DC-AC/DC Power Inverter

Team Not Platypus

Matthew Brown
Henry Godman
John Martinez
Dylan Paiton
Matthew Paiz

May 12, 2010
Abstract

An intelligent DC-DC/AC converter system was designed and implemented in the Spring of 2010 for New Mexico Tech’s Junior Design Class. The intelligent converter draws power from two energy harvesters; a 400W-12V Sunforce Wind Generator and a 60W-12V Sunforce Solar PV kit. The power is stored in an Optima 12V sealed lead acid battery. The inverter is comprised of five major subsystems: smart battery charger, inverter, measurement system, data logger and internet interface. Components were selected through decision matrices and purchased online or procured through the Electrical Engineering department at NMT. Circuits were designed in Protel 99SE and created from etching and milling processes. Data was sent via HTTP to the EE server on the NMT campus and displayed real-time information on a web page. Operation of each subsystem was demonstrated independently and in whole.
Acknowledgements

We would like to take a moment to thank the following people for all of their help on this project. Dr. Erives gave us constant guidance, constructive critique and useful advice throughout the semester. Dr. Jorgenson was kind enough to let us figure out how to use his milling machine with no prior experience. Dr. Wedeward and Dr. Rison both helped us with their expertise when we ran into walls. Norton was a great resource this semester. He let us use materials from his shop and found exactly what we needed more than a few times. David Park helped with the initial setup of the milling machine. Andy Tubesing and Chris Pauli were always available to help us get all the miscellaneous parts that made our project a success.

- Dr. Erives
- Dr. Jorgensen
- Dr. Wedeward
- Dr. Rison
- Norton Euart
- David Park
- Andy Tubesing
- Chris Pauli
# Contents

1 Introduction ........................................ 8

2 Background ........................................... 9

3 Design and Implementation .......................... 11

   3.1 Overall Design .................................... 11

   3.2 Field Components and Field Testing ............. 11

      3.2.1 Wind Generator ............................... 11

      3.2.2 Wind Generator Regulation Data ............. 12

      3.2.3 Solar Array ................................... 12

      3.2.4 Solar Panel Regulation Data ................. 13

      3.2.5 Battery ....................................... 13

   3.3 DC to DC/AC Power Inverter ...................... 13

      3.3.1 Introduction .................................. 13

      3.3.2 Background ................................... 14

      3.3.3 Design ....................................... 14

      3.3.4 Implementation ............................... 16

   3.4 Smart Battery Charger ............................. 19
A  Code For Data Logger  

B  Code For Internet Interface
## List of Tables

2.1 Intelligent Power Functions ................................................. 9

3.1 Decision Matrix for Data Logger ............................................. 26

3.2 Decision Matrix for Internet Interface ................................. 32
List of Figures

3.1 Level One Schematic of Intelligent Converter ............................... 12
3.2 Example of Various Waveforms ............................................... 15
3.3 A Diagram of The SG3524 IC .................................................. 16
3.4 Driver Schematic for Inverter Circuit ...................................... 17
3.5 Power Stage ........................................................................... 17
3.6 Filter Stage ............................................................................ 18
3.7 Driver Stage Output ............................................................... 18
3.8 Finished Inverter Circuit ......................................................... 19
3.9 Three Stage Charge ............................................................... 22
3.10 Battery Trickle Charger Results .............................................. 24
3.11 Protel PCB Schematic .......................................................... 24
3.12 Real Time Voltage ............................................................... 27
3.13 Client Mode Operation .......................................................... 27
3.14 Example of Various Waveforms .............................................. 28
3.15 Protel Schematic ................................................................. 30
3.16 Measurement Circuit Prototype .............................................. 30
3.17 Progress Comparison For Each Subsystem ............................... 34
Chapter 1

Introduction

Today we are seeing a fast growing availability of renewable energy harvesters. While the energy conversion process of these devices produces little to no pollution, the power that is generated is often intermittent and unreliable. This is best exemplified by the wind generator and photovoltaic (PV) solar cells; both work well while the wind is blowing and the sun is shining, but fail to produce when they are not. The purpose of this project is to try to take those two sources and make a power system that can provide, at least somewhat, a more stable source of energy. To accomplish this, our team has designed a system where the energy produced from both of these sources is stored into a sealed lead acid battery. An inverter is connected in series with this battery, producing 120VAC at 60Hz to emulate wall power. Power and signal characteristics will be measured, logged, and displayed in real time on the Electrical Engineering server at New Mexico Tech. The project design was implemented according the specifications for the spring 2010 EE382 guidelines [1].

The following paper outlines the system we developed. First we describe the background of intelligent chargers, and then we discuss each individual subsystem in detail. For symmetry, we included individual conclusions for each subsystem, stating its current progress. There is an overall conclusion at the end of the paper as well, which gives a broader overview of the status of the project. The scope of this project was to develop a working prototype for the inverter circuit. It was suggested about three fourths of the way into the project that we develop an enclosure for the system as well, but a working prototype with proper output was the primary concern.
Chapter 2

Background

Intelligent Inverters are currently available on the market. Microchip Technologies provides a detailed list for the functions of an intelligent inverter (below).

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital On/Off control for low standby power</td>
</tr>
<tr>
<td>Power supply sequencing and hot-swap control</td>
</tr>
<tr>
<td>Programmable soft-start profile</td>
</tr>
<tr>
<td>Power supply history logging and fault manage</td>
</tr>
<tr>
<td>ment</td>
</tr>
<tr>
<td>Output voltage margining</td>
</tr>
<tr>
<td>Current fold back control</td>
</tr>
<tr>
<td>Load sharing and balancing</td>
</tr>
<tr>
<td>Regulation reference adjustment</td>
</tr>
<tr>
<td>Compensation network control and adjustment</td>
</tr>
<tr>
<td>Full digital control of power control loop</td>
</tr>
<tr>
<td>Communications for status monitoring and con-</td>
</tr>
<tr>
<td>trol</td>
</tr>
<tr>
<td>AC RMS voltage measurement</td>
</tr>
<tr>
<td>Power factor correction</td>
</tr>
</tbody>
</table>

Table 2.1: Intelligent Power Functions

They are designed for both grid-tie and off the grid applications. They operate much like uninterruptable power supplies (UPS). The main goal is be able to supply power to a load directly from a main power source, be it a generator or a wall outlet while available and continue to provide constant power when that main source of power goes offline.

A range of technologies is implemented to meet the functions listed above. Computers, microcontrollers and FPGAs are all currently used to manage these sorts of systems. Square wave, modified sine wave and pure sine wave inverters are all implemented as well, depending on the load re-
quirements. Analog circuitry exploiting timers and op-amps are used to measure voltages and currents on these systems. ICs are also becoming more and more popular as devices are more available for high power applications. The same is true for battery charging circuitry. While the technology exists to easily display data about the system online, it is not employed often.

Meeting all of the functions above is beyond the scope of this project. For this design, our team will provide a proof of concept for an intelligent inverter system that can log data online in real time. The budget for the project was $400. Because of this limited budget, cost was a major factor when selecting components.
Chapter 3

Design and Implementation

3.1 Overall Design

Our intelligent converter will contain five major subsystems. (Figure 3.1) Each subsystem is designed to accomplish a unique goal. Each subsystem will be described in detail in the following sections. The first subsystem to be outlined is the smart battery charger. This system uses a three stage charging scheme to safely charge the battery from the photovoltaic panels. The second system is the data logger subsystem, which receives inputs from other systems and interprets them for the internet interface. The third subsystem in our overview is the measurement subsystem. This is used to record the output power characteristics to the load. The fourth subsystem is the internet interface, which displays the measured power characteristics to the end user. The fifth and final subsystem is the power inverter, which performs the actual inversion from the input renewable power sources to the output wall power.

3.2 Field Components and Field Testing

3.2.1 Wind Generator

The Air-X by Southwest Windpower Inc. was provided to test our implemented designs. It is rated for 12V operation with a boosting function that allows for power output at low RPM using a brushless permanent magnet alternator. It incorporates two main protection systems. The first is over-speed protection that dramatically reduces RPMs at around wind speeds of 35 mph and the second is hysteresis, which dramatically reduces RPMs when the battery voltage matches the regulated set point of 14.1V. When the voltage sensed in the generators internal meter mea-
sures 12.75V it will come out of regulation and begin another charge cycle. Peak power output is maintained through its microprocessor by altering the loading of the alternator [8].

### 3.2.2 Wind Generator Regulation Data

Minor testing was conducted on the wind generators’ performance characteristics. We placed the generator outside on a breezy day and applied different valued resistors ranging from 10k-10 Ohms. Whenever these loads were applied the generator stopped abruptly. We found that if there is to high of a resistance, the generator will stop supplying current to keep from forcing current to its lead terminals that could induce an Electro-static discharge (ESD). An ESD will usually damage circuitry and render our intelligent convert useless. The battery was tested to have a low internal resistance (less than one Ohm). As such, placing the sealed lead acid as the load to the generator allowed it to reach high RPMs. A voltmeter was placed across the battery terminals and an ammeter was placed between the positive terminal of the generator and battery. This configuration kept the generator from stalling and gave a reading of 12.24V-12.89V depending on wind speed. The current ranged from 1.5A- 3.2A with a spike in current no more than 4.5A from gusts.

### 3.2.3 Solar Array

The solar panel array provided is rated at 60 Watts, 12V. The array is made by SunForce and uses four 15 Watt amorphous solar panels. The array is rated at supplying 5 Amps maximum. Solar panels on their own are not regulated energy harvesters and therefore need regulation circuitry that can also maintain maximum power output. Additionally, the regulation circuitry needs to
protect the array from current flowing back into itself, which can cause damage to the crystalline silicone structure that converts photons into electrons [9].

### 3.2.4 Solar Panel Regulation Data

During the research portion of the project we recorded measurements of the solar panels voltage and current output with the solar panels loaded with different resistances. These measurements provided data to graph the panels power curve showing that the panels output a relatively linear current drop from 4.25-3.25A from 1-16V voltage range, then the current drops from 3.25-0.5A from 16-23V. This gave us a power curve relationship that was graphed to show where the ‘knee’ point that can determine the maximum power we can harvest from the solar panel array.

### 3.2.5 Battery

Lead acid batteries were developed by a French physician Gaston Plante in 1859 and are the oldest and most widely used electrical storage unit [7]. They can take a fare amount of abuse, high discharge rates and fast charging, but need to be maintained at full charge when not used to ensure that degeneration of the plates integrity and depth of discharge is not compromised. For long term storage lead acid has the most favorable characteristics, losing 40% of its charge over one year as compared to 6 months with NiMH [8]. State of charge can be determined from the terminal voltage, though care should be taken since a cold battery has higher than expected value while a hot one has lower than expected voltage which can prevent a charge cycle from terminating causing over-saturation and potentially damaging itself and nearby equipment [10]. Allowing 4-8 hours of rest from charging will provide an accurate measurement of terminal voltage to determine state of charge.

### 3.3 DC to DC/AC Power Inverter

#### 3.3.1 Introduction

A DC-AC/DC power inverter is a circuit which modifies an input varying or non-varying direct current (DC) to an alternating current (AC) of a specified voltage and frequency, and a regulated DC voltage. In the case of this project, the input DC voltage source will be a battery, which is being supplied by photovoltaic (PV) panels and a wind turbine. As such, the DC voltage will likely be inconsistent, and considerations will need to be made in order to produce the desired output. This desired AC output is a 120Vrms, 60Hz pure sine wave, or what would be seen out of a standard
US wall socket. The desired DC output is a 12V regulated. This will allow the system to output power which is usable by any load.

### 3.3.2 Background

Power inverters available on the market today vary greatly in efficiency and output type. Generally, of higher end inverters, the output waveforms seen are either pure sine or modified sine. Another characteristic which determines the quality and price of an inverter is the power output in Watts.[22] There are many inverters which vary in their strengths/weaknesses regarding the points mentioned. In addition, many circuitry examples are available for those who wish to construct their own inverter.[23][24] In our case, we decided to model our circuit after that found in [23]. We used an integrated circuit (IC) in to produce a waveform of our desired frequency, and then used a transformer to amplify the voltage to specification.

### 3.3.3 Design

The only input to the inverter subsystem is from the battery. As described previously, the battery is being charged from the PV panels through the charging subsystem and from the wind turbine through its integrated system. Our key concerns regarding the power inverter system were as follows:

1. Safety - because we are dealing with high currents, many safety concerns needed to be accounted for
2. Output Waveform - pure vs. modified sine wave vs. square wave
3. Power Output needs to handle at least 500W
4. Efficiency generally there are a lot of losses associated with converting power

The inverter will receive DC power from the battery, and convert it to usable DC and AC outputs. All other subsystems receive information from the output of the power inverter. As such, the inverter is critical to the integrity of the entire system. Safety concerns must be at the forefront of the circuit design. These concerns stem to the safety of the users, as well as the circuitry itself. We recognized this issue, and accounted for it with properly placed kill-switches and fuses/circuit breakers. The circuit was designed so that if a power spike occurred, or something malfunctioned, the key components of the system would be saved, and the system would be shutdown.

Our next major concern was the output waveform. The three major waveform options were a square wave, modified sine wave, and pure sine wave. The difficulty of the system increases...
greatly with each waveform option, respectively. Figure 3.2 shows the three different waveforms on a single graph. A pure sine wave is required to run any type of load, but most loads can run on a modified sine wave. The primary side effect to a modified sine wave is a decrease in efficiency and an increase in noise in the system. Fewer (but still a significant amount of) loads can run on a square waveform. Because of the immense difficulty associated with designing a pure sine wave inverter, and the limited amount of loads which can run off of a square wave inverter, our team decided to go with a modified sine wave output. This can be obtained by producing a square wave from the output, and filtering it to resemble a sine wave. The desired result would look like an average of the three waveforms presented in Figure 3.2.

Figure 3.2: The vertical axis is Voltage, while the horizontal is Time. As can be seen, the red waveform is a pure sine wave, blue is a modified sine wave, and green is a square wave. The specified period (20ms) would vary based on desired frequency.

We used a SG3524 IC to generate the PWM signal, off of which we would create the square wave output. A general overview of the internal circuitry of this IC can be observed in Figure 3.3. The IC was the heart of our inverter circuit, and was heavily protected by fuses. The circuitry which utilized the SG3524 to create an output signal can be seen in Figure 3.4. Following the circuitry surrounding the SG3524 was the power amplification circuit.

The frequency-regulated output of the SG3524 was then sent to a power amplifying circuit, which created the desired output. To account for the output power required, we decided to mill the entire inverter circuit onto copper plating, so as to increase the allowed current flow. We also set up a parallel transistor configuration before the power was amplified. In order to dissipate the power delivered to the transformer. This can be seen in the power stage of the circuit, in Figure 3.5. The circuits displayed in figures ?? and 3.4 were connected via pads J3, J4, J8, and J9. Combined, these circuits represent the power inverter.

To accommodate for efficiency and power output concerns, we designed the entire circuit with bipolar junction transistors (BJTs). BJTs can typically handle much higher currents than the alternative options, MOSFETs. We also placed four BJTs in parallel in the power stage of the circuit.
Figure 3.3: Relevant pins: Pins 1 and 2 are used for reference pins, to verify that the output is as desired (using a feedback system described later). Pin 6 is used to determine the desired output frequency. Pins 7 and 9 are also reference pins. Pin 10 is a shutdown pin, which is used to stop the output. Pins 11 through 14 produce the square wave output. Pin 15 receives 8v DC power from a regulator.

to dissipate more power, observable in Figure 3.5.

The output of this circuit as shown is not as expected. It currently outputs a square wave, which works with any resistive load. The next stage in design would be to convert this into a modified sine wave, via a filter circuit. Although a prototype is built and tested, the actual circuit was not integrated. The band-pass filter circuit can be seen in Figure 3.6.

### 3.3.4 Implementation

The power inverter design chosen will output a standard modified sine wave at 120Vrms and 60Hz. We initially constructed the circuit on a proto-board and were able to achieve the output we expected from the driver stage. Figure 3.7 shows the output of the driver stage.

We were unable to test the power circuit on a proto-board, so instead we milled the circuit. The milling process was quite extensive. With the help of David Park, we set up an industrial circuit mill to build our circuit. We then soldered all of the components on, and tested the circuit with a test load of a 75 Watt bulb. A picture of the finished circuit can be seen in Figure 3.8.

Our inverter was able to output a maximum 43Vrms at 60Hz. This was tested using a voltmeter. The primary reason that the circuit did not achieve a 120Vrms output is that the power transformer
Figure 3.4: A detailed description of the main component of the circuit, the SG3524, can be seen above. The red boxed region is the input of the feedback system, which receives the AC signal generated by the power stage (Figure 3.4), rectifies it to a DC signal, and steps the voltage down to within the SG3524s specifications. The green region is the input from the battery, which goes through a kill switch and a power regulator to step the voltage to 8v DC. The blue region is designed to adjust the frequency of the pulse width output. The black box outlines the output of the driver stage, which will split the waveforms between the positive and negative portions of the square wave.

Figure 3.5: This circuit receives a square wave from the driver stage of the circuit. It distributes the power across the BJTs, and sends it to transformers via J10, J11, and J12. The output of the transformer is then fed to the load, and also back to the J1 and J2 of the driver circuit, for reference.
Figure 3.6: This circuit is a standard band-pass filter, to smooth the edges of the square wave. The theoretical output of the circuit is expected to resemble a sine wave. Actual testing wasn't completed.

Figure 3.7: This output shows the desired square wave generated between pads J3 and J4 of the driver circuit. The output is 10Vp-p at 60Hz, which is as expected.

not work like we expected it to. The secondary side of the transformer (the side with the center tap) we ordered only allows 3A of current to pass through it, which severely limited the power output. The part was mislabeled as a step up transformer, when in reality it is just a step down
Figure 3.8: We used a circuit miller to outline the traces seen between components. The right side is the power stage, and the left side is the driver stage. The left transformer is used to step the AC power from the output for a feedback.

wired backwards. We strongly believe that replacing the transformer with a proper one would allow enough current to flow to step up the voltage to the full 120Vrms.

In spite of the input not being exactly to specification, we were able to record a power reading through the measurement subsystem from the output of the inverter. Power was received from the battery, and outputted to a 75W bulb. The power output was successfully measured and displayed on the internet interface.

The inverter subsystem for the power converter circuit was very close to being a finished product. We milled the driver and power stages, and powered a resistive load. Future work would include constructing the filter circuit design which is mentioned above, and replacing the step-up transformer with one more suited for our purposes. Beyond that the circuit should work as expected.

### 3.4 Smart Battery Charger

The intelligent power converter requires a consistent DC voltage to be supplied. As such, a battery coupled with a smart charging subsystem is needed to buffer the unpredictable characteristics of
the power sources. The energy stored needs to be maintainable over long periods and have a large charge capacity. Sealed lead-acid batteries are suited for this task and are reasonably priced compared to NiMH and Li-ion of similar charge capacity.

Smart charging is used in medical equipment, laptops, mobile phones, and cameras that allow optimized charging of what are usually expensive batteries that provide long discharge periods [3]. Using smart charging, the device running on the battery can be alerted to the battery being at the end its discharge point. Once alerted, the device can either power down without losing data or provide an opportunity for the user to switch power sources to avoid discontinuous operation of equipment. Smart charging and monitored charging schemes are being applied to many energy storage devices, resulting in little human intervention in the charging process.

3.4.1 Background

Typically, smart battery charging incorporates a battery packaged with integrated circuitry that contains algorithms specific to the battery’s chemistry and charge cycle. The amount of voltage and current supplied by the charger is controlled by the battery which modifies its internal resistance to constrict current flow. The circuitry uses voltage, temperature, and duration of charging to determine the battery's state of charge (SoC) and state of health (SoH), which can be used to optimize charging and help increase a battery’s life expectancy [3].

Life expectancy is decreased by reducing the amount of overcharging and discharging too deep. Over charging causes grid corrosion on the positive plates and gassing that exhausts the electrolyte, while discharging to deep causes the negative plate to form sulfate. In addition to decreasing the battery’s life, these adverse actions can reduce the amount of Amp-hours it can deliver.

There is still much debate as to the definition and classification of intelligent charging systems by battery manufactures and organizations dedicated to the topic [4]. The term smart/intelligent charging is used loosely in consumer electronics marketing that generally describes a charging system that does not require human attention. Many low end electronics and automotive chargers employ multi-stage charging that supply constant current up to 70-80% of the usable charge range and then supply a constant peak voltage to provide a topping off charge without overcharging, thus keeping the battery at optimum performance [5]. When the current supplied has decreased to a substantial value, indicating over-saturation is eminent, a lower float voltage is supplied to counter internal discharge. These low end systems are designed for specific battery types (i.e. SLA, NiMH, Li-ion) which are, in general, lacking charging algorithms which are based on manufacturing processes or nothing at all.

Some batteries integrated circuitry communicate their desired voltage and current to the charger by means of the SMBus. The SMBus is based on the I2C bus which uses a data line and a synchronization/clock line for transmitting data [6]. With this configuration the battery’s supervisory circuitry is the master and the charger circuit is the slave. This system requires a much more
complex battery setup than was available to us, and as such we will not be using it.

3.4.2 Design

Because the wind generator has the capability of charging batteries on its own, it can be treated as an independent part of the smart battery charging subsystem. Therefore, the main focus is to regulate the solar panels to ensure maximum power output to the batteries and inverter subsystem without interfering with the wind generators internal regulation. We found that there are three prominent approaches to intelligently charging a battery.

Fast charging is typically used for smaller rechargeable batteries (i.e. AA, golf carts, and automotive batteries). The side effect of this method is an increase in temperature during charging. This, in turn, increases gassing and can mislead measurements of terminal voltage, causing over-charging and reducing the lifespan. An automotive lead-acid can be fully charged from a deep discharge in roughly six to eight hours using this method.

Trickle charging is a scheme which allows more accurate measurement of terminal voltage enabling a full charge to be attainable without risk of over-charging or over-heating. The drawback to this method is that it can take up to sixteen hours to fully charge a battery [5].

The third approach is three-stage charging. First a constant current is supplied until 70-80% of the total charge is reached, which takes roughly five hours. A constant peak voltage is then supplied to finish charging the battery, which also takes about five hours. The final stage uses a float voltage that is lower than the peak voltage to prevent corrosion from forming on the positive plate. This float voltage will maintain the battery’s shelf life and counter its natural internal discharge [5]. This three-stage charging technique takes advantage of the speed of a fast charge, but only when within safe charge levels. Once the battery is beyond the margin of error created by fast charging, the trickle charge takes over. This was the scheme we chose for our system. Figure 3.9 gives the charging characteristics.

Options for the monitored charging scheme vary from simple transistor circuits to complex Op-Amp, transistor and IC regulators attempting to employ intelligent charging [14]. Attempts at designing the PV charger from scratch proved to be a very daunting task since our circuit engineering skills beyond simple Op-Amps is novice at best. In addition, trying to understand and employ the amateur designs found online ended up being a waste of time. A lot of time was invested reading datasheets from the Linear Technologies and Analog Devices web sites to find a solution. Linear Technologies became our vendor of choice because of their extensive online videos, help documentation, and supplementary applications with parts lists and diagrams which proved to be very insightful for developing designs.

For the PV charger design we settled on using the LT3652 monolithic step down battery charger that provides constant-current/constant-voltage charge characteristics, with a maximum charge
current externally programmable up to 2A [11]. Float voltage and charge duration are also externally programmable allowing for design flexibility. A 1MHz switching regulator regulates the maximum programmed average current by taking voltage measurements about an external current sensing resistor that is compared to internal voltage references. These are then integrated to produce an error current ITH that is proportional the average inductor current. This IC will allow implementing our desired multi-stage charging scheme in addition to the option of designing a trickle charge feature that can be used to initially charge up a battery from a deep discharge, increasing battery life. When the battery is fully charged, power from the solar cells is directed to the inverter subsystem. A continuous portion of the PV power is diverted to the battery when needed. Along with the LT3652, the LT4151 high side current and voltage monitor with I2C interface allows our data logger subsystem to record the batteries state of charge for the user. Minimal external components are required to operate this IC: a current sense resistor, and a pull up resistor for the I2C. The IC also has an additional voltage measurement pin that is less accurate then its primary and could be used to assist the data logger by monitoring the solar panels output voltage. The battery would be set up as the central hub to the wind generator, PV charger and the inverter subsystem. This configuration can be modeled as a parallel network where wind, PV and battery can be assumed as current sources for easy of analysis.

A 1A maximum was desired for the constant-current/constant-voltage charge characteristic up to 14.4V, where the current supplied drops, over time, to a tenth of the design maximum (0.1A).
The battery will then be floated at 13.5V, if the battery drops below 13.2V the 14.4V charge cycle is initiated, if the battery is below 10V it will be supplied a trickle charge of 0.15A until it reaches 13.2V, were the normal charging cycle takes over [11]. The external components of the LT3652 were built on a proto board to test the operation of the IC in trickle mode since the low current should not damage the proto board.

In order to properly test the circuit designs, break-out boards were etched for the two ICs. The LT3652 and LT4151 use the MSOP12 and MSOP10 footprints for printed circuit board (PCB) placement [11][12]. This allows us to test our designs with reasonable flexibility, for if we built a PCB with the components and IC installed, only to find that there was a wiring problem would have set us behind an IC that is non recoverable once installed. Also, the same chip could be used in the final design reducing cost and labor to getting/using a new/spear IC. Both are little more than a tenth of an inch long, their input/output pins are roughly 0.035 by 0.0197 of an inch, making fabrication of the test circuits challenging.

3.4.3 Implementation

Circuit Etching

Using a guide created by our lab manager, Andy Tubesing, we etched our own PCB to house the ICs for testing their operation [13]. To design the circuit layout and paths we used Protel 99 SE. Once designed we could print the circuit on blue etching paper that will be used to etch the circuit image into copper plated circuit board material. The copper was first cleaned free of oxide, dirt and oil using steel wool and then washed using gel soap and dried. The blue circuit transcribing paper was printed onto the circuit board using heat from a clothes iron. The copper plate was placed in diluted ferric acid and agitated until all of the excess copper was dissolved. The pin holes were drilled and then the ICs and header pins were soldered to the PCB.

1st Phase of Testing Lab Test

The power generator was set to 17V because of an advisement from the LT3562 data sheets to have a minimum 2V over head from the peak battery voltage. A current of 1.4-2.8mA was observed in addition to a voltmeter output of 12.21V. When the battery was not connected it maintained a potential of 12.19V. Next, the power supply voltage was increased to just shy of maximum output, where the ammeter measurements increased as well, though randomly fluctuated. The highest current value was 10.4mA using the lab power supplies.

Panel Test
The solar panels (all four) were taken outside and aligned to receive near normal incident sunlight. The panels’ were connected to the charging circuit, and the charging circuit was wired to the battery. We initiated a trickle charge and observed that the battery was being supplied with a constant 0.12A throughout testing. The battery read 12.19V before connection, and then read 12.39V once a connection was made. It then increased in potential for the next 14 minutes until the voltage stopped at 12.60V. The charging curve can be observed in Figure 3.10. Investigating into the battery we found that some of the other design teams that had used it connected it straight to the power supply to charge it up quickly, and some had put low ohm resistances across it to discharge the battery quickly for their testing purposes. Both methods are damaging to the batteries integrity and reliability. This abuse causes a reduction in charge capacity which explains our lower maximum potential. Before additional testing could be conducted on another battery, the negative terminal to the battery came disconnected and fell onto a 0.1 Ohm current sense resistor. The resistor exploded, and brought our charger circuit testing to a prompt end.

### 3.4.4 Conclusion

Our tests showed that the solar panels were much more stable in voltage and current outputs than the wind turbines.

While waiting for a new current sense resistor to arrive, our team designed the now tested charging circuit in Protel. The end goal of this design was to mill the circuit using a professional circuit mill. Figure 3.11 shows the PCB schematic designed. We were sure to make the traces wide as possible for the high current paths since increasing the cross sectional area will decrease resistance and power loss. This circuit will be used to test the multi-stage charging functionality. It will also be used to determine the peak charging voltage, the float voltage, the trip voltage (which starts another charge cycle), and the maximum average inductor current. The data acquired will be used to assist in testing integration with the other subsystems. Fixing the bugs associated with integrating the subsystems will bring the project to completion. Once the lab testing with the fully integrated project is complete the only thing left to do is field test the Intelligent
DC-DC/AC Convert to determine any more errors in the project.

3.5 Data Logger Subsystem

3.5.1 Introduction

Data Loggers are any piece of digital and/or analog hardware systems that keep a log of information about a given system. In our case, we are logging real-time voltage and current information of different parts of our system. We are logging information about voltage and current characteristics being delivered to our load, voltage and current characteristics of the battery and voltage characteristics of our energy harvesters. This information is not stored on the system, but rather piped via Ethernet to the EE server at the school. The information is then displayed and stored on the server side of our system.

3.5.2 Background

Current technology for data loggers for intelligent inverter subsystems has not advanced much in recent years. This is because technology has long since met the requirements for such a system. The real considerations for data loggers come down to the type of hardware that one wishes to use. For these systems, there is a balance of computing power to energy used that needs to be considered. For our case we do not need to crunch that much data and also energy is not abundant or reliable, so we are mainly looking at finding a very low power system that can start and stop operation automatically. Again, these technologies are available. We broke down the hardware options into four categories:

1. Computer with a CPU
2. Microcomputer with a CPU
3. Microcontroller
4. FPGA or CPLD

We found examples of several of these in the industry. Green Champ Co., Ltd produces an intelligent grid-tie inverter that uses a CPU to control the operation of the inverter and display characteristics of its operation[15]. Also, we found an IEEE paper explaining how a microcontroller can be adapted for power systems [16].
3.5.3 Design

Our selection process for the data logger started by analyzing how much data were going to need to handle. We realized that we would be taking inputs from up to four sources simultaneously: the measurement system, the battery bank, the wind generator and the solar panels. This information would then have to send all of this data, as it was collected, over the internet. Assuming a couple inputs from both the battery and the measurement system, we figured that we would need anywhere up to eight simultaneous inputs and one continuous output. We would also need to utilize several data busses including SPI and IIC and possibly several ADCs to receive and send data from the logger. We would need to display the data in real-time on the internet, so we will need to sample the data on the millisecond scale and output at a somewhat slower rate.

The second requirement we analyzed was the ability for system to be able to start and stop automatically from power down/up. Because the implementation of the project would be such that the power supplied would be intermittent, we needed a system that would be able to restart operation after a loss of power without any assistance from a human operator.

The third requirement was cost. We are working with a very limited budget of $400 so we needed find something that would not use a lot of our budget.

The last requirement was power consumption of the data logger. Again, the purpose of the entire system is to supply consistent power to a load for as long as possible so we want a data logger that consumes as little power as possible. We weighted all of these requirements into decision matrix (Table 3.1).

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Computer</th>
<th>Microcomputer</th>
<th>Microcontroller</th>
<th>FPGA/CLPD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Stability</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Power</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Processing</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>9</td>
<td>17</td>
<td>22</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 3.1: Decision Matrix for Data Logger

From this matrix we immediately saw that the computer and microcomputer we not suitable for this project. They were too expensive and consumed too much power. Also, it would be more difficult to have these systems operate independently after a power down. Our choices then were narrowed down to either the Microcontroller or the FPGA/CPLD. They both offered about the same functionality and both consume little amounts of power. However, we did not have an FPGA available that we could easily interface with such as on an evaluation board. We did have an HCS12 available to us on and evaluation board, the Dragon12 board. This board also has ADCs on board as well as modules to handle communication over both the SPI and IIC buses that we
were planning on using. Furthermore, all of the team members had used this before and were very to somewhat capable with it. This made the HCS12 the clear winner for our project.

**Internet Connection**

To implement the data logger, this first thing we needed to do was establish internet connectivity. Because the microcontroller had no way to do this by itself, we needed to purchase an Ethernet module. We considered several, including the SPINET using the ENC28J60 Ethernet controller [17] and the ENC28J60 Ethernet to SPI Header Board [18]. Without much information to decide the functionality of these devices, we went with one of the cheaper models that the team members agreed upon. This was the Wiznet WIZ811MJ [19]. Once the part arrived and we connected it to the microcontroller, we started writing the driver for it. The driver was written in C (see the supplementary material for the code). We began by testing the ability to read and write to the setup registers on the device. Once this was completed we connected the device to a router and tried to establish a connection to a simple server written in Java. After a successful connection we then worked on sending packets of data. This took some time as the Wiznet module had many peculiarities about it. Once we were able to send the sort of data that we wanted, we then established a connection to the EE server and sent sample data to a real-time display on a webpage (Figure 3.12).

At this point, we were able to execute the full client mode operation of the Wiznet 5100 (Figure 3.13). This part of the implementation took some time to make successful as well. We were able to successfully send packets of information at about 100ms intervals without losing any packets. We successfully increased the size of each packet to about 1kb. This was much more than enough to send data collected from each subsystem simultaneously at 1ms intervals and display selected data in real-time. This surpassed all requirements for a live link described in the outline of the project.
Incoming Data

The next stage of testing this subsystem was to generate varying A/D signals on the Dragon12 board, have the HCS12 read that data at intervals of 1ms and see if we could pipe that data as we got it to the server. We were successful in doing so and proved the full functionality of the data logger subsystem.

Powering the System

Once we were able to verify a full-path functionality for the data logger, we then tested the subsystem from power from the battery. I first measured the output of the AC-DC converter the board came with to transform wall power to DC. I measured an output of 11.42V. We were unsure as to how we were going to generate this DC power needed by the Dragon12 Board. We then realized that at the input of the power plug, the board had a simple 5V linear regulator, so any DC source above roughly 8V should work. We connected another Dragon board directly to the battery and verified that this was the case. We then connected our board directly to the battery and again verified the operation of the board (Figure 3.14). The Wiznet was mounted on the Dragon12s breadboard and powered from the 5V bus on the board. This subsystem was now fully functional and could be powered out in the field.

![Figure 3.14: Example of Various Waveforms](image)
Automation

The last step of making sure that this system could begin operation automatically upon startup was fairly simple. Previously we had been storing all of the code into the HCS12s RAM. Once all completed, we placed the code into EEPROM (non-volatile memory) and switch the Dragon12 Board to Load from EEPROM on startup. We then powered down the system, powered it back up and verified that it continued operation as it should.

3.5.4 Summary

To summarize this system, we were able to establish full connectivity to the EE server, send 1kb packets of data at 100ms intervals, simulate real time data by interrupt driven A/D readings, power the system off of the battery bank and have it be able to continue operation after a power loss. If we were to attempt this again, we would not use the Wiznet 5100. Its interface was very unordinary and hard to use. We would definitely use the Dragon12 Board again as it was very easy to use and satisfied all of our requirements.

3.6 Measurement Subsystem

3.6.1 Introduction

The Measurement system could be any hardware that gathers measurements from any particular output. As for our project, this is a subsystem of the intelligent DC-DC/AC converter which monitors and collects measurements from the Inverter. The outputs that are needed to be measured are the frequency, voltage, current, and power factor. Those measurements need to be outputted to a data logger which will then be put out to the internet.

3.6.2 Background

Today many tools are made to measure voltage, current, frequency, and power from systems. These include ICs, circuit components, and voltmeters. A four quadrant multiplier allows us to multiply an analog signal and output a signal to tell us what direction it is going. That is to tell us whether or not the signal is leading or lagging. This was our reasoning for using a four quadrant multiplier in our system. This led us in a curtain direction in our decision for a design.
3.6.3 Design

There are many design examples for the measurement of the values that are needed in a system such as the one we have to create. The system selections started by knowing what kind of measurements were being outputted. An estimated voltage of 120 and current of 4A were outputted from the inverter. Two main ways to going about creating the measurement system is to create it with an IC that has a four quadrant multiplier and building the circuit with implementing the multiplier. Cost was not a big issue for either design since we had a budget of 400 and the estimated cost for either design was 30. After discussion we decided to go with the IC design. The LT2940 is an IC with the four quadrant multiplier with outputted voltage and current proportional to power and current. The LT2940 was donated for our junior design project.

3.6.4 Implementation

The implementation of the design started with the LT2940 IC and creating a Fully Isolated Power and Current Monitor (Figure 3.6.4). It started by getting the chip from Linear Technology and surface mounting that on a board for testing. Circuit etching was the process in which we went about doing that task. Next all the components had to be gathered and implemented into the circuit for testing (Figure 3.6.4). After much trial and error we got an output that was proportional to voltage on a breadboard stimulus.

3.7 Internet Interface

3.7.1 Background

The Internet is becoming more vital to our society everyday. It allows billions of people to share huge amounts of information with relative ease. The Internet also allows users to communicate in real-time from the farthest corners of the globe. The widespread adoption of this service makes
it a prime candidate for a way of publicizing information. The Internet is not just limited to news articles, social networking, and blogs. It can be used to publish status information about a device.

3.7.2 Problem Description

The specifications for this subsystem require that information be uploaded to the electrical engineering web server (geek.nmt.edu) in real-time. The specifications also require that the information is accessible from a publicly available web page. In addition, we have added a few more features that we believed would benefit the end user. These are:

- Highly Graphical User Interface
- Support for Information Archiving
- Email / Text Messaging Alerts

These additional features were selected to provide the end user with more tools for data analysis.

3.7.3 Design

The major design restriction that we encountered was the limited access to the electrical engineering server. Too better explain this issue, we must first discuss some background information. The Internet is a complex network of computers that communicate with each other. The computers can be broken up into two different categories: clients and servers. Servers on the Internet are responsible for providing information to the clients. For example, Google.com is the address of a server that provides information to a person's computer, the client. Typically this information is in the form of a web page but it can also include other types of information like email. Clients and servers communicate using a predefined language called a protocol. Depending on what type of service is needed (web page, email, chat, etc.) determines the appropriate protocol to use. To connect to the electrical engineering server, we needed to know what services were available, and then select the appropriate protocol. After talking to Chris Pauli (network administrator for geek.nmt.edu), we found that the following services were available.

**HTTP** (Hypertext Transfer Protocol) is the most commonly used protocol on the Internet. It allows clients to request web pages and submit form data to a remote server. It has a simple syntax that would make it easy to implement on a device like a microcontroller. [20]

**SSH** (Secure SHell) is used to allow clients to remotely login to a remote server using a terminal. It allows users to both send and receive information in a very secure manner. SSH is a complex protocol due to its encryption and authentication mechanisms. [21]
Both methods were based on the assumption that the smart inverter would act as a client. A third option was considered that would have our device behave as a server. This would require a special hardware module that had a fully functional web server on board. To view the real-time properties of the device, the end user would be required to go to the address of the board in a web browser, instead of geek.nmt.edu. All three options were compared using a weight matrix (Table 3.2).

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Weights</th>
<th>HTTP</th>
<th>SSH</th>
<th>HTTP Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
<td>0.21</td>
<td>0.32</td>
<td>0.36</td>
<td>0.32</td>
</tr>
<tr>
<td>Reliability</td>
<td>0.23</td>
<td>0.36</td>
<td>0.27</td>
<td>0.36</td>
</tr>
<tr>
<td>Cost</td>
<td>0.23</td>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
</tr>
<tr>
<td>Performance</td>
<td>0.15</td>
<td>0.45</td>
<td>0.25</td>
<td>0.30</td>
</tr>
<tr>
<td>Accessibility</td>
<td>0.18</td>
<td>0.43</td>
<td>0.13</td>
<td>0.43</td>
</tr>
<tr>
<td>Score</td>
<td></td>
<td>0.37</td>
<td>0.27</td>
<td>0.35</td>
</tr>
</tbody>
</table>

Table 3.2: Decision Matrix for Internet Interface

The first option we ruled out was making the device the web server. This would only allow the end user to access the device information in the same local area network, defeating the original purpose of the Internet Interface. In addition, the specifications clearly state that information was to be uploaded to geek.nmt.edu. After comparing the two remaining options, we decided to use the HTTP protocol. The deciding factor was simplicity. It would have been very difficult to implement a SSH client on a microcontroller.

Our design was focused at making our microcontroller behave like a web browser. To allow us to send information to the server, we programmed our device to mimic the act of submitting a form on the web. In the event that information was ready to send, the following HTTP command would be transmitted to geek.nmt.edu using TCP/IP:

```
GET /˜matthew.paiz/jd/update.php?voltage=120&current=1
&pf=0&frequency=60
```

This command would send information about voltage, current, power factor, and frequency to a program stored on the server called update.php. The program update.php was programmed using the PHP (PHP hypertext processor) programming language. It would receive information from our device and store it on the server as an archive. The server also contained a separate web page responsible for displaying real-time data. This web page also contained links to the archived information. The archive was organized by date and hour, and would only show the average for every minute of the hour to conserve space. In addition, update.php was responsible for determining if user defined thresholds were surpassed. These thresholds could be defined in a separate web page. In the event that the threshold was surpassed, the update.php script would
email or text the end user with a notification describing the event. This was done using the send mail program on the geek.nmt.edu server.

The biggest emphasis in this subsystem was ease of use. To make data analysis more convenient, the data was presented in a graphical manner (Figure 3.12). To prevent frequent page reloads, we used an embedded Java program called an applet in the web page. Java applets allow us to update a web page graphically without reloading the page. It made it possible for us to display a real time wave form of the data.

Overall we were able to complete every one of our goals for this subsystem. It was successfully tested using data from the data logger. The Email and Texting Notifications were also successfully tested, yielding the following message:

Smart Inverter Voltage Limit Exceeded
Voltage: 203 Volts

3.8 Conclusions

3.8.1 Progress

The objective of the circuit was to invert power from renewable energy sources into usable power for any load. We were able to partially meet this objective. Our power output is usable for any resistive load, and most other loads which are not sensitive to having sinusoidal inputs. We were also able to output power characteristics to the user and charge a battery from the renewable sources. Each subsystem was completed enough to give some form of desirable output, establishing that it is in line with completion. Figure 3.17 shows the estimated percent completion for each subsystem.

One note to be made is that although the figure shows the project as being only 70% done, the three most difficult subsystems are either completed or very close to completion. The fact that we were able to integrate the whole system and achieve a desired output of both the power and the internet interface shows that we have much more than 70% of the project completed.

3.8.2 Future Work

Our progress showed a proof of concept, and that creating a practical intelligent power converter is definitely an obtainable goal. In its current state, the converter demonstrates functionality for
Figure 3.17: Progress Comparison For Each Subsystem

Each subsystem. We were under our required budget, and we project that any purchases which need to be made to complete the subsystems will not push the project over budget.

There are a few changes that we recommend for future work, however. The Wiznet Ethernet module was very finicky, and not practical for our application. We would definitely recommend looking into alternative modules for the final design. As mentioned earlier, the transformers used in the power inverter were not suitable for the amount of power required. Proper transformers are recommended, such as a current sense transformer. The measurement subsystem required a very unique transformer which was not readily available to us, or easy to locate. As such, we would recommend modifying the measurement circuit to require a more basic transformer for future work. The charger circuit only had two of the three charging modes completed, so the third needed to be worked out still. In addition, we would recommend housing even the prototype boards in enclosures to avoid unwanted contact with the high power sources. We were set back significantly when part of our charger circuit blew because of a stray wire from the battery.

3.8.3 Conclusion

The converter performed as we expected, and our level of completion was in accordance with the amount of time given. We successfully proved the feasibility of the circuit, and demonstrated an example output of a functioning prototype.
Bibliography


[10] “Can the lead-acid battery compete in modern times?” [online].


[18] “ENC28J60 Ethernet to SPI Header Board.” [Online].


[26] “Introduction to MOSFETS” [Online].
Appendix A

Code For Data Logger

```c
#include "hcs12.h"
#include "DBug12.h"
#include "vectors12.h"
#include <string.h>

// DEFINITIONS
#define TRUE 1
#define FALSE 0
#define enable() asm(" cli")
#define disable() asm(" sei")
#define S0_TX_MASK 0x07FF
#define S0_TX_BASE 0x4000
#define RESET_WIZNET PORTA = 0x00; delayby1ms(1); PORTA = 0xFF

// FUNCTIONS
void spi_setup(void);
void wiznet_setup(void);
void spi_send_byte(unsigned char byte);
unsigned char spi_rec_byte(void);
void send_wiznet_data(unsigned char upper, unsigned char lower, unsigned char data);
unsigned char rec_wiznet_data(unsigned char upper, unsigned char lower);
void send_wiznet_packet(char *message);
void wiznet_testing(void);
void delayby1ms(int k);
```

// GLOBAL VARIABLES
unsigned int RR_TX_POINTER, WR_TX_POINTER;

void INTERRUPT rti_isr(void);

char *gmessage = "GET /˜matthew.paiz/jd/update.php?data=
    ";

/*********************************************/
MAIN
/*********************************************/
// 1) receive data via interrupts and places into a buffer
// 2) when buffer is full, sends data to EE server
// 3) repeat
main()
{
    ATD0CTL2 = BIT7;       // Enable ATD0
    ATD0CTL3 = 0x00;      // Eight conversions
    ATD0CTL4 = 0x85;      // 8-bit mode
    ATD0CTL5 = 0xA6;      // Sample Ch7 only, Right Justified
    // Unsigned, Continuous Sequence
    RTICTL = 0x47;        // Prescales OSClock to 8ms
    CRGINT = 0x80;        // Enables RTI Interrupt (every 8ms)
    CRGFLG = 0x80;        // Clears RTI Interrupt Flag
    UserRTI = (unsigned short)&rti_isr; // Sets Interrupt Vector

    // HTTP HEADER
    // gmessage[38] = '1';
    gmessage[38] = '2';
    gmessage[39] = '0';
    gmessage[40] = 10;    // LF
    gmessage[41] = 0;     // NULL

    // SETUP
    disable();            // Disable Interrupts
    TSCR1 = TSCR1 & ˜BIT7;  // Disable Timer
    spi_setup();          // Sets up parameters for SPI_0 on HCS12
    DDRA = 0xFF;          // PORTA set to output
    RESET_WIZNET;
    TSCR1 = BIT7;         // Enable Timer
    enable();             // Enable Interrupts

    DB12FNP->printf("end of setup\n\r");
}
wiznet_setup(); // set IP info and establish connection to server
DB12FN->printf("end of wiznet setup \n\r");
// SEND TEST MESSAGE
//send_wiznet_packet(gmessage);
DB12FN->printf("end of send wiznet packet\n\r");

char d = '1';
for(d = '1'; d <= '9'; d++) {
  if (d == '8')
    d = '0';
  delayby1ms(100);
  send_wiznet_packet(gmessage);
}
asm("swi");

void spi_setup(void) {
  MODR |= 0x10; // Reroute SPI0 to PORTM
  DDRM = 0x38; // Configure SPI pins as output/input appropriately
    //0x38
    // 0 0 1 1 | 1 0 0 0
    // | | | | | | || PM0 RXB (don’t care)
    // | | | | | | || PM1 TXB (don’t care)
    // | | | | | | || PM2 MISO_0 (input)
    // | | | | | | || PM3 /SS_0 (output)
    // | | | | | | || PM4 MOSI_0 (output)
    // | | | | | | || PM5 SCK_0 (output)
    // | | | | | | || PM6 RXCAN4 (don’t care)
    // | | | | | | || PM7 TXCAN4 (don’t care)
  SPI0BR = 0x41; /* Set frequency to 4MHz/(4+1)*2^(1+1) = 25MHz/20 = 1.25MHz */
    // BaudRateDivisor = (SPPR+1)*2^3*SPR+1)
// BaudRateDivisor = (5) * 2^2
// = 5 * 4 = 20
// BaudRate = BusClock / BaudRateDivisor = 25MHz / 20 = 1.25MHz
// **NOTE**: Max SPI rate is 14.28MHz for the wiznet w5100

SPI0CR1 = 0x52;  // 0 1 0 1 | 0 0 1 0
   // | | | | | | LSB First Enable (Disabled, so MSB is first)
   // | | | | | Clock Select Output Enable
   // | | | Clock Phase
   // | Clock Polarity
   // Slave Select Output Enable
   // Clock Polarity

void wiznet_setup(void) {
    unsigned char upper, lower;
    unsigned int i, connected, pointer;
    unsigned int dummy;

    // Gateway
    send_wiznet_data(0x00, 0x01, 0xA); // 10
    send_wiznet_data(0x00, 0x02, 0x00); // 0
    send_wiznet_data(0x00, 0x03, 0x00); // 0
    send_wiznet_data(0x00, 0x04, 0xFE); // 254

    // Gateway
    // send_wiznet_data(0x00, 0x01, 192); // 192
    // send_wiznet_data(0x00, 0x02, 168); // 168
    // send_wiznet_data(0x00, 0x03, 0); // 0
    // send_wiznet_data(0x00, 0x04, 1); // 1

    // Subnet Mask
    send_wiznet_data(0x00, 0x05, 0xFF); // 255
    send_wiznet_data(0x00, 0x06, 0xFF); // 255
    send_wiznet_data(0x00, 0x07, 0xFF); // 255
    send_wiznet_data(0x00, 0x08, 0x0); // 0

    // Mac Address
    send_wiznet_data(0x00, 0x09, 0x00); // 00
    send_wiznet_data(0x00, 0x0A, 0x00); // 08
    send_wiznet_data(0x00, 0x0B, 0x00); // DC
    send_wiznet_data(0x00, 0x0C, 0x01); // 01
    send_wiznet_data(0x00, 0x0D, 0x02); // 02
    send_wiznet_data(0x00, 0x0E, 0x03); // 03

    // Source IP Address
    send_wiznet_data(0x00, 0x0F, 0x00); // 10
    send_wiznet_data(0x00, 0x10, 0x00); // 00
    send_wiznet_data(0x00, 0x11, 0x00); // 00
}
send_wiznet_data(0x00, 0x12, 0x4B); // 75

send_wiznet_data(0x00, 0x0F, 192); // 10
send_wiznet_data(0x00, 0x10, 168); // 0
send_wiznet_data(0x00, 0x11, 0); // 0
send_wiznet_data(0x00, 0x12, 75); // 75

// Socket 0 Destination IP-Address
send_wiznet_data(0x04, 0x0C, 10);
send_wiznet_data(0x04, 0x0D, 0);
send_wiznet_data(0x04, 0x0E, 0);
send_wiznet_data(0x04, 0x0F, 40);
129.138.40.11 GEEK
send_wiznet_data(0x04, 0x0C, 204);
send_wiznet_data(0x04, 0x0D, 236);
send_wiznet_data(0x04, 0x0E, 233);
send_wiznet_data(0x04, 0x0F, 39);

// Socket 0 Local Port
send_wiznet_data(0x04, 0x04, 0x04);
send_wiznet_data(0x04, 0x05, 0x22); // 1058

// Socket 0 Destination Port
send_wiznet_data(0x04, 0x10, 0);
send_wiznet_data(0x04, 0x11, 88);

// Socket 0 Mode Register
send_wiznet_data(0x04, 0x01, 0x01); // Set to TCP Mode

// Fill all bufferspace with whitespace
pointer = 0x4000;
for (i = 0; i < 0x2000; i++) {
    send_wiznet_data( (pointer & 0xFF00) >> 8, (pointer & 0x00FF), 0x00);
    pointer++;
}

// Set the WR_TX_POINTER
WR_TX_POINTER = 0x0001;
send_wiznet_data(0x04, 0x24, (WR_TX_POINTER & 0xFF00) >> 8);
send_wiznet_data(0x04, 0x25, (WR_TX_POINTER & 0x00FF));

// Socket 0 – Open
send_wiznet_data(0x04, 0x01, 0x01); // OPEN Command

// Socket 0 – Connect
send_wiznet_data(0x04, 0x01, 0x04); // CONNECT Command
while ( !(rec_wiznet_data(0x04, 0x02) && 0x01) ); // Wait until Connection Established

send_wiznet_data(0x04, 0x01, 0x20); // SEND Command
while(!rec_wiznet_data(0x04, 0x02) && 0x10); // Wait until SEND successful

// Get the RR_TX_POINTER
upper = rec_wiznet_data(0x04, 0x22);
lower = rec_wiznet_data(0x04, 0x23);
RR_TX_POINTER = (upper << 8) + lower;

// Get the WR_TX_POINTER
upper = rec_wiznet_data(0x04, 0x24);
lower = rec_wiznet_data(0x04, 0x25);
WR_TX_POINTER = (upper << 8) + lower;

// Fill bufferspace with whitespace
// try to get RR_TX_POINTER to 0x0000;
dummy = 0x0000 - RR_TX_POINTER;
pointer = 0x4000;
for(i = 0; i < dummy; i++) {
    send_wiznet_data((pointer & 0xFF00) >> 8, (pointer & 0x00FF), 0x00);
    pointer++;
    if (pointer > 0x6000)
        pointer = 0x4000;
}

send_wiznet_data(0x04, 0x01, 0x20); // SEND Command
while(!rec_wiznet_data(0x04, 0x02) && 0x10); // Wait until SEND successful

// Get the RR_TX_POINTER
upper = rec_wiznet_data(0x04, 0x22);
lower = rec_wiznet_data(0x04, 0x23);
RR_TX_POINTER = (upper << 8) + lower;

// Get the WR_TX_POINTER
upper = rec_wiznet_data(0x04, 0x24);
lower = rec_wiznet_data(0x04, 0x25);
WR_TX_POINTER = (upper << 8) + lower;

// send_wiznet_data(0x04, 0x01, 0x10); // CLOSE Command
while(!rec_wiznet_data(0x04, 0x02) && 0x02); // Wait until CLOSE successful

void spi_send_byte(unsigned char byte) {
    unsigned char dummy;
    PTM &= ~BIT3; // send SPI0SS low
    /* Make sure transmit data register empty */
    while((!SPI0SR & SPTEF));
    /* Send data */
    SPI0DR = byte;
unsigned char spi_rec_byte(void) {
    unsigned char dummy;
    PTM &= ~BIT3; // send SPI0SS low
    /* Make sure transmit data register empty */
    while (!(SPI0SR & SPTEF));
    /* Send dummy data to receive some */
    SPI0DR = 0x00;
    /* Wait for byte to be shifted in */
    while (!(SPI0SR & SPIF));
    dummy = SPI0DR;
    PTM |= BIT3; // send SPI0SS high
    return dummy;
}

void send_wiznet_data(unsigned char upper, unsigned char lower, unsigned char data) {
    spi_send_byte(0xF0);   // write op-code
    spi_send_byte(upper);  // upper byte of address
    spi_send_byte(lower);  // lower byte of address
    spi_send_byte(data);   // data
}

unsigned char rec_wiznet_data(unsigned char upper, unsigned char lower) {
    unsigned char dummy;
    spi_send_byte(0x0F);   // read op-code
    spi_send_byte(upper);  // upper byte of address
    spi_send_byte(lower);  // lower byte of address
    dummy = spi_rec_byte(); // data
    return dummy;
}
```c
int len(char *message) {
    int count = 0;
    while (message[++count] != 0) {
        return count;
    }
}

void delayby1ms(int k)
{
    int ix;
    TSCR1 = 0x90; /* enable TCNT and fast timer flag clear */
    TSCR2 = 0x06; /* disable timer interrupt, set prescaler to 64 */
    TIOS |= BIT0; /* enable OC0 */
    TC0 = TCNT + 375;
    for(ix = 0; ix < k; ix++) {
        while(!(TFLG1 & BIT0));
        TC0 += 375;
    }
    TIOS &= (~BIT0); /* disable OC0 */
}

void send_wiznet_packet(char *message) {
    unsigned char upper, lower;
    int i, j, connected, pointer;
    unsigned int physical_rr_addr, physical_wr_addr;
    unsigned int send_size = len(message);
    // Socket 0 - Open
    // send_wiznet_data(0x04, 0x01, 0x01); // OPEN Command
    // Socket 0 - Connect
    // send_wiznet_data(0x04, 0x01, 0x04); // CONNECT Command
    // while( !(rec_wiznet_data(0x04, 0x02) & 0x01) ); // Wait until Connection Established
    // Socket 0 - Send
    // send_wiznet_data(0x04, 0x01, 0x20); // SEND Command
    // while( !(rec_wiznet_data(0x04, 0x02) & 0x10) ); // Wait until SEND successful
    // Get the RR_TX_PTRNER
    upper = rec_wiznet_data(0x04, 0x22);
    lower = rec_wiznet_data(0x04, 0x23);
    RR_TX_PTRNER = (upper << 8) + lower;
    // Get the WR_TX_PTRNER
```
upper = rec_wiznet_data(0x04, 0x24);
lower = rec_wiznet_data(0x04, 0x25);
WR_TX_POINTER = (upper << 8) + lower;

for (j=0; j<1; j++) {
    // Convert the RR_TX_POINTER to its real physical address
    physical_rr_addr = (RR_TX_POINTER & S0_TX_MASK) + S0_TX_BASE;
    physical_rr_addr = physical_rr_addr;
    // Write to Socket 0 buffer, starting at the read pointer
    for (i=0; i<send_size; i++) {
        if (physical_wr_addr > 0x4800)
            physical_wr_addr = 0x4000;
        physical_wr_addr[0xFF00] >> 8, (physical_wr_addr & 0
        physical_wr_addr++;  
    }
    send_wiznet_data((physical_wr_addr & 0xFF00) >> 8, (physical_wr_addr & 0x00FF), message[i]);
    physical_wr_addr++;
    send_wiznet_data((physical_wr_addr & 0xFF00) >> 8, (physical_wr_addr & 0x00FF), 3); // End Of Text
    physical_wr_addr++;
    send_wiznet_data((physical_wr_addr & 0xFF00) >> 8, (physical_wr_addr & 0x00FF), 4); // End of Transmission

    // Update WR_TX_POINTER
    WR_TX_POINTER = RR_TX_POINTER + send_size + 2;
    send_wiznet_data(0x04, 0x24, (WR_TX_POINTER & 0xFF00) >> 8);
    send_wiznet_data(0x04, 0x25, (WR_TX_POINTER & 0x00FF));

    // Socket 0 - Open
    send_wiznet_data(0x04, 0x01, 0x01); // OPEN Command

    // Socket 0 - Connect
    send_wiznet_data(0x04, 0x01, 0x04); // CONNECT Command
    while( !(rec_wiznet_data(0x04, 0x02) & 0x01) ); // Wait until Connection Established

    send_wiznet_data(0x04, 0x01, 0x20); // SEND Command
    while( !(rec_wiznet_data(0x04, 0x02) & 0x10) ); // Wait until SEND successful

    // Get the RR_TX_POINTER
    upper = rec_wiznet_data(0x04, 0x22);
    lower = rec_wiznet_data(0x04, 0x23);
    RR_TX_POINTER = (upper << 8) + lower;

    // Get the WR_TX_POINTER
    upper = rec_wiznet_data(0x04, 0x24);
    lower = rec_wiznet_data(0x04, 0x25);
    WR_TX_POINTER = (upper << 8) + lower;
}
//while( !(rec_wiznet_data(0x04, 0x02) & 0x02) ); // Wait until CLOSE successful
void INTERRUPT rti_isr(void) {
    int a, b, c;
    int value;
    value = ATD0DR0;
    // value = value*196/100;
    a = value/100;
    b = (value-a*100)/10;
    c = (value-a*100-b*10);
    gmessage[38] = a + 0x30;
    gmessage[39] = b + 0x30;
    gmessage[40] = c + 0x30;
    gmessage[41] = 10;
    gmessage[42] = 0;
    // static float DC, k = .0000025, Sd;
    // average = (ATD0DR0 + ATD0DR1 + ATD0DR2 + ATD0DR3 + ATD0DR4 + ATD0DR5 + ATD0DR6 + ATD0DR7) >> 3;
    // Sd = (0.25 + 0.75*average/255)*3050;
    // DC = DC + k*(Sd - RPM);
    // if (DC > 1.0) DC = 1.0;
    // if (DC < 0.2) DC = 0.2;
    // PWMDTY4 = (DC*255);
    // print_volts = 1;
    CRGFLG = BIT7;
}

void wiznet_testing(void) {}
Appendix B

Code For Internet Interface

Graph.java

```java
import java.applet.*;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.*;

public class Graph extends Applet implements Runnable {
    private boolean error = false;
    private String message;
    private final int MAX_Y = 250; // Maximum Y Value
    private final int MAX_X = 100; // Maximum X Value
    private final int GRID_Y = 5; // Number Of Splits In the Y Axis
    private final int AXIS_Y = 20; // Maximum Y-Axis Resolution
    private final int AXIS_X = 50; // Maximum X-Axis Resolution
    private float x_inc;
    private float y_inc;
```
private Thread drawing_thread;
private Thread networking_thread;
private int points[];
int last_value;

// Initialize Java Applet
// Create Buffers to Store Data
public void init() {
    points = new int[MAXX];
    for(int x = 0; x < points.length; x++)
        points[x] = 0;
    x_inc = ((float) getWidth() - AXIS_X) / ((float) MAXX);
    y_inc = ((float) getHeight() - AXIS_Y) / ((float) MAXY);
}

private void add_then_shift(int y) {
    int previous = y;
    int tmp;
    for(int x = 0; x < points.length; x++) {
        tmp = points[x];
        points[x] = previous;
        previous = tmp;
    }

    // Requests Updated Information From The Server
    public int getValue() throws Exception {
        HttpURLConnection connection = (HttpURLConnection) new URL("http://geek.nmt.edu/˜matthew.paiz/jd/log.txt").openConnection();
        connection.setUseCaches(false);
        Scanner scanner = new Scanner(connection.getInputStream());
    }
}
```java
int x = scanner.nextInt();
scanner.close();
return x;
}
// Responsible For Running Both Threads
// One Thread Handles Graphics
// One Thread Handles Networking

public void run() {
  if (Thread.currentThread().getName().equals("Draw")) {
    try {
      last_value = 0;
      for (;;) {
        add_then_shift(last_value);
        repaint();
        try {
          Thread.sleep(100);
        } catch (InterruptedException ec) {
        }
      }
    }
    catch (Exception ec) {
      error = true;
      message = ec.getMessage();
      repaint();
    }
    } else {
      for (;;) {
        try {
          last_value = getValue();
          Thread.sleep(100);
        } catch (Exception ec) {
        }
        }
    }
  } else {
    for (;;) {
      try {
        last_value = getValue();
        }
      catch (Exception ec) {
      }
    }
  }
  }

public void start() {
  drawing_thread = new Thread(this, "Draw");
  networking_thread = new Thread(this, "Reload");
  drawing_thread.start();
  networking_thread.start();
}
// Paint Graphics

public void paint(Graphics g) {
    g.setColor(Color.BLACK);
    float grid_inc = (getHeight() - AXIS_Y) / GRID_Y;
    for(int y = 0; y < GRID_Y + 1; y++) {
        if(y != GRID_Y)
            g.drawString(((GRID_Y - y) * MAX_Y / GRID_Y) + "", AXIS_X - 30, (int)(y * grid_inc) + 10);
        g.drawLine(AXIS_X, (int)(y * grid_inc), getWidth(), (int)(y * grid_inc));
    }
    g.drawLine(AXIS_X, 0, AXIS_X, getHeight() - AXIS_Y);
    g.setColor(Color.BLUE);
    for(int x = 0; x < points.length - 1; x++)
        g.drawLine((int)(AXIS_X + x * x_inc), (int)(getHeight() - AXIS_Y - points[x] * y_inc), (int)(AXIS_X + (x + 1) * x_inc), (int)(getHeight() - AXIS_Y - points[x + 1] * y_inc));
    g.setColor(Color.BLACK);
    g.drawLine(AXIS_X, getHeight() - AXIS_Y, getWidth(), getHeight() - AXIS_Y);
    if(error) {
        g.setColor(Color.RED);
        g.drawString("Error: " + message, AXIS_X + 100, getWidth() / 2);
    }
}
}
update.php

<?php

// Open Log Files For Output
$myFile = "log.txt";
$fh = fopen($myFile, 'w') or die("can’t open file");
$stringData = $_GET["data"];
fwrite($fh, $stringData);
fflush($fh);
fclose($fh);

// Create Archive Folders
$file = date('H');
$folder = date('m. d. y');
$minute = date('s');

$timestamp = date('H:i');

exec("mkdir archive/$folder"); // Create Directory
exec("chmod 777 archive/$folder"); // Set Permissions For Directory

$contents = file_get_contents("archive/$folder/$file");

if (strpos($contents, $timestamp) == FALSE) {
    exec("echo \($timestamp\) Voltage: $stringData Volts >> archive/$folder/$file");
    exec("chmod 777 archive/$folder/$file");
}

$blocked = file_get_contents("block.txt");
$enabled = file_get_contents("enabled.txt");
$limit = file_get_contents("limit.txt");
$email = file_get_contents("email.txt");

// Handle Email If Lock is Not Enabled
if (strstr($blocked, "no")) {
    if ($enabled == "yes") {
        $a = (int) $stringData;
        $b = (int) $limit;

    }
}
echo "$a > $b\n";

if ($a > $b) {
   exec("./text $email $stringData");
}

echo $minute;

?>

</BODY>
</HTML>