

Engine Control Workstation Using Simulink / DSP Platform

By

Mark Bright , Mike Donaldson

Advisor: Dr. Dempsey

An Engine Control Workstation was designed to simulate the thermal environments found in liquid-based cooling systems. The control workstation consists of a PC liquid cooling system, a motor-generator set for the engine, two Texas Instruments Control DSP boards, hardware interface circuitry, and a user friendly GUI for precision temperature and engine control. Software packages used for implementation include MATLAB, Simulink, Real-Time Workshop, and Code Composer Studio. These packages provide auto-code generation from a Simulink model of the control system and allow users to design, implement, and test controllers to more precisely regulate the thermal dissipation of the engine with the goal of reducing energy use.

Contents

Introduction

- 1.1 Introduction.....3
- 1.2 Functional Description.....3
- 1.3 Necessity of Liquid Cooling.....6

Cooling System Components

- 2.1 Overall Plant.....6
- 2.2 Motor/Generator Set.....7
- 2.3 Active Load.....7
- 2.4 Flow meter.....8
- 2.5 Reservoir/Pump.....8
- 2.6 Fan/Radiator9

Engine Side

- 3.1 Controller Design11
- 3.2 Simulink Modeling12

Thermal Side

- 4.1 Hardware Interfacing22
- 4.2 Simulink Modeling25
- 4.3 Simulink Implementation/Results31

Future Additions

- 5.1 Future Additions36

Appendices

- Appendix A Cooling System Components

1.1 Introduction

Cooling control systems used in vehicles have become more sophisticated in recent years due to demands for lower fuel or energy consumption. A thermostat has been the common control element but better accuracy, system speed, and minimization of fan and pump energy can be achieved with a proportional actuator. Multiple temperature sensors can also improve system performance.

An Engine Control Workstation will be designed to simulate the thermal environments found in liquid-based cooling systems. The workstation will allow users to design, test and implement controllers to more precisely regulate the thermal dissipation of a motor-generator system with the goal of reducing energy use. A user friendly GUI for temperature and engine control will be designed using MATLAB and Simulink software. Workstation controller and monitoring application software will be auto-code generated within the same program with the processing of the data being done on a DSP board.

1.2 Functional Description

The system will interface with the user via a GUI developed using Simulink and MATLAB. There will be several parameters that the user will be able to input and simulate, with the hopes of designing an engine due to the user's desired specifications. These inputs are, but not limited to: command velocity, controller parameters, controller types – proportional, integral, derivative, (P, PI, PID) and their perspective values for K_p , K_i , and K_d , and load changes to the system. Based upon these inputs, system identification can be performed to obtain a mathematical model of the system.

The developed closed-loop controller for velocity and acceleration will be active for load and no load conditions on the system. This will be implemented using auto-code generation and real-time control in Simulink interfaced with a fixed-point 32-bit TMS320F2812 DSP Evaluation board (See Figure 1-1). The outputs of the system are, but not limited to: output velocity, motor current, SS error, transient response – analyzing down to overshoot percentage, settling time, and time to first peak, PWM percentage, and controller signal. These outputs will be organized on a GUI that group these outputs in order to analyze desired specifications or continue design iterations. From a hardware level, the system will also require a DSP to motor interface utilizing similar design techniques as in the 2008 Mini Project. Also, a deliverable of this project is to explore all areas of Simulink and DSP Evaluation board interfacing and document controller limitations and performance. From this another goal was created, to minimize both C code and its execution time – down to an assembly level to ensure there are no possible real-time execution constraints created by the Simulink auto-code generation.



Figure 1-1 Texas Instruments 2812 DSP Board

Board was chosen for its easy integration into a Simulink model and capability of Auto-Code Generation. Two DSP Boards were used in the Workstation: one drives the engine controls while the other drives the thermal control.

The Thermo control portion of the project will focus on developing controllers for the pump and the fan of the system. Multiple types of controllers are to be developed and implemented. These controllers will then be used to control the temperature of the coolant and/or the plant to a specific value. In order to develop controllers to control temperature it is crucial to develop a transfer function and model of the thermo portion of the project by system identification.

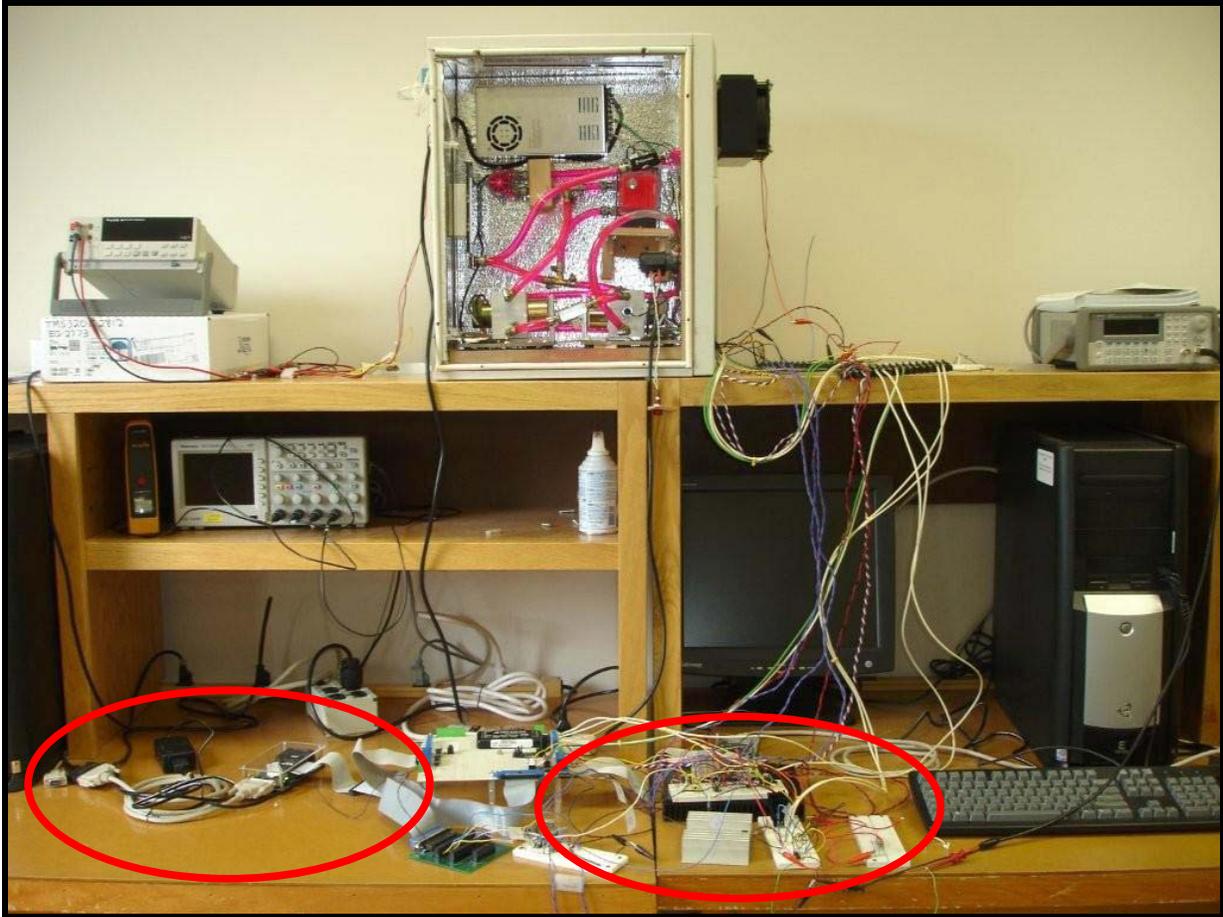
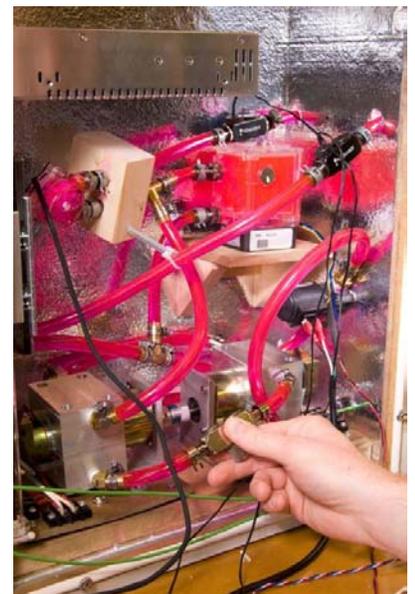


Figure 1-2 Overall System

Above is a picture of the system at the end of the project. Note: left side is engine control system, while the right side contains the thermal control system.

Figure 1-2 shows the overall integration of the plant and external hardware. The proposed engine control workstation will consist of a total of 5 subsystems: (1) motor-generator system to mimic an engine, (2) an active thermal load for the engine, (3) a liquid cooling system consisting of a pump, tank, radiator, fan, cooling block, flow meter, and multiple temperature sensors, (4) a fixed-point DSP evaluation board and software for controller implementation, and (5) interfaced electronics. A close up of the cooling system can be seen in Fig 1-3 and is further described by Fig 2-1 through Fig 2-5.



1.3 Necessity of liquid cooling

Fig 1-4 shows the thermal ratings for the Pittman DC motor that is being used in this project. The maximum temperature and thermal impedance ratings were taken from the datasheet. The current data was gathered by measuring from the motor when the generator was short-circuited for maximum load.

The resulting temperature far exceeds the maximum rating and shows the necessity of liquid cooling for this motor/generator set.

Pittman Motor

- ▶ **Tmax = 311 deg F**
- ▶ **Thermal impedance 75.9 deg F/watt**
- ▶ **$(2.9 \text{ A})^2 * (3.91 \text{ ohm}) = 36 \text{ watts}$**
- ▶ **$36\text{W} * (75.9 \text{ deg F/watt}) = 2732 \text{ deg F}$**

Figure 1-4 Thermal Comparison

2.1 Overall Plant

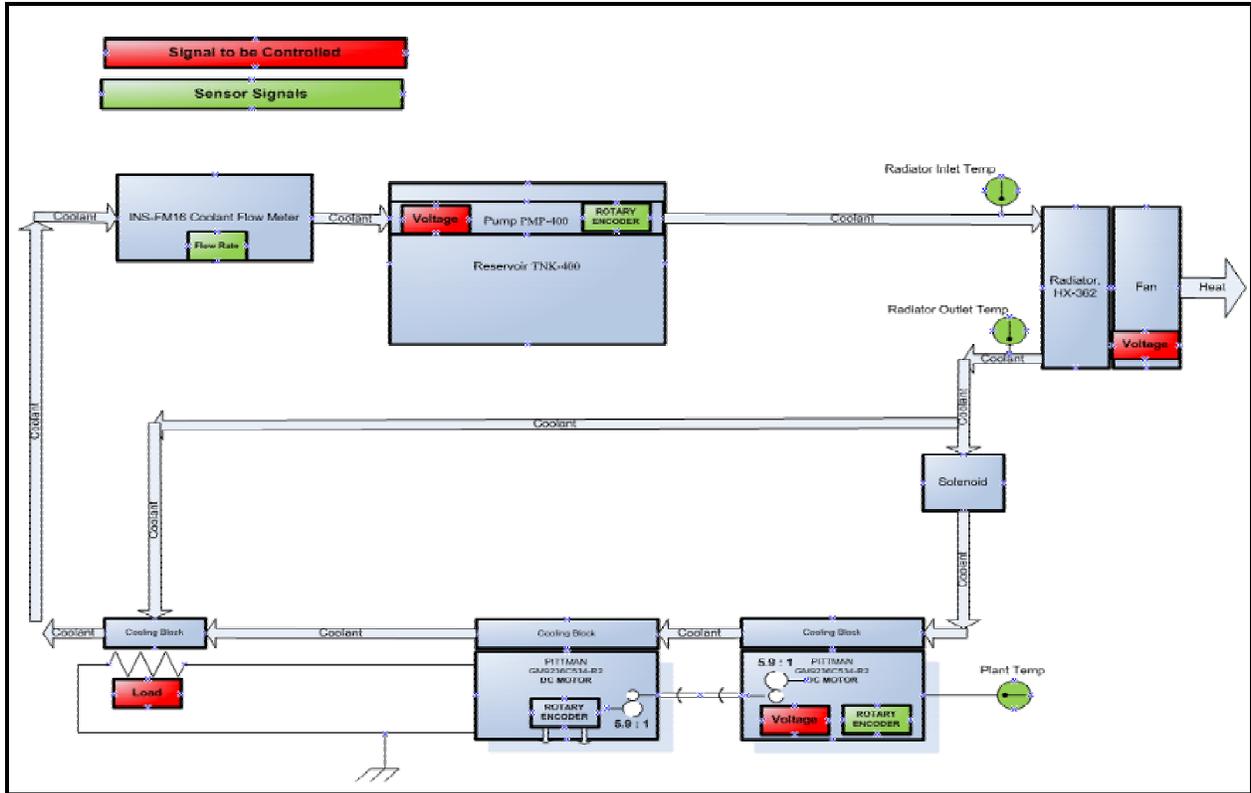


Figure 2-1 Plant System

2.2 Motor / Generator Set

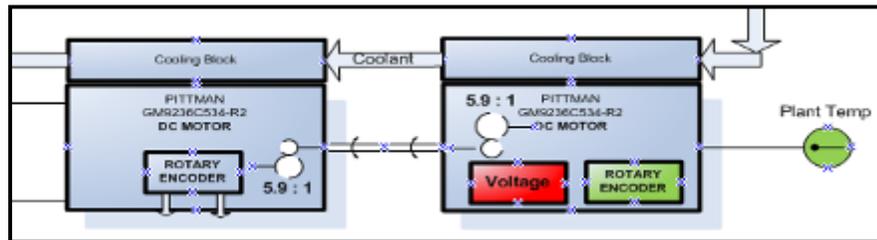


Figure 2-2 Motor/Generator Set

The motor/generator set consists of 2 DC Pittman motors as shown in Fig 2-2. One motor is arranged as a DC generator. The voltage to that motor will be controlled by the engine portion of the workstation. The rotary encoder is used to monitor the velocity. This motor is mechanically coupled to the generator. The generator's voltage terminals are connected to an active load to control the loading placed on the driving motor. Cooling blocks encompass the motor and generator to allow for use of liquid cooling. Mounted in the cooling block is a thermistor that measures the temperature of the plant.

2.3 Active Load

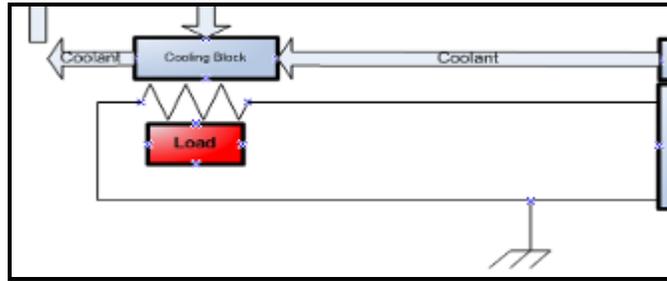


Figure 2-3 Active Load

The active load shown in Fig 2-3 is used to vary the load placed on the generator. This circuit uses a change in PWM duty cycle to vary the load. More detail on pg 28.

2.4 Flow meter

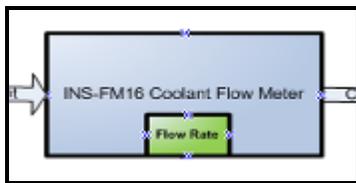


Figure 2-4.1 Flow meter

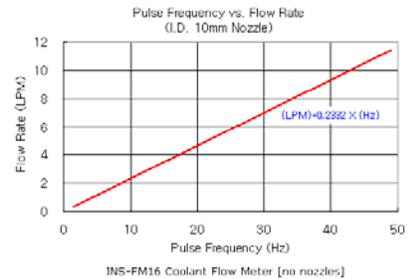


Figure 2-4.2 Flow meter Response

The flow meter shown in Fig 2.4.1 is a mechanical device that had a paddlewheel within the coolant flow. This device can be used to measure the flow rate within the system. The flow rate conversion is shown in Fig-2-4.2. The device was tested to be working but was not implemented into this system.

2.5 Pump / Reservoir

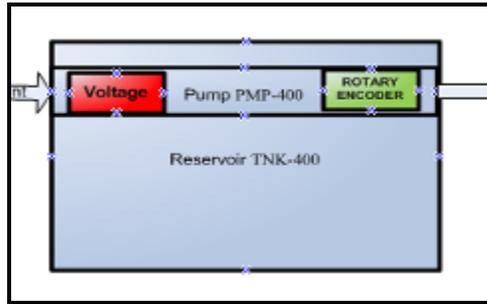


Figure 2-5 Pump/Reservoir

The pump, shown in Fig 2-5 is a brushless DC motor that is used to provide coolant flow to the cooling system. Voltage is manipulated to control the velocity of the pump. The pump is contained in a reservoir that houses extra coolant for the cooling system.

2.6 Fan/Radiator

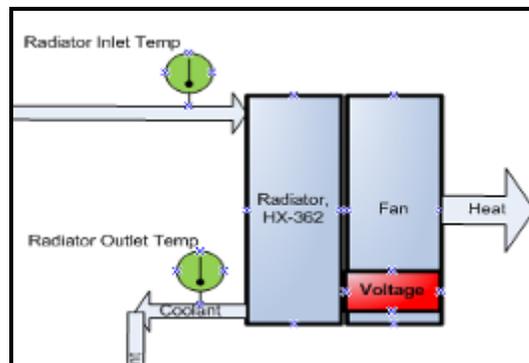


Figure 2-5 Radiator/Fan

Figure 2-5 shows the radiator and fan configuration. The coolant flows through the radiator which has numerous fins to transfer heat to the ambient environment. The fan increases the cooling effect by moving air across the radiator. There are two

thermistors here, one at the inlet and one at the outlet of the coolant flow of the radiator.

Engine Side

Project Goals

- (1) Learn software packages for auto-code generation and real-time control via Simulink/DSP interface. A major focus will be minimization of C-code and execution time.
- (2) Design DSP/motor hardware interface – will build off 2008 Mini Project
- (3) Design software for PWM generation and velocity calculation from rotary encoder.

- (4) Design closed-loop controllers for velocity and acceleration control.
- (5) Determine the limitations of the Simulink/TMS320F2812 DSP interface in terms of real-time execution and program memory.
- (6) Evaluate controller performance based on system accuracy, speed, and energy use.
- (7) Design Simulink/MATLAB GUI for controller parameter modification. This will include ability to graph critical output data for controller design evaluation.

Controller Design

As stated in project goals, controllers for both velocity and acceleration were designed. The implemented controllers were: proportional, proportional-integral, and feed forward. Using both the 2008 Mini-Project and both EE431 and 432 Control Theory class, design began. All controllers were designed in the s-plane, and then converted to the z-plane with the Bilinear Transformation function in MATLAB. The sampling rate in the models and within the implemented system was 1mS. The transfer function for the plant

was in the simulation model and had a system crossover frequency of 899 rad/sec, found in EE432 Design Project 1. The equation: $G_p = 17.1/(s/146+1)(s/776+1)$

The first implemented controller was Proportional control. As seen on the system model, the controller loop is closed by the summation of the command signal and the negative feedback implemented by the DSP board's Quadrature Encoder block (Fig 3-2). This error signal is then sent to a K_p value that was tuned to be .08, which resulted in a $e_{ss} = 75$ at 350 RPM step input. Once this was tested in simulation, it was assembled to the model and only needed a data conversion to work properly with the GUI – this was necessary, otherwise the GUI would freeze. Implementation into the GUI proved to be a great troubleshooting tool to tune all Gain values and achieve real-time results.

Once the GUI was working with proportional control, proportional-integral control was tested and implemented. The integral controller used in the Mini-Project was used. Again, as in the last controller, data type conversions were necessary. K_p was tuned once implemented into the system model and found to be .005 and $e_{ss} = 3$ at 350 RPM step input, the controller was now within the specified range for e_{ss} spec.

The next controller deliverable was to implement feed forward in order to decrease response time and control overshoot. From EE432 Control Theory class, a practical feed forward controller takes the general form of the inverse of the plant with the lowest plant pole placed a decade out from the zero. Also, to eliminate any gain here, it was compensated for by the inverse of the plant gain of 17.1, thus $1/17.1$. The placement of this controller is branched off the initial step input and runs in parallel to the proportional-integral. The idea is to let the proportional-integral controller take care of the steady state error, while the feed forward controller controls the acceleration.

Once tested for in the s-plane, significant improvement was seen in the motor response. As shown in Fig 3-6, for a 100 RPM step input, response time was decreased by 20mS and overshoot was eliminated, as well as time to first peak reduced to 25mS from 45mS. For conversion into the discrete world, the Bilinear Transformation function in MATLAB was used. Testing within the GUI showed similar results to simulation as shown in both the GUI and MATLAB axis in Fig 3-4 and Fig 3-5.

Simulink Modeling

The simulation model for Simulink started at the finishing point of the 2008 Mini-Project model. This model takes into account the engine model, as well as PWM conversion, system H-Bridge, and closed loop control via the quadrature encoder (Fig 3-3). The actual Simulink model used for the system included command conditioning, and closed-loop control, before sending signals to the PWM output (Fig 3-7). Data is also captured for the GUI from here, which adds an out-channel (ochan) to the system. These blocks consist of data buffers and conversions to allow for board to GUI communication. Once data is sent here, it can be referenced in the GUI code. Another element of the Simulink model is the DSP Board's Quadrature Encoder (QEP) at Port 8: Pins 6 and 7. This was implemented in the system to allow for our project specification of $e_{ss} = \pm 5$ RPM, as opposed to an $e_{ss} = \pm 20$ RPM if only one channel is implemented. The QEP block in Simulink brings in the motor's inner shaft velocity, this number was converted to outer shaft RPM for the GUI.

Also in the model were several data conversions. Some of the blocks in Simulink require specific data types in order to be used, for example the Discrete Transfer

Function block only allows for a double, so data conversion appears before and after this block. Trial and error was the main way to discover blocks of this nature. The starting point always came in knowing int32 was being sent out of the board. As detailed in the 'Controller Design' section, three types of control were achieved: proportional, proportional-integral, and feed forward. As seen on the model, these were implemented into the system in a parallel fashion in order to allow different controllers to be implemented within the GUI and no changes to the model are necessary. The user controller portion of the gain comes from an in-channel (ichan). Same as an ochan, these are used for DSP to GUI communication. Specifically ichan are GUI data to DSP motor commands. It is also of value to note that the DSP Board's blocks are in its own Simulink library – they are obtained in the software that the board was purchased with.

GUI Design

One of the main goals of this project was to design a nice Graphical User Interface (GUI) that would allow for real-time feedback of motor control, as well as user interaction for: input RPM, controller selection, and controller design. The MATAB GUI creator, Guide, is accessed by entering 'guide' into the command window. From here, several components can be added and interfaced in the accompanying .m file code including: buttons, sliders, axis, tables, menus, and ActiveX control. The implementation of this began with a template built into the development kit. This template was specifically designed for our target system because it included a basic Simulink model to run a motor from the 2812 DSP Board. Much time was spent with this model learning how auto-code generation from the GUI to Simulink to DSP Board was created. The

understanding can be aided by looking at Fig 3-1. Also note that auto-code generation only allows for the designed system to run on the board correctly and do not replace any design steps; the GUI .m code and Simulink models are where the system is really created.

The best development of the GUI came in small steps, just as in any other programming language. At several stages throughout the project, adding as little as one line of code resulted in a frozen GUI. This was pinpointed to two common problems: incorrect data types between the DSP Board and Simulink, or the MATLAB object handler in the GUI's object handler was not being used before being sent to grab another object. The data type problems occurred from trial and error troubleshooting. MATLAB GUI return parameters in several different data types – integer, strings, and doubles most commonly, these must be converted properly to and from data types within the code for the GUI to respond correctly. Also, one must keep note on what data type they are sending to GUI due to the heavy amount of data conversions required in the Simulink models, all due to block-specific data types. All in all, slow implementation is the best starting with remembering int32 (floating point processor) is coming out of the board.

The final version of the GUI, seen in Fig 3-8, allowed for several plots of control data, these include: Engine RPM, PWM duty cycle percentage, transient response, PI controller output, and feed forward controller output. These made use of the aforementioned ochan blocks from the Simulink model that allow for Board to GUI interaction. All data was plotted on a 0 to 5 sec axis over its operation range: RPM 0 to 834 RPM. Specifically transient response was the hardest to implement and was done

using counters inside the code that only allowed for a few cycles of code to be plotted, instead of constant real-time updating. From here, max RPM data is show in a text box, important for a percent overshoot calculation.

The most controls oriented part of the GUI lies in the adjustability of the systems control type. A user can interact with all three gain types (P, PI, and FF) via radio buttons to select their desired types, and use a slider to take the Kp gain to a range of 0 to 1 – adjustable to a .001 precision; this produces a noticeable range of variation in the system. With that, steady state error is displayed next to the current RPM and duty cycle percentage.

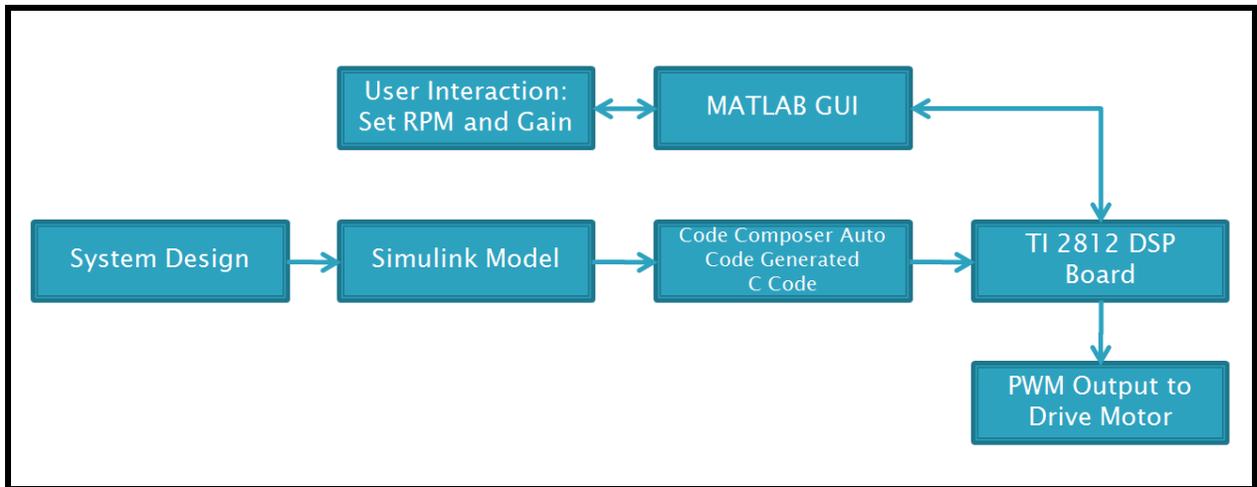


Figure 3-1 Engine Side Software Flowchart

The flowchart shows the role each piece of software and hardware plays within the engine system. Also the same for the thermal system, but has multiple outputs.

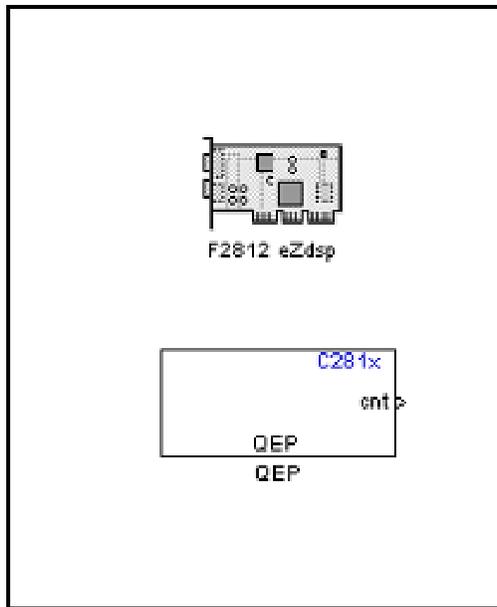


Figure 3 - 2 Quadrature Encoder Simulink Block

The QEP Simulink block brings in inner shaft RPM into the model. This value must be converted to outer shaft via a gain block. Available from the DSP 2812 Simulink library.

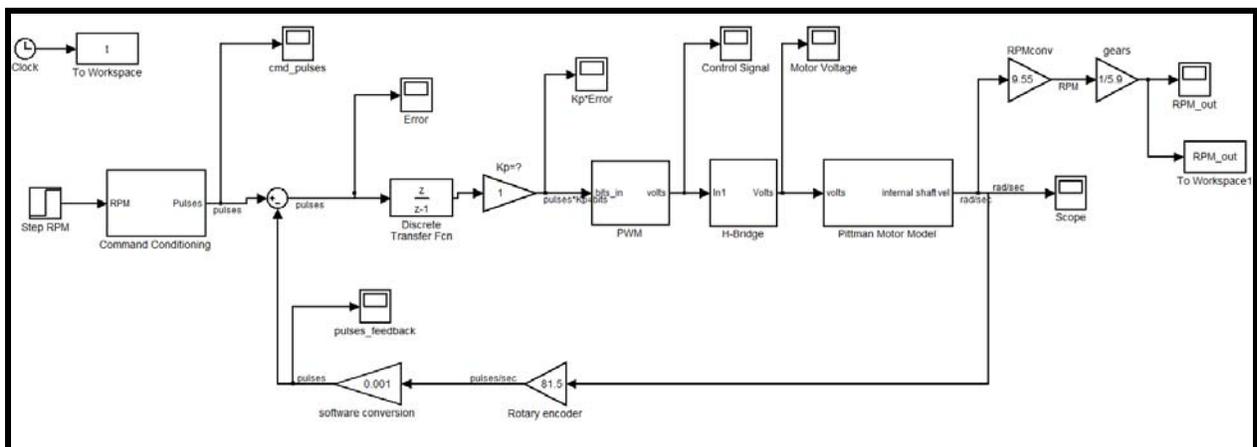


Figure 3 - 3 Simulation Simulink Model

Above is the Simulink model used in the 2008 Mini-Project. This was the starting point for controller design. All controllers were tested in both the s-plane and z-plane before being brought into the system Simulink model.

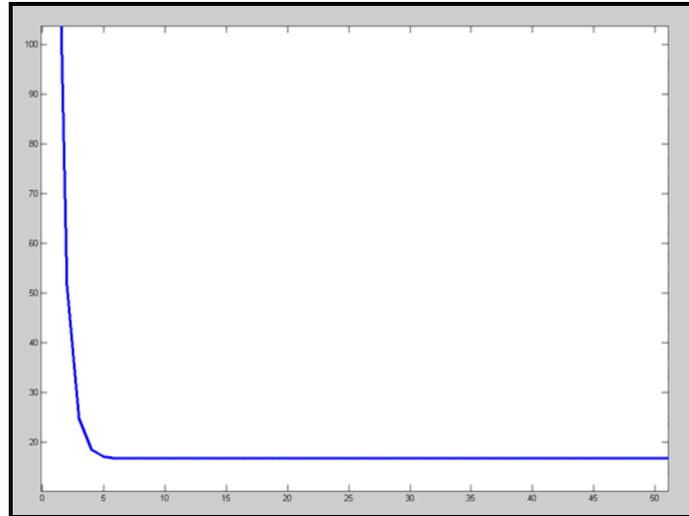


Figure 3-4 Simulation FF Controller Output

Axis: RPM vs. mS.

MATLAB Data gathered from the Feed Forward Controller Output during Simulation. There is an impulse with length of 2mS show with a steady-state value of 17 RPM.

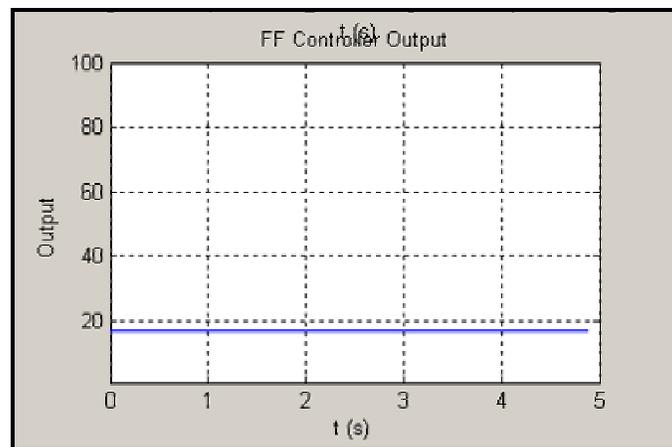


Figure 3-5 GUI FF Controller Output

Axis: RPM vs Sec.

GUI Data gathered from the Feed Forward Controller Output during Testing. The impulse is not seen here due to the 1mS sampling rate for the GUI. Steady-state value of 17 RPM.

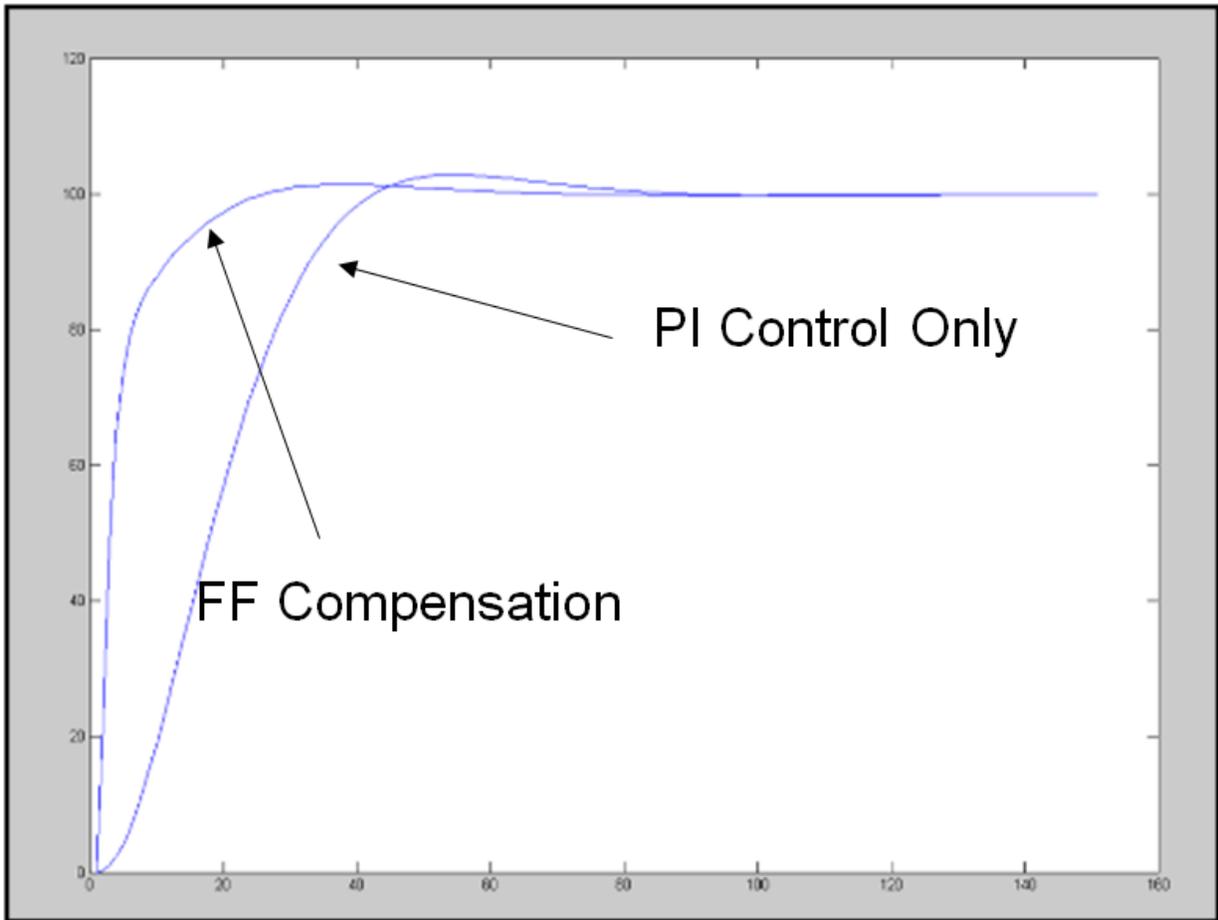


Figure 3-6 Controller Testing

Axis: RPM vs. mS. Input: 100 RPM Step command.

MATLAB Data gathered from Simulation of both FF and PI Control. Note: decreased response time, no overshoot, and smaller time to first peak for FF Controller.

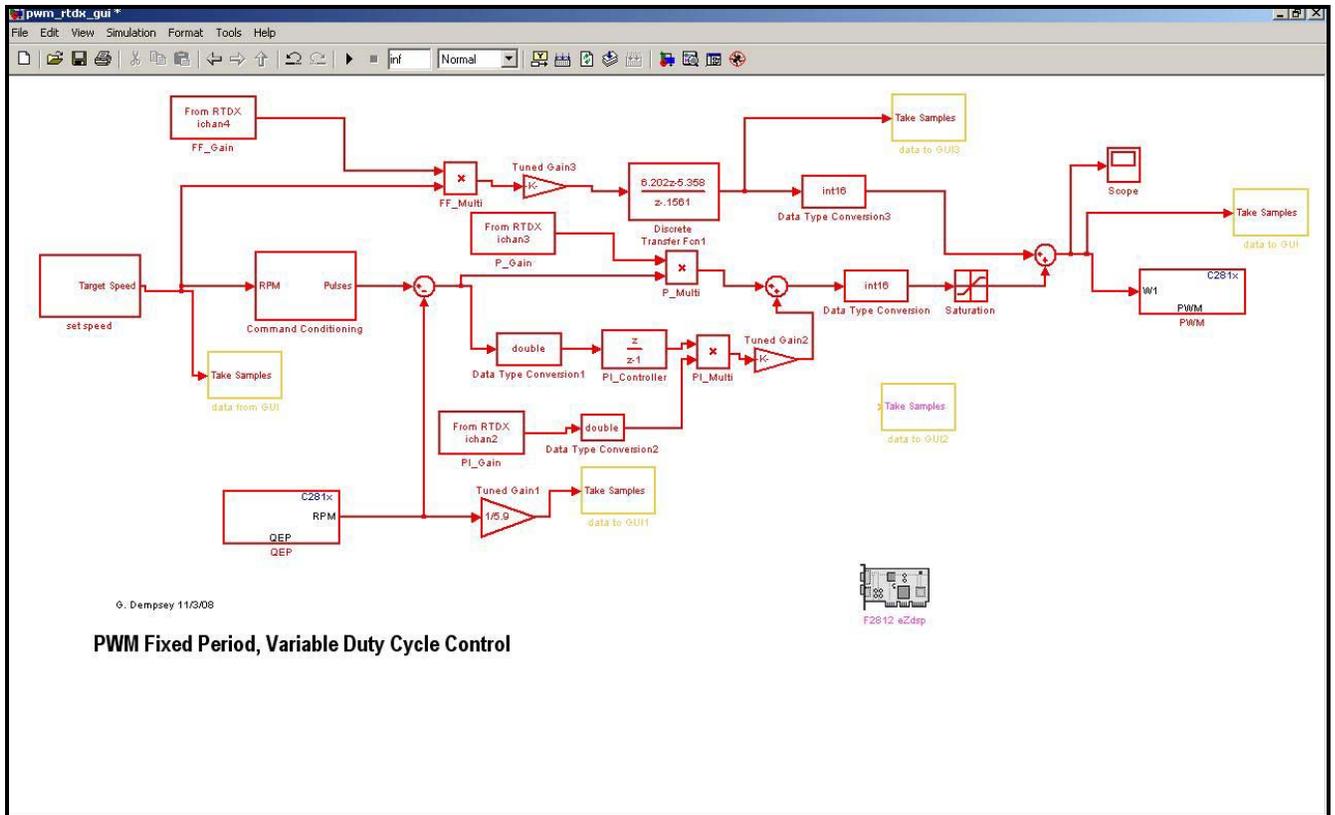


Figure 3-7 System Simulink Diagram

The Simulink model for the DSP Board contains three types of control and sends data to the GUI. All controllers are in the discrete world. QEP and PWM blocks are from the DSP 2812 Simulink library.

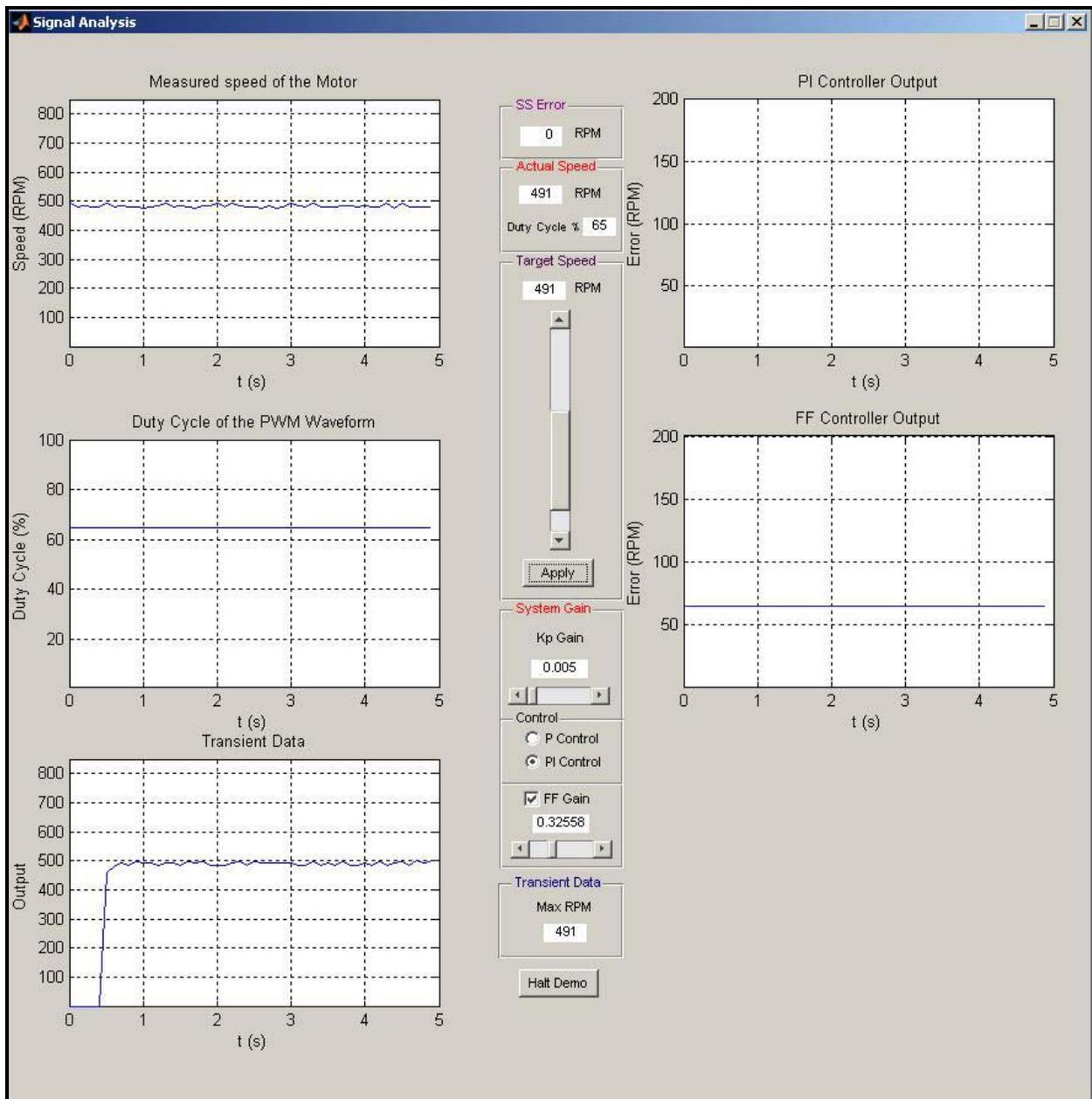


Figure 3-8 Final Engine Graphical User Interface

The final implemented GUI displays several control specifications and graphs crucial system data.

Thermal Side

Project Goals

- (1) Design closed-loop controller for temperature regulation of cooling system.
- (2) Design Simulink/MATLAB GUI for controller parameter modification. This will include ability to graph critical output data for controller design evaluation.
- (3) Design energy management control system in Simulink to regulate voltage/current to each subsystem based on its energy usage.
- (4) Evaluate controller performance based on system accuracy, speed, and energy use.
- (5) Determine the limitations of the Simulink/TMS320F2812 DSP interface in terms of real-time execution and program memory.

Functional Description

4.1 Hardware Interfacing

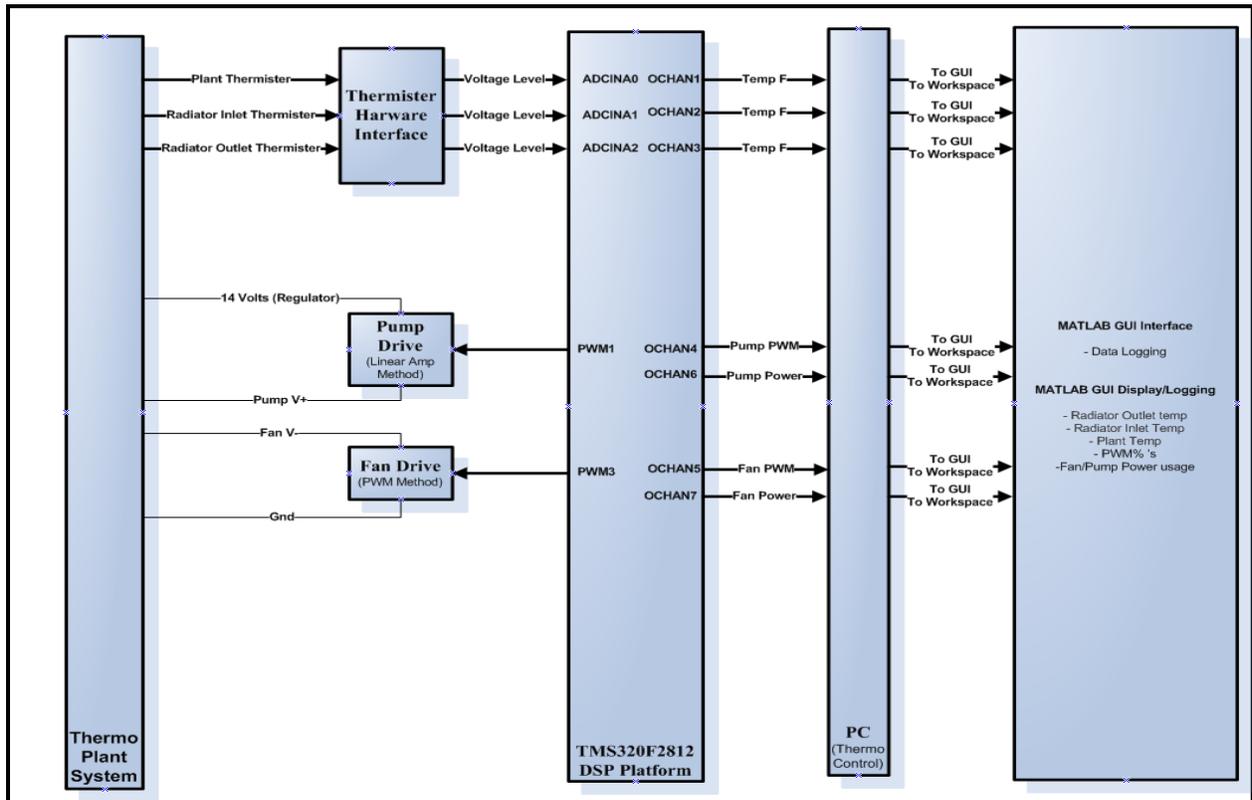


Figure 4-1 Hardware Interface Block Diagram

Thermistor Hardware Interface

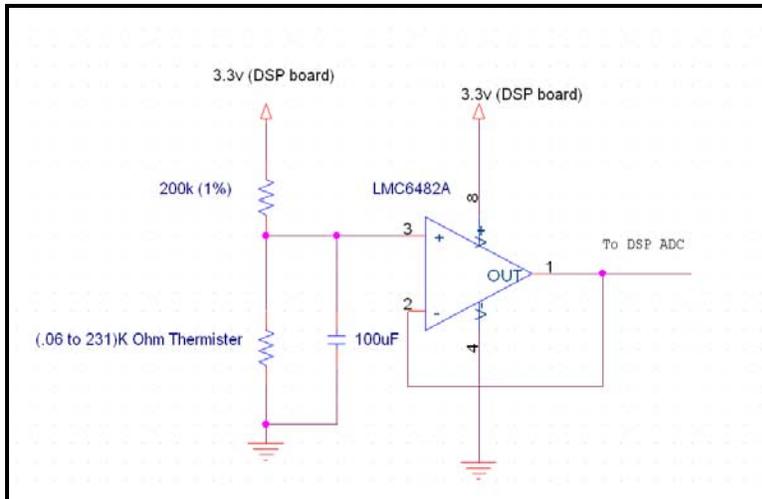


Figure 4-2 Thermistor Interfacing Circuit

The thermistor is a variable resistance that responds to changes in temperature. The thermistor is placed in series with a resistance to create a voltage divider. This voltage divider is fed into a LMC6482A operation amplifier which is used as a unity gain buffer. A capacitance is placed on the input of the LMC6482A to reduce the amount of aliasing/reduce noise. Due to the positioning of the capacitance the location of the filter pole moves as the thermistor resistance changes. The pole is located at $S = (200K + R_t) / (100\mu F * 200K * R_t)$. For 60 °F to 130 °F the pole will move from 29mHZ to 117mHZ respectively. Note that R_t does not change linearly with respect to temperature (see Fig ...)

Fan Drive Circuit

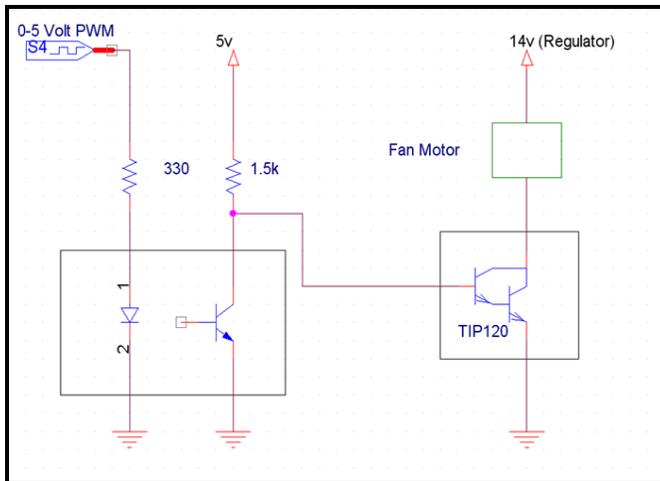


Figure 4-3 Fan Drive Circuit

The DSP board provides PWM outputs for use in velocity control. This signal is fed into a 4N25 opto-isolator. The opto-isolator is used to isolate the multiple grounds of the system to prevent electrostatic damage of the DSP board. The PWM leaves the opto-isolator (note the duty cycle is reversed) to be used with the TIP 120 Darlington pair power transistor. The TIP120 is used as a switch for full on/off operation based on a high/low of the PWM signal. The TIP120 is connected to the negative lead of the fan. The positive lead is connected to its voltage source from the 12-volt regulator.

Pump Drive Circuit

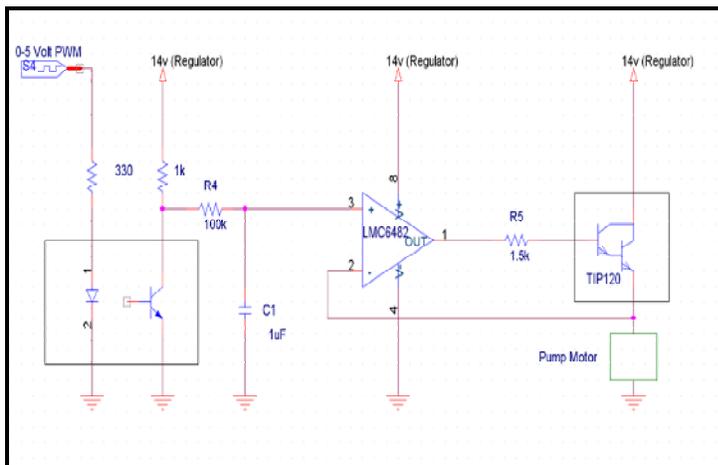


Figure 4-4 Pump Drive Circuit

The pump response was too slow to use PWM directly. The same opto-isolator setup is used for isolation purposes. The PWM is filtered (pole at 1.59Hz) to a voltage level. Control of pump velocity is achieved by controlling the voltage seen by the pump. The location of the feedback path for the operational amplifier allows for the voltage at the non-feedback input of

the operational amplifier to be the same voltage seen by the pump. With this setup we drive the TIP120 on the linear portions of its voltage/current curve instead of the on/off operation for the fan.

12 Volt Regulator Enhancement

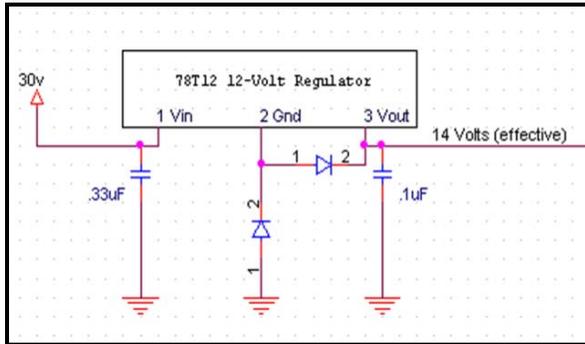


Figure 4-5 12 volt regulator circuit

The voltage output has been increased to compensate for the 13.5volt maximum of the fan/pump and to overcome the V_{ce} of the TIP120. Note the diode from Pin2 to Gnd is in the configuration for a zener diode.

Active Load

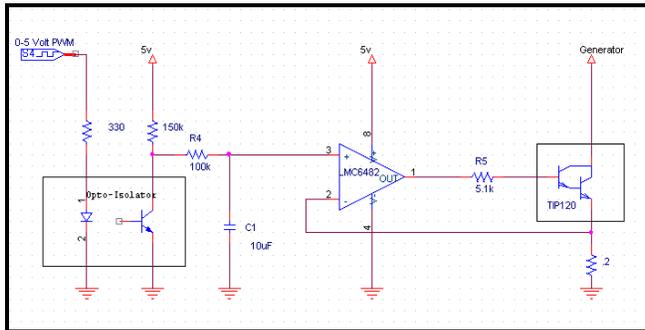


Figure 4-5.1 Active Load Circuit Load Performance

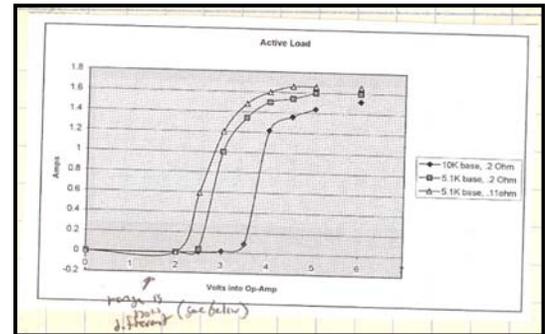


Figure 4-5.2 Active

The Active Load circuit (Fig 4-5.1) uses the same idea as the pump drive circuit. The opto-isolator is used to isolate the separate grounds. In Fig 4-5.2 a plot of voltage into the op-amp vs. the current allowed by the TIP120 for 3 sets of resistances. Poor grounding resulted in those curves being shifter to much lower voltages, 2 volts being the max value. With this in mind our PWM signal not only needs to be 'smoothed' into a voltage level but a voltage division needs to occur to be the range from 0 to 2

volts from the 5 volt regulator. The .11 ohm configuration was chosen since the 2 ohm resistor would not fit in the breadboard.

4.2 Simulink Modeling

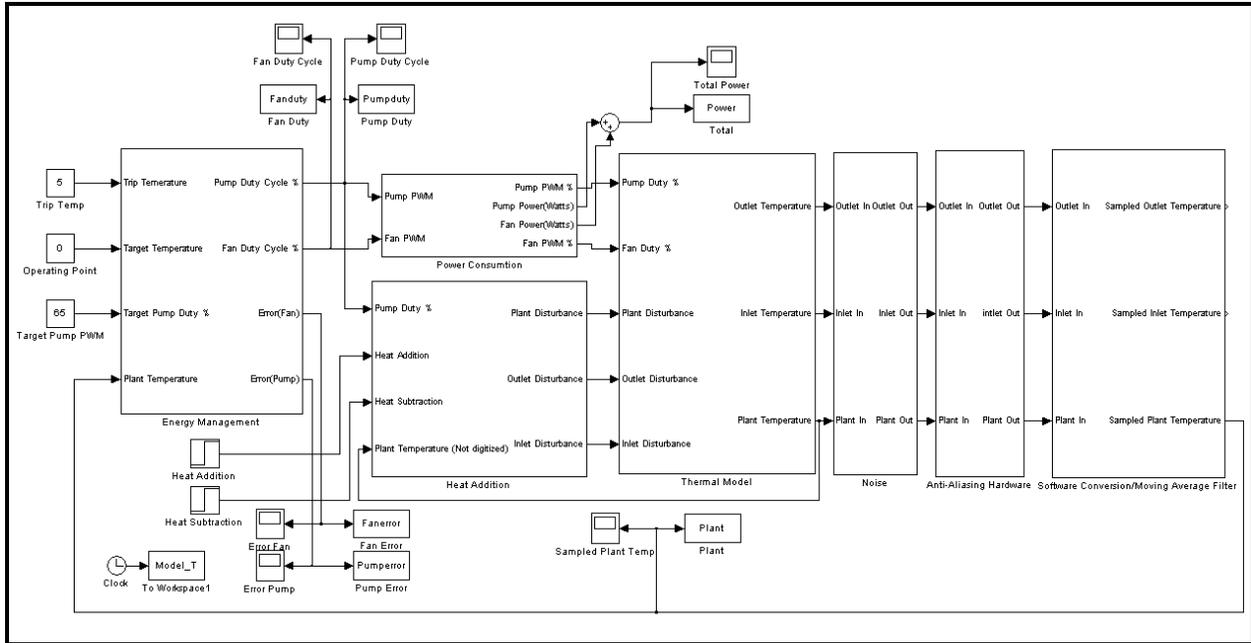


Figure 4-6 12 Overall Simulink Model for the Thermal System

Thermal Model

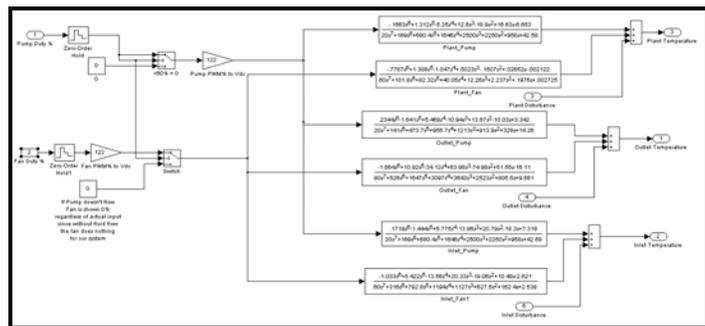


Figure 4-7 Thermal Model

The thermal model was developed using techniques from EE431. There are a total of six first order models with time delay, as seen in Fig 4-7. Three of the models are associated with the temperature response with respect to the pump, while the other

three model the response associated with the fan. Each of those model pairs is associated with the temperature of the inlet, outlet and plant thermistor.

One switch shown in the model is used to account for the pumps lack of ability to move fluid below a 60% PWM. The other switch turns off the temperature response of the fan when the pump in not flowing. If there is no fluid flow then the cooling ability of the fan will not be seen by the thermistor.

Noise

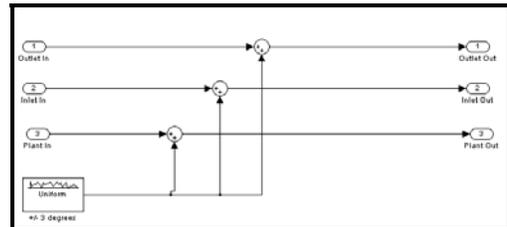


Figure 4-8 Noise Model

Initial data logging with no filtering has shown a constant +/- 3 degrees F due to noise levels. The noise block in Fig 4-8 simply models that behavior.

Anti-Aliasing hardware

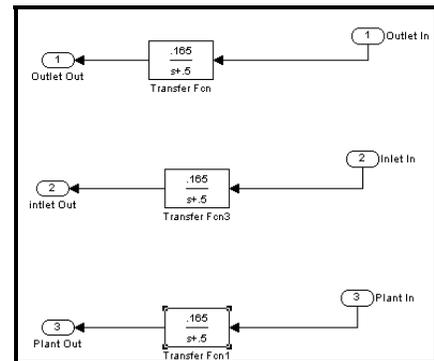


Figure 4-9 Anti-Aliasing Model

The Low-pass anti-aliasing filter is modeled and is shown in figure 4-9. Even though the pole location will move due to the variable resistance the pole location always remains extremely low. It has been decided to model it as a fixed location. The pole location seen above would be associated with a temperature of 90 degrees F.

Software Conversion and Moving Average

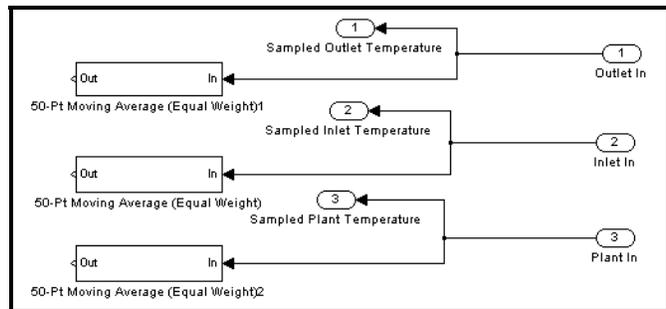


Figure 4-10 Software

Conversion Model

The software conversion block (Fig 4-10) models the conversion of the temperature data as it enters the DSP board. The ZOH block (which models sampling) is placed in front of the thermal model to add the effects of phase lag from the sampling. The moving average filtering of the GUI is also modeled here even though its output is not used in the control scheme of the energy management.

Energy Management

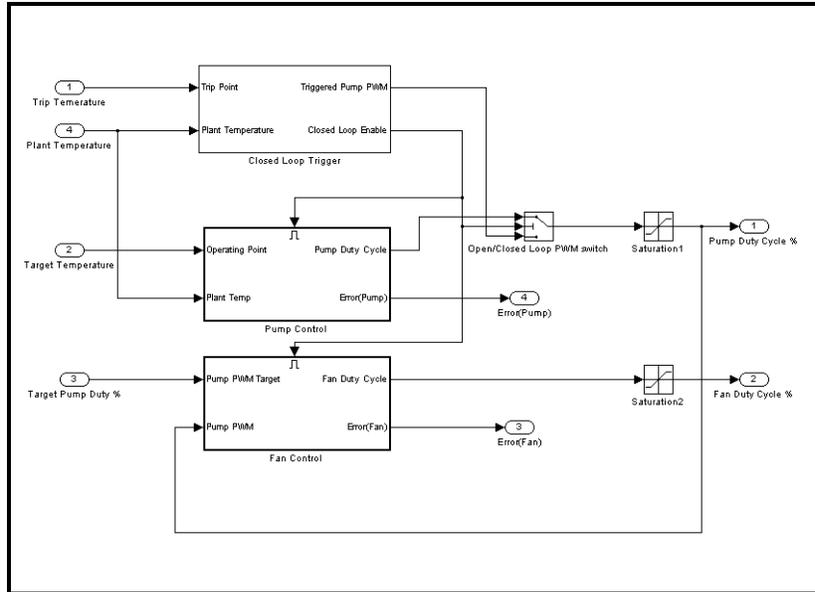


Figure 4-10.1 Energy Management

The Energy Management Block shown in Fig 4-10.1 is where the closed loop controllers for both the fan and pump velocity are contained. It was decided to have a trigger system upon initial turn on of the temperature control. A trip temperature is set (this system uses 5 degrees F above the operating point), once this trip temperature is seen at the outlet the trigger system begins to operate. The trigger system sends a 100% PWM duty cycle to the pump to circulate the fluid around the thermal system to move heat around the system that has been localized to the plat/generator. This system operates for 20 seconds before it sends a high-bit to the closed loop controllers for both the fan and pump. This high bit signifies to engage the operation of closed loop control. Fig 4-10.2 show the trigger and controller subsystems.

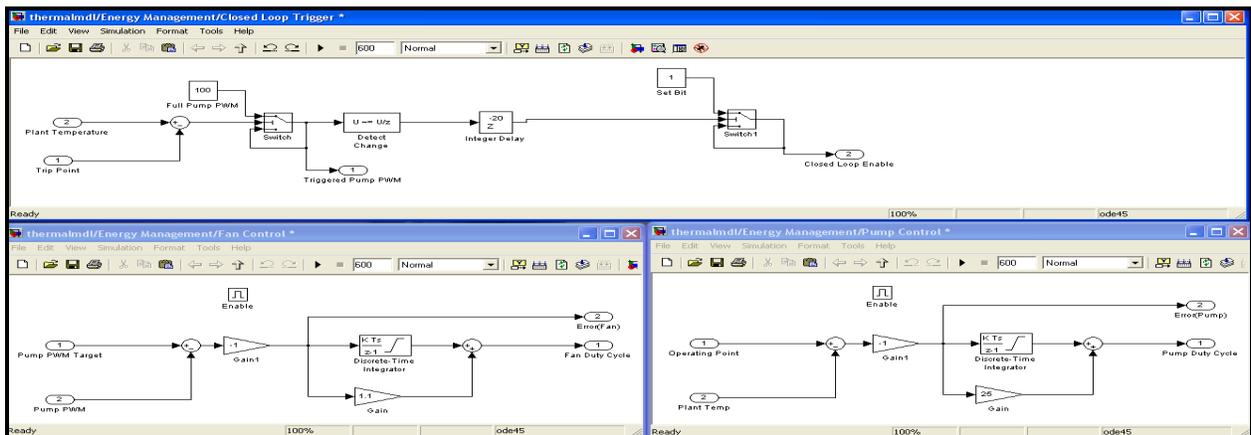


Figure 4-10.2 Trigger/Controller Subsystem

Power Calculation

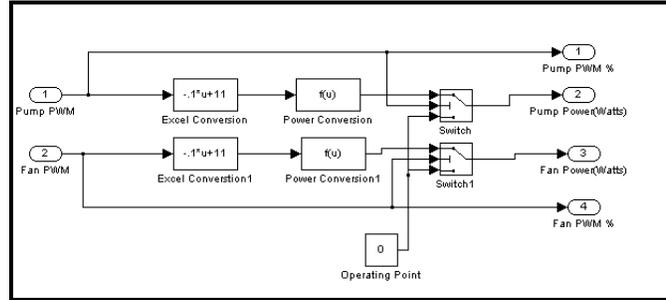


Figure 4-11 Power Conversion

The Power Conversion Block monitors the PWM for the fan/pump and calculates the power (in watts) for data-logging and display on the GUI interface.

Heat Addition

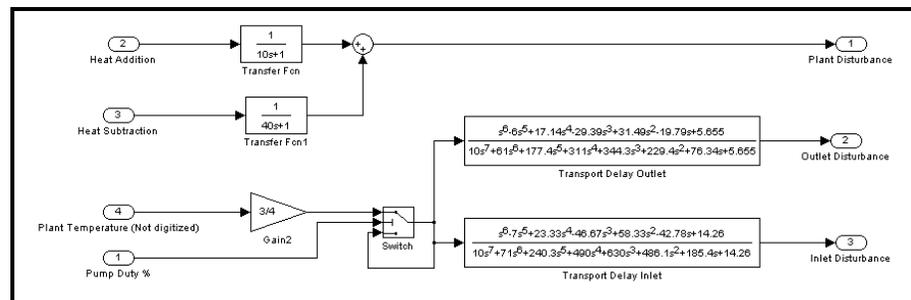


Figure 4-12 Disturbance Block

The Heat Addition model is used to test the disturbance rejection potential of the closed loop controllers. Heat is added/subtracted from the system as an exponential change. This is summed with the output of the thermal model of the plant. The plant's temperature is fed in and multiplied by a gain (3/4 to model the heat loss from the coolant path) then passed to a delay to model the coolant transport delay to the inlet and outlet.

This model is not a precise model of the heat generation of our system (time constraints) but it does accurately portray the temperature disturbance process.

Model vs. Actual

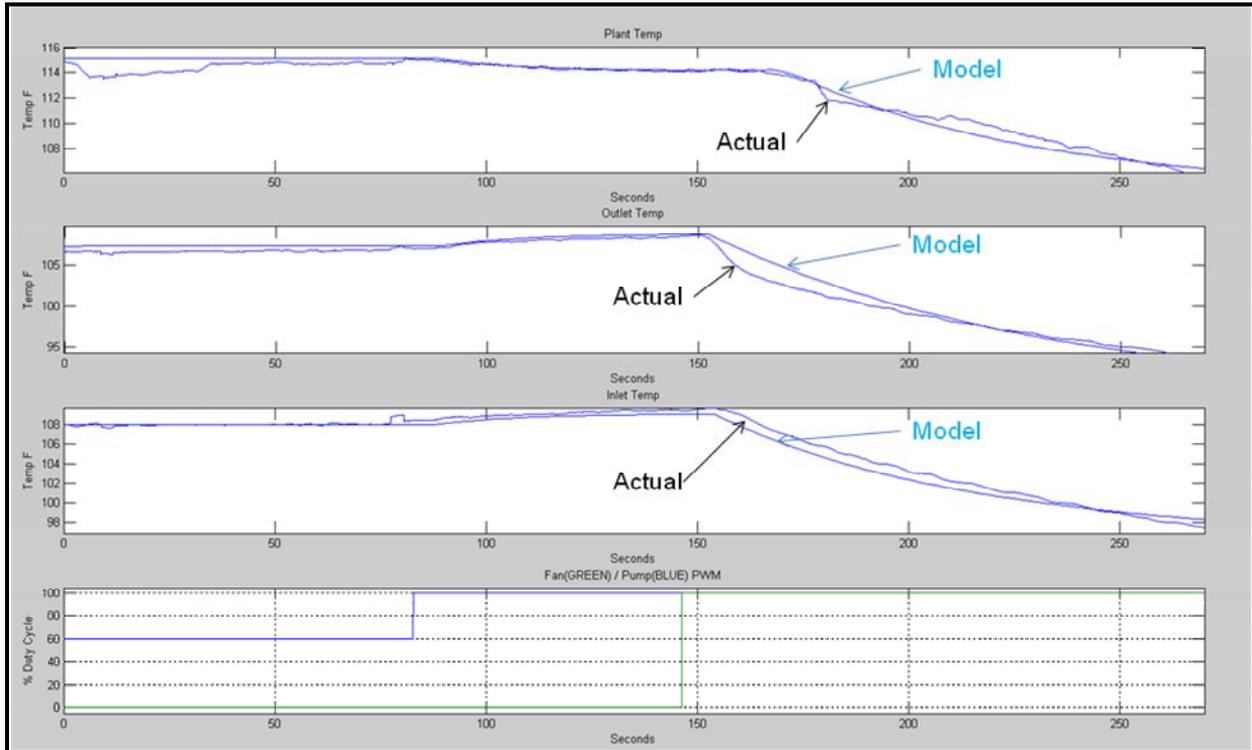


Figure 4-13 Model Data vs. Actual Data

Fig 4-13 shows the model data vs. the actual data. The response is shown for both a step of 0% to 100% for the pump PWM and a 0 to 100% step change for the fan PWM.

4.3 Simulink Implementation/Results

Conversions

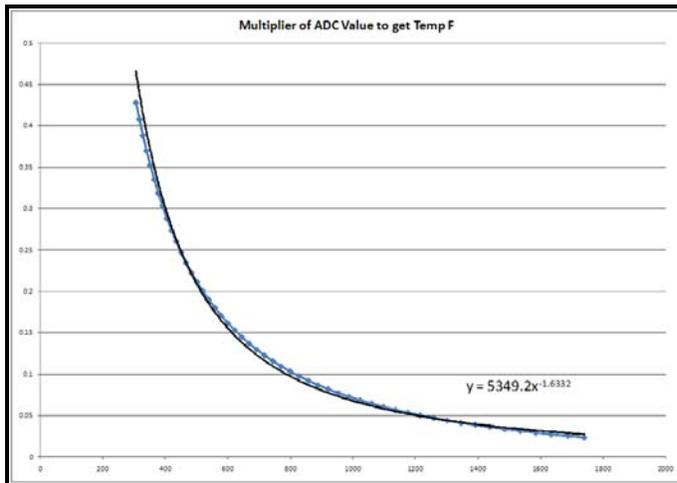


Figure 4-14.1 ADC Value to Temp F Conversion
Power Conversion

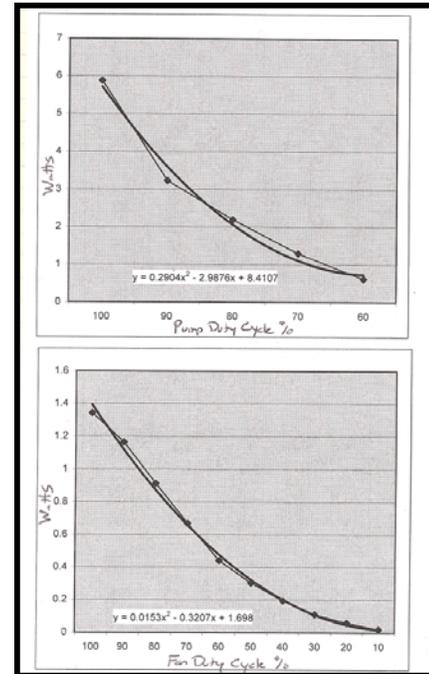


Figure 4-14.2 Duty Cycle to
Power Conversion

Figure 4-14.1 shows the multiplier needed to convert the ADC value into the corresponding temperature in degrees F. The function was created from the datasheet data on the thermistor resistance values for a given temperature. The graph also shows the strong non-linearity of these devices.

Figure 4-14.2 shows the function to convert the PWM duty cycle for both the pump and fan into its power usage in watts. The functions were created to measure the voltage drop and current for each of the devices in 10% PWM steps. Notice that the Excel generated functions do not directly use the PWM %. The first entry on the x-axis, 100%, corresponds to 1, 90% corresponds to 2 etc. The functions used in Simulink also needed to have a conversion to compensate for this.

Controller Types and Results

Bang Bang Control

Bang Bang control is simply a controller which abruptly changes between two states. For state one the fan and pump are at 0 % PWM. State 2 the fan and pump are at 100% PWM.

Improved Bang-Bang

Improved Bang Bang control uses the same concept as bang bang with the exception that state one now keeps the pump PWM at 60%.

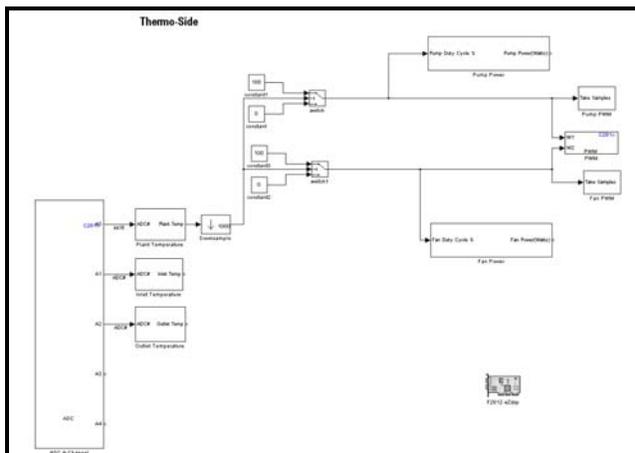


Figure 4-15 Simulink Auto-Code Implementation

Fig 4-15 shows the Simulink diagram for auto-code generation onto the DSP board. Both bang bang and improved bang bang use this same diagram. The only change is to the constant block for pump PWM from 0% (shown in Fig 4-15) to 60%. The temperature threshold can be set within both switches which monitor the plant temperature.

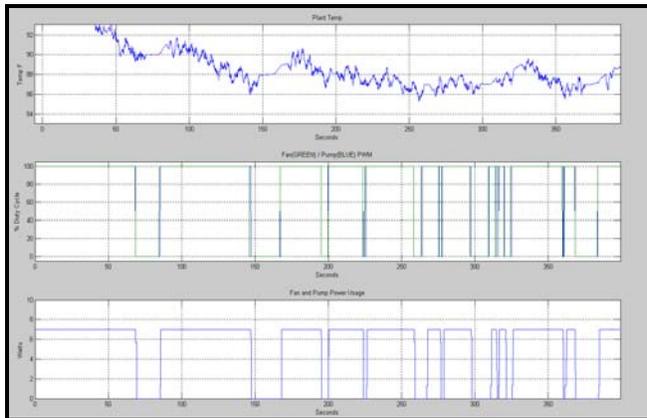


Figure 4-16.1 Bang-Bang Control Data (300Watts/Min)

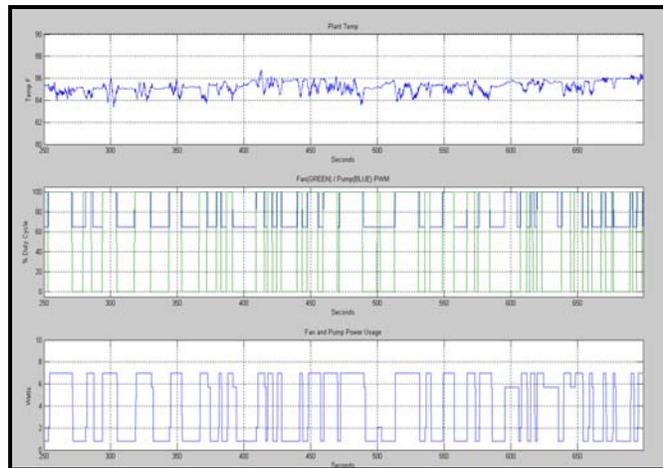


Figure 4-16.2 Improved Bang-Bang Control Data (211Watts/Min)

Fig 16 shows the performance of the two bang bang style controllers. Both were set with a threshold temperature of 85 degrees F. The power consumption was found by integrating the power usage and dividing out the sample number to determine power usage per minute.

P-Control

Proportional Control is a feedback controller that generates an error signal which is multiplied by a determined constant to change the behavior of the monitored system. This system monitors the plant temperature. The pump controller error signal is generated by the difference in plant temperature from the desired temperature. This error is multiplied by a constant (K_p) to change the behavior of the pump velocity. The fan error signal is based off the error between the pump pwm and a desired pump pwm (set to 65%). This error, multiplied by a constant is used to change the velocity of the fan.

PI – Control

PI control uses the same scheme as P-control however an integration is added into the error signal. This is done to reduce the steady state error of the thermal system.

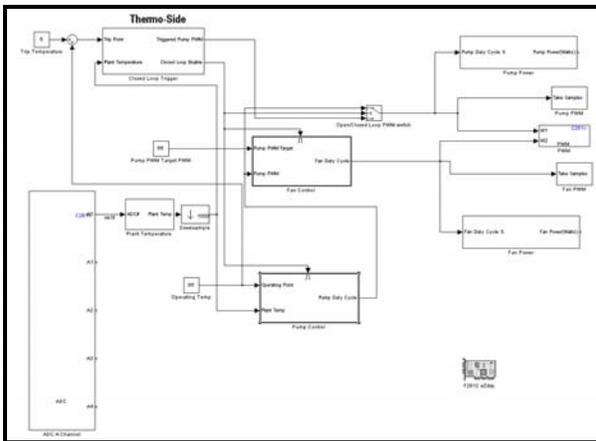


Figure 4-17 Simulink Auto-Code Implementation

Fig 4-15 shows the Simulink diagram for auto-code generation onto the DSP board. Both P and PI use this same diagram. The only change is a discrete integrator is added to both the pump and fan controllers. Notice that the monitoring of outlet and inlet temperature has been moved.

They had to be removed since the auto-code generation generated more than 8k of C-code (on-chip default setting).

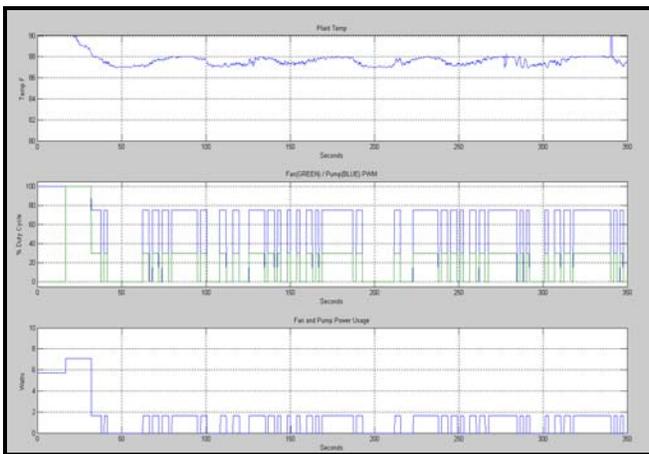


Figure 4-18.1 P Control Data (62Watts/Min)

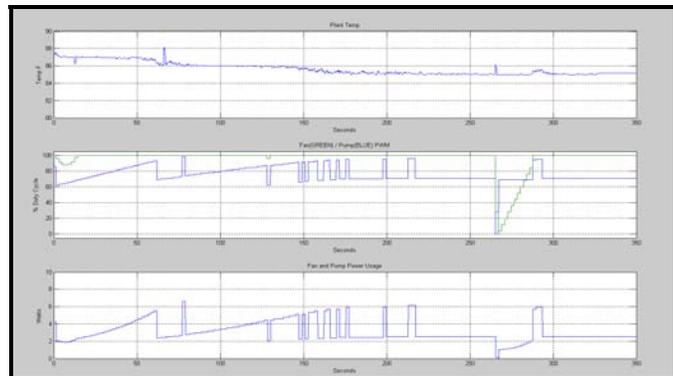


Figure 4-18.1 PI Control Data (143Watts/Min)

Fig 18 shows the performance of the two feedback controllers. Both were set with a threshold temperature of 85 degrees F. The power consumption was found by integrating the power usage and dividing out the sample number to determine power usage per minute.

5.1 Future Additions

Working on the development of the workstation has been essential in understanding the full potentials this system can have. With that, there are several additions that would continue to enhance the capabilities of this platform. These were not implemented in the system due to time constraint and because these were thought of at different points in the project.

The most significant addition to the workstation would be to use the Controller Area Network (CAN) interface that the boards have. The idea of this vehicle bus is to allow microprocessors to talk to each other without a host computer. With that, data can be sent between the DSP Boards to use anywhere. What would be ideal is for the thermal system to send information to display on the engine side GUI to allow full access to the system from one location.

Another addition to the project would be the ability to change the input command type. Currently the only type is step, but ramp and parabolic would be useful to see the effects of the load, but also how steady state error would be effected. The different command inputs can simulate different road conditions for the engine –

load going up a hill, etc. In addition to this, the implemented feed forward control can adjust the acceleration of the engine.

Future control students could be able to use this workstation on an EE431 or 432 homework or design project if a data logging feature were added to the engine side GUI. Ideally a user would be given control specifications and design a controller to meet those specs. So, to make use of this a necessary addition is a table on the GUI that records all control specs, which the user could load into the MATLAB workspace and plot their results. Also, any root locus information would be useful, too.

Appendix A

Cooling System Components

Fan: http://www.koolance.com/water-cooling/product_info.php?product_id=684

Radiator: http://www.koolance.com/water-cooling/product_info.php?product_id=555

Cooling Block: http://www.koolance.com/water-cooling/product_info.php?product_id=678

Reservoir and Pump: http://www.koolance.com/water-cooling/product_info.php?product_id=296

Pump: http://www.koolance.com/water-cooling/product_info.php?product_id=334

Flow meter: http://www.koolance.com/water-cooling/product_info.php?product_id=288

Coolant: http://www.koolance.com/water-cooling/product_info.php?product_id=491

Code Cathode: http://www.koolance.com/water-cooling/product_info.php?product_id=125

Temp Sensor: http://www.koolance.com/water-cooling/product_info.php?product_id=702