2013

Neural Network Based Control of Integrated Recycle Heat Exchanger Superheaters in Circulating Fluidized Bed Boilers

David D. Biruk
University of North Florida, david_biruk@comcast.net

NEURAL NETWORK BASED CONTROL OF INTEGRATED RECYCLE HEAT EXCHANGER SUPERHEATERS IN CIRCULATING FLUIDIZED BED BOILERS

by

David D. Biruk

A thesis submitted to School of Engineering
in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering
UNIVERSITY OF NORTH FLORIDA
COLLEGE OF COMPUTING, ENGINEERING AND CONSTRUCTION
November 2013

The thesis of David Biruk is approved: (Date)

_____ _____

Dr. Chiu Choi

_____ _____

Dr. Daniel Cox

_____ _____

Dr. O. Patrick Kreidl

Accepted for the School of Engineering

_____ _____

Dr. Murat Tiryakioglu
Director of the School of Engineering

Accepted for the College of Computing, Engineering and Construction

_____ _____

Dr. Mark A. Tumeo
Dean of the College of Computing, Engineering and Construction

Accepted for the University

_____ _____

Dr. Len Roberson
Dean of the Graduate School

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The focus of this thesis is the development and implementation of a neural network model predictive controller to be used for controlling the integrated recycle heat exchanger (Intrex) in a 300MW circulating fluidized bed (CFB) boiler. Discussion of the development of the controller will include data collection and preprocessing, controller design and controller tuning. The controller will be programmed directly into the plant distributed control system (DCS) and does not require the continuous use of any third party software.

The intrexes serve as the loop seal in the CFB as well as intermediate and finishing superheaters. Heat is transferred to the steam in the intrex superheaters from the circulating ash which can vary in consistency, quantity and quality. Fuel composition can have a large impact on the ash quality and in turn, on intrex performance. Variations in MW load and airflow settings will also impact intrex performance due to their impact on the quantity of ash circulating in the CFB. Insufficient intrex heat transfer will result in low main steam temperature while excessive heat transfer will result in high superheat attemperator sprays and/or loss of unit efficiency.

This controller will automatically adjust to optimize intrex ash flow to compensate for changes in the other ash properties by controlling intrex air flows. The controller will allow the operator to enter a target intrex steam temperature increase which will cause all of the intrex air flows to adjust simultaneously to achieve the target temperature. The result will be stable main steam temperature and in turn stable and reliable operation of the CFB.

Chapter 1 : Introduction to the Circulating Fluidized Bed (CFB) Boiler

1.1 CFB Background

In the power generation industry, the circulating fluidized bed boiler (CFB) is a relatively new technology when compared with boilers traditionally used for power generation. Fluidized bed boilers were adapted to burn petroleum coke and coal mining waste in the US in the early 1980's. Due to the ability to burn inexpensive renewable and "waste" fuels while maintaining lower emissions than standard pulverized coal units, the demand for CFB boilers has increased. As demand increased for CFB's, so has the size of the CFB. When the CFB's at JEA's Northside Generating Station were built in the early 2000's they were the largest in the world at 297MW each. By 2009 the world's largest CFB was 460 MW. Today units are available at over 600MW. (1)

The JEA owned Foster Wheeler CFB's that are the topic of this research were built as part of a demonstration project with a partnership between the US Department of Energy and JEA. (2) They have gone through years of modifications and process improvements. The process and control improvements made to the existing system eliminated the need for costly modifications to the intrexes. (3) (4) As new CFB's are designed and constructed, CFB manufacturers continue to modify designs to try to improve performance while at the same time boiler owners work to do the same to existing units. This project applies advanced controls to further improve the performance of the CFB.

1.2 CFB Steam generation and superheat

In a CFB boiler, feedwater enters the boiler drum located on top of the boiler. The water exits the boiler drum and moves into the water wall tubes that surround the combustor. As the water is heated in these tubes it turns to steam and enters the top of the boiler drum. This area of the boiler is the steam generating section.

Steam leaves the boiler drum and is heated to higher temperatures in the cyclones and superheat sections of the boiler. The superheat sections add superheat to the steam before it is sent to the turbine. The boiler that is the focus of this project has a primary superheater (PSH) with an outlet temperature between 750 and 800 degrees F followed by three intrex superheaters. Steam leaving the last intrex superheater moves to the high pressure section of the steam turbine with a steam temperature of 1000F. This temperature is controlled by attemperating the steam using feedwater between the primary superheater and first intrex and between the second and third intrex. An overview of the steam path can be seen in Figure 1-1.

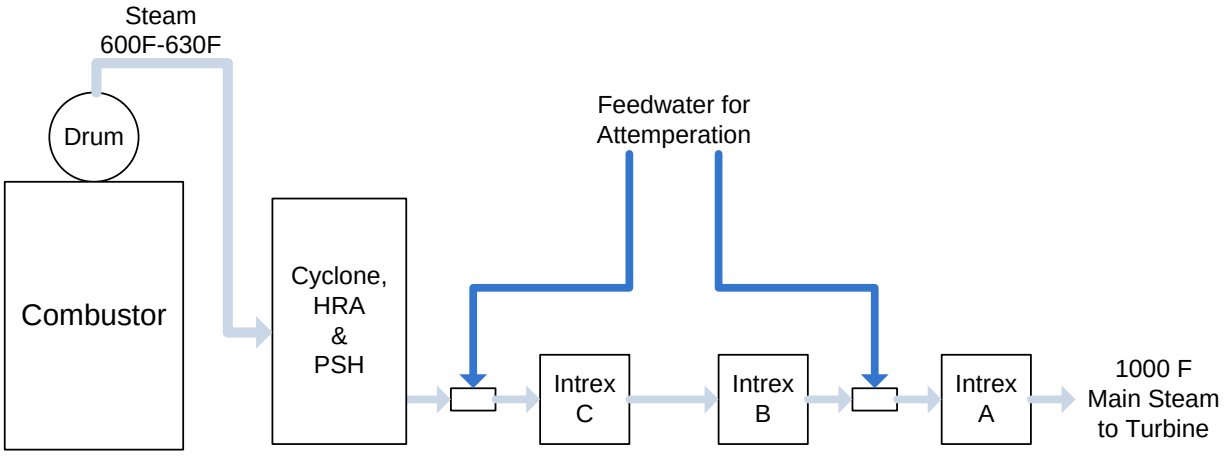


Figure 1-1 Steam Path Overview

If the steam picks up too much superheat, more feedwater is needed for attemperation. Overheating the Intrex tubes and/or excessive attemperator spray has the potential to cause metallurgical problems. If the attemperator is not able to keep the steam temperature down to 1000F, there is loss of turbine efficiency and potential to damage the steam turbine from overheating. If the intrexes do not pick up enough heat there is potential for water induction into the turbine which would also cause damage. Any deviation in main steam temperature from 1000F will impact turbine efficiency.

1.3 CFB Hot Loop

In a CFB, fuel and air are added to the combustor. The fuel mixes with bed material at the bottom of the combustor where it is fluidized by air nozzles in the floor of the boiler. Limestone is also added to the boiler combustion process in order to control SO_2 production and to act as additional bed material. The combination of fuel, ash, and limestone makes up the bed material. Some of the smaller bed material moves up through the combustor and out through the top with the boiler gas. It enters the cyclones where the heavier bed material falls out of the boiler gasses and enters the top of the intrex.

Bed materials move through the intrex and back to the combustor. The intrex provides the seal in the loop between the higher pressure combustor and the lower pressure cyclones. The tubes in the intrex have direct contact with the bed material and heat is absorbed from the bed material through the tubes into the main steam. This cycle is shown with the red arrows in Figure 1-2.

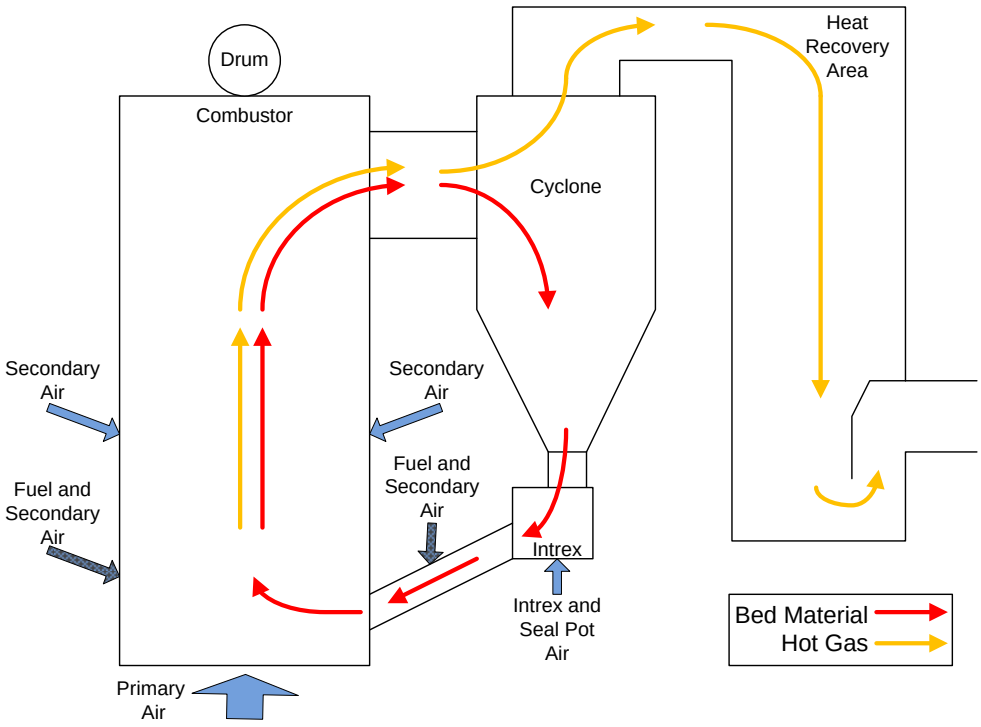


Figure 1-2 CFB Hot Loop

1.4 Current Intrex Control configuration

Many factors can impact the steam temperature increase through the intrexes including steam flow and the temperature of the bed material as well as the manner in which bed material moves through the intrexes. The intrex air flow controls can be used to change the flow of bed material through the intrexes. Each section of the intrex has an independent air flow control damper. These sections can be seen in Figure 1-3.

Using the airflow controls to move more bed material through the intrex tubes will result in more heat being added to the steam. Using the airflow controls to move more material through the bypass channel will result in less heat being added to the steam. The red arrows in Figure 1-4 show the flow of material through the tubes in an intrex superheater and the orange arrows show the bypass flow.

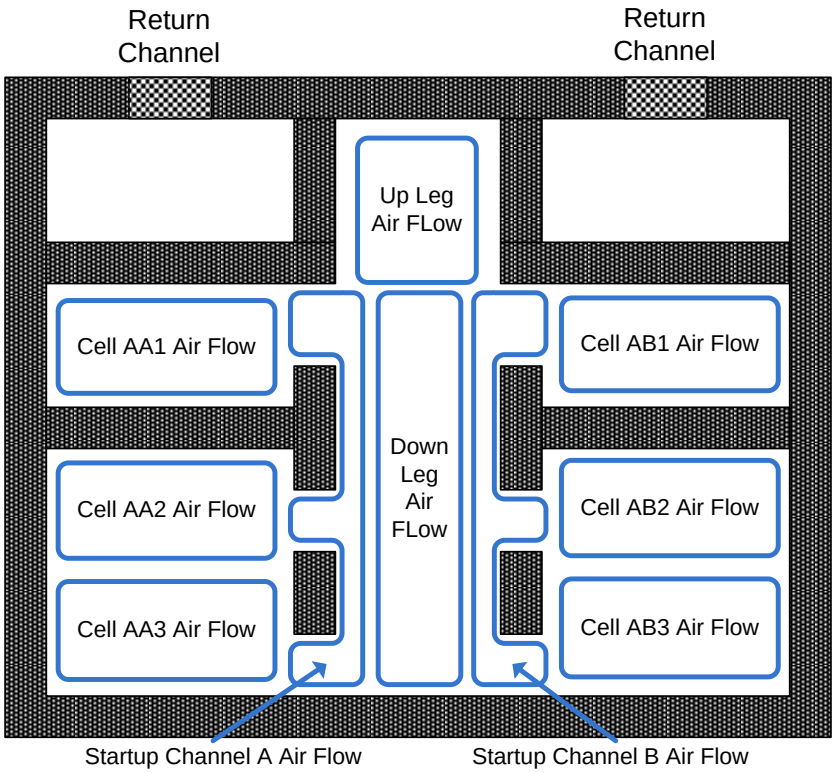


Figure 1-3 Intrex Air flow control layout

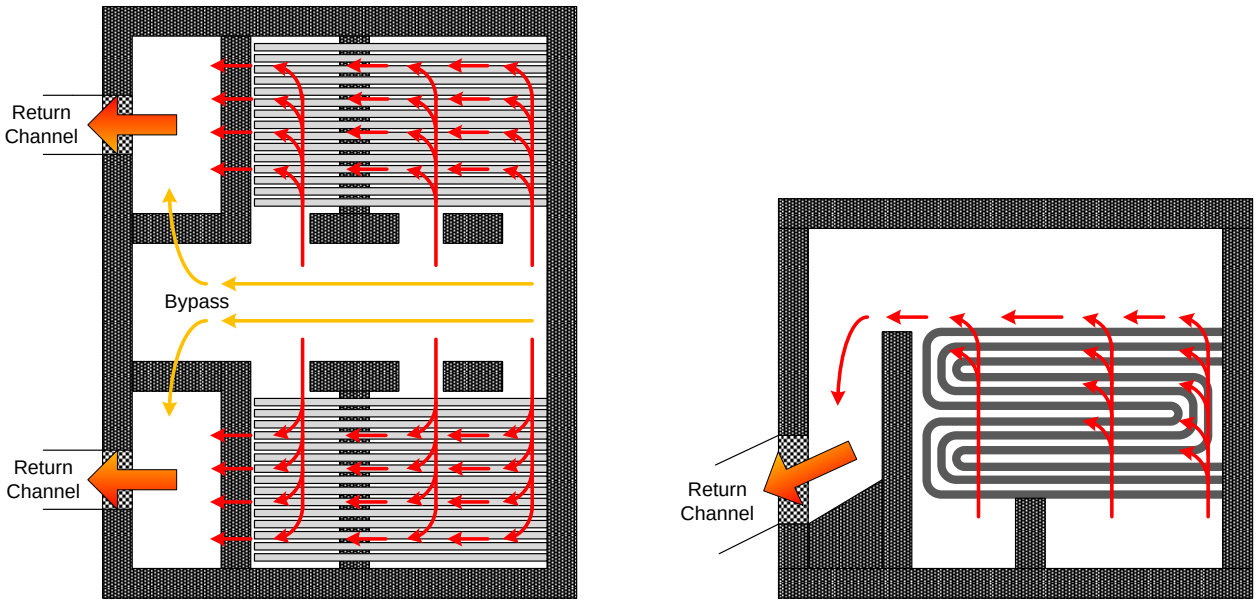


Figure 1-4 Intrex Material Flow Top View (Left) and Side View (Right)

In the previous control configuration the intrex air flows were set depending on unit MW load only so at a certain load the intrex air flows would be the same regardless of other boiler parameters. In this configuration, the steam passing through the intrexes can pick up too much superheat under certain boiler conditions. In some instances the attemperator cannot provide enough attemperation spray to keep steam temperature down to 1000F even when spraying the maximum amount of possible feedwater. This increases the potential for damage to the intrexs and turbine while at the same time reducing efficiency. There can also be times when the intrexes pick up too little superheat which can result in low main steam temperature and the potential for turbine water induction.

The rate at which the material moves through the intrexes is also an important factor. If the material does not move through the intrexes quickly enough, material will back up into the cyclone and it will plug. Once the cyclone plugs, the circulation of material through the hot loop will stop. Without proper hot loop flow, the boiler will not operate and will be forced to come off line. It is not uncommon for the operator to place the intrex air flow controls in manual and adjust them to try to move more ash through the intrexes if they have indications that the cyclones are plugging. This often has a negative impact on intrex heat transfer but enables the unit to continue to operate. The ideal intrex control system would provide intrex heat transfer control while preventing cyclone plugging.

1.5 Organization of Thesis

This Thesis will provide a solution to the current intrex control problems using a multiple input neural network model predictive controller. Other types of advanced controllers have been successfully applied to CFB boiler control applications. (5) Neural Networks have been utilized in the past for modeling and predicting CFB boiler operations. (6) The controller that is the topic of this Thesis will

maintain intrex differential temperature to stabilize main steam temperature and allow the operator to control how much superheat is added to the main steam in the intrex. In order to accomplish this, the model will use inputs from the plant along with air flows generated by an optimization algorithm to determine how to adjust the intrex air flows to compensate for changes in the properties of the bed material.

There are many considerations to be made when considering the application of a neural network model predictive controller. These considerations along with the general structure of the neural network model predictive controller will be discussed in detail in chapter 2. Many of the considerations revolve around the data that will be used for modeling. Chapter 3 will discuss data collection and preprocessing. The discussion on preprocessing will include data point selection, data set reduction, and data normalization.

A detailed discussion of the development of the neural network model specific to this thesis takes place in chapter 4. The structure of the neural network, discussed briefly in chapter 2, is selected through testing from two different structures. A genetic algorithm that uses the data selected in chapter 3 to tune the neural network is discussed in detail along with various parameters of the genetic algorithm that are tested in an attempt to find those which provide optimal tuning of the neural network. Genetic algorithms have been successfully implemented in a wide range of controls applications. (7) (8)

The development and structure of the controller optimization algorithm is discussed in chapter 5. The optimization algorithm includes a linear congruential random number generator for generating random airflows that are applied to the controller's neural network model to determine the optimum air flow setting for the current boiler parameters. The optimization algorithm and neural network model

developed in chapter 4 are programmed directly into the plant distributed control system (DCS). The implementation of the controller into the DCS is discussed in chapter 6.

The results of the controller implementation, shown in chapter 7, verify the ability of the neural network model predictive controller to successfully use the intrex air flow to control intrex differential temperature which will result in stable main steam temperature. Conclusions of this Thesis are discussed in chapter 8 along with opportunities for future research that may improve this application as well as opportunities for additional applications of this research to other areas of CFB control.

Chapter 2 : Overview of the Neural Network Model Predictive Controller

2.1 System Considerations for Neural Network Model Predictive Controllers

When considering a system for neural network control, there are many considerations to be made. Most processes can be controlled by much simpler, traditional methods. Systems that can be accurately mathematically modeled using well-established physics based relationships may not always benefit from a neural network model which is empirical in nature and requires training data to generate. (9) In order to successfully implement a neural network model predictive controller one must consider:

1. System Complexity
2. Process Knowledge
3. Reliability and Repeatability of Instrumentation
4. Data availability
5. Process Control Requirements
6. Resources available for controller implementation

For systems that require only single input-single output PID controllers, an intelligent neural network control system would not likely be necessary. (10) Neural network controllers are ideal for complex, multiple input, multiple output systems. The neural network controller can adjust many parameters simultaneously to reach a desired output. In order to control the heat transferred to the steam in the intrex, 10 air control dampers are controlled simultaneously by 5 different controller outputs. Numerous other boiler parameters will be used to model the intrex heat transfer.

Process knowledge is the starting off point for the neural network design. One of the advantages of a neural network controller is that the physics of the process do not need to be completely understood to design a neural network controller (11) (12). The neural network will "learn" how the system works by using training data. Knowing what process parameters impact the variable that will be controlled by the neural network can reduce unnecessary inputs and reduce system complexity. The list can start out large and then be reduced by analyzing the relationships between collected data. For the intrex, testing has shown that manipulating the intrex airflows has the ability to impact intrex heat transfer. In addition to the intrex air flows, there are dozens of other boiler parameters believed to impact intrex heat transfer.

Process parameters that are deemed important must have reliable and repeatable instrumentation. Unreliable instrumentation will make neural network model tuning difficult and can cause the controller model to incorrectly predict the results of control changes. Averaging values from redundant instruments can increase the availability of the network by reducing the possibility of failure from a single instrument failure. In the intrexes, both sides measure the same parameters and past experience along with historical data has shown that when all instrumentations and controls are working properly, the instrumentation from each of the two sides can be considered redundant and averaged.

For optimal neural network training, data should be available for all operating conditions. (9) If data is not available for all operating conditions, testing and data collection should be performed to expand the data set. Similar quantities of data should be available for all operating conditions as too much data at limited operating conditions will cause the network to be over trained for those conditions causing poor performance under other operating conditions (9).

Different processes can have very different control requirements. The response of the process to controls changes will have a large impact on the control scheme. The CFB has approximately a five minute lag from the time the fuel is changed to the time the MW output changes. Air flow changes in the intrex will have a much more immediate impact. In the case of the intrexes, there is not a desire to have the steam temperature change quickly but rather to be able to maintain it to a set temperature when other boiler parameters change. Having a system that doesn't require a fast response allows for a controller that has a slower response.

Using a predictive controller to control a process can require much more computing resources than a traditional PID controller as typical DCS systems have a single logic block to handle PID controls but can require the combinations of dozens to hundreds of logic blocks to implement a model predictive controller. (13) The speed at which the controller has to respond has a direct impact on the amount of required computing resources. For slower processes the computing does not have to happen as rapidly and less computing resources are needed. The requirements for the intrex are such that the controller can be programmed directly into the DCS controller without the use of external computing resources. This eliminates the need for additional communication interfaces between the DCS and a dedicated neural network machine and also eliminates the need for the continuous use of third party neural network software.

2.2 Neural Network Model Predictive Control Structure

The neural network controller for this project will be a model predictive controller. The controller structure will consist of a neural network model of the intrex and a predictive controller that will apply air flow inputs to the model and compare the model output error to the current output error. If the

applied airflows result in a lower error than those currently applied to the live plant, the airflows from the predictive controller will be applied to the live plant. The block diagram for the neural network model predictive controller can be seen in Figure 2-1.

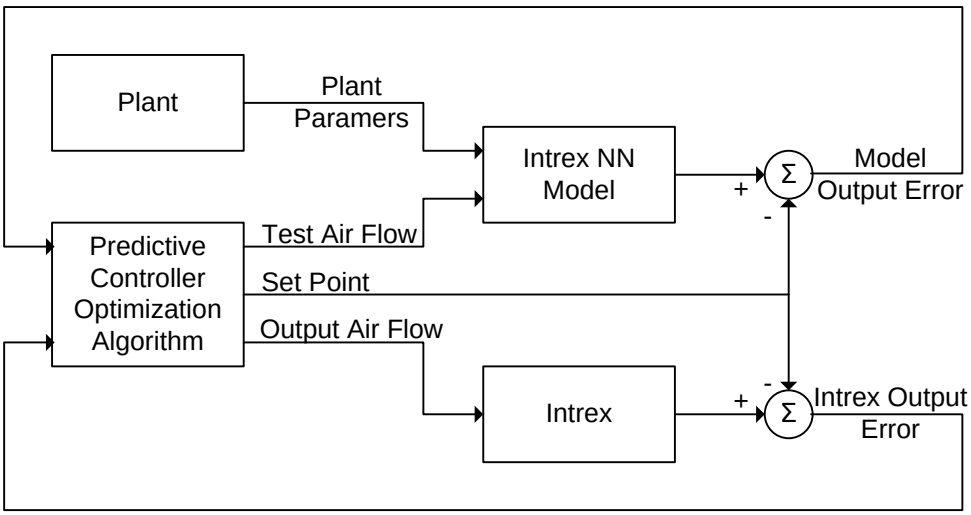


Figure 2-1 Model Predictive Controller Block Diagram

The neural network structure will consist of multiple nodes and layers. Each node will have multiple inputs multiplied by weights and then summed together with a constant. The output of the summation will be applied to an activation function. The outputs from the first layer will serve as the inputs to the next layer. The structure of the neural network node can be seen in Figure 2-2.

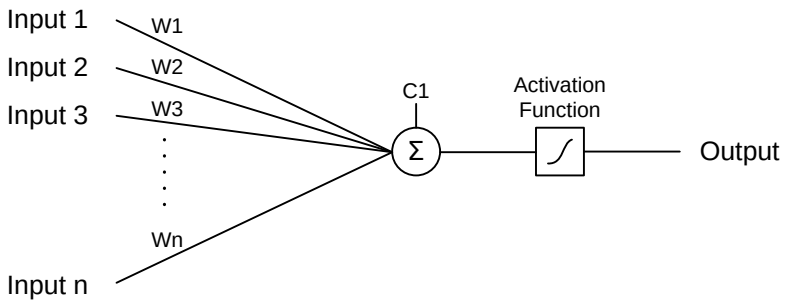


Figure 2-2 Neural Network Node

Chapter 3 : Data Collection and Pre-Processing

As discussed in Chapter 2, good data is essential for the design of a neural network model. (9) Insufficient data can result in poor performance and excessive data will require excessive computing resources to implement. The first step in creating a neural network controller is a good data collection and preprocessing plan. The focus of this project is the A intrex. The main steam is supplied to the high pressure turbine from the outlet of the A intrex. Because of this, controlling the A intrex steam temperature increase has the greatest potential for a positive impact on main steam temperature.

3.1 Data Point Selection

In order to model the intrex, the properties of the steam and bed material passing through it must be determined. Some of these properties either have a direct measurement or another measurement with a direct relationship where others do not. There are however many measurements that can be combined to determine parameters without direct measurements or direct relationships.

Data was collected from the plant information (PI) system using the PI Datalink software add on for Microsoft Excel. Data was not collected from failed redundant instruments. Data was collected for the time period from March – August 2013 in five minute intervals. Periods of operation below 178MW were excluded from the dataset as those are outside the range of normal unit operation. A list of the collected points can be seen in Table 3.1.

Table 3-1 Initial Data Point Set

Tag Name	Description
PS:N1:N01SI34TE821	Intrex Cell AB temperature 1
PS:N1:N01SI34TE822	Intrex Cell AB temperature 2
PS:N1:N01SI34TE824	Intrex Cell AB temperature 3
PS:N1:N01SI34TE825	Intrex Cell AB temperature 4
PS:N1:N01SI34TE827	Intrex Cell AB temperature 5
PS:N1:N01SI34TE828	Intrex Cell AB temperature 6
PS:N1:N01SI34TE805	Intrex Cell AA temperature 1
PS:N1:N01SI34TE806	Intrex Cell AA temperature 2
PS:N1:N01SI34TE807	Intrex Cell AA temperature 3
PS:N1:N01SI34TE808	Intrex Cell AA temperature 4
PS:N1:N01SI34TE809	Intrex Cell AA temperature 5
PS:N1:N01SI34TE810	Intrex Cell AA temperature 6
PS:N1:N01SI34TE811	Intrex Cell AA temperature 7
PS:N1:N01SI34TE812	Intrex Cell AA temperature 8
PS:N1:N01SI34TE861	Intrex Downleg Temperature
PS:N1:N01SI34TE850	Intrex Upleg Temperature 1
PS:N1:N01SI34TE851	Intrex Upleg Temperature 2
PS:N1:N01SI34TE483	Intrex Return Temperature A
PS:N1:N01SI34TE484	Intrex Return Temperature B
PS:N1:1SI34FI800A	Intrex Cell AB1 Air Flow
PS:N1:1SI34FI800B	Intrex Cell AB2 Air Flow
PS:N1:1SI34FI800C	Intrex Cell AB3 Air Flow
PS:N1:1SI34FI816A	Intrex Cell AA1 Air Flow
PS:N1:1SI34FI816B	Intrex Cell AA2 Air Flow
PS:N1:1SI34FI816C	Intrex Cell AA3 Air Flow
PS:N1:1FSHSPFL_A	Intrex Startup Channel Air Flow A
PS:N1:1FSHSPFL_B	Intrex Startup Channel Air Flow B
PS:N1:1FSHDFL	Intrex Downleg Air Flow
PS:N1:1FSHSPUPG_FL	Intrex Upleg Air Flow
PS:N1:N01SI34TE537	Main Steam Temperature to intrex A A
PS:N1:N01SI34TE538	Main Steam Temperature to intrex A B
PS:N1:1AVGBEDDP	Average Furnace Bed Pressure
PS:N1:N01BB34PT422	Furnace Freeboard Pressure 1
PS:N1:N01BB34PT472	Furnace Freeboard Pressure 2
PS:N1:N01BB34PT482	Furnace Freeboard Pressure 3
PS:N1:1TOTPAFLOW	Total Primary Air Flow
PS:N1:1TOTAIRFLOW	Total Air Flow
PS:N1:1SOLIDFUELFLW	Total Solid Fuel Flow
PS:N1:N01GG34JT003	Total Unit Megawatt Load
PS:N1:1FNHEATIN	Total Heat Input
PS:N1:1AVGFBTMP	Average Furnace Bed Temperature
PS:N1:1TOTALLIME	Total limestone flow
PS:N1:1SF_KLB_H	Main steam flow
PS:N1:1INTRXADIF_TMP	Intrex A Differential Steam Temperature

The original data points were believed to have an impact on intrex performance based on process knowledge and past experience. Additional process knowledge was used to reduce the data set. The two intrex cells each contain nine thermocouples. All of the measurements in each cell were averaged together to reduce those data points from 18 points to two. This not only reduces data points but also reduces the potential for a single instrument failure causing the neural network model to malfunction. If one of the instruments malfunctions, the control system will remove it from the average and the model will continue to function properly. The two upleg temperatures were also averaged together.

There is no desire to control the two sides of the intrex differently so controls on either side of the intrex can be averaged together. This was done for the intrex cell air flows, intrex startup channel air flows, and intrex return temperatures. Other parameters outside of the intrex can also be averaged such as redundant thermocouples and Furnace Freeboard Pressure.

Not all of the boiler parameters that are outside of the intrex have an immediate impact on intrex performance. Five minute time delays were also included for some of the parameters outside of the intrex to attempt to capture any delayed impact to intrex performance. The data set with averaged points and five minute delays included can be seen in Table 3-2.

Table 3-2 Data Points with Averages and Delays

Parameter	Description	Included Tags
Avg A1 AF	Intrex Average A1 Air Flow	PS:N1:1SI34FI800A, PS:N1:1SI34FI816A
Avg A2 AF	Intrex Average A2 Air Flow	PS:N1:1SI34FI800B, PS:N1:1SI34FI816B
Avg A3 AF	Intrex Average A3 Air Flow	PS:N1:1SI34FI800C, PS:N1:1SI34FI816C
Avg SUC AF	Intrex Average Startup Channel Air Flow	PS:N1:1FSHSPFL_A, PS:N1:1FSHSPFL_B
DNLG AF	Intrex Downleg Air Flow	PS:N1:1FSHDFL
UPLG AF	Intrex Upleg Air Flow	PS:N1:1FSHSPUPG_FL
Cell AB Ave Temp	Intrex Average Cell AB Temperature	PS:N1:N01SI34TE821, PS:N1:N01SI34TE822, PS:N1:N01SI34TE824, PS:N1:N01SI34TE825, PS:N1:N01SI34TE827, PS:N1:N01SI34TE828
Cell AA Ave Temp	Intrex Average Cell AA Temperature	PS:N1:N01SI34TE805, PS:N1:N01SI34TE806, PS:N1:N01SI34TE807, PS:N1:N01SI34TE808, PS:N1:N01SI34TE809, PS:N1:N01SI34TE810, PS:N1:N01SI34TE811, PS:N1:N01SI34TE812
DNLG Temp	Intrex Downleg Temperature	PS:N1:N01SI34TE861
UPLEG TEMP	Intrex Upleg Temperature	PS:N1:N01SI34TE850, PS:N1:N01SI34TE851
Avg RTN TE	Intrex Average Return Temperature	PS:N1:N01SI34TE483, PS:N1:N01SI34TE484
STM IN TE	Intrex Steam Inlet Temperature	PS:N1:N01SI34TE537, PS:N1:N01SI34TE538
AVG BED	Average Furnace Bed Pressure	PS:N1:1AVGBEDDP
AVG FB	Average Furnace Freeboard	PS:N1:N01BB34PT422, PS:N1:N01BB34PT472, PS:N1:N01BB34PT482
Total PA	Total Primary Air Flow	PS:N1:1TOTPAFLOW
TOT AIR	Total Secondary Air Flow	PS:N1:1TOTAIRFLOW
TOT FUEL	Total Solid Fuel Flow	PS:N1:1SOLIDFUELFLW
MW	Total unit Megawatt Load	PS:N1:N01GG34JT003
Heat in	Total Unit Heat Input	PS:N1:1FNHEATIN
AVG FB Temp	Average Furnace Bed Temperature	PS:N1:1AVGFBTMP
Limestne Flow	Limestone Flow	PS:N1:1TOTALLIME
Steam Flow	Main Steam Flow	PS:N1:1SF_KLB_H
Main stm deviation	Main Steam Temperature Deviation from 1000F	PS:N1:1INTRXADIF_TMP, STM IN TE
intrex a TEMP INCREASE	Intrex A Steam Temperature Increase	PS:N1:1INTRXADIF_TMP
TOT FUEL -5	Total Fuel Flow with 5 minute lag	PS:N1:1SOLIDFUELFLW
Limestne Flow -5	Total Limestone Flow with 5 minute lag	PS:N1:1TOTALLIME

3.2 Dataset Reduction by Stepwise Regression

In order to reduce the complexity of the model the original data set can be reduced to eliminate unnecessary variables. Stepwise regression was selected for dataset reduction. Stepwise regression is a collection of related methods that are designed to work effectively with large data sets. (14)

Regression analysis is used to explore the statistical relationships between variables. Linear regression attempts to find a line of the form y=mx+b that is the best fit of the relationship between the variables. When linear regression is used to model a relationship between two variables, the ability of the model to account for the variability in the relationship is called the coefficient of determination (R^2). In order to calculate the R^2 value, the error sum of squares and total sum of squares are needed. The error sum of squares is calculated by squaring and summing the differences between the actual output values (y_i) and the predicted model output values ($\hat{y}_i$) as seen in equation 3-1. The total sum of squares is the measure of the total variability in the response and is calculated from equation 3-2. The ratio of SS_E to SS_T is the proportion of variability in the relationship between the variables that cannot be accounted for by the regression model. By subtracting this number from 1, the proportion of variability in the relationship between the variables that can be accounted for by the regression model can be calculated. The R^2 value can be calculated from equation 3-3. The closer the R^2 value is to 1, the more accurate the regression model is. (14)

Equation 3-1: Error Sum of Squares

$$SS_E = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Equation 3-2: Total Sum of Squares

$$SS_T = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

Equation 3-3: Coefficient of Determination (R^2)

$$R^2 = 1 - \frac{SS_E}{SS_T}$$

The relevance of the inputs to a regression model can be determined through hypothesis testing. In the case of the regression model, the null hypothesis H_0 would be that the regression coefficient for a given input would equal to zero. If the null hypothesis is rejected, the alternate hypothesis, the regression coefficient is not equal to zero, would be accepted. In order to determine whether or not to reject the null hypothesis, the P-value is used. The P-value is the probability that the test statistic will take on a value that is at least as extreme as the observed value of the statistic when the null hypothesis is true. A typical cutoff value for the P-value, referred to as α, is 0.05. This can be interpreted as meaning that there is only a 5% chance that the null hypothesis is true or a 95% chance that the null hypothesis is false. (14)

In order to perform the stepwise regression for data selection, data was needed for varying operating conditions. Testing was performed for one week at which time the intrex airflows were adjusted to values that they are not normally operated at. In addition to collecting the data from the test period, points were taken from the standard operating condition data collected from March through August and added to the dataset. The combined dataset was loaded into Minitab 16 statistical analysis software for the purposes of performing a stepwise regression to reduce the size of the data set.

The stepwise regression tool in Minitab allows the user to select which data is the response and which data to use to attempt to predict that response. It also allows the user to select predictors to be used in every model. For the purposes of this project, the intrex air flows are included in every model since they are going to be the means of control. With the stepwise regression function, Minitab will automatically add/remove the other predictors from the model based on the P-value calculated for each predictor. Minitab allows for the user to set the α value and also allows for the stepwise regression to be performed by adding predictors, removing predictors, or both. The analysis of the intrex data set was performed using an α of 0.05 to add or remove predictors and with both the add and remove function active. This allowed for a reduction of the dataset from 25 variables to 20 variables which can be seen in Table 3-3.

Table 3-3 Reduced Dataset from Stepwise Regression

Parameter	Description	Regression Coefficient	P-Value
Constant	Regression Constant	805.1	N/A
Avg A1 AF	Intrex Average A1 Air Flow	0.00128	0.000
Avg A2 AF	Intrex Average A2 Air Flow	-0.00084	0.001
Avg A3 AF	Intrex Average A3 Air Flow	0.001	0.000
Avg SUC AF	Intrex Average Startup Channel Air Flow	0.0004	0.000
DNLG AF	Intrex Downleg Air Flow	0.00082	0.011
UPLG AF	Intrex Upleg Air Flow	-0.0001	0.013
Cell AB Ave Temp	Intrex Average Cell AB Temperature	0.0284	0.000
Cell AA Ave Temp	Intrex Average Cell AA Temperature	0.0154	0.000
DNLG Temp	Intrex Downleg Temperature	-0.0049	0.02
UPLEG TEMP	Intrex Upleg Temperature	0.014	0.000
STM IN TE	Intrex Steam Inlet Temperature	-0.8355	0.000
AVG BED	Average Furnace Bed Pressure	0.032	0.027
AVG FB	Average Furnace Freeboard	-0.324	0.000
Total PA	Total Primary Air Flow	0.00557	0.000
Heat in	Total Unit Heat Input	-0.00482	0.000
AVG FB Temp	Average Furnace Bed Temperature	-0.0081	0.000
Limestne Flow	Limestone Flow	-0.0092	0.000
Steam Flow	Main Steam Flow	-0.0118	0.000
Main stm deviation	Main Steam Temperature Deviation from 1000F	0.592	0.000
TOT FUEL -5	Total Limestone Flow with 5 minute lag	0.0148	0.033

The stepwise regression output from Minitab predicts an R^2 value of 92.30% with the predictors from Table 3-3. The complete output file from Minitab can be seen in Appendix A. By multiplying each variable by the associated coefficient from Table 3-3 and then adding the constant from the Table, the regression model output of the intrex differential temperature can be calculated. The regression model output equation can be seen in equation 3-4.

Equation 3-4: Regression Model Output

$$Intrex\ DT = 805.1 + \sum_{i=1}^{20} (Parameter_i \times Coefficient_i)$$

The regression model will serve as the baseline for model performance. The goal is to find a better model of the system using a neural network than that found by using the regression. In order to verify model performance, the mean squared error (MSE) and the coefficient of determination (R^2) will be calculated.

3.3 Data normalization

Before the data can be used to for neural network modeling, it must be normalized. (15) Normalization of the data effectively removes the units from the data by rescaling all of the variables to the same scale. In theory, data normalization is not necessary as the model tuning should tune out the scales. In reality, if the data is not normalized and the variables are on varying scales, the model will take a long time to tune and is more likely to get stuck in a local minimum in the error surface. Tuning weights for variables with contrasting ranges can be challenging. This will also degrade the performance any dynamic tuning algorithms. (15)

Normalization can mean different things from rescaling variables in a data set to have the same scale (vector length) to transforming data to be zero mean with a standard deviation of one. The variables for this project will be normalized to be zero mean with a standard deviation of one.

To perform the normalization, the mean and standard deviation are required for each variable in the data set. The mean for each variable is subtracted from that variable and the result is divided by the standard deviation for that variable as seen in equation 3-5. In statistics this is also called standardizing.

Equation 3-5: Normalization

$$Normalized\ X_i = \frac{X_i - \bar{X}}{\sigma}$$

The mean and standard deviation were calculated for each variable in the data set. It is important to note that if new data are added to the existing data set that these values may need to be updated. Matlab programs were written to automatically normalize and un-normalize the data set. The Matlab programs written for the normalization and inverse normalization can be seen in Figures 3-1 and 3-2 respectively.

```
function [normdata]= mmnorm(normmat,data)

% This function will take in data and an associated normalization matrix
% (normmat)containing the mean and standard deviation of the data set
% and perform normalization.  The normalized data will be returned.

normdata = zeros(size(data,1),size(data,2));    %Initialize the matrix
x=normmat(1,:);                                 %Get mean for each variable
y=normmat(2,:);                                 %Get SD for each variable
parfor i=1:size(data,2)                         %Normalize the data
    normdata(:,i) = ((data(:,i)-x(i)))/y(i);
end
end
```

Figure 3-1 Matlab Normilization Function

```
function [normdata]= immnorm(normmat,data)

% This function will take in data and an associated normalization matrix
% (normmat) containg the mean and standard deviation of the data set and
% perform inverse normalization.  The un-normalized data will be returned.

normdata = zeros(size(data,1),size(data,2));    %Initialize the matrix
x=normmat(1,:);                                 %Get mean for each
variable
y=normmat(2,:);                                 %Get SD for each variable
parfor i=1:size(data,2)                         %Un-Normalize the data
    normdata(:,i) = ((data(:,i)*y(i)))+x(i);
end
end
```

Figure 3-2 Matlab Inverse Normalization Function

Chapter 4 : Neural Network Modeling

When designing the neural network model, there are many considerations to be made. The number of input variables was previously determined by stepwise regression and the number of output variables is already known to be one. The number of layers, number of nodes in each layer, and the activation function need to be determined. The method for training the neural network must also be determined.

4.1 Neural Network Model Structure

Neural Networks with one hidden layer are considered universal approximators according to the 1989 paper written by Hornik, Stinchcombe, and White. (16) This means that in most cases a system can be successfully modeled with only one hidden layer. The model for this project will use one hidden layer with an input and output layer.

The number of input layer nodes typically matches the number of input variables which will be the case for this project. The number of output nodes is set by the number of model outputs which in this case is one. Many “rules of thumb” exist for determining the number of hidden layer nodes, one being that the number of hidden layer nodes is typically between the number of input and output nodes. (17) (18)In reality, the ideal number of nodes in the hidden layer is dependent on the system the model is based on and the “rules of thumb” are a starting point. (19) (18) This project will use testing to select the number of hidden layer nodes.

The output of each node in the neural network with the exception of the node in the output layer will be applied to an activation function. There are many types of activation functions that are commonly used. If a linear activation function is used, the neural network acts as a combination of linear regressions with each node representing a single regression.

Activation functions for neural networks are typically a form of sigmoid function. The sigmoid functions are non-linear "S" shaped functions that limit the output value of the node. (20) The sigmoid function also enables the network to model non-linear functions. For the Intrex Neural Network model, it is desired to have the output of the transfer function for each node fall between 1 and -1. This would typically be done with a tan-sigmoid activation function. The shape of the tan-sigmoid activation function can be seen in Figure 4-1.

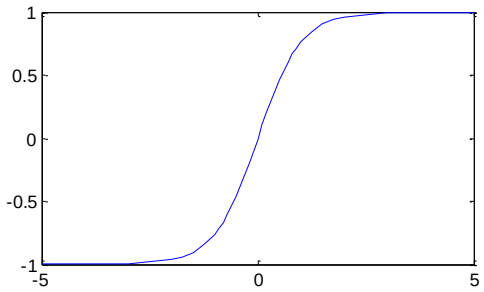


Figure 4-1 Tan-Sigmoid Activation Function

The tan-sigmoid activation function is implemented in the model program using equation 4-1. This equation can also be easily implemented into the DCS.

Equation 4-1: Tan-sigmoid Activation Function

$$Tansig(x) = \frac{2}{1 + e^{-2x}} - 1$$

A Matlab routine was written for the neural network. This routine will be called by the main program. The program takes in the input and output data, the network weights and constants, and the number of layer 1 and layer 2 nodes and returns the MSE, individual error values, maximum error and the neural network output. The Matlab routine can be seen in Figure 4-2.

```
function [MSE,err,maxer,out]=
neurnet(inA,outA,l1w,l1c,l2w,l2c,olw,olc,lay1n,lay2n)

%Network Structure
%   lay1n defines the number of neurons in the input layer.  lay2n defines
%   the number of neurons in the second layer.  The output layer will
%   always be 1 neuron.  Weights will be applied before the summing blocks
%   for each neuron.  Constants will be added at each summing block.
%   The output of each neuron will pass through an activation function

%Inputs:
%   inA = input data set (variables in different columns)
%   outA = expected output for each input
%   l1w = layer 1 weights
%   l1c = layer 1 constants
%   l2w = layer 2 weights
%   l2c = layer 2 counstants
%   olw = output layer weights
%   olc = output layer constant
%   lay1n = number of first layer neurons
%   lay2n = number of second layer neurons
%
%Outputs:
%   MSE = Mean square error
%   err = raw error values
%   maxer = maximum error
%   out = neural net output

out = zeros(1,size(inA,1));                 %Initialize Weight Matrix
weights1=reshape(l1w,size(inA,2),lay1n);    %reshape weight matrix
l1c=repmat(l1c,size(inA,1),1);              %Create l1 constant matrix
lay1out = (inA*weights1)+l1c;               %layer 1 summing node
lay1out = 2./(1+exp(-2.*lay1out))-1;        %layer 1 activation function
weights2=reshape(l2w,lay1n,lay2n);          %reshape weight matrix
lay2out = lay1out*weights2;                 %layer 2 summing node part 1
l2c=repmat(l2c,size(inA,1),1);              %create l2 constant matrix
lay2out = lay2out+l2c;                      %layer 2 summing node part2
lay2out = 2./(1+exp(-2*lay2out))-1;         %layer 2 activation function
weightsout=transpose(olw);                  %transpose out weights
out = lay2out*weightsout+olc;               %output summing node
err = outA-out;                             %calculate error
maxer = max(err);                           %find maximum error
MSE = mean((err).^2);                       %calculate MSE
end
```

Figure 4-2 Matlab Code for Neural Network Model Simulation

There are many types of algorithms used to train the weights in a neural network. Two of the more common algorithms are gradient descent and stochastic search methods. Both algorithms attempt to minimize a cost function which is typically the mean squared error (MSE). Gradient Descent tends to tune faster but also tends to find a local minimum in the error surface where stochastic search methods are better at finding the global minimum but require much more time and processing resources to implement. (21) The MSE is calculated using equation 4-2 where y_i is the actual value, $\hat{y}_i$ is the model output, *n* is the population size and *p* is the number of predictors.

Equation 4-2: Mean Squared Error

$$MSE = \frac{SS_E}{n-p-1} = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n-p-1}$$

This project employs a stochastic search method called a genetic algorithm. The genetic algorithm starts with a randomly generated population. Each member of the population is a set of neural network weights and constants. Each member is applied to a neural network in order to calculate the MSE for each member. The members of the population with the best MSE are chosen to be parents for the next generation in the algorithm and the remainder of the population is removed.

Features are randomly selected from the parents and used to generate new children to complete the population for the next generation using crossover. A percentage of the total neural network weights and constants that make up the children will then be mutated. This mutation can be adjusted to be from 0-100%. In addition to the percentage of weights and constants to be mutated, the amount of mutation must also be considered. A flow chart of the genetic algorithm can be seen in Figure 4-3

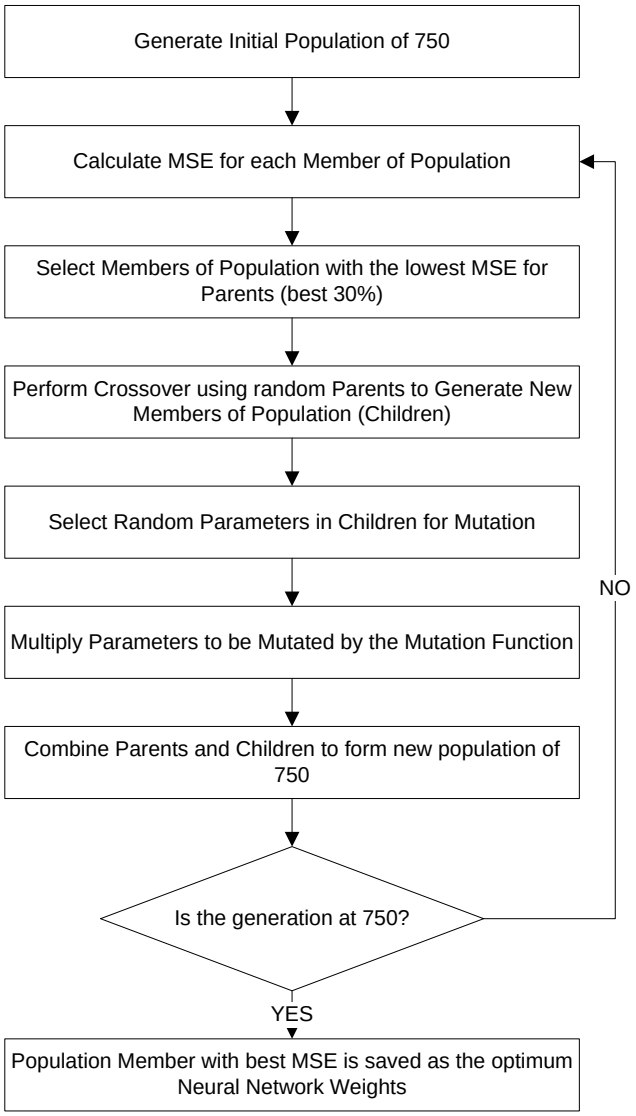


Figure 4-3 Genetic Algorithm Flow Chart

The calculation for the amount of mutation starts with a random number between -1 and 1. The random number is then multiplied by a mutation function which limits the maximum and minimum mutation. The mutation function can be set to a specific amount or varied as the algorithm progresses from one generation to the next. For this project, constant mutation is tested as well as mutations that decay as the generation increases. The linear mutation decay function can be seen in equation 4-3.

Equation 4-3: Genetic Algorithm Linear Mutation Decay Function

$$mutation(x) = .4 \times \frac{total\ generations - generation(x)}{total\ generations} + .1$$

The mutation starts at up to 50% (+/- .5) and then decreases linearly in relationship to the generation number until it reaches a maximum of 10% (+/-.1) at the final generation. The purpose of the linear decays is to promote faster learning in early generations and prevent overshoot and promote fine tuning in later generations. Figure 4-4 shows an example of the progression of the mutation over 400 generations with the linear mutation decay function applied.

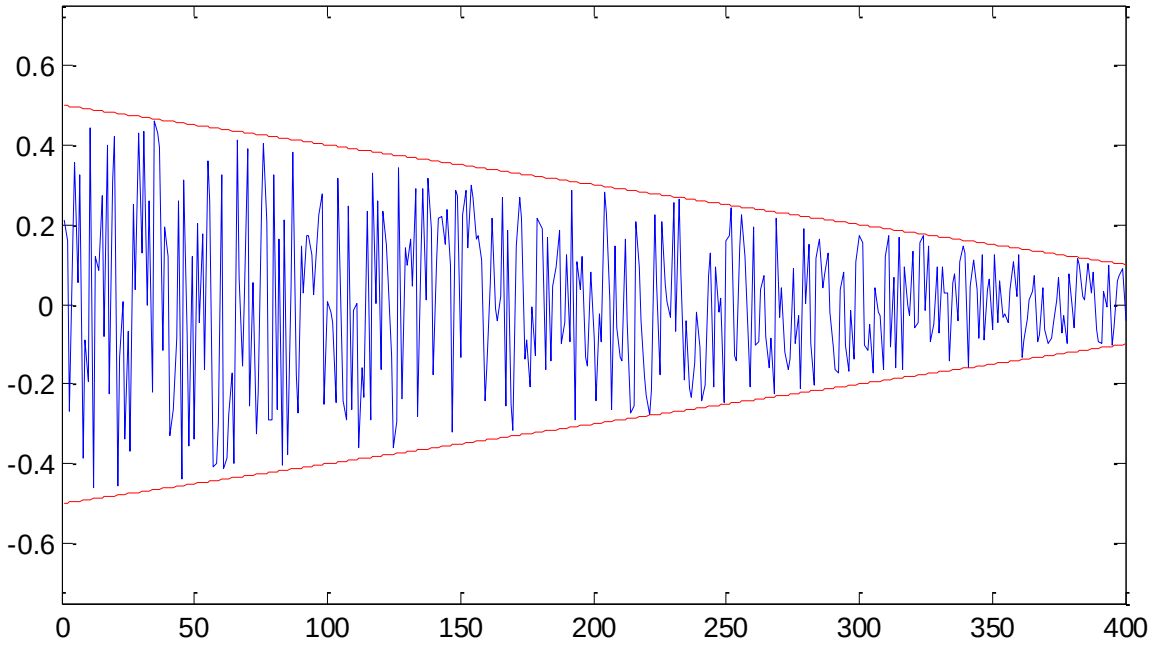


Figure 4-4 Genetic Algorithm Linear Mutation Decay Function

The cosine mutation decay function has an overall decay but will periodically increase and decrease as the generation increases. A decaying cosine function is added to the linear mutation decay function so the overall cosine decay function has an overall decay similar to the linear mutation decay function

starting at 60% and decreasing to 10%. The equation for the cosine mutation decay function can be seen in equation 4-4.

Equation 4-4: Genetic Algorithm Cosine Mutation Decay Function.

$$mutation(x) = .4 \times \frac{total\ generations - generation(x)}{total\ generations} + .1$$
$$\times \frac{total\ generations - generation(x)}{total\ generations} \times COS(\frac{20 \times generation(x) \times \pi}{total\ generations}$$

The periodic increase in mutation enhances the ability of the genetic algorithm to escape from a local minimum should one be found. Figure 4-5 shows an example of the progression of the mutation over 400 generations with the cosine mutation decay function applied.

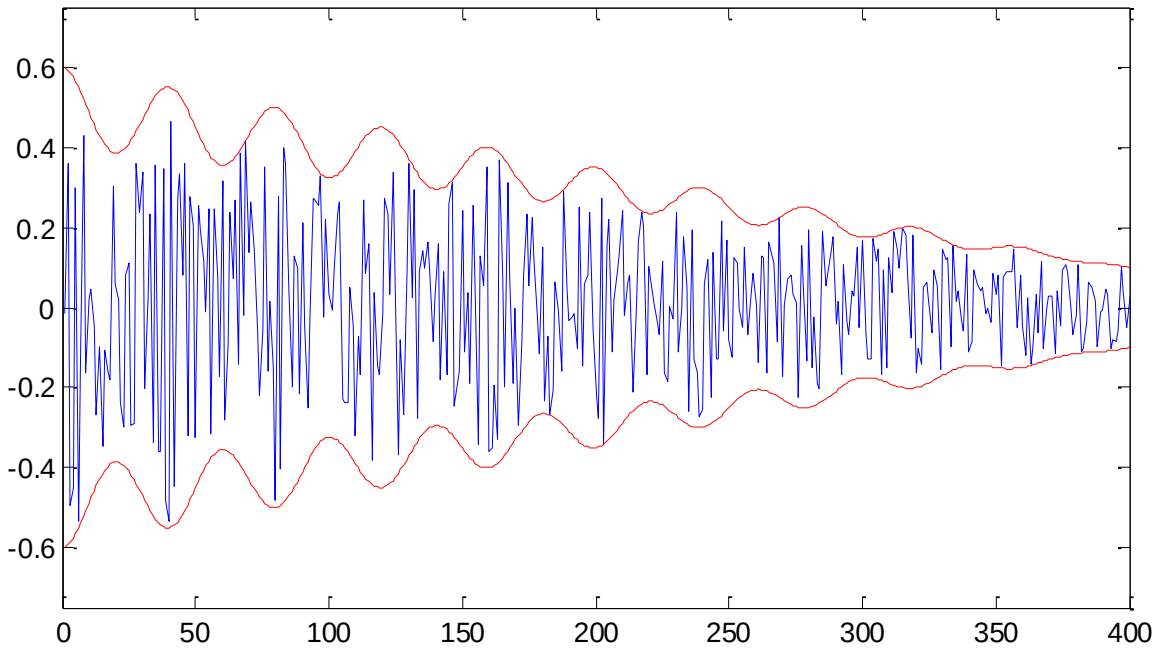


Figure 4-5 Genetic Algorithm Cosine Mutation Decay Function.

For the purposes of comparing the decay functions, the mutation functions were plotted together in Figure 4-6. A constant mutation of 25% will be compared with the mutation decay functions.

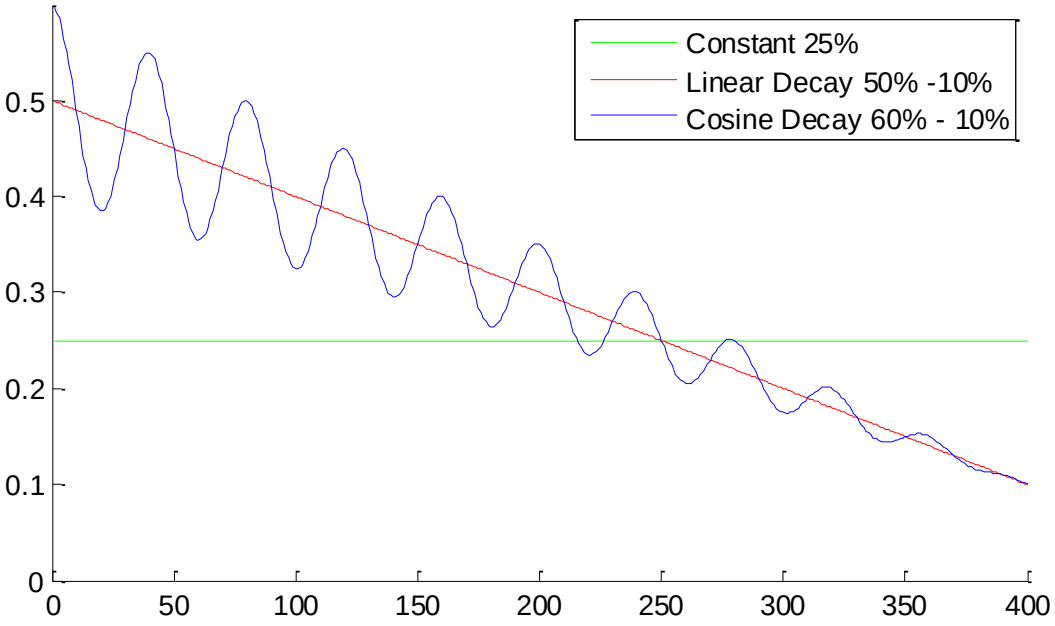


Figure 4-6 Genetic Algorithm Mutation Functions

Matlab code was written to generate and mutate the children. This code can be seen in Figure 4-7. The mutation function active in the code is the constant mutation function. The mutation decay functions are commented out and highlighted. The selection of parents for the next generation requires the neural network function from Figure 4-2 to calculate the MSE. Both the neural network and genetic algorithm functions are written into the Matlab program for the intrex model.

```
function w = genalg(parents,mut,totgen,gen,pop)

%   This program will generate a new population of weights and constants
%   using the below inputs

%Inputs
%   Parents = matrix of parent weights
%   mut = mutation
%   totgen = total number of generations
%   gen = current generation number
%   pop = Size of population to generate
%
%Outputs
%   w = weighs

numc = pop - size(parents,1);       %number of children to generate
% make children
w = zeros(numc,size(parents,2));
parfor i = 1:numc                   %for the number of children
%   generate 2x1 matrix of ints from 1:number of parents
    x = randi(size(parents,1),2,1);
%   generate 1xnumber of weights matrix of ints from 1:2
    y = randi(2,1,size(parents,2));
%   convert 2's to 1 and 1's to 0 to select first parent
    p1=y-1;
%   convert 2's to 0 to select second parent
    p2=abs(y-2);
%   combine parts frome each parent for each weight
    w(i,:)=p1(1,:).*parents(x(1),:)+p2(1,:).*parents(x(2),:);
end
% mutate children
%   determine which weights will be mutated
mutloc = randi(numc*size(parents,2),1,ceil(mut*numc*size(parents,2)));
%   mutation varies from 50% to 10% as the generation number is increased
%mutation = .4*(totgen-gen)/totgen+.1 ;
%   mutation decays from 60% to 10% with an added cosine function
%mutation =(.4*(totgen-gen)/totgen+.1)+.1*((totgen-
gen)/totgen)*cos(20*gen/totgen*pi);
%   Constant Mutation of 25%
mutation = .25;
%   determine the amount of mutation for each weight (-1:1 * mutation)
mutmul = (1-(rand(1,length(mutloc))*2)*mutation);
%   generate an empty matrix for the new children
mutmat = ones(numc,size(parents,2));

for i = 1:length(mutloc)            %for each mutation
    mutmat(mutloc(i)) = mutmul(i);  %fill in the mutation matrix
end
w = w .* mutmat;                    %generate new children
% make population of parents and children
w = cat(1,parents,w);
end
```

Figure 4-7 Matlab Code for Genetic Algorithm

4.3 Neural Network Training and Testing Programs

In order to train and test the neural network, the data previously collected must be imported into Matlab and normalized. The original data set was divided sequentially into 25 groups. The testing data set was divided sequentially into 4 groups. Group 2 from the original data set and groups 1 and 3 from the testing data set were combined to create a training data set. Group 12 from the original data set and groups 2 and 4 from the testing data set were combined to create a testing data set. The import and load functions were written into the main Matlab program and can be seen in Figure 4-8.

```
%Nerual Network Model Program
clear

%get training data
intrain = xlsread('Training_Data_in2');
outtrain = xlsread('Training_Data_out2');

%get testing data
intest = xlsread('Testing_Data_in2');
outtest = xlsread('Testing_Data_out2');

%Get normalization Matrix
innormmat=xlsread('STD_Norm_in');
outnormmat=xlsread('STD_Norm_out');

%Perform Normalization
intrain = mmnorm(innormmat,intrain);
outtrain = mmnorm(outnormmat,outtrain);
intest = mmnorm(innormmat,intest);
outtest = mmnorm(outnormmat,outtest);
```

Figure 4-8 Matlab Data Input and Normalization

The next portion of the program defines the network structure and parameters for the genetic algorithm. These parameters can be adjusted to find structure and genetic algorithm parameters that generate the best model weights for the lowest MSE. Figure 4-9 shows this portion of the program.

```
%Define Neural Network Structure
%[MSE,err,maxer,out]= neurnet(inA,outA,l1w,l1c,l2w,l2c,olw,lay1n,lay2n);
L1N = 20;                           %number of neurons in layer 1
L2N = 15;                           %number of neurons in layer 2
nL1w = L1N * size(intrain,2);       %number of layer 1 weights
nL1c = L1N;                         %number of layer 1 constants
nL2w = L2N*L1N;                     %number of layer 2 weights
nL2c = L2N;                         %number of layer 2 constants
nOLw = L2N;                         %number of output layer weights
nOLc = 1;
Totw = nL1w+nL1c+nL2w+nL2c+nOLw+nOLc;    %total number of weights
nin = size(intrain,2);

%Set Genetic Algorithm parameters
mutation = .1;                      %amount of mutation in genetic algorithm
pop = 750;                          %population (number of sets of weights)
numpar = 225;                       %number of parents to use to generate children
generations = 750;                  %number of generations

%Generate inital weights from -1 to 1
w = (rand(pop,Totw)-.5)*2;
```

Figure 4-9 Matlab Neural Network Model Structure and Genetic algorithm parameters

The previously discussed neural network and genetic algorithm programs are utilized in the main neural network program training routine. An additional program was written to convert the weight matrix to a form more easily used by the neural network. This code can be seen in Figure 4-10 and the Neural Network Training portion of the main program can be seen in Figure 4-11.

```
function [lay1w, lay1c, lay2w, lay2c, outw]=expweights(w,l1n,l2n,numins)

a=l1n*numins;   %Range for layer 1 weights
b=a+1;          %min for layer 1 constants
c=a+l1n;        %max for layer 1 constants
d=c+1;          %min for layer 2 weights
e=c+l1n*l2n;    %max for layer 2 weights
f=e+1;          %min for layer 2 constants
g=e+l2n;        %max for layer 2 constants
h=g+1;          %min for output layer weights
i=size(w,2);    %max for output layer weights
lay1w=w(1:a);   %layer 1 weights
lay1c=w(b:c);   %layer 1 constants
lay2w=w(d:e);   %layer 2 weights
lay2c=w(f:g);   %layer 2 constants
outw=w(h:i);    %output layer weights
outc=1;         %output layer Constant
end
```

Figure 4-10 Matlab Weight Conversion

```
%Training
    MSE = zeros(1,generations);
    %for each generation
    for j = 1:generations
        % calculate the error for each parent
        mse=zeros(1,size(w,1));                 %Initialize mse
        error=zeros(size(w,1),size(intrain,1)); %Initialize error
        maxer=zeros(1,size(w,1));               %Initialize maxer
        out = zeros(size(w,1),size(intrain,1)); %Initialize out
        parfor k = 1:size(w,1);                 %For each parent weight
            %Convert Weights for NN program
            [l1w, l1c, l2w, l2c, outw,outc]=expweights(w(k,:),L1N,L2N,nin);
            %Calculate the mse for the parent
            [mse(k), error(k,:), maxer(k),out(k,:)] =
                neurnet(intrain,outtrain,l1w,l1c,l2w,l2c,outw,outc,L1N,L2N);
        end

        %capture best MSE
        MSE(j) = min(mse);
        % find the best weights
        parent=zeros(numpar,Totw);
        for kk = 1:numpar;                  %for one to the number of parents
            keep = find(mse == min(mse)); %find the location of minimum error
            parent(kk,:) = w(keep(1),:);  %Store the parent with minimum error
            mse(keep) = 10000000;         %maximize error for that parent
        end
        %Generate new weights
        w = genalg(parent,mutation,generations,j,pop);
```

Figure 4-11 Matlab Neural Network Training program

The training routine will repeat and output the MSE for each generation. The best set of weights will be the parent with the lowest MSE from the final generation. The MSE is the primary metric for how well the model is performing. In order to find the best model, the outputs of models with similar MSE values will need to be looked at. Two models can have similar MSE values but very different trends and histograms. The histogram of the raw errors between the plant and the models and plots of the model outputs vs. the actual plant output were generated and reviewed to look for undesired results. Matlab code was also written to generate the histograms and plots to compare the plant output to the regression model output and the neural network model output. For the Matlab code written for testing, see appendix B.

4.4 Neural Network Testing

To determine which genetic algorithm parameters and neural network structure generate the best results, a testing matrix was generated. The testing parameters are listed in Table 4-1. Each test was run using 750 generations.

Table 4-1 Neural Network Testing Parameters

Parameter	Value 1	Value 2	Value 3
GA Mutation	10%	15%	20%
GA Parents	30% of Population	40% of Population	N/A
GA Mutation Function	Constant 25%	Linear Decay Function	Cosine Decay Function
NN Layer 2 Nodes	10	15	N/A

The MSE and neural network weights were captured for each test run. The MSE values for each set of test parameters were averaged to determine which genetic algorithm parameters produced the best results. The averaged MSE for each parameter can be seen in Table 4-2. The Cosine mutation decay function with 10% population mutation and 30% of the population being used as parents provided the best results. The complete results can be seen in appendix C.

Table 4-2 Neural Network Genetic Algorithm Parameter Performance

	Mutation Decay Function		
	Constant	Linear	Cosine
MSE ($°F^2$)	3.151	2.967	2.932
	Mutation %		
	10	20	30
MSE ($°F^2$)	2.902	3.026	3.125
	Parents		
	30%	40%	
MSE ($°F^2$)	3.008	3.027	

The optimum parameters highlighted in Table 4-2 were used to test each of the neural network structures with ten and fifteen and hidden layer nodes each an additional four times. The MSE for each structure was averaged to determine the optimal number of hidden nodes in the neural network. The averaged MSE along with the top three MSE values for each structure can be seen in Table 4-3. The top three MSE values using fifteen hidden layer nodes are all better than the best MSE value using ten neural network nodes. Fourteen of the forty four tests that were performed provided MSE values less than the 2.909 MSE value that the regression model provided with the same input data.

Table 4-3 Neural Network Results with Varied Hidden Layer Nodes

Hidden Nodes	Average MSE ($°F^2$)	MSE 1 ($°F^2$)	MSE 2 ($°F^2$)	MSE 3 ($°F^2$)
10	3.043	2.688	2.739	2.774
15	2.915	2.429	2.496	2.584

The performance of the two neural networks that generated the lowest MSE was compared with the performance of the regression model. The percentages of output values for the three best error ranges were calculated along with the percentage of errors above +/- 5.5 degrees F for each model and can be seen in Table 4-4. Beyond +/- 2.5 F the error percentages for each model are all within .3%. The Neural Network Model with the MSE of 2.496 has over 8% more errors at the +/- .5 range than either of the other two models.

Table 4-4 Model Error Percentages

Error Range (F)	+/- .5	+/- 1.5	+/- 2.5	> +/- 5.5
Regression %	43.32%	84.37%	93.14%	1.66%
NN MSE 2.429 %	42.47%	84.85%	93.14%	1.36%
NN MSE 2.496 %	51.38%	86.24%	92.93%	1.43%

Histograms were generated for all three models. These histograms can be seen in Figure 4-12. The histograms of the regression model and the neural network model with the lowest MSE appear similar. The neural network model with the second lowest MSE has a noticeably larger number of errors that are less than +/- .5 degree F from the actual plant output.

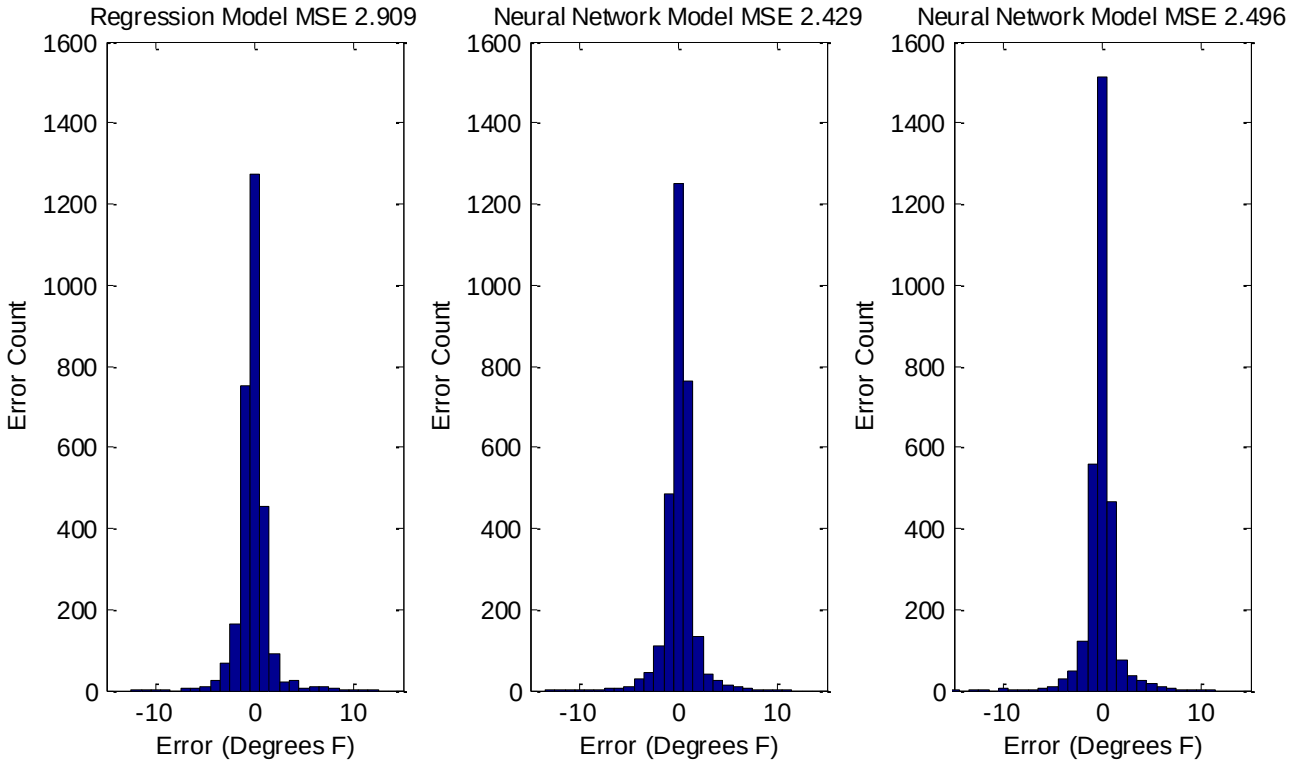


Figure 4-12 Model Output Error Histograms

The R^2 value was calculated for both neural network model sand the regression model using the testing data set. The R^2 value for the regression model that was generated using the data for data point selection was 92.30%. Re-calculating the R^2 value for the regression model using the larger testing data set resulted in an R^2 value of 92.10. The neural network models both had a better R^2 value than the regression model. The MSE and R^2 values for all three models can be seen in Table 4-5.

Table 4-5 MSE and R^2 Values for Regression and NN Models

Parameter	Regression Model	NN Model (Lowest MSE)	NN Model (Second Lowest MSE)
MSE ($°F^2$)	2.909	2.429	2.496
R^2 (%)	92.100	92.940	93.150

Data was collected from 9/1/2013 12:00PM to 9/2/2013 12:00 PM in one minute intervals, a period outside of the original dataset. All three of the model outputs for that timeframe were plotted against the plant output for the same timeframe and can be seen in Figure 4-13.

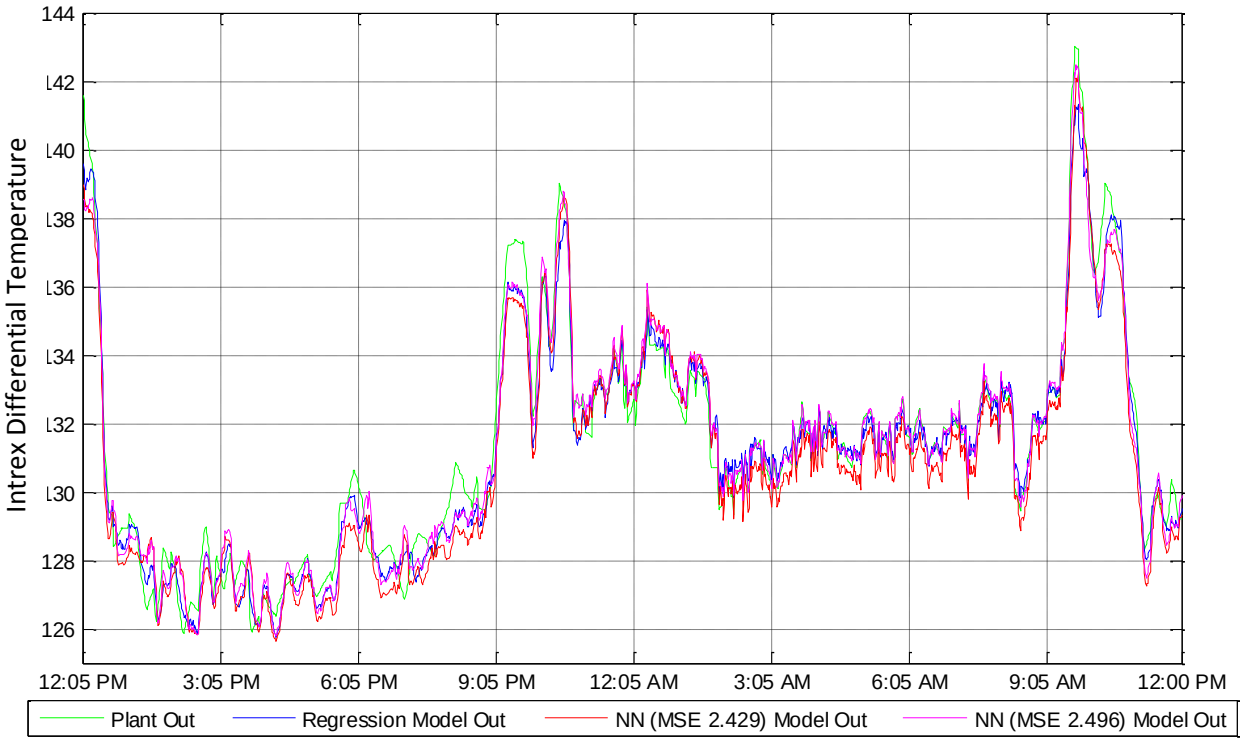


Figure 4-13 Intrex Plant Differential Temperature vs. Intrex Model Differential Temperatures

When looking at Figure 4-13 it appears that all of the model outputs are very similar over the 24 hour period. In order to be able to differentiate between the models, the timeframe from 6:05 AM to 12:00 PM on 9/2/13 was looked at in Figure 4-14. In the 130F to 135F operating range, both the regression model and the neural network (MSE 2.496) model trend the plant output very closely. The other neural network model appears to have constant -0.5 degree offset. Above 140F both neural networks are closer than the regression model.

After a comparison of MSE and R^2 values for the regression model and two neural network models as well as the histograms and trends, the neural network model with the MSE of 2.496 was selected for use with the model predictive controller. The neural network model with the MSE of 2.496 will be programmed into the DCS and act as the model for the model predictive controller. A trend from the same time period as that in Figure 4-14 was generated with the other neural network model removed and can be seen in Figure 4-15. It should be noted that even though the regression model does perform with accuracy close to that of the neural networks, for the given data set the regression model is not going to improve any further but the neural network model may be further optimized by altering the neural network structure or tuning algorithm.

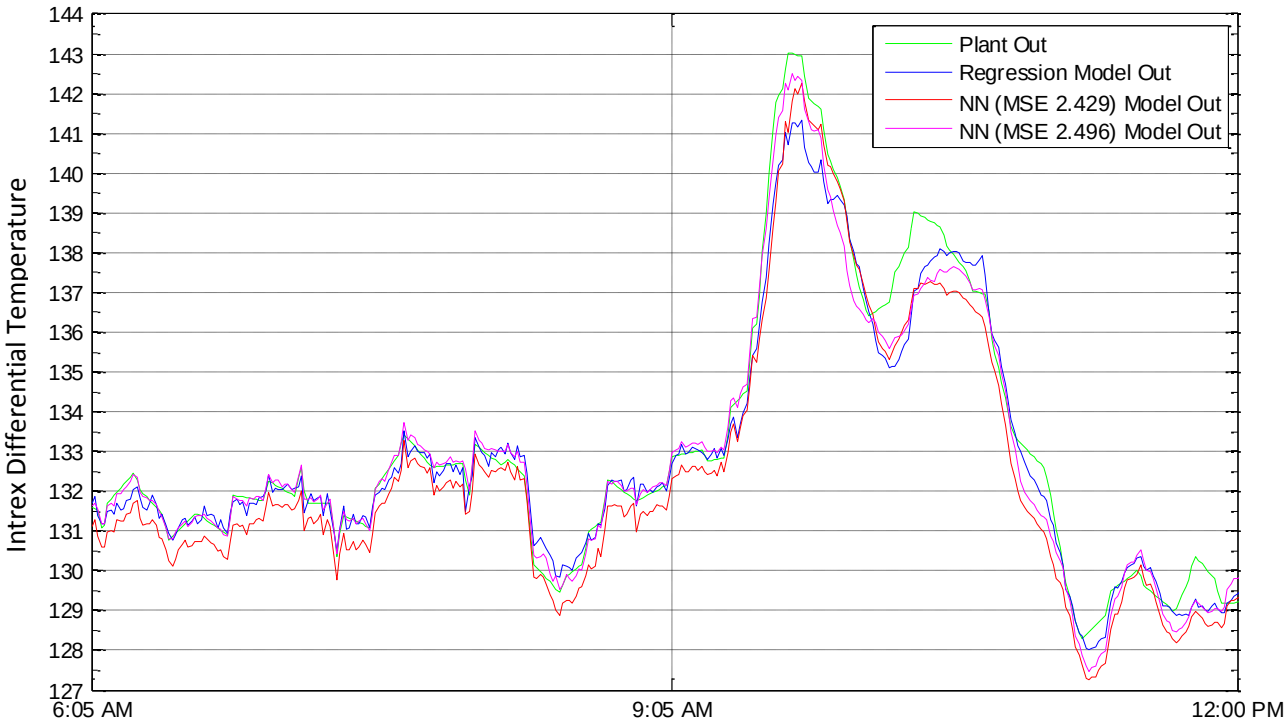


Figure 4-14 Performance of Regression and Both Neural Network Models

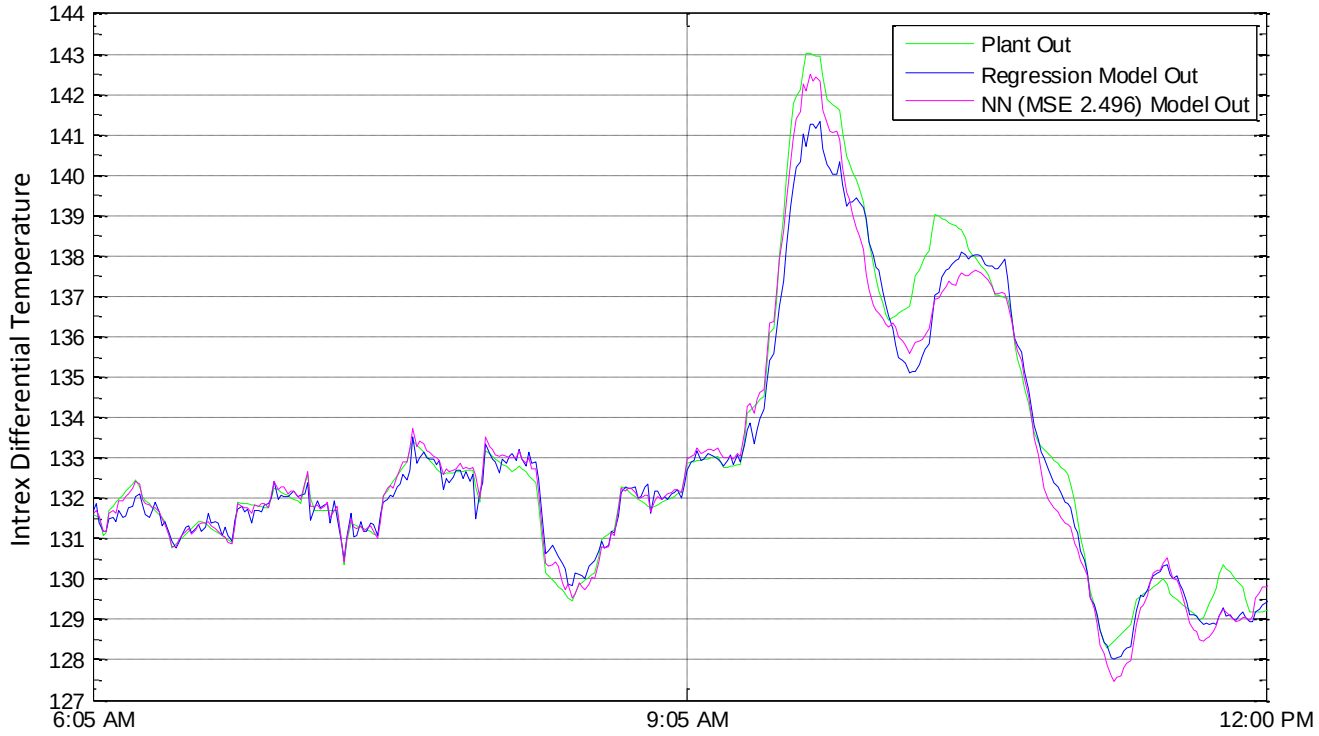


Figure 4-15 Performance of Regression Model and Best Neural Network Model

Chapter 5 : Controller Optimization Algorithm

Many optimization algorithms, sometimes referred to as cost function minimization algorithms, have been used for a model predictive controller. Mathematical Optimization Methods such as the Newton-Rapshon optimization algorithm proposed by Soloway and Haley (22) are common in model predictive control. An extended dynamic matrix control algorithm using a neural network as a non-linear prediction model was proposed by Draeger, Engell, and Ranke. (11) Advanced stochastic optimization methods such as the genetic algorithm optimization proposed by Yu and Zhu have also been researched. (23) The choice of which method to use will depend largely on the required system performance and the system resources that are available to implement the controller. In the case of the controller for this project, there are limited resources to work with but the rate of response does not have to be extremely fast as the overall plant process response is on the order of minutes.

5.1 Optimization Algorithm Operation

One of the goals of this project is to program the neural network model predictive controller directly into the plant DCS. Both the mathematical and advanced stochastic methods referenced above require programming capabilities and/or computational resources beyond what can be practically programmed into the plant DCS used for this project. The optimization algorithm for this controller uses a simple stochastic approach. The optimization algorithm generates completely random combinations of intrex

airflows using a linear congruential random number generator which will be discussed in more detail in section 5-2. Each combination is applied to the neural network model as it is generated along with the other current plant parameters. The error for the current airflows is compared with the stored previous best error. If the current airflow error is better than the previous, the new airflows become the output of the optimization algorithm. Once every 60 seconds the stored best airflows are re-applied to the model and the error value is updated. This is required to compensate for changing plant conditions. A block diagram of the optimization algorithm can be seen in Figure 5-1.

None of the airflows that are stored as a result of having the lowest error are reused to generate the next set of airflows as they would be in a learning algorithm such as a genetic algorithm or particle swarm optimization. This would require the collection of a number of results before the output could be updated. Each collection would require one complete module scan. The typical scan time of the DCS used for this project is 250ms. It can be increased to some degree but will be limited by the amount of other logic in the control module. The time required for a learning algorithm will severely slow the optimization algorithm.

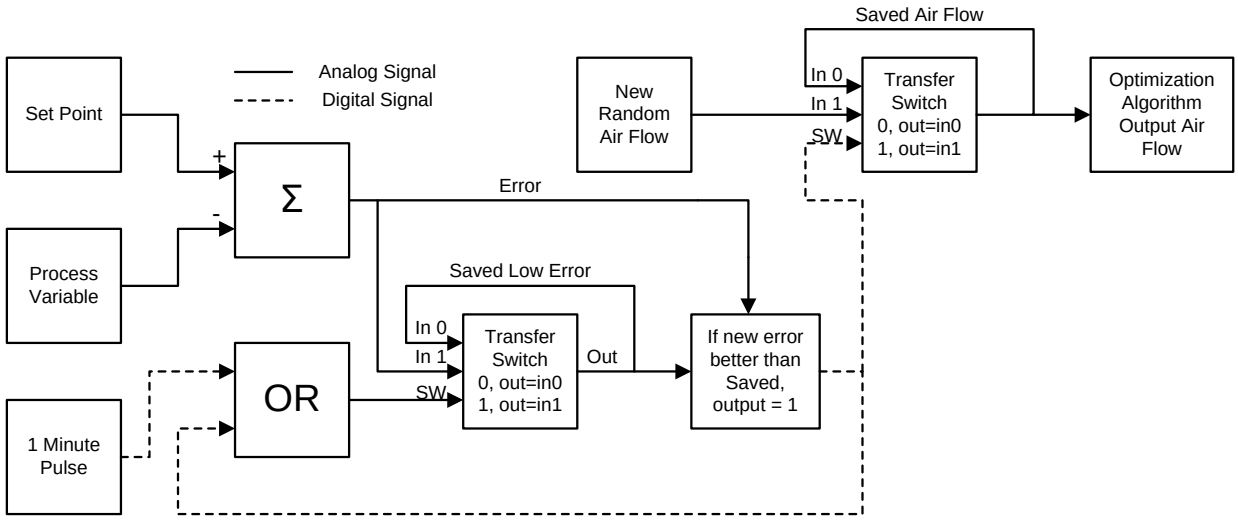


Figure 5-1 Optimization Algorithm Block Diagram

To ensure the set point of the controller remains at a realizable value, the minimum and maximum capabilities of the plant are also calculated. The setpoint for the controller is limited by these capabilities. The error signals for the minimum and maximum capabilities are also updated every 60 seconds to compensate for changing plant process variables. The block diagram for the optimization algorithm in Figure 5-1 is the same as the ones used to calculate the minimum and maximum plant capabilities. To calculate the maximum intrex capabilities, the set point is set at 300F, a point above any that will ever be reached. To calculate the minimum intrex capabilities, the set point is set at 0F, a point below any that will ever be achieved.

5.2 Linear Congruential Random Number Generator

In order to generate the random numbers for the optimization algorithm multiple linear congruential random number generators (RNG's) were programmed into the DCS. The linear congruential RNG is a common random number generator that can be implemented using DCS function codes and does not require a lot of memory.

The equation for the linear congruential RNG can be seen in equation 5-1. The "*m*" is the modulus and must be greater than 0. The "*a*" is the multiplier and the "*c*" is the increment value, both of which must be between 0 and the value of "*m*". The initial X_n is the seed value or previous value. The maximum period of the RNG will be defined by the modulus value m in the equation. In order to achieve the maximum period, *c* and *m* must be relatively prime, *a*-1 must be divisible by all prime factors of *m*, and *a*-1 must be a multiple of 4 if m is a multiple of 4.

Equation 5-1: Linear Congruential Random Number Generator.

$$X_{n+1} = (aX_n + c) Mod\ m$$

The RNG used for the optimization algorithm can be seen in equation 5-2. The values for the equation were selected to provide a full period of numbers from 0 to 99. The random number generator will generate numbers from 0 to 99. An example of the RNG output with a seed value of zero can be seen in Figure 5-2.

Equation 5-2: Linear RNG for Generating numbers from 0 to 99

$$X_{n+1} = (21 \times X_n + 7) Mod\ 100$$

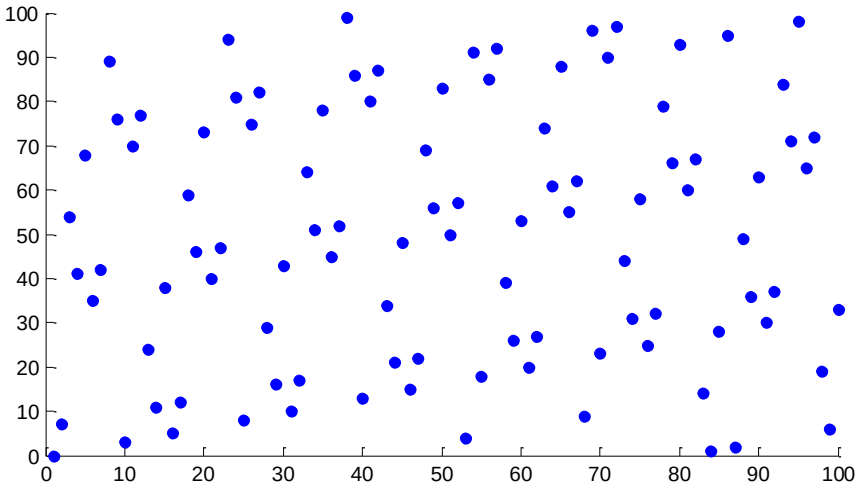


Figure 5-2 Linear Congruential Random Number Generator Output

There are five RNG's used for the optimization algorithm. Each RNG is seeded at different times using the internal DCS clock. One RNG is seeded every 13 seconds. The value of the seed is the sum of the current minute and second of the DCS clock scaled from zero to 100. An example of 40 iterations of the five RNG's can be seen in Figure 5-3. The five values on each line represent the percentage of each airflow value that will be used as an input for the neural network.

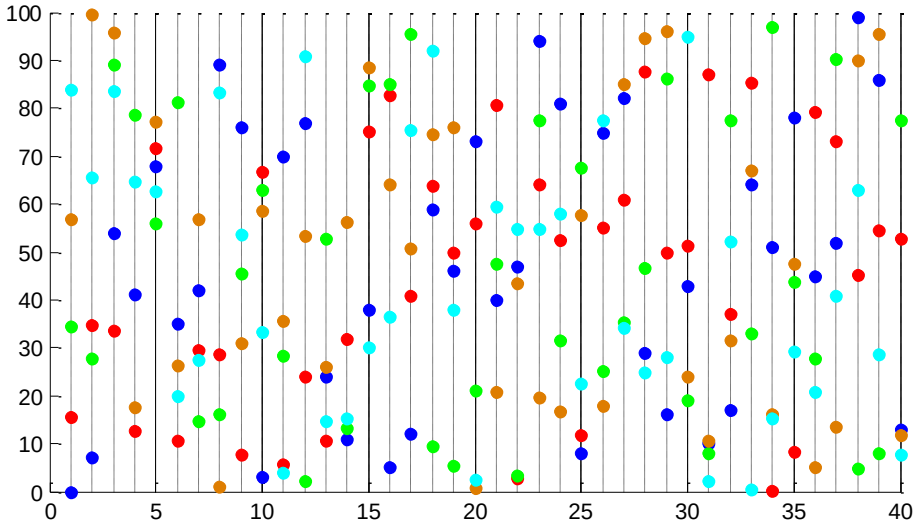


Figure 5-3 Output From all 5 Random Number Generators

The RNG outputs are each scaled to an acceptable airflow range before going to the neural network. The ranges should be limited to values for which data has been collected and used for neural network tuning. Having values outside of the tuning dataset can result in unpredictable operation. The minimum and maximum airflow values for the output of the optimization algorithm can be seen in Table 5-1.

Table 5-1 Optimization Algorithm Min/Max Values

Parameter	Minimum	Maximum
Cell A1 Air Flow	500	3000
Cell A2/A3 Air Flow	500	3000
SUC Air Flow	0	2000
DNLG Air Flow	2000	4000
UPLG Air Flow	3000	9000

Chapter 6 : Distributed Control System (DCS) Integration

6.1 DCS Function Codes and Logic Structure

The DCS utilized for this project is an ABB Symphony Harmony Infi-90 system. The DCS controller module used in this project is a BRC300 Bridge Controller module. The programming software used to program the controller is ABB Composer with Automation Architect. In order to program the DCS, function codes are tied together and conFigured to perform control functions. Function code operation and configuration instructions can be found in the ABB Function Code Application Manual. (13)

Function codes are saved as “blocks” in the controller. The BRC300 can hold 9999 blocks. Each block is assigned a block number. The blocks are scanned in order of the block number. In most applications of this type of DCS, the time for one complete scan of the DCS blocks is set to 250ms. This number is adjustable but is limited by the capabilities of the controller and amount of control logic.

For most DCS applications, the plant response is much slower than the DCS scan time making the order in which the blocks execute somewhat unimportant. Most digital signals are held for at least a second giving the processor multiple scans to read the value and react. For this project, there will be many signals that change with each scan making the order in which the blocks scan critical for proper operation. The flow chart in Figure 6-1 shows the order of operation for the DCS logic for this project with numbers representing the order in which each set of blocks is scanned.

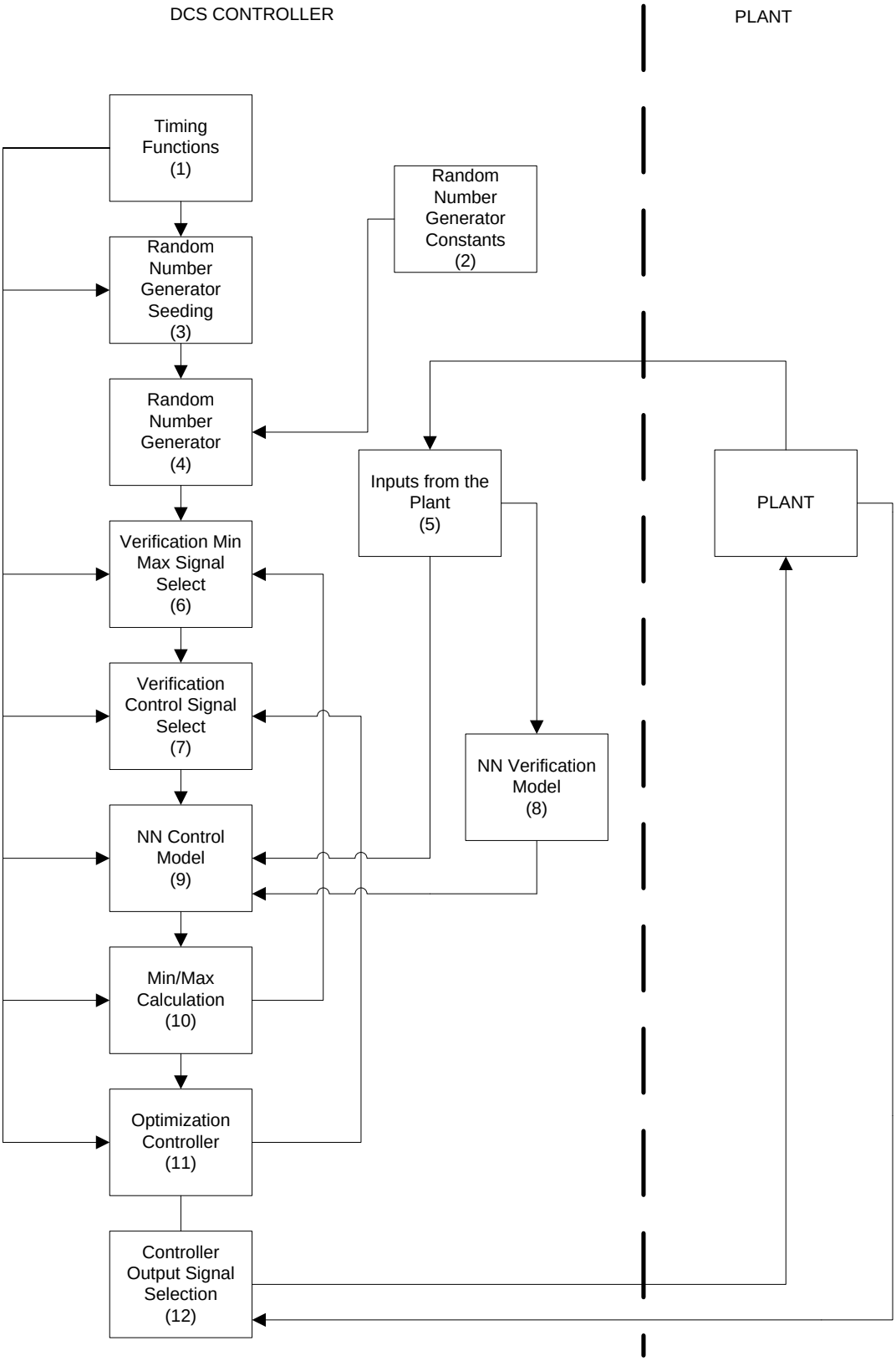


Figure 6-1 DCS Logic Order of Operation

6.2 DCS Timing Signals and Scan time

For this controller, there will be functions which will not operate on every module scan. These functions will be triggered by timing signals. There are three functions that will operate periodically which will be discussed later in more detail. Timing signals are generated using the internal DCS clock. A one scan pulse is generated at 20 seconds, one at 40 seconds, and one at 59 seconds using the seconds from the DCS clock. The “memory” function code acts like an S/R flip flop. When the seconds value for the clock is at one of the above values, the S/R is set. The digital time delay function code (TD-DIG) has a higher block number than the S/R so the output of the S/R will provide a “1” to the input of the TD-DIG. The TD-DIG will immediately provide a “1” to the “reset” on the S/R block so that on the next scan, the output of the S/R will go to “0”. The logic for the one pulse scans can be seen in Figure 6-2.

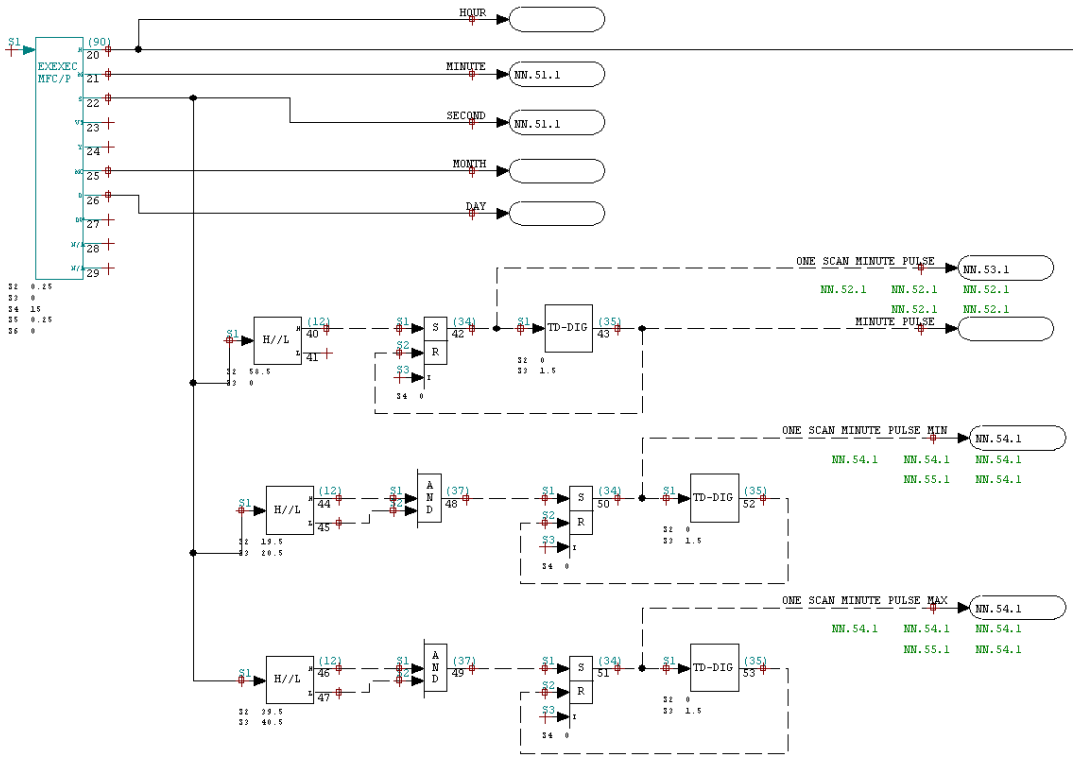


Figure 6-2 DCS Logic for timing signals

The default scan time of the controller module is 250ms. With a 250ms scan time, all blocks will be scanned four times each second. This will provide the optimization algorithm, to be discussed in more detail in section 6-9, with four different airflow input combinations per second. With these settings, the processor utilization of the BRC300 controller was less 10% so the scan time was adjusted to 100ms by using the segment control function code. This allowed for ten different input combinations of airflows per second. With these settings, the processor utilization was still less than 10%. Even though the processor utilization was less than 10%, the scan time was left at 100ms to leave room for future expansion.

6.3 DCS Random Number Generation

The random number generators will provide random inputs to the neural network for determining optimum airflow values. Five random number generators are utilized for this project. They will provide random airflow values for:

1) Intrex cell A1 airflow,
2) Intrex cell A2/A3 airflow,
3) Intrex startup channel airflow,
4) Intrex down leg airflow and
5) Intrex up leg airflow.

Each random number generator will be seeded separately at different times.

The seeding of the random number generators is done using the DCS clock. The seed is the sum of the clock seconds and minutes values scaled from 0 to 99. One random number generator is seeded every 13 seconds. This is accomplished using S/R function codes and TD-DIG function codes. The first S/R block will be set when the controller starts. The S/R block provides a “1” to two TD-DIG blocks. One TD-

DIG block sends a pulse to the associated RNG to force it to seed and the other will wait 13 seconds and then reset the S/R for the associated RNG seed and set the S/R for the next RNG seed. This will continue for each RNG and then repeat. The DCS logic for seeding the random number generators can be seen in Figure 6-3.

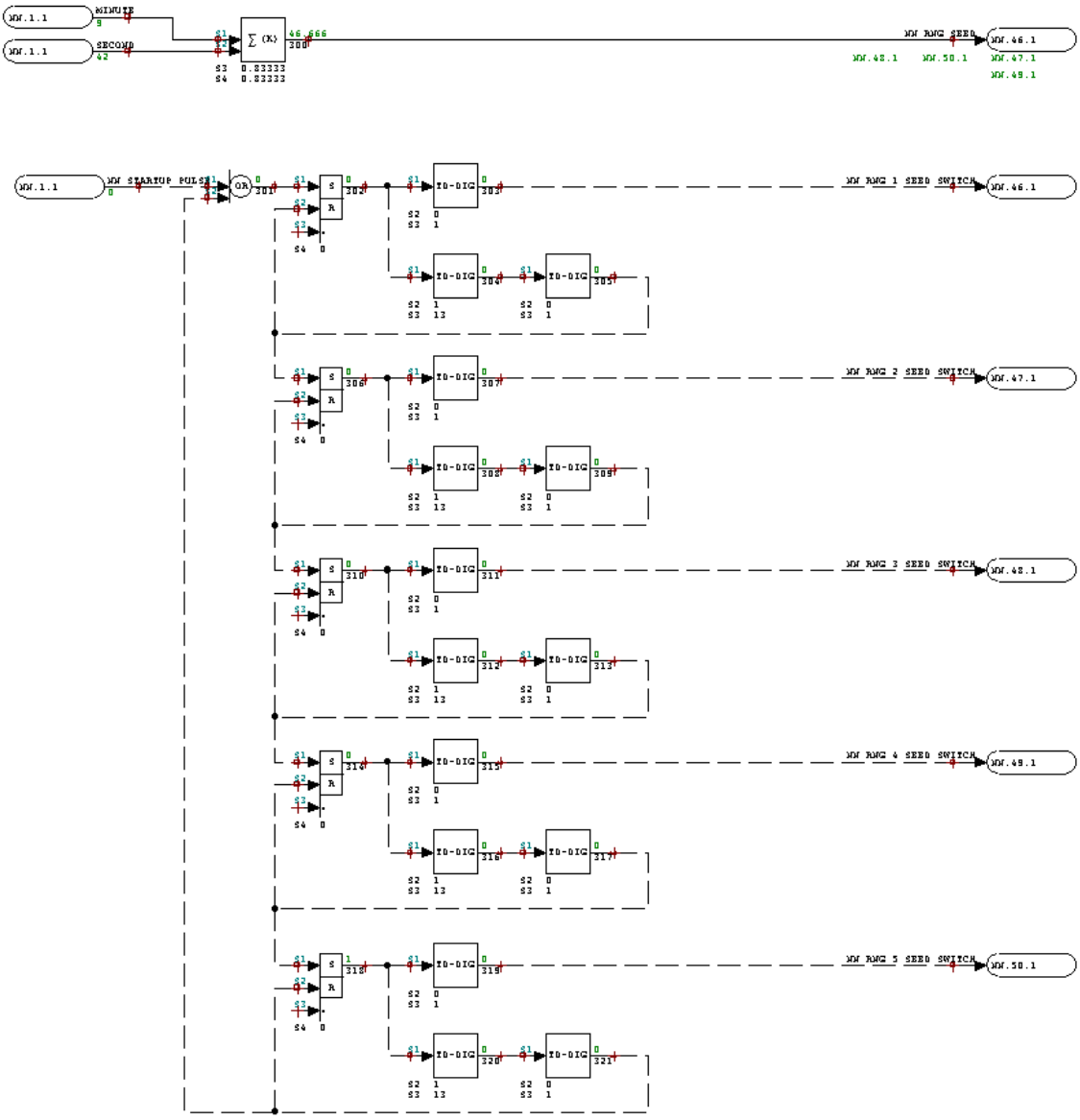


Figure 6-3 DCS Logic for Random Number Generator Seeding

The random number generators will use the seeds from the seed logic and the equation for a linear congruent number generator from equation 5-2 to generate random numbers. The benefits of using the linear congruential random number generator are that they do not require a lot of system resources and that they can be implemented using DCS function codes. The disadvantage is that there is not a dedicated DCS function code for modulus or rounding which is required for the linear congruential random number generator.

In order to get a rounded value, a series of multiplexer function codes were used. The multiplexer will round the input select value in order to select an input. The multiplexers were combined to generate a multiplexer with 100 inputs with a constant from 0 – 99 attached to each input. When the value to be rounded is used as the input to the multiplexer, the rounded value is generated at the output. The logic for the linear congruential random number generator can be seen in Figure 6-4.

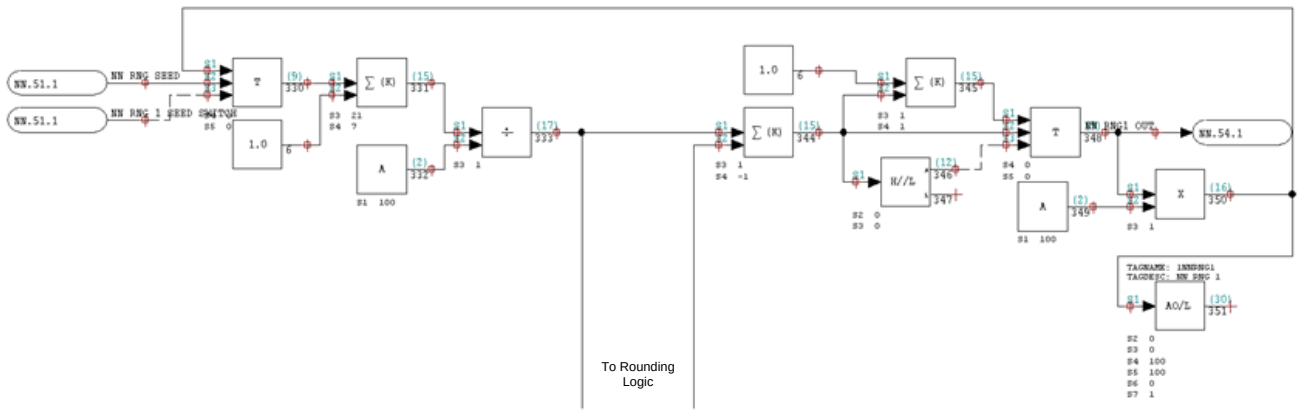


Figure 6-4 DCS Logic for Random Number Generation

6.4 DCS Signal Inputs and preprocessing

The ABB infi-90 DCS has a multi-level communication structure. The top level of communications is the plant loop. Process Control Units (PCU's) communicate with each other over the plant loop. In some cases, multiple loops can be tied together so PCU's can communicate with PCU's on other loops. The PCU's on a loop each have a loop address and a unique PCU address. Each PCU contains one or more controller modules and a communication module that ties the controllers to the loop. The controllers and communication modules communicate with each other using a communications bus called controlway. Each communication and control module in a PCU has a distinct controlway address. If a controller has associated field input modules, it communicates with those modules over an I/O expander bus.

Since the controller for this project is being tied into a pre-existing control system, the controller inputs will come from other controllers over the DCS communication system and not directly from field inputs. Input signals are brought into the controller from modules in other PCU's using analog loop input (AI/L) function codes. The AI/L function code uses the PCU address, control module controlway address and function code block number for the analog output function code (AO/L) in the PCU where the signal originates.

Once the signals are brought into the controller that will be used for the neural network, they are checked for validity and averaged where averaging is used. Any signals that are found to have bad quality resulting from communications or instrument failure will automatically be removed from any average that they are calculated into. An on/off block was also added so a signal could be "forced" out of the average if it was not indicated as bad quality but still was not reading correctly. If one of the non

redundant inputs or all of a redundant set of inputs go to bad quality or are forced out using the on/off block, the neural network model will become invalid and the logic will trigger a bad quality alarm that will automatically bypass the neural network controller. This will also occur if all of the signals for an averaged input go bad quality or are forced out of the average. The bypass logic will be discussed further in section 6-10.

The signals are normalized using equation 3-5 with the same standard deviations and averages that were used for normalization in the model development. Any signals that utilize delayed values also have the five minute delay values generated. Figure 6-5 shows the above functions programmed into the DCS using DCS function codes for the average freeboard signal. The normalized signals are tied to the inputs of the neural network model.

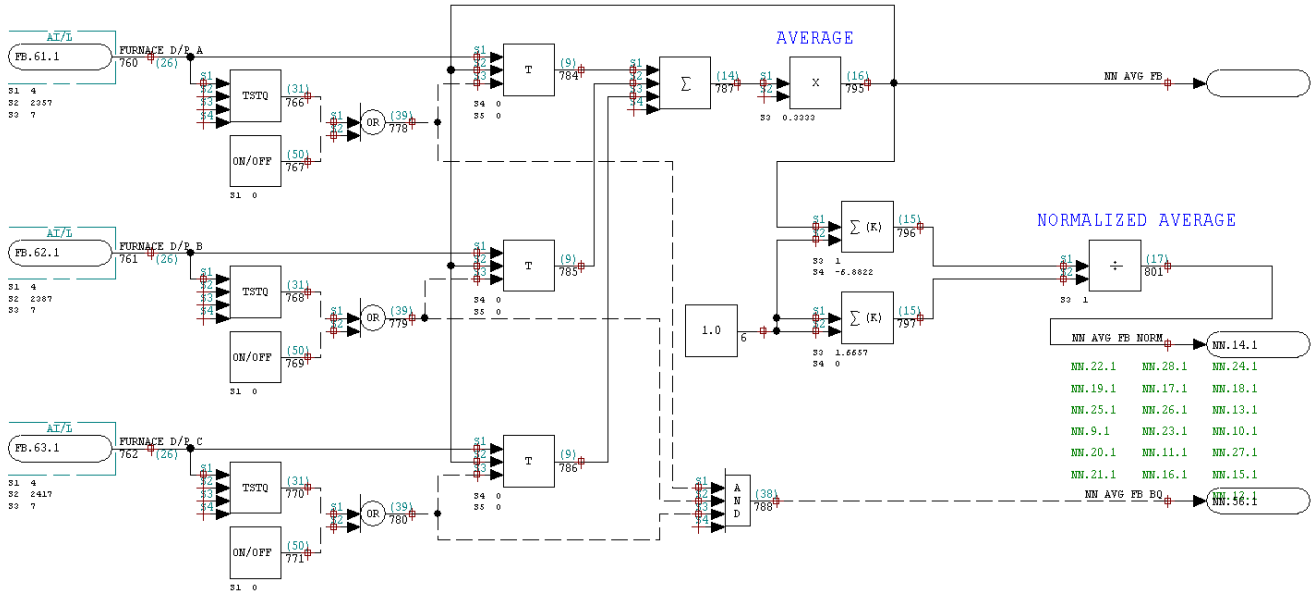


Figure 6-5 DCS Logic for Signal Input and Preprocessing

The only signal that utilized a time delay was the total fuel flow. The time delayed value of the fuel flow was generated using a “Delay” function code. The Delay function code was set up to delay the total fuel flow by five minutes and sample every two seconds. The logic for the time delayed fuel flow input can be seen in Figure 6-6.

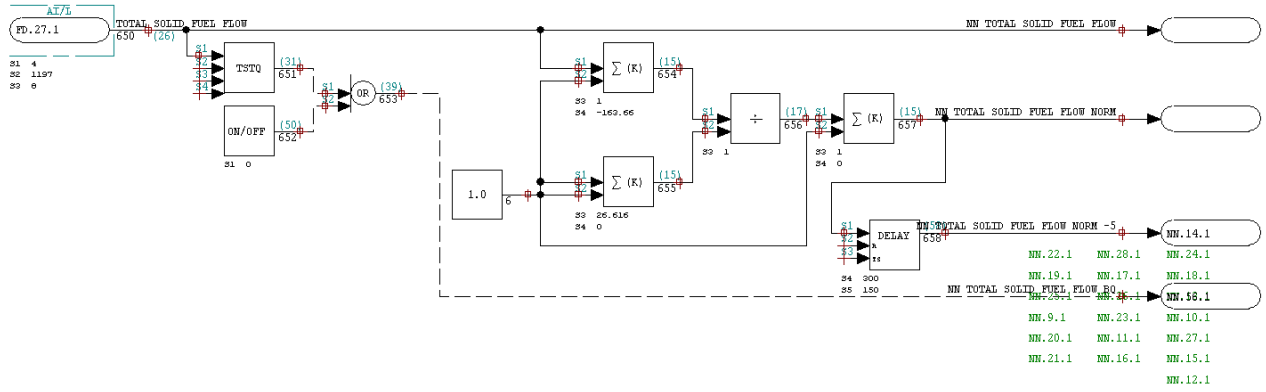


Figure 6-6 DCS Logic for time delayed inputs

6.5 DCS Minimum/Maximum Intrex Differential Temperature Airflow Verification Signal Selection

The controller will calculate the minimum and maximum intrex differential temperature that can be achieved by manipulating the airflows. The method of determining these values will be discussed in section 6-8. These values will be reapplied to the input of the neural network control model once every 60 seconds to re-verify the values.

Each air flow signal has a set of selection logic. The output from the associated random number generator (0-99) is converted to an airflow value. The value is then transferred to the input of an

“analog transfer” function code. Under normal operation this signal is passed through the analog transfer block and then passes through a second analog transfer block on to the control airflow verification signal selection logic. The first analog transfer is switched to the airflow values that are currently saved as those that generate the lowest intrex differential temperature for a single scan by the timing signal that pulses at 40 seconds. Under this condition, those values will be passed to the input of the neural network models. The second analog transfer is switched to the airflow values that are currently saved as those that generate the highest intrex differential temperature for a single scan by the timing signal that pulses at 20 seconds. Under this condition, those values will be passed to the input of the neural network models. The min/max switching logic for the intrex A1 cell airflow can be seen in Figure 6-7.

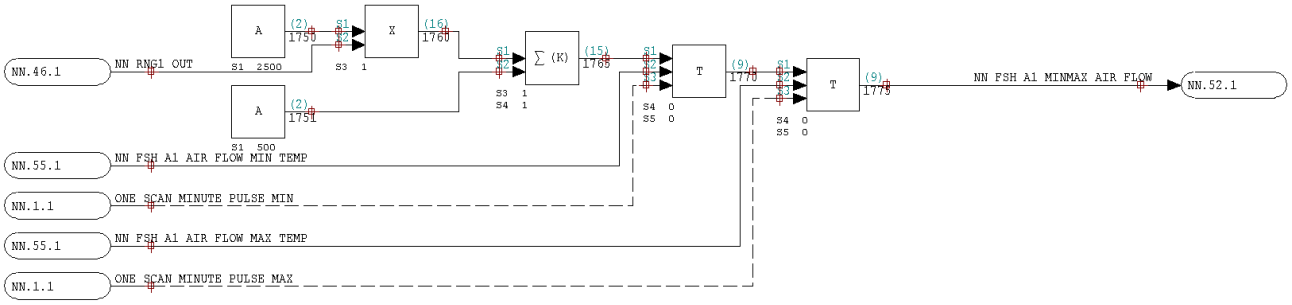


Figure 6-7 DCS Logic for Min/Max Intrex Differential Temperature Airflow Verification Signal Selection

6.6 DCS Control Airflow Verification Signal Selection

The controller will calculate the airflows required to achieve the operator entered intrex differential temperature set point. The method of determining these values will be discussed in section 6-9. These values will be reapplied to the input of the neural network control model once every 60 seconds to re-verify the values.

Each air flow signal has a set of selection logic that uses the output of the min/max selection logic and the control airflow signal as inputs to an analog transfer function code. Under normal operation the signal from the min/max airflow selection logic is passed through the analog transfer function code. The analog transfer is switched to the currently saved control airflow values for a single scan by the timing signal that pulses at 59 seconds. The value that is passed through the analog transfer is then normalized using the same normalization parameters used in model development. The normalized airflows are used as inputs for the neural network control model discussed in section 6-7. The control airflow switching logic for the intrex A1 cell airflow can be seen in Figure 6-8.

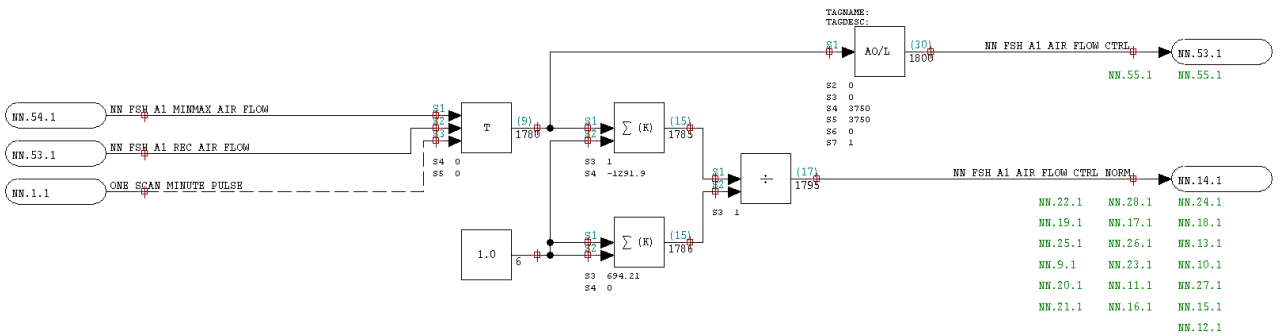


Figure 6-8 DCS Control Airflow Verification Signal Selection

6.7 DCS Neural Network Model Logic

The Neural Network model predictive controller has two neural network models. The first model, the verification model, is used to verify model accuracy and uses all inputs from the plant. The second model, the control model, uses inputs from the plant and the intrex airflow inputs from the control airflow signal selection logic in Figure 6-8. Each model has the same model structure as the neural network model generated in Matlab.

Each input into the neural network nodes in each layer is applied to a two-input summing function code. This function code has a programmable gain for each input which is where the neural network weights will be programmed. The outputs of each of the summing nodes are summed together with each other as well as with the node constant. That value is then passed on to a tan sigmoid activation function. The equation for the tan sigmoid activation function can be seen in equation 4-1. A single input layer node for the verification neural network model can be seen in Figure 6-9.

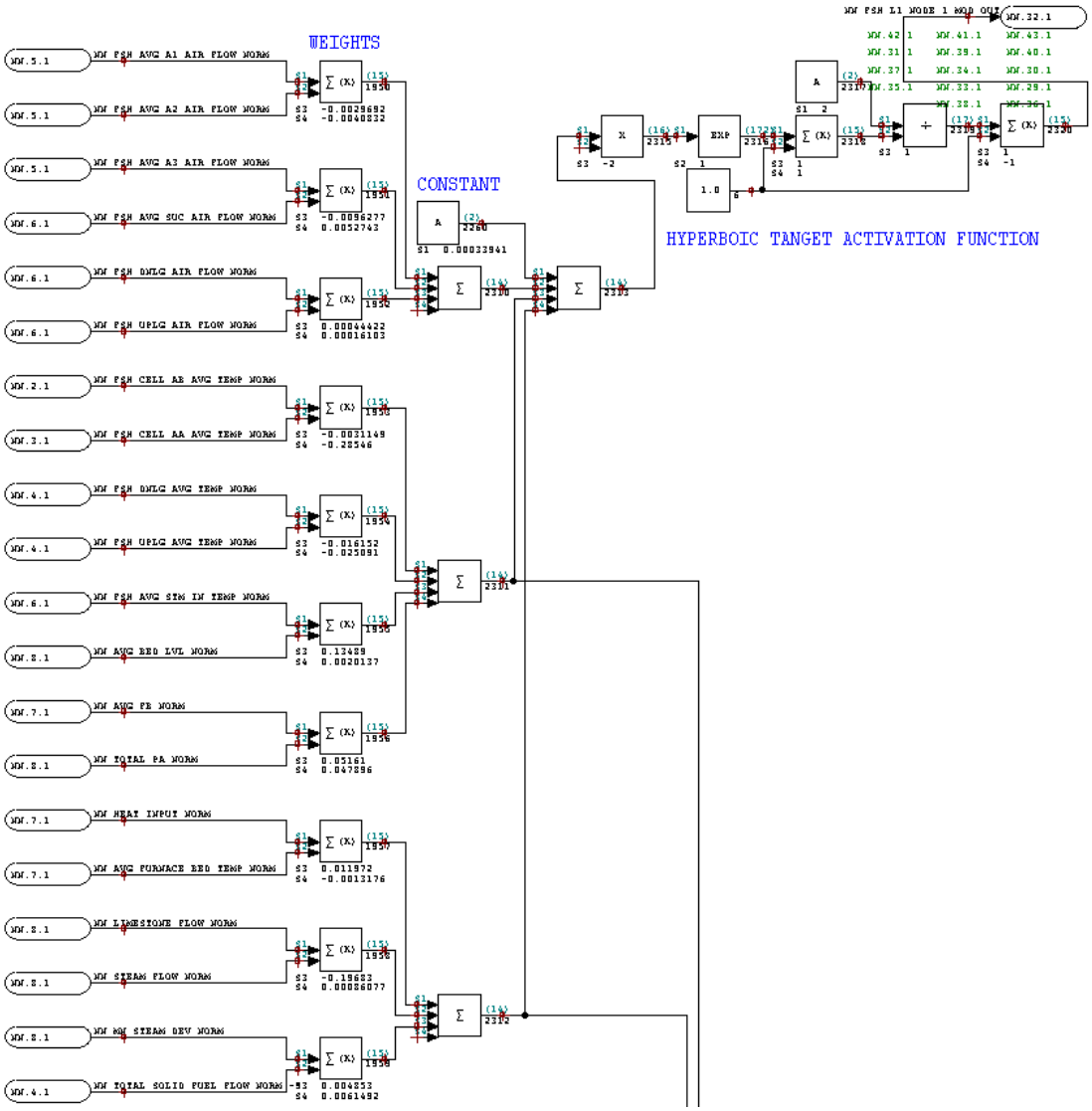


Figure 6-9 DCS Verification Neural Network Model Layer 1 Node

The input layers for the neural networks each have twenty nodes. In order to reduce the number of controller blocks required to implement the neural network models, common inputs are shared for the layer 1 logic. The only inputs that are not shared between the verification neural network model and the control neural network model are the intrex air flows. A single input layer node for the control neural network model can be seen in Figure 6-10.

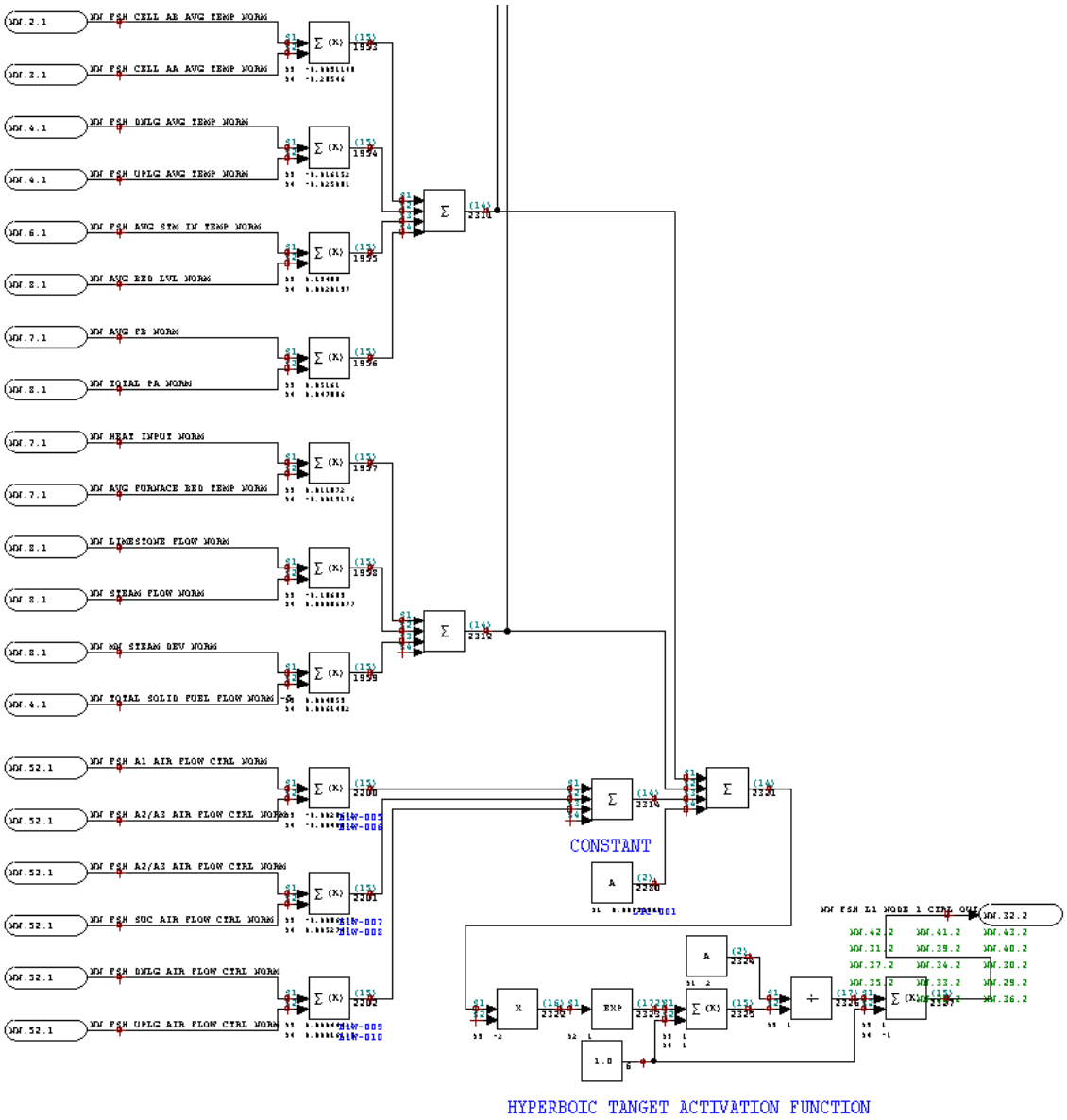


Figure 6-10 DCS Control Neural Network Model Layer 1 Node

Every output value from the layer 1 nodes is used as an input values for each of the layer 2 nodes for the same model. The layer 2 nodes for the verification and control neural network models are independent and do no share any input or output values. Each model has fifteen layer 2 nodes. The layer 2 nodes for both models have the same structure. A layer 2 node for the verification neural network model can be seen in Figure 6-11.

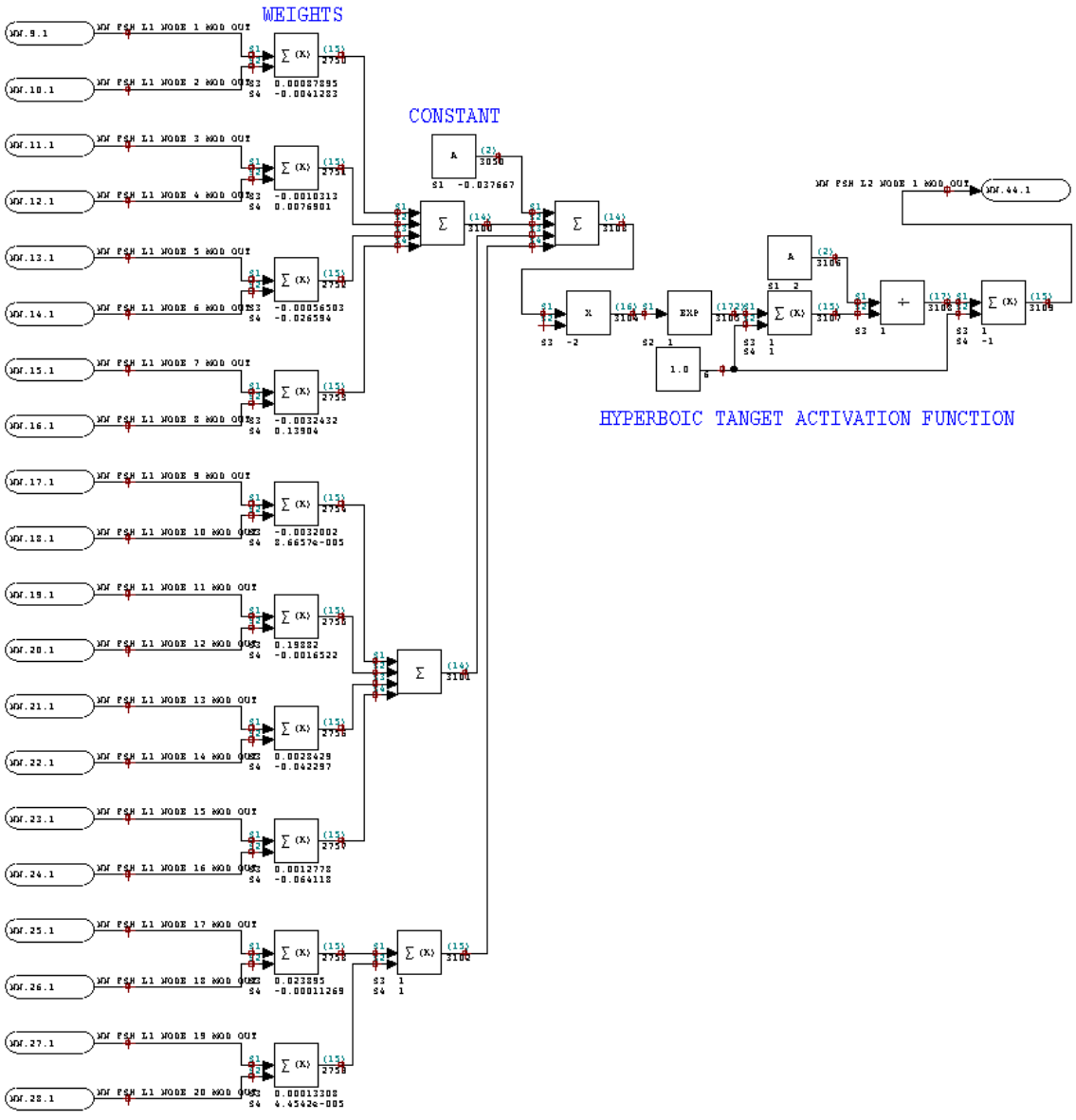


Figure 6-11 Verification Neural Network Model Layer 2 Node

The outputs from the fifteen layer 2 nodes are each used as an input to the output layer node. There is no tan-sigmoid activation function on the output layer nodes. The output values are un-normalized by reversing equation 3-5 and using the same standard deviation and average values used for normalization. The output layer for the verification model provides a value for the intrex differential temperature that is compared with the plant intrex differential temperature. The difference between the verification model output and the plant value for intrex differential temperature is calculated and passed to the control model. The output node for the verification neural network model can be seen in Figure 6-12.

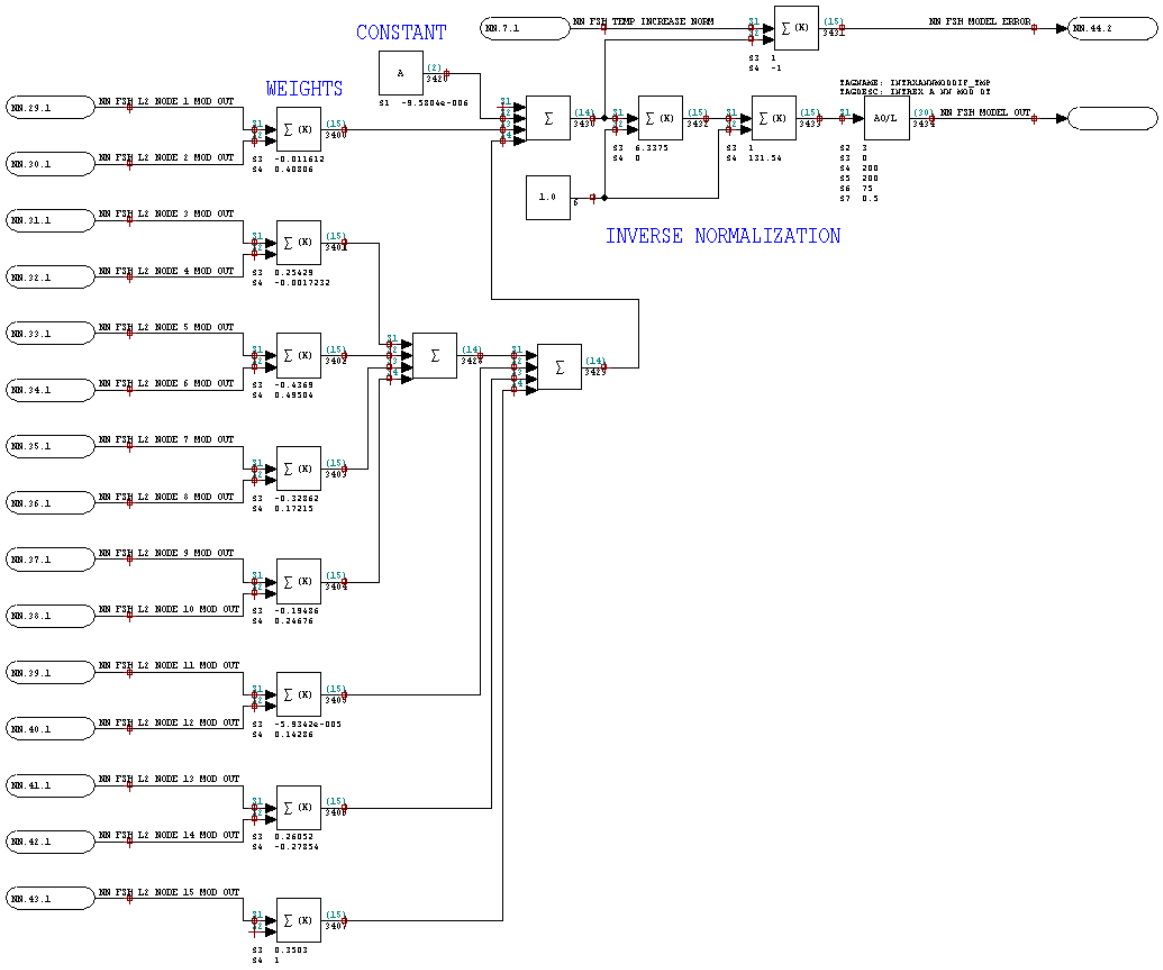


Figure 6-12 DCS Verification Neural Network Model Output Node

The verification model error value is added to the output of the last summing block in the Control model output node so the error signal of the control neural network can be properly calculated. This function will not compensate for a poorly performing model. It is only meant to fine tune the controller by a few degrees. If the verification model error signal is large, it will have a greater impact on the control model output than the intrex airflows that are being tested and the controller will not function properly. The control neural network model output node provides values to be used for calculating minimum and maximum controller capabilities as well as for calculating the optimum intrex airflows required to meet the operator entered setpoint for intrex differential temperature. The control neural network output node can be seen in Figure 6-13.

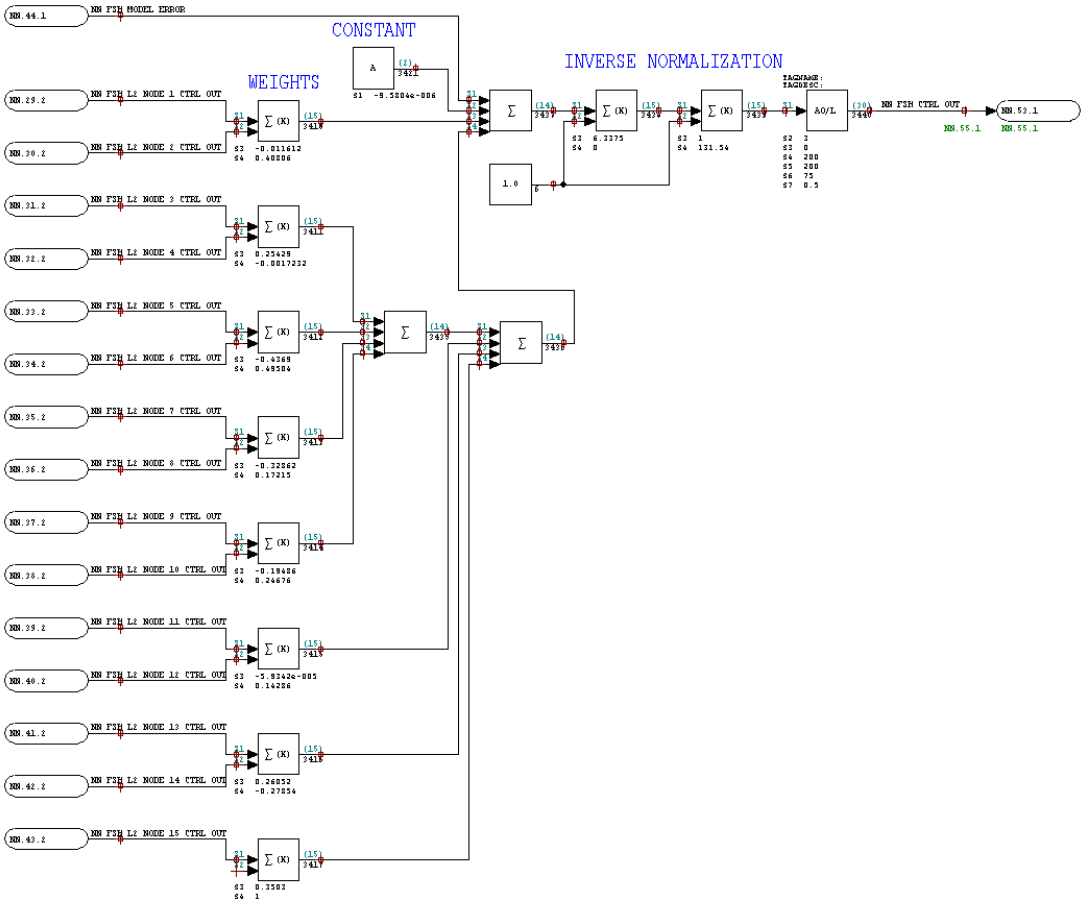


Figure 6-13 DCS Verification Neural Network Model Output Node

Each Neural network model contains 751 weights and constants. Manually entering these numbers would be very time consuming and one wrong entry will make the model malfunction. In order to ensure that the weights and constants were entered properly, they were exported from Matlab into an excel spreadsheet. The DCS Control Logic was exported into an access database. The block numbers for the weights and constants were ordered in groups so each group of weights and constants could be copied and pasted from the excel spreadsheet into the access database. The weights and constants were pasted into the access database and then the updated access database was imported back into the DCS which applied all of the weights and constants to the neural network models.

6.8 DCS Intrex Differential Temperature Minimum/Maximum Capability Calculations

In order to determine the minimum intrex differential temperature that can be achieved using the model predictive controller, the error between the output value of control model and zero set point is calculated. The current error is compared with the saved best error value. If the current error is better than the previously saved error, it will be stored along with the control model output and the airflows that provide the minimum intrex differential temperature will be updated.

One potential problem that arrives from this configuration is that when the plant parameters change in a way that causes the error to increase, a better error may not be possible and the past error is no longer relevant. In order to overcome this problem, the stored airflows that provide the minimum intrex differential temperature are reapplied to the model once every 60 seconds and the associated error value is updated. This is accomplished in a single scan using the single scan pulse that activates when the DCS clock is at 40 seconds. The logic for selecting the airflow values that generate the minimum intrex differential temperature can be seen in Figure 6-14.

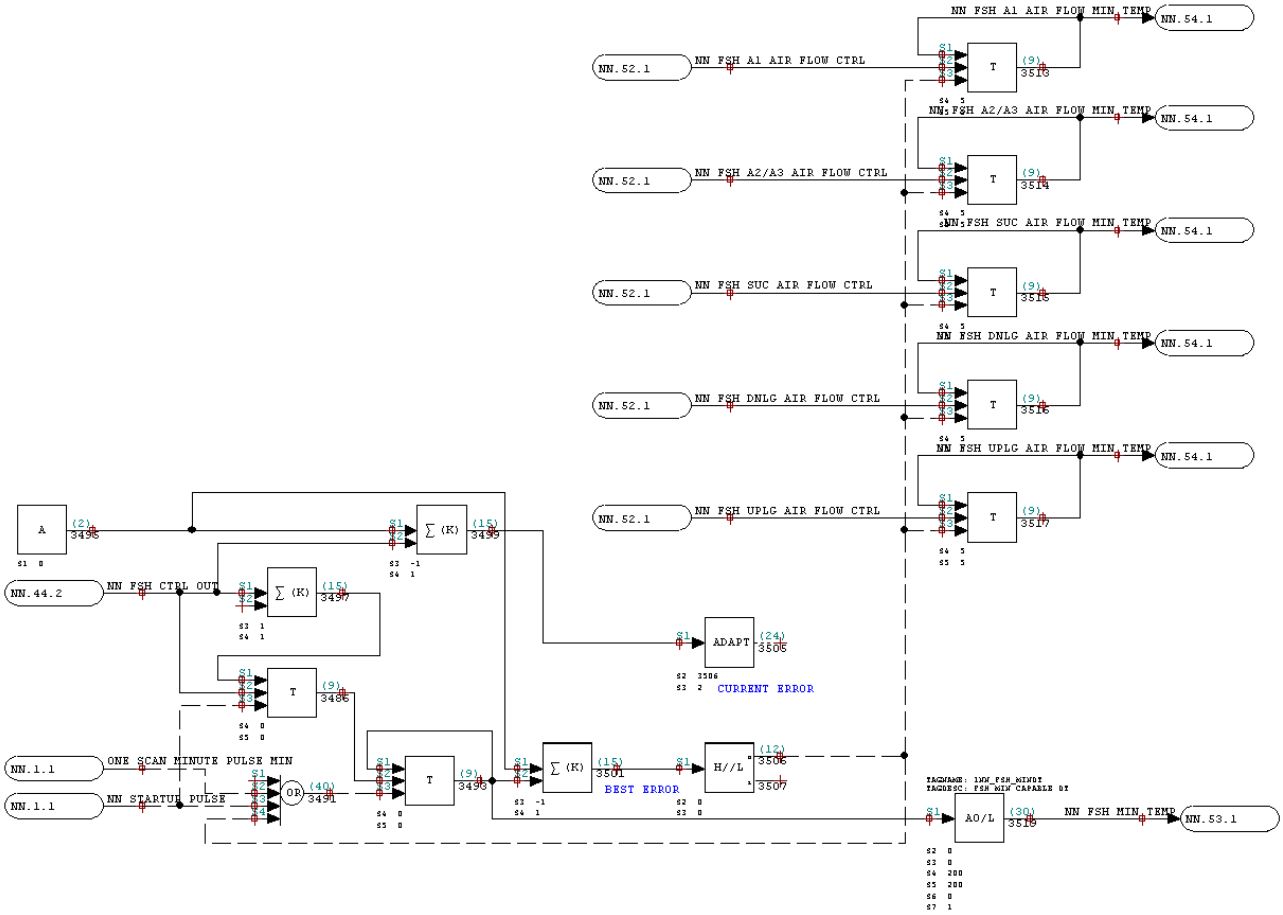


Figure 6-14 DCS Calculation of the Airflow Values for Minimum Intrex Differential Temperature

In order to determine the maximum intrex differential temperature that can be achieved using the model predictive controller, the error between the output value of control model and set point of 300 is calculated. The current error value is used to capture the airflows and error associated with the airflows required for the maximum intrex differential temperature in the same manner as that used for the minimum intrex differential temperature. The error and airflows for the maximum intrex differential temperature are verified once a minute by the single scan pulse that activates when the DCS clock is at 20 seconds. The logic for selecting the airflow values that generate the maximum intrex differential temperature can be seen in Figure 6-15.

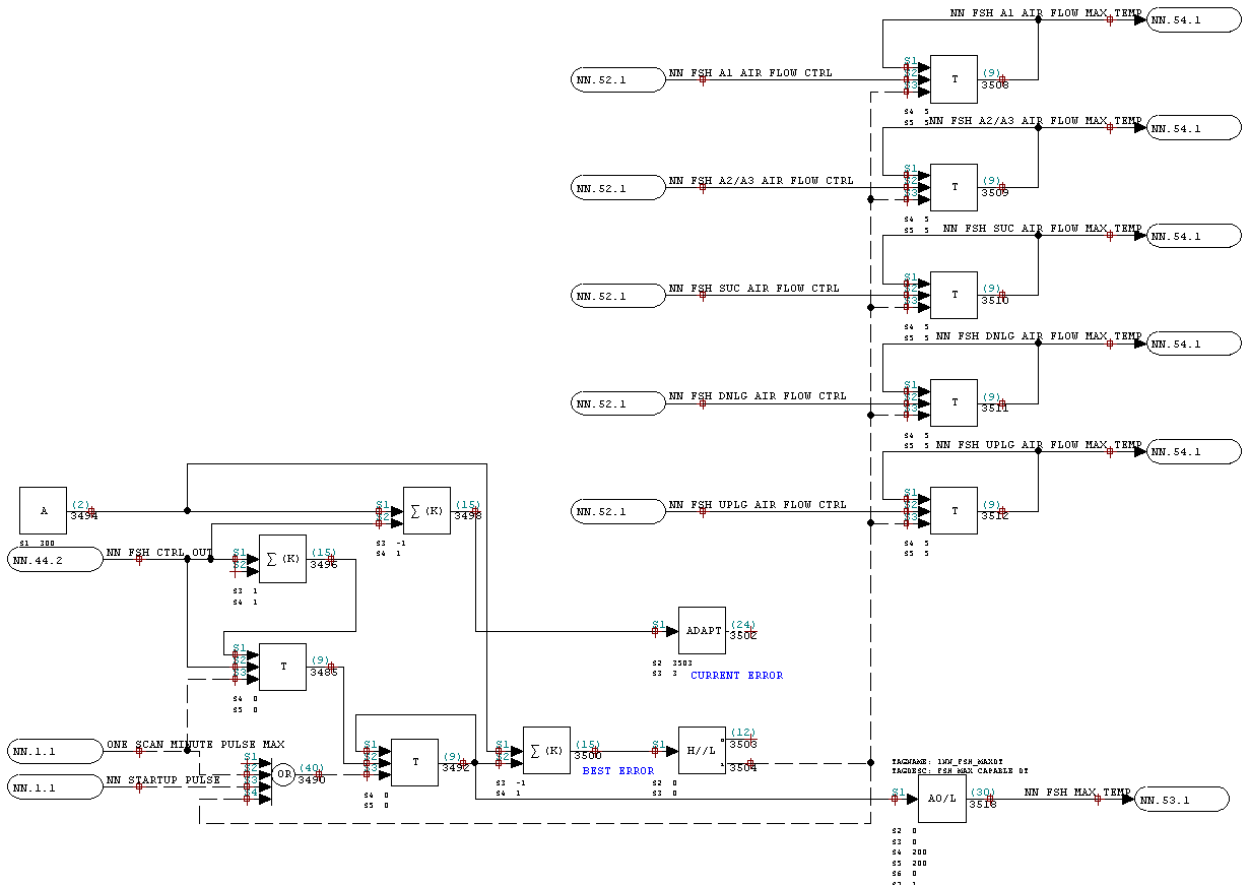


Figure 6-15 DCS Calculation of the Airflow Values for Maximum Intrex Differential Temperature

6.9 DCS Control Optimization

This controller will allow the operator to enter a setpoint for the desired intrex differential temperature. The setpoint is compared with the minimum and maximum capabilities of the controller. If the setpoint falls outside of the range that the controller is capable of controlling to, the closest value to the setpoint within the range will be selected as the setpoint and an alarm will be issued to alert the operator. The setpoint is compared to the control model output in order to generate an error signal. The current error is compared with the saved best error value. If the current error is better than the previously saved error, it will be stored along with the control model output and the airflows that provide the intrex

differential temperature closest to the setpoint will be updated. The portion of the logic used for selecting a setpoint can be seen in Figure 6-16.

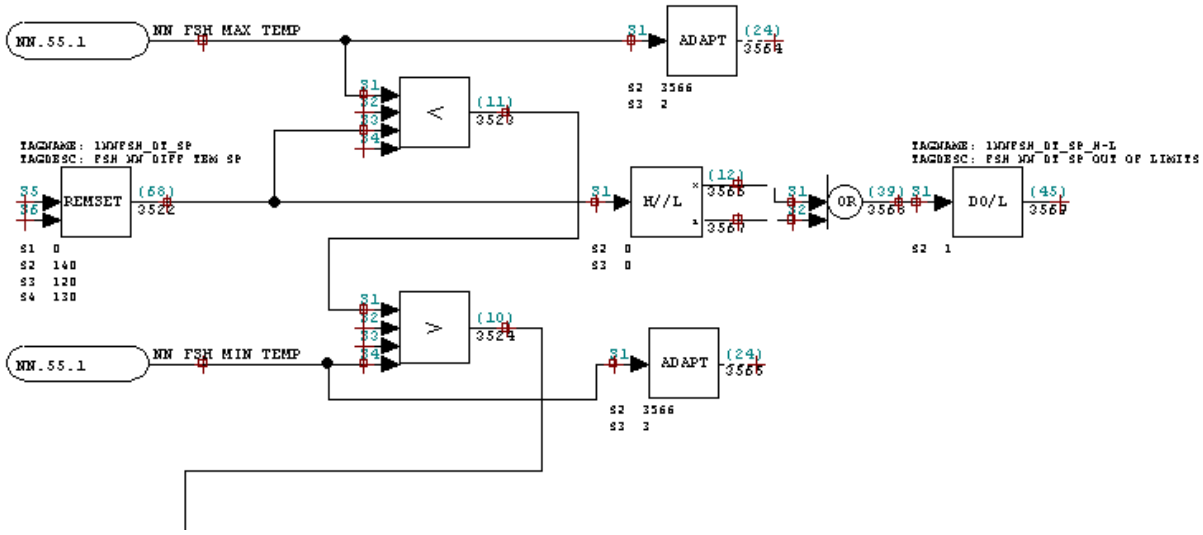


Figure 6-16 DCS Controller Setpoint Selection

As with the min/max error calculations there is the potential problem that when the plant parameters change in a way that causes the error to increase, a better error may not be possible and the past error is no longer relevant. This problem is overcome by reapplying the stored airflows that provide the lowest error between intrex differential temperature and the setpoint to the control model once every 60 seconds and updating the associated error value. This is accomplished in a single scan using the single scan pulse that activates when the DCS clock is at 59 seconds. The logic for selecting the airflow values that generate the minimum error between intrex differential temperature and the setpoint can be seen in Figure 6-17. The logic also contains a dead band that is set to +/- .25 degrees F. This is to prevent the airflows from changing unnecessarily.

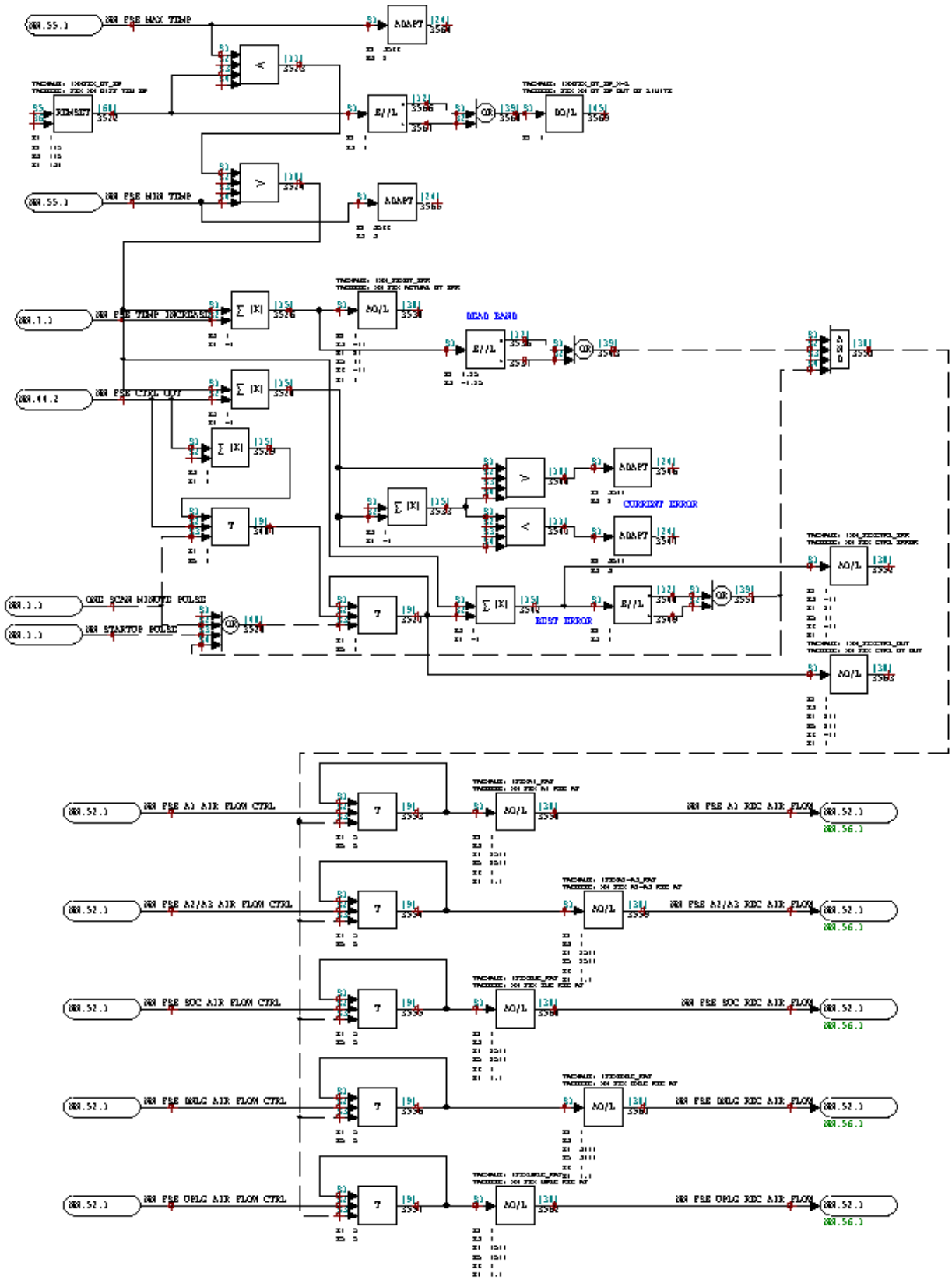


Figure 6-17 DCS Control Optimization Logic

6.10 DCS Controller Output Signal Selection

The current logic for the intrex airflows uses curves that apply a specific airflow setpoint to a PID controller for specific unit loads. The logic already contains a transfer switch that was installed previously. A “remote control memory” (RCM) function code is used to allow the operator to switch back and forth between the new neural network control and the airflow curves that are already installed. In order for the operator to be able to turn on the neural network controls, at least one input signal for each neural network input must be good quality. If all of the input signals for any neural network input go bad quality, the neural network will automatically turn off. The logic for selecting the neural network controller can be seen in Figure 6-18.

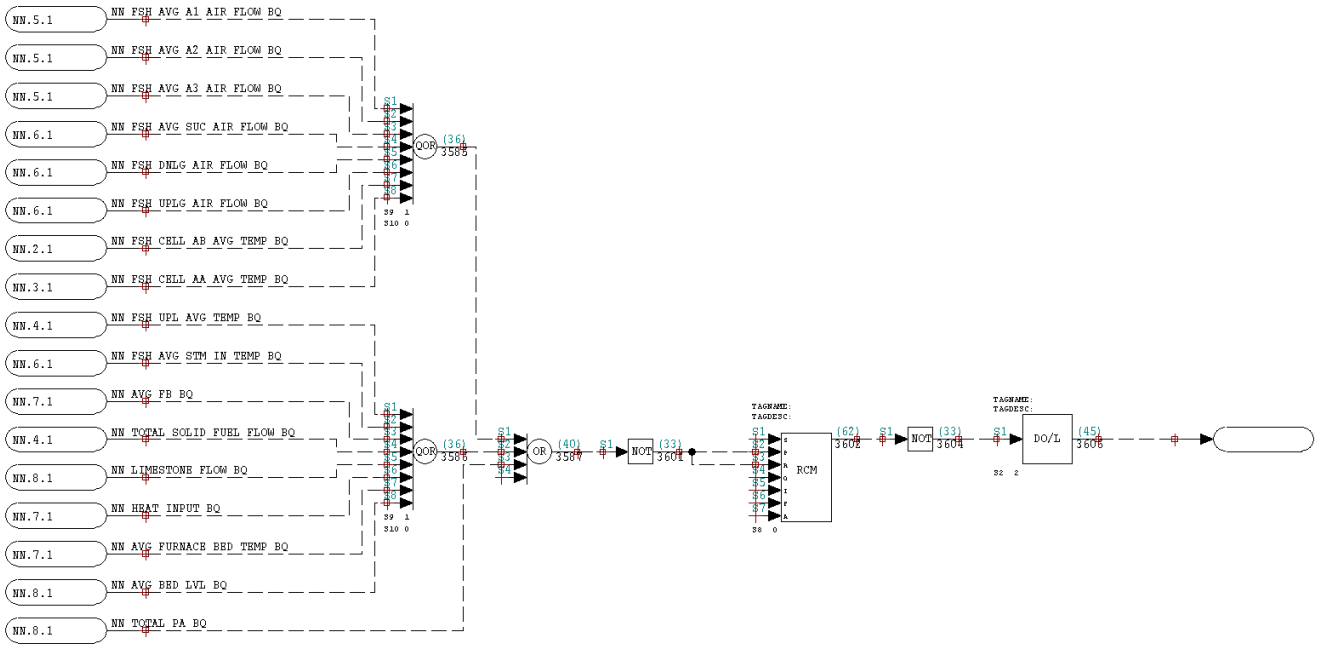


Figure 6-18 DCS Neural Network Controller On/Off Logic

In addition to intrex differential temperature, it is very important to maintain ash flow through the intrex. If the ash flow through the intrex stops, the ash will continue to build up in the cyclone. Without the circulation of ash through the hot loop, effective heat transfer cannot take place and the unit will have to come off line. In order to prevent cyclone plugging, a flush function was added to the airflow selection logic. There are five pressure indications in the inlet of the intrex. When the pressure indications are negative intrex ash flow is good. When all of the pressure indications are positive intrex ash flow is poor. The flush sequence will increase all of the airflow values until the pressure indications show that the intrex ash flow has improved. The flush will last no less than five minutes. The flush sequence will take place if four of the five pressure indications are positive for 30 seconds, all five or the pressure indications are positive, or four of the pressure indications read a pressure greater than 5" of water. The DCS logic for flushing the intrex can be seen in Figure 6-19.

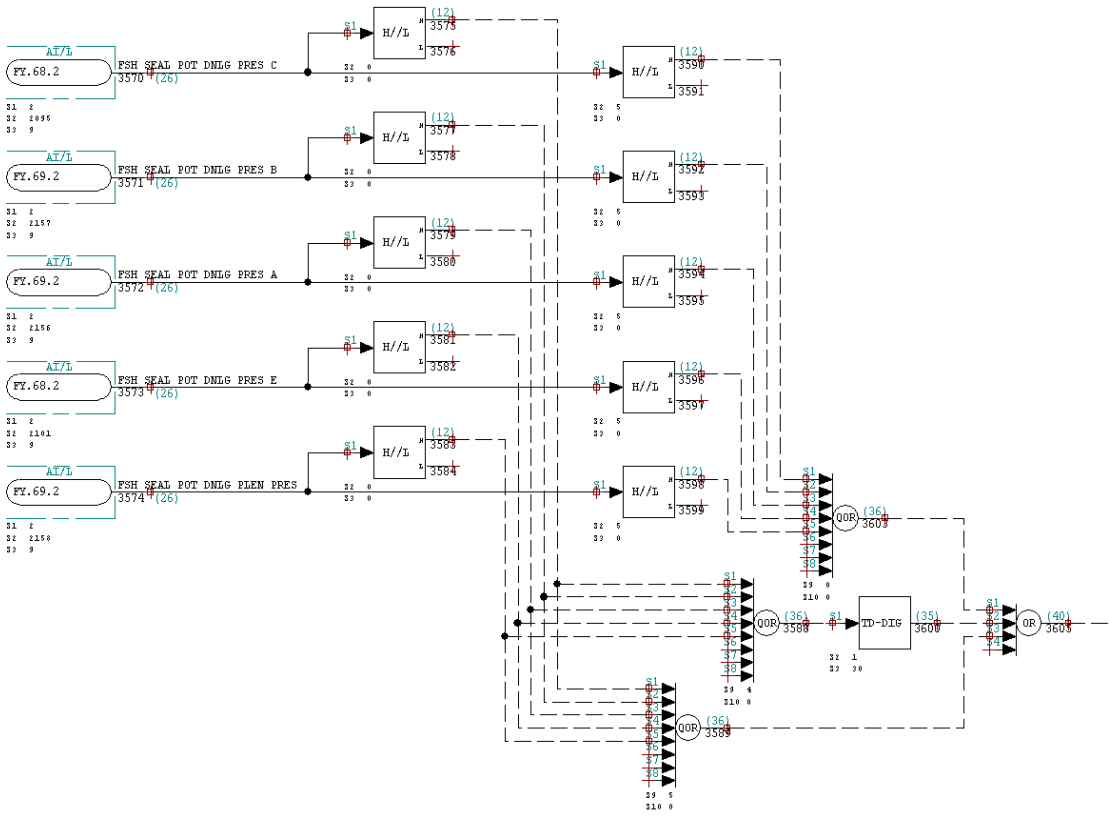


Figure 6-19 DCS Intrex Flush Logic

The output of the flush logic will stay active for at least five minutes once it is activated before it the system will switch back to normal control. This is to prevent system instability that may result from the plugged cyclone detection turning on and off if the system is operating near the threshold. To increase stability, a lag function was also added to keep the airflows from changing too rapidly. The lag blocks are set to allow the airflows to reach 63% of their change in value in 10 seconds and 99% of their change in value in 50 seconds. This logic can be seen in Figure 6-20 along with the AO/L blocks that will be used as the inputs into the live system.

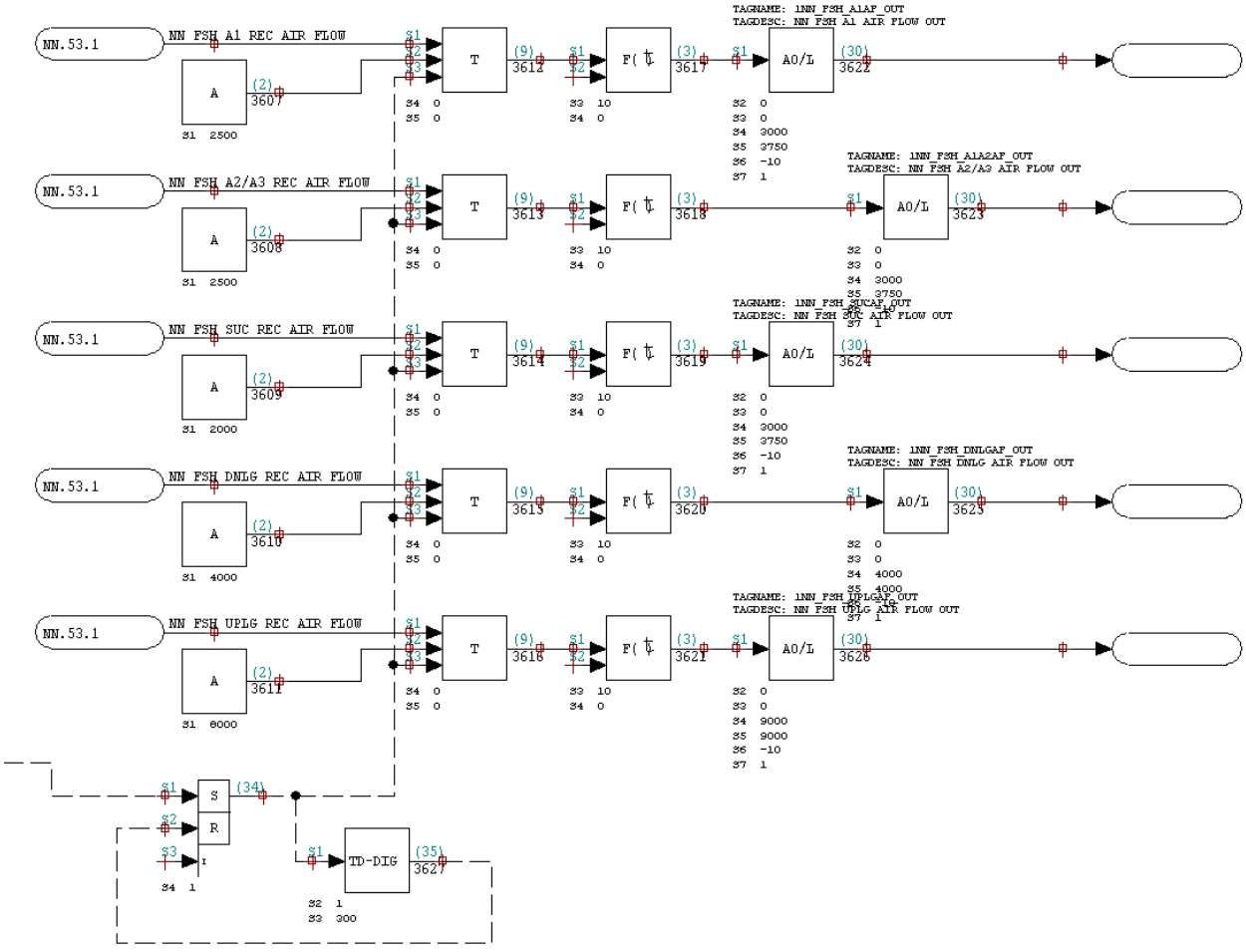


Figure 6-20 DCS Neural Network Controller Output to Plant Logic

Chapter 7 : Testing and Results

In order to verify that the neural network model was functioning properly in the DCS, the output of the verification neural network model was compared to the actual intrex differential temperature. A DCS trend was generated to compare the model performance to the plant. The largest deviation between the signals reached nearly 2.5 degrees F during a time that the actual intrex differential temperature was climbing rapidly. The deviation between the signals was less than 0.5 degrees F for the majority of the timeframe. The neural network model tracks the live plant with enough accuracy for the neural network model predictive controller to function properly. The trend of the plant intrex differential temperature and the verification Neural Network Model can be seen in Figure 7-1.

To verify the operating range of the neural network, the minimum and maximum intrex differential temperatures generated by the controller were compared to the plant intrex differential temperature. A DCS trend was generated to verify the neural network model predictive controller had a sufficient controllable range. The results show that under most circumstances the controller will have the ability to control the intrex temperature within a range of 5 degrees F. This is sufficient for the purposes of this project and can be beneficial to the plant. The range may be increased by performing additional airflow testing and using that data to re-tune the neural network. A trend of the plant intrex differential temperature vs. the minimum and maximum ranges of the controller can be seen in Figure 7-2.

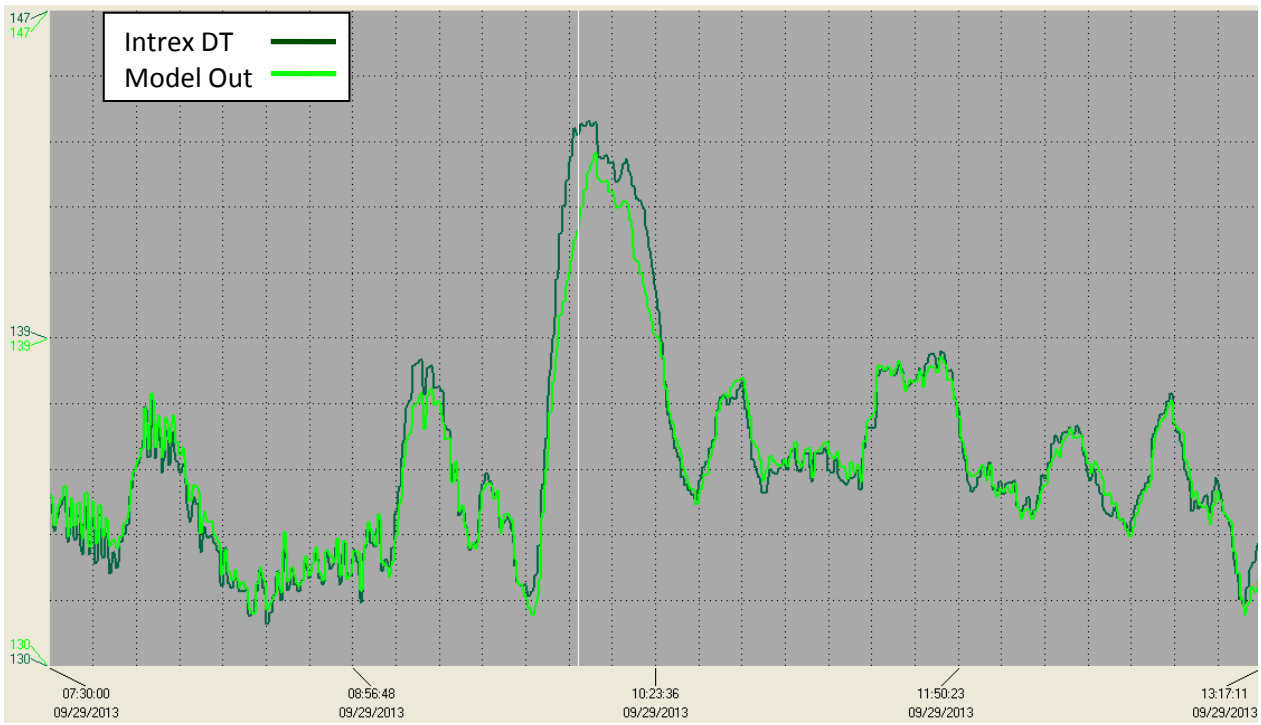


Figure 7-1 Intrex Differential Temperature vs. Verification Neural Network Model Output

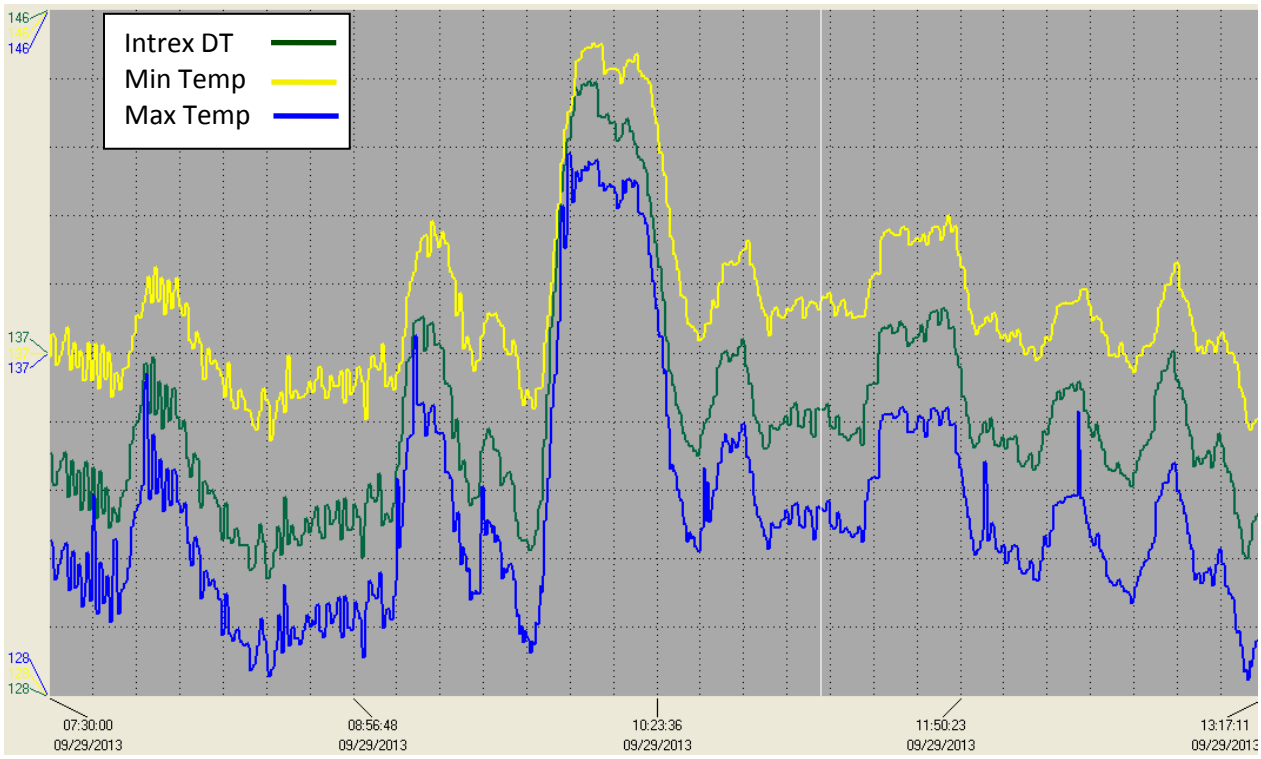


Figure 7-2 Neural Network Control Min/Max Capabilities vs. Intrex Differential Temperature

It can be seen in Figures 7-1 and 7-2 that the plant intrex differential temperature is continuously changing. With this temperature continuously changing, the main steam attemperator valves have to continuously modulate to try to control main steam temperature. If the intrex differential temperature was constant, the control of the main steam temperature would be more stable.

The set point for the neural network model predictive controller was set to 136 degrees F and left at that state for approximately 90 minutes. During this time, the actual intrex temperature was compared with the output of the neural network control model. The control model is the model that the optimization algorithm applies random airflows to in order to try to reach the operator entered intrex differential temperature. Figure 7-3 shows the output of the actual intrex differential temperature, the neural network control model output and the airflow values that would be applied to keep the neural network control model output at the set point.

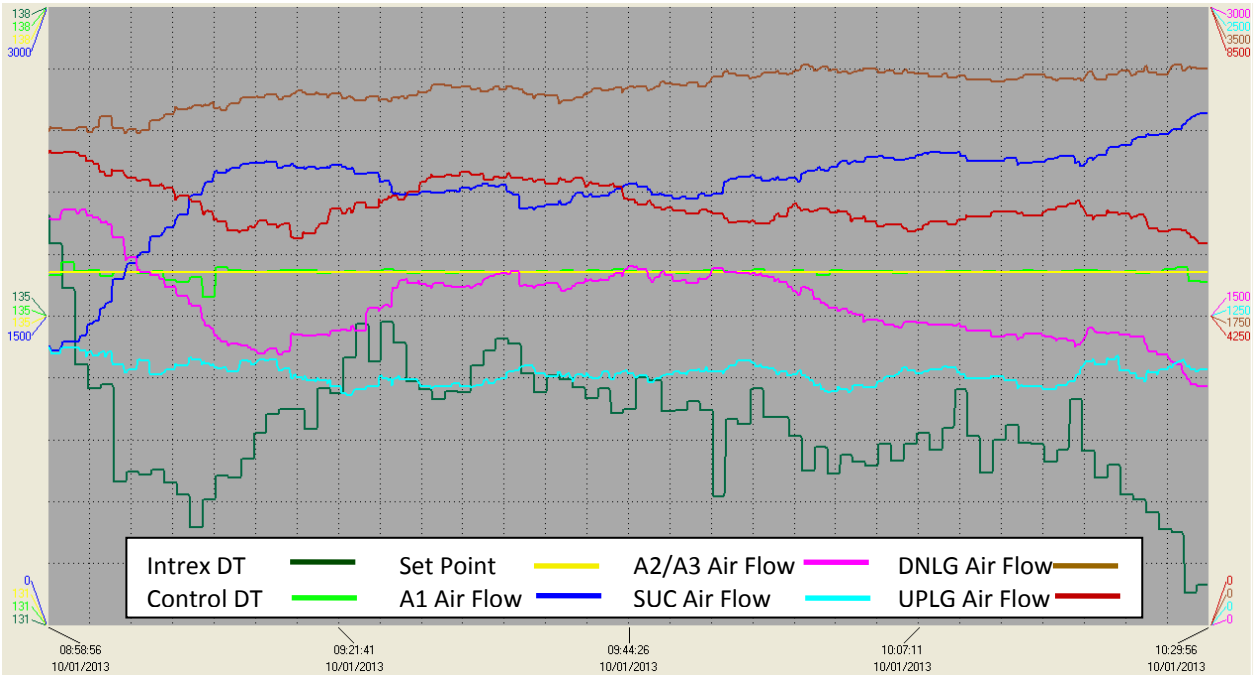


Figure 7-3 Intrex Differential temperature vs. Controller Model Output and Optimized Air Flows

It can be seen in Figure 7-3 that the controller optimization algorithm continuously modulates the intrex airflows for the controller model to keep the output of the controller model at the 136 degree F setpoint. The controller model output is able to be controlled to within +/-.1 degrees F for the majority of the time period. The maximum deviation was approximately -0.5 degrees F at approximately 9:10 AM. This was the result of the plant parameters changing to the point that the maximum controllable temperature fell below the setpoint for a short time. Another timeframe of approximately 90 minutes with a setpoint of 33 degrees F is plotted in Figure 7-4. During this timeframe it can be seen that the controller setpoint is often higher than the maximum controllable temperature and the neural network controller model output trends below the setpoint during these circumstances. With a setpoint near the edge of the controllable range, the output is less stable than that seen in Figure 7-3 but the system will still control to the setpoint when able.

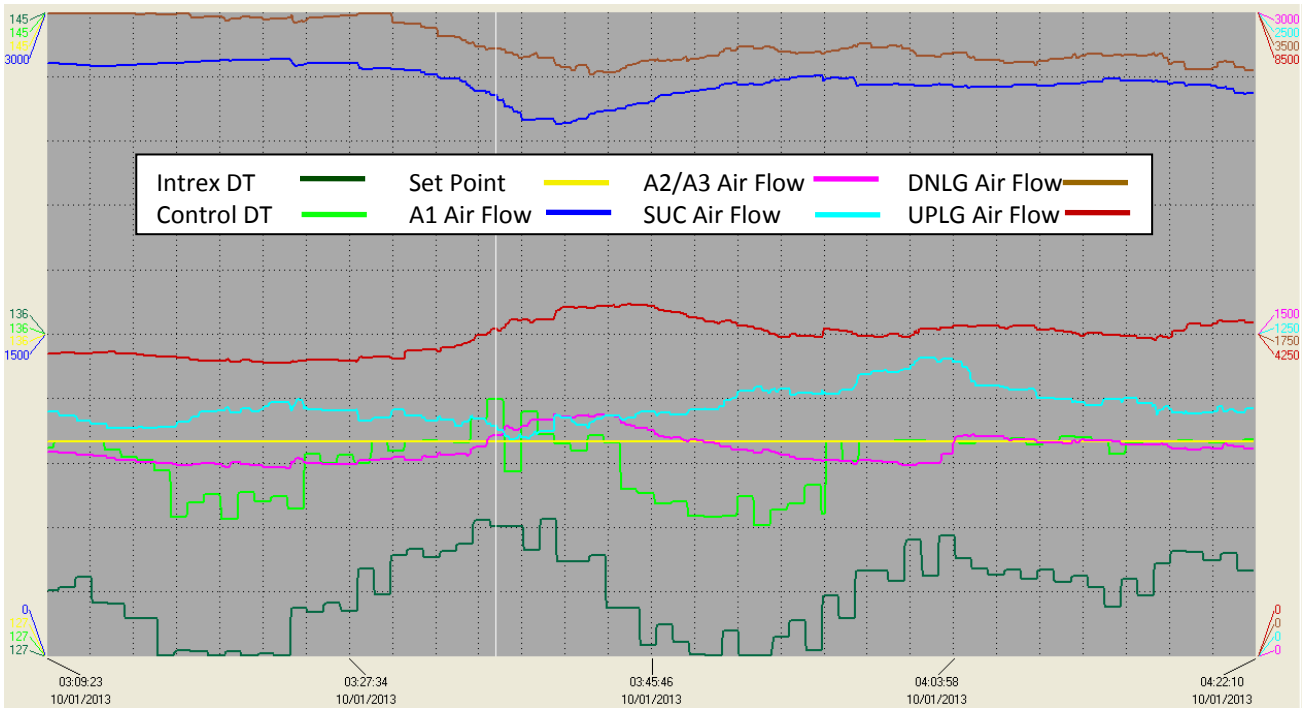


Figure 7-4 Intrex Differential Temperature vs. Controller Model Output and Optimized Air Flows with Controller Setpoint near The Edge of The Controllable Range

In addition to the stability of the controller, it is also important to look at the response. A series of step changes were made to the intrex differential temperature setpoint over a timeframe of approximately 90 minutes. The typical response time was found to be less than 60 seconds with the airflows reacting very rapidly to achieve the new setpoint. The overshoot was typically low but the controller does not have any programming that will prevent overshoot in the control model. This was considered before the programming was done and if overshoot had been an issue, logic would have been added to limit the overshoot by allowing only airflows with better errors on the same side of the setpoint as the current output to be used. The controller output airflows are sufficiently damped and the recovery from overshoot is quick enough that the additional overshoot protection was not deemed necessary. There is still the potential for deviation where the setpoint is outside of the controllable range. Figure 7-5 shows the series of step responses that were made. The airflows shown are the undamped signals within the controller. Figure 7-6 shows a closer view of approximately the first half of the step response sequence.

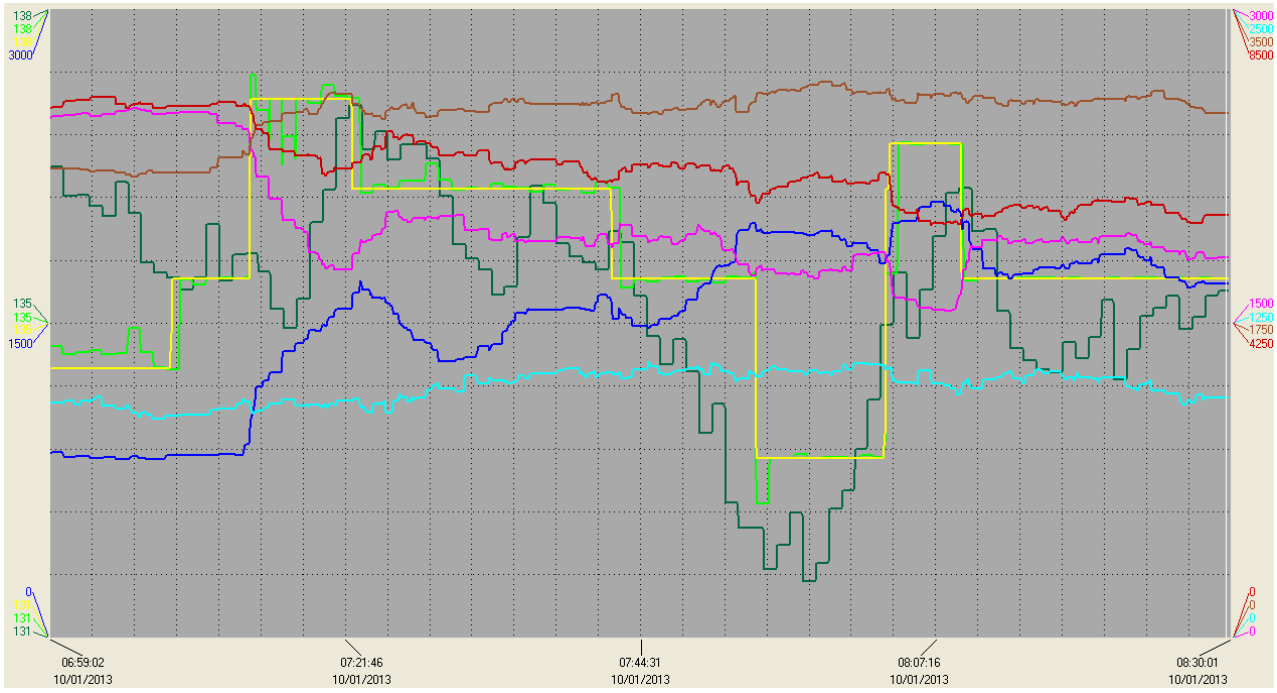


Figure 7-5 Neural Network Model Predictive Controller Step Response

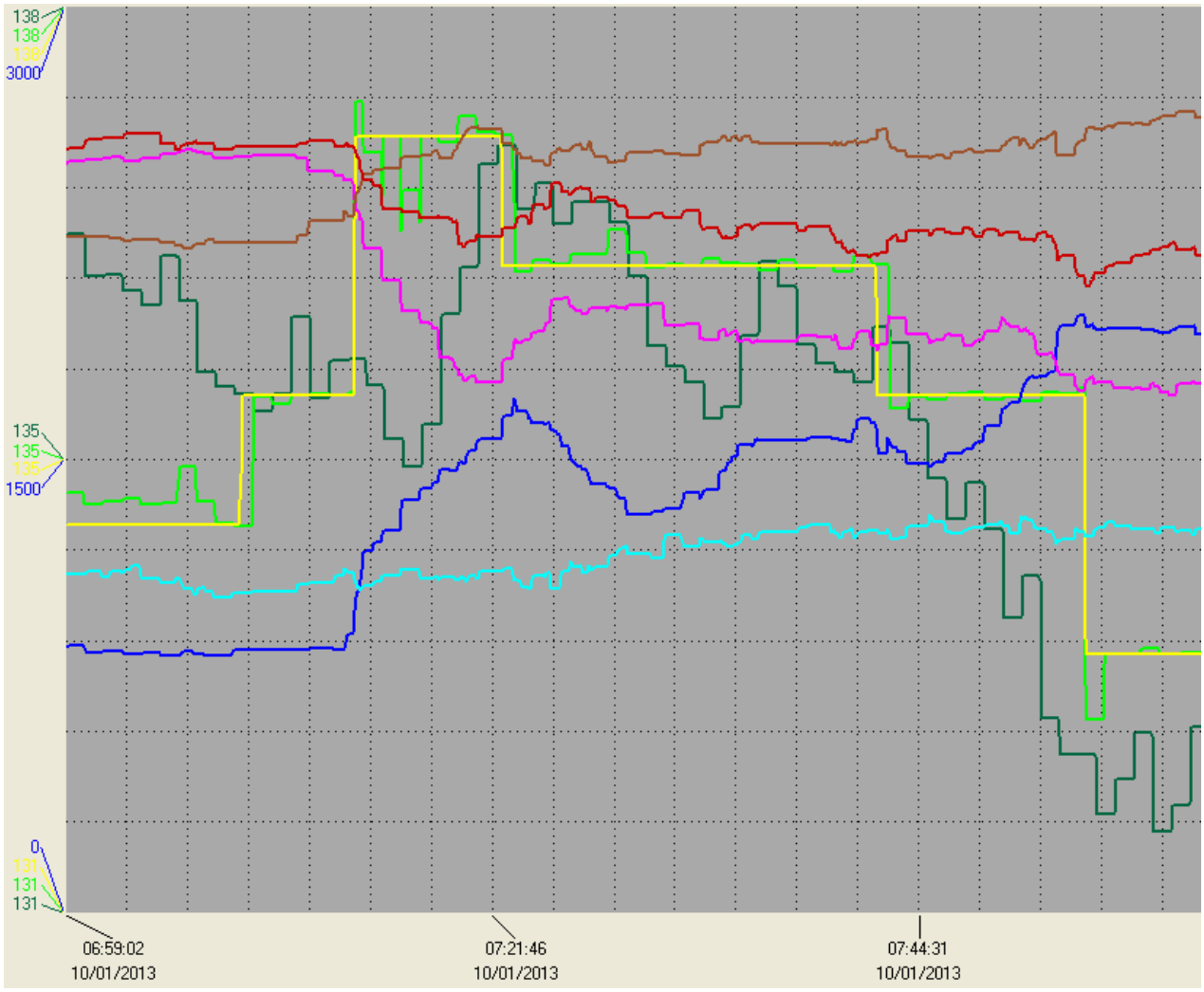


Figure 7-6 Neural Network Model Predictive Controller magnified Step Response

In order to tie the airflows setpoints from the neural network model predictive controller to the live plant, a unit outage will be required after which point a unit startup would be required to perform testing. No unit outage followed by a unit startup is scheduled within the timeframe of this project. The original intent was to manually input the airflow setpoints to match the output of the neural network controller. From the Figures above it can be seen that the airflows have to constantly adjust to maintain a temperature. At the time of the test, the plant airflows were at constant values. With this it can be seen how much the intrex differential temperature changes due to other plant parameters. The airflows cannot be set quickly enough manually to properly show controller performance.

Chapter 8 Conclusions and Areas of Future Work

8.1 Conclusions

This project has shown that a neural network model can be utilized to successfully model an intrex superheater in a circulating fluidized boiler with enough accuracy to be utilized for model predictive control. When compared to the regression model, the neural network model had better performance. The additional performance comes with additional costs in time and complexity. Training the neural network in Matlab required days of testing where Minitab was able to provide a regression model almost instantly. The neural network model required fifty pages of DCS logic to implement where the regression model would have only required one page. If accuracy is the primary objective, the neural network model is preferred even with the greater time and resource requirements.

The use of the linear congruential random number generator was found to work very well for the optimization algorithm. The majority of the resources used for the random number generators were required for performing number rounding. Of the six pages of logic required to implement the five random number generators, approximately five pages were dedicated to rounding. The remaining logic was easily implemented in the DCS and required little system resources.

The optimization algorithm as a whole had response times much better than those required and much better than what was anticipated at the start of this project. A controller scan time of 100ms was found

to be more than sufficient for the purposes of this project. The system stability was also much better than expected when the controller setpoint was within the range of the controller. The range of the controller was on the low end of what was expected but is still sufficient to be beneficial.

8.2 Areas of Future work

In order to determine which system variables to use for inputs into the neural network, a stepwise linear regression was used. This method provided sufficient results but may have eliminated other variables that did not have a linear relationship. Any such variables would not have been useful for a linear regression model but may have been useful for the neural network model and may have provided a more accurate neural network model. Future research should include alternate methods of selecting which system variables to use for inputs into the neural network model.

The neural network module utilized for the model predictive controller was tuned using a genetic algorithm. The genetic algorithm has many parameters that can be adjusted to alter how it finds optimal weights for the neural network. The ability of the genetic algorithm to find the optimal weights depends on the size of the population, number of parents in the population and the manner in which the population is mutated. This project showed three different methods of mutation with the cosine decay mutation function providing the most accurate results. With further research into the genetic algorithm parameters, it is believed that a better neural network model may be possible. There are also other stochastic optimization algorithms such as particle swarm that may provide different results.

Neural network structures with ten and fifteen hidden layer nodes were tested to determine which provided the best results. Between these two structures, the neural network with fifteen hidden layers

provided better results. There are many other combinations of layer one and layer two nodes that could be tested to find the optimum structure for this application. When the structure of the neural network changes, the number of weights changes. With different population sizes, different genetic algorithm parameters will likely be required for different neural network structures to find the optimum weights.

For the purposes of this project, the neural network model predictive controller was programmed into the DCS using pre-defined function codes. In general, the vast majority of DCS programming is done using function codes and any engineer or technician who works with a DCS system on regular basis will be familiar with the function codes associated with their DCS system. The DCS system for this project does have the ability to accept code programmed using C. This is typically only done by the DCS manufacturer for specialized applications and very little documentation is available on the topic. Future research should include a neural network model predictive controller programmed into the DCS using C instead of function codes. This would likely require less controller resources as the C programming language is more flexible than the pre-defined function blocks which would allow the programming to be done more efficiently.

In addition to the intrex, there are other systems within the CFB which can benefit from a neural network model predictive controller such as the one implemented in this project. There is little to no direct measurement of the properties of the bed material throughout the CFB hot loop. Neural network model predictive controllers may also prove beneficial to other control loops that are directly or indirectly impacted by the properties of the bed material. Future work may include neural network model predictive control of combustor bed level, fuel distribution and limestone distribution as well as numerous other processes within the CFB control system.

Stepwise Regression: intrex a TEMP IN versus Avg A1 AF, Avg A2 AF, ...

Alpha-to-Enter: 0.05 Alpha-to-Remove: 0.05

Response is intrex a TEMP INCREASE on 25 predictors, with N = 5886

Step	1	2	3	4	5	6
Constant	95.47	881.56	947.16	937.86	924.68	869.75
Avg A1 AF	0.00518	0.00097	0.00124	0.00120	0.00107	0.00105
T-Value	11.35	4.62	8.31	8.05	7.23	7.29
P-Value	0.000	0.000	0.000	0.000	0.000	0.000
Avg A2 AF	-0.00290	-0.00331	-0.00184	-0.00165	-0.00160	-0.00101
T-Value	-3.55	-8.89	-6.92	-6.19	-6.05	-3.90
P-Value	0.000	0.000	0.000	0.000	0.000	0.000
Avg A3 AF	-0.00090	0.00264	0.00076	0.00069	0.00059	0.00053
T-Value	-1.30	8.40	3.37	3.07	2.65	2.46
P-Value	0.193	0.000	0.001	0.002	0.008	0.014
Avg SUC AF	-0.00324	-0.00039	-0.00031	-0.00009	-0.00015	0.00025
T-Value	-14.44	-3.74	-4.19	-1.12	-1.87	3.15
P-Value	0.000	0.000	0.000	0.261	0.061	0.002
DNLG AF	0.01147	0.00159	0.00044	0.00044	0.00035	0.00094
T-Value	11.01	3.33	1.28	1.30	1.04	2.86
P-Value	0.000	0.001	0.201	0.195	0.298	0.004
UPLG AF	0.00129	0.00012	-0.00002	-0.00002	0.00008	-0.00029
T-Value	12.35	2.57	-0.61	-0.72	2.18	-7.16
P-Value	0.000	0.010	0.543	0.472	0.029	0.000
STM IN TE		-0.8688	-0.9401	-0.9283	-0.9247	-0.8833
T-Value		-150.18	-221.97	-209.39	-209.01	-183.35
P-Value		0.000	0.000	0.000	0.000	0.000
Main stm deviation			0.6851	0.6796	0.6728	0.6348
T-Value			75.29	74.93	74.39	70.48
P-Value			0.000	0.000	0.000	0.000
AVG FB				-0.166	-0.239	-0.178
T-Value				-8.45	-11.24	-8.55
P-Value				0.000	0.000	0.000
Cell AB Ave Temp					0.00642	0.02614
T-Value					8.78	20.76
P-Value					0.000	0.000
Heat in						-0.00607
T-Value						-18.96
P-Value						0.000
S	5.87	2.67	1.90	1.89	1.88	1.83
R-Sq	14.01	82.22	90.95	91.06	91.17	91.68
R-Sq(adj)	13.92	82.20	90.94	91.05	91.16	91.67

Step	7	8	9	10	11	12
Constant	852.4	833.0	811.5	806.9	808.9	807.1
Avg A1 AF	0.00112	0.00113	0.00117	0.00117	0.00118	0.00130
T-Value	7.83	8.03	8.37	8.32	8.42	9.17
P-Value	0.000	0.000	0.000	0.000	0.000	0.000
Avg A2 AF	-0.00097	-0.00081	-0.00083	-0.00087	-0.00083	-0.00093
T-Value	-3.81	-3.21	-3.30	-3.47	-3.32	-3.72
P-Value	0.000	0.001	0.001	0.001	0.001	0.000
Avg A3 AF	0.00064	0.00080	0.00097	0.00097	0.00093	0.00099
T-Value	2.99	3.75	4.57	4.56	4.39	4.70
P-Value	0.003	0.000	0.000	0.000	0.000	0.000
Avg SUC AF	0.00025	0.00019	0.00027	0.00031	0.00032	0.00031
T-Value	3.19	2.53	3.51	4.07	4.16	3.99
P-Value	0.001	0.011	0.000	0.000	0.000	0.000
DNLG AF	0.00112	0.00090	0.00095	0.00084	0.00089	0.00081
T-Value	3.44	2.79	2.94	2.63	2.77	2.53
P-Value	0.001	0.005	0.003	0.009	0.006	0.011
UPLG AF	-0.00026	-0.00012	-0.00013	-0.00014	-0.00013	-0.00012
T-Value	-6.67	-2.86	-3.16	-3.47	-3.22	-2.88
P-Value	0.000	0.004	0.002	0.001	0.001	0.004
STM IN TE	-0.8727	-0.8567	-0.8443	-0.8331	-0.8324	-0.8348
T-Value	-179.68	-170.69	-162.96	-148.60	-148.78	-149.06
P-Value	0.000	0.000	0.000	0.000	0.000	0.000
Main stm deviation	0.6255	0.6075	0.6018	0.5970	0.5952	0.5930
T-Value	69.92	67.46	67.09	66.34	66.25	66.08
P-Value	0.000	0.000	0.000	0.000	0.000	0.000
AVG FB	-0.229	-0.316	-0.283	-0.312	-0.320	-0.306
T-Value	-10.85	-14.17	-12.55	-13.47	-13.79	-13.17
P-Value	0.000	0.000	0.000	0.000	0.000	0.000
Cell AB Ave Temp	0.0306	0.0243	0.0286	0.0330	0.0328	0.0281
T-Value	23.43	17.25	19.28	19.35	19.26	14.66
P-Value	0.000	0.000	0.000	0.000	0.000	0.000
Heat in	-0.00972	-0.01166	-0.00499	-0.00566	-0.00437	-0.00448
T-Value	-21.59	-24.35	-5.57	-6.27	-4.69	-4.81
P-Value	0.000	0.000	0.000	0.000	0.000	0.000
Total PA	0.00670	0.00647	0.00759	0.00610	0.00610	0.00595
T-Value	11.41	11.12	12.83	9.28	9.31	9.09
P-Value	0.000	0.000	0.000	0.000	0.000	0.000
Cell AA Ave Temp		0.0126	0.0164	0.0170	0.0179	0.0150
T-Value		11.06	13.50	14.00	14.60	11.23
P-Value		0.000	0.000	0.000	0.000	0.000
Steam Flow			-0.0110	-0.0093	-0.0105	-0.0108
T-Value			-8.79	-7.23	-8.05	-8.25
P-Value			0.000	0.000	0.000	0.000
AVG FB Temp				-0.0076	-0.0102	-0.0109
T-Value				-5.18	-6.59	-7.01
P-Value				0.000	0.000	0.000

Limestne Flow					-0.0072	-0.0075
T-Value					-5.25	-5.49
P-Value					0.000	0.000
UPLEG TEMP						0.0109
T-Value						5.28
P-Value						0.000
S	1.81	1.79	1.78	1.77	1.77	1.76
R-Sq	91.86	92.03	92.13	92.17	92.21	92.24
R-Sq(adj)	91.85	92.01	92.11	92.15	92.18	92.22

Step	13	14	15
Constant	805.0	802.9	805.1
Avg A1 AF	0.00130	0.00129	0.00128
T-Value	9.21	9.13	9.06
P-Value	0.000	0.000	0.000
Avg A2 AF	-0.00086	-0.00085	-0.00084
T-Value	-3.38	-3.34	-3.32
P-Value	0.001	0.001	0.001
Avg A3 AF	0.00101	0.00101	0.00100
T-Value	4.76	4.76	4.72
P-Value	0.000	0.000	0.000
Avg SUC AF	0.00036	0.00040	0.00040
T-Value	4.53	4.85	4.87
P-Value	0.000	0.000	0.000
DNLG AF	0.00085	0.00085	0.00082
T-Value	2.66	2.66	2.56
P-Value	0.008	0.008	0.011
UPLG AF	-0.00012	-0.00011	-0.00010
T-Value	-2.87	-2.57	-2.49
P-Value	0.004	0.010	0.013
STM IN TE	-0.8351	-0.8341	-0.8355
T-Value	-149.14	-148.41	-147.68
P-Value	0.000	0.000	0.000
Main stm deviation	0.5926	0.5915	0.5920
T-Value	66.06	65.85	65.90
P-Value	0.000	0.000	0.000
AVG FB	-0.298	-0.316	-0.324
T-Value	-12.67	-12.59	-12.78
P-Value	0.000	0.000	0.000
Cell AB Ave Temp	0.0295	0.0292	0.0284
T-Value	14.76	14.58	13.94
P-Value	0.000	0.000	0.000
Heat in	-0.00451	-0.00480	-0.00482
T-Value	-4.84	-5.10	-5.12
P-Value	0.000	0.000	0.000
Total PA	0.00600	0.00615	0.00557
T-Value	9.17	9.34	7.81

P-Value	0.000	0.000	0.000
Cell AA Ave Temp	0.0152	0.0155	0.0154
T-Value	11.36	11.50	11.45
P-Value	0.000	0.000	0.000
Steam Flow	-0.0113	-0.0111	-0.0118
T-Value	-8.53	-8.37	-8.64
P-Value	0.000	0.000	0.000
AVG FB Temp	-0.0074	-0.0079	-0.0081
T-Value	-3.55	-3.73	-3.86
P-Value	0.000	0.000	0.000
Limestne Flow	-0.0079	-0.0077	-0.0092
T-Value	-5.73	-5.59	-5.94
P-Value	0.000	0.000	0.000
UPLEG TEMP	0.0129	0.0134	0.0140
T-Value	5.82	6.02	6.25
P-Value	0.000	0.000	0.000
DNLG Temp	-0.0051	-0.0047	-0.0049
T-Value	-2.46	-2.25	-2.33
P-Value	0.014	0.025	0.020
AVG BED		0.030	0.032
T-Value		2.05	2.21
P-Value		0.040	0.027
TOT FUEL -5			0.0148
T-Value			2.13
P-Value			0.033
S	1.76	1.76	1.76
R-Sq	92.25	92.26	92.26
R-Sq(adj)	92.23	92.23	92.24

Appendix B – Matlab Code for Model Development

B-1 Matlab Code to Calculate Neural Network Model Output

```
function [MSE,err,maxer,out]=
neurnet(inA,outA,l1w,l1c,l2w,l2c,olw,olc,lay1n,lay2n)

%Network Structure
%   lay1n defines the number of neurons in the input layer.  lay2n defines
%   the number of neurons in the second layer.  The output layer will
%   always be 1 neuron.  Weights will be applied before the summing blocks
%   for each neuron.  Constants will be added at each summing block.
%   The output of each neuron will pass through an activation function

%Inputs:
%   inA = input data set (variables in different columns)
%   outA = expected output for each input
%   l1w = layer 1 weights
%   l1c = layer 1 constants
%   l2w = layer 2 weights
%   l2c = layer 2 counstants
%   olw = output layer weights
%   olc = output layer constant
%   lay1n = number of first layer neurons
%   lay2n = number of second layer neurons
%
%Outputs:
%   MSE = Mean square error
%   err = raw error values
%   maxer = maximum error
%   out = neural net output

out = zeros(1,size(inA,1));                     %Initialize Weight Matrix
weights1=reshape(l1w,size(inA,2),lay1n);        %reshape weight matrix
l1c=repmat(l1c,size(inA,1),1);                  %Create l1 constant matrix
lay1out = (inA*weights1)+l1c;                   %layer 1 summing node
lay1out = 2./(1+exp(-2.*lay1out))-1;            %layer 1 activation function
weights2=reshape(l2w,lay1n,lay2n);              %reshape weight matrix
lay2out = lay1out*weights2;                     %layer 2 summing node part 1
l2c=repmat(l2c,size(inA,1),1);                  %create l2 constant matrix
lay2out = lay2out+l2c;                          %layer 2 summing node part2
lay2out = 2./(1+exp(-2*lay2out))-1;             %layer 2 activation function
weightsout=transpose(olw);                      %transpose out weights
out = lay2out*weightsout+olc;                   %output summing node
err = outA-out;                                 %calculate error
maxer = max(err);                               %find maximum error
MSE = mean((err).^2);                           %calculate MSE
end
```

B-2 Matlab Code for Genetic Algorithm Population Generation

```
function w = genalg(parents,mut,totgen,gen,pop)

%Inputs
%   Parents = matrix of parent weights
%   mut = mutation
%   totgen = total number of generations
%   gen = current generation number
%   pop = Size of population to generate
%
%Outputs
%   w = weighs

numc = pop - size(parents,1);       %number of children to generate
% make children
w = zeros(numc,size(parents,2));
parfor i = 1:numc                       %for the number of children
%   generate 2x1 matrix of ints from 1:number of parents
    x = randi(size(parents,1),2,1);
%   generate 1xnumber of weights matrix of ints from 1:2
    y = randi(2,1,size(parents,2));
%   convert 2's to 1 and 1's to 0 to select first parent
    p1=y-1;
%   convert 2's to 0 to select second parent
    p2=abs(y-2);
%   combine parts frome each parent for each weight
    w(i,:)=p1(1,:).*parents(x(1),:)+p2(1,:).*parents(x(2),:);
end
% mutate children
%   determine which weights will be mutated
mutloc = randi(numc*size(parents,2),1,ceil(mut*numc*size(parents,2)));
%   mutation varies from 50% to 10% as the generation number is increased
%mutation = .4*(totgen-gen)/totgen+.1 ;
%   mutation decays from 60% to 10% with an added cosine function
mutation =(.4*(totgen-gen)/totgen+.1)+.1*((totgen-
gen)/totgen)*cos(20*gen/totgen*pi);
%   Constant Mutation of 25%
%mutation = .25;
%   determine the amount of mutation for each weight (-1:1 * mutation)
mutmul = (1-(rand(1,length(mutloc))*2)*mutation);
%   generate an empty matrix for the new children
mutmat = ones(numc,size(parents,2));

for i = 1:length(mutloc)            %for each mutation
    mutmat(mutloc(i)) = mutmul(i);  %fill in the mutation matrix
end
w = w .* mutmat;                    %generate new children
% make population of parents and children
w = cat(1,parents,w);
end
```

B-3 Matlab Code for Data Normalization

```
function [normdata]= mmnorm(normmat,data)

% This function will take in data and an associated normalization matrix
% (normmat)containing the mean and standard deviation of the data set
% and perform normalization.  The normalized data will be returned.

normdata = zeros(size(data,1),size(data,2));    %Initialize the matrix
x=normmat(1,:);                                 %Get mean for each variable
y=normmat(2,:);                                 %Get SD for each variable
parfor i=1:size(data,2)                         %Normalize the data
    normdata(:,i) = ((data(:,i)-x(i)))/y(i);
end
end



function [normdata]= immnorm(normmat,data)

% This function will take in data and an associated normalization matrix
% (normmat) containg the mean and standard deviation of the data set and
% perform inverse normalization.  The un-normalized data will be returned.

normdata = zeros(size(data,1),size(data,2));    %Initialize the matrix
x=normmat(1,:);                                 %Get mean for each variable
y=normmat(2,:);                                 %Get SD for each variable
parfor i=1:size(data,2)                         %Un-Normalize the data
    normdata(:,i) = ((data(:,i)*y(i)))+x(i);
end
end
```

B-4 Matlab Code for Neural Network Model Training and Testing

```
%Nerual Network Model Program
clear

%get training data
intrain = xlsread('Training_Data_in2');
outtrain = xlsread('Training_Data_out2');

%get testing data
intest = xlsread('Testing_Data_in2');
outtest = xlsread('Testing_Data_out2');

%Get normalization Matrix
innormmat=xlsread('STD_Norm_in');
outnormmat=xlsread('STD_Norm_out');

%Perform Normalization
intrain = mmnorm(innormmat,intrain);
outtrain = mmnorm(outnormmat,outtrain);
intest = mmnorm(innormmat,intest);
outtest = mmnorm(outnormmat,outtest);

%Define Neural Network Structure
%[MSE,err,maxer,out]= neurnet(inA,outA,l1w,l1c,l2w,l2c,olw,lay1n,lay2n);
L1N = 20;                               %number of neurons in layer 1
L2N = 15;                               %number of neurons in layer 2
nL1w = L1N * size(intrain,2);           %number of layer 1 weights
nL1c = L1N;                             %number of layer 1 constants
nL2w = L2N*L1N;                         %number of layer 2 weights
nL2c = L2N;                             %number of layer 2 constants
nOLw = L2N;                             %number of output layer weights
nOLc = 1;

Totw = nL1w+nL1c+nL2w+nL2c+nOLw+nOLc;    %total number of weights
nin = size(intrain,2);

%Set Genetic Algorithm parameters

mutation = .1;                        %amount of mutation in genetic algorithm
pop = 750;                             %population (number of sets of weights)
numpar = 225;                          %number of parents to use to generate
children
generations = 750;                     %number of generations

%Generate inital weights from -1 to 1
w = (rand(pop,Totw)-.5)*2;
%load('C:\Documents and Settings\I&C ENGINEER\Desktop\N01 Intrex A NN\MATLAB
Final\Test Weights\Test07.mat')
%w=weightout;

%Training
    MSE = zeros(1,generations);
    %for each generation
```

```
    for j = 1:generations
        % calculate the error for each parent
        mse=zeros(1,size(w,1));                 %Initialize mse
        error=zeros(size(w,1),size(intrain,1)); %Initialize error
        maxer=zeros(1,size(w,1));               %Initialize maxer
        out = zeros(size(w,1),size(intrain,1)); %Initialize out
        parfor k = 1:size(w,1);                 %For each parent weight
            %Convert Weights for NN program
            [l1w, l1c, l2w, l2c, outw,outc]=expweights(w(k,:),L1N,L2N,nin);
            %Calculate the mse for the parent
            [mse(k), error(k,:), maxer(k),out(k,:)] =
neurnet(intrain,outtrain,l1w,l1c,l2w,l2c,outw,outc,L1N,L2N);
        end

        %capture best MSE
        MSE(j) = min(mse);
        % find the best weights
        parent=zeros(numpar,Totw);
        for kk = 1:numpar;                  %for one to the number of parents
            keep = find(mse == min(mse)); %find the location of minimum error
            parent(kk,:) = w(keep(1),:);  %Store the parent with minimum
error
            mse(keep) = 10000000;         %maximize error for that parent
        end
        %Generate new weights
        w = genalg(parent,mutation,generations,j,pop);

    end

%Plot the MSE
Figure('Name','MSE','numbertitle','off','color','w')
plot(MSE)
%Capture the weight with the lowest MSE
weightout = parent(1,:);

%Training Verification
%Generate Intrex Output using best weights and training data
[l1w, l1c, l2w, l2c, outw,outc]=expweights(weightout,L1N,L2N,nin);
[msetr, errortr, maxertr,outtr] =
neurnet(intrain,outtrain,l1w,l1c,l2w,l2c,outw,outc,L1N,L2N);
outtrn= immnorm(outnormmat,outtr);
%Get Actual intrex differential temperature
outtrainx = xlsread('Training_Data_out2');
%Plot the training output data vs the NN output with the best weights
Figure('Name','Training Verification','numbertitle','off','color','w')
plot(outtrainx)
hold on
plot(outtrn,'r')

%Testing
%Generate Intrex Output using best weights and testing data
[l1w, l1c, l2w, l2c, outw,outc]=expweights(weightout,L1N,L2N,size(intest,2));
[msetst, errortst, maxertst,outtst] =
neurnet(intest,outtest,l1w,l1c,l2w,l2c,outw,outc,L1N,L2N);
outtst= immnorm(outnormmat,outtst);
%Get Actual intrex differential temperature
```

```
outtstx = xlsread('Testing_Data_out2');
%Plot the testing output data vs the NN output with the best weights
Figure('Name','Testing Verification','numbertitle','off','color','w')
plot(outtstx)
hold on
plot(outtst,'r')

%Calculate Regression output
intestx=xlsread('Testing_Data_In2');    %Get Input Data
re=xlsread('regresscon');               %Get regression coefficients
regtesta=transpose(intestx);
regtesta(21,:)=1;
regouta=re*regtesta;                    %Caclulate output of regression model

plot(regouta,'g')

%Calculate MSE for regression and NN models
regmse=mean((outtstx-transpose(regouta)).^2)
nnmse=mean((outtstx-outtst).^2)

%Plot Regression Error Histogram
Figure('Name','Regression Error Histogram','numbertitle','off','color','w')
hold on
E=outtstx-transpose(regouta);          %Calculate Raw testing error
range=round(min(E)):1:round(max(E));   %Determine the error range
hist(E,range)                          %Plot error histogram

teststdr=std(E);                        %calculate error standard deviation
testmeanr=mean(E);                      %calculate error mean

%Training data error
Figure('Name','Training Error Histogram','numbertitle','off','color','w')
hold on
E=outtrainx-outtrn;                    %Caclulate Raw training error
range=round(min(E)):1:round(max(E));   %determine the error range
hist(E,range)                          %Plot error histogram
trainstd=std(E);                       %calculate error standard deviation
trainmean=mean(E);                     %calculate error mean
clear E range

%Testing data error
Figure('Name','Testing Error Histogram','numbertitle','off','color','w')
hold on
E=outtstx-outtst;                      %Calculate Raw testing error
range=round(min(E)):1:round(max(E));   %Determine the error range
h42=hist(E,range);                          %Plot error histogram
teststd=std(E);                        %calculate error standard deviation
testmean=mean(E);                      %calculate error mean

%get testing data
intest1m = xlsread('Testing_Data_in_1min');
outtest1m = xlsread('Testing_Data_out_1min');
intest1mx=intest1m;
intest1m = mmnorm(innormmat,intest1m);
outtest1m = mmnorm(outnormmat,outtest1m);
```

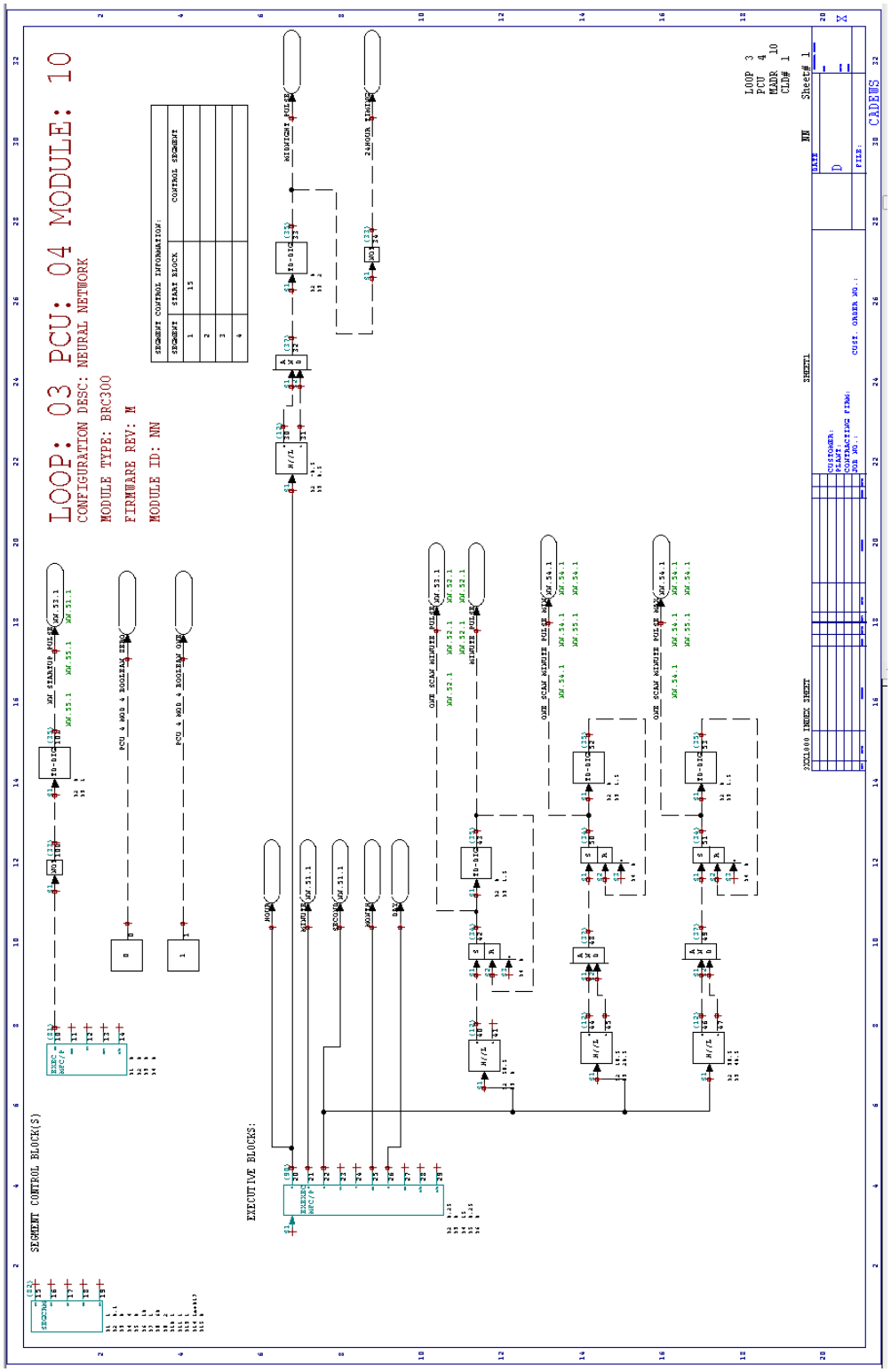
```
%Generate Intrex Output using best weights and testing data
[l1w, l1c, l2w, l2c, outw,outc]=expweights(weightout,L1N,L2N,size(intest,2));
[msetst1m, errortst, maxertst,outtst1m] =
neurnet(intest1m,outtest1m,l1w,l1c,l2w,l2c,outw,outc,L1N,L2N);
outtst1m= immnorm(outnormmat,outtst1m);
%Get Actual intrex differential temperature
outtstx1m = xlsread('Testing_Data_out_1min');
%Plot the testing output data vs the NN output with the best weights
Figure('Name','Testing 1 min Verification','numbertitle','off','color','w')
plot(outtstx1m,'g')
hold on
plot(outtst1m,'m')

re=xlsread('regresscon');
regtest=transpose(intest1mx);
regtest(21,:)=1;
regout=re*regtest;
plot(regout,'b');
```

Appendix C – Neural Network Testing Results

All tests performed with a population of 750							
Test #	Parents	L2 Nodes	Decay	% Mut	MSE	STD	Mean
1	225 (30%)	10	None(25%)	10	3.198	1.7866	-0.0837
2	225 (30%)	10	None(25%)	15	3.1486	1.7709	0.1161
3	225 (30%)	10	None(25%)	20	3.1458	1.7739	-0.0062
4	225 (30%)	10	Linear	10	2.9768	1.7256	0.0016
5	225 (30%)	10	Linear	15	2.8604	1.691	0.0429
6	225 (30%)	10	Linear	20	2.9528	1.7183	-0.0357
7	225 (30%)	10	Cosine	10	2.6875	1.6383	0.0666
8	225 (30%)	10	Cosine	15	3.0278	1.7385	0.0801
9	225 (30%)	10	Cosine	20	3.3537	1.8314	0.0272
10	225 (30%)	15	None(25%)	10	2.9695	1.708	-0.2309
11	225 (30%)	15	None(25%)	15	3.0335	1.742	0.0125
12	225 (30%)	15	None(25%)	20	3.0268	1.7395	-0.0447
13	225 (30%)	15	Linear	10	2.8601	1.6901	-0.0675
14	225 (30%)	15	Linear	15	3.094	1.7592	-0.0145
15	225 (30%)	15	Linear	20	3.2209	1.7946	-0.0389
16	225 (30%)	15	Cosine	10	2.496	1.5785	-0.0722
17	225 (30%)	15	Cosine	15	2.8213	1.6799	0.01114
18	225 (30%)	15	Cosine	20	3.2687	1.8083	0.000217
19	300 (40%)	10	None(25%)	10	3.3289	1.812	-0.2162
20	300 (40%)	10	None(25%)	15	3.3397	1.806	-0.2803
21	300 (40%)	10	None(25%)	20	3.1988	1.788	-0.0539
22	300 (40%)	10	Linear	10	2.9628	1.7173	-0.1206
23	300 (40%)	10	Linear	15	2.9203	1.7091	-0.0128
24	300 (40%)	10	Linear	20	3.1373	1.7715	-0.0109
25	300 (40%)	10	Cosine	10	2.7738	1.6657	-0.0096
26	300 (40%)	10	Cosine	15	2.9907	1.7301	0.0693
27	300 (40%)	10	Cosine	20	3.146	1.779	-0.0123
28	300 (40%)	15	None(25%)	10	3.1005	1.7554	-0.1417
29	300 (40%)	15	None(25%)	15	3.2298	1.7956	0.0823
30	300 (40%)	15	None(25%)	20	3.0855	1.7561	0.0521
31	300 (40%)	15	Linear	10	2.7674	1.6457	-0.245
32	300 (40%)	15	Linear	15	2.8389	1.6848	-0.0347
33	300 (40%)	15	Linear	20	3.048	1.7564	-0.0152
34	300 (40%)	15	Cosine	10	2.6993	1.6431	0.0233
35	300 (40%)	15	Cosine	15	3.0085	1.7348	-0.0092
36	300 (40%)	15	Cosine	20	2.9145	1.7073	-0.1385

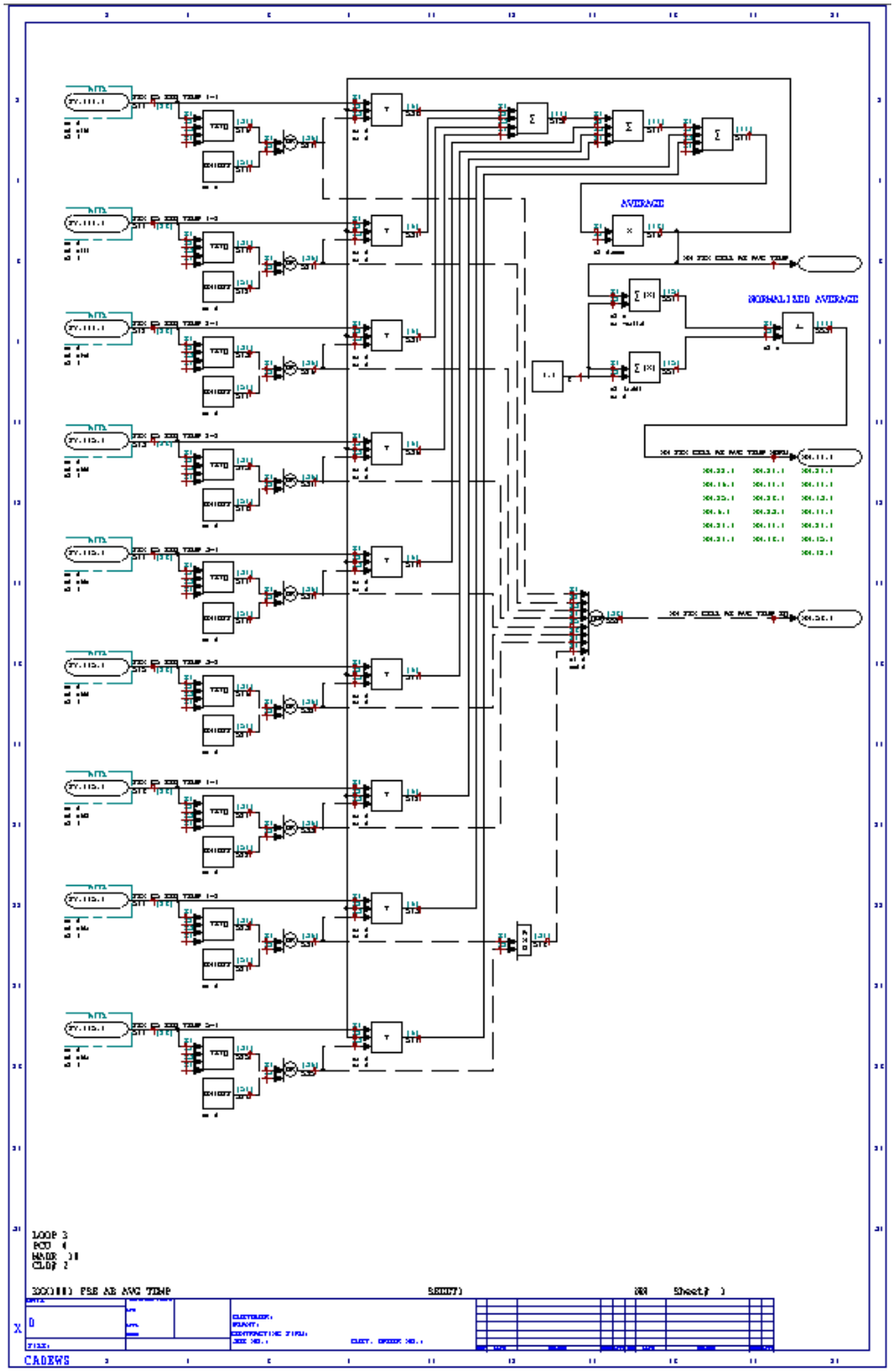
Appendix D – DCS Logic for Neural Network Model Predictive Controller Implementation

D-1 DCS Timing Logic and Executive blocks

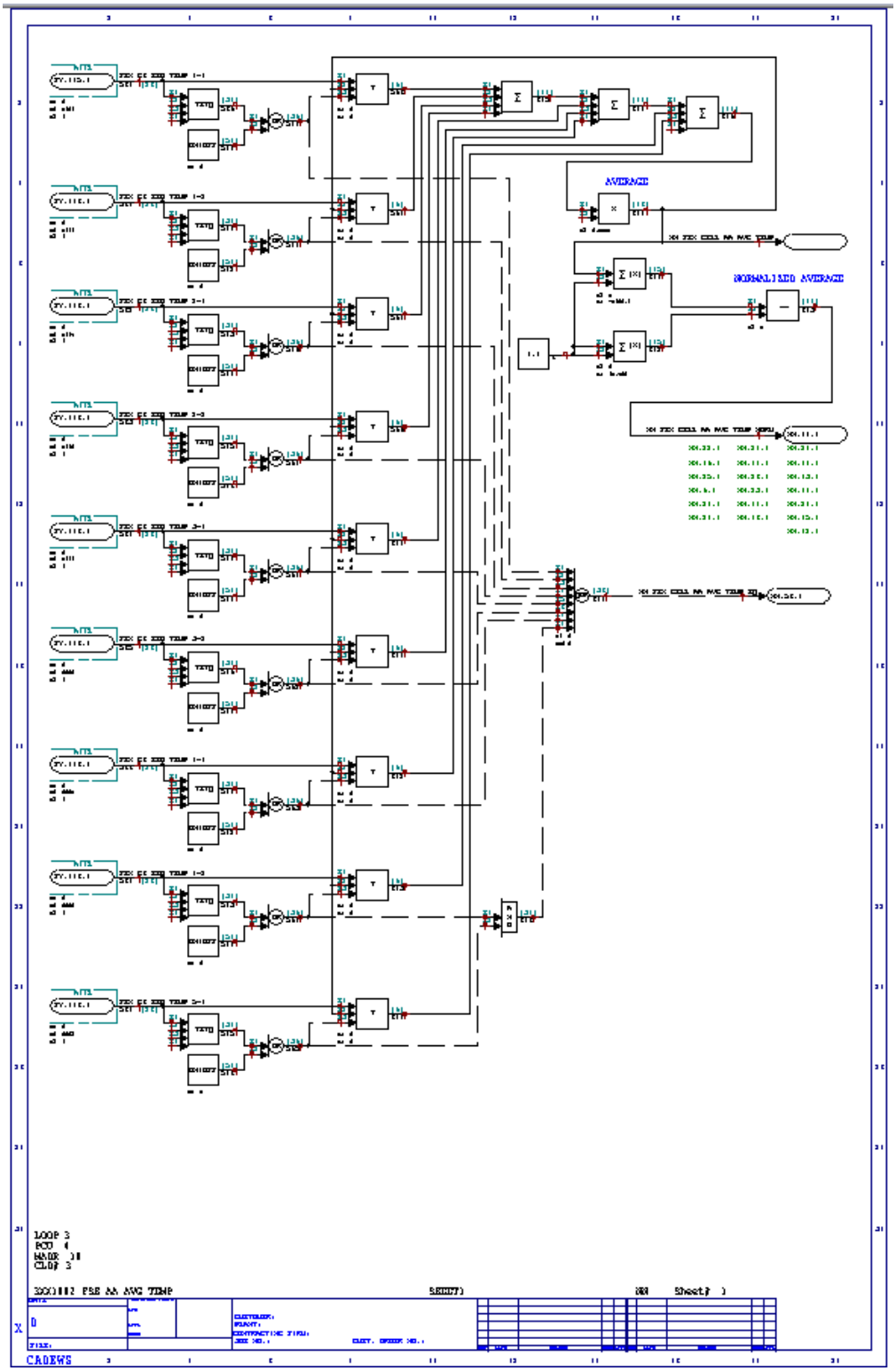


D-2 DCS Input Logic

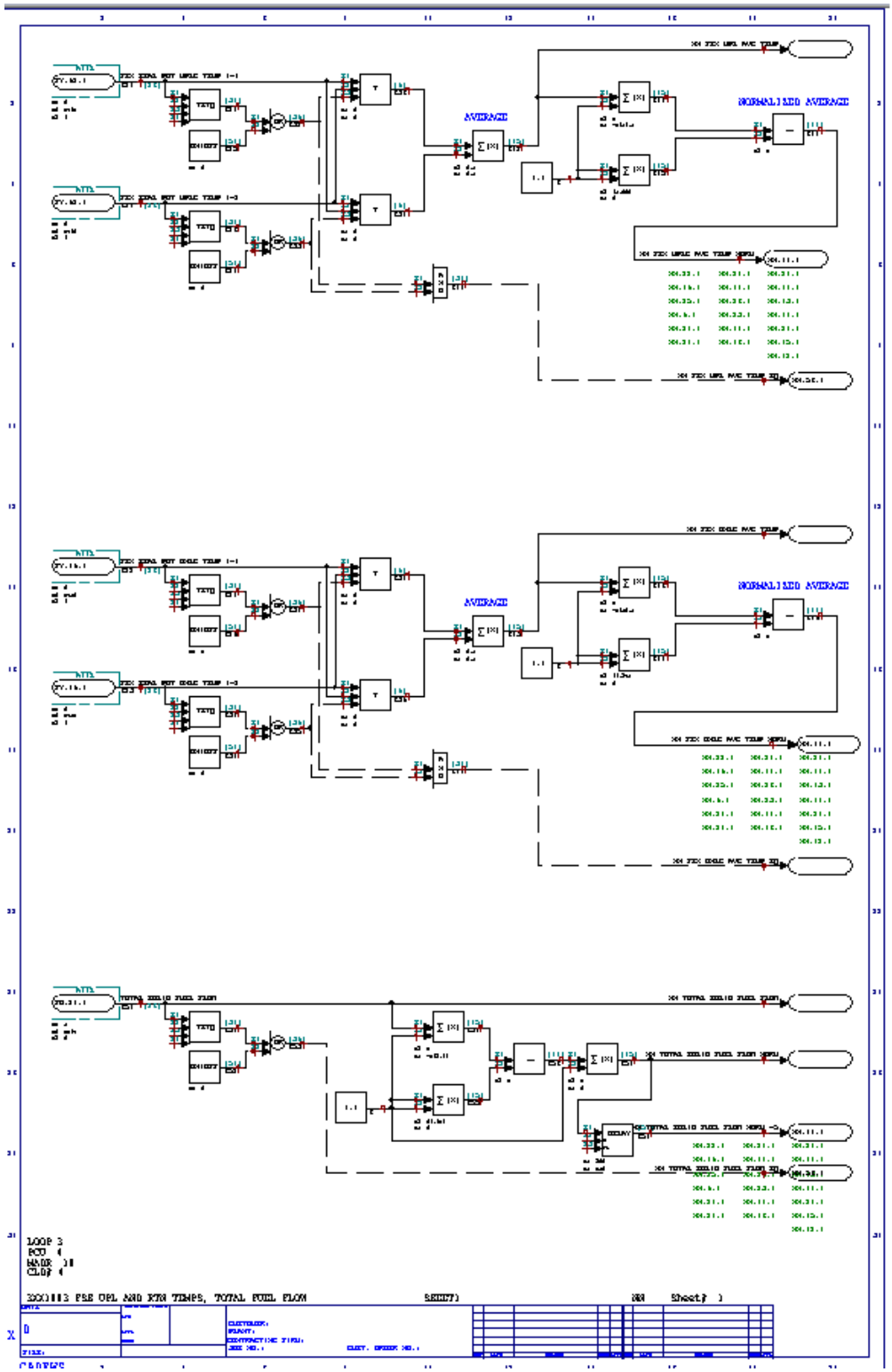
Intrex Cell AB Average Bed Temperature



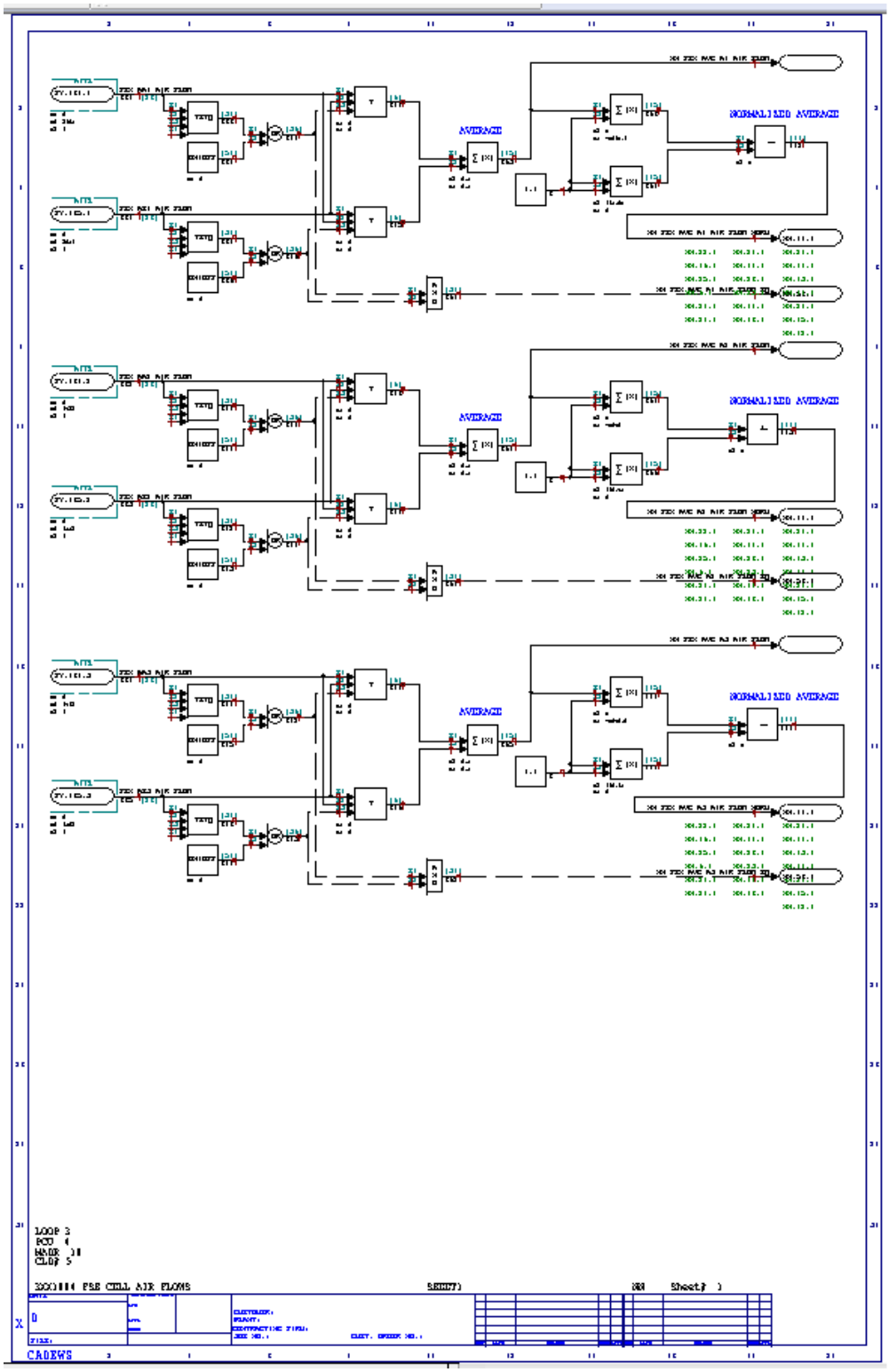
Intrex Cell AA Average Bed Temperature



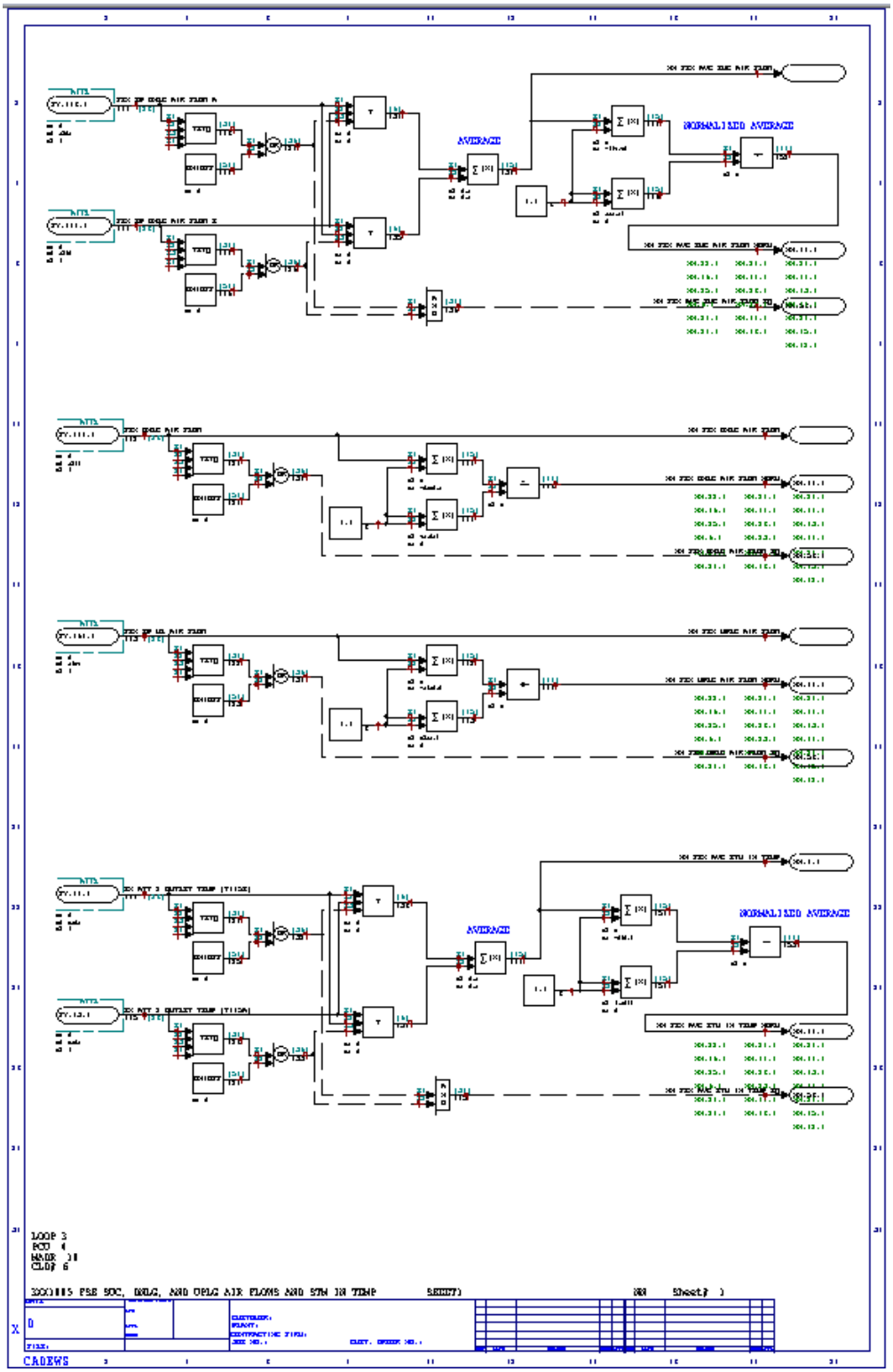
Intrex Upleg and Downleg Temperatures, Total Solid Fuel Flow



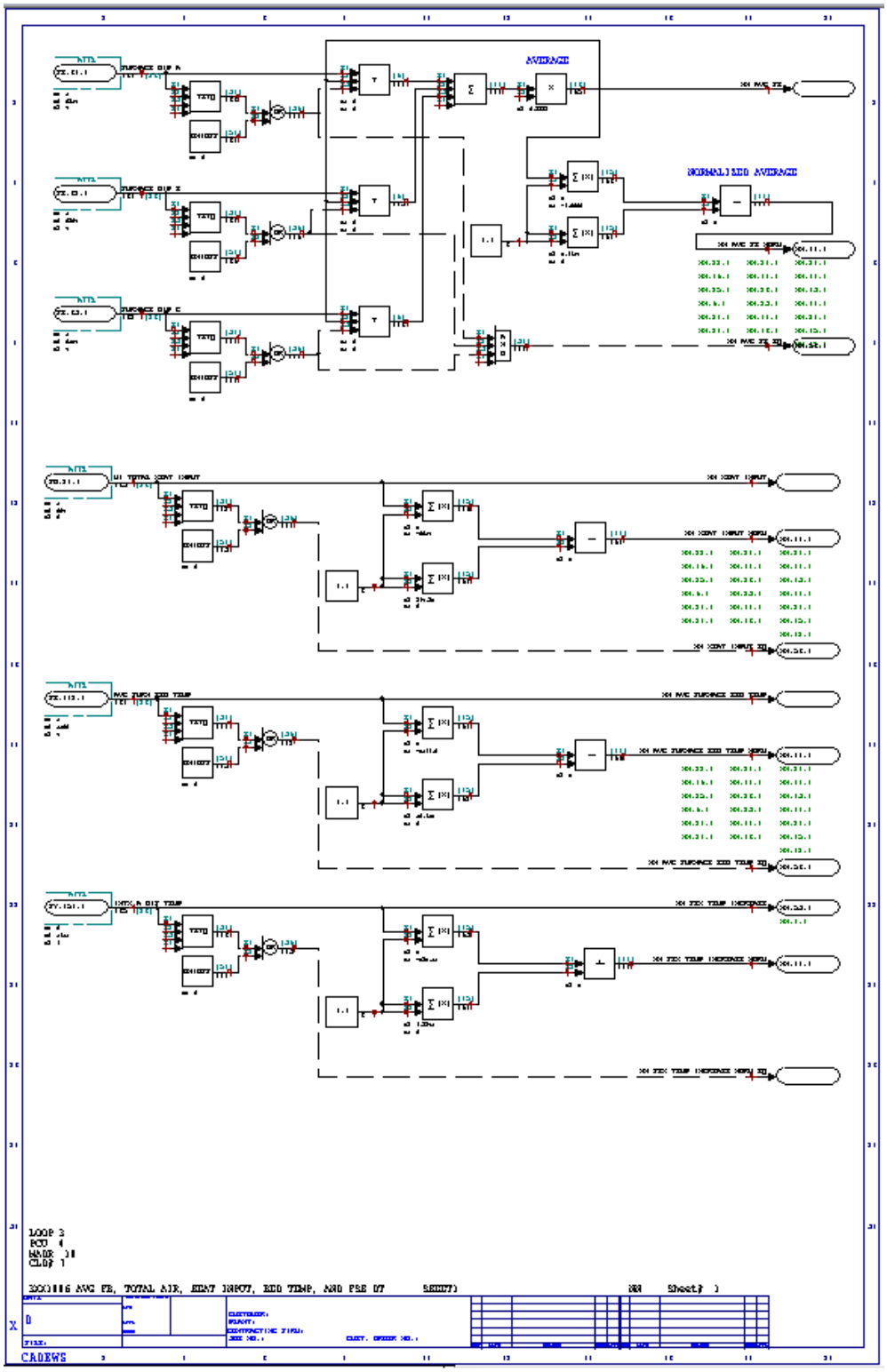
Intrex Cell A1, A2, and A3 Average Air Flows



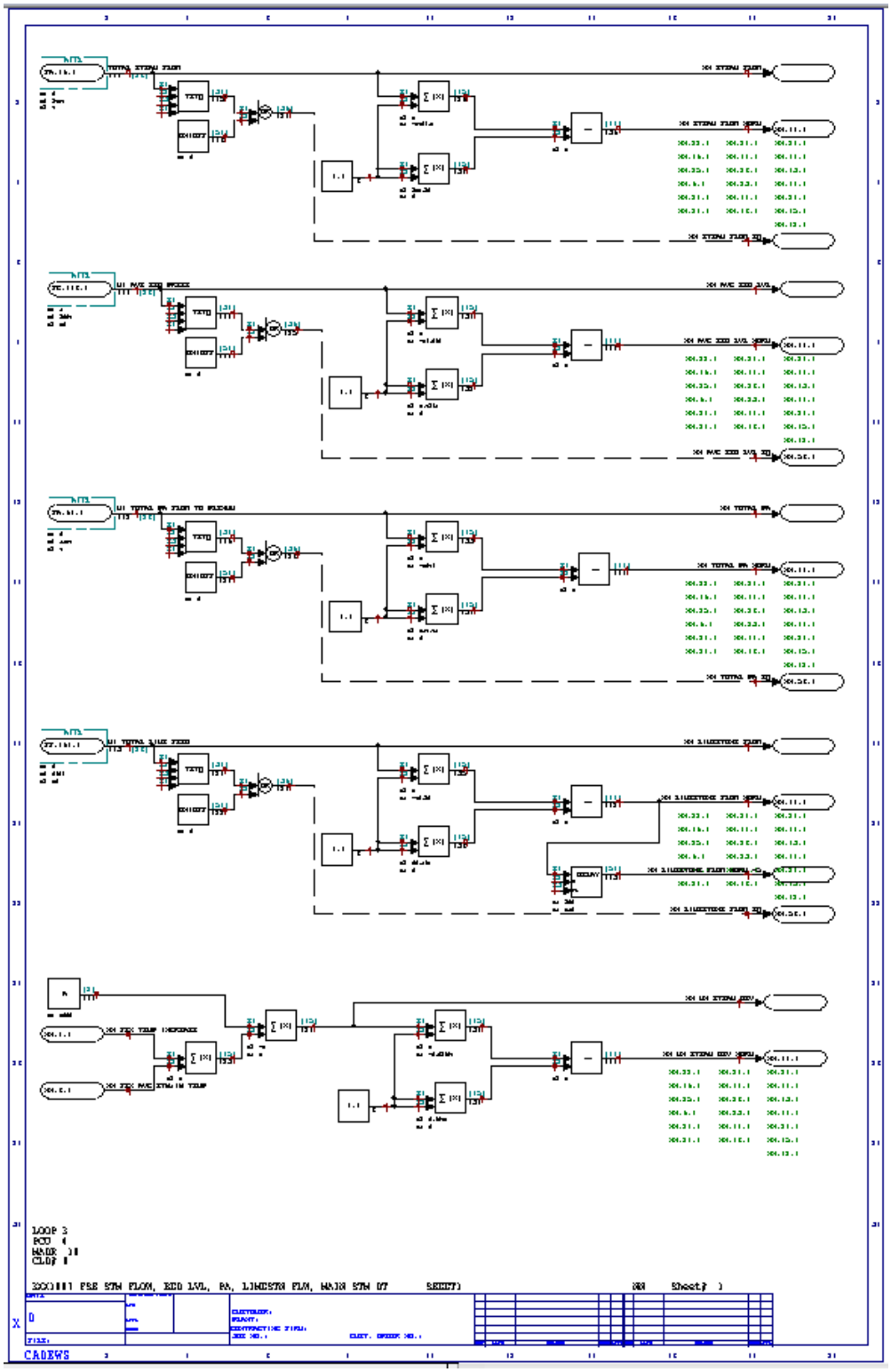
Intrex Average Startup Channel, Downleg, and Upleg air flows



Average Furnace Freeboard, Heat input, Furnace Bed Temperature and Intrex Differential Temperature

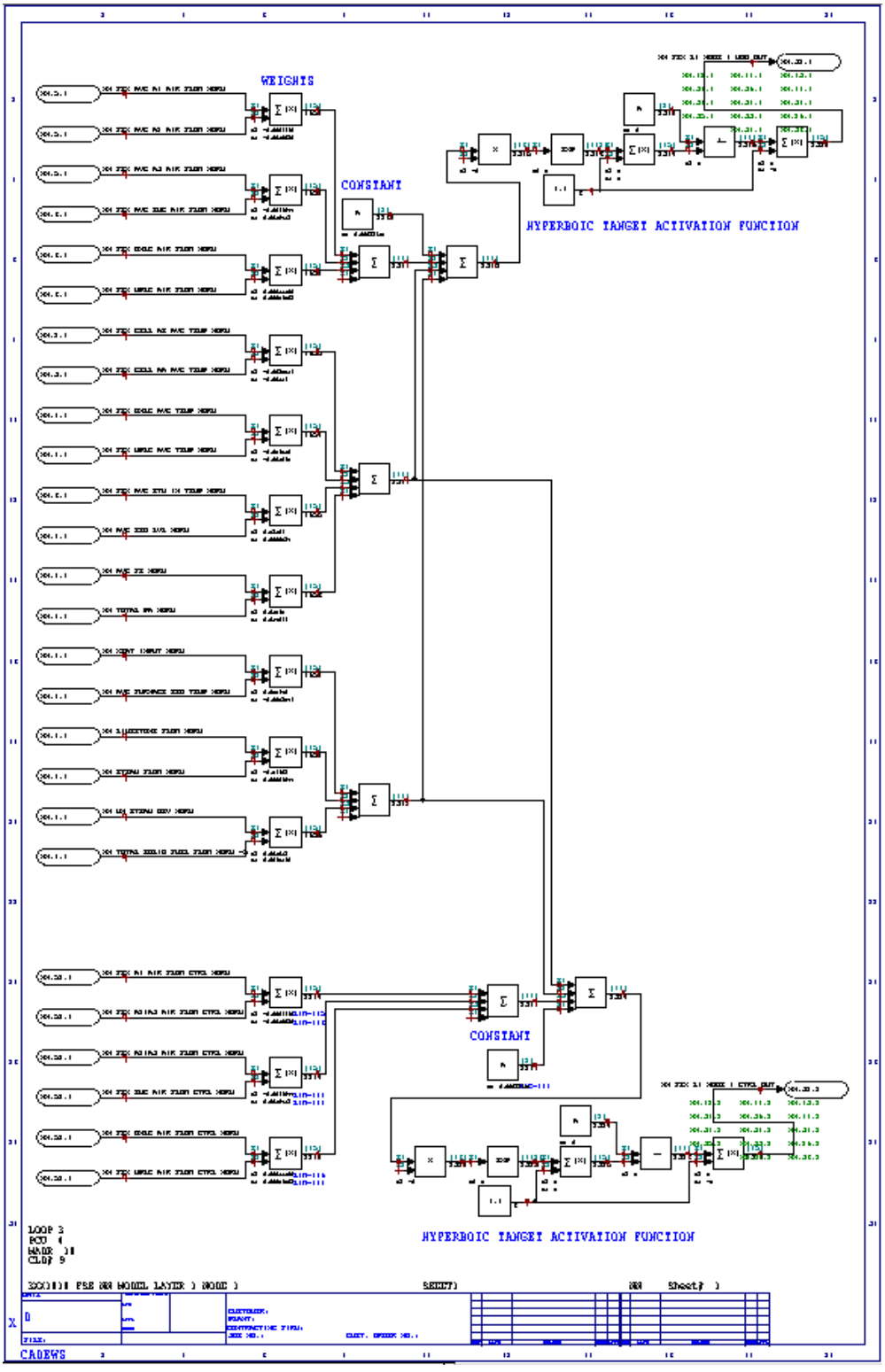


Main Steam Flow, Furnace Bed Level, Primary Air Flow, Total Limestone Flow and Main Steam Temperature Deviation

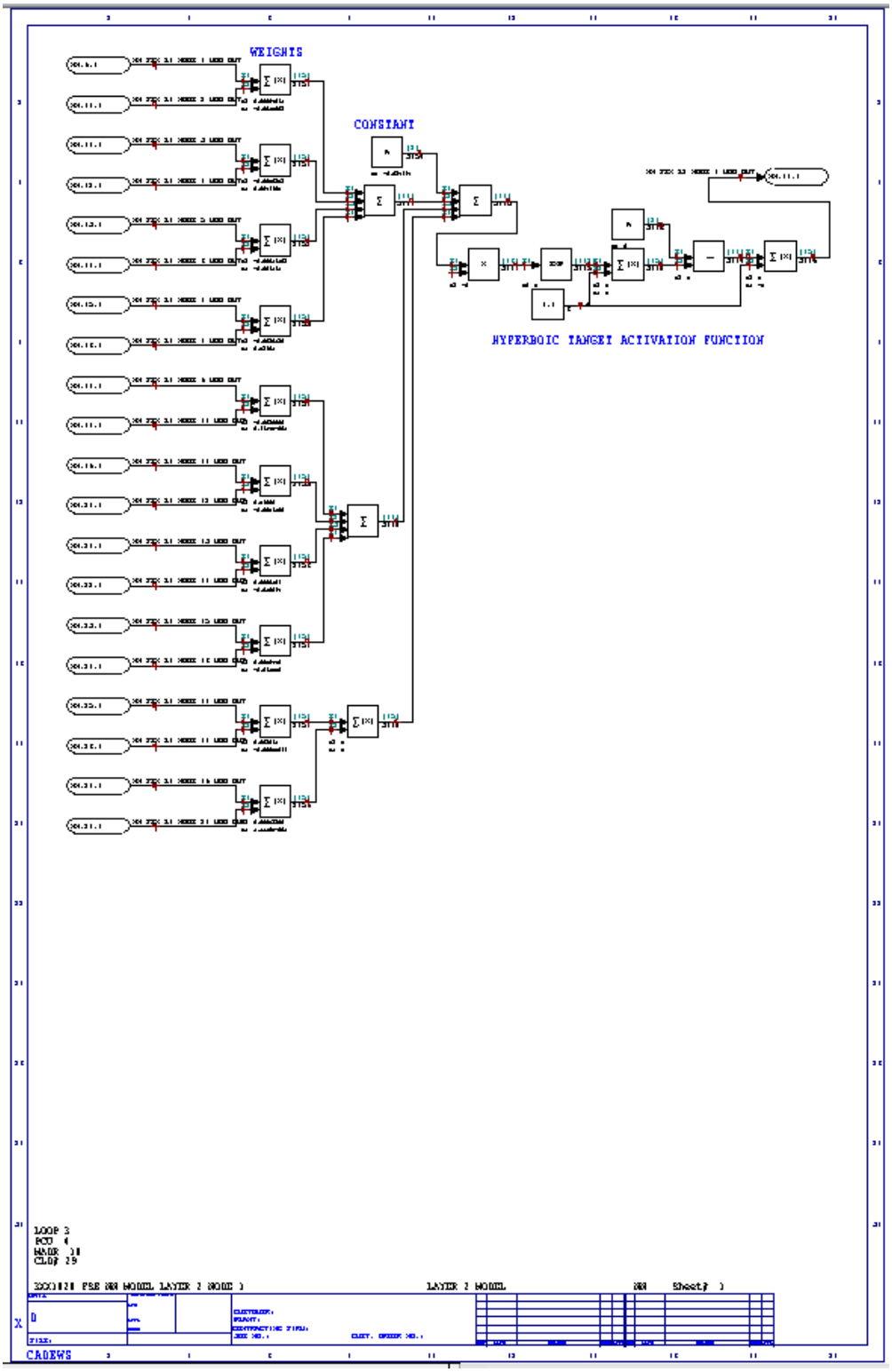


D-3 DCS Neural Network Model Logic

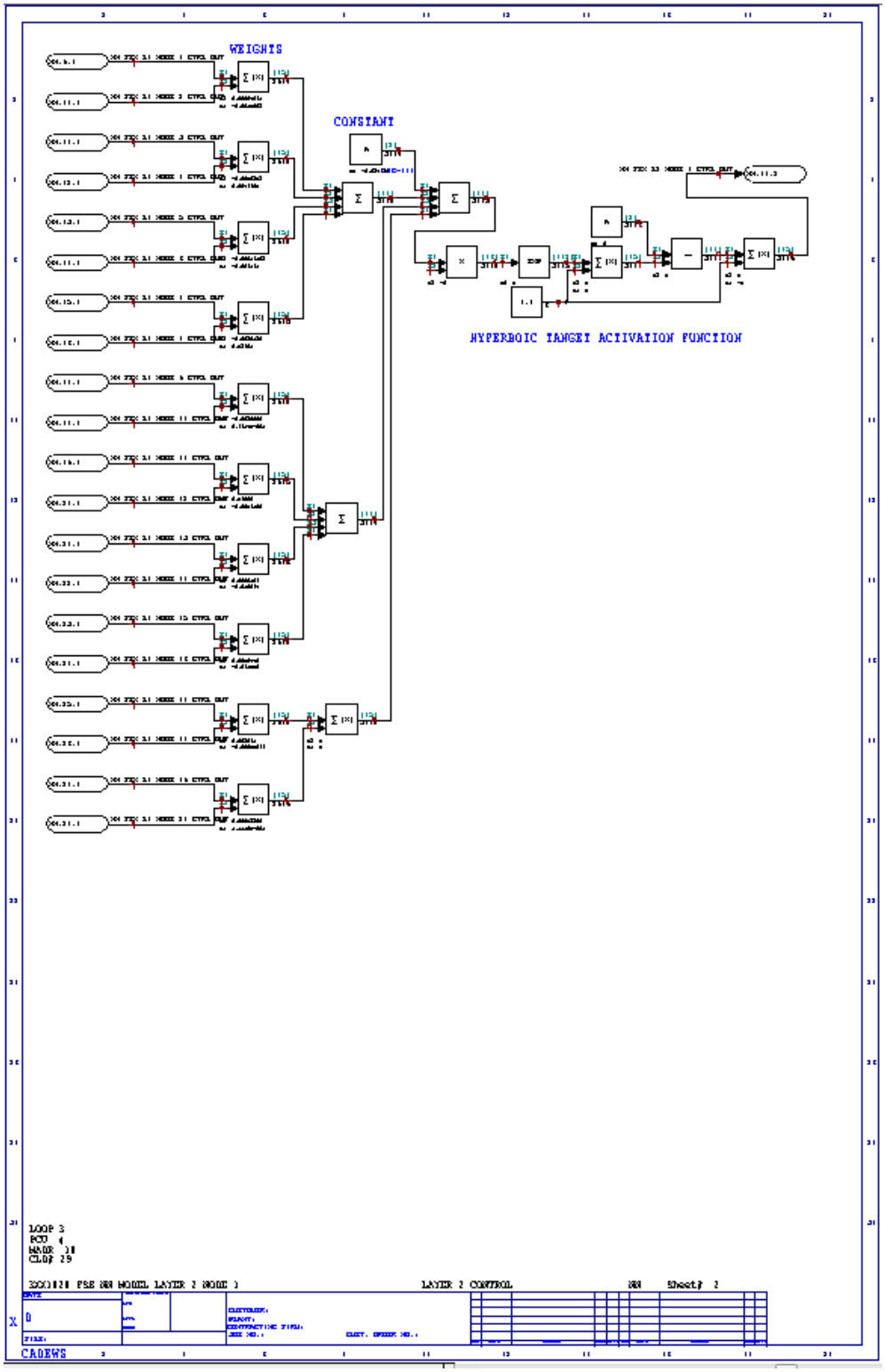
Model Verification and Control Neural Networks Layer 1 Node 1. **Only the first node of the layer is shown since only the weights and constants differ for the remaining 19 nodes.*



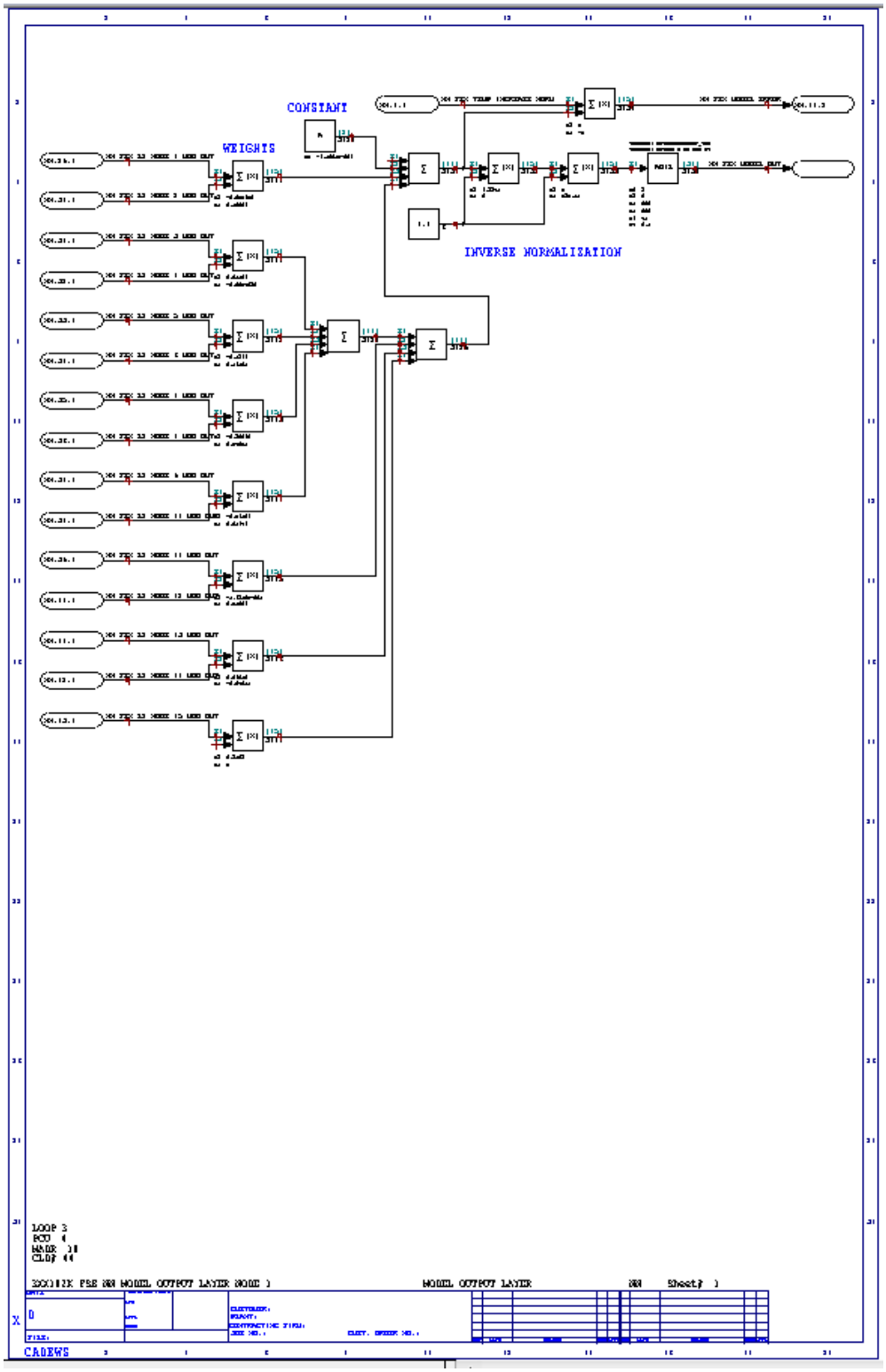
Model Verification Neural Network Layer 2 Node 1. **Only the first node of the layer is shown since only the weights and constants differ for the remaining 14 nodes.*



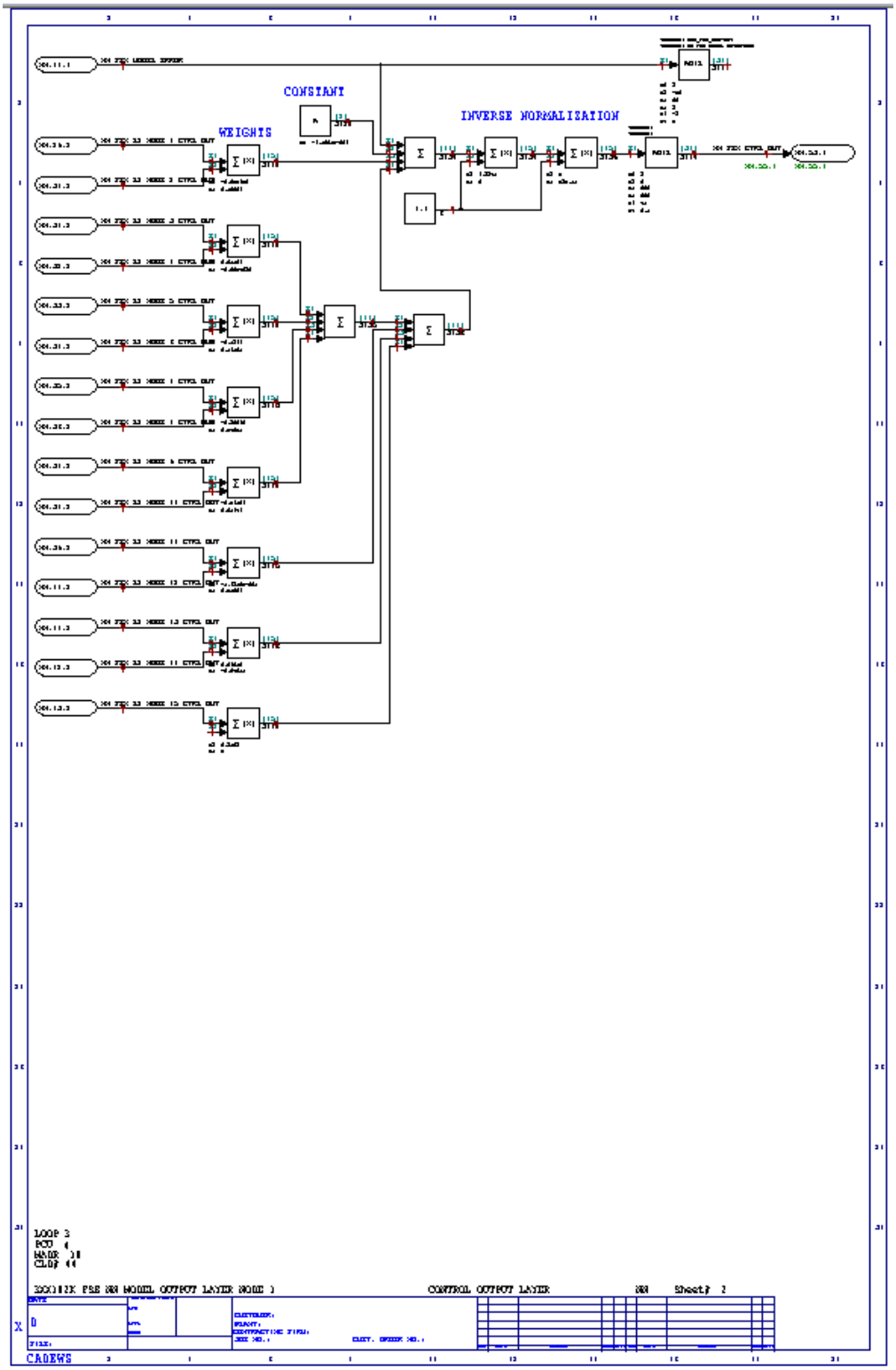
Control Verification Neural Network Layer 2 Node 1. **Only the first node of the layer is shown since only the weights and constants differ for the remaining 14 nodes.*



Model Verification Neural Network Output Node

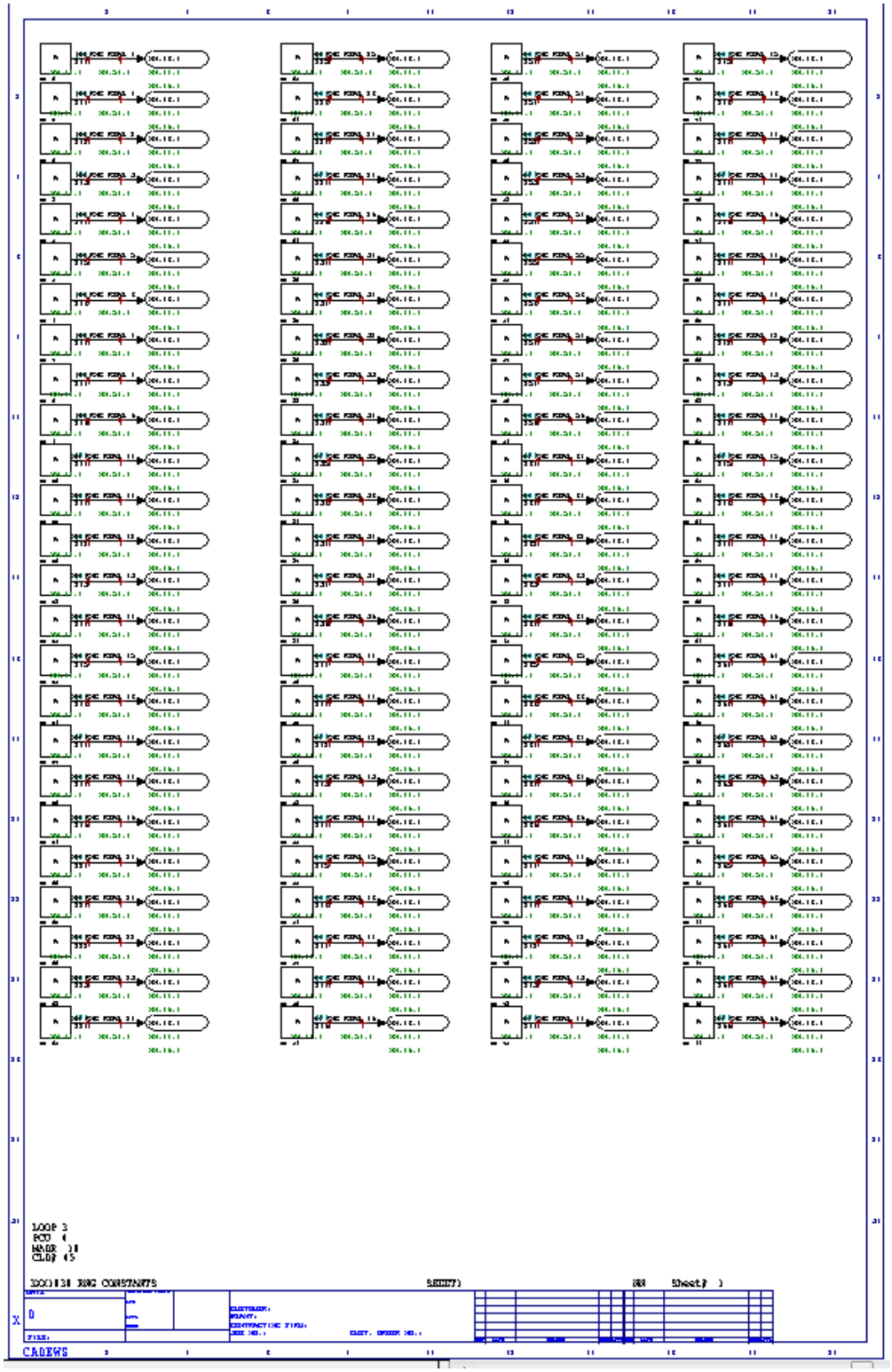


Control Neural Network Output Node.

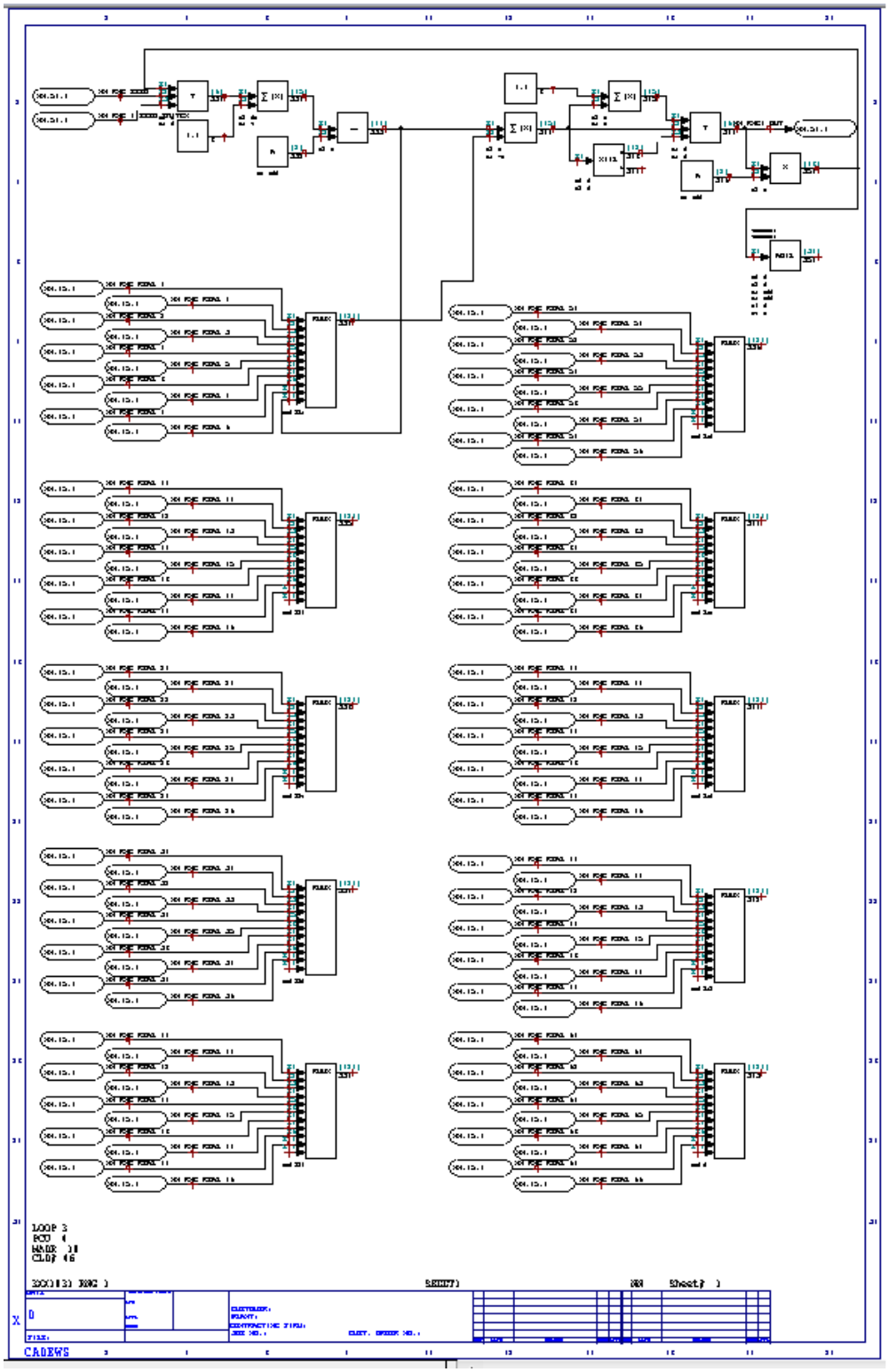


D-4 DCS Random Number Generation Logic

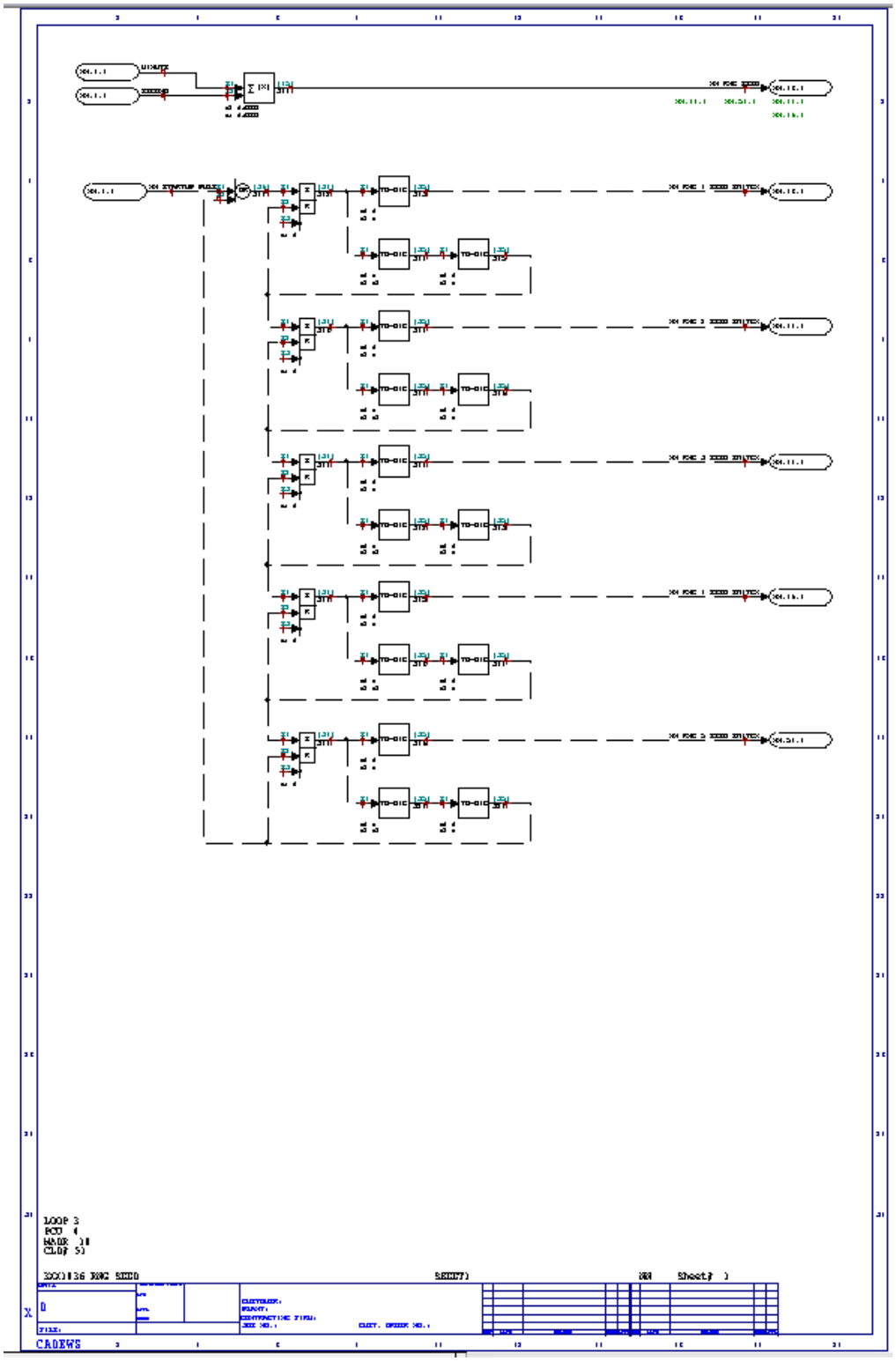
Random Number Generator Rounding Constant Blocks



Random Number Generator 1. **Only the first random number generator is shown since only the seed input and output connections differ for the other 4 random number generators.*

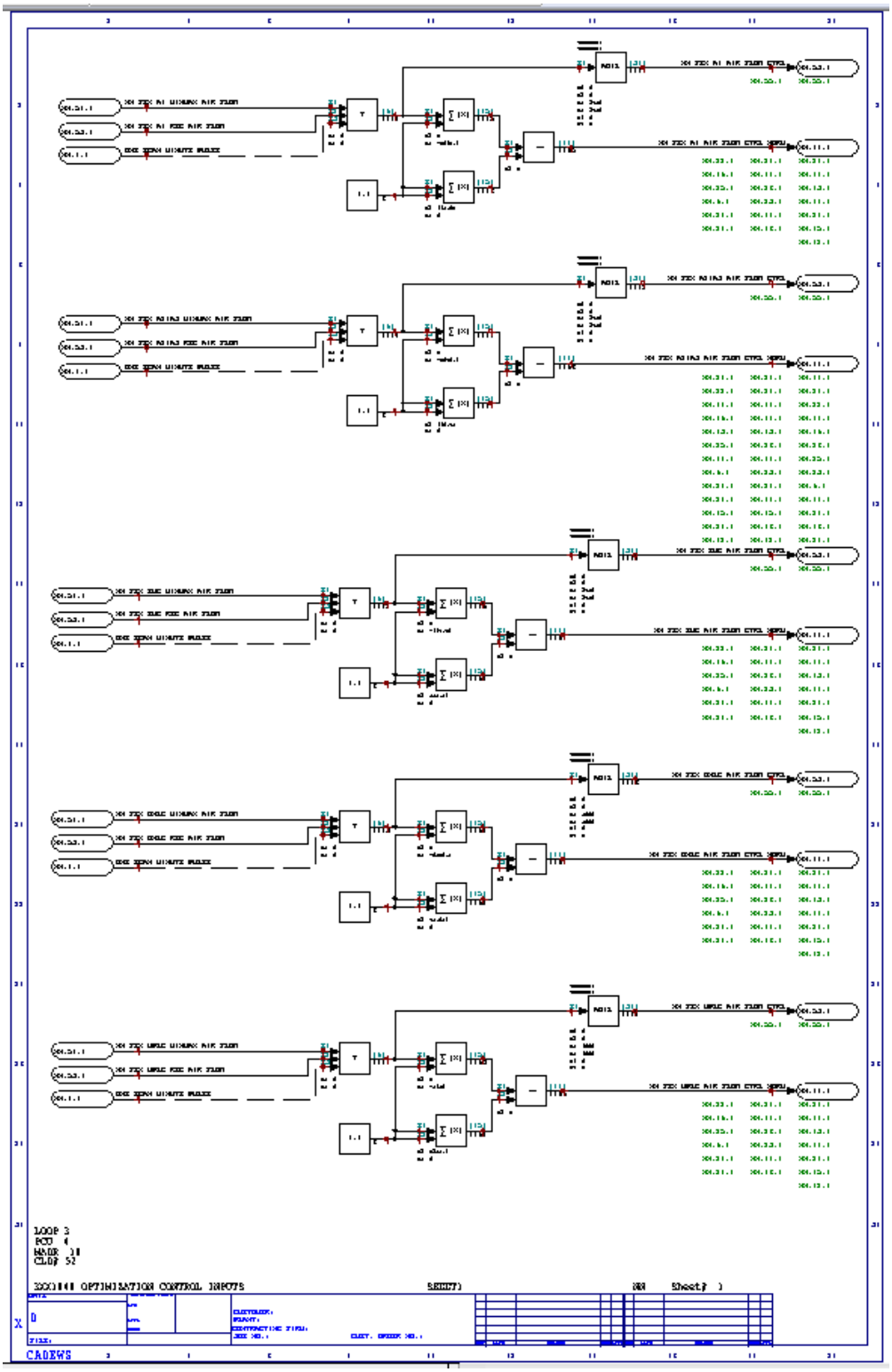


Random Number Generator Seed Logic

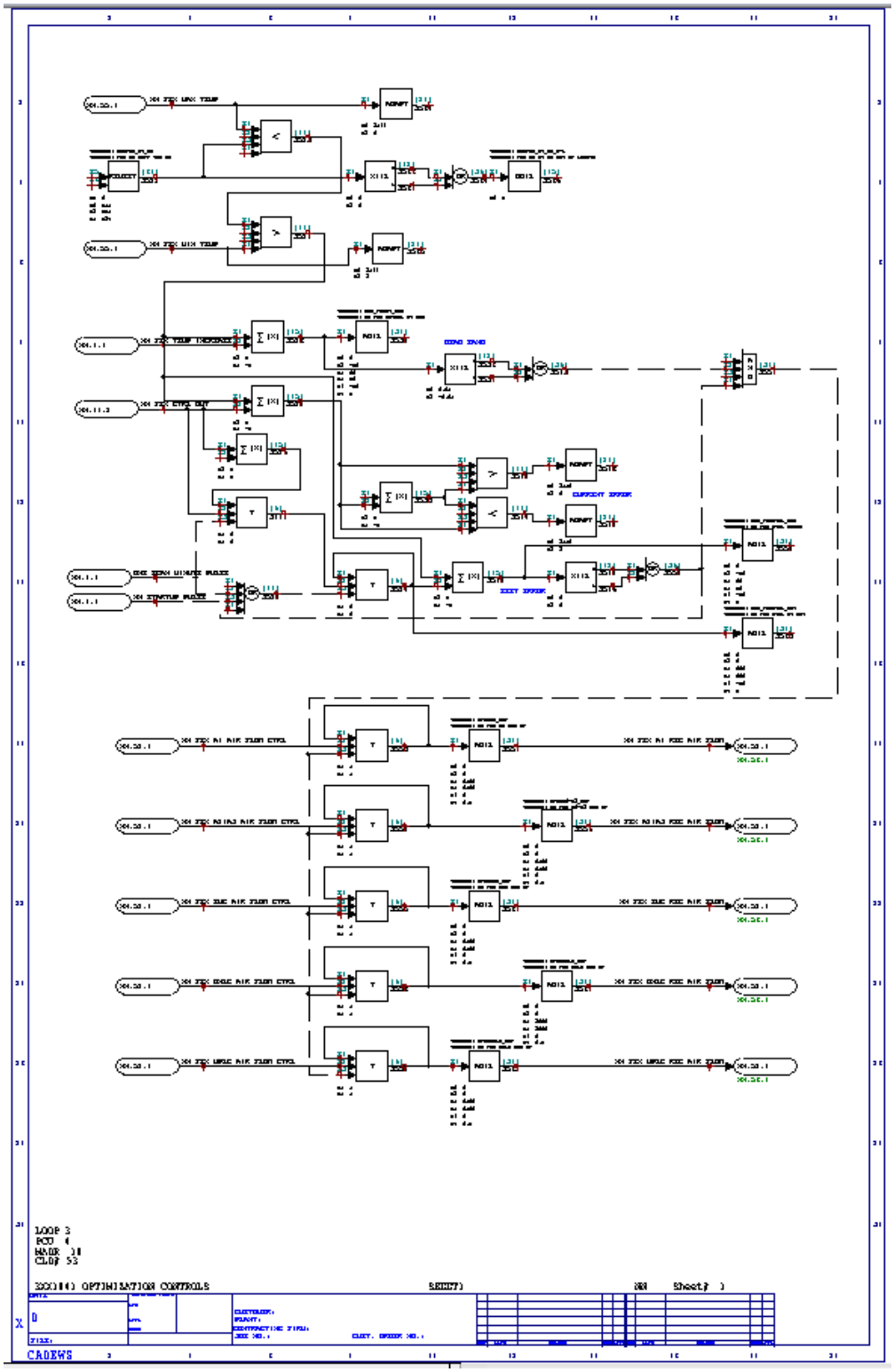


D-5 DCS Optimization Logic

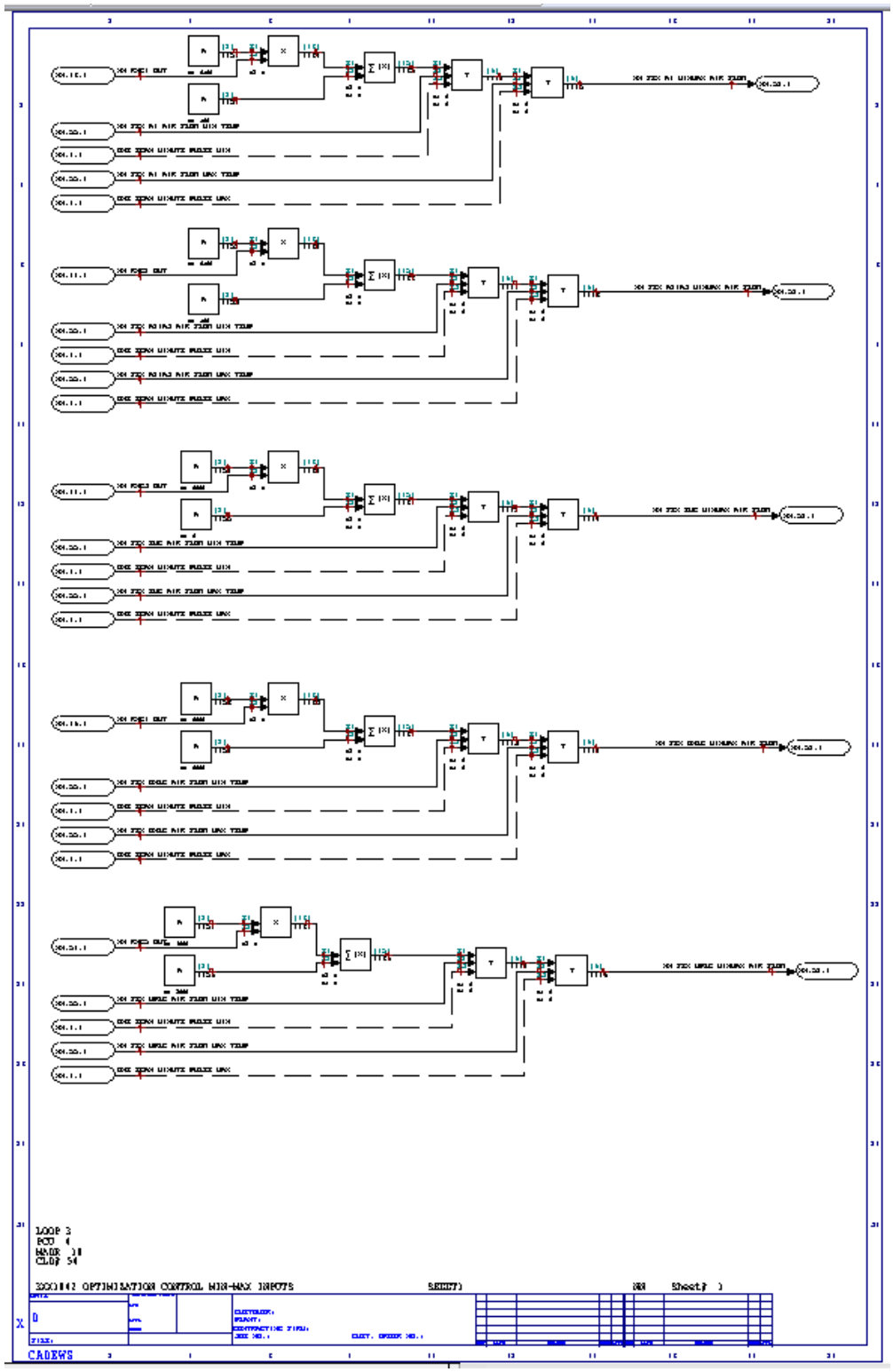
Optimization Algorithm Air Flow Input Signal Normalization



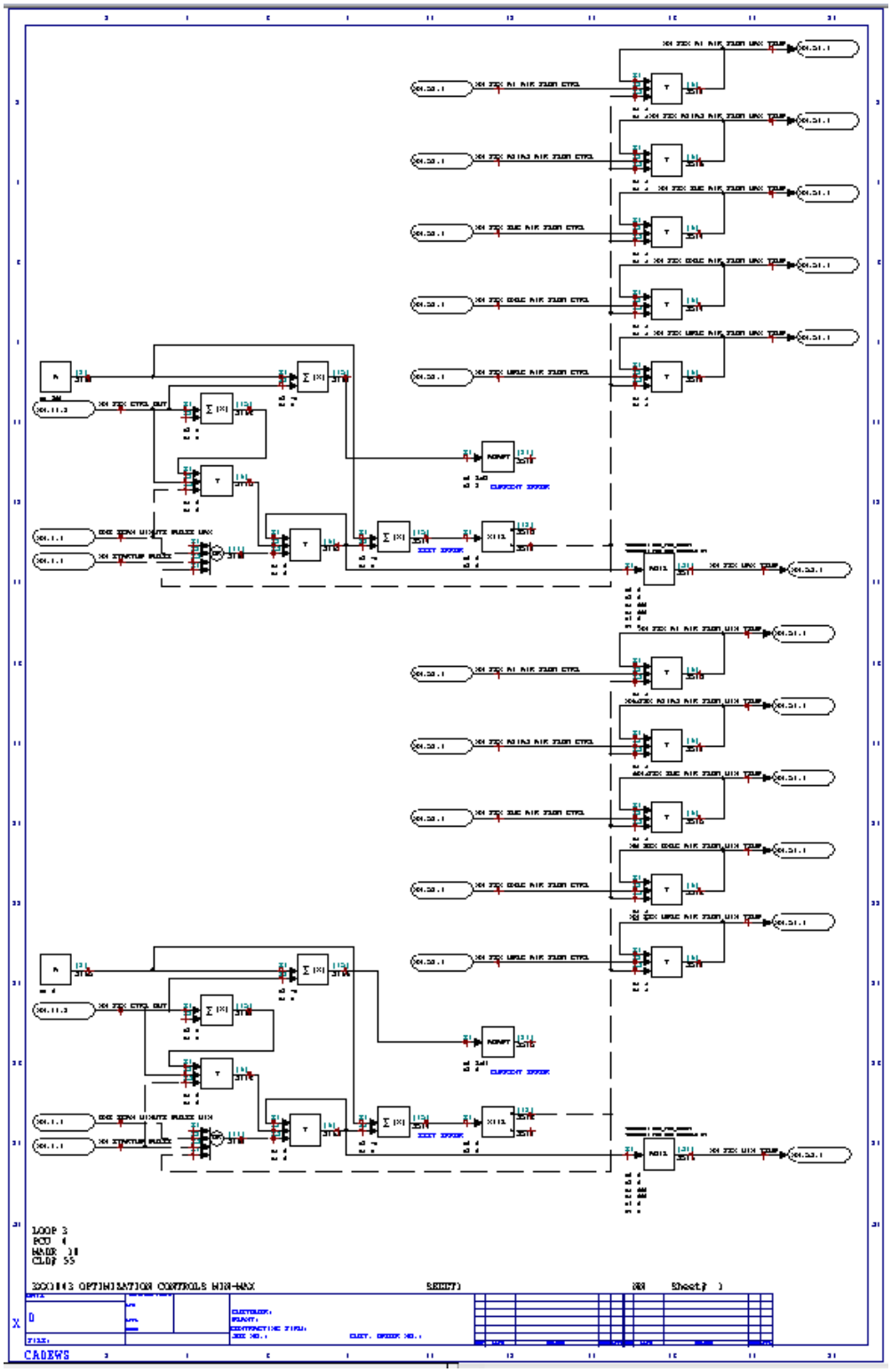
Optimization Algorithm Controller Airflow Selection



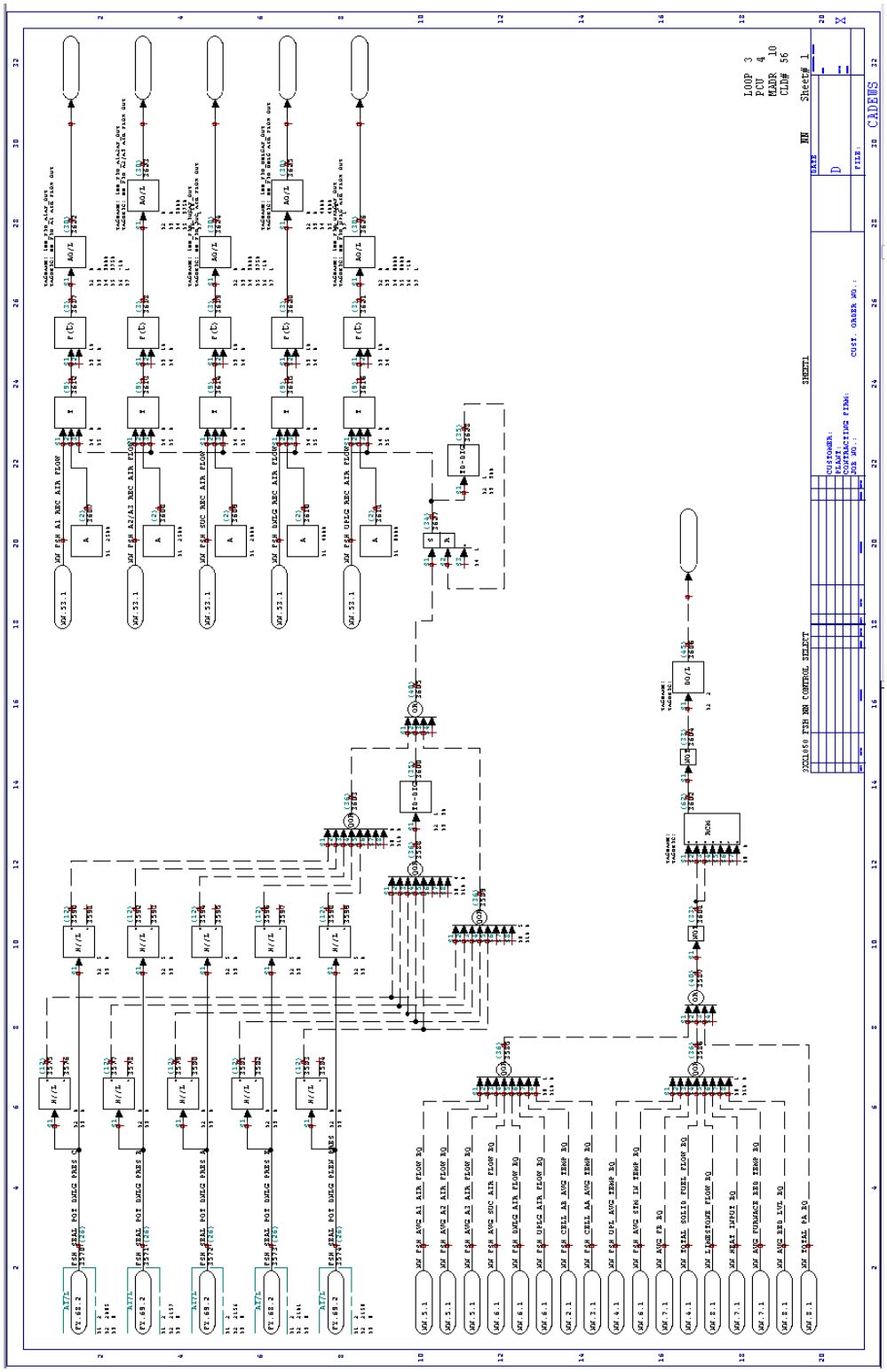
Random Number Generator Output Signal Conversion to Air Flow Values



Optimization Algorithim Minimum/Maximim Controller Capability Air Flow Selection



Intrex Flush Logic and Controller Output Signal Conditioning



Bibliography

1. [Online] http://en.wikipedia.org/wiki/Fluidized_bed_combustion.

2. The JEA Large-Scale CFB Combustion Demonstration Project. *National Energy Technology Laboratory.* [Online] 2003. http://www.netl.doe.gov/technologies/coalpower/cctc/topicalreports/pdfs/topical22.pdf.

3. **Kang, John and Thomas, Frank.** JEA Increases Power Output Through Efficiency Improvements. *Power.* October, 2011, Vol. 155, 10.

4. **Kang, John, et al.** Reducing Ash Agglomeration in JEA's CFB Boilers. *Power.* October, 2012, Vol. 156, 10.

5. *Intelligent coordinated control of circulating fluidized bed boiler-turbine unit.* **Li, Xiao-Feng, Chen, Shi-He and Zhong, Qing.** Toronto : IEEE, 2010. 2010 Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS). pp. 1-7. ISBN 978-1-4244-7859-0 .

6. *Neural network predictor of a circulating fluidized bed.* **Davari, A., Macha, S. and Sankar, R.** Athens : IEEE, 2001. Proceedings of the 33rd Southeastern Symposium on System Theory. pp. 355-357. ISBN 0-7803-6661-1 .

7. *IEE Colloquium on `Genetic Algorithms for Control Systems Engineering'.* London : IEEE, 1993. Digest No. 1993/130.

8. *Genetic Approach to Pole Placement in Linear State Space Systems.* **Cassell, Arnold and Choi, Chiu.** Jacksonville, FL : s.n., March 2012. Proceedings of the 44th IEEE Southeastern Symposium on System Theory. pp. 24-29.

9. **Barnard, E and Wessels, L.F.A.** Extrapolation and Interpolation in Neural Network Classifiers. *IEEE Control Systems Magazine.* October 1992, Vol. 12, 4.

10. **Samad, T and Annaswamy, AM.** *The Impact of Control Technology.* s.l. : IEEE Control Systems Society, 2011.

11. **Draeger, Andreas, Engell, Sebastian and Ranke, Horst.** Model Predictive Control Using Neural Networks. *IEEE Control Systems magazine.* October 1995, Vol. 15, 5.

12. **Nguyen, Derrick and Widrow, Bernard.** Neural Networks for Self-Learning Control Systems. *IEEE Control Systems magazine.* April 1990, Vol. 10, 3.

13. **ABB.** Composer for Harmony Function Code Application Manual. *WBPEEUI210504E0_V1/V2.* 2005.

14. **Montgomery, Rugner, Hubele.** Engineering Statistics. New York : John Wiley & Sons, 2004.

15. **Sola, J. and Sevilla, J.** Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on Nuclear Science.* June 1997, Vol. 44, 2.

16. *Multilayer Feedforward Networks are Universal Approximators.* **Hornik, Stinchcombe, White.** 5, Oxford, UK : Elsevier Science Ltd., 1989, Neual Networks, Vol. 2, pp. 359-366.

17. **Richards, J.A. and Jia, X.** *Remote Sensing Digital Image Analysis.* 5th Edition. New York : Springer-Verlag, 2005.

18. **Leverington, David.** A Basic Introduction to Feedforward Backpropagation Neural Networks. *Tennessee Tech University.* [Online] 2009. http://www.webpages.ttu.edu/dleverin/neural_network/neural_networks.html.

19. **Gallant, S.I.** *Neural Network Learning and Expert Systems.* Cambridge : MIT Press, 1993.

20. **Smith, Steven W.** *The Scientist's and Engineers Guide to Digital Signal Processing.* San Diego : California Technical Publishing, 1997. 0-9660176-3-3.

21. **D. J. Krusienski, W. K. Jenkins.** Design and Performance of Adaptive Systems Based on Structured Stochastic Optimization Strategies. *IEEE CIRCUITS AND SYSTEMS MAGAZINE.* First Quarter 2005, 2005, pp. 8-20.

22. *Neural Generalized Predictive Control, A Newton Raphson Implementation.* **Soloway, Donald and Haley, Pamela J.** Dearborn : IEEE, 1996. Proceedings of the 1996 IEEE International Symposium on Intelligent Control. 0-7803-2978-3 .

23. *Neural Network Model Predictive Control with Genetic Algorithm Optimization and Its Application to Turbofan Engine Starting.* **Yu, Bo and Zhu, Jihong.** s.l. : IEEE Computer Society, 2010. Proceedings of the 2010 Second International Conference on Intelligent Human-Machine Systems and Cybernetics - Volume 02. ISBN 978-0-7695-4151-8.

VITA

David Graduated from the State University of New York at Morrisville in the spring of 1999 with an associates in applied science in Electrical Engineering Technology. He worked a summer internship at Marquardt Switches in Cazenovia New York as a manufacturing engineering intern after which time he worked nearly a year for Ametek Rotron in Woodstock New York as a technician performing root cause analysis on the failure of electronically commutated DC motors.

David moved to Jacksonville in late 2000 and started working for AT&T Broadband, later Comcast, as a communications technician. He began attending classes part time at the University of North Florida in the spring of 2002 and continued work and school until he graduated from UNF in December 2007 with a Bachelors of Science Degree in Electrical Engineering with a minor in Mathematics.

In January 2008 David began pursuing his Masters of Science in Electrical Engineering from the University of North Florida. In February 2008 he began working at JEA's Northside Generating Station as a controls engineer with the instrumentation and process controls group. David received his Professional Engineering license from the state of Florida in January 2013 and his Six Sigma Green Belt Process Improvement Certification in May 2013. David is scheduled to receive his Master of Science in Electrical Engineering from the University of North Florida in December of 2013.