

# NSF Lab Furnace Control System

FINAL REPORT

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>

# Table of Contents

<b>0. Executive Summary</b>	<b>2</b>
<b>1. Requirement Specification</b>	<b>3</b>
1.1 Functional Requirements	3
1.2 Non-Functional Requirements	4
1.3 Intended Users and Intended Uses	5
1.4 Assumptions and Limitations	5
1.5 Expected End Product and Other Deliverables	6
<b>2. System Design and Development</b>	<b>7</b>
2.1 Design Plan	7
2.2 Standards	8
2.3 Interface	9
<b>3. Implementation</b>	<b>9</b>
3.1 Hardware and software	10
<b>4. Testing, Validation, and Evaluation</b>	<b>12</b>
4.1 Functional Testing	12
4.2 Non-Functional Testing	13
4.3 Process	14
4.4 Results	15
4.4.1 Successes	15
4.4.2 Challenges	16
4.4.3 Modeling & Simulation	17
<b>5. Project and Risk management</b>	<b>17</b>
<b>6. Conclusions</b>	<b>19</b>
6.1 References	19
6.3 Appendices	21
6.3.1 Arduino Mega main code	21
6.3.2 LTC1660 Block Diagram	23

## 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

## o. Executive Summary

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 have implemented allows both temperature control and gas flow control with a microcontroller. This microcontroller is able to set and read temperature

from the temperature controller. It can also 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. 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, allows the user to navigate through and control each of the 4 furnaces.

## 1. Requirement Specification

### 1.1 FUNCTIONAL REQUIREMENTS

A computer program that has to accomplish the following:

Temperature control - For each zone in each furnace the system allows the user to:

- set temperature setpoints

Temperature monitoring - For each zone the system allows the user to:

- view current temperature
- view temperature setpoints

Gas flow control - For each gas in each furnace the user must be able to:

- set gas flow rate

Gas flow monitoring - For each gas in each furnace the user must be able to:

- view current gas flow rate

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

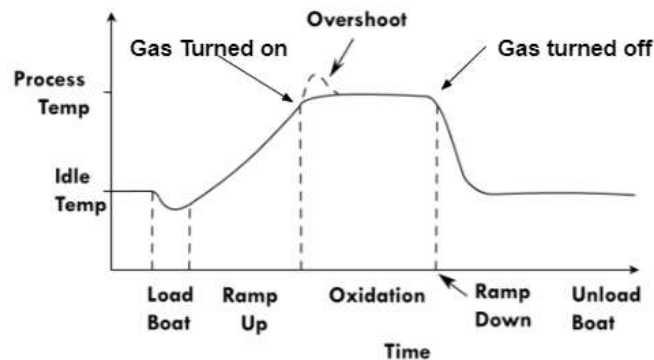


Figure 2. Timing diagram showing the furnace's active ramping cycle.



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

- Display the 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.

## 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 remain open and continue 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 with 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 is in fact modifying the system.

### 1.3 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.

### 1.4 ASSUMPTIONS AND LIMITATIONS

Assumptions:

- We will not need to do any PID tuning of the OTC's.
- 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 the end of the Spring 2019 semester, so we cannot do final testing of our product until then

### 1.5 EXPECTED END PRODUCT AND OTHER DELIVERABLES

The end product that we are delivering to our client is a user friendly interface to control the OTC's and MFC's. This has been accomplished with a GUI and attached computer, an Arduino Mega 2560, and an Arduino shield and Python API of our own design. One interface is a continuously running status readout running on a the arduino and displayed on a screen that can be mounted to the furnace equipment, and the other runs on a laptop and will be used to control the furnace settings. Both use a serial interface to communicate with the Arduino that intelligently passes commands to and reads results from the OTC's and MFC's. In order to successfully communicate with these devices, the

shield for the Arduino includes a digital-to-analog converter (DAC) and an RS232 transceiver, as well as the necessary cable connectors.

The GUI includes temperature settings and readouts. It also has 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  $\pm .2\%$  or  $\pm 2^{\circ}\text{C}$ . 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.

The user interface as a whole is very intuitive and most users will not need documentation to figure out the majority of controls. Additionally, the Design Document and Project Plan are 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.

## 2. System Design and Development

### 2.1 DESIGN PLAN

Our design connects the MFC's and OTC's to an Arduino Mega. This Arduino Mega is controlled by a Python GUI running off a computer. This allows 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.

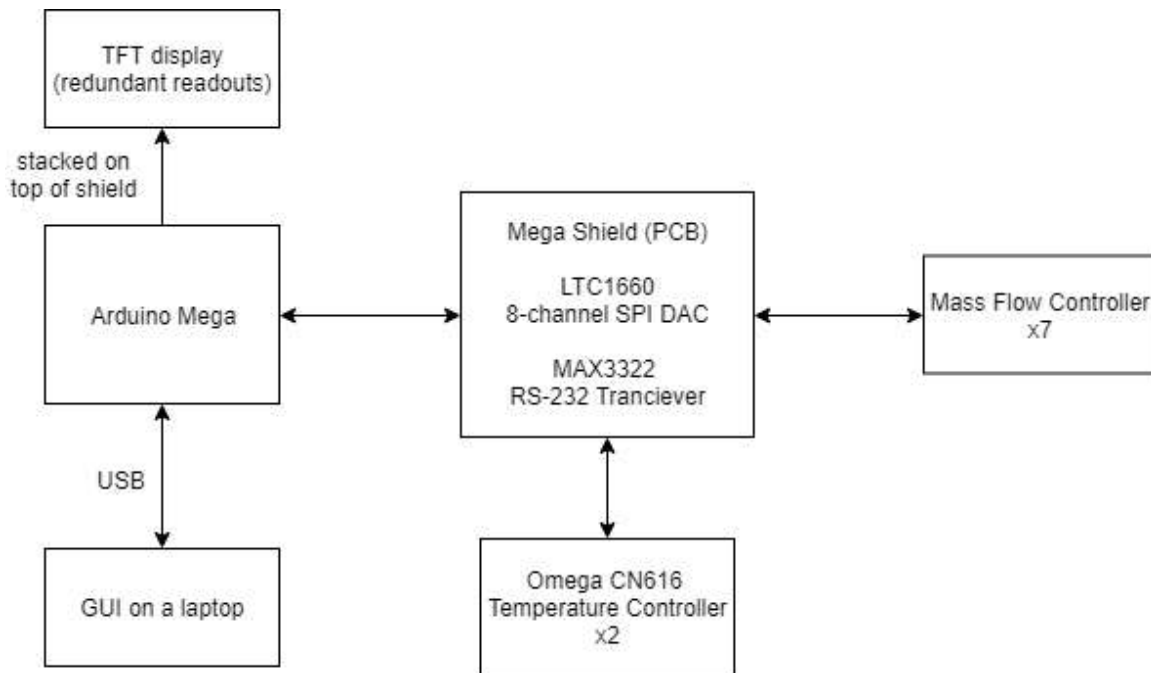


Figure 1. Block diagram showing the major parts of the project.

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 have 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.

### 2.11 DESIGN OBJECTIVES, SYSTEM CONSTRAINTS, AND DESIGN TRADE OFFS

Of course our design objectives were to interface hardware and software components together to create a better system to control both the temperature of the furnace and the flow of gasses into the furnace pipes. one of the biggest system constraints we had were the MFC could only output and read 0-5 Volts off of them so we had to find a way to allow the

computer to communicate with the MFC. Design trade offs included the use of an arduino mega which was more expensive but had the correct amount of pins so we would only need to buy one mega instead of two uno's. We also decided to have the GUI on an actual computer/laptop because a laptop has a larger screen plus it has an independent power source.

## 2.2 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.

## 2.3 INTERFACES

The interface is a very important aspect of our project. It is the gateway used by students, TA's, and professors to access the many important functions of the furnaces, OTC's, and MFC's. The simpler and more user friendly it is, the more productive lab time will be for students. Our control system has two main interfaces. First, the GUI, which is on a laptop

and allows the user to input and read several different settings to the MFC and OTC, and second we have a small always-on display that is directly connected to the arduino and displays current readout measurements and settings 24/7.

The GUI on the laptop consists of several different tabs that bring the user to different pages with different settings. The main home screen, Figure 2, displays the current settings and current values for each furnace, zone, and mass flow controller.



Figure 2: GUI Settings Display Page

Additionally, the laptop GUI has pages to set each of the values shown in Figure 2. It also has a profiles page which allows the user to choose the amount of steps (up to twenty) that the furnace will take. Then the temperature, slope, and time values of each step can be adjusted to the users needs. The profiles page is shown in Figure 3.

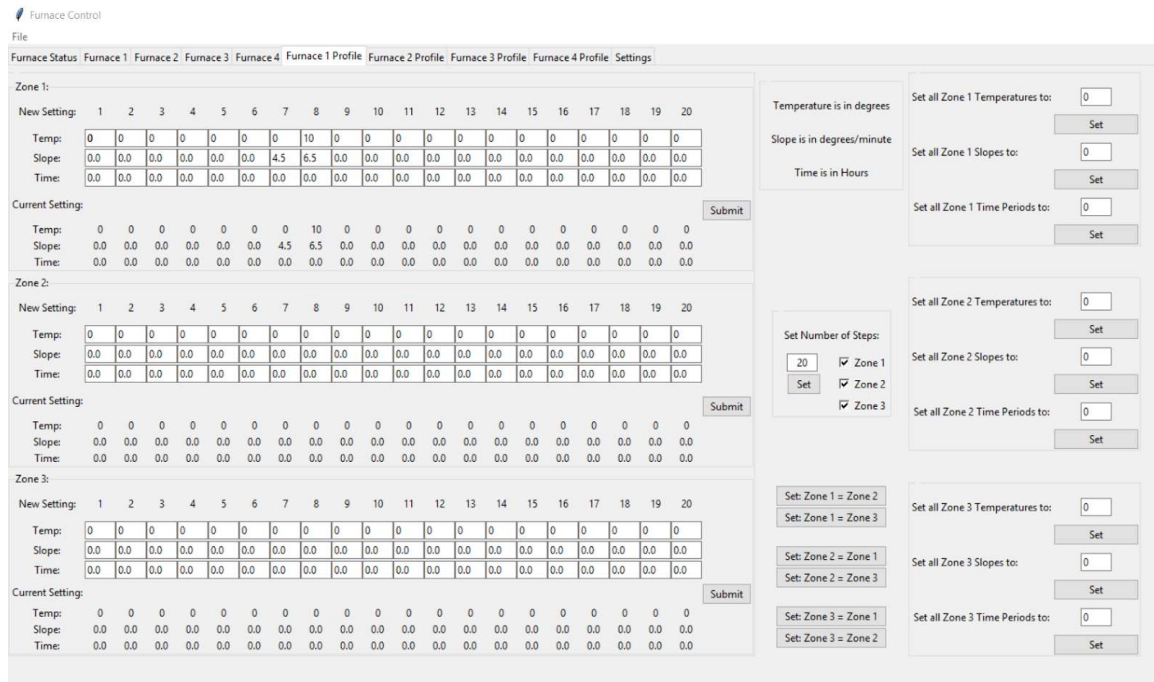


Figure 3: GUI Profiles Page

The 24/7 readout display that is directly connected to the Arduino simply shows the current reading and current setting of each furnace zone and MFC.

### 3. Implementation

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 our 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.

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 then uses 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 communicates 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 vice-versa. Refer back to Figure 1 for an overview of the system's various interfaces. This transceiver connects the Arduino Mega's serial to the RS232 level serial of the OTC. It is soldered to the PCB shield and then



connected to the TXRX of the Mega. The 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 is 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 4. This has been implemented in C code which can be seen in appendix 4.3.1.

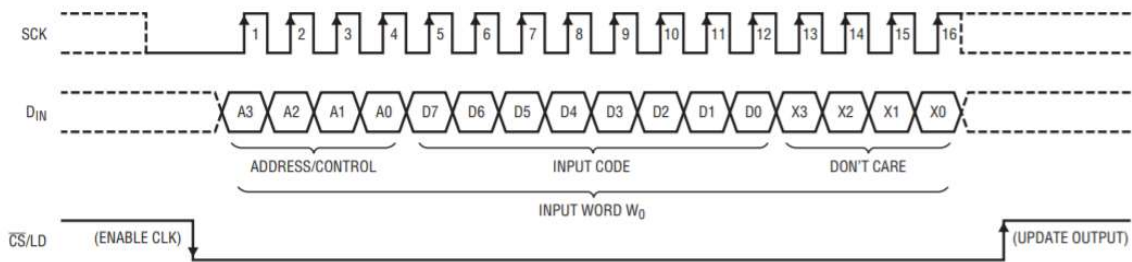


Figure 4: LTC1660 SPI DAC timing diagram

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

### 3.1 HARDWARE AND SOFTWARE

#### Computer

The computer is used to display the current readings of temperatures and gas flows via a GUI written in python. The user interface allows for keyboard and mouse entry. This is sent to a display over HDMI. There is also 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 are able to be made. Initially we were planning on using a raspberry pi to create the GUI but decided the bigger screen and power independence of a laptop would be better for this project.

#### Arduino Mega

The arduino Mega is connected to the two OTCs over serial, relaying commands from the RPi to Max3323 to be converted to the RS232 standard. The mega also has the key function of doing analog voltage reads off the 7 MFCs. It also communicates with the DAC to control the MFCs via a SPI interface. The Mega also is taking in any analog user input including buttons and analog switches. Code was developed for the Mega using the Arduino IDE. We needed 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

are also done, but the ADC for this is quite easy to use via Analogread(). The arduino mega was picked over the arduino uno because the latter did not have the correct amount of pins for the MFC so if we wanted to use an arduino uno we would have to have bought two of them and have them work in sync.

### **LTC 1660 DAC**

The DAC block performs the function of setting the flow rate for the 7 MFCs. This DAC is 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 connects the arduino Mega's serial to the RS232 level serial of the OTC. It is soldered to the PCB shield and then connected to the TXRX of the Mega. There are also some capacitors soldered around this to satisfy the timing requirements of the transceiver..

### **Arduino PCB Shield**

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

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

We assembled the PCB with parts found online using surface mount components

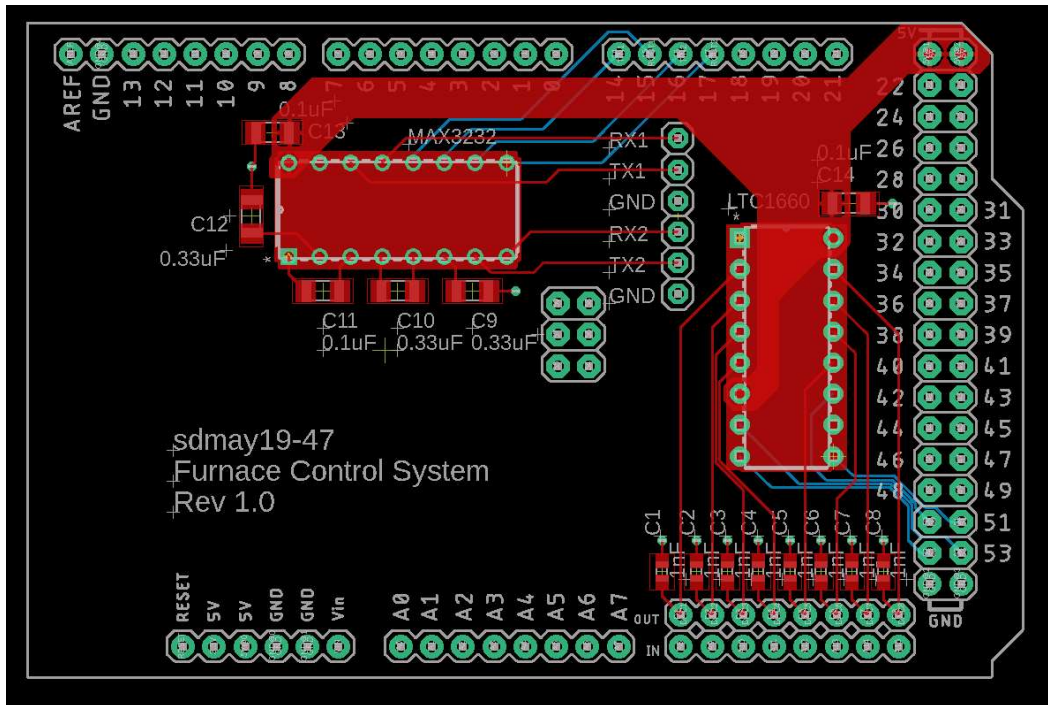


Figure 5: PCB Shield Schematic

Above is the result. This is a two layer board, the top for power and signals and the bottom is ground plane and a few extra signals. The large red trace is the 5V copper pour. The blue traces are routed on the ground plane. There are output caps on each of the DAC outputs that are not currently filled, but our client can solder on if he wishes (if the output is not stable).

## Software

The software for our project is coded using python and the arduino IDE. The python element of our project uses Tkinter for creating a GUI similar to the one made by Omega. It effectively shows the various states of the furnaces, allows for changing settings, changing temperatures and gas flow rates, and helps prevent the user from entering something erroneously, preventing some dangerous situations. The layout of the GUI's main window looks somewhat similar to Omega's. It also has a profiling capability which allows the user to set time, slope, and temperature over "steps." This python GUI interfaces with the Mega over USB serial via an API. The Arduino Mega Software contains 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.

## 4. Testing, Validation, and Evaluation

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

The test cases 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 has occurred as well. The only major area where this 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

We have tested all of these different areas so at least a passable degree and in most cases even more exhaustively.

### 4.1 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 are tested by multiple test cases to ensure that sending given messages and commands will trigger appropriate changes to data and send corresponding messages as well.

GUI testing can be performed in two sections. The major testing is just using it; we 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 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 has been tested. The Arduino Mega being used in our system does not have enough integrated DACs to communicate with 7 different MFCs. Therefore, we use a separate, 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). Unfortunately, as the furnaces have not been available for true testing, this may have to be done after project completion.

Additionally, our control system has 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.

## **4.2 NON-FUNCTIONAL TESTING**

### **Performance**

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

### **Security**

No real security requirements have been given to us as the furnaces and the system will be in a closed lab.

### **Usability**

We have made our GUI extremely user friendly. This entails simplifying the commands to change the furnace temperature to checking the status of the MFC. We have tested 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.

### 4.3 PROCESS

The process we are using for testing our project is dependent mostly on which part of the system is being tested. The Figure 6 shows the overall process for testing the various pieces of the project separately and together.

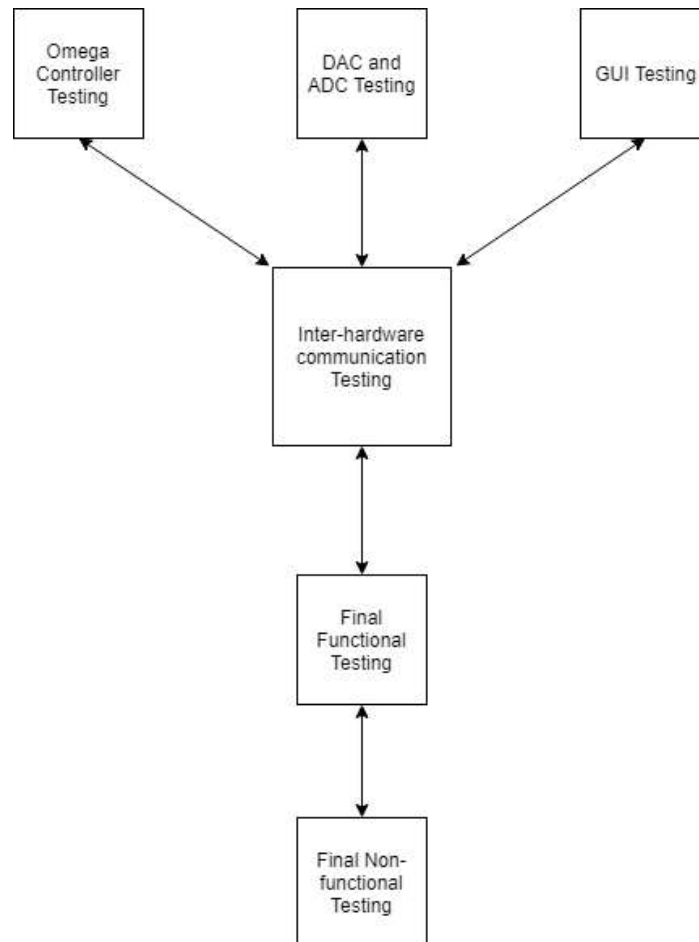


Figure 6: Flowchart showing the testing process.

For communicating with the Omega temperature controller all the testing that was required was to just read the Omega User Manual and to follow its 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

## 4.4 RESULTS

### 4.4.1 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.

### 4.4.2 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



do not have any incomplete commands. The parsing method we have chosen can be seen in detail in appendix 4.3.1.

Another challenge 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

#### 4.4.3 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 are 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 their 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](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.

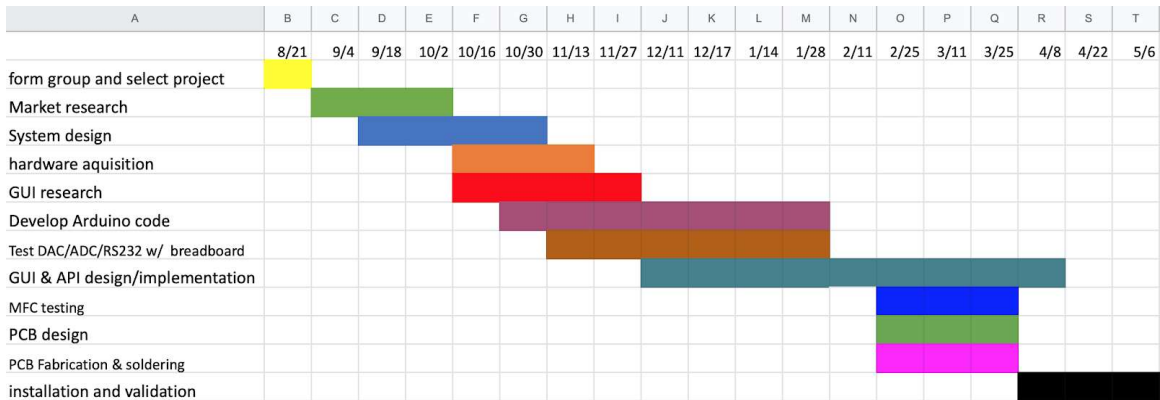
## 5. Project and Risk management

### 5.1 TASK DECOMPOSITION & ROLES AND RESPONSIBILITIES

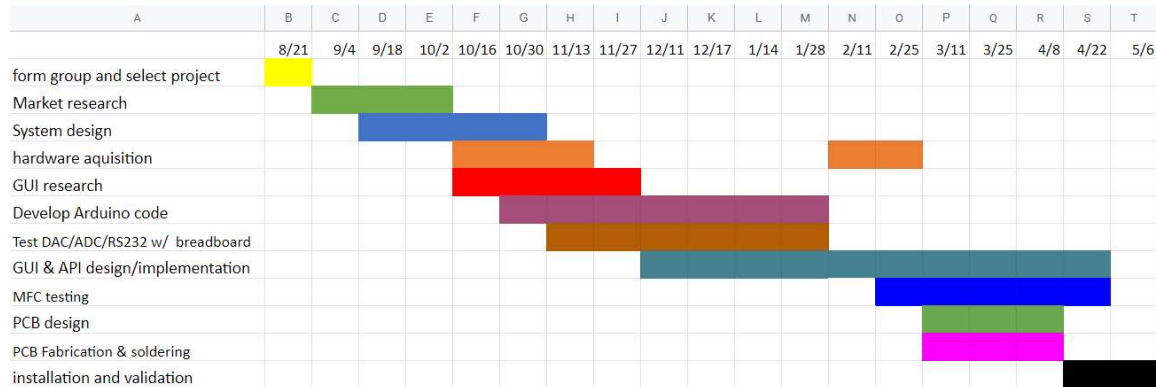
This project had both hardware and software aspects to it, but the software task was much more substantial. For this reason we had the majority of our team (Jeremy, Adam, and Chris) working on the GUI and API. Nick and Kevin were tasked with doing the hardware design and soldering jobs.

Further subdividing the work, Chris was head GUI lead, with Jeremy working with him. Jeremy was in charge of making the profile settings. Adam made the API which talked with the Arduino for the GUI.

On the Hardware side, Nick was in charge of PCB design with Kevin assisting him. Kevin did the majority of soldering once the board got in.



## Proposed Project Schedule



## Actual Project Schedule

As you can see, we stayed fairly true to our original GANTT chart

### 5.2 RISKS AND MITIGATION

For potential risks, we were worried that the OTC would not be able to communicate fast enough with the arduino. For this reason we made the refresh rate 5 seconds, allowing enough time to communicate. What actually happened was that we could not get reliable comms even with a high screen refresh rate. We had to revert the GUI refresh rate to a user click, instead of happening all the time. We also had to change the structure of the arduino code, because reading from the OTC at 4800 baud was very slow. We added buffers to store all data displayed, whereas originally we were only storing the DAC outputs on the arduino.

### 5.3 LESSONS LEARNED

We learned some extremely important lessons in this project. First off all, do as much testing as possible. We ended up not having as much time as we would have liked for testing and this caused us to rush some of the “features” in the end. There were some last minutes modifications that were done to the PCB where we had to solder on a capacitor and some wires. If we would have had more time we could have made a second revision for the PCB with these changes. These problems were due to some unintended arduino quirks.

The first last minute change was due to the fact when the serial port is opened on the arduino, the arduino resets. This is not desired as if the DAC outputs are set (there a gas

flow going) they will reset to zero when the GUI connects. This can be fixed by connecting a 100µF cap between reset and ground. We soldered this cap under the bottom of the shield.

The second last minute change was necessary after we added the TFT screen. When the screen refreshes the CLR pin for the DAC does not maintain its assumed constant 5V output. If this goes below the falling edge value the dac will reset to zero. This was fixed by connecting the CLR pin to VCC instead of an output of the arduino, using a wire we solder the pins together. This essentially loses the functionality of the CLR pin (easy reset) but we were able to make our own instead.

Another thing we learned was that good team communication is very important. When working on a hardware/software project, there is a lot of room for error. Making sure that each team member knows their responsibilities well, and where they fit into the grand scheme of things is necessary for project success. The last thing we learned was that brainstorming with others is very important, because when one person goes and does something on their own without the input of the entire team.

## 6. Conclusions

In our project, we have accomplished 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.

We have accomplished much this semester. The software team finished the GUI and API in python. The hardware team finished the design and soldering of an arduino PCB shield with the needed ICs and connectors.. Finally, much integration testing was done to make sure that our hardware and software play nicely together, while providing the best functionality to the user as possible.

## 6.1 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

## 6.2 Team Information

Christopher Pohlen: Senior in Computer Engineering and Psychology,  
cmpohlen@iastate.edu

Jeremy Hartl: Senior in Electrical Engineering, jmhartl@iastate.edu

Lingkai Lang: Senior in Electrical Engineering, llang@iastate.edu

Adam Matthews: Senior in Computer Engineering and Spanish, ajrmatt@iastate.edu

Nick Brylski: Concurrent M.Eng in Electrical Engineering, nbrylski@iastate.edu

## 6.3 APPENDICES

### 6.3.1 Arduino Mega main code

```
1 // include the SPI library:
2 #include <SPI.h>
3
4 void dacWrite(int val);
5 void clearDac(void); //resets all output to zero
6 void dacRead(char * buf);
7
8 // set pin 10 as the slave select for the digital pot:
9 const int slaveSelectPin = 53;
10 const int CLR = 7;
11
12 void setup() {
13
14     Serial.begin(4800);
15     Serial1.begin(4800);
16     // pinMode(13,OUTPUT); LED test pins
17     // digitalWrite(13,LOW);
18
19     // set the slaveSelectPin as an output:
20     pinMode(slaveSelectPin, OUTPUT);
21     // initialize SPI:
22     SPI.begin();
23
24     pinMode(CLR, OUTPUT);
25     digitalWrite(CLR,HIGH);
26 }
27
28 void loop() {
29
30     //dac variables
31     float userVal;
32     unsigned int val;
33     unsigned int channel;
34
35     //serial
36     byte incomingByte = 0;
37     char incomingString[256];
38     int numBytes;
39
40     char mfcReads[33];
41
42     // USB
43     if (Serial.available()) {
44
45         // Read bytes until newline character is reached, excluding newline character
46         numBytes = Serial.readBytesUntil('\n',incomingString,243);
47
48         // If command is meant for OTC
49         if (incomingString[0] == 'L') {
50
51             // Write contents of buffer to OTC
52             Serial1.write(incomingString,numBytes);
53
54         }
55
56         //DAC set for channel x @ a.bcd volts = Dxa.bcd
57         else if (incomingString[0] == 'S'){ //S for set DAC
58
59             channel = incomingString[1];
60             incomingString[numBytes] = '\n'; //ad newline so we can use easy parser
61             userVal = atof(incomingString + 2); //parse a.bcd string
62
63             val = userVal*1023/5 ;
64             val = (channel << 12) | (val << 2) ;
65
66             dacWrite(val);
67
68         }
69
70     }
```

```

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

```

### 6.3.2 LTC1660 Block Diagram

## BLOCK DIAGRAM

