A TEST CASE FOR IMPLEMENTING FEEDBACK CONTROL IN A MICRO HYDRO POWER PLANT

by

AHMAD SULIMAN

B.Sc., Kabul University, Afghanistan, 2003

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2010

Approved by:

Major Professor
Dwight Day

Copyright

AHMAD SULIMAN

2010

Abstract

Micro-hydro turbines generate power for small villages and industries in Afghanistan. They usually produce less than 100 kW of power. Currently the flow into the turbine is controlled manually and the voltage is controlled automatically with an electronic load controller. Excess power not used by the village is dumped into a community water heater. For larger sites that have a reservoir and/or large variable load throughout the day and night, the turbine needs to be fitted with an automatic flow control system to conserve water in the reservoir or deal with the variable loads.

Large turbines usually use hydraulic governors that automatically adjust the flow of water into the turbine. For micro-hydro sized plants this method would be too expensive and be difficult to build and maintain locally. For this reason, a 3 phase AC induction motor will be used to move the internal flow control valve of the turbine. Because a sudden change in load is possible (30 – 40%) for micro-hydro plants, the electronic load controller will also be needed to respond to quick changes in load so that the village voltage does not exceed 220V.

This report documents the process of building a test system comprising of a dynamic resistive load, microcontroller controlled resistive load, a three phase AC generator and a DC Motor. Where the dynamic resistive load represents the load of the village, the computer controlled resistive load would represent the community water heater, the three phase AC generator represents the Generator on site and the DC Motor together with its DC input voltage would emulate the turbine and its water flow respectively. The DC input voltage would be also controlled with a PWM signal through a delay loop to represent the water gate delay effects on the turbine as close as possible. With this, it would be possible to completely build and test a control system that emulates the dynamics of a water turbine generator.

Table of Contents

List of Figures ... v
List of Tables ... vii
Acknowledgements ... viii
Dedication ... ix
Chapter 1 - Introduction ... 1
Chapter 2 - Emulating the actual system with a test system ... 9
  Hardware Setup ... 10
Chapter 3 - Experiments and Tests and Results ... 27
  Experiment 1 – Generating the PWM ... 27
  Experiment 2 – Building the zero crossing detection circuit ... 28
  Experiment 3 – Building the Heater connection controller ... 30
  Test 1 – Reading Frequency under no load condition ... 31
  Test 2 – Running the system under full load of the Heaters ... 33
Chapter 4 - Conclusions and Future Work ... 36
References ... 43
Appendix A - Hardware and Software Codes ... 45
Appendix B - List of Pins for different chips ... 73

List of Figures

Figure 1.1 The block diagram for the hybrid (half automated – half manual) system 2
Figure 1.2 The block diagram of the fully automated system 4
Figure 2.1 The schematic diagram showing the H-Bridge interconnections:.............................. 12
Figure 2.2 MCS08 and H-Bridge circuit picture 13
Figure 2.3 Dynamic Resistive Load Circuit picture 14
Figure 2.4 Observation Unit, Heater Connection Controller and Rectifier Unit mounted on a single board 16
Figure 2.5 Pin connections on the ADC Chip.. 18
Figure 2.6 Extension board with ADC and DAC chips 18
Figure 2.7 Block diagram of the IP Interconnections 19
Figure 2.8 The FPGA Board 23
Figure 2.9 DC Power Supply Unit 24
Figure 2.10 The initial power generating setup 25
Figure 2.11 The generator set currently used for the test purpose 26
Figure 3.1 Microcontroller connection 27
Figure 3.2 Schematic for the zero crossing detection circuit 28
Figure 3.3 Output of Zero Crossing detector in correspondence of AC voltage 29
Figure 3.4 The zoomed in version 29
Figure 3.5 Circuit of the Heater Resistor controller 30
Figure 3.6 Correspondence of “logic low” on digital channel 1 with analog channel 2 31
Figure 3.7 Zoomed in version 31
Figure 3.8 The graph showing the variations of numbers around 2,000,000. 32
Figure 3.9 Frequency variations over time 32
Figure 3.10 Graph of maximum voltage across the Heater Resistor (Blue) in correspondence with the AC input Voltage (Yellow) while 5% of the number was fed back to FMHLC unit 34
Figure 3.11 Graphs of voltage across Heater Resistor (Blue) and input AC Voltage while 40% of the number was fed back to FMHLC unit 35
Figure 4.1 Frequency variations over a period of 3 seconds 36

Figure 4.2 Frequency variations over time – speed control rate: once per sec. 37
Figure 4.3 Frequency variations over time – speed control rate: twice per sec. 38
Figure 4.4 Frequency variations over time – speed control rate: four times per sec. 39
Figure 4.5 shows how the generated frequency is brought to the desired range 40
Figure B.1 Pin details for the L6203 Chip 73
Figure B.2 Pin details for the LTC1867 Chip 73
Figure B.3 Pin details for MCS08 Chip 74
Figure B.4 Pin details for PS2501 – 4 Chip 74
Figure B.5 Pin details for MOC3031 Chip 74

List of Tables

Table 2-1 Description of MCS08 pins used 10
Table 2-2 Combination of pins 9 and 10 and their effects on pin 7 11
Table 2-3 Pin description for the L6203 H-Bridge 12
Table 2-4 List of Inputs and Outputs on the FPGA board 22
2-5 List of output voltages of the DC Power Supply 23

Acknowledgements

All thanks and praise is due for Allah (the God) who makes all things possible.

This report is a result of the continuous support of my major advisor Dr. Dwight Day. I would like to convey him my heartily felt thanks and appreciations. In addition, I am sincerely thankful to the committee members Dr. Andrew Rys and Dr. Anil Pahwa whose generous advices and comments helped me get through the program. Also, I would like to thank all the department members and staff for facilitating such a great educational environment. I am also thankful to the Afghan Government and specially the Ministry of Higher Education, Kabul University and Engineering Faculty for funding this education and trusting me.

Special thanks to my family, my wife and my friends who never let me feel alone and have been always supportive.

I am also acknowledging the friendly and honest cooperation of the Remote HydroLight Company. They have been always helpful in regard to answering my questions and queries at any time.

Dedication

To those who are tirelessly working to encourage use of clean energy resources, and fighting against global warming.

Chapter 1 - Introduction

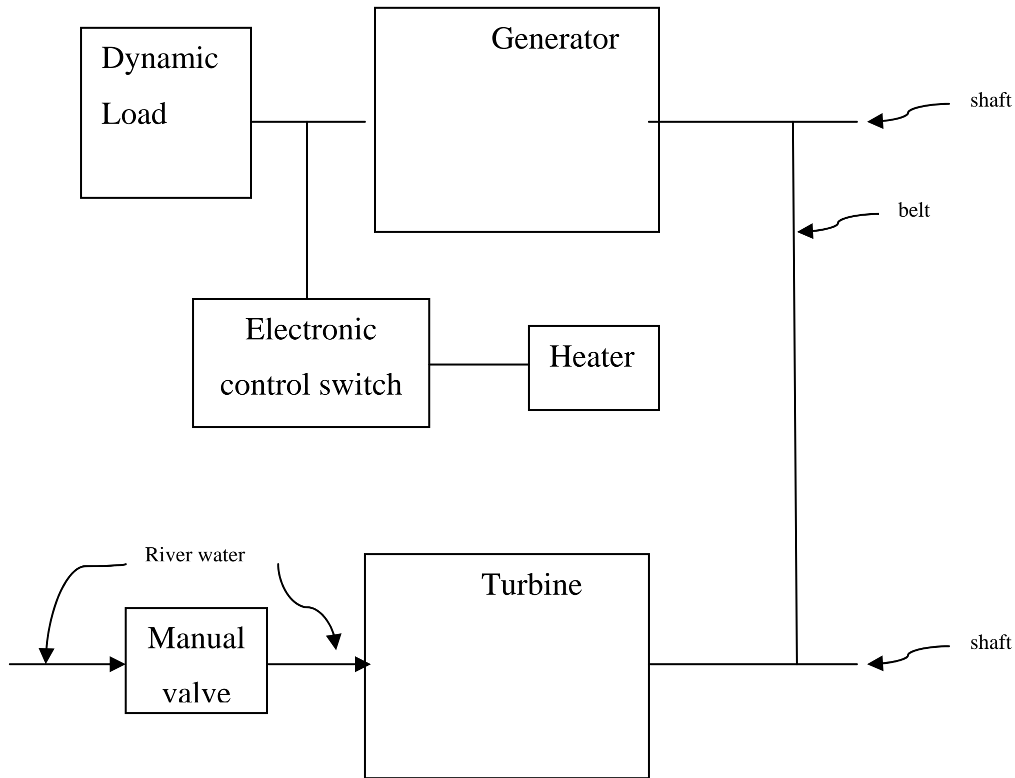
Afghanistan is a mountainous country. This fact has caused this country to be rich in permanent and flash water flows which could be good resources of energy for hydro power plants. Depending on the geography of the land, amount of water, and many other factors, there are a large variety of options in selecting different sizes of hydro power plants. Although, there have been few number of large hydro power plants built during 70s, currently there are lots of ongoing micro hydro power plant projects.

Generating power by micro hydro power plants is cheap, efficient and easily accessible way of generating power especially in rural areas where relatively large amounts of water and good heads are available, and providing grid power is somewhat challenging. There have been many efforts done by different organizations in this rally and one of them as an example could be the Remote HydroLight Company.

Generally, most of the micro hydro power plants are built locally and in a very manual and classic fashion, but a distinctive approach taken by the mentioned company is that it introduces feedback control system concepts and an automated fashion of controlling speed to keep the terminal voltage and frequency stable. As mentioned in the abstract of this report, the plant is setup in a way that it produces extra power than what would be sufficient for the village in order to keep the terminal voltage in a stable value of 220V with a frequency of 50Hz. When the village load falls down below its 70% of the peak value, the terminal voltage starts increasing and this is when they have to connect the water heater to gradually take over the place of reduced load and keep the terminal voltage stable. This approach still keeps the turbine running at the same speed as it was running while it was under full village load and water flow will be kept unchanged. The flow will be kept constant because the village load does not get settled at a constant value, so the generated power will be distributed between village load and heaters. How much current should flow to which load is decided based on the amount of village load. The lower the village load would get the higher the current would flow to the heating load. This mechanism will continue until the village load drops below 50% of its peak value and settles at that range. At this point of time the flow will be reduced manually and the heating load will also decrease but automatically. The flow is not always touched because the speed control of turbine

with flow has much longer response time compared to rapid changes in the village load. Rather, the flow has some specific values to which it is set during a period of 24 hour and always greater than what is needed by the village and the excess power is always routed automatically to the heaters. Figure 1.1 shows a diagram of the above mentioned system.

Figure 1.1 The block diagram for the hybrid (half automated – half manual) system



The above mentioned process works well as long as the plant is over (or on) a running stream where saving water wouldn't be a concern and maximum generated power is below 40KW. It gets more important to take care of turbine speed and its flow rate when the turbine is serving to generate more than 50KW of electric power. This would also mean that we have to have some kind of reservoir where we store water and use it to turn the turbine which in turn raises the concern of using water efficiently. Note has to be also taken that we still have a dynamic village load which varies in a specific range throughout 24 hours.

Knowing that flow control with a higher frequency to meet the dynamic load requirements would be impossible unless we use complicated and expensive hydraulic systems, we should come up with some other solutions. This fact is also supported in [3] and [4]. By other solutions we mean that we should be able to take care of dynamics of the fast changing village load. Furthermore, we should be controlling the flow if not as fast as in case of complicated flow control mechanism but at least much better and faster that the manual approach. One of the solutions would be to keep those heater loads in place and change the manual flow control to an automatic control. This method would emphasis more on flow control though. The heaters would be there only to take care of fast dynamic changes until the water flow is adjusted by the controller.

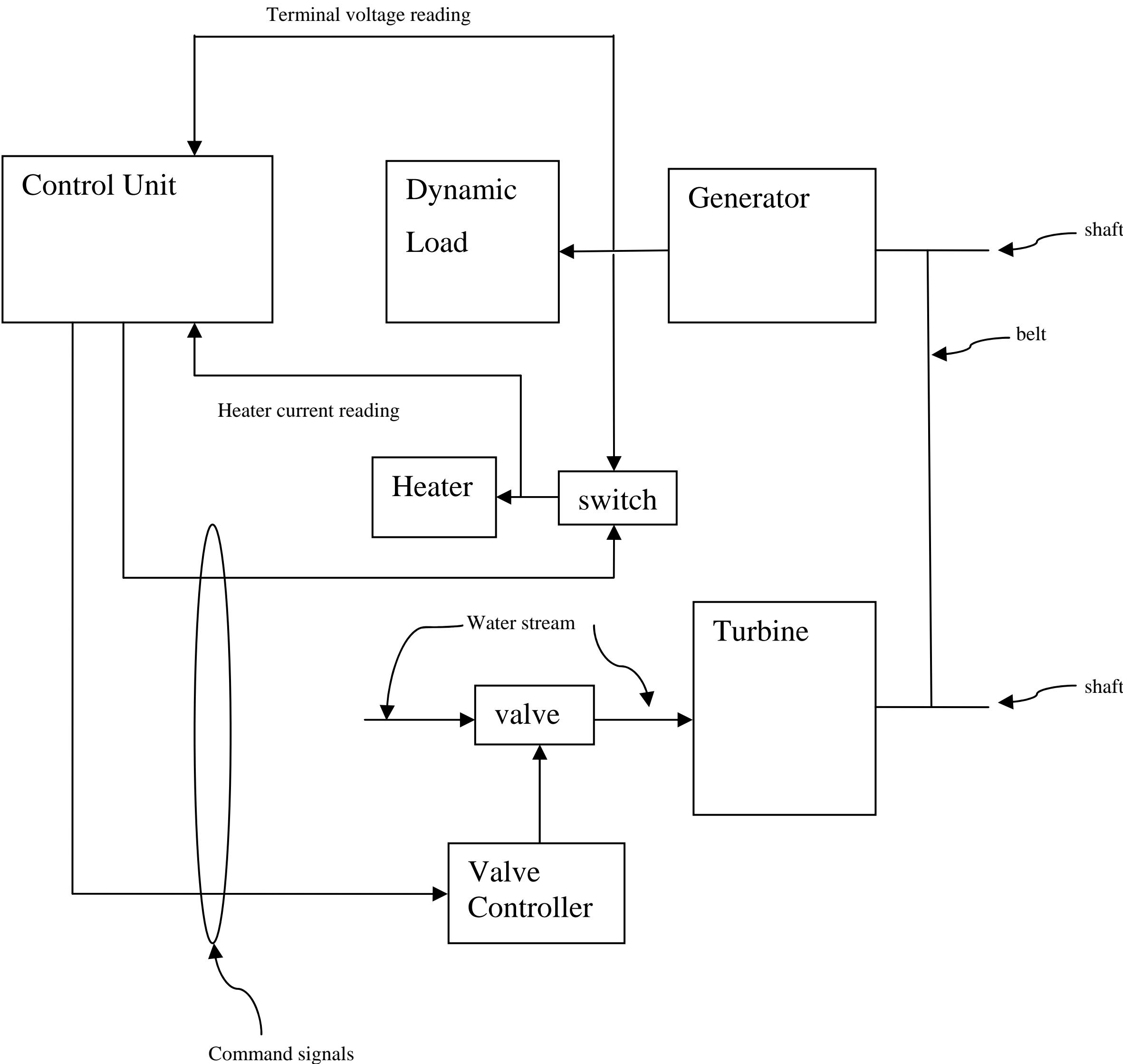
Procedure:

We will be constantly reading the terminal voltage and frequency of the generator with a microcontroller. At the same time we will also be observing the current running to the heaters. If we are reading a higher current to the heaters we will reduce the water flow simultaneously with the current flow to the heaters. Of course, it should be taken into account that water flow reduction affects slowly the overall power output. Therefore, we should have a delay in decreasing the current flow to the heaters. There has to be a threshold value set for the current flowing to the heater where we have to always keep the current above that threshold value. If the village load demand increases suddenly which would cause the heater current flow to go below the preset threshold, the controller should command the water gate driver to start opening the gate until the current flowing to the heater increases to the threshold. We would assume that at the very moment while the gates are getting opened, the power which was consumed by the heaters should suffice the village demand for as short period as it takes the water to be regulated. So we have to be quite certain about the value we choose for the threshold so that at one hand it should be sufficient enough to help us have enough power to cover those instant increases in the village load and on the other hand it should be as low as possible so that it should make sense when we claim that we are saving water.

As mentioned earlier, the generator terminal voltage and frequency should be monitored constantly in order to insure that they are always in their desired range. This way the very transient changes in the village's demand, like current drawn by a welding machine, would be

reflected as the variations of the current flowing to the heater, without requiring an abrupt change to the water flow through the turbine. Once it is observed that the village load demand has remained consistent for a certain amount of time, then the water flow could be readjusted and the current to the heaters will also be set accordingly. Figure 1.2 illustrates further this process.

Figure 1.2 The block diagram of the fully automated system



The whole process as written so far is one way to have a relatively stable micro hydro power plant while saving water as well as increasing the lifetime of the system. Although the water flow rate control is not an instantaneous mechanism to save maximum amount of water, it would give much better result than manual control of flow because this mechanism is controlling the flow every couple of seconds while a manual control could only change the flow in periods of two to three hours depending on the location of site. Furthermore, the flow rate control is obtained by just building a very simple setup which could be comprised of a 3-phase synchronous motor that would open or close a valve directly controlling the water flow through the turbine. The latterly mentioned fact also supports the idea to have a cheap and locally maintainable system.

It is worth to mention that a lot of research has been done on controlling frequency using Dump Resistive load. In fact, this is an economical and convenient approach to control the frequency when dealing with micro hydro power plants [1, 2, 3]. However, the literature reviewed, doesn't address any delayed speed control issues while using rapid means of frequency control. Reference [1] describes how a "Load Governor" was used for control of the generated frequency. Basically, the "Load Governor" increases or decreases the Dump Resistive Load in steps based on the generated frequency readings. The paper further elaborates the use and practicality of Dump Resistive Load with both Synchronous and Induction Generators. At the same time, it highlights the advantages of Induction over Synchronous Generators. In addition, the author confirms the results through simulations, assuming a constant shaft speed.

Tamrakar et. la. in reference [2] studies the terminal voltage and frequency control using a "Static Compensator (STATCOM) together with an Induction Generator connected on the Synchronous Generator Bus. The "STATCOM" takes care of power connection to the Dump Resistive Load using a chopper, whereas the Induction Generator together with the "STATCOM" takes care of the reactive power of the system. Finally, the author validates the simulation results through some experiments. Once again, this paper emphasizes more on use of the Dump Resistive Load and how an Induction Generator helps on the reactive power part. An important result of this research was that it is easier and cheaper to connect an Induction Generator in parallel with a Synchronous Generator than to connect two Synchronous Generators.

Profumo et. la. in reference [3] deals again with frequency control but applied while paralleling with network power and using an Induction Generator. After the system is connected to the network, the frequency controller is removed. This scheme is using the speed of the turbine shaft as the feedback signal to the frequency controller rather than the generated frequency. The generated frequency is only used as reference.

Bhim Singh and his coauthor in their write up [4] describe use of "IELC (Improved Electronic Load Controller)" for self excited induction generators in micro hydro power plants. They include the issue of load balance in three phase generating plants together with frequency control. This paper deals more on modeling and analysis aspect of the design. Although the source covers a bit wider area than what this report covers, it gives an idea of better use of power electronics.

In other research [5] a chopper is again used to connect and disconnect the Dump Resistive Load to and from the generator. However, the chopper this time is controlled by a PWM generated by the controller. This paper further includes a voltage regulator as well. The proposed control scheme is studied for both Synchronous and Induction Generators. The similarity of this control scheme to that employed here is the way the chopper is being controlled. Of course, the control signal is still different in my case and it is not a PWM signal. Nothing much different is there in the literature [6]. It merely connects the excess power to a heater through a PWM controlled switch and studies the behavior through some simulations. Moreover, it doesn't explain the control scheme of the PWM and how the PWM is being generated. Likewise, reference [7] studies the VAR Compensation in a micro hydro turbine plant using a Self Excited Induction Generator ("SEIG"). The control scheme and use of solid state switches to compensate for VAR generated by the "SEIG" is another approach. However, this approach is quite different from the one studied in this report in the sense that in this method the "SEIG" terminal voltage is used as the feedback signal and desired terminal voltage is used as the reference voltage and the controller only minimizes the error between those two. Whereas this report uses the terminal frequency as the feedback signal and the desired terminal frequency is the reference. Furthermore, it doesn't talk much about the frequency control or if any assumptions being made in regard to the frequency.

Profumo et. la. in reference [8] reports the research and simulation of the voltage and frequency control in a similar scheme, using "IGBT (Insulated Gate Bipolar Transistor) based

voltage source converter (VSC) and a DC chopper with an auxiliary load at the DC bus of the VSC." So the scenario is still to control the frequency and terminal voltage by use of a Dump Resistive Load connected through a chopper. The reference further analyses the use of two or more Induction Generators connected in parallel. The results recorded in this reference show that it is possible to control two or more Induction Generators connected in parallel with a single voltage and frequency controller.

Generally speaking, the only focus has been always to control the terminal frequency using solid state devices while assuming a constant mechanical power input due to the fact that controlling the mechanical power would be expensive. In addition, most of the literature studied the issue of frequency control with Induction Generators. While agreeing with the advantages of Induction over Synchronous Generators, this report uses a Synchronous Generator for the tests and experiments. The first and very important reason is that the sites where they would use the solutions documented in this report are using Synchronous Generators. Besides, due to the fact that Induction Generators are consuming reactive power [2, 3, 6, 8], additional capacitive circuits are needed which will increase complexity of the system and has to be avoided. In the same manner, Induction Generators only generate power when they are running in a speed slightly higher than synchronous speed [9] which requires a constant flow rate. Latterly mentioned characteristic of Induction Generators may cause complete power shut down in the cases where the controller starts controlling the water gate and consequently causing temporary speed variations which sometimes could go below synchronous speed. Whereas in case of using a Synchronous Generator only transient frequency fluctuations would be observed in the network not a total shutdown. Furthermore, in the case of Afghanistan, since the Synchronous Generators are readily and excessively available in the market, there is no need to make further investment in importing Induction Generators.

This section has tried to give an overall picture of the whole plant and how it should work. In addition, this chapter includes a review of work of some other researchers in the area of micro hydro power plants. The next two chapters will elaborate further about the steps taken to establish a work bench to test and model the whole system and develop an algorithm applicable to an actual site. Chapter two will describe the hardware used to establish the test system based on the requirements of an actual site. Chapter three will include some experimental steps of building and programming some hardware units, and test scenarios together with their results.

Finally, Chapter four will cover the conclusion remarks and future work will be discussed as well. The appendices will mostly contain the hardware and software details including the codes used to implement the developed algorithm.

Chapter 2 - Emulating the actual system with a test system

The system we are going to build for micro hydro turbines has to be modeled first. This chapter is explaining how we would approach to build a system to depict the actual system as close as possible taking most of the dynamics of the actual system into account.

There are lots of pieces that do not match exactly to the site situation. For example, the microcontroller which is used here and the one used on site, the prime mover of the generator being a motor instead of a turbine and so on. Not to mention the small pieces used as the observing instruments. Nonetheless, the overall system would give us feedback on how our algorithm, and to some extent software, would work in an actual site. Moreover, as mentioned earlier, it has been the effort to build the system as similar to the actual site as possible. For instance, the pulse width modulation controller for the prime mover has been completely separated from the main microcontroller. The main microcontroller is just giving signals to the pulse width modulation controller to increase or decrease the duty cycle resulting in faster and slower motor speeds respectively. This approach helps increase or decrease the motor speed as slow as the water gate affects the turbine. The analogy would be to think of pulse width modulation controller as the motor controlling the water gate based on the command received from the AC machine driver. And the duty cycle would perform as water flowing to the turbine.

Similarly, there are other points that are taken care somehow and to some extent. However, there are also some unavoidable problems with the test system that can give more feel of a real system. For example, the way the motor and generator is coupled and its affect on the frequency is more likely to be the same as it would be on site.

I will go into details of each piece of hardware used in the system and would further detail about the purpose of each piece used in the system. It is also important to mention that the term “Heater Resistors”, from now on, is used for those loading resistors in the test system that are representing the water heater load on an actual site.

Hardware Setup

The hardware setup is comprised of many different pieces of electronics working together to build the overall test system. Now let's go into details of each piece one by one.

MCSO8 Microcontroller (PWM for the Prime Mover):

This microcontroller is merely responsible for generating the pulse width modulation (PWM) signal for the prime mover i.e. the motor turning the generator. In addition, this microcontroller controls the duty cycle of the generated pulse width modulation signal based on the inputs that it gets from the Xilinx Microcontroller i.e. the main controller.

The purpose of having this control unit separate from the main controller is to emulate the delay between the command sent to the AC motor driver and its effect on the turbine speed. In this case as the main controller commands the PWM controller to speed up or slow down the motor, the controller starts to increase or decrease the duty cycle after a delay similar to that of the water affecting the real turbine. This way the motor will speed up or slow down gradually in the same manner as the turbine would be affected, gradually increasing or decreasing the amount of water flowing through it.

Below is the table listing the input and output signals to/from the chip.

Table 2-1 Description of MCS08 pins used

Pin #	Function	Purpose
3	Vcc (3.3v)	Power supply
4	Ground	Grounding the chip
7	PWM Output	Provides PWM signal for the H- Bridge
9	Port B Input pin 3	Receives signal from microprocessor.
10	Port B Input pin 2	Receives Signal from microprocessor.

Table 2-2 shows the relation between PWM duty cycle and combination of pins 9 and 10

Table 2-2 Combination of pins 9 and 10 and their effects on pin 7

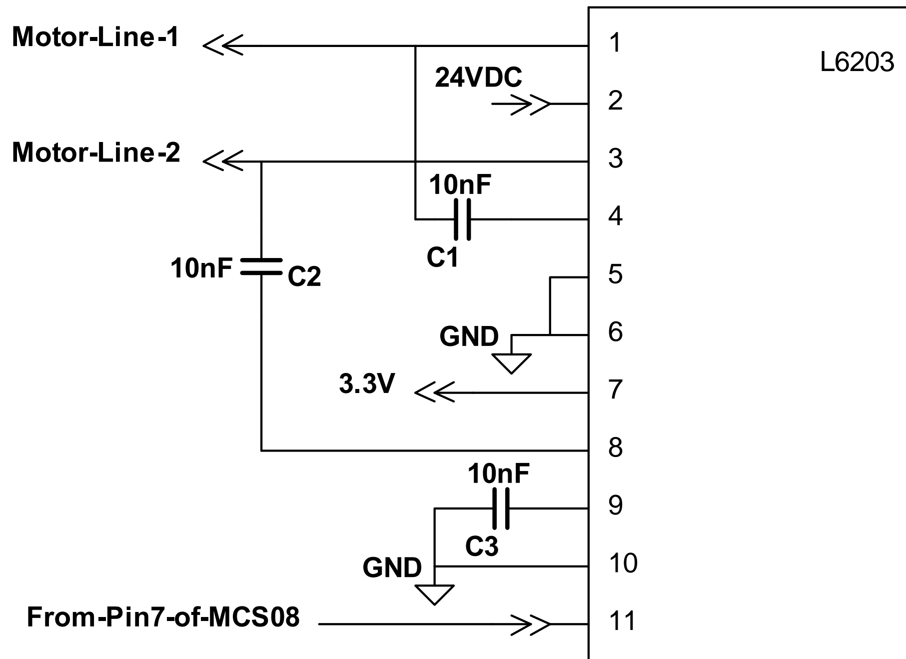
Pin 9	Pin 10	Duty Cycle
0	0	No change
0	1	Decreased
1	0	Increased
1	1	Drops down to almost 0

Prime Mover (Motor) driver circuit:

This is obvious that the PWM signal coming out of the microcontroller is in no way capable of driving the motor. Therefore, a power electronic device such as an H-Bridge should be deployed. The H-Bridges are commonly used where bidirectional motion of the motor would be needed. Although there is no need for bidirectional motion, the reason for using an H-Bridge is just because it is readily available and it works well in a range of 12 to 48 volts which gives me a better flexibility in choosing a power source.

All a motor driver does is to connect or disconnect the power supply to the motor. The connecting and disconnecting decision is made by an input signal to their Enable Pin. The mentioned input signal is nothing but the PWM signal coming from the microcontroller. How long the power should be connected to the motor is determined by the duty cycle of the PWM signal and that is what the MCS08 microcontroller is increasing and decreasing. The higher the duty cycle is the longer the power is connected to the motor which translates to higher speed. The H-Bridge has a pair of additional inputs too. Those inputs are used to set the direction of motion. In our case since we needed only one direction, we didn't need to feed them from any automated source. It was simply hard wiring one to 3.3v (logic one) and the other to ground (logic zero). Figure 2.1 shows the interconnections.

Figure 2.1 The schematic diagram showing the H-Bridge interconnections:



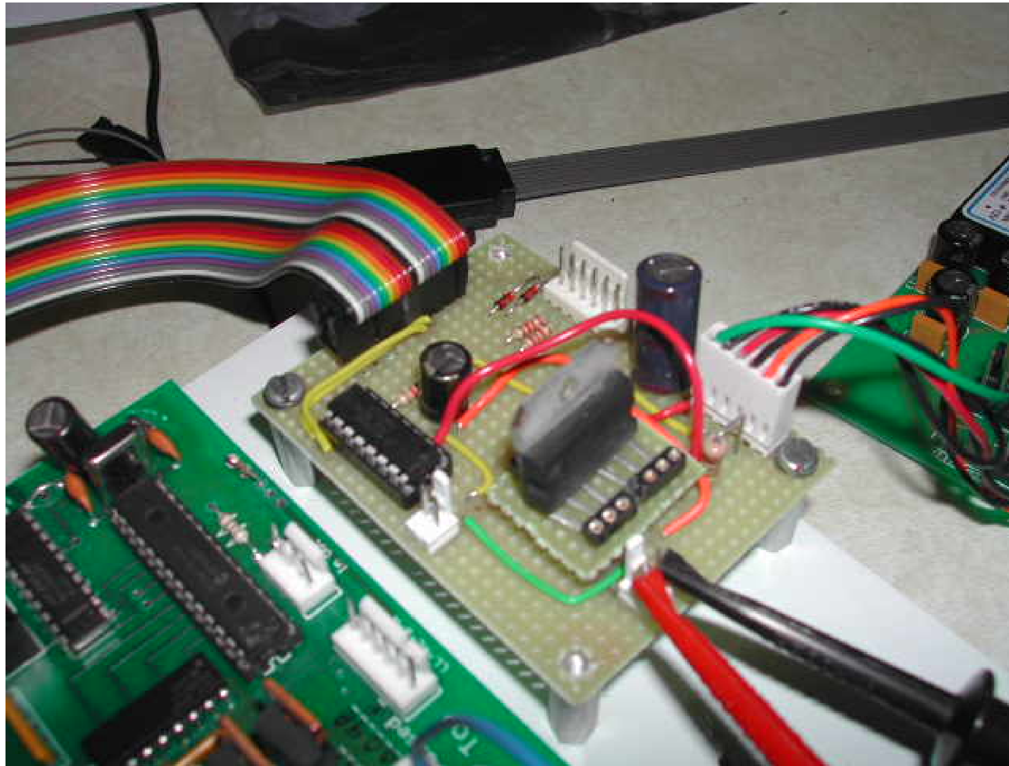
The chip with its pin names is included in Appendix B. The table below lists the input and output pins of the H-Bridge used in this project:

Table 2-3 Pin description for the L6203 H-Bridge

Pin #	Description	Connection
1	Out 2	connected to one wire of the motor
2	Vs	Takes in the power supply for the motor
3	Out 1	connected to the other wire of the motor
4	Boot 1	connected to Out 1 through a capacitor
5	IN 1	Used to set the direction (Grounded in our case)
6	GND	Ground Pin connected to ground
7	IN 2	Sets the direction (Connected to 3.3v in our case)
8	Boot 2	connected to Out 2 through a capacitor
9	Vref	Grounded through a capacitor
10	SENSE	Grounded
11	ENABLE	Connected to pin 7 of MCS08 through a 1k resistor

The MCS08 and the H-Bridge are mounted on a single board as shown in the figure below:

Figure 2.2 MCS08 and H-Bridge circuit picture



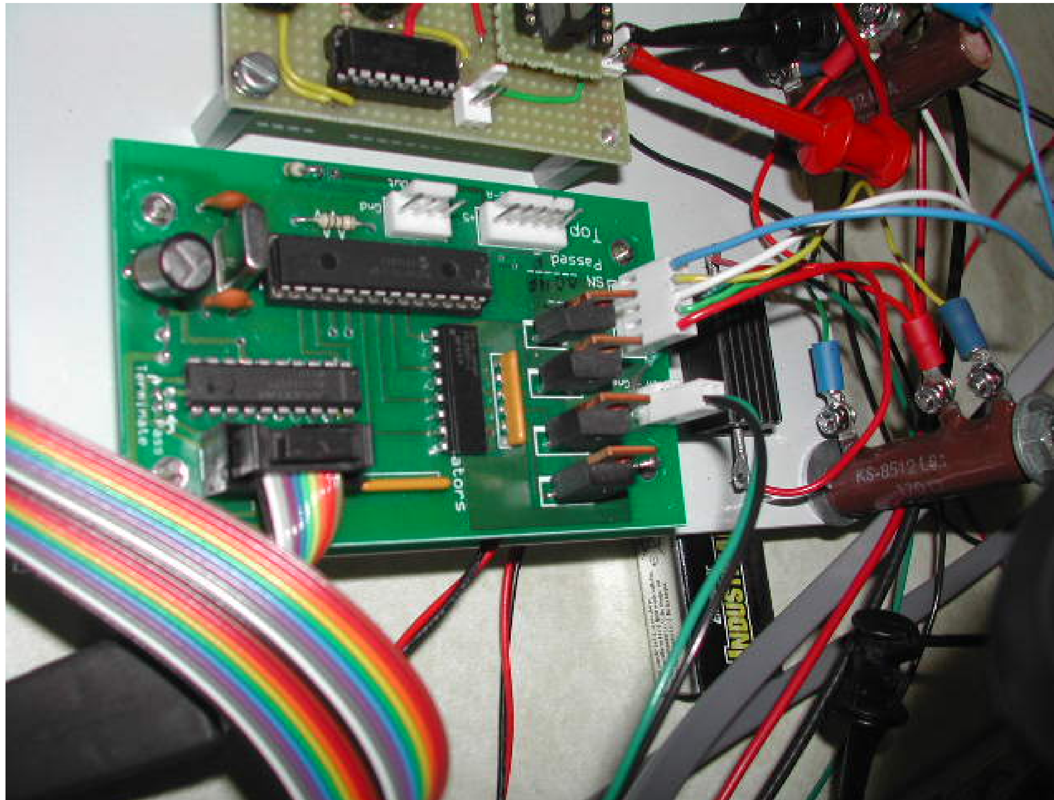
The dynamic resistive load:

This part represents our village load. Since in a village the main consuming elements are mostly the electronics, some heaters and lighting, it wouldn't be the case to worry much about the reactive part of the village load. Thus, a set of high power resistors could be a good approximation to the village load. Should there be any real issues with reactive power consumption on a site, it would be handled on that spot and would be very site specific.

The resistive load has to be connected to the generator such that it should really act as a randomly changing load as a village load would behave. We would achieve that by using an additional electronic circuit. This circuit is connecting different number of resistors based on some random numbers that it receives from a PC through its serial port. The bigger number it would receive the more resistors it would connect to the generator. This circuit is completely isolated from all the controlling unit and it is even powered independently. The reason behind that is to have the load connected to the generator completely independent of all system

dynamics. Thus, we will have a load which would randomly vary based on the random numbers sent to its controlling circuit. At this point we have the dynamic load that acts as the village load and will have its random effects on the generator that has to be handled by the control system that is being built. Below we can see a picture of the board together with its load resisters.

Figure 2.3 Dynamic Resistive Load Circuit picture



Observation Unit (zero crossing detectors):

The observation unit is a small circuit that detects the zero crossing points of the generated AC Power and reports it to the microcontroller. The microcontroller (FPGA the Xilinx microprocessor) will use the provided information to calculate the frequency of the generated power. Once the Xilinx microprocessor knows the frequency it can easily generate different commands to different subsystems to keep the frequency and consequently the terminal voltage stable.

Coming back to the observation unit, it is just the PS2501 chip that has a set of 4 pairs of LED and photo transistors built in it. Since we have only three phases, only 3 sets of them were

needed. The LED's are powered with the generated voltage through current limiting resistors of say 1 kilo ohms. Then each photo transistor is mounted on the chip fronting an LED which gets enabled by sensing a light from it. Since the collector pin of the transistor is already connected to the 3.3 (logic 1), it immediately allows current to flow when the LED is lit. The emitter pin is grounded (connected to logic 0) through a 100 ohms resister. So if the transistor is active the emitter pin will read a 1 and other wise 0.

In summary, this unit outputs logic High in the positive half cycle and logic Low in the negative half cycle. The delay caused by the electronics involved has some effects on the period and frequency that would be calculated by the processor. However, once the standard 50Hz is translated to a certain number of clock cycles, then we can decide if a count value shows a higher or lower frequency than 50 Hz. On the other hand, the frequency detector IP in the FPGA board is designed such that it takes the mentioned delays to a possible extend into account.

For more information about PS2501 please refer to its Data Sheet listed in the references list.

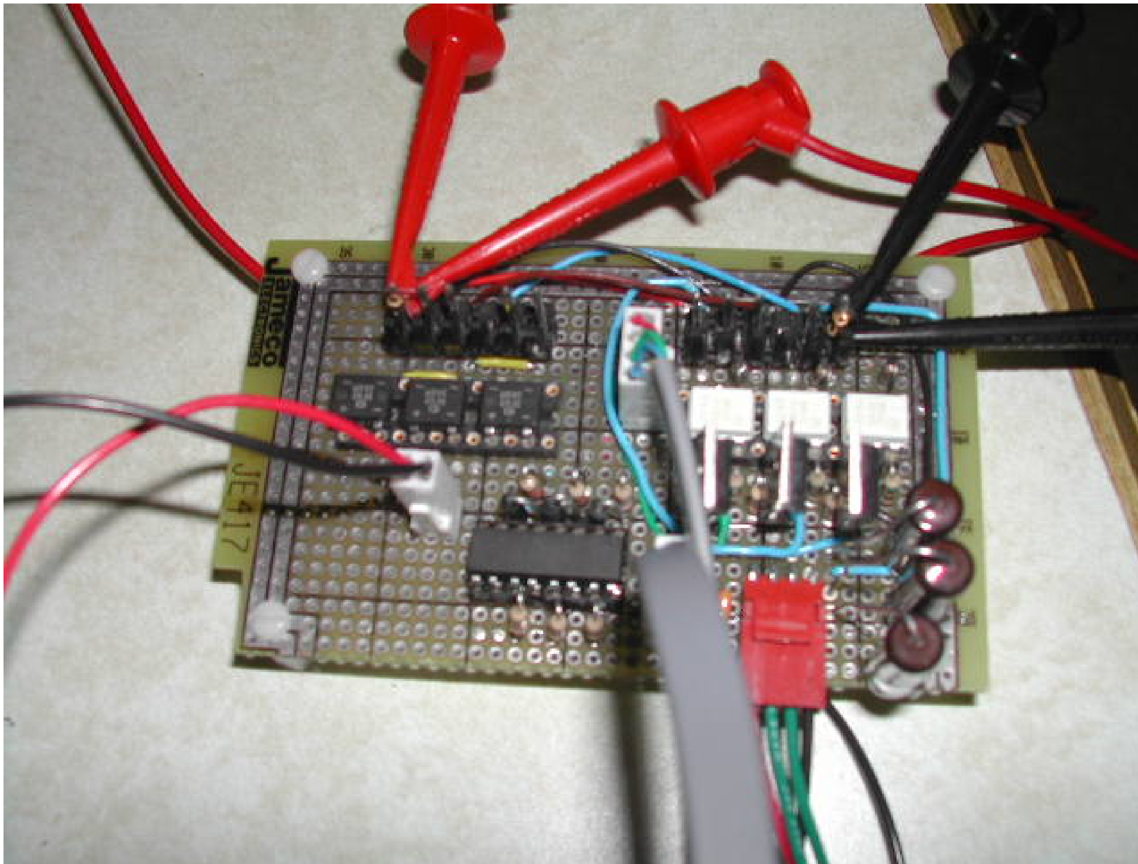
Heater Connection Controller:

This circuit consists of a set of Optic-Isolators and Triacs. The Optic-Isolators are of the same structure of PS2501 chip but in a different chip designed for Power handling issues. In addition, they use photo triacs for switching purpose instead of photo transistor. They are working the opposite way, meaning that the IR LED's on the chip would be fed from the microprocessor instead of being connected to the generator output. The anode pin of its built-in LED will be connected to the logic High (hard wired) and we connect the command signals from the microprocessor to the cathode pin. The signals should be in Logic Low in order for the LED to emit light, thus, it is an Active Low device. The photo triac allows current throw it once it detects the light from the LED. The allowed current through the photo triac would be used as the signal for the gate pin of the external Triac used to connect the power to the Heater Resistors. Therefore, the Heater Resistors would be powered up only if the optic-isolator receives a Low signal from the microprocessor. The chip number for the optic-isolator that is used in this circuit is MOC3031 to check its data sheet. The schematic drawn in chapter 3 figure 3.2 would give a better feel of how it is interconnected.

Rectifier Unit:

The system we are building is based on the assumption that we have a balanced load on the generator. As a result the issue of balancing load is not discussed in this report and neither our control algorithm will handle the unbalanced loads. On the other hand, this assumption simplifies the dynamic load circuitry and makes it easy to automatically connect different loads on the generator. To keep the load on all three phases of the generator balanced in order to comply with our assumption, we simply connect the generator terminal to a rectifier unit and we use the DC output of the rectifier to be connected to the dynamic resistive load. Below is a picture of the board which contains all three units mentioned above:

Figure 2.4 Observation Unit, Heater Connection Controller and Rectifier Unit mounted on a single board



ADC Unit:

As mentioned earlier, the idea of only connecting the heaters is good to keep the frequency stable. However, the goal is not only to keep the frequency stable but also to save water. While keeping the frequency stable, we are causing different amount of current to flow to the heaters. If we can monitor this current, we can make judgments on whether we need to decrease the amount of water flowing through the turbine or not. To monitor the current flowing through the Heater Resistors, an ADC is needed to read the voltage across a resister that would be connected in series with the Heater Resistors. Then that voltage can be easily translated to current using the Ohm's law in the software. Based on the amount of current and how long it remains in a certain range, the microprocessor will command the MCS08 microcontroller if it should increase or decrease the speed of the motor. This in turn means that we can increase or decrease the amount of water in the real site. After a while when the turbine gets slower and consequently the frequency drops down a desired value, we can simply reduce the amount of current flowing to the Heater Resistors.

The ADC used in this project is a chip from Linear Technology with the part number LTC1867. This chip provides 8 channels of analog input, a serial digital output which transmits the digital conversion result serially, a serial digital input, a clock, a conversion/serial framing signal. The conversion/serial framing pin is a dual function. When it is logic low, it lets the chip to serially transmit the converted data available from a previous conversion. On the other hand, when it is logic high, it lets the chip to start a new conversion. For further information the data sheet could be consulted. Although the ADC chip has got 8 analog channels, this ADC chip is primarily used for measuring current flowing to the Heating Resistors. However, as the need may arise, the other channels may be used as well. The board on which this chip is mounted includes a DAC as well and both are wired neatly to a header that can be easily connected to the FPGA Board. The down side is that although there is no need for the ADC, some pins will remain unused because they are connected to it. In addition, a signal through GPIO IP of the Microprocessor needs to go to the ADC to keep it always inactive.

The diagram in Figure 2.5 shows input and output pins of the ADC chip and Figure 2.6 is a picture of the board on which the ADC and DAC chips are wired. For names of all the pins of the chip refer to Appendix B.

Figure 2.5 Pin connections on the ADC Chip

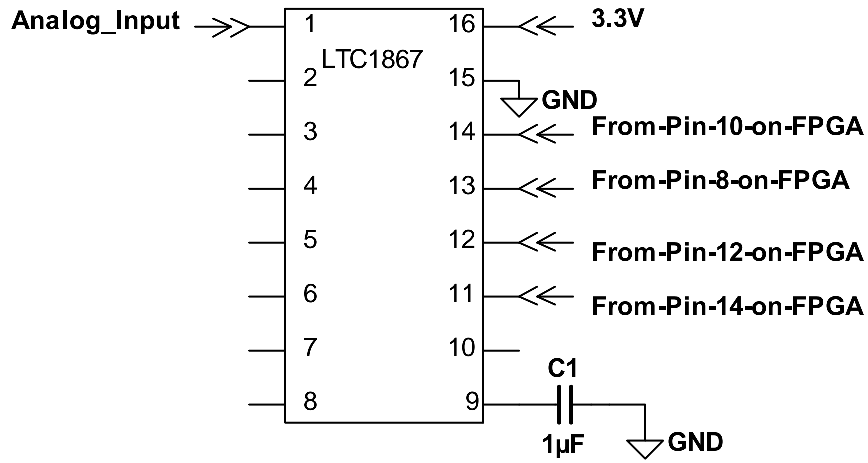
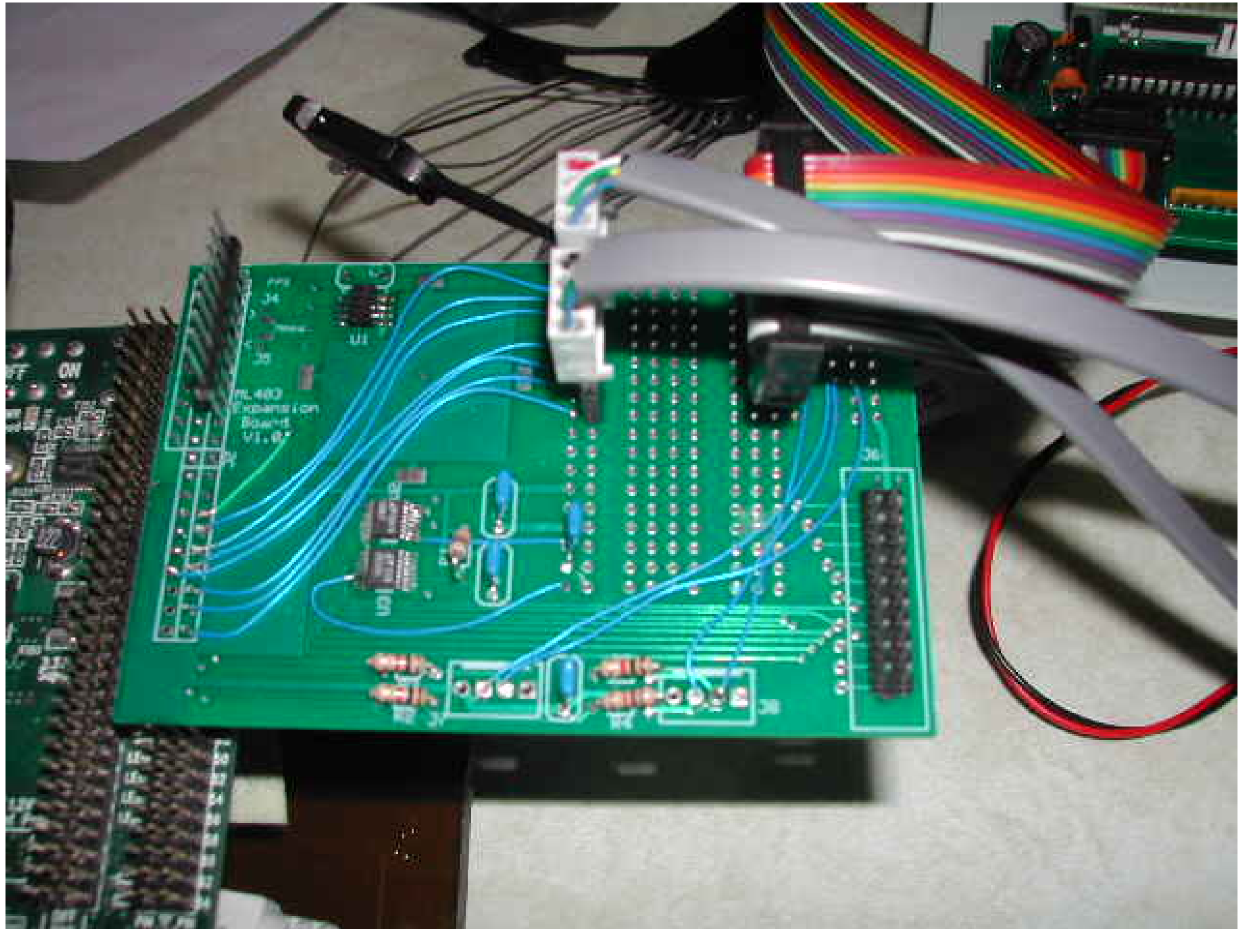


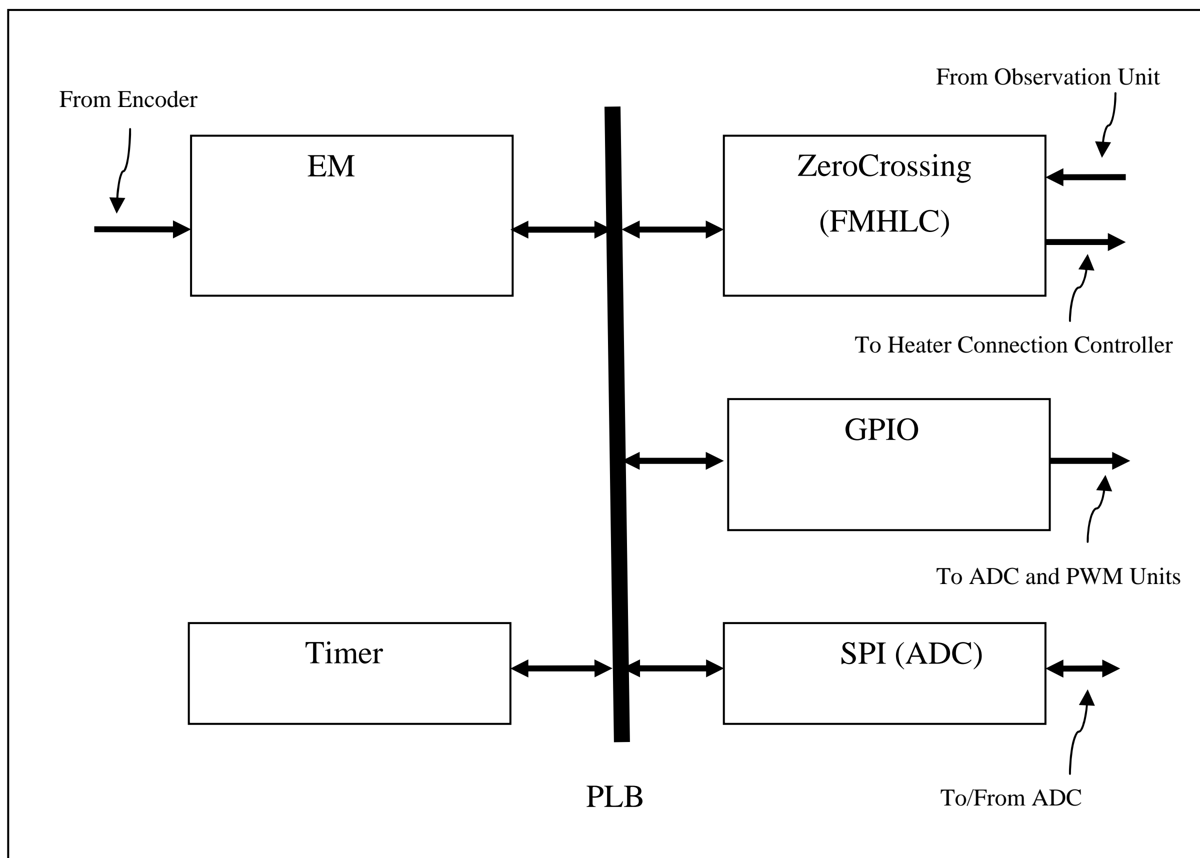
Figure 2.6 Extension board with ADC and DAC chips



The Xilinx Microprocessor on the FPGA Board:

The microprocessor chosen for this project serves as the main microcontroller that is used for the system. It is based on the PowerPC architecture. The microprocessor is mounted on an FPGA board. The input/output as well as the programming interface to the microprocessor is accessed through the same board. In fact, the Xilinx microprocessor has so many functionalities and capabilities compared to a normal microcontroller and it is way beyond our needs. However, with all of its flexibilities and user definable hardware it is an ideal microprocessor for this project. The microcontroller that built with this microprocessor establishes the core control unit for the whole system. It takes different input signals from the system and based on those inputs and the control logic, it generates proper outputs to control the speed of the prime mover and the current flowing to the Heater Resistors. Below is the block diagram showing the hardware peripheral units built and connected to the microprocessor:

Figure 2.7 Block diagram of the IP Interconnections



Some blocks which are of higher importance to us are detailed below:

Frequency Measurement and Heater Load Controlling (FMHLC) Unit:

This peripheral unit takes the output signal from the observer unit. At each positive edge (rising edge) of the received signal it will start counting the clock pulses and will continue until the next rising edge of the signal is detected. At that very moment, it latches the count to a register, clears back the count and restarts the count from 0. Based on those counts it could be decided if we are off or above 50 Hz and we can generate proper outputs accordingly.

It is very important to get those counts quite accurately and it should really represent the actual number of clock cycles that a period of a 50Hz signal would take. However, it gets challenging when the noise issue comes into the picture. Since the observer unit is responsible to send square pulses to the FMHLC unit based on the zero-crossing points of the AC voltage generated by the generator, it always decides if it should restart counting or continue the previous counting based on the input provided by the Observer Unit. At the same time, the output of the observer, due to some reason, is noisy. The noise on the mentioned signal causes the FMHLC IP to mix between real rising edges and the ones caused by the noise. Therefore, the FMHLC IP ends up providing erroneous count values.

To avoid the effects of noise on the count values, a de-bounce algorithm which is mainly used in push button applications has to be deployed. The de-bounce algorithm more or less acts as a filter. In a sense, it is getting the input, waiting for a while so that the input signal gets stable, then triggers the counter to start counting. Since noise is of much higher frequency than the actual signal, so during the waiting period the noise disappears and gives the counter a feel of steady high or steady low depending on what signal level it receives actually.

The other function of this hardware unit is to control the current flowing to the Heater Resistors. The unit performs this function by taking a number from the microprocessor and comparing it to the number of clock pulses that it counts as explained above. Each time those two numbers are equal it will send an “Enable” signal to the Heater Connection Controller. (Note: The “Enable” signal is Logic Low because the Heater Connection Controller is an active low unit).

The number received from the microprocessor varies based on the frequency measured by the microprocessor and the amount of current flowing to the Heater Resistors. The lower the number registered from the software, the longer the Heater Resistors will be connected to the generator. Therefore, we have a full control over the Heater Resistor current flow.

Encoder Monitor:

This IP receives signals from the encoder mounted on the belt connecting the motor and generator and translates those encoded signals to counts. Those counts eventually translate to the speed of the encoder wheel which will help me monitor the speed of the system. This IP together with its encoder was primarily used for instrumentation purposes and to give further insight of the system. However, such a mechanism wouldn't be available on an actual site. Besides, the motor – generator set connected with a belt had a very inconsistent output voltage. Thus it was necessary to change the set to a different setup using a motor – generator set coupled directly with their shafts. Therefore, the Encoder Monitor IP as well as its Encoder is not used in this project. Just for the sake of completion, the IP in software project is kept and its HDL code in Verilog language is included in Appendix A under HDL Verilog Code for Hardware.

General Purpose Input/output Unit:

The main purpose this IP Unit is to let the software access the external ports on the FPGA board connected to the microprocessor. At the moment four pins are accessed through this unit and it is flexible to extensions of up to 32 pins if the need may arise. The two pins are hard wired to the MCS08 Microcontroller to control its PWM signal's duty cycle. The other two pins are connected to the ADC board ADC/DAC selection purposes which will be discussed under ADC Controller. The two pins connected to the MCS08, change their level one at a time based on the instructions of the microprocessor. Moreover, the instructions from microprocessor are based on the frequency of the system monitored by the FMHLC Unit and the current flowing to Heater Resistors.

ADC Controller:

This part involves three different IPs from the microprocessor, the SPI controller, Timer, and the GPIO.

The SPI (Serial Peripheral Interface) Controller is used to communicate with the ADC. Three pins on the FPGA board are connected to this IP namely, SCK, MOSI and MISO. The SCK is connected to the SCK pin of the ADC chip providing clock for synchronizing the serial communication. [LTC 1867 Data Sheet]. The MOSI pin is connected to the SDI pin of the ADC chip providing setup information to the chip serially. Last but not least, the MISO pin is connected to the SDO pin of the ADC chip receiving the converted data serially.

The Timer IP is used internally and set up to allow time for the chip in order for it to convert an analog data to digital.

The GPIO IP as mentioned under the "General Purpose Input/output Unit" is sharing two of its pins with ADC Controller. They are merely used to activate or deactivate ADC and DAC one at a time. Since there is no need for the DAC part, the software should keep one pin always in logic high. The other pin which activates or deactivates the ADC will toggle as needed.

All inputs and outputs for the microprocessor are listed in Table 2.4 below.

Table 2-4 List of Inputs and Outputs on the FPGA board

Pin #	Pin Name	Purpose	Connected
8	AC24	receiving serial data from the ADC	to the SDO pin of the ADC Chip
10	W25	sending serial data to the ADC	to the SDI pin of the ADC Chip
12	AB24	providing clock pulses for the ADC	to the SCK pin of the ADC Chip
14	Y24	providing enable/disable bit for the ADC	to the CS/Conv pin of the ADC
18	W26	receiving pulses from the encoder	to channel A of the encoder
20	Y26	receiving pulses from the encoder	to channel B of the encoder
22	Y25	command bit for PWM control	to pin 9 of the MCS08 Chip
24	AA26	command bit for PWM control	to pin 10 of the MCS08 Chip
26	AA23	providing disable bit for the DAC	to CS/LD pin of the DAC Chip
30	AB26	active low signal to Heater control	to pin 2 of MOC3031 Chip Phase C
32	AC23	active low signal to Heater control	to pin 2 of MOC3031 Chip Phase B
34	AB25	active low signal to Heater control	to pin 2 of MOC3031 Chip Phase A
36	AD23	receiving signal from zero-crossing detect.	to pin 15 of PS2501 Chip (Phase A)
38	AC26	receiving signal from zero-crossing detect.	to pin 15 of PS2501 Chip (Phase B)
40	AD26	receiving signal from zero-crossing detect.	to pin 15 of PS2501 Chip (Phase C)

Figure 2.8 shows how the FPGA board looks like.

Figure 2.8 The FPGA Board

DC Power Supply:

So far, whatever has been discussed were all about electronic circuits that are needed for accomplishing the project. All the mentioned units are to receive their power from an external independent DC power source. The DC power supply used for this project outputs three different voltages. The different output voltages and their use are listed in table (2-4).

2-5 List of output voltages of the DC Power Supply

Value	Unit	Use
3.3	V	Supply voltage for MCS08 Microcontroller
5.0	V	Supply voltage for the Encoder
24	V	Supply voltage for the Motor

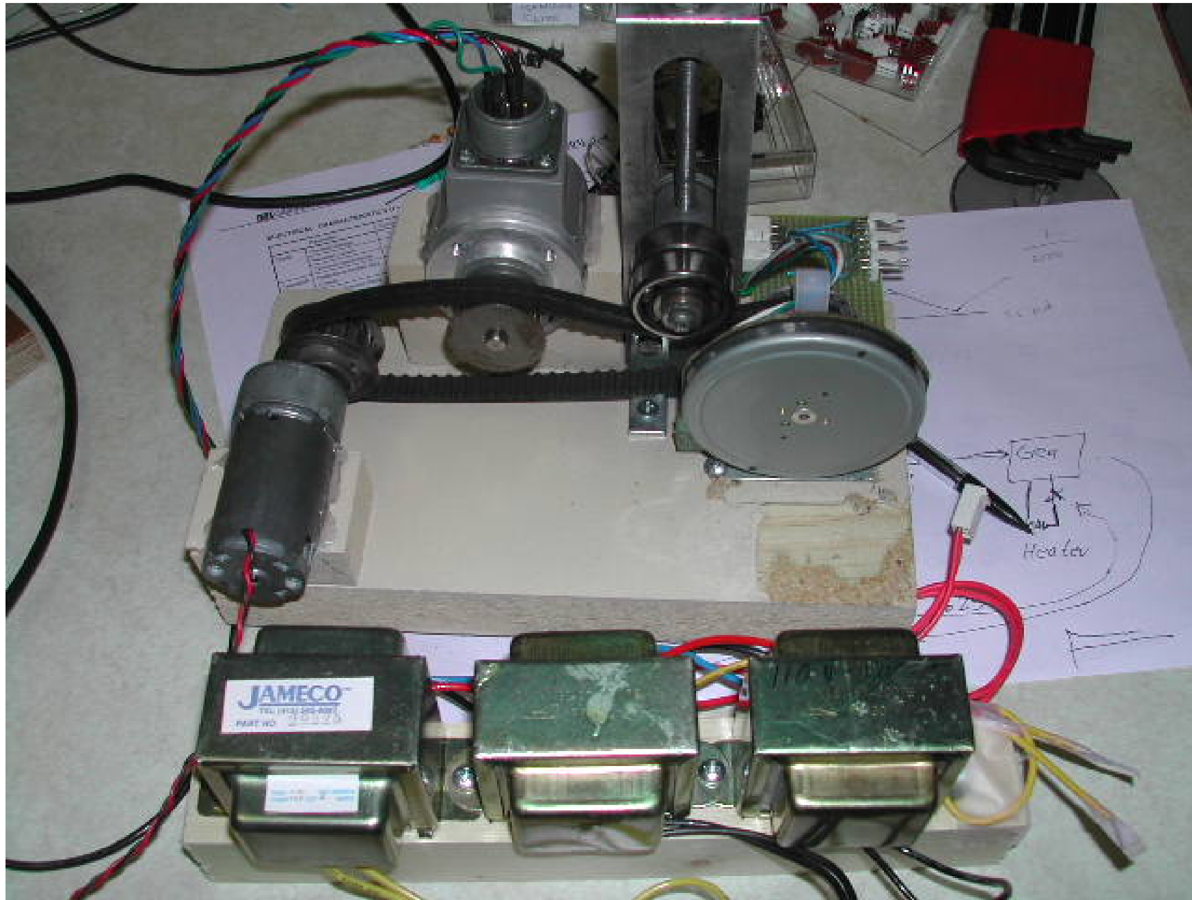
The input voltage for the power supply is a 12V DC that is provided from the bench power supply. Depending on how we would implement our algorithm on an actual plant, it will be decided how we would take care of the power supply for the hardware on site. Figure 2.9 shows the picture of the DC power supply board.

Figure 2.9 DC Power Supply Unit

The Generator Set:

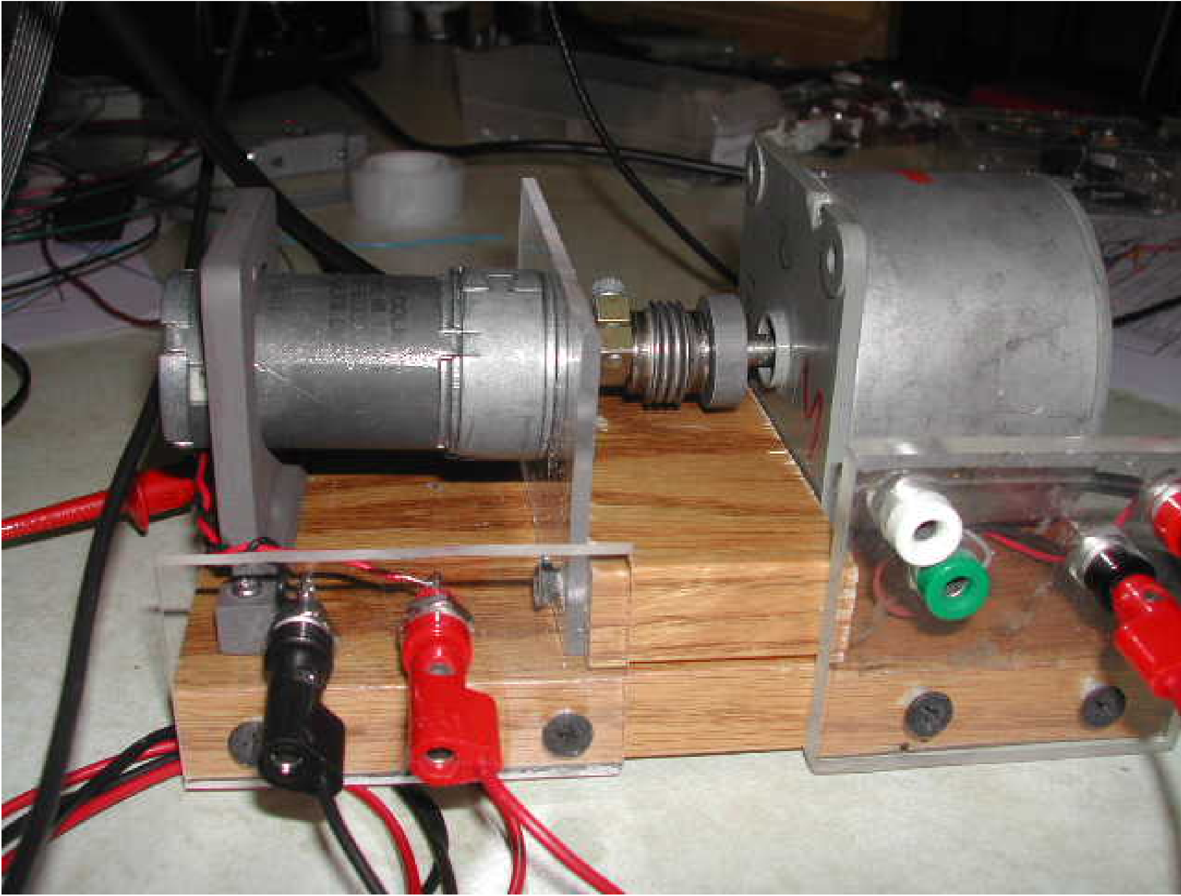
Initially it was composed of a motor, a generator, a belt that was coupling the tow and an encoder. Since the generator was providing a small voltage of almost 1.5 V AC, there was need to connect some sort of transformers to it. The transformers helped to step up the generated voltage to approximately 10 volts. The other side of the story is that the transformers affected badly on the signal shape. In addition, the way the generator set was setup caused that it would never generate a frequency in a stable range. Hence, a replacement of the whole set with a different setup was necessary. Figure 2.10 shows the old generator set with the transformers.

Figure 2.10 The initial power generating setup



The new setup is only a motor and generator which are coupled through their shafts. This generator has gotten its excitation DC voltage and provides an output of 10 to 15 volts depending on its excitation input voltage. With help of this generator set it is easy to remove the transformer unit so the generator set gets much simpler. The bottle neck is the fact that this generator has two set of windings generating two phase voltage. As a result we may end up using only one phase for all the experiments and tests. Figure 2.11 shows the new generator set.

Figure 2.11 The generator set currently used for the test purpose



Chapter 3 - Experiments and Tests and Results

This chapter will document all experiments done throughout the project. Later it will also include the tests done on the complete system. The results of every individual test will accompany the test at the end of each test write up.

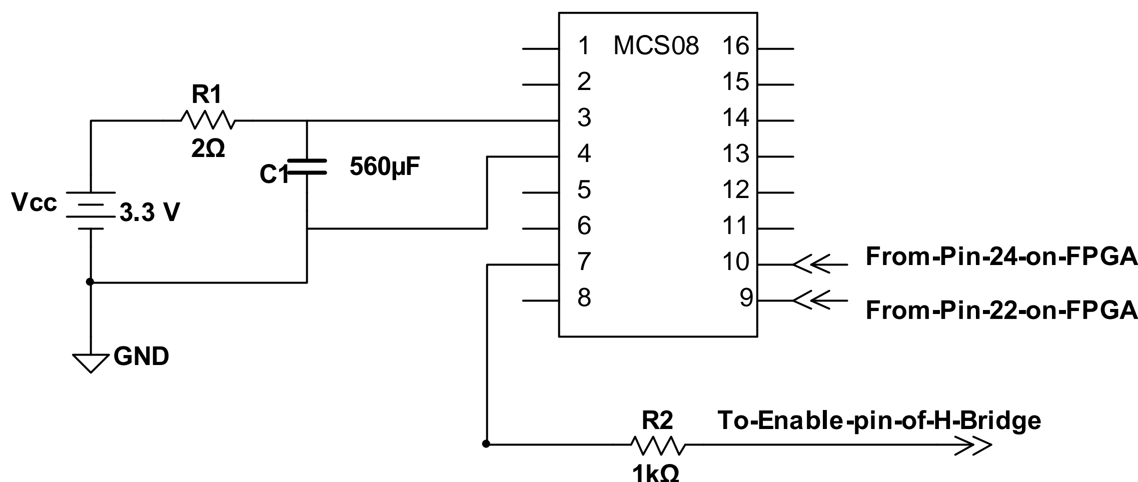
Experiment 1 – Generating the PWM

This experiment involves programming the MCS08 Microcontroller, connecting it to power and to the H-Bridge. Below is the list of material needed for this experiment. The material will be permanently used as this experiment builds a small part of the whole system.

1. MCS08 Microcontroller
2. A 0.5 watts 20 ohm resistor for connecting pin 3 of the chip to the 3.3v power supply
3. Filter capacitor used between pin 3 and 4 (grounding pin) for power supply stability
4. A 1 kilo ohm resistor connecting pin 7 to “Enable” pin of the H-Bridge

The chip is programmed such that pins 9 and 10 are configured as input receiving signal from the main microprocessor and pin 7 is used to output the PWM signal to the H-Bridge. The code written for this purpose is included in Appendix A under MCS08 Assembly Code. Figure 3.1 shows the complete interconnections. Appendix B includes all the names for all the pins of the chip.

Figure 3.1 Microcontroller connection



After the chip is programmed it is put in the circuit as shown above and then connect its PWM pin to the scope. The input pins are not connected to the main microprocessor yet. The input pins are connected to 3.3 each at a time and the scope is monitored to see how the duty cycle varies as we change the connection of input pins. This process takes a while till you can figure out the upper and lower limits for the duty cycle. It is much helpful if we have the motor also connected to the H-Bridge and PWM connected to the enable pin of the H-Bridge. The motor would turn the generator and if we connect the output of the generator to the scope as well, we can soon figure out what our highest and lowest duty cycles should be.

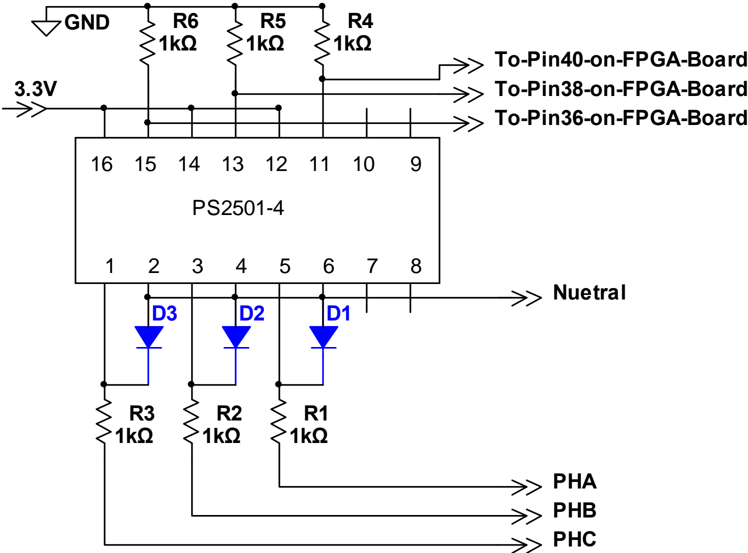
Experiment 2 – Building the zero crossing detection circuit

This circuit as explained in chapter 2 gets the generated AC voltage from the generator as input and sends square waves to the Frequency Measurement and Heater Load Controlling (FMHLC) Unit of the main microprocessor. The rising and falling edges of the wave correspond to each zero's crossed by the AC voltage. The list below details about the material needed.

1. A four channel PS2501 chip
2. Three rectifier diodes used to protect the internal diodes when reverse biased by the negative half cycle
3. Six 1 kilo ohms resisters for current control issues

The interconnections are done as Figure 3.2. Appendix B details on internal connections of the chip.

Figure 3.2 Schematic for the zero crossing detection circuit



The functionality of this circuit can be tested easily. Only one phase is enough to be connected to the resistor connected to say pin 1 of the chip. Then connect pin 16 to 3.3 volts and the scope to pin 15. It is better to connect the analog probe of the scope for better seeing the slope of the output signal. You can see in Figure 3.3 that at start of each cycle of the input voltage, we get a rising edge and at the end of the first half cycle we get the falling edge.

Figure 3.3 Output of Zero Crossing detector in correspondence of AC voltage

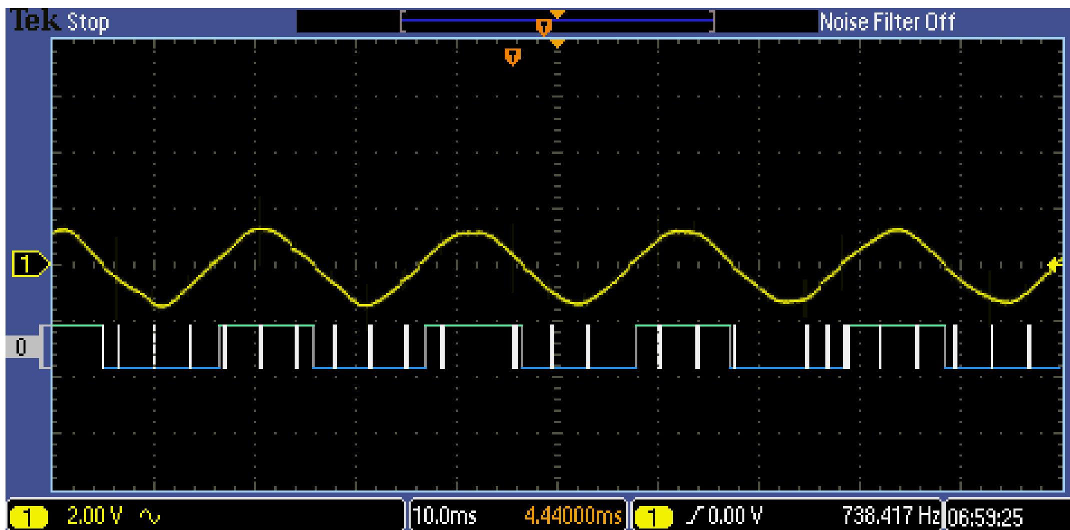
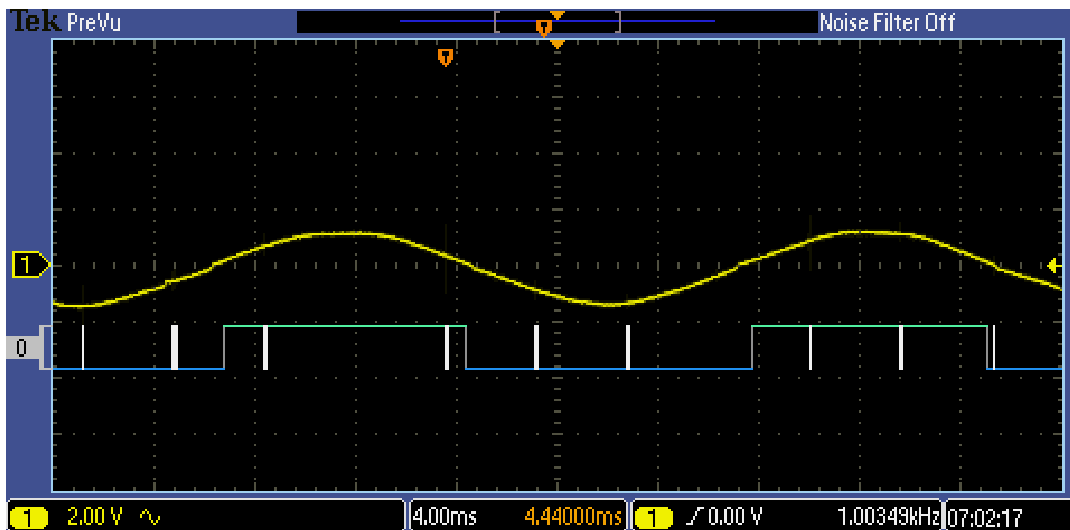


Figure 3.4 The zoomed in version



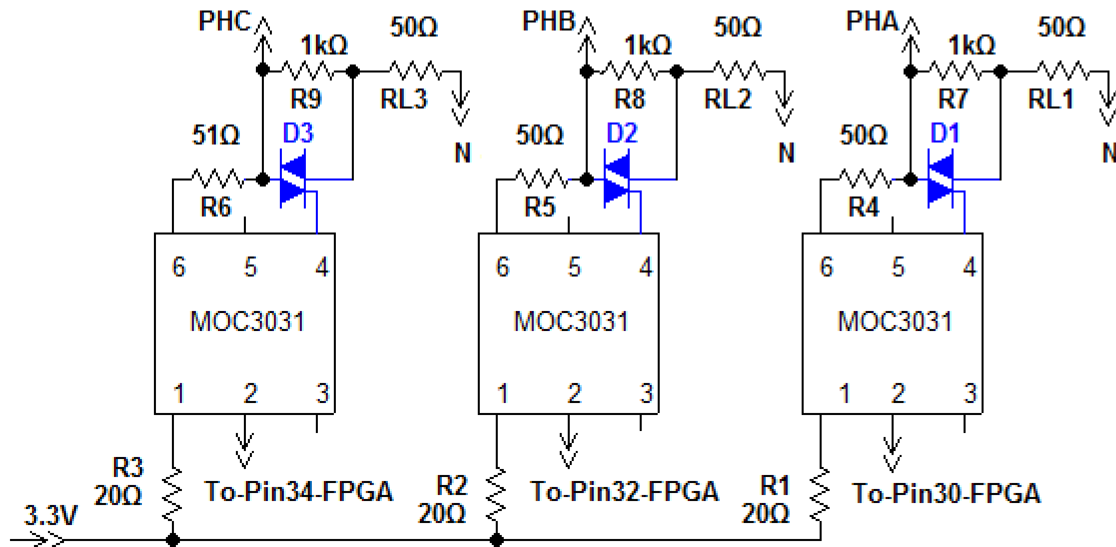
Experiment 3 – Building the Heater connection controller

This circuit is built to automatically connect the Heater Resistors to the AC voltage based on the signal received from the Frequency Measurement and Heater Load Controlling (FMHLC) Unit of the main microprocessor. Below is the list of material needed for this circuit.

1. Three MOC3031 chips
2. Three 51 ohm resistors
3. Three 20 ohm resistors
4. Three 39 ohm resistors
5. Three 1 kilo ohm resistors
6. Three 0.01 micro Farad capacitors

Figure 3.5 is the diagram for the controller. RLs represent the Heater Resistors. The diagram for internal connections of the MOC3031 chip is available in Appendix B.

Figure 3.5 Circuit of the Heater Resistor controller



This experiment was done after the Frequency Measurement and Heater Load Controlling (FMHLC) Unit were properly functional. To monitor the current switching, connect a digital probe of the scope to pin 2 of the MOC 3031 chip and an analog probe of the scope to a Heater Resistor. It can be seen in figure 3.6 that whenever there is a logic low on pin 2, a portion of the sinusoidal AC voltage shows up on the resistor which means the resistor starts loading the generator.

Figure 3.6 Correspondence of “logic low” on digital channel 1 with analog channel 2

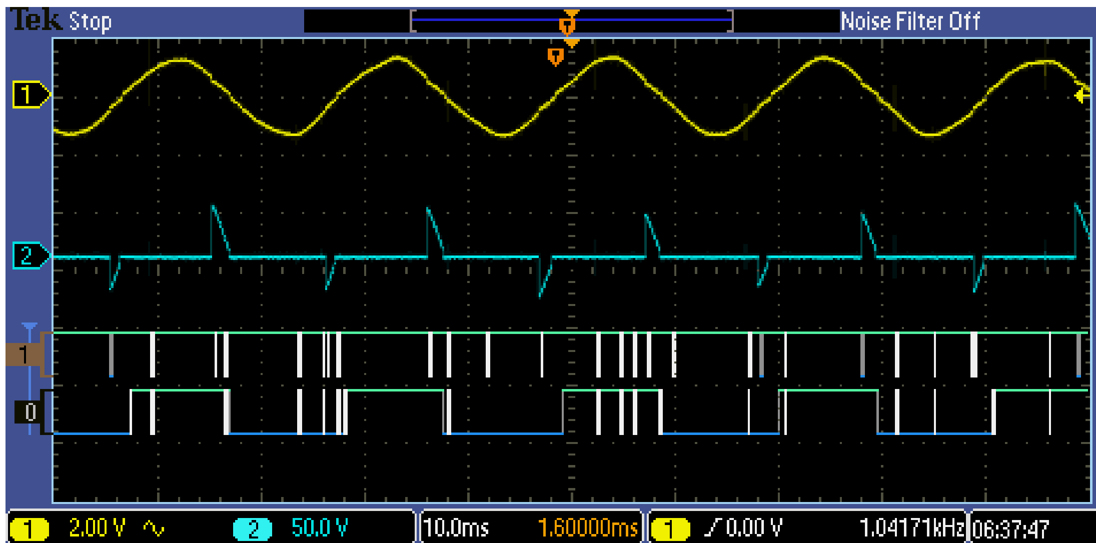
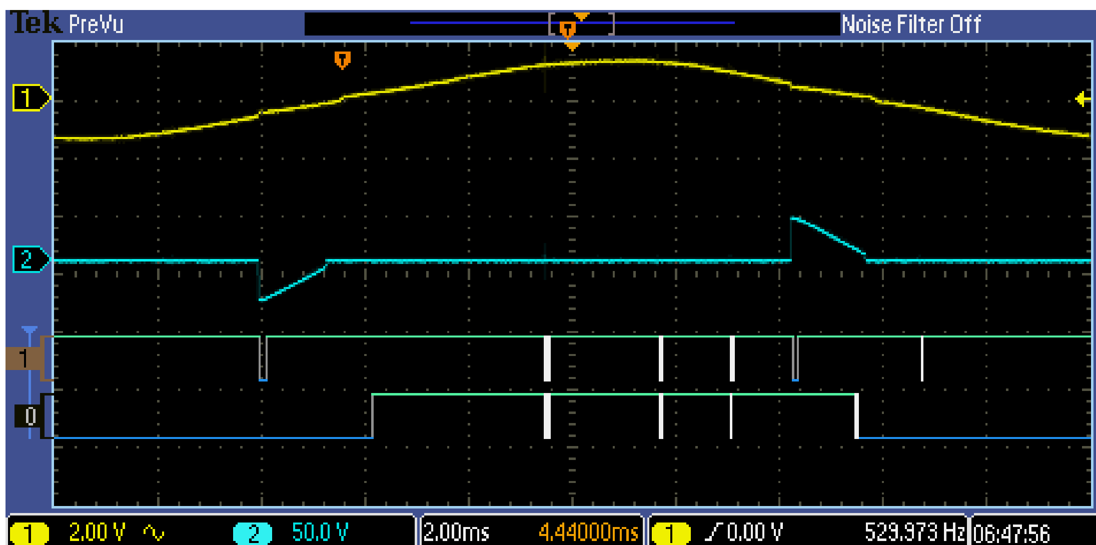


Figure 3.7 Zoomed in version

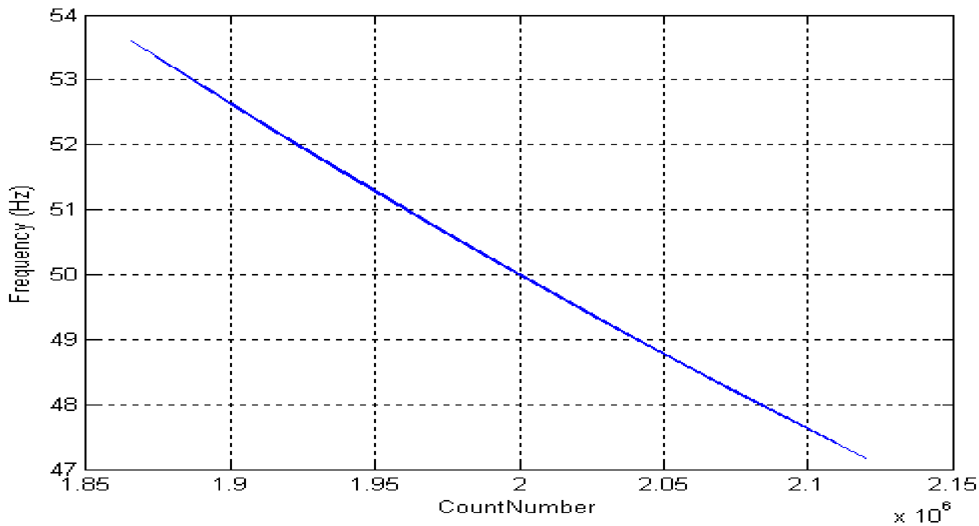


Test 1 – Reading Frequency under no load condition

The purpose of this test is to figure out a count number from the Frequency Measurement and Heater Load Controlling (FMHLC) Unit that would correspond to 50 Hz. Once this number is obtained, it would be the basis of the comparison between other numbers which the software would receive when the system gets loaded. The software will always compare each number that it receives from FMHLC unit with the 50Hz corresponding number. Then it will decide if it should send back a bigger or a smaller number to the FMHLC unit to increase or decrease the amount of current flowing to the Heater Representing resistors.

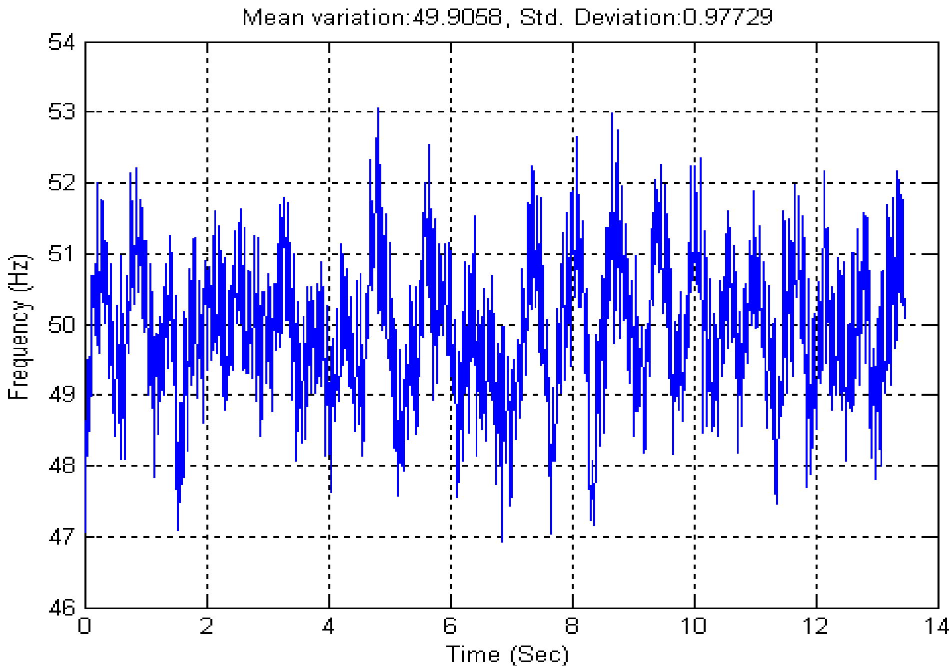
After performing this test it was possible to record numbers going up and down around (2,000,000) number of counts as the number corresponding to range of 48 to 52 Hz. Figure 3.8 shows the variations of numbers with respect to time.

Figure 3.8 The graph showing the variations of numbers around 2,000,000.



It can also be seen that how frequency changes over short period of time. Figure 3.9 demonstrates this fact further.

Figure 3.9 Frequency variations over time



Since the numbers represent time, they have inverse relation with the frequency. Keeping that in mind, if the software receives bigger numbers than the above mentioned one, it should decrease the amount of current flowing to the Heater Resistors. In other words, it should decrease the load on the generator so that the frequency could get adjusted back to 50 Hz. On the other hand, once the village load decreases the frequency will increase. Consequently, the numbers received by the software start decreasing. Hence, that would be the time when software would start loading the generator with the Heater Resistors.

The current limit up to which we can keep current flowing to the Heater Resistors will be examined in Test 2. Test 2 will also elaborate on how the software is letting lower or higher current flows to the Heater Resistors.

Test 2 – Running the system under full load of the Heaters

In this test the goal is to have an insight of maximum current drawn by the Heater Resistors. Besides, the limits based on which the control system should take care of the speed of the driving motor should be recorded. Prior to doing so, it would be helpful to include some details on how the software controls the current flowing to the Heater Resistors.

Letting specific amounts of current flow through the Heater Resistors is done by software. In a sense, software determines for the hardware when to trigger a logic low to the Heater Connection Controller unit. Only then, when the Heater Connection Controller unit detects that zero voltage, it pulses the "gate" pin of the Triac which in turn will allow current to flow to the Heater Resistors. How the software determines for the hardware when to trigger a logic low is based on frequency counts read from the FMHLC unit. The software takes 80% of half of the number it has read from the FMHLC unit and sends it back. The FMHLC unit uses the received number as a threshold value and whenever the frequency count reaches that threshold, the FMHLC unit switches the pins connected to Heater Connection Controller unit (i.e. pins 30, 32 and 34 on the FPGA board) to logic low. This, in turn, will signal the "gate" pin of the Triacs switching power from the generator to the Heater Resistors.

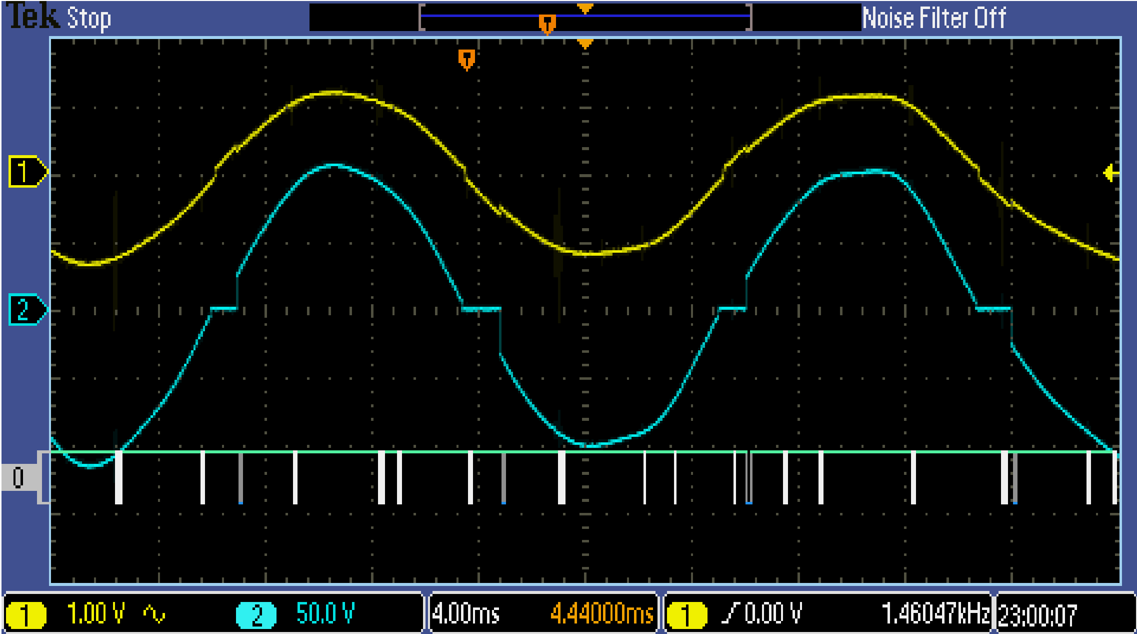
Sending numbers back and forth in the manner just described above leads us to some limits where we would definitely have to do something with the speed of the driving motor. In other words, when the dynamic load decreases to its minimum, there would be almost all the generated current flowing to the Heater Resistors. Or on the other hand, if the increased Dynamic

Load causes the current flowing to the Heater Resistors fall below the preset threshold as we discussed earlier. In such cases we have to decrease or increase the speed of the driving motor.

This test does not include the increase or decrease of the driving motor speed. However, this test will set forth for us the limits based on which we can decide when to speed up or down the driving motor.

The 40% of the counts that software receives from the FMHLC unit could be a good value for the number that could cause a minimum current to flow through the Heater Resistors. If we name this number X, then we should always have the software to keep all numbers below X while sending them to FMHLC unit. At the same time, guessing the smallest number to cause the Heater Resistors to draw a maximum current is not hard. Starting from 0 to some value say 1000 would cause a maximum current flow, but due to overlapping issues, let's avoid using range of 0 to 100. However, since we may not want to anyway draw the whole generated power to the Heater Resistors, a smaller percentage say 5% of the number received from the FMHLC unit could help much better. After some tests the maximum current flowing to the Heater Resistors was recorded as 3 mA. Figure 3.10 shows the voltage across the Heater Resistor when the maximum current is flowing through it.

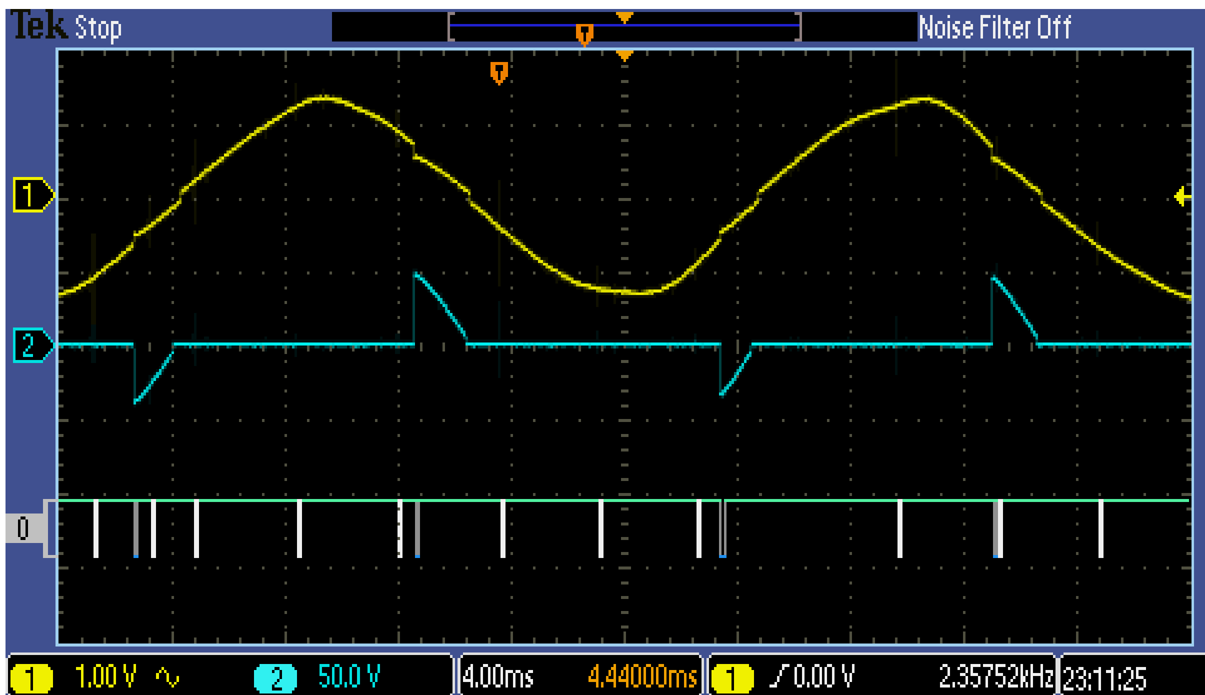
Figure 3.10 Graph of maximum voltage across the Heater Resistor (Blue) in correspondence with the AC input Voltage (Yellow) while 5% of the number was fed back to FMHLC unit



In order to make sure that we are not going below minimum current limit or above maximum current, we may try to have a voltage reading across the Heater Resistors. The voltage will be then input to the ADC. Then the software will read the ADC output and translate it to current using Ohm's law.

Using the number X resulted in a maximum current flow of 0.4 mA to the Heater Resistors which is 13% of the maximum current. Figure 3.11 shows the graphs while X was sent by the software to the FMHLC unit.

Figure 3.11 Graphs of voltage across Heater Resistor (Blue) and input AC Voltage while 40% of the number was fed back to FMHLC unit

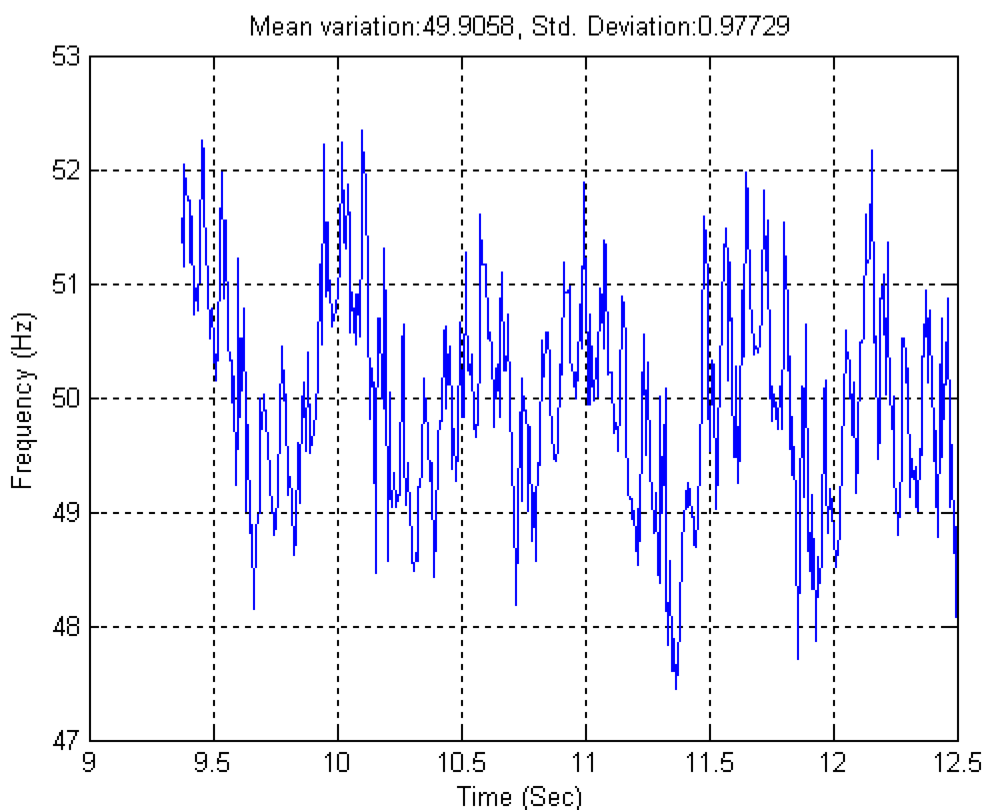


Chapter 4 - Conclusions and Future Work

As we saw through previous tests, the results and data mostly depend on the hardware used for this project. Especially the Zero Crossing detector as well as the generator set has played a major role in estimating the frequency of the generated power. Small changes in the generated power frequency caused a huge difference in the numbers read as a measure of frequency. Besides, the electronic setup of the Zero Crossing detector and its related delays introduce quite a bit of error that has to be taken into consideration.

Frequency stability regardless of the load on the generator has been always an issue. Throughout the tests, three generators were evaluated to get a stable frequency, however, it remained a challenge to the end. As was observed in Test 2 of Chapter 3, the frequency variations seemed to be quite high. Figure 4.1 shows a zoomed in version of the frequency deviations over a very short period of time.

Figure 4.1 Frequency variations over a period of 3 seconds



To try and minimize these variations a closed loop control system was implemented in the software. The approach was to compare the readings from the FMHLC unit with the 50 Hz corresponding number i.e. 2,000,000 counts (when using a 100 MHz Clock) and based on the comparison results either speed up or slow down the driving motor. Changing the speed of the driving motor takes several milliseconds as the nature of the PWM controller suggests. Thus, keeping track of each number received from the FMHLC unit instantaneously would not make any sense. Instead, it was decided to compare the average of these numbers with the target count and then decide to speed up or slow down the driving motor.

For completeness, some tests were done to figure out what could be the best averaging and speed controlling rate. Finding the best rate would help in the case of not being able to get a stable frequency from any generator. The following figures demonstrate how different averaging rates affect the deviations of the frequency. Throughout the tests, it was also revealed that in different speed control rates, we had to compare the average with different count numbers to keep the frequency around 50 Hz. For example, Figure 4.2 shows the frequency changes over time when the averaging and speed control were taking place at every second. To achieve a proper speed an average of received numbers was compared to 1,950,000. This difference is believed to be the offset seen in many proportional control systems.

Figure 4.2 Frequency variations over time – speed control rate: once per sec.

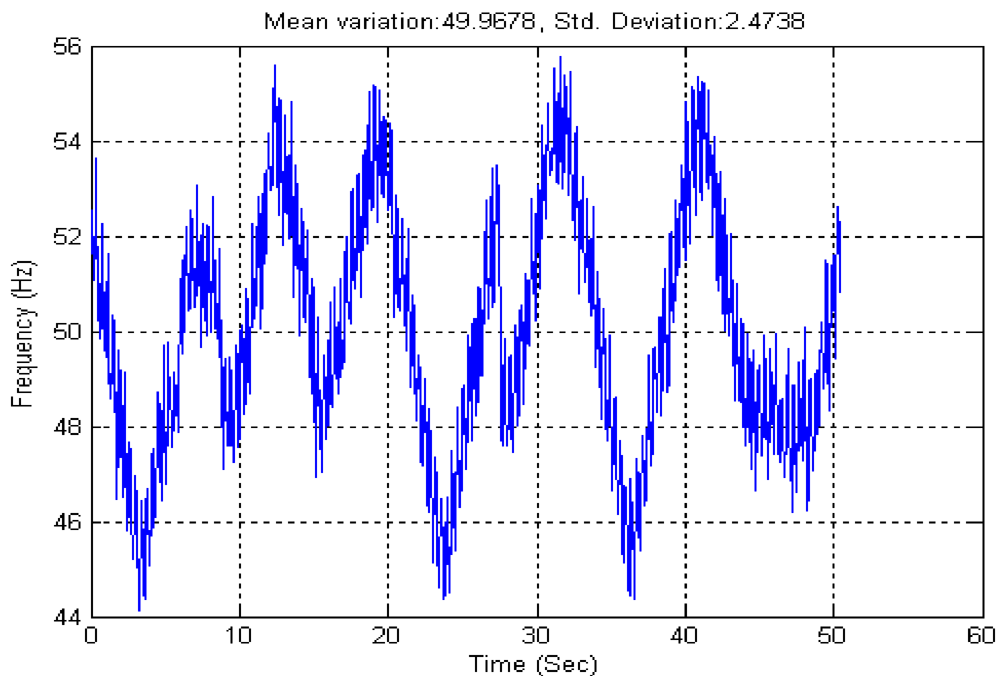
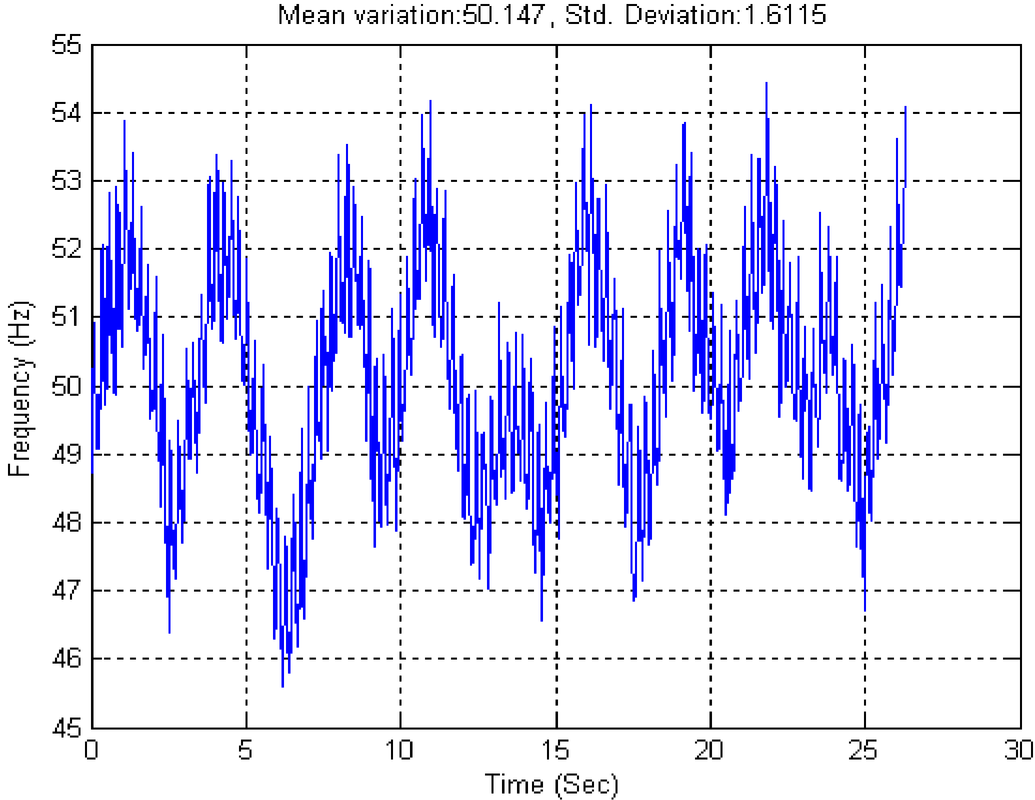


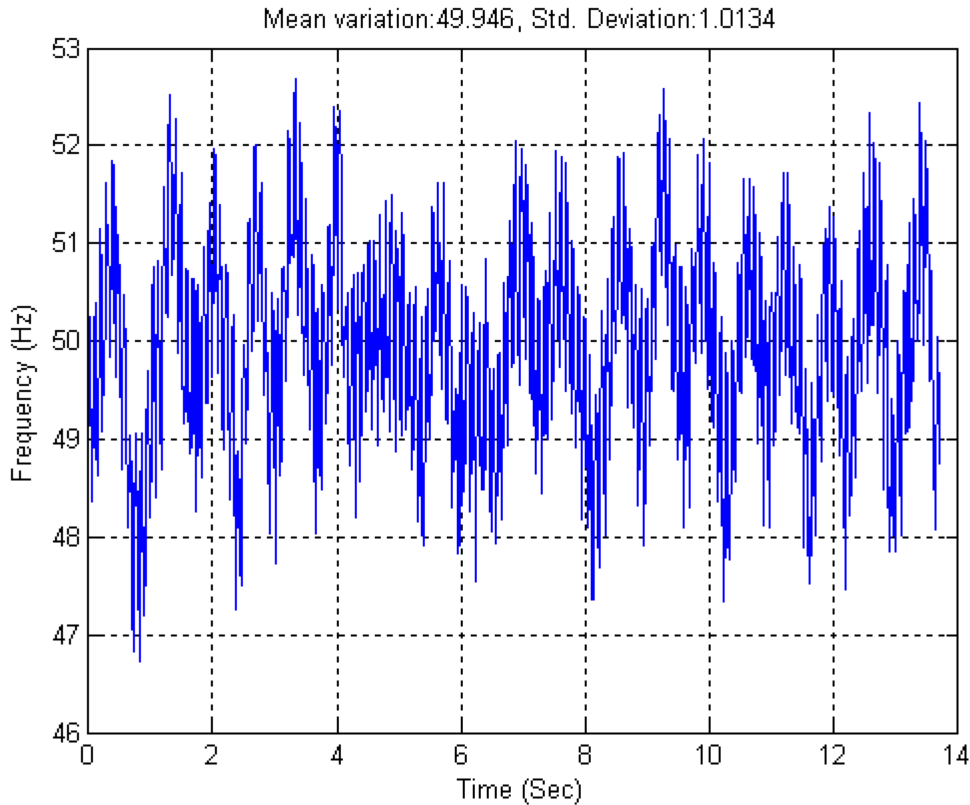
Figure 4.3 shows frequency deviations under speed control rate of twice a second. The average was compared with the number 1,900,000.

Figure 4.3 Frequency variations over time – speed control rate: twice per sec.



And Figure 4.4 shows the changes when the speed control was taking place four times in a second. The average comparison was done against **1,850,000**.

Figure 4.4 Frequency variations over time – speed control rate: four times per sec.

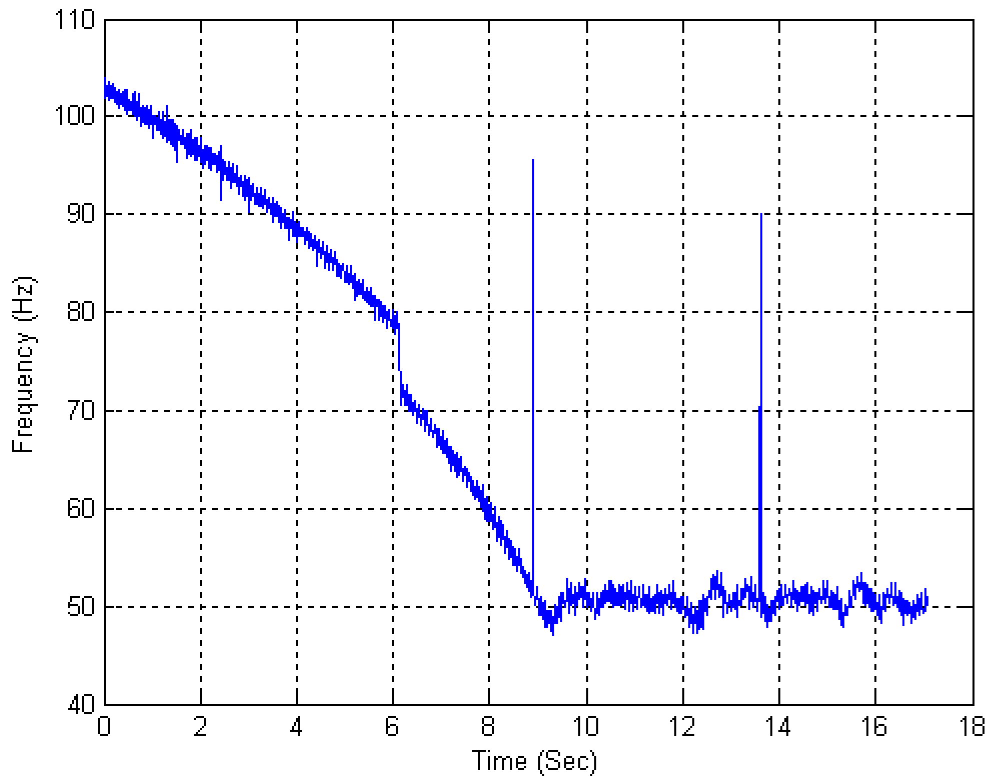


Based on the generator sets available, it was not possible to observe the small loads that these generators could handle to the motor side as a torque as well. In other words, the driving motors were too powerful to get loaded by the generator as the generator was connected to a load. Instead, it was the terminal voltage that always varied regardless of how much speed the generator set had. As a result, it was not possible to test the generator speed control based on different loads on the generator, which thus remains open for future work on this project.

In spite of all the above challenges and short comings, we were able to build and make ready all the pieces of the hardware necessary to establish a working test system. As explained in Chapters 2 and 3 all the parts have been tested and resembled the expected results individually. In addition, the parts have been assembled together and tested as well. From a functional point of view, the system performs as expected except for the accuracy and measurement part and the belief is that it could be addressed in the future work once a dependable generator set is obtained. The results included in Chapter 3 confirm this opinion. Besides, it was tested and confirmed that the control algorithm works well to keep the frequency in the range of 47 to 53 Hz as

demonstrated in the above figures. Furthermore, as shown in Figure 4.5, the system can be brought to the mentioned frequency range from any frequency it would generate from the beginning.

Figure 4.5 shows how the generated frequency is brought to the desired range



For the future work on this project, it would be appropriate that some further steps be taken to fine tune, and closing the control loop with more appropriate constraints for the system. By fine tuning, it is meant that a more stable generator to the system needs to be acquired. Then, it is anticipated that a much more reliable and stable frequency, that can be accurately monitored will be generated. Closing the control loop for the system will be also accomplished once there is a better generator set. After having a generator set that could reflect small amounts of electric load to the motor shaft as a mechanical load, it will be possible to control the speed of the motor based on the accurate frequency measurements from the system. A detailed test scenario to determine if the system can switch between Dynamic Load and Heater Resistor Load and then

whether it can control the speed and frequency will be executed as well. This test will also be performed in the future when a more stable generator set is found.

Amongst the future works that have to be considered as well are below mentioned tow additional points. Namely, implementation of Kalman Filters in the closed loop control system and implementing the control algorithm using multi processors.

Implementing a Kalman filter insures system accuracy. Although controlling the closed loop system in its basic form would be so easy and less time consuming, if we expect the system to minimize the mean squared error of the terminal voltage and frequency we may want to implement a Kalman Filter in the software. Although Kalman Filter has a vast variety of applications, one could be to apply it to minimize mean squared error [1]. Details about Kalman Filter implementation is out of the scope of this report and would be experimentally discussed as part of the future work.

Use of multi processors to implement a certain algorithm would be cheaper in comparison with the use of a single multifunctional microprocessor to implement the same algorithm. The cost of the Xilinx Microprocessor and the FPGA board leads to choose the option of multi processors. By multi processor, it is intended that microcontrollers perform the task of the hardware that are done by a single microprocessor. In this case, each microcontroller will assume a single task. Depending on the criticality of synchronization, we may use a common clock source or we can have each single microcontroller with its own clock source. In our case, for example, one microcontroller will perform the tasks of the FMHLC unit; the other one would take care of speed control, ADC control and communication with FMHLC unit. Or we can even avoid using the ADC unit and instead employ a microcontroller to do both conversion and control parts of ADC. In the latter case, we would need three microcontrollers. One would be used for the FMHLC unit as before. Likewise, a second microcontroller will handle the speed control and communication with FMHLC unit. Finally, a third one should be used to completely take care of the ADC and its controlling issues. Of course, it may be that one processor be used to control the other three and to also control the interconnections between all the microcontrollers, servicing all the communications between them. In such a situation, a common clock source is preferred for all the microcontrollers. However, the clock must be generated by the main or central microcontroller.

This report ends and leaves me with some lessons that I learned throughout this project. It would help to mention some of the main ones for the sake of emphasize. The first and foremost issue that should be always considered is the noise issue. Noise simply spoils all the assumptions and accurate calculations. For example, the noise causes at the beginning to read random numbers from the Frequency Detector unit. Eventually, you have to compensate for that and employ a de-bouncing unit. Second important issue is the delay caused by the electronics used outside the microprocessor world for the instrumentation. As an instance, the need of offsetting the counter for the frequency detector in order to account for the delay caused by the Zero-Crossing Detector circuit and so on. Or the time when it was necessary to keep the signal going to Triac low for a set number of counts in order to have them properly trigger a Triac Gate. Two more points should be mentioned as helpful ideas to these issues. One is to use a slower clock generator which would help not to accumulate more error while counting the clock cycles for any purpose. The second point is that the magic number, 2,000,000 counts, does not always resemble the 50 Hz frequency as long as noise is involved there or the clock frequency is 100 MHz.

To conclude, there are a lot of other ideas that could start the future work on this project. Nevertheless, the last two or three paragraphs give an idea of what improvements could be expected in the future to have a better control over the terminal voltage and frequency of a micro hydro power plant.

References

1. Elder, J.M.; Boys, J.T.; Woodward, J.L.; , "Integral cycle control of stand-alone generators," Generation, Transmission and Distribution, IEE Proceedings C , vol.132, no.2, pp.57-66, March 1985 doi: 10.1049/ip-c:19850012
   URL:
   http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4646439&isnumber=4646438
2. Tamrakar, I.; Shilpakar, L.B.; Fernandes, B.G.; Nilsen, R.; , "Voltage and frequency control of parallel operated synchronous generator and induction generator with STATCOM in micro hydro scheme," Generation, Transmission & Distribution, IET , vol.1, no.5, pp.743-750, September 2007 doi: 10.1049/iet-gtd:20060385
   URL:
   http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4295001&isnumber=4294995
3. Profumo, F.; Colombo, B.; Mocci, F.; , "A frequency controller for induction generators in stand-by minihydro power plants," Electrical Machines and Drives, 1989. Fourth International Conference on (Conf. Publ. No. ??) , vol., no., pp.256-260, 13-15 Sep 1989
   URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=55402&isnumber=2002
4. Singh, B.; Murthy, S.S.; Gupta, S.; , "An improved electronic load controller for self-excited induction generator in micro-Hydel applications," Industrial Electronics Society, 2003. IECON '03. The 29th Annual Conference of the IEEE , vol.3, no., pp. 2741- 2746 Vol.3, 2-6 Nov. 2003 doi: 10.1109/IECON.2003.1280681
   URL:
   http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1280681&isnumber=28610
5. Marinescu, C.; Clotea, L.; Cirstea, M.; Serban, I.; Ion, C.; , "Controlling variable load stand-alone hydrogenerators," Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE , vol., no., pp. 6 pp., 6-10 Nov. 2005 doi: 10.1109/IECON.2005.1569308
   URL:
   http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1569308&isnumber=33243
6. Marra, E.G.; Pomilio, J.A.; , "Self-excited induction generator controlled by a VS-PWM bi-directional converter for rural applications," Applied Power Electronics Conference and Exposition, 1998. APEC '98. Conference Proceedings 1998., Thirteenth Annual , vol.1, no., pp.116-122 vol.1, 15-19 Feb 1998 doi: 10.1109/APEC.1998.647678
   URL:
   http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=647678&isnumber=14120
7. Khan, P.K.S.; Chatterjee, J.K.; , "Modelling and control design for self-excited induction generator with solid-state lead-lag VAr compensator in micro-hydro energy conversion scheme," *TENCON '98. 1998 IEEE Region 10 International Conference on Global Connectivity in Energy, Computer, Communication and Control* , vol.2, no., pp.398-401 vol.2, 1998
   doi: 10.1109/TENCON.1998.798187
   URL:
   http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=798187&isnumber=17304

8. Singh, B.; Kasal, G.K.; Chandra, A.; Al Haddad, K.; , "Voltage and frequency controller for an autonomous micro hydro generating system," Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE , vol., no., pp.1-9, 20-24 July 2008 doi: 10.1109/PES.2008.4596127
   URL:
   http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4596127&isnumber=4595968
9. Henderson, D.S.; , "Synchronous or induction generators? The choice for small scale generation," Opportunities and Advances in International Electric Power Generation, International Conference on (Conf. Publ. No. 419) , vol., no., pp.146-149, 18-20 Mar 1996 doi: 10.1049/cp:19960137
   URL:
   http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=643462&isnumber=14025
10. Greg, Welch, & Gary, Bishop (2006). "An Introduction to the Kalman Filter", 1.
    URL:
    http://www.google.com/url?sa=t&source=web&cd=4&sqi=2&ved=0CDYQFjAD&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.79.6578%26rep%3Drep1%26type%3Dpdf&rct=j&q=Kalman%20filter&ei=bNT9TP_qB8yUnQegpIHSCQ&usg=AFQjCNGAd51Re2xzuzSTB-ae9NNQa-O0uA
11. Freescale, "MC9S08QG8, MC9S08QG4 Datasheet", HCS08 Microcontrollers, MC9S08QG8, Rev. 4, 2/2008.
    URL: www.freescale.com
12. Xilinx, "Embedded System, Reference Guide, EDK 11.3.1", UG111, September 16, 2009.
    URL: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/est_rm.pdf
13. Xilinx, "Virtex–4 FPGA, User Guide", UG070 (v2.6), December 1, 2008.
    URL: http://www.xilinx.com/support/documentation/user_guides/ug070.pdf

Appendix A - Hardware and Software Codes

This appendix includes codes for hardware setup of some IP's in the Xilinx microprocessor chip and as well as software for using the hardware and implementing the control algorithm.

HDL Verilog Code for Hardware

This part contains the Hardware Description Language to setup different IP's used in the project.

Code for the Encoder Monitor

Although Encoder Monitor is not used anymore in the project, the code is included as a matter of instrumentation and for the sake of completeness.

```
//----------------------------------------------------------------------------
// user_logic.vhd - module
//----------------------------------------------------------------------------
//
// ***************************************************************************
// ** Copyright (c) 1995-2009 Xilinx, Inc.  All rights reserved.            **
// **                                                                       **
// ** Xilinx, Inc.                                                          **
// ** XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         **
// ** AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND       **
// ** SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,        **
// ** OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,        **
// ** APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION           **
// ** THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,     **
// ** AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      **
// ** FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              **
// ** WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               **
// ** IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        **
// ** REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       **
// ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       **
// ** FOR A PARTICULAR PURPOSE.                                             **
// **                                                                       **
// ***************************************************************************
```

```
//
//----------------------------------------------------------------------------
// Filename:          user_logic.vhd
// Version:           1.00.a
// Description:       User logic module.
// Date:              Mon Sep 27 11:32:57 2010 (by Create and Import Peripheral
Wizard)
// Verilog Standard:  Verilog-2001
//----------------------------------------------------------------------------
// Naming Conventions:
//   active low signals:                    "*_n"
//   clock signals:                         "clk", "clk_div#", "clk_#x"
//   reset signals:                         "rst", "rst_n"
//   generics:                              "C_*"
//   user defined types:                    "*_TYPE"
//   state machine next state:              "*_ns"
//   state machine current state:           "*_cs"
//   combinatorial signals:                 "*_com"
//   pipelined or register delay signals:   "*_d#"
//   counter signals:                       "*cnt*"
//   clock enable signals:                  "*_ce"
//   internal version of output port:       "*_i"
//   device pins:                           "*_pin"
//   ports:                                 "- Names begin with Uppercase"
//   processes:                             "*_PROCESS"
//   component instantiations:              "<ENTITY_>I_<#|FUNC>"
//----------------------------------------------------------------------------

module user_logic
(
  // -- ADD USER PORTS BELOW THIS LINE ---------------
  // --USER ports added here
  TimRefer,
  A,
  B,
  // -- ADD USER PORTS ABOVE THIS LINE ---------------

  // -- DO NOT EDIT BELOW THIS LINE ------------------
  // -- Bus protocol ports, do not add to or delete
  Bus2IP_Clk,                     // Bus to IP clock
  Bus2IP_Reset,                   // Bus to IP reset
  Bus2IP_Data,                    // Bus to IP data bus
```

```
  Bus2IP_BE,                    // Bus to IP byte enables
  Bus2IP_RdCE,                  // Bus to IP read chip enable
  Bus2IP_WrCE,                  // Bus to IP write chip enable
  IP2Bus_Data,                  // IP to Bus data bus
  IP2Bus_RdAck,                 // IP to Bus read transfer acknowledgement
  IP2Bus_WrAck,                 // IP to Bus write transfer acknowledgement
  IP2Bus_Error                  // IP to Bus error response
  // -- DO NOT EDIT ABOVE THIS LINE ------------------
); // user_logic

// -- ADD USER PARAMETERS BELOW THIS LINE ------------
// --USER parameters added here
// -- ADD USER PARAMETERS ABOVE THIS LINE ------------

// -- DO NOT EDIT BELOW THIS LINE --------------------
// -- Bus protocol parameters, do not add to or delete
parameter C_SLV_DWIDTH                  = 32;
parameter C_NUM_REG                     = 5;
// -- DO NOT EDIT ABOVE THIS LINE --------------------

// -- ADD USER PORTS BELOW THIS LINE -----------------
// --USER ports added here
input     [0 : 63]                       TimRefer;   //new
input
       A;          //new
input
       B;          //new
// -- ADD USER PORTS ABOVE THIS LINE -----------------

// -- DO NOT EDIT BELOW THIS LINE --------------------
// -- Bus protocol ports, do not add to or delete
input                                    Bus2IP_Clk;
input                                    Bus2IP_Reset;
input      [0 : C_SLV_DWIDTH-1]          Bus2IP_Data;
input      [0 : C_SLV_DWIDTH/8-1]        Bus2IP_BE;
input      [0 : C_NUM_REG-1]             Bus2IP_RdCE;
input      [0 : C_NUM_REG-1]             Bus2IP_WrCE;
output     [0 : C_SLV_DWIDTH-1]          IP2Bus_Data;
output                                   IP2Bus_RdAck;
output                                   IP2Bus_WrAck;
output                                   IP2Bus_Error;
// -- DO NOT EDIT ABOVE THIS LINE --------------------
```

```
//----------------------------------------------------------------------------
// Implementation
//----------------------------------------------------------------------------

  // --USER nets declarations added here, as needed for user logic
  reg                  [0 : 63]                        Decode;
  reg                  [0 : 63]                        TR;
  reg                  [0 : 31]                        Decode1;
  reg                  [0 : 31]                        Temp;
  reg                  [0 : 31]                        Temp2;
  reg                  [0 : 2]                         delayA;
  reg                  [0 : 2]                         delayB;
  reg                                                  OutA;
  reg                                                  OutB;
  reg                                                  NewA;
  reg                                                  NewB;
  reg                  [0 : 64]                        FiFoOut;
  reg                                                  EMT;
  reg                                                  FW;
  reg                                                  STEP;
  wire                                                 Stp1;
  wire                                                 FB1;
  wire                [0 : 64]                         FiFo_Out;
  wire                                                 empty;
  wire                                                 Step;
  wire                                                 Up;
  wire                                                 full;

  // Nets for user logic slave model s/w accessible register example
  reg         [0 : C_SLV_DWIDTH-1]           slv_reg0;
  reg         [0 : C_SLV_DWIDTH-1]           slv_reg1;
  reg         [0 : C_SLV_DWIDTH-1]           slv_reg2;
  reg         [0 : C_SLV_DWIDTH-1]           slv_reg3;
  reg         [0 : C_SLV_DWIDTH-1]           slv_reg4;
  wire        [0 : 4]                        slv_reg_write_sel;
  wire        [0 : 4]                        slv_reg_read_sel;
  reg         [0 : C_SLV_DWIDTH-1]           slv_ip2bus_data;
  wire                                       slv_read_ack;
  wire                                       slv_write_ack;
  integer                                    byte_index, bit_index;
```

```
// --USER logic implementation added here

// ------------------------------------------------------
// Example code to read/write user logic slave model s/w accessible registers
//
// Note:
// The example code presented here is to show you one way of reading/writing
// software accessible registers implemented in the user logic slave model.
// Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
// to one software accessible register by the top level template. For example,
// if you have four 32 bit software accessible registers in the user logic,
// you are basically operating on the following memory mapped registers:
//
//    Bus2IP_WrCE/Bus2IP_RdCE   Memory Mapped Register
//                     "1000"   C_BASEADDR + 0x0
//                     "0100"   C_BASEADDR + 0x4
//                     "0010"   C_BASEADDR + 0x8
//                     "0001"   C_BASEADDR + 0xC
//
// ------------------------------------------------------

assign
  slv_reg_write_sel = Bus2IP_WrCE[0:4],
  slv_reg_read_sel  = Bus2IP_RdCE[0:4],
  slv_write_ack     = Bus2IP_WrCE[0] || Bus2IP_WrCE[1] || Bus2IP_WrCE[2] ||
                      Bus2IP_WrCE[3] || Bus2IP_WrCE[4],
  slv_read_ack      = Bus2IP_RdCE[0] || Bus2IP_RdCE[1] || Bus2IP_RdCE[2] ||
                      Bus2IP_RdCE[3] || Bus2IP_RdCE[4];

assign Step         = (OutA ^ NewA) || (OutB ^ NewB);
assign Up           =  NewA^OutB;

// implement slave model register(s)
always @( posedge Bus2IP_Clk )
  begin: SLAVE_REG_WRITE_PROC

           delayA             <= {delayA[1:2],A};
           delayB             <= {delayB[1:2],B};
           OutA               <= (delayA==3'b111)? 1 : (delayA==0) ? 0 : OutA;
           OutB               <= (delayB==3'b111)? 1 : (delayB==0) ? 0 : OutB;
           NewA               <=  OutA;
           NewB               <=  OutB;
```

```
        Decode      <= (slv_reg_write_sel[0])? 0 :(Step && !Up)? Decode-1:
                          (Step && Up)? Decode+1 : Decode;

if ( Bus2IP_Reset == 1 )
  begin
    slv_reg0 <= 0;
    slv_reg1 <= 8'h03ff03ff;
    slv_reg2 <= 0;
    slv_reg3 <= 0;
    slv_reg4 <= 0;
  end
else
  case ( slv_reg_write_sel )
    5'b10000 :
      for ( byte_index = 0; byte_index <= (C_SLV_DWIDTH/8)-1; byte_index =
            byte_index+1 )
        if ( Bus2IP_BE[byte_index] == 1 )
          for ( bit_index = byte_index*8; bit_index <= byte_index*8+7; bit_index
                = bit_index+1 )
            slv_reg0[bit_index] <= Bus2IP_Data[bit_index];
    5'b01000 :
      for ( byte_index = 0; byte_index <= (C_SLV_DWIDTH/8)-1; byte_index =
            byte_index+1 )
        if ( Bus2IP_BE[byte_index] == 1 )
          for ( bit_index = byte_index*8; bit_index <= byte_index*8+7; bit_index
                = bit_index+1 )
            slv_reg1[bit_index] <= Bus2IP_Data[bit_index];
    5'b00100 :
      for ( byte_index = 0; byte_index <= (C_SLV_DWIDTH/8)-1; byte_index =
            byte_index+1 )
        if ( Bus2IP_BE[byte_index] == 1 )
          for ( bit_index = byte_index*8; bit_index <= byte_index*8+7; bit_index
                =bit_index+1 )
            slv_reg2[bit_index] <= Bus2IP_Data[bit_index];
    5'b00010 :
      for ( byte_index = 0; byte_index <= (C_SLV_DWIDTH/8)-1; byte_index =
            byte_index+1 )
        if ( Bus2IP_BE[byte_index] == 1 )
          for ( bit_index = byte_index*8; bit_index <= byte_index*8+7; bit_index
                = bit_index+1 )
            slv_reg3[bit_index] <= Bus2IP_Data[bit_index];
    5'b00001 :
```

```
          for ( byte_index = 0; byte_index <= (C_SLV_DWIDTH/8)-1; byte_index =
                byte_index+1 )
            if ( Bus2IP_BE[byte_index] == 1 )
              for ( bit_index = byte_index*8; bit_index <= byte_index*8+7; bit_index
                    = bit_index+1 )
                slv_reg4[bit_index] <= Bus2IP_Data[bit_index];
        default : ;
      endcase
    end // SLAVE_REG_WRITE_PROC

  // implement slave model register read mux
  always @( slv_reg_read_sel or slv_reg0 or slv_reg1 or slv_reg2 or slv_reg3 or
slv_reg4 )
    begin: SLAVE_REG_READ_PROC

      case ( slv_reg_read_sel )
               5'b10000 :
                    begin
                           slv_ip2bus_data <= Decode[32 : 63];     //register 0
                           Temp            <= Decode[0  : 31];
                    end
               5'b01000 : slv_ip2bus_data            <= Temp;
       //register 1
        5'b00100 : slv_ip2bus_data             <= slv_reg2;
       //register 2
               5'b00010 : slv_ip2bus_data      <= slv_reg3;
       //register 3
               5'b00001 : slv_ip2bus_data      <= slv_reg4;
       //register 4
        default : slv_ip2bus_data              <= 0;
      endcase

    end // SLAVE_REG_READ_PROC

  // ------------------------------------------------------------
  // Example code to drive IP to Bus signals
  // ------------------------------------------------------------

  assign IP2Bus_Data    = slv_ip2bus_data;
  assign IP2Bus_WrAck   = slv_write_ack;
  assign IP2Bus_RdAck   = slv_read_ack;
  assign IP2Bus_Error   = 0;
```

```
endmodule
//---------------------------------------------------------------------------------
```

Code for Zero Crossing IP

This part includes two subsystems:

The main "User Logic":

```
//----------------------------------------------------------------------------
// user_logic.vhd - module
//----------------------------------------------------------------------------
//
// ***************************************************************************
// ** Copyright (c) 1995-2009 Xilinx, Inc.  All rights reserved.            **
// **                                                                       **
// ** Xilinx, Inc.                                                          **
// ** XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         **
// ** AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND       **
// ** SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,        **
// ** OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,        **
// ** APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION           **
// ** THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,     **
// ** AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      **
// ** FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              **
// ** WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               **
// ** IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        **
// ** REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       **
// ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       **
// ** FOR A PARTICULAR PURPOSE.                                             **
// **                                                                       **
// ***************************************************************************
//
//----------------------------------------------------------------------------
// Filename:          user_logic.vhd
// Version:           1.00.a
// Description:       User logic module.
// Date:              Tue Nov 02 15:09:07 2010 (by Create and Import Peripheral
Wizard)
// Verilog Standard:  Verilog-2001
//----------------------------------------------------------------------------
```

```
// Naming Conventions:
//   active low signals:                    "*_n"
//   clock signals:                         "clk", "clk_div#", "clk_#x"
//   reset signals:                         "rst", "rst_n"
//   generics:                              "C_*"
//   user defined types:                    "*_TYPE"
//   state machine next state:              "*_ns"
//   state machine current state:           "*_cs"
//   combinatorial signals:                 "*_com"
//   pipelined or register delay signals:   "*_d#"
//   counter signals:                       "*cnt*"
//   clock enable signals:                  "*_ce"
//   internal version of output port:       "*_i"
//   device pins:                           "*_pin"
//   ports:                                 "- Names begin with Uppercase"
//   processes:                             "*_PROCESS"
//   component instantiations:              "<ENTITY_>I_<#|FUNC>"
//------------------------------------------------------------------------------

module user_logic
(
  // -- ADD USER PORTS BELOW THIS LINE ---------------
  // --USER ports added here
  PhA,
  PhB,
  PhC,
  ThyA,
  ThyB,
  ThyC,
  CS_Trig,
  CS_Data,
  // -- ADD USER PORTS ABOVE THIS LINE ---------------

  // -- DO NOT EDIT BELOW THIS LINE ------------------
  // -- Bus protocol ports, do not add to or delete
  Bus2IP_Clk,                   // Bus to IP clock
  Bus2IP_Reset,                 // Bus to IP reset
  Bus2IP_Data,                  // Bus to IP data bus
  Bus2IP_BE,                    // Bus to IP byte enables
  Bus2IP_RdCE,                  // Bus to IP read chip enable
  Bus2IP_WrCE,                  // Bus to IP write chip enable
  IP2Bus_Data,                  // IP to Bus data bus
```

```
  IP2Bus_RdAck,                    // IP to Bus read transfer acknowledgement
  IP2Bus_WrAck,                    // IP to Bus write transfer acknowledgement
  IP2Bus_Error                     // IP to Bus error response
  // -- DO NOT EDIT ABOVE THIS LINE ------------------
); // user_logic

// -- ADD USER PARAMETERS BELOW THIS LINE ------------
// --USER parameters added here
// -- ADD USER PARAMETERS ABOVE THIS LINE ------------

// -- DO NOT EDIT BELOW THIS LINE --------------------
// -- Bus protocol parameters, do not add to or delete
parameter C_SLV_DWIDTH                  = 32;
parameter C_NUM_REG                     = 1;
// -- DO NOT EDIT ABOVE THIS LINE --------------------

// -- ADD USER PORTS BELOW THIS LINE -----------------
// --USER ports added here
input                                                                      PhA;
input                                                                      PhB;
input                                                                      PhC;
output                                                                     ThyA;
output                                                                     ThyB;
output                                                                     ThyC;
output [0 :  7]                                                            CS_Trig;
output   [0 : 39]                                                          CS_Data;
// -- ADD USER PORTS ABOVE THIS LINE -----------------

// -- DO NOT EDIT BELOW THIS LINE --------------------
// -- Bus protocol ports, do not add to or delete
input                                   Bus2IP_Clk;
input                                   Bus2IP_Reset;
input       [0 : C_SLV_DWIDTH-1]        Bus2IP_Data;
input       [0 : C_SLV_DWIDTH/8-1]      Bus2IP_BE;
input       [0 : C_NUM_REG-1]           Bus2IP_RdCE;
input       [0 : C_NUM_REG-1]           Bus2IP_WrCE;
output      [0 : C_SLV_DWIDTH-1]        IP2Bus_Data;
output                                  IP2Bus_RdAck;
output                                  IP2Bus_WrAck;
output                                  IP2Bus_Error;
// -- DO NOT EDIT ABOVE THIS LINE --------------------
```

```
//----------------------------------------------------------------------------
// Implementation
//----------------------------------------------------------------------------

  // --USER nets declarations added here, as needed for user logic
  reg              [0 : 31]                              count;
  reg              [0 : 31]                              countA;
  reg              [0 : 31]                              countB;
  reg              [0 : 31]                              countC;
  reg              [0 : 31]                              reg1;
  reg              [0 : 31]                              FreqCntA;
  reg              [0 : 31]                              FreqCntB;
  reg              [0 : 31]                              FreqCntC;
  reg              [0 : 31]                              ThresA;
  reg              [0 : 31]                              ThresB;
  reg              [0 : 31]                              ThresC;
  reg              [0 :  2]                              ThysP;
  reg              [0 :  2]                              ThysN;
  reg                                                    PhA_old;
  reg                                                    PhB_old;
  reg                                                    PhC_old;
  wire            [0 : 31]                               count1;
  wire            [0 : 31]                               count2;
  wire            [0 : 31]                               count3;
  wire            [0 : 31]                               offset;
  wire                                                   resetA;
  wire                                                   resetB;
  wire                                                   resetC;
  wire                                                   PosEdA;
  wire                                                   PosEdB;
  wire                                                   PosEdC;
  wire                                                   NegEdA;
  wire                                                   NegEdB;
  wire                                                   NegEdC;
  wire                                                   FltA;
  wire                                                   FltB;
  wire                                                   FltC;

  // Nets for user logic slave model s/w accessible register example
  reg        [0 : C_SLV_DWIDTH-1]           slv_reg0;
  wire       [0 : 0]                        slv_reg_write_sel;
  wire       [0 : 0]                        slv_reg_read_sel;
```

```
reg        [0 : C_SLV_DWIDTH-1]          slv_ip2bus_data;
wire                                     slv_read_ack;
wire                                     slv_write_ack;
integer                                  byte_index, bit_index;

// --USER logic implementation added here

// ------------------------------------------------------
// Example code to read/write user logic slave model s/w accessible registers
//
// Note:
// The example code presented here is to show you one way of reading/writing
// software accessible registers implemented in the user logic slave model.
// Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
// to one software accessible register by the top level template. For example,
// if you have four 32 bit software accessible registers in the user logic,
// you are basically operating on the following memory mapped registers:
//
//    Bus2IP_WrCE/Bus2IP_RdCE   Memory Mapped Register
//                     "1000"   C_BASEADDR + 0x0
//                     "0100"   C_BASEADDR + 0x4
//                     "0010"   C_BASEADDR + 0x8
//                     "0001"   C_BASEADDR + 0xC
//
// ------------------------------------------------------

assign
  slv_reg_write_sel = Bus2IP_WrCE[0:0],
  slv_reg_read_sel  = Bus2IP_RdCE[0:0],
  slv_write_ack     = Bus2IP_WrCE[0],
  slv_read_ack      = Bus2IP_RdCE[0];

// implement slave model register(s)
always @( posedge Bus2IP_Clk )
  begin: SLAVE_REG_WRITE_PROC

           PhA_old      <= FltA;
           PhB_old      <= FltB;
           PhC_old      <= FltC;
           reg1         <= 35700;
           /*count      <= (PhA_old && ~PhA)?count1:(Bus2IP_Reset)?0:count;
           countA       <= (Bus2IP_Reset)?0:count1;
```

```
        countB        <= (Bus2IP_Reset)?0:count2;
        countC        <= (Bus2IP_Reset)?0:count3;*/
        FreqCntA <= (Bus2IP_Reset)?0:(PosEdA)?count1-reg1:FreqCntA;
        FreqCntB <= (Bus2IP_Reset)?0:(PosEdB)?count2-reg1:FreqCntB;
        FreqCntC <= (Bus2IP_Reset)?0:(PosEdC)?count3-reg1:FreqCntC;
        ThresA <= FreqCntA[0 : 30] + slv_reg0 + 40000;
        ThresB <= FreqCntB[0 : 30] + slv_reg0 + 40000;
        ThresC <= FreqCntC[0 : 30] + slv_reg0 + 40000;
        ThysP[2] <= (count1 == slv_reg0)?0:(count1 == slv_reg0  +
                     10000)?1:ThysP[2];
        ThysP[1] <= (count2 == slv_reg0)?0:(count2 == slv_reg0 +
                      10000)?1:ThysP[1];
        ThysP[0] <= (count3 == slv_reg0)?0:(count3 == slv_reg0 +
                      10000)?1:ThysP[0];
        ThysN[2] <= (count1 == ThresA)?0:(count1 == ThresA + 10000)?1:ThysN[2];
        ThysN[1] <= (count2 == ThresB)?0:(count2 == ThresB + 10000)?1:ThysN[1];
        ThysN[0] <= (count3 == ThresC)?0:(count3 == ThresC + 10000)?1:ThysN[0];
  if ( Bus2IP_Reset == 1 )
    begin
      slv_reg0 <= 0;
    end
  else
    case ( slv_reg_write_sel )
      1'b1 :
        for ( byte_index = 0; byte_index <= (C_SLV_DWIDTH/8)-1; byte_index =
               byte_index+1 )
          if ( Bus2IP_BE[byte_index] == 1 )
            for ( bit_index = byte_index*8; bit_index <= byte_index*8+7; bit_index
                  = bit_index+1 )
              slv_reg0[bit_index] <= Bus2IP_Data[bit_index];
      default : ;
    endcase

end // SLAVE_REG_WRITE_PROC

    /*assign resetA = PhA_old^PhA;
    assign resetB = PhB_old^PhB;
    assign resetC      = PhC_old^PhC;*/
    assign PosEdA      = FltA & ~PhA_old;
    assign PosEdB      = FltB & ~PhB_old;
    assign PosEdC      = FltC & ~PhC_old;
    assign NegEdA      = PhA_old & ~FltA;
```

```
      assign NegEdB      = PhB_old & ~FltB;
      assign NegEdC      = PhC_old & ~FltC;
      assign ThyA        = ThysP[2]&ThysN[2];
      assign ThyB        = ThysP[1]&ThysN[1];
      assign ThyC        = ThysP[0]&ThysN[0];
      assign offset = reg1;
      assign CS_Trig = {FltA, PosEdA, NegEdA, (count1 == ThresA), (count1 ==
                        slv_reg0), 0, 0, 0};
      assign CS_Data = {ThresA, CS_Trig};
     counter MyCounterA(.Clk(Bus2IP_Clk), .reset(Bus2IP_Reset), .PosEdg(PosEdA),
                        .offset(offset), .count(count1));
     counter MyCounterB(.Clk(Bus2IP_Clk), .reset(Bus2IP_Reset), .PosEdg(PosEdB),
                        .offset(offset), .count(count2));
     counter MyCounterC(.Clk(Bus2IP_Clk), .reset(Bus2IP_Reset), .PosEdg(PosEdC),
                        .offset(offset), .count(count3));

     debouncer MyDebounceA(.Clk(Bus2IP_Clk), .Reset(Bus2IP_Reset), .Phase(PhA),
                           .OutPhase(FltA));
     debouncer MyDebounceB(.Clk(Bus2IP_Clk), .Reset(Bus2IP_Reset), .Phase(PhB),
                           .OutPhase(FltB));
     debouncer MyDebounceC(.Clk(Bus2IP_Clk), .Reset(Bus2IP_Reset), .Phase(PhC),
                           .OutPhase(FltC));

// implement slave model register read mux
always @( slv_reg_read_sel or slv_reg0 )
  begin: SLAVE_REG_READ_PROC

    case ( slv_reg_read_sel )
      1'b1 : slv_ip2bus_data <= (slv_reg0 == 20000)?FreqCntA:(slv_reg0 ==
                                20001)?FreqCntB:FreqCntC;//slv_reg0;
     //1'b10 : slv_ip2bus_data <= ThyA;//slv_reg0;
      default : slv_ip2bus_data <= 0;
    endcase

  end // SLAVE_REG_READ_PROC

// ------------------------------------------------------------
// Example code to drive IP to Bus signals
// ------------------------------------------------------------

assign IP2Bus_Data     = slv_ip2bus_data;
assign IP2Bus_WrAck    = slv_write_ack;
```

```
  assign IP2Bus_RdAck   = slv_read_ack;
  assign IP2Bus_Error   = 0;

endmodule
```

The Counter:

```
module counter(Clk, reset, PosEdg, offset, count);

input Clk;
input reset;
input PosEdg;
input  [0 : 31] offset;
output [0 : 31] count;

reg    [0 : 31] increment;

       always @(posedge Clk)
              begin: counting
                     if (reset)
                            begin
                                   increment <= 0;
                            end
                     else
                            begin
                                   increment <= (PosEdg)? offset:increment + 1;
                            end
              end
       assign  count = increment;
endmodule
```

The Debouncer:

```
module debouncer(Clk, Reset, Phase, OutPhase);

input Clk;
input Reset;
input Phase;
output OutPhase;

parameter Max = 200;

reg [0 : 31] Counter;
reg                        regOutPhase;

      always @(posedge Clk)
            begin
                  if (Reset)
                        begin
                              Counter <= 0;
                              regOutPhase <=0;
                        end
                  else
                        begin
                              regOutPhase <= (Counter == Max)? 1: (Counter == 0)?
                                                0: regOutPhase;
                              Counter     <= (Phase & (Counter != Max))? Counter+1:
                                               (~Phase & (Counter != 0))? Counter-1:
                                                Counter;
                        end
            end
            assign OutPhase =  regOutPhase;
endmodule
```

User Configuration File

In addition to the above hardware setup code, there is a file where the hardware port mapping is done. That is the port mapping between FPGA board and the pins of the Xilinx chip. Below are the mapping needed the project:

```
Net em_0_Aa_pin LOC=W26 | IOSTANDARD = LVCMOS33;
Net em_0_Bb_pin LOC=Y26 | IOSTANDARD = LVCMOS33;
Net xps_gpio_0_GPIO_IO_O_pin<3> LOC=Y25 | IOSTANDARD = LVCMOS33;
Net xps_gpio_0_GPIO_IO_O_pin<2> LOC=AA26 | IOSTANDARD = LVCMOS33;
Net SpiAdcDac_MOSI_O_pin LOC=W25 | IOSTANDARD = LVCMOS33;
Net SpiAdcDac_MISO_I_pin LOC=AC24 | IOSTANDARD = LVCMOS33;
Net SpiAdcDac_SCK_O_pin LOC=AB24 | IOSTANDARD = LVCMOS33;
Net xps_gpio_0_GPIO_IO_O_pin<0> LOC=Y24 | IOSTANDARD = LVCMOS33;
Net xps_gpio_0_GPIO_IO_O_pin<1> LOC=AA23 | IOSTANDARD = LVCMOS33;
Net zero_crossing_0_Tri_A_pin LOC=AB25 | IOSTANDARD = LVCMOS33;
Net zero_crossing_0_Tri_B_pin LOC=AC23 | IOSTANDARD = LVCMOS33;
Net zero_crossing_0_Tri_C_pin LOC=AB26 | IOSTANDARD = LVCMOS33;
Net zero_crossing_0_PHA_pin LOC=AD23 | IOSTANDARD = LVCMOS33;
Net zero_crossing_0_PHB_pin LOC=AC26 | IOSTANDARD = LVCMOS33;
Net zero_crossing_0_PHC_pin LOC=AD26 | IOSTANDARD = LVCMOS33;
```

Codes needed as software

Here is the listing of the software codes used to program the hardware and implement the control algorithm.

C Code for Xilinx

This part includes the software that communicates with different IP's on the Xilinx platform. In addition, it implements the control algorithm.

```
/*
 *
 * Xilinx, Inc.
 * XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" AS A
 * COURTESY TO YOU.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION AS
 * ONE POSSIBLE   IMPLEMENTATION OF THIS FEATURE, APPLICATION OR
 * STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS IMPLEMENTATION
```

```
 * IS FREE FROM ANY CLAIMS OF INFRINGEMENT, AND YOU ARE RESPONSIBLE
 * FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE FOR YOUR IMPLEMENTATION
 * XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO
 * THE ADEQUACY OF THE IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO
 * ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE
 * FROM CLAIMS OF INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.
 */

/*
 * Xilinx EDK 11.1 EDK_L.29.1
 *
 * This file is a sample test application
 *
 * This application is intended to test and/or illustrate some
 * functionality of your system.  The contents of this file may
 * vary depending on the IP in your system and may use existing
 * IP driver functions.  These drivers will be generated in your
 * XPS project when you run the "Generate Libraries" menu item
 * in XPS.
 *
 * Your XPS project directory is at:
 *    C:\Suliman\Test2\
 */


// Located in: ppc405_0/include/xparameters.h
#include "xparameters.h"

#include "stdio.h"

#include "xbasic_types.h"
#include "xgpio.h"
#include "gpio_header.h"

#include "xspi.h"
#include "xspi_l.h"
#include "xtmrctr.h"
#include "xtmrctr_l.h"

#define NoEffect 0
#define ADC_ACT 4
```

```
#define ADC_DACT 12
#define OPEN 1
#define CLOSE 2
#define SHUTDOWN 3
#define NoLoad 7
#define LoadPHA 3
#define LoadPHB 5
#define LoadPHC 6
#define FreqCnt 2000000
#define DEVIATE   12500

//=====================================================
       long long Count, Count1, OldCount;
       int Count0, i, resultgen, resultheat, state, count1;
       int ThyA_Up, Sum = 0, Avg, i = 0, j = 1;
       float gen_result, heat_result;
       char ch;
       int PeriodCount[5] = {0, 0, 0, 0, 0};
       Xuint32 DataRead;
       //PeriodCount[1] = 0;

void printout();

int main (void) {


   print("-- Entering main() --\r\n");

   /*
    * Peripheral SelfTest will not be run for RS232_Uart
    * because it has been selected as the STDOUT device
    */

       XSpi Spi_Ptr; // System structures
       XGpio Gpio_Ptr;
       XGpio AdcMon_Ptr;
       XTmrCtr Tmr_Ptr;
       unsigned char Rd0Cmd = 0x84,
                     Rd1Cmd = 0xc4; // Commands for converting Channel 0 and 1
       unsigned char AdcRead0[2],
                     AdcRead1[2];
       short int Control = 0;
```

```
    int LongControl = 0;

    // Initialize Hardware
    print("\r\nADC Sampling Starting. Initialization.\r\n");
    XSpi_Initialize( &Spi_Ptr, XPAR_SPIADCDAC_DEVICE_ID );
    XGpio_Initialize( &Gpio_Ptr, XPAR_XPS_GPIO_0_DEVICE_ID );
    //XGpio_Initialize( &Gpio_Ptr, XPAR_ADCDACSELECT_DEVICE_ID );
    XTmrCtr_Initialize( &Tmr_Ptr, XPAR_TMRCTR_0_DEVICE_ID);

    //Set Gen IO
    XGpio_SetDataDirection( &Gpio_Ptr, 1, 0 ); // bits to output.
    XGpio_DiscreteWrite( &Gpio_Ptr, 1, NoEffect ); // ADC and DAC off and so
                                                     the open/close bits.

    //Set Timer parameters.
    XTmrCtr_mSetLoadReg( Tmr_Ptr.BaseAddress, 0, 12500 ); // Divide 100M to
                                                                   8K
    LongControl = XTC_CSR_ENABLE_TMR_MASK // Enable on timer 0
                | XTC_CSR_AUTO_RELOAD_MASK // Reloads Counter at time out.
                | XTC_CSR_INT_OCCURED_MASK // Reset Timer Overflow bit.
                | XTC_CSR_DOWN_COUNT_MASK; // Set as a down counter.
    XTmrCtr_mSetControlStatusReg( Tmr_Ptr.BaseAddress, 0, LongControl );

    // Set up SPI
    Control = XSP_CR_ENABLE_MASK /* System enable */
            | XSP_CR_MASTER_MODE_MASK; /* Enable master mode */
    XSpi_mSetControlReg( &Spi_Ptr, Control );

    // Clear out SPI Buffer, Not sure if this is necessary
    while( XSpi_mGetStatusReg( &Spi_Ptr ) & XSP_SR_RX_FULL_MASK )
    AdcRead0[0] = XIo_In8( Spi_Ptr.BaseAddr + XSP_DRR_OFFSET);
                // XSpi_mRecvByte( Spi_Ptr.BaseAddr );


{
   Xuint32 status;

   print("\r\nRunning GpioOutputExample() for LEDs_4Bit...\r\n");

   status = GpioOutputExample(XPAR_LEDS_4BIT_DEVICE_ID,4);

   if (status == 0) {
```

```
      print("GpioOutputExample PASSED.\r\n");
   }
   else {
      print("GpioOutputExample FAILED.\r\n");
   }
}

    {
   Xuint32 status;

   print("\r\nRunning GpioInputExample() for Push_Buttons_Position...\r\n");

   Xuint32 DataRead;

   status = GpioInputExample(XPAR_PUSH_BUTTONS_POSITION_DEVICE_ID,
                             &DataRead);

   if (status == 0) {
      xil_printf("GpioInputExample PASSED. Read data:0x%X\r\n", DataRead);
   }
   else {
      print("GpioInputExample FAILED.\r\n");
   }
}

    XIo_Out32( XPAR_EM_0_BASEADDR, 0 );
    XIo_Out32( XPAR_ZERO_CROSSING_0_BASEADDR, 750000      );

    while(1)
    {
           //for (i=0; i<1000; i++);

           /*print("Enter the command\r\n");
           ch = getchar();
           getchar();*/
           GpioInputExample(XPAR_PUSH_BUTTONS_POSITION_DEVICE_ID, &DataRead);
           if (DataRead == 1)
                  XIo_Out32( XPAR_ZERO_CROSSING_0_BASEADDR, 750000);
           if (DataRead == 4)
                  XIo_Out32( XPAR_ZERO_CROSSING_0_BASEADDR, 10000);
           if (DataRead == 2)
           {
```

```
                //XGpio_DiscreteWrite( &Gpio_Ptr, 1, OPEN );
                //print("Openning\r\n");
                //printout();
                state = OPEN;
        }
        else if (DataRead == 8)
        {
                //XGpio_DiscreteWrite( &Gpio_Ptr, 1, CLOSE );
                //print("Closing\r\n");
                //printout();
                state = CLOSE;
        }
        else if (DataRead == 16)
        {
                //XGpio_DiscreteWrite( &Gpio_Ptr, 1, SHUTDOWN );
                //print("Closing\r\n");
                //printout();
                state = SHUTDOWN;
        }
        else
        {
                //XGpio_DiscreteWrite( &Gpio_Ptr, 1, NoEffect );
                //printout();
                //state = NoEffect;
        }

        if( XTmrCtr_mHasEventOccurred( Tmr_Ptr.BaseAddress, 0) )
        {
                // Reset Timer
                XTmrCtr_mSetControlStatusReg( Tmr_Ptr.BaseAddress, 0,
                                              LongControl );

                // Read from ADC
                XGpio_DiscreteWrite( &Gpio_Ptr, 1, ADC_ACT|state );
                // Set bit 0 Low, activating ADC.
                // Send over command to read channel 0.
                XIo_Out8(Spi_Ptr.BaseAddr + XSP_DTR_OFFSET, (u8) Rd0Cmd );
                // Wait while Results reg empty
                while( XSpi_mGetStatusReg( &Spi_Ptr ) &
                        XSP_SR_RX_EMPTY_MASK );
                // Read back results.
                AdcRead0[0] = XIo_In8( Spi_Ptr.BaseAddr + XSP_DRR_OFFSET);
```

```
        // Read to clear SPI
XIo_Out8(Spi_Ptr.BaseAddr + XSP_DTR_OFFSET, 0 );
         // Send dummy to read second byte.
        // Wait while Results reg empty
while( XSpi_mGetStatusReg( &Spi_Ptr ) &
        XSP_SR_RX_EMPTY_MASK );
AdcRead0[1] = XIo_In8( Spi_Ptr.BaseAddr + XSP_DRR_OFFSET);

XGpio_DiscreteWrite( &Gpio_Ptr, 1, ADC_DACT|state );
        // Turn off ADC

resultheat = ( ( AdcRead0[0] << 8 ) & 0x0ff00 )
           | ( ( AdcRead0[1]      ) & 0x000ff );
heat_result = resultheat*4/65535;

        // Read from ADC
XGpio_DiscreteWrite( &Gpio_Ptr, 1, ADC_ACT|state );
        // Set bit 0 Low, activating ADC.

        // Send over command to read channel 1.
XIo_Out8(Spi_Ptr.BaseAddr + XSP_DTR_OFFSET, (u8) Rd1Cmd );
        // Wait while Results reg empty
while( XSpi_mGetStatusReg( &Spi_Ptr ) &
        XSP_SR_RX_EMPTY_MASK );
        // Read back results.
AdcRead1[0] = XIo_In8( Spi_Ptr.BaseAddr + XSP_DRR_OFFSET);
        // Read to clear SPI

XIo_Out8(Spi_Ptr.BaseAddr + XSP_DTR_OFFSET, 0 );
        // Send dummy to read second byte.

        // Wait while Results reg empty
while( XSpi_mGetStatusReg( &Spi_Ptr ) &
        XSP_SR_RX_EMPTY_MASK );
AdcRead1[1] = XIo_In8( Spi_Ptr.BaseAddr + XSP_DRR_OFFSET);

XGpio_DiscreteWrite( &Gpio_Ptr, 1, ADC_DACT|state );
        // Turn off ADC

resultgen = ( ( AdcRead1[0] << 8 ) & 0x0ff00 )
          | ( ( AdcRead1[1]      ) & 0x000ff );
gen_result = resultgen*4/65535;
```

```
                i++;

            } // if timer.

            PeriodCount[0] = XIo_In32(XPAR_ZERO_CROSSING_0_BASEADDR);
            XIo_Out32( XPAR_ZERO_CROSSING_0_BASEADDR, (int) (40*Avg/100) );
            if ((PeriodCount[0] != PeriodCount[1]))
            {
                PeriodCount[1] = PeriodCount[0];
                //if (PeriodCount[1] > 800000)
                    xil_printf("%d\r\n",PeriodCount[1]);
                Sum += PeriodCount[1];
                j++;
            }
            if (i == 2000)
            {
                Avg = (int)(Sum/j);
                print("0\r\n");
                i = 0;
                j = 1;
                Sum = 0;
                if (Avg > (FreqCnt + DEVIATE)) { print("Openning\r\n");
                   state = OPEN;
                   /*XGpio_DiscreteWrite( &Gpio_Ptr, 1, OPEN );*/}
                else if (Avg < (FreqCnt - DEVIATE)) { print("Closing\r\n");
                    state = CLOSE;}
                    //XGpio_DiscreteWrite( &Gpio_Ptr, 1, CLOSE );}
                else
                {
                    state = NoEffect;
                    //XGpio_DiscreteWrite( &Gpio_Ptr, 1, NoEffect );
                }
            }
        }

    {
       Xuint32 status;

       print("\r\nRunning GpioInputExample() for Push_Buttons_Position...\r\n");

       Xuint32 DataRead;
```

```
      status = GpioInputExample(XPAR_PUSH_BUTTONS_POSITION_DEVICE_ID,
                                &DataRead);

      if (status == 0) {
         xil_printf("GpioInputExample PASSED. Read data:0x%X\r\n", DataRead);
      }
      else {
         print("GpioInputExample FAILED.\r\n");
      }
   }

   print("-- Exiting main() --\r\n");
   return 0;
}

void printout()
{
       Count0 = XIo_In32(XPAR_EM_0_BASEADDR);
       count1 = XIo_In32(XPAR_EM_0_BASEADDR + 4);
       //OldCount = (Count1<<32) + Count0;
       //xil_printf("OldCount: %d\r\n",(int) OldCount);
       xil_printf("countH countL: %d %d\r\n",count1, Count0);
       xil_printf("ResultGen: %d\r\n",resultgen);
       xil_printf("ResultHeat: %d\r\n",resultheat);
       xil_printf("GpioInputExample PASSED. Read data:0x%X\r\n", DataRead);
       //printf("Gen_Result: %f\r\n",gen_result);
       //printf("Heat_Result: %f\r\n",heat_result);
}
```

Assembly Code for MCS08

This code is used in the MCS08 Microcontroller to control PWM signal based on the input received from Xilinx microprocessor.

```
; lab7shell.asm - time-of-day clock using MTIM ISR

         include 'mc9s08qg8.inc'

     org     $60

delay     ds.b 1
shtdn     ds.b 1
oldval    ds.b 2

       org    $e000

init
    lda  #$53
    sta  SOPT1           ;kill cop timer
    ldhx  #$260
    txs                  ;initialize stack pointer
    lda  NV_ICSTRM
    sta  ICSTRM          ;trim internal oscillator
    ldhx #1000
    initlp:              ; do until 2 ms expired to stabilize osc
      aix  #-1
      cphx #0
      bne  initlp
    clr  ICSC2           ;up freq to 8 MHz
    lda  #$f3
    sta  PTBDD
    sta  PTBPE
    lda  #$ff
    sta  PTAPE         ;calm all unused port pins

    mov  #8,MTIMCLK     ;use bus clock divided by 256 for 8 ms intervals
    mov  #249,MTIMMOD   ;make period exactly 8 ms
    mov  #$40,MTIMSC    ;start timer and enable MTIM interrupts
    mov  #4, delay
```

```
    clr  shtdn
    mov  #$0a,TPMSC     ; activate PWM with bus clock divided by 4
    mov  #$38,TPMC1SC   ; Edge aligned PWM Low-True Pulses (Set Output on compare)
                          with no interrupt enabled
    ldhx #$fff
    sthx TPMMOD
    ldhx  #$6ff
    sthx  TPMC1V  ;generates duty cycle (at 0x0000 the duty cycle will be 0%)

    cli                         ;Turn all enabled interrupts loose

mainlp ; do forever
 lda PTBD
 bra  mainlp

; mtimisr counts down from 125 (or 1000) to detect
;   the passing of one second.

mtimisr
     pshh
     pshx
     dbnz delay,here1
     mov  #4,delay
     lda PTBD
     and #$0c
       beq  here1
       cbeqa #$08, open
       cbeqa #$04, close
       cbeqa #$0c, shutdown
       bra   here1
    open:
        lda  #1
        cbeq shtdn,here3
        ldhx TPMC1V
        cphx #$dff
          beq  here1
        aix #1
        sthx  TPMC1V
        bra  here1
       here3:
          clr  shtdn
          ldhx #$7ff
```

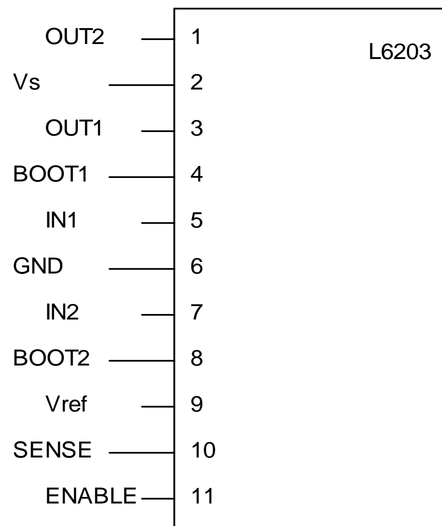
```
        sthx TPMC1V
        bra here1
  close:
      lda  #1
      cbeq shtdn,here2
      ldhx TPMC1V
      cphx #$1ff
        beq  here1
      aix #-1
      sthx  TPMC1V
      bra   here1
     here2:
        clr  shtdn
        ldhx #$2ff
        sthx TPMC1V
        bra here1
  shutdown:
      mov    #1,shtdn
      ldhx   #3
      sthx   TPMC1V
  here1:
    bclr MTIMSC_TOF,MTIMSC ; clear event flag
    pulx
    pulh
 rti

; end of mtimisr

     org  Vmtim
     dc.w mtimisr
     org  Vreset
     dc.w init

;    end  follows
 end
```

Appendix B - List of Pins for different chips

As a courtesy of the related data sheets, this Appendix lists the pin diagrams of the chips used in this report.

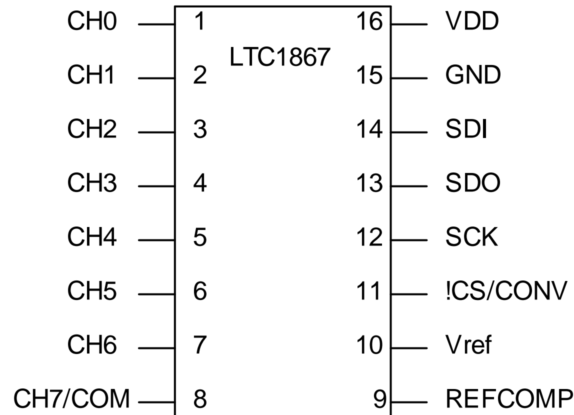
B1 - Pin details of the H-Bridge IC

Figure B.1 Pin details for the L6203 Chip



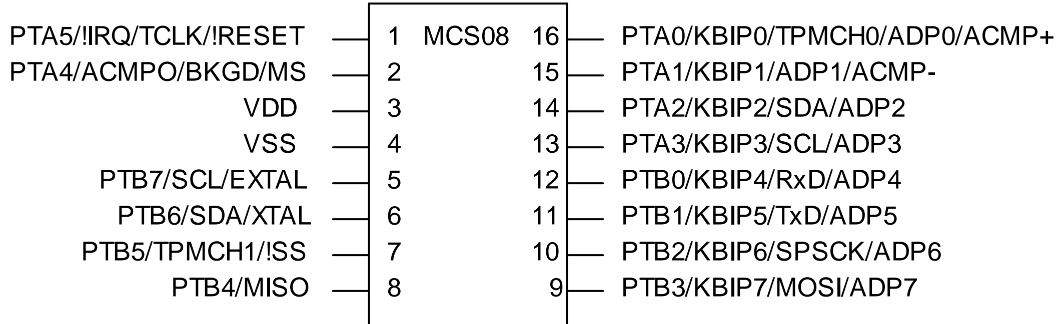
B2 – Pin details of the ADC Chip

Figure B.2 Pin details for the LTC1867 Chip



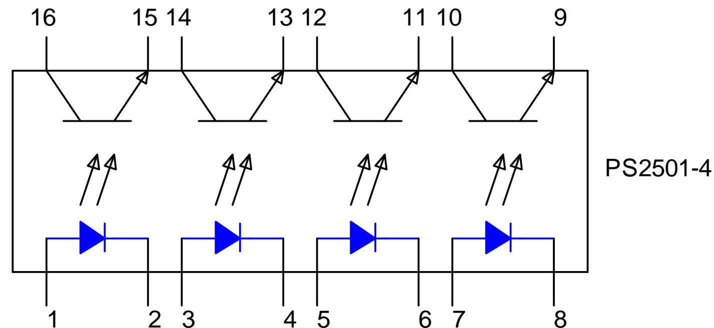
B3 – Pin details of the Microcontroller Chip

Figure B.3 Pin details for MCS08 Chip



B4 – Pin details of the Zero Crossing detector Chip

Figure B.4 Pin details for PS2501 – 4 Chip



B5 – Pin details of the Heater Connection Controller Chip

Figure B.5 Pin details for MOC3031 Chip