Design and Testing of an Electronic Load Board for Power Supply Validation and Verification

by

Justin Moses

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 4, 2014

Keywords: Electronic Load, Embedded Code, DC Load

Approved by

Robert Dean, Chair, Associate Professor of Electrical and Computer Engineering
Stuart Wentworth, Associate Professor of Electrical and Computer Engineering
John Evans, Technology Management Professor of Industrial and Systems Engineering

Abstract

As the need for high efficiency in power supplies increases, so does the need to test and verify the designs of the supplies. To test the design, efficiency, and reliability of a low-voltage, high-current power supply, an accurate and stable electronic load is needed. The suitable electronic load needs to be able to sink a large amount of current at a low voltage, have a selectable resistance, and be reliable over a wide temperature range. The load board also needs to interface with a computer for automated testing to help remove the human element from the experiment to provide consistent results. Many times, especially for a high current sinking load board, the electronic load is expensive. Thus the need to create an electronic load board that meets all of the requirements while being less expensive than a commercial load.

This thesis presents the process of designing, building, and testing an electronic load board for testing low-voltage, high-current power supplies. There are four designs and revisions covered and the differences between them. While the first revision merely added a control board to the first design, revision two was a completely stand-alone device which has undergone extensive testing and calibration to ensure the board's accuracy.

Acknowledgments

The author would like to thank his advisor Dr. Robert Dean for his patience, guidance, and enthusiasm while working on this project. Without him, this project would not have been possible or pursued to the extent it was. The author would also like to thank Dr. Christopher Wilson for his help in the integration of the project into an automated system and guidance while coding. He would also like to thank Stephan Henning for his help in the first design. Finally he would like to thank all of the undergrads who helped solder countless parts onto test boards.

Table of Contents

Abstract ii

Acknowledgments iii

List of Figures vi

List of Tables ix

1 A Background in Load Boards 1

1.1 AC Loads 5

1.2 DC Loads 7

1.2.1 Constant Resistance 8

1.2.2 Constant Current 9

1.2.3 Constant Voltage 10

1.2.4 Constant Power 11

1.3 Heat Dissipation 12

2 Load Board Version 1 15

2.1 Specifications 15

2.2 Design 15

2.3 Results 17

3 Revision One (Daughter Board) 18

3.1 Why add the Control Board? 18

3.2 Specifications 18

3.3 Design 19

3.4 Coding 22

3.5 Results 25

3.6 Design Problems 25

4 Load Board Revision Two (The Combined Board) . . . 26
4.1 specifications . . . 26
4.2 Design . . . 26
4.2.1 Current Distribution . . . 28
4.3 Coding . . . 33
4.4 Calibration . . . 34
5 Power Supply Test Integration . . . 38
6 Conclusion . . . 42
7 Further Research . . . 43
7.1 Load Board Revision Four . . . 43
7.1.1 Revisions . . . 43
7.1.2 Design . . . 44
7.1.3 Coding . . . 44
Bibliography . . . 45
Appendices . . . 48
A Control board version 1 bill of materials . . . 49
B Resistor calculation code . . . 50
C Resistor Values from Calculations . . . 53
D First control board code . . . 54
E Second load board bill of materials . . . 76
F Second load board CAD . . . 79
G Second load board code . . . 90

List of Figures

1.1 A 1260 W resistor load [1] 1

1.2 Hydraulic control load Simulator [2] 3

1.3 BK 8540 DC Load [3] 4

1.4 Agilent N3302A DC Load [4] 4

1.5 AC calibration load set up 6

1.6 Constant resistance load graph 9

1.7 Constant current load graph 10

1.8 Constant voltage load graph 11

1.9 Constant power load graph 12

1.10 Solder joint crack [5] 14

2.1 Version 1 populated board 16

2.2 Version 1 CAD with all layers 16

2.3 Version 1 CAD top copper layer 17

2.4 Version 1 CAD bottom copper layer 17

3.1 A populated control board 20

3.2 Control board CAD design 20

3.3 Control board CAD top layer 21

3.4 Control board CAD bottom layer 21

4.1 Fully populated revision 2 load baord 29

4.2 CAD drawing of version 2 load board 30

4.3 A quarter of the resistor load layout 31

4.4 PCB design toolkit calculations 32

4.5 Current difference 35

4.6 Calibration with Keithley multimeter 36

4.7 Initial current difference 37

5.1 Complete test setup with working load board 39

5.2 Load board setup in testing software 40

5.3 Commands sent and received from the load board during testing 41

5.4 Test data gathered with load board 41

C.1 Calculated resistance values 53

F.1 V2 top copper 80

F.2 V2 second layer 81

F.3 V2 third layer 82

F.4 V2 fourth layer 83

F.5 V2 fifth layer 84

F.6 V2 bottom copper 85

F.7 V2 top left resistor schematic 86

F.8 V2 top right resistor schematic 86

F.9 V2 bottom left resistor schematic 87

F.10 V2 bottom right resistor schematic 87

F.11 V2 current sensor schematic 88

F.12 V2 display schematic 88

F.13 V2 microcontroller and overall schematic 89

List of Tables

A.1 Control Board V1 BOM 49

E.1 Load Board V2 BOM 78

Chapter 1

A Background in Load Boards

Load simulations have been needed since the advent of technology whether it be for simulating a mechanical, thermal, or electronic load. A very basic load is one that is manually controlled and can be set before the power source is turned on to see if the calculated output is achieved, or to calibrate the power source from the controlled load. A good example of this manual load is a large resistor bank with manual switches such as a dummy load for an alternative energy system as seen in Figure 1.1. The resistors are loaded in a rack and according to the amount and how the resistors are hooked up, the researcher can provide a very precise resistance to test with. While the manual load does a good job at holding a specific point for a long time, it is not very good at dynamic changes since it can only change as fast at the operator can move. This causes a large problem when trying to test the transient response of a power source because the time it takes for the source to change may be too fast for the operator to respond to.

Figure 1.1: A 1260 W resistor load [1]

The other category of load is the dynamic load. The dynamic load is built so the load can be programmed to a set profile or programmed to react to the power input. This allows for a complete simulation of the desired load the power supply will be hooked to. When the entire load can be simulated, the research can save both time and money by being able to simulate a real life event without having to buy all the expensive equipment that will be used on the final product and have to use the time to hook up multiple end-units if the final product will have different options for the power supply load. One early example of a dynamic load is one that tests a electro-hydraulic system as described by E. H. Gamble and B. W. Hatten in 1951 [2]. In their system, as seen in Figure 1.2 the item to be tested is connected to the shaft of a hydraulic motor controlled by a computer. The computer reads the angle of the shaft and uses a computer to compute the current torque, velocity, and acceleration values. These are compared to the desired values and fed to a servo controller which limits the amount of oil flowing through the hydraulic motor attached to the end of the test shaft. This allows the operator to control the properties of the test and ensure the test is accurate and can be used to qualify a test item for ratings and production. In this basic form, the hydraulic motor is acting as the mechanical load for the mechanical system as a transistor or resistor act as the electrical load for an electrical system.

Many of the early loads for simulating a load on a electronic power system were developed with the power grid in mind. In the Corless and Aldred simulator, they use an analog computer to allow for analog voltage regulators, testing of prime-mover characteristics, and magnetic circuit saturation[6]. Their load simulator can simulate both a steady-state load and a dynamic load on a power grid such as a faulty power transmission line which opens and closes the circuit to the grid. Like the previous load simulator, a shaft is used to connect the power source (a generator) to the load. Unlike the previous simulator, Corless and Aldred use electrical impedance to control the torque on the shaft and simulate the load. Their dynamic testing was able to reproduce an open in the transmission line, and a subsequent closure of the transmission line circuit.

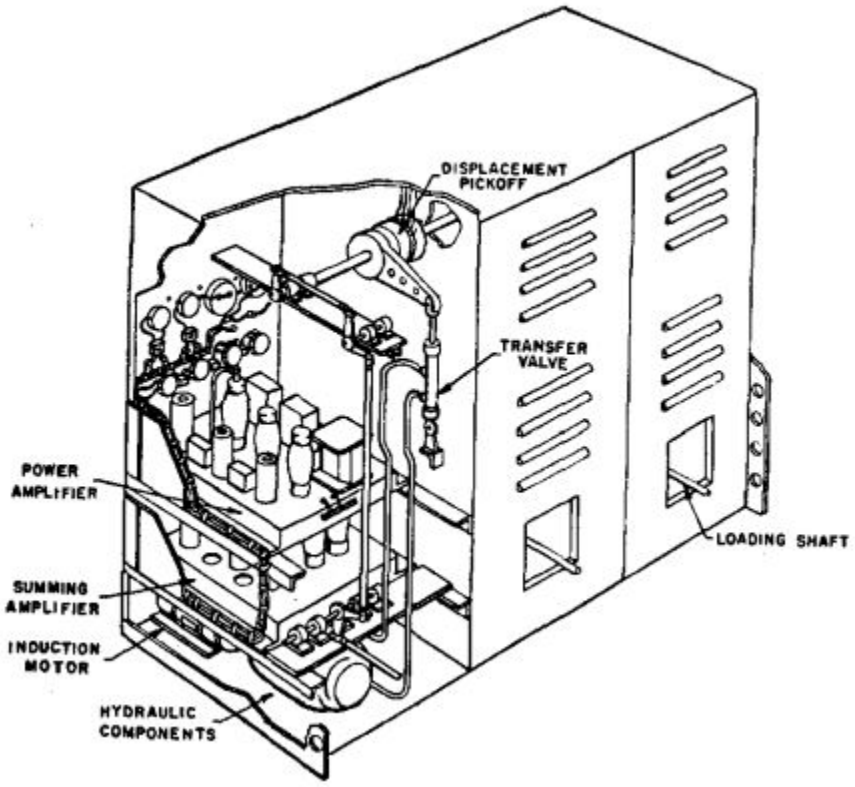

Figure 1.2: Hydraulic control load Simulator [2]

While the use of dynamic test loads for design validation and verification may have started by using them for testing mechanical or grid power, the test load commonly used in modern testing is used for testing power supplies and the power output of various circuits. These electronic loads come in AC and DC versions each with different abilities. Unfortunately, many of the electronic loads sold for testing and research are expensive. The cheapest offering from BK Precision is the 8540 DC Electronic Load which cost $525.00 and is only able to sink a maximum of 30 A at 5 V for a total of 150 W [3]. The cheapest offering from Agilent Technologies, the N3302A 150 Watt Electronic Load Module has a base price of $1,833.00 and also has a maximum power rating of 150 W [4]. While these models are good at what they do, they are expensive and in the case of the Agilent N3302, they need to be used in a large, external, housing.

Figure 1.3: BK 8540 DC Load [3]

Figure 1.4: Agilent N3302A DC Load [4]

1.1 AC Loads

There are two different types of electronic loads, an AC load and a DC load. AC electronic loads are used to emulate both time-varying and invariant loads for AC sources such as inverters, AC sources, and uninterruptable power supplies. AC loads can also be used for testing energy sources such as solar arrays, wind turbines, fuel cells, and batteries [7]. In his research, M. Kazerani found that like DC electronic loads, commercially available AC loads are expensive and limited in their use. Kazerani proposes a controllable AC load designed to simulate constant-current, constant-resistance, constant-power, or nonlinear loads with variable power/crest factors. His proposed design uses two consecutive stages for emulating a load. The first stage converts the AC input to a DC source and serves as a way to control the phase shift of the voltage source as well as a way to realize the power factor or reactive power transfer between the power source and the load. The second stage uses a buck converter to emulate a variable resistor and control the real power consumed by the electronic load. Because Kazerani chose an A/D to buck topology for his AC load, he can bypass the analog to digital converter and connect a DC source directly to the terminals of the buck converter, with the addition of a low-pass filter to protect the source against high current ripples, to create a DC load capable of being controlled as a constant-current, power, or resistance DC load.

While the main use of AC electronic loads is to bench test power supplies for accuracy, speed, and longevity, a new use has sprung up for them: on-site calibration of energy meters [8]. Due to upcoming EU regulations, measuring instruments that affect the daily life of citizens are subject to "legal metrological controls" to verify their calibration and accuracy in common operating conditions. The test is similar to the tests used for petroleum pumps, and store scales to ensure they are accurately providing data to both the store employee, and the owner. Without the proper calibration and accuracy, the customers could be overcharged for the electricity provided to them by the power company. Although most meters come calibrated from the factory, they are calibrated in respect towards their future use. For

example, an industrial meter may be better calibrated to handle imaginary power in the system, while a residential meter might be calibrated to be more accurate in sensing real power used. To solve the calibration and accuracy issue, a load had to be developed to handle the power meter in real world conditions at the facility of use. To do this, Aurilio proposes an AC electronic load using a hysteresis current control scheme. The power meter being measured is hooked to the AC load, and the load is then connected to a DC power supply as seen in Figure 1.5. After simulation of the AC load, it was found that the proposed design would be acceptable for the on-site calibration of any energy meter. While this design requires a simple, and robust design, it looses the power input power to heat, making it an inefficient system.

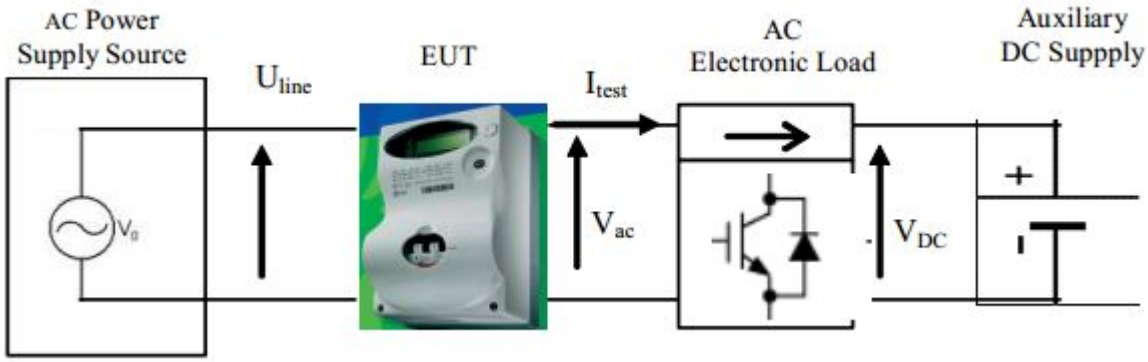

Figure 1.5: AC calibration load set up

The alternative to the common AC electronic load is a regenerative AC electronic load which sends the energy absorbed by the load back to the grid. This approach minimizes the heat produced by the load because the energy does not hit a dead end with a bank full of resistors, capacitors, and inductors and instead goes back to the power grid. This configuration also allows for a real impedance load to be emulated without losses, and without a large bank of parts [9]. An H-bridge design allows for the input rectifier to emulate the specified load and the output inverter to feed the absorbed energy back into the grid. Through the combination of the two H-bridges, a resistive, inductive, capacitive, active and reactive power, and nonlinear loads can be emulated. Their design is controlled by a field

programmable analog array one-cycle controller. In conjunction with a microcontroller and a screen, the device can emulate any impedance load as a stand alone system. A program was also written to calculate the different inputs so the user can properly set the system. The final system was able to successfully emulate the power load and push the normally dissipated energy back to the grid. A side effect of the back to back configurations is the electronic load could be used as a variable AC or DC power source which many electronic loads cannot do.

1.2 DC Loads

Like Jeong's regenerative AC electronic load, it is possible to build a DC electronic load which, instead of dissipating the energy into a bank of resistors, stores the energy into a rechargeable battery system [10]. Like AC loads, DC loads are used to test both the dynamic characteristics of power supplies and to provide a base for long term testing. In order to recover the energy normally lost in a resistive bank, Tsang uses a DC load based on a single ended primary inductor converter (SEPIC). The back end of the converter is then connected to a rechargeable battery instead of a resistive load. Unfortunately, the control for the SEPIC converter involves a fourth order controller with high precision. To overcome this, the controller is divided into four single order proportional plus integral converters. Tsang found that the load was able to correctly preform in both constant current and constant voltage applications. While recharging a battery with the energy flowing through the load may not be practical for laboratory applications, or continuous long term testing due to the finite storage available in a battery, and the lifespan of a battery, this technology can be applied for use in hybrid or fully electric systems to regenerate some of the electricity normally lost when converting a DC motor from a power load to a power source when breaking in a car.

While Tsang's DC electronic load is a novel approach to the use of electronic loads, most DC electronic loads use a bank of resistors connected to the power source through

an electronic switch. This is the method followed and a board was developed for testing the output of high-power, low-voltage power supplies. It was not only used to both test the transient response of the power sources, but also to test the longevity of such devices by providing an uninterrupted load for long periods of time. A high-performance DC load needs to be able to have a high accuracy, fast response to a control signal, high power density, and low cost [11].

While the AC electronic loads are used to mainly test power grid and other one to three phase power sources, DC electronic loads can be used to test sources such as photovoltaic panels and fuel cells to ensure they can react to the loads needed and the available storage in the case the full output of a photovoltaic system cannot be utilized immediately [12][13]. They can also be used to test batteries to verify their capacities and output characteristics [14][15]. Most of the time, the I-V characteristic curves of the DC supplies are being tested. These curves can be used to determine the basic characteristics and parameters of the power supply. While it would be nice to have one set of curves that can determine all of the parameters of the power supply, there are an infinite number of curve combinations. The ones that will be focused on are a constant resistance curve, constant current curve, constant voltage curve, and constant power curve. By testing each of these curves, the full parameters of a DC power supply can be determined [16]. Each of the graphs seen in the following sections can be calculated by using Ohm's Law (1.1) and Joule's Law (1.2).

$$V = I \cdot R \tag{1.1}$$

$$P = I \cdot V \tag{1.2}$$

1.2.1 Constant Resistance

A constant resistance load is the easiest load to build. This type of load doesn't have to involve any active components, just resistors hooked to the output of the power supply.

A constant resistive load is good for testing the longevity of power supplies as everything should stay constant, and if the voltage output of the power supply increases or decreases, the opposite effect would be had on the current output, and vise versa if the current was to change. If they do change over time with a truly constant resistance load, then the voltage or current would only change if parts within the power supply were failing or breaking down. A graph of the I-V curve can be seen in Figure 1.6. Note that as the supplied voltage increases, so does the current.

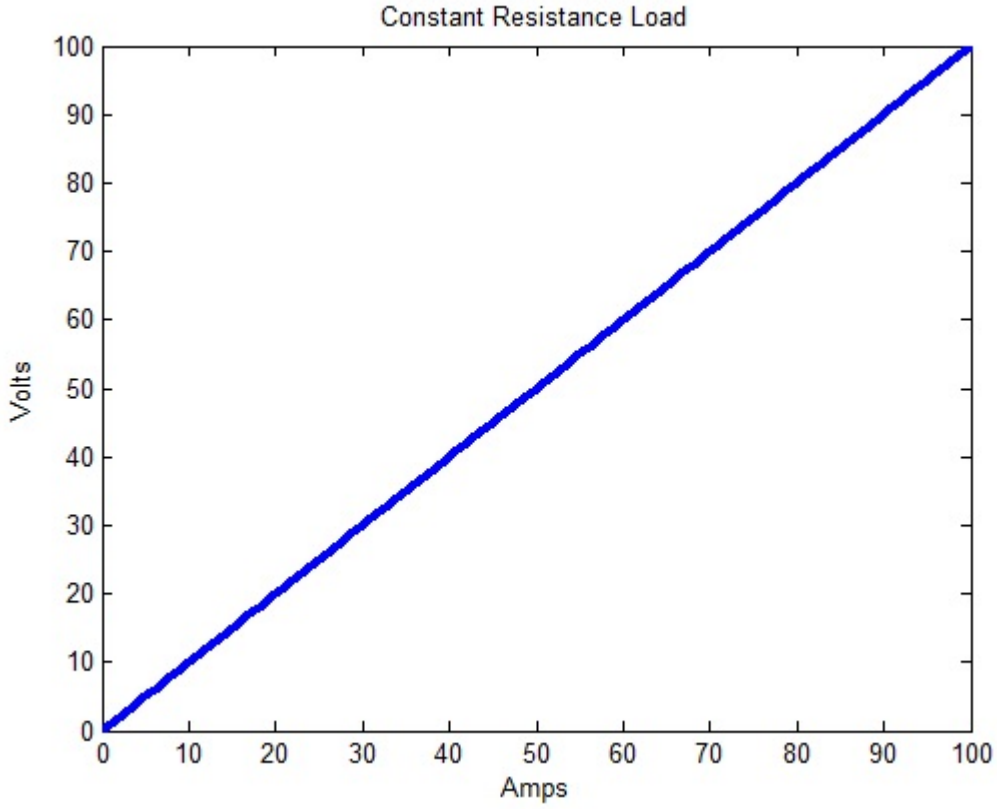

Figure 1.6: Constant resistance load graph

1.2.2 Constant Current

A constant current load is a load that changes its resistance according to the voltage input so the power supply outputs a constant current. An example of a power supply that would need to output a constant current is an LED driver [17]. If the current through an

LED passes its absolute maximum current rating, the LED will begin to break down. The luminous intensity and chromaticity of an LED is also best controlled when driving it with a constant current. A graph of an I-V curve with a constant current can be seen in Figure 1.7. Notice how no matter what the voltage is, the current stays the same, creating a perfectly vertical line. This means as the voltage increases, so does the resistance in the load.

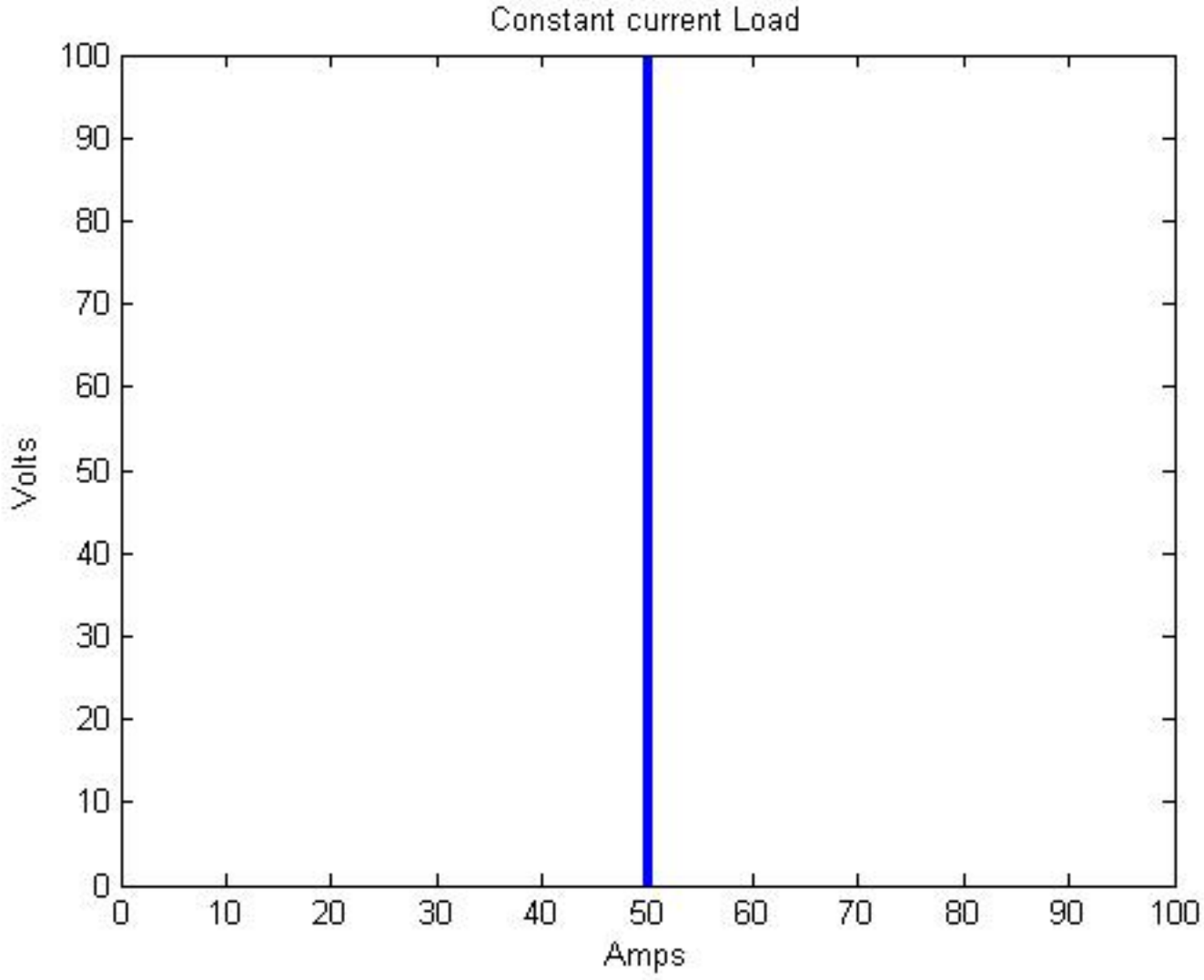

Figure 1.7: Constant current load graph

1.2.3 Constant Voltage

A constant voltage load is a load that ensures a constant voltage is delivered no matter what the resistance or current is output to the load. One example of the use of a constant voltage load is to emulate a battery charger. In the second step of a normal battery charger, the voltage is held the same, and the current is decreased from the maximum allowed to very little as the battery charges [18]. If the maximum voltage is exceeded during this step, there is a chance the battery could explode, which is why the power supply would need to be tested to be able to stay stable for the length of the charging time before a battery is fully charged. A graph of an I-V curve with a constant current can be seen in Figure 1.8. Notice how the current increases and decreases as the voltage stays the same, forming a horizontal line. As the current increases, the resistance on the load needs to decrease to compensate and keep a constant voltage.

1.2.4 Constant Power

A constant power load is a load which increases or decreases resistance while a variable voltage is applied to increase and decrease the input current to keep the same power in as before the change. An example of the need for a constant power load is to test a supply to a LCD monitor. As the monitor heats up, the internal resistance increases which complicates the delivery of a constant power level [19]. A graph of an I-V curve showing a constant power can be seen in Figure 1.9. Notice that as the voltage falls, the current increases keeping the power constant.

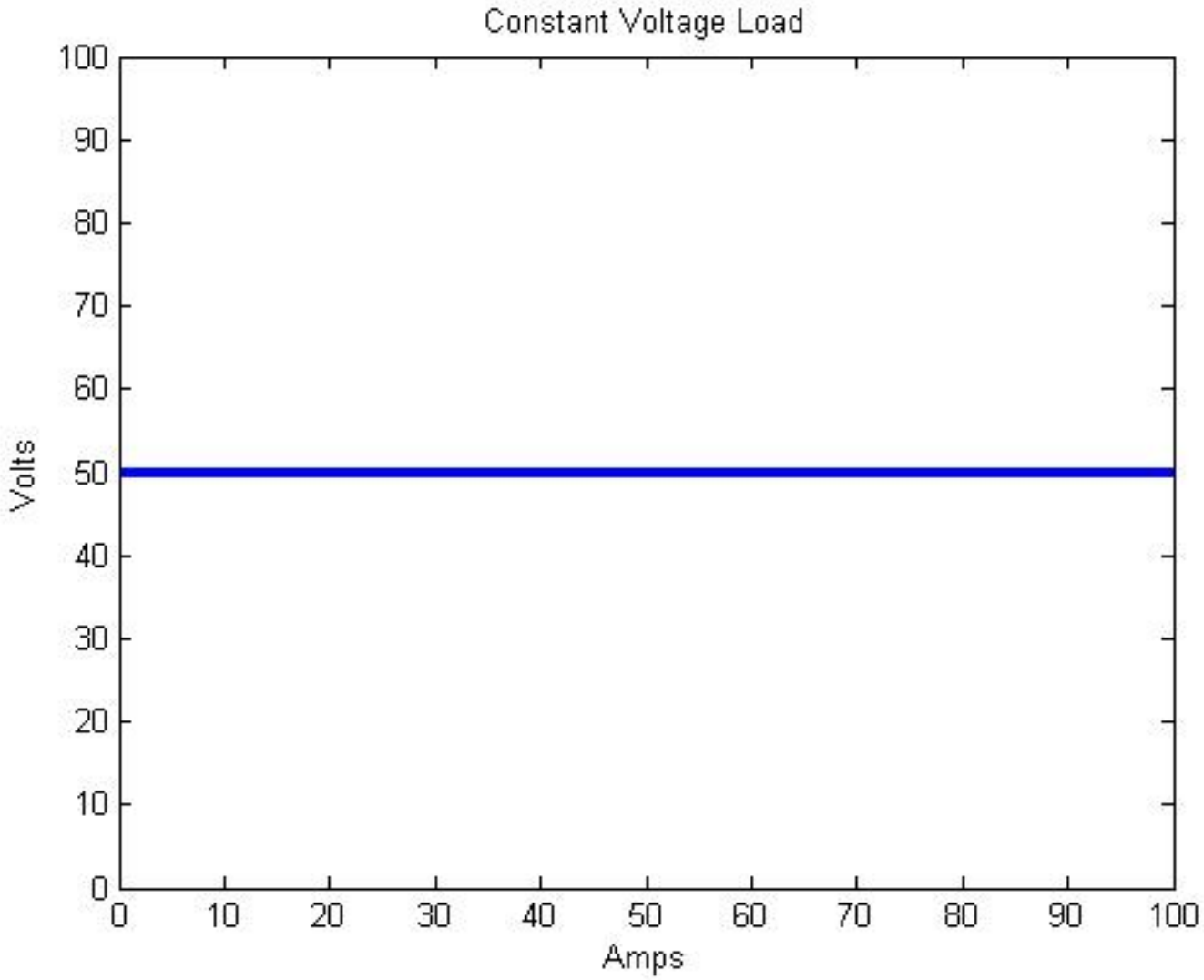

Figure 1.8: Constant voltage load graph

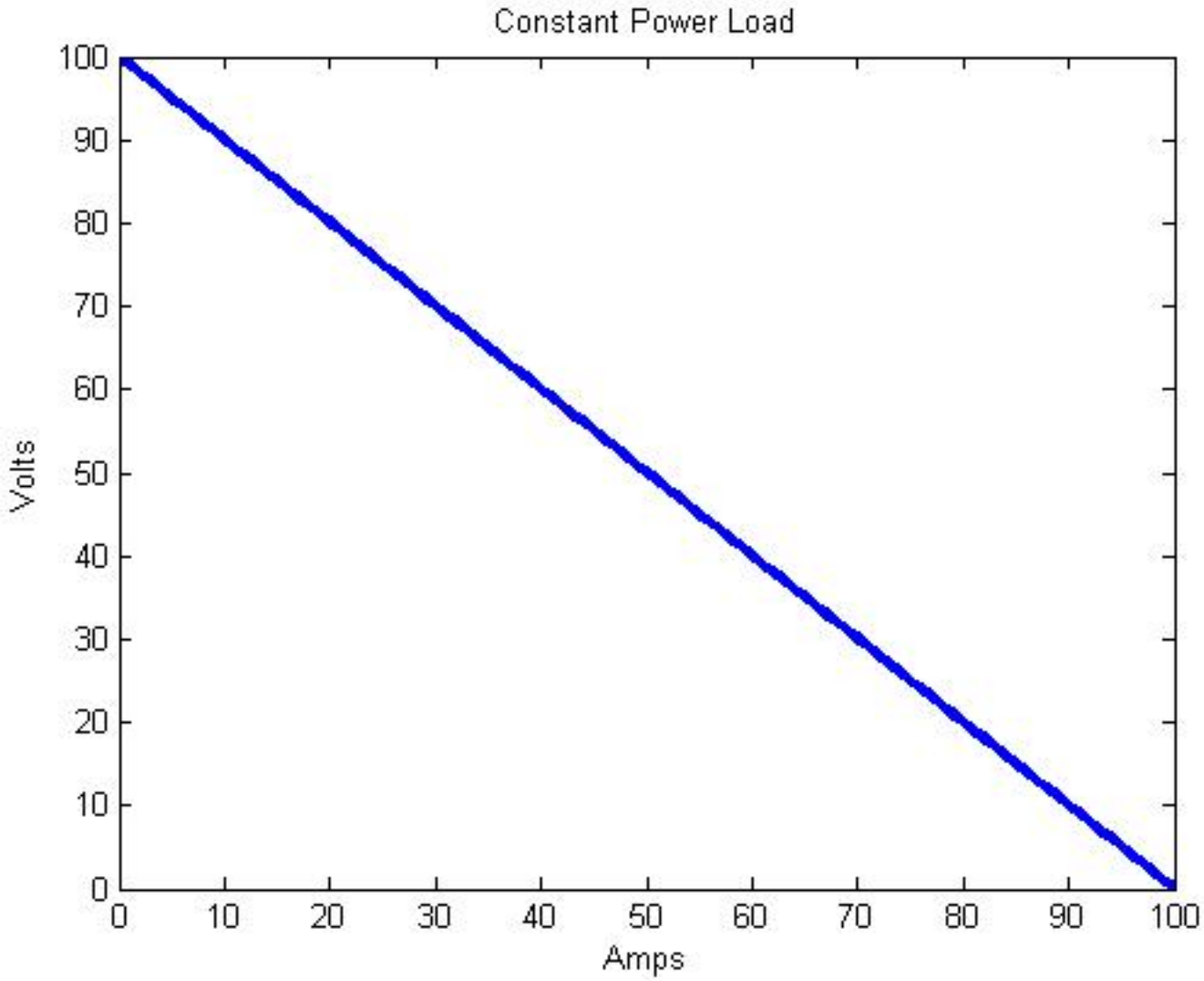

Figure 1.9: Constant power load graph

1.3 Heat Dissipation

Heat is the bane of precision electronics, especially high power precision electronics. As the heat rises in a circuit, the internal resistance increases. This is a problem for a resistive load board because as the board heats up, the extra internal resistance will have to be accounted for to make sure the load is staying within its defined boundaries. To help combat the heat problem, traces have to be wide and thick enough to handle the current flowing through them, there have to be enough vias to exchange the current from one side of the board to the other efficiently so they are not under stress and heat up, and the parts have to be chosen carefully with their heat dissipation characteristics in mind [20]. Different part packages will have different heat characteristics. Some parts are made to dissipate very little power, such as a small surface mount resistor, while other parts are made with high power applications in mind, such as resistors made to be attached to a heat sink [21]. Another problem with part packages and heat is if the parts are too close together, and not able to properly dissipate heat, then the board will experience thermal runaway where the parts heat each other up to the point of failure [20].

Besides the heat being able to damage the components themselves, there is also the possibility that over time large fluctuations in heat can damage the printed circuit board the parts are mounted on. As the board heats and cools, the properties of the board change. The layers of the board can delaminate themselves, destroying vias and other electrical features because of air gaps or thicker dielectric boundaries between copper layers. Heat can also cause the circuit board to warp. This can put extra stress on the solder joints causing them to crack or shift from the intended pad due to their coefficient of thermal expansion mismatch.

While any thermal fluctuation is bad for solder joints and will cause them to wear out due to their expansion and contraction causing cracks and eventual breaks over time, large thermal fluctuation will exaggerate the effect causing the solder joints to fail quicker [5] as seen with the solder crack in Figure 1.10. Unlike thicker layers, scorch marks, or other signs

of high heat on boards, the cracking of solder normally can't be seen with the naked eye and would only be detectable under a microscope. A failed solder joint would ruin the board and make is become unusable after a long period of time. You wouldn't want your load board to fail before the power supply you were stress testing.

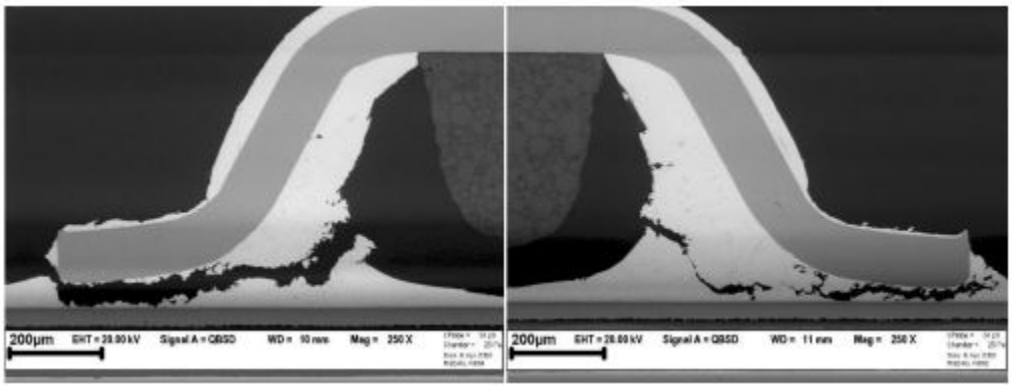

Figure 1.10: Solder joint crack [5]

Chapter 2

Load Board Version 1

The first version of the electronic load developed was co-developed with Stephan Henning. This very basic board was only capable of producing a constant resistance load.

2.1 Specifications

For the first design, the board only had a couple of specifications:

- Had to be able to sink up to 150 A at 1 V
- Had to be adjustable so different currents could be tested at 1 V
- Had to be a small size to easily integrate into the testing setup
- Had to be low cost
- Had to use resistors to produce the load needed
- Had to be a simple circuit so it could be quickly diagnosed if a problem arose

2.2 Design

The populated design of this board can be seen in Figure 2.1 and the CAD drawings can be seen in Figures 2.2, 2.3, and 2.4. Surface mount resistors with the values of 1 Ω, 0.5 Ω, 0.2 Ω, and 0.1 Ω in D-PAK packages were used because of their small footprint and high power characteristics. The resistors are turned on using a International Rectifier IRF6201 HEXFET® Power MOSFET turned on by jumping the gate pin between +5 V and ground. The International Rectifier FET was chosen both because of its low on resistance

of R_{DSOn} = 2.45 mΩ and its high continuous drain current of I_D = 27 A. Anderson Power Pole connectors are used for the +5 V power inputs so it can be easily disconnected, and through hole terminals are used to connect the output of the power supply to the input of the load. The resistors are laid out symmetrically on the left and right sides with the largest resistance, 1 Ω, on the outside, and the smallest resistance, 0.01 Ω on the inside. This allowed for the resistors flowing the largest amount of current to be as close to the input as possible so the resistance of the copper traces was not a large factor when calculating the resistances needed to obtain each current rating. At a 1 V input, the 1 Ω resistor would be flowing 1 A, and the 0.1 Ω resistor would be flowing 10 A of current.

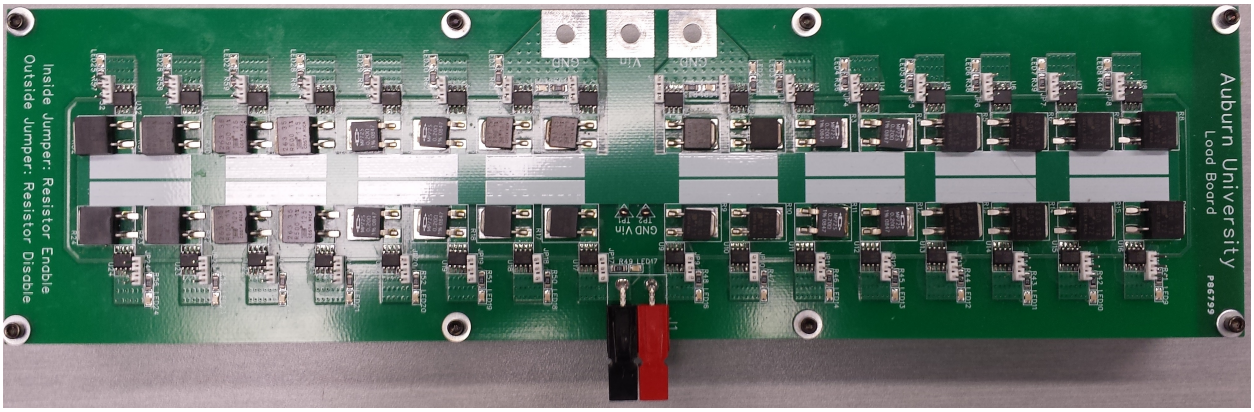

Figure 2.1: Version 1 populated board

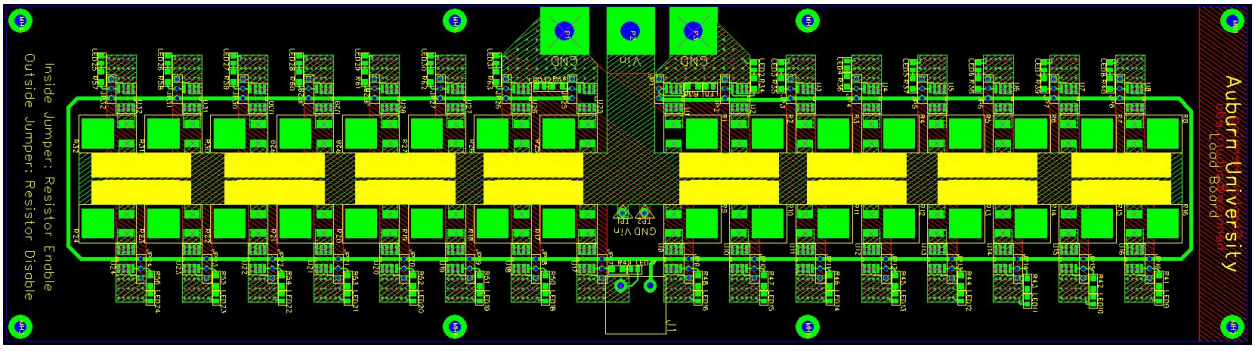

Figure 2.2: Version 1 CAD with all layers

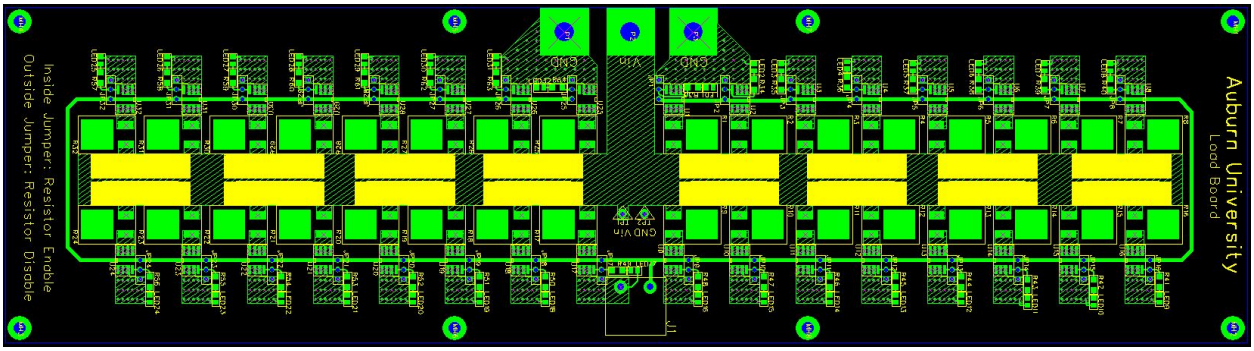

Figure 2.3: Version 1 CAD top copper layer

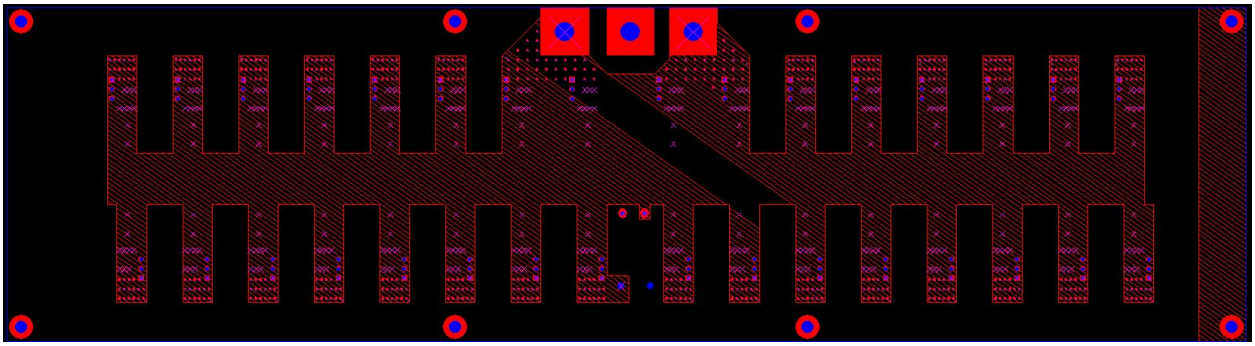

Figure 2.4: Version 1 CAD bottom copper layer

2.3 Results

While the board worked as designed, it had its drawbacks. Even though the resistance could be changed, it had to be changed by hand with the power supply for the load board turned off as well as the power supply being tested. This meant that the board would have to be updated to implement it into an automated testing system. The resistor-FET setup worked well and is implemented in the further revisions.

Chapter 3

Revision One (Daughter Board)

With the first version of the electronic load board working, the automated control needed to be worked on. This new board would not be a completely new revision of the load board, but an add on to the top of the version 1 board.

3.1 Why add the Control Board?

The control board was needed for several reasons. One is that it allows for the almost instantaneous turn on and turn off of all of the resistors. This is good because the load and power supplies can be left on when selecting the new resistances needed and allows for the transient response of the input power supply to be measured. Another reason a control board is needed is to have a load profile the board can cycle through to simulate the different power needs of a system with respect to time. A third reason to implement a load board is to provide an interface with a computer. This interface allows the electronic load to be incorporated into an automated testing system. The automated testing allows the human error in the testing to be decreased as much as possible so consistent data can be taken through the span of testing. The computer integration also allows for the remote changing of the load. This is essential for testing the power supplies in a safe and enclosed environment so there is less risk to the test operator in case something goes wrong.

3.2 Specifications

The control board needed to have some specific specifications:

- An interface with the current V1 load board

- Manual controls to set the load in case a computer was not available for testing
- An on board display to show the expected current flowing through the load board
- Connection to the computer through an FTDI cable and communicate effectively with the testing software used.
- A built in system to test if the connections were working properly.

3.3 Design

The control board was designed as a symmetrical system of two boards on either side of the load board. Both of the boards contained a Microchip Technology PIC16F887 microcontroller to handle the processing, eight FAN3227 gate drivers to turn on and off the FETs on the load board, headers to interface with the load board, three push button switches for selecting modes and other manual interface options, a potentiometer to manually set the resistance if needed, two 7-segment displays and their control circuitry to show the current set to flow on each side of the board, SPI communication to allow for communication between the two sides, and a UART to allow for the communication to the computer through the FTDI cable. The bill of materials can be found in appendix A.

The original run of the control boards did not contain any push button switches and had the wrong pinout for the PIC microcontroller and did not function properly. A revised design containing the push buttons and an improved microcontroller pinout was immediately procured for and populated. The populated board can be seen in Figure 3.1, and the cad drawings can be seen in Figures 3.2, 3.3, and 3.4. The original Microchip PIC16F724 was unable to run even though the pinout on the control board was correct. The PIC16F724 was mounted to an evaluation board and still did not respond to programming. The PIC16F887 was purchased to replace the PIC16F724 and worked in both the control board and evaluation board.

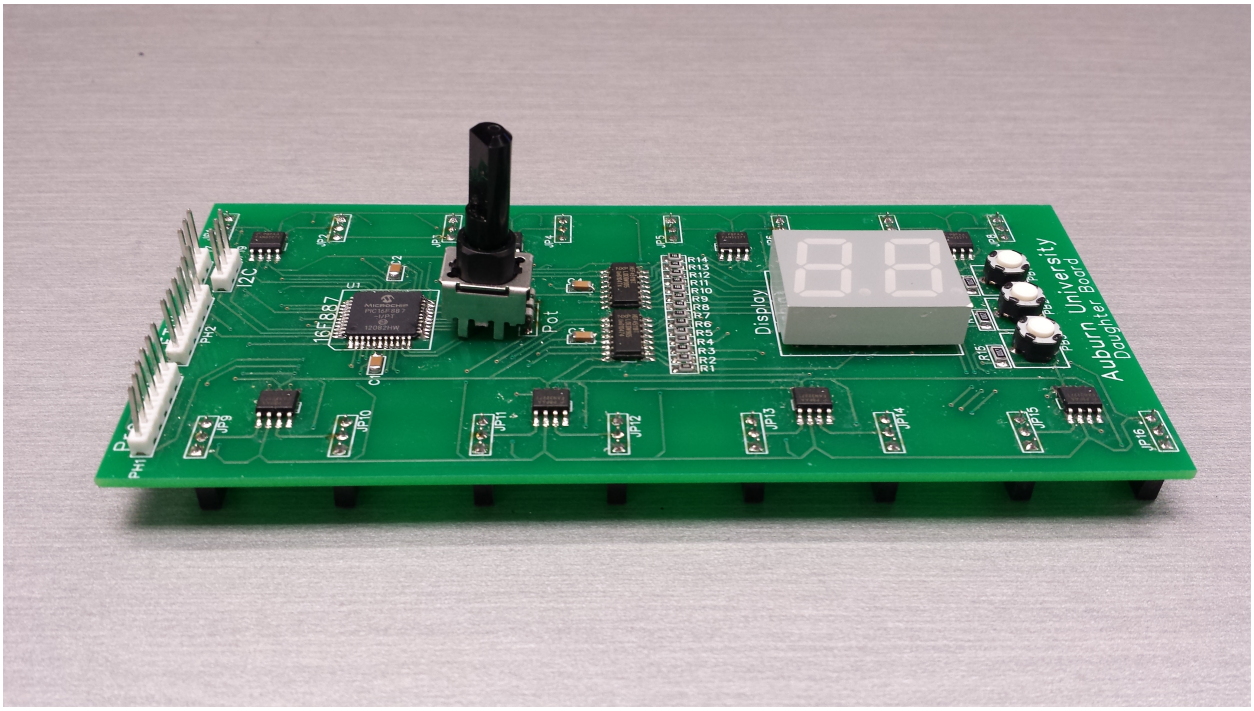

Figure 3.1: A populated control board

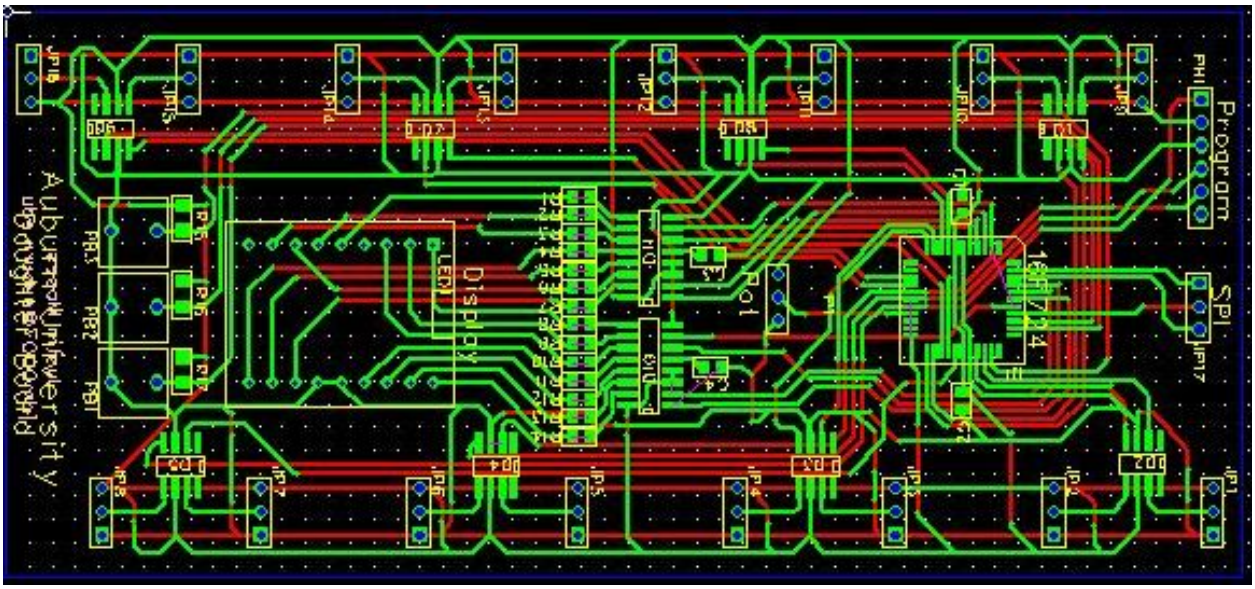

Figure 3.2: Control board CAD design

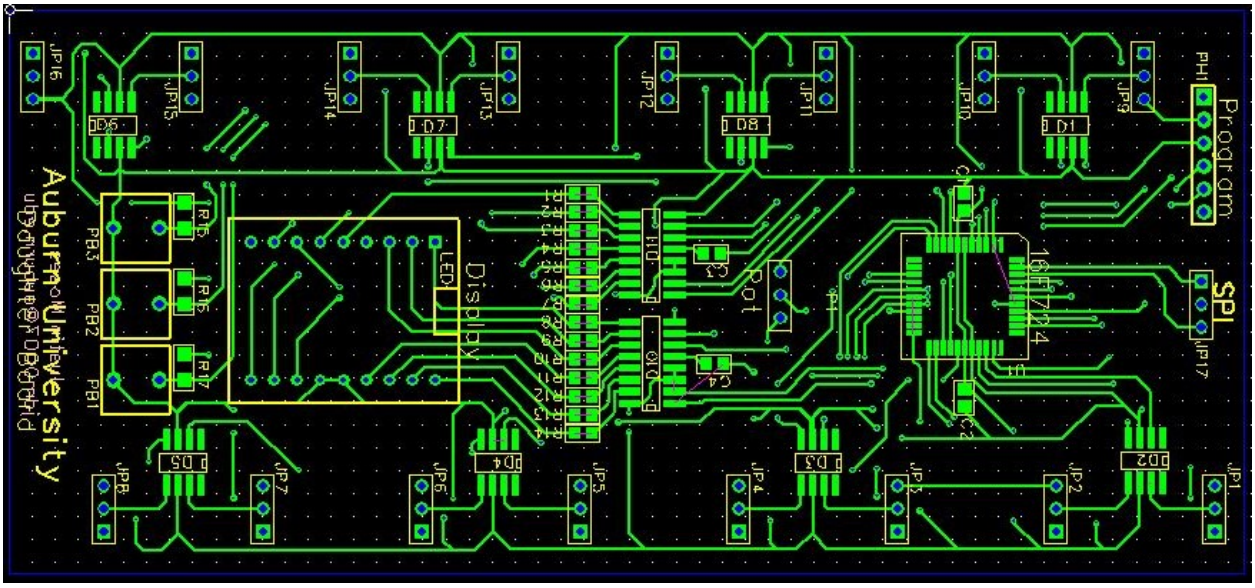

Figure 3.3: Control board CAD top layer

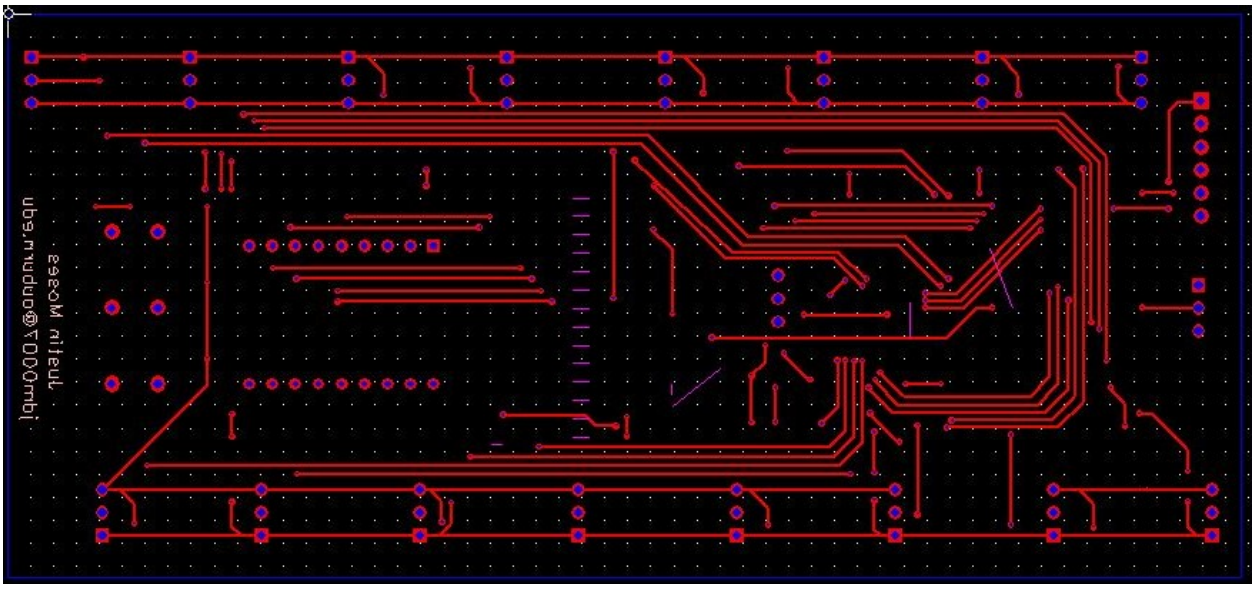

Figure 3.4: Control board CAD bottom layer

3.4 Coding

Besides the board design, another important aspect of the control board was the embedded programming of the Microchip PIC microcontrollers. They were programmed using the MPLABX environment using the C programming language. The code for the control board can be seen in appendix D. The embedded code has to handle everything from turning the resistors on and off, sending the signals to display the current, to communicating between the boards and the computer.

Because of the heat generated on the load board during testing, the heat needed to be spread out as much as possible. To do this the MATLAB script in appendix B was written to calculate the most efficient way of using each resistor. The theory behind the calculations is say for 10 A, one could just use the 0.1 Ω resistor, but that would push all of the current through one small section of the board. Instead, if eight 1 Ω resistors and one 0.5 Ω resistor is used, the current would be able to spread out across the board both reducing the heat building up in the copper connections, and the heat being built up by the resistors. The output from the MATLAB script can be seen in appendix C. Once the resistors required were calculated, they were coded into the main program. The original code used a switch statement to pull the needed resistor values for currents up to 72 A. This allowed each side to only be in control of the resistors covered by the board without having to worry if the microcontroller was trying to turn on more resistors than were available. The limit of the current to 72 A on each side also reduced the number of lines required to operate the output in half. Even though it is not an optimized way of calculating the output, it was quick and made it easily adjustable if a resistor value was found to be off from the specifications.

The second action the embedded code had to handle was the communication between the two PICs. The communication was obtained using the Serial Peripheral Interface bus (SPI) as it only needs three wires and is relatively fast for internal communications between embedded devices [22]. In order to use the SPI interface, each PIC had to be selected as the master, and the other had to be selected as a slave. To set which configuration the

PIC had when each was started up, it would look to see if there was a state already saved in the Electrically Erasable Programmable Read Only Memory (EEPROM). If there was not a state already saved, a ten second period was started where the displays would flash. If the middle push button was pressed during this time, the PIC would store the master state in the EEPROM, and if either of the outer two buttons were pressed, the slave state would be stored. The storage in the EEPROM guaranteed that even if the power was lost to the load boards, the state would be saved so the load could be started back without human intervention. In the SPI protocol, the master PIC controls the clock and initiates the communication between the two. This allows the master to control most of the functions of the load board, and leave the slave to only control the resistors on its side of the load board. The master first looked to see if the desired current was an even or odd number. If it was even, it would divide the current by two and send half of it to the slave board. If the desired current was odd, it would subtract one from the total and send half of what was left to the slave. The master would then add the one back to its half of the desired current so the correct overall current could be reached.

The third action the embedded code had to handle was interfacing with a computer. The communication between the computer and the master control board was achieved by using the Universal Synchronous Asynchronous Receiver Transmitter (USART) [23]. The key to achieving proper communication using the USART protocol is making sure the baud rate is the same on both ends. On the computer side, it is easy to set up as it is specified when setting up the Communications port. On the PIC side a special formula had to be used to calculate the values for the Serial Port Baud Rate Generator (SPBRG) register as seen in (3.1) where the F_{OSC} is the oscillator frequency of the microcontroller (4 MHz was used) and the Baud rate is the desired rate for communications (9600 was used).

$$SPBRG = \frac{F_{OSC}}{16 \times Baudrate} - 1 \tag{3.1}$$

Once the baud rate was correct, the other parameters of the communication had to be set. For the control board, the communications were set to 8-bit with a stop bit and no parity. When set on the computer, the communication packet is 8 bits long and ends with a high ninth bit. The computer was programmed to send the data in the American Standard Code for Information Interchange (ASCII) format. Once it got to the PIC, it was decoded. If the sent packet was not a "g" for go or an "x" for stop, there is a problem, the code looked for a newline packet or a number. The code was set up so that each place in the desired output could be received one at a time. The place was set to zero at the start of the program and would increase as numbers were received. The first number received represented the hundreds place, the place counter would increment, and the second number received would represent the tens place. The counter would increment again and the third number received would represent the ones place. This number scheme could go on indefinitely, but only the first three numbers received would be saved. An example of sending a number to the board would be sending 124 if the desired output was 124 A. For a desired output of 3 A, the sequence 003 would have to be sent. If the new line command was received, it would add up the hundreds, tens, and ones place holders to provide the total desired output. At this time, the place counter would reset so the next three numbers could be sent if needed. The total output would then be sent back to the computer to make sure the numbers sent and the numbers stored were the same. The process of entering and storing the total output could be looped infinitely. If the letter "g" was received, the last desired total output saved would be called, divided, output through SPI, and the resistors would be activated. If at any point in the process the letter "x" was received by the PIC, all resistors would be turned off except a 1 Ω resistor to keep a steady current in case of a fault in one of the systems. Whenever the total output was activated, the PIC also sent the hex values over eight pins to control the 7-segment displays to show the operator the current desired current flow.

3.5 Results

While this iteration of the control board worked most of the time, it still had problems. One was the boards had a hard time communicating with one another. The slave board needed to have an older version of the code to run while the master needed to have the newest version to properly communicate with the computer. The PICs were also very susceptible to Electrostatic Discharge (ESD) and failed several times due to the damaged it caused. While the communication between the PICs did not work as expected, the communication between the computer and master controller worked flawlessly once the computer software was set up. The use of only one side of the modified code was very useful as only a small amount of current was being pushed through the load board at this time in the testing process. The interface with the automated test set up allowed tests which took hours to be brought down to thirty minutes or less because the load could be changed quickly and without interruption.

3.6 Design Problems

While the basic design was good, several problems were encountered during implementation. Since the controller board was a stand alone board and was not included in the original load board, the proper fitting on the control board on the load board was difficult. If any of the male headers on the load board warped or were placed crooked during the assembly, they had to be bent by hand until they would fit into the female headers on the control board. The division of the two control boards also caused a problem. One problem was that because the SPI header was not keyed, it was easy to reverse the leads from the desired orientation. The display was also hard to read in this application. One board had the numbers right-side-up, and the other board showed the numbers upside-down. This made it difficult to try and add them together to calculate the total current. The control boards sitting on top of the load board also obstructed air flow to the resistors preventing the heat from properly dissipating.

Chapter 4

Load Board Revision Two (The Combined Board)

The main goal of the third revision was to integrate the controller board into the load board to form one coherent board. Besides reducing the number of boards that needed to be produced, the part count was also reduced because the extra switches and displays became redundant. Combining the boards also allows for only one PIC microcontroller to be used, cutting out the need for SPI communication between devices.

4.1 specifications

There are several specifications that needed to be met in the second revision of the load board:

- Load and controller boards combined
- Fully alphanumeric display
- Current and voltage sensing
- Handle up to 300 A at 1 V
- Use mostly the same parts from previous load board

4.2 Design

The design of the second revision of the load board can be seen in populated in Figure 4.1, and the CAD layout can be seen in Figure 4.2. The board begins with the same basic layout of the original load board with the resistors on the outside of the board. The board incorporated eight 0.05 Ω resistors and four 2 Ω resistors on top of the previous 32. The

small 0.02 Ω resistors allowed for an increase of current sinking capability by 160 A. The 2 Ω resistors allowed a fine tuning of the circuit during calibration since they only flow 0.5 A at 1 V. This configuration allows for the easiest routing of control and power lines to the gate drivers and FETs; a schematic of a quarter of the board can be seen in Figure 4.3, and the whole board schematic and PCB layout can be seen in appendix F. The screen on the left side of the board is a back lit Newhaven NHDC0220BIZ−FS(RGB)−FBW−3VM Liquid Crystal Display (LCD), which is able to show two lines with twenty characters each. The display allows both the desired current and actual current to be displayed as well as the voltage running through the board and the resistance turned on. To power the 3 V display, a 3 V voltage regulator was used to shift the 5 V from the board to the proper voltage. To the left side of the board is a set of four push buttons to allow the user to manually select options or resistances. In the middle of the board is a PIC18F87K22 to handle all of the controls and communications. The PIC communicates with the computer through the USART connection with the same 9600 baud rate as before, and to the Newhaven display through an Inter-Integrated Circuit (I^2C) interface. This allows all of the information needed to run the display to be fed over three lines instead of eight.

On either side of the PIC microcontroller is an Allegro ACS758XCB Hall Effect-based current sensor. These sensors are able to sense -150 to 150 A with a maximum of 1% error. These sensors eliminate the need for an inline current sensor or current sense resistor between the input power supply and the load board. They send the current data to the PIC through the PIC's analog inputs. A voltage of +5 V means there is 150 A flowing though the sensor, and a voltage of 0 V means there is -150 A flowing. A voltage reading of 2.5 V means there is no current flow. The PIC also uses an external, high resistance, voltage divider on another analog input port to sense the input voltage of the load board. These allow both for an automated control stream, and a source of feedback for the computer.

The printed circuit board is six layers thick. This allows for the four inner layers to be solid copper pours while leaving the top and bottom layers to carry the traces for device

power and control. The layers can be seen in appendix F The inner two layers carry the input from the power supply and the next two layers carry the ground plane. The position of the ground planes help block any interference from reaching the signal wires on the top and bottom planes.

4.2.1 Current Distribution

One of the biggest problems in a board that can sink 300 W of power at full capacity is the distribution of the current throughout the board. Like the first load board, the resistors carrying the least amount of current are placed the farthest away from the power supply input, and the resistors carrying the largest amount of current are placed the closest to the input. As before, this allows the least amount of copper traces and planes the current has to travel through before reaching back to the ground of the input.

The other factor besides the traces that needed to be addressed is the use of vias. Like traces, vias can only carry a maximum amount of current before they heat up and increase resistance [24]. Using the PCB Calculator Toolkit from Saturn PCB Design Incorporated [25] as seen in Figure 4.4 it was calculated that a 25 mil diameter via with a pad diameter of 35 mil with a 1 mil plating through the 62 mil thick board is able to carry up to 3.7822 A while keeping the temperature rise in the via at or below 25 °C. While a resistance of 0.57 mΩ may seem high, when the via is paralleled with several other vias, the overall resistance is greatly reduced and the total current carrying capacity is increased. The choice to use 16 vias was made because at the highest concentration of current, the vias would be flowing a total of 20 A. The 16 vias allows for a maximum of 60 A to flow before their temperature reaches the 25 °C above ambient temperature. In the original design, the current sensors had plated slots for the legs. Unfortunately, the company manufacturing the boards, Advance Circuits, could not plate a hole in the PCB that was not a perfect circle, so large planes of vias were placed on either side of the legs to carry the current to and from the top and bottom layers of the board that were coated for soldering.

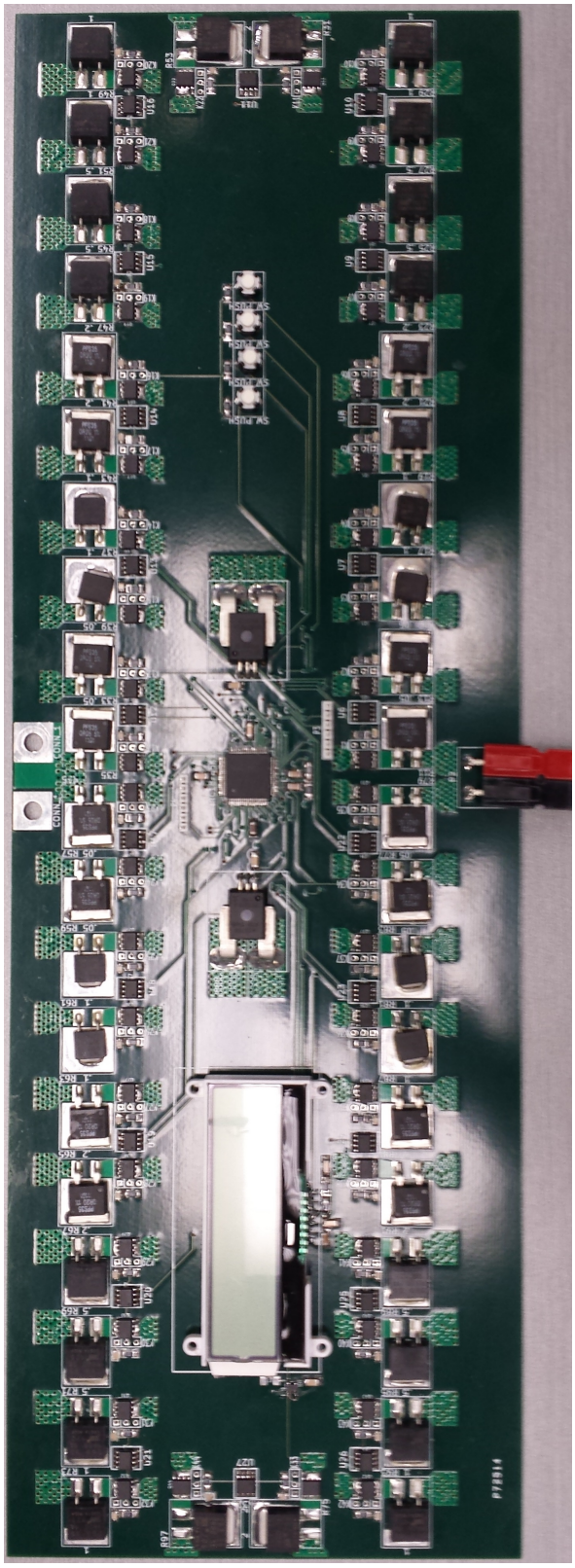

Figure 4.1: Fully populated revision 2 load baord

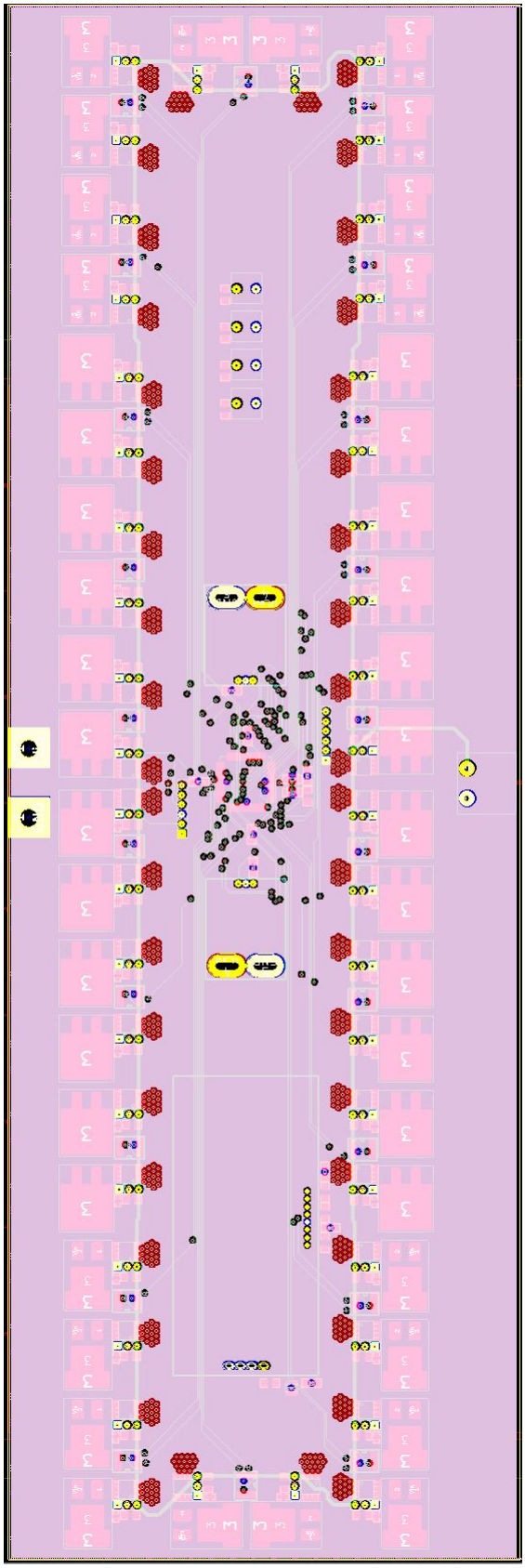

Figure 4.2: CAD drawing of version 2 load board

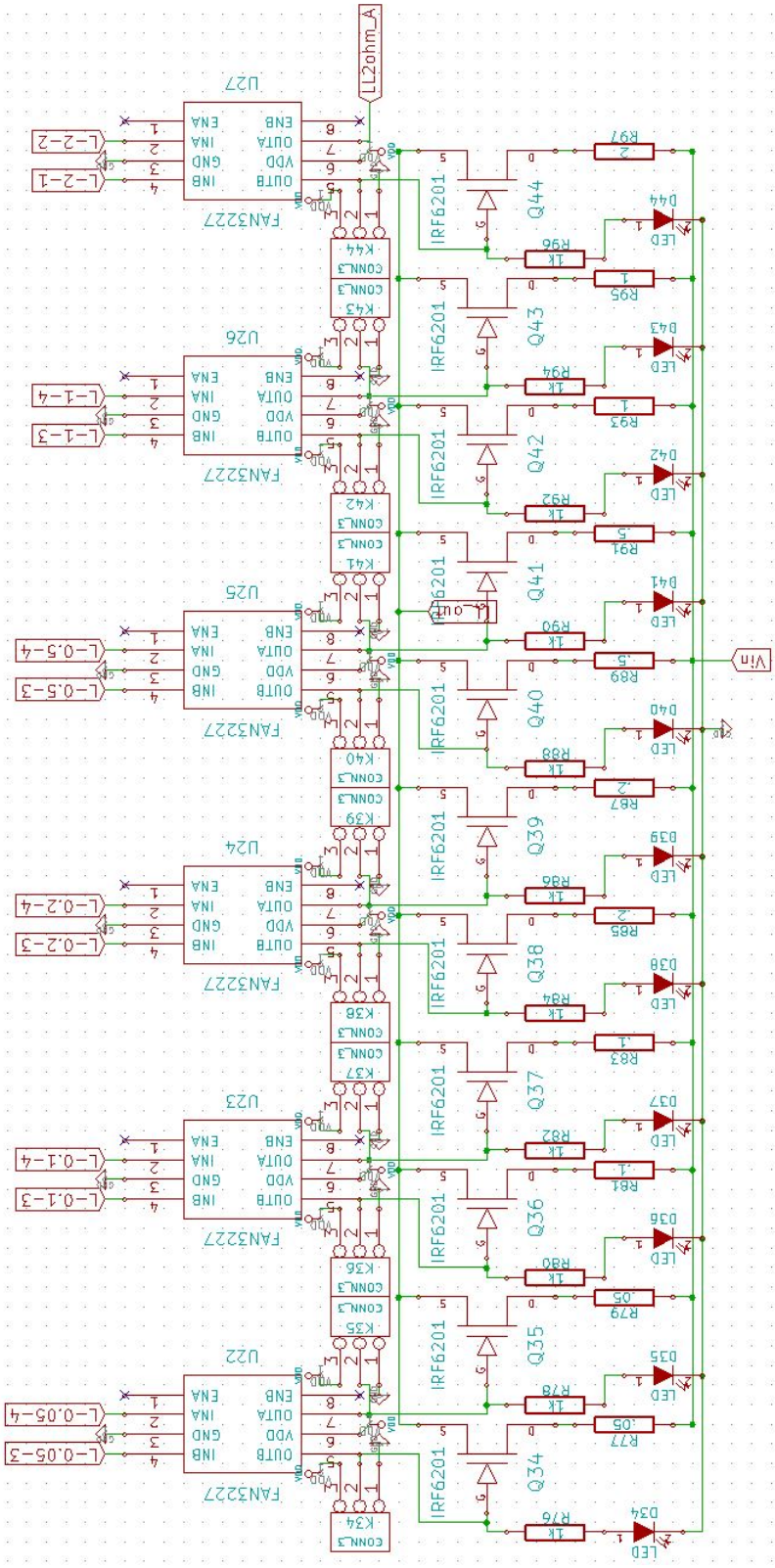

Figure 4.3: A quarter of the resistor load layout

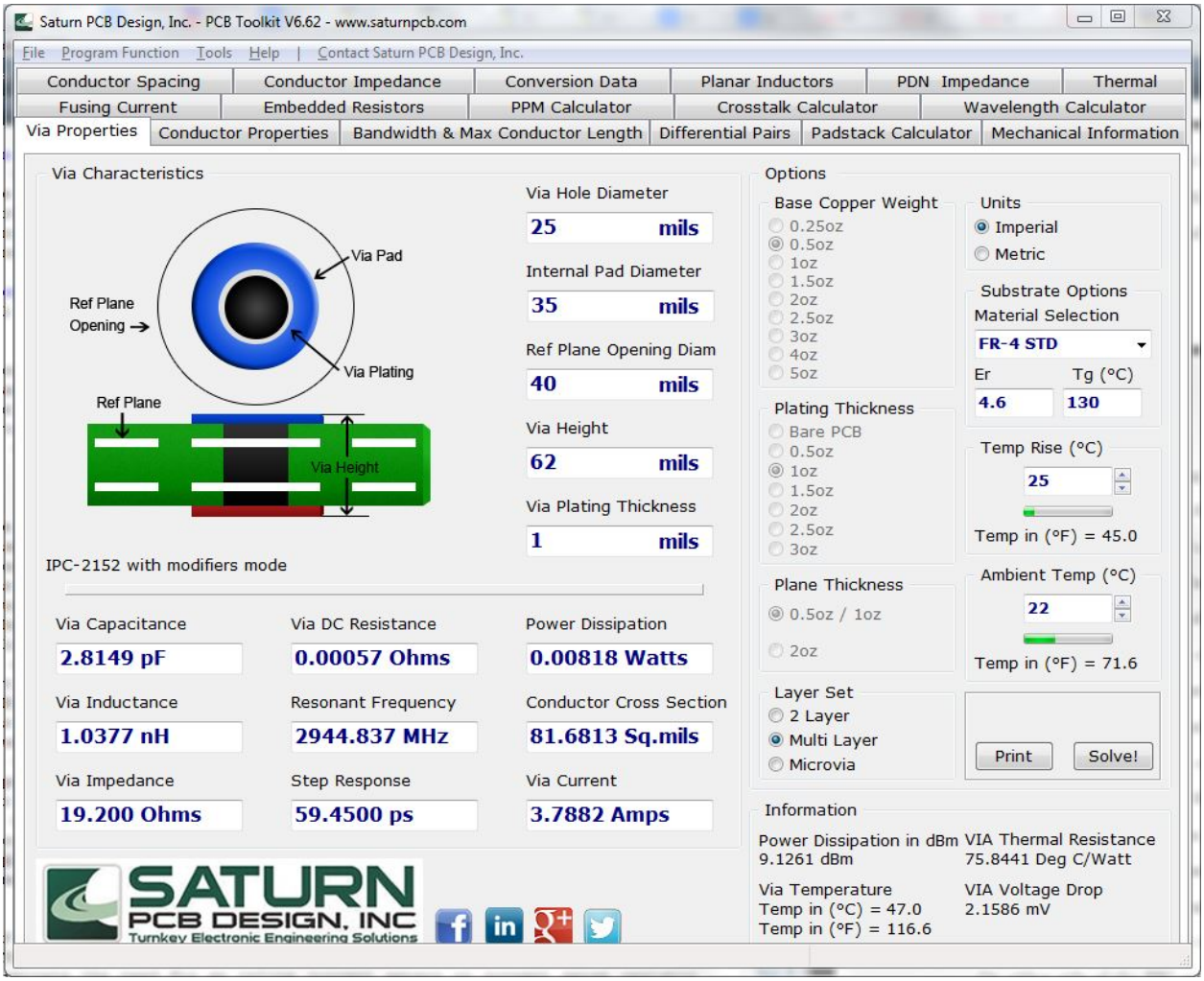

Figure 4.4: PCB design toolkit calculations

4.3 Coding

For the code of the version two of the load board, many of the same approaches were used as in the first control board. The full code can be seen in appendix G. The PIC communicates with the computer using the USART port and an FTDI cable. Since the board is not divided into two different PICs, the only communication protocol needed between components on the board is the I^2C used by the LCD screen. While the I^2C bus uses the same pins as the SPI previously used, the implementation is easier as the LCD is defined as the slave, and has a specific address for the PIC to send signals to. To turn the LCD screen on, and start the communication process, the commands and delays found in the data sheet [26] were used. At first, the display would not turn on, but after troubleshooting the problem was found to be the transmit and receive lines had been switched when making the PCB. Once the problem was fixed, the display worked as it should. To send any text to the LCD, the screen clear command was sent, and then the text was sent with a new line command to push text to the second row when needed.

While the commands through the USART bus were not changed, and extra command was added. In the start up routine for the computer program, the program polled all of the communication ports of the computer sending the ASCII character "?". If the load board received the question mark, it would reply back "LBC" to indicate that the load control board was on the specific com port. This made it easy for the test program to be moved to different computers because the COM port for each device did not have to be known or hard programmed in before hand. The command for what the load board did when the character "x" was received was also changed to completely turn off the resistors so no current would flow in case of an emergency.

To try and improve the efficiency of the output routine on the control board, the old switch statement controlled output was replaced with a modular approach. The desired current was broken into different sections for each resistor value. The resistors were turned on if the desired resistance was within or outside of certain values in a combination of if

and else if statements. While the code for this takes up as much space as the original switch statements, it can be expanded to encompass greater current outputs without much additional code. This was later changed back to an extended version of the switch statements because of issues while calibrating the board.

4.4 Calibration

While the design and embedded programming of the load board are important, it cannot be used as a design validation and verification device if it is not properly calibrated. As mentioned earlier, this is the reason the output sequence had to be reverted to the old style. In an ideal design with no losses, with a 1 V input, the current flowing through the board would be the inverse of the resistance selected. Unfortunately on the load board, this was not the case. While this inconsistency may have been because of the tolerances in the resistors, resistance in the vias, traces, and R_{DSON} of the FETs, it had to be calibrated out.

To start the calibration process, the power supply inputs of the board were hooked to a Keithley multimeter. Using a four-wire setup, the total resistance was measured between the terminals. This total resistance seen by the multimeter was what controlled the current output of the power supply. Because a very small amount of current was flowing through the board during this test, resistance due to heat in the parts and traces on the board was negligible. After the first run through of the resistances with the Keithley, the data was recorded and can be seen in Figure 4.5. While the lowest and highest expected resistances are within 1 Ω of each other, the resistances throughout the range were off by an average of 4.6%. The difference between the desired resistance and measured resistance peaked around a resistance of 200 Ω. It would make sense that as the resistance increased, so would the difference in the measured values because any small series resistance through lines and connectors would be increased because of the small total resistance of all of the resistors in parallel, but this is not the case. After the initial test, the resistors were remapped for each desired current based on the measured resistance by the Keithley. While this got closer to

the desired output, there was still a difference in the desired and measured currents as seen in Figure 4.6.

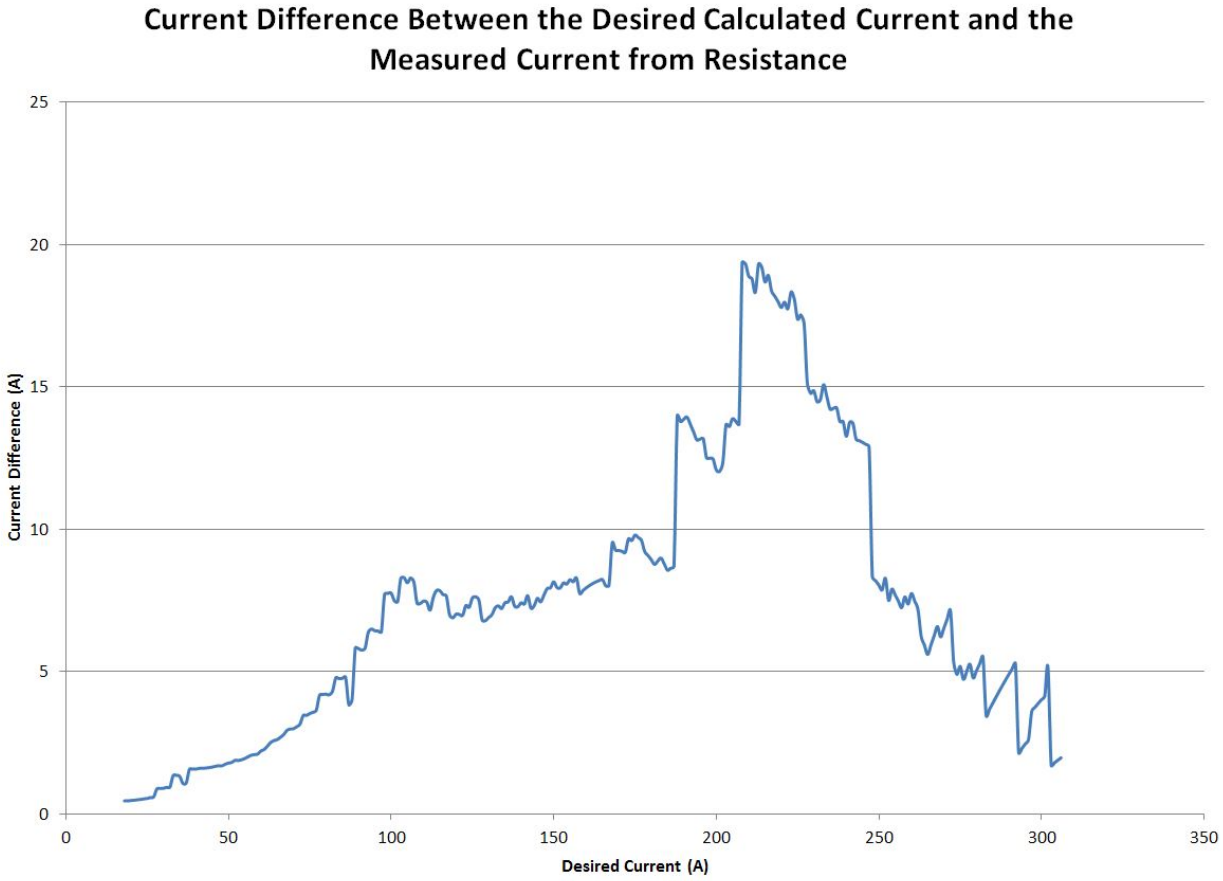

Figure 4.5: Current difference

After the board was tested on the Keithley, it was apparent the board needed to be calibrated. To do this, the load board was hooked to the output of an Agilent N5761A power supply rated for 0-6 V 180 A. The board was first put through its paces and tested to a theoretical 180 A. Instead of reaching the expected 180 A, the board only reached 62.1 A when 140 A was the desired output as seen in Figure 4.7. This is due to many factors. One of the largest is fluctuation in the resistance of the resistors combined with the internal resistance of connectors, traces, and parts. The inconsistencies in the resistances between resistors of the same values accounts for the fluctuation of the desired current when different resistors of the same value are used instead of the original resistors. The second issue is

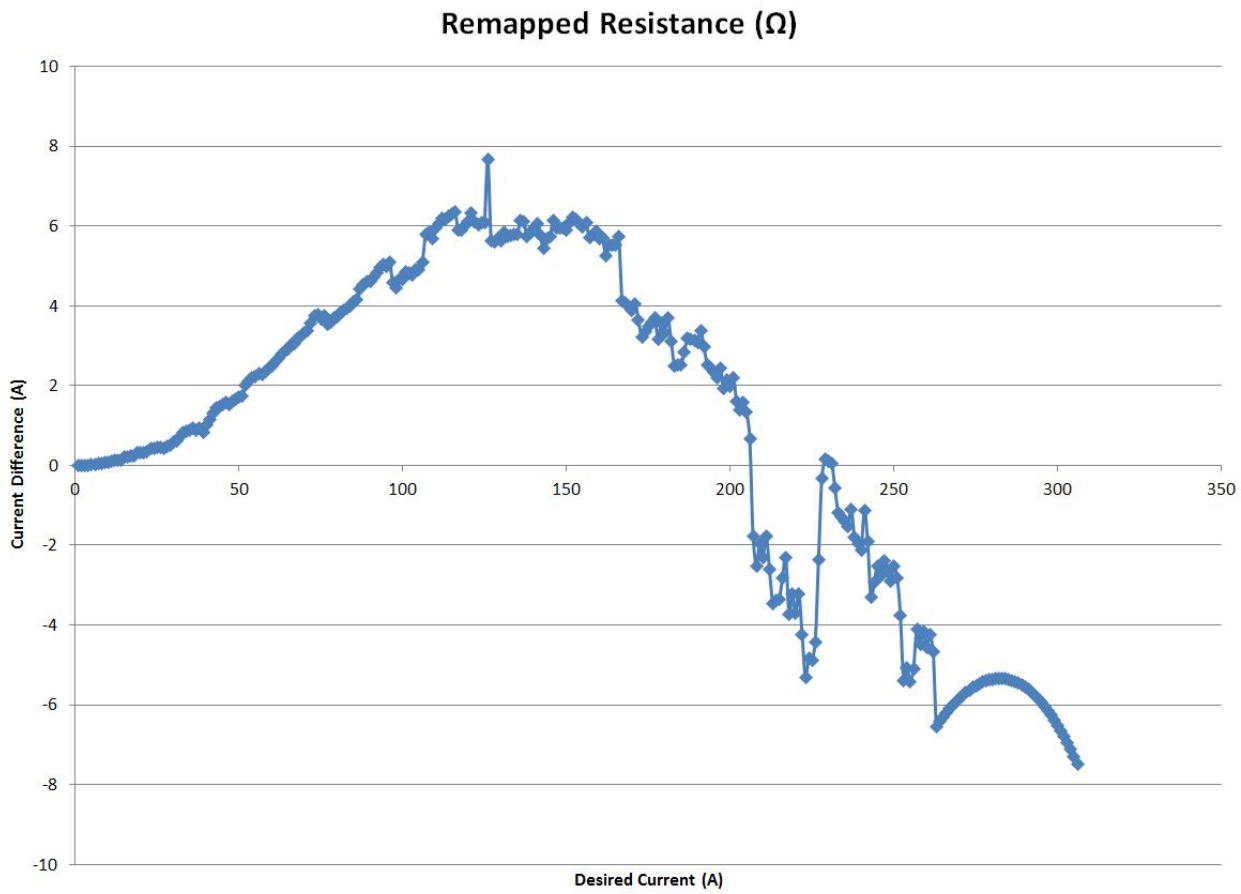

Figure 4.6: Calibration with Keithley multimeter

the heat being dissipated by the board while at the higher currents. While each via can transfer 3.78 A, the input of the load board only contained 36 vias meaning the input can only handle 136 A before becoming saturated and heating up, causing even more losses. This can especially be seen in the higher currents without any air circulation cooling the board off because the current can be seen decreasing from the power supply as the resistance of the board was increasing because of heat. If a fan was used, the board did not gain resistance as quickly. The Agilent power supply was then used to calibrate the board for currents up to 62 A. The board was not calibrated to the maximum current due to board constraints, time constraints for the rest of the project, and the need was not great to be able to handle more than 62 A during that phase of power supply testing.

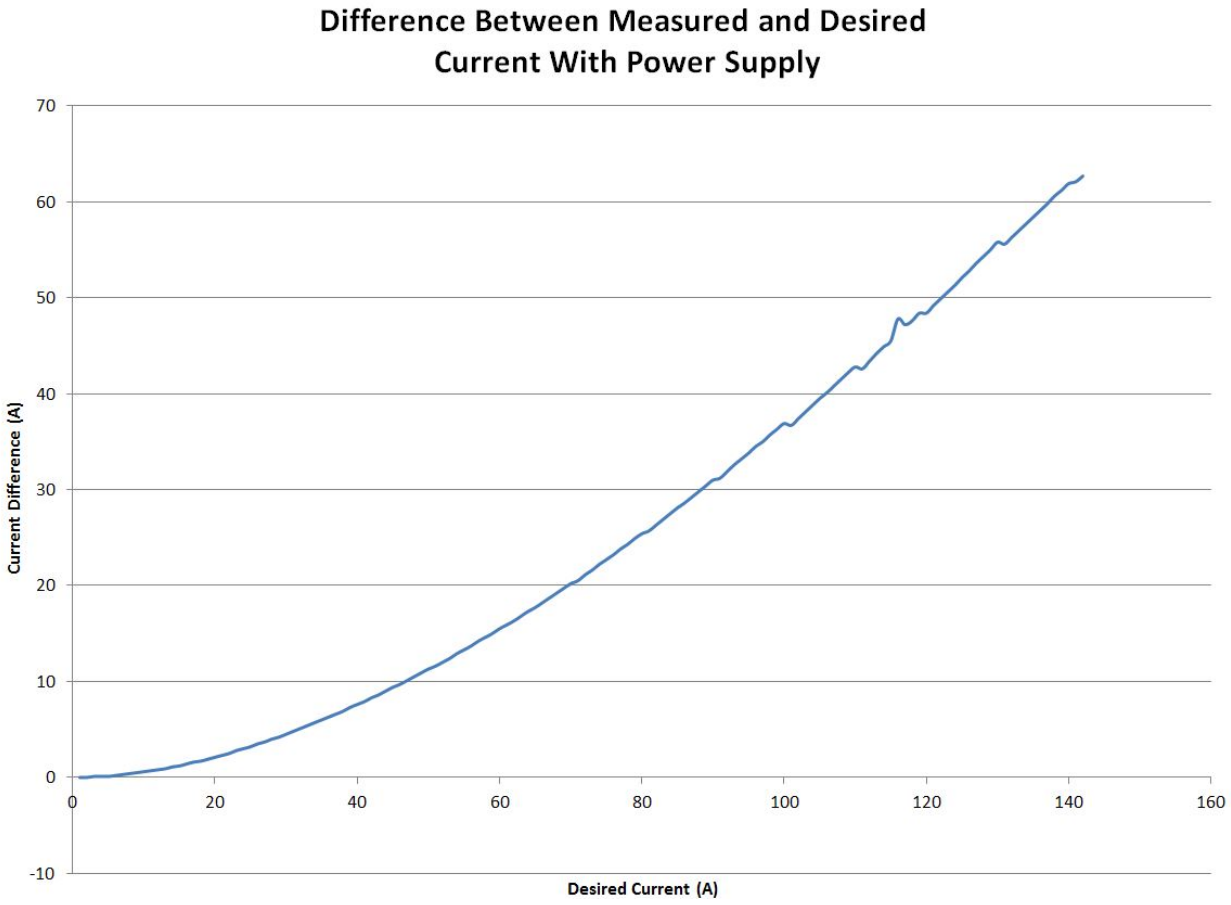

Figure 4.7: Initial current difference

Chapter 5

Power Supply Test Integration

After the electronic load board was tested and calibrated, it was used to test the validity and efficiency of buck converters. The buck boards were placed in their own carrier board so they could be swapped throughout testing. A current sense resistor was then placed between the buck carrier board and the load board. This helped get accurate current readings from the load board. The load board was then placed under a high flow fan to help cool the resistors and traces while the tests were running. This helped keep the resistances and current flow stable throughout the buck board testing process to ensure the same data was being gathered for each buck board. The test setup can be seen in Figure 5.1. In the figure, the load board is at the top right. To the left is a high flow fan ensuring the board stays cool. Directly below the load board is the current sense resistor and a current probe. The dual current sense setup made sure the correct data was being collected and allowed a way to fix any discrepancies. At the very bottom of the figure is the power supply being tested. Only one of the five possible buck phases was being tested at the time the picture was taken.

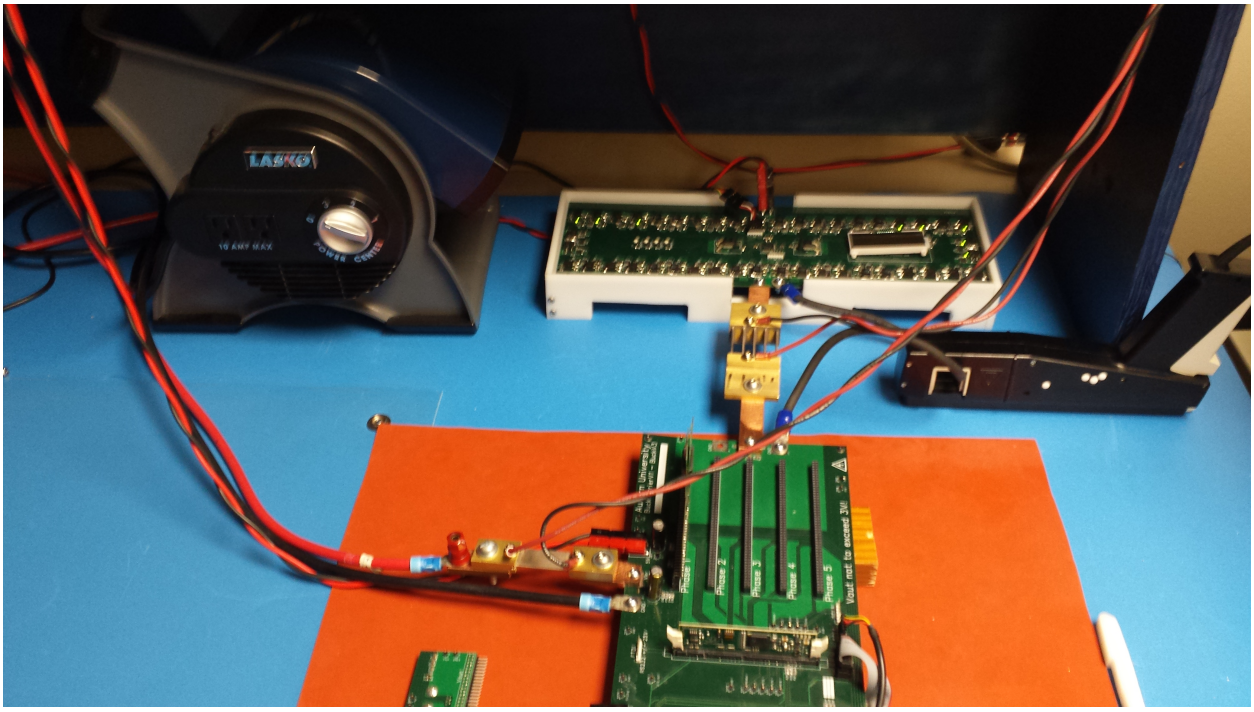

Figure 5.1: Complete test setup with working load board

Along with the hardware integration with the power supply test setup, the load board was integrated into the software testing environment. The software used to communicate with the load board was developed by Dr. Christopher Wilson and controlled the load board, power supplies, and took the data from the testing. A screenshot of the load board being setup in the program can be seen in Figure 5.2. The automated testing software communicated with the load board by sending ASCII characters throughout the testing process. The software started out by sending a "?" and received the response "LBC" if the load board was connected. The software then sent out the desired load and the load board sent back the total load to the computer to confirm it was correct. The communications for a portion of the testing procedure can be seen in Figure 5.3 where the board is testing a ramp up of 4 to 25 A of current and then starting over again. A portion of the data collected from this typical test of a buck board can be seen in Figure 5.4. In the data shown, the green column is the desired current, and the blue column is the actual current flowing through the

load board as read from the current sense resistor. The load board averages a difference of only 0.477 A from the desired current, and at a desired 25 A only has a 2.75% error.

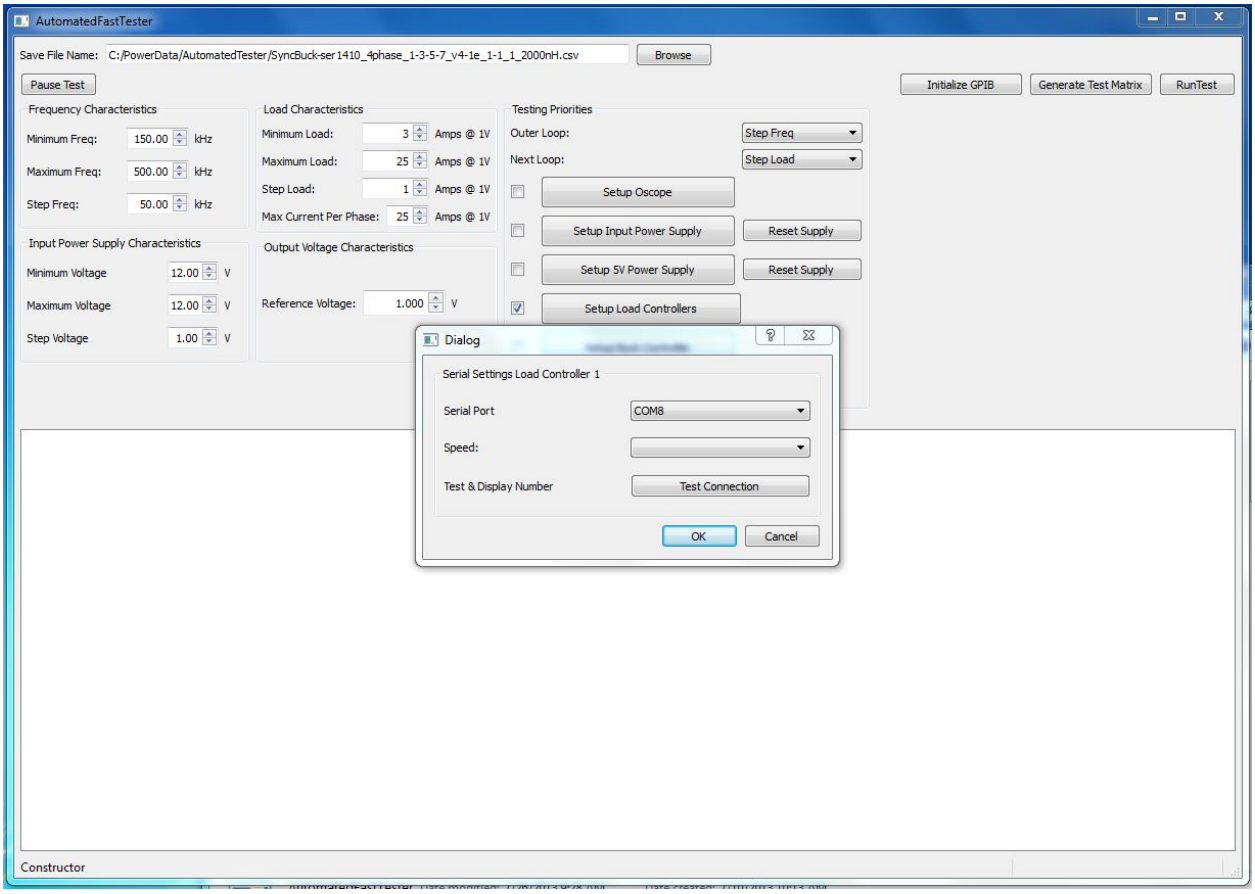

Figure 5.2: Load board setup in testing software

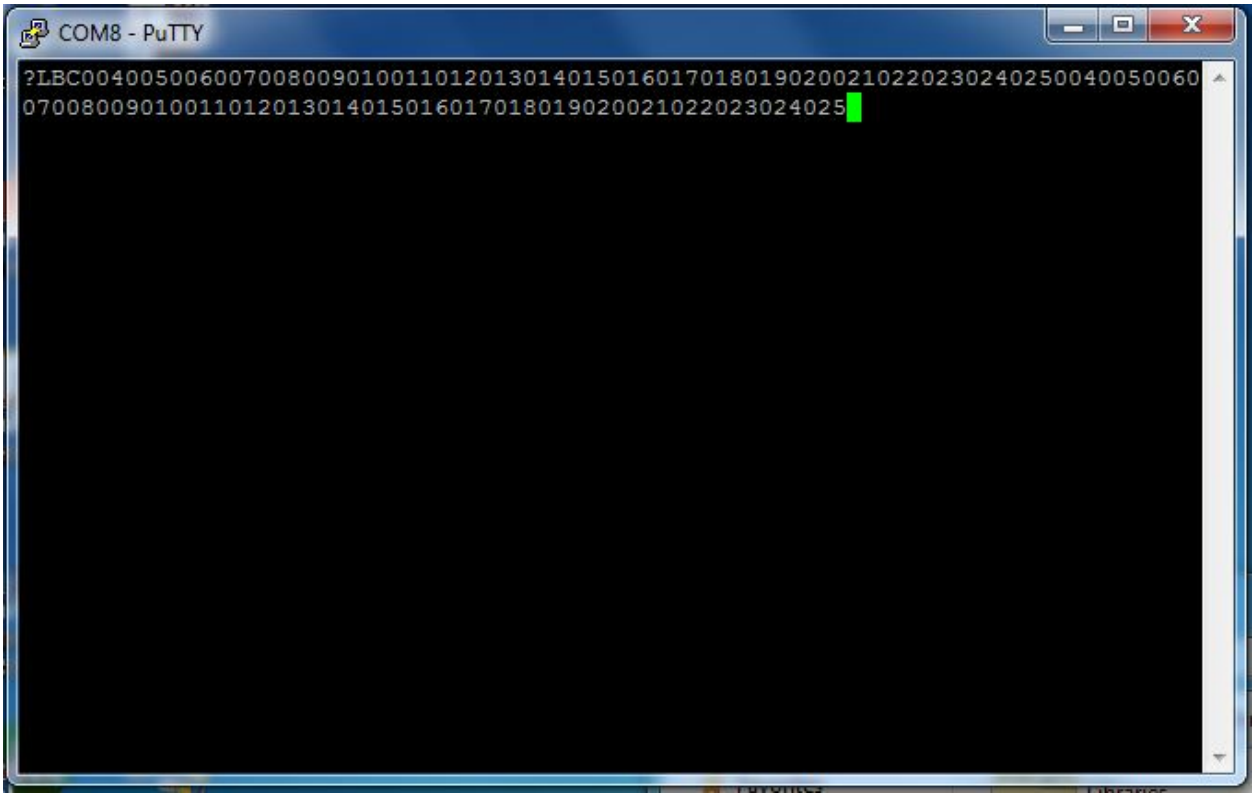

Figure 5.3: Commands sent and received from the load board during testing

D. Load	D. Freq	D Vin	Freq	PDuty	Vin	Iin	V5V	I5V_OFFS	I5V	Vout	VoutMin	VoutMax	Iout	Pin	Pin (+5)
4	120	12	119980	8.6153	11.876	0.36	4.50321	9.22E-05	0.00497	0.99733	0.48	1.384	3.7821	4.27536	4.29776
5	120	12	119988	8.6644	11.88	0.45	4.50323	9.22E-05	0.00485	0.99683	0.488	1.464	4.7066	5.346	5.36782
6	120	12	119972	8.7151	11.8822	0.51	4.50325	9.22E-05	0.00505	0.99588	0.368	1.344	5.5903	6.05992	6.08267
7	120	12	119988	8.7656	11.8724	0.6	4.50326	9.22E-05	0.00502	0.99699	0.392	1.4	6.5044	7.12344	7.14603
8	120	12	119977	8.8592	11.873	0.75	4.50325	9.22E-05	0.00493	0.99669	0.488	1.464	7.8556	8.90475	8.92697
9	120	12	119982	8.8888	11.8691	0.81	4.50326	9.22E-05	0.00489	0.99572	0.384	1.488	8.729	9.61397	9.63599
10	120	12	119976	8.9357	11.8752	0.9	4.5032	9.22E-05	0.00498	0.99522	0.448	1.52	9.5963	10.6877	10.7101
11	120	12	119980	9.0152	11.8671	0.99	4.50322	9.22E-05	0.00482	0.99681	0.344	1.552	10.4573	11.7484	11.7701
12	120	12	119990	9.084	11.8641	1.14	4.50321	9.22E-05	0.0049	0.99621	0.248	1.584	11.7671	13.5251	13.5471
13	120	12	119982	9.0761	11.9625	1.2	4.50319	9.22E-05	0.00471	0.99726	0.216	1.64	12.6283	14.355	14.3762
14	120	12	119983	9.1037	11.9638	1.29	4.50324	9.22E-05	0.00494	0.99556	0.296	1.744	13.4471	15.4333	15.4555
15	120	12	119984	9.179	11.9613	1.44	4.50329	9.22E-05	0.00491	0.99484	-0.02397	1.76	14.7267	17.2243	17.2464
16	120	12	119975	9.2598	11.9547	1.53	4.50322	9.22E-05	0.00473	0.99716	-0.02397	1.936	15.5771	18.2907	18.312
17	120	12	119977	9.2937	11.9537	1.62	4.5032	9.22E-05	0.00489	0.99577	-0.02397	1.936	16.3786	19.365	19.387
18	120	12	119977	9.3511	11.9552	1.71	4.50327	9.22E-05	0.00489	0.99574	-0.02397	1.936	17.2232	20.4434	20.4654
19	120	12	119980	9.4126	11.9515	1.83	4.50319	9.22E-05	0.00486	0.99532	0.016	2.024	18.4376	21.8712	21.8931
20	120	12	119981	9.4904	11.9492	1.92	4.50319	9.22E-05	0.00469	0.99698	-0.02397	2.024	19.2782	22.9425	22.9636
21	120	12	119982	9.5722	11.9518	2.07	4.50323	9.22E-05	0.00467	0.99684	-0.02397	2.024	20.5026	24.7402	24.7613
22	120	12	119985	9.6563	11.9425	2.16	4.50325	9.22E-05	0.00462	0.99623	-0.75	1.89	21.3397	25.7958	25.8166
23	120	12	119988	9.6811	11.9504	2.28	4.50324	9.22E-05	0.00481	0.9963	-0.70797	1.34	22.4331	27.2469	27.2686
24	120	12	119986	9.7224	11.9474	2.37	4.50327	9.22E-05	0.00459	0.99592	-0.69597	1.352	23.2195	28.3153	28.336
25	120	12	119972	9.7837	11.9413	2.49	4.50326	9.22E-05	0.0047	0.99529	0.012	1.748	24.315	29.7338	29.755

Figure 5.4: Test data gathered with load board

Chapter 6

Conclusion

Overall, the goal of creating an inexpensive, accurate DC electronic load board that could handle high currents was achieved. The first iteration of the board laid down the stepping stones for a load board with its manual controls, FETs , and resistors. The second board produced, the control board, allowed the connectivity of the original load board with a computer as well as an automated platform for changing the load seen by the power supplies. The third board, which combined both the original load and control boards into a single unit, expanded upon the capabilities already used by the previous boards and expanded upon them by having a larger load capability and more and stable control options for the integration with testing software and equipment. The final board cost less than $500 with the most expense spent on the fabrication of the board. The cost of the board can be reduced if ordering in larger quantities, and longer fabrication times. The load board was able to interface with a computer and send and receive signals using USART and an FTDI cable for integration in an automated testing environment. The third board even had extra capabilities that where not utilized during the span of the power supply testing period. While the board may not have been able to reach the 306 A of the final goal, it was able to sink as much power as was being tested at the time after a proper calibration.

Chapter 7

Further Research

There are many outlets for further research on the low cost, high power DC electronic load. One is to complete the coding, and build the fourth revision of the load board. This would provide a much finished and complete DC load while still being able to be tweaked and upgraded. Another option is to look at the board more in terms of a commercial product. After the bugs were worked out, a case can be created, and more energy can be spent into heat dissipation and easier interfaces such as integrating the USB connection and creating more user-friendly controls for a stand alone device. Software also needs to be created which is dedicated to the load board so different load profiles can be loaded in to simulate different needs such as processors over a period of time, or to simulate the charging of a battery.

7.1 Load Board Revision Four

The board design expressed in the further research section was already being worked on before funding ran out, and is explained below. The fourth iteration of the board is an improvement of the third revision, especially in the thermal management areas.

7.1.1 Revisions

The design of the third load board had several revisions from the second load board:

- Input power planes placed on the outside of the board for better heat transfer and thicker copper
- Signal traces were run on the two middle planes
- Through hole resistors used instead of surface mount

- Heat sinks were included to help dissipate heat from the resistors
- Fans built in to blow the hot air away from the load board instead of a large external fan
- Input and ground plane sizes increased as well as adding vias to improve current flow
- New power supply input connectors
- Include holes for standoffs

7.1.2 Design

Again, the version three of the load board had the same general layout of the first and second boards with the resistor around the edge. The screen, PIC microcontroller, push buttons, and current sensors were kept in the middle of the board for easy access. Holes for standoffs were also added to increase the air circulation around the bottom of the board while the added fans increased the circulation on the top of the board. The input connectors for the power supply being tested were also being changed to be able to handle the heavy currents needed to test the power supplies while being quicker and easier to disconnect than the bolted on "I" loops.

7.1.3 Coding

The coding of the board was also being worked on. While many of the functions were being kept the same as the version two load board, the current sensors, voltage sensor, and automatic calibration were being added. The addition of the current sensors in the board would allow for the board to sense the resistance selected was too high or too low to produce the correct current and toggle resistors until the correct current was met. The addition of the sensors would also allow for control algorithms to be embedded onto the board so the board could operate with the other functions of constant current, voltage, and power instead of only being able to handle the constant resistance.

Bibliography

[1] C. Air, "85a/12v 1260 watt diversion dummy load resistor heater." [Online]. Available: http://www.colemanair.us/vp_asp/scripts/shopexd.asp?id=602

[2] E. H. Gamble and B. W. Hatten, "Design and analysis of a conservative dynamic load simulator," *Journal of Applied Physics*, vol. 22, no. 10, pp. 1250–1257, 1951. [Online]. Available: http://scitation.aip.org/content/aip/journal/jap/22/10/10.1063/1.1699836

[3] B. Precision, *DC Electronic Load Model 8540*, BK Precision Data Sheet, 2008. [Online]. Available: http://www.bkprecision.com/downloads/datasheets/8540_datasheet.pdf

[4] A. Technoilogies, *Agilent N3300 Series DC Electronic Loads*, Agilent Technologies Data Sheet, December 2012. [Online]. Available: http://cp.literature.agilent.com/litweb/pdf/5980-0232E.pdf

[5] L. N. Lu, H. Huang, B. Wu, Q. Zhou, X. X. Su, and M. Cai, "Investigation of thin small outline package (tsop) solder joint crack after accelerated thermal cycling testing," in *Electronic Packaging Technology High Density Packaging, 2009. ICEPT-HDP '09. International Conference on*, Aug 2009, pp. 923–926.

[6] K. Corless and A. S. Aldred, "An experimental electronic power-system simulator," *Proceedings of the IEE - Part A: Power Engineering*, vol. 105, no. 23, pp. 503–511, October 1958.

[7] M. Kazerani, "A high-performance controllable ac load," in *Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE*, Nov 2008, pp. 442–447.

[8] G. Aurilio, D. Gallo, C. Landi, and M. Luiso, "Ac electronic load for on-site calibration of energy meters," in *Instrumentation and Measurement Technology Conference (I2MTC), 2013 IEEE International*, May 2013, pp. 768–773.

[9] I. W. Jeong, M. Slepchenkov, K. Smedley, and F. Maddaleno, "Regenerative ac electronic load with one-cycle control," in *Applied Power Electronics Conference and Exposition (APEC), 2010 Twenty-Fifth Annual IEEE*, Feb 2010, pp. 1166–1171.

[10] K. Tsang and W. Chan, "Fast acting regenerative dc electronic load based on a sepic converter," *Power Electronics, IEEE Transactions on*, vol. 27, no. 1, pp. 269–275, Jan 2012.

[11] M. Kazerani, "A high-performance controllable dc load," in *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*, June 2007, pp. 1015–1020.

[12] J. Trapp, J. Lenz, F. Farret, L. Kehler, and J. Shoenhalz, "Dc electronic load based on an interleaved chopper converter to determine photovoltaic panel and fuel cell biasing curves," in *Industry Applications (INDUSCON), 2012 10th IEEE/IAS International Conference on*, Nov 2012, pp. 1–6.

[13] F. Locment, M. Sechilariu, and I. Houssamo, "Dc load and batteries control limitations for photovoltaic systems. experimental validation," *Power Electronics, IEEE Transactions on*, vol. 27, no. 9, pp. 4030–4038, Sept 2012.

[14] J. A. Hadfield, "Development of an economical, software-controlled battery load testing system," in *Telecommunications Energy Conference, 1990. INTELEC '90., 12th International*, Oct 1990, pp. 553–555.

[15] S. Girard, R. E. Thompson, and G. Miller, "New architectures for battery test systems," in *Battery Conference on Applications and Advances, 1999. The Fourteenth Annual*, 1999, pp. 129–133.

[16] A. Technoilogies. (2013, May) How does an electronic load regulate it's input voltage, current, and resistance? Element 14. [Online]. Available: http://www.element14.com/community/docs/DOC-54529/l/how-does-an-electronic-load-regulate-it-s-input-voltage-current-and-resistance

[17] M. Integrated, *Why Drive White LEDs with a Constant Current*, Maxim Integrated Products, August 2004, application Note.

[18] D. Corporation. (2002, April) Battery charging basica ans charging fundamentals. [Online]. Available: http://batterytender.com/includes/languages/english/resources/Battery_Charging_Basics.pdf

[19] D. Sackett. (2009, October) Constant-power source. Maxim Integrated. [Online]. Available: http://pdfserv.maximintegrated.com/en/an/AN4470.pdf

[20] R. Prasad, *Surface Mount Technology: Principles and Practice*. Chapman & Hall, 1997.

[21] O. M. Company. (2014) Application notes: Resistor selection. [Online]. Available: http://www.ohmite.com/techdata/res_select.pdf

[22] M. T. Inc., *Overview and Use of the PICmicro Serial Peripheral Interface*, Microchip Std. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/spi.pdf

[23] ——, *Using the USART in Asynchronous Mode*, Microchip Std. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/usart.pdf

[24] D. Brooks and D. Graves, "Current carrying capacity of vias," *Printed Circuit Design*, January 2003.

[25] S. P. D. Incorporated. Pcb toolkit v6.62. Application. [Online]. Available: http://www.saturnpcb.com/pcb_toolkit.htm

[26] N. D. Technology, *NHD-C0220BiZ-FS(RGB)-FBW-3VM COG (Chip-On-Glass) Character Liquid Crystal DisplayModule*, Newhaven Display Std. [Online]. Available: http://www.newhavendisplay.com/specs/NHD-C0220BiZ-FSRGB-FBW-3VM.pdf

Appendices

Appendix A

Control board version 1 bill of materials

Daughter Board					
Part	Part #	# of parts	Price	New part#	Price
Gate Driver	LM5111-2MYDKR	8	1.47	FAN3227TMXCT	0.9828
LED Display	516-1209-5	1	1.43		
LED Driver	568-8243-1	2	0.36		
potentiometer	P3K1203	1	0.85		
pic	PIC16F724-I/PT	1	2.44		
LED resistors	P330ADKR	14	0.04		
Female Headers	S9008E-03	16	0.92		

Table A.1: Control Board V1 BOM

Appendix B

Resistor calculation code

```
clc
clear all
R2 = .1;
R3 = .2;
R4 = .5;
R5 = 1;
R6 = 2;
R1 = .005;
Roff = 0;
R_max = R1;
R_min = 1/(8/R1+8/R2+8/R3+8/R4+8/R5+4/R6);
R_values = zeros(626,10);
count = 2;

for a = 0:8
    for b = 0:8
        for c = 0:8
            for d = 0:8
                for e = 0:8
                    for f = 0:4
                R_values(count,1) = 1/(a/R1+b/R2+c/R3+d/R4+e/R6+f/R6);
                R_values(count,2) = a;
                R_values(count,3) = b;
                R_values(count,4) = c;
                R_values(count,5) = d;
                R_values(count,6) = e;
                R_values(count,7) = f;
                R_values(count,6) = 1/R_values(count,1);
                if R_values(count,2) == 0
                    R_values(count,7) = 0;
                end
                if R_values(count,3) == 0
                    R_values(count,8) = 0;
                end
                if R_values(count,4) == 0
                    R_values(count,9) = 0;
```

```
end
if R_values(count,5) == 0
    R_values(count,10) = 0;
end
if R_values(count,2) == 1
    R_values(count,7) = 1;
end
if R_values(count,3) == 1
    R_values(count,8) = 1;
end
if R_values(count,4) == 1
    R_values(count,9) = 1;
end
if R_values(count,5) == 1
    R_values(count,10) = 1;
end
if R_values(count,2) == 2
    R_values(count,7) = 3;
end
if R_values(count,3) == 2
    R_values(count,8) = 3;
end
if R_values(count,4) == 2
    R_values(count,9) = 3;
end
if R_values(count,5) == 2
    R_values(count,10) = 3;
end
if R_values(count,2) == 3
    R_values(count,7) = 7;
end
if R_values(count,3) == 3
    R_values(count,8) = 7;
end
if R_values(count,4) == 3
    R_values(count,9) = 7;
end
if R_values(count,5) == 3
    R_values(count,10) = 7;
end
if R_values(count,2) == 4
    R_values(count,7) = 16;
end
if R_values(count,3) == 4
    R_values(count,8) = 16;
```

```
                end
                if R_values(count,4) == 4
                    R_values(count,9) = 16;
                end
                if R_values(count,5) == 4
                    R_values(count,10) = 16;
                end
                count = count+1;
                    end
                end
            end
        end
    end
end

for x = 1:626
    for y = x+1:626
        if R_values(x,1) == R_values(y,1)
            R_values(y,1) = 0;
        end
    end
end

count2 = 1;
for z = 1:626
    if R_values(z,1) ~= 0
        R_values2(count2,:) = R_values(z,:);
        count2 = count2+1;
    end
end
Final_combinations = sortrows(R_values2,1);
xlswrite('Resistance_values_and_on_resistors',Final_combinations);
```

Appendix C

Resistor Values from Calculations

Resistance	On resistors				Current	Binary Value			
0.013888889	4				72	16			
0.014084507	4	4	4	3	71	16	16	16	7
0.014285714	4	4	3	4	70	16	16	7	16
0.014492754	4	4	3	3	69	16	16	7	7
0.0147	4	4	2	4	68	16	16	3	16
0.0149	4	3	4	4	67	16	7	16	16
0.0152	4	3	4	3	66	16	7	16	7
0.0154	4	3	3	4	65	16	7	7	16
0.0156	4	3	3	3	64	16	7	7	7
0.0159	4	3	2	4	63	16	7	3	16
0.0161	3	4	4	4	62	7	16	16	16
0.0164	3	4	4	3	61	7	16	16	7
0.0167	3	4	3	4	60	7	16	7	16
0.0169	3	4	3	3	59	7	16	7	7
0.0172	3	4	2	4	58	7	16	3	16
0.0175	3	3	4	4	57	7	7	16	16
0.0179	3	3	4	3	56	7	7	16	7
0.0182	3	3	3	4	55	7	7	7	16
0.0185	3	3	3	3	54	7	7	7	7
0.0189	3	3	2	4	53	7	7	3	16
0.0192	2	4	4	4	52	3	16	16	16
0.0196	2	4	4	3	51	3	16	16	7
0.0200	2	4	3	4	50	3	16	7	16
0.0204	2	4	3	3	49	3	16	7	7
0.0208	2	4	2	4	48	3	16	3	16
0.0213	2	3	4	4	47	3	7	16	16
0.0217	2	3	4	3	46	3	7	16	7
0.0222	2	3	3	4	45	3	7	7	16
0.0227	2	3	3	3	44	3	7	7	7
0.0233	2	3	2	4	43	3	7	3	16
0.0238	1	4	4	4	42	1	16	16	16
0.0244	1	4	4	3	41	1	16	16	7
0.0250	1	4	3	4	40	1	16	7	16
0.0256	1	4	3	3	39	1	16	7	7
0.0263	1	4	2	4	38	1	16	3	16
0.0270	1	3	4	4	37	1	7	16	16

Figure C.1: Calculated resistance values

Appendix D

First control board code

```
/*-------------------------------------------------------------------------
|   File:    actual_main.c
|   Author: Justin Moses
|
|   This program is for the PIC16F887 microcontroller used on the load control
|   board for Auburn University. Connects through USART to computer at 9600 ba
|   rate and can transfer and receive data. Boards are connected to each other
|   through SPI.
|
|   Procedure for starting and running device:
|
|   1. If first run, select master or slave (don't have to worry after first
|      time)
|
|      Buttons on board
|        _     _     _
|       |_|   |_|   |_|
|         |     |     |
|      slv    mstr   slv
|
|   2. Set output from 0-144 amps (0-999 possible in code, but not with curren
|      resistor configuration)
|   3. Press enter or send newline to clear previous desired output and load
|      newest desired output
|   4. Press 'g' to output current desired output
|   5. Press 'x' to output 0 (only use 1 resistor) to stop program or in case
|      emergency
|   6. Steps 2-5 can be repeated at any time when the code is running
|
|   On board variable resistor cirrently doesn't do anything
|-------------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <htc.h>
```

```
#include <pic.h>

//Set configureation bits
__CONFIG(FOSC_INTRC_NOCLKOUT & WDTE_OFF & PWRTE_OFF & MCLRE_ON & CP_OFF & CPD
__CONFIG(BOR4V_BOR40V & WRT_OFF);

int slave = 0;
int s_m_selected = 0;
int time = 25000;
unsigned char eeprom_address = 0x0000;
unsigned char data;
int eeprom_written = 0;
int master = 0;
int comm_clk;
char spi_out;
char spi_input;
char usart_in;
int usart_out;
int usart_first_comm = 0;
int spi_first_comm = 0;
unsigned char ch;
int ascii_in;
int out;
int out2;
int rx_yes = 0;
int place = 0;
int dec_in_hundreds = 0;
int dec_in_tens = 0;
int dec_in_ones = 0;
int total_output = 0x0001;
int total_output2 = 0x0000;

//*****************************************************************************
//PORTE -- 1ohm resistors
//PORTB -- .5 and .2 resistors
//PORTA -- .1ohm resistors
//PORTD -- 15 segment display
void output()
{
        switch (out)
    {
         case 2:
             PORTE = 0x03;
             PORTB = 0x00;
             PORTA = 0x00;
```

```
        PORTD = 0x20;
        break;
    case 3:
        PORTE = 0x07;
        PORTB = 0x00;
        PORTA = 0x00;
        PORTD = 0x30;
        break;
   case 4:
        PORTE = 0x0F;
        PORTB = 0x00;
        PORTA = 0x00;
        PORTD = 0x40;
        break;
    case 5:
        PORTE = 0x07;
        PORTB = 0x10;
        PORTA = 0x00;
        PORTD = 0x50;
        break;
    case 6:
        PORTE = 0x0F;
        PORTB = 0x10;
        PORTA = 0x00;
        PORTD = 0x60;
        break;
    case 7:
        PORTE = 0x07;
        PORTB = 0x30;
        PORTA = 0x00;
        PORTD = 0x70;
        break;
    case 8:
        PORTE = 0x0F;
        PORTB = 0x30;
        PORTA = 0x00;
        PORTD = 0x80;
        break;
    case 9:
        PORTE = 0x07;
        PORTB = 0x70;
        PORTA = 0x00;
        PORTD = 0x90;
        break;
    case 10:
```

```
        PORTE = 0x0F;
        PORTB = 0x70;
        PORTA = 0x00;
        PORTD = 0x01;
        break;
    case 11:
        PORTE = 0x07;
        PORTB = 0xF0;
        PORTA = 0x00;
        PORTD = 0x11;
        break;
    case 12:
        PORTE = 0x0F;
        PORTB = 0xF0;
        PORTA = 0x00;
        PORTD = 0x21;
        break;
    case 13:
        PORTE = 0x0F;
        PORTB = 0x31;
        PORTA = 0x00;
        PORTD = 0x31;
        break;
    case 14:
        PORTE = 0x07;
        PORTB = 0x71;
        PORTA = 0x00;
        PORTD = 0x41;
        break;
    case 15:
        PORTE = 0x0F;
        PORTB = 0x71;
        PORTA = 0x00;
        PORTD = 0x51;
        break;
    case 16:
        PORTE = 0x07;
        PORTB = 0xF1;
        PORTA = 0x00;
        PORTD = 0x61;
        break;
    case 17:
        PORTE = 0x0F;
        PORTB = 0xF1;
        PORTA = 0x00;
```

```
        PORTD = 0x71;
        break;
    case 18:
        PORTE = 0x0F;
        PORTB = 0x33;
        PORTA = 0x00;
        PORTD = 0x81;
        break;
    case 19:
        PORTE = 0x07;
        PORTB = 0x73;
        PORTA = 0x00;
        PORTD = 0x91;
        break;
    case 20:
        PORTE = 0x0F;
        PORTB = 0x73;
        PORTA = 0x00;
        PORTD = 0x02;
        break;
    case 21:
        PORTE = 0x07;
        PORTB = 0xF3;
        PORTA = 0x00;
        PORTD = 0x12;
        break;
    case 22:
        PORTE = 0x0F;
        PORTB = 0xF3;
        PORTA = 0x00;
        PORTD = 0x22;
        break;
    case 23:
        PORTE = 0x0F;
        PORTB = 0x37;
        PORTA = 0x00;
        PORTD = 0x32;
        break;
    case 24:
        PORTE = 0x07;
        PORTB = 0x77;
        PORTA = 0x00;
        PORTD = 0x42;
        break;
    case 25:
```

```
        PORTE = 0x0F;
        PORTB = 0x77;
        PORTA = 0x00;
        PORTD = 0x52;
        break;
    case 26:
        PORTE = 0x07;
        PORTB = 0xF7;
        PORTA = 0x00;
        PORTD = 0x62;
        break;
    case 27:
        PORTE = 0x0F;
        PORTB = 0xF7;
        PORTA = 0x00;
        PORTD = 0x72;
        break;
    case 28:
        PORTE = 0x0F;
        PORTB = 0x3F;
        PORTA = 0x00;
        PORTD = 0x82;
        break;
    case 29:
        PORTE = 0x07;
        PORTB = 0x7F;
        PORTA = 0x00;
        PORTD = 0x92;
        break;
    case 30:
        PORTE = 0x0F;
        PORTB = 0x7F;
        PORTA = 0x00;
        PORTD = 0x03;
        break;
    case 31:
        PORTE = 0x07;
        PORTB = 0xFF;
        PORTA = 0x00;
        PORTD = 0x13;
        break;
    case 32:
        PORTE = 0x0F;
        PORTB = 0xFF;
        PORTA = 0x00;
```

```
        PORTD = 0x23;
        break;
    case 33:
        PORTE = 0x0F;
        PORTB = 0x37;
        PORTA = 0x10;
        PORTD = 0x33;
        break;
    case 34:
        PORTE = 0x07;
        PORTB = 0x77;
        PORTA = 0x10;
        PORTD = 0x43;
        break;
    case 35:
        PORTE = 0x0F;
        PORTB = 0x77;
        PORTA = 0x10;
        PORTD = 0x53;
        break;
    case 36:
        PORTE = 0x07;
        PORTB = 0xF7;
        PORTA = 0x10;
        PORTD = 0x63;
        break;
    case 37:
        PORTE = 0x0F;
        PORTB = 0xF7;
        PORTA = 0x10;
        PORTD = 0x73;
        break;
    case 38:
        PORTE = 0x0F;
        PORTB = 0x3F;
        PORTA = 0x10;
        PORTD = 0x83;
        break;
    case 39:
        PORTE = 0x07;
        PORTB = 0x7F;
        PORTA = 0x10;
        PORTD = 0x93;
        break;
    case 40:
```

```
        PORTE = 0x0F;
        PORTB = 0x7F;
        PORTA = 0x10;
        PORTD = 0x04;
        break;
    case 41:
        PORTE = 0x07;
        PORTB = 0xFF;
        PORTA = 0x10;
        PORTD = 0x14;
        break;
    case 42:
        PORTE = 0x0F;
        PORTB = 0xFF;
        PORTA = 0x10;
        PORTD = 0x24;
        break;
    case 43:
        PORTE = 0x0F;
        PORTB = 0x37;
        PORTA = 0x30;
        PORTD = 0x34;
        break;
    case 44:
        PORTE = 0x07;
        PORTB = 0x77;
        PORTA = 0x30;
        PORTD = 0x44;
        break;
    case 45:
        PORTE = 0x0F;
        PORTB = 0x77;
        PORTA = 0x30;
        PORTD = 0x54;
        break;
    case 46:
        PORTE = 0x07;
        PORTB = 0xF7;
        PORTA = 0x30;
        PORTD = 0x64;
        break;
    case 47:
        PORTE = 0x0F;
        PORTB = 0xF7;
        PORTA = 0x30;
```

```
        PORTD = 0x74;
        break;
    case 48:
        PORTE = 0x0F;
        PORTB = 0x3F;
        PORTA = 0x30;
        PORTD = 0x84;
        break;
    case 49:
        PORTE = 0x07;
        PORTB = 0x7F;
        PORTA = 0x30;
        PORTD = 0x94;
        break;
    case 50:
        PORTE = 0x0F;
        PORTB = 0x7F;
        PORTA = 0x30;
        PORTD = 0x05;
        break;
    case 51:
        PORTE = 0x07;
        PORTB = 0xFF;
        PORTA = 0x30;
        PORTD = 0x15;
        break;
    case 52:
        PORTE = 0x0F;
        PORTB = 0xFF;
        PORTA = 0x30;
        PORTD = 0x25;
        break;
    case 53:
        PORTE = 0x0F;
        PORTB = 0x37;
        PORTA = 0x70;
        PORTD = 0x35;
        break;
    case 54:
        PORTE = 0x07;
        PORTB = 0x77;
        PORTA = 0x70;
        PORTD = 0x45;
        break;
    case 55:
```

```
        PORTE = 0x0F;
        PORTB = 0x77;
        PORTA = 0x70;
        PORTD = 0x55;
        break;
    case 56:
        PORTE = 0x07;
        PORTB = 0xF7;
        PORTA = 0x70;
        PORTD = 0x65;
        break;
    case 57:
        PORTE = 0x0F;
        PORTB = 0xF7;
        PORTA = 0x70;
        PORTD = 0x75;
        break;
    case 58:
        PORTE = 0x0F;
        PORTB = 0x3F;
        PORTA = 0x70;
        PORTD = 0x85;
        break;
    case 59:
        PORTE = 0x07;
        PORTB = 0x7F;
        PORTA = 0x70;
        PORTD = 0x95;
        break;
    case 60:
        PORTE = 0x0F;
        PORTB = 0x7F;
        PORTA = 0x70;
        PORTD = 0x06;
        break;
    case 61:
        PORTE = 0x07;
        PORTB = 0xFF;
        PORTA = 0x70;
        PORTD = 0x16;
        break;
    case 62:
        PORTE = 0x0F;
        PORTB = 0xFF;
        PORTA = 0x70;
```

```
        PORTD = 0x26;
        break;
    case 63:
        PORTE = 0x0F;
        PORTB = 0x37;
        PORTA = 0xF0;
        PORTD = 0x36;
        break;
    case 64:
        PORTE = 0x07;
        PORTB = 0x77;
        PORTA = 0xF0;
        PORTD = 0x46;
        break;
    case 65:
        PORTE = 0x0F;
        PORTB = 0x77;
        PORTA = 0xF0;
        PORTD = 0x56;
        break;
    case 66:
        PORTE = 0x07;
        PORTB = 0xF7;
        PORTA = 0xF0;
        PORTD = 0x66;
        break;
    case 67:
        PORTE = 0x0F;
        PORTB = 0xF7;
        PORTA = 0xF0;
        PORTD = 0x76;
        break;
    case 68:
        PORTE = 0x0F;
        PORTB = 0x3F;
        PORTA = 0xF0;
        PORTD = 0x86;
        break;
    case 69:
        PORTE = 0x07;
        PORTB = 0x7F;
        PORTA = 0xF0;
        PORTD = 0x96;
        break;
    case 70:
```

```
            PORTE = 0x0F;
            PORTB = 0x7F;
            PORTA = 0xF0;
            PORTD = 0x07;
            break;
        case 71:
            PORTE = 0x07;
            PORTB = 0xFF;
            PORTA = 0xF0;
            PORTD = 0x17;
            break;
        case 72:
            PORTE = 0x0F;
            PORTB = 0xFF;
            PORTA = 0xF0;
            PORTD = 0x27;
            break;
        default:
            PORTE = 0x01;
            PORTB = 0x00;
            PORTA = 0x00;
            PORTD = 0x10;
            break;
    }
}

//*****************************************************************************
int delay(time)
{
    while (time != 0, time--){}//delays
}

//*****************************************************************************
void eeprom_write1(unsigned char eeprom_add, unsigned char eeprom_data)
{
    GIE = 0;                    //disable interrupts
    EEADR = eeprom_add;         // set the address location in EEADR
    EEDATA = eeprom_data;       // data to be written is taken in EEDATA
    WREN = 1;                   //enable write
    EECON2 = 0x55;              //
    EECON2 = 0xAA;              //
    WR = 1;                     //
    WREN = 0;                   //disable write
    while(WR == 1);             // wait until write procedure gets completed.
    GIE = 1;                    //enable interrupts
```

```
}

//*********************************************************************************
void init_usart()
{
    TXEN = 1;                       //enable TX port
    SENDB = 0;                      //no byte sent
    BRGH = 0;                       //Fosc/16
    BRG16 = 1;                      //don't divide by 16
    SPBRG = 55;                     //9600 baud rate
    SPEN = 1;                       //sereal port enabled
    RCIE = 1;                       //enable RC interrupt
    PEIE = 1;                       //enable usart interrupt
    GIE = 1;                        //enable inerrupts
    RX9 = 0;                        //no 9th bit
    CREN = 1;                       //continuous recieve
    SYNC = 0;                       //synchronous
    CSRC = 1;                       //device cinfigured as slave
    TX9 = 0;                        //no 9th bit
    RCIF = 0;                       //Clear RCIF Flag
    TXIF = 0;                       //Clear TXIF flag
    SCKP = 0;                       //not synchronous

}


//*********************************************************************************
void usart_tx(unsigned int usart_out)
{
    //transformation//
    unsigned int nibble = usart_out & 0x000F;   //pick only first 4 bits
    unsigned int ascii[4];                      //ascii variable for storing
    int counter = 0;                            //set at 0
    for (counter = 0; counter < 2; counter ++)  //fills in ascii values
   {
        if (nibble > 9)                         //if a leter
        {
            ascii[1-counter] = nibble + 55;     //input letter
        }
        else                                    //if numeral
        {
            ascii[1-counter] = nibble + 48;     //save numeral
        }
        usart_out = usart_out >> 4;             //save next nibble
        nibble = usart_out & 0x000F;            //set nibble
```

```
    }

    TXREG = ascii[1];              //set and start usart transmit
    while(!TRMT)                   //wait while transmitting
        continue;
}

//*****************************************************************************
int usart_rx()
{
    while(!RCIF)
        continue;
    usart_in = RCREG;              //recieve usart data
    usart_first_comm = 1;          //shows usart has communicated at least once
    CREN = 0;                      //disable continous receive
    CREN = 1;                      //enable continuous receive
    rx_yes = 1;                    //set button pressed to 1
    return;                        //store data
}

//*****************************************************************************
// initialize SPI register for master and slave
void init_spi()
{
    if (master == 1)
    {
        SSPSTAT = 0b11000000;      //SMP & CKE = 1
        SSPCON = 0b00100000;       //serial port enabled, clock idle state is lo
    }
    if (master == 0)
    {
        SSPSTAT = 0b01000000;      //SMP = 0, CKE = 1
        SSPCON = 0b00100101;       //serial port enabled, clock idle state is lo
    }
}

//*****************************************************************************
//function to send a byte in SPI
int send_spi(out)
{
    int dummy;                     //initialize "fake" byte
    SSPBUF = out;                  //outputs over SPI
    while (BF)                     //wait until done
        continue;
    dummy = SSPBUF;                //discard recieved value
```

```
}

//*****************************************************************************
//function to receive a byte in SPI
int rec_spi()
{
    while (!BF)                 //waits until SPI is finished
        continue;
    out = SSPBUF;               //stores SPI value
    return;
}

//*****************************************************************************
//Interrupt routine for communication
static void interrupt uart_ISR(void) //
{                               //
    if (RCIF)                   //
    {                           //
        RCIF = 1;               //clear interrupt flag
        usart_rx();             //goto usart read
    }                           //
}

//*****************************************************************************
//sets the desired current output places for the incoming USART numbers
int dec_in(char usart_in)
{
    if (place == 1)             //sets number for the hundreths place
    {
        switch (usart_in)
        {
            case 49:
                dec_in_hundreds = 0x0064;//hex for 100
                break;
            case 50:
                dec_in_hundreds = 0x00C8;//hex for 200
                break;
            case 51:
                dec_in_hundreds = 0x012C;//hex for 300
                break;
            case 52:
                dec_in_hundreds = 0x0190;//hex for 400
                break;
            case 53:
                dec_in_hundreds = 0x01F4;//hex for 500
```

```
            break;
        case 54:
            dec_in_hundreds = 0x0258;//hex for 600
            break;
        case 55:
            dec_in_hundreds = 0x02BC;//hex for 700
            break;
        case 56:
            dec_in_hundreds = 0x0320;//hex for 800
            break;
         case 57:
            dec_in_hundreds = 0x0384;//hex for 900
            break;
         default:
            dec_in_hundreds = 0x0000;//hex for 000
            break;
    }
}
if (place == 2)                         //sets tens pace
{
    switch (usart_in)
    {
        case 49:
            dec_in_tens = 0x000A;    //hex for 10
            break;
        case 50:
            dec_in_tens = 0x0014;    //hex for 20
            break;
        case 51:
            dec_in_tens = 0x001E;    //hex for 30
            break;
        case 52:
            dec_in_tens = 0x0028;    //hex for 40
            break;
        case 53:
            dec_in_tens = 0x0032;    //hex for 50
            break;
        case 54:
            dec_in_tens = 0x003C;    //hex for 60
            break;
        case 55:
            dec_in_tens = 0x0046;    //hex for 70
            break;
        case 56:
            dec_in_tens = 0x0050;    //hex for 80
```

```
            break;
        case 57:
            dec_in_tens = 0x005A;    //hex for 90
            break;
        default:
            dec_in_tens = 0x0000;    //hex for 00
            break;
    }
}
if (place == 3)                      //set ones place
{
    switch (usart_in)
    {
        case 49:
            dec_in_ones = 0x0001;    //hex for 1
            break;
        case 50:
            dec_in_ones = 0x0002;    //hex for 2
            break;
        case 51:
            dec_in_ones = 0x0003;    //hex for 3
            break;
        case 52:
            dec_in_ones = 0x0004;    //hex for 4
            break;
        case 53:
            dec_in_ones = 0x0005;    //hex for 5
            break;
        case 54:
            dec_in_ones = 0x0006;    //hex for 6
            break;
        case 55:
            dec_in_ones = 0x0007;    //hex for 7
            break;
        case 56:
            dec_in_ones = 0x0008;    //hex for 8
            break;
        case 57:
            dec_in_ones = 0x0009;    //hex for 9
            break;
        default:
            dec_in_ones = 0x0000;    //hex for 0
            break;
    }
}
```

```
}

//*********************************************************************
unsigned int ADCRead(ch)
{
unsigned int ADC_result;

    ADCON0=0b00000001;              //set Fosc/2, RA0, enable ADC
    ADON=1;                         //switch on the adc module
    GO_DONE=1;                      //Start conversion
    while(GO_DONE);                 //wait for the conversion to finish
    ADON=0;                         //switch off adc
    return ADRESH;                  //return final digital value
}

//*********************************************************************

void eeprom_checker()
{
    EEADR = 1;                      //set address as 1
    RD = 1;                         //read from that address
    eeprom_written = EEDATA;        //store read data

    if (eeprom_written == 1)        //check if eeprom has been written
    {
        EEADR = 0;                  //sed eeprom address to 0000
        RD = 1;                     //read byte
        master = EEDATA;            //set if master or not
        if (master == 0)            //if not master
        {
            slave = 1;              //set pic as slave
            PORTD = 0xF5;           //output 5 for Slave
            delay(time*2);          //delay
            TRISC = 0b10011000;     //set C port configuration for SPI slave
        }
        else
        {
            PORTD = 0xF3;           //output 3 for Master
            delay(time*2);          //delay
            TRISC = 0b10010000;     //set port C configuration for SPI Master
        }
    }
    else
    {
        while (s_m_selected == 0)   //if slave/master has not been selected
```

```
        {
          PORTD = 0x55;          //output 55 for Slave Select
          if (RC0 == 0 || RC2 == 0)//if outside buttons are pressed
          {
              slave = 1;         //set board as slave
              s_m_selected = 1; //set slave/master as selected
              TRISC = 0b11010000; //set C port configuration for SPI slave
              PORTD = 0xF5;      //output 5 for Slave
              delay(time);       //
              PORTD = 0x5F;      //
              delay(time);       //
              PORTD = 0xF5;      //
              delay(time);       //
              eeprom_write1(0,0x00);//save eeprom 0000 as slave
              eeprom_write1(1,0x01);//save eeprom 0001 as s/m selected
          }
          else if (RC1 == 0)     //if inside button is pressed
          {
              slave = 0;         //set board for master
              s_m_selected = 1; //set slave/master as selected
              TRISC = 0b11011000; //set port C configuration for SPI Master
              PORTD = 0xF3;      //output 3 for Master
              delay(time);       //
              PORTD = 0x3F;      //
              delay(time);       //
              PORTD = 0xF3;      //
              delay(time);       //
              eeprom_write1(0,0x01);//save eeprom 0000 as master
              eeprom_write1(1,0x01);//save eeprom 0001 as s/m selected
          }

        }
    }
}



//*****************************************************************************
void main()
{

    OSCCON = 0b01110101;         //<-|
    OSCTUNE = 0b00001111;        //<-|these set the clock to 8MHz
    INTCON=1;                    //
```

```
    /*set pin I/O*/
    TRISA = 0x01;                   //RA0 - input, all other output
    TRISB = 0x00;                   //all outputs
    TRISD = 0x00;                   //all outputs
    TRISE = 0x00;                   //all outputs

    EEPGD = 0;                      //address of first eeprom programmable bit
    EEADR = 0;                      //address of first eeprom bit

    eeprom_checker();               //checks eeprom value
    init_usart();                   //initialize serial connection and USART port
    init_spi();                     //initialize spi port

    PORTE = 0x01;                   //1 ohm always on
    PORTD = 0x00;                   //
    PORTB = 0x00;                   //
    PORTA = 0x00;                   //

/*Flash display after startu routines completed*/
    PORTD = 0xFF;                   //
    delay(time);                    //
    PORTD = 0x88;                   //
    delay(time);                    //
    PORTD = 0xFF;                   //
    delay(time);                    //
    PORTD = 0x88;                   //
    delay(time);                    //
    PORTD = 0xFF;                   //
    delay(time);                    //
    PORTD = 0x88;                   //
    delay(time);                    //

/*sit in this loop until computer is talked to, spi has been used or one of t
while (usart_first_comm == 0 && RC0 == 1  && RC1 == 1 && RC2 == 1 && spi_first
   {
        PORTE = 0x01;               //leave 1 ohm resistor on
        PORTD = 0x00;               //
        PORTB = 0x00;               //
        PORTA = 0x00;               //
        if (RC3 == 1) spi_first_comm = 1;
    }
/*main part of the main program that controls the output and communication of
   for(;;)                          //run indefinitely
    {
       if (master == 1)             //checks to see if save or master
```

```
{
if (usart_in != 0x67 &&  rx_yes == 1)//if keyboard is pressed but not
{
    if (usart_in == 13)  //if enter (newline) is pressed
    {
        total_output = dec_in_hundreds + dec_in_tens + dec_in_ones;//g
        total_output2 = total_output >> 8;//shifts the total output by
        usart_tx(total_output2 && 0x0F); //tansmit the first two numbe
        usart_tx(total_output && 0xFF);  //transmit the last number
        rx_yes = 0;       //reset button pressed
        place = 0;        //set the place in the input number to hundre
    }
    else
    {
        place++; //increment digits place
        dec_in(usart_in); //function that stores digits
        usart_tx(usart_in);//send back key pressed
        rx_yes = 0;       //set place in input number to hundreds
    }
}
if (usart_in == 0x67)     //if 'g' pressed
{
    rx_yes = 0;           //reset button pressed
    if (total_output & 0b00000001 == 1)        //if output is odd numbe
    {
        total_output2 = total_output - 1;     //subtract 1 from output
        out = total_output2/2;                //divide output in 2
        out2 = total_output2/2 + 1;           //add the 1 back to the o
    }
    else                                      //if even number
    {
        out = total_output/2;                 //divide output by 2 and
        out2 = out;                           //output the divided outp
    }
    send_spi(out2);                           //send output2 to second
    while (rx_yes == 0)                       //until another button is
    {
        output(out);                          //output desired output f
    }
}
if (usart_in == 0x78)     //if 'x' is pressed
{
    rx_yes = 0;           //set buttons pressed to 0
    out = 0;              //set output to 0
    send_spi(out);        //send output of 0 to slave
```

```
            while (rx_yes == 0)  //
            {
                output(out);     //output nothing until another button pressed
            }
        }
    }
    if (slave == 1)              //if not master pic
    {
        while (!BF)              //wait for SPI to complete
            continue;
        out = SSPBUF;            //read in SPI data
        output(out);             //output SPI data
    }
  }
    return;
}
```

Appendix E

Second load board bill of materials

Index	Manuf#	Qty
R11, R13, R33, R35, R55 R57, R77, R79	PFS35-0R05J1	8
R15, R17, R37, R39, R59 R61, R81, R83	PFS35-0R2F1	8
R19, R21, R41, R43, R63 R65, R85, R87	PFS35-0R2F1	8
R23, R25, R45, R47, R67 R69, R89, R91	PWR263S-35-R500F	8
R27, R29, R49, R51, R71 R73, R93, R95	PWR263S-35-1R00F	8
R31, R53, R75, R97	PWR263S-35-2R00F	4
R7	RMCF1206FT15R0	1
R5	ERJ-8ENF1003V	1
R1, R2, R3, R4, R6	ERJ-8ENF1002V	5
R8, R9, R10, R12, R14 R16, R18, R20, R22, R24 R26, R28, R30, R32, R34 R36, R38, R40, R42, R44 R46, R48, R50, R52, R54 R56, R58, R60, R62, R64 R66, R68, R70, R72, R74		

R76, R78, R80, R82, R84 R86, R88, R90, R92, R94 R96	ERJ-8ENF1001V	46
C10, C11	C3216X7R2A104K160AA	2
C2, C3, C4, C5, C7 C8, C9	C3216X7R1E105K085AA	7
C1	C3216X5R1A106K160AB	1
C6		1
P3, P4	3-644456-6	2
K1, K2, K3, K4, K5 K6, K7, K8, K9, K10 K11, K12, K13, K14, K15 K16, K7, K18, K19, K20 K21, K22, K23, K24, K25 K26, K27, K28, K29, K30 K31, K32, K33, K34, K35 K36, K37, K38, K39, K40 K41, K42, K43, K44	640452-3	44
Sw1, SW2, SW3, SW4	EVQ-11U04M	4
U6, U7, U8, U9, U10 U11, U12, U13, U14, U15 U16, U17, U18, U19, U20 U21, U22, U23, U24, U25 U26, U27	FAN3227TMX	22
Q1, Q2, Q3, Q4, Q5 Q6, Q7, Q8, Q9, Q10 Q11, Q12, Q13, Q14, Q15		

Q16, Q17, Q18, Q19, Q20 Q21, Q22, Q23, Q24, Q25 Q26, Q27, Q28, Q29, Q30 Q31, Q31, Q32, Q33, Q34, Q35 Q36, Q37, Q38, Q39, Q40 Q41, Q42, Q43, Q44	IRF6201TRPBF	44
D1, D2, D3, D4, D5 D6, D7, D8, D9, D10 D11, D12, D13, D14, D15 D16, D17, D18, D19, D20 D21, D22, D23, D24, D25 D26, D27, D28, D29, D30 D31, D32, D33, D34, D35 D36, D37, D38, D39, D40 D41, D42, D43, D44	LG R971-KN-1	44
U1	PIC18F87K22-I/PTRSL	1
U2	TC1015-3.3VCT713	1
U3	NHD-C0220BIZ-FS(RGB)-FBW-3VM	1
U4, U5	ACS758KCB-150B-PFF-T	2

Table E.1: Load Board V2 BOM

Appendix F

Second load board CAD

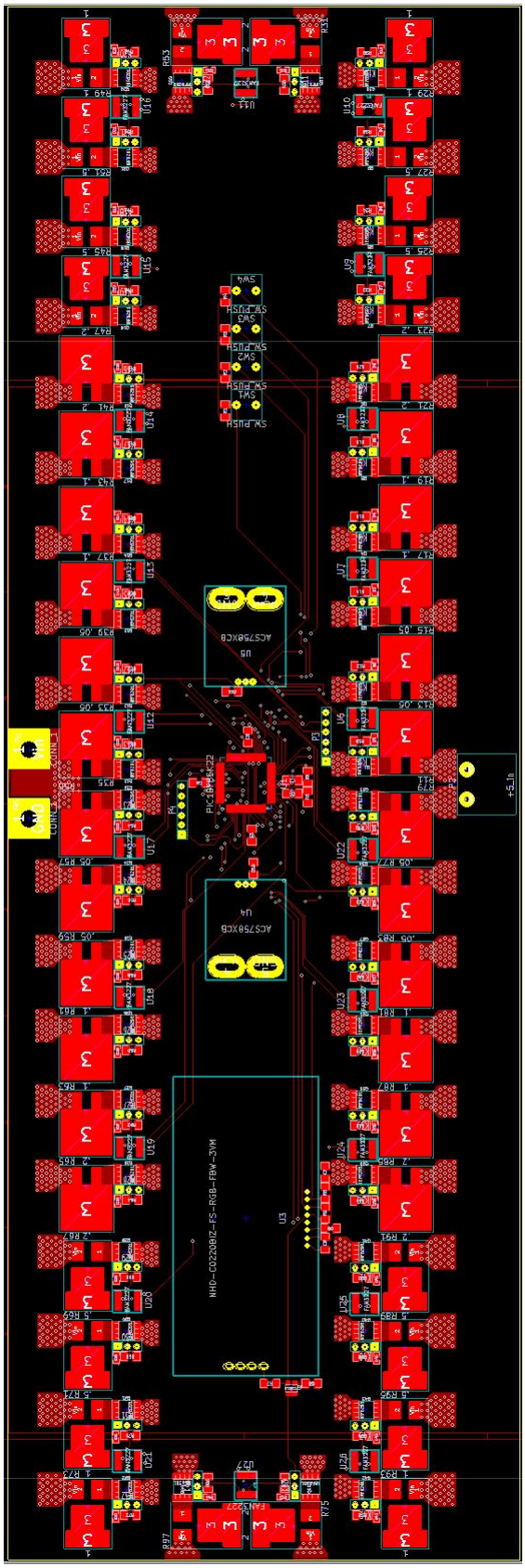

Figure F.1: V2 top copper

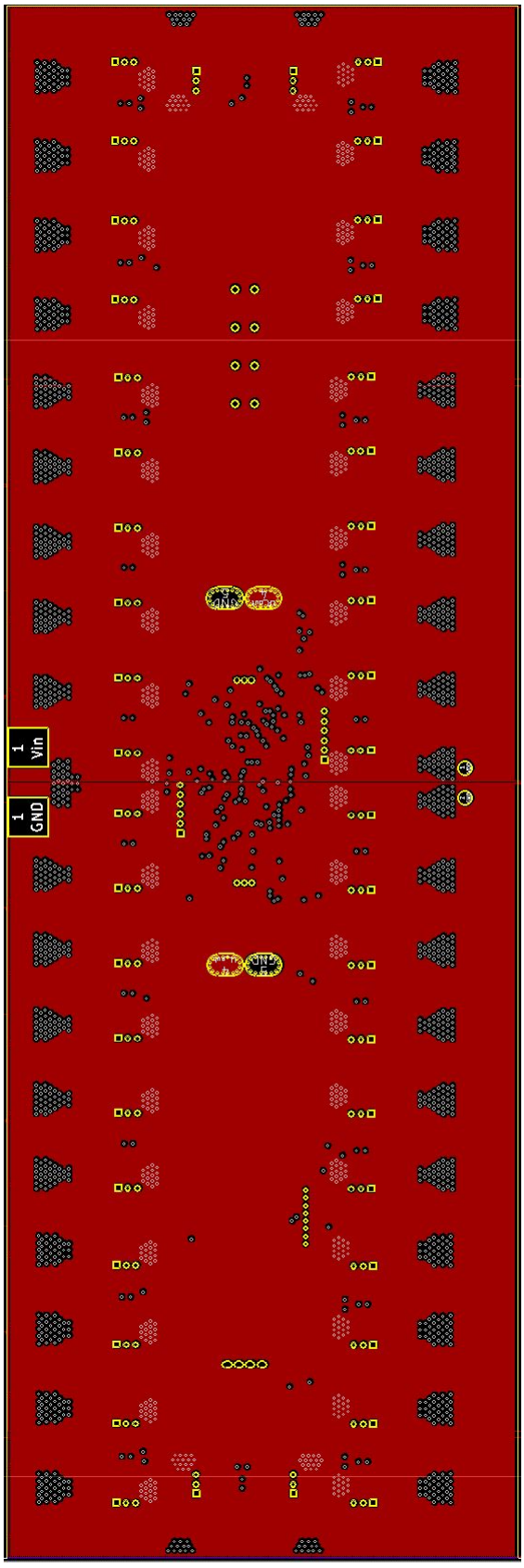

Figure F.2: V2 second layer

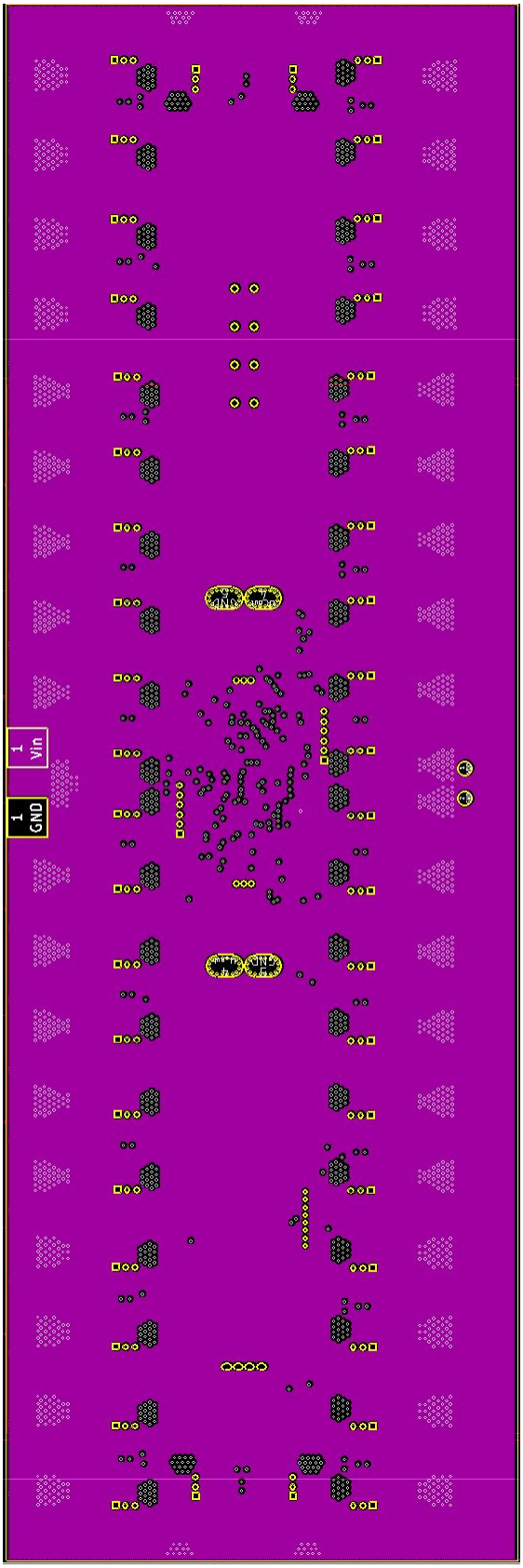

Figure F.3: V2 third layer

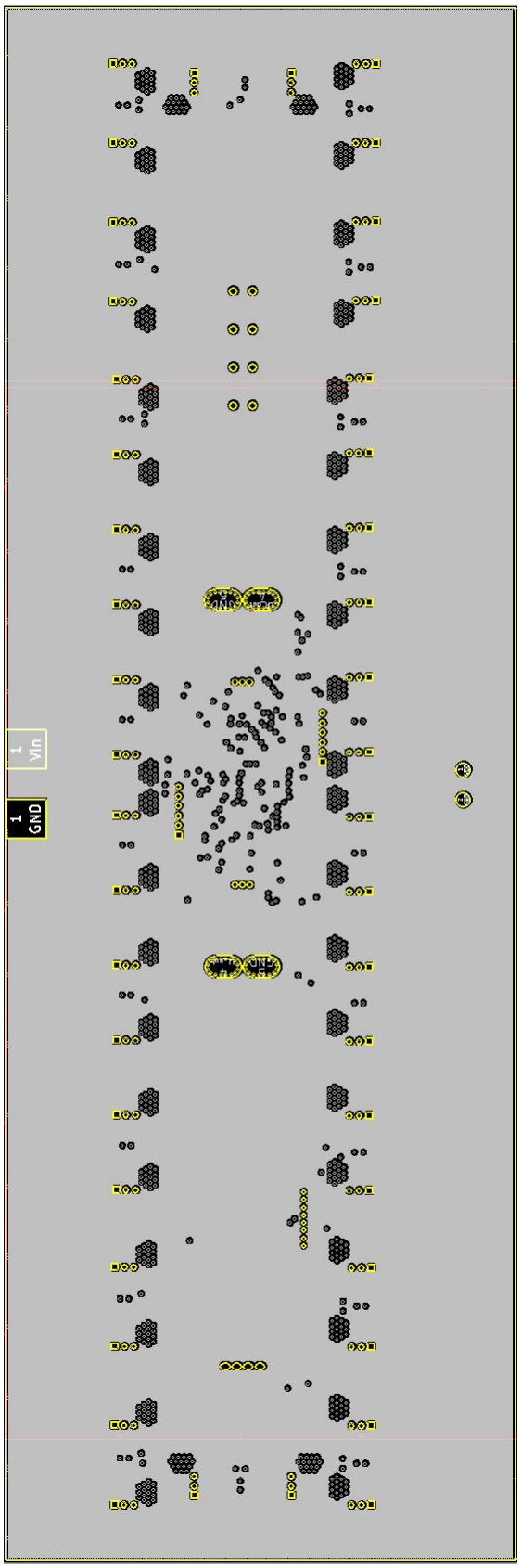

Figure F.4: V2 fourth layer

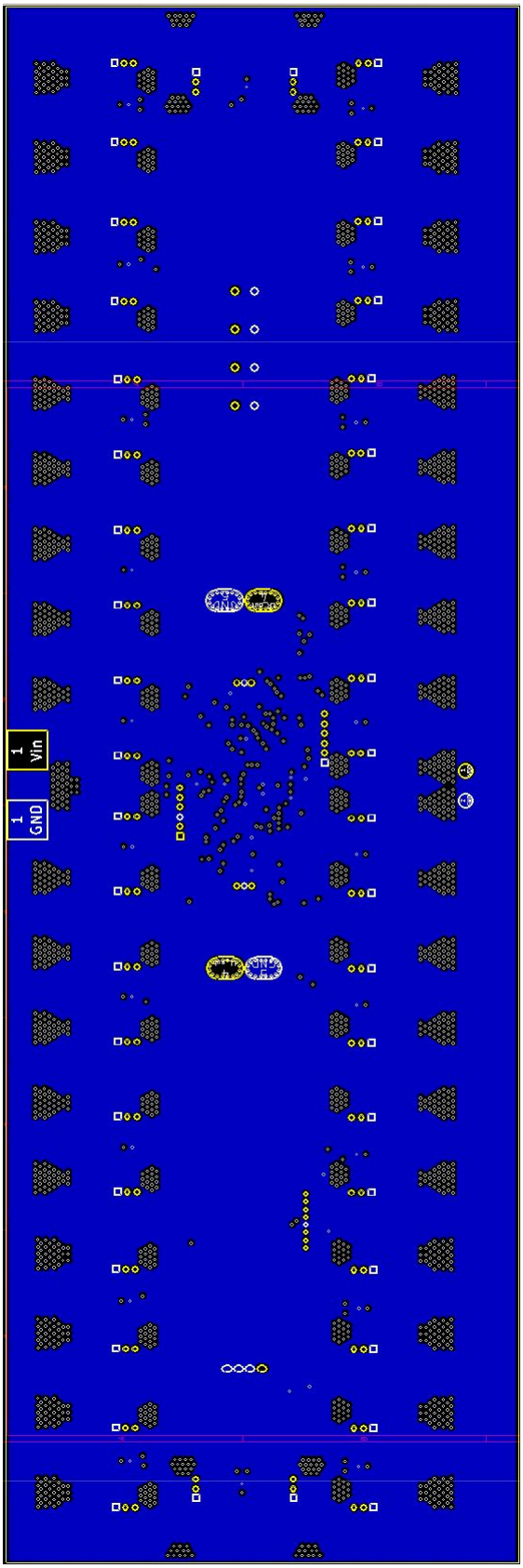

Figure F.5: V2 fifth layer

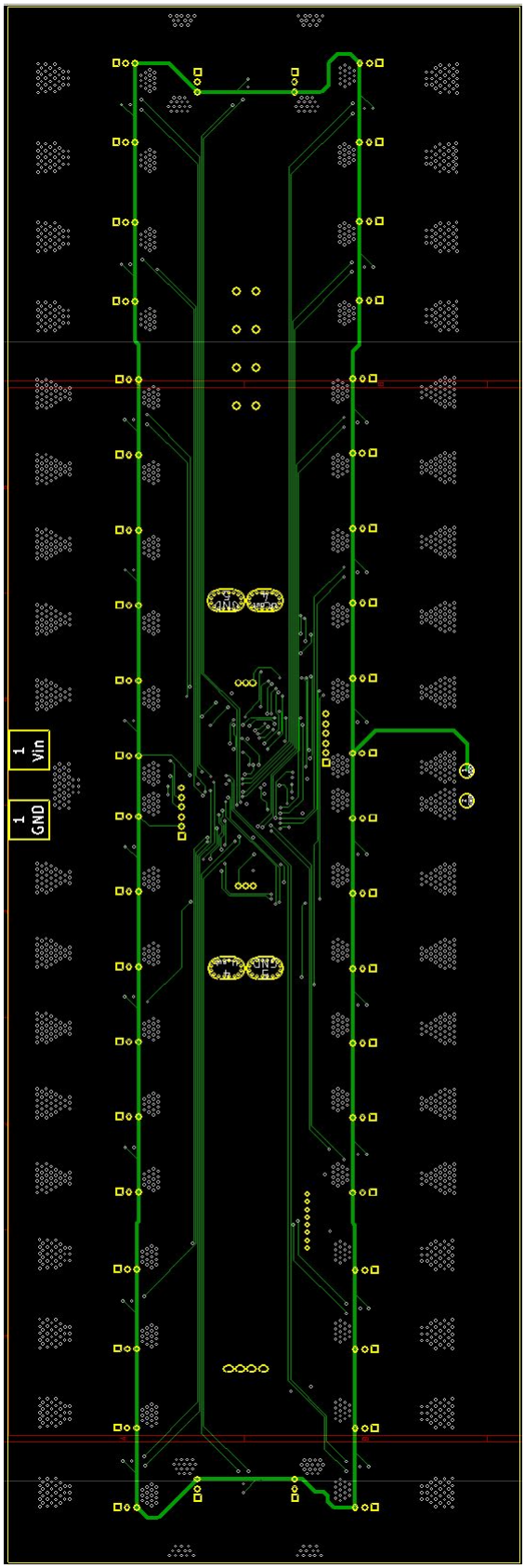

Figure F.6: V2 bottom copper

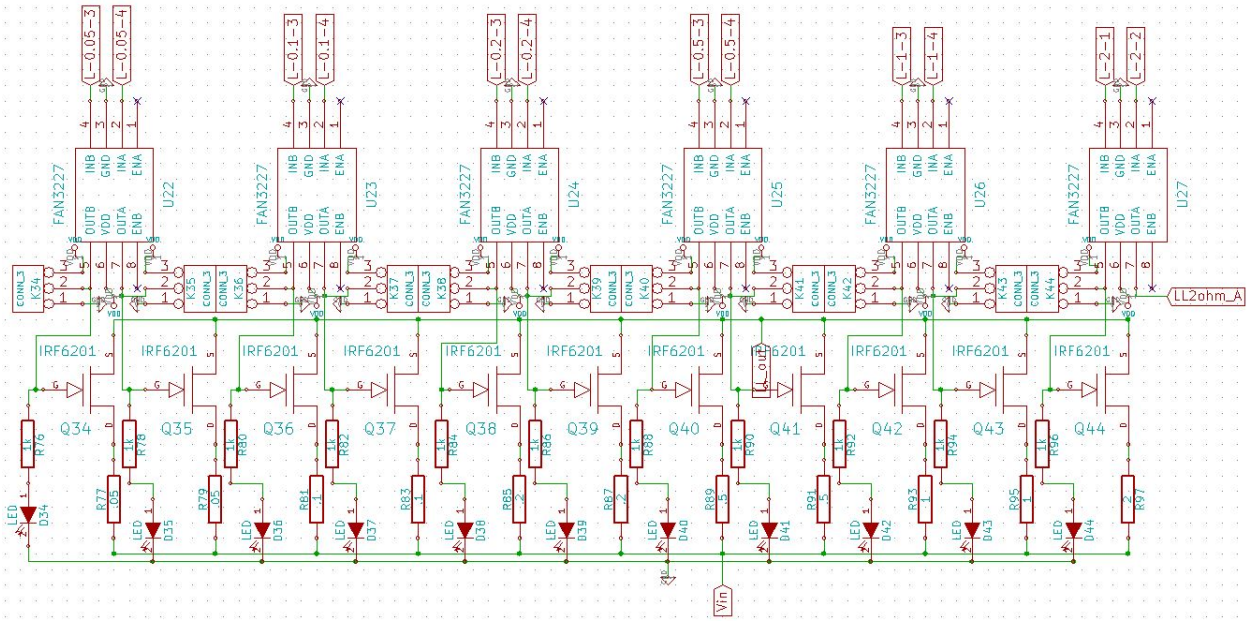

Figure F.7: V2 top left resistor schematic

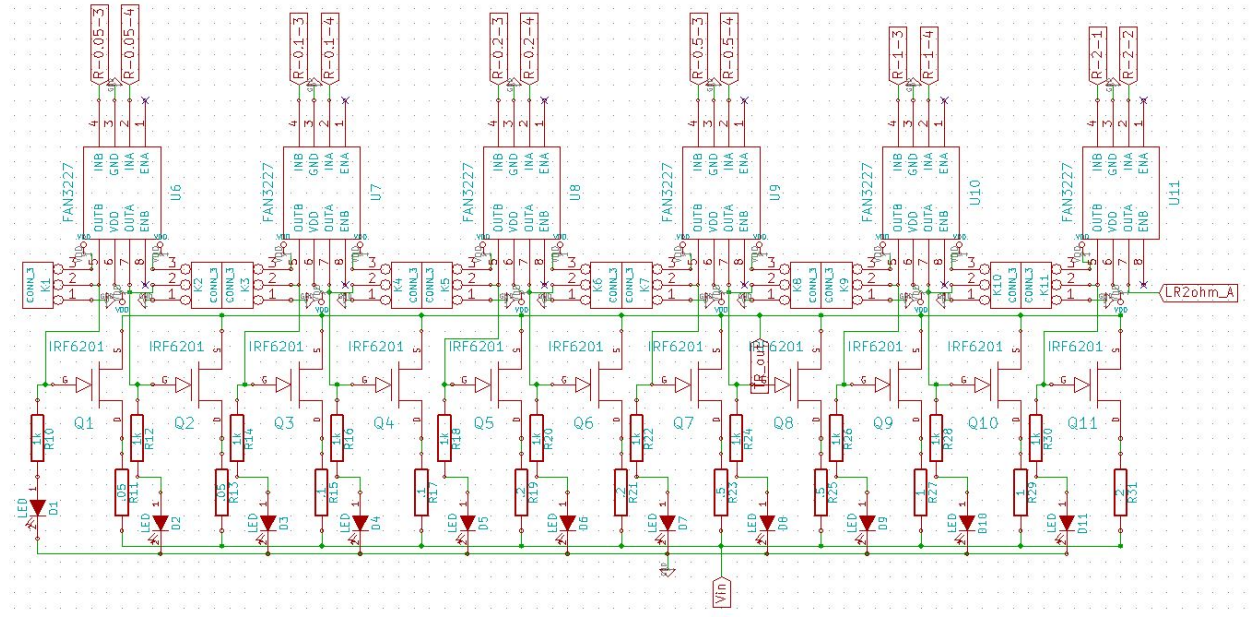

Figure F.8: V2 top right resistor schematic

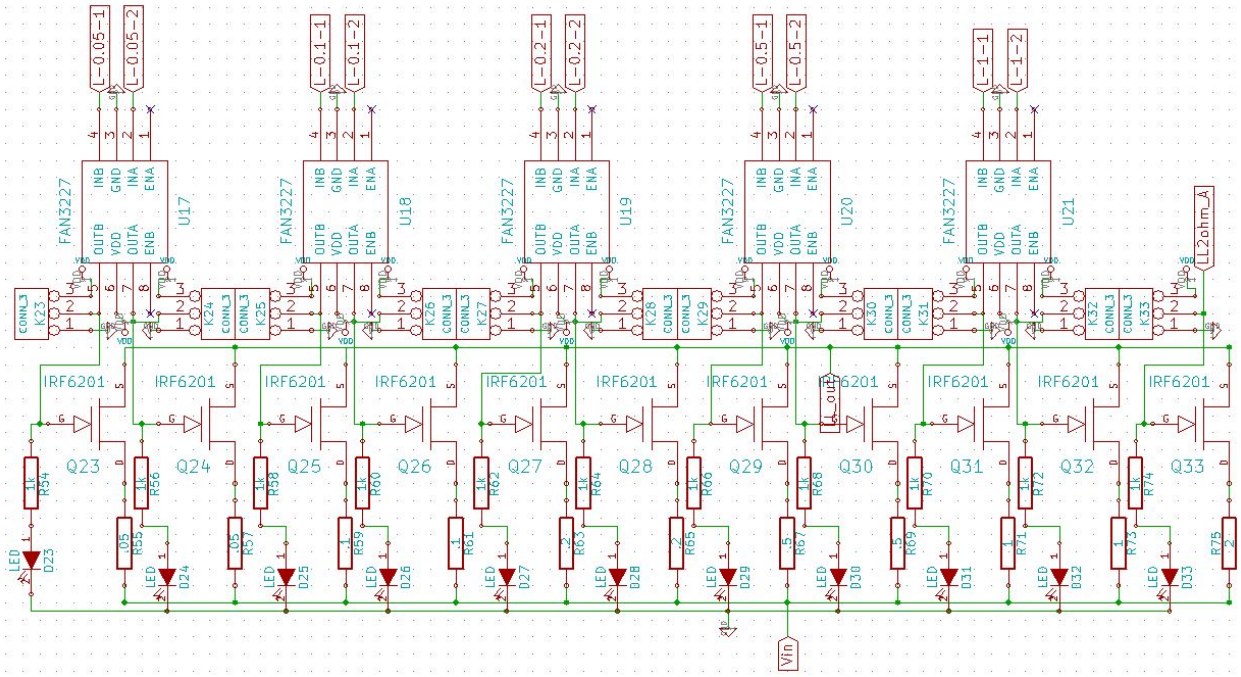

Figure F.9: V2 bottom left resistor schematic

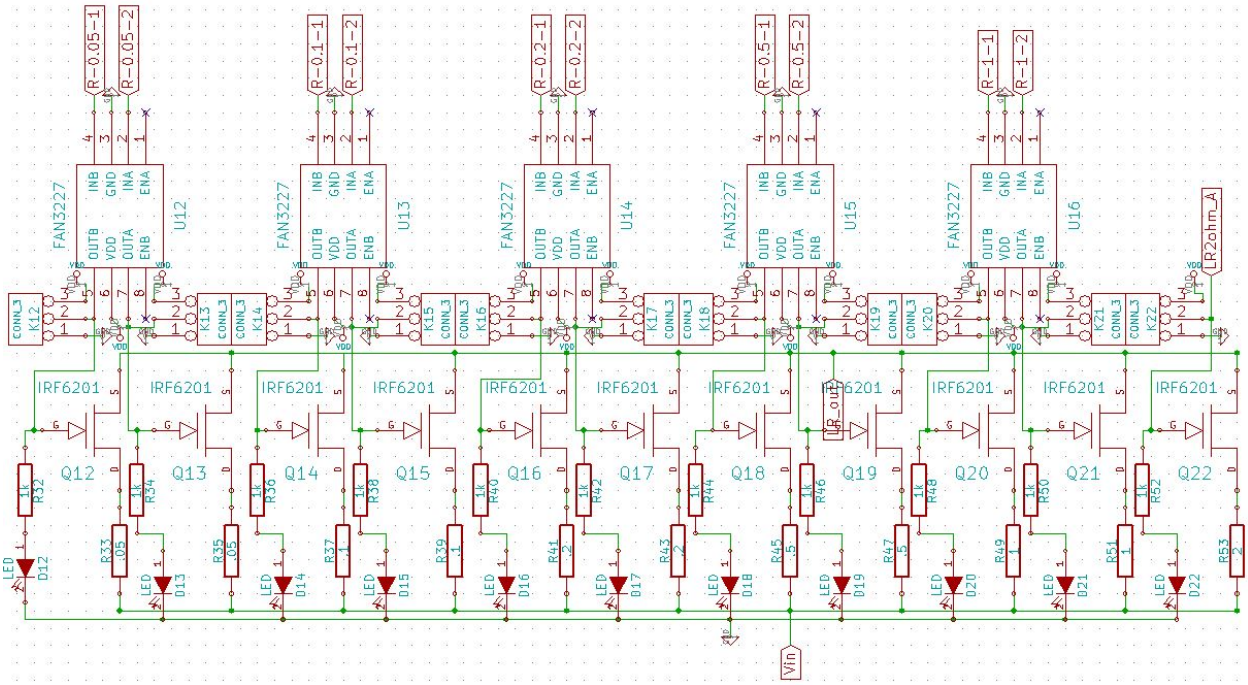

Figure F.10: V2 bottom right resistor schematic

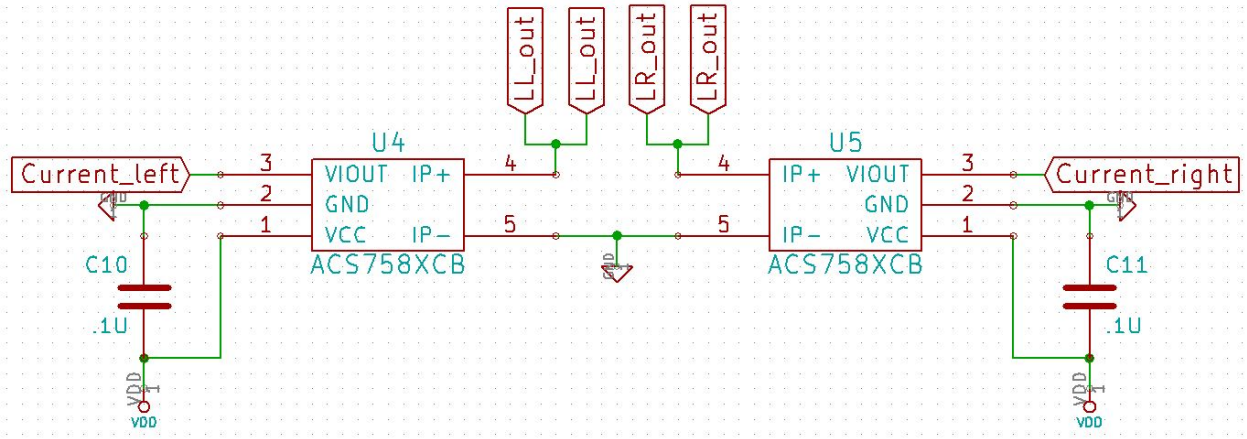

Figure F.11: V2 current sensor schematic

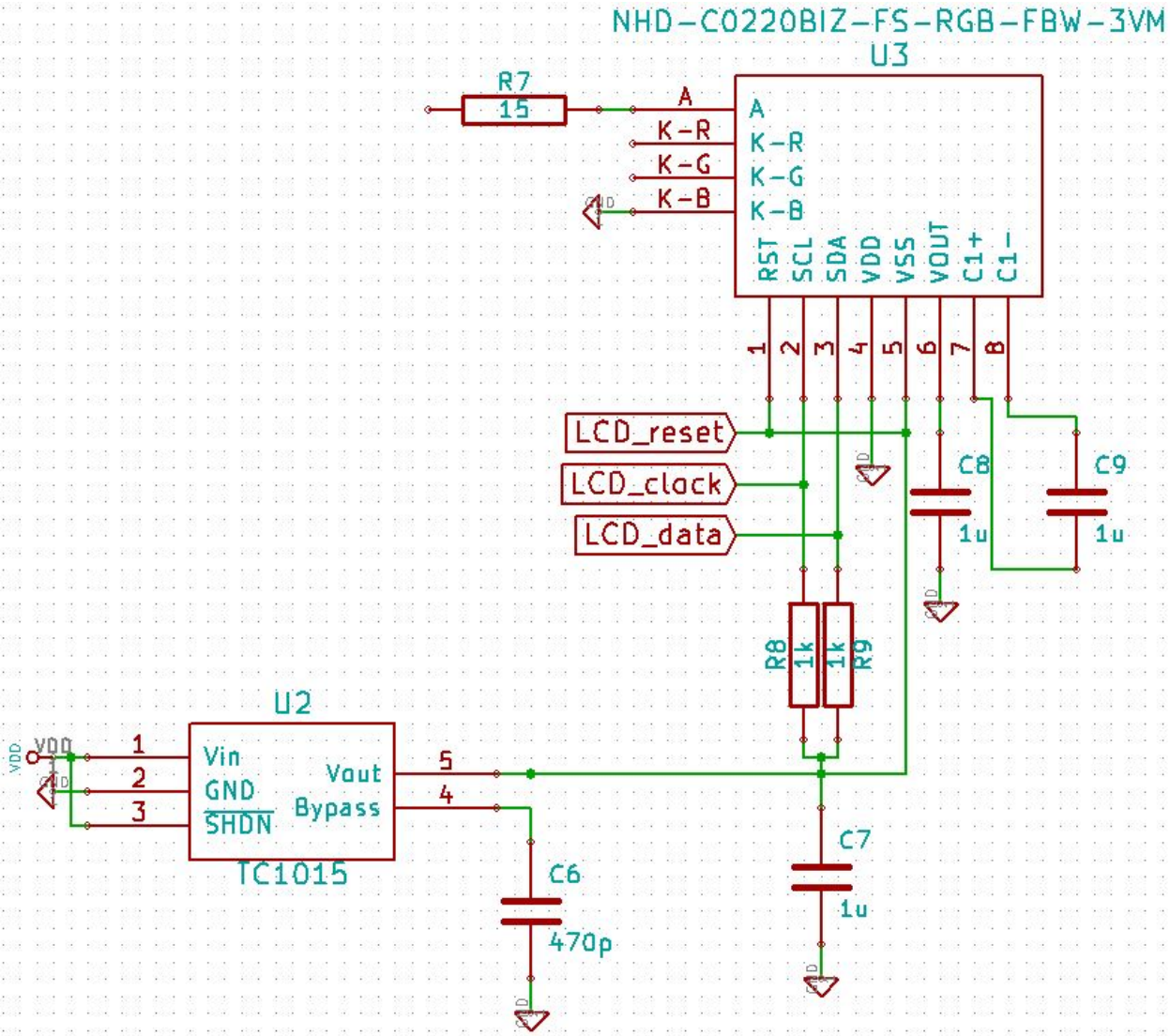

Figure F.12: V2 display schematic

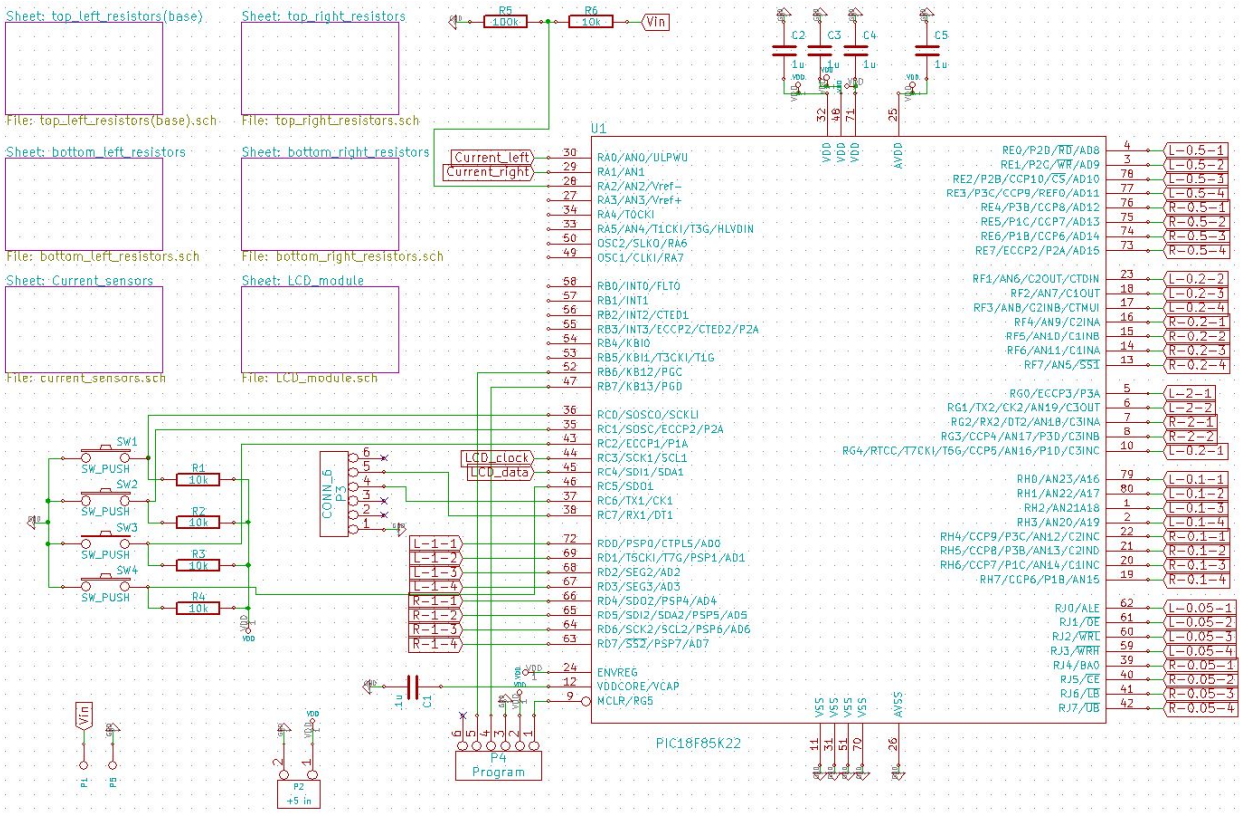

Figure F.13: V2 microcontroller and overall schematic

Appendix G

Second load board code

```
/*
 * File:   loadmain_v3_0.c
 * Author: JMoses
 *
 * Created on June 19, 2013, 3:33 PM
 *
 * PORTD - 1 ohm
 * PORTE - .5 ohm
 * PORTF - .2 ohm
 * PORTG - G0-3 2 ohm, G4 - .2ohm
 * PORTH - .1 ohm
 * PORTJ - .05 ohm
 */

#include <xc.h>
#include <plib.h>
#include <pconfig.h>
#include <usart.h>
#include <adc.h>
#include <stdio.h>
#include <stdlib.h>




#define _XTAL_FREQ 16000000
#define USE_OR_MASKS

// PIC18F87K22 Configuration Bit Settings

// CONFIG1L
#pragma config FOSC = INTIO2, SOSCSEL = DIG, XINST = OFF, IESO = OFF, WDTEN =

int usart_in;
int rx_yes = 0;
int place = 0;
```

```
int dec_in_hundreds = 0;
int dec_in_tens = 0;
int dec_in_ones = 0;
int total_output = 0x0000;
int total_output2 = 0x0000;
int tens_digit = 0;
unsigned int ADCResult[3]= {0};
float voltage;
float current;
unsigned char ResultStr[10];
unsigned char buffer[20];
unsigned char buffer2[20];
int count = 0;
float current2;
float current3;
int modified_ones = 0;

float difference = 0.0;
    int pos_20_count;
    int pos_10_count;
    int pos_5_count;
    int pos_2_count;
    int pos_1_count;
    int pos_05_count;
    int desired = 0;
    int mode = 0;        //1 for active, 2 for passive
    int button_pressed = 0;


static void interrupt uart_ISR(void) //
{                                    //
    if (RC1IF)                        //
    {                                //
        RC1IF = 1;                    //clear interrupt flag
        usart_in = getc1USART();//goto usart read
        while(Busy1USART());
        rx_yes = 1;                      //set button pressed to 1
    }                                //
}

//I2C stattup routine-------------------------------------------------------------
void i2c_start(void)
{
    PORTCbits.RC3 = 1;
    PORTCbits.RC4 = 1;
```

```
    PORTCbits.RC4 = 0;
    PORTCbits.RC3 = 0;
}

//I2C Stop routine-------------------------------------------------------------
void i2c_stop(void)
{
    PORTCbits.RC4 = 0;
    PORTCbits.RC3 = 0;
    PORTCbits.RC3 = 1;
    PORTCbits.RC4 = 1;
}

//I2C output-------------------------------------------------------------------
void i2c_out(unsigned char d)
{
    int n;

    for (n=1; n<9; n++)
    {
        if(d & 0x80)
        {
            PORTCbits.RC4 = 1;
        }
        else
        {
            PORTCbits.RC4 = 0;
        }
        __delay_us(5);
        PORTCbits.RC3 = 1;
        __delay_us(5);
        PORTCbits.RC3 = 0;

        d = d << 1;
    }

    TRISC = 0b10110111;
    PORTCbits.RC3 = 1;
    while (PORTCbits.RC4 == 1)
    {
        PORTCbits.RC3 = 0;
        PORTCbits.RC3 = 1;
    }
    PORTCbits.RC3 = 0;
    TRISC = 0b10100111;
```

```
}

//Initializig the LCD-------------------------------------------------------------
void init_lcd(void)
{
    i2c_start();

    i2c_out(0x78);
    i2c_out(0x00);
    i2c_out(0x38);
    __delay_us(25);
    i2c_out(0x39);
    __delay_us(25);
    i2c_out(0x14);
    i2c_out(0x70);
    i2c_out(0x5D);
    i2c_out(0x6D);
    i2c_out(0x0C);
    i2c_out(0x01);
    i2c_out(0x06);
    __delay_us(10);

    i2c_stop;
}

//Clearing the display-----------------------------------------------------------
void clear_display(void)
{
    i2c_start();

    i2c_out(0x78);
    i2c_out(0x00);
    i2c_out(0x01);

    i2c_stop();
}

//Displays on LCD----------------------------------------------------------------
void display (int addr, unsigned char *text)
{
    int n;
    int c = 0;

    while(*(text+c))
      c++;
```

```
    i2c_start();

    i2c_out( 0x78 );            // Slave address of panel.
    i2c_out( 0x80 );            // Next byte is command, followed by control
    i2c_out( 0x80 | addr );     // move to address addr
    i2c_out( 0x40 );

    for (n=1; n<c+1; n++)
    {
        i2c_out(*text);
        text++;
    }

    i2c_stop();
}

//Initialize the ADC-----------------------------------------------------------
void init_ADC(void)         //Configure ADC with 3 analog channels
  {
    ADCON0 = 0b00001000;
    ADCON1 = 0b00000000;
    ADCON2 = 0b10001000;

    PIR1bits.ADIF = 0;
    PIE1bits.ADIE = 0;
    INTCONbits.PEIE = 1;
    ADCON0bits.ADON = 1;
  }

//Does the conversion for each channel-----------------------------------------
void runADC(void)
{
    for(unsigned char i=0;i<=2;i++)     //Loop three times to sample and read
        {
        ADCON0bits.CHS=i;                   //Select a channel to sample (index
                                            //runs for the first time, i will be
        __delay_us(18);                       //A small delay before we convert
        ConvertADC();                           // Convert Analog to Digital
        while(BusyADC());
        ADCResult[i] = (unsigned int) ReadADC();        //Read each channel
        voltage = (ADCResult[2]*5.0)/4096;                  //convert 12-bit ADC
        current = (((ADCResult[0]-2041.0)*60.0)+((ADCResult[1]-2049.0)*60.0))
        }
}
```

```
//Transmits Characters to PC-----------------------------------------------------
void putrs1USART(const char *data)
{
    do
    {  // Transmit a byte
        while(Busy1USART());
        putc1USART(*data);
   }
    while( *data++ );
}

//Converts the ASCII to decimal--------------------------------------------------
int dec_in(char usart_in)
{
    if (place == 1)              //sets number for the hundreths place
    {
        switch (usart_in)
        {
            case 49:
                dec_in_hundreds = 0x0064;//hex for 100
                break;
            case 50:
                dec_in_hundreds = 0x00C8;//hex for 200
                break;
            case 51:
                dec_in_hundreds = 0x012C;//hex for 300
                break;
            case 52:
                dec_in_hundreds = 0x0190;//hex for 400
                break;
            case 53:
                dec_in_hundreds = 0x01F4;//hex for 500
                break;
            case 54:
                dec_in_hundreds = 0x0258;//hex for 600
                break;
            case 55:
                dec_in_hundreds = 0x02BC;//hex for 700
                break;
            case 56:
                dec_in_hundreds = 0x0320;//hex for 800
                break;
             case 57:
                dec_in_hundreds = 0x0384;//hex for 900
```

```
            break;
        default:
            dec_in_hundreds = 0x0000;//hex for 000
            break;
    }
}
if (place == 2)                         //sets tens pace
{
    switch (usart_in)
    {
        case 49:
            dec_in_tens = 0x000A;    //hex for 10
            tens_digit = 1;
            break;
        case 50:
            dec_in_tens = 0x0014;    //hex for 20
            tens_digit = 2;
            break;
        case 51:
            dec_in_tens = 0x001E;    //hex for 30
            tens_digit = 3;
            break;
        case 52:
            dec_in_tens = 0x0028;    //hex for 40
            tens_digit = 4;
            break;
        case 53:
            dec_in_tens = 0x0032;    //hex for 50
            tens_digit = 5;
            break;
        case 54:
            dec_in_tens = 0x003C;    //hex for 60
            tens_digit = 6;
            break;
        case 55:
            dec_in_tens = 0x0046;    //hex for 70
            tens_digit = 7;
            break;
        case 56:
            dec_in_tens = 0x0050;    //hex for 80
            tens_digit = 8;
            break;
        case 57:
            dec_in_tens = 0x005A;    //hex for 90
            tens_digit = 9;
```

```
                break;
            default:
                dec_in_tens = 0x0000;    //hex for 00
                tens_digit = 0;
                break;
        }
    }
    if (place == 3)                      //set ones place
    {
        switch (usart_in)
        {
            case 49:
                dec_in_ones = 0x0001;    //hex for 1
                break;
            case 50:
                dec_in_ones = 0x0002;    //hex for 2
                break;
            case 51:
                dec_in_ones = 0x0003;    //hex for 3
                break;
            case 52:
                dec_in_ones = 0x0004;    //hex for 4
                break;
            case 53:
                dec_in_ones = 0x0005;    //hex for 5
                break;
            case 54:
                dec_in_ones = 0x0006;    //hex for 6
                break;
            case 55:
                dec_in_ones = 0x0007;    //hex for 7
                break;
            case 56:
                dec_in_ones = 0x0008;    //hex for 8
                break;
            case 57:
                dec_in_ones = 0x0009;    //hex for 9
                break;
            default:
                dec_in_ones = 0x0000;    //hex for 0
                break;
        }
    }
}
```

```
//Output sequence for the active mode----------------------------------------------------
void output_active()
{
/*1 ohm resistors-------------------------------------------------------------------------
    if (total_output > 22)                  //Sets ones place rule for all output
    {
        if (dec_in_ones == 0 | dec_in_ones == 3  | dec_in_ones == 5 | dec_in_
        {
            PORTD = 0x7F;                   //turns on 3x1 ohm resistors
        }
        else
        {
            PORTD = 0xFF;                   //turns on 4x1 ohm resistors
        }
    }
    else
    {
        switch(total_output)
        {
            case 1:
                PORTD = 0x00;               //0 A
                break;
            case 2:
                PORTD = 0x00;               //0 A
                break;
            case 3:
                PORTD = 0x01;               //1 A
                break;
            case 4:
                PORTD = 0x11;
                break;
            case 5:
                PORTD = 0x13;
                break;
            case 6:
                PORTD = 0x33;
                break;
            case 7:
                PORTD = 0x37;
                break;
            case 8:
                PORTD = 0x77;
                break;
            default:
```

```
                if (dec_in_ones & 0b00000001 == 1)
                {
                    PORTD = 0x7F;
                }
                else
                {
                    PORTD = 0xFF;
                }
                break;
            }
        }
/*.5 ohm resistors-------------------------------------------------------------
    if (total_output > 22)
    {
        if (dec_in_ones == 2 | dec_in_ones == 7)
        {
            PORTE = 0x77;
        }
        else if (dec_in_ones == 3 | dec_in_ones == 4 | dec_in_ones == 8 | dec
        {
            PORTE = 0x7F;
        }
        else if (dec_in_ones == 0 | dec_in_ones == 1 | dec_in_ones == 5 | dec
        {
            PORTE = 0xFF;
        }
    }
    else
    {
        if (total_output < 11)
        {
            PORTE = 0x00;
        }
        else if (total_output == 11 | total_output == 12)
        {
            PORTE = 0x01;
        }
        else if (total_output == 13 | total_output == 14)
        {
            PORTE = 0x11;
        }
        else if (total_output == 15 | total_output == 16)
        {
            PORTE = 0x13;
        }
```

```
        else if (total_output == 17 | total_output == 18)
        {
            PORTE = 0x33;
        }
        else if (total_output == 19 | total_output == 20)
        {
            PORTE = 0x37;
        }
        else if (total_output == 21 | total_output == 22)
        {
            PORTE = 0x77;
        }
    }
/*.2 ohm resistors-------------------------------------------------------------
    if (total_output > 26)
    {
        if (total_output < 32)
        {
            PORTF = 0x11;
            PORTG = 0x0F;
        }
        else if (total_output > 31 & total_output < 37)
        {
            PORTF = 0x13;
            PORTG = 0x0F;
        }
        else if (total_output > 36 & total_output < 42)
        {
            PORTF = 0x33;
            PORTG = 0x0F;
        }
        else if (total_output > 41 & total_output < 47)
        {
            PORTF = 0x37;
            PORTG = 0x0F;
        }
        else if (total_output > 46 & total_output < 52)
        {
            PORTF = 0x77;
            PORTG = 0x0F;
        }
        else if (total_output > 51 & total_output < 57)
        {
            PORTF = 0x7F;
            PORTG = 0x0F;
```

```
        }
        else if (total_output >56)
        {
            if (dec_in_ones == 7 | dec_in_ones == 8 | dec_in_ones == 9 | dec_
            {
                PORTF = 0xFF;
                if (total_output == 1)
                {
                    PORTG = 0x05;                          //turn 1 A on
                }
                else                                       //if even number
                {
                    PORTG = 0x0F;                          //turn 2A on
                }
            }
            else
            {
                PORTF = 0xFF;
                if (total_output == 1)
                {
                   PORTG = 0x15;                          //turn 1 A on
                }
                else                                       //if even number
                {
                    PORTG = 0x1F;                          //turn 2A on
                }
            }
        }
    }
    else
    {
        PORTF = 0x00;
        if (total_output == 1)
        {
            PORTG = 0x05;                          //turn 1 A on
        }
        else                                       //if even number
        {
            PORTG = 0x0F;                          //turn 2A on
        }
    }
/*.1 ohm resistors-------------------------------------------------------------
    if (total_output > 66)
    {
        if (total_output > 66 & total_output < 77)
```

```
{
    PORTH = 0x01;
}
else if (total_output > 76 & total_output < 87)
{
    PORTH = 0x11;
}
else if (total_output > 86 & total_output < 97)
{
    PORTH = 0x13;
}
else if (total_output > 96 & total_output < 107)
{
    PORTH = 0x33;
}
else if (total_output > 106 & total_output < 117)
{
    PORTH = 0x37;
}
else if (total_output > 116 & total_output < 127)
{
    PORTH = 0x77;
}
else if (total_output > 126)
{
    modified_ones = dec_in_ones + 3;

    if (modified_ones > 9)
    {
        tens_digit = tens_digit + 1;
    }
    else
    {
        tens_digit = tens_digit + 0;
    }
    if (tens_digit & 0b00000001 == 1)
    {
        PORTH = 0x7F;
    }
    else
    {
        PORTH = 0xFF;
    }

}
```

```
    }
    else
    {
        PORTH = 0x00;
    }
/*.05 ohm resistors-------------------------------------------------------------
    if (total_output > 146)
    {
        if (total_output < 167)
        {
            PORTJ = 0x01;
        }
        else if (total_output > 166 &  total_output < 187)
        {
            PORTJ = 0x11;
        }
        else if (total_output > 186 &  total_output < 207)
        {
            PORTJ = 0x13;
        }
        else if (total_output > 206 &  total_output < 227)
        {
            PORTJ = 0x33;
        }
        else if (total_output > 226 &  total_output < 247)
        {
            PORTJ = 0x37;
        }
        else if (total_output > 246 &  total_output < 267)
        {

         PORTJ = 0x77;
        }
        else if (total_output > 266 &  total_output < 287)
        {
            PORTJ = 0x7F;
        }
        else if (total_output > 286)
        {
            PORTJ = 0xFF;
        }
    }
    else
    {
        PORTJ = 0x00;
```

```
    }
}

//Output passiveroutine to resistors and desired value on LCD----------------
void output_passive()
{
    switch(total_output)
    {
        case 0:
            PORTD = 0x00;          //1
            PORTE = 0x00;          //.5
            PORTF = 0x00;          //.2
            PORTG = 0x00;          //2,.2
            PORTH = 0x00;          //.1
            PORTJ = 0x00;          //.05
            display(0x0E,"0A");
            break;
        case 1:
            PORTD = 0x00;          //1
            PORTE = 0x00;          //.5
            PORTF = 0x00;          //.2
            PORTG = 0x05;          //2,.2
            PORTH = 0x00;          //.1
            PORTJ = 0x00;          //.05
            display(0x0E,"1A");
            break;
        case 2:
            PORTD = 0x00;          //1
            PORTE = 0x00;          //.5
            PORTF = 0x00;          //.2
            PORTG = 0x0F;          //2,.2
            PORTH = 0x00;          //.1
            PORTJ = 0x00;          //.05
            display(0x0E,"2A");
            break;
        case 3:
            PORTD = 0x01;          //1
            PORTE = 0x00;          //.5
            PORTF = 0x00;          //.2
            PORTG = 0x0F;          //2,.2
            PORTH = 0x00;          //.1
            PORTJ = 0x00;          //.05
            display(0x0E,"3A");
            break;
        case 4:
```

```
        PORTD = 0x11;          //1
        PORTE = 0x00;          //.5
        PORTF = 0x00;          //.2
        PORTG = 0x0F;          //2,.2
        PORTH = 0x00;          //.1
        PORTJ = 0x00;          //.05
        display(0x0E,"4A");
        break;
    case 5:
        PORTD = 0x13;          //1
        PORTE = 0x00;          //.5
        PORTF = 0x00;          //.2
        PORTG = 0x0F;          //2,.2
        PORTH = 0x00;          //.1
        PORTJ = 0x00;          //.05
        display(0x0E,"5A");
        break;
    case 6:
        PORTD = 0x33;          //1
        PORTE = 0x00;          //.5
        PORTF = 0x00;          //.2
        PORTG = 0x0F;          //2,.2
        PORTH = 0x00;          //.1
        PORTJ = 0x00;          //.05
        display(0x0E,"6A");
        break;
    case 7:
        PORTD = 0x37;          //1
        PORTE = 0x00;          //.5
        PORTF = 0x00;          //.2
        PORTG = 0x0F;          //2,.2
        PORTH = 0x00;          //.1
        PORTJ = 0x00;          //.05
        display(0x0E,"7A");
        break;
    case 8:
        PORTD = 0x7F;          //1
        PORTE = 0x00;          //.5
        PORTF = 0x00;          //.2
        PORTG = 0x07;          //2,.2
        PORTH = 0x00;          //.1
        PORTJ = 0x00;          //.05
        display(0x0E,"8A");
        break;
    case 9:
```

```
        PORTD = 0xFF;           //1
        PORTE = 0x00;           //.5
        PORTF = 0x00;           //.2
        PORTG = 0x07;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"9A");
        break;
    case 10:
        PORTD = 0xFF;           //1
        PORTE = 0x01;           //.5
        PORTF = 0x00;           //.2
        PORTG = 0x01;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"10A");
        break;
    case 11:
        PORTD = 0x77;           //1
        PORTE = 0x11;           //.5
        PORTF = 0x00;           //.2
        PORTG = 0x07;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"11A");
        break;
    case 12:
        PORTD = 0x7F;           //1
        PORTE = 0x11;           //.5
        PORTF = 0x00;           //.2
        PORTG = 0x0F;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"12A");
        break;
    case 13:
        PORTD = 0xFF;           //1
        PORTE = 0x11;           //.5
        PORTF = 0x00;           //.2
        PORTG = 0x0F;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"13A");
        break;
    case 14:
```

```
        PORTD = 0xF7;           //1
        PORTE = 0x13;           //.5
        PORTF = 0x00;           //.2
        PORTG = 0x0F;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"14A");
        break;
    case 15:
        PORTD = 0x7F;           //1
        PORTE = 0x33;           //.5
        PORTF = 0x00;           //.2
        PORTG = 0x07;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"15A");
        break;
    case 16:
        PORTD = 0xFF;           //1
        PORTE = 0x33;           //.5
        PORTF = 0x00;           //.2
        PORTG = 0x07;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"16A");
        break;
    case 17:
        PORTD = 0x7F;           //1
        PORTE = 0x37;           //.5
        PORTF = 0x00;           //.2
        PORTG = 0x07;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"17A");
        break;
    case 18:
        PORTD = 0xFF;           //1
        PORTE = 0x37;           //.5
        PORTF = 0x00;           //.2
        PORTG = 0x07;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"18A");
        break;
    case 19:
```

```
        PORTD = 0xFF;           //1
        PORTE = 0x77;           //.5
        PORTF = 0x00;           //.2
        PORTG = 0x03;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"19A");
        break;
    case 20:
        PORTD = 0xF7;           //1
        PORTE = 0x7F;           //.5
        PORTF = 0x00;           //.2
        PORTG = 0x03;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"20A");
        break;
    case 21:
        PORTD = 0xFF;           //1
        PORTE = 0x7F;           //.5
        PORTF = 0x00;           //.2
        PORTG = 0x07;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"21A");
        break;
    case 22:
        PORTD = 0x7F;           //1
        PORTE = 0xFF;           //.5
        PORTF = 0x00;           //.2
        PORTG = 0x07;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"22A");
        break;
    case 23:
        PORTD = 0xF7;           //1
        PORTE = 0x77;           //.5
        PORTF = 0x10;           //.2
        PORTG = 0x0F;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"23A");
        break;
    case 24:
```

```
        PORTD = 0xF7;           //1
        PORTE = 0x7F;           //.5
        PORTF = 0x10;           //.2
        PORTG = 0x03;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"24A");
        break;
    case 25:
        PORTD = 0x77;           //1
        PORTE = 0xFF;           //.5
        PORTF = 0x10;           //.2
        PORTG = 0x07;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"25A");
        break;
    case 26:
        PORTD = 0x7F;           //1
        PORTE = 0xFF;           //.5
        PORTF = 0x10;           //.2
        PORTG = 0x07;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"26A");
        break;
    case 27:
        PORTD = 0xFF;           //1
        PORTE = 0xFF;           //.5
        PORTF = 0x10;           //.2
        PORTG = 0x0F;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"27A");
        break;
    case 28:
        PORTD = 0xFF;           //1
        PORTE = 0x77;           //.5
        PORTF = 0x12;           //.2
        PORTG = 0x0F;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"28A");
        break;
    case 29:
```

```
        PORTD = 0xFF;           //1
        PORTE = 0x7F;           //.5
        PORTF = 0x12;           //.2
        PORTG = 0x07;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"29A");
        break;
    case 30:
        PORTD = 0x7F;           //1
        PORTE = 0xFF;           //.5
        PORTF = 0x10;           //.2
        PORTG = 0x1F;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"30A");
        break;
    case 31:
        PORTD = 0xFF;           //1
        PORTE = 0xFF;           //.5
        PORTF = 0x10;           //.2
        PORTG = 0x1F;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"31A");
        break;
    case 32:
        PORTD = 0x7F;           //1
        PORTE = 0x7F;           //.5
        PORTF = 0x12;           //.2
        PORTG = 0x17;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"32A");
        break;
    case 33:
        PORTD = 0xFF;           //1
        PORTE = 0x7F;           //.5
        PORTF = 0x12;           //.2
        PORTG = 0x1F;           //2,.2
        PORTH = 0x00;           //.1
        PORTJ = 0x00;           //.05
        display(0x0E,"33A");
        break;
```

```
case 34:
    PORTD = 0xFF;          //1
    PORTE = 0xFF;          //.5
    PORTF = 0x12;          //.2
    PORTG = 0x17;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"34A");
    break;
case 35:
    PORTD = 0xFF;          //1
    PORTE = 0x77;          //.5
    PORTF = 0x32;          //.2
    PORTG = 0x17;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"35A");
    break;
case 36:
    PORTD = 0xF7;          //1
    PORTE = 0x7F;          //.5
    PORTF = 0x32;          //.2
    PORTG = 0x1F;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"36A");
    break;
case 37:
    PORTD = 0xF7;          //1
    PORTE = 0xFF;          //.5
    PORTF = 0x32;          //.2
    PORTG = 0x17;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"37A");
    break;
case 38:
    PORTD = 0xFF;          //1
    PORTE = 0x77;          //.5
    PORTF = 0x36;          //.2
    PORTG = 0x13;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"38A");
    break;
```

```
case 39:
    PORTD = 0x7F;          //1
    PORTE = 0xF7;          //.5
    PORTF = 0x36;          //.2
    PORTG = 0x17;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"39A");
    break;
case 40:
    PORTD = 0xFF;          //1
    PORTE = 0xF7;          //.5
    PORTF = 0x36;          //.2
    PORTG = 0x1F;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"40A");
    break;
case 41:
    PORTD = 0xFF;          //1
    PORTE = 0xFF;          //.5
    PORTF = 0x36;          //.2
    PORTG = 0x17;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"41A");
    break;
case 42:
    PORTD = 0xFF;          //1
    PORTE = 0x77;          //.5
    PORTF = 0x76;          //.2
    PORTG = 0x1F;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"42A");
    break;
case 43:
    PORTD = 0xFF;          //1
    PORTE = 0x7F;          //.5
    PORTF = 0x76;          //.2
    PORTG = 0x17;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"43A");
    break;
```

```
case 44:
    PORTD = 0xF7;          //1
    PORTE = 0xFF;          //.5
    PORTF = 0x76;          //.2
    PORTG = 0x1F;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"44A");
    break;
case 45:
    PORTD = 0xFF;          //1
    PORTE = 0x77;          //.5
    PORTF = 0x7E;          //.2
    PORTG = 0x17;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"45A");
    break;
case 46:
    PORTD = 0x7F;          //1
    PORTE = 0xF7;          //.5
    PORTF = 0x7E;          //.2
    PORTG = 0x1F;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"46A");
    break;
case 47:
    PORTD = 0xF7;          //1
    PORTE = 0xFF;          //.5
    PORTF = 0x7E;          //.2
    PORTG = 0x17;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"47A");
    break;
case 48:
    PORTD = 0xF7;          //1
    PORTE = 0x77;          //.5
    PORTF = 0xFE;          //.2
    PORTG = 0x1F;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"48A");
    break;
```

```
case 49:
    PORTD = 0xF7;          //1
    PORTE = 0xF7;          //.5
    PORTF = 0xFE;          //.2
    PORTG = 0x1F;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"49A");
    break;
case 50:
    PORTD = 0xF7;          //1
    PORTE = 0xFF;          //.5
    PORTF = 0xFE;          //.2
    PORTG = 0x17;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"50A");
    break;
case 51:
    PORTD = 0xFF;          //1
    PORTE = 0xFF;          //.5
    PORTF = 0xFE;          //.2
    PORTG = 0x1F;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"51A");
    break;
case 52:
    PORTD = 0xF7;          //1
    PORTE = 0x7F;          //.5
    PORTF = 0x7E;          //.2
    PORTG = 0x17;          //2,.2
    PORTH = 0x01;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"52A");
    break;
case 53:
    PORTD = 0xF7;          //1
    PORTE = 0xFF;          //.5
    PORTF = 0x7E;          //.2
    PORTG = 0x17;          //2,.2
    PORTH = 0x01;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"53A");
    break;
```

```
case 54:
    PORTD = 0x77;          //1
    PORTE = 0x7F;          //.5
    PORTF = 0xFE;          //.2
    PORTG = 0x13;          //2,.2
    PORTH = 0x01;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"54A");
    break;
case 55:
    PORTD = 0xFF;          //1
    PORTE = 0x7F;          //.5
    PORTF = 0xFE;          //.2
    PORTG = 0x13;          //2,.2
    PORTH = 0x01;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"55A");
    break;
case 56:
    PORTD = 0xFF;          //1
    PORTE = 0xFF;          //.5
    PORTF = 0xFE;          //.2
    PORTG = 0x11;          //2,.2
    PORTH = 0x01;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"56A");
    break;
case 57:
    PORTD = 0xFF;          //1
    PORTE = 0xFF;          //.5
    PORTF = 0xFE;          //.2
    PORTG = 0x1F;          //2,.2
    PORTH = 0x01;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"57A");
    break;
case 58:
    PORTD = 0xFF;          //1
    PORTE = 0x7F;          //.5
    PORTF = 0x7E;          //.2
    PORTG = 0x13;          //2,.2
    PORTH = 0x11;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"58A");
    break;
```

```
case 59:
    PORTD = 0xFF;          //1
    PORTE = 0xFF;          //.5
    PORTF = 0x7E;          //.2
    PORTG = 0x17;          //2,.2
    PORTH = 0x11;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"59A");
    break;
case 60:
    PORTD = 0xFF;          //1
    PORTE = 0x7F;          //.5
    PORTF = 0xFE;          //.2
    PORTG = 0x10;          //2,.2
    PORTH = 0x11;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"60A");
    break;
case 61:
    PORTD = 0xFF;          //1
    PORTE = 0xFF;          //.5
    PORTF = 0xFE;          //.2
    PORTG = 0x10;          //2,.2
    PORTH = 0x11;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"61A");
    break;
case 62:
    PORTD = 0xFF;          //1
    PORTE = 0xFF;          //.5
    PORTF = 0xFE;          //.2
    PORTG = 0x1F;          //2,.2
    PORTH = 0x11;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,"62A");
    break;

default:
    PORTD = 0x00;          //1
    PORTE = 0x00;          //.5
    PORTF = 0x00;          //.2
    PORTG = 0x01;          //2,.2
    PORTH = 0x00;          //.1
    PORTJ = 0x00;          //.05
    display(0x0E,".5A");
```

```
            break;
    }
}

//Main program-----------------------------------------------------------------
void main(void)
{
    if(rx_yes == 1)
        {
            if (usart_in != 0x67)//if keyboard is pressed but not 'g'
            {
                if (usart_in == 13)  //if enter (newline) is pressed
                {

                    total_output = dec_in_hundreds + dec_in_tens + dec_in_one
                    total_output2 = total_output >> 8;//shifts the total outp
                    Write1USART(total_output2 && 0x0F); //tansmit the first t
                    Write1USART(total_output && 0xFF);  //transmit the last n
                    rx_yes = 0;       //reset button pressed
                    place = 0;        //set the place in the input number to h
                }
                else
                {
                    place++; //increment digits place
                    dec_in(usart_in); //function that stores digits
                    while(Busy1USART());
                    putc1USART(usart_in);//send back key pressed
                    rx_yes = 0;       //set place in input number to hundreds
                }
            }
            if (usart_in == 0x67)     //if 'g' pressed
            {
                rx_yes = 0;           //reset button pressed
                clear_display();
                display(0x00,"Desired: 1V");
                output_active(total_output);
//output desired output from before
                runADC();
                current3 = current;
                sprintf(buffer2,"%f",current3);
                sprintf(buffer,"%.3f",voltage);
                display(0x40,"Actual:");
                display(0x47,buffer);
                display(0x4C,"V");
                display(0x4E,buffer2);
```

```
                display(0x53,"A");

                pos_20_count = 0;
                pos_10_count = 0;
                pos_5_count = 0;
                pos_2_count = 0;
                pos_1_count = 0;
                pos_05_count = 0;
                desired = total_output;

            }
            if (usart_in == 0x78)     //if 'x' is pressed
            {
                rx_yes = 0;           //set buttons pressed to 0
                int out = 0;              //set output to 0
                while (rx_yes == 0)  //
                {
                    output_active(out);      //output nothing until another bu
                }
            }
            if (usart_in == 0x3F)    //if ? is sent
            {
                putrs1USART("LBC");
            }
        }
    else
    {
    runADC;
    difference = desired - current;

    if (abs(difference) > 0.5)
    {
        difference = total_output - current;
        total_output = total_output + difference;


/*This is to add or subtract a step at a time (active plan 2)-----------------
        if(difference >= 20)
        {
            total_output = total_output + 20;
        }

        if (difference >= 10 && difference < 20)
        {
            total_output = total_output + 10;
```

```
}

if (difference >= 5 && difference < 10)
{
    total_output = total_output + 5;
}

if (difference >= 2 && difference < 5)
{
    total_output = total_output + 2;
}

if (difference >= 1 && difference < 2)
{
    total_output++;
}

if (difference >= 0.5 && difference < 1)
{
    total_output = total_output + 0.5;
}

if(difference <= -20)
{
    total_output = total_output - 20;
}

if (difference <= -10 && difference > -20)
{
    total_output = total_output - 10;
}

if (difference <= -5 && difference > -10)
{
    total_output = total_output - 5;
}

if (difference <= -2 && difference > -5)
{
    total_output = total_output - 2;
}

if (difference <= -1 && difference > -2)
{
    total_output--;
```

```
        }

        if (difference <= -0.5 && difference > -1)
        {
            total_output = total_output - 0.5;
        }
*/

/* This is for addressing individual resistors (active plan 3)----------------
        if(difference > 20 && pos_20_count != 8)
        {
            difference = difference - 20;
            pos_20_count++;
            total_output = total_output + 20;
        }

        if (difference > 10 && difference < 20)
        {
            total_output = total_output + 10;
            difference = difference - 10;
        }

        if (difference > 10 && pos_20_count == 8 && pos_10_count != 8)
        {
            total_output = total_output + 10;
            difference = difference - 10;
            pos_10_count++;
        }

        if (difference > 5 && difference < 10)
        {
            total_output = total_output + 5;
            difference = difference - 5;
        }

        if (difference > 5 && pos_10_count == 8 && pos_20_count == 8 && pos_5
        {
            total_output = total_output + 5;
            difference = difference - 5;
            pos_5_count++;
        }

        if (difference > 2 && difference < 5)
        {
            total_output = total_output + 2;
```

```
            difference = difference - 2;
        }

        if (difference > 2 && pos_20_count == 8 && pos_10_count == 8 & pos_5_
        {
            total_output = total_output + 2;
            difference = difference - 2;
            pos_2_count++;
        }

        if (difference > 1 && difference < 2)
        {
            total_output++;
            difference--;
        }

        if (difference > 1 && pos_20_count == 8 && pos_10_count == 8 && pos_5
        {
            total_output++;
            difference--;
            pos_1_count++;
        }

        if (difference >= 0.5 && difference < 1)
        {
            total_output = total_output + 0.5;
            difference = difference - 0.5;
        }

        if (difference >= 0.5 && pos_20_count == 8 && pos_10_count == 8 && po
        {
            total_output = total_output + 0.5;
            difference = difference - 0.5;
            pos_05_count++;
        }
 */
        output_active(total_output);
        }
    }
}

void passive_mode(void)
{
    if(rx_yes == 1)
        {
```

```
            if (usart_in != 0x67)//if keyboard is pressed but not 'g'
            {
                if (usart_in == 13)  //if enter (newline) is pressed
                {

                    total_output = dec_in_hundreds + dec_in_tens + dec_in_one
                    total_output2 = total_output >> 8;//shifts the total outp
                    Write1USART(total_output2 && 0x0F); //tansmit the first t
                    Write1USART(total_output && 0xFF);  //transmit the last n
                    rx_yes = 0;       //reset button pressed
                    place = 0;        //set the place in the input number to h
                }
                else
                {
                    place++; //increment digits place
                    dec_in(usart_in); //function that stores digits
                    while(Busy1USART());
                    putc1USART(usart_in);//send back key pressed
                    rx_yes = 0;       //set place in input number to hundreds
                }
            }
            if (usart_in == 0x67)     //if 'g' pressed
            {
                rx_yes = 0;           //reset button pressed
                clear_display();
                display(0x00,"Desired: 1V");
                output_passive(total_output);
//output desired output from before
                runADC();
                current3 = current;
                sprintf(buffer2,"%f",current3);
                sprintf(buffer,"%.3f",voltage);
                display(0x40,"Actual:");
                display(0x47,buffer);
                display(0x4C,"V");
                display(0x4E,buffer2);
                display(0x53,"A");


            }
            if (usart_in == 0x78)     //if 'x' is pressed
            {
                rx_yes = 0;           //set buttons pressed to 0
                int out = 0;              //set output to 0
                while (rx_yes == 0)  //
```

```
                {
                    output_passive(out);      //output nothing until another b
                }
            }
            if (usart_in == 0x3F)    //if ? is sent
            {
                putrs1USART("LBC");
            }
        }

        else //Updates the display with current values for V and A
        {
            current2 = current;
            runADC();
            if (abs(current2-current)>0.5)
            {
                sprintf(buffer2,"%f",current);
                sprintf(buffer,"%.3f",voltage);
                display(0x40,"Actual:");
                display(0x47,buffer);
                display(0x4C,"V");
                display(0x4E,buffer2);
                display(0x53,"A");
            }
        }
}

//Main program-----------------------------------------------------------------
void main(void)
{
    __delay_ms(25);

    current = 10;

    INTCON=1;                       //
    OSCCON = 0b01101111;
    GIE = 1;
    PEIE = 1;
    RC1IE = 1;

    TRISA = 0b11111111;
    TRISB = 0b11111111;
    TRISC = 0b10100111;
    TRISD = 0b00000000;
    TRISE = 0b00000000;
```

```
    TRISF = 0b00000000;
    TRISG = 0b00000000;
    TRISH = 0b00000000;
    TRISJ = 0b00000000;

    BAUDCON1bits.BRG16 = 1;                 // 1 = 16-bit Baud Rate Generator ? S
    SPBRGH1 = 0b0000;                       // BRGH is 4 bits only
    SPBRG1  = 0b11010000;                   // 9600 Bauds = 16000000 / (4 (416 +
(n=416 -> 0001 10100000)
    TXSTA1 = 0b00100100;                    // ( 8 bits - Asynchrone )
    RCSTA1 = 0b10010000;

    PORTD = 0x00;
    PORTE = 0x00;
    PORTF = 0x00;
    PORTG = 0x00;
    PORTH = 0x00;
    PORTJ = 0x00;

    init_lcd();
    init_ADC();
    __delay_ms(25);
    init_lcd();
    clear_display();
    display(0x00,"Press 1 for active");
    display(0x40,"Press 2 for passive");

    while (button_pressed == 0)
    {
        if (PORTCbits.RC0 == 0)
        {
            mode = 1;
            button_pressed = 1;
        }
        if (PORTCbits.RC1 == 0)
        {
            mode = 2;
            button_pressed = 1;
        }
    }


    if (mode == 1)
    {
        clear_display();
```

```
            display(0x00,"A Desired: 1V    0A");

            for(;;)
            {
                active_mode();
            }
        }

        if (mode == 2)
        {
            clear_display();
            display(0x00,"P Desired: 1V    0A");

            for(;;)
            {
                passive_mode();
            }
        }

        return;
}
```