

**Engineering Team:**

Matthew Belden

Arash Ebad

Cesar Elizondo

Matthew Garcia

Darren George

Thomas Gerstenberg

Chad Higgins

Cynthia Lopez

Kristin Makowski

Mark Parangue

Robert Paulino

Louis Tiger

**Sponsored By:**

San Diego Gas & Electric

**Submitted To:**

John Kennedy and Dr. Lal Tummala

San Diego State University

May 16, 2014

Table of Contents

[Abstract 1](#_Toc387917509)

[Introduction 2](#_Toc387917510)

[System Design 4](#_Toc387917511)

[User Interface/Web Application 4](#_Toc387917512)

[OSI PI Server 7](#_Toc387917513)

[Central Communication Hub 8](#_Toc387917514)

[Zigbee Communication and Protocol 11](#_Toc387917515)

[Arm Cortex M3 12](#_Toc387917516)

[Variable Frequency Drive (VFD) 14](#_Toc387917517)

[Sensors 16](#_Toc387917518)

[Pump 17](#_Toc387917519)

[Relays 18](#_Toc387917520)

[Printed Circuit Boards (PCBs) 19](#_Toc387917521)

[Fabrication 21](#_Toc387917522)

[Conclusion and Recommendations 23](#_Toc387917523)

[References 25](#_Toc387917524)

[Appendices 26](#_Toc387917525)

[Appendix A: Main PCB Schematic 26](#_Toc387917526)

[Appendix B: 80/20 Inc. Aluminum Piping 27](#_Toc387917527)

[Appendix C: AutoCad Fabrications 28](#_Toc387917528)

[Appendix D: Final Project Schedule 30](#_Toc387917529)

[Appendix E: Complete System Fabrication 31](#_Toc387917530)

[Appendix F: Bill of Materials 31](#_Toc387917531)

# 

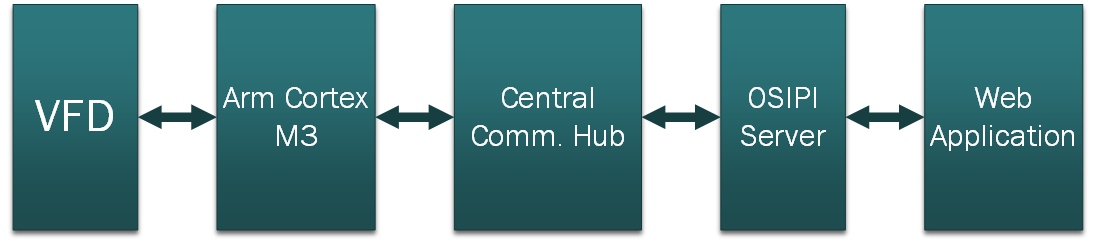
# Abstract

The cost of energy and total power consumption continue to rise on an annual basis, making power-intense devices such as pools, air conditioners and appliances cost more and more every year. How can San Diego residents save money while still being able to use expensive motor-driven devices?

A Variable Frequency Drive (VFD) gives the ability to control the frequency being delivered to a motor, ultimately giving control over the power consumption of the motor. Without the use of a VFD, a motor is always running at its full potential, and most likely using more energy than it needs. A user-friendly interface to a VFD control system will provide a power-management system for the user to ultimately save money.

# Introduction

As of now, there is not much information about using a VFD for residential purposes. Many opinions online are based off of random testing without any real validation of their results. Our goal was to develop a system that will let us test a residential application of a VFD in order to determine whether it is worth the investment to purchase a VFD. The end result of the system is a user-friendly interface to control a VFD, which will be powering a small water pump, and collects data about the current power consumption in addition to other metrics. The system diagram can be seen in **Figure 1**.



**Figure 1:** System Diagram

Starting on the left, the VFD is connected to a motor (in our case, a circulating pump). The VFD controls the frequency of the motor, which alters the flow rate of the water based on the chosen frequency.

Our secondary microcontroller, an Arm Cortex M3, will send and receive commands for the VFD to operate. Three sensors (flow, pressure, and temperature) and two relays are also controlled by the Arm Cortex. The data from the sensors is gathered every second. Flow rate is measured in gallons per minute (gpm), pressure in pounds per square inch (psi), and temperature in degrees Fahrenheit (°F). All of the connections are via physical connection. Once all of the data is collected on the secondary microcontroller, it sends all of the information to the Central Communication Hub through wireless communication.

The Central Communication Hub (CCH), a Raspberry Pi microcomputer, acts as our primary microcontroller and as a wireless gateway between the Zigbee protocol (802.15.4) of the Arm Cortex and the more common WiFi protocol (802.11) of the home network. In addition, the CCH hosts a touchscreen user interface that can control the VFD and set up the system. The data received from the Arm Cortex is then forwarded to be stored on the OSI Pi Server.

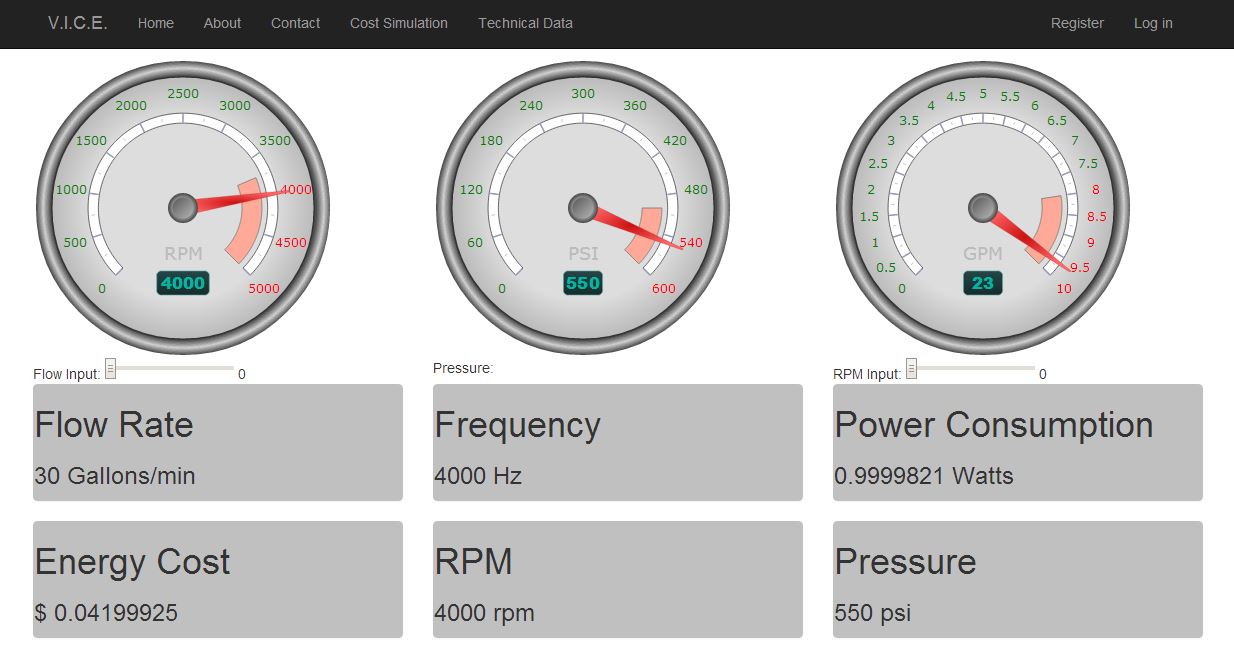
The OSI Pi server stores all of the data collected from the system with their timestamp. A web application is connected to the OSI Pi server so the user can view the current data as well as send commands to the VFD. Data shown on the web application includes the current flow rate, pressure, and temperature, and power usage of the system. This data is used to compare the differences of powering a motor from the VFD versus powering the motor from a wall outlet. Our project integrated a previous project, the Energy Wise Outlet (EWO), as our power monitoring system and provides the 120V at 60Hz AC source our system. The EWO uses wireless capabilities to transfer data consisting of voltage, current, real and reactive power, and power factor directly to the OSI Pi server.

# System Design

## User Interface/Web Application

Our design acts as a VFD demonstration model for residential energy users, specifically SDG&E customers. The problem this model solves is that most of those consumers are uninterested or uneducated about how they are charged for energy and that there are easy ways to consume less energy. In order to show that incorporating a VFD is a worth-while energy solution, we needed a proper user interface. The main purpose of the user interface is to give customers hands-on control of our demonstration while quickly delivering data to show the significant energy savings at lower frequencies.

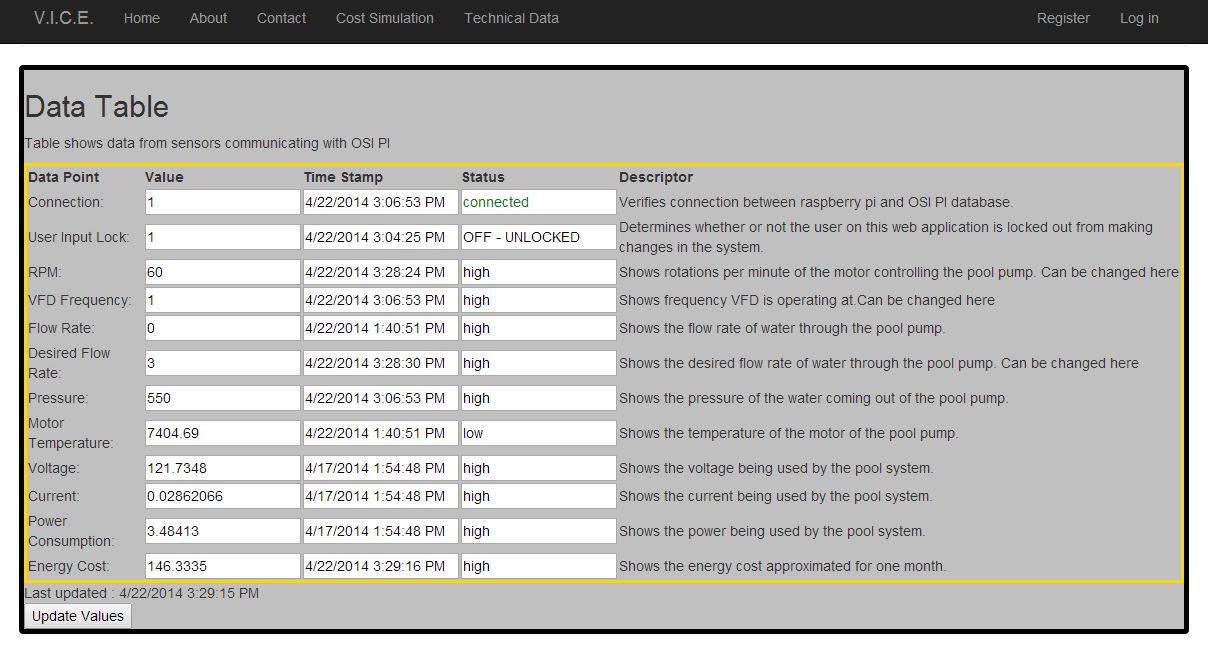
The first task in developing the interface was to determine a platform. We described our user as a residential energy consumer with multiple small motors throughout the house, including a pool pump. Although the ultimate goal was to create a product someone would want to incorporate into their home, our immediate focus was to put on a demonstration for multiple users in a trade-show setting. So knowing that the interface is for residents and multiple users, we decided on a web interface hosted on a local area network (LAN). We knew most homes have local area networks that they could connect all of the components of this project to, just like we would in a demonstration on our own LAN. Also, having multiple users at a demonstration would mean each individual’s mobile device could have a different operating system and web browser, making Microsoft’s ASP.NET framework favorable since it would work on all platforms and devices. **Figure 2** shows the main page of the web application based user interface.



**Figure 2:** Web Application User Interface

Choosing an ASP.NET Web Application allowed us to have a responsive interface which could actively access our server and run on personal computers as well as mobile devices. This framework also allowed us to write functions for events, graphical enhancements, and server communications in C#. On the main page where our user will be the viewing the system most of the time, we wanted continuously updating data values and some way to graphically represent them. We wanted the main page to be informative on a quick glance, but also interesting to look at and play around with for learning purposes. The final main page consisted of six panels as shown in **Figure 2**, displaying values updated from or calculated from our server’s database. The values for flow rate, VFD frequency, motor rpm, and water pressure were updated every second from the server. The values for power consumption and monthly energy cost were calculated and updated every second based on the voltage, current, and power factor values we received from the EWO. To make these values more visually appealing, we added three gauges above the panels. On these gauges, the values for flow rate, pressure, and rpm are updated every second along with the panels. We decided the values the user would be most interested in changing would be the flow rate mainly, and possibly the motor rpm. Under the gauges are inputs for the user to enter the desired value, which is then updated in the server database to be sent through our communication path to the VFD.

Because a future benefit of customers using VFDs could be for SDG&E to have more control of the grid, we added a control page to our web app as seen in **Figure 3**.



**Figure 3:** Control and Data Page

This is the technical data page that displays all of the values we store in our database as well as calculated values for power and cost. From this page, SDG&E and our group’s engineers can monitor the system values and change important values, such as the current VFD state and the desired flow rate. In order to really prove VFD cost savings, we also included a simulations page. From this page the user can run a one minute simulation, see the pump running, and then see the results appear with a comparison between power consumption with and without a VFD. All of the data that we are able to see is retrieved from the OSI PI server.

## OSI PI Server

OSI PI is a real-time data management infrastructure and secure database for storing and searching data. This is a historian database, which means it stores data values with its timestamp inside the database as points, and is also a staple to SDG&E. We used system management tools to create multiple PI points in the server to track all values for the Energy Wise Outlet (EWO), the VFD, and system management. The list of the PI points being tracked is shown in **Figure 4**.

|  |  |  |
| --- | --- | --- |
| **Points Monitored in the OSI PI Server** | | |
| **EWO Points** | **VFD Points** | **Control Points** |
| Voltage | Frequency | State (On - using VFD,  On - not using VFD, Off) |
| Current | RPM |
| Real Power | Pressure | Web Lock (lock the web app from updating the VFD) |
| Apparent Power | Temperature |
| Power Factor | Flow Rate | Desired Flow Rate |

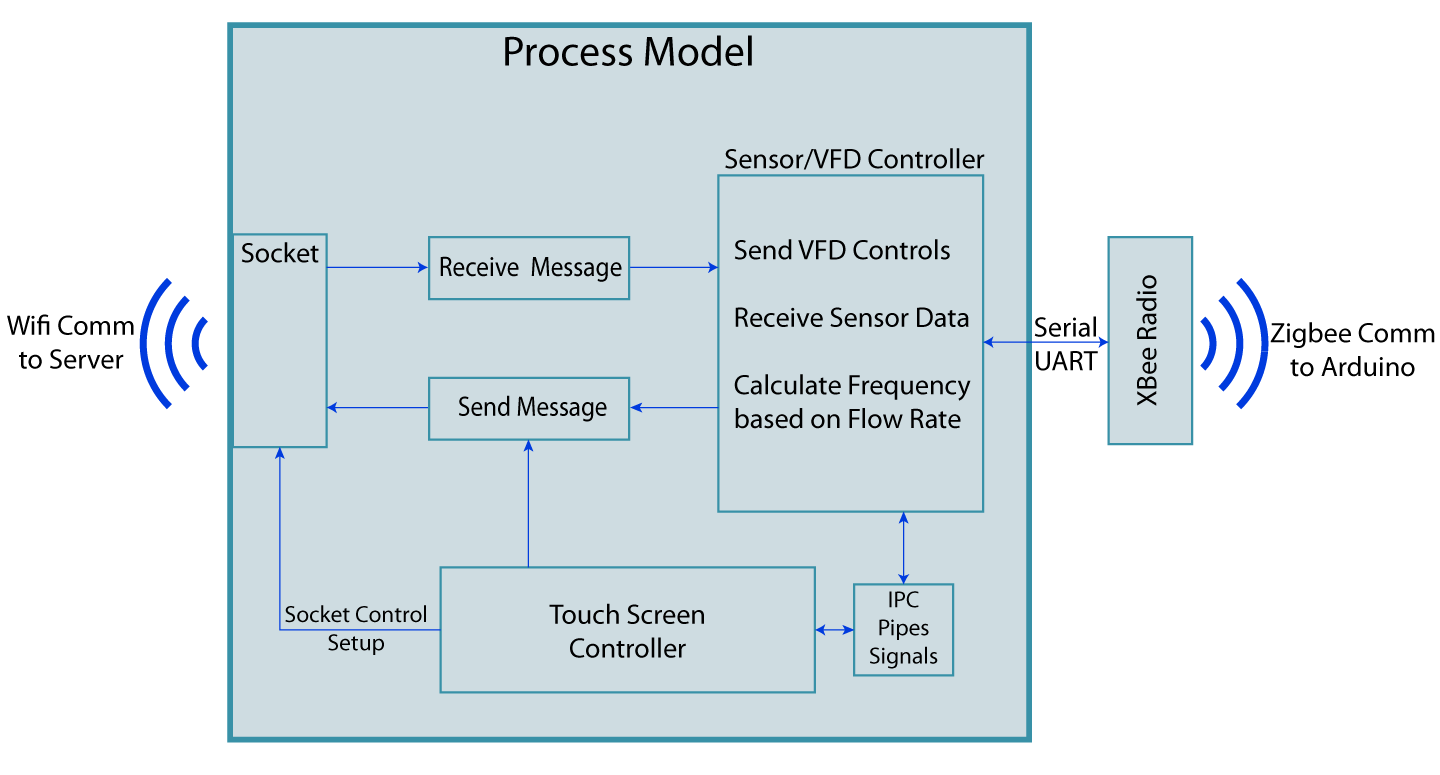
**Figure 4:** List of points in the OSI PI database.

To store these points, we developed a multi-threaded C# program incorporating two threads. One thread takes in data from the Central Communication Hub (CCH), which is forwarding the sensor data, and from the EWO. The data received from the EWO and the CCH is in the form of ASCII strings. These strings are then parsed and update the necessary Pi points in the database. The server is continuously listening to a port in this thread for either a connection from the CCH or from the EWO. When a connection is accepted, the client sends its data, and then closes the socket. By never keeping an open connection, this enables us to use the same port for the multiple clients.

The second thread monitors the PI points in the PI database to see if the web-application has updated any control points with new data. If there is updated data, it sends the updated controls to the CCH in the form of a string, which then relays the controls to the VFD (i.e. changing the frequency or state of the VFD).

## Central Communication Hub

The Central Communication Hub is a Raspberry Pi microcomputer fitted with a 2.8 inch touchscreen and a Zigbee XBEE transceiver. The Raspberry Pi is a small, cheap, powerful device that runs a Linux operating system with a 700MHz processor. The reason we chose this as our CCH is because it provides easy networking services, a large support base, and gives the opportunity for expansion in our system to multiple devices. The CCH serves three main purposes: control the VFD frequency based off of flow rate, host a small user interface for system controls, and be the gateway between WiFi and Zigbee wireless protocols. An overall process model of the CCH is shown in **Figure 5**.

**Figure 5:** Process model of the Central Communication Hub

The VFD controller process was coded in C and is a super-loop that primarily focuses on communication with the Arm Cortex. The CCH to Arm Cortex is a master/slave communication scheme where the CCH (master) sends a data packet to the Arm Cortex (slave) every one second. The data packet is either a request for sensor data or controls that the Arm Cortex must execute. The response from the Arm Cortex is always a data packet containing the sensor data. This allows us to collect sensor data every one second and give us a maximum control delay of one second. When the CCH receives the data packet from the Arm Cortex, it parses the packet and sends the data to the server to be stored in the OSI PI database. Next, it checks whether the server sent any controls to the receive thread and determines what type of packet to send to the Arm Cortex (request or control).

Since the user only has access to changing the desired flow rate, a frequency must be determined to send to the Arm Cortex. This was implemented by a lookup table and a proportional feedback loop. When controls are received from the server, the initial frequency is set by the lookup table, which was formed by linear interpolation of tested frequency/flow rate pairs. For our system, the lookup table for frequency is 10.41\*Desired Flow Rate + 13. After the initial frequency is set, the feedback loop checks whether the actual flow rate meets the desired flow rate. If the difference in flow rates is outside of a given threshold, it will adjust the frequency until it reaches inside the threshold.

The second main purpose of the CCH is the touchscreen user interface. The touchscreen is a small 2.8 inch screen which allows for basic controls of the VFD and system setup. The controller for the user interface is done entirely in Python, and the skeleton of the program is based off of David Hunt’s Lapse-Pi project (see references below). It implements the PyGame library and custom button and icon classes provided by David Hunt. It is an event-driven program where each button has a callback function associated with it when pressed. The background and each button image was custom made using Adobe Illustrator for this user interface. The main page of the user interface, displayed in **Figure 6**, primarily gives the user control over the flow rate and the current state of the VFD. In addition, there is an option to lock the settings and state with a passcode, which is useful for a trade show environment where the users should only be allowed to change the flow rate. The settings button contains another page with options to set the server IP address, enable the web lock, and restart the services pertaining to the socket (i.e. if the server IP address was updated, the services will restart pointing to the new IP address).

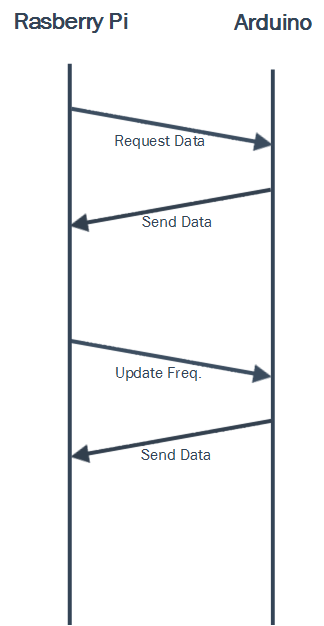


**Figure 6:** User interface on the CCH touchscreen

The idea for having a CCH allows further expansion in the system to accommodate multiple VFDs or other devices while only needing one controller for all of them. With the industry split on whether small sensory nodes should communicate over WiFi or Zigbee, fitting the CCH with both allows it to aggregate data from both types of communication protocols. While the current system is a linear network topology, the addition of a Zigbee transceiver gives the CCH the opportunity to be the hub of a star network topology.

## Zigbee Communication and Protocol

**Figure 7** below models the data structure when data is passed from the Arm Cortex to the Central Communication Hub. The data from the sensors, relays, and the VFD will be gathered and organized into a data packet in the Arm Cortex. This packet is composed of five parameter values. The parameters consist of the flow rate, frequency, temperature, pressure, and the relay state. Within the packet, the data type for the first four parameters are floating point, while the last parameter will be comprised of an integer.



**Figure 7:** Zigbee Protocol

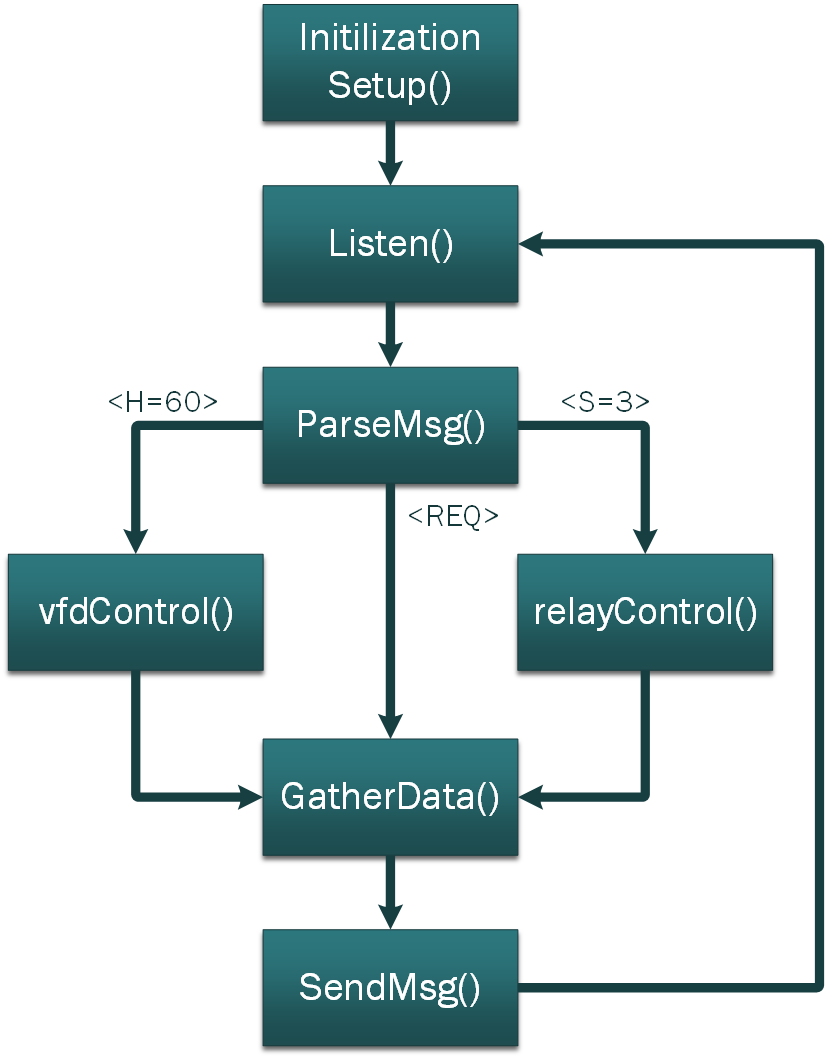
## Arm Cortex M3

Within the early stages of our design, we planned on using just one device that would control both the hardware devices and communication. We decided to split that job between two devices. One of these devices was the Arduino Due microcontroller board. The hardware team selected this board based on multiple properties. These properties would include processing power, power consumption, I/O pin availability, serial UART port availability, reference availability and programming compatibility.

The Arduino Due contains an ARM Cortex-M3 32-bit microcontroller. This allows the Due to outperform many 8-bit microcontroller boards. The ARM core runs at 84MHz clock speed and contains 0.5MB of flash memory, 96KB of SRAM, and a DMA controller to further increase the speed of memory intensive tasks. Another advantage of this board is the ability to operate at a voltage of 3.3V. In our design we chose to power it with a 5V supply from an AC-to-DC converter, which was regulated to 3.3V by the Arm Cortex board. This allowed for enough voltage to power the sensors and XBee module.

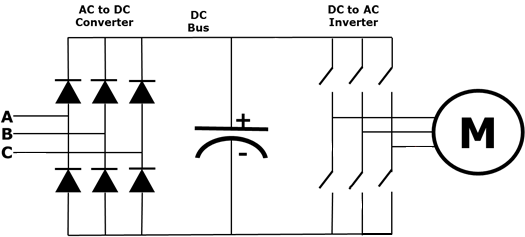
With this board, we also had more than enough pin availability for each of our devices. The flow sensor, temperature sensor, and two relays are each connected to a digital pin and the pressure sensor is connected to an analog pin. In addition, we also needed a board with at least three serial UART ports that would be used for the XBee transceiver, the VFD, and a PC for debugging. Also with the popularity of the Arduino Due board, we were able to come across multitudes of references to help in our design of the project. We were able to obtain detailed datasheets on the microcontroller as well as the schematics and pin-outs of the Arduino board and its RS485 shield, which is used for communication with the VFD. With the schematic for the Arduino RS485 shield, we were able to replicate the design onto a PCB. Another big factor in selecting the Arduino Due microcontroller board involved its programming compatibility. Unlike many microcontroller boards, the Arduino Due comes with its own software development environment, the Arduino IDE. With this software, it allowed us to easily develop, program, and test our software on the Arm Cortex-M3.

The function of the Arm Cortex is to act as the brain for our hardware components, and was written in C++. The process begins when the Arduino Due board is powered on and initializes all settings necessary, including setting up the board's I/O pins, setting each serial UART baud rate, and declaring the global variables. After initialization, the microcontroller waits for a message from the Central Communication Hub. Once a message is received, it will be passed onto the parser. The parser determines if the message is a control statement or a request for data. If the message is a control statement, the program will then decide whether the statement will adjust the VFD frequency, the relay state, or both. If the message is a request for data, the microcontroller will begin to gather the values from the flow rate, pressure, and temperature sensors. The program will then assemble these sensor values, along with the frequency and relay state into a data packet to be sent to the CCH. Once the data has been sent, the program will then go back to listening for a new message, thus repeating the cycle, as seen in **Figure 8**, until the system is powered off.



**Figure 8:** Arduino processing loop

## Variable Frequency Drive (VFD)

A Variable Frequency Drive (VFD) is a device that can be used to control the frequency and voltage of an electric motor. Without the VFD, the motor is connected directly to a supply, which will cause the motor to always run at maximum speed. This is not an efficient way to use the motor because it always consumes the greatest amount of energy it possibly can. With a pool pump as an example, it is not necessary for the motor to always run at full speed. Thus, with the VFD we can set the frequency to a value that we prefer or set it up so that it only uses the amount of energy required for the motor to run. The motor will consume less energy and the customer can save money on their electric bill.

**Figure 9:** 3-Phase Variable Frequency Drive

The VFD consists of an AC-DC converter, a DC bus, and a DC-AC inverter, shown in **Figure 9**. The supply voltage first goes through the AC-DC converter and the DC bus to make sure that the voltage is at a steady level before entering the DC-AC inverter. Depending on the user-chosen frequency to be set, the inverter will output pulses generating the amount of voltage desired to power the motor at the chosen frequency.

The main criterion for our choice in VFD was that it supported some type of communication protocol. The VFD used in our system supported Modbus RTU communication allowing us to send commands to our VFD (slave) by means of a microcontroller (master). Modbus Protocol is a serial communication protocol that allows a master device to control one or multiple slave devices through an eight byte data stream. The eight byte stream consists of a one byte slave address, one byte function code, two byte register address, two byte register value, and a two byte cyclic redundancy checksum (CRC) for errors. **Figure 10** below portrays the byte stream sent to the VFD. Using a UART, the microcontroller passes the data through a RS485 differential signal converter to the VFD. A RJ45 Ethernet cable was used in order to access the RS485+ and RS485- pins on the VFD.

**Figure 10:** Data stream sent using Modbus Protocol

## Sensors

The flow sensor was the most important sensor in our design.  The original idea was to allow the user to be able to set the flow rate and have the system adapt and maintain the user’s chosen flow rate by changing the frequency according to the flow rate read from the sensor.  We chose a basic Hall Effect flow sensor.  As the water would flow through the sensor, the spindle inside the sensor would spin.  Each time there was a spin, it would create a pulse.  The code implemented in the Arm Cortex counts the number of pulses by measuring the rising edge of each pulse. To convert the pulses to gallons per minute, the following equation was used: , where the period is the sampling period for pulses in seconds. The sensor was ½ inch on each side, which was perfect for our system as we had all ½ inch piping on the output of the pump.  The sensor has only 3 wires: a ground wire, a power wire, and a signal wire.  The sensor was powered from our 5V rail and used a 10kΩ pull-up resistor for the signal into the Arm Cortex.   Because the Arm Cortex has a max input voltage of 3.3v, we used diodes as protection for the input pin.

A temperature sensor was selected to allow for monitoring of the motor temperatures. This information is useful because we can detect if a motor is overheating as well as warn users if temperatures were too hot. The temperature sensor utilizes the 1-wire protocol, which consists of a configurable digital reading of the current temperature.

The pressure sensor outputs an analog voltage in the range of 0.5V to 4.5V on a scale from 0 psi to 1000 psi. To test the sensor, we used an analog pressure gauge, which allowed us to get an accurate reading that we could compare to our electronic pressure sensor.  The analog voltage is converted to a digital value in the Arm Cortex’s 10-bit analog-to-digital converter.  This gave us a range of 0 to 1023, corresponding to 0V and 5V respectively. Because our system max pressure was around 7 psi, we used linear modeling based off of the analog pressure gauge in order to get the correct psi. The pressure sensor was also supplied by the 5V rail and used the diode configuration for safety on the Arm Cortex.

## Pump

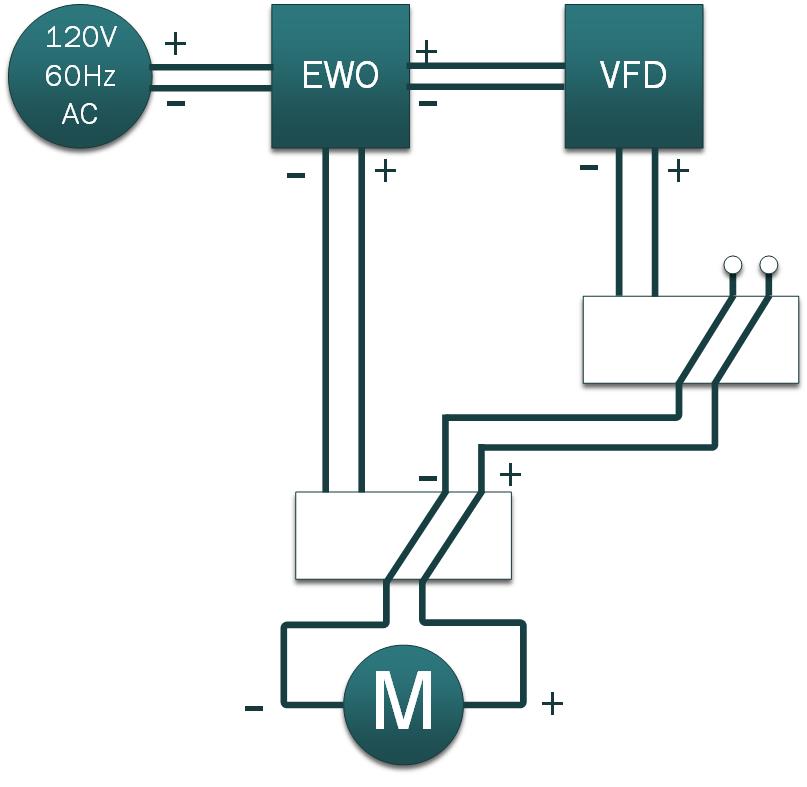
When considering the type of pump that would be required for our project, we needed to consider two factors. First, our pump was chosen to be single phase as that was the average type of pump for residential customers that would own a pool. It also needed to be a split capacitor motor due to the single phase VFD that we chose. For our pump, we used a GRUNDFOS UP1542-F Hot Water Circulation Pump, 1/25HP, 115V model. Since it was designed as a circulation pump for water, it fit our requirement for a device that could simulate the workings of a pool-pump. On a mechanical level, it was a reciprocal "windmill" style pump which utilized a rotating apparatus to circulate water.

Once the pump was integrated within the system, numerous issues presented themselves and were dealt with systematically. One such issue was that the pump required constant water flow, which meant that the system had to be nearly airtight and not allow unwanted gasses into the tubing. Another problem was that the flow rate for our original pump was too low, as it only provided around 3 gal/min of circulation. A stronger pump of the same style was acquired, and this was able to provide a more acceptable 5 gal/min. The biggest problem we had to be concerned with was powering it with the use of relays to prevent damage to the output of the VFD.

## Relays

Our system was designed so that we didn't have to constantly unplug or plug in different cords to switch between the 120V 60Hz wall outlet and the VFD.  To solve this issue, we used a system of relays controlled by the Arm Cortex.  The design concept was to have 3 different states.  The first state is an open circuit and nothing was powering the motor.  The second state sets the pump power to a 120v at 60Hz AC power source.  The third state sets the pump connection through the VFD.  In order to accomplish this, the output of the VFD is fed into a relay with normally closed contacts while the wall outlet feeds into the same relay with normally open contacts.  This relay when energized allows us to switch between the wall outlet and VFD respectively.  Now to have a completely open state, another relay was installed on the line of the wall outlet before it went into the normally open contacts of the first relay.  This allowed a completely open system when both relays were energized.  The complete relay system designed is shown in **Figure 11**.

In order to control the relays from the Arm Cortex, the output pins for each relay are fed into the base of a BJT, with 5V at the emitter and the relay coil at the collector. Setting the output pin high placed the BJT into saturation and allowed the 5VDC to energize the coils.  Two LEDs were also incorporated to allow a visual representation of which state the system was in.  This allowed us to see it was switching properly and more importantly allowed users who were not familiar with our system to understand how the switching is working with a visual aid.



Relay

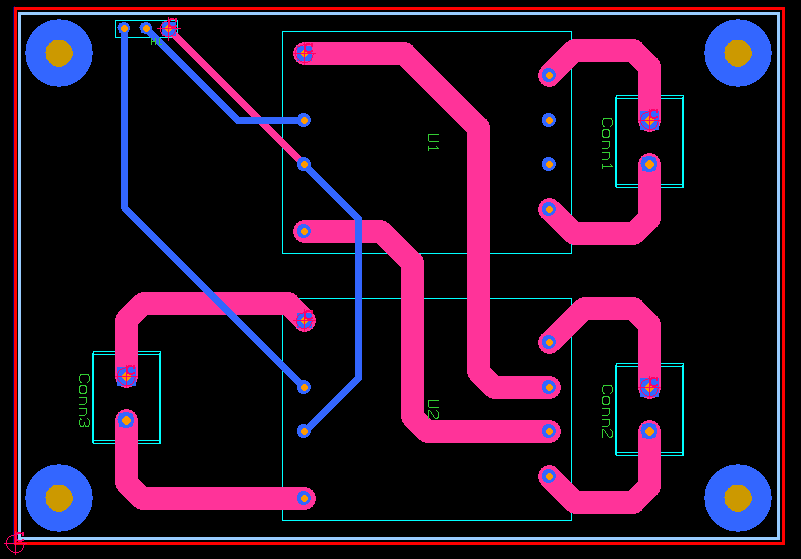
Relay

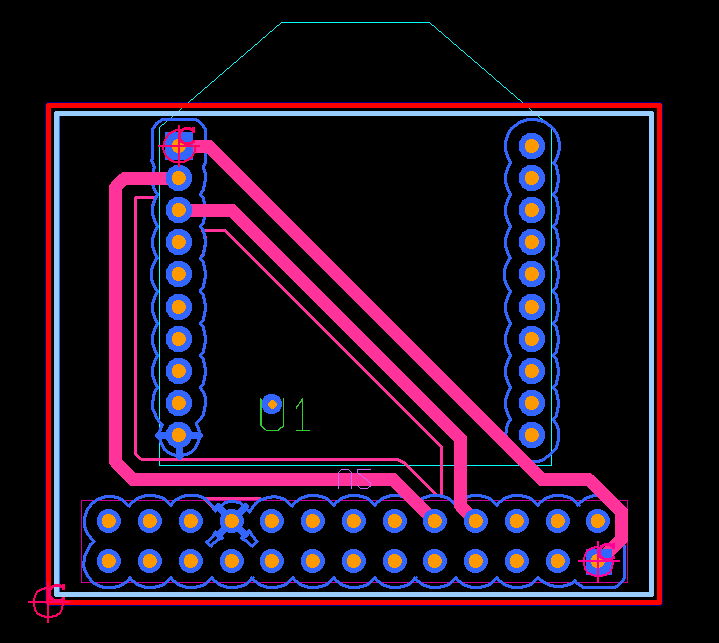
**Figure 11:** Relay configuration with the open state shown

## Printed Circuit Boards (PCBs)

One of the most important aspects of our system is the use of printed circuit boards (PCBs). We have a total of three PCBs, one of which is specifically for the software side of the project. The simplest PCB designed was the Zigbee XBEE transceiver mount for the Central Communication Hub, as seen in **Figure 12.1**. This was designed to take up the least amount of space possible. The second PCB that was completed is the board housing the switching relays, shown in **Figure 12.2**. Although our system does not work with currents exceeding 1A, we wanted to separate the relays from the rest of the components because they deal with 120V, requiring creepage distance to be considered. Creepage is the shortest distance between two conductive parts measured along the surface. According to the IEC60950 standards, the minimum creepage required for a working voltage of 100Vrms is 1.4mm and for 150Vrms it is 1.6mm. Thus, the traces have a creepage greater than 2mm, or approximately 78.74mils, apart. Although it was not necessary, it was decided to take an extra precaution and modify the trace width to accommodate higher than expected currents. The trace width was calculated using formulas from IPC-2221. The minimum trace width for boards with 1oz of copper to accommodate 1A of current is 30.8mils. Thus, the traces were changed to be 100mils wide, making the board capable of allowing over 2A of current to safely pass through. The main PCB, which is located in **Appendix A**, did not require such specific considerations as the relay board did, but it was designed so that each component has easy access whenever troubleshooting is necessary.

The biggest issue faced on the boards was the fact that the grounds weren’t connecting. For instance, the ground pins of the XBee modules were not connecting to the ground planes of the PCBs. To correct this problem, a via was placed in the ground plane of the bottom layer, as can be observed in **Figure 12.1**. In the final version of the main PCB in **Appendix A**, four vias were placed throughout the board to ensure there would be no further grounding issues. The addition of the vias allowed for the ground plane of the bottom layer to connect with the ground plane of the top layer, reassuring that the ground pins of each component are in fact grounded.



**Figure 12.1:** Raspberry Pi XBEE PCB **Figure12.2:** Switching Relays

## Fabrication

The goal for the full system fabrication was to design a unit to be safe, secure, compact, light, accessibility to all parts, and visually appealing for audiences to be able to follow the procedure of the project. We started implementing our design by hand, drawing them during our meetings, creating them on Microsoft Visio and AutoCAD, and finally building the system. AutoCAD is a software application for 2D and 3D designing and drafting. AutoCAD is useful because you can create your design based on the dimensions needed and then send your design to a distributor for them to build; however, using this software was a struggle in itself since none of us had ever used AutoCAD before. In order to learn AutoCAD, our team reviewed many YouTube videos and had the opportunity to be tutored by a colleague. While designing the case of our system, each component was given a specific dimension, while keeping in mind that the entire structure had to fit in a trunk of a car. The end result is shown in **Appendix E**. and consists of a ten gallon tank, a water pump to circulate water, and an electrical panel to hold the VFD and all of the electrical components.

The ten gallon tank was first chosen to be 10 inches wide; setting the entire width of our system was set to ten inches, the height to thirteen inches, and the length to thirty-five inches. In order for the cage not to over compensate for the dimensions we set, it was decided to use one inch PVC piping. Due to the weight of the tank with water, we had to keep the unit as light as possible, so we chose aluminum extrusion with mounting capabilities as our frame for the unit. The details of the different shapes of the aluminum piping can be found under **Appendix B**. These parts were selected from 80/20 Inc. but were not ordered from them. The best option for fabrication of the unit was single, double, and triple one inch aluminum extrusion. In order to be able to carry the unit easily, four handles were set up in the four top corners of the unit, allowing easy carrying for two people. Three-quarter inch piping was chosen for the connection between the water tank and the water pump on the front side of the tank and one-half inch piping was chosen for the output of the water pump flowing back into the tank. Overall, the water inside of the tank flows into the water pump through the three-quarter inch pipeline and circulates back into into the water tank through the one-half inch pipeline. Finally, the electrical panel is cased using clear acrylic panels in order to protect the electronics from possible water damage. For the electrical grounding protection, a ground bar was installed inside of the panel. The images of the entire shield of our fabrication that was designed through AutoCAD are illustrated under **Appendix C** as well as a full system layout in **Appendix E**.

# Conclusion and Recommendations

Ultimately, the end system result is a feature-complete demonstration model for San Diego Gas and Electric to educate the public on the money and energy savings a variable frequency drive can give. In addition, it provides an overall awareness that such devices exist and can be utilized to cut back on energy and ultimately save money. When testing the device, we found that the VFD provides substantial savings when operating at lower frequencies, as shown in **Figure 13**. The first row is the power consumption using our water pump powered directly from a 120V/60Hz wall outlet, and the rest of the rows are the power consumption using the VFD.

While running the motor through the VFD at 60 Hertz is actually consuming more energy than the wall outlet, the idea is that the motor does not need to be running at its full potential constantly. If we consider a scenario where a user runs a motor using a VFD half of the time at 60Hz and half of the time at 40Hz, the user still cuts down on energy usage by 15% compared to not using a VFD, with about 16% loss in flow rate compared to control group. The savings continue to increase with even lower frequencies used. For example, if we take the above scenario and run a motor continually at 50Hz, the energy savings are over 17% with 16% loss in flow rate. While the VFD is not significantly more efficient (savings are roughly equal to the productivity loss), the freedom to set a frequency based on the current user needs is a more cost-efficient option than always being tied to the motor’s full potential.

Since this was demonstrated on a scaled-down version of a pool pump, one recommendation is to continue testing and gathering data about the cost-effectiveness on a full-scale pump. In addition, further research can be done to see how a VFD can improve other motors in a residential setting, such as washing machines, dryers, and air conditioners. Moreover, with the use of the Central Communication Hub these devices can easily be integrated into our system to create a “smart home” environment.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Variable Frequency Drive Power Measurements** | | | | | |
| **Frequency** | **Voltage Measured** | **Current Measured** | **Flow Rate** | **Power** | **Power Saved** |
| 60Hz line powered from wall (Control Group) | 120V | 230 mA | 3.0GPM | 27.60W | --- |
| 60Hz | 127.35V | 235 mA | 3.0GPM | 29.93W | -8.44% |
| 50Hz | 116V | 196 mA | 2.5GPM | 22.74W | 17.61% |
| 40Hz | 104V | 166 mA | 2.0GPM | 17.26W | 37.46% |
| 30Hz | 89.7V | 149 mA | 1.0GPM | 13.36W | 51.59% |

**Figure 13:** Power use comparison of the VFD versus a standard wall outlet

# References

Arduino Due overview and specs:

http://arduino.cc/en/Main/ArduinoBoardDue#.Uxty4IWwUYA

Arduino Due pin-out:

http://arduino.cc/en/Hacking/PinMappingSAM3X#.Uxt04IWwUYA

Arduino reference libraries:

http://arduino.cc/en/Reference/HomePage

Raspberry Pi touchscreen setup guide:

https://learn.adafruit.com/adafruit-pitft-28-inch-resistive-touchscreen-display-raspberry-pi/software-installation

Raspberry Pi touchscreen reference project:

http://www.davidhunt.ie/lapse-pi-touch-a-touchscreen-timelapse-controller/

Information on Variable Frequency Drives:

http://www.vfds.com/what-is-a-vfd

Printed Circuit Board specifications and standards:

http://blog.optimumdesign.com/clearance-and-creepage-rules-for-pcb-assembly

Information on PCB creepage:

http://www.silabs.com/Support%20Documents/TechnicalDocs/AN583.pdf

Information on PCB trace design

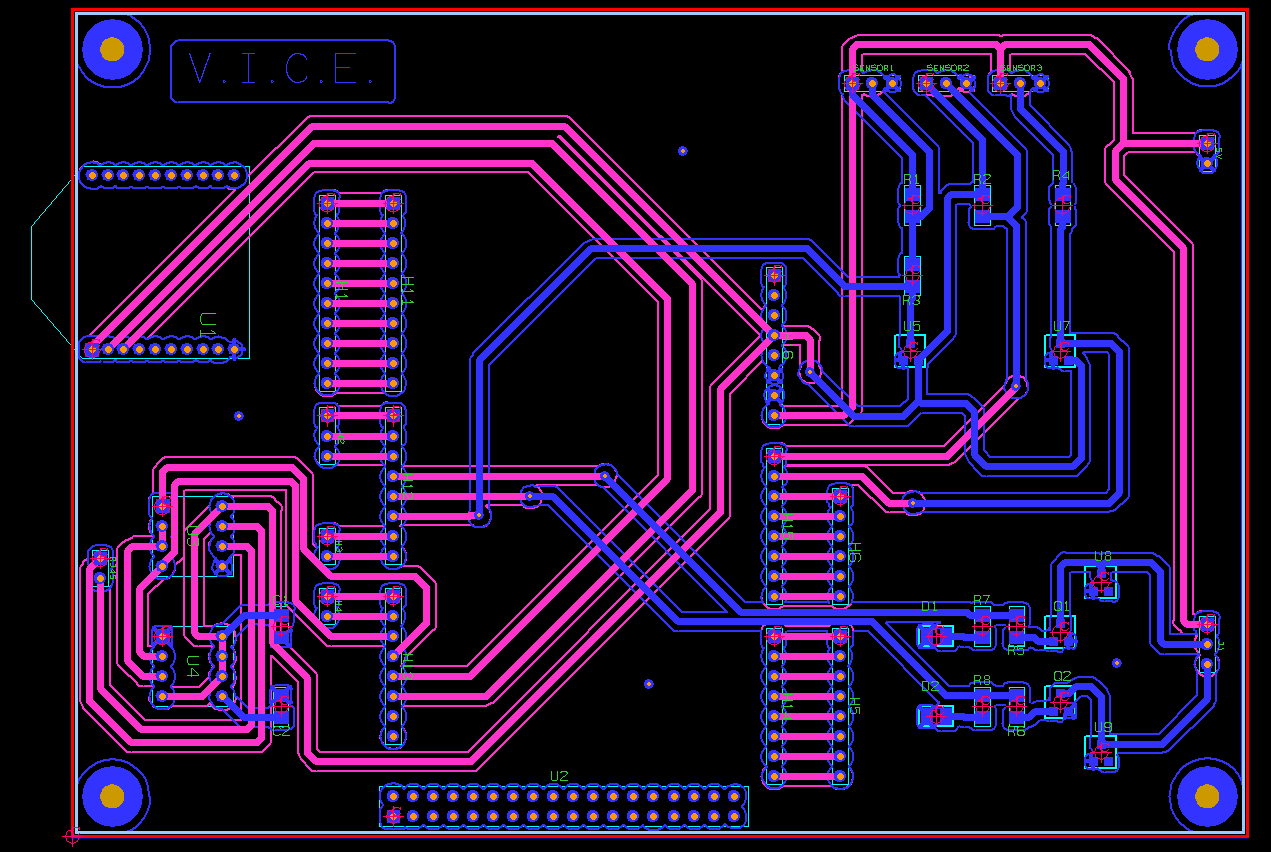
http://www.4pcb.com/trace-width-calculator.html

Information on single-phase motors:

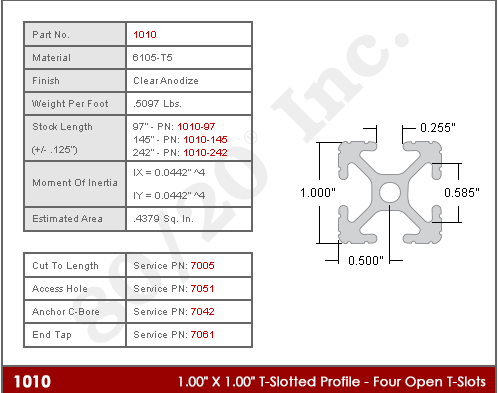
http://www.grundfos.com/content/dam/Global%20Site/Industries%20%26%20solutions/waterutility/pdf/motorbook.pdf

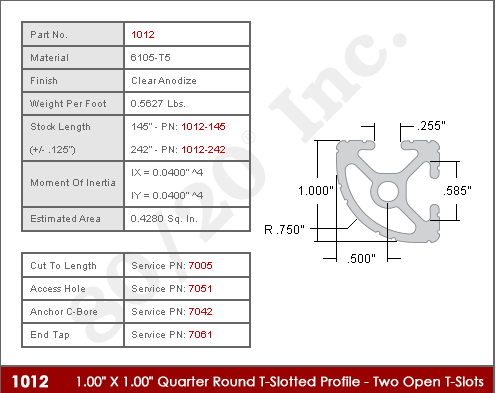
# Appendices

## Appendix A: Main PCB Schematic

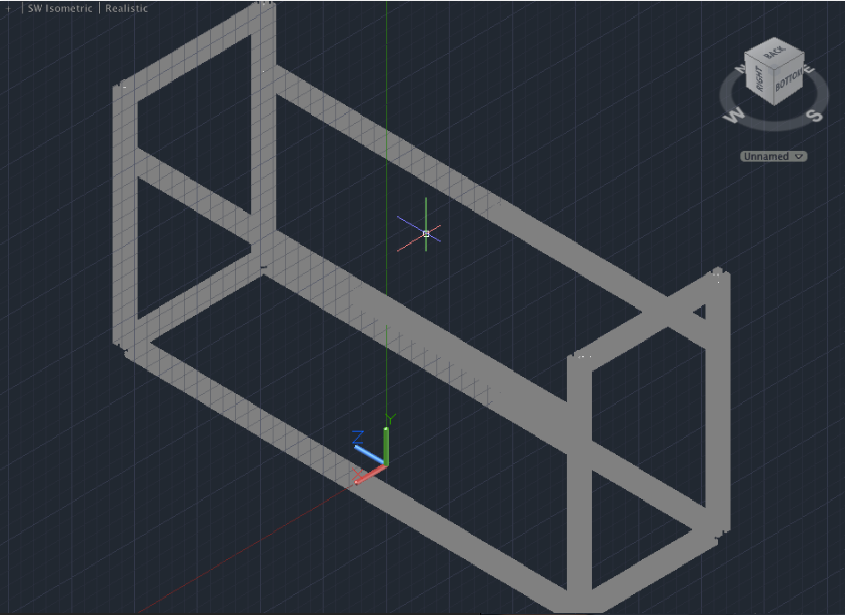
****

## Appendix B: 80/20 Inc. Aluminum Piping

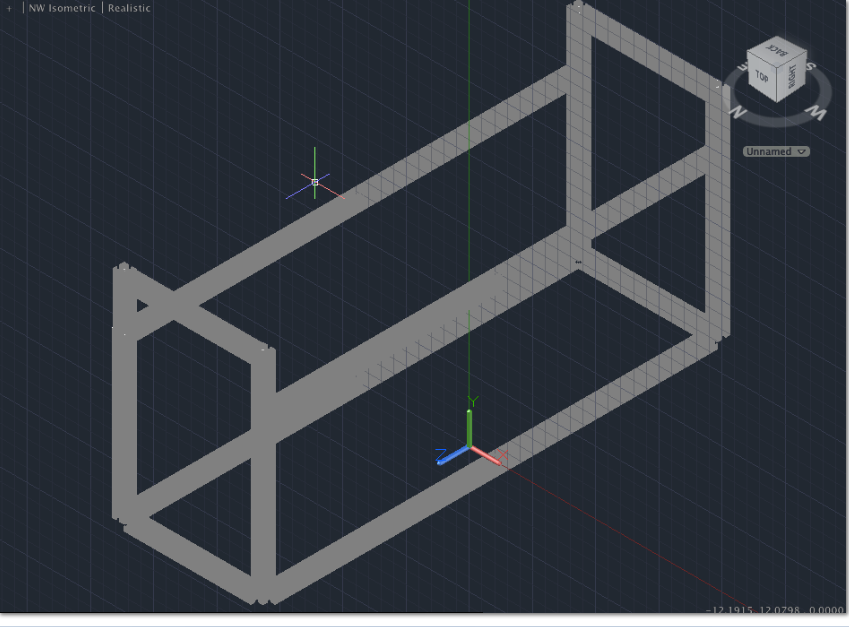




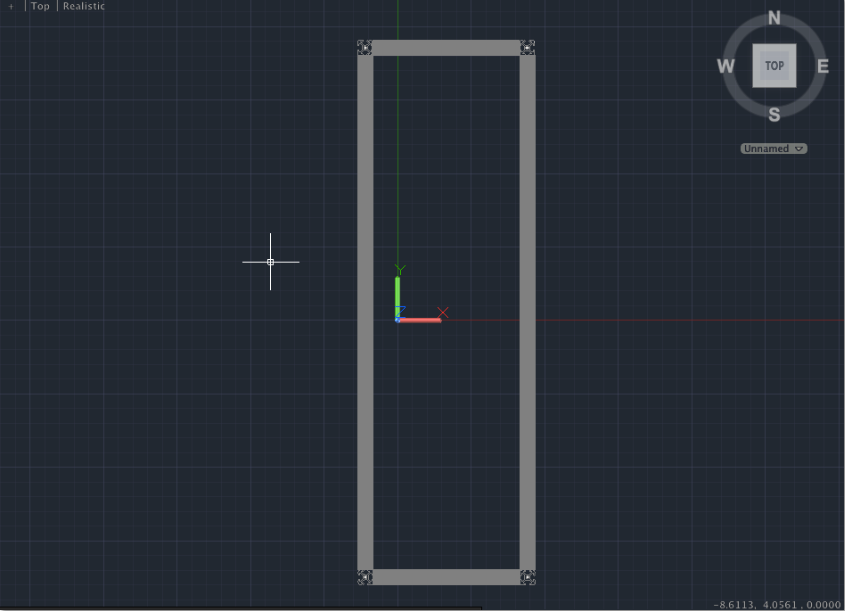
## Appendix C: AutoCad Fabrications



Front Angle View

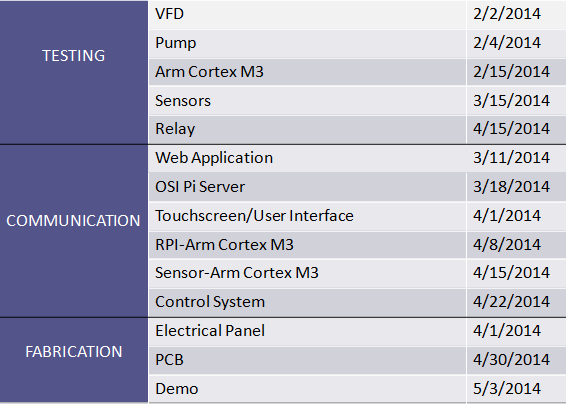


**Back Angle View**



**Top View**

## Appendix D: Final Project Schedule



## Appendix E: Complete System Fabrication

****

## Appendix F: Bill of Materials

