Affordable Power Supply

Senior Design Final Report

Team SDMay21-47

Client/Advisor: Dr. Gary Tuttle

Group Members:
Ben Almquist, Mary Le, Michael Neilson,
Adam Simodynes, Krishan Sritharan, Chance Webster

Team Website:
https://sdmay21-47.sd.ece.iastate.edu/

1 Introduction

1.1 Acknowledgements

Our team would like to thank Professor Tuttle for advising and supporting us in our project. We appreciate all the advice he has given for our project.

1.2 Problem and Project Statement

With Covid-19 limiting lab availability to students, it is difficult for electrical engineering students to work with any hardware because they do not have power supplies to test their circuits. It would be beneficial for students studying electrical engineering to have access to a power supply at home during this pandemic.

The solution for this problem is to design an affordable power supply with a small form factor that any electrical engineering student can easily build and operate from home. This will allow students to perform any hardware labs for their electrical engineering courses without requiring access to an on-campus lab.

1.3 Requirements

- Input Voltage
  - $120V_{RMS}$AC wall power
- 4 voltage outputs
  - 2 to 25 VDC
    - 1 A max current
    - Standard binding posts output connector
  - -2 to -25VDC
    - 1 A max current
    - Standard binding post output connector
  - 1 to 10VDC
    - 1 A max current
    - Standard binding post output connector
  - Fixed 5 VDC
    - 2 A max current
    - Standard USB type A output connector
- Box dimensions of 7 inches long, 4 inches wide, 2 inches deep

- Output display to display the voltage levels
- Cost of final product to be below $100 to purchase
- (Optional) Output voltage indications for the various supplies provided by up to 4 displays — LED, LCD, or OLED. Displays may be shared between individual supplies.
- (Optional) Output current indications for the various supplies provided by up to 4 displays — LED, LCD, or OLED. Displays may be shared between individual supplies.

1.4 Intended Users and Uses

The intended users of the product are electrical and computer engineering students. The uses for the product are to test hardware for electrical and computer engineering courses, more specifically for the hardware laboratories within the courses.

1.5 Related Products

- Keysight E3633A - Single output
  - One adjustable 20V, 10A output
  - $1,663
- Rigol DP832 - Triple output
  - Two adjustable 30V, 3A outputs
  - One adjustable 5V, 3A output
  - $473
- Extech 382270 - Quad output
  - Two adjustable 30V, 5A outputs
  - One fixed 6.5V, 3A output
  - One fixed 15V, 1A output
  - $490
- B&K 1513 - Single output
  - One selectable output, up to 12V, 1A
  - $50.50

The vast majority of market power supplies are significantly more expensive than our final price goal of $100. The cheapest power supply listed on DigiKey is $50.50. However, this supply only has one, low-power output. The cheapest quad output power supply listed on DigiKey is $490. Meeting our price goal makes our design the best option in terms of cost-utility tradeoff.

1.6 Operational Environment

The operational environment of the power supply will be a students dorm or apartment. These locations may not be as clean and orderly as lab spaces, so the device should be rugged enough to survive being moved frequently. It may also be exposed to other hazards not found in labs such as food and liquids.

1.7 Expected End Product and Deliverables

The deliverables for our product are a working prototype of the product, an instruction manual for building the power supply and how to use it. The prototype will meet all the requirements specified previously. It will come with a PCB, complete set of parts, and a box.

The instruction manual will consist of the list of parts needed to build the power supply. It will also include a list of instructions for building the power supply and how to operate it.

1.8 Development Standards and Practices Used

- IEEE 1100 - 2005: Recommended Practice for Powering and Grounding Electronic Equipment
  - This standard provides guidance on how to enhance performance and keep safety protocols. It also includes how to protect the devices and resolve any problems with select instruments.
- IEEE 1332 - 2012: Standard Reliability Program for the Development and Production of Electronic Products
  - This standard provides guidance for communication between designer and consumer on best practices for reliability and consistency of electronic products.

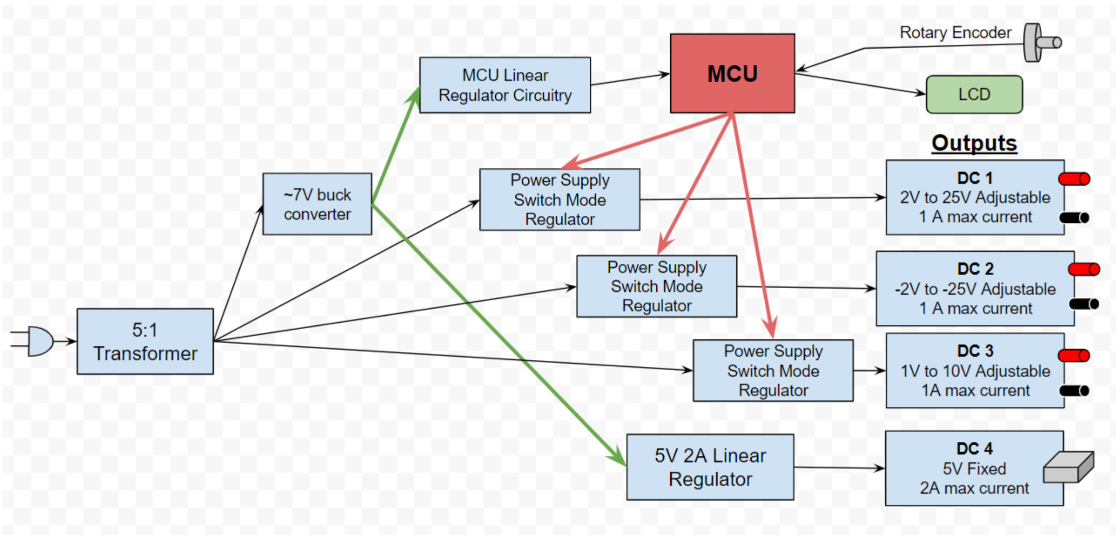
2 Design

2.1 Overview of the Design

The initial aspect of the design was deciding what types of I/O we were going to use. The I/O uses a rotary encoder, a simple 2x16 LCD display, four buttons, and four different voltage outputs. The way in which we have decided to implement the three variable voltage outputs (+25, -25, and +10) is via Buck converters which are controlled by a microcontroller, efficient, and cheaper to implement. The last of the four voltage outputs is done with a linear voltage regulator that has a current limit feature built in at

2A. The fixed output is meant to provide a 5V output, and using a regulator is an easy and cost effective solution to this. All of the design aspects have been chosen to minimize cost while maintaining as much quality as possible.

Block Diagram



2.2 Electrical Component Selection and Schematic Capture

Once we had decided upon our general I/O and block diagram, our next step was to select parts for our most critical components such as the MCU, switching regulators, and linear regulator. Because the board is intended to be assembled by students who may be inexperienced at soldering, we attempted to use as many through-hole components as possible. When selecting the MCU, our initial choice was to use the Atmega 328, as we each had experience with it and knew that it would make prototype testing easier because we could test with an Arduino. However, as the project evolved, it was discovered that we would need more I/O pins than the Atmega 328 could provide, and we wanted to include extra I/O pins to allow for user expansion in the future. This led us to switch to a PIC18F26Q10 microcontroller, which has more I/O pins, an internal oscillator, overall cheaper, and one of our group members had experience working with.

The next pieces we needed to select were the three different switching regulators. For each we first defined the parameters that we needed, such as output voltage range, max output current, input voltage, and price. For each of the regulators we searched digikey for switching regulators using filters on these categories. It was difficult to find regulators that fit our requirements and were inexpensive. For this reason, we had to end up using surface mount parts for the +10V and -25V regulators, as they were the only parts that fit our specifications. We were however able to find through hole components for the +25V switching regulator and the 5V linear regulator.

Once we had our regulator ICs picked out, we were able to begin picking out components for the surrounding circuitry. For this portion of the component selection process, we had to dig into each datasheet to find how to pick values and specifications for the surrounding circuitry. Once we had requirements for each component, we searched for them on DigiKey and found the closest matches we could at a reasonable price. The final step was selecting the peripheral components such as the LCD, buttons, and rotary encoder. These pieces were easier to select as they had less defined specifications, and were easier to find cheap options.

In parallel to this process, we began to build the circuit schematics in KiCad. For each of the regulators we generated the schematics with the help of the datasheet, and filled in specific values and part numbers as we found them. On the microcontroller schematics we mapped out the specific pins we would use for inputs and outputs, such as for the LCD, rotary encoder, buttons, and digital potentiometer controls. Finally, we combined all of the schematics by using functional blocks to link to each of the individual schematics' sheets.

2.3 Evolution of the Design

The biggest change from our initial design to our final design was the input power transformation. At first, we planned to use an AC wall adapter to get the voltage down to around 24Vrms. However, after looking at part we had specified for this, we found out it didn't have the power capabilities needed and it was eating one fourth of our budget, so we decided to switch from the external supply to an internal one which includes a 5:1 transformer.

Another design change we made was switching from a metal enclosure to a 3D printed enclosure. Our main motivation for this change was that none of us had experience with or access to a CNC machine, so hand drilling into the enclosure would have been a time consuming and potentially dangerous process, and would also require access to a shop. For this reason, we decided that 3D printing an enclosure would be much quicker and simpler for our prototype, as one of our team members has experience with 3D printing. However, we also decided that if the design is to be mass produced (>100 units) in the future, we would recommend using metal enclosures and a CNC machine, as this would be a much cheaper option for a larger-scale manufacturing operation.

2.4 Safety and Security Considerations

Considering that this is a power supply unit that will be used by underclass students who may be new to working with electronics, it was important that we consider the safety of the user and of the unit. The first step was limiting the output current of the outputs. We made sure that the switching regulators had max output currents of only 1 amp, and 2 amps for the linear regulator. This ensures that the circuits will not accidentally output more current than they are supposed to.

Another safety measure we took was limiting the amount of exposed wire that would be carrying high voltages. It was important that the transformer be as close as possible to the input power connector so that long wires carrying 120 volts would not be exposed to the user.

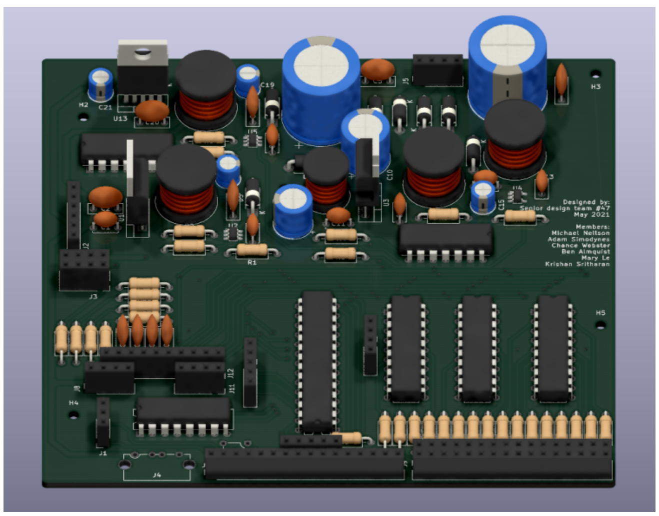
Next, we made sure to discuss safe operation of the power supply in our manual. Some safety tips we give are keep food and liquids away from the product and keep the product on a flat, level surface in a stationary position. We also recommend safety and supervision during soldering, as students may be inexperienced.

3 Implementation

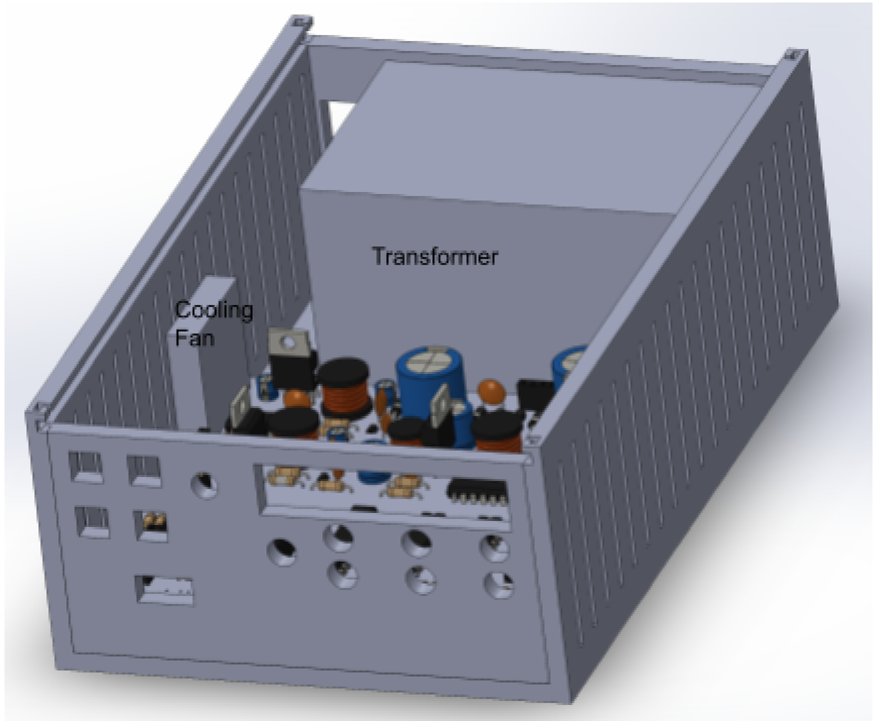
3.1 Implementation Details

Once the initial block diagram was complete we started the implementation by selecting the key components for each section. With the components selected, we designed the schematics to accompany the selected components to meet the requirements of the device. The next step in implementation was to combine all parts of the schematic into one full schematic. With the complete schematic finalized, we completed the layout for the PCB. Finally, we designed the case to house the PCB and other final components, as well as allow for an organized user interface.

PCB with components



Complete Model of Final Product



4 Testing

4.1 Unit Testing

Unit testing will be completed by testing each individual component separately to verify each component's functionality. For each of the voltage regulator ICs, we built the designed schematic on a breadboard and tested it using a lab supply as input and a multimeter to read the voltage output. During the initial run of testing, we used standard analog potentiometers in place of the digital potentiometers in the design. We would, however, use the digital potentiometers during interface testing as well.

4.2 Interface Testing

Our interface is designed to be an LCD display to display the output voltages, a button to select the output the user chooses to change, and a rotary encoder to modify the voltage level at the specified output. The tests that will need to be completed are verifying that the LCD displays the correct voltage, the button changes the selected output voltage on the LCD display, and the rotary encoder correctly changes the output voltage level. All in all, the three components of the user interface will be tested individually first, to verify they are operational, before testing the functionality of the components together.

4.3 Acceptance Testing

Our acceptance testing for the power supply is done by testing each output voltage with a multimeter to ensure the voltage ranges are correct. Testing accuracy of the voltage on the LCD will be another factor within our acceptance testing. This will also be done using a multimeter at each output and using the encoder to change the values to verify the values shown on the LCD are within the restraints.

4.4 Results

The results from our testing were exactly as expected. The unit test results for each individual section were as follows:

- 1 to 10VDC Variable Output
  - Results: The output was accurately within the 1 to 10 VDC range.
- 2 to 25VDC Variable Output
  - Results: The output was close to a range 1 to 24 VDC and with a small change in resistance value we achieved the 2 to 25 VDC range.
- -2 to -25 VDC Variable Output
  - Results: The initial result was -4 to -21 VDC and with a change in component values the voltage range is now -2 to -25 VDC.
- Fixed 5 VDC
  - Results: The results were exactly 5 VDC.

Testing the interface was done by connecting the microcontroller and the IO via breadboards to ensure the IO functioned as expected with the circuitry. The results for these tests were close to as expected as each variable output measured by a multimeter was within 0.1 VDC of the voltage on the LCD. For the acceptance testing the PCB is needed to complete the testing, and we encountered an issue with our PCB order and have not received the PCB. The results that we have received thus far have been very good and expect nothing less from the PCB design.

5 Closing Material

5.1 Conclusion

This report for our power supply details our procedure for designing and building a power supply in an at-home environment. The goal was to give students an option for completing labs and projects in a personal environment. The requirements include having four total outputs, assembled all on a PCB board, and powered by a 24-V transformer. A main priority was minimizing the cost to keep it affordable for students. Our final approximate price per unit is $95. This is below our cost ceiling of $100 and is significantly cheaper than quad output power supplies on the market.

We completed a fundamental design that includes a schematic and layout meeting all technical requirements. We added the additional features of a microcontroller and LCD for improved accessibility and ease-of-use. We implemented and tested each individual component for functionality before implementing and testing a larger system prototype. We successfully completed all technical requirements, achieving the necessary voltage and current values at each output while keeping the cost as low as possible.

5.2 Future Plans

The casing for this design is not intended for mass production. The 3D printing procedure can be time consuming and expensive. In the event that numerous models are requested, an alternate casing choice would be more cost and time efficient. A 3D printed case was the ideal choice for our prototype because of the customization options. Once an effective case layout is determined, a pre-made casing could be selected that requires minimal labor from the user to prepare for the electronic components and could be ordered in bulk.

5.3 References

Masoud Farhoodnea, Azah Mohamed, Hussain Shareef, "A comparative study on the performance of custom power devices for power quality improvement", *Innovative Smart Grid Technologies - Asia (ISGT Asia) 2014 IEEE*, pp. 153-157, 2014.

Ming Yang, Digvijay Deswal, Francisco de León, "Mitigation of Half-Cycle Saturation of Adjacent Transformers During HVDC Monopolar Operation—Part I: Mitigation Principle and Device Design", *Power Delivery IEEE Transactions on*, vol. 34, no. 6, pp. 2232-2239, 2019.

Tuttle, Gary. "EE 333 Labs." *EE 333 : Lab*, Iowa State University, 29 Oct. 2020, tuttle.merc.iastate.edu/ee333/lab.htm.

5.4 Appendices

5.4.1 Appendix I - Operational Manual

Instructions for Assembly:

1. Ensure you have all the materials listed in the parts list
2. **Soldering**
   Note: for easiest assembly, solder components in the order they are listed below
   a. SMD components
   b. Smallest non-electrolytic capacitors
   c. Resistors and diodes
   d. DIP ICs
   e. Smallest electrolytic capacitors
   f. Inductors
   g. Large electrolytic caps
   h. Large transistors
   i. Off-board components
   Be sure to have your soldering checked by a TA or professor before beginning enclosure assembly
3. **Enclosure Assembly**
   a. After soldering, align the PCB board in the case at the indicated location.
   b. Insert the push buttons, LCD, binding posts, and rotary encoder at the indication locations and wire them to the PCB
   c. Place the transformer in the back of the enclosure.
   d. Insert the fan at the indicated position.
   e. Insert the front and back pieces into the base.
   f. Slide the top onto the base and insert the locks to ensure the parts stay together.

Product Operation Procedure

1. Insert female end of power cord into power outlet connector socket on the enclosure
2. Insert male end of power cord into wall outlet
   a. Note: there is not a power switch on the assembly, so the assembly will power on at this point
3. Insert banana cables to desired outputs and ground binding posts
4. Use toggle button to select output to modify
   a. On startup, the +25 V output will be selected
5. Use rotary encoder to adjust output value on desired voltage output
6. Use output enable button to enable voltage to the output pins

Button Function Quick Reference

- Output Enable(Button 2)
  - Toggles the selected output on or off
- Output Select(Button 1)
  - Output select cycle: +25 → -25 → +10 → +5 → +25
  - Starts on + 25 supply
- Output Adjust(Rotary encoder)
  - Either increases or decreases the selected output's voltage(if not on the fixed 5) based upon which direction the encoder goes and how far

Guide for Safe Operations

- Although the power supply is intended to be portable, it is important to still follow electronics lab best practices. This includes making sure that your workspace is clean and free from food or drink
- Be sure to have your soldering checked soldered and checked by a TA or professor
- While the device is plugged in, keep the enclosure sealed
- When banana cables are connected, make sure that the leads do not ever touch each other
- Keep the enclosure in a well ventilated area, and ensure there is adequate airflow to the fan and air intake vents

Parts List:

Onboard Components

Part:	**Reference:**
330 nF Cap	C1
100 nF Cap	C2, C5, C6, C8, C18, C20, C22, C23, C24, C25
2.2 mF Cap	C3, C4
2.2 uF Cap	C7, C17
10 uF Cap	C9, C19
220 uF Cap	C10
600 pF Cap	C11
82 uF Cap	C12
4.7 uF Cap	C13

1 uF Cap	C14
22 uF Cap	C15, C21
220 pF Cap	C16
Schottky Diode	D1-D9
1x3 2.54mm Header Pin	J1
1x6 2.54mm Header Pin	J2
2x4 2.54mm Header Pin	J3
USB Type A	J4
Power Outlet Connector Socket - IEC320C13	J5
1x16 2.54mm Header Pin	J7
2x16 2.54mm Header Pin	J9
47 uH Inductor	L1, L5
68 uH Inductor	L2
22 uH Inductor	L3, L4
100 kΩ Res	R1
8.2 kΩ Res	R2
1 kΩ Res	R3, R9-R33
6.8 kΩ Res	R4, R6
820 Ω Res	R5, R7
3.3 kΩ Res	R8
10 kΩ Res	R34
1 MΩ Potentiometer	RV1
24 V Transformer - F8-24	T1
5 V Linear Regulator - L7805	U1
1-40 V Switching Regulator - LMR14010	U2, U5
1-37 V Switching Regulator - LM2595T	U3
Negative 1-34 V Switching Regulator - LT1931ES5	U4

Microcontroller - PIC18F26Q10	U6
Digital Potentiometer - MCP4231	U7, U8
3-to-8 Decoder - 74HC238	U9
D Flip-Flops - 74HC377	U10, U11, U12
Linear LDO Regulator - BA50DD0WT	U13

Off-Board Components (connected by wire to PCB)

Part:	**Quantity:**
Red LED	16
Rotary Encoder	1
LCD	1
Push Button	4
Fan	2
Binding Post (Red)	4
Binding Post (Black)	1
Binding Post (Green)	1
Casing	1

5.4.2 Appendix II - Alternative versions of the Design

The original plan for this design did not include a microcontroller or an LCD. The user would rotate mechanical potentiometers to change the voltage levels and could read the voltage levels using a multimeter. However, we decided that this implementation would not be ideal for our intended users. Many students do not have access to a multimeter outside of labs, making it difficult to tell the precise voltage level. The incorporation of a microcontroller allowed the voltage level to be read and displayed on an LCD, making it much simpler for students to complete lab work and projects. The use of a microcontroller also allowed us to switch from mechanical potentiometers to digital potentiometers. Instead of rotating a unique potentiometer for each output, the user can select which output to adjust with push buttons and adjust using a single rotary encoder. This change was implemented in order to improve the ease-of-use for the user.

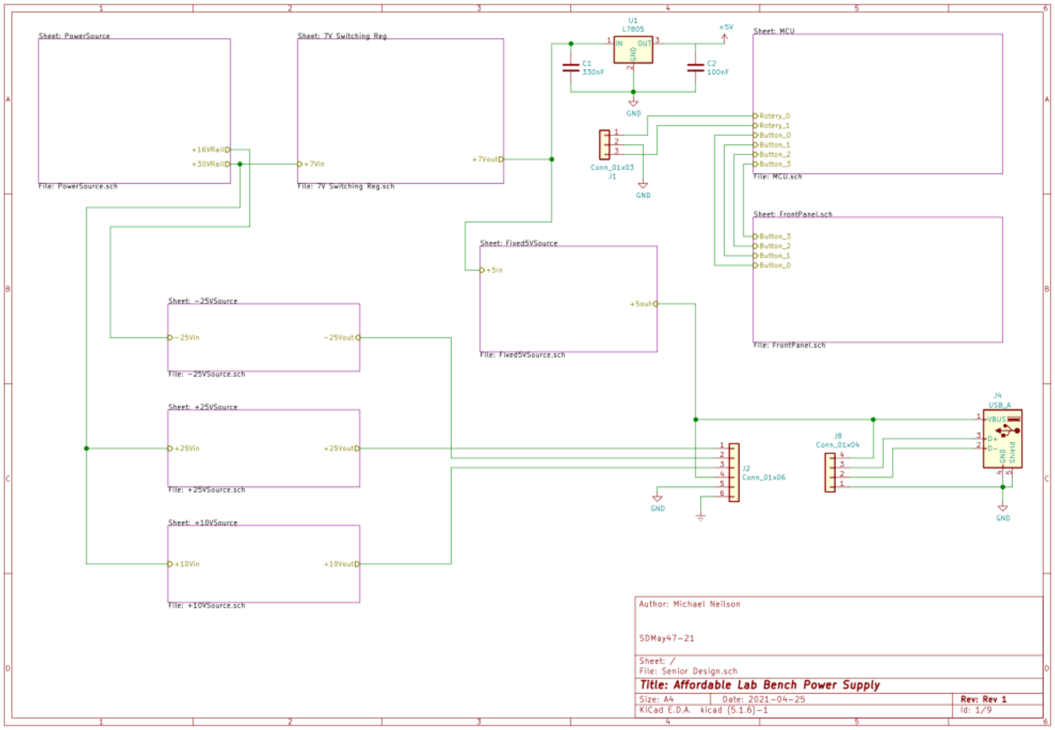
Another alternate version that was used included ordering a pre-assembled casing to house the PCB. This decision would have meant drilling holes in the case for the I/O components. It was then decided that this extra step would add too much burden to the assembly and preparation of the design. We settled on a 3D printed casing to maximize the amount of customization in the design without requiring user assembly for this component.

5.4.3 Appendix III - Other Considerations

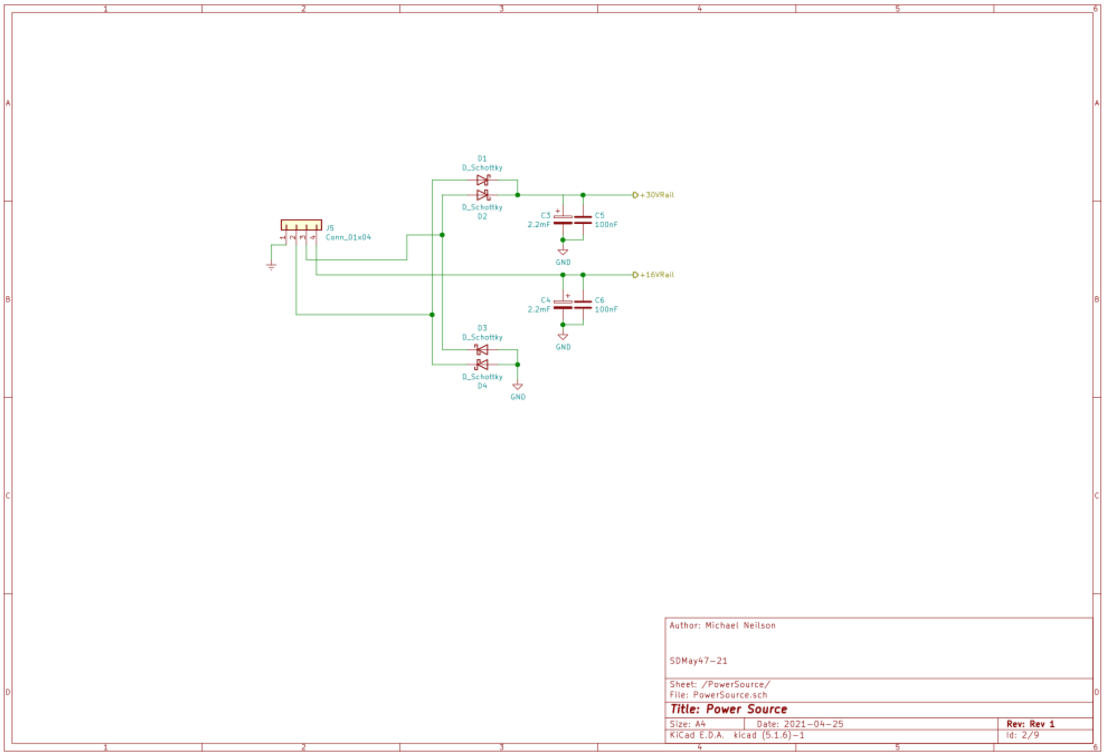
This project applied many of the skills that we have learned as engineering students. It required both technical and non-technical skills to balance technical requirements with project management. We applied technical skills such as part selection, breadboard prototype construction, and schematic and layout design. For non-technical skills, this project required time management, communication, and task delegation. As a group, we saw improvement in the non-technical skills as the project progressed.

5.4.4 Appendix VI - Schematics

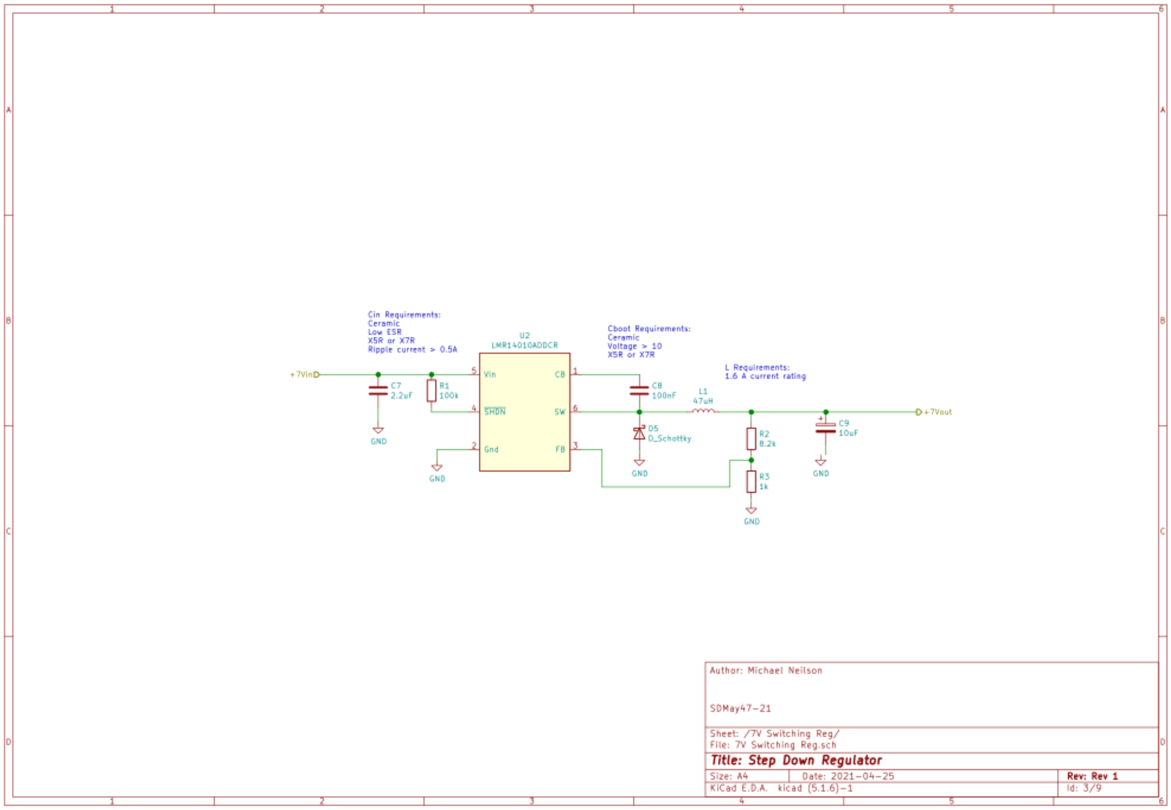
System-Level Drawing



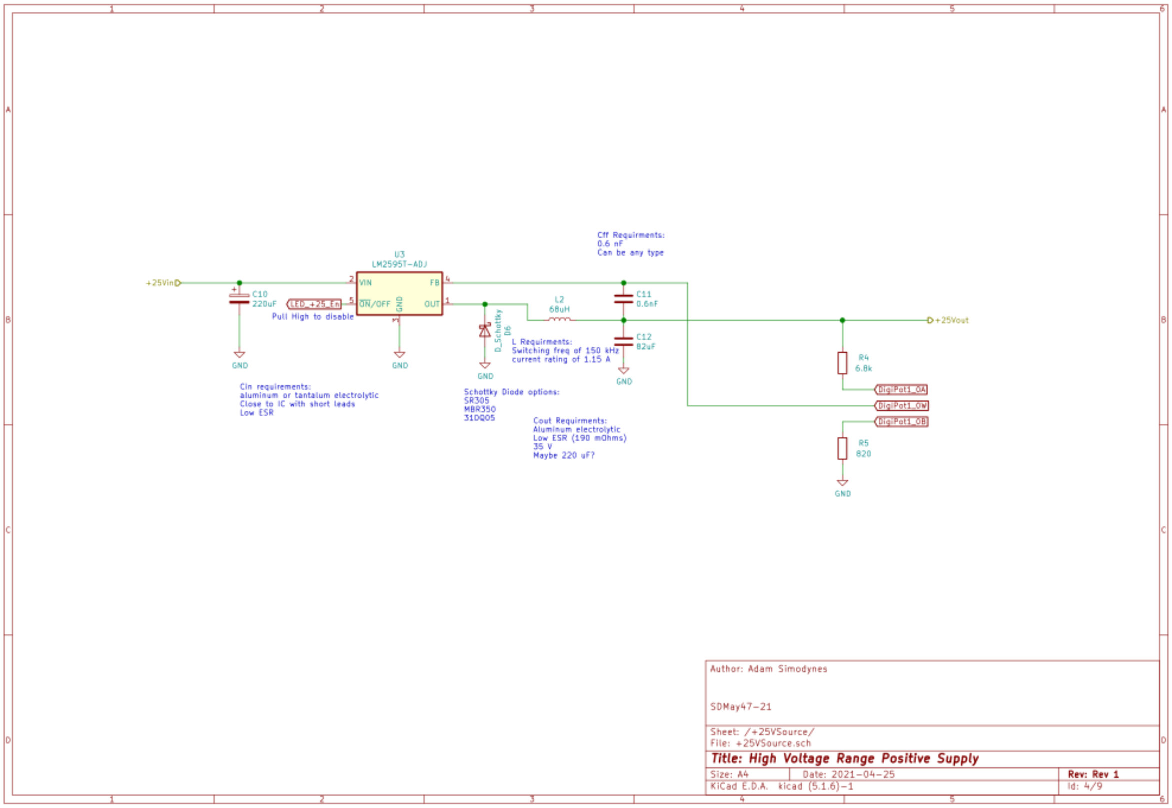
Power Source



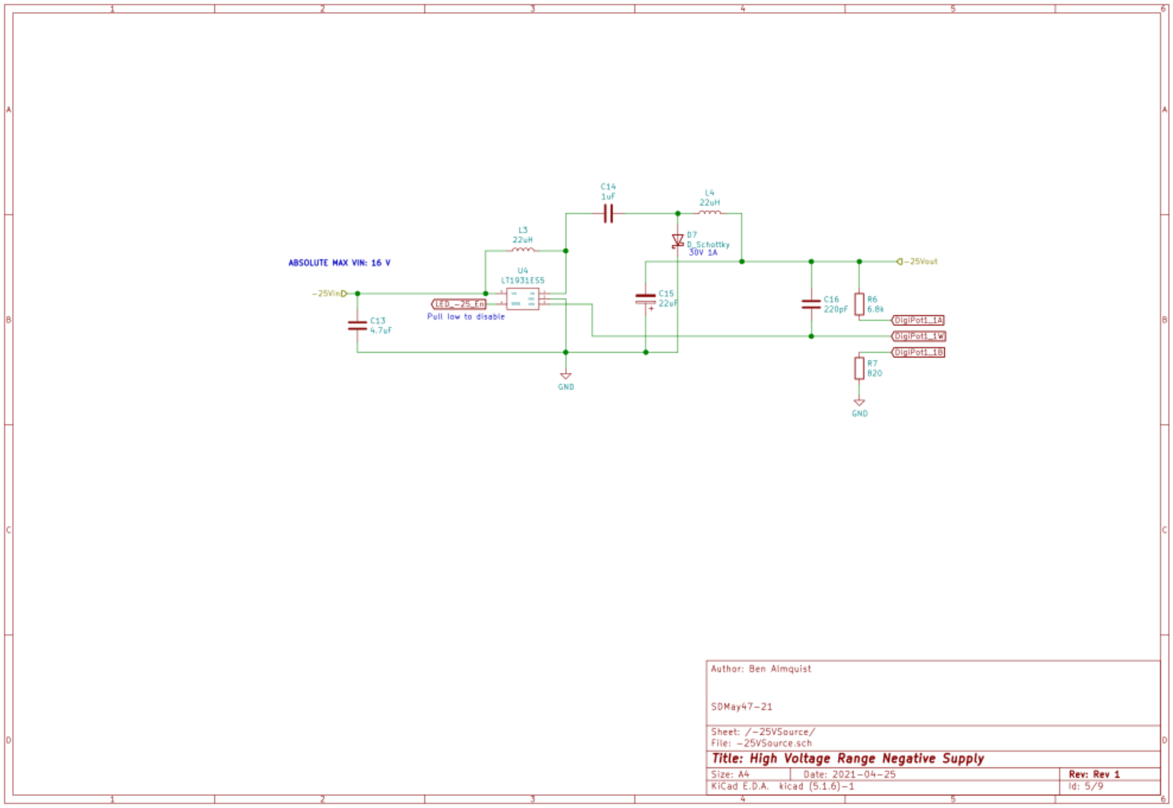
Step-Down Regulator



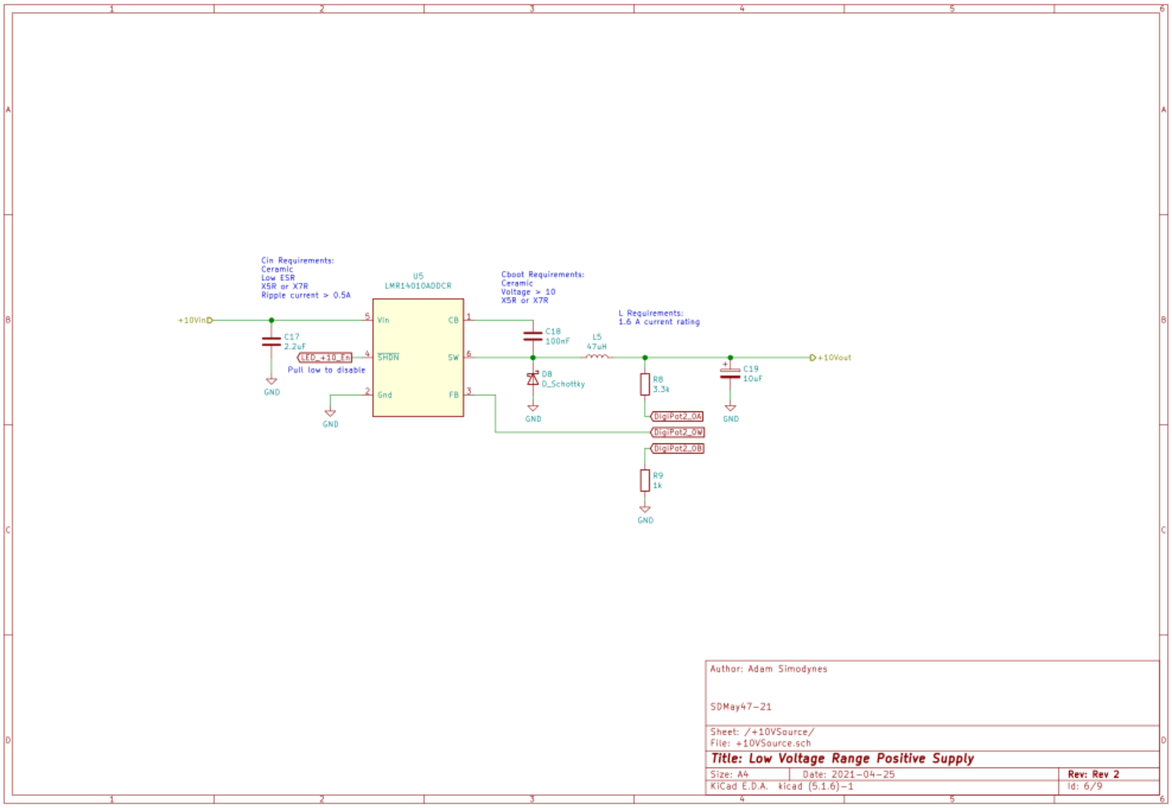
High Voltage Positive Supply (+2-25 Vdc)



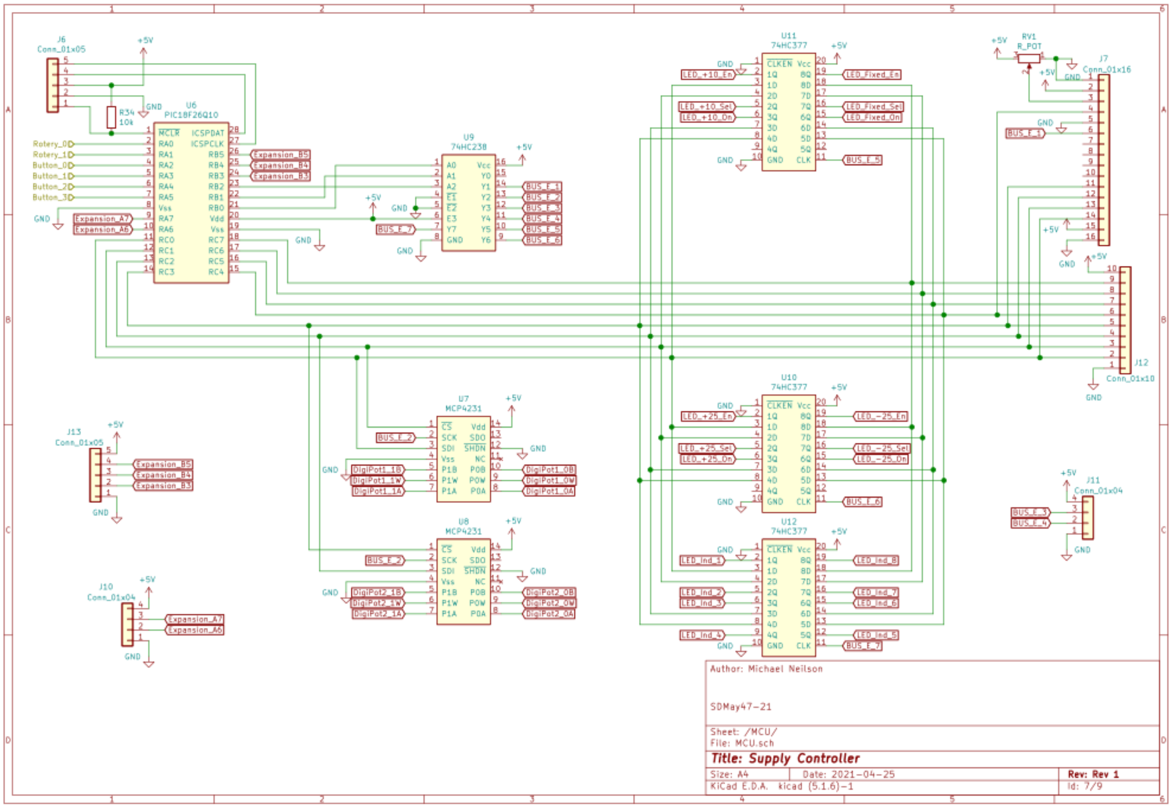
High Voltage Negative Supply (-2 - -25Vdc)



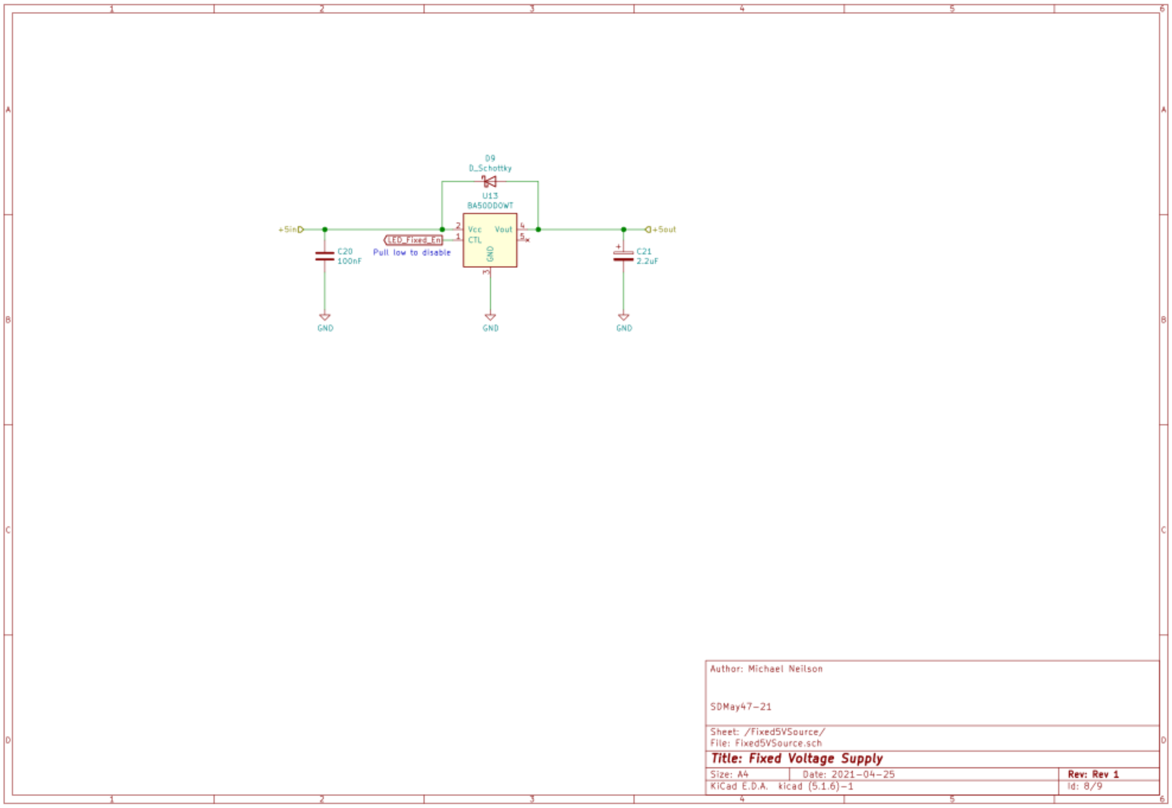
Low Voltage Positive Supply (+ 1-10 Vdc)



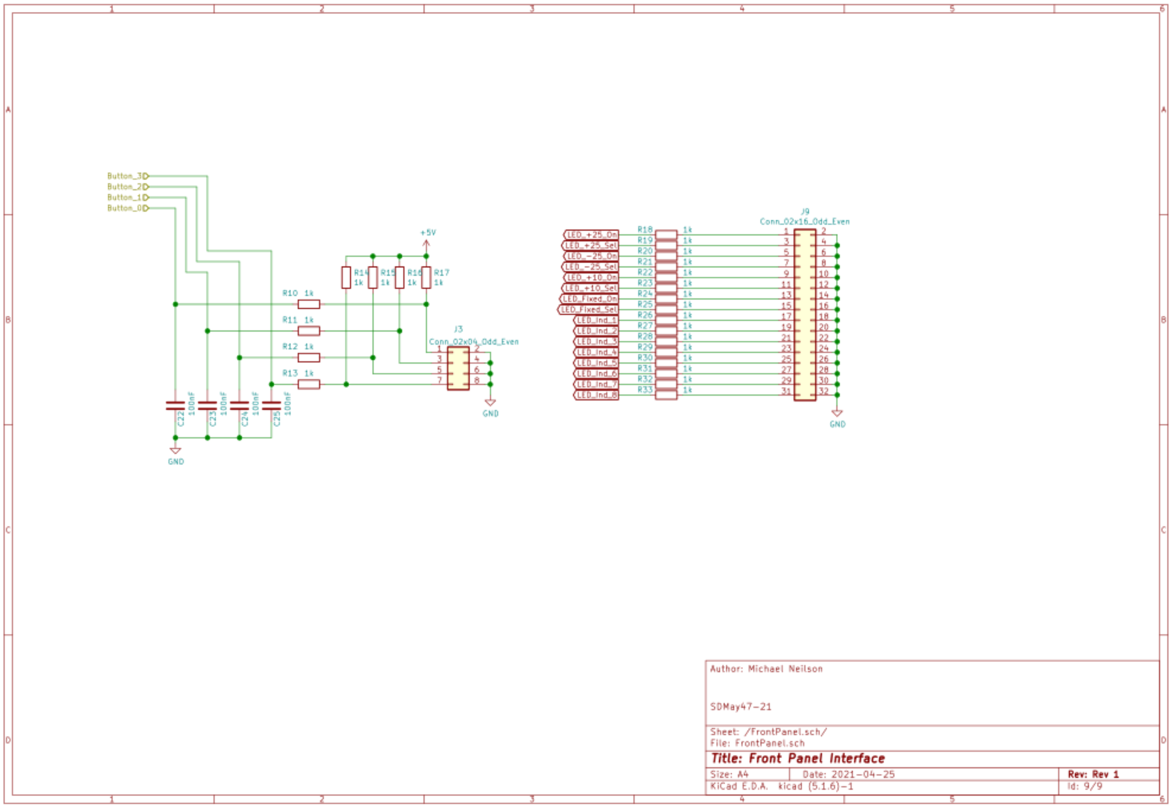
Microcontroller



Fixed Voltage Supply

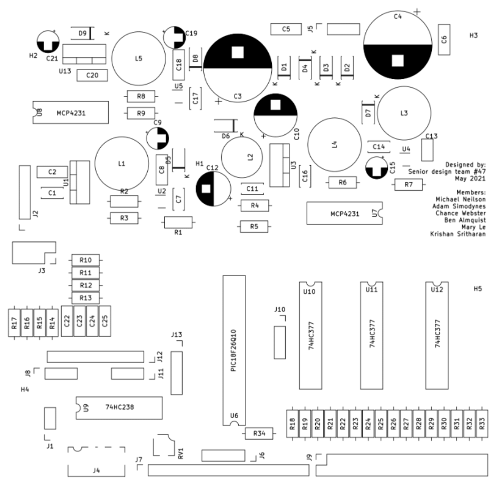


Front Panel Interface

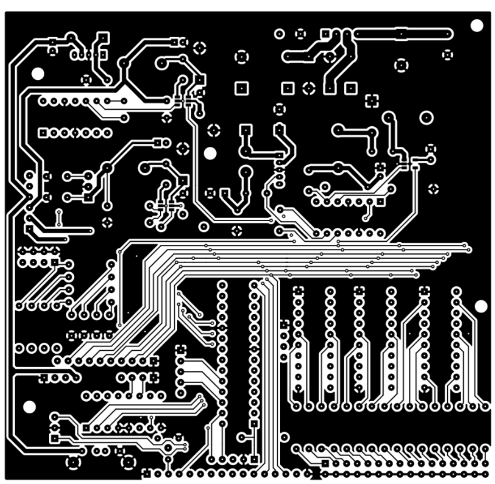


5.4.5 Appendix VI - PCB layout

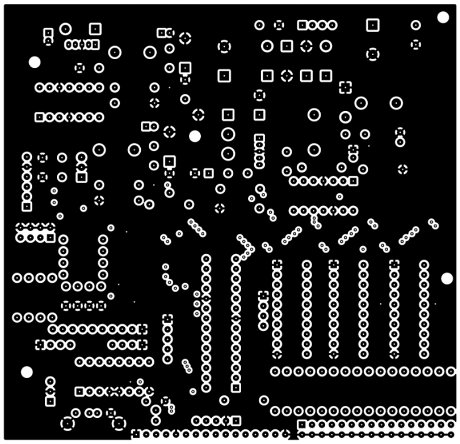
Front Silkscreen



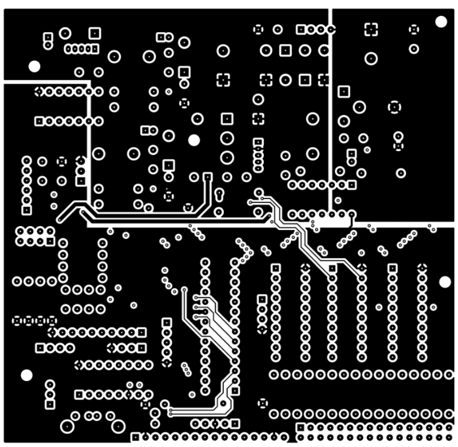
Front Copper



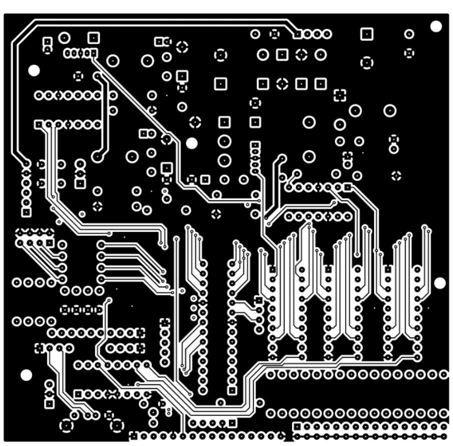
Inside 1 Copper



Inside 2 Copper



Back Copper



5.4.6 Appendix VI - Code

main.c

```c
/*
 * File:   main.c
 * Author: Michael
 *
 * Created on February 5, 2021
 */




#include <xc.h>
#include <stdint.h>
#include <stdlib.h>

#include "Configs.h"
#include "LCD.h"
#include "Interrupts.h"
#include "Rotary.h"
#include "DigiPot.h"
#include "Button.h"
#include "LED.h"
#include "PowerSupply.h"

void init();
void postInit();
//void intToString(char*,int,int);

void main(void) {
    init();
    postInit();
    int rot = 0;
    while(1){
        rot = Rotary_Change();
        if((Supply[sel] + rot >= 0) &&(Supply[sel] + rot <= 128)){
            Supply[sel] += rot;
            static char ints[4] = {2,3,0,1};
            SupplyDisplay(LCD_Buffer);
            LCD_Clear();
            LCD_Print_String(LCD_Buffer,40);
            DigiPot_set(Supply[sel],ints[sel]);
        }
        rot = 0;
        int x = Button_Rising_All();
        if(x & 0x01)
            Out_En();
        if(x & 0x02)
            Out_Sel();
        for(int j = 0; j < 3;j++)
            __delay_ms(100);
    }
    return;
}

void init(){
    TRISC |= 0x40;
    LCD_init(DISPLAY_2_LINES,DISPLAY_ON,INCREMENT_CURSOR);
    LCD_Clear();
    Rotary_init();
    DigiPot_init();
    Button_init();
    LED_init();
    powerSupply_init();
}

void postInit(){
    __delay_ms(10);
    Interrupt_init();
}
```

Button.c

```c
#include "Button.h"


void Button_init(){
    TRISA   |= BUTTON_PINS;
    ANSELA  &= ~BUTTON_PINS;

    prevButton = Button_Get_All();
}

char Button_Get_All(){
    return (PORTA & BUTTON_PINS) >> BUTTON_SHIFT;
}

char Button_Rising_All(){
    char k = Button_Get_All();
    char result = (k^prevButton)&k;
    prevButton = k;
    return result;
}
```

Button.h

```
/*
 * File:   Button.h
 * Author: Michael
 *
 * Created on February 5, 2021
 */

#ifndef BUTTON_H
#define BUTTON_H

#ifdef  __cplusplus
extern "C" {
#endif

    #include <xc.h>

    void Button_init();
    char Button_Get_All();
    char Button_Rising_All();

    char prevButton;

    #define BUTTON_PINS 0x3C
    #define BUTTON_TOTAL 4

    #define BUTTON_SHIFT 2

    #define Button_0 0x04
    #define Button_1 0x08
    #define Button_2 0x10
    #define Button_3 0x20



#ifdef  __cplusplus
}
#endif

#endif  /* BUTTON_H */
```

Configs.h

```
/*
 * File:   Configs.h
 * Author: Michael
 *
 * Created on February 11, 2021, 12:53 AM
 */

#ifndef CONFIGS_H
#define CONFIGS_H

#ifdef  __cplusplus
extern "C" {
#endif

#include <xc.h>


// PIC18F26Q10 Configuration Bit Settings
// 'C' source line config statements
// CONFIG1L
#pragma config FEXTOSC = ECH    // External Oscillator mode Selection bits (EC (external clock) above 8 MHz; PFM set to high power)
#pragma config RSTOSC = HFINTOSC_64MHZ// Power-up default value for COSC bits (HFINTOSC with HFFRQ = 64 MHz and CDIV = 1:1)

// CONFIG1H
#pragma config CLKOUTEN = OFF   // Clock Out Enable bit (CLKOUT function is disabled)
#pragma config CSWEN = ON       // Clock Switch Enable bit (Writing to NOSC and NDIV is allowed)
#pragma config FCMEN = ON       // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor enabled)

// CONFIG2L
#pragma config MCLRE = EXTMCLR  // Master Clear Enable bit (MCLR pin (RE3) is MCLR)
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (Power up timer disabled)
#pragma config LPBOREN = OFF    // Low-power BOR enable bit (Low power BOR is disabled)
#pragma config BOREN = SBORDIS  // Brown-out Reset Enable bits (Brown-out Reset enabled , SBOREN bit is ignored)

// CONFIG2H
#pragma config BORV = VBOR_190  // Brown Out Reset Voltage selection bits (Brown-out Reset Voltage (VBOR) set to 1.90V)
#pragma config ZCD = OFF        // ZCD Disable bit (ZCD disabled. ZCD can be enabled by setting the ZCDSEN bit of ZCDCON)
#pragma config PPS1WAY = ON     // PPSLOCK bit One-Way Set Enable bit (PPSLOCK bit can be cleared and set only once; PPS registers remain locked after one clear/set cycle)
#pragma config STVREN = ON      // Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)
#pragma config XINST = OFF      // Extended Instruction Set Enable bit (Extended Instruction Set and Indexed Addressing Mode disabled)

// CONFIG3L
#pragma config WDTCPS = WDTCPS_31// WDT Period Select bits (Divider ratio 1:65536; software control of WDTPS)
#pragma config WDTE = OFF       // WDT operating mode (WDT Disabled)

// CONFIG3H
#pragma config WDTCWS = WDTCWS_7// WDT Window Select bits (window always open (100%); software control; keyed access not required)
#pragma config WDTCCS = SC      // WDT input clock selector (Software Control)

// CONFIG4L
#pragma config WRT0 = OFF       // Write Protection Block 0 (Block 0 (000800-003FFFh) not write-protected)
#pragma config WRT1 = OFF       // Write Protection Block 1 (Block 1 (004000-007FFFh) not write-protected)
#pragma config WRT2 = OFF       // Write Protection Block 2 (Block 2 (008000-00BFFFh) not write-protected)
#pragma config WRT3 = OFF       // Write Protection Block 3 (Block 3 (00C000-00FFFFh) not write-protected)

// CONFIG4H
#pragma config WRTC = OFF       // Configuration Register Write Protection bit (Configuration registers (300000-30000Bh) not write-protected)
#pragma config WRTB = OFF       // Boot Block Write Protection bit (Boot Block (000000-0007FFh) not write-protected)
#pragma config WRTD = OFF       // Data EEPROM Write Protection bit (Data EEPROM not write-protected)
#pragma config SCANE = ON       // Scanner Enable bit (Scanner module is available for use, SCANMD bit can control the module)
#pragma config LVP = ON         // Low Voltage Programming Enable bit (Low voltage programming enabled. MCLR/VPP pin function is MCLR. MCLRE configuration bit is ignored)

// CONFIG5L
#pragma config CP = OFF         // UserNVM Program Memory Code Protection bit (UserNVM code protection disabled)
#pragma config CPD = OFF        // DataNVM Memory Code Protection bit (DataNVM code protection disabled)

// CONFIG5H


// CONFIG6L
#pragma config EBTR0 = OFF      // Table Read Protection Block 0 (Block 0 (000800-003FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR1 = OFF      // Table Read Protection Block 1 (Block 1 (004000-007FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR2 = OFF      // Table Read Protection Block 2 (Block 2 (008000-00BFFFh) not protected from table reads executed in other blocks)
#pragma config EBTR3 = OFF      // Table Read Protection Block 3 (Block 3 (00C000-00FFFFh) not protected from table reads executed in other blocks)

// CONFIG6H
#pragma config EBTRB = OFF      // Boot Block Table Read Protection bit (Boot Block (000000-0007FFh) not protected from table reads executed in other blocks)

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.



#ifdef  __cplusplus
}
#endif

#endif  /* CONFIGS_H */
```

DigiPot.c

```
#include "DigiPot.h"


    void DigiPot_Pulse_SCK(){
        LATB &= ~DIGIPOT_EN;
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        LATB &= ~DIGIPOT_EN;
        LATB |= (DIGIPOT_EN & DIGIPOT_ADDR);
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
    }
```

```
    void DigiPot_init(){

    }



    void DigiPot_set(short Val,char n){
        if(n > 5){
            return;
        }
        LATC |= 0x2A;
        LATC &= ~(0x01<<(((n>>1)<<1)+1));
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        NOP();
        short k= Val % DIGIPOT_TAPS;

        short command = 0x0000 | ((n%2) << 12);
        command |= (k & 0x0FFF);
        for(int i = 15;i >= 0;i--){
            char x = (((command>>i)&0x01)<<((n>>1)<<1));
            LATC &=  ~(0x01<<((n>>1)<<1));
            LATC |=  x;
            __delay_us(1);
            LATB &= ~DIGIPOT_EN;
            __delay_us(1);
            LATB |= (DIGIPOT_EN & DIGIPOT_ADDR);
            __delay_us(1);
        }
```

```



        NOP();
        NOP();
        NOP();
        NOP();

        LATB &= ~DIGIPOT_EN;
        __delay_us(1);
        LATC &= ~0x2A;
    }
```

DigiPot.h

```
/*
 * File:   DigiPot.h
 * Author: Michael
 *
 * Created on February 5, 2021
 */

#ifndef DIGIPOT_H
#define DIGIPOT_H

#ifdef  __cplusplus
extern "C" {
#endif

#include <xc.h>
#include "LCD.h"

void DigiPot_init();
void DigiPot_set(short,char);
//void intToHexString(char* buffer, int i,int p);



#define DIGIPOT_EN                      0x07
#define DIGIPOT_ADDR                    0x02

#define DIGIPOT_TAPS                    129


#ifdef  __cplusplus
}
#endif

#endif  /* DIGIPOT_H */
```

Interrupts.c

```c
#include "Interrupts.h"



void __interrupt(high_priority) HP_ISR(){

}


void __interrupt(low_priority) LP_ISR(){
   //interrupt-on-change handler
    if(PIR0 & 0x10){
        //rotary encoder state change
        if(IOCAF & 0x03){
            IOCAF &= ~0x03;
            Rotary_Update();
        }


        PIR0 &= ~0x10;
    }


}




void Interrupt_init(){

    INTCON |= 0xE0;
    PIE0 |= 0x10;
    IPR0 &= ~0x10;


    //rotary Interrupts
    IOCAP   |= 0x03;
    IOCAN   |= 0x03;

}
```

Interrupts.h

```c
/*
 * File:   Interrupts.h
 * Author: Michael
 *
 * Created on February 5, 2021
 */

#ifndef INTERRUPTS_H
#define INTERRUPTS_H

#ifdef  __cplusplus
extern "C" {
#endif

#include <xc.h>
#include "Rotary.h"


//high priority interrupt handler
void __interrupt(high_priority) HP_ISR();


//low priority interrupt handler
void __interrupt(low_priority) LP_ISR();



void Interrupt_init();

#ifdef  __cplusplus
}
#endif

#endif  /* INTERRUPTS_H */
```

LCD.c

```
#include "LCD.h"


void LCD_Pulse_Enable(){
    LATB &= ~LCD_EN;
    LATB |= (LCD_EN & LCD_ADDR);
    NOP();
    LATB &= ~LCD_EN;
}

void LCD_init(char config1,char config2,char config3){
    for(int i = 0;i < LCD_MAX_LINES;i++){
        for(int j = 0;j < LCD_MAX_WIDTH;j++){
            LCD_Buffer[i][j] = ' ';
        }
    }

    TRISB &= 0xF8;
    TRISC &= 0xC0;
    NOP();
    LATC &= 0xF0;
    __delay_ms(15);
    LATC |= 0x02;
    LCD_Pulse_Enable();
    __delay_ms(3);

    LCD_Command(FUNCTION_SET|(config1 & 0x1F));
    LCD_Command(DISPLAY_CONTROL|(config2 & 0x07));
    LCD_Command(ENTRY_MODE|(config3 & 0x03));

}
```

```
void LCD_Command(uint8_t cmd){
    LATC &= 0xC0;
    LATC |= cmd >> 4;
    LATC &= ~LCD_RS;
    __delay_us(1);
    LATB |= (LCD_EN & LCD_ADDR);
    NOP();
    LATB &= ~LCD_EN;
    __delay_ms(1);
    LATC |= (0x0F & cmd);
    LCD_Pulse_Enable();
    __delay_ms(3);
}

void LCD_Print_Char(char c){

    LATC &= 0xF0;
    NOP();
    LATC |=  (c & 0xF0) >> 4;
    __delay_us(1);
    /*Send higher nibble of data first to PORT*/
    LATC |= LCD_RS;
    NOP();
    LCD_Pulse_Enable();
    __delay_us(1);
    LATC &= 0xF0;
    NOP();
    LATC |=  (0x0F & c);
    __delay_us(1);
    LCD_Pulse_Enable();
    __delay_us(1);

}

void LCD_Print_String(char S[],uint8_t len){
    for(int i = 0; i < len;i++){
        LCD_Print_Char(S[i]);
        __delay_us(50);
    }
}
```

```
void LCD_Clear(){
    LCD_Command(CLEAR_DISPLAY);
    __delay_ms(2);
}

LCD_Update_W_Cursor(){

}

LCD_Update(){

}
```

LCD.h

```c
/*
 * File:   LCD.h
 * Author: Michael
 *
 * Created on February 5, 2021
 */

#ifndef LCD_H
#define LCD_H

#ifdef  __cplusplus
extern "C" {
#endif

#include <xc.h>
#include <stdint.h>
#define _XTAL_FREQ 64000000

#define LCD_LINES 2
#define LCD_WIDTH 16

#define LCD_MAX_LINES 2
#define LCD_MAX_WIDTH 40

char LCD_Buffer[LCD_MAX_LINES][LCD_MAX_WIDTH];

//Start up the LCD
//
//parameters
//char config1 -> use macros under FUNCTION_SET
//char config2 -> use macros under DISPLAY_CONTROL
//char config3 -> use macros under ENTRY_MODE
//
void LCD_init(char,char,char);

//Send a command to the LCD
//
//parameters
//uint8_t com -> macros are listed below
//
void LCD_Command(uint8_t);

//Send a character to the LCD
//
//parameters
//char c -> character to be sent to the LCD
//
void LCD_Print_Char(char);

//Sends multiple characters to the LCD
//
//parameters
//char* S -> pointer of a string of characters to be sent to the LCD
//uint8_t len-> length of the string of character to be sent to the LCD
//
void LCD_Print_String(char*,uint8_t);

//Clears the LCD and sets the cursor to position 0
//
void LCD_Clear();

//sets the cure to a specified position [WIP]
//
//parameters
//char x -> x coordinate of the new position
//char y -> y coordinate of the new position
//
void LCD_Move_Cursor(char,char);


#define CLEAR_DISPLAY           0x01

#define RETURN_HOME             0x02

#define ENTRY_MODE              0x04
#define INCREMENT_CURSOR        0x02
#define SHIFT_WITH_CHAR         0x01

#define DISPLAY_CONTROL         0x08
#define DISPLAY_ON              0x04
#define CURSOR_ON               0x02
#define CURSOR_BLINK            0x01

#define CURSOR_DISPLAY_SHIFT    0x10
#define DISPLAY_SHIFT           0x08
#define LEFT_TO_RIGHT           0x04

#define FUNCTION_SET            0x20
#define DATA_LINE_8BIT          0x10
#define DISPLAY_2_LINES         0x08
#define DOTS_5X10               0x04

#define SET_DDRAM               0x40

#define LCD_RS                  0x10

#define LCD_EN                  0x07
#define LCD_ADDR                0x01




#ifdef  __cplusplus
}
#endif

#endif  /* LCD_H */
```

LED.c

```
#include "LED.h"

void Reg_CLK(char addr){
    LATB |= addr & ADDR_LINES;
    NOP();
    LATB &= ADDR_LINES;
    NOP();
}

void LED_init(){
    sel = P25;
    Reg[0] = 0x00;
    Reg[1] = 0x03;
    Reg[2] = 0x00;

    Reg_Set(Reg[0],REG_ADDR);
    Reg_Set(Reg[1],REG_ADDR+1);
    Reg_Set(Reg[2],LED_ADDR);

}

void LED_Opp(char O, char A,char X,char addr){
    char k = Reg[addr - ADDR_LINES];
    k &= A;
    k |= O;
    k ^= X;
    Reg[addr - ADDR_LINES] = k;
    Reg_Set(k,addr);

}

void Out_En(){
    char addr = REG_ADDR + (sel/2);
    if(sel % 2){
        LED_Opp(0x00,~0x00,LED_N25_ON | LED_N25_EN,addr);
        return;
    }
    LED_Opp(0x00,~0x00,LED_P25_ON | LED_P25_EN,addr);
}
```

```
void Out_Sel(){
    LED_Opp(0x00,~(LED_P25_SEL|LED_N25_SEL),0x00, REG_ADDR + (sel/2));
    sel++;
    sel%=4;
    char addr = REG_ADDR + (sel/2);
    if(sel%2){
        LED_Opp(LED_P25_SEL,~0x00,0x00,addr);
        return;
    }
    LED_Opp(LED_N25_SEL,~0x00,0x00,addr);

}

void Reg_Set(char r,char addr){
    LATC &= 0x00;
    LATC |= r;
    Reg_CLK(addr);
    LATC = 0;
}
```

LED.h

```
/*
 * File:   LED.h
 * Author: Michael
 *
 * Created on February 5, 2021
 */

#ifndef LED_H
#define LED_H

#ifdef  __cplusplus
extern "C" {
#endif

    #include <xc.h>


    void LED_init();
    void LED_Set(char,char);
    void LED_Opp(char,char,char,char);

    void Out_En();
    void Out_Sel();

    void Reg_Set(char,char);

    #define LED_ADDR 7

    char Reg[3];
    char sel;


    #define LED_P10_EN      0x01
    #define LED_P10_SEL     0x02
    #define LED_P10_ON      0x04

    #define LED_FIXED_EN    0x80
    #define LED_FIXED_SEL   0x40
    #define LED_FIXED_ON    0x20

    #define LED_P25_EN      0x01
    #define LED_P25_SEL     0x02
    #define LED_P25_ON      0x04

    #define LED_N25_EN      0x80
    #define LED_N25_SEL     0x40
    #define LED_N25_ON      0x20

    #define P10     0
    #define FIXED   1
    #define P25     2
    #define N25     3

    #define REG_ADDR 5
    #define ADDR_LINES 0x07


#ifdef  __cplusplus
}
#endif

#endif  /* LED_H */
```

PowerSupply.c

```
#include "PowerSupply.h"

void powerSupply_init(){
    Supply[0] = 0;
    Supply[1] = 0;
    Supply[2] = 0;
}


void SupplyDisplay(char* ch){
    if(sel == 1){
        if(Reg[0]& LED_FIXED_ON){
            sprintf(ch,"+5V: ON");
        }else{
            sprintf(ch,"+5V:OFF");
        }
        return;
    }
    float k;
    if(sel == 0){
        sprintf(ch,"+10V: ");
        k = 0.7*(10010/((Supply[0]*(625/8))+1000));
    }else if(sel == 2){
        sprintf(ch,"+25V: ");
        k = 1.23*(17620/((Supply[1]*(625/8))+820));
    }else{
        sprintf(ch,"-25V:-");
        k = 1.255*(17620/((Supply[2]*(625/8))+820));
    }
    sprintf(ch+6,"%6.2f",k);
}
```

PowerSupply.h

```
/*
 * File:   PowerSupply.h
 * Author: Michael
 *
 * Created on February 5, 2021
 */

#ifndef POWERSUPPLY_H
#define POWERSUPPLY_H

#ifdef  __cplusplus
extern "C" {
#endif

    #include <xc.h>
    #include <stdio.h>
    #include "LED.h"
    #include "LCD.h"

    void powerSupply_init();

    short Supply[3];

    void SupplyDisplay(char*);



#ifdef  __cplusplus
}
#endif

#endif  /* POWERSUPPLY_H */
```

Rotary.c

```
#include "Rotary.h"

void Rotary_init(){

    TRISA   |= 0x03;
    ANSELA  &= ~0x03;
    WPUA    |= 0x03;

    TotalRotation = 0;
    prevRState = PORTA & 0x03;

}


int Rotary_Change(){
    int result = 0;
    if (TotalRotation >= NPerCycle) {
        result = TotalRotation/NPerCycle;
        TotalRotation %= NPerCycle;
    }
    else if (TotalRotation < 0) {

        result = ((TotalRotation + 1) / NPerCycle)-1;
        TotalRotation -= result * NPerCycle;
    }
    return result;
}

void Rotary_Update(){

    int newState = PORTA & 0x03;

    if(newState == 0 && prevRState == 2){
        TotalRotation++;
    }
    if(newState == 2 && prevRState == 0){
        TotalRotation--;
    }
    prevRState = newState;
}
```

Rotary.h

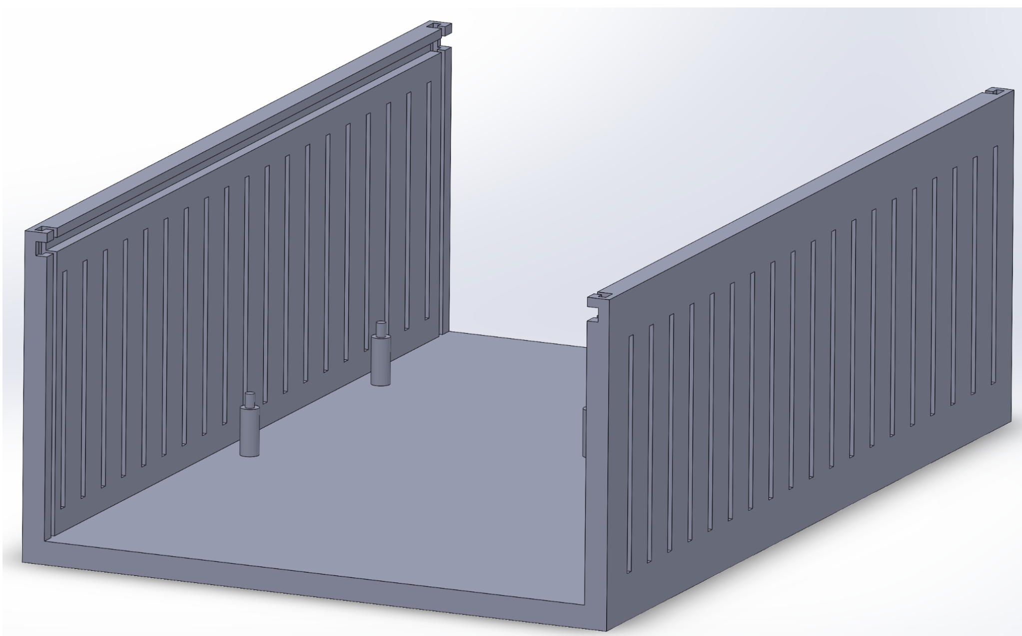
```
/*
 * File:   Rotary.h
 * Author: Michael
 *
 * Created on February 5, 2021
 */

#ifndef ROTARY_H
#define ROTARY_H

#ifdef  __cplusplus
extern "C" {
#endif

#include <xc.h>
#include "Rotary.h"

void Rotary_init();
int Rotary_Change();
void Rotary_Update();


volatile int TotalRotation;
volatile char prevRState;

#define NPerCycle           24





#ifdef  __cplusplus
}
#endif

#endif  /* ROTARY_H */
```

5.4.6 Appendix VII - Enclosure

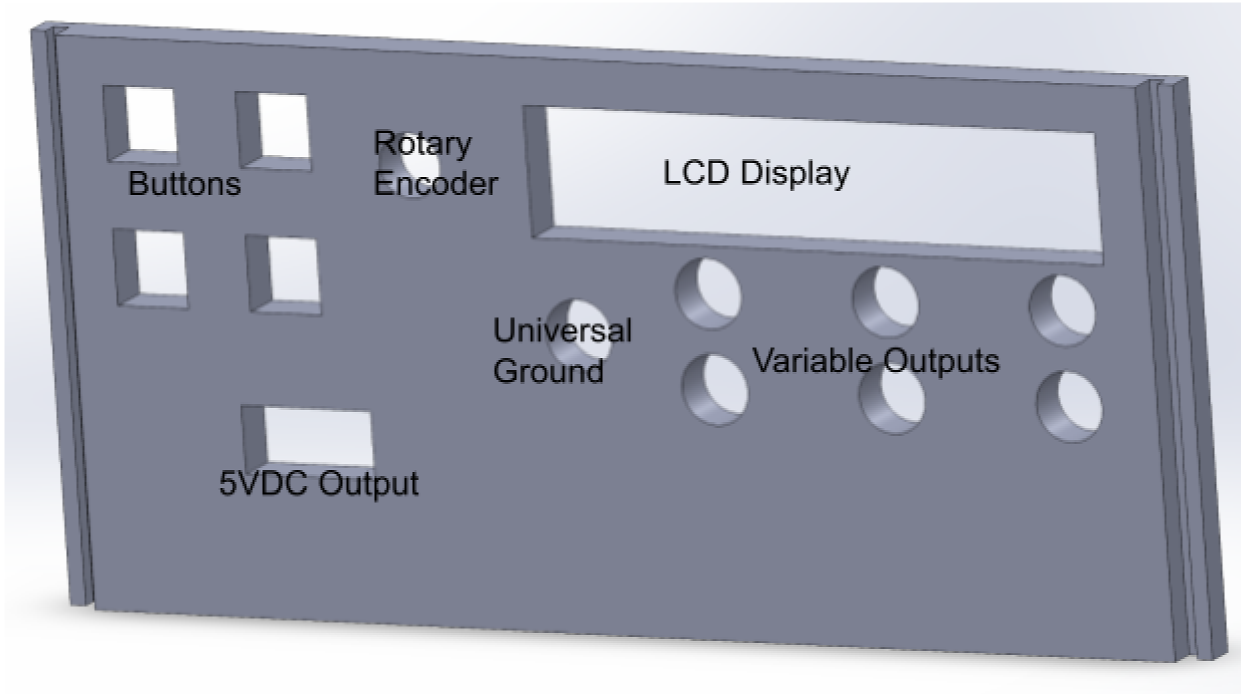
The enclosure was designed to be 3D printed for efficiency and to allow the IO to be organized. The enclosure was designed in five parts to allow the user easy access to build the product. The first part is the base which has two of the walls attached to it, which has cuts into it to allow airflow for cooling. The base also has five pegs where the PCB will be installed. Lastly, the base has slots for the front, back, and top panels. The next part of the enclosure is the front panel. This part has slots for the IO components to be inserted and also has slits to be inserted into the base.

The next piece of the enclosure is the back panel, which is attached to the base using the slits on the sides of the part. The back also has a cut for the power cable to attach to the transformer. The top part is simply used to close off the inside of the power supply. It slides into the slots on the top of the base and is held in place by two locks. The locks are designed to be inserted into the slits on the sides to keep the top from moving forward or backward.

Base of the Enclosure

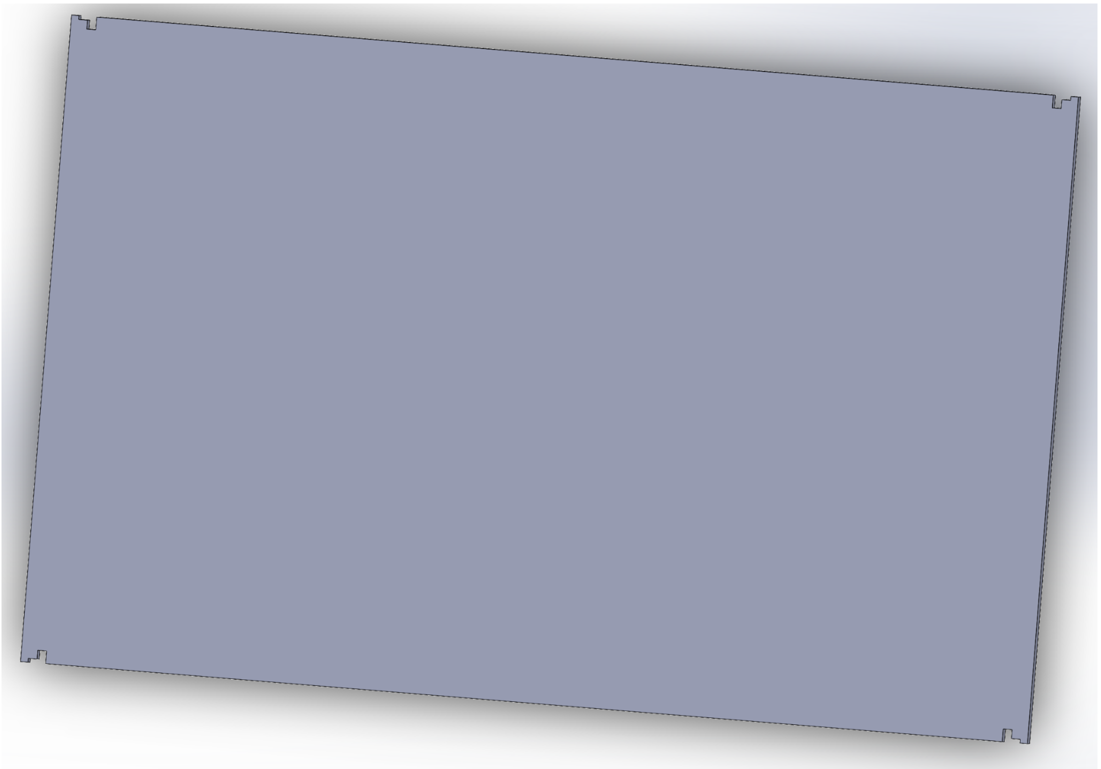


Front of the Enclosure

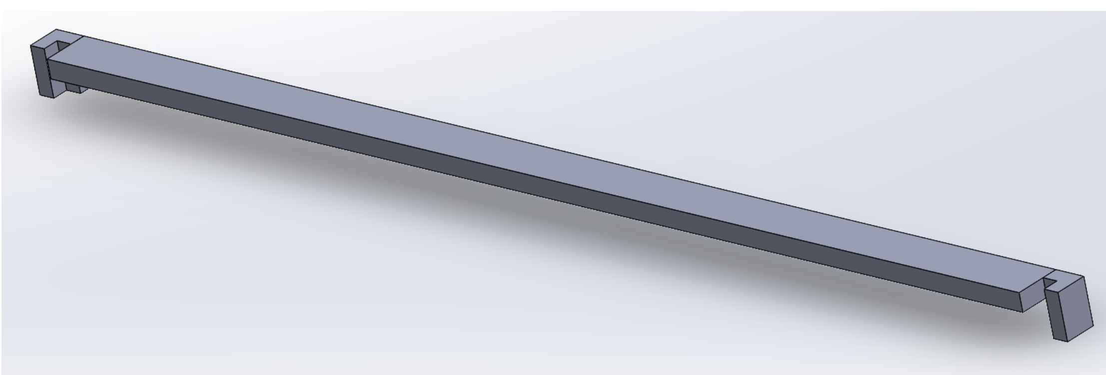


Back of the Enclosure

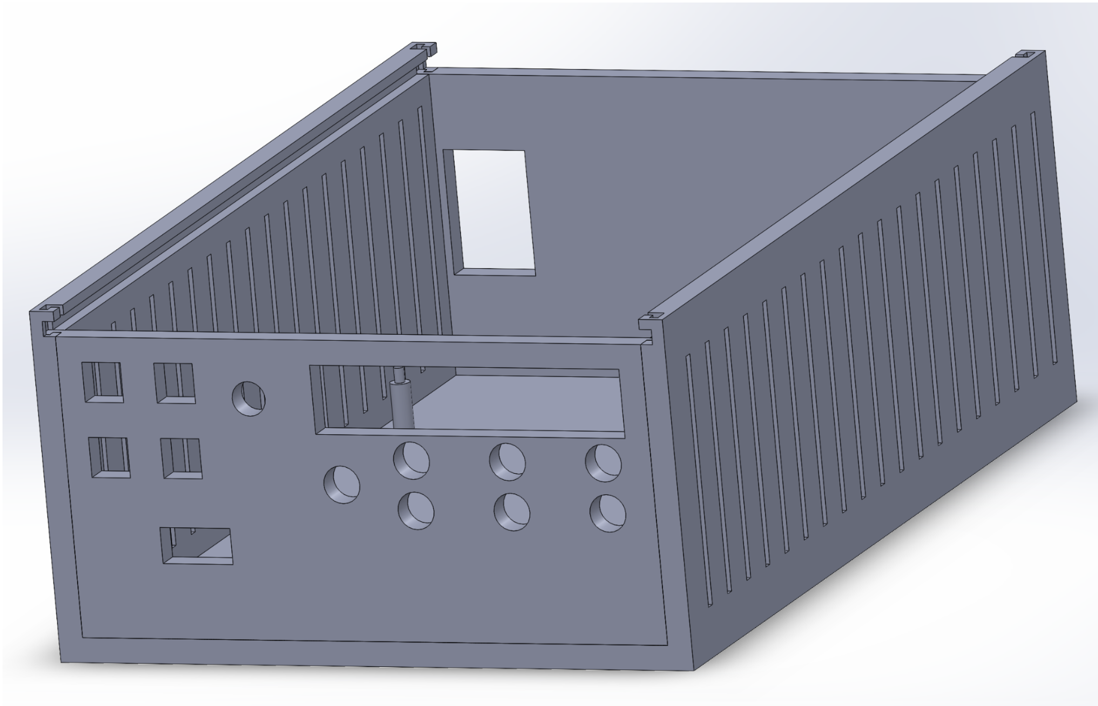
Top of the Enclosure



Lock for the Enclosure



Enclosure Model without Top



Complete Enclosure Model