NSF Lab Furnace Control System

DESIGN DOCUMENT

Team 47
Client & Advisor: Dr. Gary Tuttle

Team Members
Nick Brylski - Systems Engineer
Jeremy hartl - Hardware Engineer / Report Manager
Adam Matthews - Embedded Systems Engineer
Kevin Lang - Hardware Engineer
Chris Pohlen - Software Engineer / Gitlab monitor

https://sdmay19-47.sd.ece.iastate.edu/team.html

Revised: 12/4/18 Version 2

Table of Contents

List of Figures

List of Tables

N/A

List of Definitions

OTC - Omega CN616 PID Temperature Controller. A device that can set and monitor up to 6 independent temperature channels.

MFC - Mass flow controller. A device that sets and monitors the rate of gas flow by measuring the mass of the gas.

Oxidation - The forming of an oxide layer, commonly silicon dioxide, on top of a substrate.

Doping - The injection of chemicals into a material to create more electrons or holes

DAC - Digital to analog converter

ADC - Analog to digital converter

RS-232 - asynchronous serial communication standard used to transmit data over distance

I2C - synchronous serial communication standard , 2 wire

SPI - synchronous serial communication standard ,3 wire

1 Introduction

1.1 Acknowledgement

Senior design team 47 would like to thank our client and advisor Dr. Gary Tuttle. He has provided us with great insight and project materials. We would also like to thank Dr. Meng Lu, the current instructor for EE 432, and PhD student Harsh Gaonkar, a lab TA for EE 432.

We would also like to thank Texas Instruments and Analog Devices for providing us free samples for this project.

We'd also like to thank our TA Tim Lindquist for providing valuable input on our project.

1.2 Problem Statement

The NSF lab contains four furnaces that are used for the oxidation and doping of silicon wafers in EE 432: Microelectronic Fabrication. Three of these furnaces are operational. During operation, the state of any furnace is determined by two categories of parameters, temperature and gas flow rate. Each furnace has 3 temperature zones and either 1 or 3 gas channels. Temperature is controlled by two Omega CN616 PID Temperature Controllers (OTC) that can control set points and time profiles for up to 6 independent zones each, which can be seen in appendix 4.3.1. One controls 6 zones across two furnaces, and the other controls the 3 zones of the third. If the fourth furnace becomes operational, it will presumably be controlled by the second OTC. Gas flow is controlled by mass flow controllers (MFC) with one MFC per channel. Currently, set points and time profiles for the OTCs are controlled by three onboard buttons, and all feedback is displayed on 7 seven segment display digits. Each MFC has a dedicated control knob and seven segment display, as seen in appendix 4.3.2.

The control segment of this system can be greatly improved. Simply setting the temperature on a single zone requires a lengthy sequence of semi-ambiguous button presses that is prone to user error. Setting time profiles is even more complex. The controls for gas flow are simple, but there is no way to automate changes in flow rate.

The solution that we will implement will automate both the temperature control and gas flow control with a microcontroller. This microcontroller will be able to set and read temperature from the temperature controller. It will also be able to create profiles; e.g ramps 4 celsius/ min up to 1000c then hold for 15 hours--then ramp back down at 3 celsius/min to room temp. These profiles will include setting and shutting off the gas flow at certain time points. All temperature and gas flow information will be read on a mounted display connected to the microcontroller. A program on a separate computer, also connected to the microcontroller, will allow the user to navigate through and control each of the 4 furnaces.

1.3 Operating Environment

The operating environment for our product is the NSF lab in the Applied Science Complex. Our device, while interfacing with a temperature controller, will not come into contact with the 1000°C+ furnace. The user interface however, will be used on a weekly basis by students, TA's and researchers for years to come. This means that we will need to choose components that can last for many years. The display that we will show this information on will also need to be in robust housing to prevent anyone from bumping into it or getting scratched.

1.4 Intended Users and Intended Uses

The intended users for our NSF Furnace Control System are students, researchers, professors, and teaching assistants.

A heavy portion of the furnace usage is for the academic purposes of Iowa State University's EE 432 course. This Microelectronic Fabrication Techniques course consists of approximately 5 labs with 5 students and 1 teaching assistant per lab. These labs meet every week and last between 3-4 hours. Most students using the equipment have no previous experience with the NSF furnace. The heavy usage by inexperienced students requires our design to be user friendly and intuitive.

Researchers and professors who have previous experience using the furnace will be using it for longer, more precise processes. This kind of usage will desire more control over the equipment along with a better automated control. That way it can run on weekends without requiring the professor/researcher to come in and make adjustments.

1.5 Assumptions and Limitations

Assumptions:

- We will not need to do any PID tuning of the OTC's, but our project will allow for tuning commands to be sent.
- We will not need modify any existing thermocouples on the OTC's
- Has to withstand continuous usage and run 24/7
- The funding for the project is fairly large, we should have no problem getting all the hardware we need

Limitations:

- Can't be readily tested due to heavy lab usage and the immobility of the furnace
- Must be compatible with existing power supplies and hardware rack
- Will need to be compatible with 5V logic of MFCs
- will need to be compatible with RS232 protocol of OTCs
- The furnace will not be usable for testing until after Thanksgiving and Spring breaks, when the class finishes using the furnace.
    - This means that the final phase of testing and installation may not happen until around April 2019

1.6 Expected End Product and Other Deliverables

The end product that we will be delivering to our client will be a user friendly interface to control the OTC's and MFC's. This will be accomplished with a GUI's, a computer running the GUI, an Arduino Mega 2560, and an Arduino shield and Python API of our own design. One GUI will be an always-on status readout running on a computer and displayed on a screen that can be mounted to the furnace equipment, and the other will run on a laptop and will be used to control the furnace settings. Both will use a serial interface to communicate with the Arduino that will intelligently pass commands to and read results from the OTC's and MFC's. In order to successfully communicate with these devices, the shield for the Arduino will include a digital-to-analog converter (DAC) and an RS232 transceiver, as well as the necessary cable connectors.

The GUI will include temperature settings and readouts. It will also have gas flow values for each of the furnaces. The control interface will allow users to set temperatures, timers, and temperature profiles.

The accuracy of the furnace control system is limited by the Omega Temperature controller and Mass Flow Controller (MFC). The Omega Controller has an accuracy rating of ± .2% or ± 2°C. The Mass Flow Controller is rated at an accuracy of one percent. The chosen DAC that will communicate with the MFC is rated at .5% which will allow the MFC to retain its accuracy at about one percent.

Additionally, several useful documents will be delivered to our client. For the user interface, a brief, descriptive document will be delivered. This document will contain basic operation instructions. However, the user interface as a whole will be very intuitive and most users will not need documentation to figure out the controls. Additionally, the Design Document and Project Plan will be available to the client. These documents are for the client to review and understand the process that we went through when designing our control system. To allow our client to find express concerns with our process and overall design, these documents are constantly available to him long before we deliver our final project.

The final completed, end product will be delivered by week 13 of spring semester (April 8-12).

2. Specifications and Analysis

2.1 Proposed Design

Our design will connect the MFC's and OTC's to an Arduino Mega. This Arduino Mega will be controlled by a Python GUI running off a computer. This will allow complete digital control of the system. This can be summarized with the following block diagram below. This blocks represent the main functional aspects of the system, like controlling the gas flow or displaying information. The arrows represent the interfaces we are using to connect the devices together, what type of information is being sent. This system accomplishes the goal by integrating hardware, software, communication specifications, and design requirements.

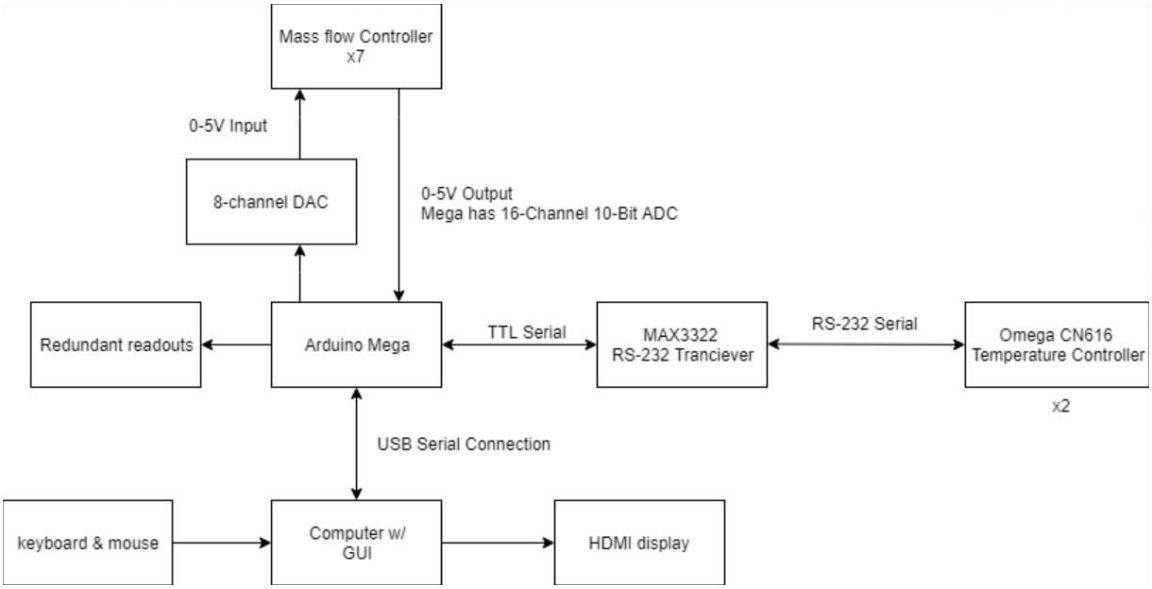

Figure 1. Block diagram showing the major parts of the project.

These specifications include the majority of the functional requirements at a high level, but there are a few non-functional requirements as well. The current control system's complexity and difficulty to use were the major motivators for this project. The user interface we create must therefore address those issues. The control system we create should be able to be easily learned and used, require fewer individual actions from the user, and be a bit more resistant to user error.

Our proposed design uses an Arduino Microcontroller to act as a middleman between a user interface and both the temperature controllers and the mass flow controllers. This allows for modularization of the project to a degree. A change in a type of hardware only requires a few changes to the code. We have created a block diagram to show the solution we will be implementing; this can be seen below. The diagram implies a system with four distinct parts: the temperature controllers and their transceivers, the mass flow controllers and their DACs, the user interface and controls, and finally the microcontroller to connect the different parts together to allow for a common protocol between the different parts.

2.1.1 Functional Requirements

A computer program that will accomplish the following:

Temperature control - For each zone in each furnace allow user to:

- set temperature setpoints
- set alarm setpoints

Temperature monitoring - For each zone in each furnace allow user to:

- view current temperature
- view temperature setpoints
- view alarm status
- view alarm setpoints

Gas flow control - For each gas in each furnace allow user to:

- set gas flow rate

Gas flow monitoring - For each gas in each furnace allow user to:

- view current gas flow rate

Profiling - For each furnace allow user to setup and monitor a profile (as seen in the image below taken from reference 1) that automates furnace control:

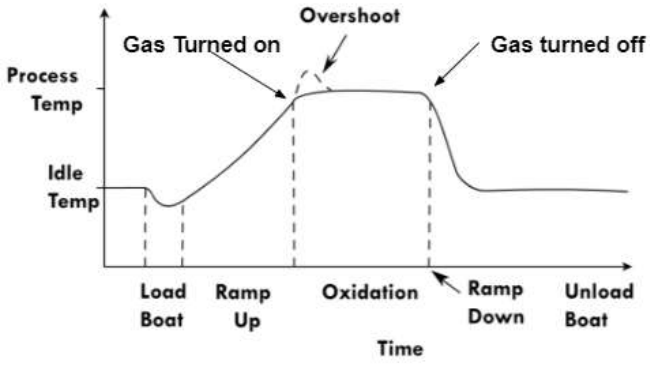

Figure 2. Timing diagram showing the furnace's active ramping cycle.

An always-on device and display that will accomplish the following:

- Display all data that the control program monitors. This allows for fast monitoring of system status and removes the need for the computer to be connected to monitor the system.

2.1.2 Non-Functional Requirements

Control program:

- When the control computer becomes physically connected to the Arduino, the control program, if running or once initialized, must recognize and initialize communication with the Arduino without user interaction.
- The control program must allow the user to complete all functional requirements reliably and without crashing.

Status monitor:

- The computer and mounted display should be able to run 24/7
- In the event the computer restarts, the status monitor program should open and begin monitoring again without user interaction.

Arduino and Shield:

- The controller must be capable of communicating to the OTC's via the RS232 protocol.
- The controller must be capable of communicating with the MFC's, which use an analog 0-5V range for getting and setting flow rate values. This requires our system to have both ADC and DAC for 7 MFC's.

Ease of use

- The users of the lab, whether TA or student, should be able to easily operate the furnaces via our GUI
- Readouts should be clear and there should be little ambiguity to what furnaces are at what temperature and what MFC is at what flow rate. There should be a unified readout/setpoint menu that shows current system setpoints
- changing system setpoints should be done in another tab, when the user has finished putting in the new setpoints, the program should confirm the users choices when they are done so no mistakes are made
- This means little to no instruction should be needed, the GUI should be made in such a way that someone new to the system would understand the setting they are changing and be able to verify that the actions they are taking our in fact modifying the system.

2.1.3 Standards

IEEE 1028-1998 - IEEE Standard for Software Reviews

This standard concerns only reviews and audits; procedures for determining the need for a review or an audit are not defined and the determination of the results of the review or audit is not specified. This is needed in groups where software is written like ours. We need procedures to follow to review our code to make sure we are following the best practices. To make sure that it will have a long lifetime and will be able to be modified easily if need be.

IEEE 1233-1998 - IEEE Guide for Developing System Requirements Specifications

This standard details the operations, constraints, configurations, of systems and how they can be explicitly specified. It also goes over the software that is sometimes used to represent systems requirements. It talks about laying out the necessary qualities and

characteristics of the work requirements put forth by individuals, what they need to show to management etc.

IEEE C2-1997 - National Electric Safety Code (NESC)

This code goes into how wiring should be implemented in connecting our devices to the 3-phase electrical grid. It goes over the basic security precautions that are needed and why they are needed to protect the devices. It also talks about various types of power supplies and the safety requirements needed when using one or another.

We believe that all 3 of these standards are relevant. The software review aspect is very relevant because we will be steadily adding a lot of various features to our GUI that the user to control. THe system requirements standard is also very relevant. We are creating a system with a set of specifications that must be met. The guidelines show us how best to go about characterizing these specifications. The national electrical safety code is also relevant. We will be connection our devices to the grid and thus must have a good understanding of the risks involved and what protection mechanisms are needed.

2.2 Design Analysis

Our attempts to communicate with the Omega temperature controller with our computer via the RS-232 cable have been successful by following the instructions of the Omega controller User Manual.

With this communication line established we have been able to set different temperature for 6 different zones which using two controllers we will be able to control all 9 different operational zones in the three furnaces. In addition, we are able to tell the controller to read the set temperature value of each zone and the current temperature value of the zones as well allowing us to get up to date data of each and every zone.

For the microcontroller we had to decide which one to implement into our design. It was decided that the Arduino Mega would be the best to use compared to the Arduino Uno for the following reasons:

The Mega has four serial ports instead of the Uno which only has one. This will allow us to connect the microcontroller to the computer and and the two Omega controllers .

The Arduino Uno only has six 10-bit ADC channel which is one short of what we need.

Our group was given a RS-232 shield chip to use which for testing purposes is what we have been using. However we realized that the shield chip that was given might be too large so our group decided to create out own shield chip to save space in the final design.

For DACs our group realized that to match the resolution of the Arduino Mega ADC they would need to be 10-bit DAC. Also they would also need to have 8-channels for the 7 Mass Flow Controllers (MFCs). We found the appropriate chips and ordered them from TI. Once we got the chips we then got break-out boards to make testing easier and attempted to soldered them together. This initially was a challenge as the DAC chips were smaller than what most of us were used to.

Eventually we managed to create a method to solder the pins into the board by using a smaller soldering head pin and also using the solder head to heat up multiple pins on the chip as well.

We got the DAC to successfully communicate with the Arduino allowing us to determine which channel and voltage value we would like to use thus allowing us to control the MFC. Testing was done using a digital multimeter and the results concluded that the device's voltage output was accurate to 0.5% of the specified voltage(anywhere between 0 and 5 volts).

We have created the design for the GUI interface. This design, we believe, is very well suited for the task and we are looking into different libraries and methods for enacting it. As we move forward it is likely that we will make changes to the design due to the constraints of the project, hardware changes, or programming roadblocks as they come up. As for what has been done to this point, the GUI is not fully prototyped. Through research and following some tutorials, we have been able to learn a lot about Tkinter, the library we are using to create the GUI, and have made many test programs with it. We are planning to begin true prototyping as soon as possible.

3. Testing and Implementation

To test the functionality of the system, we will need to test: changing temperature setpoints, changing the gas flow rate, checking the states of furnaces, and observe profiles run from start to finish.

The test cases will include communication between the different hardware pieces of the project and that the furnaces react correctly to the inputs.

Examples:

- Send a command to increase furnace temperature > get status from the temperature controller > get the actual furnace temperature
- send command to lower gas flow > check the voltage level of the mass flow controllers > get the actual flow rate

The results of such tests can only really go two ways: after sending the command, the given piece of hardware either receives the command and performs the operation, or it does not. There are, of course, edge cases involving communication problems or human error that will need to be looked at as well.

The items to be tested include:

- Communication with the Omega Temperature Controllers
- DAC and ADC Tests
- Microcontroller testing
- GUI functional testing

There is some non-functional testing that needs to occur as well. The only major area where is testing is necessary is the GUI, but it is vital to fulfilling the projects goals.

The GUI items that need testing include:

- Effective data display
- Smooth and efficient control usage
- Realtime changes based on user input

The tested that has been completed thus far includes the temperature controller communication, the DAC and ADC tests, and some microcontroller testing.

3.1 Interface Specifications

Our project encompasses many different communication interfaces between the components within our control system. The Arduino Mega is the central hub that connects all of the devices through several different communication interfaces. The computer that has the Python UI in our design will use an HDMI interface to communicate with the display. The user will then use a keyboard and mouse to interact with the display through the computer.

Once settings are adjusted or information is requested from the UI on the display, the computer will communicate through the Arduino Mega through a USB Serial connection. The Arduino Mega is then used to communicate with our two main controllers, the temperature controller and the mass flow controller. For the Arduino to properly communicate with the temperature controller, a MAX3223 RS-232 Transceiver is used. This IC converts TTL Serial to RS-232 Serial and vise-versa. Refer back to Figure 1 for an overview of the system's various interfaces. This transceiver will connect the arduino Mega's serial to the RS232 level serial of the OTC. It will be soldered to the PCB shield and then connected to the TXRX of the Mega. RS232 transceiver is needed because the Mega does not have a differential driver, only 0-5V GPIO.

The interface specification for the DAC is SPI, we are utilizing the Mega's built in SPI hardware. TO meet this interface specification the DAC will be soldered to the PCB shield which is in turn will be wired to the appropriate pins on the Mega: this includes pins for the data, clock, latch-in, and reset. A timing diagram for the SPI communication between the Mega and DAc can be seen below in figure 3. This has been implemented in C code which can be seen in appendix 4.3.1.

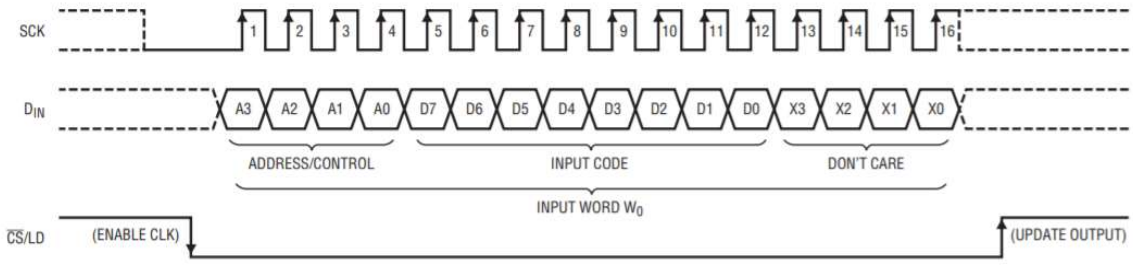

Figure 3: LTC1660 SPI DAC timing diagram

This interface specification allows us to set the 8 outputs of the DAC used in setting the flow rate of the mass flow controllers.

3.2 Hardware and software

Computer

The computer will be used to display the current readings of temperatures and gas flows via a GUI written in python. The user interface will allow for keyboard and mouse (or some other control scheme) entry. This will be sent to a display over HDMI. There will be a serial USB connection between the Mega and computer to allow the GUI to run the Mega

as a slave. Besides the calibration of the system setpoints, temperature profiles will be able to be made.

Arduino Mega

The arduino Mega will be connected to the two OTCs over serial, relaying commands from the RPi to Max3323 to be converted to the RS232 standard. The mega will also have the key function of doing analog voltage reads off the 7 MFCs. It will also be communicating with the DAC to control the MFCs via a SPI interface. The Mega also will be taking in any analog user input including buttons and analog switches. Code will developed for the Mega using the Arduino IDE. We will need to use some more advanced features when we start dealing with the DAC, as this will not be using the 2-channel DAC present on the arduino. Analog reads will be also done, but the ADC for this is quite easy to use via Analogread().

LTC 1660 DAC

The DAC block performs the function of setting the flow rate for the 7 MFCs. This DAC will be soldered to the PCB shield which is in turn will be wired to the appropriate pins on the Mega: this includes pins for the data, clock, latch-in, and reset.

RS232 Transceiver

This block will connect the arduino Mega's serial to the RS232 level serial of the OTC. It will be soldered to the PCB shield and then connected to the TXRX of the Mega. There will also be some capacitors soldered around this to satisfy the timing requirements of the transceiver..

Arduino PCB Shield

Further along in our project after we have completed the prototyping we will develop a Arduino shield PCB. This PCB will contain the two RS232 transceiver chip and the 8-channel DAC. It will also contain the needed connectors to hook on top of the Mega and connect the MFCs and OTCs. This shield will be created using Eagle PCB software. The parts we will be using are readily available on ultralibrarian and snapeda. The design of the PCB will follow the following process:

1. Schematic creation- wire the circuit with components
2. place components footprints on PCB
3. route the traces

We will be assembling the PCB with parts found online using surface mount components

Software

The software for our project will be coded using python and the arduino IDE. The python element of our project will be using Tkinter for creating a GUI similar to the one made by

Omega. It needs to effectively show the various states of the furnaces, allow for changing settings, change temperatures and gas flow rates, and prevent the user from entering something erroneously and causing any kind of dangerous situations. The layout of the GUI's main window will probably look very similar to Omega's. This python GUI will interface with the Mega over usb serial via an API. The Arduino Mega Software will contain code to transform the users commands into the appropriate calibration of the MFCs, parsing the information based on our instruction encoding. This arduino code can be seen in appendix 4.3.1.

3.3 Functional Testing

Unit Testing

DAC and ADC functional testing is purely composed of their ability to convert digital signals into a given voltage and vice versa. A given digital signal needs to have a corresponding voltage. We will need to test and make sure it can produce such a thing in the first place and then make sure the output voltage or digital signal is correct.

The OTC Python API functions will be tested by

GUI testing can be performed in two sections. The major testing will be just using it; we will test and make sure all the buttons and boxes, when clicked, typed in, etc., have the correct background and front end behavior. As for actual unit tests for the GUI, the only real tests to be done will be to make sure the functions that are used by the buttons or other controls output and call the correct things.

Integration Testing

The ability for each component to properly send and receive the correct data is vital to the overall operation of our control system. Testing the integration of the OTC into our system was done through several different steps. First we tested the RS 232 communication with the OTC directly and tested each send and receive signal. We did this by using a computer and PuTTY to send and receive the RS 232 data. Then we sent codes to set temperature, read temperature, and read set temperature. These three test codes all provided positive results. Next, we used those results to test our serial to RS 232 converter. We ran the same test but sent serial data through the Arduino Mega to the converter instead of directly sending RS 232 to the OTC. These tests also passed, thus proving the proper technical ability to integrate the OTC into our system.

The ability to integrate the MFCs into our system also needs to be tested. The Arduino Mega being used in our system does not have enough integrated DACs to communicate with 7 different MFCs. Therefore, we will use a seperate, external DAC IC chip to communicate with the MFCs. The current mass flow controllers are controlled through an input between 1 and 5, so the testing of the external DAC will judge its ability to correctly output the necessary voltage value when given a digital signal from the Arduino Mega. Unfortunately, we can not test the DAC with the actual MFCs because the Microelectronics Research Center has not yet decided on purchasing new ones.

Acceptance Testing

We will have users test the control systems interface. These users are researchers, professors, EE 432 students and TA's, and Dr. Tuttle (our advisor/client). We will have these users operate the furnace using our new control system and ask them for feedback. This feedback will then be taken into consideration and we will redevelop our control system using our new feedback.

Additionally, our control system will have a timed and ramped feature that allows users to have the system run autonomously. This, along with the accuracy of our temperature setting will be tested by performing EE 432 lab projects and see if the desired outcomes are achieved.

3.4 Non-Functional Testing

Performance

The status monitor and the computer have to remain on and be constantly updating. Testing this could be done by setting the computer to be on 24/7 and then check up on it every couple hours to see if there were any errors that occured that perhaps shut off the monitor and/or computer.

Security

As security requirements are added we will add testing and programs that allow us to meet those requirements.

Usability

We plan on making our GUI extremely user friendly. This entails simplifying the commands to change the furnace temperature to checking the status of the MFC. We would test this by setting the steps to initiate certain commands and then check if they did what we wanted them to do. If not we would just have to go back into the GUI code and alter/test it over and over again until we got the right results

Compatibility

Our team had to make sure the OTC was able to connect with a computer using an old RS232 cord. Testing consisted of reading the user manual for the OTC and following the direction stated in it.

The correct microcontroller must be chosen that can be hooked up to seven MFC's and two OTC's which made the Arduino Mega a very viable choice.

The DAC's must be compatible with the MFCs. We currently do not have access to an MFC, but we have verified that the DAC's can be programmed to set the range of DC voltages required by the MFC, 0-5V.

The communication between the OTC and our user interface was tested by sending commands to the controller through an RS232 cable. We could then see if those commands were accepted and followed by sending another command to verify changed data. The Python API for communication between GUI and OTC was written with the Arduino programmed as a serial passthrough between a computer and the OTC. API functions could be tested by running a test script from the command line and asserting the output.

The mass flow rate meters are controlled by using simple, analog voltages. This will make our testing for our controller interface pretty simple. We will adjust settings on the interface, then using

a multimeter, check and make sure that our voltage outputs are the same as the desired voltage outputs that correlate with our settings on the controller interface.

3.5 Process

The process we are using for testing our project is dependent mostly on which part of the system is being tested. The Figure 4 shows the overall process for testing the various pieces of the project separately and together.

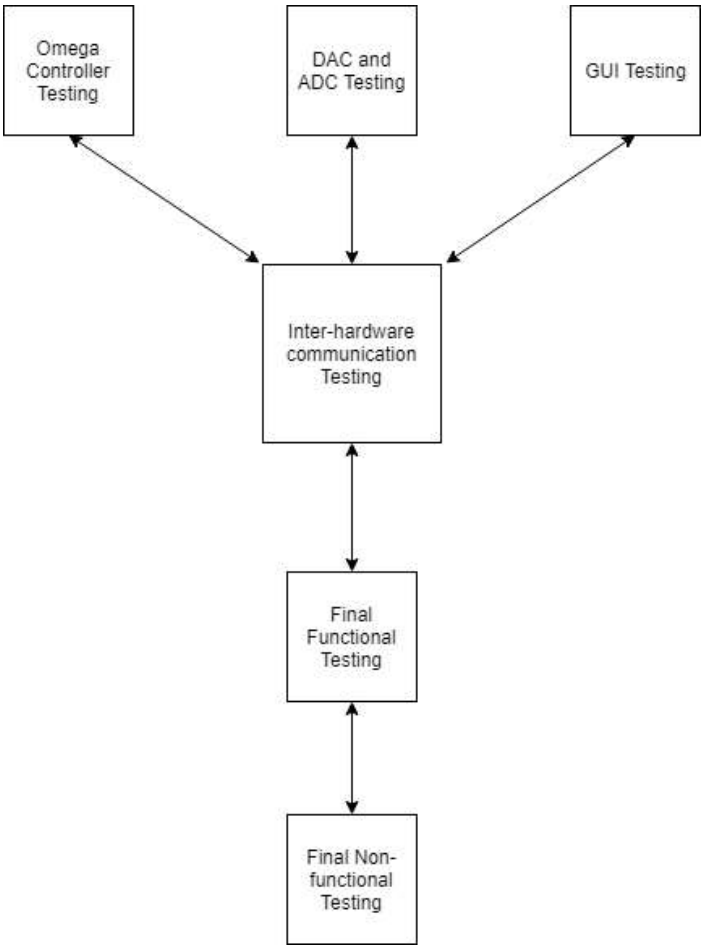

Figure 4. Flowchart showing the testing process.

For communicating with the Omega temperature controller all the testing that was required was to just read the Omega User Manual and to follow it's instruction on how to send and receive information to and from the OTC. We then connected the controller to a computer and (utilizing PuTTY) we tested how to send it with trial and error to make sure we could always extract the necessary data and change whatever settings and temperatures we needed to.

The DAC was tested by one of our group members wrote code for the Arduino Mega and then when the code was debugged they selected a value for which the DAC would output and then a digital multimeter was hooked up to the output to test the accuracy of the actual value over the expected value which resulted in an accuracy of 0.5% of the targeted value

3.6 Results

3.6.1 Successes

We have succeeded in communicating with the OTC that our client has given us to test our implementation. We have sent commands to set and read the temperature for each channel on the OTC. We have also been able to run many other commands that are present on the OTC datasheet like changing the ID and temperature alarms. This was tested via the serial monitor within the arduino IDE from a computer to the Arduino Mega via USB, and a Python API was written to facilitate communication with a GUI.

The LTC 1660 DAC has been successfully implemented into the arduino mega code. This required setting up SPI communication on the Mega and correctly wiring up the IC on a breadboard. We verified that each of the 8 channels could be set by user input over the Arduino IDE serial monitor using a digital multimeter.

The Arduino Mega's internal ADC was successfully integrated into the main code. We verified that a user could send a command ('R') and receive back a string containing the analog voltages on the Mega's analog pins using a variable voltage source.

Another Success was the overall structure of the arduino code we developed (see appendix 4.3.1). It is structured such that there should not be anything unpredictable happening. The code waits until the serial port receives the newline character, which will always come at the end of the command. Then it parses the user input based on the first character, then does one of three functions: relays the data to the OTC, reads the ADC, or sets the DAC. We tested all 3 functionalities at once and everything was working together just as expected.

3.6.2 Challenges

We performed the temperature controller tests using a computer hooked up to an extra controller and setup to received and send information to the controller. We used PuTTY to echo back the controllers responses to our commands. We initially encountered some confusion because the documentation for the communication to the controller was written in a confusing manner. However, after some trial and error we were able to understand the whole protocol. This was due to some unexpected data being sent back to us when setting values on the OTC, this is not in the documentation for the OTC so we will have to flush the serial buffer after these types of commands to account for this.

Another challenge we faced in encoding the commands for the Mega to parse. We would need to find a way that the Arduino would only execute commands given to it with no error involved. To do this we decided on using characters that would distinguish each of the 4 commands available that the arduino does. L signifies an OTC pass through, no action besides relaying the info to the RS232 transceiver is done. R signifies an ADC read, reading the 7 ADC channels and relaying the info back to the GUI. S is for set, meaning the DAC will be set with a to a specified voltage for a given channel. All of these commands end with the newline character, which the Mega will wait for before it executes any command. This ensures that no commands execute until all data is received, meaning we will not have any incomplete commands. The parsing method we have chosen can be seen in detail in appendix 4.3.1.

Another challenges we will face is ensuring that our solution can be successfully integrated into the system. We will be removing all of the current manual controls for the furnace leaving only digital

controls. We do not have a schematic (our client does not have one to give us) for the current system (the MFC wiring), so we are unsure of what we will be removing and what will be staying put. Our biggest question is still the power supplies for the MFCs. Without schematics or specifications we don't have a way of confidently integrating them into our system. We understand how to power the MFCs given by the datasheet specifications so we should be able to keep the current power supply. The current manual controls for MFC can be completely scrapped as we are implementing a digital solution.

Some of the more minor issues we have faced along the way with our project include:

- finding an easy to use and readily available GUI library and tools, as well as documentation for it
- creating a test program for the DAC/ADC
- having too few serial ports on our microcontroller

3.6.3 Modeling & Simulation

For modeling our system, our client gave us an OTC to perform a full range of testing on. This includes reading & setting temperature, calibrating alarms, setting ID, and any other function available to us on the OTC data sheet. This allows us to confidently say that our modeling outside of the real system will be easily integrated.

Modelling the mass flow controllers was done through testing the LTC1660 DAC and Mega ADCs with multimeters and power supplies. This was done because our client did not have a mass flow controller to give us (and everything else that is needed to go along with it like a gas tank, power supply, and gas tubing). The result of testing can be seen in the two video links below. We our confident our modeling of the MFC will translate well into the real system, as it controlled very simply by applying and reading analog voltages. The tests show us entering commands into the serial monitor and watching an output or serial response from the DAC.

ADC Demo - https://drive.google.com/file/d/1s41MxXiI9wN7UJzD7piw7pomQIcKS1Hc/view?usp=sharing

This test was completed by connecting analog pin one to a voltage source. We then used the serial monitor within the arduino IDE on a computer to send a command to the Mega, 'R. The arduino parses this command as a request for ADC data, reads the first 8 analog pins and relays this information back to the serial monitor. The first 4 numbers in this video represent analog pin 0. We can see it varies from 0 to 1023 (2^10-1) depending on what the voltage on the power supply is set to. The rest of the pins are floating so there values are undefined. This will transfer directly to the MFC in the real system when we get the MFCs wires routed to the DAC. This test shows only one channel, but it does work well for all channels.

DAC Demo - https://drive.google.com/file/d/1p8Y_GyOjyHt6NCGHbjEiqqzc4lyTuMX6/view?usp=sharing

This test was completed by connection the pin 1 of the LTC1660 DAC (an image of this can be seen in appendix 4.3.2) to a digital multimeter. Again, the arduino IDE serial monitor was used to send a command that the Mega would parse and then complete the instruction based on our instruction encoding. Our encoding for the DAC follows Sbc.ef where S signifies a set command, b is the dac

channel (1-8), and c.ef is the floating point value we want to set on the DAC. This works for all 8 channels of the DAC.

4 Closing Material

4.1 Conclusion

So far in our project, we have accomplished much of the embedded systems programming and testing to ensure that our solution will successfully control the furnace system. We have confirmed that our hardware will in fact respond to user input for setting and reading analog voltages 0 - 5V, which is what is exactly required to work with the MFC. The OTC serial communication has also been thoroughly implemented over the arduino Mega & RS232 transceiver, with the OTC responding to all commands we have given it. This work aligns well with the goals of this project, *to implement a new furnace control system that is less prone to user error and allows for a more unified system of controlling and viewing the system setpoints and readouts.*

Our solution surpasses all other methods we have considered because we will be able to provide a highly functional and integrated user experience for the furnace system. With a single arduino, it would be much more difficult to integrate software functionality like temperature profiling and make a user experience that is easy to navigate. Using a python GUI library like tkinter allows us to create this GUI in such a way that the system will be much less prone to user error and misinput compared with the existing system.

Our plan of action for next semester is threefold: to complete the GUI and API in python, to design and fabricate an arduino PCB shield with the needed ICs and connectors, and then install our solution into the system. This will keep us quite busy in 492, especially on the software end where some of the functionality we wish to implement (like temperature profiling) will not be a simple task.

4.2 References

1. *Lecture Number - 23: Oxidation*, Electronic Materials, Devices and Fabrication, Dr. S. Parasuraman, Department of Metallurgical and Materials Engineering: Indian Institute of Technology, Madras, 2012

4.3 Appendices

4.3.1 Arduino Mega main code

```
// inslude the SPI library:
#include <SPI.h>

void dacWrite(int val);
void clearDac(void); //resets all output to zero
void dacRead(char * buf);

// set pin 10 as the slave select for the digital pot:
const int slaveSelectPin = 53;
const int CLR = 7;

void setup() {

  Serial.begin(4800);
  Serial1.begin(4800);
//  pinMode(13,OUTPUT); LED test pins
//  digitalWrite(13,LOW);

   // set the slaveSelectPin as an output:
  pinMode(slaveSelectPin, OUTPUT);
  // initialize SPI:
  SPI.begin();

  pinMode(CLR, OUTPUT);
  digitalWrite(CLR,HIGH);
}

void loop() {

  //dac variables
  float userVal;
  unsigned int val;
  unsigned int channel;

  //serial
  byte incomingByte = 0;
  char incomingString[256];
  int numBytes;

  char mfcReads[33];

  // USB
  if (Serial.available()) {

    // Read bytes until newline character is reached, excluding newline character
    numBytes = Serial.readBytesUntil('\n',incomingString,243);

    // If command is meant for OTC
    if (incomingString[0] == 'L') {

      // Write contents of buffer to OTC
      Serial1.write(incomingString,numBytes);

    }

    //DAC set for channel x @ a.bcd volts =  Dxa.bcd
    else if (incomingString[0] == 'S'){ //S for set DAC

      channel = incomingString[1];
      incomingString[numBytes] = '\n'; //ad newline so we can use easy parser
      userVal = atof(incomingString + 2); //parse a.bcd string

      val = userVal*1023/5 ;
      val = (channel << 12) | (val << 2) ;

      dacWrite(val);

      }

```

```
70      else if (incomingString[0] == 'R'){
71
72        dacRead(mfcReads);
73        Serial.write(mfcReads,32);
74
75
76      }
77
78      else if (incomingString[0] == 'C'){
79
80        clearDac();
81
82      }
83
84    }
85
86    // OTC
87    if (Serial1.available()) {
88
89      // Read byte from OTC
90      incomingByte = Serial1.read();
91      // Write byte to USB
92      Serial.write(incomingByte);
93
94    }
95  }
96
97  void dacWrite(unsigned int val) {
98
99    // take the SS pin low to select the chip:
100   digitalWrite(slaveSelectPin, LOW);
101   delay(10);
102   //  send in the address and value via SPI:
103   SPI.transfer16(val);
104   delay(10);
105   // take the SS pin high to de-select the chip:
106   digitalWrite(slaveSelectPin, HIGH);
107
108 }
109 void dacRead(char * buf){
110
111   int reads[8];
112
113   for (int i = 0;i<8;i++){
114     reads[i] = analogRead(i);
115     }
116
117

      sprintf(buf,"%04d%04d%04d%04d%04d%04d%04d%04d",reads[0],reads[1],reads[2],reads[3],read
      s[4],reads[5],reads[6],reads[7]);
118
119 }
120
121
122 void clearDac(void){
123
124   digitalWrite(CLR,LOW);
125   digitalWrite(CLR,HIGH);
126
127 }
128
```

4.3.2 LTC1660 Block Diagram