An Intelligent Alternative Energy Controller

Final Report

Junior Design: Team E

Alan Huynh
Blaine Martinez
Chuck Dahl
David Breen
David Park

May 11, 2010

New Mexico Institute of Mining and Technology

Abstract

This document details our (Junior Design Team E's) design, development, construction, and testing of an intelligent alternative energy controller that takes in power from natural energy sources and converts them into a usable form. This converted energy is used to charge a bank of batteries and supply AC and/or DC power to almost any electrical item.

A brief background on energy conversion is provided, followed by a description of the design parameters we were to fulfill. Each subsystem and its components are discussed in detail, covering hardware as well as software (when applicable) aspects of the design. Testing and analysis data are provided, as well as a summary of expenses incurred for the project. The report concludes with an evaluation of our final design and problems encountered along the way. Code for the software portions of the project and other miscellaneous information are included at the end as appendices.

Acknowledgments

- Dr. Hector Erives
- Dr. Robert Bond
- Dr. Anders Jorgensen
- Andrew Tubesing
- Chris Pauli
- Henry Godman

Table of Contents

Table of Figures

List of Tables

Chapter 1

Introduction

This paper presents the outline and results of our Intelligent Alternative Energy Controller. The project goal is to take energy from two renewable energy sources, a wind turbine and photovoltaic panel, and use it to charge a bank of batteries. The batteries will then be used to provide a 12VDC and 120VAC at 60 Hz output. All the while, the current and voltages between the energy generators and the subsystems will be measured, as well as the outputs voltages and current. For the AC output case, the frequency and power factor of the signal will also be measured. Once all the data has been collected, it will be uploaded on to the EE web-server, which then hosts the information on the internet.

The project motivation is for society to become more independent of non-renewable resources. Currently the majority of the electricity supplied in the United States is provided by coal and natural gas. The largest emission free resources used are nuclear power plants. The problem with the power plants is that they require uranium as a source. Usable uranium is difficult to harvest and it remains radioactive after it has been used, which requires proper disposal. Coal and natural gas, on the other hand, emit large amounts of pollution into the atmosphere. For these reasons and more, wind turbines and photovoltaic panels are good choices for sources of energy. Each provide energy without any emissions and will continue to supply energy until for their life's entirety.

Chapter 2

Background

There are wide selections of intelligent inverters available on the market today. These inverters have a wide variety of features, from low battery and temperature alarms, to pure sign-wave output, and fan-less operation. The requirements of this project included data logging, and internet access. These features, while available as separate instrumentation packages, are not generally available as part of an integrated system. This integration is part of the requirements of the project, and necessitated the design and construction of our own systems.

Chapter 3

Design

Our design began with the abstraction of the system as a whole and the interaction of the different modules. The project specifications required that the system utilizes the charge controller of the wind turbine and therefore be connected directly to the battery without any further design. The photovoltaic panels (PV) are to be connected indirectly to the battery via a charge controller. The regulated AC and DC power supplies originate directly from the battery and employ an automated means of disconnection in the event of low battery voltage. The RabbitCore RCM4000 microcontroller is responsible for data acquisition and uploading the data to the web. Since the design must be a standalone system, regulated power supplies are used to provide the various voltages used throughout the system. Figure 3.1 shows the general layout of system components.

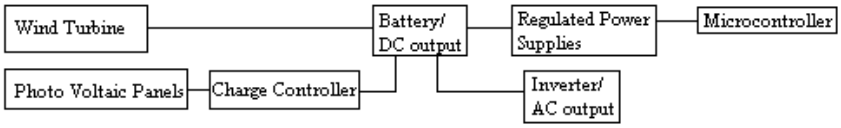


Figure 3. 1: Block Overview of System Components

Over-current protection is provided at various key locations within the system. The PV panels, system supply wire, inverter, DC output, and low current circuitry are all protected by individual fuses. Figure 3.2 shows the location of these fuses, later discussion will explain the use of relays which aren't shown in this figure.

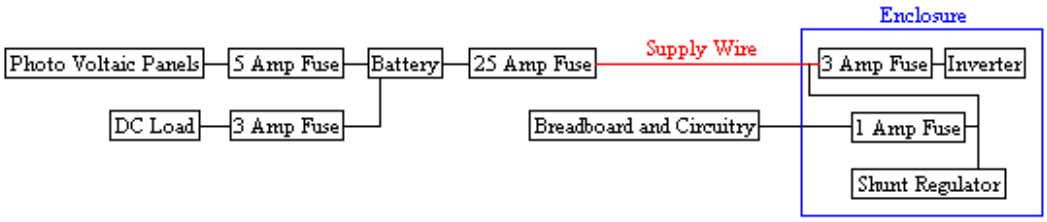


Figure 3. 2: Block Overview of Safety Components

3.1 Battery Charger Subsystem

The charge control of the PV panel is provided through the use of a shunt load. The discovery of this practical and easy method of regulation was found through a renewable energy website[1]. The alternative to this method of charge control was to use an integrated circuit that provided multiple charge rates that were dependent on changing parameters of the external circuit. The complexity of this IC and its external circuit made its implementation very

time consuming and impractical. The use of the shunt regulator, in conjunction with the microcontroller, is an adequate means of regulating a low wattage alternative energy source such as the PV panels used on this project.

3.2 AC Inverter Subsystem

The inverter topology used was chosen based on the review of a previously proven design[2]. This design involves changing the direction of current through the center tapped primary winding of a step up transformer. MOSFETS are employed as switches to control the direction and time that current is allowed to flow through an individual side of the center tapped winding. By changing the direction of the current in the primary winding, the voltage induced into the secondary winding will change polarity, resulting in an AC voltage output. The output is then filtered providing a sinusoidal output waveform. Figure 3.3 shows the basic design of the inverter.

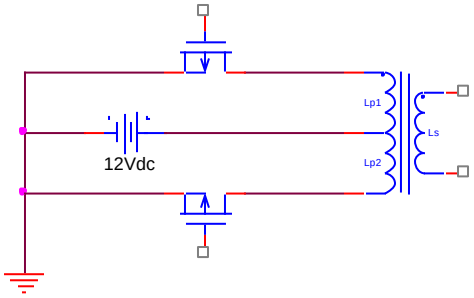


Figure 3. 3: Inverter Design

This general design can be implemented in various ways. Originally, the inverter was chosen to deliver 300 watts, but time restrains forced upon an alternative design with a much lower wattage output. The switching of the MOSFETS is done by generating a pulse train with the use of a Field Programmable Gate Array (FPGA) and a driver circuit. The alternative way of generating the pulse train was to use a carrier signal along with a reference sine wave which is fed into a comparator circuit.

3.3 Microcontroller and Configuration

As the core component of the entire system, the microcontroller controls and monitors the entire system, programmed to measure voltages and currents at specific points within the system with its analog-to-digital (A/D) converters and to control various aspects of the system. The data is posted on the web through the integrated network module and is also used by the system itself to determine the state of several outputs which are interfaced to other system components. The microcontroller is essential to the operation of the battery charge controller and the disconnection of DC and AC loads when the battery voltage becomes too low, and controls the pulse train module that provides the inverter's AC output.

To achieve this intended functionality, a pool of suitable products was to be chosen from: the Arduino Mega, the Dragon12, and the RCM4000 RabbitCore. The Arduino's main advantage is its user-friendly expandability by use of "shields" – add-on modules which attach to the Arduino to provide additional functionality – and an IDE (Integrated Development Environment) which uses a variant of C, a programming language the team is experienced in. However, the Arduino inherently lacks Internet connectivity, which is a requirement. The Dragon12's main advantage is the team's inherent familiarity with developing on it and its wider array of features and modules, but it too lacks Internet connectivity. Both of these would have to interface with external components to achieve connectivity. While the Arduino and Dragon12 were also suitable for this application, the Rabbit was chosen since it has a fast clock, built-in Ethernet capabilities, and was kindly donated to our project.

The RCM4000 is a development platform designed to be used in embedded systems, communications, and remote control purposes. It is powered by a 58.98MHz clock, 22 general purpose digital I/O pins, 8 analog inputs, 32MB flash memory, real-time clocks and timers, and a built-in Ethernet port. The package comes with Dynamic C software, a user-friendly IDE that facilitates rapid development and prototyping, and comes with code libraries for utilizing the microcontroller's various components, which became useful for the Internet interface and measurement subsystems of our design.

3.3.1 Monitor/Data Logger Subsystem

Options for the measurement portion of the project were to use an integrated chip for frequency and power factor, use a hall-effect sensor to measure current, use a current sense resistor for current, and a simple voltage divider for the voltage measurements. The integrated chip solution became very complicated design in itself. The circuit layout is large and the outputs are fed directly to the microcontroller through a serial port. The hall-effect sensor was a reasonable solution for the current measurements, but the sensor itself had a higher price. So to keep the design inexpensive, the current sense resistor and voltage divider ideas were implemented.

3.3.2 Interface to Internet Subsystem

The project specifications required that the system be able to send and store data to an Internet-connected server for logging and viewing by a user. The data may be sent using either the HTTP or SSH protocol, and may be stored in any manner – SQL (Structured Query Language) databases, text files, etc. The team chose to use the HTTP protocol to communicate and send data to the server and MySQL database software for persistent storage of data. A PHP (Hypertext Preprocessor) script pulls data from the SQL database for display in a web browser.

As noted above, the team required a microcontroller with communication capabilities. The first choice was to use the Arduino Mega with an Ethernet shield that contained the WIZnet 811MJ network module to achieve connectivity. The Dragon12, lacking Ethernet, would also

have to do the same, using some sort of serial communication protocol like I^2C (Inter-Integrated Circuit) or SPI (Serial Peripheral Interface). In an effort to save funds and simplify the design and programming aspects, the team chose to utilize the Rabbit's built-in Ethernet module and code libraries to interface with the Internet. This decision saved the team from having to spend time programming serial communication routines. Figure 3.4 shows a block diagram of how the system information is taken and stored to the EE web-server, then to the internet.

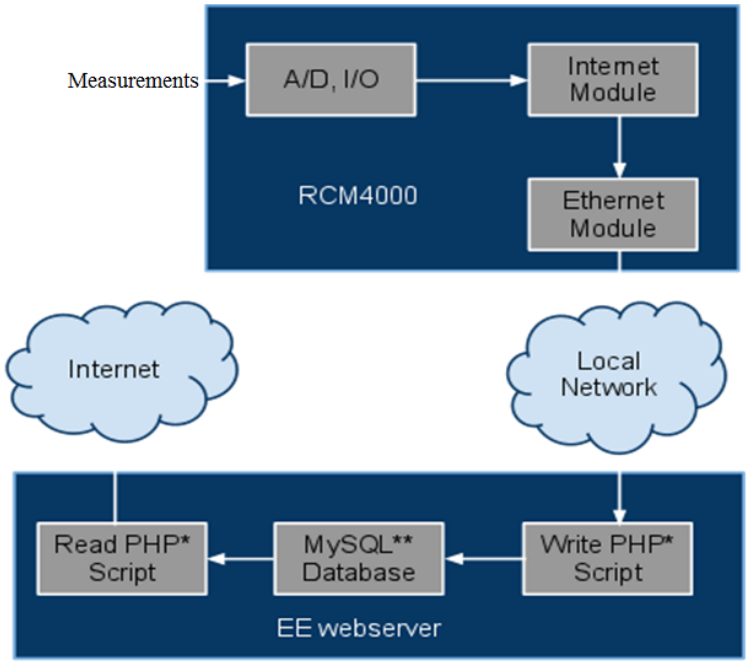


Figure 3. 4 : Internet Interface Outline

Chapter 4

Implementation

4.1 AC Inverter Subsystem Testing

The inverter schematic shown in Appendix Figure 1 is courtesy of Didik Rostyono and Harsono Hadi. This design was used as a template and was subjected to various changes before its final implementation; figure 4.1 shows the basic inverter layout. The transformer selection was paramount to the operation and capabilities of the inverter.

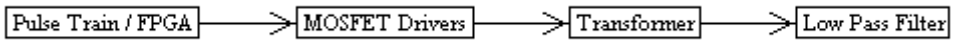


Figure 4. 1 : Basic Inverter Layout

The expectations of the inverter were to provide approximately 300 watts of power. An attempt was made to design and build a transformer suitable for a 300 watt inverter (with the use of a donated transformer core and bobbins). Extensive P-spice simulations were done modeling the effects of applying 12 volt pulses of different lengths to the primary side of step-up transformers of different parameters. Appendix Figure 2 shows the P-spice circuit used for these simulations. Unfortunately, during testing it was discovered that the finished transformer didn't have enough impedance in the primary winding and would not limit currents in the primary circuit to reasonable levels with no load. High currents are inevitable in the primary circuit of an inverter while running a high wattage load, and therefore required the design of a primary circuit capable of carrying these large currents. Number 10 AWG (American wire gauge) was used for the interconnections between the battery and the primary circuit components; switch, fuse, buss-bars, and transformer. Although the MOSFETS are rated for 60 amps, the connection of the MOSFETs to the external circuit could greatly decrease this capability and a unique design was needed to minimize the resistance from connection of the TO247 MOSFET packages to the external circuit. Figure 4.2 show the lay of the bus bars used for the high current MOSFETs and Diodes.

Figure 4. 2 : High Current MOSFET and Diode Layout

The pulse train originates from the Altera Max II field programmable gate array. The code for generating the pulse can be found in the appendix. The signal was then passed through a transistor/resistor network which would develop and apply a 10 volt pulse train to the gates of the MOSFETS, which would fully turn on or off the MOSFETS to the primary side of the transformer. Figure 4.3 shows the waveform of the pulse trains.

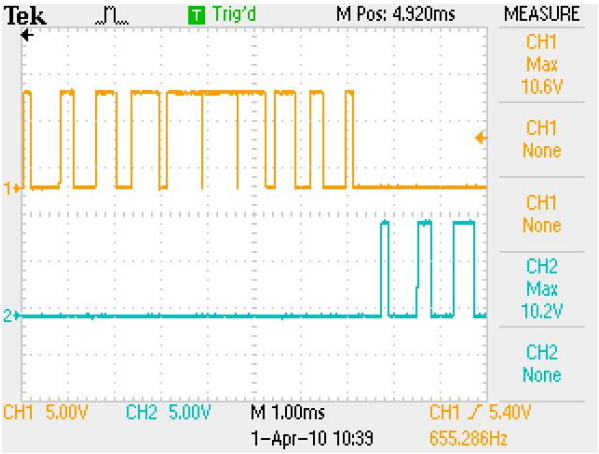


Figure 4. 3 : MOSFET Gate Drivers

In order to maximize current carrying ability and minimize resistance, parallel MOSFETs were used to switch each side of the center tapped primary winding. Flyback diodes are used in the primary circuit to provide a discharge path when attempting to interrupt the current in the inductor during switching. The particular switching of the MOSFETS allows a corresponding current to flow in the primary winding which thereby induces a voltage into the secondary that is proportional to the turns ratio of the transformer. Figure 4.4 shows the unfiltered output of the secondary. The changing direction of the current in the primary winding will result in an AC voltage on the secondary.

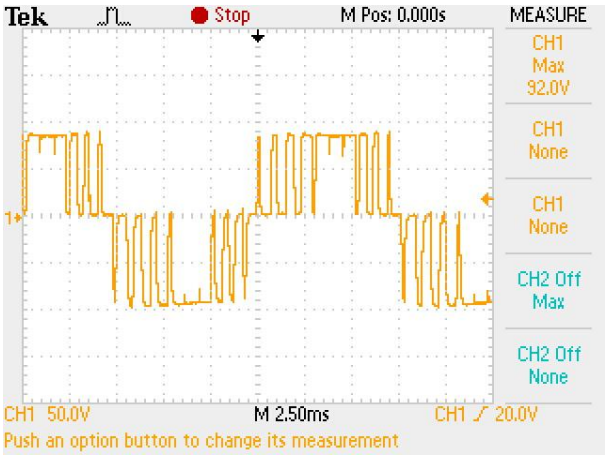


Figure 4. 4 : Unfiltered Inverter Output

A low pass filter is then used to filter out the higher frequencies of the output. A P-spice simulation was done using the 2.2uF capacitor as specified in our design template. The simulation verified that this would be a suitable capacitor to use. The filtered output is shown in figure 4.5 below.

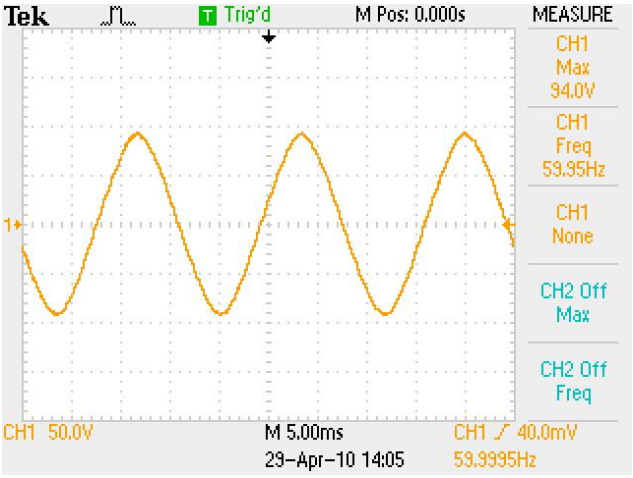


Figure 4. 5 : Filtered Inverter Output

The above waveform shows that the output isn't very close to our target voltage of 120 V_{rms}. This can be attributed to the turns ratio of the transformer that was resorted to in the final decision. As previously discussed, the inverter didn't utilize the high wattage transformer that was preferred. Instead, four small 5 VA transformers were used to make two separate center tapped transformers. The primary windings are connected in parallel and the secondary windings are connected in series, they are connected in this manner in order to get the greatest turns ratio possible out of these transformers. The primary impedance of this equivalent center tapped transformer had approximately 130 times the impedance of the transformer we designed and built. This added impedance was necessary for the proper operation of the inverter.

4.2 Measurement Subsystem Testing

This system was implemented in two sections; the measurement calculations such as the power factor take place in code on the microcontroller. The microcontroller measures the frequency of the voltage and current wave forms. This is accomplished by reading two of the digital I/O lines, and measuring the time high, as well as phase of two pulse trains. These pulse trains are generated by comparator circuits located on a breadboard. These comparators take an attenuated sine-wave as input, and switch on zero crossings. With the addition of a small amount of hysteresis to combat noise problems, the comparators produce a zero to three volt square wave. Magnitude of the current and voltage is measured using half wave rectification. To measure the magnitude of the voltage sine-wave, the voltage is attenuated to a three volt peak to peak wave, which is then fed to the half wave rectifier. The average output of the half

wave rectifier is then measured with the ADC. The magnitude of the current is measured in a similar manner, with the voltage taken over the current sense resistor.

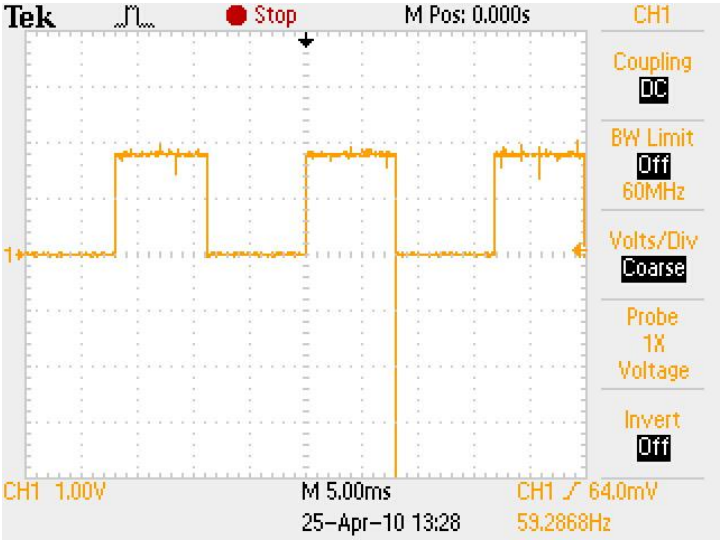


Figure 4. 6 : Example of Current and Voltage Pulse Trains

As is visible in the figure 4.6 above, the comparator output waveforms exhibit an occasional sharp negative voltage spike. This is most probably caused by noise generated by the switching power supply. Bypass capacitors were used to alleviate this noise, but this technique was not totally successful. Due to the extremely short nature of these spikes, they are not harmful to the micro-controller, or its functionality.

These comparator outputs were obtained in a laboratory setting, with clean sine-waves as inputs. Due to the small value of the current sense resistor, the voltage drop over it at relatively low currents is swamped by noise as seen in the figure 4.7 below. This made it impossible for the current comparator to operate properly. Due to time constraints, the current measurement doesn't work outside the laboratory.

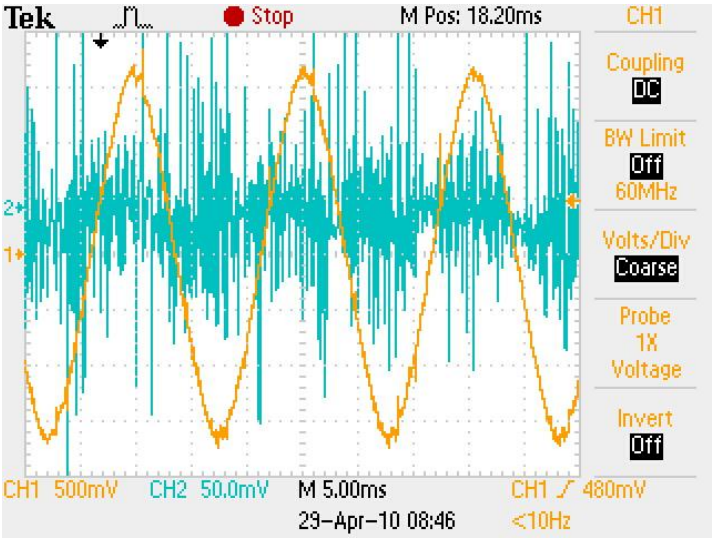


Figure 4. 7 : Current Sense Resistor Output

The half wave rectified magnitude measurement system was also tested. The output signal has some ripple, but the average of the output is within reasonable tolerance, as seen in figure 4.8 below.

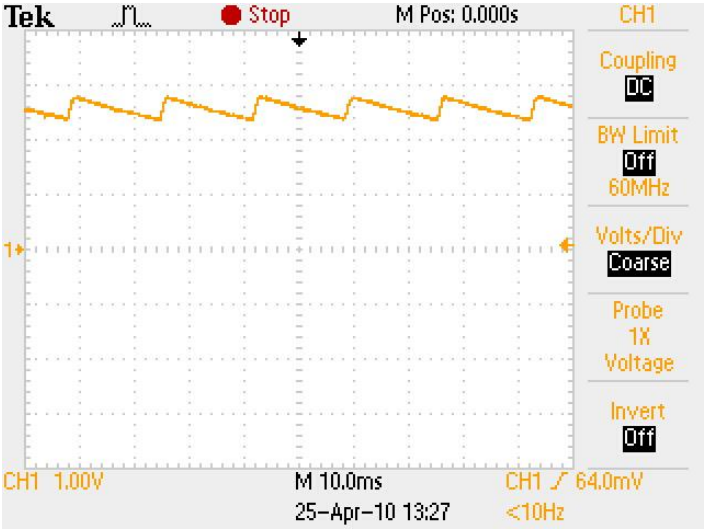


Figure 4. 8 : Output of Half-Wave Rectifier

4.3 Interface to Internet Testing

The network interface module was tested initially with fake data. This data was passed to the EE department student server utilizing the HTTP protocol and the GET method. The data is received by a php script on the server, and written to a MySQL database along with a time stamp. After the measurement system was finished, the network interface was tested with data produced by the measurement system with two sine-waves as input.

The code based part of the system was tested in the lab, with the square wave pulse trains simulated by two function generators with a 90 degree phase shift between the outputs. The phase and frequency measurements were then tested. All amplitude measurements were simulated with variable voltage sources connected to the inputs of the on-board ADC.

time_stamp	batt_voltage	load_voltage	load_current	frequency	pf
2010-05-04 20:36:43	12.1898	3.01791	0.887003	55.5556	-0.448053
2010-05-04 20:36:33	12.1898	3.00311	0.940684	55.5556	-0.448053
2010-05-04 20:34:46	12.1898	3.00311	0.927903	62.5	-0.448085
2010-05-04 20:36:22	12.1898	3.01791	0.871666	62.5	-0.448085
2010-05-04 20:35:07	12.1898	3.00311	0.930459	62.5	-0.448085
2010-05-04 20:36:11	12.1898	3.00311	0.866553	62.5	-0.448085
2010-05-04 20:36:01	12.1898	3.00311	0.86911	55.5556	-0.448053
2010-05-04 20:35:50	12.1898	3.01791	0.927903	62.5	-0.448085
2010-05-04 20:35:40	12.1898	3.01791	0.86911	62.5	-0.448085
2010-05-04 20:35:29	12.1898	3.01791	0.86911	62.5	-0.448085
2010-05-04 20:35:18	12.1898	3.01791	0.871666	55.5556	-0.448053
2010-05-04 20:34:57	12.1898	3.01791	0.866553	71.4286	-0.448053

Figure 4. 9 : Sample data collected in the lab

4.4 Battery Charger Subsystem

The battery charger subsystem was implemented very simply using a power shunt resister. This resistor has to be sized correctly to ensure proper regulation. The solar panel has characteristics as shown in the illustration in Figure 4.10,

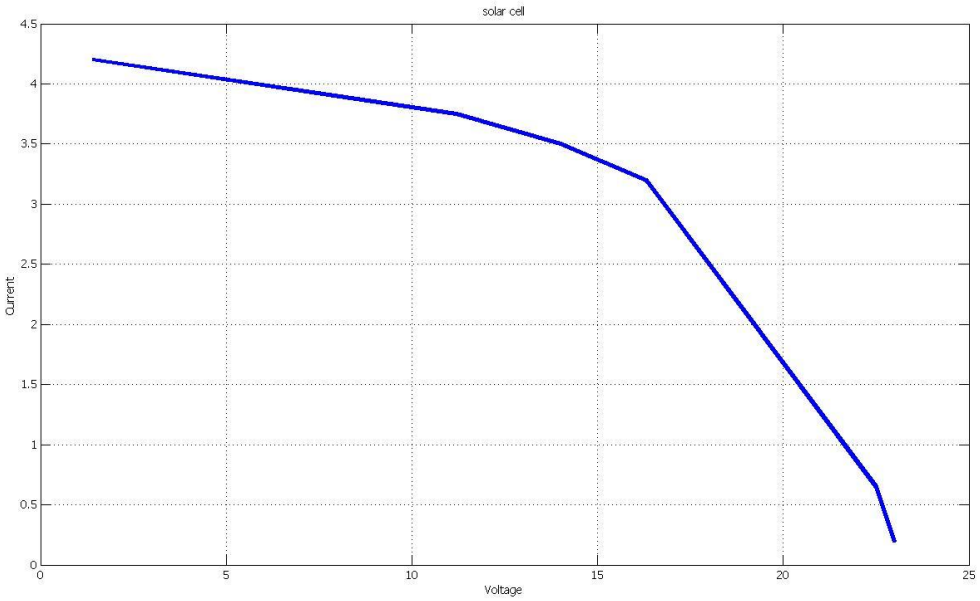


Figure 4. 10 : Solar Cell Characterization:
Courtesy of John Martinez and Henry Godman

These characteristics will cause a change in output current, as well as changes in the amount of sun. These changes cause corresponding changes in the charging current. The shunt resistors should be sized such that the average current output by the solar panel is dissipated through the resistors when the shunt load is on. This should maintain a constant average charge on the battery when no load is present, or the battery is underutilized. Oscillations in the on and off cycles of the shunt load are prevented through the use of hysteresis. This control system is implemented in code on the microcontroller. This method allows simple adjustment of the hysteresis levels, as well as the control of other parts of the circuit, by the modification of the constants containing the hysteresis levels in the code. Simple control of the shunt load MOSFET is achieved using standard digital I/O, as well as driver circuitry to provide the voltage necessary to fully turn on the power MOSFET.

4.4.1 Battery Charger Subsystem Testing

Testing of the shunt regulator was achieved using the battery, as well as a variable logic signal simulating the output from the digit. This allowed testing of the shunt regulator in both modes of regulation.
The hysteresis levels were tested using a variable voltage source to simulate the variable battery voltage, and the outputs of the microcontroller were observed utilizing a logic probe as the voltage was varied. This testing was done without the battery, or the shunt regulator present.

4.5 Safety Implementations

Safety implementations were added to the system to protect the customers and users. First, all high current carrying wire and devices are enclosed in a metal casing. The casing is grounded to the negative terminal of the battery, which can also be grounded to Earth Ground. Second, all the wires used are appropriately sized and gauged according to the American Wire Gauge. Fuses are placed on the high current carrying wire to protect the wire insulation and system components. Thirdly, normally open relays are placed in between the PV panels and battery to protect the battery. This relay can be seen in figure 4.11 as Relay 1. This relay will disconnect the PV panels from the battery in the event that the microcontroller loses power because the microcontroller is regulates the battery charging subsystem. By disconnecting the PV panels from the battery, overcharging of the battery will be prevented. Lastly, another relay is placed between the battery and the DC output. The purpose of this normally open relay (Relay 2 in figure 4.11) is to disconnect the load from the battery when the battery voltage becomes too low. The disconnection is necessary to prevent the battery to be completely once it reaches its low voltage state.

Fuses of the appropriate size and ratings are placed to limit the current through each subsystem and outputs. Also, there is an on/off switch which controls the AC output.

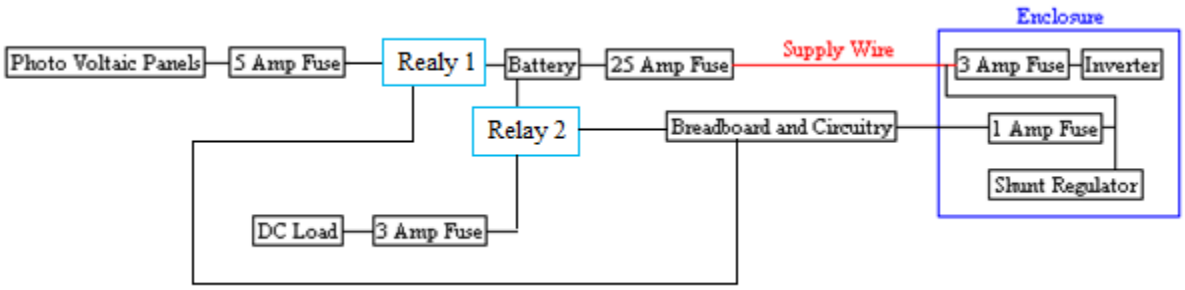


Figure 4. 11 : Safety Fuses and Relays

Figure 4.12 shows the metal enclosure with the lid open. Here is where all the high current wire, buses, and components are stored. The enclosure is well ventilated to allow air circulation. Not visible in figure 4.12 is the base of the enclosure with ventilating holes through the bottom of the case. Figure 4.13 shows the enclosure with the lid on. It can be seen that there more ventilating slots along the sides of the casing. Also, the wires coming through the front are held in place with strain relief devices to prevent stripping of the insulation and wire damage.

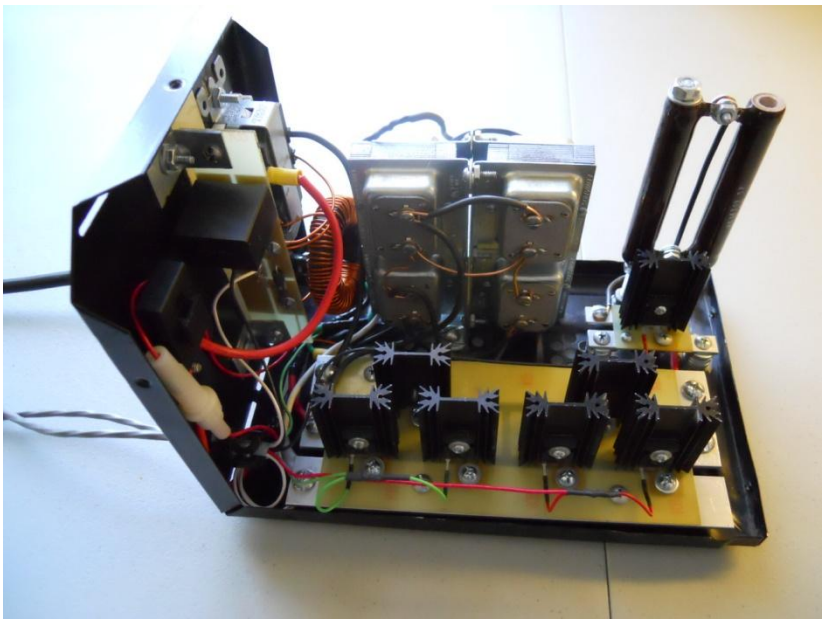


Figure 4. 12 : Open Lid Enclosure

Figure 4. 13 : Closed Lid Enclosure

Chapter 5

Conclusion

The Intelligent Alternative Energy Controller works as a standalone system, powered directly off a 12VDC battery. The system is able to take energy from the photovoltaic panels and wind turbine and use it to charge the battery. The battery will then provide the 12VDC output to a DC load, supply power to the microcontroller and signal level components, and will be used to generate an AC output. All the while, the voltage, phase, frequency, and power factor of the AC out will be taken and uploaded to the EE server. Currently the issues found are that the AC voltage is low, current measurement is very noisy, and few subsystems of total integrated system are not working.

Currently the system will allow the wind turbine to connect directly to the battery because of its internal regulation, the PV panels however, are connected to the charge controller, which then feeds the battery. The 12VDC output connects to the load through a 5 amp fuse and a normally-open relay used to disconnect the load when battery is low. The inverter has a manual on/off switch to the AC output in order to prevent power loss when not in use. Several safety implementations were added to protect the system, the battery, and the loads. Fuses were properly placed, and the enclosing case is well ventilated and grounded when the natural end of the battery is grounded. For convenience, a three terminal AC outlet is used as the connection between the AC load and the inverter. The current, voltage, power factor, and frequency measurements can be uploaded on to the EE web-server. The uploaded information can be seen through the internet at http://geek.nmt.edu/~JD2010_team1/ . Login name and password are needed to view the page if trying to access the site through a different server, other than the EE server. If already connected to the EE server, the login name and password are needed to clear data old data.

The AC output voltage is about 100V peak with nominal voltage in the battery, which is equivalent to about 70.7 V_{rms}. The project specification required a 120Vrms output, so this goal was not attained. A possible solution is to use higher switching frequency for the inverter pulse train. Implementation of this would require higher frequency components due to propagation delays through each component.
A second issue found during implementation it that the signal coming from the current sense resistor from the AC output is very low and noisy. The signal is amplified before it is measured and sent to the comparator circuit, but once reaching this stage, the noisy signal causes the op-amps to saturate to the positive and negative rails.
When the entire system was integrated together, portions of the code for the microcontroller were causing the entire program to crash. It is believed to be the functions where the status of the system is read, but further debugging is still required.

5.1 Future Work

Since the testing of this project was done in a lab setting, controlled sources were used to simulate the battery voltage that was measured by the microcontroller, thus allowing microcontroller the turn on and off the load DC load as well as the control the battery charger. For that reason, the real effect of the varying battery voltage was never tested. To accomplish the proper testing of the circuit, several battery states are needed: low voltage (dead battery), nominal voltage, full voltage. Without a quick and time efficient way of charging and discharging the provided battery (by means other than our system, to ensure it is done properly), it became very unpractical to constantly monitor the entire system while battery voltage was transitioning between its high, low, and nominal states. As a result, a weather proof casing should be used to protect the components if the system were to be left alone (unmonitored) for long periods of time. Also, more safety precautions will need to be added to protect the system (both internally and externally) and the surrounding environment from any system failures.

Active noise filters are recommended to reduce the noise one the current sense resistor. Presently, the signal being sent to comparator is too small and noisy to accurately measure the current of the AC load.

The currents needed to operate the entire system are fairly inefficient due to the relays used in the safety implementations. An improvement of this design would be to use MOSFETs with pull down resistors. By using MOSFETS in place of the relays, the system with would draw about 200 mA less current.

References:

[1] HURD Solar: Renewable Energy
http://ghurd.info/

[2] *Mathematical Manipulation of Pure Sine Wave Inverter Using Atmel 89S2051*
Didik Rostyono & Harsono Hadi

[3] *Microelectronic Circuits, Fifth edition*
Adel S. Sedra and Kenneth C. Smith

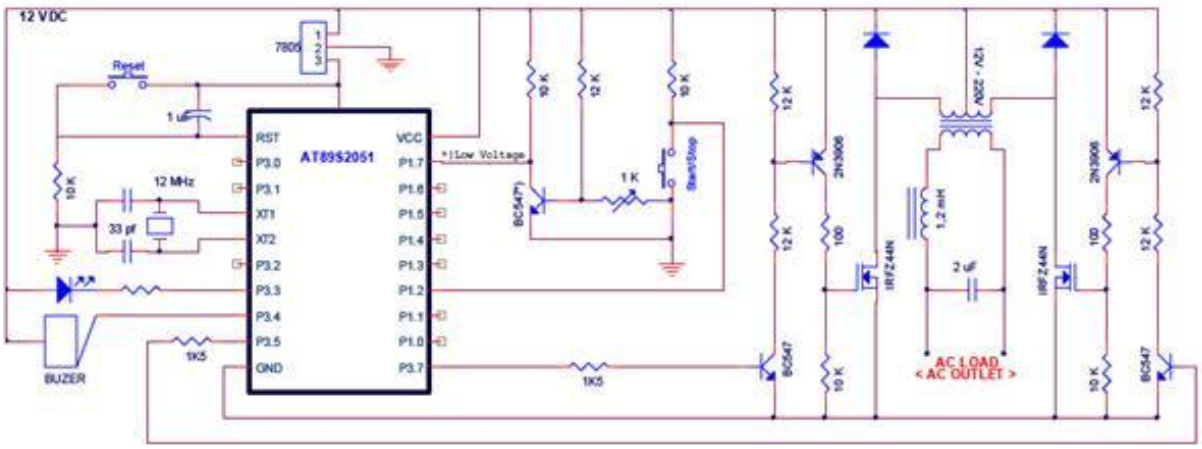
[4] *The Art of Electronics, Second edition*
Paul Horowitz and Winfield Hill

[5] *Engineering Circuit Analysis*
William Hayt, Jack Kemmerly, Steven Durbin

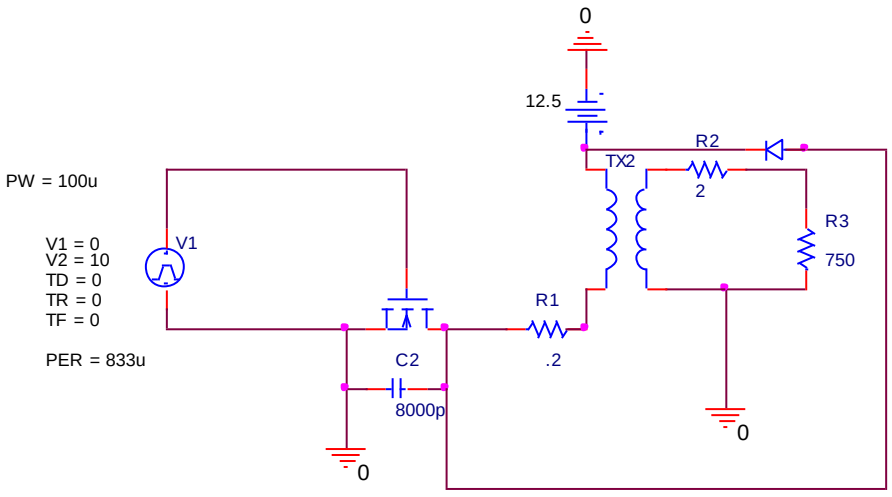
Appendix

Project Expenses

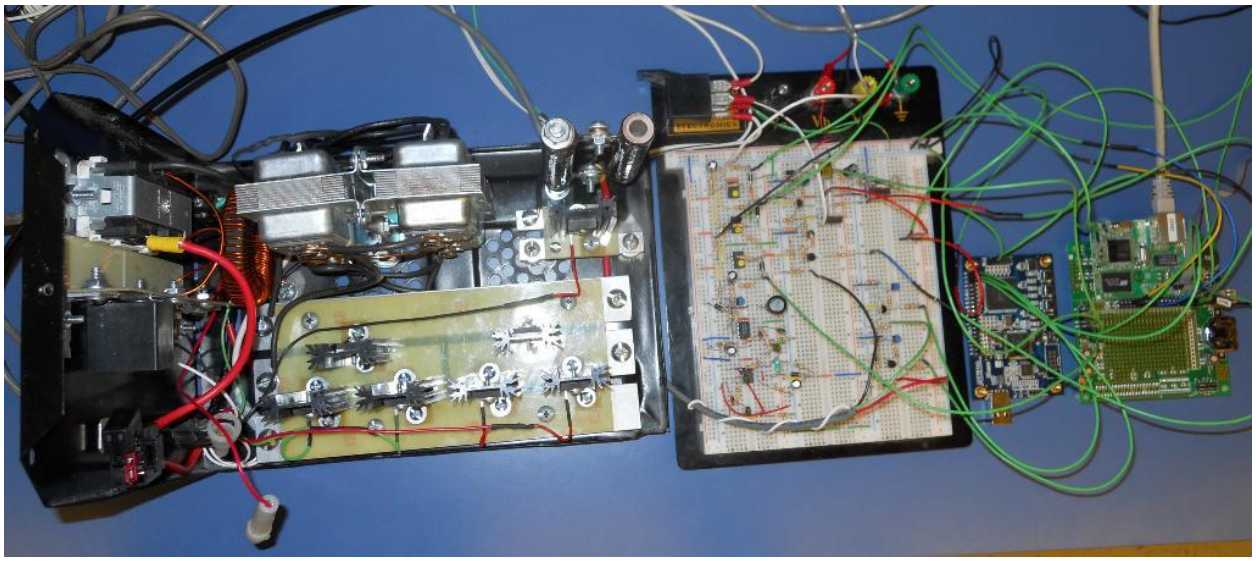
ITEMS	QUANTITY	PRICE/QUANTITY	SUB TOTAL
MOSFET (High Current)	5	$ 2.06	$ 10.30
MOSFET (Low Current)	1	$ 1.41	$ 1.41
Diodes (High Current)	2	$ 3.18	$ 6.36
Diodes (Low Current)	6	$ 0.02	$ 0.12
Capacitor (High Voltage)	1	$ 1.52	$ 1.52
Capacitor (Low Voltage)	10		$ 2.00
Sense Resistors	3	$ 1.26	$ 3.78
PCB board	1	$ 8.81	$ 8.81
Heatsink	7	$ 0.60	$ 4.16
Ferrite Core	1	$ 3.07	$ 3.07
Schottky Diode (15v)	2	$ 3.18	$ 6.36
Opamp	3	$ 1.54	$ 4.62
Voltage Regulators	3		$ 4.62
BJT	9	$ 0.20	$ 1.80
Inductors	2		$ 0.50
Fuse Holder & Fuses	4		$ -
Switch/AC outlet	1		$ -
Current Sense Resistors	3	$ 1.26	$ 3.78
FPGA			$ -
Transformer (5VA)	4	$ 13.00	$ -
Resistors	44		$ -
Rabbit RCM4000	1	$ 249.00	$ -
Metal Enclosure	1		$ -
Miscellaneous Items			$ 20.78
Total			**$ 83.99**



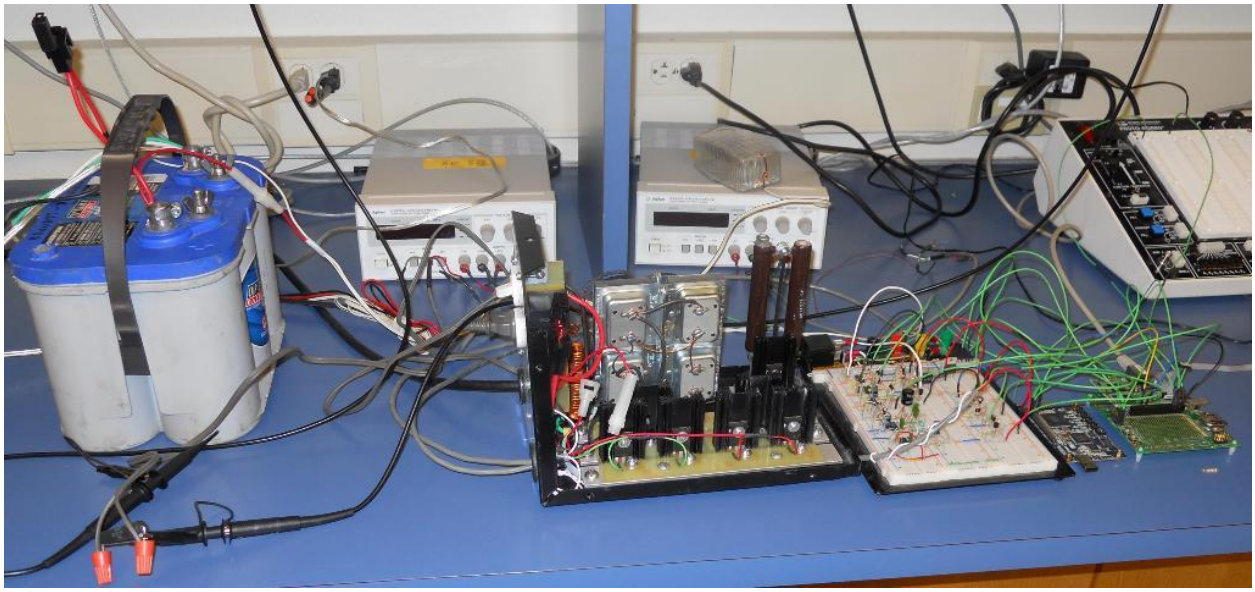
Appendix Figure 1 : Reference Inverter Circuit



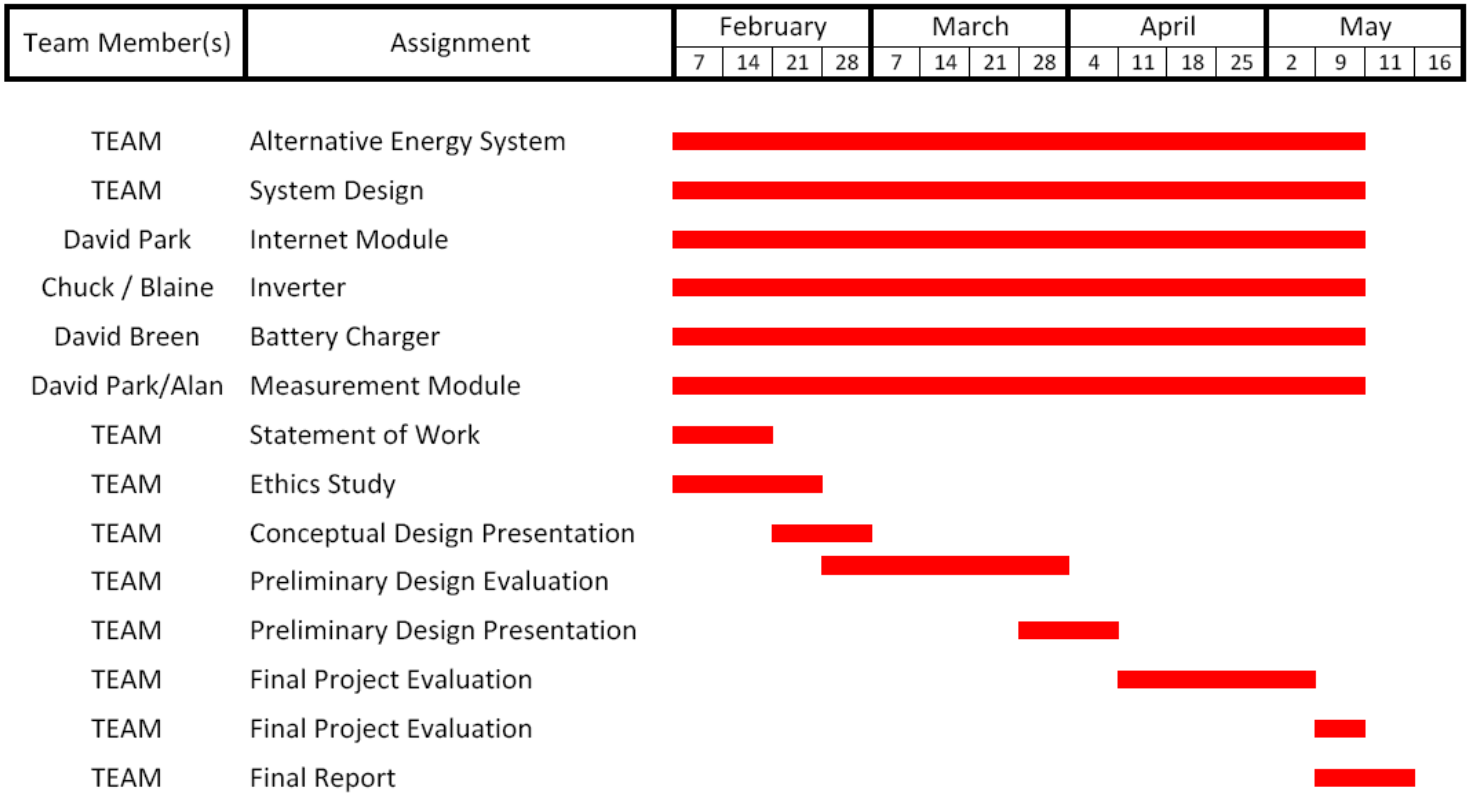
Appendix Figure 2 : Simulated Inverter Circuit



Appendix Figure 3 : System Setup without Battery



Appendix Figure 4 : System Setup with Battery



Appendix Figure 5 : Gantt Chart

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

   core.c
   David Park
   Alan Huynh

   Last updated: 4/27/2010

    Description
    ===========
    This program is the core of the project, and is responsible
    for measurements and communicating with the server.

    Instructions:
    -------------
    Load the code. Run it.

*****************************************************************************/

//ADC declarations////////////////////////////////////////////////////////
#class auto
#define ADC_SCLKBRATE 115200ul // 115200
#use RCM40xx.LIB
#include "math.h"
#ifndef ADC_ONBOARD
   #error "This core module does not have ADC support.  ADC programs will not "
   #fatal "   compile on boards without ADC support.  Exiting compilation."
#endif
#define NUMSAMPLES 10       //change number of samples here
#define STARTCHAN 0
#define ENDCHAN 2
#define GAINSET GAIN_1      //other gain macros
                                //GAIN_1     (0 - 20.00 Volts)
                                //GAIN_2     (0 - 11.11 Volts)
                                //GAIN_4     (0 -  5.56 Volts)
                                //GAIN_5     (0 -  4.44 Volts)
                                //GAIN_8     (0 -  2.78 Volts)
                                //GAIN_10    (0 -  2.22 Volts)
                                //GAIN_16    (0 -  1.39 Volts)
                                //GAIN_20    (0 -  1.11 Volts)

//////////////////////////////////////////////////////////////////////////
//Http client declarations////////////////////////////////////////////////

// define HTTPC_VERBOSE to turn on verbose output from the HTTP Client library
#define HTTPC_VERBOSE

/*
 * NETWORK CONFIGURATION
 * Please see the function help (Ctrl-H) on TCPCONFIG for instructions on
 * compile-time network configuration.
 */

//for dhcp configuration, TCPCONFIG 5
//for manual cinfiguration, TCPCONFIG 1
#define TCPCONFIG           1
//manual options
#define _PRIMARY_STATIC_IP  "10.0.0.76"
#define _PRIMARY_NETMASK    "255.255.255.0"
#define MY_GATEWAY          "10.0.0.254"
#define MY_NAMESERVER       "10.0.0.254"

///// End of Configuration Options /////
```

```
//  Set a default of declaring all local variables "auto" (on stack)
#class auto

// default functions to xmem
#memmap xmem

#use "dcrtcp.lib"

#use "http_client.lib"

////////////////////////////////////////////////////////////////////////
//Begin function declarations/////////////////////////////////////////////
void http_header(tcp_Socket *sock,  float ad_inputs[ENDCHAN+3])
{
    char url[200];
    httpc_Socket hsock;
    int retval;
    int is_text;
    char far *value;
    sprintf(url,"http://geek.nmt.edu/~JD2010_team1/data_write.php?batt_voltage=%f&load
_voltage=%f&load_current=%f&frequency=%f&pf=%f&submit=Go",ad_inputs[0],ad_inputs[1]/.7
43,ad_inputs[2]/28.6667/.15,ad_inputs[ENDCHAN+1],ad_inputs[ENDCHAN+2]);

   //The initialization function httpc_init_if() must be called before any other HTTP
client function.
   //This function lets you specify an interface. httpc_init() is a macro for httpc_in
it_if(), using IF_ANY as the interface designator.
   retval = httpc_init (&hsock, sock);
   if (retval)
   {
      printf ("error %d calling httpc_init()\n", retval);
   }
   else
   {
    //Connect to the HTTP server referenced in "url" and GET the resource referenced i
n "url".
      //This function is like httpc_get(), but with a URL instead of separate host, au
thentication, port and file parameters.
    retval = httpc_get_url (&hsock, url);
      if (retval)
      {
        printf ("error %d calling httpc_get_url()\n", retval);
      }
      //else
      //{
          //    printf ("Data sent\n");
     // }
   }

}

// set the STDIO cursor location and display a string
void DispStr(int x, int y, char *s)
{
   x += 0x20;
   y += 0x20;
   printf ("\x1B=%c%c%s", x, y, s);
}

////////////////////////////////////////////////////////
// millisecond delay
////////////////////////////////////////////////////////
nodebug
void msDelay(unsigned int delay)
```

```
{
    auto unsigned long time0;

    for (time0 = MS_TIMER; MS_TIMER - time0 < delay; ) ;
}

////////////////////////////////////////////////////////////////////////

// read the A/D with using the low level driver
nodebug
unsigned int sample_ad(int channel, int num_samples)
{
    auto unsigned long rawdata;
    auto unsigned int sample;
    auto unsigned int cmd;

    //convert channel and gain to ADS7870 format
    // in a direct mode
    cmd = 0x80|(GAINSET*16+(channel|0x08));

    for (rawdata=0, sample=num_samples; sample>0; sample--)
    {
        // execute low level A/D driver
        rawdata += (long) anaInDriver(cmd);
    }
    return ((unsigned int) (rawdata/num_samples));
}

float convert_volt(int channel, int value)
{
    auto float voltage;

    // convert the averaged samples to a voltage
    voltage = (value - _adcCalibS[channel][GAINSET].offset) * (_adcCalibS[channel][GAI
NSET].kconst) * 9;

    return voltage;
}

// It's safer to keep sockets as globals, especially when using uC/OS-II.  If
// your socket is on the stack, and another task (with its own stack, instead
// of your task's stack) calls tcp_tick, tcp_tick won't find your socket
// structure in the other task's stack.
// Even though this sample doesn't use uC/OS-II, using globals for sockets is
// a good habit to be in.
tcp_Socket demosock;

main ()
{
    // frequency and power factor variables
    unsigned long startf, endf, startp, endp, difff, diffp;
    float freq = 0;
    float phase = 0;
   char freqtxt[16], phasetxt[160];
   int flag = 0;

    // ADC variables
    auto int channel, cycle;
    auto unsigned int avg_sample;
    auto char s[80];
    auto char display[80];

    auto float ad_inputs[ENDCHAN+3];
```

```
    ///HTTP client initilization//////////////////////////////////////////////
   // initialize tcp_Socket structure before use
    memset( &demosock, 0, sizeof(demosock));

   // initalize board
    brdInit();

    // initalize HTTP client
   printf ("http client v" HTTPC_VERSTR "\n");

    printf ("Initializing TCP/IP stack...\n");
    sock_init_or_exit(1);

    // initially start up A/D oscillator and charge up cap
    anaIn(0,SINGLE,GAINSET);

    //DispStr(1, 1, "\t\t\t<<< Voltage Readings (V) >>>");
    //DispStr(1, 3, "\t Solar\t Wind\t Batt.\t DCL\t ACL\t LN5IN\t LN6IN");
    //DispStr(1, 4, "\t------\t------\t------\t------\t------\t------\t------");

    // frequency and power factor initilization
   // configure Port E as input
    WrPortI(PEDDR, &PEDDRShadow, 0x00);
   startf = 0;
   endf = 0;
   startp = 0;
   endp = 0;
   difff = 0;
   diffp = 0;

   // control routine initialization
   // configure Port A as output
   WrPortI(SPCR, &SPCRShadow, 0x084);
   //initilize port A values (pulse on, shunt off, load on)
   WrPortI(PADR,&PADRShadow,0xFD);

   //wait for stuff to initilize
   printf("Initializing Ports....\n");
   msDelay(1000);

    // Main routine
    while (1)
    {
        // ADC: voltage and current measurements
        //display[0] = '\0';
        for(channel = STARTCHAN; channel <= ENDCHAN; channel++)
        {
            // sample each channel
            avg_sample = sample_ad(channel, NUMSAMPLES);
            ad_inputs[channel] = convert_volt(channel, avg_sample);
         //sprintf(s, "\t%6.3f", ad_inputs[channel]);
            //strcat(display, s);
        }

       // ADC: self-monitor/control routine
      // PA0 connects to load relay, PA1 connects to shunt load relay, PA2 connects to
 pulse train enable/reset
      // check for normal operation

      if (ad_inputs[0] <= 14.0)
      {
      // Disable/keep disabled the shunt load, output PA1 low
      printf("batt voltage:%f shunt off\n", ad_inputs[0]);
      WrPortI(PADR, &PADRShadow,PADRShadow & 0xFD);
```

```
        // if battery voltage < 11V, stop supplying power to load and make pulse train
 low
        if (ad_inputs[0] < 11.0)
        {
            //output PA0 low and PA1 low and PA2 low
            printf("batt voltage:%f all off\n", ad_inputs[0]);
            WrPortI(PADR, &PADRShadow,PADRShadow & 0xF8);
        }
        //otherwise keep supplying power, output PA0 high, pulse PA2
        else
        {
            printf("batt voltage:%f loads on\n", ad_inputs[0]);
            WrPortI(PADR, &PADRShadow, PADRShadow | 0x05);
        }
      }

      //if battery voltage is over 14V, turn on/keep on shunt load, while turning/keep
ing the pulse train on (high).
      else
      {
        //output PC1 high and PC2 high
          printf("batt voltage:%f pulse shunt on\n", ad_inputs[0]);
        WrPortI(PADR,&PADRShadow, PADRShadow | 0x7);
      }

    //  //if pulse train is enabled measure frequency and power factor
        if (PADRShadow & 0x04 != 0x00)
        {
        printf("measuring frequency\n");
            // Frequency and power factor measurements
            // wait for passing high signal, if any
            while ((RdPortI(PEDR) & 0x01) != 0x00) { }

            // wait for low signal
            while ((RdPortI(PEDR) & 0x01) != 0x01) { }

            // now we can measure frequency, timestamp high signal
                if ((RdPortI(PEDR) & 0x01) != 0x00)
            {
                // frequency timestamp
                startf = MS_TIMER;
            }

            // wait until high signal goes low
            while ((RdPortI(PEDR) & 0x01) != 0x00)
              {
                // wait while checking to see if 2nd signal is high
                 // and get time both are high
                if (((RdPortI(PEDR) & 0x03) == 0x03) && (flag == 0))
                 {
                    // start phase timestamp
                    startp = MS_TIMER;
                    flag = 1;
                 }

                 if (((RdPortI(PEDR) & 0x03) != 0x03) && (flag == 1))
                 {
                    // end phase timestamp
                    endp = MS_TIMER;
                    flag = 2;
                 }
              }
```

```
            // while loop exits next time a low is encountered, take time difference
            endf = MS_TIMER;

              if (flag == 1)
              {
                endp = endf;
              }

              // else printf("current is leading\n");

                //printf("startp %d\n",startp);
            difff = 2 * (endf - startf);
              diffp = (endp - startp);
              //printf("diffp %d\n",diffp);

            // calculate frequency and phase shift
                freq = (float) 1 / difff;
              phase = (float) 180.0 - (180.0 * diffp / difff);
            //printf("Frequency: %f Hz", freq * 1000);
            //printf("\nPhase shift: %f degrees\n\n", phase);
              flag = 0;
           ad_inputs[ENDCHAN+1] = freq * 1000;
              ad_inputs[ENDCHAN+2] = cos(phase);

        }
        else
        {
        ad_inputs[ENDCHAN+1] = 0;
            ad_inputs[ENDCHAN+2] = 0;
        }
      printf("Frequency measured!! phase is %f", phase);

      //DispStr(1, 5, display);
        // store frequency and power factor values to AD array



        // send it off to the server
        http_header(&demosock, ad_inputs);
        printf("Data sent\n");

    // repeat every 10 seconds
        msDelay(10000);
    }
}
```

```
1   module pulse_train (output reg pos_train, neg_train, input clk,
    reset);
2       /*Coded by David Park
3       4/7/2010
4       Module to genarate varing width pulse train for inverter.
5       total train length is 833333 cycles (16.6667e-3sec, 60 hz)
6       total pulse length is 41667 cycles (833.33e-6 sec)
7
8       pulse widths vary as multiples of 50 us*/
9
10      parameter PERIOD_CYCLES = 41667;
11
12      reg [19:0] count;
13
14      always @ (posedge clk, negedge reset)
15      //keep output low while reset is low.
16      if (~reset)
17          begin
18              count<=0;
19              pos_train<= 1'b0;
20          end
21      else
22          begin
23          //clock pulse 1 7500 on cycles (3*50us)
24          if (count<7500)
25          begin
26              pos_train = 1'b1;
27              count<=count + 1'b1;
28          end
29          //clock pulse 2 15000 on cycles (6*50us)
30          else if ((count>PERIOD_CYCLES) && (count<(PERIOD_CYCLES +
    15000)))
31          begin
32              pos_train = 1'b1;
33              count<=count + 1'b1;
34          end
35          //clock pulse 3 22500 on cycles (9*50us)
36          else if ((count>(2*PERIOD_CYCLES)) && (count<(2*
    PERIOD_CYCLES + 22500)))
37          begin
38              pos_train = 1'b1;
39              count<=count + 1'b1;
40          end
41          //clock pulse 4 30000 on cycles (12*50us)
42          else if ((count>(3*PERIOD_CYCLES)) && (count<(3*
    PERIOD_CYCLES + 30000)))
43          begin
44              pos_train = 1'b1;
45              count<=count + 1'b1;
46          end
47          //clock pulse 5 37500 on cycles (15*50us)
```

```
48          else if ((count>(4*PERIOD_CYCLES)) && (count<(4*
    PERIOD_CYCLES + 37500)))
49          begin
50              pos_train = 1'b1;
51              count<=count + 1'b1;
52          end
53          //clock pulse 6 37500 on cycles (15*50us)
54          else if ((count>(5*PERIOD_CYCLES)) && (count<(5*
    PERIOD_CYCLES + 37500)))
55          begin
56              pos_train = 1'b1;
57              count<=count + 1'b1;
58          end
59          //clock pulse 7 30000 on cycles (12*50us)
60          else if ((count>(6*PERIOD_CYCLES)) && (count<(6*
    PERIOD_CYCLES + 37500)))
61          begin
62              pos_train = 1'b1;
63              count<=count + 1'b1;
64          end
65          //clock pulse 8 22500 on cycles (9*50us)
66          else if ((count>(7*PERIOD_CYCLES)) && (count<(7*
    PERIOD_CYCLES + 22500)))
67          begin
68              pos_train = 1'b1;
69              count<=count + 1'b1;
70          end
71          //clock pulse 9 15000 on cycles (6*50us)
72          else if ((count>(8*PERIOD_CYCLES)) && (count<(8*
    PERIOD_CYCLES + 15000)))
73          begin
74              pos_train = 1'b1;
75              count<=count + 1'b1;
76          end
77          //clock pulse 10 7500 on cycles (3*50us)
78          else if ((count>(9*PERIOD_CYCLES)) && (count<(9*
    PERIOD_CYCLES + 7500)))
79          begin
80              pos_train = 1'b1;
81              count<=count + 1'b1;
82          end
83          //clock pulse 11 7500 on cycles (3*50us)
84          else if ((count>(10*PERIOD_CYCLES)) && (count<(10*
    PERIOD_CYCLES + 7500)))
85          begin
86              neg_train = 1'b1;
87              count<=count + 1'b1;
88          end
89          //clock pulse 12 15000 on cycles (6*50us)
90          else if ((count>(11*PERIOD_CYCLES)) && (count<(11*
    PERIOD_CYCLES + 15000)))
```

```
 91          begin
 92              neg_train = 1'b1;
 93              count<=count + 1'b1;
 94          end
 95          //clock pulse 13 22500 on cycles (9*50us)
 96          else if ((count>(12*PERIOD_CYCLES)) && (count<(12*
     PERIOD_CYCLES + 22500)))
 97          begin
 98              neg_train = 1'b1;
 99              count<=count + 1'b1;
100          end
101          //clock pulse 14 30000 on cycles (12*50us)
102          else if ((count>(13*PERIOD_CYCLES)) && (count<(13*
     PERIOD_CYCLES + 30000)))
103          begin
104              neg_train = 1'b1;
105              count<=count + 1'b1;
106          end
107          //clock pulse 15 37500 on cycles (15*50us)
108          else if ((count>(14*PERIOD_CYCLES)) && (count<(14*
     PERIOD_CYCLES + 37500)))
109          begin
110              neg_train = 1'b1;
111              count<=count + 1'b1;
112          end
113          //clock pulse 16 37500 on cycles (15*50us)
114          else if ((count>(15*PERIOD_CYCLES)) && (count<(15*
     PERIOD_CYCLES + 37500)))
115          begin
116              neg_train = 1'b1;
117              count<=count + 1'b1;
118          end
119          //clock pulse 17 30000 on cycles (12*50us)
120          else if ((count>(16*PERIOD_CYCLES)) && (count<(16*
     PERIOD_CYCLES + 30000)))
121          begin
122              neg_train = 1'b1;
123              count<=count + 1'b1;
124          end
125          //clock pulse 18 22500 on cycles (9*50us)
126          else if ((count>(17*PERIOD_CYCLES)) && (count<(17*
     PERIOD_CYCLES + 22500)))
127          begin
128              neg_train = 1'b1;
129              count<=count + 1'b1;
130          end
131          //clock pulse 19 15000 on cycles (6*50us)
132          else if ((count>(18*PERIOD_CYCLES)) && (count<(18*
     PERIOD_CYCLES + 15000)))
133          begin
134              neg_train = 1'b1;
```

```
135                 count<=count + 1'b1;
136             end
137             //clock pulse 20 7500 on cycles (3*50us)
138             else if ((count>(19*PERIOD_CYCLES)) && (count<(19*
        PERIOD_CYCLES + 7500)))
139             begin
140                 neg_train = 1'b1;
141                 count<=count + 1'b1;
142             end
143             else
144             //clock pulse end
145             begin
146                 pos_train = 0;
147                 neg_train = 0;
148                 count<=count + 1'b1;
149             end
150             //end of 60 hz cycle
151             if (count>833333)
152                 begin
153                     count<=1'b0;
154                 end
155         end
156     endmodule
```