STARS

Electronic Theses and Dissertations

2019

SAW Correlator Temperature Compensation Using a Pulse Width Modulated Temperature Controller

Daniel Betancourt
University of Central Florida

Find similar works at: https://stars.library.ucf.edu/etd

University of Central Florida Libraries http://library.ucf.edu

STARS Citation

SAW CORRELATOR TEMPERATURE COMPENSATION USING A PULSE WIDTH
MODULATED TEMPERATURE CONTROLLER

by

DANIEL BETANCOURT
B.S. University of Central Florida, 2018

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Sciences
in the Department of Electrical & Computer Engineering
in the College of Engineering & Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2019

ABSTRACT

A Surface Acoustic Wave (SAW) correlator built on a Lithium Niobate substrate is temperature compensated in order to maintain a constant center frequency. Frequency shifts as a result of temperature variations limit device performance. An Arduino®-based PWM temperature controller is developed to read the device temperature from a resistance temperature detector located on the SAW wafer and to regulate its temperature to a specified setpoint by providing current to a heater which is co-located with the temperature sensor on the SAW correlator substrate. The final temperature controller achieves frequency shifts of 0.013 MHz from room temperature with a worst-case PPM experienced over 30°C of temperature variation of 0.48 PPM/°C. Linear and non-linear plant models are developed successfully to predict the device's temperature based on any input setpoint. Although there are alternatives to limit temperature drift at different temperatures, this thesis presents a simple method that works on a standard Lithium Niobate substrate.

ACKNOWLEDGMENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

ADC – Analog to Digital Converter

D – Derivative

I – Integral

IDT – Interdigital Transducer

P – Proportional

PID – Proportional Integral Derivative

PCB – Printed Circuit Board

PPM – Parts Per Million

PWM – Pulse Width Modulation

RF – Radio Frequency

RTD – Resistance Temperature Detector

SAW – Surface Acoustic Wave

TCF – Temperature Coefficient of Frequency

VNA – Vector Network Analyzer

CHAPTER 1: INTRODUCTION

This thesis presents the design considerations of temperature variations in the implementation of a SAW correlator used in communication-based applications. The velocity of the acoustic wave is dependent on the substrate material's temperature coefficient of frequency (TCF) (Bruckner 2). Hence, temperature change results in a drift in a SAW Correlator's center frequency. Despite their wide bandwidth, this can lead to poor correlator results under high or low temperatures (Brocato 17). This effect is even more prevalent when $LiNbO_3$ substrates are used due to their large TCF of (94 ppm/°C) (Yurish 160). Figure 1 shows the shift in frequency of a SAW correlator when temperature changes as measured directly from a VNA. The light trace shows the S21 parameter data when the correlator is operating at 40°C. The darker waveform shows the S21 data when the correlator is at 25°C. The total shift in frequency is about 1.5 MHz. Higher temperatures lead to a decrease in center frequency and lower temperatures increase it. This change corresponds to a change of 89.82 PPM/°C.
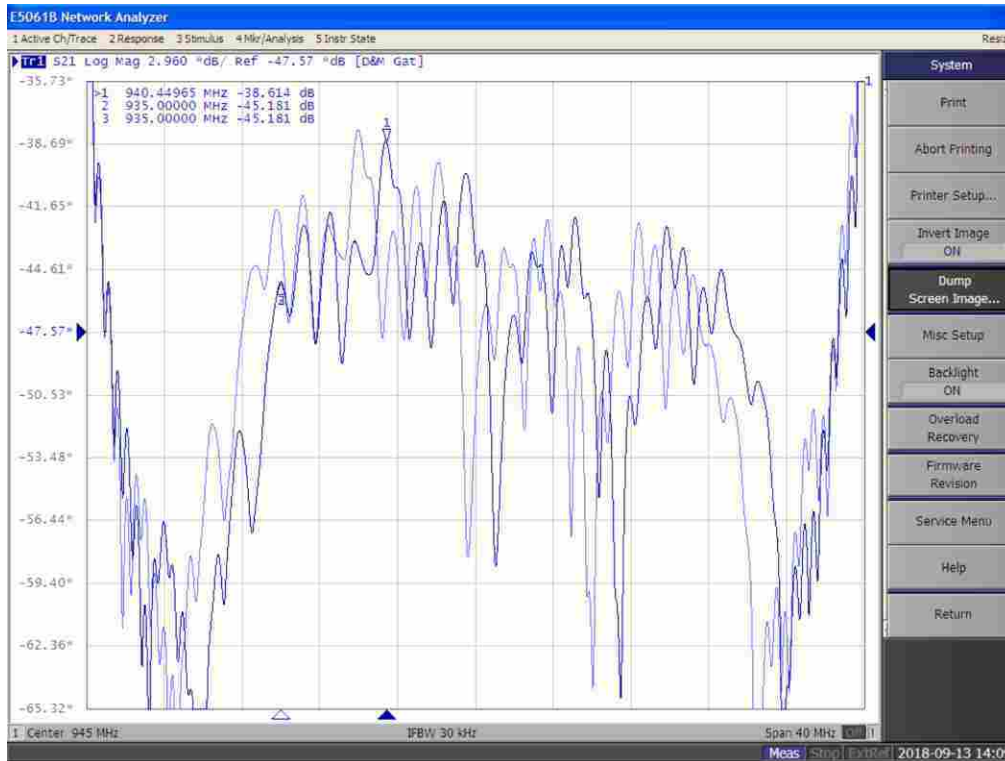
Figure 1: Initial frequency shift demonstration.

One way of reducing this shift in center frequency is by using St-Quartz substrates since they feature a negligible temperature coefficient (Yurish 160). However, LiNbO$_3$ achieves better insertion loss performance due to its higher electromechanical coupling coefficient and low propagation loss (Brocato 18). Layering piezoelectric materials to produce a composite substrate with optimal characteristics has been shown to produce lower temperature coefficients while maintaining a high electromechanical coupling coefficient (Gong).

This thesis explores an alternative way of improving temperature stability. The center frequency of a LiNbO$_3$ SAW correlator can be kept constant by implementing a closed-loop system that reads the current device temperature and adjusts the power supplied in order to keep

2

the correlator at a fixed temperature. Figure 2 shows a diagram of the block components that make up a closed-loop system.
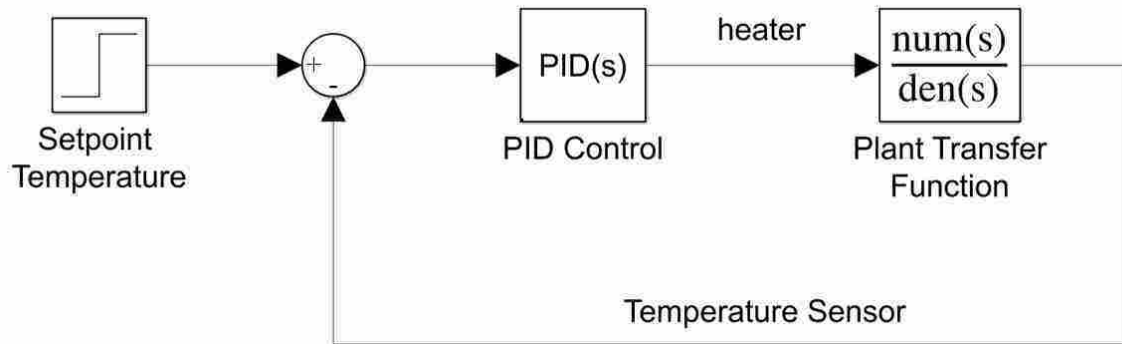
Figure 2: Closed-loop temperature controller diagram.

For this thesis, the PID control block of the control system is implemented through an Arduino®-based temperature controller PCB. The controller can adjust the setpoint and the PID terms for achieving the desired output response based on code uploaded through the Arduino® IDE. The heater and the temperature sensor are both implemented on the wafer along with the correlator. Both, the heater and temperature sensor are essentially resistors that rely on the temperature controller PCB for receiving power and reading the temperature sensor's output. Connections from the heater and temperature sensor to the temperature controller are possible by soldering the SAW device's package to a second PCB and using wire to connect them together. The design and programming of the PID controller PCB using the Arduino® IDE in Chapter 2. It is also programmed using a Simulink® block diagram to allow for real-time data acquisition and plotting, as well as on-the-fly tuning and setpoint changes of the controller.

3

MATLAB$^\circledR$ System Identification Toolbox$^{\text{TM}}$ is used in Chapter 3 to derive both, linear and non-linear models, and predict the output based on a given input for the open and closed-loop systems. Chapter 4 covers the experiment setup including all connections and the tuning and testing of the controller.

By gathering and post-processing the device's S21 data under different temperatures, its shift in center frequency can be calculated. This shift in frequency can be expressed as a PPM/°C value by examining how much PPM changes per degree Celsius. It is possible test how well its temperature is being regulated by the controller by taking a second S21 measurement with the controller regulating the SAW's device temperature. This can be done by measuring how much PPM changes and using the measured PPM/°C value to obtain the error in the temperature readings. An ideal temperature controller would maintain the shift in frequency constant. Results pertaining to the shift in frequency with temperature, along with the required S21 processing, are discussed in Chapter 6. Further background information about SAW device theory and device packaging is also provided in this chapter.

CHAPTER 2: DESIGN OF A PID TEMPERATURE CONTROLLER

Control Theory

Maintaining a temperature constant using a heater requires the use of a feedback loop. A closed-loop system is continuously taking the difference between the setpoint temperature and the process variable (the sensed temperature) at a specified sample rate. The system's goal is to maintain this difference, the error term, constant at 0. One way to control how the system behaves when there is error is through a PID algorithm. The output of a PID controller dictates how much power is applied to the actuator of the system (the heater). For example, if the error term is large and if the PID controller is tuned appropriately, the heater will power on at full power. This algorithm consists of proportional, integral, and derivative parameters and is represented mathematically by Equation 1 (Messner, PID Controller). The controller output u(t) is used to power an actuator based on the PID parameters and the error between the setpoint and the measured process variable. K_p is the proportional term, K_i is the integral term, K_d is the derivative term, and e(t) is the error.

$$u(t) = K_p e(t) + K_i \int e(t)\, dt + K_d \frac{d\, e(t)}{dt} \qquad\qquad (1)$$

The Laplace transform can be applied to transform Equation 1 into Equation 2, which gives an algebraic expression that can be easily used to generate a control system model. Equation 2 shows that the integral and derivative terms are functions of the proportional term K_p and the integral or derivative times T_i and T_d (Haugen 44).

$$U(s) = E(s)(K_p + K_i \frac{1}{s} + K_d s) = E(s)K_p(1 + \frac{1}{T_i s} + T_d s) \tag{2}$$

The proportional element of the controller simply changes the output in proportion to the error e(t). The proportional band is the temperature range around the setpoint where the heater can be driven to duty cycles greater than 0% and lower than 100% (PWM). The controller behaves as an on-off controller outside this range. If the proportional band is too small (high P term), the controller will behave as an on-off controller that never reaches the setpoint temperature (Haugen 34-35). The integral term reduces the steady state offset from the setpoint that is not accounted by the proportional term. A shorter integral time (higher I term) allows the controller to reduce this error (Haugen 38). An integral time that is too short will result in overshoot that takes time to correct, while an integral time that is too long is avoided because offset should be eliminated quickly (Skogestad 295).

The derivative element of the controller senses the change in error over a short time period and adds a portion of the error to the output (Haugen 40). This is the part of the controller that quickly adjusts the output closer to the set point in the case of an unexpected disturbance or overshoot (Messner, PID Controller). In other words, it keeps the output within the proportional band. A higher derivative term leads to a decreased response to disturbances (Haugen 92-93). However, if the term is 0, the controller will behave as a PI controller that does not try to correct for disturbances.

The plant is the part of the system that is controlled; in this case, it is the heater. This block can be modeled using mathematical models, such as a transfer function, that describe a system's output based on an input. This serves to characterize and predict the behavior of a

system. To close the loop, a sensor needs to repeatedly gather samples of the process variable.

Without this, temperature is not regulated and the system changes to open-loop configuration.

The PID algorithm, setpoint temperature, and feedback loop are all implemented in software

written for the Arduino® temperature controller for this thesis.

Board Schematic & Layout

The temperature controller schematic design is based on the Arduino® UNO schematic

since this is used for prototyping (Banzi). 5 volts (VCC) is used across the schematic for power.

Additional components and connections are added to the Arduino® chip, the Atmega328PU, as

required. The Arduino® chip connections are shown in Figure 3.
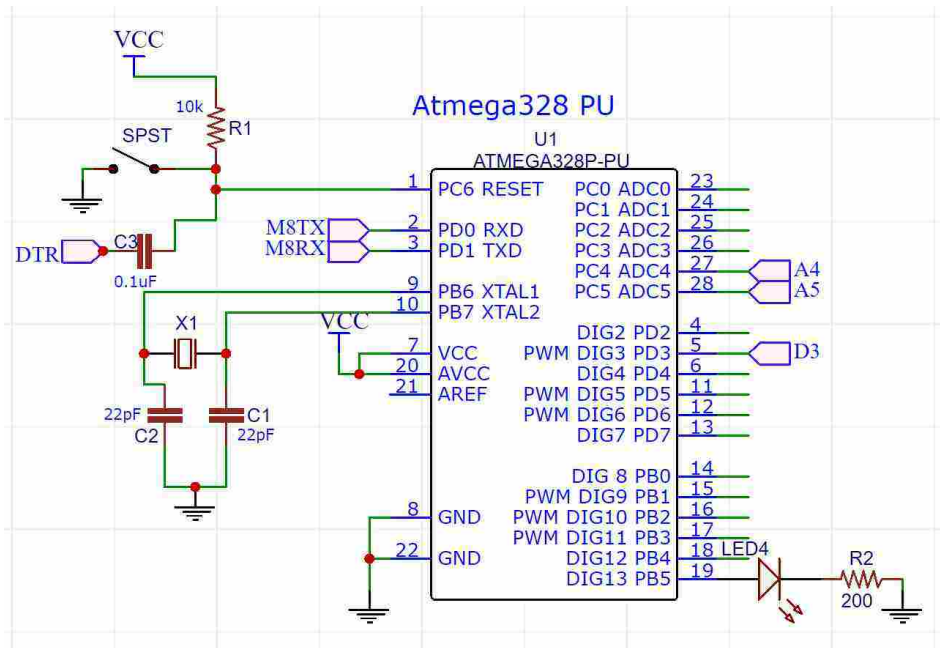
Figure 3: Schematic showing Atmega328PU connections (Banzi).

An FT232RL USB to Serial UART interface allows external programming of the Arduino® chip (FTDI). It connects the data lines D^+ and D^- of the USB to the RX and TX pins on the Arduino® chip. LED lights for TX and RX activity are added. This portion of the schematic is shown in Figure 4.
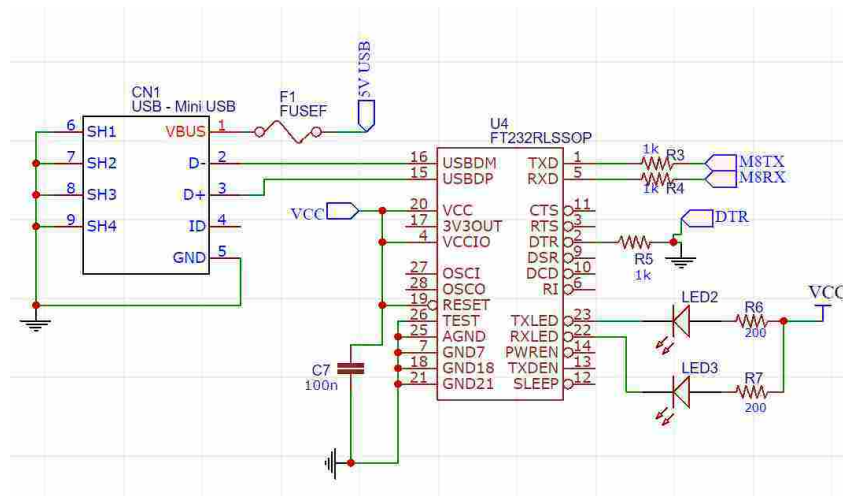
Figure 4: Schematic showing USB to serial interface and USB connector (FTDI).

The part of the schematic used for the temperature sensor is a voltage divider with the RTD set to R_{RTD} and a resistor of known value (1kΩ) set to R_{12}. The voltage divider's output is read by a 16-bit ADC, the ADS 1115 module, to allow for greater accuracy when compared to the Arduino® UNO's built-in 10-bit ADC. Using a 16-bit ADC gives the Arduino® the ability to read 65,536 (2^{16}) discrete analog levels, whereas a 10-bit ADC only allows for 1,024 (2^{10}) discrete analog levels. For the temperature controller, this means being able to read changes of 0.07°C instead of 2.5°C. This specific module is chosen because it uses the versatile I^2C

interface, it is powered by 5 volts, and it has programmable gain (Earl). The connections between

the voltage divider and the module are shown in Figure 5.
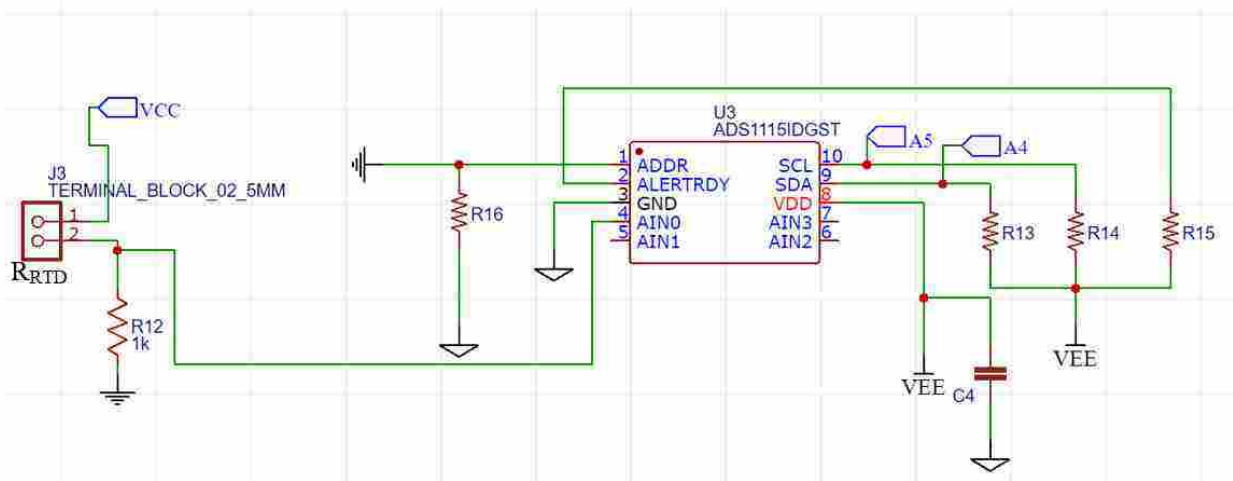
Figure 5: Schematic for reading temperature (Earl).

The part of the schematic used for powering the heater is based on a logic-level N-

channel MOSFET configured to be used as a switch, as shown in Figure 6. The drain of the

MOSFET is connected to the heater, which is powered using 22 volts. The gate is connected to

the output PWM signal (D3) and the source is connected to ground. This configuration works as

a switch since power is supplied to the heater resistor only when the output PWM pin is toggled

high by the PWM output. The duty cycle of the output controls the average power that is

supplied to the heater.
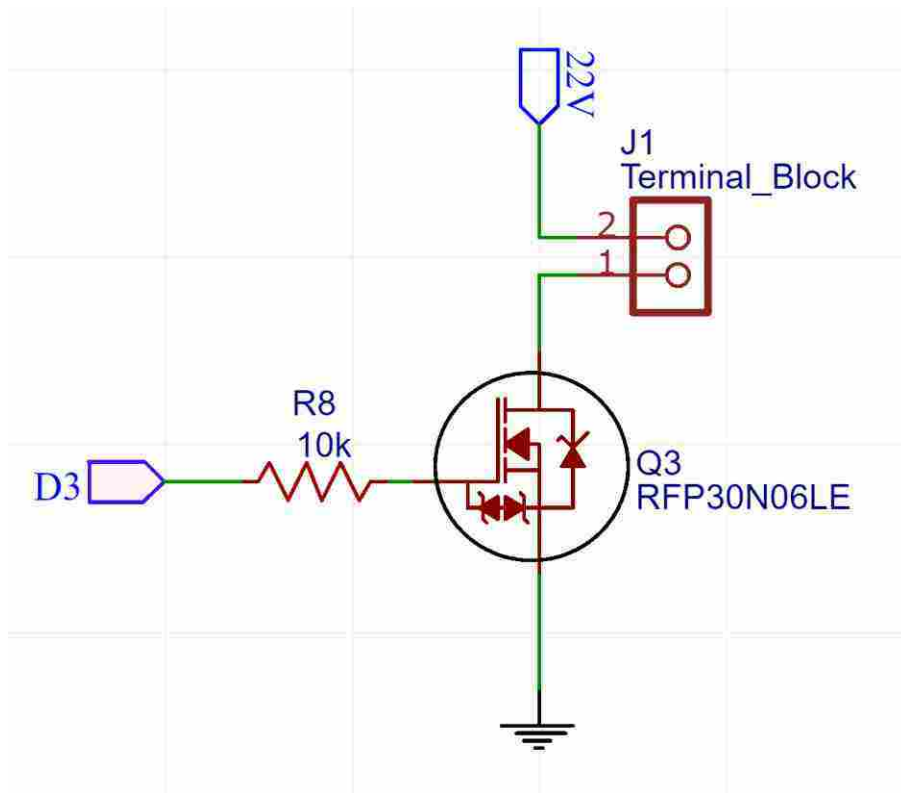
Figure 6: Schematic for controlling power to heater.

A 5-volt regulator is used for regulating the output voltage from the DC input connector. A switch is used to toggle whether the 5 volts for powering the PCB is sourced directly from the USB or from the regulator. The regulator and switch connections are shown in Figure 7.
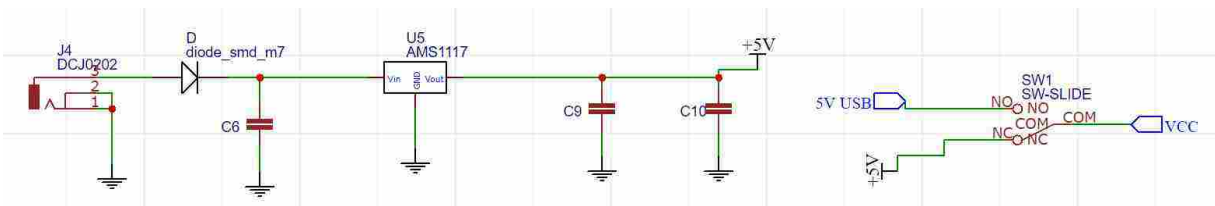
Figure 7: Schematic for power supply.

The temperature controller is a 2.170 x 3.175 inch FR4 Printed Circuit Board (PCB) with surface mount components, as shown in Figure 8. Previous iterations of the PCB included an LCD screen and rotary encoder where it was possible to change temperature setpoint using the PCB. These two components were removed for testing since setpoint changes are implemented directly from a computer connected to the controller. Changes from the original PCB include the addition of the 16-bit ADC for accurate temperature readings and the ability to change the source of the 5 volts for power supply via a switch. Connections from the PCB to the power supply and device are done using screw terminal blocks.
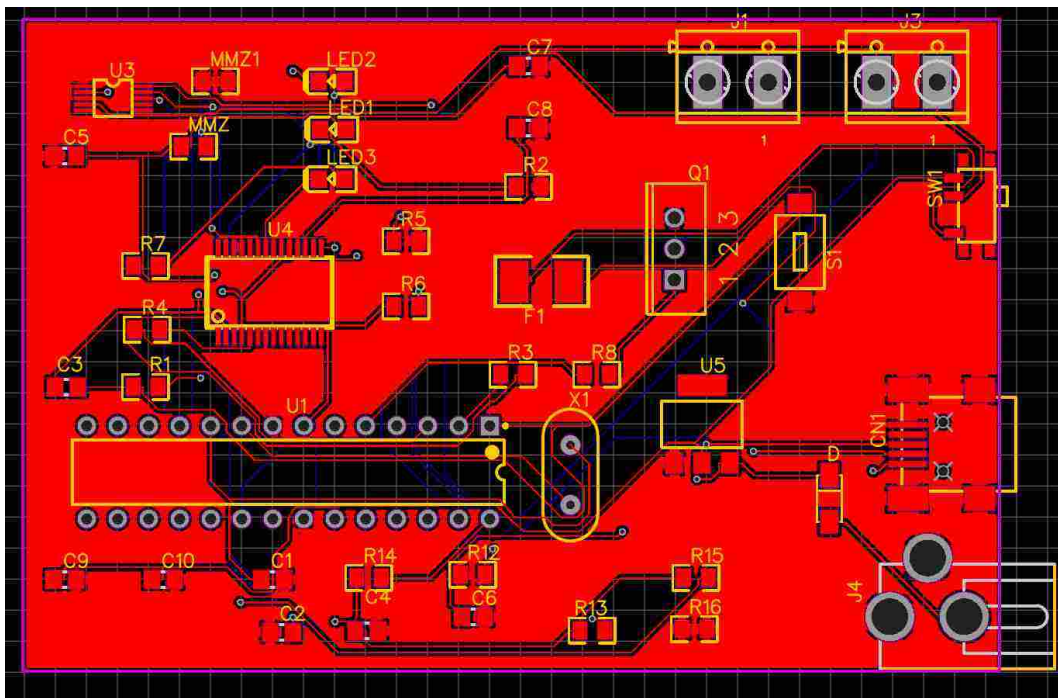
Figure 8: PCB layout design.

Programming the Controller

Programming and uploading code to the ATMEGA328-PU is done using the open-source

Arduino IDE. External libraries for the PID algorithm and for interfacing with the ADS1115 16-

bit ADC are implemented. The PID library used allows for on-the-fly tuning and setpoint

changes and a constant sample time (Beauregard). The ADS 1115 16-bit ADC uses I^2C and

allows for four single-ended readings to reach an accuracy of 0.125 mV for each additional bit

(Earl). The code simply sets up the setpoint temperature and input (measured temperature) to

obtain an output (duty cycle) using the PID algorithm (Beauregard). It repeatedly prints these

values to the serial port for viewing. The function reading_to_temp is created to average ten

ADC voltage readings and convert them to temperature each time the loop executes. The full

code is shown in Appendix B.

Using the Simulink® Support Package for Arduino® Hardware, it is possible to program

the Arduino® to perform the same functions as the code in Appendix B for both, closed-loop and

open loop-controllers (MathWorks, Simulink® Arduino®). This makes it simple to view and

record all input-output data in real time and to examine the accuracy of any plant model. The

Simulink® diagram developed, which is shown in Figure 9, gives the user the option to choose

between open and closed loop configurations by means of a software switch. When the switch is

configured to open-loop, a percent duty cycle is sent to the output PWM pin. This number is first

multiplied by 2.55 to account for the fact that the maximum Arduino® PWM value is 255. When

the switch is in the closed-loop configuration, the value written to the Arduino® digital pin is the

result of summing the P, I, and D terms multiplied by 2.55. The input to the PID term blocks is

the error, or the difference between setpoint and measured temperature. The block labeled

'Query Instrument' outputs the measured temperature for use in the closed-loop case. The

measured temperature is plotted against the output of a linear or non-linear model that predicts

temperature based on the controller PID terms and setpoint.
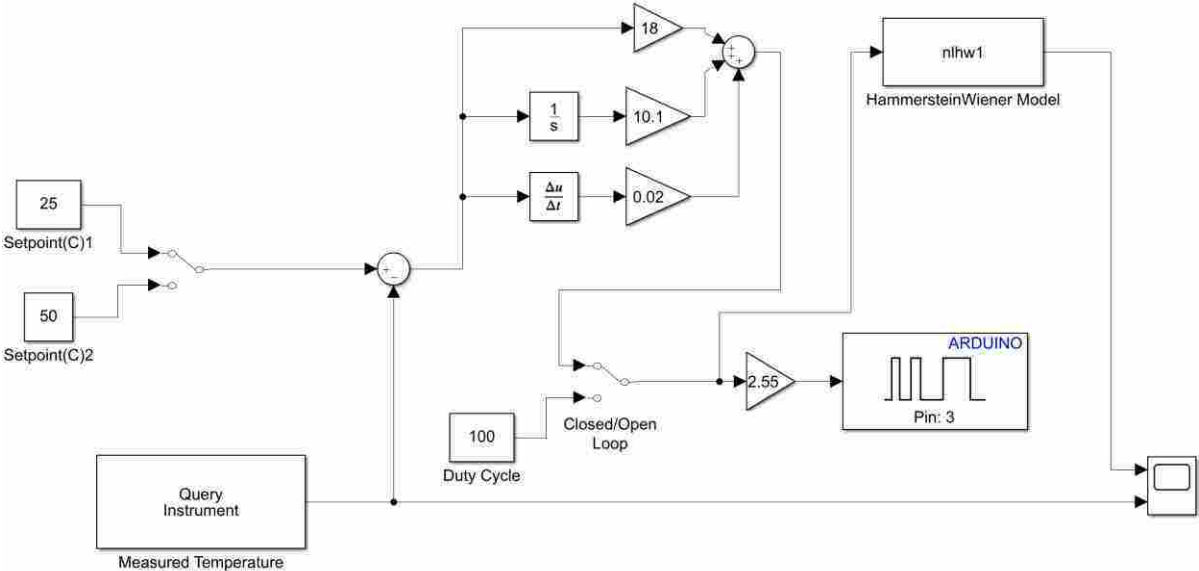
Figure 9: Simulink® diagram used for programming controller.

CHAPTER 3: MODEL RESULTS

Linear Model Results

Obtaining Linear Model from Open-Loop Controller

A diagram for the open-loop control system is shown in Figure 10. Since there is no

feedback loop, the plant relies on the specified input to supply varying or constant power to the

plant. Since the system uses a PWM-based controller, the input power is described as a duty

cycle ranging from 0% to 100% and not a setpoint temperature.
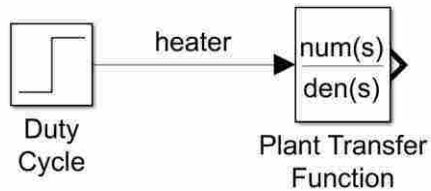
Figure 10: Open-loop diagram of control system.

There are two ways to obtain plant models from an open-loop step response. The first

method is to do it manually by obtaining gain, time delay, and time constant directly from a step

response plot (Messner, System Analysis). The manual method for deriving a first-order transfer

function from an open-loop step response starts by examining the general form of a first-order

transfer function P(s).

$$P(s) = e^{-sT} * \frac{K}{(\tau*s)+1}$$ (3)

K is the gain found by dividing the change in temperature by the change in setpoint temperature. τ is the time constant and it describes the speed at which the system's response changes. For first-order transfer functions, it is described as the time at which the output is 63% of its steady-state value (Messner, System Analysis). T is the time delay and it is calculated by finding the amount of time it takes for the temperature to start increasing when there is a step change in the setpoint temperature. Transfer function models should only be of high enough order to characterize the dynamics of a system to the needed degree of accuracy. Hence, being able to model the dynamics of the system using a first-order model would be the best-case scenario.

This method is applied to the step response of the open-loop temperature controller. After applying the manual procedure and modifying the variables for best fit to the SAW correlator controller to account for measurement error, the plant transfer function obtained is shown in Equation 4.

$$P(s) = e^{-s*0.3} * \frac{0.447}{(25*s)+1} \qquad\qquad (4)$$

Figure 11 shows a Simulink® diagram used to compare the step responses of the open-loop controller with the that of the first-order model for the plant. The time delay is added using Simulink® transport delay block (MathWorks, Transport Delay).
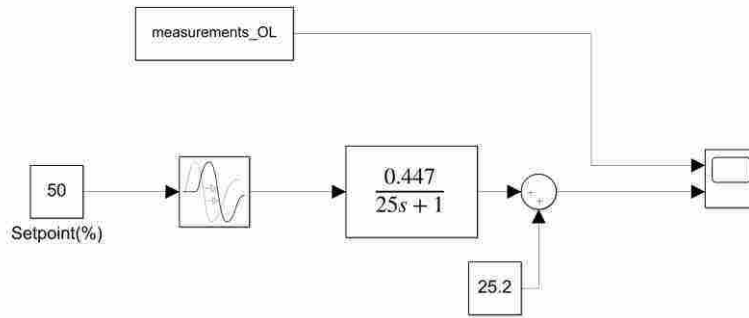
Figure 11: Simulink® diagram for first-order transfer function model.

The second way to derive a linear model is through MATLAB® System Identification Toolbox™. This facilitates the process of finding transfer function models that describe a system. In addition, it makes it much simpler to derive higher order models and to make comparisons between them to see which one is a better fit (MathWorks, Estimating Models). It also allows the merging of many different input-output relationships and estimating other model types such as state space and non-linear models. Below is the procedure of deriving a 4-pole 3-zero transfer function model.

First, the open-loop input and output measurements are imported into the System Identification Toolbox™ as input-output data objects with a sample time of 0.1 seconds. This type of object groups the input duty cycle data with its corresponding output temperature read, as shown in Figure 12 (Ljung 50; ch. 2).

16
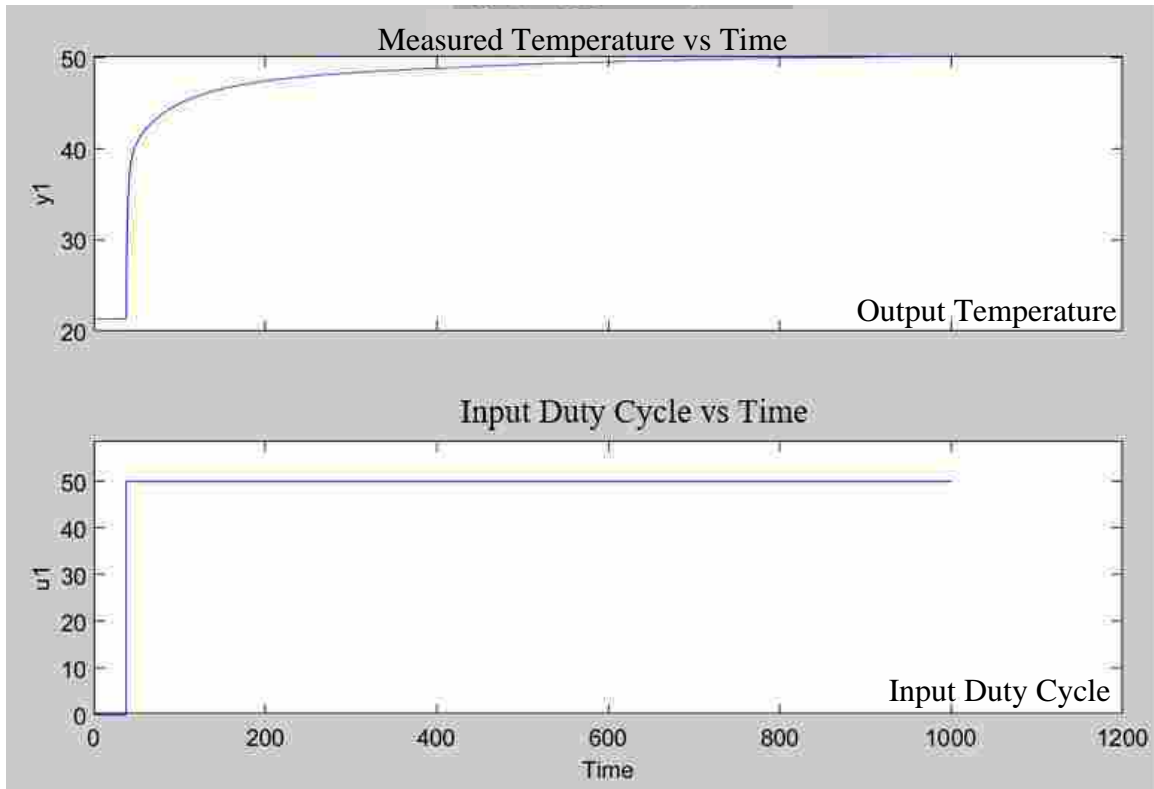
Figure 12: Open-loop Measurements used to derive higher order transfer function model.

After preprocessing the data by removing the starting delay, the toolbox is used to derive transfer function models of different combinations of poles and zeros. The best-fit to the open loop data is a transfer function with 4 poles and 3 zeros. The transfer function model in Equation 5 matches the measured open-loop step response by 85.55%, as shown in Figure 13. Throughout this section, any writing specifying a linear transfer function refers to the one in Equation 5.

$$P(s) = \frac{0.5598\,s^3 + 0.08757\,s^2 + 0.0004332\,s + 1.171e{-}07}{s^4 + 2.336\,s^3 + 0.2098\,s^2 + 0.0006743\,s + 1.998e{-}07} \qquad (\,5\,)$$

17

Table 1 shows the poles and zeros of the generated transfer function. Although the pole -0.00033 and the zero -0.000287 are very close together, eliminating this pair in order to simplify the transfer function to a 3rd order one seems to shift and distort the output response significantly. Both, the step response of the 4th order (green) and 3rd order (yellow) transfer functions are compared with the open-loop measurements (black). The figure shows that the pair of poles and zeros should not be eliminated for producing an accurate linear transfer function model.

Table 1: Poles and zeros of 4th order transfer function.

Poles	Zeros
-2.2431	-0.1513
-0.0901	-0.00482
-0.0030	-0.000287
-0.00033	
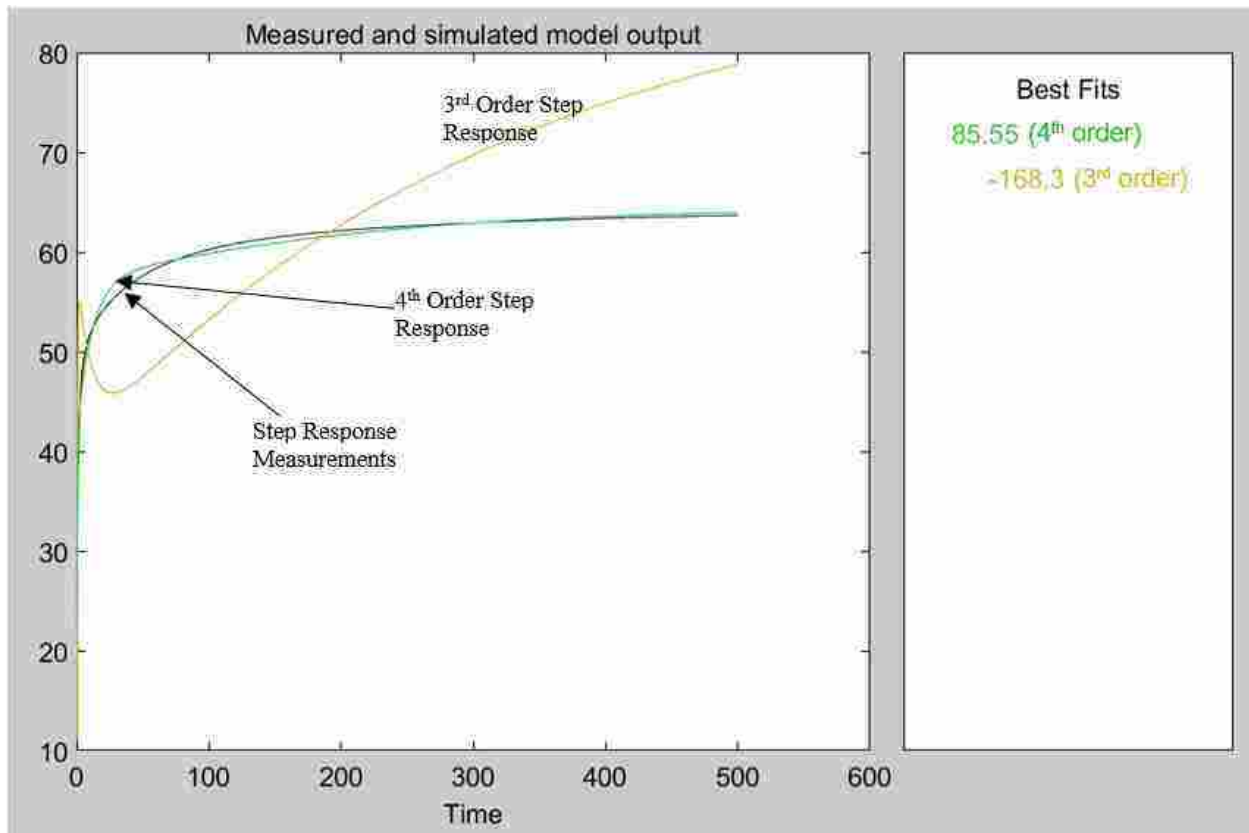
Figure 13: Open-loop step response of linear transfer functions.

Open-Loop Controller Operation and Linear Model

Figure 14 shows the temperature reached in steady-state when the controller is in open-loop configuration for various duty cycles. For example, if the duty cycle is held constant at 25.49%, the steady-state temperature is 40.2°C. This figure suggests that it is possible to make an accurate linear model since the relationship between duty cycle and temperature reached appears to be linear for the measured temperature range.
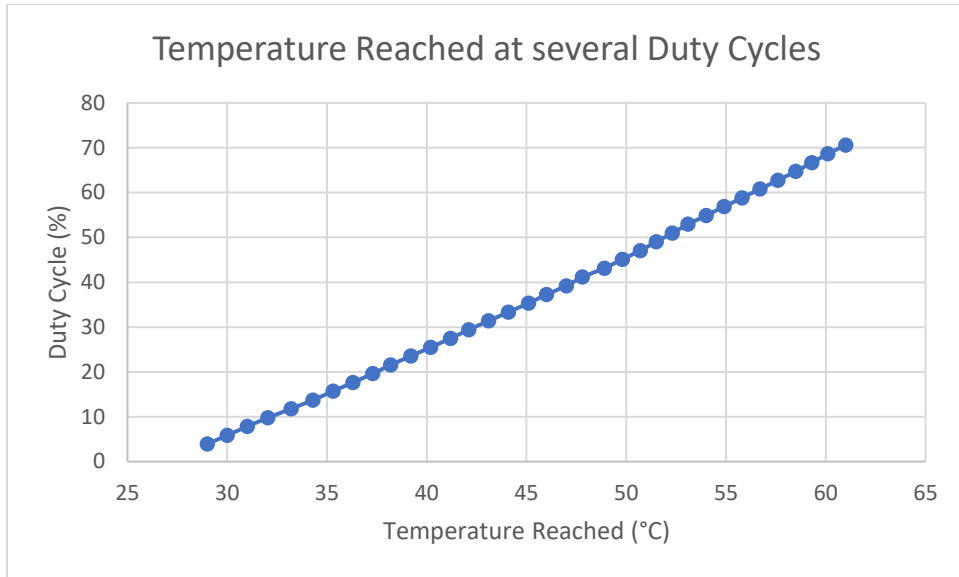
Figure 14: Temperature reached for constant duty cycles.

Figure 15 compares the temperatures in the linear region from the open-loop controller's measurements (yellow) and the 4th order transfer function model (blue) from Equation 5. In this case, the input duty cycle changes from 0% to 50% instantly at a time of 0 seconds. Once the temperature reaches 47°C, both the measurements and the model reach steady state. The model takes longer to increase during the first 50 seconds but reaches steady state quicker. In contrast, the measured value quickly rises in temperature, but stagnates after 50 seconds.
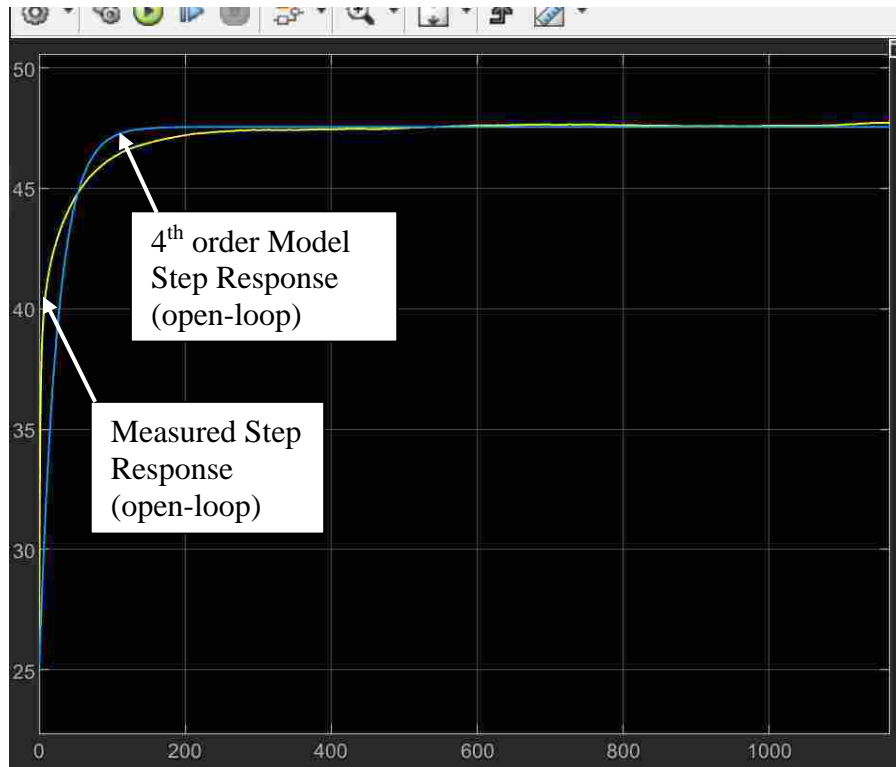
Figure 15: Open-loop step response of linear model and measurements.

Figure 16 uses the same 4th order transfer function to test the system for a varying duty cycle input by plotting temperature in Celsius vs time. The predicted temperature from the model is shown in red and the measurements in black. In this case, the input duty cycle changes every 100 seconds at eight different times. The linear transfer function model is able to predict the output with an accuracy of 92.24%. Figure 16 shows that the model can predict the output accurately using just the linear dynamics of the system. However, the model struggles to predict the temperature after some transitions in input duty cycle.
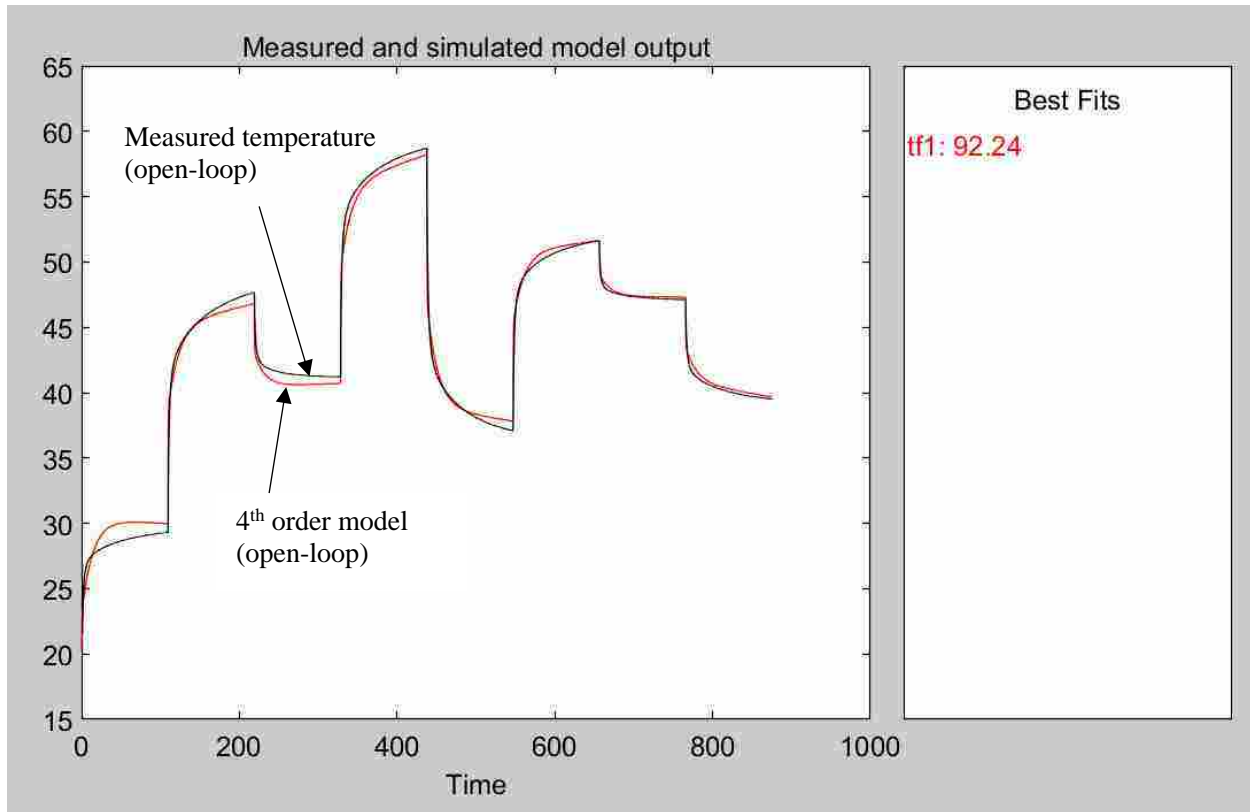
Figure 16: Open-loop response of linear model and measurements.

Closed-Loop Controller Operation and Linear Model

Figure 17 shows the output temperatures of the closed-loop controller (blue) and the same 4th order transfer function model (yellow) from Equation 5. In this case, the input varies as a result of the PID parameters in order to maintain a setpoint of 50°C. When the controller is in closed-operation, it is the model that reacts faster to the step change in temperature setpoint. Both the model and measurement experience a very small amount of overshoot, but the measured plot reaches steady-state quicker than the model.

22
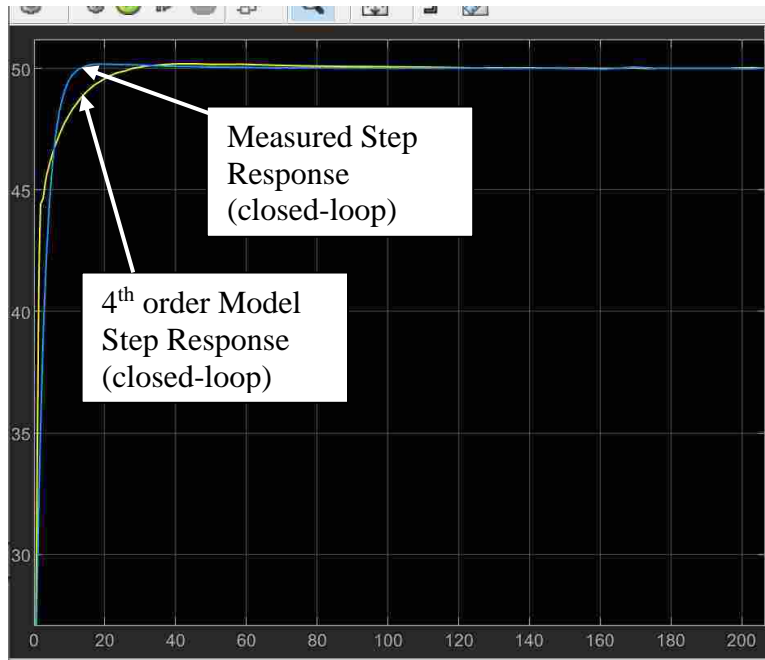
Figure 17: Closed-loop response of linear model and measurements.

Figure 18 shows that the closed-loop response of a linear model with varying duty cycle inputs (red) also matches the measurements (black) well. According to the best-fit percentage, there is a match of 92.49%. Upon visual inspection, the model does not appear to be as accurate as the open-loop response from Figure 16 at the upper and lower temperatures.
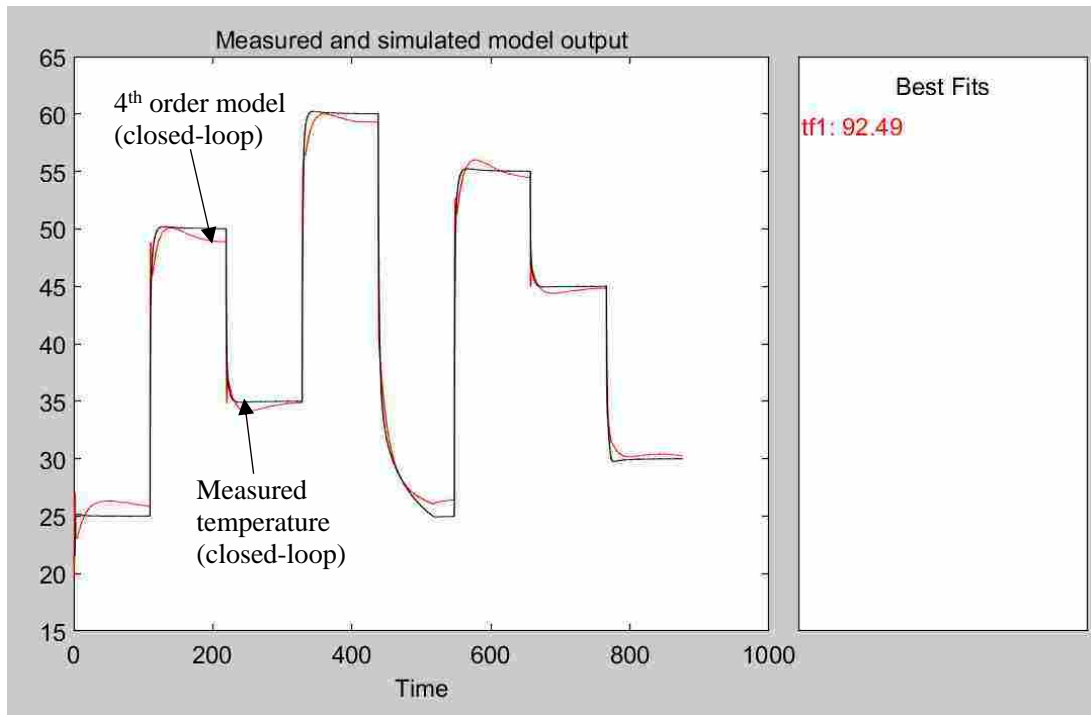
Figure 18: Closed-loop response of linear model and measurements with a varying input.

Non-Linear Model Results

Obtaining Non-Linear Model from Open-Loop Controller

Non-linear models are used when a system's output depends non-linearly on the inputs to the system. One non-linear Hammerstein Wiener non-linear block-structured model consists of one dynamic linear element between two static, or memoryless, non-linear elements connected in series. In this way, the model can represent both, the linear and non-linear aspects of the system (Ljung 13, ch.7). It essentially combines both, the Hammerstein and the Wiener non-linear models into one model (Ljung 13, ch.7). In a Hammerstein model, a static non-linear function is followed by a linear dynamic transfer function, whereas in a Wiener model, the linear dynamic

24

function is followed by a static non-linear function (Ljung 58, ch.7). The block diagram in Figure 19 shows the Hammerstein-Wiener model (Ljung 57, ch.7). It describes how the outputs of the two non-linear blocks, w(k) and y(k), are non-linear functions of their inputs, u(k) and x(k). The non-linear functions of the input and output are f and h, respectively. The linear block is a linear transfer function formed by the ratio of the rational polynomials B(z) and F(z) (Ljung 57, ch.7). This non-linear model was chosen since it is part of the System Identification Toolbox^{TM} (Ljung 13, ch.7).
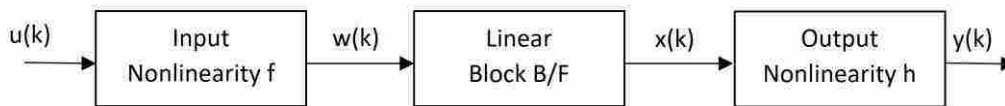
Figure 19. Hammerstein-Wiener diagram (Ljung 57, ch.7).

To represent the linear block, a discrete transfer function is used because the System Identification Toolbox^{TM} does not allow continuous linear transfer functions when generating a non-linear model. The linear block is based on the general-linear polynomial model in Equation 6, where y(k) is the system output, u(k) is the system input, e(k) is the system disturbance, n is the system delay, and A(z), B(z), C(z), D(z), and F(z) are all polynomials that use the backward-shift operator z^{-1}, and Z is the Z-transform variable (National Instruments). The coefficients of these rational polynomials are parameters determined from the provided input and output by using estimation procedures (Ljung 68, ch.4). The first ratio of polynomials represents the measured input-output relationship and the second ratio of polynomials represents the

relationship between the measured output and the disturbance (Ljung 11, ch. 1). Hence, F(z)

determines the poles unique to the system dynamics and D(z) determines the poles unique to the

disturbances (Mathworks, Polynomial).

$$Z\{y(k)\} = \left[\frac{B(z)}{A(z)F(z)}\right]Z\{u(k-n)\} + \left[\frac{C(z)}{A(z)D(z)}\right]Z\{e(k)\} \qquad (6)$$

In practical applications, one or more of these polynomials is fixed to unity depending on

the model structure applied to increase estimation efficiency and flexibility when modeling a

system's dynamics and noise (Ljung 42, ch.4). For example, a Box-Jenkins model uses

polynomials B(z), F(z), C(z), and D(z), whereas the output-error model only uses B(z) and F(z).

The reduced version of the general-linear polynomial model used by the System Identification

Toolbox is the output-error model, meaning A(z), C(z), and D(z) from Equation 6 are all set to

unity. This simplifies the problem by parametrizing the system's dynamics, while ignoring the

parametrization of disturbance (MathWorks, Polynomial). The resulting equation of the output-

error model and its block diagram are shown in Equation 7 and Figure 20 (National Instruments).

The diagram shows how the output y(k) of the model is the input u(k) multiplied with the ratio of

B(z) and F(z) polynomials and added with the disturbance term e(k).

$$Z\{y(k)\} = \left[\frac{B(z)}{A(z)F(z)}\right]Z\{u(k-n)\} + Z\{e(k)\} \qquad (7)$$
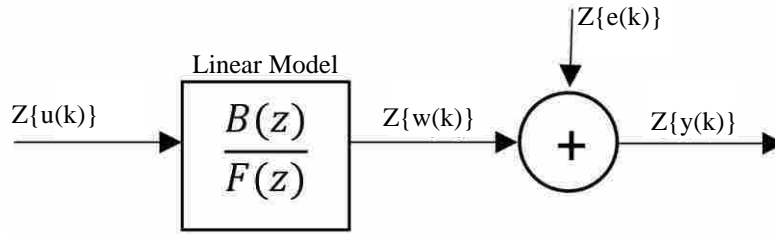
Figure 20: Output-error model diagram (National Instruments).

The estimated B(z) and F(z) polynomials using the System Identification Toolbox^{TM} are shown in Equations 8 and 9 (Ljung 40-48, ch.4).

$$B(z) = -0.001155\ z^{-1} + 0.2171\ z^{-2} - 0.4259\ z^{-3} + 0.21\ z^{-4} \qquad (\ 8\ )$$

$$F(z) = 1 - 1.552\ z^{-1} - 0.1869\ z^{-2} + 1.058\ z^{-3} - 0.3361\ z^{-4} + 0.0171 z^{-5} \qquad (\ 9\ )$$

As shown in Figure 21, the output temperature of this polynomial model (pink) appears to be approaching the real system's measurements (black). However, it is clear it does not fully characterize the system since the static non-linearities of the system, functions f and h, are not modeled. This is the best polynomial model that the System Identification Toolbox^{TM} was able to obtain.
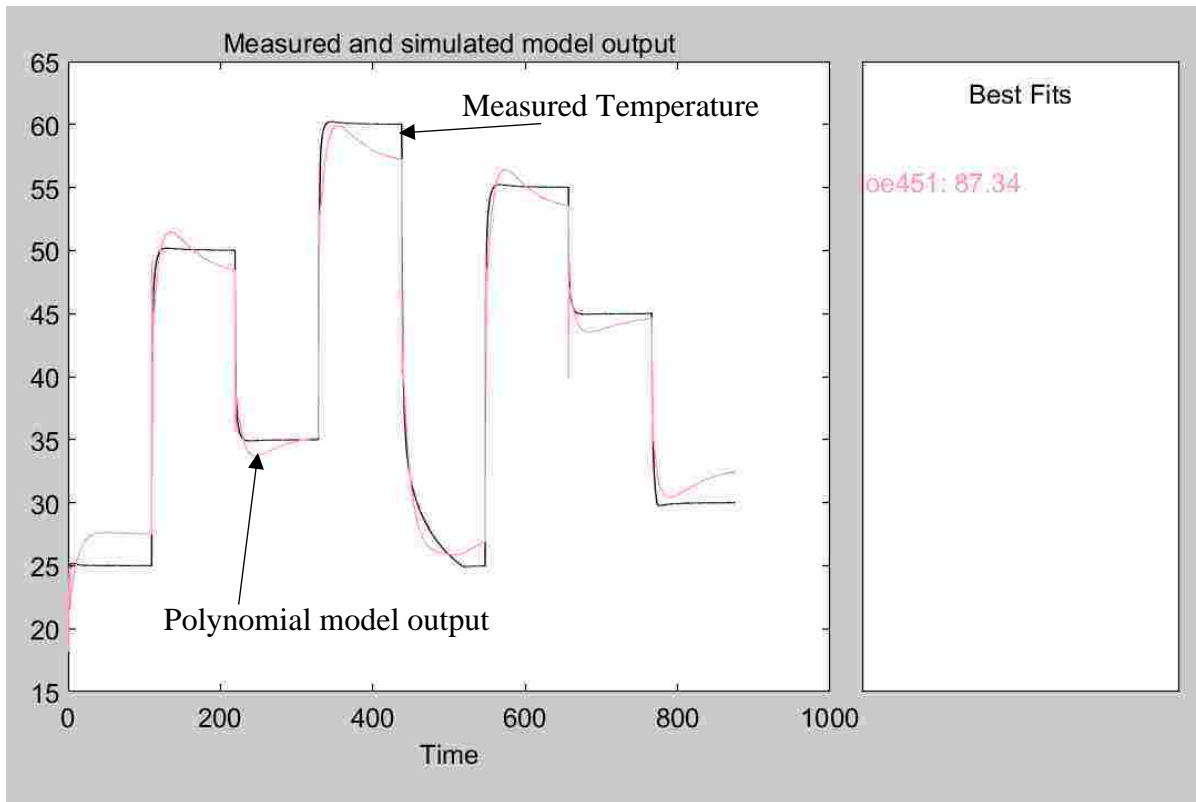
Figure 21: Polynomial model response (closed-loop).

For finding the full Hammerstein-Wiener model, many open-loop measurements are recorded to give the Control System Toolbox a good understanding of how the system behaves. A total of 15 input-output datasets recorded are inputted into the MATLAB® toolbox. However, adding even more data sets can improve the accuracy of the models obtained. Some examples of these input-output datasets are provided in Figure 22. Inputs u1 measured in duty cycle percentage are plotted below their corresponding outputs y1 measured in degrees Celsius. To model the dynamics of the system, inputs include a variety of square waves, triangle waves, and step inputs at different operating points.

28
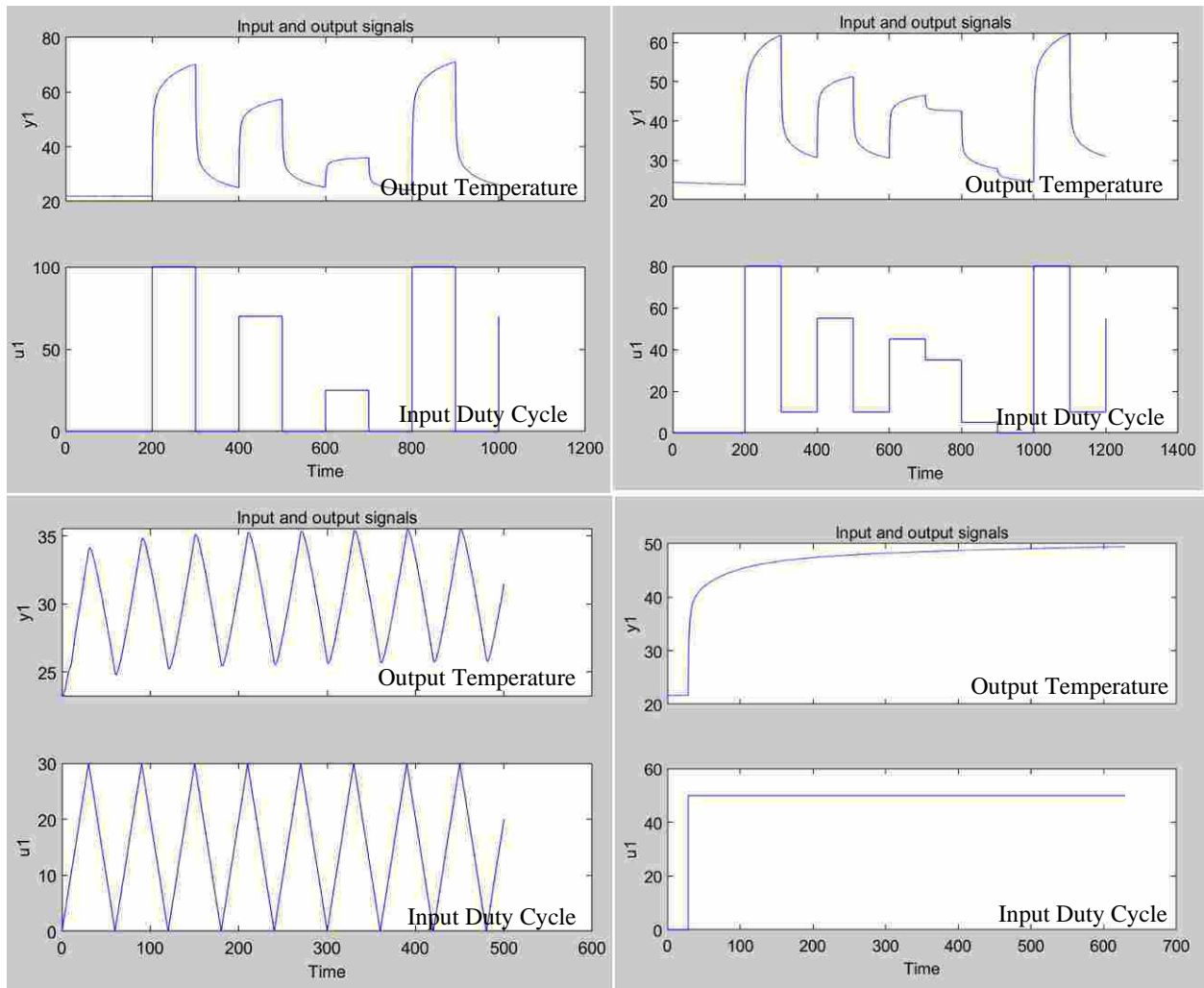
Figure 22: Open-loop measurements taken to derive a non-linear model.

The generated Hammerstein-Wiener model can be checked by using the input duty cycles used to generate the model as inputs into the generated model. The measured temperature in black is compared to the theoretical temperature in red for two different inputs in Figure 23. Both plots show that the model was generated successfully since both theoretical outputs match their corresponding measurements.
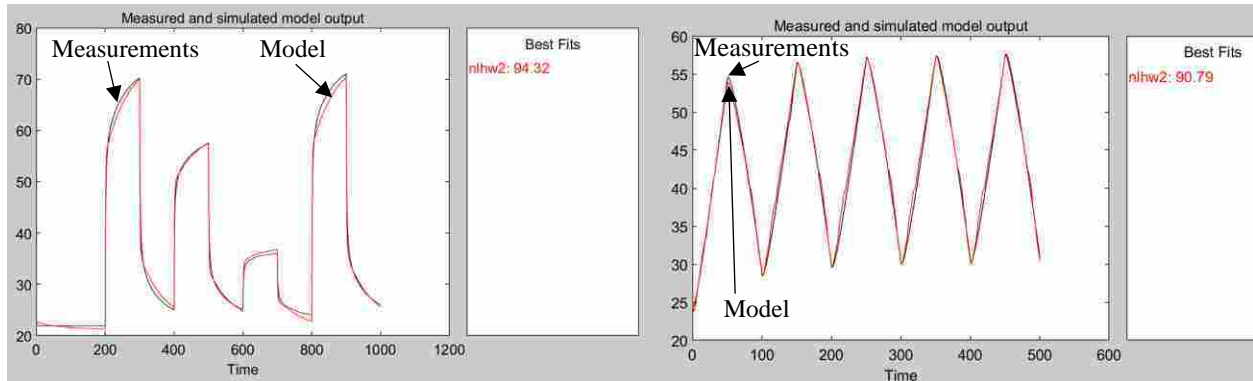
Figure 23: Hammerstein-Wiener model open-loop output response.

This shows a better match compared to Figure 21 because input and output nonlinearities are added to the polynomial model. The derived model (red) has a 94.32% and 90.79% fit compared to some of the measurements (black) used to derive it.

To generate the static non-linearities of the system, the System Identification Toolbox™ generates a linear polynomial model and uses a piecewise linear function parametrized by breakpoint locations to model the system's non-linearities at the input and output of the system (Ljung 59, ch. 7). Basically, the input nonlinear block shown in Figure 19 maps the input nonlinearities as a function of the input data u(k) and the output nonlinear block maps the output nonlinearities as a function of the linear block's output, x(k) (Ljung 57, ch 7). Both f and h non-linear functions are static with no memory, meaning their outputs only depend on the input at a particular time (Ljung 58, ch 7).

Figures 24 and 25 show the input and output data for both non-linear blocks in Figure 19 when the toolbox generates a Hammerstein-Wiener model directly from the input-output data sets. Both plots show a linear trend, suggesting that the system can be approximated using a linear transfer function.

30
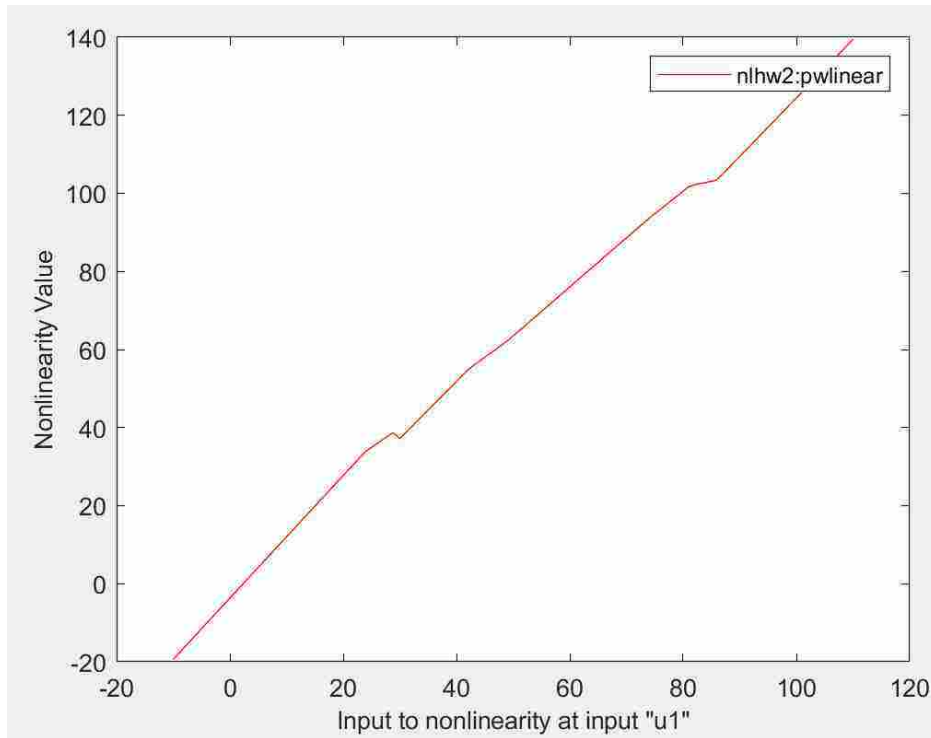
Figure 24: Hammerstein-Wiener input non-linearities.
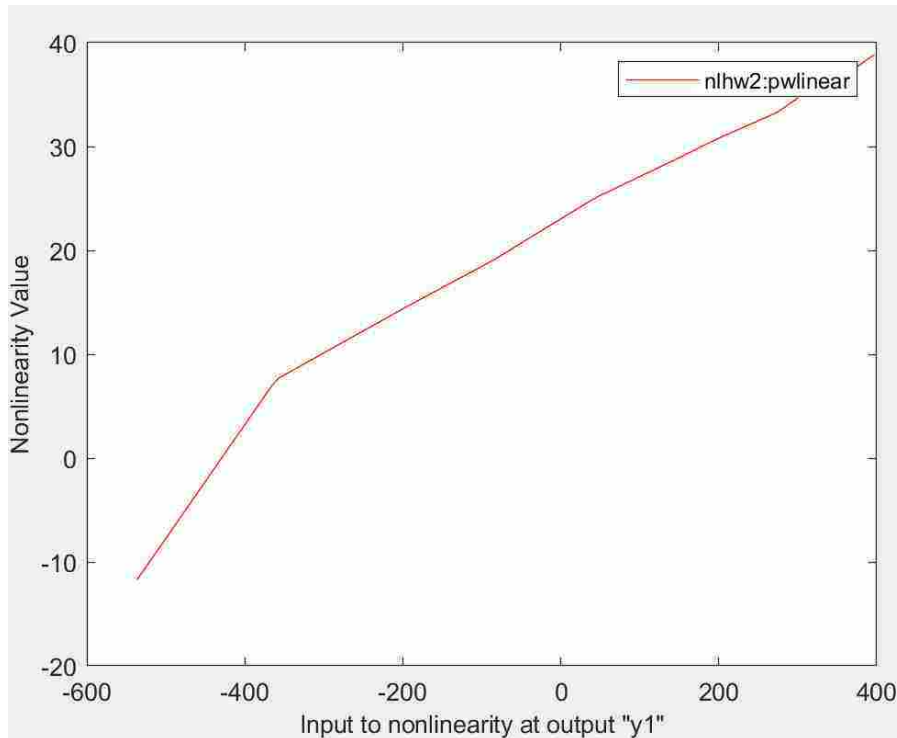
Figure 25: Hammerstein-Wiener output nonlinearities.

The MATLAB® System Identification Toolbox™ also offers the ability to implement the

Hammerstein-Wiener model completely and learn both input and output non-linearities as well

as the polynomial model, which is different than that of Equation 8 and 9. Although it is possible

to create a Hammerstein-Wiener model from a specific linear model using non-linearity

estimators created in the command line, MATLAB® does a better job in terms of modelling the

complete Hammerstein-Wiener model (Ljung 64-86, ch.7). Figure 26 compares the output

responses of all the elements of the Hammerstein-Wiener model generated directly from the

toolbox with closed-loop measurements. The method used in Figure 26 for removing input

and/or output non-linearities in order to produce a Hammerstein, Wiener or polynomial model is

specified in the toolbox's user guide (Ljung 79, ch. 7). The bad fit between measurements and

the new polynomial model extracted from the Hammerstein-Weiner model indicates that the

process the toolbox uses to generate the polynomial model during this procedure is not the same

as the process used to obtain the polynomial model in Equations 8 and 9. This is evident in

Figure 26, where the polynomial model from Equations 8 and 9 has a fit of 87.34%, while the

polynomial model used in the full Hammerstein-Wiener model implemented by the MATLAB®

toolbox has a poor fit of -329.9%. However, when combined with input and output non-

linearities, the model reaches a fit of 91.43%. The non-linear results presented in this thesis are

from the final Hammerstein-Wiener model, which reaches a fit of 91.43% in Figure 26.
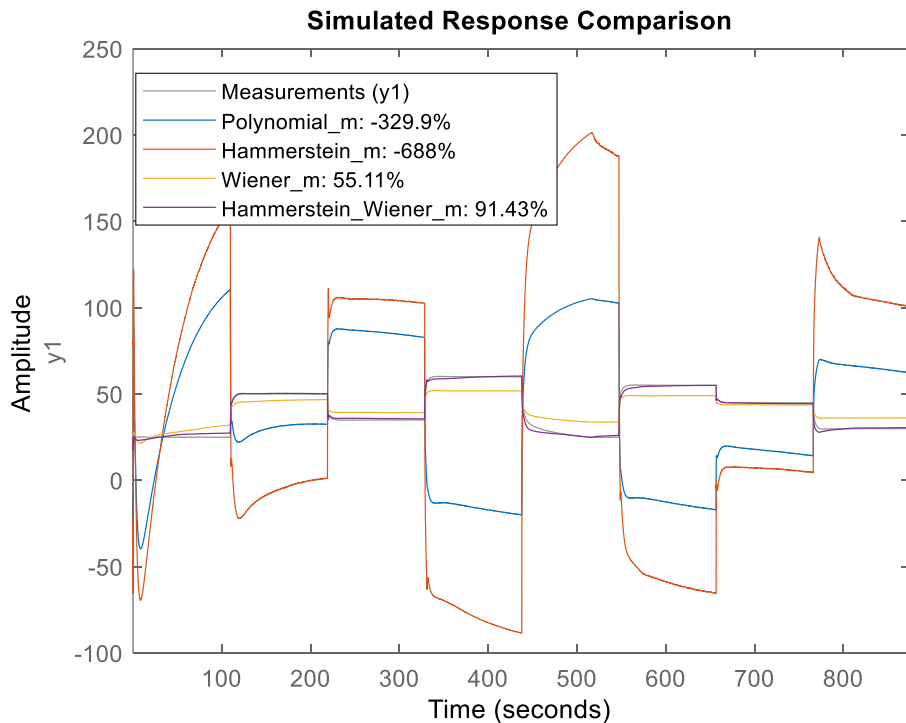
Figure 26: MATLAB® process for generating Hammerstein-Wiener Model

Open-Loop Controller Operation and Non-Linear Model

The plant's non-linearity is tested through the test shown in Figure 27. A function generator is used to generate a very low frequency (10 mHz) voltage as an input to the heater, instead of using the controller's output PWM signal. The results suggest that the system is not purely linear because the output temperature does not increase in a perfect linear fashion when an input voltage is applied. The output response is also not symmetrical since the output does not decrease at the same rate it increases, even though the increasing and decreasing portions of the input have the same slope magnitude.
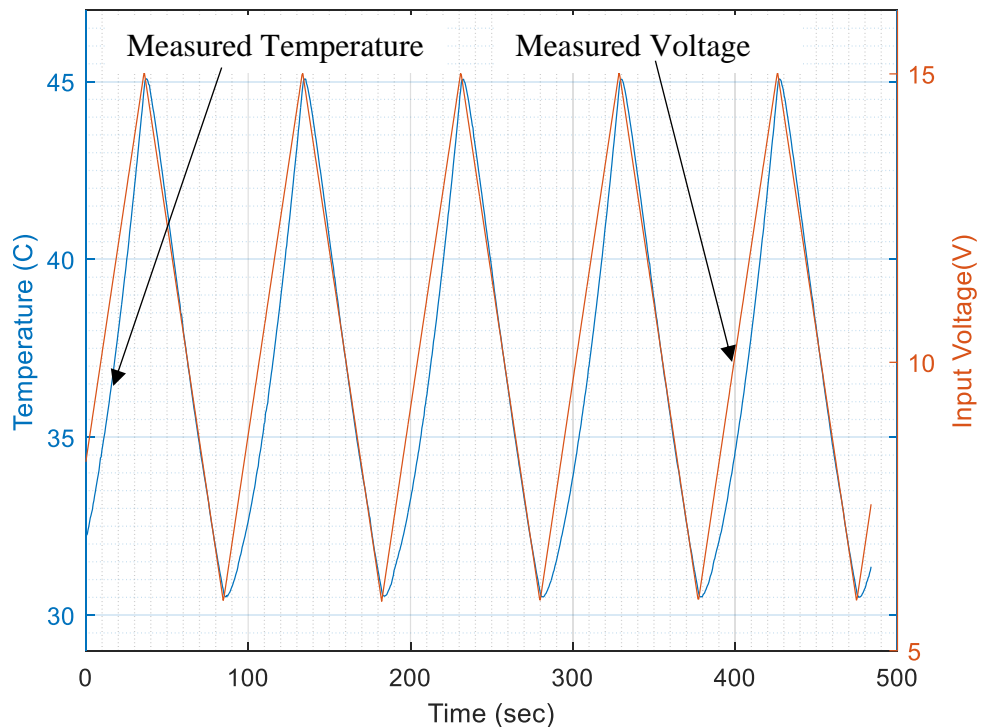
Figure 27: Measured Temperature when triangular wave is used to power heater.

In figure 28, the Hammerstein-Wiener non-linear model output is compared against the measured open-loop output response. The open-loop temperature plot for non-linear model is shown in green and open-loop measurements in black. Similar to the linear model, there is a good match when the setpoint duty cycle changes. There is also a better prediction of the output temperatures after the duty cycle changes. This suggests that the system is primarily a linear system with some non-linearities as it approaches steady-state.
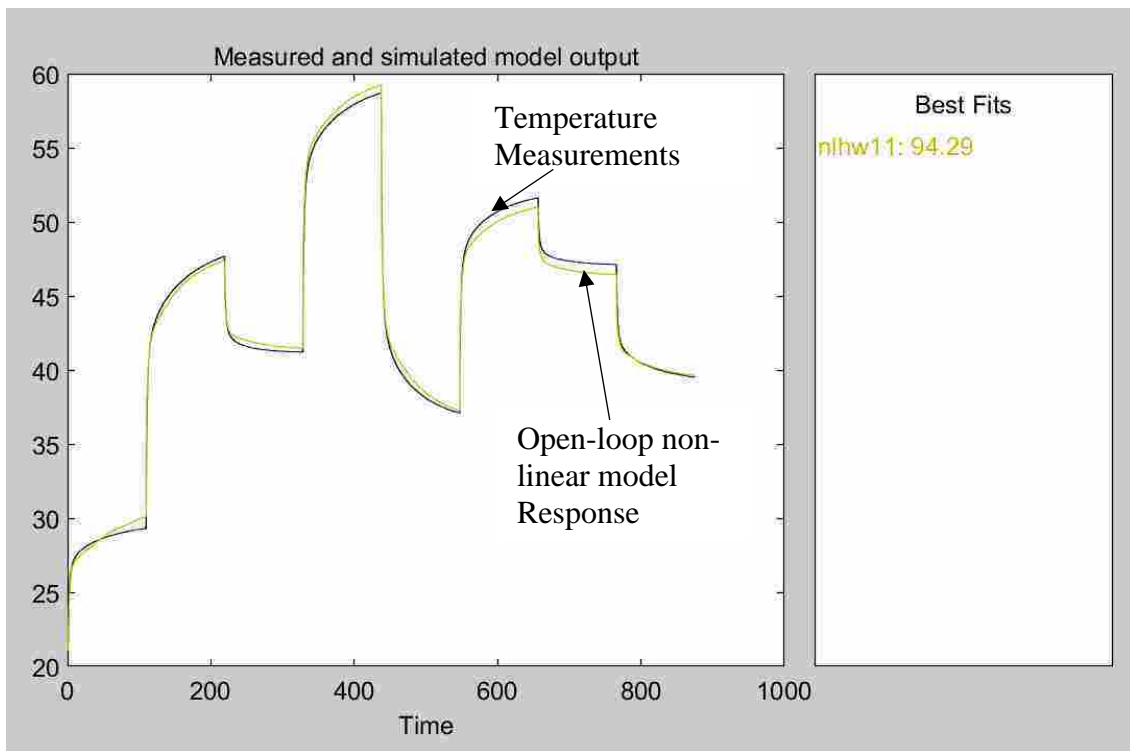
Figure 28: Open-loop temperature for non-linear model.

Closed-Loop Controller Operation and Non-Linear Model

The closed-loop temperature plot for non-linear Hammerstein-Wiener model (green) and measurements (black) are shown in Figure 29. There is also a good match when the setpoint duty cycle changes. The fit of this plot shows that the 4th order transfer function model and the non-linear model results are approximately equal for the closed-loop controller. The only possible conclusions from this study is that the polynomial model did not perform as well as the transfer function model, but a Hammerstein-Wiener model was equivalent or better to the transfer function model.
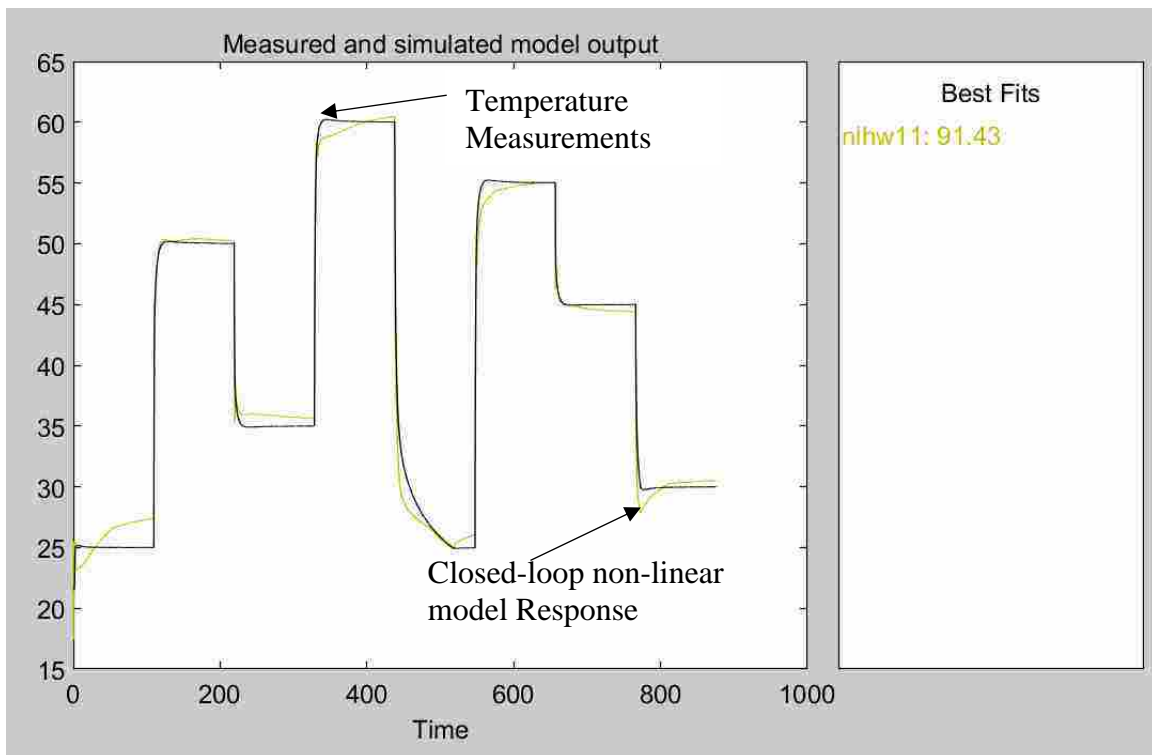
Figure 29: Closed-loop temperature for non-linear model.

CHAPTER 4: EXPERIMENT SETUP

External Connections to SAW Device and Oven Setup

The experiment block diagram in Figure 30 shows connections required for the experiment setup. The diagram shows all connections using arrows and the colored blocks are sub-sections of either the SAW device or the temperature controller. Other connections required are the VNA connections for measuring S21, the connection to a computer via USB for reading data and providing 5 volts, and the connections from the power supply to the circuit used for heating. Further details about all the necessary connections are mentioned in this section.
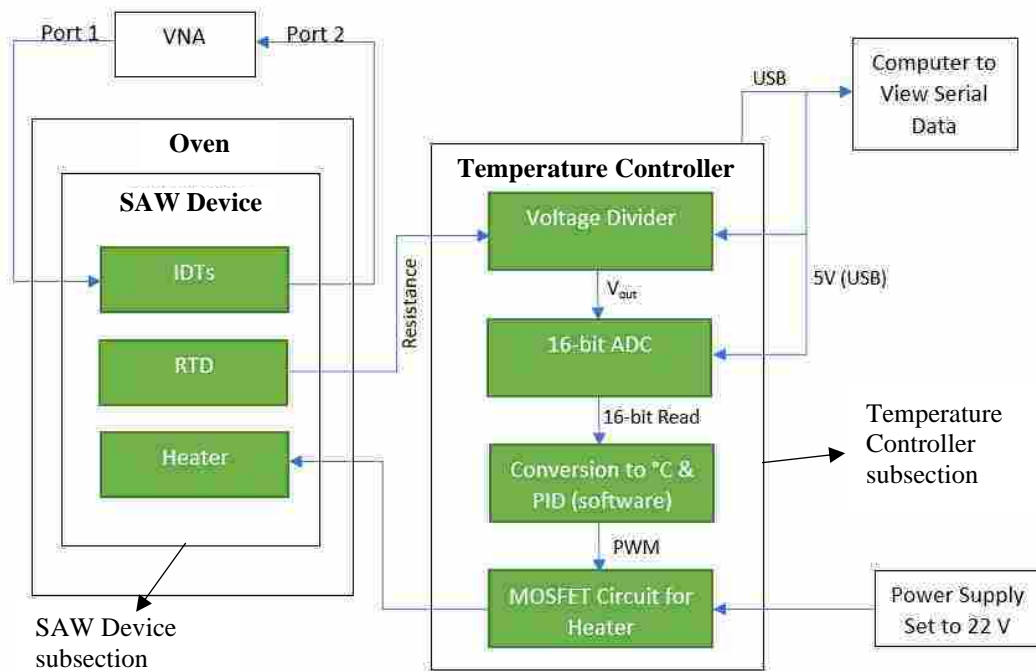
Figure 30. Experiment Block Diagram.

A basic setup of the BK Precision 1666 Bench Switching Power Supply above the Sun Systems temperature chamber is shown in Figure 31 (BK Precision and Sun Systems). These are shown along with the HP 8753ES S-Parameter Network Analyzer used (Keysight).
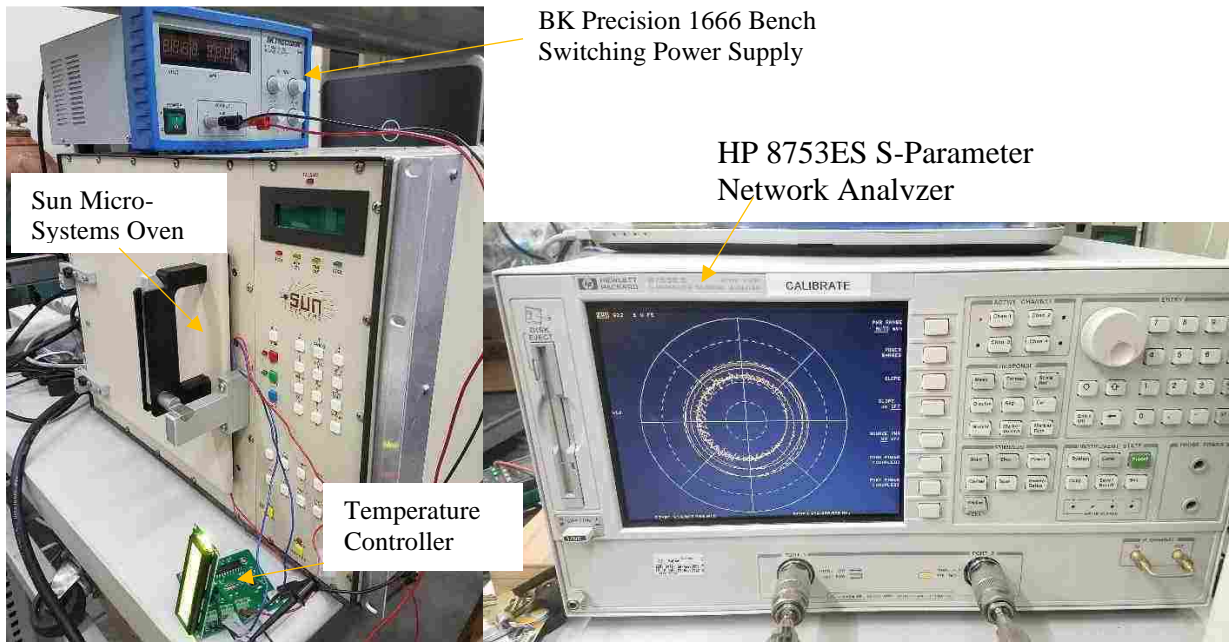
Figure 31: Experiment setup and equipment.

A second PCB is needed to connect the pins of the SAW package to the controller. A FR4 board is etched using hydrochloric acid to form the four traces needed for these connections. The packaged device is soldered on top of this board, as seen in Figure 32. The on-device heater and RTD sensor are both wired to the controller using 22-gauge wire. The on-wafer heater is connected in series between the positive terminal of a power supply set to 22 V and the drain of the N-channel MOSFET. The RTD is connected in series with a resistor of known value (1 kΩ) to form a voltage divider with an output voltage proportional to temperature.
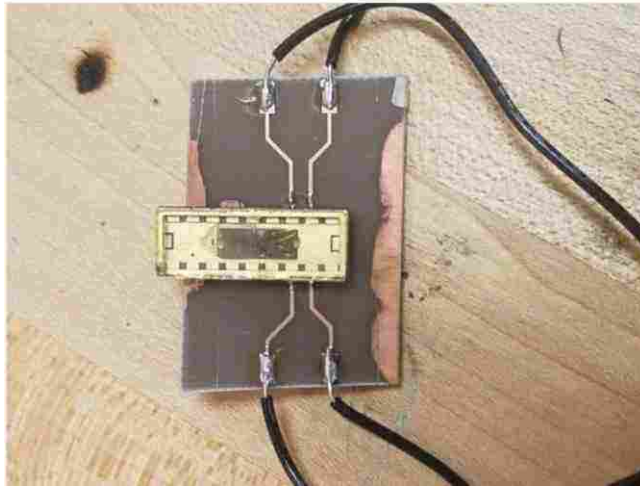
Figure 32: SAW device PCB traces.

Placing the device inside the oven, as seen in Figure 33, allows for testing the controller's ability to maintain the SAW device's temperature stable under different ambient temperatures. The thermocouple regulating the temperature of the oven is placed close to the SAW device to ensure that oven temperature is similar to the temperature experienced by the device. The packaged SAW correlator is connected to the Vector Network Analyzer (VNA) using coaxial cables that enter the oven through a sealed opening on the side. These are soldered directly to the package pins bonded to the input and output transducers.
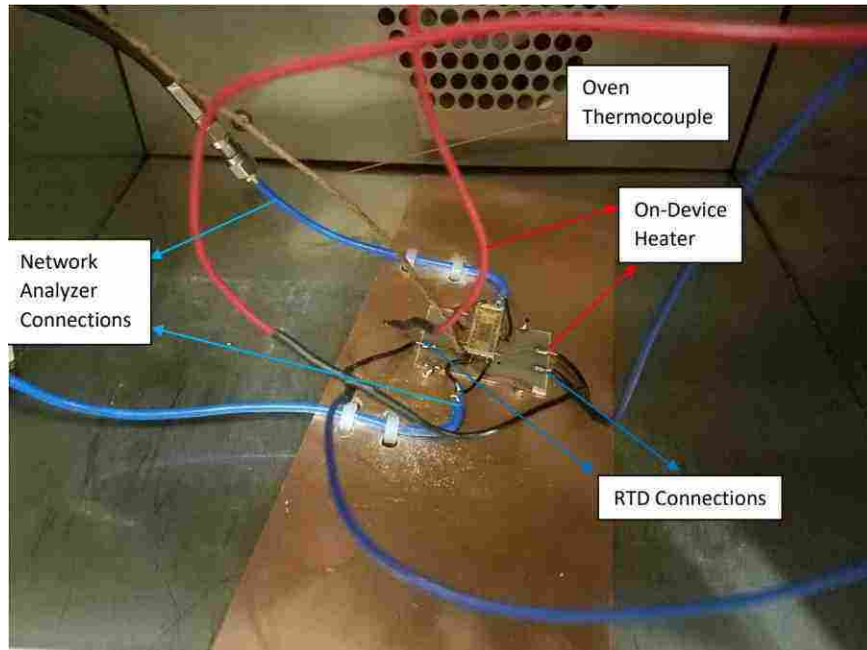
Figure 33: SAW device external connections.

A 2-port calibration is performed at frequencies 920 to 970 MHz using SOLT calibration standards connected to the SMA cables. The input power is set to 10 dBm with a sweep time of 1 second and 11207 points. While running VNA measurements, the oven is turned off to avoid distortions in the S21 response. The power supply, controller, and VNA are all placed outside of the oven.

Relating Resistance to Temperature

Connecting an RTD with a resistor in series creates a voltage divider with a temperature dependent output voltage. Hence, by applying a known input voltage to the RTD and measuring the output voltage at the node between both components, it is possible to obtain the RTD's resistance. Using the voltage divider equation in Equation 10 is the simplest way of measuring

40

resistance using an Arduino® because it can only read voltage. V_{in} is the input voltage, V_{out} is the

output voltage, R_{12} is a 1kΩ resistor, and R_{RTD} (from Figure 5) is the current resistance of the

RTD.

$$V_{out} = V_{in} * (\frac{R_{12}}{R_{12}+R_{RTD}}) \tag{10}$$

$$V_{out} = 5 * (\frac{1000}{1000+R_{RTD}}) \tag{11}$$

$$(1000 + R_{RTD}) = 5 * (\frac{1000}{V_{out}}) \tag{12}$$

$$R_{RTD} = 5 * \left(\frac{1000}{V_{out}}\right) - 1000 \tag{13}$$

In order to relate resistance readings to temperature, the device is placed inside the oven and a

multimeter connected in parallel to the on-wafer RTD. Resistances are measured as oven

temperature is increased from room temperature to 70°C. These results are plotted in Figure 34.
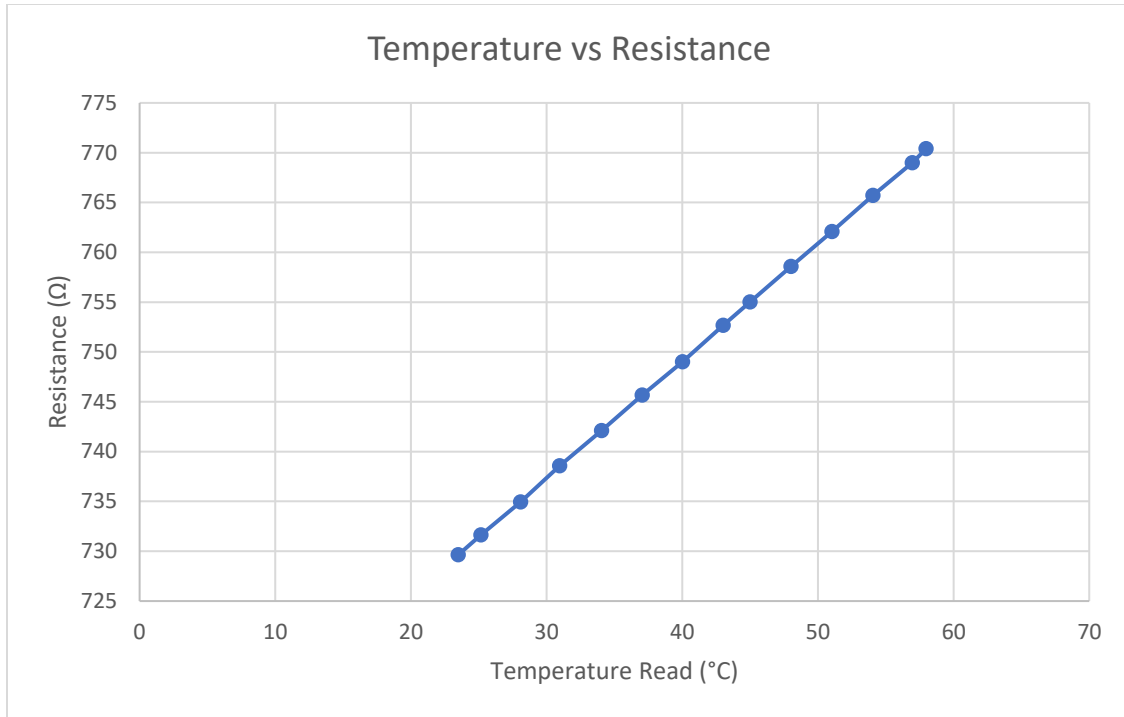
Figure 34: Resistance of SAW device inside oven at different oven temperatures.

Figure 34 shows that there exists a linear relationship between RTD resistance and ambient temperature. This data was also measured using a second thermocouple to ensure accuracy. A linear equation of the line is obtained using Excel's trend-line feature and solved for temperature. The resulting formula, Equation 14, shows how the RTD's resistance R_{RTD} can be converted to a temperature reading in Celsius.

$$Temperature \; (°C) = \frac{R_{RTD} - 705.51}{1.1481} \qquad\qquad (14)$$

The disadvantage of this method is that the Arduino® 10-bit ADC limits the accuracy of the output voltage read because it only has the ability to detect 10-bits or 1,024 (2^{10}) discrete analog

levels. Each time the ADC senses a change of one bit or 4.9 mV, the measured temperature

changes by 2.5°C. This is demonstrated by Table 2. The output of the voltage divider responsible

for measuring RTD resistance is measured by the Arduino® built-in ADC and converted to

temperature. This ADC is incapable of reading voltages in between those listed in Table 2.

Voltage values are the result of one sample reading, not the average of many samples.

Table 2: 10-bit ADC temperature reading accuracy (1-bit change).

Voltage Read (V)	Temperature (°C)
2.8662	28.2221
2.8613	30.6999
2.8564	33.1862
2.8516	35.6809
2.8467	38.1843
2.8418	40.6962
2.8369	43.2167
2.8320	45.7460
2.8271	48.2840
2.8223	50.8308
2.8125	55.9509
2.8076	58.5243
2.8027	61.1067

Using a 16-bit ADC module allows the ADC to read voltage changes smaller than 1 mV and temperature changes lower than 0.08°C. The output voltage of the same voltage divider is again responsible for measuring RTD resistance. This time, voltage is read by the ADS 1115 ADC and sent to the Arduino® processor.

Using a 1kΩ resistor as R_{12} from (Figure 5) limits the voltage read to 2.7-2.9 V when the input voltage is 5 volts for the temperatures under test. This meets the ADS 1115 requirement that the input voltage should be no greater than 4.096 volts (Earl). The output voltage values listed in Table 3 are the result of one sample reading, not the average of many samples. By taking the average of several readings, it is possible to further increase the precision of the readings. For this experiment, the average of 10 samples was used.

Table 3: 16-bit ADC temperature reading accuracy (1-bit change).

Voltage Read	Temperature (°C)
2.876375	31.93242263
2.87649993	31.86847686
2.87662506	31.8044281
2.87674999	31.74058914
2.87687492	31.67669296

Tuning the Controller

Tuning the controller to achieve a desired output response is essential when it comes to implementing a PID controller in a closed-loop control system. Without proper tuning, the

actuator will provide too little or too much power when there is error between the setpoint and

the measured SAW correlator temperature. Figure 35 shows how the temperature and duty cycle

change for the temperature controller PCB when it is not tuned correctly.

Untuned Closed-Loop Plots of Duty-Cycle and Temperature Measurements vs Time
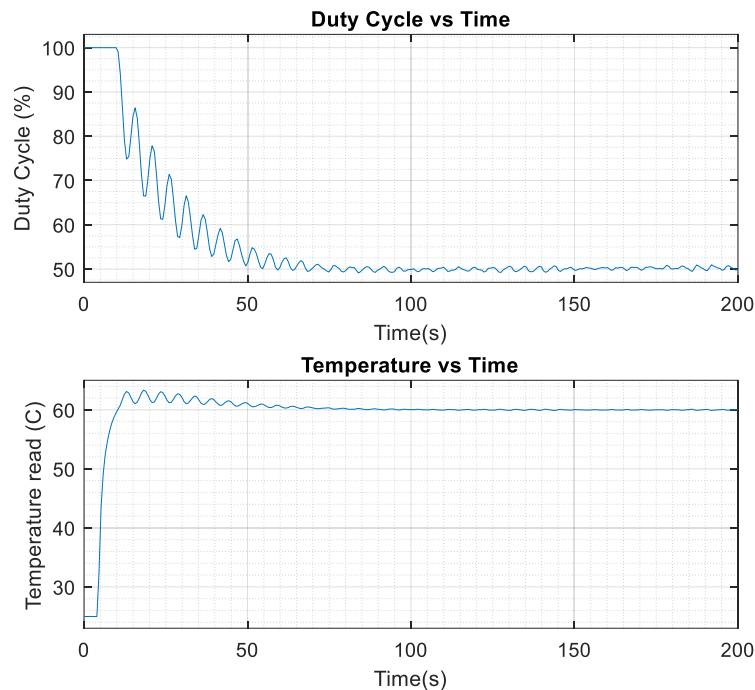
Figure 35: Untuned controller output response (SAW temperature).

First, the Ziegler-Nichols tuning method was attempted following the process in *PID*

Control by Finn Haugen (Haugen 94). This requires incrementing the proportional term until

sustained oscillation is achieved. From this, the period of oscillation and the proportional term at

which sustained oscillation is achieved are plugged into formulas to solve for the P, I, and D

terms (Haugen 94). The root locus plot of the 4th order transfer function used to model the plant

in Equation 15 is shown in Figure 36. The root locus reveals that the transfer function is stable

since all the poles are in the left-hand axis of the root locus plot and it is overdamped since all its

poles are in the real axis. Hence, sustained oscillations or neutral stability are impossible to

achieve since the root locus does not cross the imaginary axis (Cheever).

$$P(s) = \frac{0.5598\, s^3 + 0.08757\, s^2 + 0.0004332\, s + 1.171e{-}07}{s^4 + 2.336\, s^3 + 0.2098\, s^2 + 0.0006743\, s + 1.998e{-}07} \qquad (\ 15\ )$$
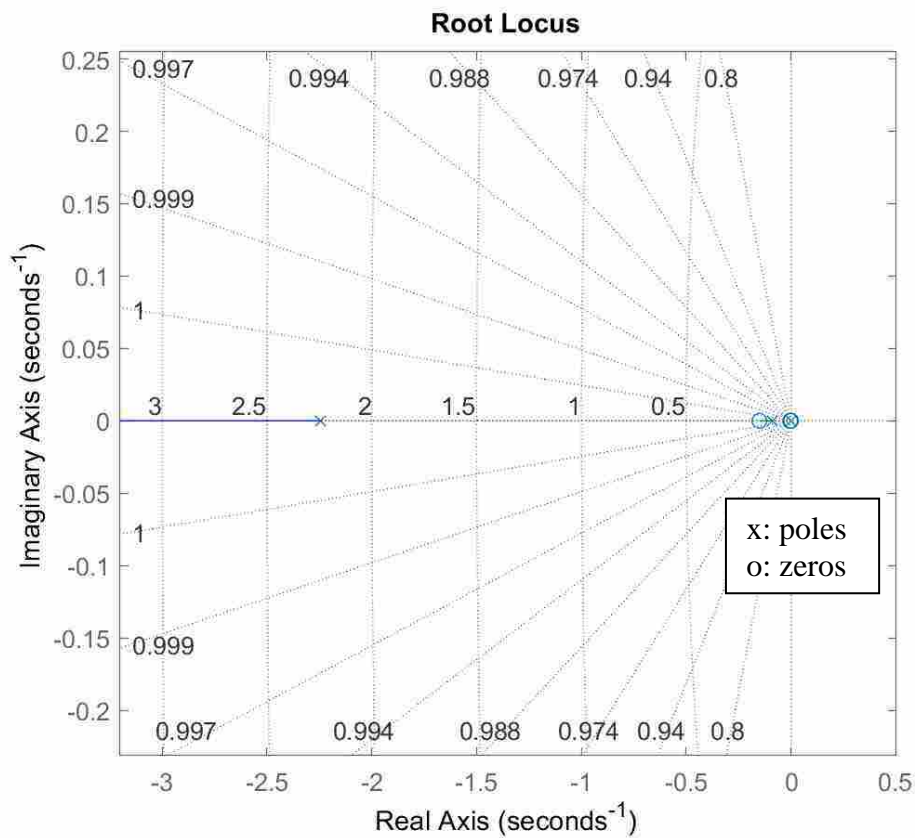
Figure 36: Root locus of 4th order transfer function.

Instead of using the Ziegler-Nichols method, the PID parameters are tuned empirically using the simulated step response of the 4th order linear transfer function model in Equation 15. The step response for gains K_p= 1, K_i=0, and K_d= 0 is shown in Figure 37, along with very conservative constraints of a rise time lower than 5 seconds, a settling time of less than 10 seconds, and a maximum overshoot of 10%. Only the plant and controller are modeled during this simulation since the sensor is assumed to be ideal.
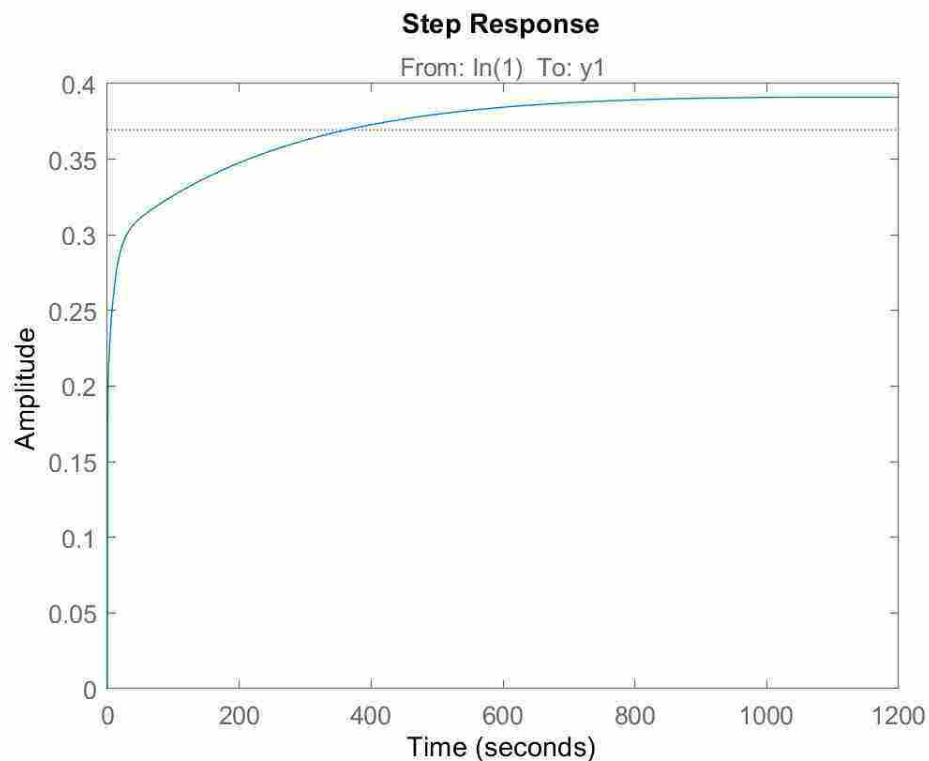
Figure 37: Step response for uncompensated linear transfer function.

Then, the gains are adjusted by using MATLAB$^{®}$ PID Tuner, which is part of the Control System Toolbox^{TM} (MathWorks, PID Tuner). By importing a 4th order linear transfer function

47

model of the plant, it is possible to plot the response time and transient behavior of the output

response. Changing the response time changes how fast the output response reacts to a step input

change, whereas changing the transient behavior adjusts the aggressiveness of the controller

when it senses disturbances in the output response (MathWorks, PID Tuner). After toggling

these two options, the output response achieves a rise time of 0.766 seconds and a settling time

of 3.17 seconds with very minimal overshoot of just 0.456%, as seen in Figure 38. The PID

parameters obtained are K_p=18, K_i=10.1, and K_d=0.02. These correspond to an integral time of

1.78 seconds and a derivative time of 0.001 seconds.
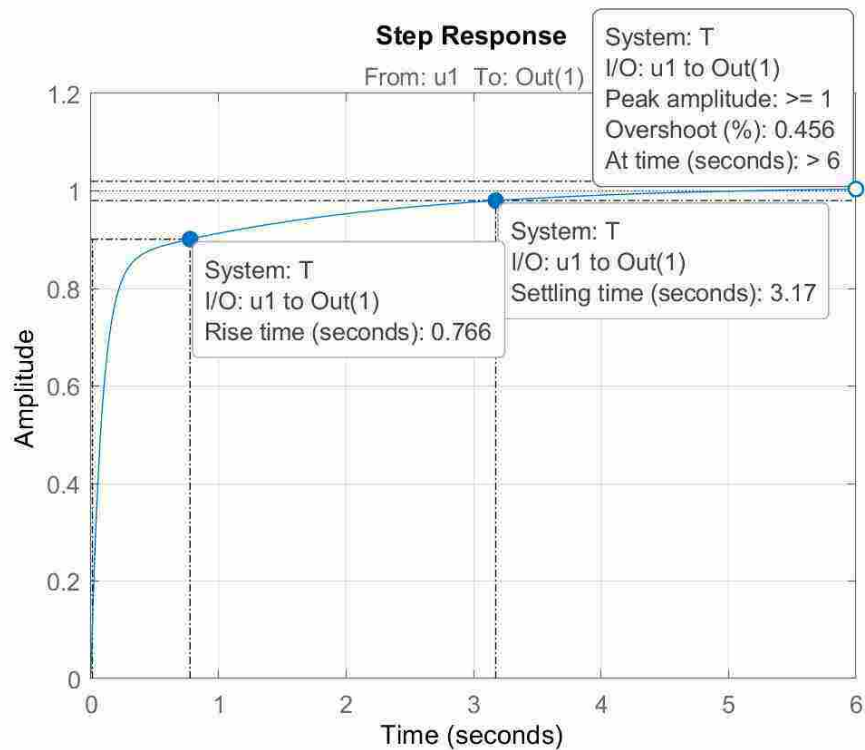
Figure 38: Step response for compensated transfer function.

The transfer function of the closed-loop system can be obtained by first deriving the transfer function of the closed-loop system diagram shown in Figure 39. Then, transfer functions for each block are substituted into this equation. The process for deriving the closed-loop system is shown below (Messner, PID Controller).
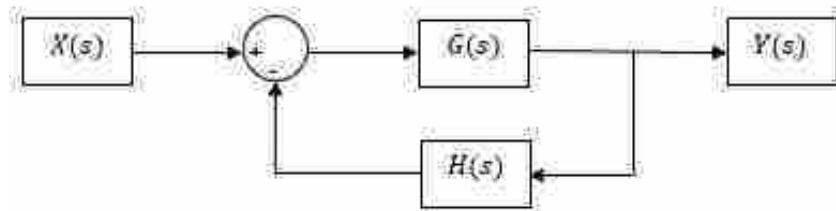
Figure 39. Closed-loop block diagram.

X(s) and Y(s) are the system's input and output, respectively. H(s) represents the gain of the feedback path and G(s) is the open-loop gain. Equations 16 through 19 show the derivation of this closed-loop system. This is done by first setting the output Y(s) equal to the error multiplied by the gain. This equation is then solved for the ratio of the output Y(s) to the input X(s) to form the transfer function of the system (Gajic).

$$Y(s) = G(s) * \left(X(s) - \left(H(s) * Y(s)\right)\right) \tag{16}$$

$$Y(s) = \left(G(s) * X(s)\right) - \left(G(s) * H(s) * Y(s)\right) \tag{17}$$

$$Y(s) * \left(1 + \left(G(s) * H(s)\right)\right) = \left(G(s) * X(s)\right) \tag{18}$$

$$\frac{Y(s)}{X(s)} = \frac{G(s)}{1+\big(G(s)*H(s)\big)} \qquad\qquad (19)$$

In the temperature controller, G(s) is the multiplication of the PID controller transfer function, C(s), with the plant transfer function, P(s). The transfer functions C(s), P(s) and their multiplication are shown in Equations 20 through 23.

$$C(s) = (K_p + K_i\frac{1}{s} + K_d * s) \qquad\qquad (20)$$

$$P(s) = (\frac{0.5598\,s^3 + 0.08757\,s^2 + 0.0004332\,s + 1.171e-07}{s^4 + 2.336\,s^3 + 0.2098\,s^2 + 0.0006743\,s + 1.998e-07}) \qquad\qquad (21)$$

$$C(s) * P(s) = (\frac{K_d*s^2+K_p*s+K_i}{s}) * (\frac{0.5598\,s^3 + 0.08757\,s^2 + 0.0004332\,s + 1.171e-07}{s^4 + 2.336\,s^3 + 0.2098\,s^2 + 0.0006743\,s + 1.998e-07}) \qquad\qquad (22)$$

$$C(s) * P(s) = (\frac{0.0112\,s^5 + 10.08\,s^4 + 7.231s^3 + 0.8923\,s^2 + 0.004377\,s + 1.183e-06}{s^5 + 2.336\,s^4 + 0.2098\,s^3 + 0.0006743\,s^2 + 01.998e-07\,s}) \qquad\qquad (23)$$

The sensor of the system is assumed to be ideal, meaning that the output and input of the feedback path block are equal, resulting in a feedback path transfer function of H(s)=1. Comparing the derived closed-loop transfer function from Equation 19 and substituting the appropriate transfer functions, the closed-loop transfer function of the temperature controller becomes Equation 24. This is solved using MATLAB® and the resulting transfer function is shown in Equation 25.

$$\frac{Y(s)}{X(s)} = \frac{C(s)*P(s)}{1+H(s)*C(s)*P(s)} \qquad\qquad (24)$$

$$\frac{Y(s)}{X(s)} = \left(\frac{0.0112\,s^5 + 10.08\,s^4 + 7.231 s^3 + 0.8923\,s^2 + 0.004377\,s + 1.183e{-}06}{1.011\,s^5 + 12.42\,s^4 + 7.44\,s^3 + 0.893\,s^2 + 0.004378\,s + 1.183e{-}06}\right)$$ (25)

Similar to the plant model, the closed-loop system for the temperature controller is stable and overdamped since its root locus never crosses the imaginary axis and all the poles are real, as seen in Figure 40 (Cheever). The plot's poles and zeros are shown in Table 4.

Table 4: Poles and zeros for closed-loop transfer function.

Poles	Zeros
-11.6526	-899.4385
-0.4631	-0.5615
-0.1567	-0.1513
-0.0048	-0.0048
-0.0003	-0.0003
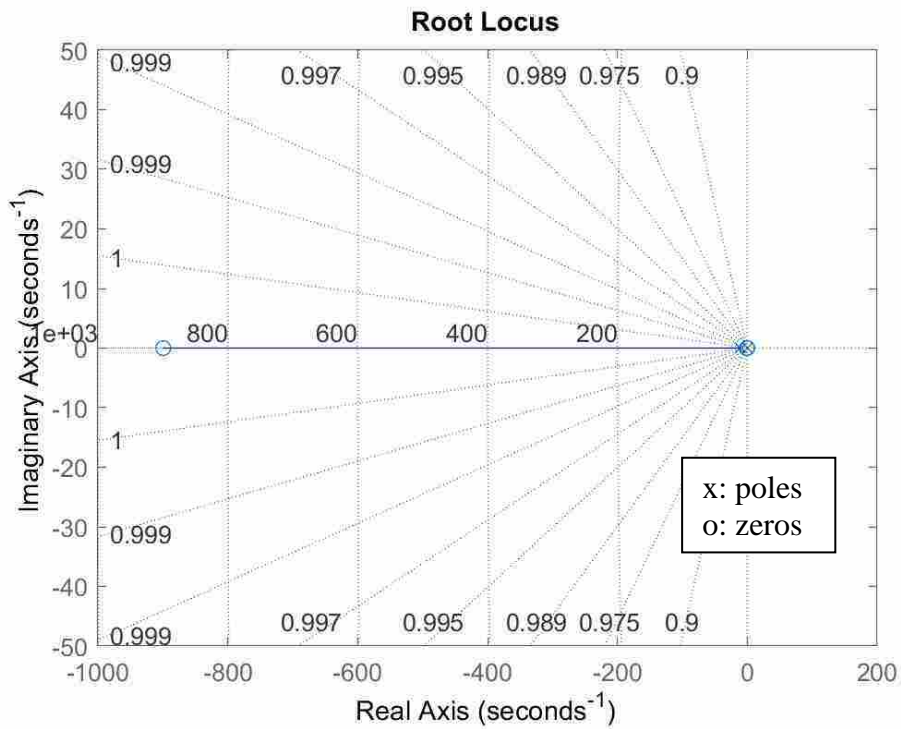
Figure 40: Root locus for compensated transfer function.

The same PID terms found are used in the temperature controller PCB to ensure that the output response behaves as expected. In contrast to Figure 35, Figure 41 shows the measured smooth duty cycle and temperature changes expected from the applied PID terms. As expected, the output reaches steady state at around 7 seconds and there is very minor overshoot. This shows that using the linear model to tune the non-linear system yields acceptable results.

Tuned Closed-Loop Plots of Duty-Cycle and Temperature Measurements vs Time
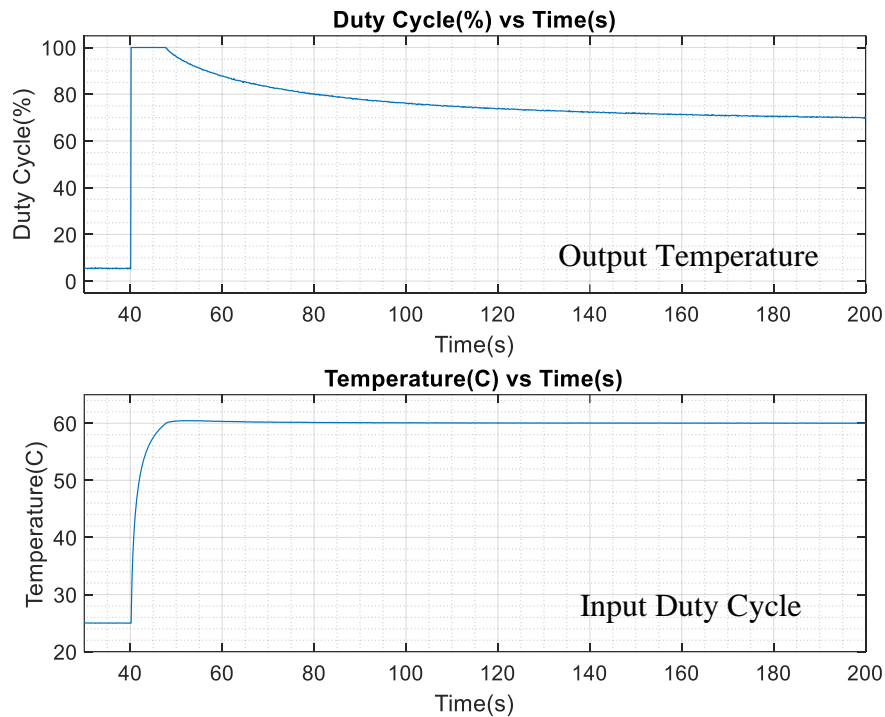
Figure 41: Closed-Loop measurements for tuned controller output response.

Testing Temperature Controller PCB

The first test involves ensuring that the temperature controller regulates temperature despite disturbances in external temperature. This is done by allowing the controller to reach steady-state and observing the change that occurs as the oven ambient temperature is increased from 25 to 55°C. Figure 42 shows that the controller returns to its original setpoint temperature of 60°C without ever exceeding 0.1°C of error.
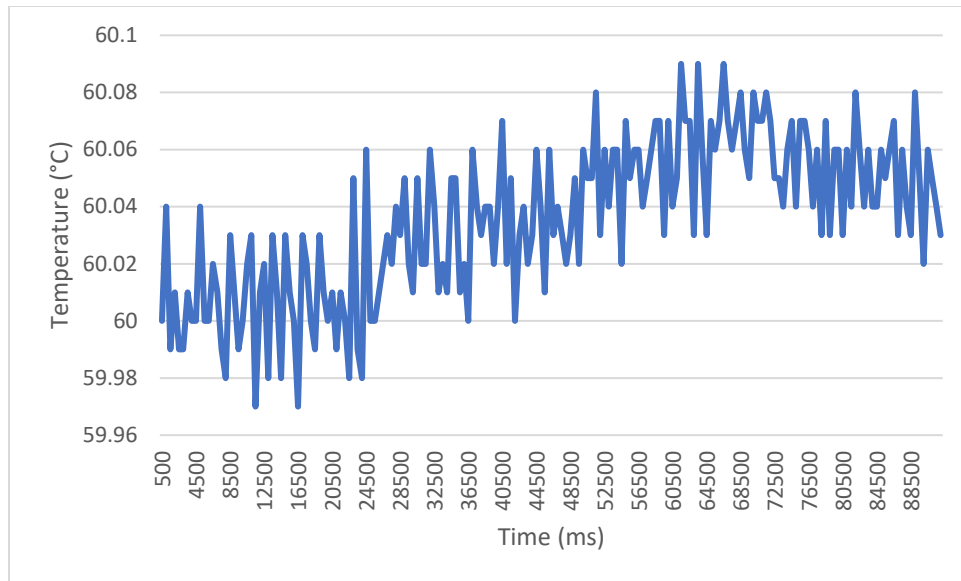
Figure 42: Measured controller behavior as oven temperature increases.

The second test performed is measuring duty cycle manually to ensure the controller's digital pin outputs the expected values. This is possible by connecting the output PWM pin to an oscilloscope and using the cursors to measure duty cyle while the contoller is regulating the temperature of the SAW device. Table 5 shows the duty cycle measurements recorded for ambient temperatures ranging from 25°C to 60°C. Duty cycle is calculated by measuring the times when the PWM output pin is toggled high (powering heater) and low (not powering heater). The columns labeled heater on and heater off are the times during which the controller was on and off, respectively. The duty cycle percent is calculated by dividing the total time the PWM is high by the total time and multiplying this by 100.

Table 5: Controller duty cycle measurements.

Oven Temperature (°C)	Heater On (ms)	Heater Off (ms)	Duty Cycle (%)
25	42	9	82.35294118
30	42	10	80.76923077
35	42	10	80.76923077
40	33	10	76.74418605
45	41	10	80.39215686
50	32	19	62.74509804
55	10	40	20
60	10	40	20

The maximum power supplied to the on-device heater by using the controller is calculated using a multimeter connected in series to measure current. The measurements are taken as soon as the power supply is turned on because it is at this moment that the difference between the read temperature and the setpoint is the largest. The controller setpoint is 60°C. The measurements recorded are shown below in Table 6.

Table 6: Power supplied to the on-device heater for a controller setpoint of 60°C.

Oven Temperature (°C)	Maximum Power (mW)
25	22 V x 25.35 mA = 557.75
30	22 V x 25.38 mA = 558.44
35	22 V x 24.94 mA = 548.78
40	22 V x 22.77 mA = 500.94
45	22 V x 20.19 mA = 444.13
50	22 V x 15.38 mA = 338.33
55	22 V x 10.33 mA = 227.24
60	22 V x 4.76 mA = 104.65

The results from Tables 5 and 6 confirm that the controller is working as expected. Table 5 shows that the duty cycle decreases from 82.35% to 20% as ambient temperature increases. Similarly, Table 6 shows that when the oven temperature is 25°C, a power of 557.75 mW is reached, whereas only 104.65 mW is measured when the oven temperature is at 60°C. Both tables confirm that the controller needs to use less power to regulate temperature at the setpoint as the ambient (oven) temperature increases. Note that the duty cycle for the controller is not constant as temperature is being regulated, meaning the duty cycles listed in Table 5 are actually the averages of several duty cycles.

<u>Comparing Arduino® and Simulink® Results</u>

The output temperatures recorded should be same whether the controller is programmed

from the Arduino® IDE or from Simulink® as long both are set to the same PID parameters,

setpoint and sampling time. Having similar results for this section is important because the

transfer function models obtained using Simulink® should be similar to the case when the

controller is programmed using the Arduino® IDE. These two methods of programming the

controller are compared for the open-loop controller during the linear-region of operation by

changing the duty cycle from 30% to 35%.

As expected, both output responses seen in Figure 43 are very similar. This simply

indicates that both are sending the expected PWM output value to heater.
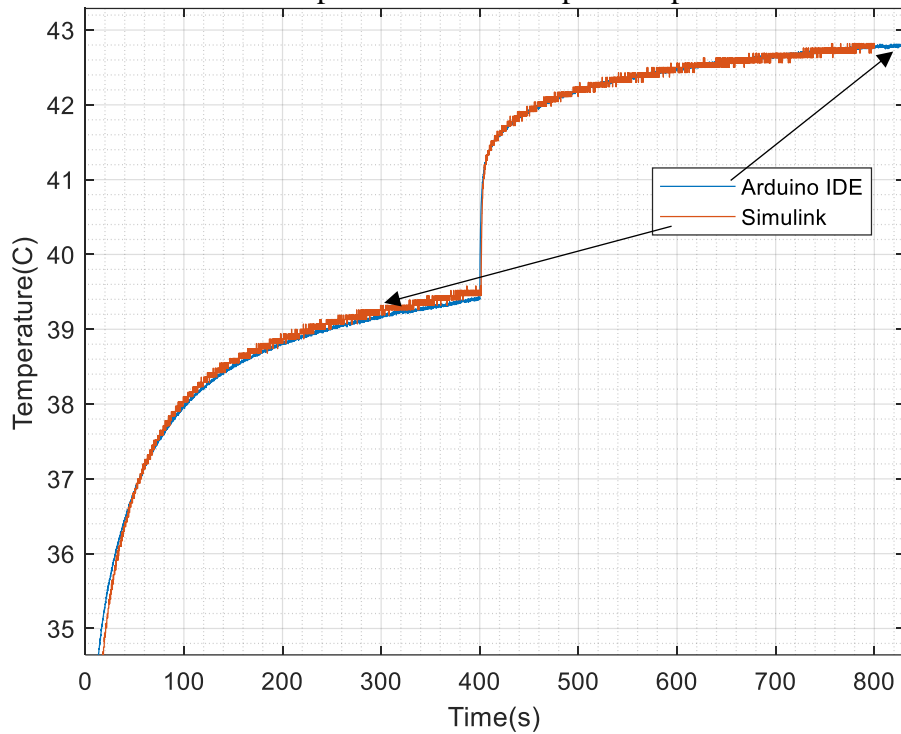
Figure 43: Open-loop comparison between Arduino® IDE and Simulink® (linear).

Next, the closed-loop linear data between Arduino® IDE and Simulink® are compared in Figure 44. These are obtained by changing the setpoint temperature from 30°C to 35°C at the same time. There appears to be a difference during the setpoint change. This is possibly due to differences in the PID implementation. The Arduino® measurements appear to be more accurate because each new measurement is the average of 10 different measurements.
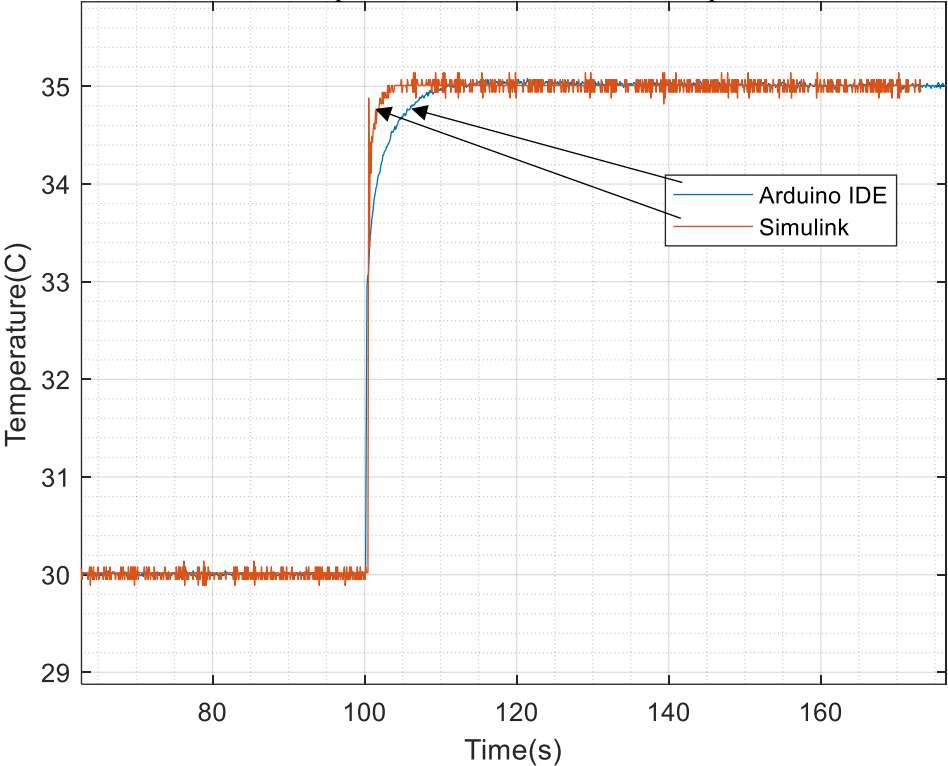
Figure 44: Closed-loop linear comparison between Arduino® IDE and Simulink® (linear).

Figure 45 checks for differences in closed-loop and non-linear measured temperature when programming the controller using the Arduino® IDE compared to using Simulink®. This plot shows minimal differences between the two ways of programming the controller.
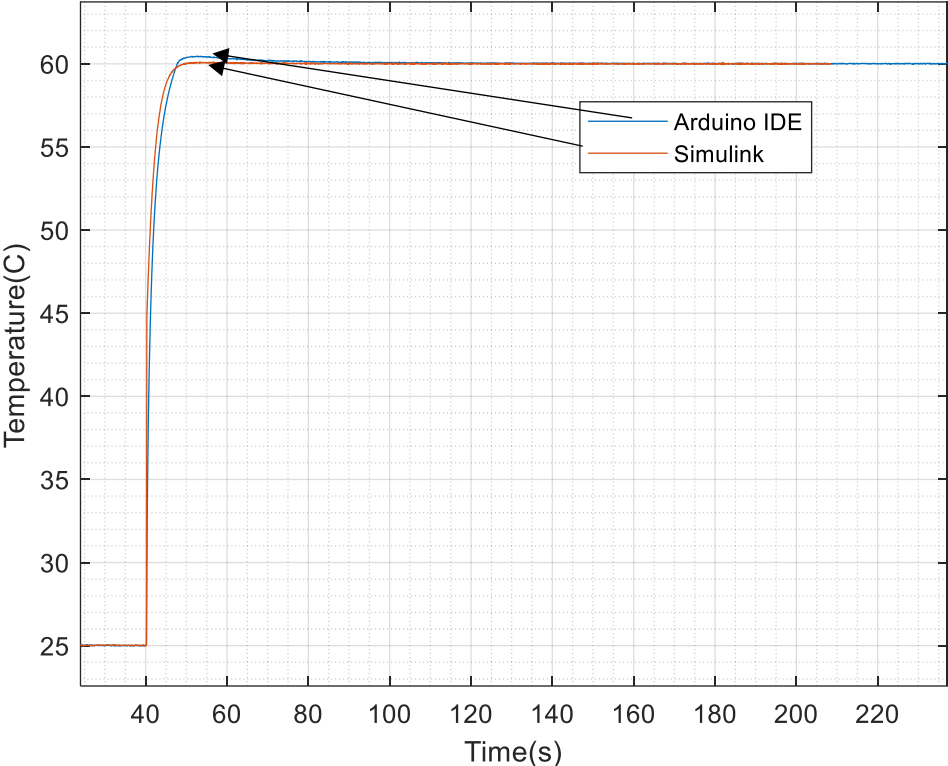
Figure 45: Closed-loop non-linear data in Arduino® IDE and Simulink® (non-linear).

CHAPTER 5: SAW RESULTS

SAW Device Theory

A SAW device uses two interdigital transducers (IDTs) on a piezoelectric substrate to convert an RF signal into an acoustic wave and back into an RF signal (Royer 59-60). SAW IDTs take advantage of the fact that the acoustic wave velocity is approximately 5 orders of magnitude slower than the speed of light in free-space in order to form compact delay lines (Brocato 14). In addition, its design versatility lends itself to other applications such as filters and signal processing (Brocato 9). An example of a SAW device is shown in Figure 46 (Morgan 5).
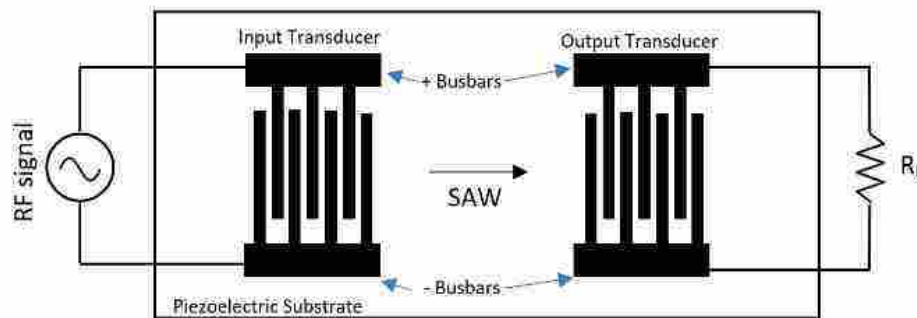
Figure 46. Example of SAW device input and output IDTs (Morgan 5).

Both, the input and output IDTs are made up of inter-digitated metal electrodes connected to positive or negative bus bars. The RF input connects to the input IDT's busbars, while the output connects to the output IDT's busbars (Morgan 4). At the input, the applied voltage produces a periodic electric field, which produces a stress in the substrate material due to its

piezoelectric properties (Morgan 5). The output transducer's bus bars connected to a load convert the acoustic wave back into an RF signal due to the piezoelectric effect's reciprocity (Morgan 5).

The second-order effect of concern in this thesis is the effects of temperature variation in the device response. The delay can be described as the length between the input and output IDTs divided by the velocity of the wave, both of which depend on temperature variation (Morgan 101). The velocity of the wave can be calculated using the temperature-dependent bulk constants of the substrate material, whereas the length is affected by thermal expansion (Morgan 101).

<u>SAW Device Packaging</u>

The SAW correlator is fabricated using the standard photolithography process shown in Appendix A. The fabricated devices are tested using the Cascade Probe and diced using a dicing saw. The device is glued onto a 20-pin package and wire-bonded using 1 mil gold wire to the package pins, as shown in Figure 47. The device embodiment consists of an input transducer and an output transducer surrounded by two aluminum thin film resistors. The RTD (red) is closer to the transducer and the heater (blue) surrounds the RTD. A coaxial cable is soldered to the pins connected to the input transducer in a common ground configuration. The signal, ground, signal outputs are connected to another coaxial cable. The packaged device is then soldered to the traces of the etched PCB using the pins on the package. Wires soldered to these PCB traces also connect the pins on the package to the temperature sensor and heater.
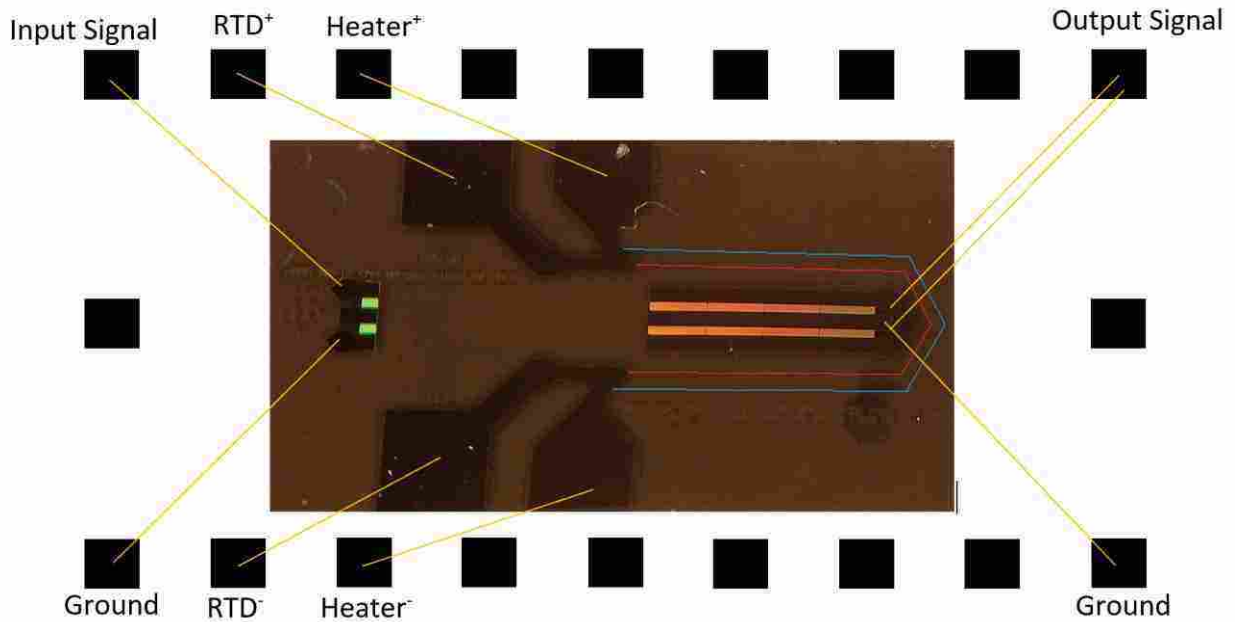
Figure 47: SAW correlator packaging and bonding.

MATLAB® S21 Processing

The transmission of the input RF signal is examined by first measuring S21 data in the expected frequency range of 850 MHz to 1050 MHz in Figure 48. It is possible to use the Vector Network Analyzer's time-gating tool to process these measurements. However, doing this in MATLAB® provides more flexibility in manipulating the data for reporting. The measurements are done for oven temperatures ranging from 25 to 55°C and the controller kept off. The enclosed oven is turned off during the measurement to reduce measurement distortion. Figure 48 shows that the non-gated frequency responses exhibit small ripples throughout a small range of frequencies. This is where the device response occurs.
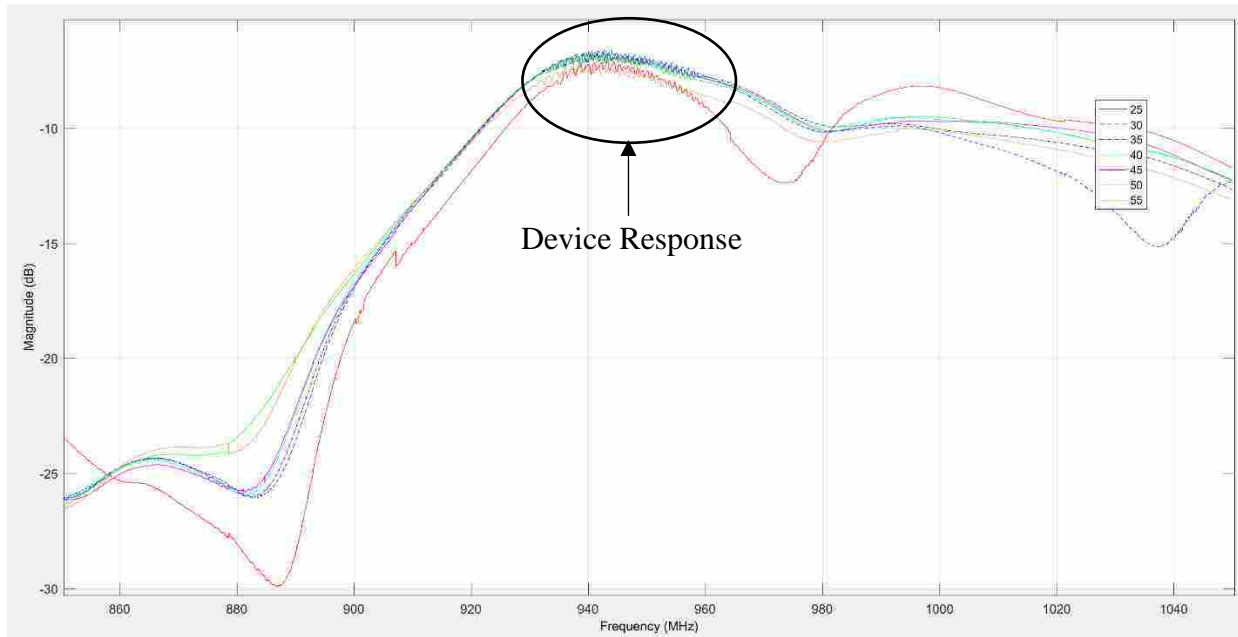
Figure 48: S21 (dB) response of the SAW correlator for different oven temperatures (non time-gated).

The raw S21 data is first converted into time-domain using the inverse Fast-Fourier transform function in MATLAB® in Figure 49. Then, most of the electrical response is zeroed out through time-gating so that the device response remains. The time data is zeroed from 0 to 0.6 μs and then from 3 μs to the end. This essentially extracts the time period during which the device response occurs at and reduces RF feed-through or spurious reflections that occur outside this time period.
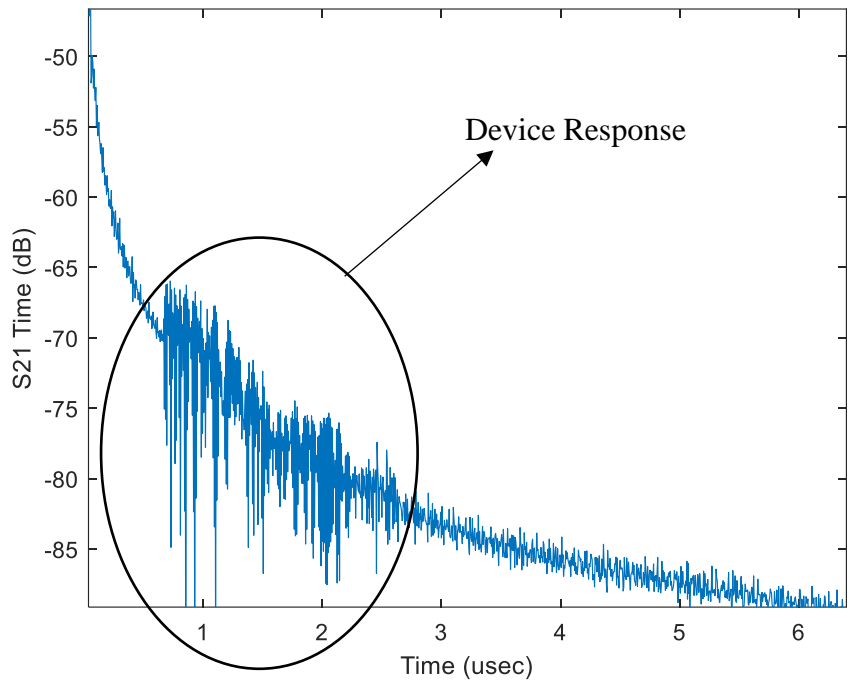
Figure 49: Time domain representation of S21 at 25°C (non time-gated).

The next step is to convert the resulting time-domain response back into frequency domain using the Fast-Fourier transform. The resulting post-processed S21 frequency response is shown in Figure 50 for oven temperatures 25 to 55°C. At this step, it is now possible to observe the shifts in frequency that occur when a SAW correlator's temperature changes. The frequency appears to decrease linearly as the temperature is increased from 25 to 55°C. These results are discussed further in the next section.
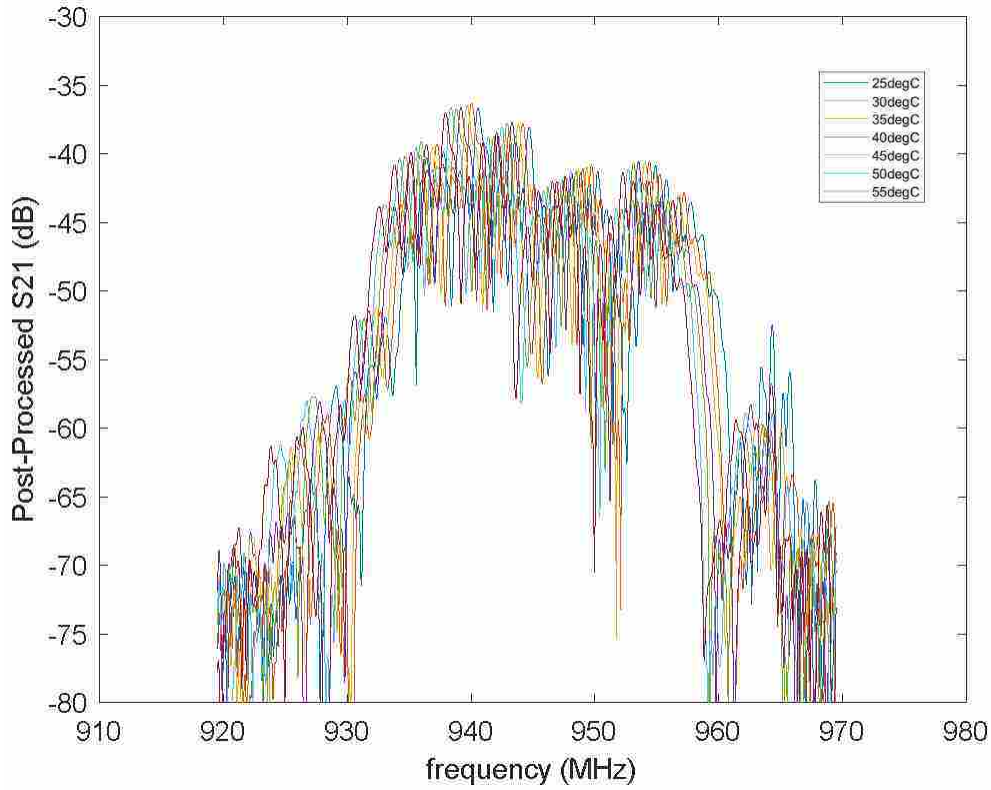
Figure 50: Post-processed S21 response of device for the different oven temperatures.

Further post-processing can be applied to make the shift in frequency as temperature changes clearer. The typical method to identify changes in frequency is by cross-correlating the current frequency to a baseline frequency. Cross-correlation Φ_{xy} in the frequency domain is obtained for this experiment by using the following formula, where X(f) and Y(f) are the responses from Figure 50 at room and the temperature of interest, respectively.

$$\Phi_{xy} = X^*(f)Y(f), \text{ where } * = \text{complex conjugate} \tag{26}$$

For this experiment, cross-correlation in the frequency domain between each of the oven

temperatures Y(f) and room temperature X(f) is performed in Figure 51.
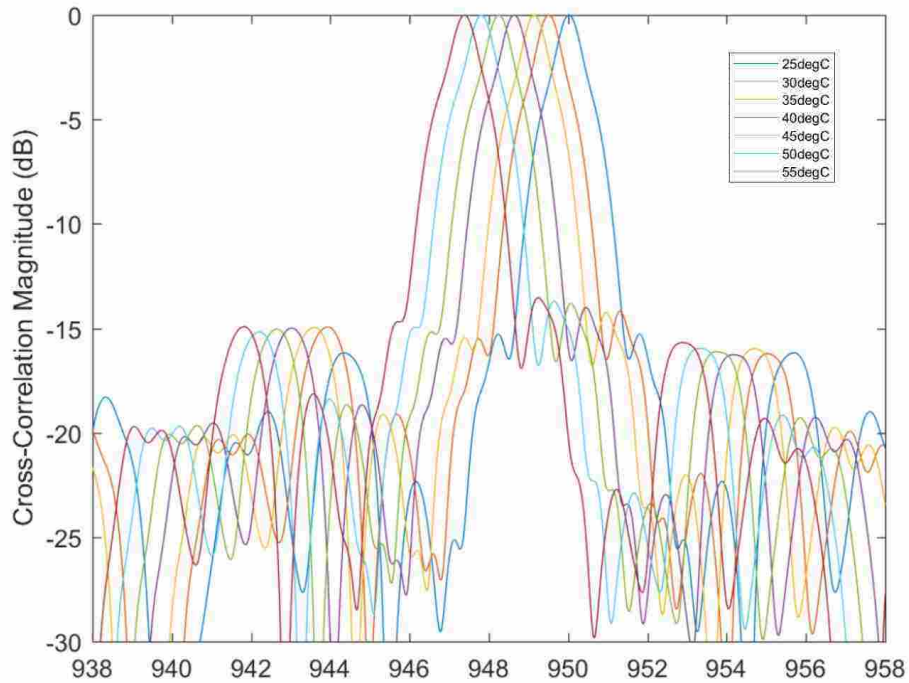
Figure 51: Cross-Correlated S21 for different oven temperatures.

All of the post-processing shown in the steps above can be done using the lines of code shown in

Appendix C.

Frequency Shift as a Function of Temperature & PPM Calculation

The shifts in frequency from the last section are recorded and plotted against temperature

in Figure 52 for the controller off case. As expected, the plot shows a linear relationship between

temperature and frequency shift. On average, the center frequency shifts 0.43 MHz for every 5°C

increment in oven temperature. The maximum shift in frequency is 2.6 MHz.
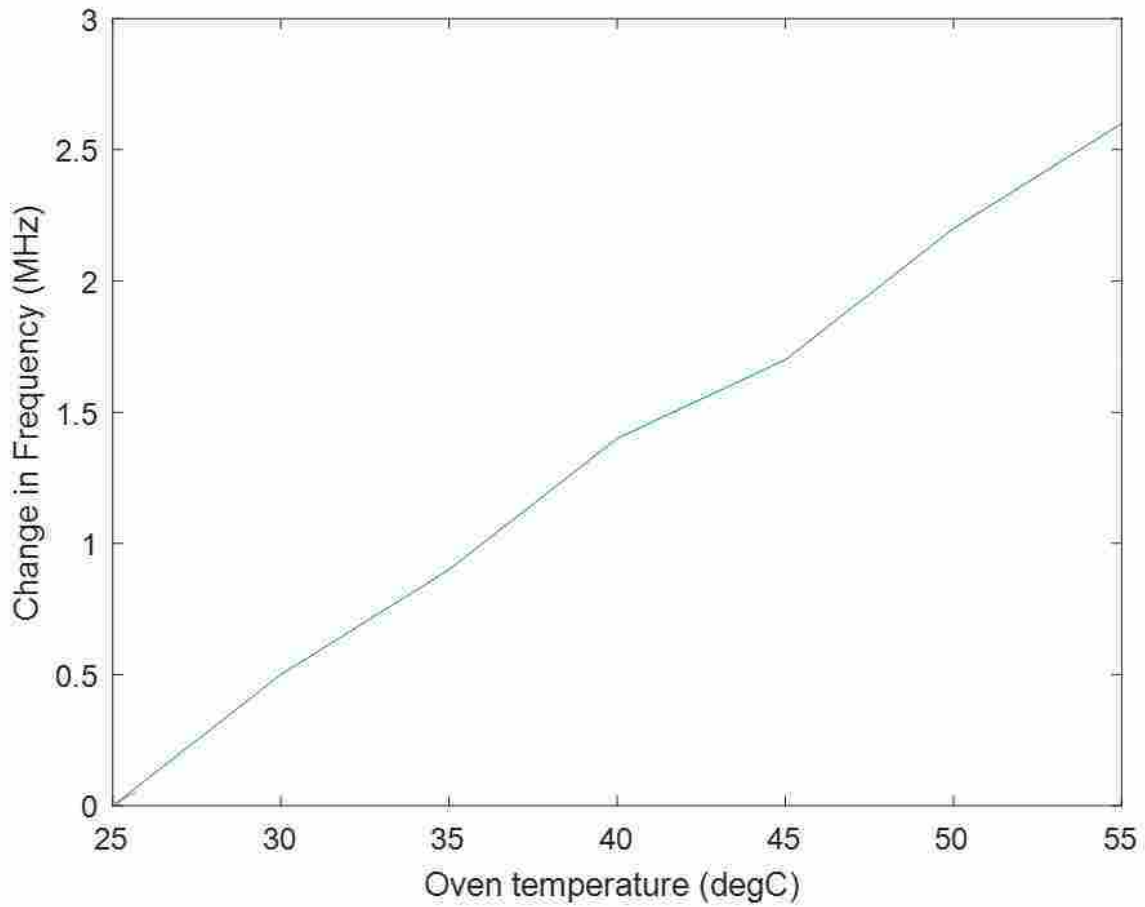
Figure 52: Change in device center frequency when the controller is off.

The S21 data is retaken for the case with the temperature controller on and regulating temperature at 60°C. The same post-processing procedure from the last section is repeated to produce Figure 53. As expected, there are minimal shifts in frequency.
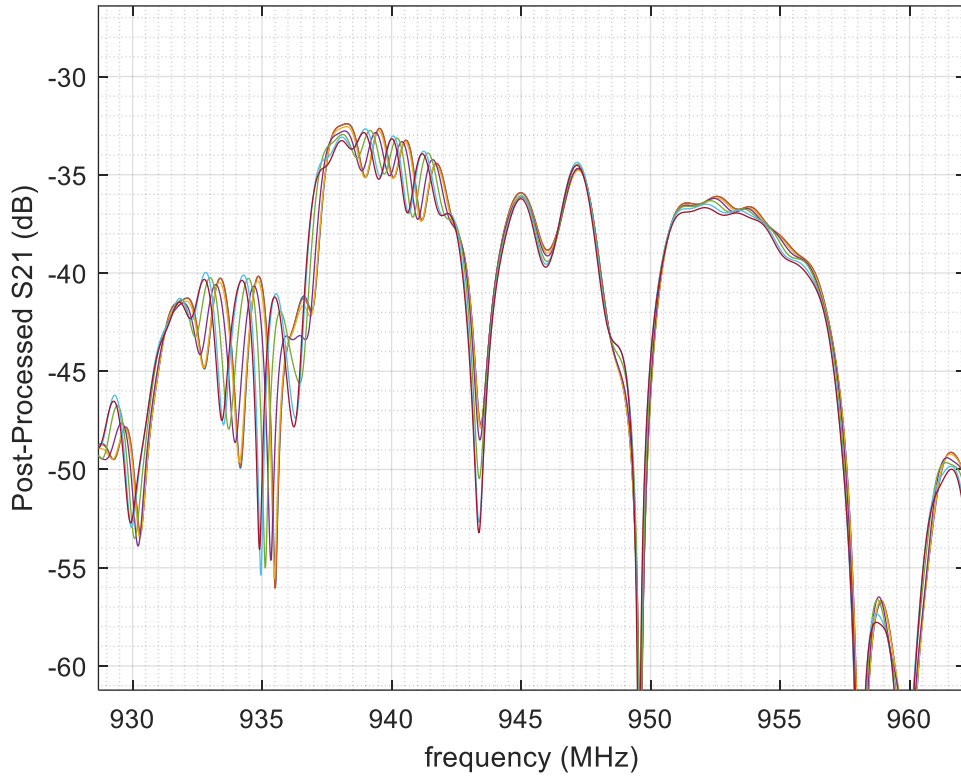
Figure 53: S21 for different oven temperatures with controller on.

The shifts in frequency are measured for seven different oven temperatures in order to evaluate the controller's ability to maintain temperature stable at the setpoint. These are plotted against ambient temperature in Figure 54. The data is measured by first setting the oven to 55°C and letting it cool down slowly. These results show that the device's center frequency is not a function of the oven temperature due to the controller maintaining a nearly constant temperature.
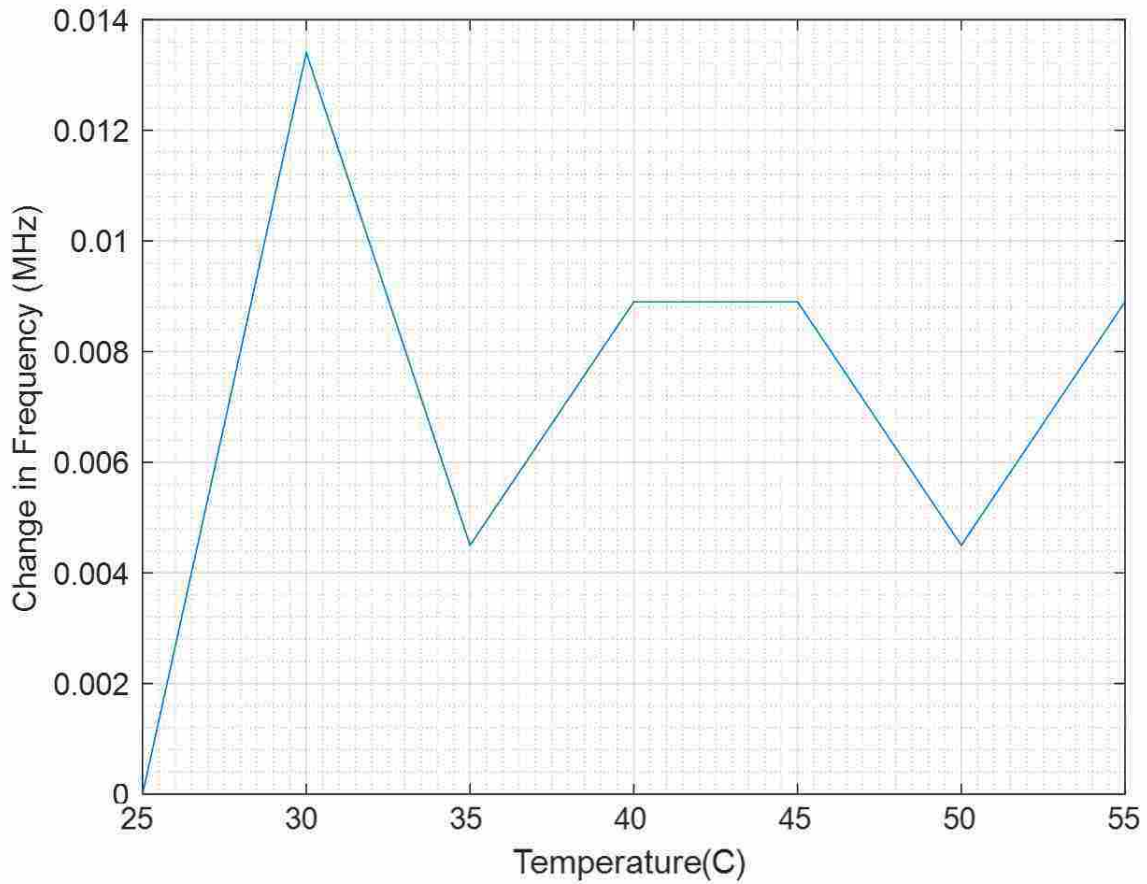
Figure 54: Change in frequency for closed-loop controller at different oven temperatures.

The PPM change at each temperature is then calculated to describe the change in frequency in units of parts per million. This is done in Figures 55 and 56 from Figures 52 and 54 using Equation 27. In this equation, $f_{currentT}$ is the center frequency at the current temperature and f_{roomT} is the center frequency measured at room temperature.

$$PPM = \frac{|f_{currentT} - f_{roomT}|}{f_{roomT}} \qquad (27)$$
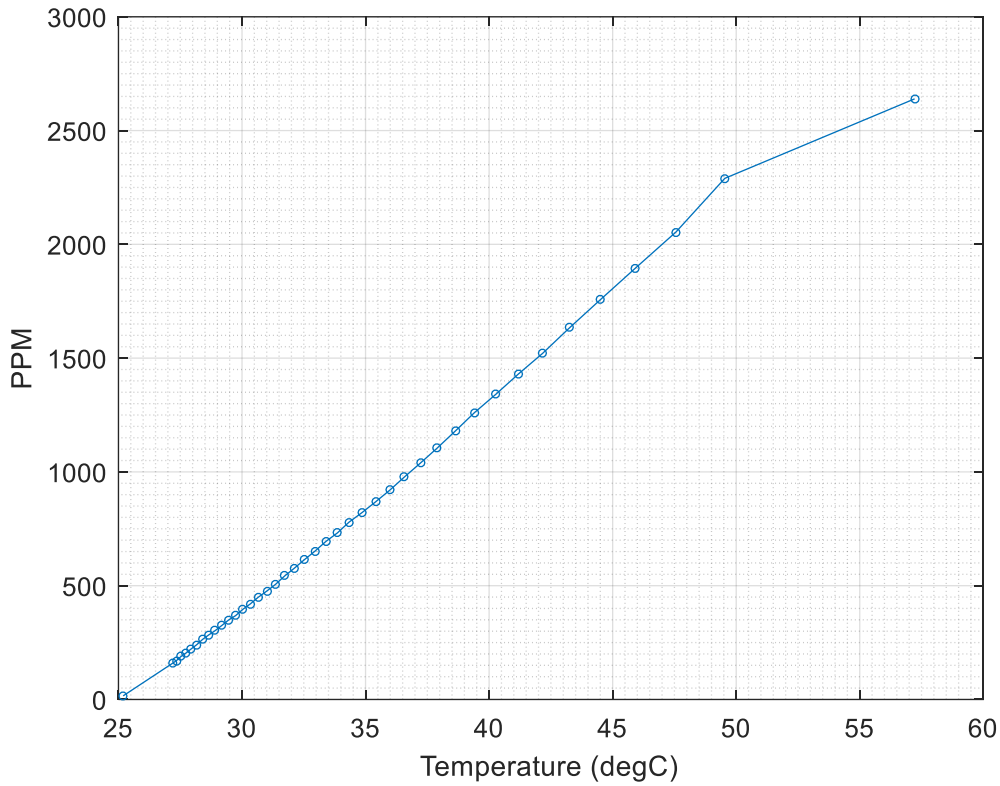
Figure 55: PPM as a function of temperature when the controller is kept off.

PPM varies mostly linearly as temperature of the device changes. The derivative of the line is found using Microsoft Excel to find a linear trendline in order to solve for PPM/°C. The generated equation of the line is shown in Equation 28 and its derivative results in 89.82 PPM/°C, which is close to the expected value of 94 PPM/°C for Lithium Niobate (Yurish 160). T in Equation 28 is the x-axis of Figure 55 and PPM represents the y-axis.

$$PPM = 89.82\,T - 2291.3 \qquad\qquad (\,28\,)$$

By examining PPM with the controller on from Figure 56, it is possible to describe how well the controller regulates temperature. Changes in PPM/°C show the ability of the controller to maintain the device's temperature stable as oven temperature changes.
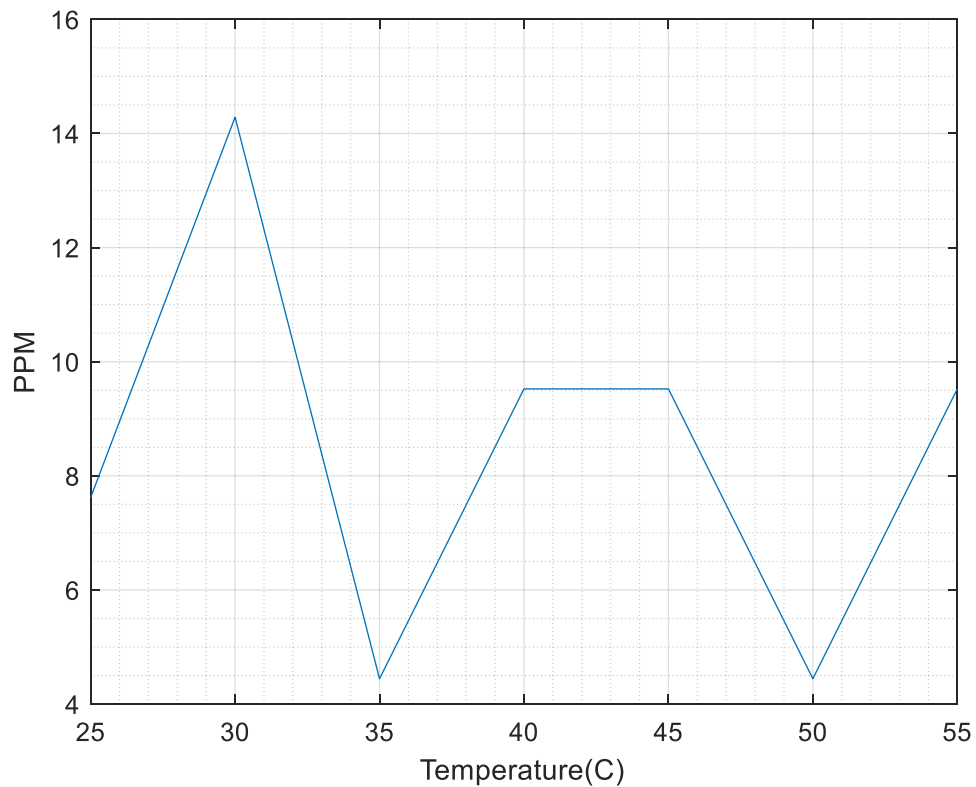
Figure 56: Closed-loop PPM as oven temperature changes.

The worst-case PPM/°C occurs whenever the maximum change in PPM occurs. This occurs between 30°C, at 14.29 PPM, and 35°C at 4.48 PPM. PPM/°C is calculated in Equation 29 by finding the ratio of the largest PPM change over the temperature change it occurs on.

$$\frac{14.29-4.48 \, PPM}{35-30 \, °C} = \frac{9.81 \, PPM}{5°C} = 1.962 \, PPM/°C \qquad (29)$$

Alternatively, this can be expressed as the worst PPM experienced over the entire range of

temperature changes, as shown in Equation 30.

$$\frac{14.29 \, PPM}{55-25 \, °C} = 0.48 \, PPM/°C \qquad (30)$$

73

CHAPTER 6: CONCLUSION

A PWM-based PID controller for regulating temperature of a SAW correlator was implemented successfully using an on-wafer heater and RTD resistors. The accuracy of the controller is shown by its stability in the measured temperatures and by the low variation in frequency shift that amounted to a worst-case PPM experienced over 30°C of 0.48 PPM/°C. The developed linear and non-linear plant models were both shown to be good models of the plant. As shown in Table 7, the results did not show which is the better model. Therefore, using a linear model of at least 4th order is recommended because of simplicity and for easier implementation with other tools that may not allow for a Hammerstein-Wiener non-linear model.

Table 7: Linear and non-linear model fit to measurement data.

	Linear Model	Non-Linear Model
Closed-Loop	92.49%	91.43%
Open-Loop	92.24%	94.29%

Future work would include developing a method of cooling the SAW correlator when it is in a hot environment. Combining the presented method of heating the device with a method for cooling would result in a controller capable of regulating temperature under any environment. One easy way of accomplishing this is by placing a Peltier device in proximity to the correlator and using another output PWM pin to drive it. The threshold for changing between heating and cooling is all controlled in programming.

Another possibility for future work is to develop a Graphical User Interface using

MATLAB® App Designer (MathWorks, App Designer). Programming and editing the

temperature controller using Simulink® requires installation of the program and basic know-how.

Packaging an app designed in App Designer would fix both problems. A prototype for this is

shown in Figure 57.
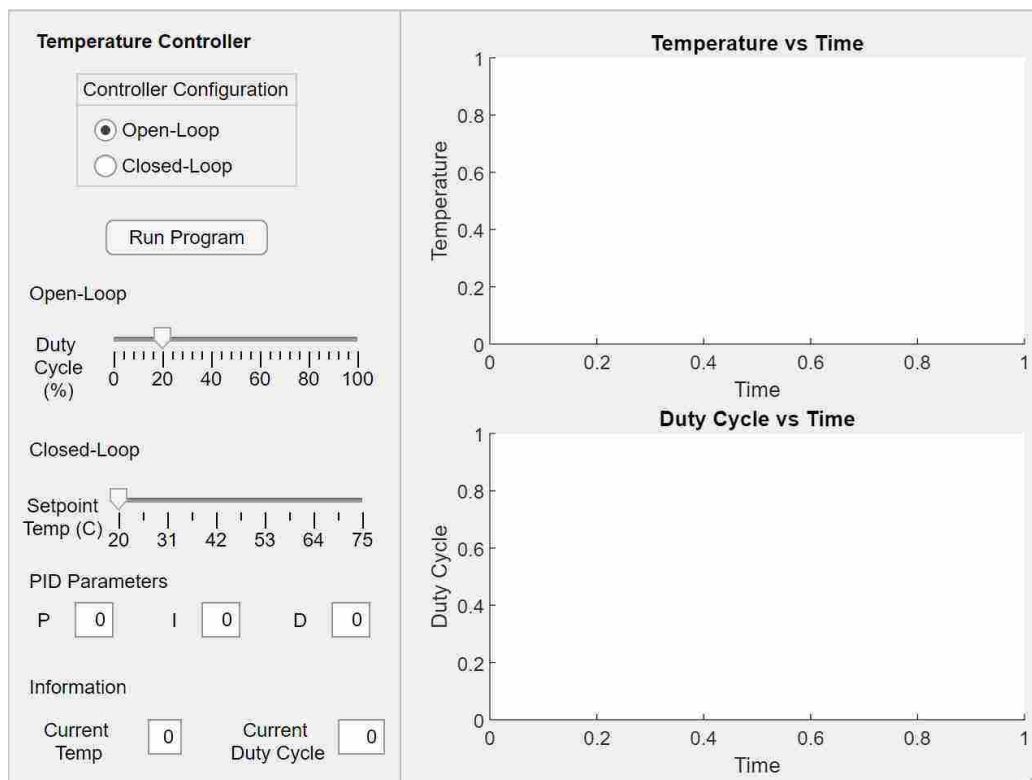
Figure 57: GUI for temperature controller.

Although the thesis required measuring the temperature experienced by the SAW

correlator, the method implemented is not the best way to measure temperature. Electronics in

the temperature controller limit the environment temperature of this system. A more practical

method for temperature sensing involves using an antenna to transmit the device response using a SAW temperature sensor. The response, as shown throughout this thesis, can be used to sense temperature quite accurately after signal processing.

APPENDIX A: FABRICATION OF SAW DEVICE

1. The LiNbO$_3$ wafer is prepared by rinsing it with deionized water, acetone, and methanol to get rid of contaminants. The mask is cleaned using acetone and methanol. Both are dried using Nitrogen.

2. The wafer is placed on a spinner and adhesion promoter (hexamethyldisilazane) is added uniformly across the top using a disposable pipettes. It is important to ensure there are no bubbles before setting the spinner to 4000 RPM for 40 seconds.

3. Positive photoresist is applied uniformly and the spinner is set to 4000 RPM for 40 seconds. A foam swab damped with acetone is used to remove the bead around the wafer before it stops spinning.

4. The wafer is placed on a hot plate pre-heated to 100°C for 90 seconds. After 90 seconds, the wafer is placed on a metal plate while it cools down.

5. The wafer is placed on the Karl Suss mask aligner with the flat side of the wafer aligned to the chuck. The substrate is exposed to UV light for 30 seconds.

6. Once exposed, PLSI Type 2 is poured on a petri-dish. This is a 11:1 solution of 1 part PLST to 11 parts deionized water. The wafer starts developing once it comes into contact with the developer. The petri dish is agitated in circular motions for 30-45 seconds until the wafer is developed. Once the features clear out, the substrate is rinsed with deionized water and dried with N$_2$.

7. The wafer is verified under a microscope for any flaws. After discarding the developer, the wafer is placed in the plasma cleaner for 30 seconds.

8. The wafer is placed in the Electron Beam machine's planetary for metallization. Step by step instructions for operating this machine are listed next to it.

9. The wafer is placed in an acetone-filled petri dish and the petri-dish in the ultrasonic bath with water. The petri-dish is removed once the excess metal comes off.

10. The wafer is removed from the petri dish slowly while it is rinsed with acetone to ensure the excess metal does not stay on top of the wafer.

11. The wafer is cleaned with methanol, dried with N_2, and examined under microscope.

APPENDIX B: PID CONTROLLER ARDUINO® IDE CODE

```
#include <PID_v1.h> //PID library
#include <Adafruit_ADS1015.h> //ADS 1115 ADC library
Adafruit_ADS1115 ads(0x48);

int PWM_pin = 3; //PWM output pin used to power heater
float Voltage = 0.0;
double Setpoint, Input, Output; //variables needed for PID library
double time; //tracks time. Used for processing the data.

int16_t adc0; //creates 16-bit integer for 16 bit ADC
int analogPin= 0;
float raw= 0.0;

//Voltage Divider variables
float Vin= 5.02; //Vin measured with DMM
float Vout= 0.0; //Vout instantiated
float R12= 994; //R12 measured with DMM
float Rrtd= 0.0; //RTD instantiated

float buffer= 0.0;
float sensedtemperature= 0.0; //same as temperature_read
float temperature_read = 0.0; //same as sensedtemperature

double Kp=18, Ki=10.1, Kd=0;  //PID constants

//create PID using corresponding variables
PID myPID(&Input, &Output, &Setpoint,Kp,Ki,Kd, DIRECT);

void setup() {
  Serial.begin(9600);
  ads.begin(); //call ADC library
  ads.setGain(GAIN_ONE); //input reading range of +/-4.096V
  pinMode(PWM_pin,OUTPUT); //set pin 3 as an output
  Setpoint = 50; //sets setpoint
  Input = temperature_read;
}

void loop() {
  time = millis(); //counts time

  temperature_read = readThermcouple(); //reads the temperature
  Input = temperature_read;
  myPID.Compute(); //Computes the PID results using current i/o values
  analogWrite(PWM_pin,Output); //Write duty cycle to output PWM pin
```

```
  Serial.print(Input); //Prints temperature read to serial
  Serial.print("\t");
  Serial.print(Output); //prints PWM duty cycle to serial
  Serial.print("\t");
  Serial.println(time); //Prints time to serial
  }
}

float reading_to_temp() { //function that converts reading to temperature
float rawsum=0; //used to store sum of multiple readings
for (int j=1; j<=10; j++) //averages 10 ADC readings for higher accuracy
 {
 rawsum += ads.readADC_SingleEnded(0);
 }
 raw=(rawsum/10); //average of 10 readings

 Vout= (raw * 0.125)/1000; //output voltage measured using voltage divider
 buffer= (5.02/Vout) -1; //obtains multiplication factor used to find Rrtd
 Rrtd= R12 * buffer; //calculates RTD or R12 resistance
 sensedtemperature=  (Rrtd-705.51)/(1.1481); //gets temperature using Rrtd
 return sensedtemperature;
 }
```

APPENDIX C: S21 PROCESSING MATLAB® CODE

```matlab
pathNameIn = 'C:\' //folder containing file

fileNameIn_25deg = sprintf( '25.txt' ); %store VNA data in variable
headerLength = 1; %size of header in text file
fid = fopen( [pathNameIn, fileNameIn_25deg] ); %open file in path
txtData_25deg = textscan( fid,'%f %f %f %f %f %f %f %f %f','HeaderLines',headerLength,
        'Delimiter',',' );  %imports data into columns
fclose( fid ); %close the file that is open

S21_25deg = (txtData_25deg{:,6})+1j*(txtData_25deg{:,7});%store columns 6 & 7 (s21)
s21_25deg = ifft(S21_25deg); %convert frequency-domain data to time-domain data
s21_25degFilt = s21_25deg; %copy variable contents
s21_25degFilt(1:43) = 0; %sets to 0 undesired portion in time-domain data
s21_25degFilt(178:end) = 0; %sets to 0 undesired portion in time-domain data
S21_25degFilt = fft(s21_25degFilt); %convert back to frequency domain

plot(freq,db(S21_25degFilt)) %plot S21
```

LIST OF REFERENCES

Banzi, Massimo. "Arduino Uno SMD." *Arduino*, 2012,

    www.arduino.cc/en/Main/ArduinoBoardUnoSMD.

Beauregard, Brett. "Arduino PID Library." *Arduino Playground*,

    playground.arduino.cc/Code/PIDLibrary/.

BK Precision. "1665 Series Bench Switching DC Power Supplies Model 1666."

    bkprecision.com/products/power-supplies/1666-1-40v-5a-switching-dc-power-

    supply.html.

Brocato, Robert W. "Programmable SAW Development." Sandia National Laboratories, 2004.

Bruckner, Gudrun, and Jochen Bardong. "Wireless Readout of Multiple SAW Temperature

    Sensors." *MDPI*, 12 July 2019.

Cheever, Erik. "The Root Locus." The Root Locus , Swarthmore College,

    lpsa.swarthmore.edu/Root_Locus/RootLocus.html.

Earl, Bill. "Adafruit 4-Channel ADC Breakouts." *Adafruit*, 5 Feb. 2019, cdn-

    learn.adafruit.com/downloads/pdf/adafruit-4-channel-adc-breakouts.pdf.

FTDI Chip. FT232R USB UART IC Datasheet. 4 Apr. 2019,

    ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf.

Gajic, Zoran. 4.4 Block Diagrams. Rutgers University,

    eceweb1.rutgers.edu/~gajic/solmanual/slides/chapter4_BD.pdf.

Gong, Xun, and De Zhang. "A Novel Temperature Stable Composite Substrates for Surface

    Acoustic Wave Applications." Symposium on Piezoelectricity, Acoustic Waves,

    and Device Applications, 2008.

Haugen, Finn. *PID Control*. Tapir Academic Press, 2004.

Keysight Technologies. 8753ES S-Parameter Network Analyzer. keysight.com/en/pd-
1000002292:epsg:pro-pn-8753ES/s-parameter-network-analyzer?cc=US&lc=eng.

Ljung, Lennart. *System Identification Toolbox User's Guide*. MathWorks, 2015.

MathWorks. Analyze Design in PID Tuner. mathworks.com/help/control/getstart/analyze-
design-in-pid-tuner.html

MathWorks. MATLAB App Designer. mathworks.com/products/matlab/app-designer.html

MathWorks. Estimating Linear Models. mathworks.com/help/ident/gs/identify-linear-models-
using-the-gui.html

MathWorks. What Are Polynomial Models?. mathworks.com/help/ident/ug/what-are-
polynomial-models.html

MathWorks. Simulink® Support Package for Arduino® Hardware. 21 Oct. 2019,
mathworks.com/hardware-support/arduino-simulink.html.

MathWorks. Transport Delay. mathworks.com/help/simulink/slref/transportdelay.html

Messner, Bill, and Dawn Tilbury. "Introduction: PID Controller Design." *Control Tutorials for
MATLAB & Simulink®*, MathWorks.

Messner, Bill, and Dawn Tilbury. "Introduction: System Analysis." *Control Tutorials for
MATLAB & Simulink®*, MathWorks.

Morgan, David. Surface Acoustic Wave Filters: With Applications to Electronic
Communications and Signal Processing. 2nd ed., Academic Press, 2007.

National Instruments. "General-Linear Model Definitions (System Identification Toolkit)." June

    2013, zone.ni.com/reference/en-XX/help/372458D-

    01/lvsysidconcepts/modeldefinitionsgl/.

Royer, Daniel, and Eugene Dieulesaint. Elastic Waves in Solids II. Springer, 2011.

Skogestad, Skogestad. "Simple analytic rules for model reduction and PID controller tuning." J.

    Process Control 2001, 13, 291–309

Sun Electronic Systems, Inc. "Temperature Test Chambers: Sun Electronic Systems, Inc."

    Temperature Test Chambers , sunelectronics.com/Temperature-Test-Chambers.html.

Yurish, Sergey Y. Smart Sensors and MEMS: Proceedings of the NATO Advanced Study

    Institute on Smart Sensors and MEMS. Vol. 181, Kluwer Academic Publishers, 2004.