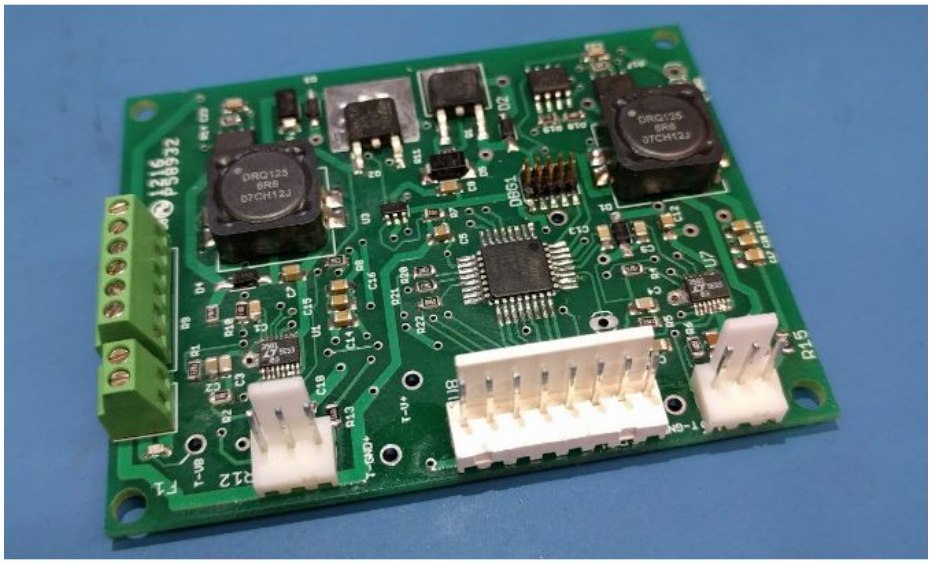


FoE Adjustable Power Supply

Liam Mulshine, Scott Sun, Sandra Schumann

ES91r: The Frustration of Electronics
Harvard University
Spring 2016

Table of Contents

1 Introduction

In ES52: The Joy of Electronics, at the end of every semester, students present their final projects at a design fair. Many of those projects require power levels that you cannot get simply from a standard 9V battery (i.e. single 5V supply, dual +-5V supply). The solution so far has been to carry around a benchtop power supply from the lab. However, said power supply is quite large, heavy, and, in general, not the optimal solution for something as simple as supplying power to small breadboard circuits.

Our project designs an adjustable breadboard power supply: a small box that can be plugged into the breadboard and used to satisfy all ES52 power supply needs. The final result should be able to supply enough power for every project likely to be built as part of ES52. The exact specifications initially defined were as follows:

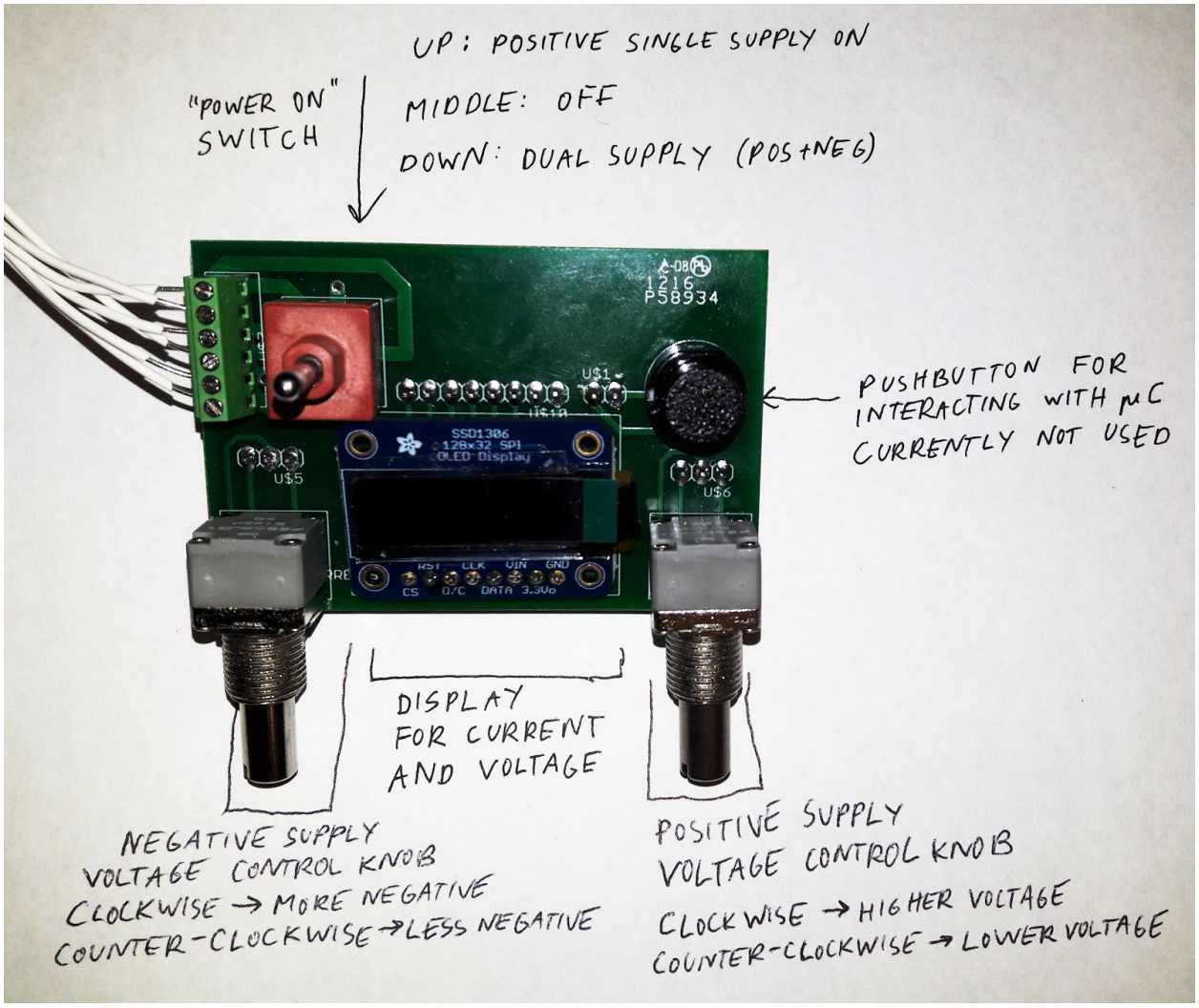
- Output voltage ranges from 1.5V to 10V on the positive side, and from -1.5V to -10V on the negative side
- Maximum output current: 1A
- Maximum noise at output: 5%
- Linearly adjustable voltage
- Fits the size of a breadboard
- Runs off of 4.8V (4 NImH batteries in series)
- Real time current and voltage display

This document describes observations made during the production of the first prototype. The first section of the document contains instructions for operating the power supply. The second section is dedicated to the design process: component selection, schematic and layout design. The third section describes the soldering, testing and debugging of the first prototype. The fourth section discusses how well the first prototype meets the predefined specifications. The fifth section describes sources of error in the first prototype and the sixth section necessary and potential changes for the second prototype, including changes to the specifications.

The authors of breadboard power supply, Liam Mulshine, Scott Sun and Sandra Schumann would like to thank our instructor, Avinash Uttamchandani for advice and support, professor David Abrams and several other professors for their recommendations for improving our design and all of our “Frustration of Electronics” classmates for their help during the design reviews and their moral support later on. We would also like to thank Jim MacArthur, and Al Takeda, whose assistance in lab made assembling our board a good deal less (although not completely) painless.

2 Power Supply Operation

The following image describes the power supply's user interface:



For operating the power supply, it must first be turned on. This can be done using the "power on" switch in the upper left hand corner of the user interface. The switch has three positions: up, middle and down. The "up" position turns on only the positive supply. The "down" position turns on both positive and negative supply. The "middle" position turns the entire power supply off (this includes the microcontroller and the display).

Adjusting the voltage can be done using two control knobs. Positive supply is controlled by the knob on the right and negative supply by the one on the left. The voltage difference from ground gets greater as the knob is turned clockwise for both positive and negative supply, which for us seemed to be the intuitive direction, leading to hopefully very straightforward operation.

The OLED display in the middle is meant for providing feedback about the voltage and current outputs of the power supply. However, in our first prototype this functionality was still incomplete (see section “Microcontroller Code” below). The user interface also includes a pushbutton intended for interacting with the microcontroller, for example, in order to switch between positive and negative voltage display on the OLED display; however, due to the display functionality being still incomplete it currently does nothing.

3 Design Process

The process by which we went to design our adjustable breadboard power supply can be broken up into 5 main stages: defining the *Problem and Project Specifications*, creating a *Block Diagram*, *Part Selection, Schematic Design,* and implementing the *PCB Layout.* In this section, we will summarize each stage of the design process. However, please note that these stages had overlap, and that moving from one stage to the next did not prevent us from jumping back a stage to redefine our specs, alter our schematic, choose a new part, etc.

3.1 Problem and Project Specifications

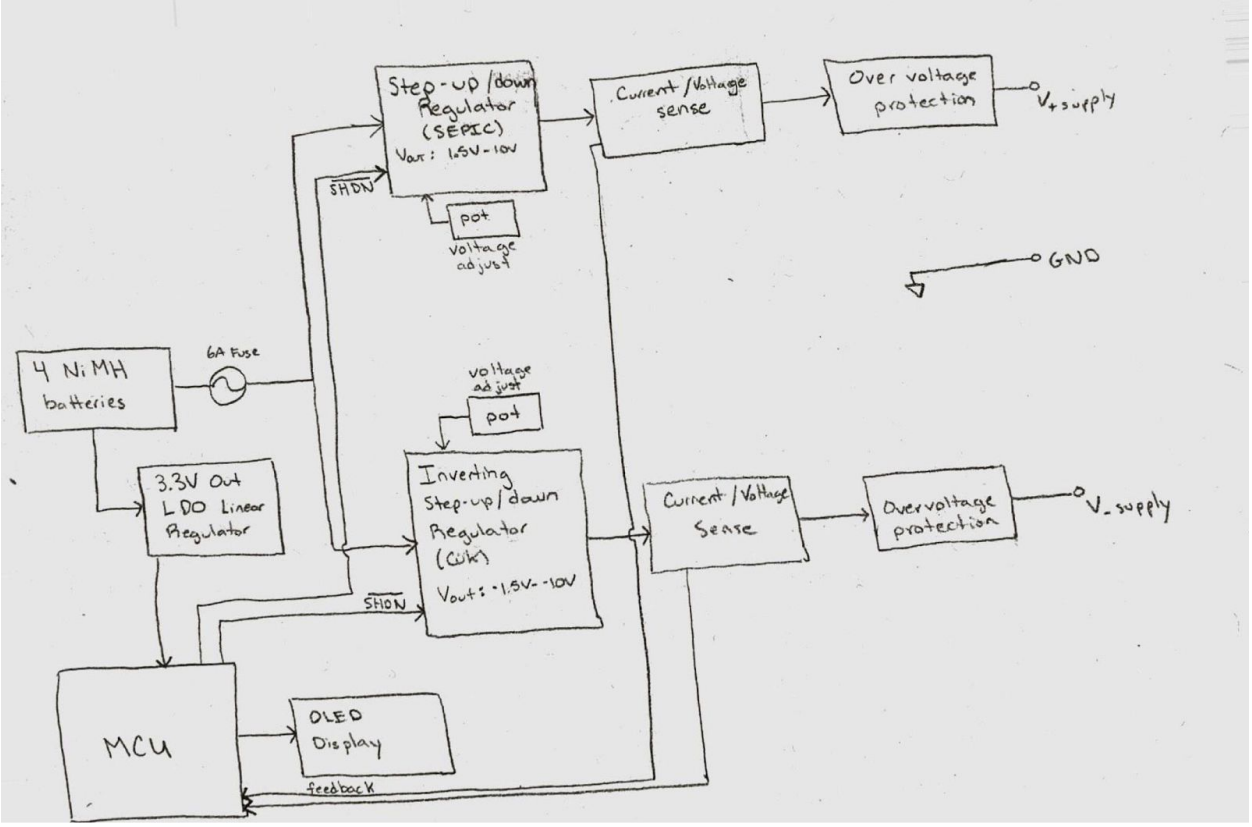
The first step of our design process involved defining our problem and creating a set of specifications upon which we would base our design decisions. The bulk of this process was outlined above in the introduction to this project report. Our problem was largely defined for us (to create a breadboard power supply for use by first year undergraduate engineering students in Harvard’s class, ES52), and some specifications were suggested. Yet we had the freedom of defining specs like continuous or discrete voltage adjustment, what mechanism we would employ to provide user feedback, maximum current ratings, voltage range, etc. In retrospect, we were a bit bold when defining our initial specifications, considering that this was the first PCB design project for the majority of our group. Yet doing so challenged us to stretch our boundaries, and made achieving the end product truly satisfying.

3.2 Block Diagram

The complete block diagram can be seen on the next page.

In order to create the voltage ranges that we desired, which is from -10 to -1.5V and 1.5V to 10V, we opted to use two separate buck-boost topologies--the SEPIC (for positive voltages) and the Cuk (for negative voltages) converters. Because we wanted linear continuous control of voltages via potentiometers by the user, we needed to be able to display the current and voltage to the user. This necessitated the use of a microcontroller to interface with current and voltage sensing in order to display these figures to the user through an OLED display. Since our switching regulators would not always be set to the necessary 3.3V for the microcontroller power supply, we needed a separate 3.3.V linear regulator to power the microcontroller. To guard against user stupidity, we wanted to add overvoltage and short protection at the outputs.

Through current and voltage sensing, the microcontroller can shut down the switching regulators as necessary, which turns off all current flow through the power supply. In order to provide ease of maintenance, we wanted to power this off of 4 easily-obtainable NiMh rechargeable batteries, which are limited by a 6A fuse.



3.3 Part Selection

As the switching regulator was the most important component in our variable power supply, a lot of time was spent deciding which switching regulator matched our specifications. We ended up settling on the LT3581 because it was capable of being arranged in SEPIC and Cuk topologies to cover both positive and negative voltage ranges from a single supply. The SEPIC configuration allows us to buck or boost our battery voltage to a range from 1.5V to 10V. The Cuk inverts our battery voltage to provide a range from -1.5V to -10V. Furthermore, the exceptional datasheet was very descriptive and user-friendly, providing step-by-step instructions for calculating the various inductor and capacitor values. This greatly simplified the design process. Furthermore, this chip was able to handle the 1A current limit that we had originally set on our power supply throughout the majority of our voltage range.

Because we also needed to provide a 3.3V reference to the microcontroller, we decided to use the AS1363 linear regulator connected directly to the battery pack as we needed a stable 3.3V supply regardless of what our switching regulators were providing. We felt the simplest way to

accomplish that was with a simple linear regulator to drop the 4.5V nominal voltage of the battery to 3.3V, which wouldn't dissipate a lot of power given the low current requirements of our microcontroller, which we chose to be a SAMD-21E.

We chose to use the SAMD-21E for several simple reasons. We did not need a very advanced microcontroller to perform the operations with which we were concerned--mostly measuring voltage and current outputs via a reasonably accurate 12 bit ADC, shutting down the converters in the event of errors, and driving an OLED display via SPI. Thus, we could use a relatively simple microcontroller. Furthermore, this was essentially "free" as it was already provided, which reduced cost. Our specs made integrating a microcontroller quite essential as it would have been very difficult to display voltage information to the user in a way easier than using a microcontroller. Atmel Start and sample code for the OLED greatly simplified the coding process.

To power our device, we chose 4 AA NiMH rechargeable batteries that each have a nominal voltage of 1.2 V. This is because we wanted to provide a high enough battery voltage so that the 3.3V regulator could work and so that the switching regulators could provide more power at low voltages through bucking a higher voltage down. We also wanted the batteries to be easily replaceable, and since NiMH batteries can be found in stores, they were the perfect match. According to some discharge curves we found, they are able to reach at least 5 A of discharge current, though such high discharge rates have a negative effect on their battery life of 2500mAh. Regardless, as the purpose was for portable charging of ES52 breadboards, they would last in the range of half an hour to an hour in the worst case situation. In most cases, the current draw would not nearly be as high, so we deemed this an acceptable tradeoff. As protection for our batteries, we used a 6A fuse as we anticipated that each switching regulator would at most draw around 2-3A when operated simultaneously in conjunction with the microcontroller (which doesn't use that much power).

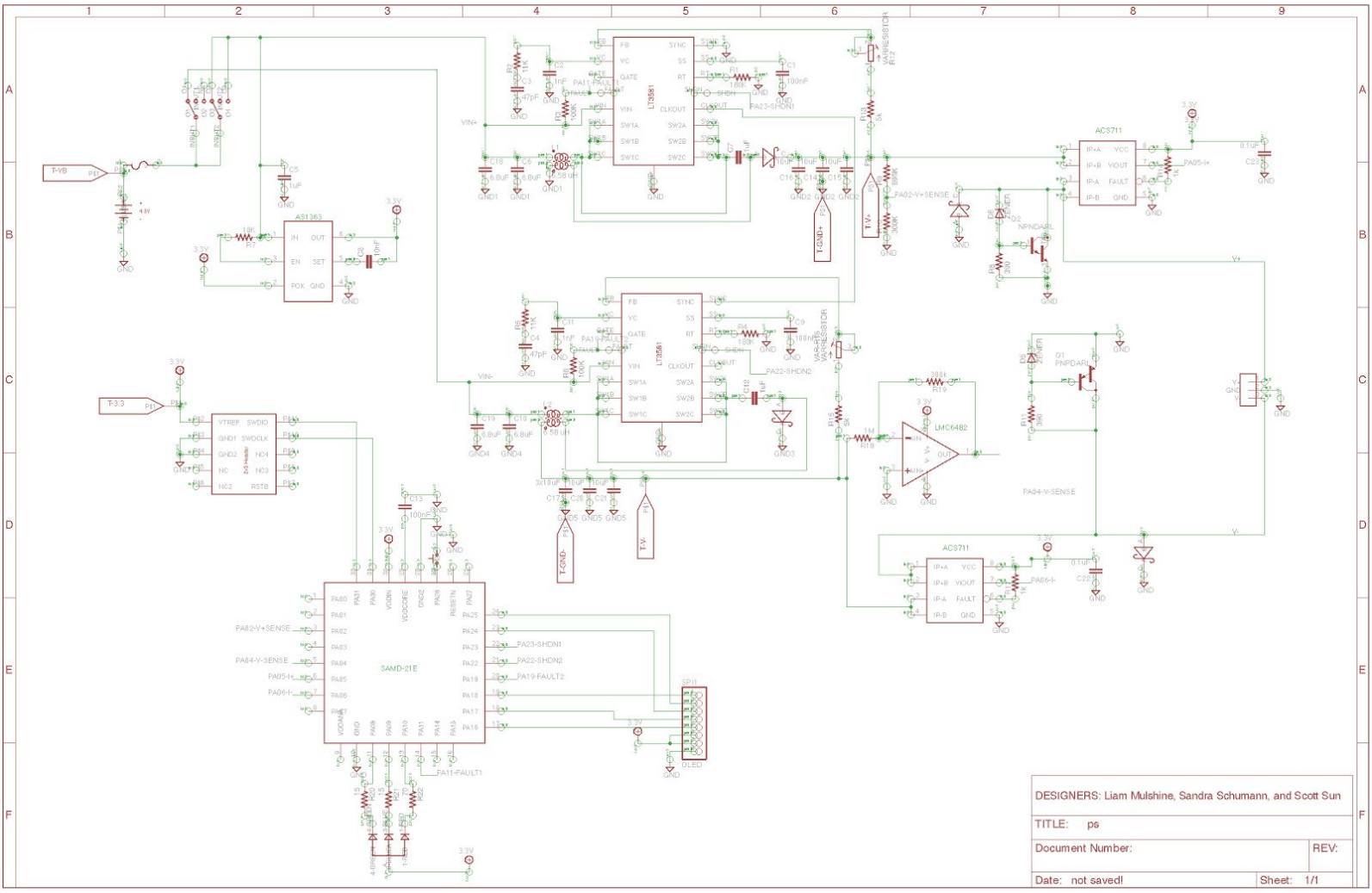
In order to provide current sensing, after considering an instrumentation amplifier to measure the voltage drop across a small resistance in series with the output, we opted to use the ACS711, a special current-sense chip. This decision was motivated by the fact that using the instrumentation amplifier and resistance would not have been as accurate, nor as simple to design, as purchasing a specially-made component. We wanted to eliminate this source of unnecessary complexity for a far superior component that had a smaller footprint.

To provide over-voltage protection at the outputs, we could have used a power zener that conducted at voltages greater than our maximum output voltage. However, we chose to use an active power zener, involving a zener and resistor connected to the gate of a Darlington transistor pair because a Zener diode that dissipates 12W is difficult to acquire and fit onto our pcb.

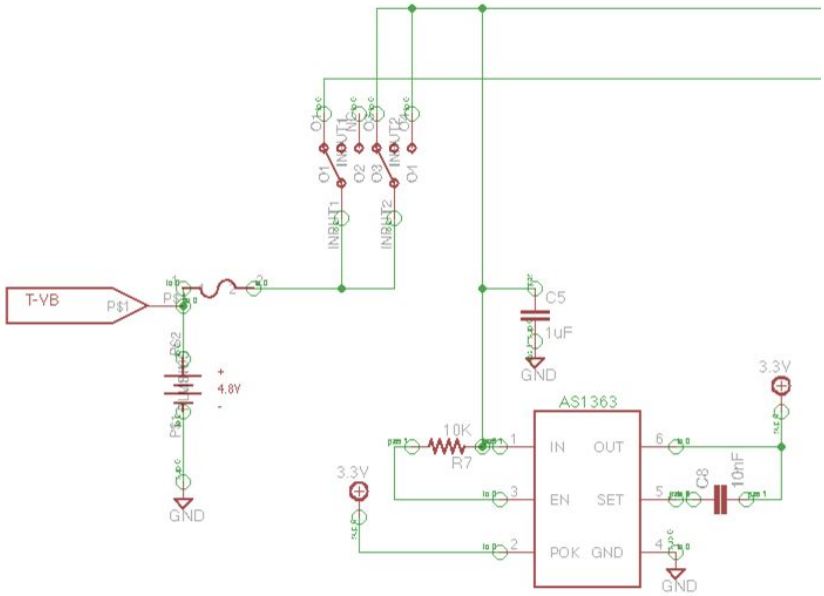
We decided to use an OLED display because our user voltage control potentiometers required that the current voltage be displayed to the user. While we could have used a 7 segment

display, we would have had to use more pins, and it would not have displayed as much information due to the far lower resolution. In addition, as these had already been purchased for us with breakout boards and an onboard voltage regulator, these were free and simple to incorporate. As the OLED uses SPI as its interface, it is relatively easy to control, especially using the sample code provided by Adafruit..

3.4 Schematic



Our schematic can be broken up to match with our block diagram to improve readability. In the picture below is our power supply's input voltage stage:

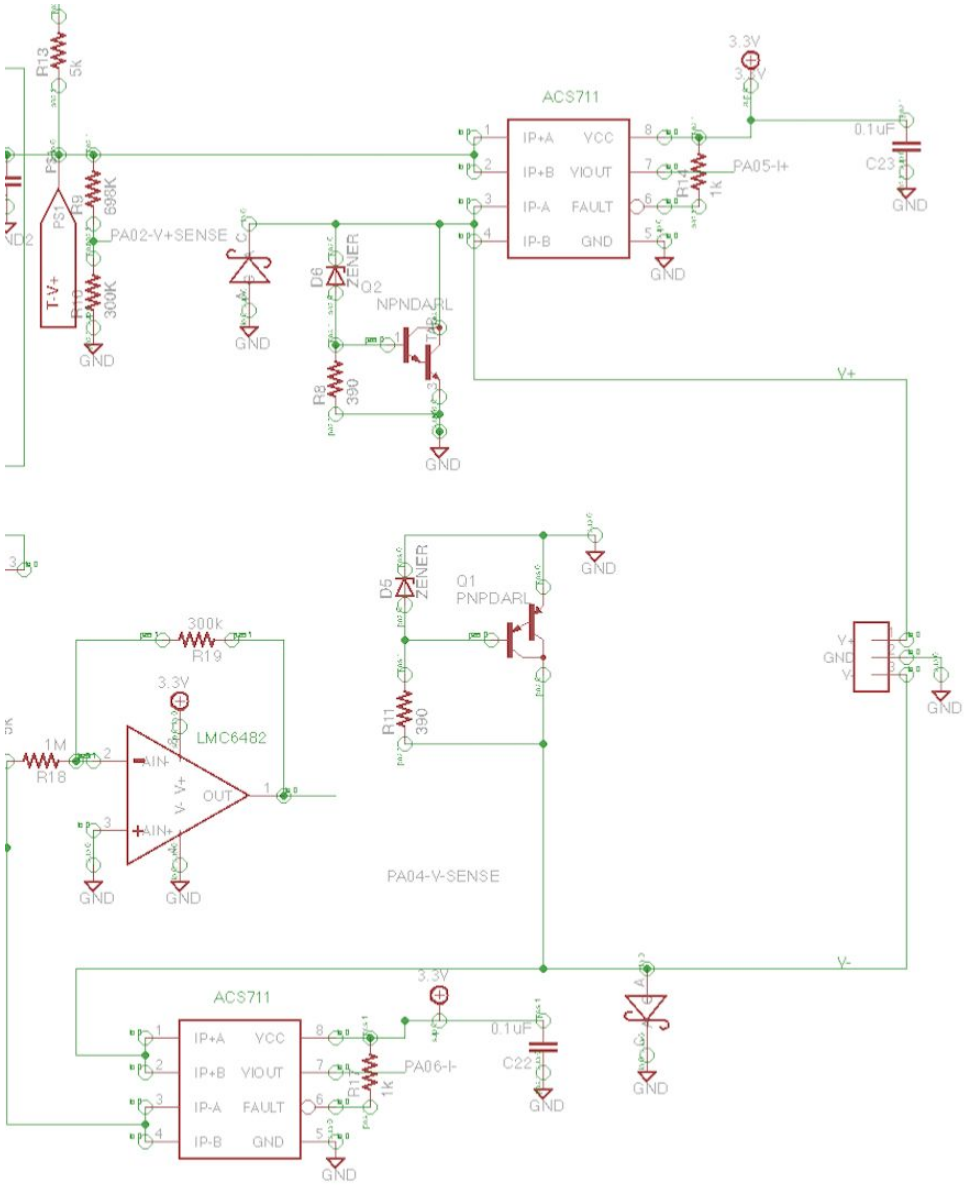


This stage consists of our 4 NiMH AA batteries with a nominal voltage of 4.8V. This is connected to a 6A fuse for overcurrent protection and a DPDT switch in on-off-on configuration. The first position sets both power supply rails on, the middle turns both off, and the last position is positive single-supply operation. Whenever the circuit is on, the AS1363 3.3V linear regulator is powered to provide 3.3V to the microcontroller.

The next stage is our switching regulator stage, with the top regulator being in SEPIC configuration and the bottom in Cuk:

All of the component values in these configurations were either provided by the datasheet, either as recommended values, or, in the case of the inductors, capacitors, clock timing resistor, and feedback resistor, as equations that needed to be solved to provide a range of acceptable values. We adjusted our values for a clock speed of 500kHz to give reasonably low values for our inductors and capacitors while preventing noise from being too high. This clock generated by the top regulator was fed into the sync pin of the second to ensure they switched synchronously. We chose feedback resistor values as 100K potentiometers to attain the voltage span of 10V per the spec.

The output of these two regulators is our current/voltage sense circuitry followed by our output protection circuitry:

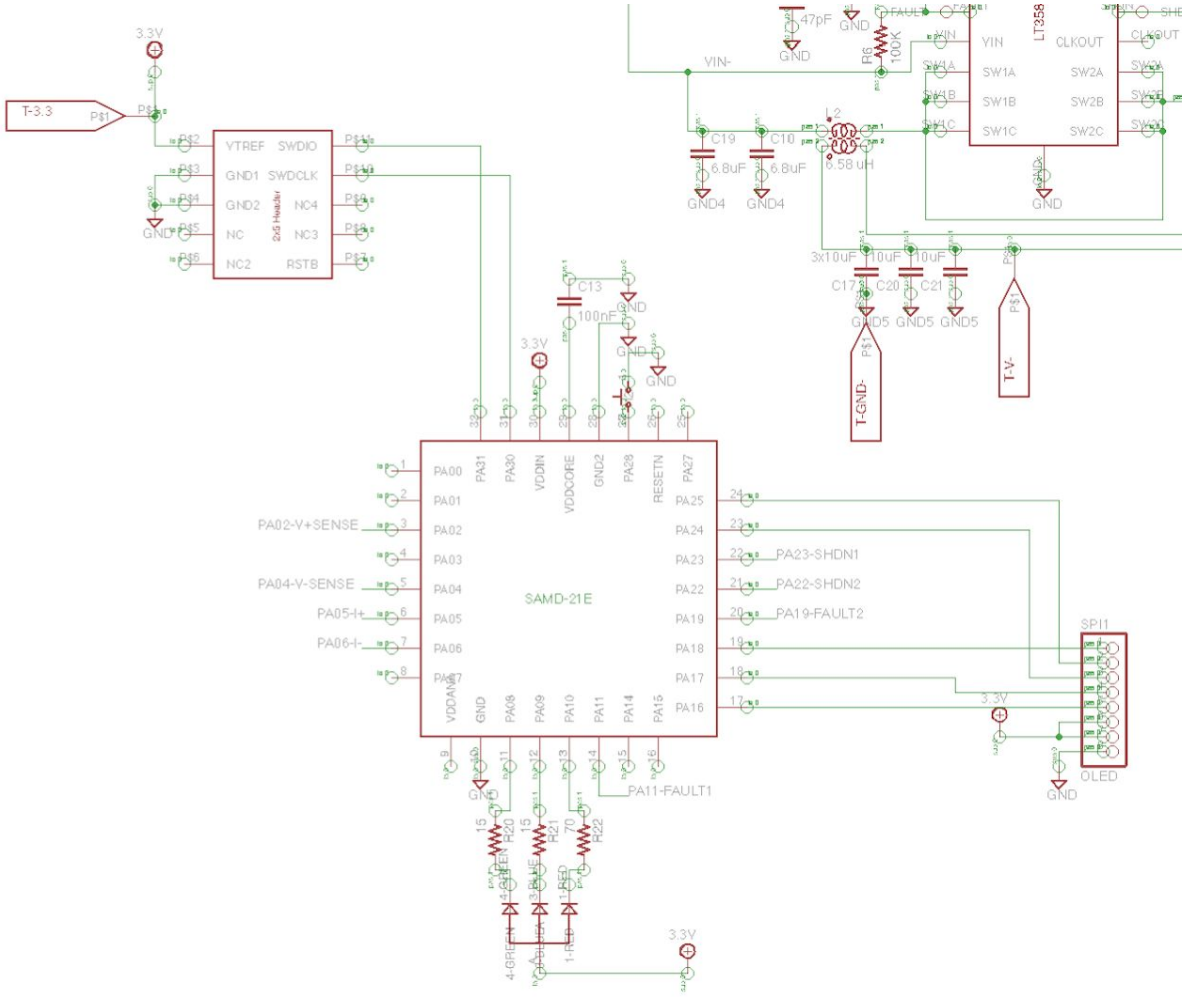


In order to sense output voltage, we constructed simple voltage divider with very large resistances to minimize loading and current flow. Their outputs are connected to PA02 and PA04 in the microcontroller where they will be processed by the internal ADC. For current sense, the ACS711 chips are able to provide a very accurate voltage to the microcontroller based on the current flowing through them.

For circuit protection, we included active power Zeners and Schottkys in order to avoid large voltages and currents destroying our switching regulators should someone apply a large voltage to the output or short the two output pins together. The LT3581 also already provides some degree of short protection as per the datasheet, but it also mentions that the safety of the chip is not guaranteed, which is why we added our own. The power Schottky diodes are meant to protect against connecting the two output pins together, because when a negative voltage is applied to the positive rail or vice-versa, the Schottky will conduct. These Schottkys cannot withstand a very large power (<1W), but we decided that this was acceptable as most situations where the user would touch the two output lines together would be accidental and occur in timescales of a second.

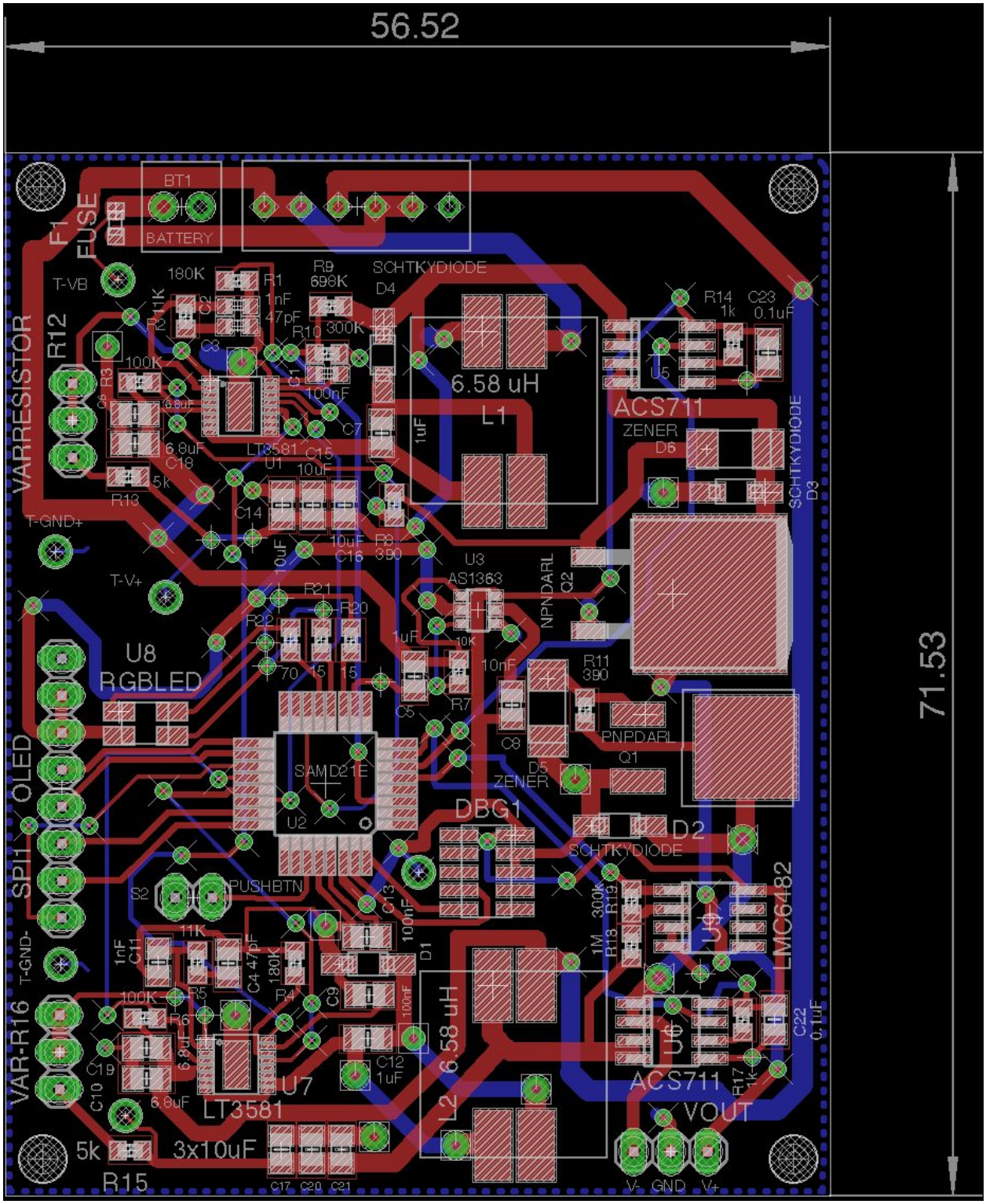
The active power zener is supposed to protect against a large magnitude overvoltage condition (where the output voltage has been raised above 12V on the positive or -12V on the negative supply), whereby our appropriately chosen 12V Zener diode would conduct and turn on the Darlington transistor pairs, which can handle 20W of power. Thus, the large voltage and current would primarily flow through the active power Zener. As a side-note, we noticed afterward pcb production that the positioning of our zener and resistor were reversed in the negative supply's active power Zener array. We immediately fixed this issue during pcb assembly.

Finally, the wiring of the microcontroller is included below:



In addition to the connections the microcontroller has with the LT3581 regulators and the ACS711 current sense chips, the microcontroller also has an RGB LED as a status indicator and a pushbutton for additional user control. The remaining connections are to the debugger and the SPI for the OLED display. Unfortunately, we did not wire up our reset pin and miswired our debug header, so we had to rearrange the debugger wire in order to upload code to the microcontroller.

3.5 Layout


56.52
71.53
BT1
BATTERY
F1
FUSE
T-VB
180K
R1
1nF
47pF
R9
698K
SCHTKYDIODE
D4
R14
1k
C23
0.1uF
VARRESISTOR
R12
R2
11K
C1
C8
R10
300K
R3
100K
6.8uF
100nF
C7
6.58 uH
L1
1uF
ACS711
U5
ZENER
D6
SCHTKYDIODE
D3
C18
6.8uF
LT3581
U1
C15
10uF
R13
5k
C14
10uF
C16
10uF
R8
390
T-GND+
T-V+
U3
AS1363
NPNDARL
Q2
R21
R20
R22
70
15
15
1uF
10K
10nF
R11
390
U8
RGBLED
C5
R7
C8
PNPDARL
Q1
D5
ZENER
SAMD21E
U2
DBG1
D2
SCHTKYDIODE
OLED
SPI1
T-GND-
S2
PUSHBTN
11K
C13
100nF
D1
300K
R19
1M
R18
LMC6482
U4
1nF
C11
C4 47pF
180K
R4
C9
100nF
100K
R5
R6
6.8uF
C19
C10
6.8uF
U7
LT3581
C12
1uF
6.58 uH
L2
ACS711
U6
R17
1k
C22
0.1uF
VAR-R16
5k
R15
3x10uF
C17
C20
C21
VOUT
V-
GND
V+

We conceived of our power supply as being a pcb packaged inside a small 3D printed box that would have all the peripherals for the user to control the power supply and connect it to a breadboard. We drilled holes on the four corners to have standoffs, which could attach the board to a case. This could then be placed on top of a breadboard as a nice, clean package. By using a case, we would also be able to adjust the positioning of things like header connections to the breadboard because their positioning could easily be adjusted if we had measured incorrectly, whereas our pcb would not be easily alterable upon fabrication. As such, we wanted to make a board that was the same width as a breadboard, which was roughly 5.5 cm. Unfortunately, as our circuit ended up being quite large, the length of our pcb slowly increased to cover over ⅓ of the length of a standard breadboard. A major design decision for us was to consider the tradeoffs between having a smaller pcb and being able to effectively and easily place components onto the board during assembly. We wanted adequate room for thicker traces due to our high current requirements as well as making the small components easy to solder.

Furthermore, as our intention was for this to be attached inside a box, we used 0.1” header connectors for most external connections to the main pcb, even fabricating a second pcb that could be attached to the case that contained all the buttons and switches as the main board would be inaccessible to the user. This change was also necessitated by the need for additional space on our main pcb, which would have been too large had we included all of the switches and buttons. For the heavy current connectors, we bought screw terminal connectors that were rated to 10A to ensure that they could handle the fuse burnout current of 6A. All power input connections were placed at the top, while the outputs are placed at the bottom right. Pushbutton, potentiometer, and OLED headers were placed to the left. This was done to maximize the amount of usable space in the center of the board for all the traces and components.

In the layout of our board, we tried to keep things symmetrical as we were creating dual supply rails which almost mirrored each other in appearance. We placed the microcontroller in the center for easier trace laying. We wanted to also keep all digital components and signal traces confined to the immediate area around the microcontroller in order to reduce the noise that could be coupled onto the supply lines. Thus our OLED connectors are also near the microcontroller.

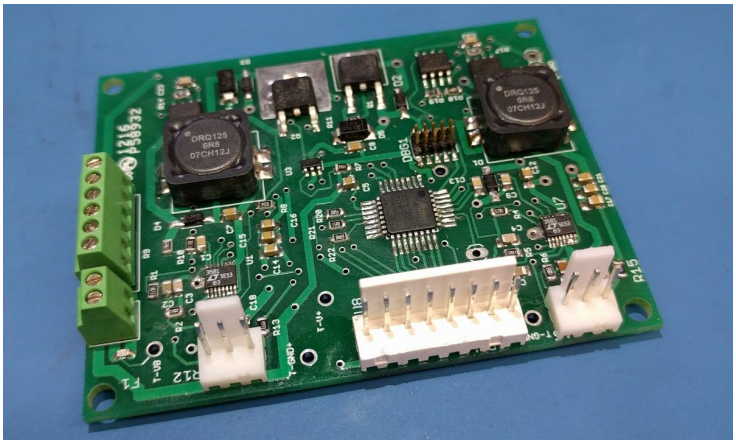
In laying down traces, we used thicker traces for power connections and thinner ones for data lines. To reduce coupling, we opted for a ground plane, although not a separate one for both analog and digital because we felt there would not be a large impact from the digital system due to how little we were actually using the microcontroller. Pads for most components were increased 50% in size in order to make soldering the smaller pins easier.

4 Soldering, Debugging, and Testing

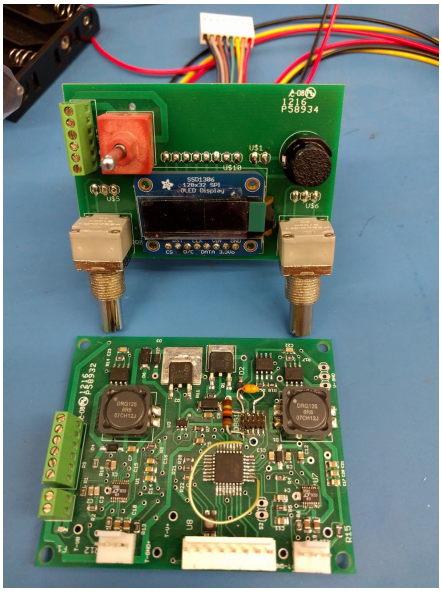
Now came the real challenge. Assembling the board, testing it, and, of course, debugging any mistakes we made in the design process.

4.1 Assembly

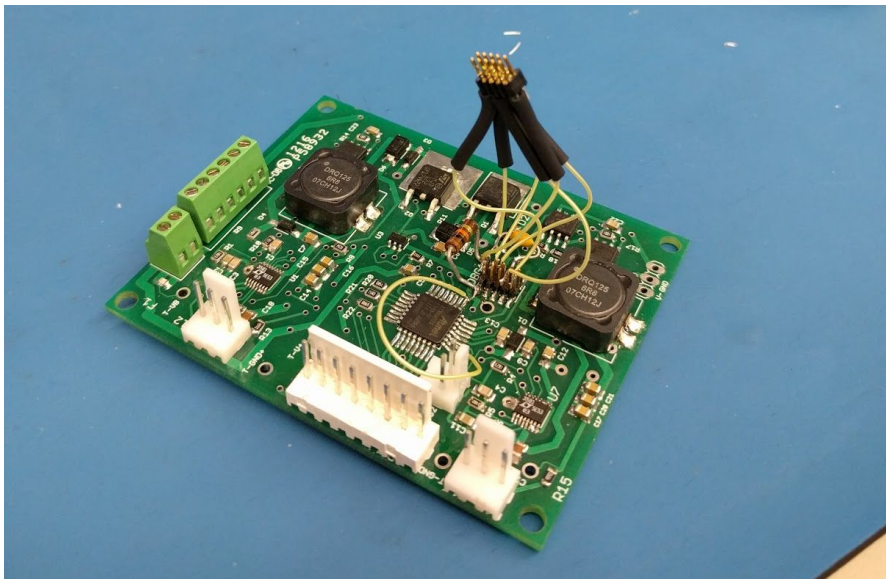
We determined our first design error before beginning board assembly: we had neglected to include the necessary reset circuitry for the microcontroller and debugger. This is our completed board prior to any corrections:



This was not terribly difficult to correct for; all that it required was for us to pull-up and connect together the reset pins on the microcontroller and debug header as shown below:



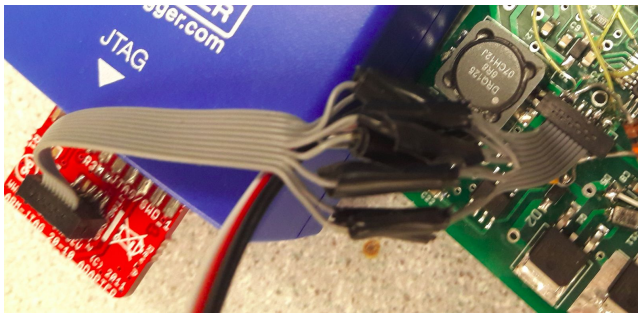
However, while correcting this, and taking a closer look at our layout for the board, we came to a chilling realization. We had improperly configured the pins on the debug header connector. This issue was a bit more difficult to correct for. Our initial strategy involved soldering thin wire-wrap wire onto the pins of the debug header - which was a challenge in and of itself - and then individually routing each wire to the appropriate pins (which we determined through some research on the olimex breakout board for the SAM-ICE debugger) on the debugger ribbon cable. While this strategy allowed us to upload code to the microcontroller on occasion, it was quite unreliable, since the wires often broke off of the header pins (due to how thin they were) causing poor connection between the debugger and the MCU as well as inducing shorts between 3.3V and ground on multiple occasions. Here is what it looked like:



After two days of battling with poor connection and sporadic shorts, we settled on a new strategy to connect the debug header to the debugger. We found an additional debugger ribbon cable, connected it to the improperly configured debug header on our PCB, then routed thicker wires from the end of the ribbon cable to the appropriate pins at the end of another ribbon cable that connected to the debugger. Here is what it looked like:

Although we still experienced poor connection between the debugger and the MCU on occasion, we did not experience the same short circuits that occurred so often in our previous fix. Also, when the connection was bad in our new setup, we knew that we simply needed to re-secure each connection by pushing each wire back into the debugger ribbon cables; we did not need to re-solder each connection onto the debug header connector pins (which were each merely 0.05” apart), risking more shorts, and more frustration. At last, we had fixed this issue sufficiently well to be able to move on with assembly, testing and debugging.

Although we were able to test things, uploading code was still relatively time-consuming. Because of this, we decided to try a third approach: cutting one of the cables and soldering two ends of it together such that it acted as an adapter between our improperly configured debug header and a proper one. Here is a picture of the described solution:



The end result was still not bulletproof, but did allow us to upload new code for a while. However, the power connection started failing after a while and due to time constraints we were not able to fix it entirely.

The most difficult component to solder on our board was the LT3581, switching regulator (perhaps the most important component on our power supply!). The regulator came in a 16-Lead MSOP package, which inherently is a rather difficult component to hand solder due to its small size. However, the component was even more difficult to solder since (1) the component had a ground pad on its underside, and (2) we had neglected to extend pads on the component footprint beyond the length of the legs of the component itself. We were a bit lucky when it came to soldering the ground pad. Since we had placed a wide trace from the ground pad to a large via right next to the component, we were able to scrape off the solder mask from the top of the trace, apply solder to the ground pad, and then place the soldering iron onto the exposed trace to melt and fuse the solder to the bottom ground pad of the regulator. Having finished this, we then proceeded to solder each individual pin on the regulator. Although this process was quite tedious, we were able to solder each pin successfully. We made sure that each connection was complete and that we had no unintended shorts between pins by using an ohmmeter. We had one poor connection, but this was easily fixed (thanks to Cruft Lab’s amazing soldering equipment).

Two additional minor issues that presented themselves to us were (1) that we had neglected to tie the analog supply pin of the uC to 3.3V and (2) that we had accidentally ordered the wrong package size for the LMC6482 package. Tying the supply voltage high was simply done with some wire-wrap wire. Also, fortunately, Avi had additional SOIC-8 op-amp packages available for our use.

Assembling the rest of the board was rather painless. We needed to make sure that components that had a “right” and “wrong” orientation were appropriately oriented and that all of the components were placed in the correct locations. After two-days in the lab, we were ready to start debugging!

4.2 Debugging!

The majority of time spent debugging related to securing the connection between the debugger and the uC, which we described in detail above. After we gained a reliable connection we began to test the regulators. We used the Agilent workbench power supply to test our board because its supply voltage is clean and because it allowed us to limit the supply current to ~100 mA for testing purposes.

We were quite relieved to find that our single supply (positive) regulator seemed to work as expected upon our first test. We achieved an open circuit voltage range from ~1.6V to ~9V. The biggest issue we found was that our potentiometers were oriented such that adjusting the voltage was somewhat unintuitive (increasing the voltage required a counter-clockwise turn). However, this was a simple fix, since the wires connecting the potentiometer to the mainboard were removable; we simply needed to reroute each wire to the appropriate mainboard pins.

The negative supply, in contrast, did not work upon the initial test. Before making the hasty assumption that the regulator, or another component needed to be replaced, we looked back to the schematic to check if the unexpected results stemmed from a design flaw. Our testing indicated that there was a short circuit between the negative supply and ground when powered on, but not when powered off. This led us to look directly at our voltage protection circuitry. Had we improperly oriented the diode in the active power zener? Indeed we had! In fact, the resistor and the zener diode in the active power zener needed to be swapped. After fixing this in the soldering lab, we tested the negative rail again and found that it behaved as expected… sometimes.

Our negative rail would work briefly, then short out, and then work again. This indicated to us that there was some sort of bad connection. After studying our PCB, we found that the connection between our potentiometer and the main board was not secure. This was problematic since the potentiometer is in the feedback path of the voltage regulator. When the connection is bad, feedback is lost, and the regulator output becomes unpredictable. We were able to fix the bad connection, but determined that upon our next design iteration we will have to use more secure connectors between the potentiometers and the main board. This bad

connection was present in both our positive and negative regulators. To shed some positive light on this situation, however, it showed us that out overvoltage protection circuitry was working properly, since the output of each terminal was pulled to ground when the output voltage spiked too high.

We also determined that the fixed resistor that we had placed in the feedback loop of the inverting topology of the LT3581 was too small; we were not able to achieve a voltage higher than approximately 3.5V with the 5k fixed resistor. We looked back at the datasheet and determined that we had improperly calculated the value of this feedback resistor. After replacing it with a 20k resistor, we achieved the voltage range that we had hoped for.

4.3 Testing

Once we were able to reliably adjust the open circuit voltage outputs on the positive and negative supplies from approximately +/- 1.6V to +/- 9V, we began to test how our power supply behaved with a load.

To test our power supply with a load, we plugged the power supply outputs into a breadboard using 0.1” headers, wires and connectors. We placed resistors between power and ground (starting with higher resistances, then moving to lower resistances, and eventually placing small resistors in parallel to avoid exceeding the 0.25W absolute maximum power rating for the resistors we had access to in lab). We used an oscilloscope to measure the DC voltage across the resistor as we adjusted the output voltage. We then AC coupled our output so that we could analyze the cleanliness of our signal, and we used the FFT functionality on the oscilloscope to analyze the signal’s noise density at different frequencies.

5 Firmware

Several components of our design rely on the microcontroller (SAMD-21E). The most important job of the microcontroller is pulling the LT3581 shutdown pins high, allowing the chips to function. Additionally, the microcontroller is responsible for displaying output voltage and current on the OLED screen.

Our approach for writing firmware code starts with initializing shutdown pins and the microcontroller’s internal analog-to-digital converter with the help of Atmel Start (http://start.atmel.com). The exact pin numbers can be found from the schematic (see “Design Process”). Additionally, the board includes an RGB LED useful for debugging purposes, which also got initialized in Atmel Start.

For operating the OLED screen, we built a library based on Adafruit’s SSD1306 Arduino library. Our version currently only includes the most basic functionality necessary to draw single pixels

on our 128x32 OLED screen using software SPI. The library files can be found in Appendix A along with pin declaration header file generated by Atmel Start and a brief description of usage.

Getting OLED display to work took a lot of effort, because of which we did not have enough time to get it to display anything relevant (for example, voltage or current output). However, this is a functionality that we would like to extend in the future (see section “Future Steps”).

6 Results

For the first iteration of our design, the results we achieved were respectable.

6.1 Voltage Range

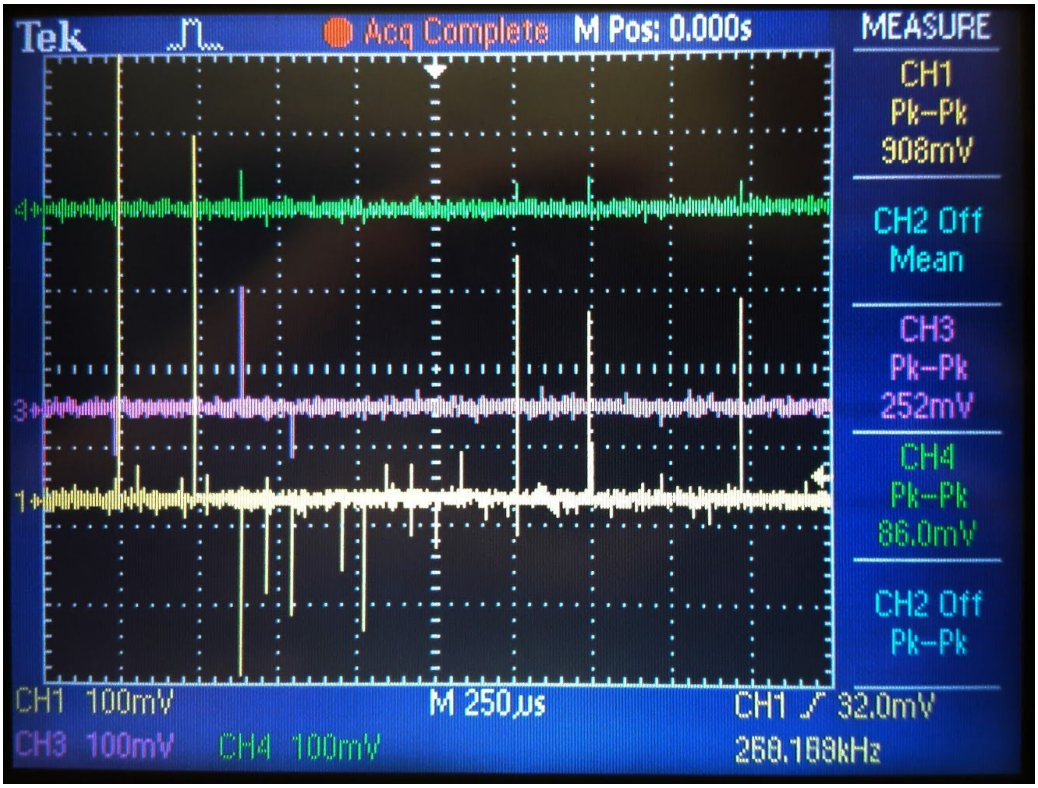
The voltage range that we achieved was very close to the range that we initially specified. As mentioned above, we achieved a range that spans approximately +/- 1.6V to +/- 9V. If we wanted to widen this range, we would need a potentiometer/variable resistor with a maximum resistance larger than 90k (we ordered a 100k potentiometer, which would have extended our voltage range up to +/- 10V, however, the pots that we received both had a maximum resistance of just 90k).

6.2 Noise Levels

Unfortunately, the noise levels that we achieved were a good amount higher than we had hoped, particularly on the positive supply. In testing we determined that the noise levels increased with increasing current demands from the load. Analyzing the FFT for the positive supply voltage, we found that we had noise peaks at 11.8 kHz, 38.5 kHz, 214 kHz and 290 kHz. With loads as small as 25 ohms, at 9V supply, we saw as large as 3V of noise peak to peak.

We are not entirely sure where noise of said frequencies could have originated, especially considering that the switching frequency of our regulators was set at ~450 kHz during testing. Speculation leads us to believe that the bulk of this noise comes from improper layout design. The datasheet for the LT3581 gives suggestions on how to place and rout connections for the device’s supporting circuitry. However, given that each member of our group had little background using the Eagle design tool, and given that we had a somewhat tight time constraint in designing the board, we were unable to take their suggestions fully. Another possible source of error could be the possible high ESR of our output capacitors.

We decided to see how much adding LC filters would reduce the voltage noise at the output. We scoped the open circuit output for the positive supply with a (1) zero stage, (2) one stage, and (3) two stage LC filter with a critical frequency of approximately 5 kHz and found that the noise reduced considerably after each stage:



Here, channel 1 corresponds to voltage (1) (see above), channel 3 to (2) and channel 4 to (3). As can be seen from the measured peak-to-peak voltages (on the right), zero stage LC filter gave us noise with amplitude close to 1V, one stage 250mV and two stages less than 100mV.

The only downside to adding an LC filter to the output of the supply is that it adds a small time delay to the user adjustable voltage. We believe that this is acceptable, however, because our supply will have rather precise user feedback.

6.3 Maximum Current

Our maximum current range is limited by (1) the heat sinking capabilities of the LT3581 and our PCB in general and (2) the fact that as current increases, noise increases, which interferes with the feedback voltage level leading to output instability in severe cases. In our circuit the latter limitation holds precedence.

We grew aware of the second limitation when testing our circuit with small loads (less than 25 ohms). When our voltage hit the upper rail (9V), and our load was pulling approximately 350mA, the noise level at our output was quite large and, ultimately the output began to oscillate. We analyzed the LT3581 datasheet for possible causes of the oscillation and found that the issue could lie in our choice of switching frequency for the component. The switching frequency is dictated by the resistance in the path from the RT pin of the regulator and ground. Choosing a larger resistor reduces the component's switching frequency, while choosing a smaller resistor increases the switching frequency. If the switching frequency is too high, the switches at the

output of the LT3581 could always remain on (or remain on for longer than the “Max On Time”), and if the frequency is too low the switches could stay off for longer than the “Max Off Time” for the component. We decided to swap the resistor between RT and ground first with a larger value, and then with a smaller value, and found that the stability of the circuit increased considerably with the smaller value RT resistor. Choosing an RT of 43k gave us a switching frequency of ~2MHz, and allowed us to supply as much as 400mA with no circuit instability, despite considerable voltage noise.

The voltage noise is likely a major contributor to our circuit’s instability. If we could reduce the voltage noise (by improving the layout, etc), we believe that our maximum current rating would increase.

6.4 Sizing and User Interface

We were very happy with how small we were able to make our circuit board. It is about as wide as a standard breadboard, and spans a little more than a third of the length of the board. Also, our external user display/interface board and battery pack causes the power supply to extend about 3-4 inches vertically. These results fall approximately within our project specifications.

Our user interface is quite intuitive, and allows the user to easily adjust the output voltage by turning potentiometer knobs, as discussed earlier.

We were unable to complete the user display and current limiting functions for our project due to difficulties that we ran into on the firmware side. While we did manage to get OLED screen to display single pixels, we did not have enough time to implement anything beyond that. However, we hope to get the ADC working and the OLED screen to display characters as soon as possible.

7 Future Steps

Given the chance to iterate upon our first design, we would make the following changes:

1.) Redesign the layout of the LT3581 such that it matches more closely with the suggestions made in the LT3581 datasheet. This would hopefully reduce the noise levels on the supply rails (particularly the positive rail).
2.) Design the regulator’s voltage feedback loop such that there is no open circuit when the potentiometers are left disconnected. Alternatively, swap the potentiometer connectors between the mainboard and the external board with more secure/reliable connectors that reduce the chance of cutting off the feedback.
3.) Correct the pin locations on the debug header connector to save us the pain of dealing with poor connections between the debugger and the microcontroller. Also, add the reset circuitry for uC and debugger.

4.) Add a one/two stage LC filter at the output of the power supply to reduce the voltage noise.
5.) Use very low ESR capacitors for the output capacitors of the regulators
6.) Replace the fixed resistor in the feedback loop of negative regulator with a 20k resistor so that voltage range spans from ~-1.6 volts to ~-9 volts.
7.) Order SOIC-8 package for LMC6482.
8.) Use lithium-polymer batteries for improved discharge current performance compared to NiMh.
9.) Reduce the overall length of the board so that it takes up less of the breadboard. Possibly mount the chips on the bottom of the pcb too.
10.) Configure the ADC on the microcontroller.
11.) Extend the OLED display library to show characters in addition to single pixels.

Appendix A: SSD1306 Library and Atmel Start Pin Declarations

SSD1306.h

```
/*
 * SSD1306.h
 * Created: 9.04.2016 0:29:12
 * Author: Sandra Schumann
 * Based on Adafruit_SSD1306.h Arduino library
 */


#ifndef SSD1306_H_
#define SSD1306_H_

#include "atmel_start_pins.h"

#define BLACK 0
#define WHITE 1
#define INVERSE 2

#define SSD1306_128_32

#define SSD1306_LCDWIDTH                  128
#define SSD1306_LCDHEIGHT                 32

#define SSD1306_SETCONTRAST 0x81
#define SSD1306_DISPLAYALLON_RESUME 0xA4
#define SSD1306_DISPLAYALLON 0xA5
#define SSD1306_NORMALDISPLAY 0xA6
```

```
#define SSD1306_INVERTDISPLAY 0xA7
#define SSD1306_DISPLAYOFF 0xAE
#define SSD1306_DISPLAYON 0xAF

#define SSD1306_SETDISPLAYOFFSET 0xD3
#define SSD1306_SETCOMPINS 0xDA

#define SSD1306_SETVCOMDETECT 0xDB

#define SSD1306_SETDISPLAYCLOCKDIV 0xD5
#define SSD1306_SETPRECHARGE 0xD9

#define SSD1306_SETMULTIPLEX 0xA8

#define SSD1306_SETLOWCOLUMN 0x00
#define SSD1306_SETHIGHCOLUMN 0x10

#define SSD1306_SETSTARTLINE 0x40

#define SSD1306_MEMORYMODE 0x20
#define SSD1306_COLUMNADDR 0x21
#define SSD1306_PAGEADDR   0x22

#define SSD1306_COMSCANINC 0xC0
#define SSD1306_COMSCANDEC 0xC8

#define SSD1306_SEGREMAP 0xA0

#define SSD1306_CHARGEPUMP 0x8D

#define SSD1306_EXTERNALVCC 0x1
#define SSD1306_SWITCHCAPVCC 0x2

// Scrolling #defines
#define SSD1306_ACTIVATE_SCROLL 0x2F
#define SSD1306_DEACTIVATE_SCROLL 0x2E
#define SSD1306_SET_VERTICAL_SCROLL_AREA 0xA3
#define SSD1306_RIGHT_HORIZONTAL_SCROLL 0x26
#define SSD1306_LEFT_HORIZONTAL_SCROLL 0x27
#define SSD1306_VERTICAL_AND_RIGHT_HORIZONTAL_SCROLL 0x29
#define SSD1306_VERTICAL_AND_LEFT_HORIZONTAL_SCROLL 0x2A

void ssd1306_begin(uint8_t switchvcc);
void ssd1306_command(uint8_t c);

void clearDisplay(void);
void display(void);

void drawPixel(int16_t x, int16_t y, uint16_t color);
```

```
void fastSPIwrite(uint8_t d);

int8_t _vccstate;


#endif /* SSD1306_H_ */
```

SSD1306.c

```
/*
 * SSD1306.c
 *
 * Created: 9.04.2016 0:34:02
 *  Author: Sandra Schumann
 */

/*********************************************************************
The following is Adafruit SSD1306 Arduino library licensing text.
This library is built on Adafruit SSD1306 Arduino library.

This is a library for our Monochrome OLEDs based on SSD1306 drivers
  Pick one up today in the adafruit shop!
  ------> http://www.adafruit.com/category/63_98
These displays use SPI to communicate, 4 or 5 pins are required to
interface
Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing
products from Adafruit!
Written by Limor Fried/Ladyada  for Adafruit Industries.
BSD license, check license.txt for more information
All text above, and the splash screen below must be included in any redistribution
*********************************************************************/

#include "SSD1306.h"
#include <hal_delay.h>
#include <string.h>

// the memory buffer for the LCD

static uint8_t buffer[SSD1306_LCDHEIGHT * SSD1306_LCDWIDTH / 8] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80,
0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x80, 0x80, 0xC0, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
0x00, 0x00, 0x00, 0x00, 0x80, 0xC0, 0xE0, 0xF0, 0xF8, 0xFC, 0xF8, 0xE0, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x80,
0x80, 0x80, 0x00, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x80, 0x00, 0xFF,
#if (SSD1306_LCDHEIGHT * SSD1306_LCDWIDTH > 96*16)
0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x00, 0x00, 0x80, 0x80, 0x00, 0x00,
0x80, 0xFF, 0xFF, 0x80, 0x80, 0x00, 0x80, 0x80, 0x00, 0x80, 0x80, 0x80, 0x80, 0x00, 0x80, 0x80,
0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x00, 0x00, 0x8C, 0x8E, 0x84, 0x00, 0x00, 0x80, 0xF8,
0xF8, 0xF8, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xE0, 0xE0, 0xC0, 0x80,
0x00, 0xE0, 0xFC, 0xFE, 0xFF, 0xFF, 0xFF, 0x7F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFE, 0xFF, 0xC7, 0x01, 0x01,
0x01, 0x01, 0x83, 0xFF, 0xFF, 0x00, 0x00, 0x7C, 0xFE, 0xC7, 0x01, 0x01, 0x01, 0x01, 0x83, 0xFF,
0xFF, 0xFF, 0x00, 0x38, 0xFE, 0xC7, 0x83, 0x01, 0x01, 0x01, 0x83, 0xC7, 0xFF, 0xFF, 0x00, 0x00,
0x01, 0xFF, 0xFF, 0x01, 0x01, 0x00, 0xFF, 0xFF, 0x07, 0x01, 0x01, 0x01, 0x00, 0x00, 0x7F, 0xFF,
0x80, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x7F, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x01, 0xFF,
0xFF, 0xFF, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x03, 0x0F, 0x3F, 0x7F, 0x7F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xE7, 0xC7, 0xC7, 0x8F,
0x8F, 0x9F, 0xBF, 0xFF, 0xFF, 0xC3, 0xC0, 0xF0, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFC, 0xFC, 0xFC,
0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xF8, 0xF8, 0xF0, 0xF0, 0xE0, 0xC0, 0x00, 0x01, 0x03, 0x03, 0x03,
0x03, 0x03, 0x01, 0x03, 0x03, 0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x03, 0x03, 0x01, 0x01,
0x03, 0x01, 0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x03, 0x03, 0x01, 0x01, 0x03, 0x03, 0x00, 0x00,
0x00, 0x03, 0x03, 0x00, 0x00, 0x00, 0x03, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
0x03, 0x03, 0x03, 0x03, 0x03, 0x01, 0x00, 0x00, 0x00, 0x01, 0x03, 0x01, 0x00, 0x00, 0x00, 0x03,
0x03, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
#if (SSD1306_LCDHEIGHT == 64)
0x00, 0x00, 0x00, 0x80, 0xC0, 0xE0, 0xF0, 0xF9, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x3F, 0x1F, 0x0F,
0x87, 0xC7, 0xF7, 0xFF, 0xFF, 0x1F, 0x1F, 0x3D, 0xFC, 0xF8, 0xF8, 0xF8, 0xF8, 0x7C, 0x7D, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7F, 0x3F, 0x0F, 0x07, 0x00, 0x30, 0x30, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xFE, 0xFE, 0xFC, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0xC0, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x30, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xC0, 0xFE, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7F, 0x7F, 0x3F, 0x1F,
0x0F, 0x07, 0x1F, 0x7F, 0xFF, 0xFF, 0xF8, 0xF8, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFE, 0xF8, 0xE0,
0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFE, 0xFE, 0x00, 0x00,
0x00, 0xFC, 0xFE, 0xFC, 0x0C, 0x06, 0x06, 0x0E, 0xFC, 0xF8, 0x00, 0x00, 0xF0, 0xF8, 0x1C, 0x0E,
0x06, 0x06, 0x06, 0x0C, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0xFE, 0xFE, 0x00, 0x00, 0x00, 0x00, 0xFC,
0xFE, 0xFC, 0x00, 0x18, 0x3C, 0x7E, 0x66, 0xE6, 0xCE, 0x84, 0x00, 0x00, 0x06, 0xFF, 0xFF, 0x06,
0x06, 0xFC, 0xFE, 0xFC, 0x0C, 0x06, 0x06, 0x06, 0x00, 0x00, 0xFE, 0xFE, 0x00, 0x00, 0xC0, 0xF8,
0xFC, 0x4E, 0x46, 0x46, 0x46, 0x4E, 0x7C, 0x78, 0x40, 0x18, 0x3C, 0x76, 0xE6, 0xCE, 0xCC, 0x80,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x01, 0x07, 0x0F, 0x1F, 0x1F, 0x3F, 0x3F, 0x3F, 0x3F, 0x1F, 0x0F, 0x03,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x0F, 0x00, 0x00,
0x00, 0x0F, 0x0F, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x0F, 0x00, 0x00, 0x03, 0x07, 0x0E, 0x0C,
0x18, 0x18, 0x0C, 0x06, 0x0F, 0x0F, 0x0F, 0x00, 0x00, 0x01, 0x0F, 0x0E, 0x0C, 0x18, 0x0C, 0x0F,
0x07, 0x01, 0x00, 0x04, 0x0E, 0x0C, 0x18, 0x0C, 0x0F, 0x07, 0x00, 0x00, 0x00, 0x0F, 0x0F, 0x00,
0x00, 0x0F, 0x0F, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x0F, 0x00, 0x00, 0x00, 0x07,
0x07, 0x0C, 0x0C, 0x18, 0x1C, 0x0C, 0x06, 0x06, 0x00, 0x04, 0x0E, 0x0C, 0x18, 0x0C, 0x0F, 0x07,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

```
#endif
#endif
};

// the most basic function, set a single pixel
void drawPixel(int16_t x, int16_t y, uint16_t color) {
  if ((x < 0) || (x >= SSD1306_LCDWIDTH) || (y < 0) || (y >= SSD1306_LCDHEIGHT))
    return;

  // x is which column
    switch (color)
    {
      case WHITE:   buffer[x+ (y/8)*SSD1306_LCDWIDTH] |=  (1 << (y&7)); break;
      case BLACK:   buffer[x+ (y/8)*SSD1306_LCDWIDTH] &= ~(1 << (y&7)); break;
      case INVERSE: buffer[x+ (y/8)*SSD1306_LCDWIDTH] ^=  (1 << (y&7)); break;
    }

}

void ssd1306_begin(uint8_t vccstate) {
  _vccstate = vccstate;

  // set pin directions
  gpio_set_pin_direction(DC, GPIO_DIRECTION_OUT);
  gpio_set_pin_function(DC, GPIO_PIN_FUNCTION_OFF);
  gpio_set_pin_direction(CS, GPIO_DIRECTION_OUT);
  gpio_set_pin_function(CS, GPIO_PIN_FUNCTION_OFF);

  // set pins for software-SPI
  gpio_set_pin_direction(SID, GPIO_DIRECTION_OUT);
  gpio_set_pin_function(SID, GPIO_PIN_FUNCTION_OFF);
  gpio_set_pin_direction(CLK, GPIO_DIRECTION_OUT);
  gpio_set_pin_function(CLK, GPIO_PIN_FUNCTION_OFF);

  // Setup reset pin direction (used by both SPI and I2C)
  gpio_set_pin_direction(RST, GPIO_DIRECTION_OUT);
  gpio_set_pin_function(RST, GPIO_PIN_FUNCTION_OFF);
  gpio_set_pin_level(RST, true);
  // VDD (3.3V) goes high at start, lets just chill for a ms
  delay_ms(1);
  // bring reset low
  gpio_set_pin_level(RST, false);
  // wait 10ms
  delay_ms(10);
  // bring out of reset
  gpio_set_pin_level(RST, true);

  // Init sequence
  ssd1306_command(SSD1306_DISPLAYOFF);                    // 0xAE
  ssd1306_command(SSD1306_SETDISPLAYCLOCKDIV);            // 0xD5
  ssd1306_command(0x80);                                  // the suggested ratio 0x80
```

```
  ssd1306_command(SSD1306_SETMULTIPLEX);                  // 0xA8
  ssd1306_command(SSD1306_LCDHEIGHT - 1);

  ssd1306_command(SSD1306_SETDISPLAYOFFSET);              // 0xD3
  ssd1306_command(0x0);                                   // no offset
  ssd1306_command(SSD1306_SETSTARTLINE | 0x0);            // line #0
  ssd1306_command(SSD1306_CHARGEPUMP);                    // 0x8D
  if (vccstate == SSD1306_EXTERNALVCC)
    { ssd1306_command(0x10); }
  else
    { ssd1306_command(0x14); }
  ssd1306_command(SSD1306_MEMORYMODE);                    // 0x20
  ssd1306_command(0x00);                                  // 0x0 act like ks0108
  ssd1306_command(SSD1306_SEGREMAP | 0x1);
  ssd1306_command(SSD1306_COMSCANDEC);

 #if defined SSD1306_128_32
  ssd1306_command(SSD1306_SETCOMPINS);                    // 0xDA
  ssd1306_command(0x02);
  ssd1306_command(SSD1306_SETCONTRAST);                   // 0x81
  ssd1306_command(0x8F);

#elif defined SSD1306_128_64
  ssd1306_command(SSD1306_SETCOMPINS);                    // 0xDA
  ssd1306_command(0x12);
  ssd1306_command(SSD1306_SETCONTRAST);                   // 0x81
  if (vccstate == SSD1306_EXTERNALVCC)
    { ssd1306_command(0x9F); }
  else
    { ssd1306_command(0xCF); }

#elif defined SSD1306_96_16
  ssd1306_command(SSD1306_SETCOMPINS);                    // 0xDA
  ssd1306_command(0x2);   //ada x12
  ssd1306_command(SSD1306_SETCONTRAST);                   // 0x81
  if (vccstate == SSD1306_EXTERNALVCC)
    { ssd1306_command(0x10); }
  else
    { ssd1306_command(0xAF); }

#endif

  ssd1306_command(SSD1306_SETPRECHARGE);                  // 0xd9
  if (vccstate == SSD1306_EXTERNALVCC)
    { ssd1306_command(0x22); }
  else
    { ssd1306_command(0xF1); }
  ssd1306_command(SSD1306_SETVCOMDETECT);                 // 0xDB
  ssd1306_command(0x40);
```

```
  ssd1306_command(SSD1306_DISPLAYALLON_RESUME);           // 0xA4
  ssd1306_command(SSD1306_NORMALDISPLAY);                 // 0xA6

  ssd1306_command(SSD1306_DEACTIVATE_SCROLL);

  ssd1306_command(SSD1306_DISPLAYON);//--turn on oled panel
}

void ssd1306_command(uint8_t c) {
    // SPI
    gpio_set_pin_level(CS, true);
    gpio_set_pin_level(DC, false);
    gpio_set_pin_level(CS, false);
    fastSPIwrite(c);
    gpio_set_pin_level(CS, true);
}

void display(void) {
  ssd1306_command(SSD1306_COLUMNADDR);
  ssd1306_command(0);   // Column start address (0 = reset)
  ssd1306_command(SSD1306_LCDWIDTH-1); // Column end address (127 = reset)

  ssd1306_command(SSD1306_PAGEADDR);
  ssd1306_command(0); // Page start address (0 = reset)
  ssd1306_command(3); // Page end address

  // SPI
  gpio_set_pin_level(CS, true);
  gpio_set_pin_level(DC, true);
  gpio_set_pin_level(CS, false);

  for (uint16_t i=0; i<(SSD1306_LCDWIDTH*SSD1306_LCDHEIGHT/8); i++) {
    fastSPIwrite(buffer[i]);
  }
  gpio_set_pin_level(CS, true);
}

// clear everything
void clearDisplay(void) {
  memset(buffer, 0, (SSD1306_LCDWIDTH*SSD1306_LCDHEIGHT/8));
}


void fastSPIwrite(uint8_t d) {
  for(uint8_t bit = 0x80; bit; bit >>= 1) {
    gpio_set_pin_level(CLK, false);
    if(d & bit) gpio_set_pin_level(SID, true);
    else        gpio_set_pin_level(SID, false);
    gpio_set_pin_level(CLK, true);
  }
```

```
}
```

atmel_start_pins.h

```
/*
 * Code generated from Atmel Start.
 *
 * This file will be overwritten when reconfiguring your Atmel Start project.
 * Please copy examples or other code you want to keep to a separate file
 * to avoid losing it when reconfiguring.
 */
#ifndef ATMEL_START_PINS_H_INCLUDED
#define ATMEL_START_PINS_H_INCLUDED

#include <hal_gpio.h>

#define LEDG  GPIO(GPIO_PORTA, 8)
#define LEDB  GPIO(GPIO_PORTA, 9)
#define LEDR  GPIO(GPIO_PORTA, 10)
#define SHDN2 GPIO(GPIO_PORTA, 22)
#define SHDN1 GPIO(GPIO_PORTA, 23)
#define SID   GPIO(GPIO_PORTA, 16)
#define CLK   GPIO(GPIO_PORTA, 17)
#define CS       GPIO(GPIO_PORTA, 18)
#define DC       GPIO(GPIO_PORTA, 24)
#define RST   GPIO(GPIO_PORTA, 25)

#endif // ATMEL_START_PINS_H_INCLUDED
```

SSD1306 Library Usage

The recommended usage of SSD1306 library for this power supply is as follows (an example).

```
// Initializing SSD1306
ssd1306_begin(SSD1306_SWITCHCAPVCC);
// Displaying contents of the buffer
display();
// Clearing the contents of the buffer
clearDisplay();
// Drawing a white pixel at location x,y
drawPixel(x, y, WHITE);
display();
```