the prefixes kilo-, milli-, and micro-, respectively; the space is blank if there is no prefix. The final character shows the base unit of the measurement: “A” for current in amps, “V” for voltage in volts, “W” for power in watts, and “E” for energy in watt-hours. As there is not enough room for “Wh”, “E” seemed the next most obvious character to put, though it may cause confusion if users expect to see only SI units. At present, the power and energy measurements are totals from all three phases, while the current and voltage read only from the first phase.

The touch-button will turn on the LCD backlight if it has gone to sleep. If the backlight is on, the button will advance the display to the next mode. This very simply interface has not been user-tested, but its simplicity.

7.3.3 Current Status

The present software for the E-meter reads the current and voltage from three separate inputs using six of the seven hardware ADC channels. These channels are grouped together, so they only trigger the ISR when all six have finished measuring. The

At present, the software collects readings from two of the MSP430’s ADC “channels”, simulating what an individual “phase” would be measured. As mentioned in the MSP430 user’s guide, these two channels are

grouped together, so they wait for each other to resolve before triggering the ISR. This approach can be extended to the other channels; with only a few changes in software, all of the six ADCs needed for this application can group together.

Laboratory execution and examination using a stopwatch show that the current software can make approximately 760 measurements per second on both channels when 120 measurements are taken to compute the RMS data before outputting over RS232.

7.4 Printed Circuit Board

Team PICA decided for the purposes of this project that a PCB would provide the best presentation of the final E-Meter. By designing a PCB for the project, the team would gain valuable experience in fabrication, and be able to present a cleaner project. Additionally, a PCB can help prevent parasitic capacitance inherent in terminal connections (in particular between the microprocessor and the LCD). During the time spent debugging the LCD by email with Chuck Cox and John Lupien of SynchroSystems, John mentioned that the stuttering and dim segments seen in our demonstration video could be caused by even a slight extra capacitance. Other components, such as the microprocessor itself, have pins spaced in such a way that a PCB becomes a logical method for easily connecting external components.

In order to design the PCB the team used the freely available program Eagle, produced by Cadsoft Computer USA. Several versions of the software are available, ranging in price and features from a free board-size limited version to an expensive professional no-limits version. As the team budget did not include software licenses, the free boad-size limited version was used to produce the E-Meter PCB.

Originally the E-Meter would be composed of a single PCB containing all of the components except for the power supply. However, as the free version of Eagle only allows for an 80 x 100 mm board, the E-Meter became two linked boards; one for the ADC input networks, and the other for the main E-Meter processing and data transmission components. These two boards can be seen in figures 22 and 23 (larger versions, and complete schematics, can be found in appendix B.6 and B.5 respectively). Each of these boards are composed of 2 layers, a front and a back. The team chose a two-layer board for several reasons:

1. The rapid prototype machine, on which Johnson Controls, Inc. (JCI) allowed us to etch our PCBs only supported two layers.
2. The free version of Eagle only supports creating two-layer boards.

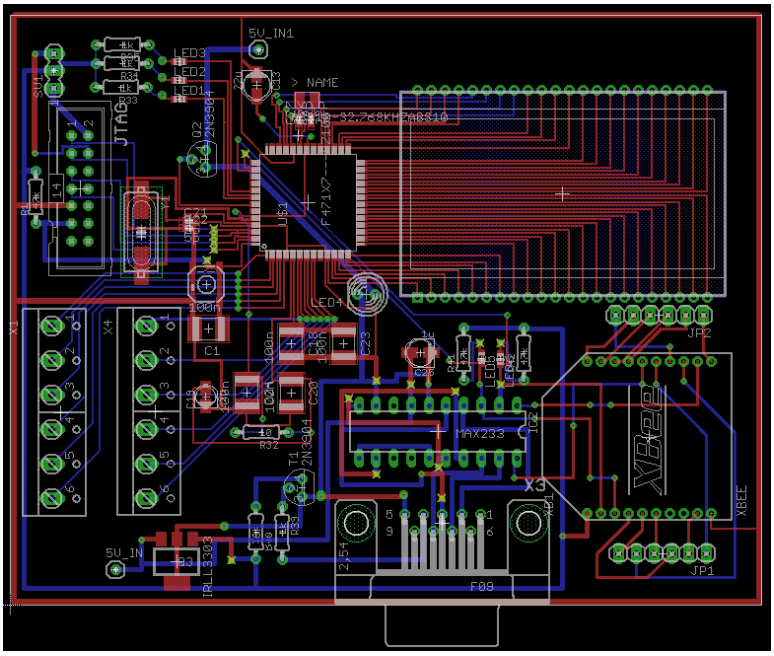


Figure 22: E-Meter main processing and data transmission board.

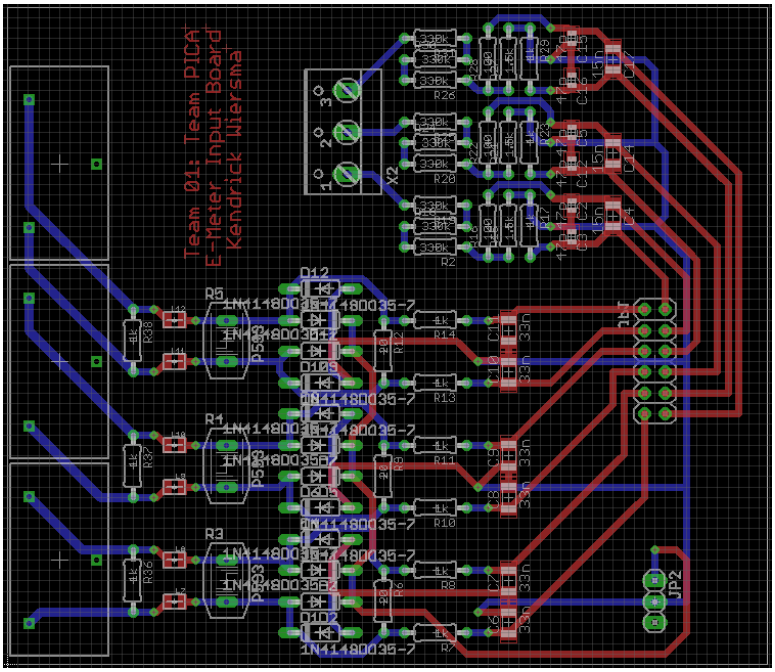


Figure 23: E-Meter input board.

Signals are routed on either side with vias inbetween the two sides. All components are mounted on the top of the board with through-hole components being soldered on the rear.

The main board brings together the MCU, RS232 transceiver, Xbee radio, LCD, user pushbuttons, and JTAG interface. Along with these components, several MCU peripherals, such as clocks, are also included on this PCB. Spatial locality of the components with respect to the MCU determined which components appeared on this board, and which components were allocated to the secondary board. Peripherals such as clocks and transmission signals that are susceptible to noise need to be close to the MCU to prevent interference from electromagnetic radiation in the environment or the surroundings. In order to further reduce the possibility of electro-magnetic radiation, traces carring signals remain isolated (as much as possible) from traces carrying voltage supply. Where not possible, signal traces were routed between ground traces to provide extra shielding.

Eagle provides part libraries containing CAD schematics for many common parts for use in PCB development. Many companies also provide custom part libraries with their part to aid designers in during PCB layout. For this project, the team used the MSP430 library freely available from TI , the SparkFun library, a library from SynchroSystems, and a custom library in addition to the built-in part libraries. Eagle allows users to create custom part CAD files to expand the available library of usable components.

Upon receiving the etched PCBs from JCI, the team verified that the routed traces matched the schematics, and no short circuits were present from mislabeled nets. To check conductivity the team used a Digital Multi-Meter (DMM) set to continuity test; placing the leads on connected traces would cause the DMM to emit a beep, while non connected traces would result in no sound. After the boards passed verification they were returned to JCI to have all surface mount components populated by Joshua Sliter who works in the labs at JCI populating prototype boards. The team then added the remainder of the parts. The completed input board and main board can be seen in figures 24 and 25 respectively.

7.5 Testing

In order to verify the performance of the PICA E-Meter, several tests stressing various components of the system are described below. These tests verify both functional and behavioral correctness of the E-Meter in various situations, responding to several different stimuli.

1. Verify that the connection on the board are reliable. To do this, perform continuity tests on pins that lie on the same PCB traces.

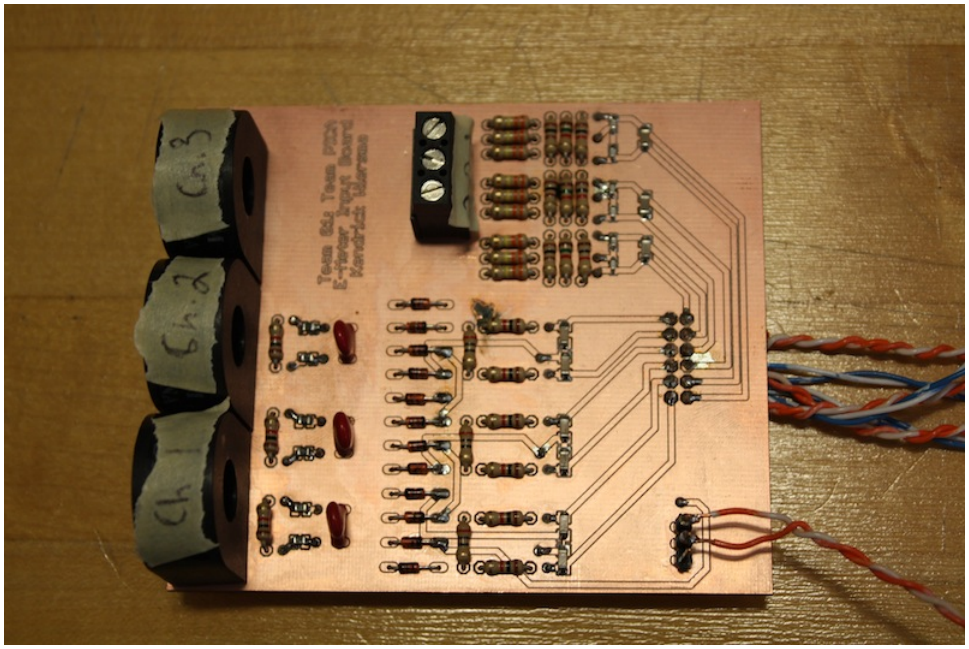


Figure 24: E-Meter input board fully assembled.

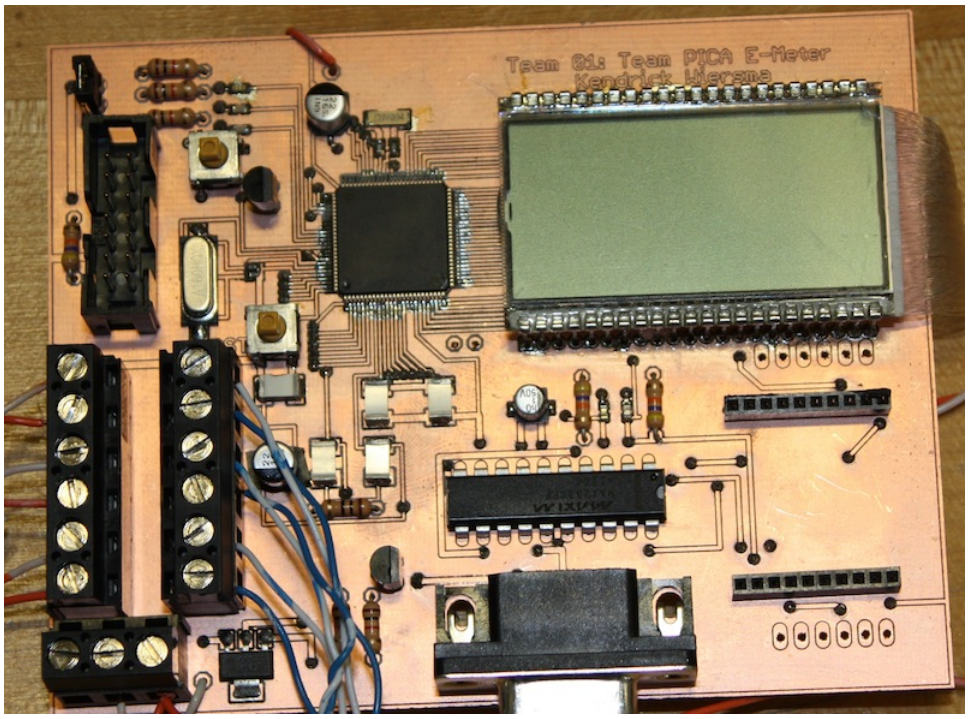


Figure 25: E-Meter main board fully assembled.

2. Verify that the board contains no ground or board shorts. Perform continuity tests between the ground pin and the signal pins, as well as between the VCC pin and the signal pins. If any are connected, verify that this should be the case.
3. Verify that the processor can be controlled via JTAG. With external voltage applied, attach the JTAG pod and re-program the device with the E-meter software by using Code Composer Studio. On the prototype board, removing the RS232 cable increases the success rate of the this procedure.
4. Verify that the processor can run without the JTAG pod. After programming completes, remove the JTAG pod and press the reset button. The processor should launch the software automatically. The LCD screen should display "00 W".
5. Verify that the SD16s are clocked correctly. The three signal LEDs should display a binary counting-up sequence and the LCD screen should show some changes.
6. Verify that the 32.768kHz clock is correctly connected by enabling the backlight and waiting for the timeout condition to turn off the backlight.
7. Verify that the RS232 transmission functions properly by attaching an RS232 cable between the E-Meter and a computer, opening a terminal and verifying that characters are being transmitted.
8. Verify that the E-Meter is responding to commands via typing the character 'q' after performing the previous test. The E-Meter should halt.
9. Verify that the input board is properly attached to the E-Meter by attaching a load and verifying that the display reads a value within 5V of 120V.

7.6 Future Work

7.6.1 Hardware

The prototype of the PICA E-Meter functions properly but it requires several improvements before mass production could begin. First and foremost, the Xbee radio socket does not allow for transmission of data over a wireless link. In fact, the Xbee radio that team PICA first attached to the Xbee socket no longer functions properly when attached to the Xbee developer boards. This observation leads to the natural conclusion that there exists an error in the MSP430 to Xbee interface developed for this project. Likely, the problem lies in the flow control circuitry which did not undergo a full test before being integrated into the

final prototype. Secondarily, and potentially related, the RS232 port on the prototype seems to create issues during the JTAG process. Mainly, when attached, the RS232 cable prohibits the ability to reprogram or debug the MSP430. Team PICA does not fully understand the nature of this particular point of failure, however a simple workaround is to detach the RS232 during program loading.

A second revision of the E-Meter PCB would allow for many of the lessons learned during assembly of the first PCB to be incorporated:

1. Plan on soldering the LCD to the back of the PCB.
   (a) In the current design the traces between the MSP430 and the LCD are routed on the top of the board, meaning that the LCD must be top-soldered to the board.
   (b) An unfortunate side-effect of this tweak would be the addition of 40 vias to the back of the board, making the initial assembly more difficult.
2. Fix the mislabeled net, and attach VCC to the MSP430 processor without the need of a jumper wire.
   (a) The first revsion schematic contained a mislabeled net which resulted in having to attach a jumper wire to the prototype PCB attaching the 3V supply to the correct pins of the MSP430 processor.
3. Move the TX and RX LEDs to the TX and RX lines on the other side of the RS232 driver.
   (a) Currently the TX and RX LEDs are attached between the MSP430 and the MAX233A RS232 driver IC. While the signal will still transmit, the MSP430 cannot drive the LEDs enough to illuminate them.
   (b) Moving the LEDs to the other side of the MAX233A driver would allow the higher voltage drive from the IC to illuminate the LEDs during transmit/receive activities.
   (c) This fix would greatly improve debugging, especially if the RS232 bug still exists.
4. Add better labels to components that are sensitive to orientation, reducing the difficulty of populating components to the board.
5. Ensure that the clocks are surrounded entirely by ground traces, prohibiting the clock signal from leaking into other components on the PCB.
   (a) While it is not obvious that this problem currently adversely affects any components on the PCB, a few small changes would prevent signal leakage.

6. Move the backlight LED to the correct position on the back of the PCB underneath the LCD.
    (a) This removes another small jumper wire used by the team after learning that the backlight fiber-optic lines were not as flexible as previously thought.
7. Route the 5V signal from the empty pin on the 3-pin header to the voltage regulator as opposed to using wire jumpers.
8. Fix the crossed trace on the input board that prevents the second current sense network from operating properly.

A second revision of the E-Meter PCB would also allow for several enhancements:

1. Additional buttons for a simpler user interface.
2. Move buttons to more easily accessible position, further simplifying the user interface.
3. Include space for adding mounting brakets to the PCB.

7.6.2 Software

To reduce the impact of noise in the ADCs, a digital low-pass filter could be implemented to reject noise in the measurements. In order to do this, however, the E-meter must be sampling at a rate high enough to identify higher-frequency fluctuations as distinct from low-frequency signals like the 60Hz waveform from the AC lines. Insufficient sampling rates would lead to aliasing: signals of a frequency higher than the sampling frequency would appear as a low frequency, and would not be rejected. In light of this, the E-meter software can be optimized in several ways to increase the sampling frequency.

1. Using the ISR to read data from the ADC registers, square and multiply them as usual, then write these values to a shared data structure that the main loop can process between ADC conversions. This approach would require a means to keep the ISRs from overwriting data before the main process can read it, but would allow the ISR to complete more quickly, thereby allowing the ADCs to perform more conversions.
2. Creating separate variables for data being read and data to display, where the read data is copied to the display variables when the display functions have completed. This would allow the output loop to run in parallel with the ADCs, as the ISR would not endanger any data being used for calculations.

3. Using 32-bit “long” unsigned integers to store the sum-of-squares would require some mandatory loss in precision to prevent overflow, as the mutliplication and squaring produce 32-bit numbers by default. The payoff for this method would be an indirect reduction the impact of small-signal noise in the measurements, but also a substantial speed-up of the measurements in the ISR, as integer-to-float conversions and floating-point addition would be effectively eliminated. Floating-point numbers may still be required to calculate the real-world units, but converting number types once would still greatly improve the speed of the program.

These methods may not be particularly difficult to implement, but since they were not required to produce a working E-meter prototype, the current software does not use them. Building a software low-pass filter will not be a trivial task in and of itself, but it should be rewarding nonetheless.

In the event of successful wireless communication, the data transmitted should be encrypted to prevent digital eavesdropping and improve privacy. While very secure encryption schemes boost privacy, the limited ability of the MSP430 would likely call for a balance of security and speed, possibly at the user’s preference. The corresponding decryption software must be run on the base station as well.

The primitive interface can be expanded to show statistics on multiple channels, rather than just the first. To establish a good interface, the team should gather user reactions and behavior with each iteration of the design. If no buttons can be added, the software might be expanded to distinguish between a button press and a button hold.

The E-meter software currently lacks a method to be configured, aside from compiling new code. To create a more customizable system, the E-meter should be made able to accept commands from the base station. For example, these commands could alter which measurements are sent to the base station, or how often they are relayed. As the base station software currently does not support sending such commands, the E-meter software and the base station must both be revised to support this communication.

8 Smart Breakers

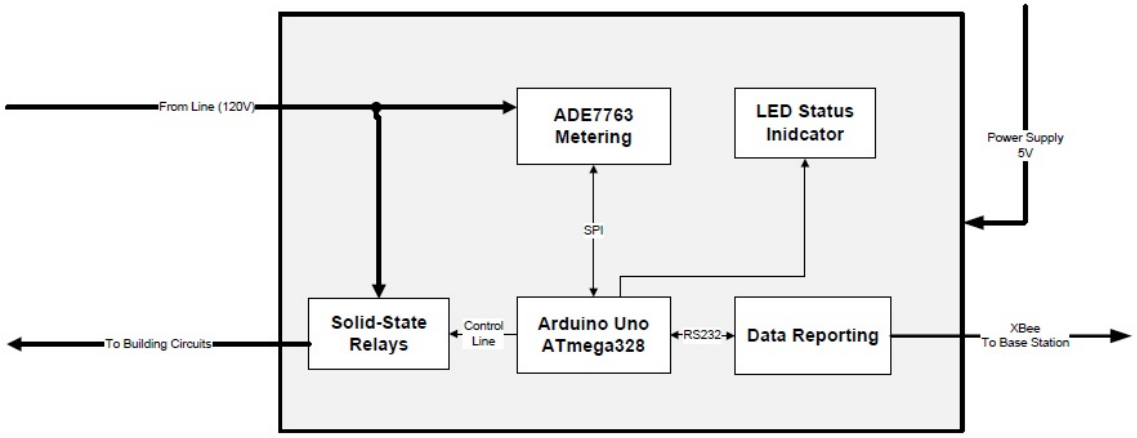


Figure 26: Block diagram of breaker system

The smart breakers make up the part of the full system that measures power to each individual circuit and consist of a monitoring section and a switching section. Designing the two functions into the same subsystem allows for a simpler and safer installation. Without the switching capability, the user would have to add the monitoring component outside of their breaker panel, which adds more electrical connections and would likely require a new design for the breaker box. As a whole, the smart breakers were designed to be more of a proof of concept than as a final design. Much of this came from the fact that very few systems exist that provide individual circuit level information on a large scale.

8.1 Breakers

8.1.1 Breaker Overview

The breaker system is half of the smart breakers system aimed at providing the consumer with better circuit protection in addition to circuit information.

In addition to allowing for a safer installation, using solid state technology improves safety through faster response time. Accuracy can also be improved since the control logic for the switch is much easier to calibrate than an induction coil which is what breakers currently use. Cost of a smart breaker is higher because of the added switching feature, but the team decided that it was acceptable because of the benefits of having a safer system and improved functionality.

8.1.2 Breaker System Diagram

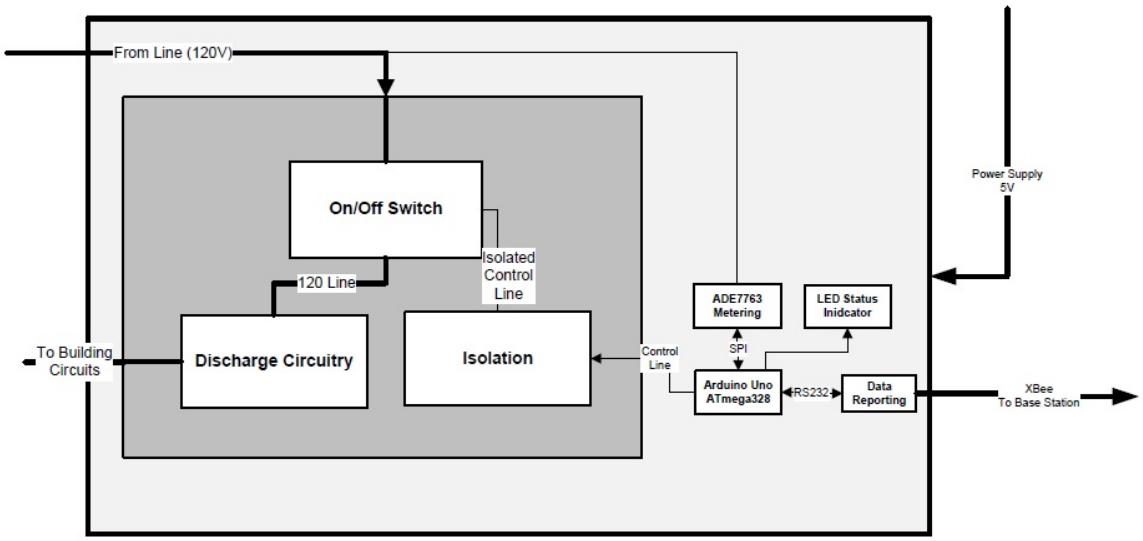


Figure 27: Block diagram highlighting the breaker components

The diagram above shows the breaker technology within the outlined box in relation to the other aspects of the smart breaker. For the design of the switching part of the breakers, the isolating circuit, the discharge circuitry and on/off switch are the three key components needed for a safe and fully functional breaker system. The controls and monitoring are also important for a fully working system, but the team decided those functions were better handled by the monitoring part of the smart breakers.

8.1.3 Component Selection

On/Off Switch

The team focused initially on selecting an appropriate switching component and looked at SCRs, triacs, FETs, thyristors and solid state relays as possible options based on their ability to work well with high voltage and current levels. The table below shows a few additional benefits and drawbacks of each.

Each of the options does require the use of heat sinks, which is not ideal for a final design because it is difficult to dissipate heat from inside a standard breaker box. This one of the first parts of the design that would need to change for the transistion from proof of concept to final product to happen. Ultimately, the team decided to use solid state relays, due to their ease of use for the designer and lack of need for additional circuitry, which would require significantly more time for parts to arrive and for assembly. While solid state relays may not be optimal for a final design, the team decided to spend more time working on the

Components				
SCR	Triac	FET	Thyristor	Solid State Relay
Benefits				
Work Well With High Current and High Voltage				
		We've Worked With Them Before		Easy to Use
Drawbacks				
Require Heat Sinks				
Require Additional Circuitry				
Require Complex Triggering				Expensive

Figure 28: Table comparing switching components

primary goal of measuring power. More testing needs to be completed to determine what the efficiency of the switch is since any additional power used by the system has to be subtracted from the total saved.

Isolation Circuit

The isolation circuitry's purpose is to protect the control logic should any line voltage cross over to the control side. The team selected an opto isolator for this purpose based on availability, ease of use, and functionality.

In the prototype design, the functionality of the isolation is somewhat compromised since it uses only a single power source. However, since the breakers were primarily a proof of concept, the team decided it was acceptable to spend more time on other areas of the project instead of physically building the second power supply needed for a final design.

Discharge Circuitry

At high currents and voltages, suddenly turning the circuit off can cause damaging transients. The discharge circuitry captures those transients before they become a problem and dissipates them later in a safe manner. In a final design, energy storing components, primairly capacitors or inductors, would be used to quickly absorb the transients before discharging them at a much slower rate. The team did not include this aspect in the prototype since it is not critical to demonstrating the smart breakers' ability to read power.

8.1.4 Testing

Following selection of the switching component, the team ran a few basic tests and found that at low current levels the solid state relay functions as well as standard air-gap breakers. The comparison to standard air-gap breakers was measured by visually watching for transients using an oscilloscope while turning the relay and breaker on and off.

With loads between 150Ω and 1.5kΩ attached to the relay, the turn on voltage remained constant at 2.8 V. This was determined by increasing the control voltage until the load circuit turned on. Similarly, the turn off voltage was tested by lowering the control voltage until the load circuit turned off, and again within the range of loads attached, the turn off voltage stayed constant at 2.8 V. Although the team hoped to design for currents up to 20 A, the ability to switch currents higher than 1 A was not tested, primarily because the discharge circuitry had not been developed. After implementing the discharge circuitry, the same test would be done, except with loads that require much higher current. The microcontroller would also have to have a new threshold value programmed, but this will not affect how the hardware works.

8.1.5 Future Work

Solid state relays work well to demonstrate the smart breakers' ability to monitor and control electrical power, but are not a good selection for a consumer device because of high cost and need for heat sinks. Given more time, the team would like to design breakers that make use of another switching component that would be selected based more on cost and minimizing the need for heat sinks instead of easy use in a design. Adding discharge circuitry is another aspect the team would like to include.

8.2 Individual Monitor

8.2.1 Problem Definition

The individual circuit monitors provide the means for the homeowner to know where power is used within the building and makes up the second half of the smart breakers, along with the switching circuitry. They measure voltage and current in the connected home circuit, use it for controlling the breakers and feed the information to the user through the base station. Homeowners can then use the information to adjust their power consumption habits, while the control logic uses the information to detect over-current and over-voltage situations and change the switch's status accordingly.

8.2.2 System Diagram

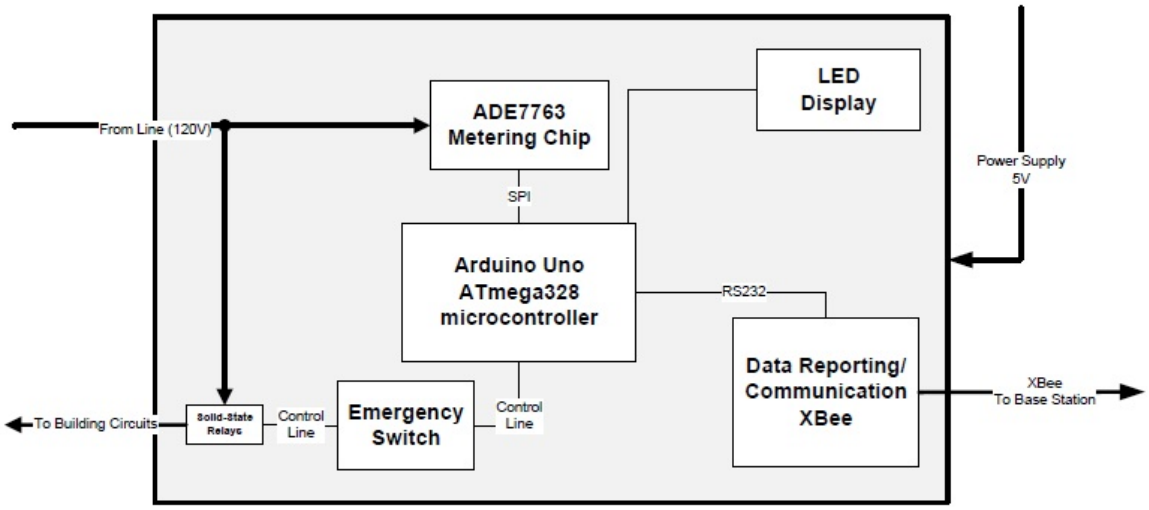


Figure 29: Detailed block diagram of monitoring part of the Smart Breakers

The above diagram shows the components of the individual monitors and their relationships to each other and the breaker section of the smart breakers.

8.2.3 System Components

Metering Chip

The purpose of the metering chip and its inputs is to measure instantaneous voltage and current as it enters the home and pass the data to the controller.

At the beginning of the year, the team found ADE7763 metering chips from Analog Devices and obtained free samples. As an IC designed for energy metering [18], the chip has all of the functionality needed for the project. At a minimum, the chip needs to read voltage and current with additional measurements being nice but not necessary. The team also found several other chips that perform similarly, but due to cost to the design team, the ADE7763 metering chips were used. The team recognizes this is not good customer driven design, but for proving that metering is feasible at the individual circuit level, the decision was acceptable.

Input methods for the ADE7763 were selected based on the AN 564 app note [1], using a current transformer and voltage divider to measure the power line. Both work well for the project because they are simple and accurate. The team had originally hoped to keep the cost to the consumer below $20 so that

replacing an average of 20 standard air-gap breakers would cost less than $400, which is comparable to similar products (see Business Competition section for more). The high turns ratio current transformer's cost of $14.23 pushes the smart breaker cost to $10 more than originallly hoped, but was easily available. An alternative considered after the decision to purchase the high turns ratio current transformer was a lower turns ratio current transformer at a lower cost, combined with a voltage divider. However, implementing this would have used up more of the budget, which the team wanted to save for other aspects of the project. The voltage divider is both easily available and low cost, although it does not provide good protection against lightning strikes, which could present a problem since replacing only one individual component of the smart breaker is not feasible. Selected components and values keep the input levels lower than the maximum value of 500 mV[18] for the ADE7763 channels using the equations below.

$$V_{secondary} = \frac{(I_{primary} * R_{burden})}{2511} \tag{4}$$

This gives the voltage across the secondary of the current transformer and comes from the datasheet for the current transformer [19]. From this equation, R_{burden} is selected as 60Ω, allowing the current transformer to be used for up to 20A.

$$V_{measure} = V_{line} * \frac{R_{measure}}{R_{total}} \tag{5}$$

$$P = \frac{V^2}{R} \tag{6}$$

Equation 5 is the standard voltage divider equation used to determine values so the maximum input voltage is not exceeded. Equation 6 is the power equation used to determine resistor values so they don't draw too much power. To keep $V_{measure}$ under 500mV, $R_{measure}$ is 1kΩ with an R_{total} of 511kΩ.

$$f_{3dB} = \frac{1}{(2 * pi * R * C)} \tag{7}$$

Equation 7 is used to determine the 3dB point of the attenuation network in Figure 30 (an-564). A 22nF capacitor with the $R_{measure}$ from above gives a cutoff of 7200Hz as suggested by the app note [1].

Microcontroller

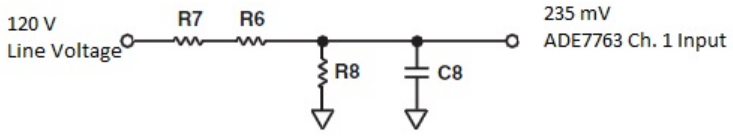


Figure 30: Attenuation network for ADE7763 [1]

The microcontroller's job is to transfer the data stored in the metering chip's registers to the base station and does so through the SPI connection built into the chip. It also provides the logic needed to detect and turn off the breaker in an over-voltage or over-current situation. Other functions include managing start up and shut down, primarily initializing registers and saving data.

$$V_{voltageMax} = V_{line} * \frac{R_{measure}}{R_{total}} * N_{safetyFactor} \tag{8}$$

$$V_{currentMax} = \frac{(I_{limit} * R_{burden})}{2511} * N_{safetyFactor} \tag{9}$$

The manufacturers of the smart breakers would use equations 8 and 9 to determine the exact resistor values based on the desired part ratings. Both $V_{voltageMax}$ and $V_{currentMax}$ equal 500 mV, as specified by the datasheet [18] and $N_{safetyFactor}$ is a value greater than 1 which the team decided to be 2 and 1.5 for voltage and current respectively. This allows spikes up to 30 A for current and 255 V for voltage without damage to the chip. To determine the threshold values, the resistor values from equations 8 and 9 are put into equations 4 and 5, giving the analog values for the microcontroller to use.

First semester, the team decided to use an FPGA for this purpose as they are very flexible and can easily manage data from the metering chip. The decision did not meet the criteria of low cost, but was justified by the higly flexible nature and availability. However, the design process for the rest of the breakers took longer than anticipated and the team decided to use the Altera Nios II processor available from Calvin. The team wanted to use the DE2 board to start testing while they continued to decide on a part for a final production design. Calvin also had some pic microcontrollers, but the team is not familiar with them and would have had to learn new protocols and language, making the design more complex than the Nios II which had been used in other classes before. After working with the Nios II function alt_avalon_spi_command(,,,,,,), the team discovered there is very little documentation for the function, making it very difficult to work with and no alternative was found for using SPI. The team decided to look

at possible alternative microcontrollers and found an Arduino Uno that uses an ATmega328 microcontroller and has more and clearer documentation. The team used the SPI.transfer() function to pass information, with full code in Appendix C This allowed the team to begin testing, although a cost comparison on the microcontroller for final production has not been completed yet.

Communication

The purpose of the communication component is to transfer data from the smart breakers to the base station and is controlled by the microcontroller.

First semester the team decided to use Ethernet to achieve this, based on compatibility with the base station, low cost and ability to connect several breakers. However, longer than expected design time for other aspects of the project made it necessary to use a simpler method. The team chose XBee radios instead, transmitting wirelessly with an RS232 connection to the individual components. XBee also removes the need to have external wires connecting the breakers and base station, making the product more aesthetically appealing to the customer in addition to providing a safer installation.

Display

The purpose of the display is to provide the user with basic on/off information relating to the breakers should the system fail and require a manual shutoff. This meant it must be simple and easy to understand, which led to the decision to use well-labeled LEDs. Because LEDs are very easy to include in a schematic, using them instead of a character-based display also reduced design time.

Controls

Because it can be critical to shut off power, the team decided to include a physical failsafe method in addition to the system's ability to do so automatically. Because of the way the automatic switching is configured, this backup switch would only allow the user to turn power off. The team decided to eliminate the option to override the automatic controls and force an over-voltage or over-current situation to protect the user if they haven't actually cleared the fault.

8.2.4 Software

Figure 31 shows the microcontroller's operating process for normal operation and what it does in both an over-voltage and over-current situation. Because the purpose of the breaker is to shut off power when unsafe conditions are detected, this is the first thing it does. Since the cause of the unsafe condition is

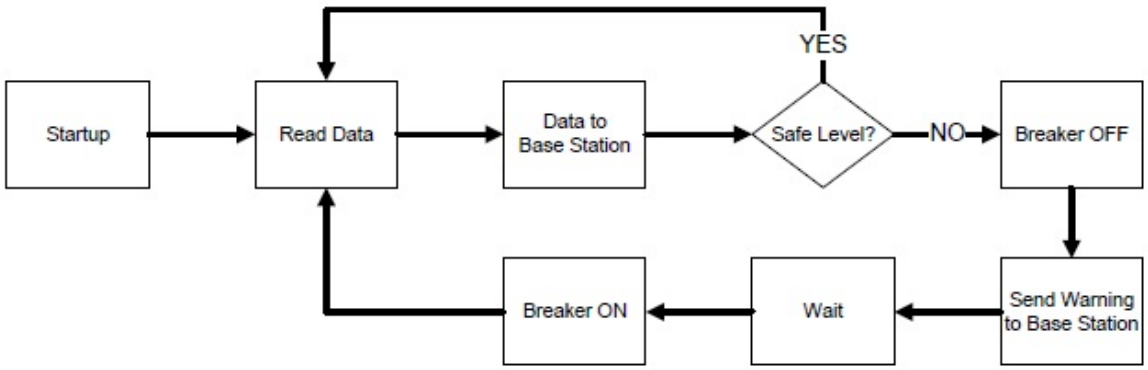


Figure 31: Software flow diagram for Arduino microcontroller

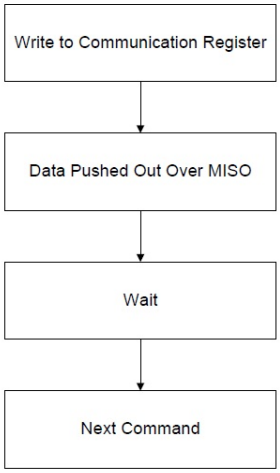


Figure 32: Read/write process for ADE7763

unlikely to remove itself automatically, the smart breaker then alerts the user so they can address the cause. In future work this alert routine may be modified so the user is not overwhelmed with alerts if they do not immediately see the early warnings. The system then waits before taking further action, allowing the user time to clear the cause. Again in future work, this routine may be modified so it waits for a longer period of time if more time has passed since the unsafe condition was initially detected. The last part of the loop for when an unsafe condition is detected is turning the breaker back on. Even if the fault has not been cleared yet, this is a necessary step if the breakers are able to automatically reset since they have no other way of measuring current. This does pose a risk to the user since it allows the over-voltage or over-current condition to continue, but with the line voltage at 60 Hz the system will be able to shut down again within 0.05 seconds, assuming 3 cycles to obtain a reading. The team decided that 3 cycles would provide accurate data without exposing the user to unnecessary danger.

Voltage and current are both necessary to calculate power, so the team focused on reading those values first. Other features of the ADE7763 may be used if the project were given more time, but as they're not critical, the team didn't deal with them since the ADE7763 makes it easy to ignore information it collects. Some features would include line frequency, apparent and real energy, and possibly sag.

Figure 32 is an expanded view of the read operation in Figure 31. According to the datasheet [18], a read operation is specified by a zero, followed by the register to be read. However, during testing of the microcontroller, this only returned meaningless data.

8.2.5 Final Decisions

To summarize, the individual circuit monitors consist of an Arduino Uno microcontroller and Analog Devices ADE7763 metering chip with input networks using a voltage divider and current transformer. The unit will communicate with the base station computer through the Arduino to computer interface directly, using SPI for communication between the ADE7763 and Arduino Uno. Minimal on/off displays and controls including LEDs and a physical switch will be provided to ensure user safety as well. The system will use a solid state relay to demonstrate that the microcontroller is able to control a switching device, but will use a yet to be determined technology for a marketable product.

8.2.6 Printed Circuit Board

The team designed a printed circuit board for the purpose of displaying and testing the prototype design of the smart breakers. It uses parts listed in the Bill of Materials found in the business section and follows the schematic and layout (figures 33 and 34). After having the board printed and populating it, some initial tests were run, during which a connection exploded, making the board unusable. A second version with a few minor modifications was laid out and put on vector board to replace the first version since the team did not have time to reprint a full board for version two.

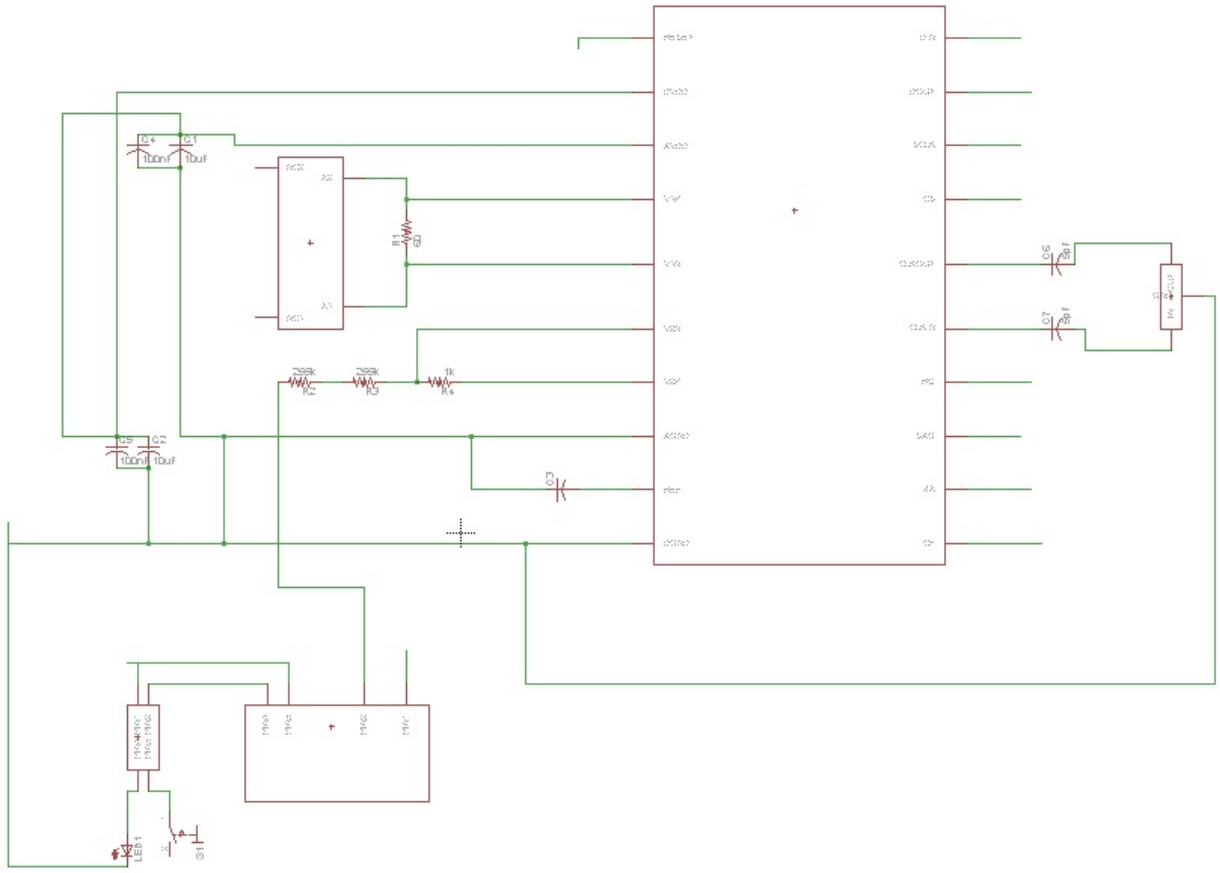


Figure 33: Schematic for Smart Breakers

Figure 33 is the first version of the smart breaker schematic as drawn in Eagle. Component values were selected as specified in the Component Selection section above. The clock and C1-C5 were specified by the ADE7763 datasheet [18] and C6 and C7 are specified by the clock datasheet [20]. Changes were made to the optocoupler after the board was printed after recognizing the improper connections. For version 2, pin 1 of the optocoupler was connected to a power source instead of the control line for the solid state relay and R_1 changed to 42Ω to account for a larger safety factor. The board and schematic were not changed

following the redesigning since the team was using vectorboard for the second version. The schematic does not include the microcontroller since the team decided to locate it off-board so it could be reused later.

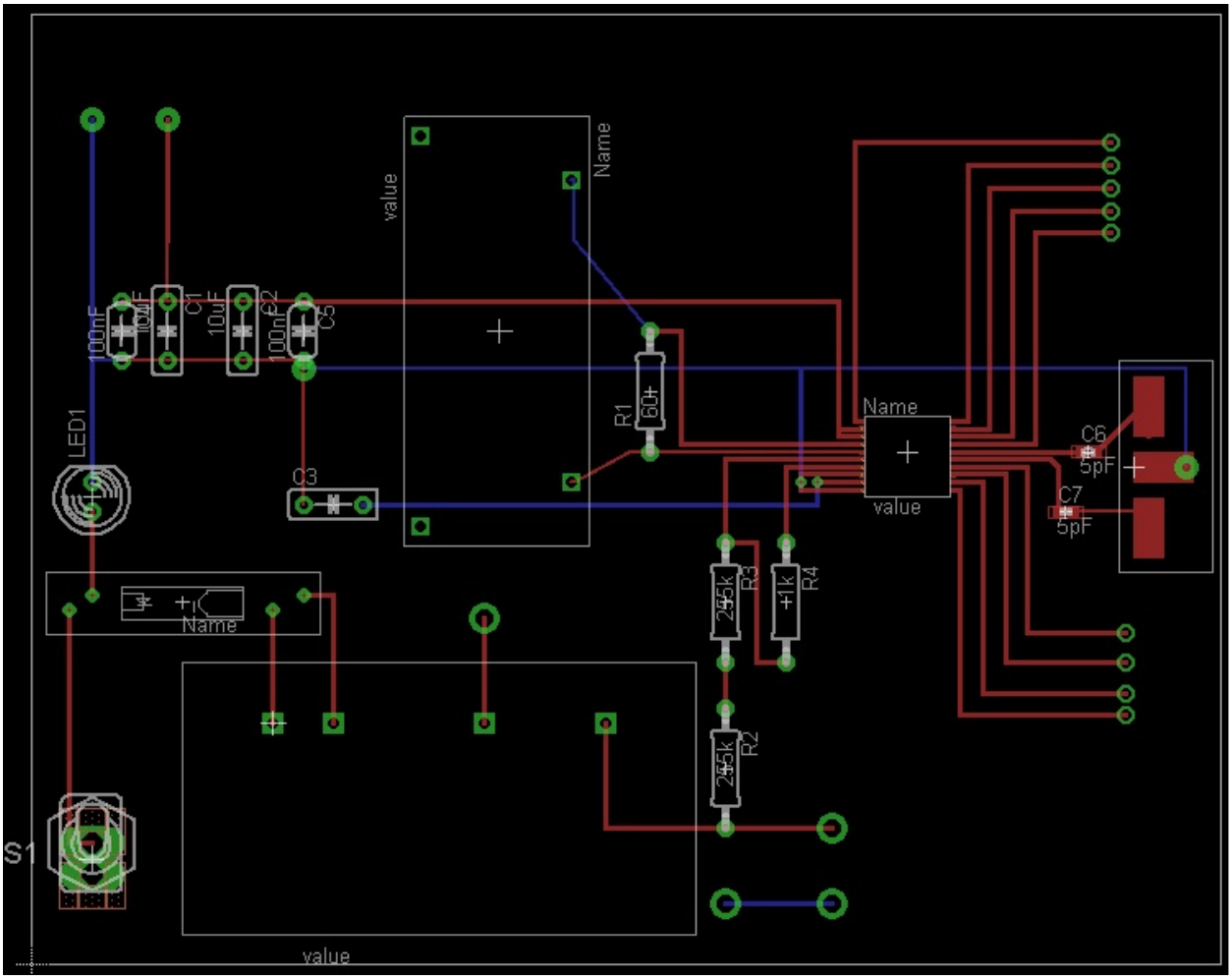


Figure 34: PCB layout for Smart Breakers

Figure 34 is the first version of the smart breaker layout for a printed circuit board as drawn in Eagle. Little consideration was given to minimizing board space since getting used to Eagle and making all the connections was more important. The ADE7763, solid state relay, current transformer, clock and optocoupler were defined by the team, following the guidelines at instructables.com [21]. The microcontroller is again not included due to it's off-board location.

For the second version of the board, a few changes would be made including a much more compact design with more complicated traces to minimize cost. Also, more traces would be put on the bottom layer to make soldering easier. The team would also use larger traces and pads for connections to the 120 V to

handle higher currents.

8.2.7 Testing

The table and graph below show the results of current transformer testing, during which a 6.2kΩ burden resistor was used instead of the 42Ω selected for final design. This made it easier to read results using an oscilloscope but significantly limited the current threshold as specified by equation 9. Table 4 gives the component values used and the resulting voltage and current measurements, both with percent error from values calculated based on measurements on the primary side. Table 3 shows the inputs used to obtain data for table 4. From the data the team confirmed that the current transformer provides a linear relationship between primary and secondary current, and concluded that the chip should be able to read accurately with minimal correction factors. Figure 35 shows this relationship graphically.

Table 3: Measured:Primary

Voltage(V)	Power (W)	Resistance (Ω)	Current (A)
117	40.00	342	0.360
117	20.00	684	0.250
117	13.33	1027	0.210
117	10.00	1369	0.200
117	80.00	171	0.690

Table 4: Measured:Secondary with Percent Error

Voltage(V)	% Error	Resistance (Ω)	Current (mA)	% Error
0.840	5.5	6200	0.135	5.8
0.580	6.0	6200	0.093	6.6
0.490	5.5	6200	0.084	0.4
0.440	10.9	6200	0.069	13.4
1.620	4.9	6200	0.265	3.6

Figure 36 shows the results of testing the low pass filter on channel 1 of the ADE7763 metering chip. From the results, the team concluded that the low pass filter works as expected, giving a corner frequency of approximately 10kHz as recommended by the datasheet [18] to prevent aliasing.

8.3 Future Work

While the system appears to work well for a prototype, there is still much that needs to be completed before a final product can be released. All of the input networks function properly, the breaker part performs the

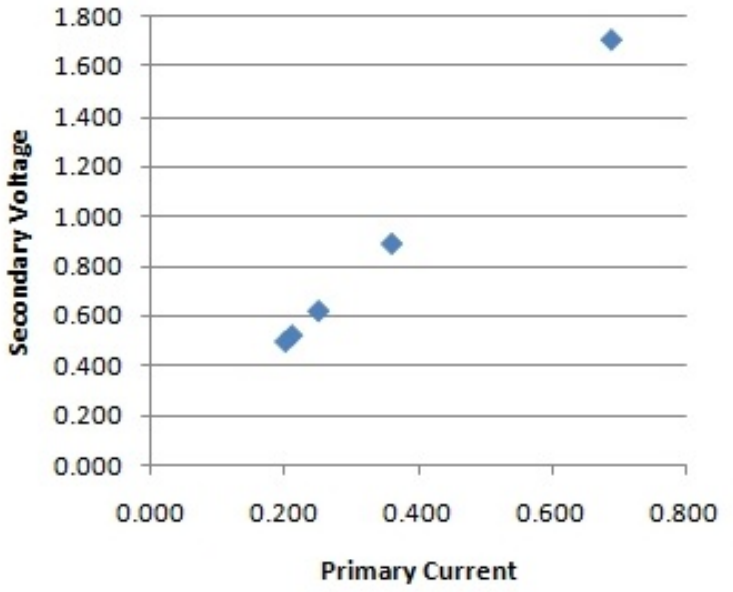


Figure 35: Current-Voltage Response of Current Transformers

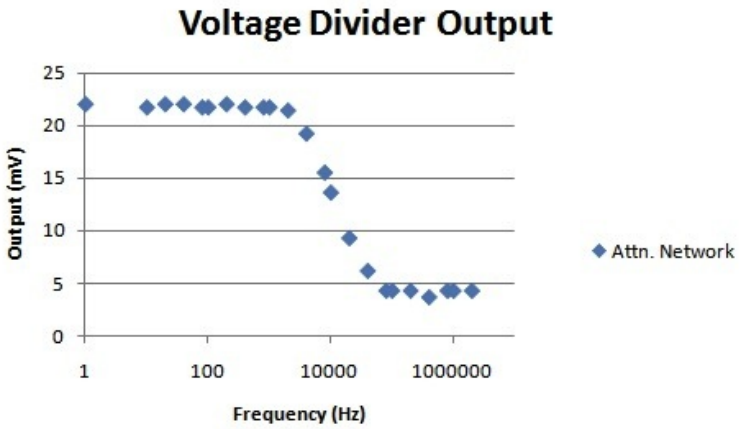


Figure 36: Graph showing results from voltage divider testing

functions needed, and the team can communicate with the microcontroller. However, the functionality of the ADE7763 has not been verified since the microcontroller could not communicate properly with the chip. The team still needs to test the smart breakers' functionality using non-linear loads, designing communications for several units at the same time, effective solid state technology for switching purposes and selecting/implementing a different metering chip and microcontroller.

No adjustments should be necessary for non-linear loads, but since large motors often draw a significant amount of current upon startup, the team would like to test to make sure this is not a limitation on the functionality.

Right now, the prototype monitor is set up so that a single monitoring chip reports directly to the computer. However, in a typical home, multiple devices are expected to be used simultaneously, requiring a more complicated process for passing data to ensure no data is lost. Assuming a system with minimal data transmission lines, the team recommends the use of a Master Control Unit to manage the monitoring devices prior to transmitting data back to the base station. Another concern is the bandwidth and ability to handle real-time data from over a dozen devices. If the user desires high precision and frequent readings from many monitoring devices, a single ZigBee system (currently the most likely option) will not be sufficient.

Designing a good solid state breaker to work in the small, un-cooled space provided in a breaker box is another step the team would like to take. However, without changing the design of a typical breaker box, the limited access to cooling and the voltage and current thresholds required make this a very difficult step forward. A new style of breaker box could be provided to allow for better cooling, but this would be completely contradictory to the team's goal of providing a simple and safe installation, requiring a skilled electrician to re-wire the box for the homeowner.

For production purposes, implementing a different metering chip is another critical stage. The ADE7763 from Analog Devices works well as a proof of concept, but is not the most cost effective solution for the consumer. Another possible benefit is that the microcontroller will work better with the chip its designed to work with. This should help utilize more of the available functionality in the metering chips, continuing to provide the customer with more information.

9 Circuit Identifier

9.1 Purpose

This device should allow users to accurately map individual outlets to a specific circuit/breaker, by temporarily using/manipulating power from that outlet, which means that it will not be permanently plugged into the circuit.

9.2 Requirements

1. Shall run off of 120V line
2. Shall use or manipulate power for a time interval such that DSP can recognize the device is “on”.
3. Shall not adversely affect other devices on the circuit.

9.3 Design Criteria

1. Should be user friendly/intuitive
2. Should be easily portable
3. Efficiency is not a concern
4. Cost
5. Overall system should require minimal user input

9.4 Ideas

9.4.1 First Idea Descriptor

The Identifier must have a master controller (or something similar that is back at the service panel) identify the circuit into which it is plugged. Something that might work is an X-10 power line control module or similar. There are some people who have a device that allows you to turn on and off lights remotely. There is a device you plug in between the socket and the lamp that receives the signal. There is also a controller you plug in to any plug (not sure the plug has to be on the same circuit, but that does not matter in this

purpose). The controller sends a high frequency signal down the line that the receiver intercepts and interprets and an on/off/dim signal. Therefore, the control box could be the thing the team is looking for. The control box is basically a box with a few buttons and a 120-VAC cord. You plug it in and press a button. The controller at the panel sees the high frequency signal come through, figures out which circuit it came from, and we have our detection.

9.4.2 Second Idea Descriptor

There are also things that were found online that might work. One of them is called an Extech RT30 Wireless Remote AC Circuit Identifier. It has a clamp that acts as a probe to connect to wires that will have an indication as to if the circuit is being used or not. The data sheet can be found at http://www.globaltestsupply.com/datasheets/RT30data.pdf

9.4.3 Third Idea Descriptor

The other thing that was found online is called an Automatic Circuit Identifier with Digital Receiver. This is a more of a manual approach to see which circuit is on and which is off. It is not the ideal method, but it is an option nonetheless. The information on this can be found at
`http://www.idealindustries.com/media/pdfs/61534_manual.pdf`

9.5 Decision

The control box idea was a great idea, but the team did not believe they had enough time to incorporate the design into their project because it was not ready for use directly out of box. It would require more design work than the team had time for. Therefore the team decided to go with idea number 3, which was the Automatic Circuit Identifier with Digital Receiver. It would work very well out of box, and provide the team with the functionality needed to complete this part of the project in a timely fashion.

10 Base Station

The original design for the base station called for a self-contained Linux system to collect and process the measurements from the other PICA systems. To accomplish this goal, the design team decided to use an FPGA development board and program it will the SPARC-compatible LEON3 soft-processor, which is licensed under the GNU General Public License (GPL). While this option showed a great deal of promise, it ultimately lead to more obstacles than the team decided could be overcome without detracting from the other areas of the project. In light of this decision, the role of the base station will be fulfilled by a standard desktop computer using custom software to perform the tasks that an embedded system would have performed.

10.1 Device Selection

One of the key factors in creating an embedded system is determining the processing power required. As most of the devices connected to the base station aggregate their own data and send only summaries to the base station, the base station does not require a lot of processing power. It does, however, need to process the data and display the information to its web interface. As the device does not generate a graphical user interface of its own, the majority of the processor usage will involve reading data from other devices, displaying the data, writing data to disk, and the overhead involved in switching between these tasks. In the current configuration, the E-meter sends measurements about twice a second. If no other devices are active, this represents a demand for handling only a few floating-point numbers with each report, an easy task for most processors, even if performed in hardware. Writing data to a disk and displaying a simple web page are also fairly low-demand functions. As such, the base station could probably perform all of its tasks with less than 100MHz for its processor clock.

While a modern desktop computer would provide more than enough computational ability, the design team focused on creating a single-purpose embedded Linux system that would perform the necessary tasks without competing with the processes a user might run on a home computer. Additionally, this smaller device could stay on to constantly gather data, typically requiring less power to remain active than a full-sized desktop computer would. In order design such a device, the design team required hardware that was flexible enough to allow extra devices such as XBee radios or large storage devices to attach to it, but would also have features like networking built-in.

10.1.1 Board Selection

In their search for suitable hardware, the design team discovered an FPGA development board that a previous senior design team had ordered for their own project. This development board, a Digilent ML-509 educational board, included a 9-pin RS232 serial port, a standard networking port, a compact-flash reader, and many configurable pins for attaching extra devices. In addition to these peripherals, the board featured a Xilinx Virtex 5 FPGA, which could be configured into a wide variety of different processors or systems for which a standard hardware description could be found. Upon finding that these features met the requirements for the base station, the design team selected the board for making a prototype of the base station.

While the Virtex 5 chip offered flexibility during development, including it in a production design would drive the price of the base station very high. (See `http://digilentinc.com/Products/Detail.cfm?Prod=XUPV5` for the price of an identical board.) As this flexibility is not needed for production editions of the base station, a traditional processor would replace the FPGA for both cost and simplicity.

10.1.2 Processor Selection

Having selected the development board, the design team looked at several different options for soft-processors. The website `http://opencores.com/projects` provides many open-source soft-processors from which to choose, such as the Sun OpenSPARC and the OpenRISC series. While many different options would be able to run the base station software, the team settled on the SPARC-compatible LEON3 processor, which is licensed under the GPL, as it seemed to provide a comprehensive package to include all of the development board's components. Additionally, the LEON3 vendor, Gaisler-Aeroflex, provided a host of cross-compilers and tools to interface with and debug the processor under free or demo licenses. They also advertise a Linux distribution containing a variety of utilities and programs. With all these tools available, the design team felt certain that the LEON3 would provide everything required of the base station.

10.2 Building the LEON3 Processor

In order to use the LEON3 soft-processor, the FPGA required a configuration file generated from the LEON3 hardware description files. As the FPGA was a Xilinx part, the design team downloaded ISE, the Xilinx software suite for creating part-specific configurations from the description files. Although Xilinx provides a "Webpack" edition of ISE free of charge, its functionality is very limited. In fact, the synthesis tools for the specific FPGA on the board, the XC5VLX110t, were explicitly disabled without a license for ISE. To work around this issue, the team requested a thirty-day evaluation license from Xilinx, which unlocked the required synthesis tools for the processor on the ML-509 board.

10.2.1 Configuring the Source Files

The LEON3 hardware description source files came with pre-made configurations for many different boards, including one for the ML-509. As such, this configuration could be used directly without requiring further changes, although the automated process took more than an hour to complete. When this task completed, it produced a bitfile that configured the specific model of FPGA to behave like the LEON3 processor. The design team later discovered that Gaisler had a functionally similar bitfile available for direct download.

10.2.2 Programming the FPGA

In order to use the Xilinx tool to program the FPGA, the workstation computer needed access to the development board's configuration. Xilinx provided a programmer pod, a Universal Serial Bus (USB) device to access the FPGA's configuration using the JTAG protocol, for exactly this purpose. Once fitted with the appropriate drivers, the workstation successfully detected the pod. The Xilinx program `impact` wrote the bitfile into the FPGA, transforming it into the LEON3 processor.

10.2.3 Managing the LEON3 Processor

Gaisler-Aeroflex provided an evaluation version of their `grmon` tool to connect to the LEON3 for loading and debugging code on it. As described in its manual, `grmon` could connect to the processor using the same JTAG pod that `impact` used to program the device. This claim held true, and `grmon` promptly gave a summary of the system it found there.

10.3 Extending the LEON3

While Gaisler-Aeroflex provided the core LEON3 and its interfaces to the development board's hardware under the GPL, they do not distribute their Floating Point Unit (FPU) under the same license. As such, on the default LEON3 system, any float-point operations would be emulated in software, rather than performed in hardware. As the intended purpose of the base station would likely involve handling floating-point numbers as measurements, including a hardware FPU would likely benefit the performance of the base station.

While the LEON3 configuration tools include a section for adding an FPU to the system, they did not include the description files for such a device. Instead, Gaisler provided its own FPUs as a separate download. By separating them from the GPL-licensed source, Gaisler could distribute the FPU in a form that could be included into the LEON3 system without exposing their source description files. Once these files were added into the project, the LEON3 configuration tool could include them into the bitfile creation process. The design team created and loaded the resulting bitfile onto the FPGA, and asked `grmon` to display information about the system. Its output is given in the appendix, listing 7.

10.4 Running Gaisler Linux on the LEON3 Processor

As previously mentioned, Gaisler-Aeroflex provided a customized Linux distribution for the LEON3, which is derived from the SnapGear embedded Linux distribution. They also provided a pre-built compiling tool chain based on GNU Compiler Collection (GCC). Using these two together should have produced a small Linux system for the LEON3. The critical components of the system, such as the Linux kernel and the LEON3 boot-loader, functioned without much extra tweaking. An example of the messages printed by the Linux kernel over a serial connection may be found in the appendix, listing 8.

Gaisler provided a configuration tool for their distribution that behaved very similarly to their previous configuration tools. This tool not only allowed for configuring the kernel, but also for creating an entire Linux system around it, complete with a shell and a wide range of utilities and applications. Unfortunately, very few of them worked without tinkering. Some options could not be built at all, others could be built with a sufficient amount of tweaking, but even then the configuration files for these programs did not install correctly into the distribution's miniature file-system. The design team tried multiple times to get the system working, but when the distribution's bundled DHCP client would not load due to library linking failures, the team decided to try a different approach. Message logs with these failures can be found in the

appendix, listings 9 and 10.

10.5 Building Linux from Scratch for the LEON3

In order to build a Linux system without dealing with Gaisler's distribution, the design team changed focus to building a Linux distribution from source code. The team referenced the guide at `http://cross-lfs.org/view/clfs-embedded/arm/`, and adapted it for the LEON3 SPARC system. While this process ran quite smoothly, it ultimately failed when trying to boot on the LEON3, where the system unexpectedly stopped at a break point. This problem persisted even when the debugging connection asserted that such breakpoints should be ignored, which may indicate a flaw in the LEON3 hardware. The design team's guide to making this distribution is given in the appendix, section E.

10.6 Software Monitor

Due to hardware difficulties, the design team decided that the time required to create a fully-functional LEON3-powered base station would be better spent on other areas of the project. In place of the hardware base station, the team created software to run on a Linux desktop computer that can monitor data from a serial port and display a graph of the data received. As the XBee radios can be treated as serial devices, this approach allows for either wired or wireless communications. The software does not include a method for saving information or for loading saved information, and it requires that the computer be on in order to read the data. While this does not satisfy all of the design targets for the base station, it does demonstrate that data can be passed between devices and does not require specific or unusual hardware.

10.6.1 Software Design

The monitoring software demonstrates that the Power Information Collection Architecture (PICA) systems output legible data using a pair of Perl scripts that feeds a stream of data between them. Combined, the two script read serial output from the E-Meter and plots the readings to a graph on the computer screen using Gnuplot, as shown in figure 37. The command-line arguments to the first script select which of the items found in the line should be passed. The arguments to the second script determine how many streams to read (typically one), how many points should be plotted at once, and the range of values to show.

The base-station software could realistically use any language that run on a Linux machine. As is

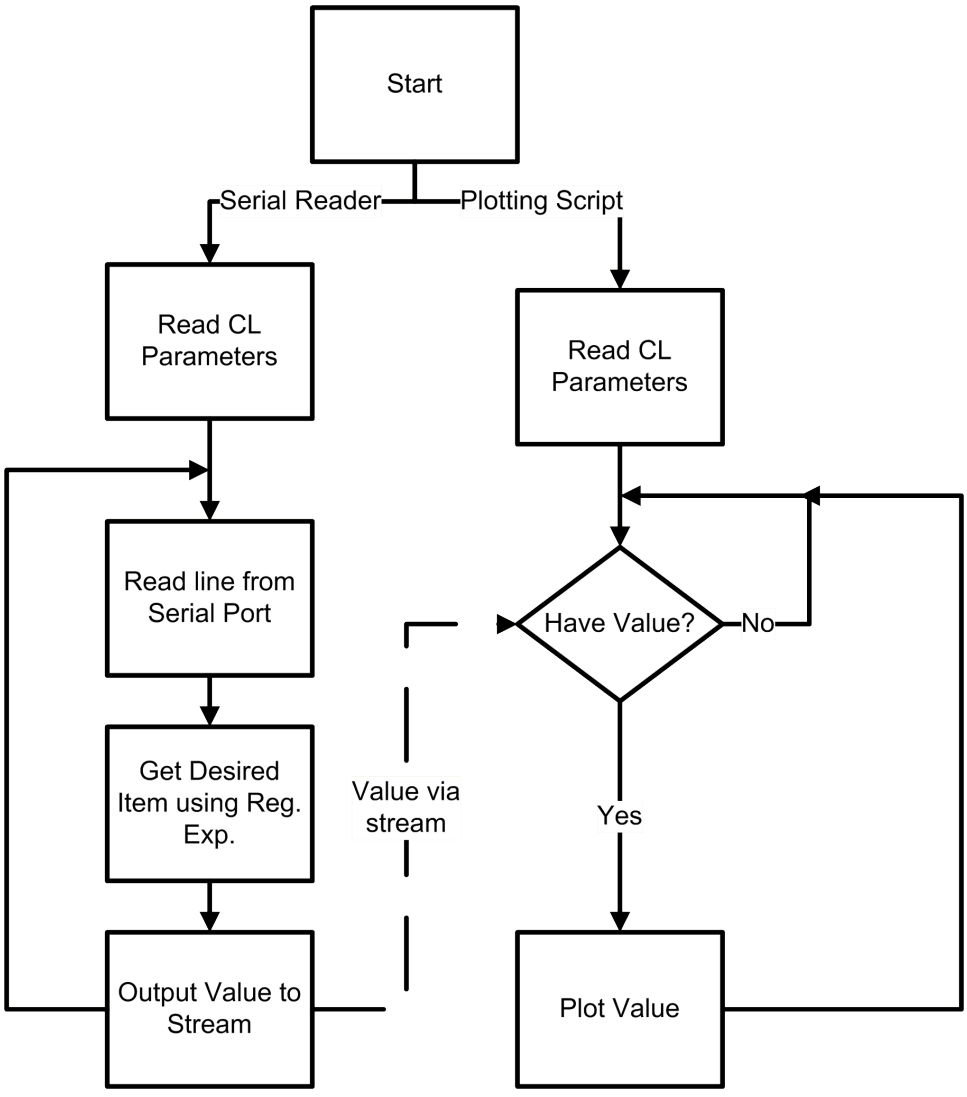


Figure 37: Flowchart for Perl Monitor Software

commonly known, compiled languages typically deliver higher performance than scripted languages. C, in particular, provides high speed and access to many low-level Linux functions. While Perl does not have quite the speed of compiled languages, it provides many packages to perform the desired functions. In particular, Perl can read whole lines of input from the serial port, use its built-in regular-expression support to select one of several numbers and pass it as a stream into another Perl script. Additionally, Perl scripts are portable, meaning that they can be transferred between different machines and even different platforms without being recompiled. The combined value of Perl's regular expressions and portability, as well as its familiarity to the software designer, made it a clear choice for the monitoring program. Gnuplot provides an automatic buffer of points, aesthetically

10.7 Final Design

Although the design team originally specified that a base station with dedicated hardware would provide the best fit for the design goals and requirements, producing a working prototype of the device proved to be too time-consuming and difficult to debug. In place of the single-purpose device, the design team created software to run on a Linux-based desktop computer that can interface with the other PICA devices.

10.8 Testing

In order to confirm that a hardware bases station conforms to this design, verify each of the following. These tests confirm the general behavior, as this should indicate that the device hardware is functioning properly.

1. When plugged in and turned on, the base station loads its software and indicates this externally, such as with status LEDs.
2. While active, inserting a network cable will cause the base station to connect to the network via DHCP. Network connectivity may be indicated on its own status LED.
3. While plugged into the network, the base station should display a web page when browsed by another computer on the network.
4. While active, the base station should connect with other PICA devices and gather their data. This should be reflected in the web interface.

As the hardware will be particular to the prototype, a generic hardware test for any base station is impractical until a prototype has been produced. In order to verify that the development board and LEON3 processor were functioning properly, the developer took the following steps:

1. Plug in the power supply and switch the board to be "on". If a change is observed, the power supply works.
2. Verify that the on-board Complex Programmable Logic Device (CPLD) is correctly programmed: the LEDs by the directional buttons should light. If not, obtain the programming image from Xilinx and flash it to the CPLD using `impact`. It may be located under the ML-505 board section.
3. Verify that the networking is functioning. If a networking cable is inserted into the ethernet port, several status LEDs along the front edge towards ethernet port should light or blink.
4. Verify that the Compact Flash (CF) port is correctly configured. Removing the CF card should cause some red error LEDs to light; inserting the card should clear them.
5. Program the FPGA using a bitfile. The status LEDs should turn off during the flashing process, then turn back on afterwards. This process typically turns on one of the red LEDs.
6. Close `impact` and start `grmon` using the Xilinx USB programmer pod. The red LED should clear, and the console should reply with a readout of all the elements of system.

Note that the FPGA heats significantly while programmed as a LEON3. A heatsink is not included, so caution must be exercised to avoid burns. The likelihood of heat damage to the processor is low, as it has been run for several hours at a time without measurable effects on performance. A production model must deal with heat, as it will be kept on for much longer than it would be for testing purposes.

10.9 Future Work

The difficulties with the hardware base station can likely be overcome with more dedicated time and effort. If this not the case, the LEON3 system may need to be abandoned in favor of another processor. This may require a complete restart of this subsystem design if the error lies with the LEON3 processor. If this is the case, much of this hardware prototype design must be revised. Concept drawings for the web interface may be seen in figures 38 and 39.

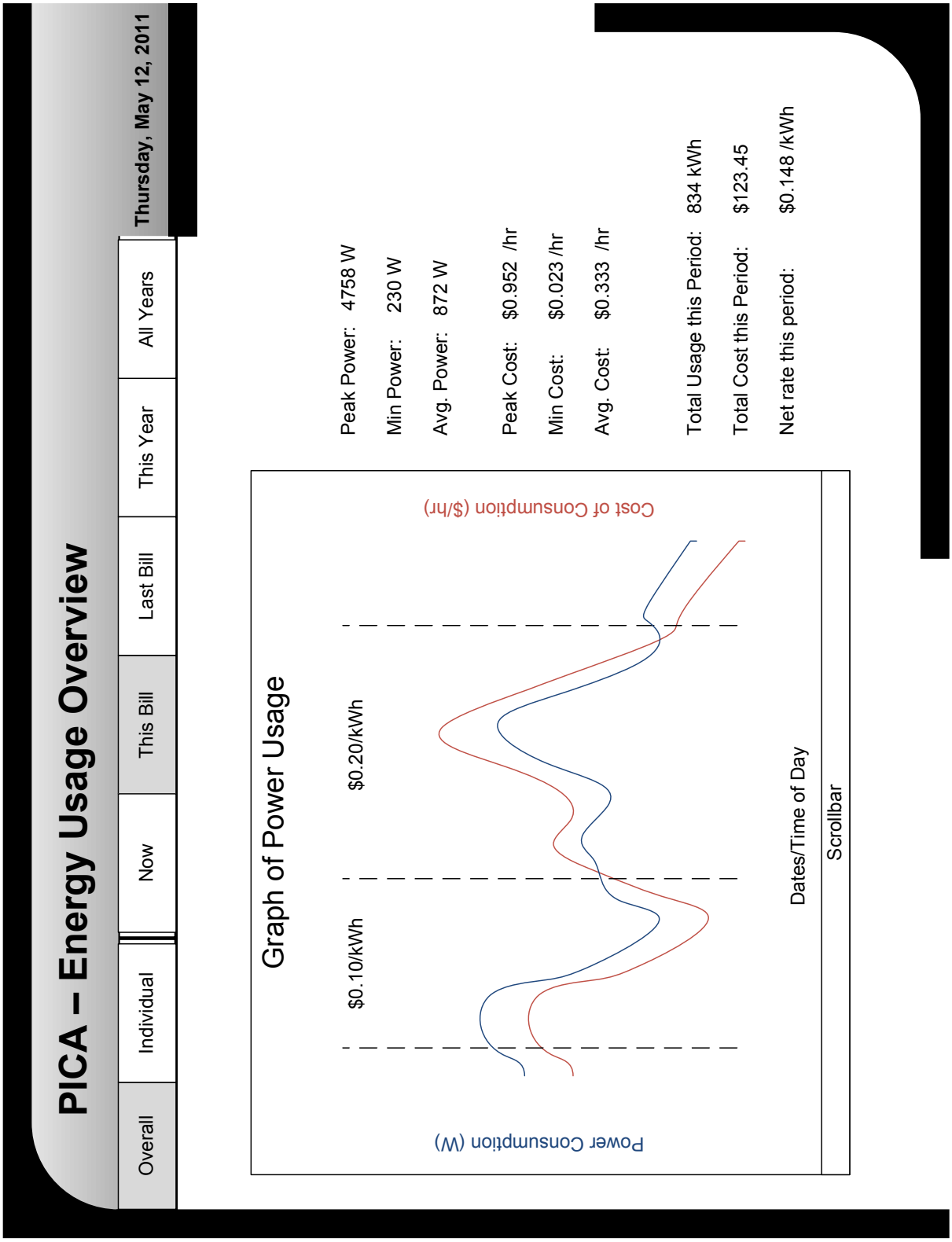


Figure 38: Conceptual Base-Station Interface: Power History

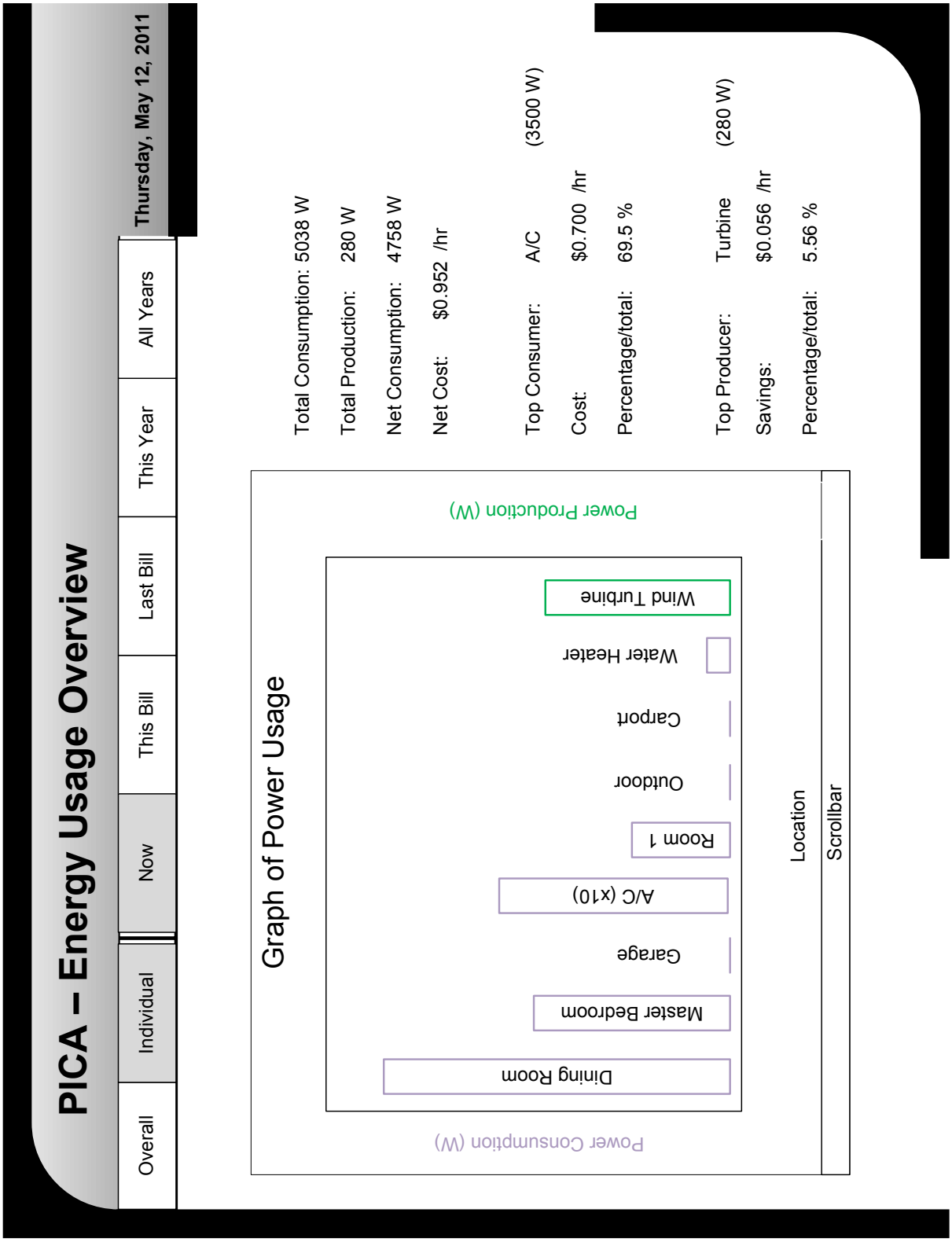


Figure 39: Conceptual Base-Station Interface: Localized Measurements

10.9.1 Software Monitor

The software monitor lacks several features intended for the original base station. Adding them to the Perl scripts would provide a good hold-over in the absence of a hardware base station. In fact, a hardware base station could use some components of the Perl scripts if Perl is installed on the base station.

1. Allow data to be backed up to a file.
2. Read data from a backup file and display information about it.
3. When supported, instruct connected systems to change their output.
4. Provide a way to restrain all devices from talking at once.
5. Distinguish devices from each other.
6. Support data encryption and decryption between devices.

11 Power Supply

11.1 Design Criteria

1. Size and weight: These could conflict however, if the size is no bigger than 4 cubic inches and all of the parts are surface mount, the heaviest part is the transformer. The power supply should not exceed 3 pounds if possible.
2. Output voltage: The output voltage will need to be sufficient for various loads. It will need to still remain at 5V
3. Efficiency: This is very important. Linear power supplies are not very efficient, which is why the team decided to build a switched-mode power supply. They are generally 30% more efficient than linear power supplies.
4. Power Dissipation: Power dissipation shall be low.
5. Electronic noise at input and outputs: The electronic noise at input and outputs shall not get to the point where compensation for noise is needed.

11.2 Alternatives

11.2.1 Buck converter

“The input voltage is converted into a lower output voltage.”[22]

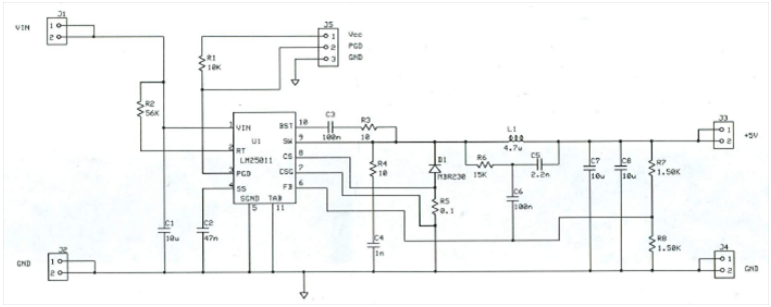


Figure 40: Buck Converter

11.2.2 A boost converter

“The input voltage is converted into a higher output voltage”[22]

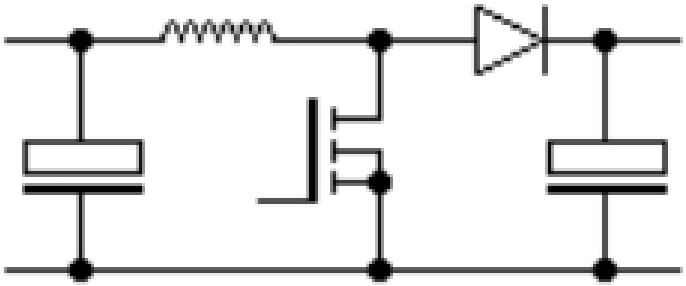


Figure 41: Boost Converter

11.2.3 A buck-boost converter

“The input voltage is converted into a negative voltage”[22]

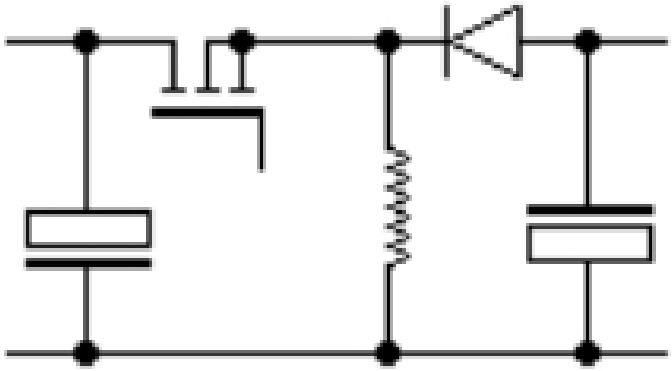


Figure 42: Buck-Boost Converter

11.2.4 A flyback converter

“Several isolated output voltages, up to approximately 250 are possible”[22]

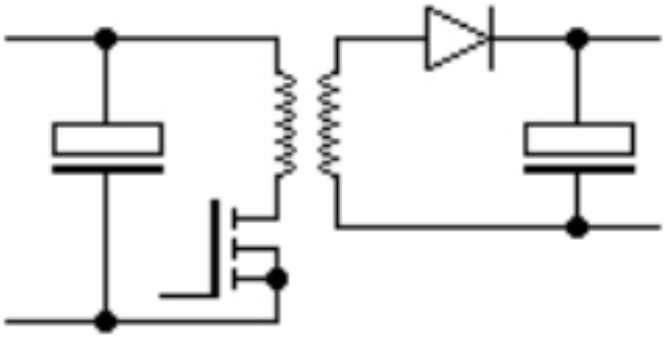


Figure 43: Flyback Converter

11.2.5 Single Transistor Forward Converter

“One electrically isolated voltage, up to approximately 100 Watts.”[22]

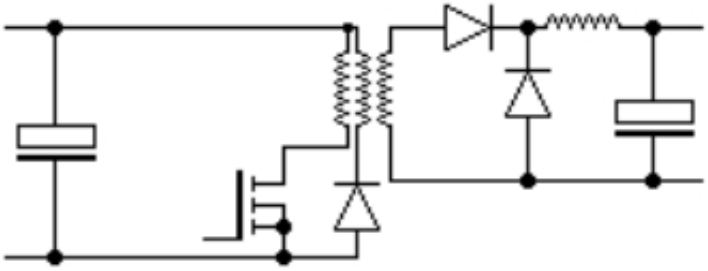


Figure 44: Single Transistor Forward Converter

11.2.6 Two Transistor Forward Converter

“One electrically isolated voltage, up to approximately 1kW.”[22]

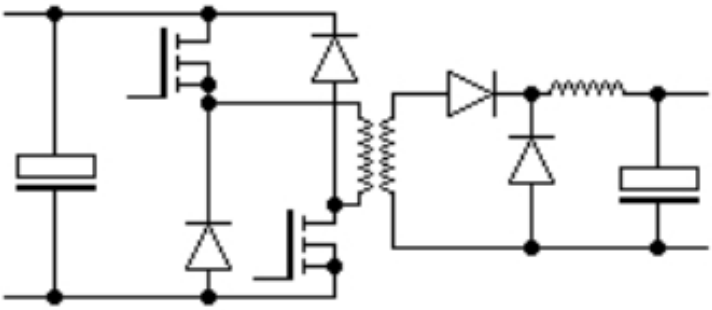


Figure 45: Two-Transistor Forward Converter

11.2.7 Half-Bridge Push-Pull Converter

“One electrically isolated voltage, up to few kilo watts.”[22]

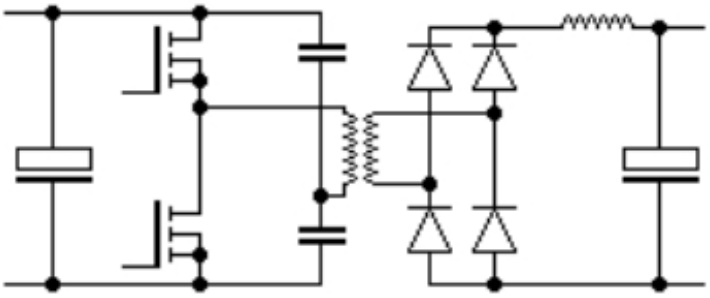


Figure 46: Half-Bridge Push-Pull Converter

11.2.8 Full-Bridge Push-Pull Converter

“One electrically isolated voltage, up to many kilo watts.”[22]

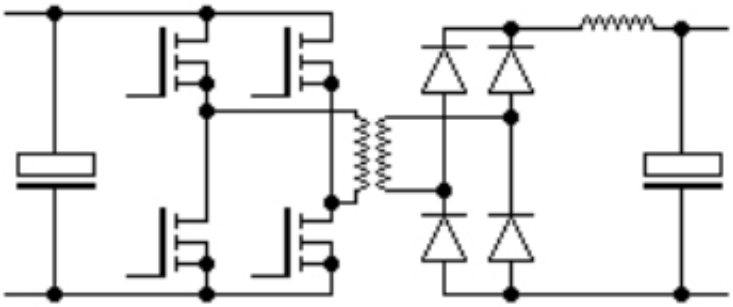


Figure 47: Full-Bridge Push-Pull Converter

11.3 Rating the Alternatives

Once the team decided on a switched-mode supply, we needed to select a topology to use. The choices include a buck converter, a boost converter, a buck-boost converter, a flyback converter, single- and two-transistor forward converters, and half- and full-bridge push-pull converters. Some of these topologies can be eliminated immediately because the design will be too complex than needed to meet the needs of this design. For example, a boost converter takes the input voltage and converts it to a higher output voltage, which does not fit line voltage being converted to 5V DC. Similarly, a buck-boost converter outputs a negative voltage, which is also not fitting for this application. The single-transistor forward converter, two-transistor forward converter, half-bridge push-pull converter, and full-bridge push-pull converter have one electrically isolated voltage up to a varying number of watts. This would work for our design, however they are more complex than needed, other designs will work and be less complex. After these eliminations, the two remaining topologies are the flyback and the buck converters. The buck takes an input DC voltage and converts it into a lower output DC voltage, while a flyback can produce several isolated output voltages from an input AC voltage. Using a transformer and rectifier bridge can adapt an AC input into a DC voltage, allowing the buck topology to work where the flyback topology could also be used. Either design is feasible for the purpose specified. The flyback would use an inductor, while the buck uses a transformer. The team has used transformers for these purposes before, which would make the design easier. A flyback also can have multiple isolated voltages, but this was not needed in the design, so the team did not feel like this design was best. The team decided on a Buck Converter. Additionally, Johnson Controls Incorporated graciously donated parts for a buck converter, as well as expert advice on designing and assembling one. As none of the members of the team have made a SMPS before, it felt very nice that the team could have an expert to go to for advice on a chip and design he has worked with several times.

11.4 Implementation

11.4.1 Scope

This section describes the analysis of the 5V/2A, LM25011-Buck Converter.

11.4.2 Module Overview and Block Diagram

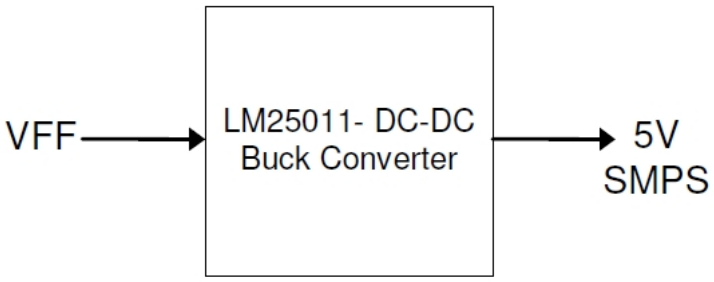


Figure 48: Buck Conversion Use-Case Diagram

11.4.3 Schematics

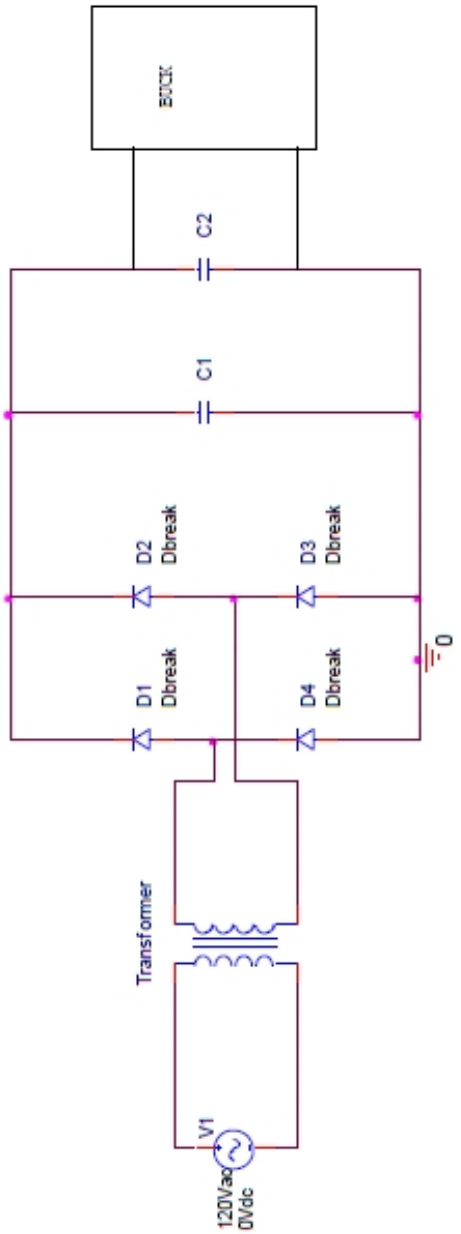


Figure 49: Schematic of AC to DC Conversion

11.4.4 Calculations

This power supply will require a transformer and four diodes as shown in the first schematic. These equations are for that part of the schematic.

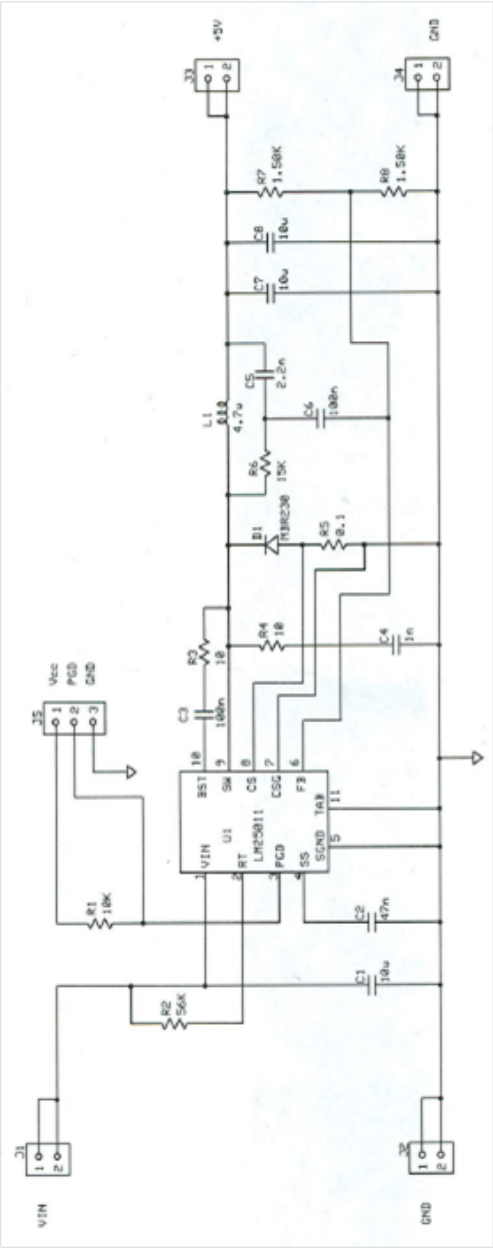


Figure 50: Schematic of Buck Converter DC to DC

$$V_{AC} = 18\text{V} \tag{10}$$

This is the voltage the transformer will output from a 120 V_{AC} outlet. The buck converter needed at least 6V, but in order to be safe a transformer was chosen that was higher than this value. the buck converter could take up to 42V, so it was also safe from the maximum value.

$$Transformer_{peak} = VAC\sqrt{2} \tag{11}$$

Calculating for the rectifier circuit. This comes from the equation for finding V_{rms}, which is

$$V_{rms} = \frac{V_p}{\sqrt{2}} \tag{12}$$

$Current = 2$ This is the desired output current. So, this is what I will equal in the following few equations.

$$dt = \frac{1}{120}\text{s} \tag{13}$$

is the duration between charging cycles and is equal to

$$\frac{1}{2 \times Main\ \ Frequency} \tag{14}$$

Therefore since in America the main frequency is 60Hz,

$$dt = \frac{1}{120} \tag{15}$$

$$Overhead = 1.4 \tag{16}$$

$$dv = Transformer_{peak} - Overhead - 8 = 16.056 \tag{17}$$

We need this 8V for the switching regulator, it needs 3V of headroom and since we want 5V out we take the voltage +3V and get 8V, but for a little bit of watermark/overhead.

$$Capacitor = Current\frac{dt}{dv} \tag{18}$$

$$Capacitor = .001\text{F} \tag{19}$$

This simply means that the capacitor(s) for filtering need to be at least 1mF.

Note:This is a minimum value This value could be achieved by one capacitor or multiple as long as the total capacitance is at least 1mF.

The diodes were selected based on the fact that Chuck Howerda gave them to our team and said they would work well for this application and for the diode bridge.

11.4.5 Capability Justification Table

Table 5: Power Supply Capabilities

Capability Parameter	Specification	Analysis Result	Remarks
Input Voltage	Min	7.09V	From Current Draw
Input Voltage	Max	17.10V	From Current Draw
Switching Frequency	Max	3.38MHz	From Current Draw
Output Voltage	Min	4.99V	From Calculation
Output Voltage	Max	5.19V	From Calculation
Output Ripple Voltage	Max	10mV	Assumed
Output Current	Max	2A	From Current Draw
Output Inductor	Typ	2.2µH	From Schematic
Power Dissipation across Inductor	Max	99mW	From Calculation
Power Dissipation across current sense resistor during current limit mode	Max	246mW	From Calculation
Efficiency	Max	86.86%	From Calculation
Softstart Time	Min	8.032ms	From Calculation
Softstart Time	Max	12.048ms	From Calculation
Input Under Voltage Protection Threshold	Min	4.6V	From Datasheet
Thermal Shutdown	Typ	155°C	From Datasheet

Continued on next page

Table 5: Continued from previous page

Capability Parameter	Specification	Analysis Result	Remarks
Line Regulation	Max	5 +/- 5%	Verify by Testing
Load Regulation	Max	5 +/- 5%	Verify by Testing
Gain Margin			Verify by Testing
Phase Margin			Verify by Testing

This table was co-produced with Hilbrand Sybesma, a hardware engineer at JCI. After we were both done working on January 27, 2011, he took the time to work with me on them. As I was creating it, he stopped me if I made any errors and he told me other things to include. He went above and beyond with help.

11.4.6 Theory of Operation

"The LM25011 Constant On-time Step-Down switching regulator is capable of supplying up to 2A of load current. This high voltage regulator contains an N-Channel Buck switch, a startup regulator, current limit detection,and internal ripple control. The constant on-time regulation principle requires no loop compensation, results in fast load transient response, and simplifies circuit implementation.

The operating frequency remains relatively constant with line and load. The adjustable valley current limit detection results in a smooth transition from constant voltage to constant current mode, when current limit is reached, without the use of current limit foldback. The PGD output indicates the output voltage has increased to within 5% of the expected regulation value.

Additional features include: Low output ripple, VIN under-voltage lock-out, adjustable soft-start timing, thermal shutdown, gate drive pre-charge, gate drive under-voltage lock-out, and maximum duty cycle limit."[23]

11.4.7 Design Assumptions

VFF value is considered as min. 7.09V and max. 17.10V from Current draw of the chip in use.

11.4.8 Analysis

1. Input voltage range:

$$P_{VFF} = 7.09, 12.3, 17.1\text{V} \tag{20}$$

   (a) Input Voltage range for the LM25011 is 6V to 42V 12.3V was choosen for a medium between 7.09 and 17.1.

2. Output Voltage

$$P_{5V} = 4.75, 5, 5.25\text{V} \tag{21}$$

   (a) These values are from the 5%tolerance.

3. Load Current

$$I_{out} = 2\text{A} \tag{22}$$

   (a) Note: As per datasheet, LM25011 can deliver up to 2Aload current [23].

11.4.9 Output Voltage Calculation

Calculations from datasheet:

$$V_{ref} := \begin{pmatrix} 2.46 \\ 2.51 \\ 2.56 \end{pmatrix} \cdot V \qquad R_{fb1} := \begin{pmatrix} 1 - 1\% \\ 1 \\ 1 + 1\% \end{pmatrix} \cdot 1.43 \cdot K\Omega$$

$$R_{fb2} := \begin{pmatrix} 1 - 1\% \\ 1 \\ 1 + 1\% \end{pmatrix} \cdot 1.47 \cdot K\Omega$$

$$V_{out} := \overrightarrow{\left(V_{ref} \cdot \frac{R_{fb1} + R_{fb2}}{R_{fb1}} \right)} \qquad V_{out} = \begin{pmatrix} 4.989 \\ 5.090 \\ 5.192 \end{pmatrix} V$$

$$V_{out\_nom} := V_{out_1} \qquad V_{out\_nom} = 5.090\,V$$

Figure 51: Calculation from datasheet

11.4.10 Selection of Switching Frequency

Information from datasheet:

The maximum allowed frequency may be limited by the minimum and maximum input voltage, as described below:

1. Operation at a low input voltage requires a high duty cycle to maintain output regulation. The maximum duty cycle is limited by the minimum off-time of the LM25011 (150ns typ., 208 ns max). The output voltage drops out of regulation at low VIN if the switching frequency does not satisfy the following condition:

$$Fsw < (Vin(min) - Vout)/(Vin(min)x208ns) \tag{23}$$

2. Operation at high input voltage requires a low duty cycle, limited by the minimum on-time of the LM25011 (90 ns). The switching frequency must satisfy the following condition to ensure the on-time is greater than 90 ns:

$$Fsw < Vout/(Vin(max)x90ns) \tag{24}$$

If the selected value of Fsw does not satisfy this condition, the LM25011 maintains regulation, but the switching frequency will be lower than the desired value.

$$f_{s\_max\_con1} := \frac{\min(P\_VFF) - V_{out\_nom}}{\min(P\_VFF)\cdot 208\cdot ns} \qquad f_{s\_max\_con1} = 1.356\cdot MHz$$

$$f_{s\_max\_con2} := \frac{V_{out\_nom}}{\max(P\_VFF)\cdot 90\cdot ns} \qquad f_{s\_max\_con2} = 3.307\cdot MHz$$

Figure 52: Condition from datasheet

From the conditions shown, maximum switching frequency should not exceed 1.356MHz.

The approximate operating frequency is calculated as follows:

$$RT_{used} = \begin{pmatrix} 1-1\% \\ 1 \\ 1+1\% \end{pmatrix} \tag{25}$$

$$V_in = P_{VFF} \tag{26}$$

$$f_s = \frac{V_{out->nominal}}{(V_{in}15ns) + [4.1 * 10^{-11}(RT_{used} + .5\text{k}\Omega)\text{As}]} \tag{27}$$

$$f_s = \begin{pmatrix} 2.121 \\ 2.035 \\ 1.961 \end{pmatrix} \tag{28}$$

$$f_{sMAX} = 2\text{MHz} \quad \text{From Datasheet} \tag{29}$$

For the selected RT value, switching frequency is higher than 2MHz. However after speaking with the contact from National Semiconductor I have confirmed that LM25011 can operate up to 2.4MHz.

11.4.11 Selection of Inductor

As per JCI recommendations as well as National Semiconductor, we opted to use a 15µHinductor since they said it is the best to use because we will want more than the minimum requirement.

11.4.12 Ripple Current

1. Maximum Ripple Current

$$I_{ripple\_max} = (max(V_{in}) - min(V_{out}))(\frac{min(T_{on\_min})}{min(Lrecom)}) \tag{30}$$

   (a) Note: This is acceptable.

$$I_{ripple\_max} = 294.851\text{mA} \tag{31}$$

2. Nominal Ripple Current

$$I_{ripple\_nom} = (V_{in_1} - V_{out})(\frac{min(T_{on\_min})}{min(Lrecom)}) \tag{32}$$

$$I_{ripple\_nom} = 175.525\text{mA} \tag{33}$$

3. Minimum Ripple Current

$$I_{ripple\_min} = (min(V_{in}) - max(V_{out}))(\frac{min(T_{on\_min})}{min(Lrecom)}) \quad (34)$$

$$I_{ripple\_min} = 95.321\text{mA} \quad (35)$$

4. Inductor Peak Current

$$I_{peak} = I_{out\_max} + \frac{I_{ripple\_max}}{2} \quad (36)$$

$$I_{peak} = 2.147\text{A} \quad (37)$$

11.4.13 Minimum Valley Threshold for Current Limit

$$I_{valley\_th} = I_{out\_max} - \frac{I_{ripple\_min}}{2} \quad (38)$$

11.4.14 Current Sense Resistor

As long as we have a current sense resistor that does not exceed RS_{max} (located in the datasheet) it will be okay. The more current we want in our power supply the smaller our current sense resistor will need to be. Which we can vary as we see fit. The proportion to which this varies depends on the ripple voltage and the calculation for which is shown below.

11.4.15 Ripple Voltage

$$RS_{ripple\_min} = min(RS_{used})min(I_{ripple\_min}) \quad (39)$$

$$RS_{ripple\_min} = 10.381\text{mV} \quad (40)$$

As per datasheet, minimum ripple voltage across current sense resistor should be greater than 10mV. In this case it is very close, but that should not be a problem.

11.4.16 Power Dissipation

We set the current sense resistor to get 1.5A out. Also, by the results of the testing we get 1.3A out instead of 1.5A. We still get 5V out. Therefore we can calculate the efficiency.

$$Eff = \frac{P_{out}}{P_{in}} \tag{41}$$

$$Eff = \frac{1.3 \times 5}{1.5 \times 5} \tag{42}$$

$$Eff = 88.7\% +/- 5\% \tag{43}$$

So, therefore $Eff = 84.23\%,\ 88.7\%,\ 93.1\%$

11.4.17 Efficiency

No adequate information is provided in the datasheet about power loss in LM25011, However to identify approximate power loss in LM25011, the Efficiency of the LM25011 buck converter is simulated using National Semiconductor Webench tool for the same specification.[24]

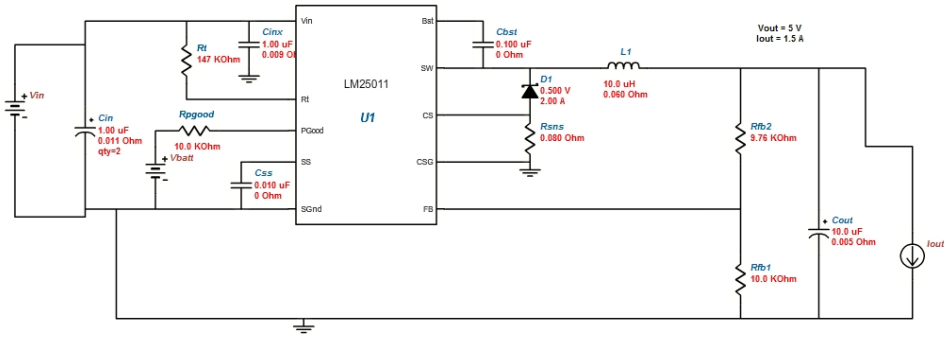


Figure 53: Circuit as shown in Webench

11.4.18 Total Output Power

$$P_{out} = max(V_{out})I_{out\_max} \tag{44}$$

$$P_{out} = 10.383\text{W} \tag{45}$$

$$P_{in} = \frac{P_{out}}{Eff} \tag{46}$$

$$P_{in} = 11.955\text{W} \tag{47}$$

Figure 54: LM25011 Efficiency graph from Webench tool $Eff = 86.85\%$

11.4.19 Approximate power loss in LM25011

$$PD_{LM25011} = P_{in} - PD_{ext} - P_{out} \tag{48}$$

$$PD_{LM25011} = 223.948\text{mW} \tag{49}$$

11.5 Testing

First I tested just the power supply with no load to make sure it gave a 5V output, which was measured with a DMM. That was a success! After which, I put a load of 2.5Ω to see how much current went through. Those values and test setup are shown in the pictures as follows:

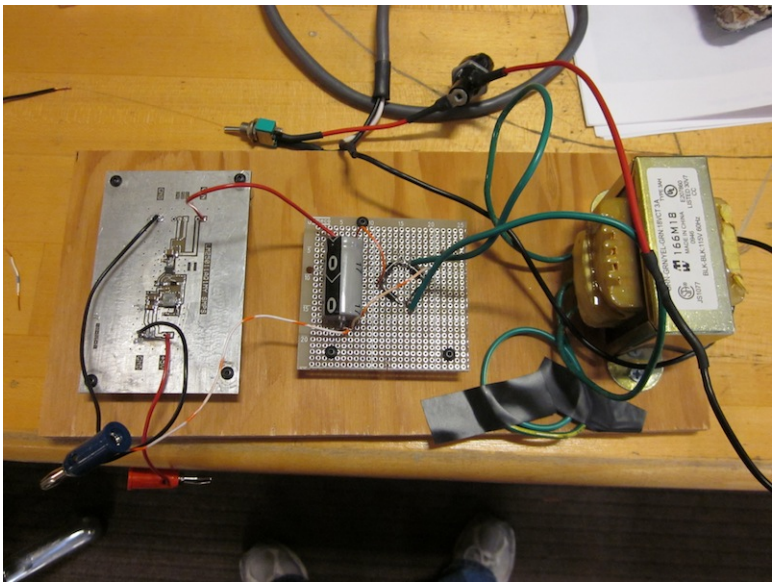


Figure 55: Full Assembly

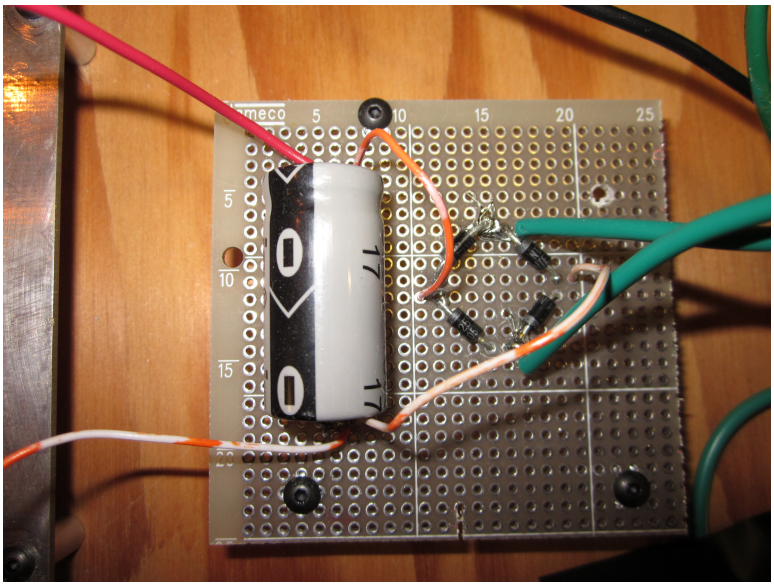


Figure 56: Rectifier Circuit

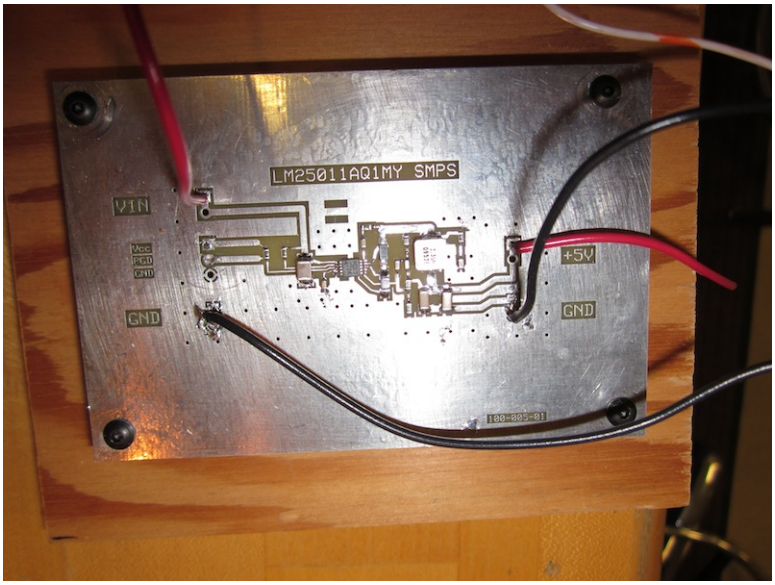


Figure 57: Buck Converter

12 Business Case

12.1 Industry Profile

12.1.1 Overview of the Problem

Standard electric meters were developed decades ago and are still used today, despite many technological advances in the last several years. Along with these technological advances, Americans have become accustomed to having access to large amounts of data, but due to the nature of the standard electric meter, data regarding the usage of power is severely limited. For the power companies, data from the meters is minimal and grid control is limited to manual operation, costing them time and money. As the cost of electricity becomes higher and higher, electricity use in buildings is becoming a bigger concern and people have few cheap or simple ways to monitor this. Of the options available, most only address part of the whole problem, giving some information to the consumer and none to the power company or vice-versa. While there are devices such as breakers and fuses that provide electrical safety for buildings, advances in technology have made it possible to further improve safety but have not been implemented in a cost-effective way or made easily available to an average consumer, which for the purpose of this project shall be defined as a person without a mathematical or scientific education beyond high-school.

12.1.2 Major Customer Groups

The two main customers of the PICA system are power companies and power consumers. The E-meter subsystem will only be sold to power companies, as they must in turn provide metering equipement to their customers. The base station and solid-state breakers will be sold to power consumers who are interested in knowing how much power they use in different regions of their buildings.

12.1.3 Regulatory Requirements

The PICA system must meet certain codes in order to be safe enough for the customer to use, which will also protect from unexpected lawsuits. Underwriters Laboratories (UL) is an independent product safety certification organization, which offers safety certifications to products [25]. In order to gain the confidence of customers, the devices of the PICA system will be UL certifiable. The specific qualifications of UL certification remain unknown to the design team, as the documents regarding the certification requirements

are not publicly available. The system will also restrict EM radiation to comply with FCC Title 47 Part 15. It will also comply with ANSI C12.19 and ANSI C12.21 standards.

While these standards should ensure the general safety of the PICA devices, defects or unforeseen circumstances could imperil users or their property. The PICA system will provide a limited warranty against defects, but cannot be expected to foresee all possible circumstances. To this end, the devices will ship and work with a disclaimer regarding safe operating conditions and the hazards of tampering with the device.

In addition to ensuring the physical safety of the users, the system should also ensure the privacy and security of the users' information. While any wireless link runs the risk of packet interception and capture by a malicious observer, this will only affect the data currently being transferred, and data encryption schemes may greatly hinder these intrusions. The stored data will likely not be encrypted, but will not be actively transmitted: the only means of accessing this data will be through the software controls set in place by the base station or by physically removing the storage medium and removing the data from it. The base station software will use permissions-based file system access and will require a user to authenticate as an administrator before accessing this information. In this way, the user's data will be stored with access controls and will be kept private.

12.1.4 Significant Trends and Growth Rate

Recently, the demand for "smarter" and more informative devices has been increasing with the awareness of resource stewardship and the effects of human activity on the environment. Currently, power companies are investigating smart meters and deploying them to their customers in pilot programs. Power consumers are becoming more energy- and economically-aware, so the time is ripe for providing the products that PICA, LLC. proposes.

12.1.5 Barriers to Entry and Exit

The major barriers to entry is customer recognition. Although power companies may still be testing smart meters before deploying them to their customers, introducing them to a new product might be difficult if they are already near to making a decision. Additionally, power consumers will not be able to buy from PICA, LLC. unless they specifically know of it already.

12.2 Business Strategy

12.2.1 Desired image and position in market

PICA seeks to be known as a leader in making consumers energy aware. This primarily means providing relevant information in an easy-to-understand format that allows users to make informed decisions about their energy usage. Our position in the market will likely be a part of the "green" market, which has been rapidly growing. However, we would like to aim for the consumer oriented part of this market, rather than reaching out to the part of the market dealing with clean power generation.

12.2.2 SWOT analysis

Strengths

One big strength of the product is that in a single system it addresses problems that previously would require multiple, independent systems and products. Another strength is the solid state breakers, which in addition to providing usage information also have a faster response time and include a safe automatic reset option.

Weaknesses

The biggest weakness is the cost of the system. While users want to know the information provided, many are not willing to pay a lot for that information.

Opportunities

The idea of solid state breakers for home usage is relatively new and has not been taken advantage of, so there is a lot of potential growth in that area.

Threats

A big threat is that other companies have already started to establish themselves in similar fields. The PICA system may not provide enough of an advantage over these systems to be successful in an area where consumers have started to use these other systems.

Competitive strategy

As there is a high cost associated with the PICA products, the company will rely less on cost to remain competitive than on the differentiation and focus of the company. The company has already started to differentiate itself primarily through the development of the solid state breakers. The company is also different from many in how we provide solutions to more than one problem the consumer and power company may have. The company hopes to remain competitive by continuing to focus on more aspects of the problems associated with power usage than potential competitors do. Instead of providing a solution relating only to single outlet usage, PICA will provide devices that can all work together, leaving room for easy expansion when the customer decides they want even more information.

12.3 Competitor Analysis

This section analyzes a few products with similar features to various components of the PICA system such as Kill-A-Watt, Cent-a-Meter, The Energy Dectective, and Watts Up? Smart Circuit. The table below gives a overview of these products compared to the PICA system.

12.3.1 Kill-A-Watt

Around the turn of the millennium P3 International introduced the Kill-A-Watt device, which they marketed as a "user-friendly power meter that enables people to calculate the cost to use their home appliances," [26]. According to Amazon.com these devices range in price from $52 to 99 Manufacturer Suggested Retail Price (MSRP) depending on features, most notably how many devices can be monitored simultaneously. P3 produces three models of the Kill-A-Watt devices:

1. Kill-A-Watt PS (P4320): A power strip capable of monitoring voltage, line frequency, amperage, KWH, and current leakage for up to eight devices simultaneously and includes built-in surge protection [27].
2. Kill-A-Watt (P4400): The original Kill-A-Watt device, capable of monitoring voltage, amperage, watts used, line frequency, KWH, uptime, power factor, and reactive power for 1 device [28].
3. Kill-A-Watt EZ (P4460): This device is functionally identical to the P4400 series except that it includes one extra feature, it can calculate how much a device costs the consumer, after being programmed with the $/KWH provided by the power company [29].

Table 6: Comparison of PICA Competitors

Product	Monitoring features	Control Features	Cost (Fixed)	Cost (Recurring)	PICA Competitor
Kill-A-Watt	voltage, line frequency, amperage, KWH, and current leakage	N/A	$52 to $99	N/A	Ciruit-by-Circuit Monitors
Cent-A-Meter	Cost of electricity, temperature, humidity, kW of demand kg/hour greenhouse gas emissions	N/A	$140	N/A	E-Panel Meter
The Energy Detective	kW load, $/hour	N/A	$119.95 to $455.80	N/A	E-Panel Meter
Watts Up? Smart Circuit	Current, Voltage, kilowatts used	Remote On/Off	$194.95 /circuit	Free to $50.00 /month	Circuit-by-Circuit monitors
Smart-Watt	voltage, line frequency, amperage, KWH, current leakage, circuit load, on/off cycles	N/A	$169 to $249	N/A	Circuit-by-Circuit Monitors

All of the Kill-A-Watt devices claim to be accurate to within 0.2% of the actual power the monitored device uses [27][28][29]. The Kill-A-Watt devices cannot replace a power meter, but simply provide a method of supplying a consumer with additional data about their power consumption.

12.3.2 Cent-a-Meter

The Australian company, Clipsal produces the Cent-a-meter also known as the Electrisave or the Owl in the UK. Clipsal only produces one version of the Cent-a-meter which displays the cost of the electricity used in the home along with the temperature and humidity [30]. The device can also measure kW of demand, and kg/hour of greenhouse gas emissions [31]. Unlike the Kill-A-Watt, the centimeter does not accumulate any data, just displays instantaneous data on a receiver unit mounted in the home [30]. Clipsal does not list an MSRP for the Cent-a-meter, however SmartHome USA sells Cent-a-meter devices for $140 [31].

12.3.3 The Energy Detective (TED)

Energy Inc., a division of 3M, recently introduced its TED (The Energy Detective) power monitor. Functionally, TED operates exactly as the meter on the exterior of a consumer's home or business but the display resides indoors in a more convenient viewing location. Energy Inc. currently produces two series of the TED device:

1. TED1000 series: The TED1000 devices monitor current energy consumption in killowatts, and current energy cost in $/hour, and log this data for 13 months to predict energy use for the current billing cycle. TED1000 devices can integrate with a proprietary software package, Footprints, provided by Energy Inc. to visually display usage data [32]. TED1000 series devices range in price from $119.95 to $229.95 depending on the amp-rating of the service installation [33].
2. TED5000 series: The TED5000 sought to improve upon the TED1000 series by extending the functionality of the TED devices. The largest selling point for the TED5000 is integration with the Google Power service to track power usage data on the web [34]. TED5000 series units range in price from $239.95 to $455.80 depending on from how many measurement units the device gathers data [35].

12.3.4 Watts Up?

In 1997 Electronic Educational Devices Inc. introduced the Watts Up? product line to the education market. The product immediately became a hit, and soon utility companies across the United States began to take notice [36]. EED markets the Smart Circuit devices as a replacement for traditional circuit breaker devices for 100V to 250V, 20 amp 50/60Hz circuits [37]. Each Smart Circuit contains a built in web-server that allows for aggregation of collected data at a maximum rate of once per second [37]. These Smart Circuits are typically installed into a standard panel enclosure box, similar to standard circuit breakers, mounting directly to the industry-standard DIN rail inside the enclosure[37]. Alternatively, if needed at one local outlet, the Smart Circuit can be housed in a standard double gang electrical box [37]. Each Smart Circuit device can turn itself on or off when it receives a certain remote-control signal or when it detects one of many programmable stimuli. This self-waking feature makes these devices ideal for home-automation projects [37].

A single Smart Circuit, capable of controlling one circuit, costs $194.95, with enclosures for one, five, or ten Smart Circuits devices going for $325.95, $1495.95, and $2495.95 respectively [37]. A basic account, to view aggregated data and control the devices is free for residential use, but data rates, historical data and devices rules are limited [38]. A top-tier account, featuring the fastest update time, 1 second, up to 25 meters, 1 year of archival data, and 25 rules costs $50.00 a month [38].

12.3.5 Smart-Watt

The Smart-Watt device from Smartworks Inc. takes a similar approach to the Kill-A-Watt device in metering a single device at a time, but monitors much more information including circuit load over any period of time, and number of on/off cycles the attached device undergoes [39]. The biggest advantage to the Smart-Watt devices comes from the proprietary network Smartworks has developed for their devices. Each device attaches to a local network where a central server collects and collates all the data [39]. The Smart-Watt comes in two versions, one for International Electrotechnical Commision (IEC) plugs and receptacles and one for National Electrical Manufacturers Association (NEMA) plugs and receptacles. Both devices are similarly priced ranging from $169 to $249 depending on the current rating [39].

12.3.6 Standard Power Meter

Most homes or businesses attached to the electric grid are metered using a standard analogue power meter. This device provided by the power company, measures the amount of electrical energy consumed over a period of time. Typically, a power meter records in billing units, such as KWH. Each meter requires periodic readings based on the billing cycle of the power company; it is safe to assume that meters are read approximately once per month. In order to read the meter, an employee of the power company will physically go out to the meter and record usage data.

12.3.7 Nonintrusive Appliance Load Monitor

All of the products discussed here use a technique known as Non-intrusive Load Monitoring (NILM) to monitor power consumption without affecting the load on the circuit [40]. However, some more sophisticated products in this area use NILM to estimate the number of individual loads on the circuit [41]. If the research in this field proves that NILM provides accurate and useful data, devices based on the NILM technology would have a large advantage over other single-device power monitors, as such a device could be inserted into the feeder lines from the utility company and monitor all devices in the entire installation from a single point. Research turned up no significant products that claim to be capable of monitoring multiple loads on a circuit from a single point on the circuit. Thus this section is included to provide information, but does not represent a viable competitor in the market just yet.

12.3.8 PICA Competitors Comparison

In order to better understand the competition in the marketplace table 6 reproduces the information laid out above as a comparative table. The column on the far right side describes the PICA component that most directly competes with the product listed in the left column.

12.4 Ten-Year Financial Forecast

To assist in evaluating the financial viability of the product, the following graph and data illustrate predictions of sales in the first ten years of the product's life. In the first several years, the sales volume will grow as consumer recognition and demand increase. In the fifth year, rival companies will likely produce a competitive product; without any further product improvement, PICA will experience a shrinking market

demand, but continue to make a profit through the end of the decade. If, however, PICA develops a new product, the long-term trend may become greater growth instead of loss. These predictions assume a per-unit cost of $337 and a sale price of $400.

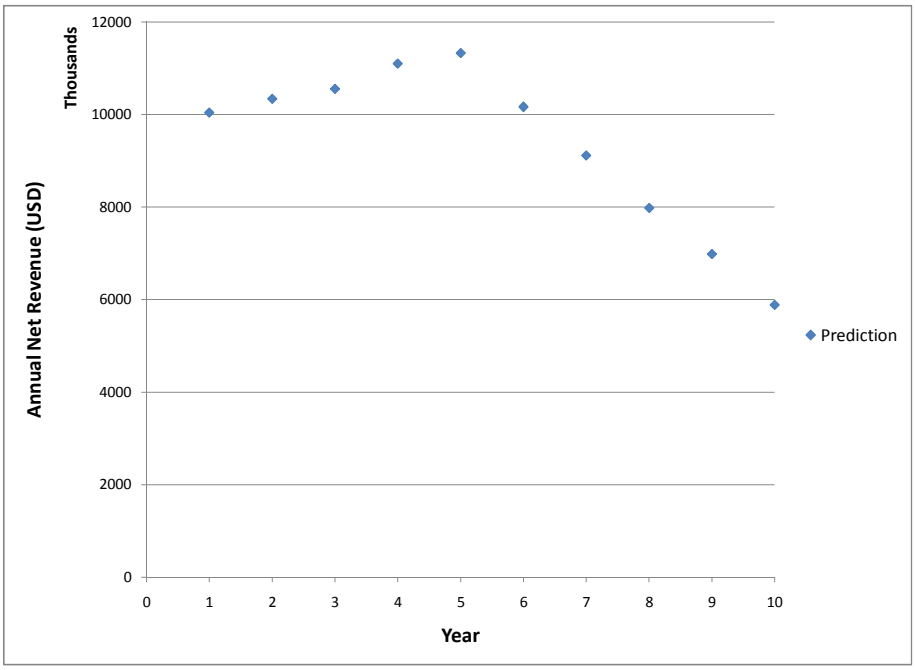


Figure 58: Graph of Ten-Year Revenue Predictions

Table 7: Ten-Year Financial Forecast

Year	Growth	Units Sold	Parts Cost	Other Cost	Income	Balance	Running Total
1	–	166000	56000000	242000	66000000	10283000	10283000
2	+2%	169000	57000000	149000	68000000	10587000	20870000
3	+2%	173000	58000000	149000	69000000	10801000	31671000
4	+5%	181000	61000000	149000	73000000	11349000	43020000
5	+2%	185000	62000000	149000	74000000	11579000	54599000
6	-10%	166000	56000000	149000	67000000	10406000	65005000
7	-10%	150000	50000000	149000	60000000	9350000	74355000
8	-12%	132000	44000000	149000	53000000	8210000	82565000
9	-12%	116000	39000000	149000	46000000	7207000	89772000
10	-15%	99000	33000000	149000	39000000	6104000	95876000

Table 8 shows the initial and recurring costs for our business. Recurring costs are further broken down into fixed and variable costs. Variable costs are shown as cost per system, independent of the volume of sales. All of the variable costs were calculated in table 6, and the initial costs were calculated in tables 4 and (hours worked; in text of document).

Table 8: Costs Overview

Cost Type	Detail	Amount
Initial Costs	Facilities	$11,000
	Prototyping	$82,000
	Total	$93,000
Fixed Recurring Costs	Marketing & Advertising	$120,000
	Legal	$17,000
	Facilities	$12,000
	Total	$149,000
Variable Costs Per System	Parts Cost	$277
	Labor	$44
	Shipping	$16
	Total	$337

Table 9 gives the cost of assembling each subsystem based on the number of hours needed to assemble each subsystem as estimated by the design team. Costs for wages are based on information from Professor Nielson's lecture first semester.

Table 9: Assembly Costs

Item	Value	Units
Hourly wage	20	USD/hour
Insurance, vacation, holiday, etc.	10	USD/hour
Per work-hour total	30	USD/hour
Time to assemble breakers	0.75	hour
Time to assemble base station	0.5	hour
Time to assemble E-meter	1.25	hour
Breakers per year	192000	
Base stations per year	64000	
E meters per year	830000	

Continued on next page

Table 9: Continued from previous page

Item	Value	Units
Total time for breaker	144000	hour
Total time for base station	32000	hour
Total time for E-meter	1037500	hour
Breakers labor cost	4320000	USD
Base station labor cost	960000	USD
E-meter labor cost	31125000	USD
Per-breaker labor	22.5	USD/Device
Per-base-station labor	15	USD/Device
Per-E-meter labor	37.5	USD/Device
Total Labor Hours	1213500	hour/year
Total Labor Cost	36405000	USD/year

Table 10 calculates the cost for a storage facility where the manufactured systems can be stored until they are shipped to the customers. Floorspace needed was calculated based on the physical volume of each subsystem and how many pallets would be needed to hold a quarter of a year's supply. From this information we found the cost of a standard warehouse facility using information found at `www.buildingsguide.com`.

Table 10: Storage Costs

	Breaker Storage	E-Meter Storage	Base Station Storage	Total
Width (ft)	0.16	0.5	1	
Height (ft)	0.25	0.67	0.67	
Length (ft)	0.33	0.25	0.33	
Volume (ft^3)	0.01	0.08	0.22	
Yearly Supply	38,000	166,000	13,000	
Quarterly Supply	9,500	41,500	3,250	
Pallet Height (ft)	3	3	3	
Pallet Stack	3	3	3	
Total Height (ft)	9	9	9	
Floor Area (ft^2)	14	384	80	479
Floor Length (sqare) (ft)	4	20	9	22

Continued on next page

Table 10: Continued from previous page

	Breaker Storage	E-Meter Storage	Base Station Storage	Total
Cost	$323	$8,832	$1,844	$11,000
Per-unit Cost	$0.03	$0.21	$0.57	
Yearly Supply (Peak)	44,636	190,000	14,879	190,000
Quarterly Supply (Peak)	11,250	47,500	3,750	
Floor Area (At Peak) (ft^2)	17	440	80	549
Floor Length (Peak, Square) (ft)	4	21	10	23

Table 11 shows calculations for how many customers our business expects to have. Using the total number of electric companies and number of homes in the U.S., we determined our percentage of the market and combined this with the estimate of the total number of potential customers from the business team to find the number of systems we can expect to sell. Final numbers are given in terms of each subsystem.

Table 11: Expected Demand

	Number	Modified	
Customers	6,400,000	1,280,000	1,280,000
Percentage Interested	0.001	0.0002	0.0002
Customers Interested	64,000	12,800	13,000
Breakers/Customer	3	0.6	0.6
Breakers Needed	192,000	38,400	38,000
Base Stations Needed	64,000	12,800	13,000
US Power Companies	3200	640	640
Number Converted Homes	8,300,000	1,660,000	1,660,000
Number Expected Conversions	24,900,000	4,980,000	4,980,000
Total Homes	130,000,000	260,000,000	260,000,000
Average Homes/Company	40,625	8,125	8000
Major Meter Providers	5	1	1
Minors Per Major	3	0.6	0.6
Minor Meter Providers	15	3	3
Ratio Homes Provided	3	0.6	0.6
Effective Meter Providers	30	6	6
PICA Homes	830,000	166,000	166,000

Continued on next page

Table 11: Continued from previous page

	Number	Modified	
E-Meters Needed	830,000	166,000	166,000

Table 12 shows the costs of the subsystems as calculated in other tables and determines the cost of a full system for use in table 1.

Table 12: Cost Per Unit

System	Detail	Value
Breaker	Cost of Parts per	$35
	Cost of Assembly per	$23
	Shipping	$3
	Total Cost Per	$61
	Number	38000
	Storage Cost	$0.03
Base Station	Cost of Parts Per	$100
	Cost of Assembly per	$15
	Shipping	$8
	Total Cost Per	$123
	Number	13000
	Storage Cost	$0.57
E-Meter	Cost of Parts Per	$200
	Cost of Assembly Per	$38
	Shipping	$15
	Total Cost Per	$253
	Number	166000
	Storage Cost	$0.21
System	Cost of Parts Per	$216
	Cost of Assembly Per	$44
	Shipping	$16
	Total Cost Per	$277
	Number	166000
	Storage Cost	$0.26

13 Parts and Project Costs

The project costs can be broken into fixed and variable costs, where fixed costs represent the costs the team will incur during the year and production start-up costs, while variable costs represent the long-term costs associated with production.

13.1 Fixed Costs

13.1.1 Prototype parts

Through the Senior Design class, the Calvin College Engineering department provided $750 for prototype parts. Since the project has a large scope, the team needs to minimize the cost of any single part to stay within the given budget. As such, the team chose several parts more because of their low cost rather than for functionality. As such, the team sought donations and free samples whenever possible, including the TI MSP430 development kit from Texas Instruments and the ADE7763 power monitoring chips from Analog Devices. The team also obtained a pre-purchased Xilinx Virtex-5 development board. Tables 13 through 18 shows the cost of parts for Team PICA's prototype, and the cost of parts for a full scale production model.

Table 13: Materials and Cost for a Single Breaker

Item	Quantity	Unit Price	Single System	Manufacturer Part #	Vendor
Solid State Relays	2	18.42	36.84	STH24D25	Mouser
5pF SMD Cap	2	0.03	0.06	C0603C0G1E050C	Mouser
Opto Isolator	2	3.03	6.06	OPI1264C	Mouser
3.579545MHz Crystal	1	0.63	0.63	ABLSG-3.579545MHZ-D-2-Y-T	Mouser
Total:	7		43.59		

Table 14: Materials and Cost for 1000 Breakers

Item	Quantity	Price per 1000	For 1000 Systems	Manufacturer Part #	Vendor
Solid State Relays	2	13.00	26000.00	STH24D25	Mouser
5pF SMD Cap	2	0.005	10.00	C0603C0G1E050C	Mouser
Opto Isolator	2	1.76	3520.00	OPI1264C	Mouser
3.579545MHz Crystal	1	0.35	350.00	ABLSG-3.579545MHZ-D-2-Y-T	Mouser
Total:	7.00		29880.00		

Table 15: Materials and Cost for a Single E-Meter

Item	Quantity	Part Price	Single System	Manufacturer Part #	Vendor
LCD Display	1	22.75	22.75	SCLBC	SynchroSystems
SSOP to DIP Adapter 20-Pin	2	3.95	7.90	BOB-00499	SparkFun
Break Away Headers – Straight	1	2.5	2.50	PRT-00116	SparkFun
1N4148 Diode	24	0.09	2.16	1N4148	Mouser
2500 Series Box Header	2	1.41	2.82	N2514-6002RB	Mouser
AVE Series Electrolytic Cap 1uF	12	0.07	0.84	AVE105M50B12T-F	Mouser
Kehmet .033uF SMD Cap	12	0.59	7.08	C1206C333KARACTU	Mouser

Continued on next page

Table 15: Continued from previous page

Item	Quantity	Part Price	Single System	Manufacturer Part #	Vendor
Kehmet .015uF SMD Cap	6	0.34	2.04	C1206C153KARACTU	Mouser
Kehmet 47pF SMD Cap	12	0.07	0.84	C0603C470J5GACTU	Mouser
Murata Inductor EFI	12	0.11	1.32	BLM21BD102SN1D	Mouser
Maxim RS233a RS232 Driver	1	8.7	8.70	MAX233CPP+G36	Mouser
D-Sub 9 Connector	1	2.32	2.32	56F404-001	Mouser
Xbee Connector 2mm pitch	2	1.6	3.20	M22-7131042	Mouser
16MHz Crystal	1	0.41	0.41	ABLS-16.000MHZ-B4-T	Mouser
32.768kHz Crystal	1	0.99	0.99	ABS10-32.768KHZ-7-T	Mouser
Tyco 6P Terminal Block	2	2.33	4.66	796949-6	Mouser
Tyco 3P Terminal Block	2	0.83	1.66	796949-3	Mouser
Varistor	3	0.26	0.78	V8ZA05P	Mouser
Current Transformer	3	6.11	18.33	CST-1020	Mouser
Linear Regulator	3	1.54	4.62	ZSR300GTA	Mouser

Continued on next page

Table 15: Continued from previous page

Item	Quantity	Part Price	Single System	Manufacturer Part #	Vendor
100nF SMD Cap	3	0.7	2.10	CB037D0104JBA	Mouser
22uF SMD Cap	4	0.54	2.16	EEE-1CA220SR	Mouser
18pF SMD Cap	2	0.09	0.18	CGA2B2C0G1H180J	Mouser
Red SMD LED	1	0.4	0.40	HSMC-C190	Mouser
Green SMD LED	1	0.88	0.88	HSMQ-C120	Mouser
Amber SMD LED	1	0.4	0.40	HSMA-C190	Mouser
7pF SMD Cap	2	0.03	0.06	C0603C0G1E070C	Mouser
Tact Switch	2	0.47	0.94	B3W-1052	Mouser
Total:	119		103.04		

Table 16: Materials and Cost for 1000 E-Meters

Item	Quantity	Unit Price	Single System	Manufacturer Part #	Vendor
Solid State Relays	2	18.42	36.84	STH24D25	Mouser
5pF SMD Cap	2	0.03	0.06	C0603C0G1E050C	Mouser
Opto Isolator	2	3.03	6.06	OPI1264C	Mouser
3.579545MHz Crystal	1	0.63	0.63	ABLSG-3.579545MHZ-D-2-Y-T	Mouser
Total:	7		43.59		

Table 17: Materials and Cost for a Single Power Supply

Item	Quantity	Part Price	Single System	Manufacturer Part #	Vendor
1000uF Cap	1	3.18	3.18	TVX1J102MCD	Mouser
10u Cap	3	0.3	0.90	C1206C106K9PACTU	Mouser
100n Cap	2	0.03	0.06	C1608Y5V1H104Z	Mouser
47n Cap	1	0.2	0.20	C1608X7S2A473K	Mouser
2.2n Cap	1	0.16	0.16	06035C222K4Z2A	Mouser
1n Cap	1	0.03	0.03	C1608X7R1H102M	Mouser
2A 30V Schottky Diode	1	0.55	0.55	RB060M-30TR	Mouser
4.7u Inductor	1	0.72	0.72	SRR0604-4R7ML	Mouser
56K Resistor	1	1.6	1.60	RG1608N-563-B-T1	Mouser
15K Resistor	1	0.1	0.10	TNPW060315K0DEEA	Mouser
10K Resistor	1	1.1	1.10	PAT0603E1002BST1	Mouser
1.5K Resistor	2	0.79	1.58	TNPW06031K50BEEN	Mouser
10 Resistor	2	0.62	1.24	TNPW060310R0BEEA	Mouser
0.1 Resistor	1	0.3	0.30	CRL1206-FW-R100ELF	Mouser
LM25011	1	4.86	4.86	LM25011MY/NOPB	Digikey
Transformer	1	23.56	23.56	166M18	Mouser
Diode	4	0.23	0.92	1N4004G	Mouser
Total:	25		41.06		

Table 18: Materials and Cost for 1000 Power Supplies

Part	Quantity	Price per 1000	For 1000 Systems	Manufacturer Part #	Vendor
1000uF Cap	1	2.02	2020	TVX1J102MCD	Mouser

Continued on next page

Table 18: Continued from previous page

Part	Quantity	Price per 1000	For 1000 Systems	Manufacturer Part #	Vendor
10u Cap	3	0.055	165	C1206C106K9PACTU	Mouser
100n Cap	2	0.007	14	C1608Y5V1H104Z	Mouser
47n Cap	1	0.036	36	C1608X7S2A473K	Mouser
2.2n Cap	1	0.08	80	06035C222K4Z2A	Mouser
1n Cap	1	0.006	6	C1608X7R1H102M	Mouser
2A 30V Schottky Diode	1	0.169	169	RB060M-30TR	Mouser
4.7u Inductor	1	0.34	340	SRR0604-4R7ML	Mouser
56K Resistor	1	0.37	370	RG1608N-563-B-T1	Mouser
15K Resistor	1	0.072	72	TNPW060315K0DEEA	Mouser
10K Resistor	1	0.472	472	PAT0603E1002BST1	Mouser
1.5K Resistor	2	0.47	940	TNPW06031K50BEEN	Mouser
10 Resistor	2	0.37	740	TNPW060310R0BEEA	Mouser
0.1 Resistor	1	0.121	121	CRL1206-FW-R100ELF	Mouser
LM25011	1	2.835	2835	LM25011MY/NOPB	Digikey
Transformer	1	17.28	17280	166M18	Mouser
Diode	4	0.042	168	1N4004G	Mouser
Total:	25		25828.00		

13.2 Board Cost

After all the dimensions were found on the parts, the Visio drawings were done in order to try to get a minimum board size to create a PCB. The PCB price was found using PCBExpress. The team did not need to pay for the PCBs originally because that service was donated by Johnson Controls Incorporated. However, if the team were to go into production they would not have this luxury. Therefore, in order to get

an accurate Bill of Materials, this cost needed to be estimated at retail price along with all of the donated parts by various organizations.

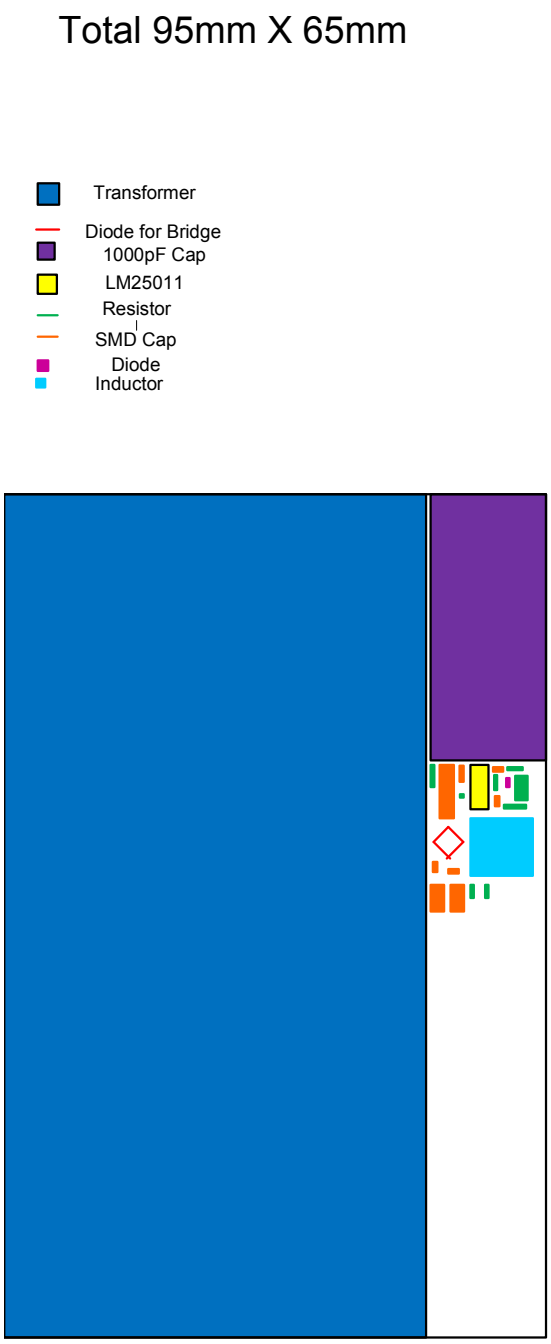
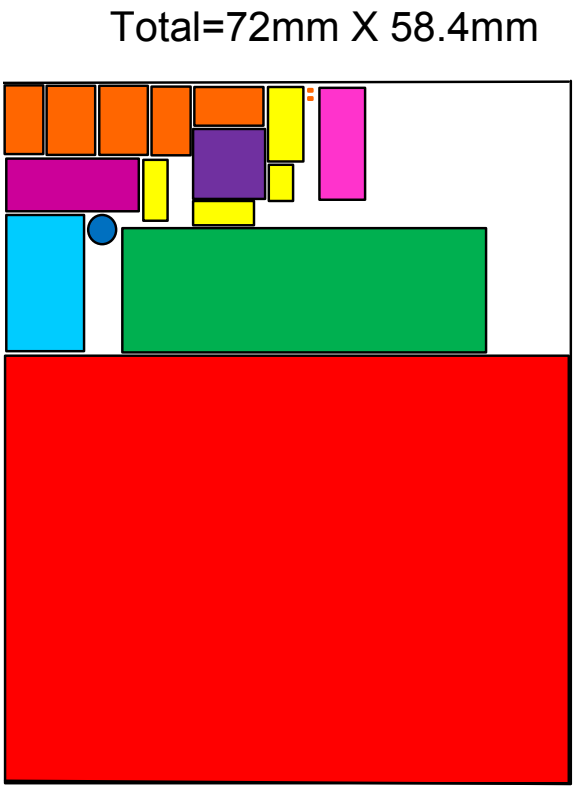
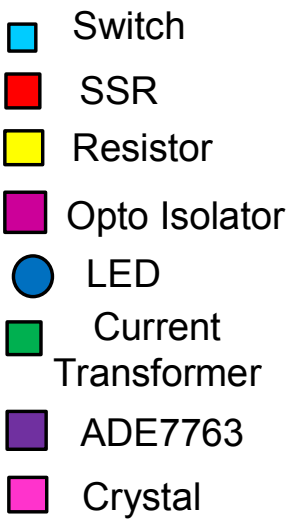


Figure 59: Power supply board mockup, 92.5 x 95 mm.

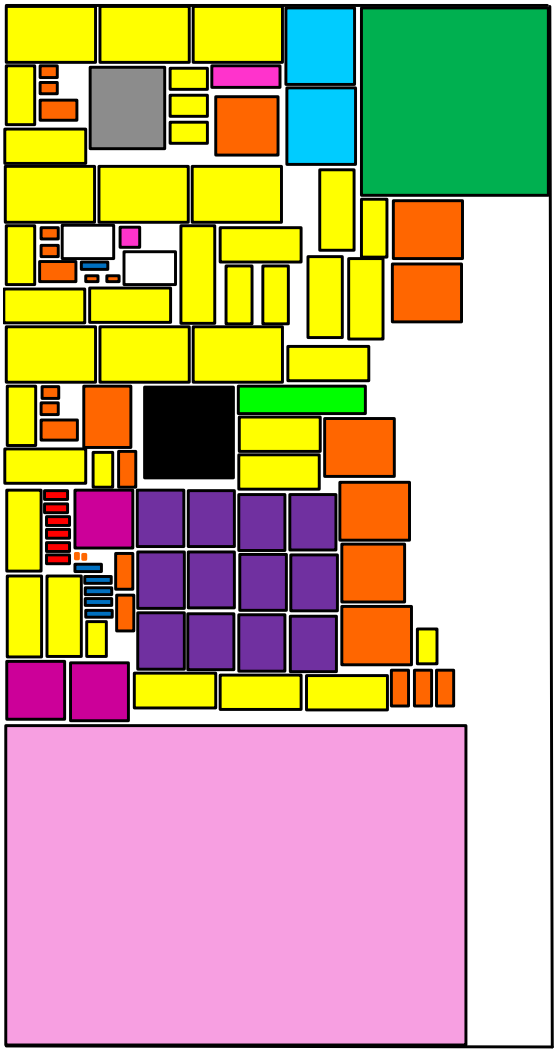


(a)

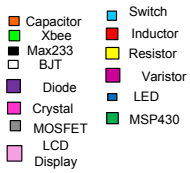


(b)

Figure 60: Smart breaker board mockup 72mm x 58.4mm.



(a)



(b)

Figure 61: E-Meter board mockup 90mm x 47mm.

13.3 Labor

Throughout the first and second semester, the team has kept a log of how many hours they worked. Determining the cost of labor for both semesters can accurately be assessed based on these records. For second semester so far, the team has logged 490.75 hours, with another 127.75 estimated before the end of the semester. The team also logged 388 hours for first semester, putting the yearlong total at about 1006.5 hours. Assuming engineers are paid $100 an hour, the first semester labor cost is $38,800, the second semester cost is $61,850, making the full labor costs for the project $100,650 as calculated in table 19. Labor has been further broken down by team member in table 20.

Timeframe	**Hours Logged**	**Cost**
First Semester	388	$38,800
Second Semester	618.5	$61,850
Total	1006.5	$100,650

Table 19: Project hours

Team Member	First Semester	Second Semester	Total
Amy Ball	75	88	163
Nathan Jen	107.75	122.75	230.5
Avery Sterk	95	209	304
Kendrick Wiersma	125.5	242	367.5

Table 20: Project labor broken down by team member and semester as of 12 May 2011.

13.4 Manufacturing start-up

In determining the cost for start-up of manufacturing, the team looked strictly at the costs of the product, ignoring many costs associated with starting a business. The team decided to contract out the work needed to build the system. Due to the high volume of systems being manufactured, the cost to manufacture will be low and is included in the cost of parts for each subsystem. The cost of constructing a storage facility is then our largest up front cost.

Table 21 shows the calculations used to determine the labor costs for manufacturing purposes. Estimates given during lecture [42] helped determine the hourly wage and additional costs of labor including insurance, vacation, holiday, sick time etc. The number of hours needed to assemble each system does not include the time needed to print and populate each circuit board as these will be completed by automatic machinery that requires minimal human interaction.

Table 21: Cost of labour for manufacturing

Line Item	Number
Hourly wage	$20
Insurance, vacation, holiday, etc.	$10
Per worker total	$30
Hours to assemble breakers	1
Hours to assemble base station	0.5
Hours to assemble E-meter	1.5
Breakers per year	1,600,000
Base stations per year	64,000
E-meters per year	830,000
Hours for breaker	1,600,000
Hours for base station	32,000
Hours for e meter	1,245,000
Breaker cost	$48,000,000
Base station Cost	$960,000
E-meter cost	$37,350,000
Total Hours	2,877,000
Total Cost	$86,310,000

13.4.1 Distribution

The team has not yet contacted a shipping company to determine exact costs, but based on experience, size and weight, the team expects a total cost of $25,400,000. Table 22 shows the cost of distribution for each subsystem.

Table 22: Cost of distribution

	Breaker	Base Station	E-meter
Number shipped	1,600,000	64,000	830,000
Cost per device	$5	$12	$20
Total cost	$8,000,000	$768,000	$16,600,000
Total Shipping Cost	$25,368,000		

13.5 Variable Costs

13.5.1 Parts

To calculate the overall cost of parts used in production, the team assumed large quantities for each of the individual components. This means that the low end of the cost range for parts is used. Based on this information and the estimated final cost of the prototype, the team calculates the cost of parts and manufacturing for the breakers, base station and e-meter to be $35, $100, and $200 per subsystem, respectively.

13.5.2 Marketing

The project includes two distinct advertising methods to better accommodate the different target consumers. The e-meter aspect of the project will be sold directly to the power company, and the breakers and base station will be sold to the home and business owners. As the number of power companies is significantly fewer than the number of home and business owners, and will be purchasing in much larger quantities, the team decided it makes sense to appeal to the power companies in a much more personal manner. This includes phone calls, letters, and visits and outside of the cost of paying a few employees will be negligible.

Most of the advertising will aim at the home and business owners, and the team decided that magazines and websites such as Popular Science and Green magazine are the best method of reaching out to potential

buyers. Green magazine features news and products related to sustainable energy, reaching thousands of people every year. Approximately 36% of those people are in the building and contracting industry and would be beneficial in spreading news about the team's product [43]. The cost to put a medium size ad on their website is $150 dollars per month [43], so for a year would be $1800. Popular science reaches over 7 million people using printed material. For a 1/3 page ad in four color for 12 months, the cost is $59,900 [44]. The team would like to target 3 to 4 magazines and using Popular Science and Green magazine as boundary cases, estimates a total cost of $120,000 for marketing and advertising.

For the home and business owners' side of the project, the team also would like to work with distributers like Lowe's and Home Depot. The team would like to use a method of advertising similar to the one used for power companies, so the cost will not noticeably increase. The distribution companies may do additional advertising, but any costs associated with that will be their responsibility, so again the cost the design team expects will stay the same.

13.5.3 Legal, warranty and support

The team expects about 10 hours of work for basic legal documentation. Because of the potential for lawsuits, the team built in money to cover the costs of 200 hours of work, assuming $80 an hour, giving a total of $16,800. The team does not intend to pursue any patents, but recognizes there may be infringement lawsuits, which were built into the above 200 hours.

The team expects 5% of all PICA systems that include all three subsystems to fail and need replacement. At a system cost of $260 per system with shipping of $30, the amount needed to cover warranties is $2,432,000.

13.6 Total Costs

Table 23 is a summary of the project costs, including both fixed and variable costs.

Table 23: Cost estimate for the project.

Full Scale Production				
Fixed Costs	Prototyping	$120,700		
	Automated Equipment			
	Marketing/Advertising	$120,000		

Continued on next page

Table 23: Continued from previous page

Full Scale Production				
	Legal	$168,000		
	Facilities	$0		
	Total	$257,500		
Variable Costs	System	Breakers	Base station	E-meter
	Number of devices	1600000	64000	830000
	Single device parts	35	100	200
	Total device parts	56000000	6400000	166000000
	Labor	$48,000,000	$960,000	$37,350,000
	Shipping	$8,000,000	$768,000	$16,600,000
	Inventory	$5,100	$31,800	$6,500
	Total (per device)	$70	$127	$265
Totals	Total Per Subsystem	$112,262,600	$8,417,300	$220,214,000
	Total per device	$70	$132	$265
	Full System	$340,893,900.00		

14 Acknowledgements

Team PICA would like to acknowledge and thank the following people for their help in the course of the project.

- Professor VanderLeest for his work as the Team Advisor, providing constructive feedback and insights, and for pushing us to always improve.
- Professor Ribeiro for his help in Engr. 315 and insights on power systems.
- Consumer's Energy, for allowing us to visit their Smart Grid Learning Center, and Brian Bushey, Mark Luehmann and Tim Voss specifically for answering many questions about their work as a power company.
- Tim Theriault for his time as the team's Industrial Consultant.
- Mark Michmerhuizen for his work providing feedback on team documents.
- Chuck Cox, and John Lupien of SynchroSystems for their assistance in debuggin the E-Meter LCD.
- The Business team from Bus. 396 for helping us understand business aspects of our project.
- Professor Medema for helping understand business aspects of our project.
- Bob DeKraker, Chuck Holwerda and Phil Jasperse for their help as shop and lab technicians.
- Glenn Remelts for assisting with team research.
- Texas Instruments for generously providing us with a MSP430 development kit.
- All of our friends and family for supporting, encouraging and putting up with us.

References

[1] S. English and D. Smith, “A power meter reference design based on ade7756,” Electronic, 2001.

[2] T. Instruments, “Msp430f471x3, msp430f471x6, msp430f471x7 mixed signal microcontroller,” March 2011.

[3] MIT, “Eecsenergy overview: Trends and motivation,” Electronic, 2009. [Online]. Available: http://www.eecs.mit.edu/eecsenergy/

[4] P. America, “Average retail cost of energy per kilowatt,” Electronic. [Online]. Available: http://www.project.org/info.php?recordID=247

[5] National power corporation. [Online]. Available: www.natpow.com

[6] M. Hill, “What are voltage sags?”

[7] The home depot. [Online]. Available: www.homedepot.com

[8] ANSI, “American national standard for utility industry end device data tables,” National Electrical Manufacturers Association, Tech. Rep., February 2009.

[9] ——, “Ansi c12.21-2006 american national standard protocol specification for telephone modem communication,” National Electrical Manufacturers Association, Tech. Rep., May 2006.

[10] FCC. Title 47: Telecommunication. [Online]. Available: http://ecfr.gpoaccess.gov/cgi/t/text/text-idx?c=ecfr&sid=72eb0650720fb4c17ca9a1f782cdb360&rgn=div5&view=text&node=47:1.0.1.1.14&idno=47

[11] A. Ball, A. Sterk, K. Wiersma, and N. Jen, “Team 01: Team pica project proposal and feasability study,” December 2010. [Online]. Available: http://ujoint.calvin.edu/academic/engr/2010-11-team1/documents/T01_Team_PICA_PPFS_B.pdf

[12] (2011, April) Synchrosystems. [Online]. Available: www.synchros.com

[13] “Softbaugh sblcd4,” April 2011. [Online]. Available: http://www.softbaugh.com/ProductPage.cfm?strPartNo=SBLCDA4

[14] T. Instruments, “slac316a.”

[15] T. Magnetics, “Cst-1020 current transformer,” 2010.

[16] S. Underwood, V. Chan, and K. Venkat, "Implementation of a three-phase electronic watt-hour meter using the msp430f471xx," Application Report, June 2009.

[17] T. Instruments, "Msp430x4xx family user's guide," January 2011.

[18] "Analog devices single-phase active and apparent energy metering ic ade7763," Electronic Data Sheet, 2004.

[19] C. Magnetics, "High ratio vertical pcb mount current transformer," Datasheet.

[20] ABRACON, "Hc/49us (at49) 3rd lead case-grounded smd low profile crystal," Datasheet, August 2010.

[21] Instructable eagle part development. [Online]. Available: www.instructables.com

[22] (2011, April). [Online]. Available: http://schmidt-walter.eit.h-da.de/smps_e/smps_e.html

[23] (2011, April). [Online]. Available: http://www.national.com/pf/LM/LM25011.html#Overview

[24] (2011, April). [Online]. Available: http://www.national.com/analog/webench/power

[25] U. Laboratories. (2010) About ul. [Online]. Available: http://www.ul.com/global/eng/pages/corporate/aboutul/?set-cookie=true

[26] P. International, "About p3," 2008. [Online]. Available: http://www.p3international.com/about.html

[27] ——, "Kill a watt ps electricity usage monitoring power strip," Electronic Data Sheet, 2007.

[28] ——, "Kill a watt electricity usage monitor," Electronic Data Sheet, 2007.

[29] ——, "Kill a watt ez," Electronic Data Sheet, 2007.

[30] Clipsal. (2010) Cent-a-meter sales. [Online]. Available: http://www.clipsal.com/homeowner/products/smart_home_technology/cent-a-meter

[31] (2010, November) Smarthomeusa webshop. [Online]. Available: http://www.smarthomeusa.com/ShopByManufacturer/eco-response/Item/CM113A/specifications/

[32] (2010) Ted 1000 features. [Online]. Available: http://www.theenergydetective.com/ted-1000/features

[33] (2010) Ted1000 store. [Online]. Available: http://www.theenergydetective.com/store/ted-1000

[34] "Ted 5000 features," 2010. [Online]. Available: http://www.theenergydetective.com/ted-5000/features

[35] "Ted 5000 store," 2010. [Online]. Available:
http://www.theenergydetective.com/store/ted-5000?limit=25

[36] (2010) About watts up? [Online]. Available: https://www.wattsupmeters.com/secure/about.php

[37] (2010) Smart circuit 20. [Online]. Available:
https://www.wattsupmeters.com/secure/products.php?pn=20&wai=256&more=2

[38] (2010) Watts up? services overview. [Online]. Available:
https://www.wattsupmeters.com/secure/products.php?pn=2

[39] SmartWorks. (2010) Inline high-resolution watt-hour meter. [Online]. Available:
http://www.smart-works.com/pdf/SmartWattBrochureInline.pdf

[40] G. W. Hart, E. C. Kern, Jr., and F. C. Schweppe, "Non-intrusive appliance monitor apparatus," U.S. Patent 4 858 141, August 15, 1989.

[41] G. W. Hart. (1992) Nonintrusive appliance load monitoring. [Online]. Available:
http://www.georgehart.com/research/nalm.html

[42] N. Nielsen, "Cost estimating and pricing," Lecture, November 2010.

[43] G. Magazine, "2010 green magazine media kit," Electronic.

[44] Advertising: Popsci media group sales information. [Online]. Available:
http://www.popsci.com/advertising

A Original Gantt Charts

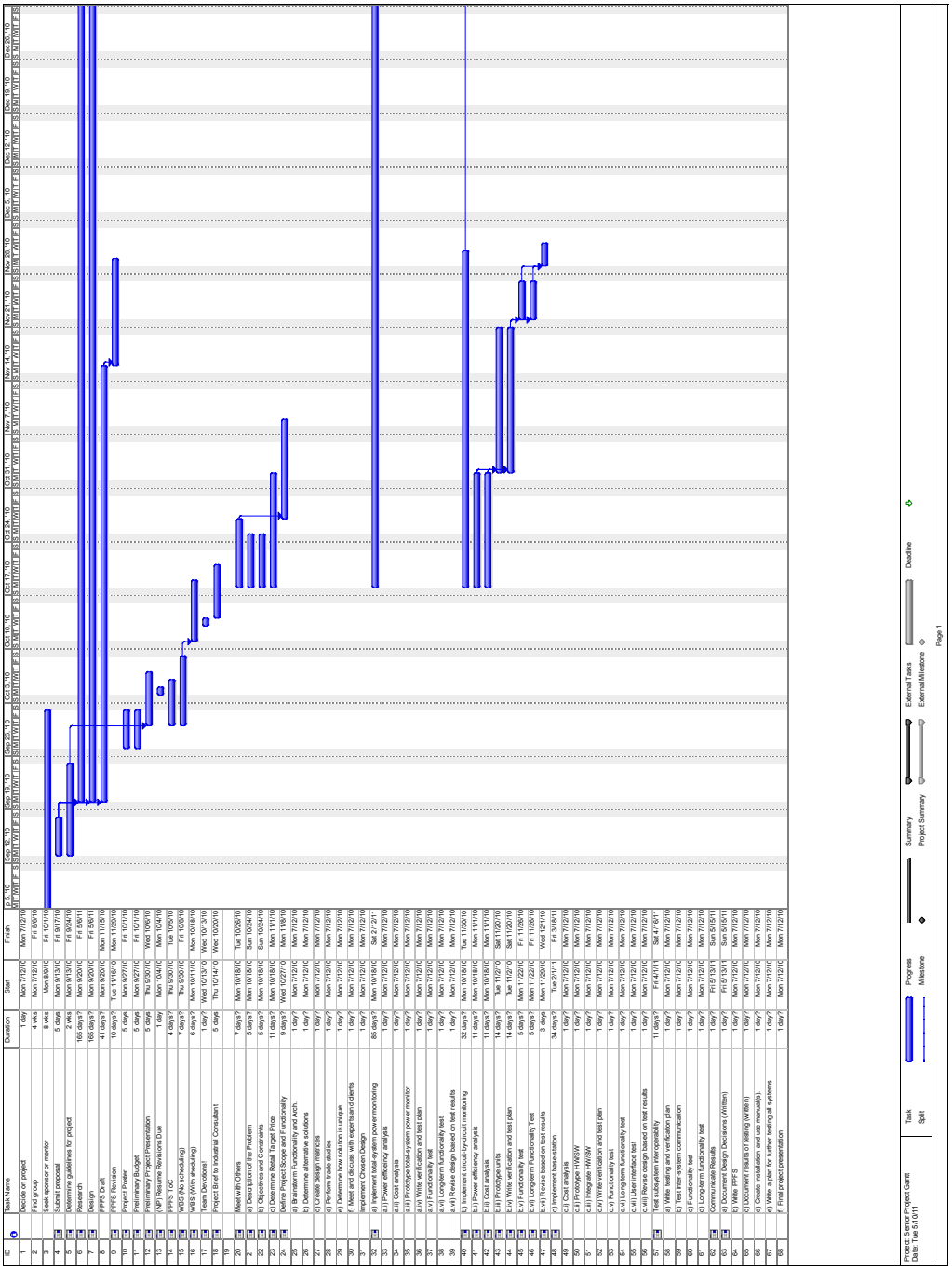


Figure 62: Original Gantt Chart for First-Semester Tasks

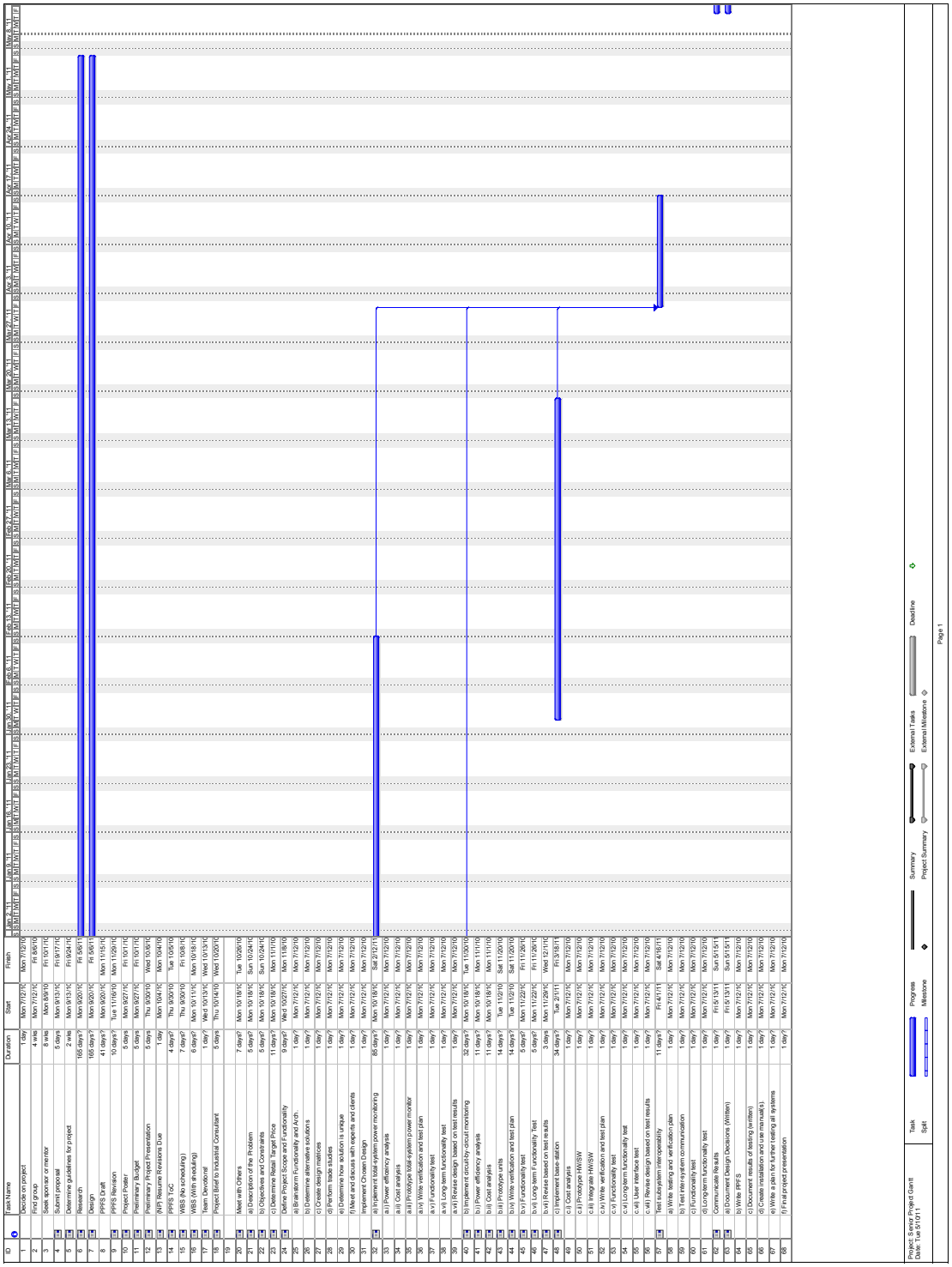


Figure 63: Original Gantt Chart for Second-Semester Tasks

B E-Meter Appendix

B.1 Device Pinout

The following table shows the pins of the MSP430 and how they are connected in the E-meter.

Table 24: MSP430F47197 Pinout

Pin Number	Pin Name	Pin Function
1	A0+	Phase 1 Voltage (Positive)
2	A0-	Phase 1 Voltage (Negative)
3	A1+	Phase 2 Voltage (Positive)
4	A1-	Phase 2 Voltage (Negative)
5	A2+	Phase 3 Voltage (Positive)
6	A2-	Phase 3 Voltage (Negative)
7	AVSS	Tie to Ground (GND)
8	AVCC	3V input
9	VREF	Virtual Ground
10	A3+	Phase 1 Current (Positive)
11	A3-	Phase 1 Current (Negative)
12	A4+	Phase 2 Current (Positive)
13	A4-	Phase 2 Current (Negative)
14	A5+	Phase 3 Current (Positive)
15	A5-	Phase 3 Current (Negative)
16	A6+	Unused
17	A6-	Unused
18	VSS	Tie to Ground (GND)
19	S0	LCD segment pin 00
20	S1	LCD segment pin 01
21	S2	LCD segment pin 02
22	S3	LCD segment pin 03
23	S4	LCD segment pin 04
24	S5	LCD segment pin 05

Continued on next page

Table 24: Continued from previous page

Pin Number	Pin Name	Pin Function
25	S6	LCD segment pin 06
26	S7	LCD segment pin 07
27	S8	LCD segment pin 08
28	S9	LCD segment pin 09
29	S10	LCD segment pin 10
30	S11	LCD segment pin 11
31	S12	LCD segment pin 12
32	S13	LCD segment pin 13
33	S14	LCD segment pin 14
34	S15	LCD segment pin 15
35	S16	LCD segment pin 16
36	S17	LCD segment pin 17
37	S18	LCD segment pin 18
38	S19	LCD segment pin 19
39	S20	LCD segment pin 20
40	S21	LCD segment pin 21
41	S22	LCD segment pin 22
42	S23	LCD segment pin 23
43	S24	LCD segment pin 24
44	S25	LCD segment pin 25
45	S26	LCD segment pin 26
46	S27	LCD segment pin 27
47	S28	LCD segment pin 28
48	S29	LCD segment pin 29
49	S30	LCD segment pin 30
50	S31	LCD segment pin 31
51	S32	LCD segment pin 32
52	S33	LCD segment pin 33

Continued on next page

Table 24: Continued from previous page

Pin Number	Pin Name	Pin Function
53	S34	LCD segment pin 34
54	S35	LCD segment pin 35
55	P5.0	Unused
56	COM0	LCD commmon 0
57	COM1	LCD commmon 1
58	COM2	LCD commmon 2
59	COM3	LCD commmon 3
60	P5.5	Unused
60	P5.5	Unused
60	P5.5	Unused
63	LCDCAP	Charge pump capactior (22µF to ground)
64	DVCC2	Digital voltage input (1.5V)
65	XIN	32.768kHz crystal
66	XOUT	32.768kHz crystal
67	DVSS2	Tie to Ground (GND)
68	S36	LCD segment pin 36
69	S37	LCD segment pin 37
70	S38	LCD segment pin 38
71	S39	LCD segment pin 39
72	P3.3	Unused
73	P3.2	Unused
74	P3.1	Unused
75	P3.0	Unused
76	P2.7	Unused
77	P2.6	Unused
78	P2.5	RS232 RX
79	P2.4	RS232 TX
80	P2.3	Button
81	P2.2	Unused

Continued on next page

Table 24: Continued from previous page

Pin Number	Pin Name	Pin Function
82	P2.1	Unused
82	P2.0	Unused
82	P1.7	Unused
85	P1.6	LED3 (Green)
86	P1.5	LED2 (Amber)
85	P1.4	LED1 (Red)
88	P1.3	Unused
89	P1.2	LCD Backlight LED
90	P1.1	Unused
91	P1.0	Unused
92	DVCC	100nF capacitor
93	XT2OUT	16.000MHz crystal
94	XT2IN	16.000MHz crystal
95	DVSS1	Tie to Ground (GND)
96	TDO/TDI	JTAG Test Data Out/In
97	TDI/TCLK	JTAG Test Data Out / Test Clock
98	TMS	Test Mode Select
99	TCK	Test Clock
100	RST	Reset (Assert Hi)

B.2 Hello World Program

Listing 1: Embedded C MSP430 Hello World program.

```
/*************************************************************
 * MSP430F47197 Hello World Demo
 * Description: Toggle LED located at P5.1 on MSP430 Dev Board
 * Hardware Setup: See diagram below, requires only JTAG clk.
 *
 *                   MSP430x471x7
 *               -----------------
 *          /|\|                  |
 *           | |                  |
 *           --|RST               |
 *             |                  |
 *             |              P5.1|-->LED
 * Author: Kendrick Wiersma -- kendrick.g.wiersma@gmail.com
 */

#include "msp430x471x7.h"

void main(void) {
  volatile unsigned int i;

  WDTCTL = WDTPW + WDTHOLD; // Ask the watchdog timer to please hold for us
  P5DIR |= BIT1;            // Set P5.1 to be an output

  while(1) {
    P5OUT ^= BIT1;
    for(i=50000; i > 0; i--); //delay for a bit
  }
}
```

B.3 Compiling the MSPGCC4 Toolchain for Ubuntu Linux

The design team used Ubuntu Linux 10.10 32-bit to compile the MSPGCC4 toolchain. The MSPGCC4 toolchain depends on several packages readily available in the Ubuntu repository:

- subversion
- gcc-4.4
- texinfo
- patch
- libncurses5-dev
- zlibc
- zlib1g-dev
- libx11-dev
- libusb-dev
- libreadline6-dev

Install all the following packages by executing the following command in Terminal:

Next, checkout the MSPGCC4 source from the online subversion repository using the command line version of subversion.

Listing 2: Install MSPGCC4 dependencies

```
sudo apt-get install subversion gcc-4.4 texinfo patch libncurses5-dev zlibc zlib1g-dev libx11-dev
     libusb-dev libreadline6-dev
```

Once all the code has been downloaded, switch to the new directory and execute the buildgcc.sh script. Accept all the default options and wait for a while until the compilation completes.

Now navigate to `http://mspdebug.sourceforge.net/download.html` and download the latest release. Extract the files from the downloaded archive. Change directories to the newly extracted code and issue a make command. Once the compilation completes, install mspdebug using make install.

This completes all the essential tools for compiling code using MSPGCC4. Some additional tools may be required to work with the programming pod.

In order to load code onto the MSP430, gdb and HAL need to be configured to properly talk to the programming pod. Change to the install directory of the MSPGCC4 (usually /opt/msp430-gcc-4.4.3/bin/msp430-gdb-proxy/) and issue the command:

```
sudo wget http://www.soft-switch.org/downloads/mspgcc/msp430-gdbproxy
```

once the file transfer completes issue the command:

```
chmod 777 msp430-gdbproxy
```

to allow the file to execute.

Now switch to /usr/lib and download the libHil.so file using wget:

```
sudo wget http://www.soft-switch.org/downloads/mspgcc/libHIL.so
```

Repeat to download the MSP430.so driver file:

```
sudo wget http://www.soft-switch.org/downloads/mspgcc/MSP430.so
```

B.4 SD16 to RS232 UART Test

The following code transmits results from the second SD16 converter to the RS232 UART

Listing 3: Embedded C MSP430 code to transmit data from the SD16 readings to the RS232 UART.

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

```
//  MSP430x471xx Demo - SD16, Single Conversion on a Single Channel Using ISR
//
//                 MSP430x471xx
//             -----------------
//         /|\|              XIN|-
//          | |                 | 32kHz
//          --|RST          XOUT|-
//            |                 |
//    Vin+ -->|A2.0+            |
//    Vin- -->|A2.0-            |
//            |                 |
//            |            VREF |---+
//            |                 |   |
//            |                 |  -+- 100nF
//            |                 |  -+-
//            |                 |   |
//            |            AVss |---+
//            |                 |
// Based on TI demo code
// Modified by Avery Sterk April 2011
//******************************************************************************
#include  <msp430x471x7.h>

unsigned int result;

void main(void)
{
  volatile unsigned int i;                  // Use volatile to prevent removal
                                            // by compiler optimization

  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  FLL_CTL0 |= XCAP14PF;                     // Configure load caps

  do
  {
    IFG1 &= ~OFIFG;                         // Clear OSCFault flag
    for (i = 0x47FF; i > 0; i--);           // Time for flag to set
  }
  while ((IFG1 & OFIFG));                   // OSCFault flag still set?

  P2SEL |= BIT4+BIT5;                       // P2.4,5 = USCI_A0 RXD/TXD
  UCA0CTL1 |= UCSSEL_1;                     // CLK = ACLK
  UCA0BR0 = 0x03;                           // 32k/9600 - 3.41
  UCA0BR1 = 0x00;                           //
  UCA0MCTL = 0x06;                          // Modulation
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  IE2 |= UCA0RXIE;                          // Enable USCI_A0 RX interrupt

  SD16CTL = SD16REFON+SD16SSEL0;            // 1.2V ref, SMCLK
  SD16CCTL2 |= SD16SNGL+SD16IE ;            // Single conv, enable interrupt
  for (i = 0; i < 0x3600; i++);             // Delay for 1.2V ref startup

  while (1)
  {
    _NOP();                                 // SET BREAKPOINT HERE
    SD16CCTL2 |= SD16SC;                    // Set bit to start conversion
    __bis_SR_register(LPM0_bits + GIE);     // Enter LPM0 w/ interrupts
  }
}

#pragma vector=SD16A_VECTOR
__interrupt void SD16AISR(void)
{
  switch (SD16IV)
  {
  case 2:                                   // SD16MEM Overflow
    break;
  case 4:                                   // SD16MEM0 IFG
```

```
    break;
  case 6:                                   // SD16MEM1 IFG
    break;
  case 8:                                   // SD16MEM2 IFG
    result = SD16MEM2;                      // Save CH2 results (clears IFG)
//    UCA0TXBUF = (result >> 9); // bitshift result down by 9 bits, so it's at least ASCII
    UCA0TXBUF = (SD16MEM2 >> 12) + 0x41; // map top 4 bits of 16-bit measurement to 'A'-'P
        '.
    break;
  }

  __bic_SR_register_on_exit(LPM0_bits);        // Exit LPM0
}
```

The following Matlab code reads the character values from the MSP430 UART and displays them in comparison with a reference sinusoid.

Listing 4: Matlab Interpreter for Character Output

```
% matlabReader.m
% Written by Kendrick, documented by Avery
% in case Avery needs to use it
s = serial('COM1'); % grab COM1
set(s,'BAUD',9600); % set its baudrate
fopen(s); % open it (as though it were a file)
k = 0;
for k=1:100 % collect 100 numbers
    char = fscanf(s, '%u', 1); % get a character from the serial port
    if ~isempty(char)
        charVec(k) = char; % add it to the vector
        % char % DEBUG: print it to console
    end % endif
end % endfor
% now, plot the vector, BUT....
% we have ascii characters starting with 'A', which is 0x65, so fix that offset
% and 'A' was the most negative reading we could get (-0.6V -> 0, 0.6V -> F)
% so adjust so that our 15-character span is centered around 0
plot( single(charVec)-65-7.5 )

fclose(s) % close the file descriptor
delete(s) % remove our link to COM1
clear all % clear the workspace
```

B.5 E-Meter Schematic

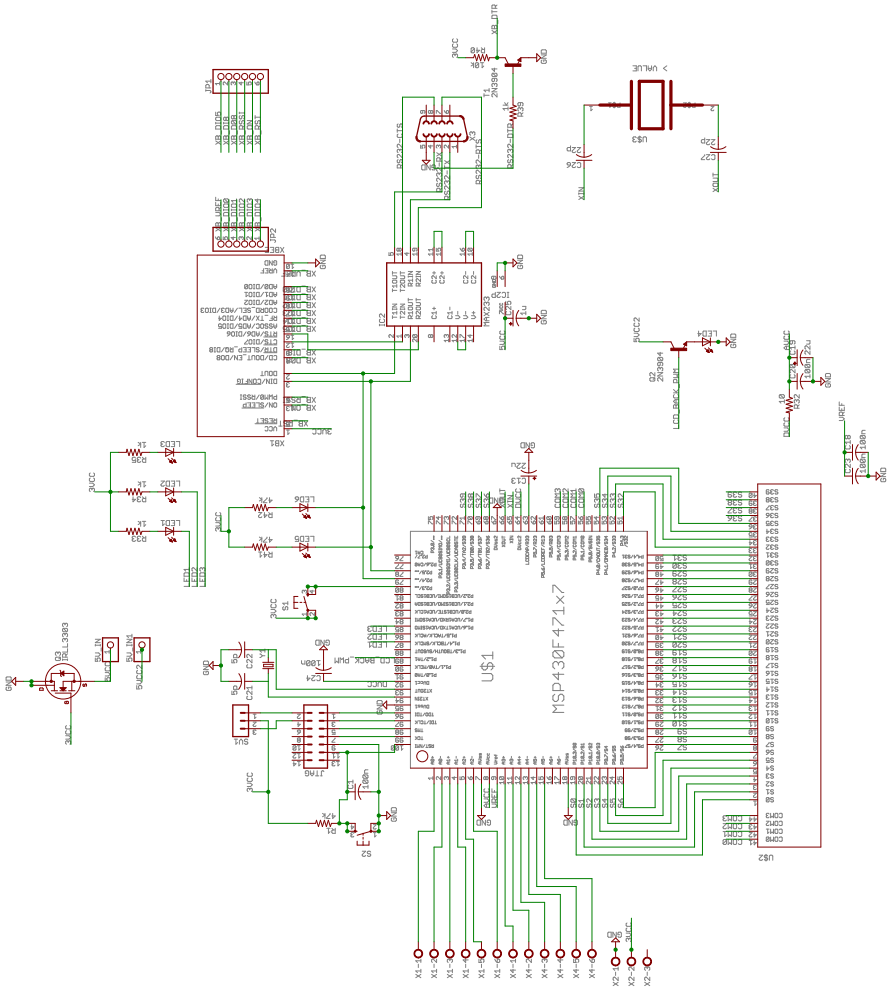


Figure 64: E-Meter Main Board Schematic

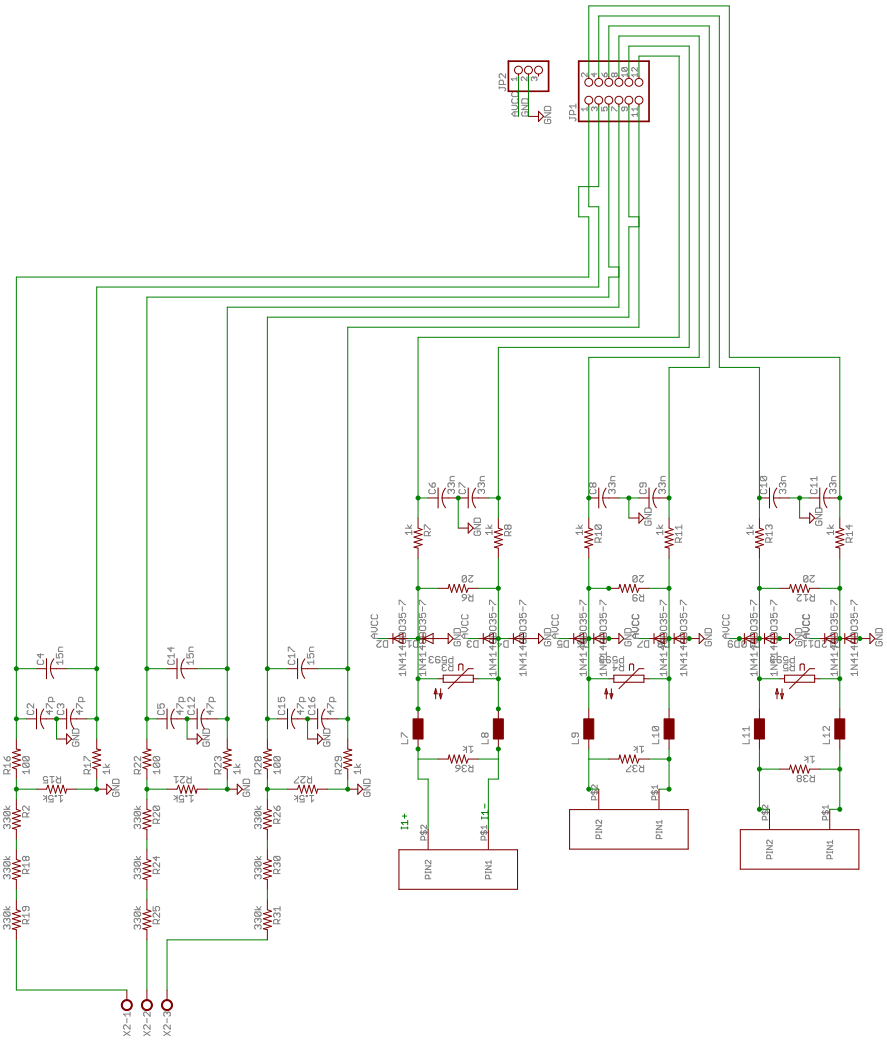


Figure 65: E-Meter input board schematic.

B.6 E-Meter Printed Circuit Board

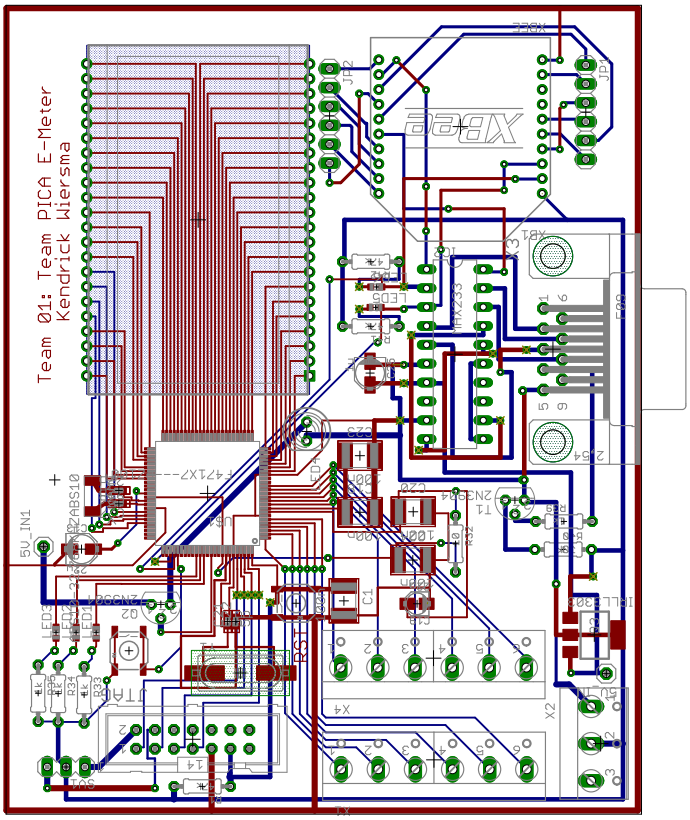


Figure 66: E-Meter main board.

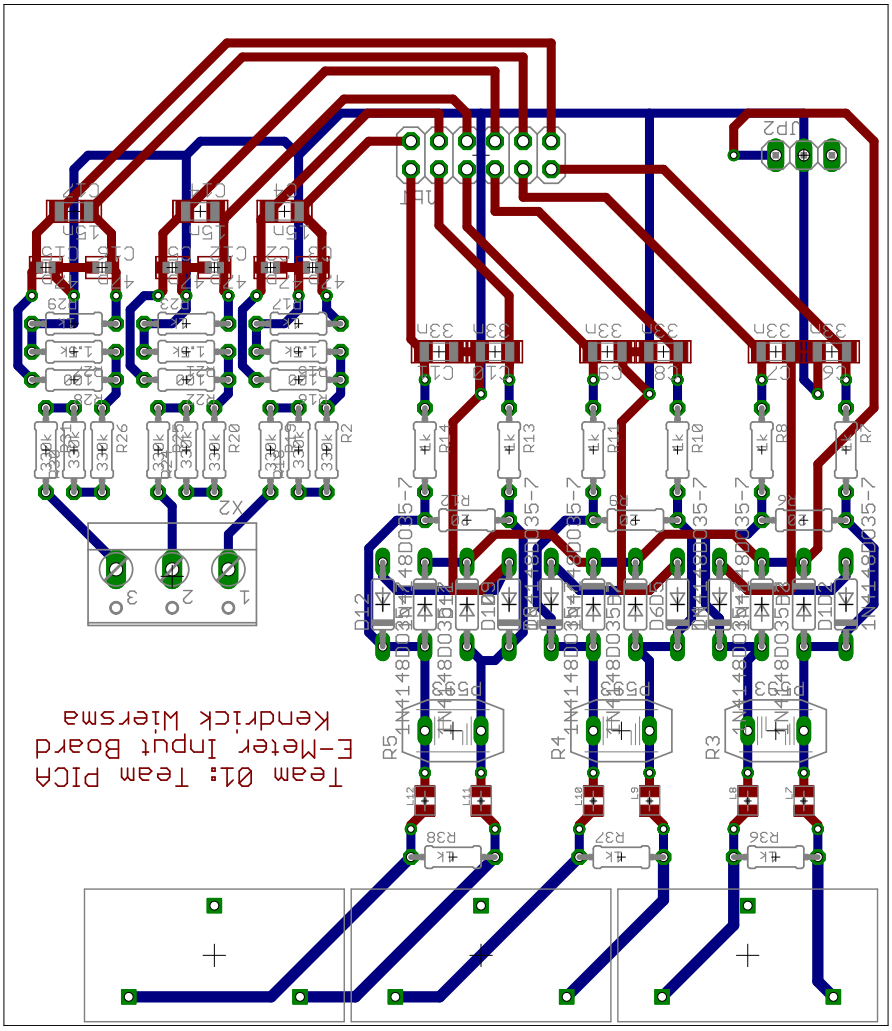


Figure 67: E-Meter input board.

B.7 E-Meter Code Listing

Listing 5: Final Measurement and Calculation Code

```
/* emeter_rc.c
 *
 * A program to read from the SD16, convert to real-world units, then
 *   output to the UART and the LCD
 *
 * Written 26 April 2011 thru 7 May 2011 by Avery Sterk
 * for Calvin College Senior Design Team 01
 *
 * Hardware: Texas Instruments MSP430F47197, with
 *   * SynchroSystems LCD screen on S00..S39 and COM0..3
 *   * 10 uF capactor between LCDCAP and ground
 *   * Abracon ABLS 16.000 MHz oscillator as Xtal 2, ~6pF shunt caps
 *   * 32,768 Hz clock as Xtal 1, 12pF shunt caps
 *   * MAX233A RS232 UART on P2.4-5
 *   * 0.1uF capcitor between pins VREF and AVss
 *   * Current-sense network inputs on ADC channels 0,1,2
 *   * Voltage network input on ADC channels 3,4,5
 ******************************************************************************/

// how many seconds to wait before the backlight goes to sleep
#define BACKLIGHT_AWAKE 5
// declare how many meaurements to take before stopping
// as we use float, this can be large
#define COUNT_TO 120
#define COUNT_TO_FLOAT 120.0;
/** begin scale factors ******/
// irms: (measured_i_rms * 0.6V)/(measured_v_rms * 0x8000)
// here, 360mA *600mV / (6.9mVRMS * 0x8000) = 0.000955 A / unit
#define irms_scale_factor 0.000955
// vrms: (line_VRMS * 0.6V )/(net_VRMS * 0x8000)
// here, (120 V * 0.6V)/(0.185V * 0x8000) = 0.01188 V / unit
#define vrms_scale_factor 0.01188
/* power scale factor: compute via product of the other two */
// here, 0.0955mA * 0.01188 V * A / 1000mA = 0.0000011345
#define power_scale_factor 0.000011345
/**** end scale factors ******/
// number of hours per ACLK tick
#define energy_factor 0.000000008477 // 2^(-15)/3600 Hz


#include "msp430x471x7.h"
#include "lcd.h"
#include <math.h>
// remember, 16 bit ints and 32-bit longs
/* 32-bit floats */
volatile double p1vsum, p1isum, p1psum; // can be changed in an ISR
volatile double p2vsum, p2isum, p2psum;
volatile double p3vsum, p3isum, p3psum;
volatile unsigned int count; // 16-bit int
volatile float tmpf; // just a placeholder
volatile long tmpl; // just another placeholder
volatile short uptime_hours, uptime_minutes, uptime_seconds, backlight_state;
volatile char uptime_separator;

void print_units_to_lcd(double value,char units);
void double_fmt(double value, char* buf, int length);
void print_uart(char* str);

void main(void) {
    volatile unsigned int i;               // don't compile away, may change
    volatile unsigned char byte;
    float mean;
```

```
    double p1energy, p2energy, p3energy, total_energy;
    double rms;
    unsigned int time;
    char display_mode = 'P'; // statistic to display
    char buffer[256]; // buffer, you'd better not be able to overrun this.
    double ptotal;


    WDTCTL = WDTPW | WDTHOLD;                    // stop the watchdog
    FLL_CTL0 |= XCAP14PF;                        // configure capacitors, TI value
    FLL_CTL1 |= SELS;                            // select 16Mhz crystal for SMCLK

    /* wait for crystal to stabilize */
    do {
        IFG1 &= ~OFIFG;                    // clear Oscillator fault flag
        for (i = 0x47FF; i > 0; i--);      // wait for a while to stabilize
    }
    while ((IFG1 & OFIFG)); // while the oscillator fault keeps flagging

    /* initialize UART */
    P2SEL |= BIT4 + BIT5;                        // P2.4 and P2.5 are UART pins
    UCA0CTL1 |= UCSSEL_2;                        // Use SMCLK for UART timing
    UCA0BR0 = 138;                               // UART tick every 138 SMCLK ticks
    UCA0BR1 = 0x00;                                  // 16MHz / 15200Baud ~= 138
    UCA0MCTL = UCBRS_7;                          // modulation for 115200 baud

    /* initialize Synchro LCD */
    // control: ACLK/64 speed = 32768/64 = 512Hz, a 4-Mux display
    LCDACTL = LCDFREQ_32 | LCD4MUX | LCDSON; // TODO: Test using the 64 clock divider
        instead of 32
    LCDAVCTL0 = LCDCPEN;   // enable charge pump
    LCDAVCTL1 = VLCD_11;   // set output to 3.02V
    LCDAPCTL0 = 0xFF; // use all pins S00-S31
    LCDAPCTL1 = LCDS32 | LCDS36; // and S32-D39
    // Port1: LED output
   P1OUT = BIT2 + BIT4 + BIT5 + BIT6; // Signal LEDs off, backlight LED on
   P1DIR = BIT0 + BIT2 + BIT4 + BIT5 + BIT6;
   P1IES = 0x00;
   P1IE  = 0x00;
   P1SEL = 0x00;
    // Port4: LCD peripheral
   P4OUT = 0x00;
   P4DIR = 0x00;
   P4SEL = 0xFF; // port 4, all pins are controlled by the LCD_A controller
    // Port5: LCD peripheral
   P5SEL = BIT1 + BIT2 + BIT3 + BIT4; // all of these belong to the LCD_A device

    /* initialize SD16 ADC */
    SD16CTL = SD16REFON + SD16SSEL0 + SD16DIV_3;      // 1.2V reference, use SMCLK /3
        /* configure SD16's to use single conversion, 2's-complement form, and group */
    SD16CCTL0 |= SD16SNGL + SD16GRP + SD16DF; // group to next channel (CH 1)
    SD16CCTL1 |= SD16SNGL + SD16GRP + SD16DF; // group to next channel (CH 2)
    SD16CCTL2 |= SD16SNGL + SD16GRP + SD16DF; // group to next channel (CH 3)
    SD16CCTL3 |= SD16SNGL + SD16GRP + SD16DF; // group to next channel (CH 4)
//  SD16CCTL3 |= SD16SNGL + SD16IE + SD16DF; // interrupt when done
    SD16CCTL4 |= SD16SNGL + SD16GRP + SD16DF; // group to next channel (CH 5)
    SD16CCTL5 |= SD16SNGL + SD16IE + SD16DF; // interrupt when done
    for (i=0x3600; i > 0; i--);             // wait for reference to settle

    /* initialize timer */
    uptime_minutes = 0;
    uptime_hours = 0;
    uptime_seconds = 0;
    uptime_separator = ':';
    TACTL = TASSEL_1; // use ACLK, which is 32kHz.
    TBCCTL0 = CCIE;                             // CCR0 interrupt enabled
    TBCCR0 = 0x4000;
    TBCTL = TBSSEL_1 + MC_2 + ID_1;           // ACLK, continuous mode, / 2
```

```
130
        /* initialize variables */
        p1energy = 0; // never explicitly set elsewhere.
        p2energy = 0;
        p3energy = 0;
135
        /* turn LCD on */
        backlight_state = BACKLIGHT_AWAKE;
        LcdOn();
        LcdClear();
140     LcdShowStr("  00 W", LCD_PRIM_AREA, 0, STEADY);
        LcdShowStr("0000", LCD_SEC_AREA, 0, STEADY);
        LcdShowChar(uptime_separator, LCD_SEC_PUNCT_AREA, 0, STEADY);
        LcdShowChar(display_mode, LCD_MODE_AREA, 0, STEADY);

145     /* start UART */
        IE2 |= UCA0RXIE;                          // Enable USCI_A0 RX interrupt
        UCA0CTL1 &= ~UCSWRST;                     // UART state machine out of reset

        /********** MAIN BEHAVIOR LOOP *****************/
150     /***********************************************/
        while( UCA0RXBUF != 'q' ) { // until we receive 'q' over RS232
             /* init variables */
             count = 0;
             p1psum = p1vsum = p1isum = 0;
155          p2psum = p2vsum = p2isum = 0;
             p3psum = p3vsum = p3isum = 0;
             ptotal = 0;

             /* change display modes if the button is pressed */
160          i = 0;
             while(P2IN & BIT3) // debouncer for the button press
                  if (i < 10000)
                       i++;
                  else break; // done with this loop
165          if ( (backlight_state > 0) && (i > 8000) ) { // yes, you can release the button
                 faster than it will wait.
                  backlight_state = BACKLIGHT_AWAKE;
                  LcdClear();
                  switch (display_mode) {
                  case 'P':
170                    display_mode = 'E';
                       LcdShowStr("Wh", LCD_PRIM_AREA, 4, STEADY);
                       break;
                  case 'E':
                       display_mode = 'V';
175                    LcdShowStr(" V", LCD_PRIM_AREA, 4, STEADY);
                       break;
                  case 'V':
                       display_mode = 'I';
                       LcdShowStr("mA", LCD_PRIM_AREA, 4, STEADY);
180                    break;
                  case 'I': default:
                       display_mode = 'P';
                       LcdShowStr(" W", LCD_PRIM_AREA, 4, STEADY);
                       break;
185               } // end switch
                  LcdShowChar(display_mode, LCD_MODE_AREA, 0, STEADY);
             } else if (i > 8000) {// wake the backlight
                  backlight_state = BACKLIGHT_AWAKE; // reset the timer
                  P1OUT |= BIT2; // turn it back on
190          }

             /* Turn Timer On */
             TACTL |= MC_2;

195          /* activate peripherals */
             SD16CCTL5 |= SD16SC;                 // start SD16A conversion
```

```
            __bis_SR_register(LPM0_bits + GIE); // light sleep, with interrupts
            // SD16A  ISR will collect data into the "sum" variables
            // and    wake us back up where we went to sleep.

            /* wakeup routine */
            P1OUT += 0x70; // adds the blinky lights

            // output power to uart
            mean = p1psum * power_scale_factor / COUNT_TO_FLOAT;
            double_fmt(mean, buffer, 4);
            print_uart(buffer);
            ptotal += mean;
            // compute channel 1 energy
            TACTL |= MC_0; // turn off clock
            time = TAR; // read elapsed time
            p1energy += energy_factor * (mean * time); // compute energy
            TACTL &= TACLR; // clear the timer

            mean = p2psum * power_scale_factor / COUNT_TO_FLOAT;
            p2energy += energy_factor * (mean * time);
            while(!(IFG2 & UCA0TXIFG));
            UCA0TXBUF = ' '; // space to separate
            double_fmt(mean, buffer, 4);
            print_uart(buffer);
            ptotal += mean;


            mean = p3psum * power_scale_factor / COUNT_TO_FLOAT;
            p3energy += energy_factor * (mean * time);
            while(!(IFG2 & UCA0TXIFG));
            UCA0TXBUF = ' '; // space to separate
            double_fmt(mean, buffer, 4);
            print_uart(buffer);
            ptotal += mean;

            total_energy = p1energy + p2energy + p3energy;
            while(!(IFG2 & UCA0TXIFG));
            UCA0TXBUF = ' '; // space to separate
            double_fmt(ptotal, buffer, 4);
            print_uart(buffer);

            if ( display_mode == 'V' ) {
                 mean = p1vsum / COUNT_TO_FLOAT;
                 rms = vrms_scale_factor * sqrt(mean); // take the square root
                 print_units_to_lcd(rms,'V');
            } else if (display_mode == 'I' ) {
                 mean = p1isum / COUNT_TO_FLOAT;
                 rms = irms_scale_factor * sqrt(mean);
                 print_units_to_lcd(rms,'A');
            } else if (display_mode == 'E' ) {
                 print_units_to_lcd(total_energy,'E');
            } else { // assume power
                 print_units_to_lcd(ptotal,'W');
            }

            /* end this line of text */
            while(!(IFG2 & UCA0TXIFG));
            UCA0TXBUF = 0x0A;                       // "newline"
            while(!(IFG2 & UCA0TXIFG));
            UCA0TXBUF = 0x0D;                       // "carriage return"
            _NOP(); // breakpoint here if you need it
       } // end while condition; shut things down
       LcdClear();
       LcdOff();
       P1OUT = 0x70; // turn off the signal LEDs
       while(!(IFG2 & UCA0TXIFG));
       __bis_SR_register(LPM3_bits); // go into a deep sleep, no interruptions
} // end void main()
```

```
265
/* format a measurment with the given units for the LCD screen
 * Receives: value, a 32-bit float; units, a character for the units
 * Effect: determines the range of value, then outputs three digits of
 *         the value in a way fit for the LCD screen:
270 *         with XYZ digits, p an SI prefix, and u the unit character,
 *         either: X.YZ pu
 *         or:     XYZ pu
 *         whichever fits the 10^3n better.   */
void print_units_to_lcd(double value, char units) {
275     long long_prefix = 0;
     long long_suffix = 0;
     int prefix_order = 0;
     int three_digits = 0;
     char decimal = ' '; // space means no decimal, '.' means decimal.
280     /* negative sign */
     if (value < 0 ) { // are we negative?
          LcdShowChar('-',LCD_PRIM_AREA,0,STEADY); // if so, show the negative sign
          value = - value; // take the absolute value
     } else LcdShowChar(' ',LCD_PRIM_AREA,0,STEADY); // if not, clear it
285
     /* figure out the output using long values */
     long_prefix = value; // implicit ftoi
     long_suffix = (value - long_prefix) * 1000000; // should give us 6 decimal places
     // figure out which prefix to use
290     if (long_prefix == 0) { // everything is in decimal places
          if ( long_suffix >= 1000 ) { // >= 1 milli-unit
               prefix_order = -1;
               if (long_suffix >= 10000) // >= 10 milli-units, cannot use decimal
                    three_digits = long_suffix / 1000;
295                else {
                    three_digits = long_suffix / 10;
                    decimal = '.';
               }
          } else {
300                prefix_order = -2;
               three_digits = long_suffix; // < 1000, so we're good
          }
     } else { // we have something > 1 unit
          if (long_prefix < 10) { // mix units and decimals
305                three_digits = 100*long_prefix + (long_suffix / 10000); // 1 lead, 2 follow
                    digits
               decimal = '.';
          } else if (long_prefix < 1000)
               three_digits = long_prefix;
          else {
310                prefix_order = 1;
                if (long_prefix < 10000) { // use decimal point with kilo-units
                    decimal = '.';
                    three_digits = long_prefix / 10;
                } else // no decimal point with kilo-units
315                    three_digits = long_prefix / 1000;
          }
     }
     LcdShowChar(units,LCD_PRIM_AREA,5,STEADY); // show units
     LcdShowChar(decimal,LCD_PRIM_PUNCT_AREA,0,STEADY); // decimal
320     switch (prefix_order) { // write prefix
     case 0: LcdShowChar(' ',LCD_PRIM_AREA,4,STEADY); break;
     case 1: LcdShowChar('k',LCD_PRIM_AREA,4,STEADY); break;
     case -1: LcdShowChar('m',LCD_PRIM_AREA,4,STEADY); break;
     case -2: LcdShowChar('u',LCD_PRIM_AREA,4,STEADY); break;
325     default: LcdShowChar('?',LCD_PRIM_AREA,4,STEADY); break;
     } // end switch
     if (three_digits >= 1000) { // something went wrong
          LcdShowStr("???",LCD_PRIM_AREA,1,STEADY);
     }
330     if (three_digits >= 100) {
          LcdShowChar(0x30 + (three_digits / 100), LCD_PRIM_AREA,1,STEADY);
```

```
            three_digits %= 100;
            LcdShowChar(0x30 + (three_digits / 10), LCD_PRIM_AREA,2,STEADY);
            LcdShowChar(0x30 + (three_digits % 10), LCD_PRIM_AREA,3,STEADY);
      } else { // < 100
            LcdShowChar(' ',LCD_PRIM_AREA,1,STEADY);
            if (three_digits >= 10) // output the 10's place
                  LcdShowChar(0x30 + (three_digits / 10), LCD_PRIM_AREA,2,STEADY);
            else
                  LcdShowChar(' ', LCD_PRIM_AREA,2,STEADY);
            LcdShowChar(0x30 + (three_digits % 10), LCD_PRIM_AREA,3,STEADY);
      }

}

/* double_fmt gives a c-string representation of a double */
void double_fmt(double value, char* buf, int want_places) {
      int place = 0;
      long long_value = 0;
      long tens_place = 1;
      int num_pre_decimal = 0;
      char byte = '\0';
      if ( value < 0 ) {
            buf[place++] = '-';
            value = - value; // invert the value
      }
      long_value = value; // implicit ftoi;
      // figure out how many places it is
      while ( long_value >= tens_place ) {
            tens_place *= 10;
            num_pre_decimal++;
      }
      while ( num_pre_decimal >= 0 ) {
            byte = 0x30 + ( long_value / tens_place );
            buf[place++] = byte;
            long_value %= tens_place;
            tens_place /= 10;
            num_pre_decimal--;
            want_places--;
      }
      if ( want_places > 0 ) {
            buf[place++] = '.';
      }
      long_value = 1000000 * value;
      long_value %= 1000000; // up to six places after the decimal!
      tens_place = 100000;
      if ( want_places > 6 ) {
            want_places = 6; // nothing more than six more places
      }
      while ( want_places > 0 ) {
            byte = 0x30 + (long_value / tens_place);
            buf[place++] = byte;
            long_value %= tens_place;
            tens_place /= 10;
            want_places--;
      }
      buf[place] = '\0'; // null-terminate
}

void print_uart(char * str) {
      while (*str != '\0') { // until we reach the null character,
            while(!(IFG2 & UCA0TXIFG)); // wait for the transmission to finish
            UCA0TXBUF = *str; // send the current character
            *str = '\0'; // clear this character
            str++; // next character
      }
      while(!(IFG2 & UCA0TXIFG)); // wait for the transmission to finish
}
```

```
/* SD16 ADC interrupt service routine */
#pragma vector=SD16A_VECTOR
__interrupt void SD16AISR(void) { // service the SD16A interrupt
     // just in case we don't want to waste time on the stack,
        // use register variables
     register int current;
     register int voltage;
     switch (SD16IV) { // who caused the interrupt?
          default: break;
          case 2: break; // SD16AMEM overflow, change your input?
          case 4: case 6: case 8: case 10: case 12: case 14:
          // SD16 done converting, save the value
               SD16CCTL5 &= ~SD16SC; // shut off the SD16
               /**** phase 1 ****/
               current = SD16MEM3; // implicitly clears any interrupt flag
               p1isum += (long) current * current; // square, change to float, add to sum
               voltage = SD16MEM0;
               p1vsum += (long) voltage * voltage;
               p1psum += (long) voltage * current;
               /**** phase 2 ****/
               current = SD16MEM4;
               p2isum += (long) current * current;
               voltage = SD16MEM1;
               p2vsum += (long) voltage * voltage;
               p2psum += (long) voltage * current;
               /**** phase 3 ****/
               current = SD16MEM5;
               p3isum += (long) current * current;
               voltage = SD16MEM2;
               p3vsum += (long) voltage * voltage;
               p3psum += (long) voltage * current;
               /* add to the count, ensure we cleared the interrupt */
               count ++; // increment count
               SD16CCTL5 &= ~SD16IFG; // hang the interrupt manually
               break;
     }
     if ( count >= COUNT_TO ) {
          __bic_SR_register_on_exit(LPM0_bits); // wake the processor from sleep
     } else {
          SD16CCTL5 |= SD16SC; // turn on the SD16
     }
} // end SD16A ISR

/* Timer B ISR */
#pragma vector=TIMERB0_VECTOR
__interrupt void Timer_B (void)
{
     int tmp;
     uptime_seconds++;
     if ( uptime_seconds == 60) {
          uptime_seconds = 0;
          uptime_minutes++;
     }
     if (uptime_minutes == 60) {
          uptime_minutes = 0;
          uptime_hours++;
     }
     uptime_hours %= 24; // if 24 or more, reduce to less than 24
     TBCCR0 += 0x4000;
     uptime_separator ^= 0x1A; // ':' to ' ' and vice-versa
     LcdShowChar(uptime_separator,LCD_SEC_PUNCT_AREA,0,STEADY);

     LcdShowChar(0x30 + (uptime_minutes % 10),LCD_SEC_AREA,3,STEADY);
     LcdShowChar(0x30 + (uptime_minutes / 10),LCD_SEC_AREA,2,STEADY);
     LcdShowChar(0x30 + (uptime_hours % 10),LCD_SEC_AREA,1,STEADY);
     LcdShowChar(0x30 + (uptime_hours / 10),LCD_SEC_AREA,0,STEADY);
     /* dial settings */
     tmp = (2 << (uptime_seconds / 5)) - 1; // add a hand to the dial every 5 seconds
```

```
    LcdShowSegs(tmp, LCD_DIAL_AREA, 0, STEADY);

    if (backlight_state > 0)
        backlight_state--;
    if (backlight_state == 0) // if it's now "asleep"
        P1OUT &= ~BIT2; // turn off the LCD backlight
}
```

C Smart Breaker Appendix

Listing 6: Software Code for Arduino Uno

```
/*
This is code for using
the Arduino Uno microcontroller to read (and write)
to the ADE7763 energy metering chip, via SPI

Based off the example provided at
http://arduino.cc/en/Tutorial/BarometricPressureSensor
for using SPI to read a pressure sensor

Connections:
CS: pin 10 :ADE pin 17
MOSI: pin 11 :ADE pin 20
MISO: pin 12 :ADE pin 19
SCK: pin 13 :ADE pin 18


Nathan Jen
06 May 2011
Engineering 340 Sr. Design Project Team PICA
*/

#include <SPI.h>                      // include the SPI library

// ADE7763 memory register addresses
const int chan1 = 0x16;               // address for channel 1, RMS current
const int chan2 = 0x17;                    // address for channel 2, RMS voltage
const byte READ = 00000000;                // read command given to communication register,
    page 50
const byte WRITE = 10000000;                  // write command given to communication
    register, page 50

// Other important values
const int currentThresh = 2;                  // set the limit for allowable current - pick a
     value, just for testing
      int voltageData = 0;                         // initialize to a known value - mostly
          for testing
      int currentData = 0;

// Any pins needed to connect master & slave
const int slaveSelect = 10;
//const int solidState = 6;               // use this pin to activate the SSR control line
    once reading is happening

// Setup
void setup() {
  Serial.begin( 9600 );                   // open serial port at 9600 bps

  SPI.begin();                            // start SPI library

  SPI.setBitOrder( MSBFIRST );
  SPI.setDataMode( SPI_MODE3 );                   // spi_mode3 = high polarity and rising
      clock
  pinMode( slaveSelect, OUTPUT );                 // digital pot example still uses this
//pinMode( solidState, OUTPUT );              // commented out until we can read
//analogWrite( solidState, 0 );                   // set the solidState pin to a default "
    off" value - commented out for same reason as above
  writeRegister( 0x0A, 0x0, 0x0 );            // disable interrupts
//writeRegister( 0x09, )              // select sample rates, filter enable & calibration;
    commented out b/c default looks acceptable
  delay( 100 );                       // allow ADE to set up as necessary
```

```
  Serial.print( "Starting Voltage is " );         // this data would be sent to the base
      station
  Serial.print( voltageData );
  Serial.print( "\n" );
  Serial.print( "Starting Current is " );         // as would this data
  Serial.print( currentData );
  Serial.print( "\n" );
}


/*
// use this loop for testing
void loop() {

unsigned testing = readRegister( 0x16 );
  Serial.print( "\n \n Vrms is " );
  Serial.print( testing );

  delay( 1000 );
}
*/


// Do Reading Here
void loop() {

unsigned voltageData = readRegister( 0x17 );   // unsigned b/c that's what the register is
    formated as, should be RMS value already by nature of the register
  Serial.print( "\n Voltage is " );
  Serial.print( voltageData );                // print to the effective Base Station
  Serial.print( "\n" );
  delay( 1000 );                                // 1000 because we can read straight off
      the console at that speed

unsigned currentData = readRegister( 0x16 );
  Serial.print( "\n Current is " );
  Serial.print( currentData );                  // print to the 'Base Station'
  Serial.print( "\n" );
  delay( 1000 );

  if( currentData < currentThresh ) {
  //analogWrite( solidState, 255 );             // allow SSR to conduct
    Serial.print( "Current OK" );
    Serial.print( "\n" );
    Serial.print( "\n" );
  } else {
  //analogWrite( solidState, 0 );                 // the shut-off commmand
    Serial.print( "WARNING: CURRENT EXCEEDS ALLOWABLE LIMIT" );
    Serial.print( "\n" );
    Serial.print( "\n" );

  }

}



/***************** FUNCTION DEFINITIONS *******************************/

// Read function, without specifying how many bytes we get back
// all of the Serial.print() commands were used for debugging
  unsigned int readRegister( byte thisRegister ) {
  int result = 0x1;
//   Serial.print( "\n Result: " );
//   Serial.print( result );
  SPI.setDataMode( SPI_MODE3 );
  digitalWrite( slaveSelect, LOW );
  byte dataToSend = READ | thisRegister;
```

```
  int printDataToSend = int( dataToSend );
   Serial.print( "\n dataToSend: " );
   Serial.print( printDataToSend );
  delay( 10 );                                    // give it a little time to actually
      transfer all the data
  result = SPI.transfer( dataToSend );
   Serial.print( "\n First result: " );
   Serial.print( result );

  // get the second byte and add to the result
  int second1Result = result << 8;
   Serial.print( "\n Shifted result: " );
   Serial.print( second1Result );

  int result2 = SPI.transfer( 0x00 );
   Serial.print( "\n Second result: " );
   Serial.print( result2 );

  int second2Result = second1Result | result2;
   Serial.print( "\n Or-ed: " );
   Serial.print( second2Result );


  // get the third byte & add to the result - this is the last shift needed since the
      registers are 24 bits
  int third1Result = second2Result << 8;
   Serial.print( "\n Shifted result2: " );
   Serial.print( third1Result );

  int result3 = SPI.transfer( 0x00 );
   Serial.print( "\n Third result: " );
   Serial.print( result3 );

  int third2Result = third1Result | result3;
   Serial.print( "\n Final result: " );
   Serial.print( third2Result );

  SPI.setDataMode( SPI_MODE2 );
  digitalWrite( slaveSelect, HIGH );
  return( second2Result );
}


// Write function, as defined by Nathan Jen
void writeRegister( byte thisRegister, byte msbValue, byte lsbValue ) {
  byte destination = WRITE | thisRegister;          // combine the write command and the
      specified register
  digitalWrite( slaveSelect, LOW );               // select the chip
  SPI.transfer( destination );                    // send destination to communication
      register
  SPI.transfer( msbValue );                  // send the first half of the data
  SPI.transfer( lsbValue );                  // send the second half of the data
  digitalWrite( slaveSelect, HIGH );              // deselect the chip
}


/****************** EXAMPLE CODE ***************************************/

/* Read from or write to register from the SCP1000:
// Taken directly from example code

unsigned int readRegister(byte thisRegister, int bytesToRead ) {
  byte inByte = 0;           // incoming byte from the SPI
  unsigned int result = 0;   // result to return
  Serial.print(thisRegister, BIN);
  Serial.print("\t");
  // SCP1000 expects the register name in the upper 6 bits
```

```
  // of the byte. So shift the bits left by two bits:
  thisRegister = thisRegister << 2;
  // now combine the address and the command into one byte
  byte dataToSend = thisRegister & READ;
  Serial.println(thisRegister, BIN);
  // take the chip select low to select the device:
  digitalWrite(chipSelectPin, LOW);
  // send the device the register you want to read:
  SPI.transfer(dataToSend);
  // send a value of 0 to read the first byte returned:
  result = SPI.transfer(0x00);
  // decrement the number of bytes left to read:
  bytesToRead--;
  // if you still have another byte to read:
  if (bytesToRead > 0) {
    // shift the first byte left, then get the second byte:
    result = result << 8;
    inByte = SPI.transfer(0x00);
    // combine the byte you just got with the previous one:
    result = result | inByte;
    // decrement the number of bytes left to read:
    bytesToRead--;
  }
  // take the chip select high to de-select:
  digitalWrite(chipSelectPin, HIGH);
  // return the result:
  return(result);
}


//Sends a write command to SCP1000
// This also directly from example code
void writeRegister(byte thisRegister, byte thisValue) {

  // SCP1000 expects the register address in the upper 6 bits
  // of the byte. So shift the bits left by two bits:
  thisRegister = thisRegister << 2;
  // now combine the register address and the command into one byte:
  byte dataToSend = thisRegister | WRITE;

  // take the chip select low to select the device:
  digitalWrite(chipSelectPin, LOW);

  SPI.transfer(dataToSend); //Send register location
  SPI.transfer(thisValue);  //Send value to record into register

  // take the chip select high to de-select:
  digitalWrite(chipSelectPin, HIGH);
}
*/ // end of their read and write commands given in example code
```

D Base Station Appendix

D.1 LEON3 Hardware Linux

Listing 7: GRMON Hardware Listing

```
GRMON LEON debug monitor v1.1.46 evaluation version

Copyright (C) 2004,2005 Gaisler Research - all rights reserved.
For latest updates, go to http://www.gaisler.com/
Comments or bug-reports to support@gaisler.com

This evaluation version will expire on 28/9/2011
Xilinx cable: Cable type/rev : 0x3
JTAG chain: xc5vlx110t xccace xc95144xl xcf32p xcf32p

Device ID: : 0x509
GRLIB build version: 4104

initialising
detected frequency:  80 MHz

Component                            Vendor
LEON3 SPARC V8 Processor             Gaisler Research
AHB Debug UART                       Gaisler Research
AHB Debug JTAG TAP                   Gaisler Research
SVGA Controller                      Gaisler Research
GR Ethernet MAC                      Gaisler Research
DDR2 Controller                      Gaisler Research
AHB/APB Bridge                       Gaisler Research
LEON3 Debug Support Unit             Gaisler Research
LEON2 Memory Controller              European Space Agency
System ACE I/F Controller            Gaisler Research
Generic APB UART                     Gaisler Research
Multi-processor Interrupt Ctrl       Gaisler Research
Modular Timer Unit                   Gaisler Research
PS/2 interface                       Gaisler Research
PS/2 interface                       Gaisler Research
General purpose I/O port             Gaisler Research
AMBA Wrapper for OC I2C-master       Gaisler Research
AMBA Wrapper for OC I2C-master       Gaisler Research
AHB status register                  Gaisler Research

Use command 'info sys' to print a detailed report of attached cores


Grmon> info sys
00.01:003   Gaisler Research  LEON3 SPARC V8 Processor (ver 0x0)
             ahb master 0
01.01:007   Gaisler Research  AHB Debug UART (ver 0x0)
             ahb master 1
             apb: 80000700 - 80000800
             baud rate 115200, ahb frequency 80.00
02.01:01c   Gaisler Research  AHB Debug JTAG TAP (ver 0x1)
             ahb master 2
```

```
03.01:063   Gaisler Research  SVGA Controller (ver 0x0)
             ahb master 3
             apb: 80000600 - 80000700
             clk0: 25.00 MHz  clk1: 25.00 MHz  clk2: 40.00 MHz  clk3: 65.00 MHz
04.01:01d   Gaisler Research  GR Ethernet MAC (ver 0x0)
             ahb master 4, irq 12
             apb: 80000b00 - 80000c00
             Device index: dev0
             edcl ip 192.168.0.52, buffer 2 kbyte
00.01:02e   Gaisler Research  DDR2 Controller (ver 0x0)
             ahb: 40000000 - 60000000
             ahb: fff00100 - fff00200
             64-bit DDR2 : 1 * 256 Mbyte @ 0x40000000, 4 internal banks
                           190 MHz, col 10, ref 7.8 us, trfc 131 ns
01.01:006   Gaisler Research  AHB/APB Bridge (ver 0x0)
             ahb: 80000000 - 80100000
02.01:004   Gaisler Research  LEON3 Debug Support Unit (ver 0x1)
             ahb: 90000000 - a0000000
             AHB trace 128 lines, 32-bit bus, stack pointer 0x4ffffff0
             CPU#0 win 8, hwbp 2, itrace 128, V8 mul/div, srmmu, lddel 1, GRFPU-lite
                   icache 2 * 8 kbyte, 32 byte/line lru
                   dcache 1 * 8 kbyte, 16 byte/line
03.04:00f   European Space Agency  LEON2 Memory Controller (ver 0x1)
             ahb: 00000000 - 20000000
             ahb: 20000000 - 40000000
             ahb: c0000000 - c2000000
             apb: 80000000 - 80000100
             16-bit prom @ 0x00000000
04.01:067   Gaisler Research  System ACE I/F Controller (ver 0x0)
             irq 13
             ahb: fff00200 - fff00300
01.01:00c   Gaisler Research  Generic APB UART (ver 0x1)
             irq 2
             apb: 80000100 - 80000200
             baud rate 38461, DSU mode (FIFO debug)
02.01:00d   Gaisler Research  Multi-processor Interrupt Ctrl (ver 0x3)
             apb: 80000200 - 80000300
03.01:011   Gaisler Research  Modular Timer Unit (ver 0x0)
             irq 8
             apb: 80000300 - 80000400
             8-bit scaler, 2 * 32-bit timers, divisor 80
04.01:060   Gaisler Research  PS/2 interface (ver 0x2)
             irq 4
             apb: 80000400 - 80000500
05.01:060   Gaisler Research  PS/2 interface (ver 0x2)
             irq 5
             apb: 80000500 - 80000600
08.01:01a   Gaisler Research  General purpose I/O port (ver 0x1)
             apb: 80000800 - 80000900
09.01:028   Gaisler Research  AMBA Wrapper for OC I2C-master (ver 0x2)
             irq 14
             apb: 80000900 - 80000a00
           Controller index for use in GRMON: 1
0c.01:028   Gaisler Research  AMBA Wrapper for OC I2C-master (ver 0x2)
             irq 11
             apb: 80000c00 - 80000d00
           Controller index for use in GRMON: 2
0f.01:052   Gaisler Research  AHB status register (ver 0x0)
             irq 7
```

```
            apb:  80000f00 - 80001000

Grmon>
```

D.2 Linux Log

Listing 8: Linux Log

```
=~=~=~=~=~=~=~=~=~=~=~=~=~= PuTTY log 2011.02.27 23:59:45 =~=~=~=~=~=~=~=~=~=~=~=~=~=
Booting Linux
Booting Linux...
PROMLIB: Sun Boot Prom Version 0 Revision 0

Linux version 2.6.21.1 (avery@atlantis) (gcc version 3.4.4) #2 Sun Feb 27 23:56:21 EST
    2011

ARCH: LEON

TYPE: Leon2/3 System-on-a-Chip

Ethernet address: 0:0:0:0:0:0

CACHE: direct mapped cache, set size 8k

Boot time fixup v1.6. 4/Mar/98 Jakub Jelinek (jj@ultra.linux.cz). Patching kernel for
    srmmu[Leon2]/iommu

64MB HIGHMEM available.

Nocache: 0xfc000000-0xfc400000, 1024 pages [128-1280]

node 2: /cpu00 (type:cpu) (props:.node device_type mid mmu-nctx clock-frequency
    uart1_baud uart2_baud )

node 3: /a: (type:serial) (props:.node device_type name )

node 4: /ambapp0 (type:ambapp) (props:.node device_type name )

PROM: Built device tree from rootnode 1 with 1478 bytes of memory.

DEBUG: psr.impl = 0xf  fsr.vers = 0x7

Built 1 zonelists.  Total pages: 64133

Kernel command line: console=ttyS0,38400 rdinit=/sbin/init

PID hash table entries: 1024 (order: 10, 4096 bytes)

Todo: init master_l10_counter

Attaching grlib apbuart serial drivers (clk:80hz):

Console: colour dummy device 80x25

Dentry cache hash table entries: 32768 (order: 5, 131072 bytes)

Inode-cache hash table entries: 16384 (order: 4, 65536 bytes)

pkbase: 0xfc800000 pkend: 0xfcc00000 fixstart 0xfcff1000
Memory: 252096k/262144k available (1516k kernel code, 9920k reserved, 168k data, 1732k
    init, 65536k highmem)

Mount-cache hash table entries: 512
```

```
NET: Registered protocol family 16
NET: Registered protocol family 2
IP route cache hash table entries: 2048 (order: 1, 8192 bytes)
TCP established hash table entries: 8192 (order: 4, 65536 bytes)
TCP bind hash table entries: 8192 (order: 3, 32768 bytes)
TCP: Hash tables configured (established 8192 bind 8192)
TCP reno registered
leon: power management initialized
highmem bounce pool size: 64 pages
io scheduler noop registered
io scheduler cfq registered (default)
grlib apbuart: 1 serial driver(s) at [0x80000100(irq 2)]
grlib apbuart: system frequency: 80000 khz, baud rates: 38400 38400
ttyS0 at MMIO 0x80000100 (irq = 2) is a Leon
Testing fifo size for UART port 0: got 4 bytes.
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
loop: loaded (max 8 devices)
xsysace xsysace.0: Xilinx SystemACE revision 1.0.12
xsysace xsysace.0: capacity: 3906560 sectors
 xsa: xsa1 xsa2
Xilinx SystemACE device driver, major=254
Probing GRETH Ethernet Core at 0x80000b00
Detected MARVELL 88EE1111 Revision 2
10/100 GRETH Ethermac at [0x80000b00] irq 12. Running 100 Mbps full duplex
TCP cubic registered
NET: Registered protocol family 10
IPv6 over IPv4 tunneling driver
Freeing unused kernel memory: 1732k freed

init started: BusyBox v1.8.2 (2011-02-27 23:11:31 EST)
```

```
starting pid 14, tty '': '/etc/init.d/rcS'
mount: mounting tmpfs on /var/tmp failed: Invalid argument

starting pid 25, tty '': '/bin/sh'
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0A:35:01:D2:66
          inet addr:192.168.0.80  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::20a:35ff:fe01:d266/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Base address:0xb00

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

/ # echo "Now I"m pl      'm plugging in the E ethernet"
Now I'm plugging in the ethernet
/ #
/ # echo "Now I'm plugging in the ethernet"
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0A:35:01:D2:66
          inet addr:192.168.0.80  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::20a:35ff:fe01:d266/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Base address:0xb00

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

/ # ifconfig eht0   th0 down
/ #
/ # ifconfig eth0 down    up
/ #
/ # ifconfig eth0 up
/ # ifconfig eth0 down
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0A:35:01:D2:66
          inet addr:192.168.0.80  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::20a:35ff:fe01:d266/64 Scope:Link
```

```
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:9 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Base address:0xb00

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

/ # dh
/ # dh
dhcprelay  dhrystone

/ # dh  exit

process '/bin/sh' (pid 25) exited. Scheduling it for restart.

starting pid 33, tty '': '/bin/sh'
/ #
```

D.3 DHClient with Busybox Log

Listing 9: DHClient with Busybox

```
=~=~=~=~=~=~=~=~=~=~=~=~= PuTTY log 2011.03.08 13:45:09 =~=~=~=~=~=~=~=~=~=~=~=~=
Booting Linux
Booting Linux...
PROMLIB: Sun Boot Prom Version 0 Revision 0

Linux version 2.6.21.1 (avery@atlantis) (gcc version 3.4.4) #13 Mon Mar 7 17:44:18 EST
    2011

ARCH: LEON

Attaching textvga drivers [0xf01fcf34 s:0x1fe0]: ##! error GAISLER_VGA(0x61) not found

TYPE: Leon2/3 System-on-a-Chip

Ethernet address: 0:0:0:0:0:0

CACHE: direct mapped cache, set size 8k

Boot time fixup v1.6. 4/Mar/98 Jakub Jelinek (jj@ultra.linux.cz). Patching kernel for
    srmmu[Leon2]/iommu

64MB HIGHMEM available.

Nocache: 0xfc000000-0xfc400000, 1024 pages [128-1280]

node 2: /cpu00 (type:cpu) (props:.node device_type mid mmu-nctx clock-frequency
    uart1_baud uart2_baud )

node 3: /a: (type:serial) (props:.node device_type name )

node 4: /ambapp0 (type:ambapp) (props:.node device_type name )

PROM: Built device tree from rootnode 1 with 1478 bytes of memory.

DEBUG: psr.impl = 0xf  fsr.vers = 0x7

Built 1 zonelists.  Total pages: 64485

Kernel command line: console=ttyS0,38400 root=254:1 rw init=/sbin/init

PID hash table entries: 1024 (order: 10, 4096 bytes)

Todo: init master_l10_counter

Attaching grlib apbuart serial drivers (clk:80hz):

Console: colour dummy device 80x25

Dentry cache hash table entries: 32768 (order: 5, 131072 bytes)

Inode-cache hash table entries: 16384 (order: 4, 65536 bytes)

pkbase: 0xfc800000 pkend: 0xfcc00000 fixstart 0xfcff1000
Memory: 253504k/262144k available (1656k kernel code, 8512k reserved, 204k data, 136k
    init, 65536k highmem)
```

```
Mount-cache hash table entries: 512
NET: Registered protocol family 16
Generic PHY: Registered new driver
SCSI subsystem initialized
NET: Registered protocol family 2
IP route cache hash table entries: 2048 (order: 1, 8192 bytes)
TCP established hash table entries: 8192 (order: 4, 65536 bytes)
TCP bind hash table entries: 8192 (order: 3, 32768 bytes)
TCP: Hash tables configured (established 8192 bind 8192)
TCP reno registered
leon: power management initialized
highmem bounce pool size: 64 pages
io scheduler noop registered
io scheduler cfq registered (default)
GRVGA: Defaulting to 640x480 8-bit resolution
Framebuffer address from node : 0xf0980000
Framebuffer address from videomemory : 0x40980000
fb0: Gaisler frame buffer device, using 300K of video memory
grlib apbuart: 1 serial driver(s) at [0x80000100(irq 2)]
grlib apbuart: system frequency: 80000 khz, baud rates: 38400 38400
ttyS0 at MMIO 0x80000100 (irq = 2) is a Leon
Testing fifo size for UART port 0: got 4 bytes.
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
loop: loaded (max 8 devices)
xsysace xsysace.0: Xilinx SystemACE revision 1.0.12
xsysace xsysace.0: capacity: 3906560 sectors
 xsa: xsa1 xsa2
Xilinx SystemACE device driver, major=254
Marvell 88E1101: Registered new driver
```

```
Marvell 88E1111: Registered new driver
Marvell 88E1145: Registered new driver
Probing GRETH Ethernet Core at 0x80000b00
Detected MARVELL 88EE1111 Revision 2
10/100 GRETH Ethermac at [0x80000b00] irq 12. Running 100 Mbps full duplex
gr_udc-gr_probe(2154): could not find amba slave with id's 00000001 00000021
Attaching grlib ps2 mouse drivers at 0x80000400, irq: 4
Attaching grlib ps2 keyboard drivers at 0x80000500, irq: 5
Netfilter messages via NETLINK v0.30.
ip_tables: (C) 2000-2006 Netfilter Core Team
TCP cubic registered
NET: Registered protocol family 17
VFS: Mounted root (ext2 filesystem).
Freeing unused kernel memory: 136k freed

init started: BusyBox v1.8.2 (2011-03-06 14:47:37 EST)
starting pid 15, tty '': '/etc/init.d/rcS'
starting pid 26, tty '': '/bin/sh'
/ # dhcline  ent eth0
Internet Software Consortium DHCP Client 2.0pl5
Copyright 1995, 1996, 1997, 1998, 1999 The Internet Software Consortium.
All rights reserved.

Please contribute if you find this software useful.
For info, please visit http://www.isc.org/dhcp-contrib.html

sh: cannot exec 'if': No such file or directory
sh: cannot exec 'then': No such file or directory
sh: cannot open '/etc/dhclient-exit-hooks'
sh: cannot exec 'fi': No such file or directory
sh: cannot exec '-r``': No such file or directory
sh: cannot exec ':': No such file or directory
sed: bad option in substitution expression
sed: bad option in substitution expression
[: 2: unknown operand
[: 2: unknown operand
sh: cannot exec 'exit_with_hooks': No such file or directory
sh: syntax error
Listening on LPF/eth0/00:0a:35:01:d2:66
Sending on   LPF/eth0/00:0a:35:01:d2:66
Sending on   Socket/fallback/fallback-net
DHCPREQUEST on eth0 to 255.255.255.255 port 67
DHCPACK from 192.168.1.1
```

```
sh: cannot exec 'if': No such file or directory
sh: cannot exec 'then': No such file or directory
sh: cannot open '/etc/dhclient-exit-hooks'
sh: cannot exec 'fi': No such file or directory
sh: cannot exec '-r``': No such file or directory
sh: cannot exec ':': No such file or directory
sed: bad option in substitution expression
sed: bad option in substitution expression
sh: syntax error
bound to 192.168.1.118 -- renewal in 43200 seconds.
```

D.4 DHClient with BASH Log

Listing 10: DHClient with BASH

```
=~=~=~=~=~=~=~=~=~=~=~=~= PuTTY log 2011.03.08 15:31:48 =~=~=~=~=~=~=~=~=~=~=~=~=
Booting Linux
Booting Linux...
PROMLIB: Sun Boot Prom Version 0 Revision 0

Linux version 2.6.21.1 (avery@atlantis) (gcc version 3.4.4) #13 Mon Mar 7 17:44:18 EST
    2011

ARCH: LEON

Attaching textvga drivers [0xf01fcf34 s:0x1fe0]: ##! error GAISLER_VGA(0x61) not found

TYPE: Leon2/3 System-on-a-Chip

Ethernet address: 0:0:0:0:0:0

CACHE: direct mapped cache, set size 8k

Boot time fixup v1.6. 4/Mar/98 Jakub Jelinek (jj@ultra.linux.cz). Patching kernel for
    srmmu[Leon2]/iommu

64MB HIGHMEM available.

Nocache: 0xfc000000-0xfc400000, 1024 pages [128-1280]

node 2: /cpu00 (type:cpu) (props:.node device_type mid mmu-nctx clock-frequency
    uart1_baud uart2_baud )

node 3: /a: (type:serial) (props:.node device_type name )

node 4: /ambapp0 (type:ambapp) (props:.node device_type name )

PROM: Built device tree from rootnode 1 with 1478 bytes of memory.

DEBUG: psr.impl = 0xf  fsr.vers = 0x7

Built 1 zonelists.  Total pages: 64485

Kernel command line: console=ttyS0,38400 root=254:1 rw init=/sbin/init

PID hash table entries: 1024 (order: 10, 4096 bytes)

Todo: init master_l10_counter

Attaching grlib apbuart serial drivers (clk:80hz):

Console: colour dummy device 80x25

Dentry cache hash table entries: 32768 (order: 5, 131072 bytes)

Inode-cache hash table entries: 16384 (order: 4, 65536 bytes)

pkbase: 0xfc800000 pkend: 0xfcc00000 fixstart 0xfcff1000
Memory: 253504k/262144k available (1656k kernel code, 8512k reserved, 204k data, 136k
    init, 65536k highmem)
```

```
Mount-cache hash table entries: 512
NET: Registered protocol family 16
Generic PHY: Registered new driver
SCSI subsystem initialized
NET: Registered protocol family 2
IP route cache hash table entries: 2048 (order: 1, 8192 bytes)
TCP established hash table entries: 8192 (order: 4, 65536 bytes)
TCP bind hash table entries: 8192 (order: 3, 32768 bytes)
TCP: Hash tables configured (established 8192 bind 8192)
TCP reno registered
leon: power management initialized
highmem bounce pool size: 64 pages
io scheduler noop registered
io scheduler cfq registered (default)
GRVGA: Defaulting to 640x480 8-bit resolution
Framebuffer address from node : 0xf0980000
Framebuffer address from videomemory : 0x40980000
fb0: Gaisler frame buffer device, using 300K of video memory
grlib apbuart: 1 serial driver(s) at [0x80000100(irq 2)]
grlib apbuart: system frequency: 80000 khz, baud rates: 38400 38400
ttyS0 at MMIO 0x80000100 (irq = 2) is a Leon
Testing fifo size for UART port 0: got 4 bytes.
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
loop: loaded (max 8 devices)
xsysace xsysace.0: Xilinx SystemACE revision 1.0.12
xsysace xsysace.0: capacity: 3906560 sectors
 xsa: xsa1 xsa2
Xilinx SystemACE device driver, major=254
Marvell 88E1101: Registered new driver
```

```
Marvell 88E1111: Registered new driver
Marvell 88E1145: Registered new driver
Probing GRETH Ethernet Core at 0x80000b00
Detected MARVELL 88EE1111 Revision 2
10/100 GRETH Ethermac at [0x80000b00] irq 12. Running 100 Mbps full duplex
gr_udc-gr_probe(2154): could not find amba slave with id's 00000001 00000021
Attaching grlib ps2 mouse drivers at 0x80000400, irq: 4
Attaching grlib ps2 keyboard drivers at 0x80000500, irq: 5
Netfilter messages via NETLINK v0.30.
ip_tables: (C) 2000-2006 Netfilter Core Team
TCP cubic registered
NET: Registered protocol family 17
VFS: Mounted root (ext2 filesystem).
Freeing unused kernel memory: 136k freed


init started: BusyBox v1.8.2 (2011-03-06 14:47:37 EST)
starting pid 15, tty '': '/etc/init.d/rcS'
starting pid 26, tty '': '/bin/sh'
sh-3.1# echo $SHELL
/bin/sh
sh-3.1# which bash
/bin/bash
sh-3.1# dhclient eth0
Internet Software Consortium DHCP Client 2.0pl5
Copyright 1995, 1996, 1997, 1998, 1999 The Internet Software Consortium.
All rights reserved.

Please contribute if you find this software useful.
For info, please visit http://www.isc.org/dhcp-contrib.html

/bin/sh: error while loading shared libraries: libdl.so.2: cannot open shared object
    file: No such file or directory
Listening on LPF/eth0/00:0a:35:01:d2:66
Sending on   LPF/eth0/00:0a:35:01:d2:66
Sending on   Socket/fallback/fallback-net
DHCPREQUEST on eth0 to 255.255.255.255 port 67
DHCPACK from 192.168.1.1
/bin/sh: error while loading shared libraries: libdl.so.2: cannot open shared object
    file: No such file or directory
bound to 192.168.1.118 -- renewal in 43200 seconds.
sh-3.1# halt

The system is going down NOW!
```

```
Sending SIGTERM to all processes
Terminated
sh-3.1#
Sending SIGKILL to all processes

Requesting system halt
System halted.

Halt
```

D.5 Perl Base Station Script

Listing 11: Caption here

```
#! /usr/bin/perl -w

###############################################################################
# Author: Kendrick Wiersma
#   Date: 6 May 2011
#  Email: Kendrick.G.Wiersma@gmail.com
#
# For Senior Design 2011 Team 01 Team PICA. This simulates our basestation
#   functionality, within a reasonable shadow of a doubt :)
###############################################################################
use Device::SerialPort;
use Getopt::Long;
use Switch;

use strict;

my $printer=0;
my $help;

GetOptions( "printer=i" => \$printer,
            "help"      => \$help);   # parse command line option for which
if($help || $printer > 4 || $printer < 1){&usage}; # ensure valid comamnds

# Setup STDOUT to auto-flush
# Note: this disables the default buffered behaviour of stdout
select((select(STDOUT), $| = 1)[0]);

my $serialPort = Device::SerialPort->new("/dev/ttyS0") or die "Can't get the serial device\n
    "; # grab the serial port
$serialPort->databits(8);       # Set the number of data bits
$serialPort->baudrate(115200);  # Set the baudrate
$serialPort->parity("none");    # Select the parity
$serialPort->stopbits(1);       # Select the number of stopbits

my $charBuffer;                 # Buffer to hold our input

print "I'm about to start listening on the serial port";
while(1) {
    my $char = $serialPort->lookfor(); # Wait for a character (blocks)
    if($char){
#     print $char . "\n";
     my @array = split(/ /, $char);
     print $array[$printer-1] . "\n";
#    print "Got char: " . $char . "\n";
#    if($char eq 0xD) {                  # Start of Transmission (STX)
```

```
#         $charBuffer = "";               # start by nuking the character buffer
#     } elsif ($char eq 0xA) {              # End of Transmission (EOT)
      #     my @array = split('/ /', $char);
      #     print $array[$printer-1];       # Here we want to print the character buffer
#       print $charBuffer;
#     } else {
#         print "Accumulated: " . $char . "\n";
#         $charBuffer .= $char;         # Otherwise just accumulate
#         print $charBuffer . "\n";
#         $char = "";
#     }
    }
}

################################################################################
# Function: Usage
#     Displays some helpful usage information :)
#     Because seriously, who doesn't give usage info...oh wait...
################################################################################
sub usage {
    print <<ENDOFHELP;
    This script is intended to be piped with the driveGnuplot script
    Commandline options include:
       -printer  Sets which power to print valid options are 1, 2, 3, 4
           1 - Prints channel 1 Power
         2 - Prints channel 2 Power
         3 - Prints channel 3 Power
         4 - Prints total power
     -help  Displays this help message
ENDOFHELP
die;
}
```

E Embedded Linux from Scratch for the LEON3 SPARC

This guide is adapted from `http://cross-lfs.org/view/clfs-embedded/arm` to work on the LEON3 Sparc-V8-compatible hardware. It assumes that the SPARC processor has an FPU built into it.

This guide was originally tested and performed on Ubuntu 10.04 LTS. Hence, `/media` for the base of the mountpoint directory.

E.1 Setting up the build environment

E.1.1 Make a new partition

Use the program “GParted” to make an Ext3 partition of 2GB or more, and label it “CLFS”. Non-Ubuntu systems will probably have their own particular way of doing this that depends on where disks are mounted.

E.1.2 Mount the new partition

- Mount the partition: see your distribution for details. On Ubuntu, click "Places > CLFS" in the GNOME Panel.
- Set up an environment variable that points to the partition

```
export CLFS=/media/CLFS
```

- Create the source directories (as root)

```
sudo mkdir -v $CLFS/sources
sudo chmod -v a+wt $CLFS/sources
```

E.1.3 Download the CLFS-embedded sources

- Enter the sources directory

```
cd $CLFS/sources
```

- Make a list of all the sources

```
cat > embedded_list.txt << EOF
http://ftp.gnu.org/gnu/binutils/binutils-2.21.tar.bz2
http://busybox.net/downloads/busybox-1.17.3.tar.bz2
http://cross-lfs.org/files/packages/embedded-0.0.1/clfs-embedded-
    bootscripts-1.0-pre5.tar.bz2
http://downloads.sourceforge.net/e2fsprogs/e2fsprogs-1.41.14.tar.gz
ftp://gcc.gnu.org/pub/gcc/releases/gcc-4.5.2/gcc-4.5.2.tar.bz2
http://ftp.gnu.org/gnu/gmp/gmp-5.0.1.tar.bz2
http://cross-lfs.org/files/packages/embedded-0.0.1/iana-etc-2.30.tar.bz2
http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.36.3.tar.bz2
http://www.multiprecision.org/mpc/download/mpc-0.8.2.tar.gz
http://gforge.inria.fr/frs/download.php/27105/mpfr-3.0.0.tar.bz2
http://www.uclibc.org/downloads/uClibc-0.9.31.tar.bz2
http://downloads.sourceforge.net/libpng/zlib-1.2.3.tar.gz EOF
```

- Download these packages: (`-no-clobber` lets you skip what you already have)

```
wget --no-clobber -i embedded_list.txt
```

- Make a list of the Leon3-specific files:

```
cat > leon3_list.txt << EOF
ftp://gaisler.com/gaisler.com/linux/linux-2.6/kernel/mklinuximg
    -2.6.36-1.0.8.tar.bz2
ftp://gaisler.com/gaisler.com/linux/linux-2.6/kernel/leon-linux
    -2.6-2.6.36.3-1.0.1.tar.bz2
ftp://gaisler.com/gaisler.com/linux/linux-2.6/kernel/grlib-linux-drvpkg
    -1.0.0.tar.bz2
EOF
```

- Download these files, too.

```
wget --no-clobber -i leon3_list.txt
```

- Make a list of all patches from CLFS:

```
cat > patch_list.txt << EOF
http://patches.cross-lfs.org/embedded-dev/busybox-1.17.3-config-1.patch
```

```
http://patches.cross-lfs.org/embedded-dev/busybox-1.17.3-fixes-1.patch
http://patches.cross-lfs.org/embedded-dev/iana-etc-2.30-update-1.patch
http://patches.cross-lfs.org/embedded-dev/uClibc-0.9.31-configs-2.patch
EOF
```

- Download the patches

```
wget --no-clobber -i patch_list.txt
```

E.1.4 Add the CLFS user

- Become root to administer users, and keep CLFS set

```
sudo -s CLFS=$CLFS
```

- (As root) Create the group and user "clfs"

```
groupadd clfs
useradd -s /bin/bash -g clfs -m -k /dev/null clfs
passwd clfs
```

- (As root) Give the clfs user access to $CLFS – Note: $CLFS should still be set. This should not error.

```
chown -Rv clfs ${CLFS}
```

- Surrender root privileges

```
exit
```

- Become the clfs user

```
su - clfs
```

E.1.5 Configuring the clfs user

- Bash profile (when in a login shell)

```
cat > ~/.bash_profile << "EOF"
exec env -i HOME=${HOME} TERM=${TERM} PS1='\u:\w\$ ' /bin/bash
EOF
```

- Bash rc (for non-login shells)

```
cat > ~/.bashrc << "EOF"
set +h
umask 022
CLFS=/media/CLFS
LC_ALL=POSIX
PATH=${CLFS}/cross-tools/bin:/bin:/usr/bin
export CLFS LC_ALL PATH
EOF
```

- Load the new bash profile

```
~/.bash_profile
```

E.1.6 Preparing the filesystem

- Create directories necessary for Linux to work

```
mkdir -pv ${CLFS}/{bin,boot,dev,{etc/,}opt,home,lib/{firmware,modules},
    mnt}
mkdir -pv ${CLFS}/{proc,media/{floppy,cdrom},sbin,srv,sys}
mkdir -pv ${CLFS}/var/{lock,log,mail,run,spool}
mkdir -pv ${CLFS}/var/{opt,cache,lib/{misc,locate},local}
install -dv -m 0750 ${CLFS}/root
install -dv -m 1777 ${CLFS}{/var,}/tmp
mkdir -pv ${CLFS}/usr/{,local/}{bin,include,lib,sbin,src}
mkdir -pv ${CLFS}/usr/{,local/}share/{doc,info,locale,man}
mkdir -pv ${CLFS}/usr/{,local/}share/{misc,terminfo,zoneinfo}
mkdir -pv ${CLFS}/usr/{,local/}share/man/man{1,2,3,4,5,6,7,8}
for dir in ${CLFS}/usr{,/local}; do
  ln -sv share/{man,doc,info} ${dir}
done
```

- Make the cross-tools directories

```
install -dv $CLFS/cross-tools{,/bin}
```

- Set up the necessary files for Linux

  Mtab symlink (list of mounted file systems)

```
ln -svf ../proc/mounts ${CLFS}/etc/mtab
```

  Passwd (list of users)

```
cat > ${CLFS}/etc/passwd << "EOF"
root::0:0:root:/root:/bin/ash
daemon:x:2:6:daemon:/sbin:/bin/false
EOF
```

  Group (list of groups)

```
cat > ${CLFS}/etc/group << "EOF"
root:x:0:
bin:x:1:
sys:x:2:
kmem:x:3:
tty:x:4:
tape:x:5:
daemon:x:6:
floppy:x:7:
disk:x:8:
lp:x:9:
dialout:x:10:
audio:x:11:
video:x:12:
utmp:x:13:
usb:x:14:
cdrom:x:15:
EOF
```

  Log files for login, agetty, and init

```
touch ${CLFS}/var/run/utmp ${CLFS}/var/log/{btmp,lastlog,wtmp}
chmod -v 664 ${CLFS}/var/run/utmp ${CLFS}/var/log/lastlog
```

E.2 Making the Cross-Compile Tools

E.2.1 Setting CFLAGS

```
unset CFLAGS
unset CXXFLAGS
echo unset CFLAGS >> ~/.bashrc
echo unset CXXFLAGS >> ~/.bashrc
```

E.2.2 Setting build settings

Note: the CPU setting below was taken from the post at

http://gcc.gnu.org/ml/gcc/2010-11/msg00440.html

- Temporarily

```
export CLFS_HOST=$(echo ${MACHTYPE} | sed "s/-[^-]*/-cross/")
export CLFS_TARGET="sparc-embedded-linux-gnu"
export CLFS_ARCH="sparc"
export CLFS_CPU="sparchfleonv8"
export CLFS_FLOAT="hard"
```

- Persistently (you can log out and this will re-load on login)

```
echo export CLFS_HOST=\""$CLFS_HOST\"" >> ~/.bashrc
echo export CLFS_TARGET=\""$CLFS_TARGET\"" >> ~/.bashrc
echo export CLFS_ARCH=\""$CLFS_ARCH\"" >> ~/.bashrc
echo export CLFS_CPU=\""$CLFS_CPU\"" >> ~/.bashrc
echo export CLFS_FLOAT=\""$CLFS_FLOAT\"" >> ~/.bashrc
```

E.2.3 Install the Linux headers

- Unpack

```
cd $CLFS/sources
tar xf linux-2.6.36.3.tar.bz2
tar xf leon-linux-2.6-2.6.36.3-1.0.1.tar.bz2
tar xf grlib-linux-drvpkg-1.0.0.tar.bz2
```

- Apply Leon patches

```
cd linux-2.6.36.3
for file in `ls ../leon-linux-2.6-2.6.36.3-1.0.1/patches/*.patch`; do
  echo $file; patch -Np1 < $file; sleep 0.5; done
cp -r ../grlib-linux-drvpkg-1.0.0/kernel/drivers .
patch -p1 < drivers/grlib/add_grlib_tree.patch
if test $? -ne 0; then
  echo "obj-\$(CONFIG_SPARC_LEON) += grlib/"; fi >> drivers/Makefile
```

- Install headers

```
make mrproper
make ARCH=${CLFS_ARCH} headers_check
make ARCH=${CLFS_ARCH} INSTALL_HDR_PATH=dest headers_install
cp -rv dest/include/* ${CLFS}/usr/include
```

- Cleanup

```
cd $CLFS/sources
rm -rf linux-2.6.36.3
rm -rf leon-linux-2.6-2.6.36.3-1.0.1
rm -rf grlib-linux-drvpkg-1.0.0
```

E.2.4 Install GMP-5.0.1

```
cd $CLFS/sources
```

- Unpack

```
tar xf gmp-5.0.1.tar.bz2
```

- Configure

```
cd gmp-5.0.1
CPPFLAGS=-fexceptions ./configure --prefix=${CLFS}/cross-tools
```

- Make, time the result (2m11s for me.)

```
time make
```

- (Optional) Check that it's correct (2m46s for me)

```
time make check
```

- Install

```
make install
```

- Cleanup

```
cd $CLFS/sources
rm -rf gmp-5.0.1
```

E.2.5 MPFR-3.0.0

- Unpack

```
tar xf mpfr-3.0.0.tar.bz2
cd mpfr-3.0.0
```

- Configure

```
LDFLAGS="-Wl,-rpath,${CLFS}/cross-tools/lib" \
./configure --prefix=${CLFS}/cross-tools --enable-shared \
  --with-gmp=${CLFS}/cross-tools
```

- Make and install (1m29s for me)

```
time { make && make install; }
```

- Cleanup

```
cd $CLFS/sources
rm -rf mpfr-3.0.0
```

E.2.6 MPC-0.8.2

- Unpack

```
tar xf mpc-0.8.2.tar.gz
cd mpc-0.8.2
```

- Configure

```
LDFLAGS="-Wl,-rpath,${CLFS}/cross-tools/lib" \
  ./configure --prefix=${CLFS}/cross-tools \
  --with-gmp=${CLFS}/cross-tools \
  --with-mpfr=${CLFS}/cross-tools
```

- Build and install (About 21 seconds)

```
time { make && make install; }
```

- Cleanup

```
cd $CLFS/sources rm -rf mpc-0.8.2
```

E.2.7 Cross Binutils-2.21

- Extract

```
cd $CLFS/sources
tar xf binutils-2.21.tar.bz2
```

- Make a temporary directory

```
mkdir binutils-cross
cd binutils-cross
```

- Configure

```
../binutils-2.21/configure --prefix=${CLFS}/cross-tools \
  --target=${CLFS_TARGET} --with-sysroot=${CLFS} --disable-nls \
  --enable-shared --disable-multilib
```

- Build and install (5m11s)

```
time { make configure-host && make && make install; }
```

- Post-install maintenance

```
cp -v ../binutils-2.21/include/libiberty.h ${CLFS}/usr/include
```

- Cleanup

```
cd $CLFS/sources && rm -rf binutils-2.21 binutils-cross
```

E.2.8 Cross GCC - 4.5.2 – Static

- Extract

```
cd $CLFS/sources tar xf gcc-4.5.2.tar.bz2
```

- Make a temporary directory

```
mkdir -v gcc-cross-static cd gcc-cross-static
```

- Configure

```
AR=ar LDFLAGS="-Wl,-rpath,${CLFS}/cross-tools/lib" \
  ../gcc-4.5.2/configure --prefix=${CLFS}/cross-tools \
  --build=${CLFS_HOST} --host=${CLFS_HOST} --target=${CLFS_TARGET} \
  --with-sysroot=${CLFS} --disable-nls --disable-shared \
  --with-mpfr=${CLFS}/cross-tools --with-gmp=${CLFS}/cross-tools \
  --with-mpc=${CLFS}/cross-tools --without-headers --with-newlib \
```

```
  --disable-decimal-float --disable-libgomp --disable-libmudflap \
  --disable-libssp --disable-threads --enable-languages=c \
  --disable-multilib --with-float=${CLFS_FLOAT}
```

- Build (13m4s)

```
time make all-gcc all-target-libgcc
```

- Install (4 seconds) time

```
make install-gcc install-target-libgcc
```

- Cleanup

```
cd $CLFS/sources && rm -rf gcc-4.5.2 gcc-cross-static
```

E.2.9 uClibc-0.9.31

- Extract

```
tar xf uClibc-0.9.31.tar.bz2
```

- Patch cd uClibc-0.9.31

```
patch -Np1 -i ../uClibc-0.9.31-configs-2.patch
cp -v clfs/config.arm.big .config
```

- Configure Set arch=sparc (v8 will be selected automatically), and set FPU to be what you have in the Leon (Y/N)

```
make menuconfig
```

- Build (1m39s)

```
time make
```

- Install

```
make PREFIX=${CLFS} install
```

- Cleanup

```
cd $CLFS/sources && rm -rf uClibc-0.9.31
```

E.2.10 GCC-4.5.2 – Full

- Extract

```
cd $CLFS/sources tar xf gcc-4.5.2.tar.bz2
```

- Make build directories

```
mkdir gcc-cross-full cd gcc-cross-full
```

- Configure

```
AR=ar LDFLAGS="-Wl,-rpath,${CLFS}/cross-tools/lib" \
  ../gcc-4.5.2/configure --prefix=${CLFS}/cross-tools \
  --build=${CLFS_HOST} --target=${CLFS_TARGET} --host=${CLFS_HOST} \
  --with-sysroot=${CLFS} --disable-nls --enable-shared \
  --enable-languages=c,c++ --enable-c99 --enable-long-long \
  --with-mpfr=${CLFS}/cross-tools --with-gmp=${CLFS}/cross-tools \
  --with-mpc=${CLFS}/cross-tools --disable-multilib \
  --with-float=${CLFS_FLOAT}
```

- Build (20m02s)

```
time make
```

- Install

```
make install
```

- Cleanup

```
cd $CLFS/sources && rm -rf gcc-cross-full gcc-4.5.2
```

E.3 Building the System

E.3.1 Set up cross-compiling variables for convenience

```
echo export CC=\""${CLFS_TARGET}-gcc\"" >> ~/.bashrc
echo export CXX=\""${CLFS_TARGET}-gcc\"" >> ~/.bashrc
echo export AR=\""${CLFS_TARGET}-ar\"" >> ~/.bashrc
echo exportAS=\""${CLFS_TARGET}-as\"" >> ~/.bashrc
echo export LD=\""${CLFS_TARGET}-ld\"" >> ~/.bashrc
echo export RANLIB=\""${CLFS_TARGET}-ranlib\"" >> ~/.bashrc
echo export READELF=\""${CLFS_TARGET}-readelf\"" >> ~/.bashrc
echo export STRIP=\""${CLFS_TARGET}-strip\"" >> ~/.bashrc
source ~/.bashrc
```

E.3.2 busybox-1.17.3

-
- Extract

```
cd $CLFS/sources tar xf busybox-1.17.3.tar.bz2
```

- Patch

```
cd busybox-1.17.3
patch -Np1 -i ../busybox-1.17.3-fixes-1.patch
patch -Np1 -i ../busybox-1.17.3-config-1.patch
```

- Configuration Check Should not require input, just checks that the configuration is acceptable

```
cp -v clfs/config .config make oldconfig
```

- Build

```
make CROSS_COMPILE="${CLFS_TARGET}-"
```

- Install

```
make CROSS_COMPILE="${CLFS_TARGET}-" CONFIG_PREFIX="${CLFS}" install
```

- Cleanup

```
cd $CLFS/sources && rm -rf busybox-1.17.3
```

E.3.3 e2fsprogs-1.41.14

- Extract

```
tar xf e2fsprogs-1.41.14.tar.gz
cd e2fsprogs-1.41.14
```

- Make an internal build directory

```
mkdir -v build
cd build
```

- Configure

```
CC="${CC} -Os" ../configure --prefix=/usr --with-root-prefix="" \
  --host=${CLFS_TARGET} --disable-tls --disable-debugfs \
  --disable-e2initrd-helper --disable-nls \
```

- Build (53 seconds)

```
time make
```

- Install

```
make DESTDIR=${CLFS} install install-libs
```

- Cleanup

```
cd $CLFS/sources && rm -rf e2fsprogs-1.41.14
```

E.3.4 iana-etc-2.30

- Extract

```
tar xf iana-etc-2.30.tar.bz2
cd iana-etc-2.30
```

- Patch

```
patch -Np1 -i ../iana-etc-2.30-update-1.patch
```

- Make information up-to-date

```
make get
```

- Build (<1 second)

```
time make
```

- Install

```
make DESTDIR=${CLFS} install
```

- Cleanup

```
cd $CLFS/sources && rm -rf iana-etc-2.30
```

E.3.5 zlib-1.2.3

- Extract

```
tar xf zlib-1.2.3.tar.gz
cd zlib-1.2.3
```

- Patch

```
cp configure{,.orig}
sed -e 's/-O3/-Os/g' configure.orig > configure
```

- Configure

```
./configure --prefix=/usr --shared
```

- Build (<5 seconds)

```
time make
```

- Install

```
make prefix=${CLFS}/usr install
mv -v ${CLFS}/usr/lib/libz.so.* ${CLFS}/lib
ln -svf ../../lib/libz.so.1 ${CLFS}/usr/lib/libz.so
```

- Cleanup

```
cd $CLFS/sources && rm -rf zlib-1.2.3
```

E.3.6 Create the fstab file

```
cat > ${CLFS}/etc/fstab << "EOF"
# Begin /etc/fstab
# This assumes that your CF card has Ext3 on partition 1
#     and swap on partition 2.
# file system mount-point type    options          dump  fsck
#                                                       order
/dev/sda1     /            ext3    defaults         1     1
/dev/sda2     swap         swap    pri=1            0     0
proc          /proc        proc    defaults         0     0
sysfs         /sys         sysfs   defaults         0     0
devpts        /dev/pts     devpts  gid=4,mode=620   0     0
shm           /dev/shm     tmpfs   defaults         0     0
# End /etc/fstab
EOF
```

E.3.7 Compile the Linux kernel

- Extract

```
cd $CLFS/sources
tar xf linux-2.6.36.3.tar.bz2
tar xf leon-linux-2.6-2.6.36.3-1.0.1.tar.bz2
tar xf grlib-linux-drvpkg-1.0.0.tar.bz2
tar xf mklinuximg-2.6.36-1.0.8.tar.bz2
```

- Apply Leon patches

```
cd linux-2.6.36.3
for file in `ls ../leon-linux-2.6-2.6.36.3-1.0.1/patches/*.patch`; do
  echo $file; patch -Np1 < $file; done
cp -r ../grlib-linux-drvpkg-1.0.0/kernel/drivers .
patch -p1 < drivers/grlib/add_grlib_tree.patch
if test $? -ne 0; then
  echo "obj-\$(CONFIG_SPARC_LEON) += grlib/"; fi >> drivers/Makefile
```

- Prepare source

```
make mrproper
```

- Configure – don't make modules.

```
make ARCH=${CLFS_ARCH} CROSS_COMPILE="${CLFS_TARGET}-" menuconfig
```

- Build

```
time make ARCH=${CLFS_ARCH} CROSS_COMPILE=${CLFS_TARGET}- zImage
```

- Make the GRMON image Look up the MAC address of your device's ethernet port, then substitute it for the 001122334455 below.

```
cd ${CLFS}/sources/mklinuximg-2.6.36-1.0.8
sed -i "s@sparc-linux-@$CLFS_TARGET-@g" mklinuximg
./mklinuximg ../linux-2.6.36.3/arch/${CLFS_ARCH}/boot/image image.dsu \
  -cmdline "console=ttyS0,38400 root=/dev/sda1" -ethmac 001122334455
cp image.dsu ${CLFS}/boot/
cd ../linux-2.6.36.3
```

- Install the kernel

```
cp arch/${CLFS_ARCH}/boot/uImage ${CLFS}/boot/clfskernel-2.6.36.3
cp System.map ${CLFS}/boot/System.map-2.6.36.3
cp .config ${CLFS}/boot/config-2.6.36.3
```

- Cleanup

```
cd $CLFS/sources
rm -rf linux-2.6.36.3 leon-linux-2.6-2.6.36.3-1.0.1 grlib-linux-drvpkg
   -1.0.0
```

E.3.8 Giving control to root

- Become root

```
su -s CLFS=\$CLFS"
```

- Change the ownership of all of $CLFS

```
chown -Rv root:root ${CLFS}
```

- Switch a few to another group

```
chgrp -v utmp ${CLFS}/var/run/utmp ${CLFS}/var/log/lastlog
```

- Create the two most important nodes

```
mknod -m 0666 ${CLFS}/dev/null c 1 3
mknod -m 0600 ${CLFS}/dev/console c 5 1
```

E.3.9 CLFS-Bootscripts-1.0-pre5

- Extract

```
cd ${CLFS}/sources
tar xf clfs-embedded-bootscripts-1.0-pre5.tar.bz2
cd clfs-embedded-bootscripts-1.0-pre5
```

item Install

```
mkdir ${CLFS}/etc/rc.d
mkdir ${CLFS}/etc/rc.d/{init.d,start,stop}
make DESTDIR=${CLFS} install-bootscripts
```

- Cleanup

```
cd ${CLFS}/sources && rm -rf clfs-embedded-bootscripts-1.0-pre5
```

E.3.10 mdev

- Fill the file

```
cat > ${CLFS}/etc/mdev.conf << "EOF"
# /etc/mdev/conf
# Symlinks:
# Syntax: %s -> %s

MAKEDEV -> ../sbin/MAKEDEV
/proc/core -> kcore
fd -> /proc/self/fd
mcdx -> mcdx0
radio -> radio0
ram -> ram1
sbpcd -> sbpcd0
sr0 -> scd0
sr1 -> scd1
sr10 -> scd10
sr11 -> scd11
sr12 -> scd12
sr13 -> scd13
sr14 -> scd14
sr15 -> scd15
sr16 -> scd16
sr2 -> scd2
sr3 -> scd3
sr4 -> scd4
sr5 -> scd5
sr6 -> scd6
sr7 -> scd7
sr8 -> scd8
sr9 -> scd9
```

```
stderr -> fd/2
stdin -> fd/0
stdout -> fd/1

# Remove these devices, if using a headless system
# You will see an error mdev: Bad line 35
vbi -> vbi0
vcs -> vcs0
vcsa -> vcsa0
video -> video0
# Stop Remove for headless system

# Devices:
# Syntax: %s %d:%d %s
# devices user:group mode

null 0:0 777
zero 0:0 666

urandom 0:0 444

console 0:5 0600
fd0 0:11 0660
hdc 0:6 0660
kmem 0:9 000
mem 0:9 0640
port 0:9 0640
ptmx 0:5 0660
tun[0-9]* 0:0 0640 =net/

sda* 0:6 0660
sdb* 0:6 0660
hda* 0:6 0660
hdb* 0:6 0660

tty 0:5 0660
```

```
tty0* 0:5 0660
tty1* 0:5 0660
tty2* 0:5 0660
tty3* 0:5 0660
tty4* 0:5 0660
tty5* 0:5 0660
tty6* 0:5 0660

ttyS* 0:20 640
EOF
```

E.3.11 Profile

- Create the file

```
cat > ${CLFS}/etc/profile << "EOF"
# /etc/profile

# Set the initial path
export PATH=/bin:/usr/bin

if [ `id -u` -eq 0 ] ; then
        PATH=/bin:/sbin:/usr/bin:/usr/sbin
        unset HISTFILE
fi

# Setup some environment variables.
export USER=`id -un`
export LOGNAME=$USER
export HOSTNAME=`/bin/hostname`
export HISTSIZE=1000
export HISTFILESIZE=1000
export PAGER='/bin/more '
export EDITOR='/bin/vi'

# End /etc/profile
```

```
EOF
```

E.3.12 Inittab

```
cat > ${CLFS}/etc/inittab << "EOF"
# /etc/inittab

::sysinit:/etc/rc.d/startup

tty1::respawn:/sbin/getty 38400 tty1
tty2::respawn:/sbin/getty 38400 tty2
tty3::respawn:/sbin/getty 38400 tty3
tty4::respawn:/sbin/getty 38400 tty4
tty5::respawn:/sbin/getty 38400 tty5
tty6::respawn:/sbin/getty 38400 tty6

# Put a getty on the serial line (for a terminal)
# uncomment this line if your using a serial console
#::respawn:/sbin/getty -L ttyS0 115200 vt100

::shutdown:/etc/rc.d/shutdown
::ctrlaltdel:/sbin/reboot
EOF
```

E.3.13 Hostname

- Pick a hostname. Set it to `CLFS_HOST` (replace localhost below)

```
CLFS_HOST="localhost"
```

- Make it into a file

```
echo "$CLFS_HOST" > ${CLFS}/etc/HOSTNAME
```

E.3.14 Hosts file

- Fill file. Replace `CLFS_HOSTNAME` with your name, as cat doesn't evaluate it.

```
cat > ${CLFS}/etc/hosts << "EOF"
# Begin /etc/hosts (network card version)

127.0.0.1 localhost
127.0.1.1 CLFS_HOSTNAME

# End /etc/hosts (network card version)
EOF
```

E.3.15 Network configuration

```
cat > ${CLFS}/etc/network.conf << "EOF"
# /etc/network.conf
# Global Networking Configuration
# interface configuration is in /etc/network.d/

# set to yes to enable networking
NETWORKING=yes

# set to yes to set default route to gateway
USE_GATEWAY=no

# set to gateway IP address
GATEWAY=192.168.0.1

# Interfaces to add to br0 bridge
# Leave commented to not setup a network bridge
# Substitute br0 for eth0 in the interface.eth0 sample below to bring up br0
# instead
# bcm47xx with vlans:
#BRIDGE_INTERFACES="eth0.0 eth0.1 wlan0"
# Other access point with a wired eth0 and a wireless wlan0 interface:
```

```
#BRIDGE_INTERFACES="eth0 wlan0"

EOF

mkdir ${CLFS}/etc/network.d
cat > ${CLFS}/etc/network.d/interface.eth0 << "EOF"
# Network Interface Configuration

# network device name
INTERFACE=eth0

# set to yes to use DHCP instead of the settings below
DHCP=no

# IP address
IPADDRESS=192.168.1.2

# netmask
NETMASK=255.255.255.0

# broadcast address
BROADCAST=192.168.1.255
EOF

cat > ${CLFS}/etc/udhcpc.conf << "EOF"
#!/bin/sh
# udhcpc Interface Configuration
# Based on http://lists.debian.org/debian-boot/2002/11/msg00500.html
# udhcpc script edited by Tim Riker <Tim@Rikers.org>

[ -z "$1" ] && echo "Error: should be called from udhcpc" && exit 1

RESOLV_CONF="/etc/resolv.conf"
RESOLV_BAK="/etc/resolv.bak"

[ -n "$broadcast" ] && BROADCAST="broadcast $broadcast"
```

```
[ -n "$subnet" ] && NETMASK="netmask $subnet"

case "$1" in
        deconfig)
                if [ -f "$RESOLV_BAK" ]; then
                        mv "$RESOLV_BAK" "$RESOLV_CONF"
                fi
                /sbin/ifconfig $interface 0.0.0.0
                ;;

        renew|bound)
                /sbin/ifconfig $interface $ip $BROADCAST $NETMASK

                if [ -n "$router" ] ; then
                        while route del default gw 0.0.0.0 dev $interface ; do
                                true
                        done

                        for i in $router ; do
                                route add default gw $i dev $interface
                        done
                fi

                if [ ! -f "$RESOLV_BAK" ] && [ -f "$RESOLV_CONF" ]; then
                        mv "$RESOLV_CONF" "$RESOLV_BAK"
                fi

                echo -n > $RESOLV_CONF
                [ -n "$domain" ] && echo search $domain >> $RESOLV_CONF
                for i in $dns ; do
                        echo nameserver $i >> $RESOLV_CONF
                done
                ;;
esac

exit 0
```

```
EOF

chmod +x ${CLFS}/etc/udhcpc.conf
cat > ${CLFS}/etc/resolv.conf << "EOF"
# Begin /etc/resolv.conf

domain example.org
nameserver 192.168.0.1
nameserver 192.168.1.1

# End /etc/resolv.conf
EOF
```

E.4 Finishing Up

E.4.1 Make a final directory

- Copy to a new directory

```
install -dv ${CLFS}-final
cp -arv ${CLFS}/* ${CLFS}-final/
```

- Remove the unnecessary things

```
rm -rfv ${CLFS}-final/cross-tools
rm -rfv ${CLFS}-final/usr/src/*
rm -rfv ${CLFS}-final/usr/include
rm -rfv ${CLFS}-final/usr/man
rm -rfv ${CLFS}-final/usr/share/man
rm -rfv ${CLFS}-final/sources
FILES="$(ls ${CLFS}-final/lib/*.a ${CLFS}-final/usr/lib/*.a)"
for file in $FILES; do rm -fv $file; done
```

E.4.2 Create a tarball

```
install -dv ${CLFS}/build
cd ${CLFS}-final
tar jcfv ${CLFS}/build/clfs-leon3-embedded.tar.bz2 *
```

F Power Supply Photographs

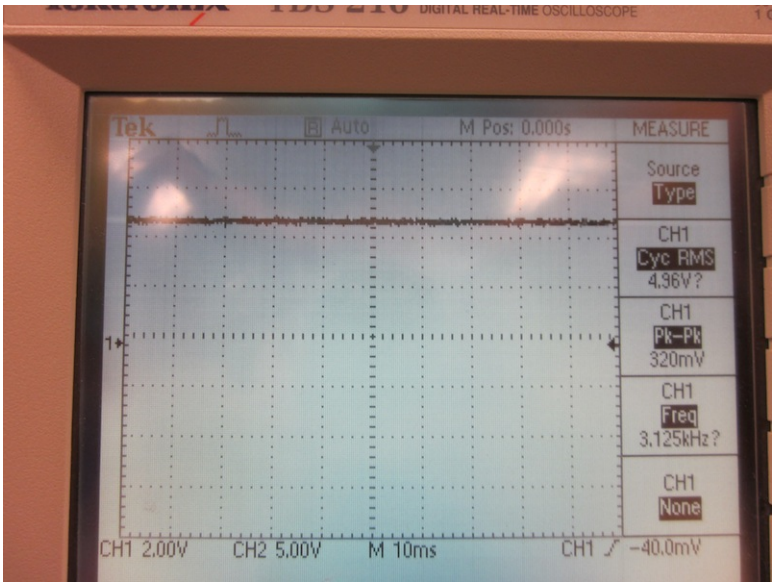


Figure 68: Oscilloscope Reading-Shows about 4.96V

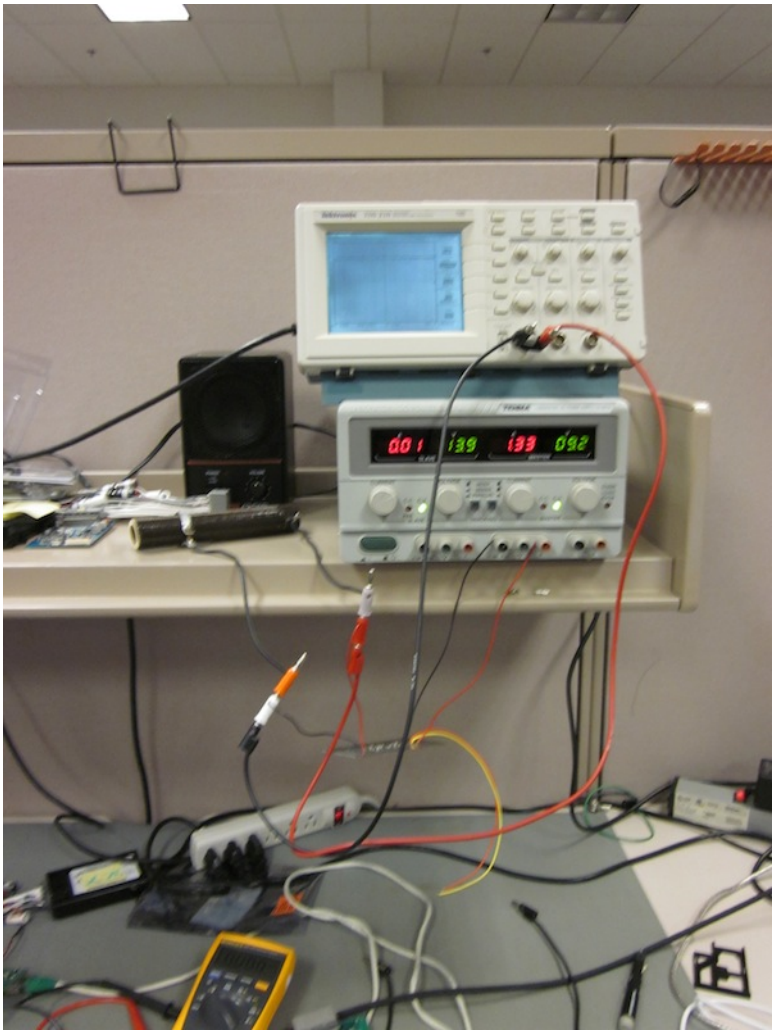


Figure 69: General Test Setup

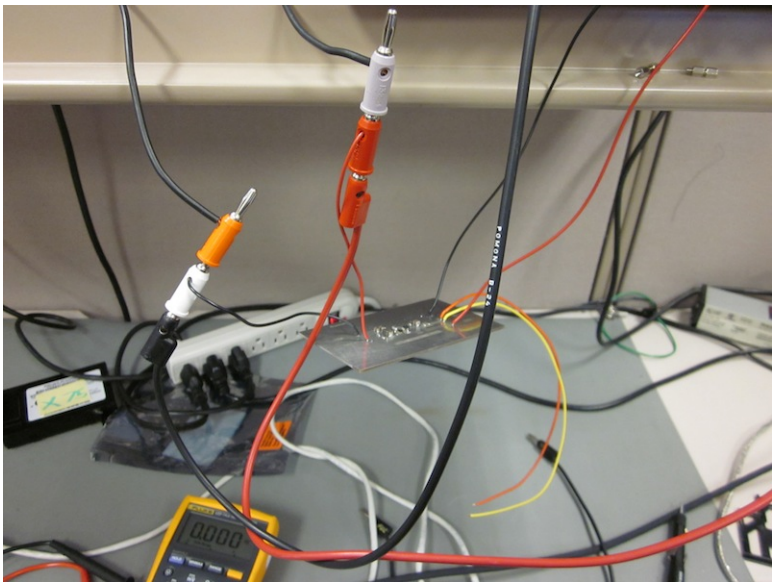


Figure 70: Power Supply Being Tested

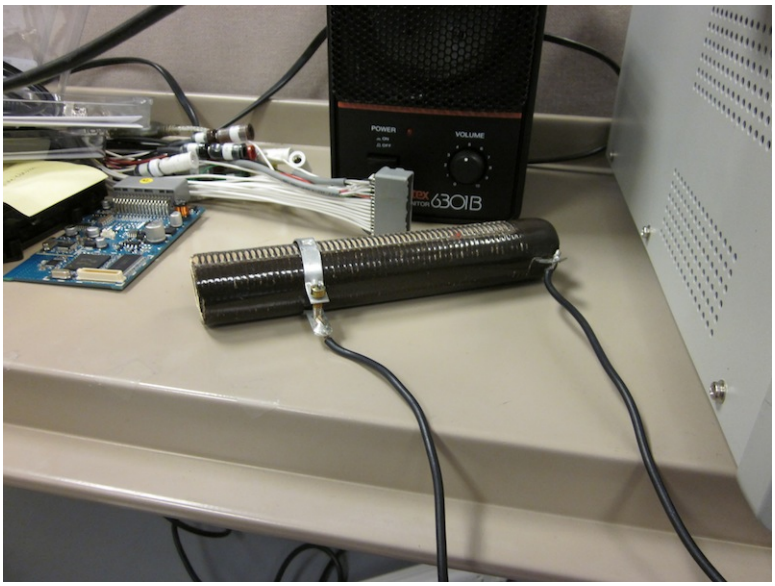


Figure 71: 2.5Ω Load