Nanogrid

Senior Capstone Project: Winter 2014
Contents

Team Members..............................................................................................................
Abstract.........................................................................................................................
Motivation......................................................................................................................
Objective.........................................................................................................................
System Overview..............................................................................................................
Approach.........................................................................................................................
  Generation.....................................................................................................................
  Circuit Protection..........................................................................................................;
  Bus Design....................................................................................................................
  Sensor Design...............................................................................................................;
  Controls........................................................................................................................
Conclusion.........................................................................................................................
Appendix........................................................................................................................
  Sensor Drivers..............................................................................................................
  Bus Relays....................................................................................................................
  Controller Logic..........................................................................................................;
References.......................................................................................................................
Team Members:

Stephen Jackson (Primary Contact):

*Electrical Engineer*

**Team Lead / Lead Hardware Engineer**

[stjacks@ucsc.edu](mailto:stjacks@ucsc.edu)

[stephen80jackson@gmail.com](mailto:stephen80jackson@gmail.com)

(323)251-7225

I am a 5th year EE major with a focus on power engineering. My upper division coursework at UCSC includes Electric Drives, Power Systems, Power Electronics, Feedback Control, Microprocessor System Design, and PCB Design. I have experience with PSCAD, PowerWorld, MATLAB(Simulink), PSpice, Allegro Layout and Allegro PCB Editor. My programming languages include C/C++, Assembly, and Verilog. I will be responsible for hardware design and PCB layout.

Dustin Bennie:

*Electrical Engineer*

**Vice Lead / Lead Generation and Subsystems Engineer**

[dbennie@ucsc.edu](mailto:dbennie@ucsc.edu)

(707)975-7204

I am a 5th year electrical engineering student with a focus on power and hardware engineering. I have taken many different upper division courses at UCSC including: Power Engineering: (EE175, 176, 177), EDA Tools for PCB Design (EE174), Digital Signal Processing (EE153). I also have experience with PSCAD, PowerWorld, PSpice, MATLAB (Simulink), AutoCAD, Inventor, SolidWorks, Data Structures and Object Oriented Programming in C++, Assembly Language, and Verilog. I also have an extensive knowledge in mechanics. My main responsibility for this project will be Systems Integration and Generation. I will also be assisting in PCB hardware design, and protection.
Seth Cochran:

Electrical Engineer
Treasurer / Lead Sensor Integration Engineer
smcochra@ucsc.edu
(530)591-5246

I am a 4th year electrical engineering student here at UCSC with a focus on power and renewable energy. I have extensive experience with analog filter design. Furthermore, I received wide-ranging training in mechanical design aspects common within engineering projects in the including motor control and fabrication in the Introduction to Mechatronics (CE118) course. These CAD tools include SolidWorks, AutoCAD, Matlab and CorelDRAW. My knowledge of programming ranges from Java and C to Assembly. For this project, I expect to incorporate power monitoring skills gained from the SHEMP (Smart Home Energy Monitoring Project) I was a part of, and I am excited to gain more experience with power monitoring.

Adrian Unkeles:

Electrical/Computer Engineer
Administrator / Lead System Controls Engineer
aunkeles@ucsc.edu
(503)901-1010

I am a 5th year EE major with additional background in CE and Physics. My academic focus has been in the Electronics/Optics track while my employment experiences are in software and systems control development. Relevant upper-division courses at UCSC I have taken: Power Engineering (EE176, taking 176 and 177 in 2014), PCB Design (EE174), Feedback Control Systems (EE154), Device Electronics (EE178), E&M (EE135), and Analog Electronics (EE171). I have extensive experience with PSpice, Matlab (Simulink), Allegro EDA Tools, and various programming languages including: C/C++, Java, Python, Verilog, Assembly, and various scripting formats. My main responsibility for the project will be Systems Control aspect.
Abstract:

The Nanogrid will be an electrical generation and distribution system, built to include multiple loads and distributed energy resources. These components will be monitored by a series of sensors and controllers that can implement load control and system stability algorithms.

Motivation:

We intend to create a Nanogrid as an educational tool for future power engineering students at UCSC; it will give a system level perspective of power flow and power system design. This project will also serve as a test bench for future graduate-level research.

Objective:

This project will demonstrate the basic concepts of power engineering and system design in order to promote education and research. The Nanogrid will be a scaled-down representation of a DC-DC distribution system with an emphasis on modularity and load control techniques. The system will contain two DC brushed motors shaft coupled together as the generation source. The prime mover will be powered with a separate power supply with a PWM driver. The shaft of the second motor, when coupled with the prime mover, spins backwards and acts as a generator. This generation is fed into a bus with various loads and transmission lines attached. Various sensors will relay information back to a microcontroller. The microcontroller will then relay its status back to a master controller relay. The master controller will implement logic based on sensor data, and automotive relays serve as the switches for load shedding. The entire system will consist of several generation points, buses, and loads with a system wide protocol to maintain stability. This system will serve as a basis for implementing a single-phase alternating current (AC) distribution system for the second half of the project.
System Overview:

The goal of this project is to design and implement a small-scale model of the power grid, where multiple parallel generation sources can support a network of loads. A master system will be developed to monitor power flow and communicate with the generation sources in order to maintain stable operating frequency, and meet power flow and load requirements.

This project will be built in modular fashion, to allow for many types of loads to be observed. We will be using several microcontrollers in conjunction with a computer to control and observe the nanogrid. Initially we will incorporate voltage and current sensors, and once 3-phase power is incorporated, we will add sensors to measure VARS and phase.

Figure 1 - System level block diagram separated by boxes to signify who will be leading which components.

The PC will serve as a master control for the system and will interface with the microcontrollers. The PC will utilize software to monitor power flow within the system.
and deliver commands to generation sources for maintaining power flow, load requirements, and frequency requirements. The master system will also deliver the commands that coordinate system recovery under fault conditions and conduct load shedding when generation capacity does not meet load requirements. This approach was designed based on the utility power management system. Control operators will manage the grid and coordinate system recovery much like the PC will in the Nanogrid. The microcontroller are merely a means for collecting and manipulating the data at each bus.

The microcontrollers will relay sensor data to the PC so the master system can perform the necessary analysis and controls. Under fault conditions, the microcontrollers will be able to read the sensor data directly and operate relays to isolate faulted sections of the nanogrid from operation sections. The generation sources will also be controlled by the microcontrollers which will be under command from the master system. Lastly, the microcontrollers will also be able to shed loads under direction of the master system. The loads will be composed of variable resistors, capacitors, and inductors so that various impedances can be simulated on the nanogrid.

**Approach:**

This project will consist of two stages. The first stage requires the implementation of the Nanogrid with direct current (DC) sources. This stage will build the initial test-bed for tying renewable energies to multiple loads of varying impedance to simulate real-world power sinks. A system similar to the Smart Home Energy Monitoring Project (SHEMP) will be developed to monitor power flow on the system by using iterative algorithms, measuring the variability present, and facilitating load shedding. SHEMP was a 2012 CITRIS (the Center for Information Technology Research)
sponsored UCSC Senior Design team that implemented real-time monitoring, and allowed the user to turn off an appliance remotely from an online interface.

Controls will be developed to adjust for demand on the Nanogrid without damage to the loads, or grid. Furthermore, each grid component will also be accompanied by a relay to handle emergency fault conditions. Testing will be implemented to ensure that the system is stable when the proper controls are in place.

The second stage will consist of implementing the same system with alternating current (AC) sources, allowing the possibility to observe transmission effects. Time-permitting, we will update the implementation to utilize 3-phase AC. A system level overview was created in Power World in order to simulate and represent a diagram of a power system seen below in Figure 2.

![PowerWorld one-line diagram showing the desired system with multiple buses interconnected multiple, separate generation sources and loads.](image-url)
Generation:

Overview:

The objective in designing the generators for the Nanogrid is to mimic the way generators operate on the real grid while maintaining an ability to interface with a microprocessor based control system. The original premise for this project was to use interchangeable sources of generation such as a solar panel. However, electric machines were chosen based on their ease of use. Another source of generation could be substituted in place of the current generation, but changes to system parameters and stability would have to be taken into consideration.

When a device is used to convert electrical to mechanical energy, it is called a motor, and when it converts mechanical energy to electrical, it is called a generator. Since any electrical machine can run and convert power in either forward or reverse direction, any machine can be used as either a motor or a generator. Generally, most machines convert energy using the principals of a rotating magnetic field and depend on the basic electro-magnetic principles. Magnetic attraction and repulsion forces are used to generate torque, and rotating motion of the machine (Chapman).

When researching the generation for this project, there were many ideas and iterations that were considered. Brushed direct current (DC) machines were ultimately chosen based on size, cost, and market availability. Most DC machines are similar to alternating current (AC) machines, in that they both contain AC voltages and currents. However, there is a class of DC machines that convert the internal AC voltages to DC voltages at their terminals with mechanisms called commutators and brushes. These are called “brushed” DC machines. This class of motor contains permanent magnets in the stator with one or more electromagnet pairs in the rotor. The stationary part is called
the stator, and the rotating part of the machine is called the rotor. A basic brushed DC machine model can be seen below in Figure 3 (Chapman).

![Brushed DC machine diagram](image)

Figure 3: Brushed DC machine diagram


Magnetic fields are generated in both the rotor and stator, with the use of electromagnets or permanent magnets. As the rotor turns, continual realignment of the magnetic fields occurs. When opposing magnetic polarity fields of the rotor and stator are as close as possible, the motor will be in a stable position at rest. However, when the attractive and repulsive forces are at right angles to the rotation of the motor, the magnetic forces will induce a torque proportional to the strength of the magnetic fields and the current flowing in the electro-magnets, causing the motor to readjust to a more stable position. Figure 4 shows the relative positions of the rotor and stator for minimum and maximum torque generation (Chapman).
When the rotor rotates 180 degrees, the direction of the voltage on the segment reverses, but the magnitude remains constant. The resulting voltage contains a sinusoidal characteristic. The voltage out of the rotor is an alternating positive and negative. In order for a brushed DC machine to produce a DC voltage at the terminals, two semicircular conducting segments are added to the end of the loop, and two contacts (brushes) are fixed at a point where the voltage in the loop is zero. The contacts then short-circuit the two segments. Using this method, the contacts switch connections every time the voltage of the switches direction, and the output of the contacts is built up in the same way to output a constant magnitude of constant polarity (Chapman).

According to Chapman, in order to maintain an alignment that produces the torque throughout a 360 degree rotation of the rotor, the magnetic fields must continuously change. This connection-switching is referred to as commutation. However, this is a very simplified explanation of brushed DC machines, and real, brushed DC machines which contain multiple loops and poles require a much more detailed analysis outside the scope of this project.

Figure 4: Torque generation in an electric machine showing the minimum (a) and maximum (b) torque generated
All generators are driven by a source of mechanical power called a prime mover. Prime movers can be anything from a steam turbine, an engine, or a shaft-coupled electric motor. The output of the generator is directly affected by the speed of the prime mover. Therefore, by varying the speed of the prime mover, the amount of generation can be controlled (Chapman).

**Machine Specifications:**

For this project, a YUYAO JUNDA 24 VDC motor (RS555-2580) was chosen as the prime mover and shaft coupled to a matching 24 VDC generator. This motor was chosen based on the size, availability, and allotted time frame. The size of the motor was important, because the BK Precision 1666 power supplies provided were limited to 40 Volts at 5 Amps with a power rating of 200W. The RS555-2580 motors were rated at 24VDC at 2.83 Amps. Larger DC motors were purchased, but they exceeded the power rating on the supplies. The RS555-2580 motors were also extremely inexpensive and available in bulk. However, the casing of the motors were poorly constructed and not fully shielded for EMI leakage. Because the motors were still considered “hobby” motors the backing of the motors where the leads were located was constructed of plastic instead of metal. A disassembled brushed motor used in this project showing the case, rotor, and brushes can be seen in Figure 5.

![Figure 5: Disassembled brushed DC machine showing the case, rotor, and brushes.](image)

**Mounting Apparatus:**

The machines were mounted to a wooden block using 1 ¼” pipe straps, and the motors and generators were shaft coupled using ¼” vacuum tubing. Rubber was used
in between the machine and the pipe straps to dampen vibrations caused by rotation or coupling. The mounting template can be seen in Figure 6 below, and the shaft coupling can be seen in Figure 7.

![Figure 6: Generation mounting template.](image)

**Figure 6: Generation mounting template.**

![Figure 7: The shaft coupling of the DC machines.](image)

**Figure 7: The shaft coupling of the DC machines.**

*Pulse Width Modulation and Motor Control:*

The generation control was obtained through a modulation technique called pulse width modulation (PWM). PWM is a technique that conforms the width or duration of a square wave, and it can be used to control the power supplied to electrical devices and machines. PWM was chosen for control because PWM provides a smooth variation in speed. The average value of the current and voltage carried to a load is controlled by switching the supply between on and off states at a rapid rate. The greater the duration for which the switch is on in comparison to off periods corresponds to a
higher power supplied to the load. Duty cycle refers to the percentage of time which the PWM pulse is, and a high duty cycle corresponds to high power delivery, since the power is on for most of the period. PWM is a preferred method for power control and delivery because the power loss in the switching device is very low. When a switch is on there is very little voltage drop across the switch, and when a switch is off there is virtually no current. Since power is the product of both voltage and current, the power loss for PWM is nearly zero.

A motor driver was built to take a PWM signal from the microcontroller and feed it into the leads of the prime mover. The purpose of the driver circuit was to use the PWM signal as an electronic control for the prime mover. A Darlington transistor was used for the switching device because of its compound structure consisting of two bipolar transistors connected in such a way so that the current amplified by the first transistor is further amplified by the second transistor. This configuration allows for it to switch a much higher current. The motor driver switching device was a TIP122 NPN epitaxial Darlington transistor. Since the motor would be operating at a max output of 24 VDC and a max current of approximately 2.5 A, the TIP122 was chosen based on the rated collector current of 5 A and an operating voltage of approximately 60 V. A current limiting resistor of 1 kΩ was used, because the microcontroller can only source approximately 18 mA. However, using a transistor as a switch for motor control can generate back emf due to sudden on/off switching form the transistor. Therefore, a silicon-power-Zener flyback (1N 5367B) was connected across the motor terminals for back emf protection. The schematic for the motor driver circuit can be seen in Figure 8.
One major drawback to working with brushed motors is the large amount of electrical noise or electromagnetic interference (EMI) they produce. When a motor spins, it generates a significant amount of current and voltage load transients, observed as high-frequency pulses on the supply lines. This noise can also be generated by the brushes during commutation, and the brushes may bounce when they interact with the rotating rotor. When coupled with the inductance of the motor coils and leads, this bouncing can lead to significant inductive spikes or noise on power lines and can potentially induce residual noise on neighboring lines. The PWM motor driver can also cause noise, but it is of much smaller magnitude than the noise produced by the brushes. This noise can interfere with sensor data, and it can potentially impair the microcontroller by causing voltage dips on regulated power lines. If these voltage dips are large enough, then they can potentially corrupt the data in microcontroller registers or even cause the microcontroller to reset.

The best brushed DC machines are fully shielded in a thick metal case and include metal end caps in order to reduce radio frequency (RF) noise leakage. A machine’s metal casing typically provides enough shielding capability for reducing over-the-air RF interference. Cheaper motors are notorious for leaking and causing RF interference. They usually have thin metal cases and plastic end caps that offer far less
protection from RF noise. However, as discussed earlier the motors used were chosen based on size, cost, availability, and ease of use, so it was anticipated that a significant amount of EMI suppression would need to be implemented.

When applying a 24 VDC source to the prime mover the generator was monitored on an oscilloscope in order to analyze the resulting generation DC signal. The oscilloscope showed that there were significant amounts of EMI that needed to be suppressed. The initial motor system can be seen below in Figure 9, and the unfiltered oscilloscope screen shot can be seen in Figure 10. Figure 10 shows an inductive spike of approximately 330 V.

![Figure 9: The initial generation system.](image)
There are many techniques of filtering DC motor noise. One technique is to keep the machine and power leads as short as possible. Noise can easily be decreased by twisting the generator leads so that they spiral around each other. This method is used in order to cancel the induced magnetic field along the wires for both the prime mover source and the generation leads. By twisting the wire leads, the area between the wires is reduced and the mutual inductance affect is minimized. There were very minimal amounts of noise suppression when performing this technique. The motor system with the twisted leads can be seen below in Figure 11 and the effect of twisting the wire leads can be seen on the oscilloscope in Figure 12. Figure 12 shows an inductive spike of approximately 299 V.
The next step in noise suppression was soldering a single capacitor across the generator terminals. Since a capacitor will only conduct currents that are changing at a high frequency, a single capacitor wired across the generator terminals will function like a short circuit for high-frequency electrical noise, while not affecting the DC current in the generator. This method reduces conduction of noise along the motor wiring, and the capacitors absorb the transient voltage spikes caused by the rotation of the commutator. For this project, there were many iterations of capacitor sizes that were tested. First, according to several electromagnetic interference suppression references, a
0.1 µF ceramic capacitor was used across the terminals, but after noticing that more noise could be suppressed if the capacitor size was increased, several other iterations of larger capacitors were used. Eventually a 2200 uF 50V polarized capacitor was employed, because beyond 2200 uF the noise suppression did not increase. It typically is not recommended to use polarized capacitors for motor suppression, because if the motor is spun backwards then the capacitor could be damaged or fail. Also, sudden inrushes of back emf could also potentially damage the capacitor. However, because the motors only being spun in one direction there was no need to utilize a ceramic capacitor.

Since the generator is spun backwards due to the shaft-coupling of the prime mover, the voltage on the generator is reversed, and it was extremely important to reverse the polarity of the capacitor across the generator leads. The capacitor will catastrophically fail if the voltage is reversed, because a reverse-bias voltage of approximately 1-1.5V will destroy the center layer of the dielectric material. The generator with the single capacitor filter and twisted leads can be seen below in Figure 13. The resulting oscilloscope screen shot from the single capacitor filter can be seen in Figure 14. Figure 14 shows an inductive spike of approximately 92 V.

Figure 13: Single 2200 uF 50V capacitor generation filter.
After the single capacitor filter, there was still residual noise on the generation lines. When a machine is particularly noisy, it is often necessary to use two additional capacitors to reduce the machine noise. One side of each capacitor (0.1 uF) is soldered to one of the generator terminals, and the other side is soldered to the generator’s case. These two additional capacitors make the motor’s case act as a shield and reduce radiated noise. Also, a 2.5A slow blow fuse was added in series with the positive terminal in order to safeguard the motor from excess current. The three capacitor filter and fuse can be seen in Figure 15, and the resulting oscilloscope screen shot in Figure 15. Figure 16 shows an inductive spike of approximately 70 V.
As seen in Figure 16, there was still residual noise on the generation lines, so ferrite beads were used for further EMI suppression. Ferrite beads are often used as passive low-pass filters, and the inherent geometry and electromagnetic properties of coiled wire over the ferrite bead result in an impedance for high-frequency signals. Two different types of ferrite beads were employed. First, there was a ferrite core that was clipped around the positive and negatively twisted generation leads. Second, there were two 6-hole ferrite beads that were implemented in series on both the positive and negative generation leads. Figure 17 shows the ferrite core and 6-hole ferrite bead on the
generator output. The resulting filtered signal can be seen in Figure 18. Figure 18 shows an inductive spike of less than approximately 2 V.

![Image](image_url)

**Figure 17:** The ferrite core and 6-hole ferrite bead on the generator output.

![Image](image_url)

**Figure 18:** The resulting filtered generation with a three capacitor filter and ferrite beads.

**AC Generation**

In a power system, voltage and frequency are the main variables that guarantee stability. Poor frequency regulation can disrupt consumer applications that operate at a constant rate such as a water pump. Transformers are also sensitive to frequency variations. Most importantly, the performance of generators is dependent on the performance of auxiliary electric motor drives that deliver fuel and air to the boiler, oil to bearings and cooling service to many systems. Underperformance of these auxiliaries due to low frequency can lead to a runaway situation with cascaded generator shutdowns and blackouts. Grid operators are also
obligated to maintain voltage within a strict range as most electrical machinery is dependent on a predetermined input voltage to avoid overheating and other possible malfunctions. When multiple generation sources are operating in parallel, it is essential that both voltage and frequency be controlled independently to avoid these consequences.

Voltage and frequency regulation is typically accomplished with specially designed synchronous generators. The key feature of a synchronous generator, as shown in Figure 19, is that the rotor’s magnetic field strength is independently controlled by a DC excitation current. With three coils positioned 120 degrees apart on the stator, three-phase AC power can be achieved. In general, all generators will follow Faraday’s law as follows:

\[ V_{emf} = A \omega B c \cos(\omega t) \]

Here, \( A \) is the area of the coils through which the magnetic field, \( B \), passes through at an angular velocity, \( \omega \). Therefore, with the area of the coils constant, the voltage of a generator is always proportional to the strength of the magnetic field in the rotor and the velocity at which it is rotated. As a result, the ability to independently control the strength of the magnetic field in the stator, as is possible in a synchronous generator, is a critical component to being able to regulate voltage between multiple generation sources.

Figure 19: in a synchronous generator, independently controlled excitation current in the rotor excites a three-phase output voltage http://www.alternative-energy-tutorials.com/images/stories/wind/alt64.gif.
Unfortunately, for the Nanogrid, all commercially available AC motors, induction and synchronous, operate with a constant rotor magnetic field, and there is apparently no commercial application for very small generators which might make the procurement of one affordable. This makes the approach used with the Nanogrid DC generation sources via the shaft coupled DC motors problematic as using a DC motor to drive an AC induction motor will result in frequency and voltage being tied together. This is the reality drove the design of fabricating AC generation for the Nanogrid.

The approach used to accomplish this design goal is one reminiscent of an AM modulator. As shown in Figure 20, using the input sine wave from the induction motor/generator as the “carrier wave,” a DC voltage from a digital-to-analog converter can be used to set output voltage at a desired level via a four-quadrant multiplier chip. This circuit allows the microcontroller to control both frequency and voltage independently.

One major limiting factor driving the overall design of the multiplier circuit was the need to operate off of one power supply since one was driving the motor and the other was powering the bus and its accompanying sensors. Since the design requires op-amps in addition to a negative reference voltage for the DAC, a split voltage supply needed to be created, as show in Figure 21. In order to accomplish this a voltage divider was used in conjunction with a voltage follower to create a virtual ground reference that would be high impedance enough that
any low impedance loading through the power pins of the chips used in the circuit would not offset the half voltage reference created by the voltage divider.

The DAC purchased, Texas Instrument’s DAC0830, featured some extra pins for use with its input buffer register feature. However, tying ILE to VCC as well as tying CS, WR1, WR2, and XFER to GND allowed the convert to operate in “flow-through” mode where the applied digital inputs directly affect the analog output. This configuration minimized the I/O pins needed as well as increased the response time of the voltage control.
Figure 22: Functional diagram for the DAC0830 digital-to-analog converter which features a double latched register.

Since the DAC0830 outputs two differential currents, $I_{OUT1}$ and $I_{OUT2}$, a second transimpedance amplifier stage, as shown in Figure 22, is needed in order to convert the DAC’s output to a voltage. Luckily the chip comes with a matched feedback resistor for use with an inverting op-amp circuit which handles the transimpedance conversion. The final output to the four-quadrant multiplier from the DAC is as follows:

$$V_{out} = -\frac{V_{ref}(digital \ input)}{256}$$
Figure 23: The DAC0830 configured for flow-through operation with transimpedance amplifier to convert the current output to voltage.

Figure 23 shows the functional diagram for Analog Device’s AD633 analog multiplier. It features two unity gain differential op-amps fed into a multiplication junction the product of which is fed into a summing junction with an offset input, Z. The voltage follower at the output ensures low output impedance. The output, W, as a function of the inputs is governed, therefore, by the equation:

\[ W = \frac{(X_1 - X_2)(Y_1 - Y_2)}{10V} + Z \]
In order to achieve the desired modulation of the AC input with the DAC output, inputs X2, Y2, and Z were tied to virtual ground (+12V). The DAC output and AC input from the induction motor (also referenced to virtual ground). Therefore, the output can be calculated from the above equation.

\[
W = \frac{(12 + V_{DAC}) - 12)((12 + v_{in}) - 12)}{10} + 12 = \frac{(V_{DAC})(v_{in})}{10} + 12
\]

With an output range for the DAC from 0V and 12V, the possible ranges for \(V_{DAC}\) are between 0V and 1.2\(v_{in}\) with reference to virtual ground (12V). The final non-inverting op-amp attenuates the signal lower to avoid railing in the audio amp stage.
Finally, since the circuit is meant to drive multiple loads and the output of the circuit up until this point is low-power, a 20-watt amplifier, NTE7143, was used to give the output enough power to drive a load. The circuit, shown in Figure 25, begins with a high pass filter (C4 and R6) to filter out any DC offset gained through the circuit. Resistor R5 ensures that current flows through the high pass filter and not through a lower impedance path. Capacitors C6, C7, C8, and C9 filter out low frequency and high frequency transients and help maintain steady DC power rails. Resistor R7 and capacitor C5 creates another high pass filter that essential makes the circuit a voltage follower for high frequencies so as to eliminate amplification of noise through the audio amp. The final passive components, R9 and C10, maintain the correct bias level between the positive and negative terminals of the output. The result of the circuit was meant to add power to the circuit for driving loads while maintaining stability and filtering out unwanted DC offsets.
Figure 26: Audio amplifier stage of the ac generation circuit adds power to enable the system to drive a load.

The results of this circuit in Figure 26 proved that the amplitude of a sine wave created by driving an induction motor could, in fact, be controlled separately with extra circuitry. The amplitude of the sine wave produced was only bounded by the voltage of the power supply due to railing in the op-amps. The 8-bit DAC provided 0.1V resolution on a 24V peak-to-peak sine wave. Driving the PWM at 100% duty cycle enabled a frequency of approximately 200Hz, which is well above the desired 60Hz target. Such a circuit could potentially be duplicated and used to deliver stable power with multiple generation sources.

One major limitation of this circuit is that it doesn’t deliver the power that was expected. One good reason for this is the way the dual supply was constructed. While the voltage follower provides a good high impedance virtual ground reference voltage, it does not supply
the current needs for a high power application such as the audio amp. This issue could be fixed by running the audio amp off of a single supply circuit instead of treating the +/-12V and virtual ground all as power rails so that a low impedance path back to ground for the current to travel is formed and full power can be delivered to the audio amp.

Even with the circuit modified to deliver enough power to the audio amp, some final remarks should be made in regards to how this circuit contributes to the overall goal of the Nanogrid project, which is to construct a realistic model of a power grid for simulation purposes. Being able to control frequency and voltage independently does facilitate the delivery of stable power, however this circuit also compromises having a more realistic feel and response on the Nanogrid. Typically, on the real power grid, step load changes affect the regulation of frequency as changes in resistance are felt as a counter-torque from the generator to the prime mover, which requires the prime mover to respond in order to keep the frequency within limits. In this circuit, with the generator fed into an op-amp circuit which has a high input impedance, the generator is unlikely to see the impedance changes on the grid and, hence, the prime mover likely won’t feel a shift and torque and have to respond to maintain frequency which is undesired in device which is supposed to simulate controls for load shedding.

**Bus Design:**

The purpose of the bus design is to provide a means by which generation sources and loads can be connected interchangeably in parallel. Another key component of the design, as showing Figure 27, is to make the buses a hub through which the main control system interfaces with the physical Nanogrid via the Uno 32 microprocessors. As shown in Figure 18, the inputs (outputs) to the bus were designed identically. The 5A fast-burn show after the connector serves as a failsafe to any abnormal current spikes. The Hall Effect sensors positioned between the relay and the fuse provides voltage and current readings back to the Uno32 microprocessor. Finally, the Panasonic JS-M relays allow the microprocessor to disconnect or connect a component from the bus. With this outline, when PCBs are made, all functionality can be easily accessible to the microprocessor.
Figure 27: Bus Schematic.

The JS-M relay was selected because it provides robust switching capability of up to 15A and 16V, which is above the operating conditions of the Nanogrid. It also requires 12V and 52mA across the operating coil in order to move the relay contact from normally closed (NC) to normally open (NO). The schematic in Figure 28 demonstrates the basic circuit used to drive the relay via the microprocessor. Since the GPIO pin of the UNO 32 provides only 3.3V and sources 10mA, it cannot switch the relay on its own. A 2N3904 NPN transistor provides the means to amplify the 10mA sources by the microprocessor by a factor of up to 200, well above the minimum required to move the contact from normally closed to normally open when power at that input needs to be switched off.
Once assembled, the overall function of this relay worked as expected. The microcontroller reliably switched the relays as needed and the sensors provided the necessary feedback needed. One anomaly noticed was that the accumulation of small amounts of resistance across the components did lead to a slight voltage drop from source to load. However, this difference was small and the overall control capability performed as needed.

**Sensor Design:**

In the Nanogrid, system control will be implemented directly from sensor feedback. In the DC system, only real power needs to be monitored since reactive and complex power equate to zero. Therefore, the only significant measurements needed are voltage and current. Since sensors were to be placed throughout each bus, it was important to find a way to monitor the power distribution in the least invasive way possible. The magnitude of the magnetic field generated from a current carrying wire may be measured to determine current flowing in a wire. This magnetic sensing allows for noninvasive current monitoring. Unfortunately, voltage must be measured using a known resistance, a consequence of Ohm’s Law, and generates finite power dissipation. Since it was to be replicated at every node in the system, an important design factor was to minimize the power drop (~25 µW) at every sensor in order to reduce the
impact on the power flow in the system. Another aspect of sensing to consider for both sensors, was the maximum analog to digital converter (ADC) input to our Uno32 as well as the input range needed to span for ideal resolution.

**Current Sensing in a DC System**

Several things were taken into consideration when selecting a current sensor. The group selected an operating range of 2 A per generation source that in turn needed sensing capabilities for this size. A current sensor capable of sensing 50 A would result in poor resolution of current flowing in this system; a sensor capable of detecting 5 A flowing at any given point in the system would be sufficient. Another goal was to make the sensor as non-invasive as possible. A shunt resistor was considered; however, quality parts were relatively expensive (approximately $10 per sensor on digikey.com) for our needs. The fact that the sensor would need to be replicated many times throughout the system made the price a valid concern. Furthermore, we needed a sensor that would be useable throughout both phases of the project: AC and DC.

Allegro’s ACS714 Hall Effect sensor cost $4.54, half the cost of the shunt resistors available, matched our desired needs and was used to monitor current in the Nanogrid. Hall effect sensors are transducers that vary their output voltage in response to a magnetic field. A current carrying wire generates a magnetic field as described by Ampere’s Law. This is the magnetic field that the sensor uses; its output voltage varies according to the magnitude of the field (185mV/A). With the use of magnetic sensing, our current sensor at each node would not interfere with the overall power flow of the system.

The Hall Effect sensor comes in a small 8 pin SOIC package that presents a problem with prototyping. This issue was resolved by a recommendation from a classmate to purchase SMT (surface mount technology) to DIP (dual in-line package) adapters from Proto Advantage. These adapters allowed for the use of the surface mount chips to be used in our breadboard and proto-board models, and later when we transition to PCB design.
Figure 29 below shows the typical application recommended by Allegro that we utilized in the system. As shown by pins 1 – 4, the ACS714 is capable of measuring between -5 A and 5 A. This is another advantage of the chip because we are able to sense current direction at a given node. This will be helpful in a transmission line connection to multiple buses. In addition to the measuring range, the chip is undefined yet undamaged at currents up to 20 A in either direction. This means, in the case of a short circuit, large currents will not damage our device. Filter capacitors are used as shown in Figure 21 from pins 6 and 8. Pin 7, labeled \textit{VIOUT}, varies 185 mV per amp. At 0 A, the output is biased at 2.5 V. This presented a problem for our ADC input as shown in Figure 30.

![Figure 29: Diagram from Allegro's ACS714 Hall Effect Sensor datasheet showing the pin configuration of the chip along with a typical application.](image-url)
First, the resolution obtained when sensing between 0 A and 3.5 A was about 200 points on the ADC reading. Therefore, for every 100 mA, the sensor changes the ADC reading by 6 points. With expected noise on the channel due to close proximity of motor generation, this range on the output was unacceptable. This problem yielded a standard op-amp problem of biasing the voltage and amplifying the signal to set an output between 0 V and 3.3 V (and therefore a 0 – 1023 ADC output which maximizes our resolution) for sensing of current between 0 A and 5 A.

For our output, we need to select a gain and a bias voltage to use. Our gain is equal to our current voltage difference to the difference we wish as seen below.

\[ G = \frac{V_{\Delta \text{New}}}{V_{\Delta \text{Old}}} = \frac{0 - 3.3 V}{3.425 - 2.5 V} = -3.567 \frac{V}{V} \]

This yields a gain of -3.567 using an inverting amplifier. From this value, we can choose \( R_1 \) and \( R_2 \) appropriately. Through simple circuit analysis, we yielded the following equations that allow us to calculate the bias voltage we need to obtain our new signal range.
\[ V_{\text{out}} = -G V_{\text{in}} + (1 + G) V_{\text{bias}} \]

Rearranging to solve for \( V_{\text{bias}} \) we have:

\[ V_{\text{bias}} = \frac{V_{\text{out}} + G V_{\text{in}}}{(1 + G)} \]

Now, selecting values of 0 V and 3.3 V for \( V_{\text{out}} \) and \( V_{\text{in}} \), respectively, yields a \( V_{\text{bias}} \) of 2.58 V. The circuit used is shown below in Figure 31 and shows the selected resistor values. The voltage follower distributes the bias voltage to each amplifier throughout the bus. The new analog to digital conversion distribution is shown in Figure 32.

Figure 31 - Shows a single op amp used to amplify our Hall Effect output voltage. In the bus, the voltage follower appears just once, and the op amp shown here as U34B is replicated for each sensor.
Figure 32 - Plot showing the associated ADC value generated from the voltage outputted by the ACS714 at varying currents after it has passed through our op amp circuit.

The amplified signal is also accompanied by amplified noise. Along with a software averaging system, the least significant bit has been removed to generate a more stable signal. Our final readings were produced on a 0 to 511 range. Using only the top 9 significant bits alongside a moving average calculation in software yielded a steady current signal that was accurate to about 1%. Rearranging the equation listed in Figure 32, representing the current to ADC relationship, we were able to convert values our ADC read into actual current to use in our control algorithm as shown below.

\[ x = \frac{y - 525.45}{-94.38} \]

Voltage Sensing in a DC System

A simple voltage divider wired in parallel with every load shown in Figure 33 below generated accurate voltage readings. The overall resistance was chosen to be much greater than the typical resistance we expected of our loads. This induced a minimal current to flow through the voltage divider, and therefore, our sensor created a negligible voltage drop in the overall system.
The motors are capable of delivering about 12 V to the system when powered by a 100% PWM. Therefore, the voltage divider needed to be mapped to a 0 V to 3.3 V scale. Using 1% rated resistors, the values of $R_1$ and $R_2$ were selected to be 2.2 MΩ and 820 kΩ, respectively. The 1% resistor tolerance is preferred, because when calibrating the sensors it is important to have accurate values to assure that the sensors all have the similar values. 5% resistors can be used but will be less accurate and will require further calibration. The equation below shows the linear relationship between the input and output voltage related to the mapping of the two resistor values.

$$V_{out} = V_{in} \frac{R_2}{R_1 + R_2}$$

With an input of 12 V, we expect to have an output of 3.258 volts. Similarly, with an input voltage of zero volts, we yield zero volts at the output. This matches the ideal ADC input span as intended and is displayed in Figure 34.

A calibration of the sensors was completed to find the voltage dividers were accurate to 1%. The calibration also yielded an equation that helped map our 0 – 1023 ADC value to an actual voltage on the software side of the system.
Figure 34 – Shows a plot of the relationship between our ADC values and the voltage divider output used to measure voltage at each node in our bus.

As expected, Figure 34 shows the linear relationship we were expecting. The $R^2$ value displayed shows a perfect fit for our line. Rearranging the equation listed, we were able to convert the values the ADC read into actual voltage to use in our control algorithm as shown:

$$x = \frac{y + 2.2929}{84.566}$$

It was important to maintain the modular ability of the overall system with the adoption of the AC system. This would allow the transition from a DC system to an AC system and back without having completely different boards. Voltage and current signals that would typically be fed to ADC ports when operating the DC system now had to be modified to provide appropriate feedback to the microcontrollers. In an AC system Root Mean Squared (RMS) values for voltage and current at each node on the grid were necessary as well as sensors to allow for power factor calculations.
**Low Voltage Rectification**

In the DC system, a high-impedance voltage divider was used to map a 24V value to a 0-3.3V microcontroller ADC port while minimizing the power dissipation at each voltage sensor. In the AC system, 0-3.3V signal needed rectification while maintaining reasonable resolution.

**Current**

The current sensor outputs data centered on 2.5V. Therefore, to rectify this signal, a biased sine wave would have to be manipulated with operational amplifiers. Again, diodes were not considered because of their non-ideal forward voltages. Figure 35 shows the configuration below.

![Figure 35: Circuit used to rectify the Hall Effect sensor output.](image)

From Figure 35, the sensor outputs to Current1 and is run through a unity gain amplifier to prevent back feed to the other sensors. Then, the signal is run through an inverting op amp with a bias voltage of 2.5V. The transfer function is shown below as:

\[
V_{out} = \left( \frac{R_{110}}{R_{111} + R_{110}} \right) \left( \frac{R_{109} + R_{108}}{R_{108}} \right) V_{bias} - \left( \frac{R_{111}}{R_{108}} \right) V_{in}
\]

The resistors were designed to accommodate the following function: 2.5 + sin(ωt). With \(R_{108} = R_{109} = R_{110} = R_{111}\), the output is reduced to \(V_{out} = V_{bias} - V_{in}\). Selecting \(V_{bias} = 2.5V\) will
remove the DC bias in the sine wave, and feeding it into an inverting op amp will remove the positive part of the sine wave due to the single supply rails. The result is a half-wave rectifier without any diodes. This technique may be smoothed over with a filtering capacitor yielding a ripple voltage as shown below:

\[ V_{\text{ripple}} = \frac{I}{fC} \]

**Voltage**

A full wave bridge rectifier was considered. This is the standard way to rectify a large signal. Full wave rectifiers require the signal to pass through two diodes, and this implies a voltage drop across each diode. With a typical voltage drop of 0.6V across each diode, our 3.3V signal automatically gets truncated to 3.3V – 2 (0.6V) = 2.1V. Furthermore, voltages that map to below 1.2V were considered a zero value. This was not acceptable, and other approaches were considered. Additionally, schottky diodes were considered. With these, resolution will be lost as well but to a lesser degree. Typically, the voltage drops with these diodes that were found fell in the range of 0.3V – 0.5V. This was still unacceptable, and a better approach to the problem was sought.

After quite a bit of research, a circuit was found via an online source, "Precision Full-wave Signal Rectifier Needs No Diodes" (Blanes). The schematic below utilizes two operational

![Figure 35: Typical full wave diode bridge rectifier](image-url)
amplifiers and is implemented without any diodes. It results in a precision full wave rectification of the signal. Figure 36 shows the circuit used for the full wave rectification of the voltage signal.

Figure 36: Shows the circuit used for full wave rectification of the voltage signal.
The circuit operates as follows: If $V_{in} > 0V$, then U34A’s output equals $V_{in}/2$, and U34B operates as a subtractor, delivering an output voltage equal to the input, Voltage1. In effect, the circuit operates as a unity-gain follower. If $V_{in} < 0V$, then the output of U34A equals 0V, and the circuit behaves as a unity-gain inverter and delivers an output of $V_{out} = -V_{in}$. Therefore, the output is a fully rectified version of the input. Selecting a large capacitor allowed the generation of a small ripple voltage. The theoretical ripple voltage may be calculated as shown below.

$$V_{ripple} = \frac{I}{2fC}$$

Figure 37 shows the output of the voltage rectifier circuit without an output capacitor, and Figure 38 shows the output of the voltage rectifier circuit with an output capacitor.

Figure 37: The output of the voltage rectifier circuit without an output capacitor.
Zero Cross Detection

As part of the AC system, it is important to sense the system wide frequency as well as if the system requires any power factor correction. In addition, with multiple AC generation points, frequency synchronization is important. To calculate phase angles and power factor, the time between zero crossings was used. The zero crossing circuitry and code is described below.

While this component was not fully implemented, the modular code allows for the implementation of zero crossings, phase angle calculations and power factor in future improvements to the project. With this information, a digital switch may be used to incorporate an inductor or capacitor bank as a means of power factor correction.

Power factor is calculated as follows

\[ PF = \cos \phi \]

Where \( \phi \) is equal to the phase angle. Power factor is a very important aspect of power systems. As a consumer, household items draw resistive and inductive power. An inductive loads, like a motors, require reactive power to generate an electric field in order to operate. Power systems with current and voltage out of phase with each other increase the power necessary to generate and transmit electricity. As a result, power companies often switch on
parallel capacitor banks to improve system efficiencies. Power factor is measured between values of -1 and 1, where -1 means the current and voltage are 180° out of phase and 1 means the waveforms are 100% in phase.

**Voltage**

The LM339N comparator was used along with one of the typical application circuits on the datasheet. The application is shown below in Figure 39.

![Figure 39: the zero cross detector circuit used for the voltage daughter board sensors.](image)

The diode (D1) prevents negative voltages (anything more negative than a forward voltage drop i.e. -0.6V) from entering the chip. The circuit can be operated without the feedback portion; however, these components improve switching time. In addition, the feedback implements a small hysteresis that acts as a debouncer (~5mV). The 10k resistor acts as a pull-up to VCC. Here, VCC is shown to be 15V; on the daughter board, a supply voltage of 3.3V was used. This allowed the pull-up resistor to output a perfect logic high or low to our microcontroller for processing. Figure 40 shows the resultant input and output waveforms.
Figure 40: The input (yellow) and output (blue) voltage from the circuit shown in Figure 39. Note the output is high when the waveform is low.

Current

The zero crossing detection for the current sensor was easier to develop. It was biased up at 2.5V as explained before, so the LM339N just compared the input to 2.5V. When the voltage was below this threshold, the output would be high and vice versa. Feedback was implemented to enhance switching time and implement hysteresis as a de-bouncer (~5mV). The circuit is shown below in Figure 41, and Figure 42 shows the resultant input and output waveforms.
Figure 41: The circuit used for zero cross detection of the current waveform.

Figure 42: The input (blue) and output (yellow) waveform from the Hall Effect sensor circuit biased at 2.5V shown in Figure 41.
Space on the daughter board likely could have been saved by tying every LM339N to a 5V supply and running a 3.3V rail to each pull up resistor. This would limit the need to use different chips for the 3.3V rails and the 5V rails. You cannot use the 3.3V rails because the Hall Effect waveform biased at 2.5V can exceed 3.3V, and the comparator is then operating out of range.

**Code**

In order to update the phase delay times for each load, the ZeroCrossCheckEvents() function must be called. This function reads the pins and checks to see if they have changed from the last state they were in. If the state has changed, the value for the load will be updated. The checking works best if you run it in an infinite loop within main so the program can check for changes as often as possible. When a change occurs, a flag is set to alert the program to access the running time. From there, the CalculatePhaseAngle() and CalculatePowerFactor() functions can be used to manipulate time into values of interest. The code for these functions along with an example implementation in a main function can be found in the project, Nanogrid_Rev6-ZeroCrossDetect.X.

With the square waves from the zero crossing circuitry, it is important to remember the inverted signal from the voltage circuit. This is corrected in software. However, when testing the circuit with the oscilloscope, you need to measure time delays from the falling edge of the voltage square wave to the rising edge of the current waveform.

For the zero cross detection, the system needed to know how much time had passed between each crossing. In order to do this, an ISR (interrupt service routine) was used to increment a count at a rate of 10 kHz. This gave the zero cross detection 0.1 millisecond resolution. Calculating how many points cycle yields:

$$\frac{10 \text{ kHz}}{60 \text{ Hz}} = 166.67 \text{ points per cycle}$$

At 166.67 points per cycle, the system has 2.16 degrees of resolution. From this one can expect errors to be as large as 0.6%. This was satisfactory as the system regularly returned exact
values matching the oscilloscope traces. The code that was implemented returned poor results when the system ran print statements between the first and second crossing. In this case, the print statements delayed the second zero crossing time reading. It resulted in botched values. Without the print statements, the system operated as desired within the theoretical 0.6% tolerance.

Potential Problems Loading the Communication system with an ISR

The control algorithm requires constant messaging to occur between a master microcontroller and a network of slaves. Running an ISR has the potential to create disturbances in the message protocol if the ISR triggers in the middle of a message. Because the AC system was not completely finished, the zero crossing code was not exposed to thorough testing with a control algorithm in place. However, in a brief test; an ISR was run (at 10 kHz and only implementing a time variable as before) while messages were passed back and forth between a single master and slave. In this interaction, messages did not appear to drop. With the incorporation of an entire algorithm, the system might be expected to either drop messages or measure longer times between crossings while the microcontroller services tasks before servicing the zero cross code. Increasing the number of slaves should have no effect because the zero cross algorithm runs local to each slave.

Load Control:

In a DC system, our single bus control scheme was focused on the fact that voltage will be uniform at any load. Therefore, if 5 volts is delivered to one load, all the other loads in the bus (all of which are connected in parallel) will receive 5 volts. This was verified experimentally to insure that loses in the system wouldn’t effect this ideal assumption. The loss was found to be negligible (less than 0.01%).

The control system runs two systems simultaneously. First, it protects against voltage irregularities. In the real world, under-voltage conditions represent a state in which power generation capabilities do not fulfil the power demanded by the load. In these conditions, it is
important to maintain a constant voltage (in the United States, this is 120 VAC +/- 5%) at every node possible rather than provide a lower voltage everywhere in times of high demand. In this case, the system needs to cut distribution (induce a blackout) to some nodes in order to maintain voltage. Our smart system ranks each load based on its priority in the system and sheds the lowest priority loads one at a time to maintain power at our highest priority location. Second, the system protects against over current faults with an auto recloser algorithm. The Nanogrid control system mimics a traditional technique used on the grid in that it attempts three quick “re-close’s”; if the system is still at fault, then it waits for a long period of time and tries to reconnect the load a final time before permanently disconnecting the load.

**Control Flow for Voltage Regulation in a Single Bus DC System**

The system involved in maintaining the proper voltage is delivered to our primary load (and the rest of our system) is shown as a flowchart in Figure 43. The system relies on feedback control to increase or decrease the PWM values of our motors to meet our system need. It begins by checking the state of our generation. Our settings initiate the PWM to a low value (approximately 30% of its total generation capacity). Since the generation is not at maximum, the system checks the state of our highest priority load. In the case where the load is under or over-voltage, it looks to the proportional-integral-derivative controller (PID controller) to correct our overall PWM signal. This consistently loops until the voltage has corrected itself. Once corrected, the system allows for a 1% deviation in supply voltage to the primary load before attempting to adjust the PWM value again (i.e. for a 5 V system, the primary load may vary between 4.95 V and 5.05 V before receiving a correction). This alleviates constant motor oscillations attempting to remedy non-significant deviations. If the PID controller indicates that a 100% PWM value or greater is necessary in order to supply the primary load with power, a load needs to be shed. This is completed by tripping the relay of the lowest priority load. The algorithm keeps track of what loads are ‘on’ and ‘off’ in order to know what loads are available. If the PWM is still at maximum, the lowest priority load remaining is shed and so forth. If the system has shed loads and the primary voltage is stable, it looks to recover lost loads. This is only done if the generation source has the capacity to support another load. In order to decide
whether or not we could regain a load, we set a threshold in software of 80% that the current PWM value must be less than.

![Flow chart](image)

**Figure 43**: Flow chart depicting the control flow of our single bus system to maintain a constant voltage across our primary load.

**Control Flow for Short Circuit Protection in a Single Bus DC System**

Along with voltage protection, the system also protects against short circuit faults and autonomously attempts to remedy the fault. Figure 44 shown below is a flowchart demonstrating the logic used to protect against short circuit faults. The system continually polls each current sensor to determine if its load has drawn too much current and has exceeded the preset threshold. If it has, the associated relay is tripped and a timer is set for a short period of time. Each load has three chances to recover. When the timer expires, the system attempts to
recover the load by reopening the connection through its relay. If the fault has not cleared (sensors still detect over-current conditions), we increment our count and repeat the process. After the second failed quick-attempt to recover the load, a long timer is set before we recheck the load a final time. If the load fails to resolve the short-circuit issue, the system permanently cuts the load from the system. Otherwise, if at any time in the recovery process the load has resolved the short circuit, the node is restored and its count is reset to zero consecutive faults.

Figure 44: Flow chart depicting the control flow of our single bus system that protects against short circuit fault conditions and attempts to remedy them.

**Voltage P.I.D Control**

On the Nanogrid, the primary concern for the motor driving the DC generator is the regulation of voltage. In a system where multiple generation points are connected in parallel, each source must be at the same potential. Without constant voltage, the system is no longer
operating in stable DC. This can be a challenge as loads change and the generator must
instantaneously respond with more or less power and little transient disturbance. The objective
of maintaining a constant voltage was achieved with a Proportional-Integral-Derivative
Controller shown in system block diagram form in Figure 45.

![PID System Block Diagram](image)

**Figure 45: System Block Diagram with PID Controller.**

As shown above, the system plant is composed of a process of the DC generator which
provides the voltage output: y(t). This variable is a function of the speed at which the generator
armature is rotated. The process is controlled by a DC motor actuator which outputs a torque to
the generator based on the input of the duty cycle set in the motor driver by the microprocessor.
The system also provides a feedback path through a voltage divider outputted through the
ADC converting the actual voltage output to a digital value.

The dominant plant disturbance caused on the Nanogrid comes from the shedding and
regaining of loads. An increase in load causes a counter torque applied from generator to the
actuator to increase. The actuator decelerates if the power being delivered by it stays constant.
In addition, the voltage will increase or decrease depending on a decrease or increase in load
respectively. Since loads are gained or shed instantaneously, it can be seen virtually as a step response. In the case of the experiment conducted in Figure 46, the voltage can vary by as much as 40% with the addition or subtraction of a load in a single bus system. To stabilize the voltage as instantaneously as possible, the actuator is controlled through a software-based PID controller.

![Voltage Response to Load Change](image)

**Figure 46: Voltage Response to Stepped Load Changes.**

Figure 47 shows the basic aspects of how a system responds to a step input. In essence, the goal of the PID controller is to maximize slew rate, minimize overshoot, and minimize settling time to reach within a certain tolerance of the desired final value. The output of the PID controller is set by the error between the desired output and the actual output, and is governed by the differential equation:

\[
y(t) = k_p e(t) + k_i \int_0^t e(\tau)d\tau + k_d \frac{de(t)}{dt}
\]
Figure 47: Aspects of a response to a step input.

(http://microchip.wdfiles.com/local--files/asp0107:settling-time-overshoot/SettlingOvershoot.png)

Reaction to the stepped change in load can be tuned by adjusting the constant in each term of the equation. The proportional constant, \( k_p \), decides the slew rate, the overall sensitivity to a change in the error signal. The derivative portion of the PID controller seeks to reduce overshoot by reducing its output depending on how quickly the target value is being approached or departed from. Finally, the integral term seeks to reduce settling time eliminating small but persistent errors between the actual and desired values. By selecting the constants associated with these terms carefully, the system can quickly and smoothly find its desired voltage (PID Lab).

In order to execute the PID controller in software, some approximations need to be made on account of not having an exact means of arriving at an integral or derivative since they are continuous functions and software can only handle discrete functions. From calculus it is known that with a \( \Delta t \) approaching zero these functions can be reduced to:

\[
\frac{de(t)}{dt} = \frac{\Delta e}{\Delta t}
\]

and,

\[
\int_{0}^{t} e(\tau)d\tau = \sum_{i=1}^{k} e(t_i)\Delta t
\]
The PID function is executed every 200 ms, a relatively short $\Delta t$. Therefore, the derivative of the error with respect to time can be found by taking the difference between the current error and the previous error and dividing by the time duration (200ms). Likewise, the integral error can be found by adding the current error to the previous error and multiplying by 200ms. Finally, in order to prevent constant oscillation, the error value is considered to be zero if the voltage value is within a one percent tolerance range. The result of the PID function is an integer correction value which is outputted to adjust the PWM duty cycle which directly controls the power delivered to the actuator from the power supply.

The resulting performance of the PID controller was considered acceptable as the system was able to adjust to a stepped load change and set a stable voltage within a second. Further fine tuning of the function can yield an even tighter response.

**Circuit Protection:**

**Auto Recloser Protection:**

A typical component in a real grid is an auto recloser, which seeks to handle faults without the need for a worker to reset the system. Most faults are temporary in nature and usually caused by a fallen tree branch which creates a short between the power line and ground, but quickly clears. With all generation sources and loads connected in parallel, the sudden low impedance caused by a ground fault will pull the voltage of the generator down to zero and cause a spike in current when the power being delivered is still constant. An auto recloser detects this spike in current, and acts to immediately shut off that portion of the grid. However, instead of remaining in the off state, the auto recloser closes the circuit a few times in quick succession to test whether the line is operating normally. If the fault does not clear and overcurrent conditions persist then, after a longer pause, the line is checked one last time before remaining permanently open until a worker fixes the situation.

The auto recloser function was implemented on the nanogrid via the Uno32 microcontroller by reorganizing the system into a state machine as shown in Figure 48.
system constantly polls for overcurrent detection so as to give quick resolution to the fault. Upon detection of an overcurrent, the system moves from the normal operating state to an emergency state. In the emergency state, the PID controller is disabled freezing the generator output from spiking in response to what the sensors will measure as well below the target voltage (theoretically zero). Likewise, load shedding and load regaining algorithms are also not used in the emergency state since it is assumed that the voltage requirements for the priority loads can’t be met when the system is short circuited. While in the emergency state, a timer is initiated for a duration depending on whether it is a quick check or a longer check. After this timer expires, the system moves back into the normal state and can resume normal operation so long as the fault is cleared. If the fault is not cleared, then overcurrent will be immediately detected, and the emergency state will be entered again. The transition from the normal to emergency state can only happen four times before a boolean value indicating a permanent fault becomes true and the system resumes normal operation, minus the faulted bus input.

![Nanogrid State Machine](image)

**Figure 48: Nanogrid State Machine**

The auto recloser algorithm was tested by inserting a physical switch on a particular load with a wire connected to ground to simulate a fault. Tests demonstrated that this algorithm worked as expected on the single bus system. The ground fault was quickly detected and the shorted load relay was tripped. The auto recloser algorithm successfully detected whether fault
was cleared or not and was able to resume normal operation, either without the load if the fault was deemed permanent or with the load if the fault was deemed temporary.

System Communication:

The Uno32 development kit (with an embedded Pic32 microprocessor) was the platform of choice for individual bus monitoring. Time-averaged voltage and current readings were calculated for each component on a bus to define its present state. The retention of steady-state voltages amongst all the loads was the primary objective; similar to a real power grid, thresholds were defined. Within these limits, the components operated as expected by the user and power suppliers.

The consolidation of similar packets of data from each point in the grid gives the total present state of the system from which to detect emergency conditions. At this point an algorithm was developed to determine emergency conditions within a bus; the next step was to enhance the emergency condition response to include the action of a transmission line between two or multiple buses. The platform of choice for this algorithm, the storage and analysis of this information, and formation of decisions to communicate back to the dispersed points about desired changes in their generation and load connectivity, is the Raspberry Pi.

Master Platform

The Raspberry Pi (RPi) was chosen for a system-wide monitoring device because of its broad capabilities, especially as the project moves forward with the AC model. Transient analysis requires the solving of differential equations and handling of complex matrices, tasks which the RPi will handle with Python libraries (SciPy, NumPy) and make user-readable solutions with Matlab plotting software. A PC could equally provide the software capabilities for this system with greater processing power, yet the on-board GPIO of the RPi bring the software one step closer to the hardware, and the absence of a hardware interface diminish the modularity of the system. The RPi has been loaded with its preferred OS (Raspbian-Wheezy) providing a streamlined and intuitive platform for C code and Python scripting. Similarly to the
process undergone with the RPi, the Uno32 was also chosen for its high level of suitability with the projects specifications.

**Slave Platform**

The Uno32 was chosen to provide sufficient implementation of ADC and PWM handling. Basic on-board port toggling allowed the implementation of load shedding techniques intrinsic to system control. Furthermore, the wealth of code already available through the CE118 roach projects enables much of the necessary peripheral functionality. Jumper placing was necessarily checked to both configure the hardware for slave operation, and enable the slave select line on the board. The CE118 repository even provided a bootloader GUI with which to flash the devices with the projects, providing a quick method to check experimental results against the expected outputs for the board. A feature common to the Uno32 and RPi of particular use for our project, were their support of the SPI communication protocol.

**SPI background**

As a serial data link that requires two configuration types of clients (master and slave), the RPi and Uno32 conveniently fit these roles. SPI is a good choice among the various communication protocols since it uses a minimum wire count, synchronizes the clocks system-wide to that of the master and contains a mechanism for selecting which Uno32 to poll amongst the dispersed network for its present state. Each slave only needs 4 wires to work within this interface. The master handles the interface similarly with the same 3 wires A, B, C, and one SS line per slave in the dispersed network. The system is shown in block diagram form in figure 49 below:
Figure 49: SPI interface between master and slave.

For clarity, following is a list of acronym definitions:

- Data transmission lines:
  - A) MOSI: Master-Out Slave-In
  - B) MISO: Master-In Slave-Out

- System clock line
  - C) SCLK: System Clock

- Slave select line
  - D) SS: Slave Select

These pins were used as shown on the Uno32 Dev board with the following layout:

Figure 21: SPI pins on the Uno32 development board

**SPI Communication Details**

In synchronicity with the clock, data is pushed out of the master’s shift register onto the MOSI line and into the slave’s shift register at the same time that data is pushed out of the slaves shift register onto the MISO line and into the master’s shift register. That is, as a bit is
pushed out of either the master or slave’s shift register, a new bit is pushed in. The shift
registers never really empty, they just exchange their contents. Research has informed the
decision that slave messages and master messages will not be sent simultaneously over the SPI
channel, but rather a message sent and fully read before the consecutive message is loaded. A
block diagram representation of this functionality is shown in figure 50 below.

![Figure 50: SPI module block diagram.](image)

Note that the shift register is a data link shared by both the master and slave devices, but
is also the docking point for both the reception and sending of data from the RXB and TXB
FIFO’s respectively, on each device’s internal SPI channel.

It is important to consider basic handshaking protocols when setting up a
communication line between two devices, they must both know when and how the hardware
will trigger software events. The two relevant variables in this case are the clock polarity (CKP)
which describes whether the clock will retain a logic high or low value in the idle time of no
data transmission. The clock edge (CKE) defines on which edge transition the reading of SPI
shift register data will occur. By dropping the peripheral bus clock low enough in the
declaration of the SPI channel, ringing in the signal was reduced and data loss minimized due to missed clock cycles.

**Message architecture**

Separate message structures were defined for the slave-to-master and master-to-slave communication lines. The slave messages consist of:

- Message type indicator (voltage or current and component type)
- ID for specified: generation source, transmission line, or load
- Sensor reading

While the master messages contain possible fields with:

- Message type indicator (relay status for load or transmission line, or PWM setting for generation source)
- ID for specified component type
- Relay status
- PWM setting

Lastly, the uppermost bit of all messages will indicate whether the message came from the master (1) or from the slave (0).

The functions to construct and deconstruct messages between the two types of devices utilize bit-shifting, bit-adding, and bit-masking techniques to either set the relevant fields for data transmission, or parse and analyze the relevant fields after reception. The common slave messages fall into one of 6 categories: a current or voltage reading for one of the three types of component: load, generation source, or transmission line. To create for instance a message transmitting the voltage on a load, the only inputs needed are the priority level of the load and its voltage ADC value. These values are bit-shifted to their appropriate indices; a message identifier is added specifically for the deconstruction of these messages, and the identifier that this message came from a slave. A bitwise representation of the message (updated in the present code structure) is depicted in Figure 51:
Receiving the slave messages on the master side is a multi-stage process. First, the message identifier is parsed out; based on its value certain indices are scanned to determine the priority level of the indicated component type. The data type is also parsed out, which informs the selection of a circular buffer (one for each current and voltage) into which to write the transmitted ADC value. The masters control messages are a bit more involved.

The master must construct messages that implement different control functionality based on what type of component it is talking about. If the component is a load or transmission line, the relevant control is the status of a relay connecting that component to the grid. If the component is a generation source, the relevant control is a desired PWM setting. Master messages should never talk about more than one component, and should always have either a relay status or a PWM setting accordingly. Similar to the slave messages, each type of master control message has a message identifier which is included for the receiving slave side.

If a slave receives a master message, it immediately begins to process it by parsing out the message identifier. Based on what it finds, the correct load or transmission line could be connected or disconnected from the grid, or a generation source could be set to a new operating point. These are functions that run locally on the Uno32’s, there will be no blocking in the communication system as the physical grid adjusts to the control. A bitwise representation of the master message is depicted in Figure 52:

```
|<1-bit>| |<1-bit>| |<5-bits>| |<2-bits>| |<3-bits>| |<8-bits>| |<2-bits>| |<10-bits>|
|Device_Type| Unused_bits | Msg_Type | RelayStat | Trans_ID | Load_ID | Gen_ID | PWMSetting |
```

Figure 52: Master message bitwise breakdown.

Message Timing Overview
While selected, the slave will always have a message to send to the master, namely a current or voltage reading from one of its components. The sequence in which these values are read is not important; load priority is a property defined by the order of load handling in the load shedding and reconnection functions (highest index to lowest index), not in the structure of the snapshot the master takes of its slaves.

The slave main function handles more than just the collection of current and voltage readings. Constant updates to the messages they would send to the master occurs after each round of data collection on the slave. Given that the ADC process cycles through all the components for a given bus on the order of milliseconds, the slaves should have sufficient time to recalculate these messages.

There is a notable corner case and easy workaround to address, what messages do the slaves send at system startup? This could be a tricky message to create since the slave may not have taken enough data to fulfill the averaging function. Furthermore, the transients observed at system startup for current and voltage quantities make this data unreliable. This issue is avoided by implementing a brief setup delay, allowing the system to reach steady-state operation, and giving the slaves enough time to construct messages before being asked to send them.

32-bit message protocols were devised for the communication from slave-to-master and master-to-slave. While the messages could have been stitched together from four separate 8-bit messages, the SPI channel is capable of an easy configuration to handle 32-bit transfers. Exhibition of an 8-bit 0xAA readout is displayed in Figure 54 below:
Communication algorithm development

Commands vs. Queries

The master Raspberry Pi was responsible for commanding the slave Pic32’s constantly in order to affect the hardware status of the buses the slave controlled. In order to make responsive and accurate decisions, the RPi also queried the Pic32’s for sensor and hardware status data it had gathered.

Differentiation between commands and queries was performed based on the message type included into each 32-bit message. The function:

```c
int cmdOrquery(uint8_t msgType)
```

on slaveResponse.c on the Pic32 clearly shows how the message type was determined. These two types of interaction required different responses from the slaves and were thus handled in two separate functions as seen in the functions:
uint32_t createResponseMsg(SLAVE_INFO *slave, uint8_t msgType, uint32_t message)

uint32_t serviceMasterMsg(SLAVE_INFO *slave, uint8_t msgType, uint32_t message)

in slaveResponse.c on the Pic32. Whereas commands initiate other processes on the slave (ex: open/close a relay, initialize a timer) the query messages were all followed by functions which created and returned a message with the requested data embedded.

**Software Simulation**

As per good engineering practices, system operations were tested with software simulations prior to hooking up a system to hardware. With the slave control algorithm, this meant using pre-defined datasets for input sensor data rather than reading values from the Pic32 ADC ports.

Datasets were defined for both currents and voltage, in the three possible modes of operation:

- under-threshold
- within threshold
- over-threshold

By switching between these datasets, the varying conditions for load-shedding and short-circuiting were simulated. The consistency with which these simulations reacted correctly to the varying stimuli suggested that issues with data transmission were not a result of deficiencies with the handshaking/acknowledgement algorithm or the SPI transfer function. Furthermore, the simulations confirmed the fact that the SPI messaging clock speed could be increased beyond the 4 kHz frequency which worked with hardware.

**PID Optimization**

During the check-off for the previous quarter, the PID function allowed for the PWM value (determining the operating point of the motor) to settle within 1 second of a step voltage change. When the RPI was brought into the mix, the division of labor between the micros was
set such that the PID control remained on the Pic32, but the control functions which asked for and used the PWM setting were removed to the master controller. This meant that the RPi would have to query the Pic32 for the present setting of the PWM at specific times in the control algorithm, notably in the following functions on the RPi in SensorControls.c:

```c
int ShedLoads(SLAVE_INFO *info)
int RegainLoads(SLAVE_INFO *info)
void PrintSensorReadings(SLAVE_INFO *info, FILE *f)
void PWMvalueRequest(SLAVE_INFO *info)
```

Whether or not the PID function was serviced depended on a 0.1 second timer, which was checked consistently in a while loop before the data ready flag was checked for the SPI messaging function. Initial runs including the RPi and Pic32 showed that the PWM control had an unstable response to changes in load connectivity (following the shedding of the first load for instance), leading to oscillations of increasing magnitude. The control voltage on LOAD_1 thus varied continually and was unable to settle within the 1% tolerance required for the system to regain loads. This was solved with a couple modifications:

- readdressing the period of the timer which allowed for the PID to be updated from 0.2 seconds to 0.1 seconds
- setting limits on the amount by which the PWM controller could adjust in any one PID adjustment call
- calibrating the PID controller gain values. Readjusting the gain parameters for the proportional, integral, and derivative control to the PID controller had a substantial effect on the settling time of the PWM signal. Values were modified from:

$$K_p = 100, K_i = 0, K_d = 10 \quad \text{to} \quad K_p = 30, K_i = 0.5, K_d = 5$$
This was the most successful optimization, bringing the response time of the PID control to less than 2 seconds with only a single cycle of overshoot.

**Multiple bus system**

The multi-bus system could not function without external hardware since the RPi only was configured to put out SPI data and clock lines on a specific GPIO ports. This meant that the SPI lines would have to be shared as seen in the following figure:

![Multiplexing circuit for multi-bus system](image)

*Figure 22: Multiplexing circuit for multi-bus system*

Further handling was performed through messages, to let the slaves know in software when they were online or offline. This system did not succeed, yet it was not fully investigated due to the need to finalize the implementation of a single bus system.

**Averaging function optimization**

Issues were noticed with the averaging function where at startup, values read across the loads were inaccurate and consistently smaller than expected. Taking a look at the averaging function in place revealed that the input readings were being averaged over the whole array, which initially was filled with zeroes. The function was rewritten as seen in runningAvg.c on the Pic32. The advantages of the new edition are an optimization of the function running as O(n) to O(1) and the ensured accuracy of the output values over all time.
Problems and Attempted Fixes

Data integrity

The DC bus system continued to be developed with attempts at maintaining data integrity for the messages between the master and slave micros. The first step taken towards this goal was to ensure that all messages sent from the master to the slave were recognized and acknowledged as having been processed on the slave micro.

Handshaking/Acknowledgement

Handshaking algorithms are a common technique used to establish a connection between two systems with assured message transfer completion. In an SPI communication system (where the master always initiates message transfers) data is transferred synchronously on a clock line, but the processing to take information from the data lines into the internal buffers on the micro-controllers is itself asynchronous between the chips. The messages can get out of sync in time so it is necessary to ensure that the slave has received and processed commands as the master sends them.

The handshaking algorithm built for this system employs a no-op message to hold back the master from moving forward in the control algorithm until the message it has just sent (MOSI) is responded to with a unique acknowledgement code and the message type of the command just sent (on the MISO line). This interaction is implemented on the RPi in the SPItransferCalls.c file with the functions:

- `uint32_t singleSPIrx(uint32_t message)`
- `int waitForAck(uint8_t ackCode, uint8_t msgType)`
- `uint32_t SPIsendFluff(void)`

and is pictured below:
As shown, the master sends no-op messages following each command/query in order to force the slave to return back the value currently on its transfer buffer. At the slowest SPI clock speeds (>8 kHz), the no-op messages almost never show up on the input message log on the master. At these speeds the slave has enough time to process inputs so that it can always respond with an acknowledgement of the MOSI message rather than returning no-op messages on MISO. At higher speeds (>16kHz) the no-op messages become prevalent in the message log of the master, meaning that the slave hasn’t had time to update its internal buffer and must return a no-op message.
The presence of no-op messages does not equate to errors, but rather are an indicator of how the Pic32 boards can operate as a part of the communication framework only at kHz frequencies. The slowest part of a process determines the operating speed of that process, and in this case while the SPI messaging frequency can be continually ramped up, the frequency of calls to the control algorithm are limited by calls to ports on the Pic32.

**Delay function**

Another attempt to increase data integrity was through the use of delays on the master side micro-controller. The idea here was that with the limited hardware I/O space on the RPi, the master’s internal SPI buffers could be building up data, leading to buffer overflow and overwrites as it processed data at a slower rate than which it was receiving that data. In the SPItransferCalls.c file on the RPi the function:

```c
void delayMasterMS(float timeMS)
```

allows the user to specify an amount of milliseconds to force the master to idle. This delay time was implemented as a buffer between consecutive messages in the waiting for acknowledgement and sending fluff loops, which continually instantiates transfers between the micros. While these delays didn’t end up solving the issues facing the SPI transfers, the internal mechanism was used to show how print statements can significantly increase the runtime of a function, adding 2-4ms of runtime per call.

From speaking to other teams implementing SPI, delays tuned to each function were the only way in which to keep the master from over-loading the slave with messages. At the speeds this system was able to handle (limited by the ADC rate on the Pic32), delays on the order of milliseconds were not useful. Delays would have only been necessary had the system SPI transfers been running in the MHz range, as the limits of the RPi in particular (discussed later) would have been pushed to the limit.

**Message Error Handling**
Errors in the messaging system appeared commonly in the transmission from slave to master, which (performing the acknowledgement checks) were interpreted as a MSG_FAIL type. Mistakes appeared in as few as a single bit, or up to a slew of consecutive bits. The umbrella solution to this problem was to take any message that didn’t acknowledge a message successfully, disregard it, and have the master resend the message that had been responded to with an inappropriate message.

This technique can be seen throughout the master’s code in the functions:

- SensorControls.c
- TimerControls.c
- SPItransferCalls.c

The master was given 5 chances to send and receive the message correctly before moving on to the consecutive command in the sequence. This introduced the possibility that a command was not handled within a single call of the slave control algorithm. On the other hand, this was seen as preferable to the default of giving each command one chance to be sent correctly, or if the 5x limit was removed, the chance that the master gets trapped in a loop sending a message without ever receiving the intended response.

This technique was fairly successful for cases when the message had been incorrect by a single bit. The following message would generally be correct. This technique was less effective when the message had been greatly distorted. It would be continually followed by an equally distorted message, leading to a message time-out and the control algorithm moving on to the next line of the control algorithm.

Microcontroller Solutions

The issue of data loss continued to plague the transmission, so a few outlier ideas were investigated, notably through the use of:

1. Real-time OS
2. Threading
3. UART

1) Concerns were raised regarding the speed of the RPi. The requirement to deal with paging and swapping, could limit its ability to deal with the control algorithm running as a program in ‘userland’ on Raspbian Wheezy OS. The PREEMPT_RT package was deemed proper to deal with this issue, but was unsuitable to run on the SD card.

2) Threading was attempted through Python, but this ended up being unsuitable and was quickly abandoned.

3) UART was recommended as a replacement to SPI, but it was far too late in the project to make the switch and it was quickly dropped.

Solution on the Horizon

Noise, EMI Interference

At a certain point, there was a shift from trying to handle the errors, to trying to find the source of the errors. Due to discussions with technical experts in industry, professors, and TA’s, the deficiencies in the data integrity had been explained as a result of insufficient data throughput capabilities on the RPi and Pic32, leading to buffer overflow and conflicting bits. When the data lines were viewed on the digital oscilloscopes however, another story was told.

The following images are a few of the scope traces showing the MISO and MOSI data lines in yellow and blue respectively, along with the master clock line in green.
Figure 57: System operation under normal conditions (1/3)

Figure 58: System operation under normal conditions (2/3)

Figure 59: System operation under normal conditions (3/3)
These scope traces show the root cause of the problem, EMI interference caused by the periodic rotation of the DC motors. The data lines are clearly affected by this interference, subject to spikes and ripples of large enough magnitude to cause logic changes as seen in the above figures.

The noise on the data lines was undoubtedly a function of the supply voltage to the motors. While the above figures being given a supply voltage of 24 [V], the following figures show the exact same microcontroller setup, with a supply voltage of 22 [V]:

![Figure 60: System operation at 22 [V]](image_url)

And the following figure showed the setup with a supply voltage of 20 [V]:

![Figure showing system operation at 20 [V]](image_url)
There are obvious incremental benefits resulting from decreasing the supply voltage, leading to a system with no message interference. This matches observations of the system performance during the system startup. The messaging function would work with stellar integrity as the system was turned on, but performance got worse and worse as the supply voltage ramped up towards the intended rail.

Handling EMI interference

Attempts were made to handle the discovery of the EMI disturbance of the data lines from various angles. The various solutions included:

- Ground wire wrapping
- Passive RC low-pass filter
- Foil-wrapping

Each of these fixes showed incremental benefits in reducing the ripples on the data lines, but were unable to eliminate the spikes as shown in the following figure:
The only complete solution was to reduce the supply voltage to around 18 [V]. This reduced the amount of generation available to the loads, but the clean transmission of data was more than desirable a trade-off for this loss. The DC system was built up to include the same functionality as provided by solely the Pic32. Albeit it occurred at a slower pace due to the need for the triggers to pass from the hardware, through the Pic32, to the RPi, and back, as opposed to simply from the hardware to the Pic32 and back. Regardless of this longer chain, the system could respond to the short-circuit protection (the most time-sensitive action) in less than a second.

**Conclusion:**

The result the Nanogrid project was a single bus DC prototype which can execute load shedding and fault handling. The step response from load changes smoothly adjusts voltage within an acceptable 1-2 second time frame thanks to accurate sensor readings and a well-tuned PID control. The system is able to preserve the correct loads based on their priority status and available generation. Expanding the tiered communication scheme will allow the system to incorporate multiple buses, allowing for investigations into topics including transmission lines and generation balancing. A more powerful and precise AC generation will be required in order to ease the issue of finding components and increase predictability of motor operation.

Future progress will require a restructuring of the communication to address throughput and data loss reduction. Additionally, the control algorithm will need to service
local buses in parallel instead of using a polling scheme. This is necessary to avoid loss of response across the system during emergency conditions. Phase locking multiple AC generators will require updating the control algorithm by measuring and manipulating frequency as well as predicting phase changes in order to compensate for delays in the communication protocol.
Appendix:

Figure 62: Sensor Drivers
Figure 63: Bus Relays
Figure 64: Controller Logic
Figure 65: Voltage Rectifier Circuit for Load 1

Figure 67: Current Rectifier for Load 1
Figure 68: Voltage Zero Cross Circuit

Figure 69: Current Zero Cross Circuit
Figure 49: Single Supply Audio Amplifier LM1875
Figure 50: Multiplier Circuit with Variable Gain Amplifier and Audio Amplifier
References:

12. “PID Control with MATLAB and Simulink” MathWorks, February 25th 2014