DEVELOPMENT OF A LOW-COST HYBRID MUSIC SYNTHESIZER

By

Spencer R. Drewry

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

Submitted June, 2021

Supervised by Art MacCarley and Vladimir Prodanov

TABLE OF CONTENTS

SECTIONS ***Page***

APPENDICES ***Page***

LIST OF TABLES AND FIGURES

TABLES ***Page***

FIGURES ***Page***

ABSTRACT

Until recently, affordable music equipment has always been seen as “budget”, providing a poor user experience. Inexpensive equipment was plagued with audible noise, signal integrity issues, and convoluted user interfaces. Companies like *Teenage Engineering* have proven that this does not have to be the case, in 2019 introducing their "Pocket Operator” series for $89. Due to the modern availability of low cost, high quality, consumer off the shelf [COTS] analog and digital components as well as creative engineering, the quality of inexpensive audio equipment has increased significantly. Despite these industry advances, the market is relatively small and shows a great potential for growth.

This senior project capitalizes on this market possibility, providing a low-cost analog/ digital hybrid synthesizer architecture without the aforementioned caveats of poor signal integrity, user interface and sound quality. The synthesizer provides a low latency, simple to use, visual interface to the user. This visual interface allows intuitive and simple-to-learn access to the synthesizer’s parameters. The value of these parameters can also be loaded or saved from non-volatile memory. The power will be provided locally by a battery. Therefore, the synthesizer’s power draw will be low enough to ensure a significant on-time. Physically, the synthesizer provides industry standard audio connectivity to be interfaced with the end user’s existing equipment.

SECTION 1 - INTRODUCTION

I am fascinated with the intersection of technology, engineering, and art. My interest in taking apart stereos, keyboards, and other devices grew by the time I was in high school. I actually had a pretty successful stereo and music equipment repair business going fixing all of my friends' secondhand gear. I love how electronic instrument companies have spent so much time engineering devices to help musicians make sounds! They have generated decades of history, building on previous innovations all in the pursuit of more effective tools for artists. I decided last year that I wanted to complete a senior project that delves into some of these interests.

The first idea I had was to build my senior project around the goal of creating a synthesizer. The synthesizer has a rich history filled with challenges, creativity and innovation. Being one of the oldest electronic musical instruments, the first few models we would recognize today emerged in the 1950s [1]. Early synthesizers were fully analog systems, essentially retrofitted analog computers converted to sound making devices. This architecture was riddled with pitfalls, as analog systems are inherently sensitive to environmental conditions. It was not uncommon for synthesizers to need to be tuned every time they were turned on, much like acoustic instruments [2].

This era of analog synthesis was bound to end, as in the 1980s digital electronics were entering the consumer musical electronics field with force. In 1983 the MIDI standard for musical devices to interface with each other over a serial bus was released [3]. This drove the digitalization of synthesizers even further. Once the floodgates were open, more and more digital

circuitry made its way into synthesizers. Instead of tuned LC oscillators, digital-to-analog converters [DAC's] served up arbitrary waveforms from memory (at perfect pitch thanks to quartz clocks) [4]. These methods of digital synthesis were originally very 'lean', as the computational power of microprocessors and the size of memory was still small and very expensive. This initial limitation actually drove engineers to develop efficient methods of reproducing sound. These methods, wavetable and FM synthesis for example live on today [5].

The explosion of consumer electronics has enabled almost every device to be designed, manufactured, and sold for cheaper than ever before. Recently this has led to a generation of capable, low-cost synthesizers to meet the market. For example, the 'Pocket Operator' series from teenage engineering are capable synthesizers that cost merely $89.00 [6]. This senior project follows this trend, combining a legacy of digital and analog synthesis circuitry into a portable, low-cost product. Combining some of the benefits of a digital system with the historically beloved sound of analog filter and compression architectures [7], [8]. Using a modern, low cost, microprocessor enables inexpensive devices with plenty of features, low latency and low cost [9], [10].

SECTION 2 - PROJECT MOTIVATION

2.1 CUSTOMER NEEDS

Determining customer needs is an essential step in the initial design of any system or project. Engineering specifications that do not satisfy a customer's needs could be the end of a product. Interestingly, engineering functionality a customer never needed could derail a project just as quickly. Thus, it is paramount to accurately determine the needs of a customer early on.

For some engineering problems it can be simple to determine what the system requirements and specifications are. For example, a drop-in component replacement has well defined requirements. The project unfortunately is much more open ended, leading to needing much more rigorous design specifications. Being a consumer device, many of the specifications are driven by cost rather than functionality. In the design process especially, I was forced to weigh the customer value of a feature versus its implementation cost. For example, the component cost required to improve the synth from a paraphonic architecture to a fully polyphonic architecture was vastly larger than the value it adds to the final project.

I found two main areas of focus while establishing customer needs. The first being the final cost of the product. Small, low-cost synthesizers are a growing market in the music equipment area. I decided it would be competitive for the product to be priced similarly as other budget synthesizers on the market. This is where I determined the $100.00 cost of manufacturing limit. This ensures that the synthesizer falls within the sub $150.00 consumer cost range roughly equivalent devices sell at.

The second aspect I focused on was the final design's compliance with established interface and other standards. As musicians often interconnect many brands and types of musical equipment, customers want the ability to easily connect devices with no issues. To insure this, I set up engineering specifications to define the levels and connectivity of all inputs and outputs.

2.2 REQUIREMENTS AND SPECIFICATIONS

TABLE I
Low Cost Hybrid Musical Synthesizer Requirements and Specifications

Marketing Requirements	Engineering Specifications	Justification
1	Total component and assembly cost <100$ with low quantity production(<100 units)	Maintains competitive pricing in consumer synthesizer market
2	All synthesizer parameters are accessible in less than two menu levels	Ensures simple/fast user interface without 'menu diving'
3	Size of device less than 3x4in by 0.5in thick	Ensures portability
3	No non ROHS components exposed to the user	Ensures user safety with device on their person
4	Device average current draw from 3V source less than 150mA	Ensures 12+ hours use out of 2 standard 2000 mAh AA size cells
5	Female ¼ in TRS line jack with audio "line" level of +4 dBu	Glenn M. Ballou, ed. (1998). Handbook for Sound Engineers: The New Audio Cyclopedia, Second Edition. Focal Press. p. 761. ISBN 0-240-80331-0
5	Female ⅛ in TRS with audio "headphone" power of 10-20mW into a 32 ohm load	Glenn M. Ballou, ed. (1998). Handbook for Sound Engineers: The New Audio Cyclopedia, Second Edition. Focal Press. p. 761. ISBN 0-240-80331-0
5	MIDI note to audio response latency less than 100ms	J. Deber, R. Jota, C. Forlines, and D. Wigdor, "How Much Faster is Fast Enough?," *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*, 2015.
6	User input to interface response and input to audio latency less than 100ms	J. Deber, R. Jota, C. Forlines, and D. Wigdor, "How Much Faster is Fast Enough?," *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*, 2015.
7	Noise at audio line out at least 60dB less than "note on" output amplitude	"Signal to Noise Ratio (SNR) - How it can destroy your recording," *Magroove Blog*, 02-Nov-2019. [Online]. Available: https://magroove.com/blog/en-us/snr/. [Accessed: 17-Nov-2020].

Marketing Requirements

1. Low final cost
2. Easy to navigate user interface
3. Small, portable, on person device
4. Long battery life
5. Standard audio connectivity
6. Responsive interface
7. Low Noise

SECTION 3 - INITIAL DESIGN

3.1 LEVEL ZERO FUNCTIONAL DECOMPOSITION

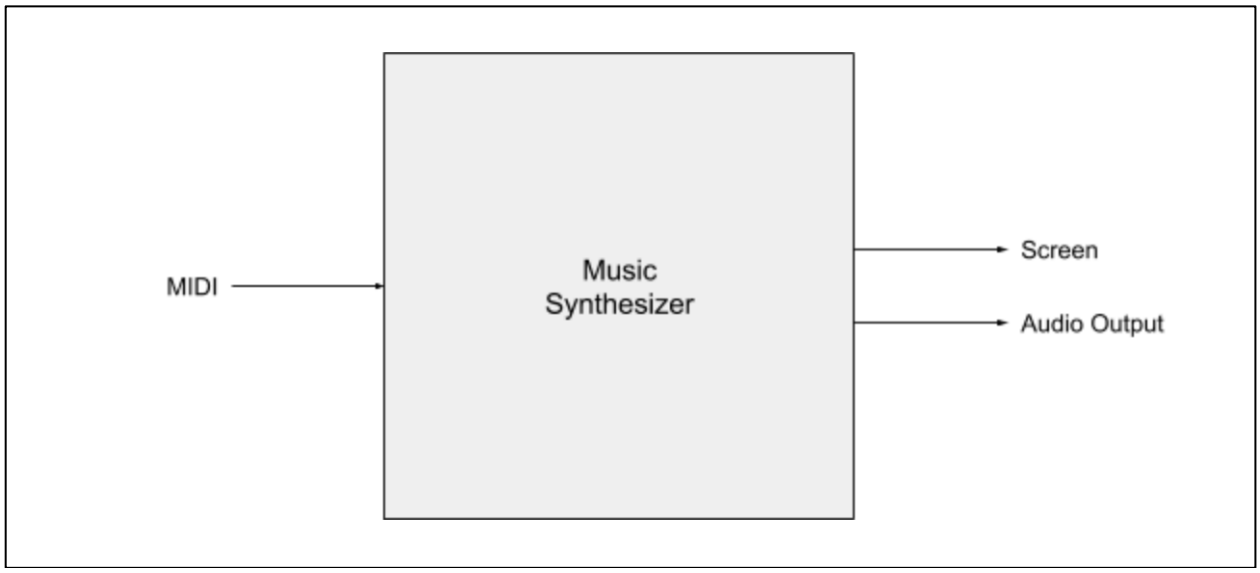


FIGURE 1: LEVEL 0 BLOCK DIAGRAM

The level zero decomposition is almost a direct reflection of the requirements and specifications. The project is one module, and therefore all inputs and outputs described in table 1 must be present. The user inputs including the knobs, buttons, and keyboard are all inputs. The input signals including the battery voltage and MIDI data connection are also inputs. The outputs follow the same convention, mirroring the specifications for audio output and screen.

TABLE II
Music Synthesizer Module Breakdown

Module	**Music Synthesizer**
Inputs	MIDI: ⅛ in trs standard MIDI input
Outputs	Screen: OLED matrix display with onboard display controller, SPI/I2C Audio Jack: ⅛ in and ¼ in trs and Line level or headphone level output 1.41Vpp
Functionality	Decode user input to synthesizer parameters, which can be saved locally. These parameters are displayed on the OLED screen. When a key is pressed or a MIDI note received, an appropriate audio tone is generated on the output.

3.2 LEVEL ONE FUNCTIONAL DECOMPOSITION

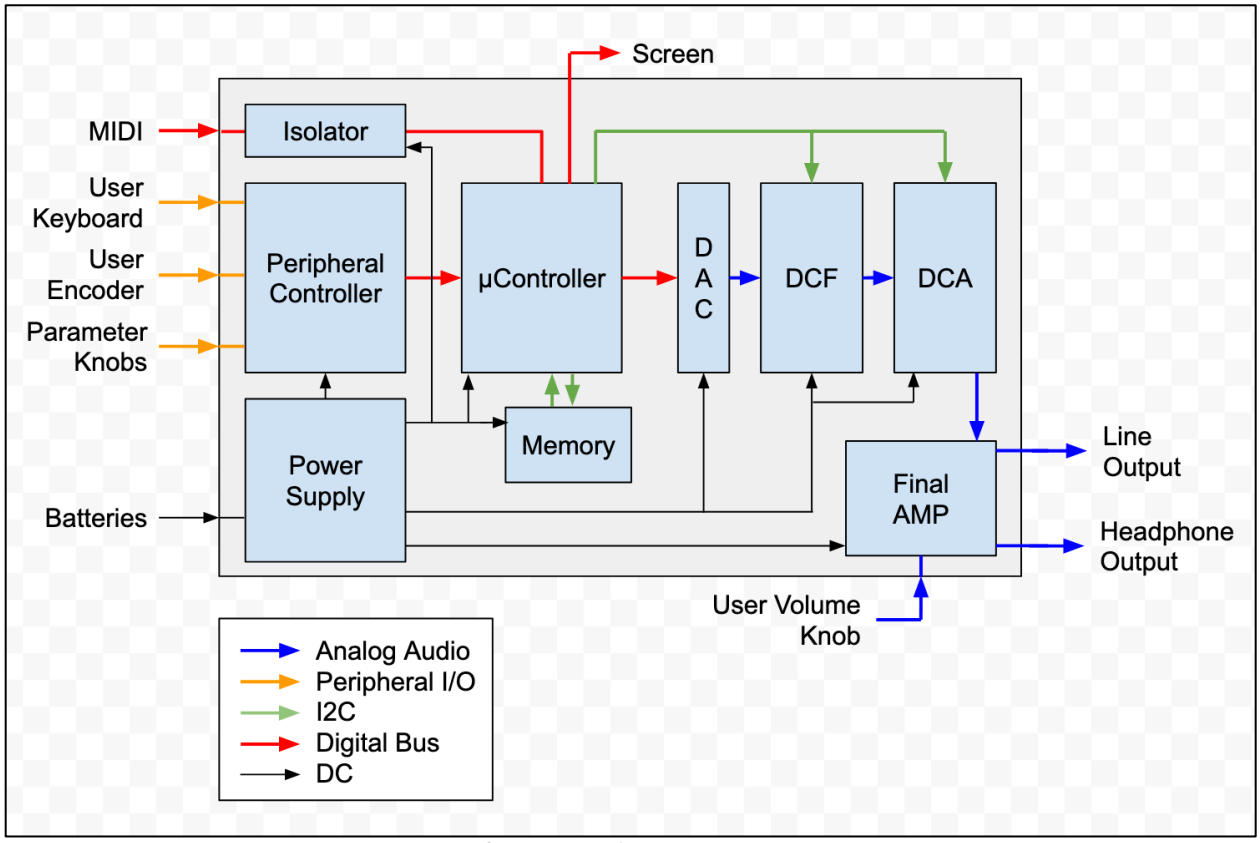


FIGURE 2: LEVEL 1 BLOCK DIAGRAM

In figure two, the level zero functional diagram is broken down further into a level one decomposition. This decomposition begins to outline the methods intended to accomplish the various function within the device. Tables 3a, 3c, 3d, and 3i outline various standard functions of a handheld device. Namely the power supply, input isolation, and peripheral controls. These functional blocks handle the functions the main synthesis engine needs to interreact with the interfaces and be powered.

The main synthesis architecture is outlined by tables 3c, 3f, 3g, and 3h. The signal chain is as follows. A micro-controller interprets user input as well as MIDI data to determine the frequency of notes required. A wavetable defined waveform is calculated for each note and presented on the DAC's digital bus, then clocked out as an analog waveform at the output of the

DAC. This portion follows a relatively standard wavetable synthesis architecture [5]. The output of the DAC is filtered by a digitally controlled analog filter [DCF] [7]. The signal then is attenuated by the digitally controlled amplifier and finally output to the amplifier to be amplified to appropriate line and headphone levels (specified in table III-i).

TABLE III-a
POWER SUPPLY MODULE BREAKDOWN

Module	**Power Supply**
Inputs	Batteries: 2X AA size rechargeable cells
Outputs	DC: 3.3V for digital electronics, +8V and +4Vref for analog circuits
Functionality	The Power Supply regulates the battery voltage to digital and analog power rails for the other electronics.

TABLE III-b
MICRO-CONTROLLER MODULE BREAKDOWN

Module	**µController**
Inputs	Digital Bus(peripheral values): Peripheral Values reported on 8bit digital bus MIDI (isolated): MIDI data, isolated and 3V3 level shifted Memory: I2C bus for non-volatile memory DC: 3V3
Outputs	Screen: SPI Peripheral Bus 3V3 Logic DAC: 8bit internal DAC Memory: I2C bus for non-volatile memory DCF, DCA: I2C bus to control filter and amplifier parameters
Functionality	The µController manages all of the user inputs, screen outputs, generating wavetable tones through the DAC, and controlling the DCF and DCA states.

TABLE III-c
ISOLATOR MODULE BREAKDOWN

Module	**Isolator**
Inputs	MIDI: ⅛ in trs standard MIDI input DC: 3.3V
Outputs	MIDI: 3.3V isolated MIDI
Functionality	The Isolator protects the microcontroller from voltages present on the MIDI input jack, as well as level shifting the MIDI signals to 3.3V logic.

TABLE III-d
PERIPHERAL CONTROLLER MODULE BREAKDOWN

Module	**Peripheral Controller**
Inputs	User Encoder: Digital quadrature signal Parameter Knobs: Analog voltage User Keyboard: Digital input per ‘key’ DC: 3.3V
Outputs	Digital Bus: Values reported on change Interrupt Line: Interrupt low edge on parameter change
Functionality	The Peripheral Control block manages muxing and polling all of the user inputs, then reporting them to the microcontroller through a digital bus and interrupt line.

TABLE III-e
MEMORY MODULE BREAKDOWN

Module	**Memory**
Inputs	Digital Bus: I2C bidirectional DC: 3.3V
Outputs	Digital Bus: I2C bidirectional
Functionality	The nonvolatile Memory saves and returns parameters and wavetable data for the uController.

TABLE III-f
DAC MODULE BREAKDOWN

Module	**DAC**
Inputs	Digital Bus: 8BIT audio stream DC: 3.3V, 5V, 2.5Vref
Outputs	Audio: 8Vpp audio signal
Functionality	The DAC converts the wavetable digital audio stream to an analog audio signal to be filtered and amplified in the analog signal path.

TABLE III-g
DCF (DIGITALLY CONTROLLED FILTER) MODULE BREAKDOWN

Module	**DCF (Digitally Controlled Filter)**
Inputs	Audio: 3.3Vpp audio signal Control Signal: I2C filter control commands from uController DC: 3.3, 5V, 2.5Vref
Outputs	Audio: 4Vpp audio signal
Functionality	The DCF filters the audio signal with a resonance and cutoff determined by control values on the I2C bus.

TABLE III-h
DCA (DIGITALLY CONTROLLED AMPLIFIER) MODULE BREAKDOWN

Module	**DCA (Digitally Controlled Amplifier)**
Inputs	Audio: 4Vpp audio signal Control Signal: I2C filter control commands from uController DC: 3.3V, 5V, 2.5Vref
Outputs	Audio: 4Vpp audio signal
Functionality	The DCA amplifies the audio signal with an amplitude and saturation value from the I2C bus.

TABLE III-i
FINAL AMP MODULE BREAKDOWN

Module	**Final AMP (Buffer Amplifier)**
Inputs	Audio: 4Vpp audio signal User Volume Knob: Analog resistance Value DC: 3.3V, 5V, 2.5Vref
Outputs	Line Output: 1.41Vpp audio signal Headphone Output: 0-15mW into 32ohm load
Functionality	The Final AMP amplifies and isolates the audio signal for the Line output. It also drives the headphone output with appropriate power (determined from user volume input).

SECTION 4 - DESIGN

4.1 HARDWARE DESIGN AND ASSOCIATED SOFTWARE CONTROL

The following chapter outlines the design process followed in constructing and testing each “module” of the circuitry. To ensure the overall design was valid for the entire project, I opted to construct and verify both the hardware and software for each portion of the power, user control, and audio circuitry. This ensured that the final construction of the entire circuit would not harbor so many issues that it proves very difficult to debug. I found this technique especially important while building the host of code that controls each portion of the synthesizer because as a whole, the interdependencies of the full software build make debugging the peripheral errors very difficult.

4.2 ESP32 S2 WORKFLOW

In the final revision of the project, an ESP32S2 microcontroller is implemented with all the supporting hardware on the main PCB. Though, as the ESP32S2 is a surface mount microcontroller and comes without a USB UART for programming, power supply, or reset buttons, I opted to use a development board while prototyping. This allowed me to get a functional microcontroller up and running without first ordering a PCB to program and easily break out the ESP32S2’s I/O to a breadboard.

The development board I choose was the ESP32S2 Saola-1, a very inexpensive development platform for the ESP32S2 chipset with USB to UART, I/O breakout, and onboard voltage regulation.

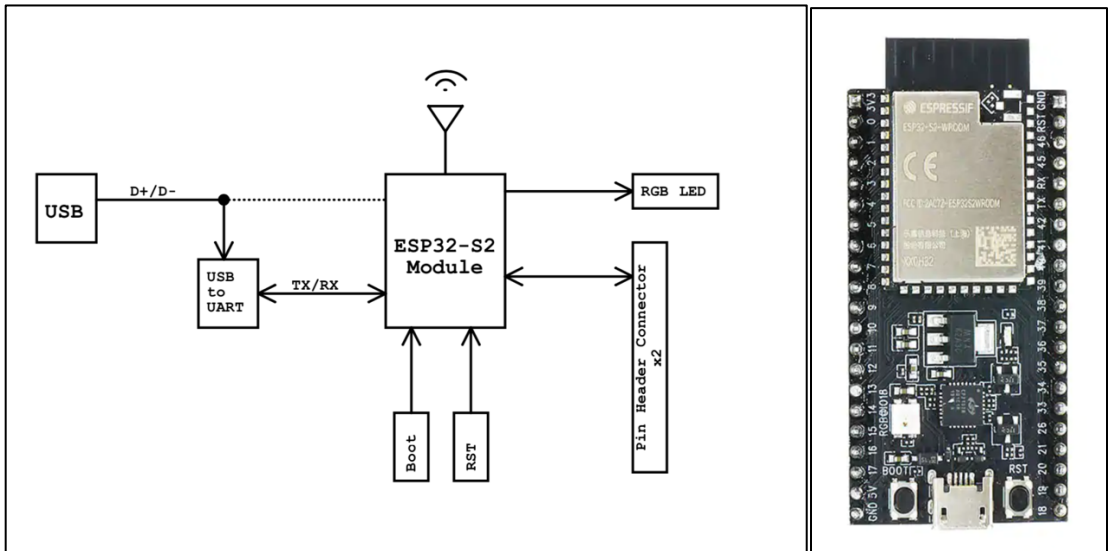


FIGURE 3: ESP32SA SOALA 1 DEVELOPMENT BOARD AND BLOCK DIAGRAM

The ESP32S2 requires a toolchain to allow the compiling/linking and uploading of code to the ESP32 S2 chipset. These tools and their installation instructions can be found on Espressif's support page for the ESP32S2 [13]. My experience was initially frustrating with these Python based tools, as linking the directory and ensuring it ran the scrips in Python 3 was difficult on my machine. Though, once installed and properly setup compiling and uploading code was as simple as running two command line scripts.

4.3 VOLTAGE REGULATION and VOLTAGE REFERENCE

The power supply for this project focuses mainly on isolating the analog and digital supplies to prevent bleed through of digital switching noise into the audio path. The first stage is tasked with boosting the battery voltage of 3V to an unregulated 6.5V volt rail that supplies the linear regulators and the LED display. I choose the LM2623 switch mode boost regulator as it

can be bootstrapped to operate on very low input voltages and boasts up to 90% efficiency even at low input voltages. In this configuration it can supply up to 2A of output current with an input above 0.8V. This low drop out voltage ensures that the regulator will provide an output as the AA cells drain and their nominal voltage drops from 3V down to 2V. The circuit below is a direct implementation of the "typical application circuit" in the LM2623 datasheet [19]

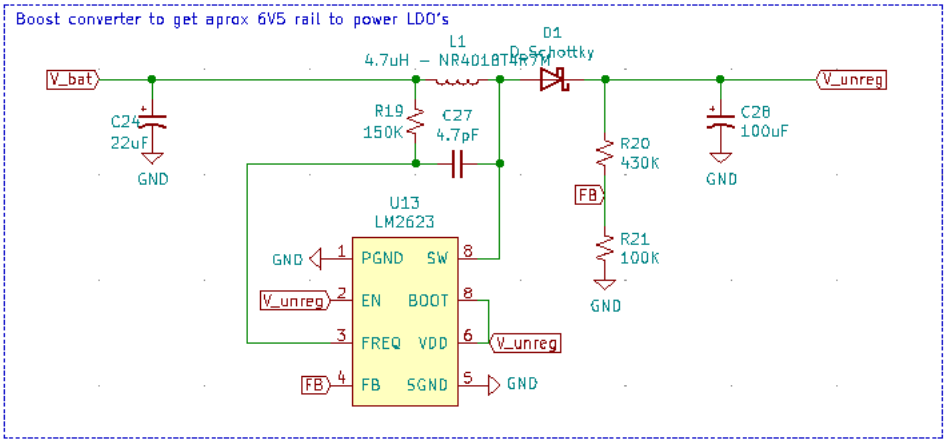


FIGURE 4: SWITCHMODE BOOST CONVERTER

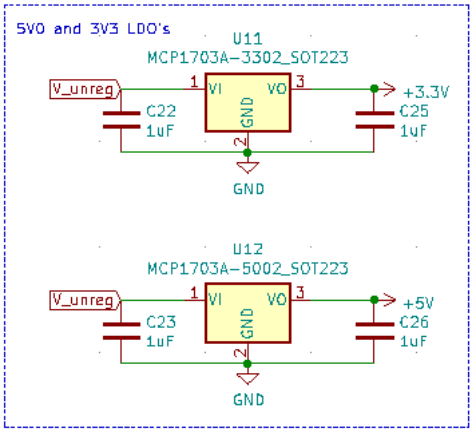


FIGURE 5: 3.3V AND 5V0 LINEAR REGULATORS

This 6.5V switch mode regulated supply is then lowered to 5V and 3.3V rails via two linear regulators. I choose two standard low-drop-out (LDO) regulators, the MCP1703A-3302 and the MCP1703A-5002. They are implemented in a standard positive rail linear regulator configuration with input and output filtering via 1uF capacitors. This provides two isolated

supplies: a 3.3V logic supply (noisy digital rail) and a 5V analog supply. The separate linear regulators ensure adequate isolation between the noise on the digital supply and the analog supply.

Because all the op amps and analog circuitry in the synthesizer are powered from a single 5V rail, a centered analog reference voltage of 2.5V is required for bias and centering. This was generated using a spare op amp in the MCP6L02 package. This op amp is wired as a unity-gain buffer and stiffens a 2.5V voltage reference created using two 10K resisters in a voltage divider configuration. Note the filtering capacitors at both the input and output of the buffer. These ensure the buffer is not sensitive to noise on in the voltage divider nor susceptible to oscillations caused by the direct negative feedback in the buffer.

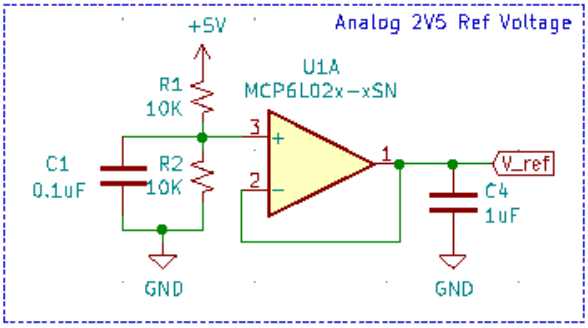


FIGURE 6: 2.5V ANALOG REFERENCE VOLTAGE GENERATION

4.4 DAC and FILTERING

Initially, the design outlined the use of a DAC IC to convert the waveform data generated in the ESP32 S2 microcontroller to an analog waveform that can be filtered and amplified by the rest of the analog audio chain. After reading the technical reference on the ES32S2 chipset, I found that there were two onboard DAC's that use an internally hardwired I2S port. Unfortunately I ran into issues with updating this I2S DAC at a sufficient rate without significant software overhead.

The final solution was to use a dithered PWM peripheral on the ESP32S2 to generate a PWM waveform with a duty cycle representative for the desired "DAC" output. This "Sigma Delta" PWM peripheral requires a setup struct to set its channel, clock prescale, initial duty, and output GPIO.

```
static void sigmadelta_init(void)
{
    sigmadelta_config_t sigmadelta_cfg = {
        .channel = SIGMADELTA_CHANNEL_0,
        .sigmadelta_prescale = 1,
        .sigmadelta_duty = 0,
        .sigmadelta_gpio = GPIO_NUM_21,
    };
    sigmadelta_config(&sigmadelta_cfg);
}
```

FIGURE 7: SIGMA DELTA INITIALIZATION

To change the PWM duty, or value of the DAC output, we simply need to load a new duty value into the register using the "set_duty" call.

```
sigmadelta_set_duty(SIGMADELTA_CHANNEL_0, current_sample);
```

FIGURE 8: UPDATING THE SIGMA DELTA DUTY CYCLE

This output is not suitable for our audio chain, as it is not centered about the 2.5V analog reference and contains high frequency content from the PWM switching. The following circuit conditions the PWM signal by decoupling it with a 10uF film capacitor then passing this signal into an MCP6l02 OP amp in a unity gain configuration. The op amp both buffers and low-pass filters the signal to frequencies below the GBW of the OPAMP. Also note the 470K resistor between the decoupled input and the 2.5V reference rail. This resistor ensures the decoupled signal tends to center at the 2.5V reference, it also ensures that the op amp stage remains biased.

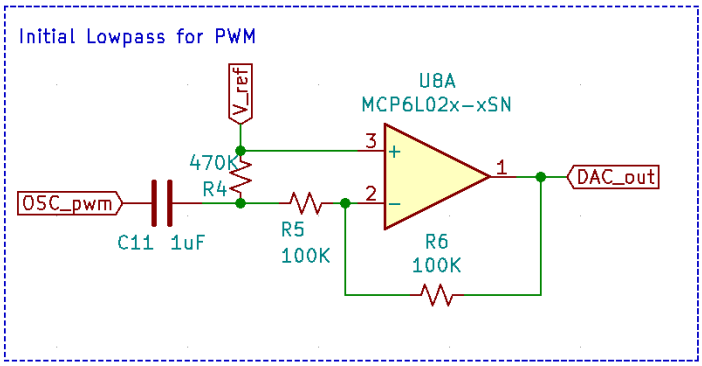


FIGURE 9: LOWPASS AND BUFFERING OF SIGMA DELTA PWM

4.5 DIGITALLY CONTROLLED FILTER

The digitally controlled filter is one of the most important circuits in the synthesizer. In many synthesizers the filter's "character" defines the sound of the synth. For this synthesizer I needed a low cost, simple to implement, two pole lowpass with a cutoff that can be controlled by the microcontroller.

This is a design challenge in many ways, as often a voltage or clock controllable filter is a complicated and expensive circuit. I elected to implement the MAX7490 switched capacitor building block. This IC has two identical filter stages that contain a clocked capacitor element to change their pole locations.

The majority of this design was obtained from the MAX7490 datasheet reference example section [14]. I implemented the mode 1 second order filter in a lowpass configuration.

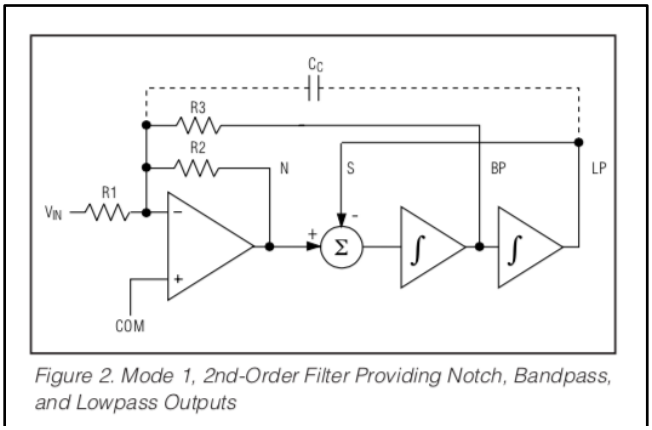


FIGURE 10: MAX7490 MODE 1 ARCHITECTURE [14]

The mode 1 design equations from the same section provide the derived cutoff frequency, peak gain, and resonance/Q of the entire system.

Mode 1 Design Equations

$$f_O = \frac{f_{CLK}}{100}$$

$$f_{notch} = f_O$$

$$Q = \frac{R3}{R2}$$

$$H_{OLP} = \frac{-R2}{R1}$$

$$H_{OBP} = \frac{-R3}{R1}$$

$$H_{ON1}(\text{as } f \rightarrow 0\text{Hz}) = \frac{-R2}{R1}$$

$$H_{ON2}(\text{at } f = f_{CLK}/2) = \frac{-R2}{R1}$$

FIGURE 11: MAX7490 MODE 1 DESIGN EQUATIONS [14]

The first stage of the design sets R1 and R2 and R3 to a value of 100K to obtain a unity gain single pole lowpass with a Q-factor of 1. This is fed to the second lowpass stage. This stage is different as R3 and R2 are ganged together as a combination of a 100K potentiometer and two 47K resistors. This allows the resonance of the second lowpass stage to be modified via RV2. As the variable resistor is adjusted from one extreme to the other, the ratio of R3/R2 is varied from 147K/47K to 47K/147K varying the Q from 0.32 to 3.13.

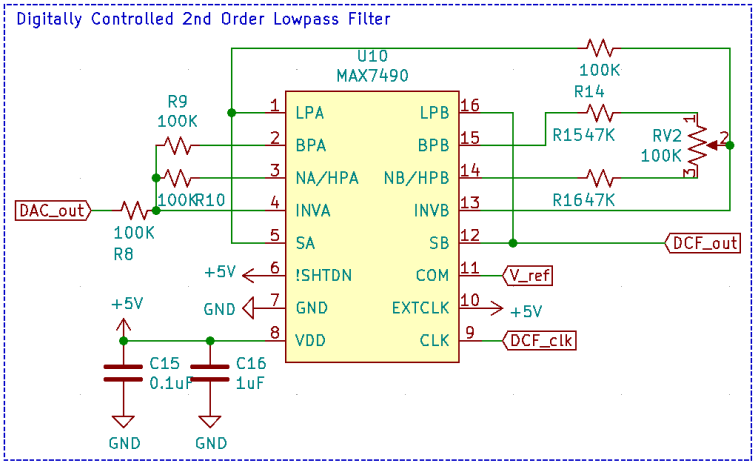


FIGURE 12: SECOND ORDER VARIABLE Q DCF USING MAX7490 IC

Also of important note is that the cutoff of the whole filter block is controlled by the clock signal present on pin 9. As shown in the design equation, the cutoff is proportional to this clock by a ratio of 1/100. For example the cutoff would be set to 1kHz if the clock provided is 100kHz. The benefit of this architecture is that no digital to analog conversion of a control voltage is required, only a clock signal 100x the desired cutoff.

The code block below sets up a PWM output peripheral using the MSP32S2 LEDC peripheral. Typically, LEDC is used to control the brightness of an LED or similar PWM controlled system. I leveraged this peripheral to generate a variable frequency clock signal at a 50% duty cycle. This LEDC peripheral is initialized to a 50% duty which we never change. The clock output from this GPIO is connected directly to pin 9 of the MAX7490.

```
ledc_channel_config_t ledc_channel_fltr_clk = {
        .channel    = LEDC_CHANNEL_1,
        .duty       = 0,
        .gpio_num   = FLTR_CLK_PIN,
        .speed_mode = LEDC_LOW_SPEED_MODE,
        .hpoint     = 0,
        .timer_sel  = LEDC_TIMER_2
};
```

FIGURE 13: INITIALIZING THE LEDC PERIPHERAL TO GENERATE THE DCF CLK SIGNAL

To update the cutoff frequency, the LEDC register is first updated to load the last frequency that was requested. Then, a new desired cutoff is loaded into the “ledc_freq_hz” register to be updated on the next call.

```
//push last val to pwm reg, do this first for timing reasons
ledc_timer_config(&ledc_timer_fltr_clk);
update_DCA(DCA_aten);
```

FIGURE 14: CHANGING THE DCF CLK FREQUENCY

4.6 DIGITALLY CONTROLLED AMPLIFIER

The next large building block of any synthesizer is an automated gain stage called a DCA or VCA, both controllable amplifiers. Since the synthesizer’s parameters are all controlled by the ESP32S2 microcontroller, I opted to implement a three-wire serial bus controlled volume integrated circuit. The LM1971 was a good option as its price is relatively low, it requires very few external components, operates at a 5V analog rail (with 3.3V tolerant I/O), and has a “mute” mode in which the input and output are essentially isolated.

The final schematic design is obtained entirely from the typical application design in the datasheet [15]. The only difference is that in the implemented design the output is not buffered because it is driving an op amp in the final amplifier section not a low impedance line load directly.

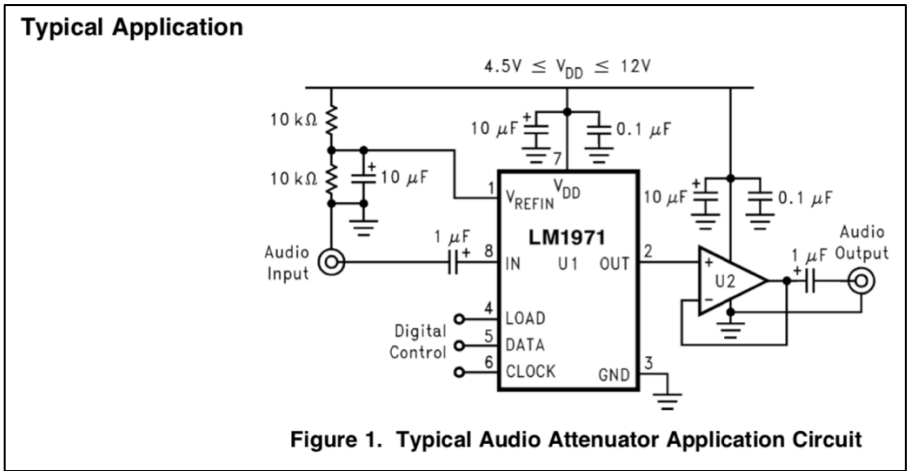


FIGURE 15: LM1971 RECOMMENDED APPLICATION [15]

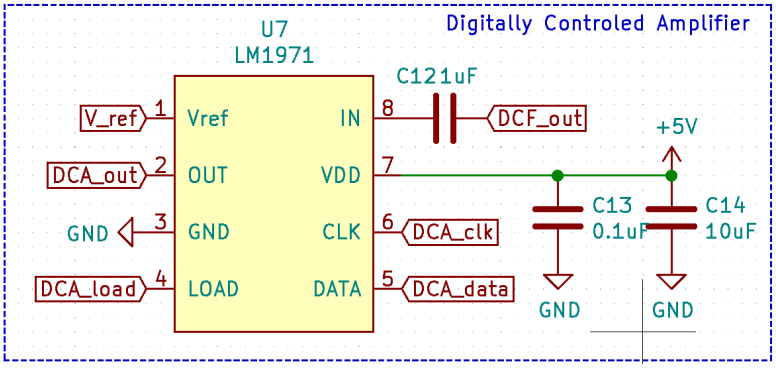


FIGURE 16: IMPLEMENTATION OF LM1971 VOLUME CONTROL IC

To control the LM1971 IC I implemented a simple “bit banged” 3 wire serial bus interface. First the I/O is initialized in an output mode and set to the default states of the bus.

```
    // 3 WIRE BUS FOR DISPLAY
    gpio_reset_pin(DIS_DAT);
    gpio_reset_pin(DIS_CLK);
    gpio_reset_pin(DIS_LAT);
    gpio_set_direction(DIS_DAT, GPIO_MODE_OUTPUT);
    gpio_set_direction(DIS_CLK, GPIO_MODE_OUTPUT);
    gpio_set_direction(DIS_LAT, GPIO_MODE_OUTPUT);
    gpio_set_level(DIS_CLK, 0);
    gpio_set_level(DIS_LAT, 0);
```

FIGURE 17: INITIALIZATION OF DCA 3-WIRE SERIAL I/O

The following code loops over the requested 2 bytes to be sent, masking each bit and clocking and latching it onto the bus. The first byte is a channel selection byte which is ignored by the LM1971 as it is a single channel device. The second byte sets the channel attenuation between 0dB (0X00) and 96dB (0X3F).

```
//// UPDATES DCA OVER 3WIRE BUS ////
void update_DCA(int volume){
    gpio_set_level(DCA_CLK, 0); //clk low
    gpio_set_level(DCA_LAT, 0); //load low
    uint16_t data_bytes = 0x0000 | volume;

    for(int bit=0; bit<=15; bit++){
        gpio_set_level(DCA_CLK, 0); //falling edge clk line

        //check if cur bit is high
        if(((0x8000 >> bit) & data_bytes) > 0){gpio_set_level(DCA_DAT, 1);}
        else{gpio_set_level(DCA_DAT, 0);}

        gpio_set_level(DCA_CLK, 1);//rising edge clk line
    }

    gpio_set_level(DCA_LAT, 1); //latch high
    gpio_set_level(DCA_DAT, 1);
}
```

FIGURE 18: IMPLEMENTATION OF 3-WIRE SERIAL FOR DCA

4.7 LINE and HEADPHONE AMPLIFIER

The line and headphone amplifier sections are tasked with amplifying the final audio signal in the path and conditioning it to both a line level output and a headphone output while also allowing user volume control. The first stage amplifies the output of the DCA with an inverting amplifier setup of gain 4.7X. This amplified signal is then attenuated by the user volume control knob and coupled to the line output via a 10uF coupling capacitor.

The second stage handles duplicating and amplifying the line level output to two suitable headphone output channels. This is accomplished by the OPA1688, a purpose built low voltage headphone amplifier OP AMP.

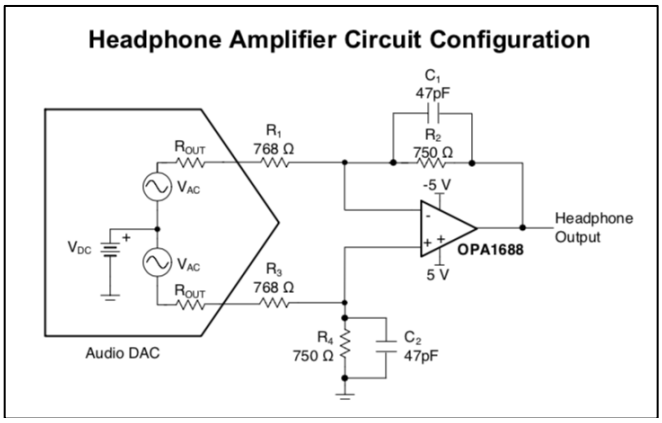


FIGURE 19: OPA1688 RECOMMENDED HEADPHONE AMPLIFIER CIRCUIT [20]

The OPA1688 is implemented differently than the recommended datasheet configuration [20]. The setup is modified to be used in single supply mode of 5V. Instead of driving the input with a balanced signal, the non-inverting input is tied to VREF (the 2.5 volt reference). The output is also coupled using two 100uF capacitors. The coupling capacitors ensure the output has no DC bias, lowering the drive power at idle and ensuring the headphones never experience a damaging DC bias.

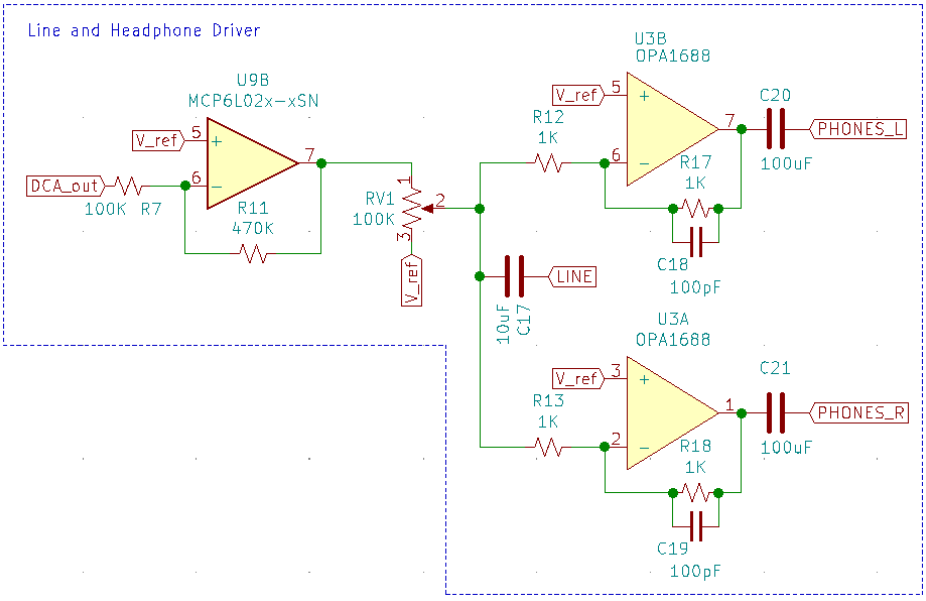


FIGURE 20: LINE AND HEADPHONE AMPLIFIER CIRCUIT AS IMPLEMENTED

4.8 MIDI IMPLEMENTATION

The MIDI implementation is an important part of the synthesizer as it allows a MIDI keyboard or sequencer to control what notes are played as well as whether the filter or amplifier envelopes are triggered. The MIDI standard has been in place since 1983 and is well documented [3].

The circuit below converts the midi current loop from the controlling device into a 3v3 logic level signal the ESP32S2 can interpret. The midi signal is first passed through switch that allows the user to invert the polarity of the 1/8in midi jack. This jack is now included in the MIDI standard with a standardized pinout, but unfortunately many manufacturers already implemented 1/8 in TRS MIDI with a reverse polarity in many devices. The switch ensures that a user will be able to use this synth with a MIDI device of either polarity.

The signal is then current limited with a 330 ohm resistor and used to drive the LED in the H11L1 Schmidt triggered optocoupler. A reverse polarity protection diode is also connected to shunt current when the polarity switch is in the wrong position for the incoming midi current loop. The output of the optoisolator is pulled up to logic 3.3V with a 10Kohm resistor then applied directly to pin 19 of the ESP32 to be read in as serial data.

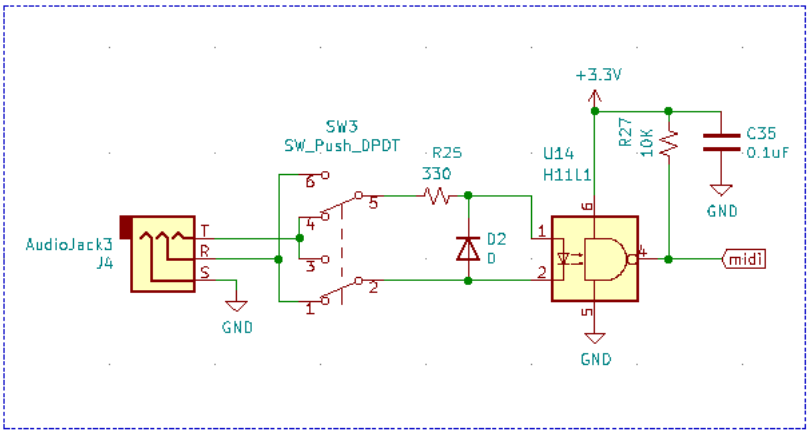


FIGURE 21: MIDI ISOLATION AND LEVEL CONVERSION

Because MIDI follows a standard serial format with a baud rate of 31,250, the ESP32S2's serial UART can be setup to handle the receiving and buffering of the MIDI data directly without any software polling or loops.

```
//// UART DRIVER INIT FOR MIDI ////
void midi_uart_init(void){
    /* Configure parameters of an UART driver,
     * communication pins and install the driver */
    uart_config_t uart_config = {
        .baud_rate =  ECHO_UART_BAUD_RATE,
        .data_bits = UART_DATA_8_BITS,
        .parity    = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
        .source_clk = UART_SCLK_APB,
    };
    int intr_alloc_flags = 0;

    #if CONFIG_UART_ISR_IN_IRAM
    intr_alloc_flags = ESP_INTR_FLAG_IRAM;
    #endif

    ESP_ERROR_CHECK(uart_driver_install(ECHO_UART_PORT_NUM, BUF_SIZE * 2, 0, 0,
NULL, intr_alloc_flags));
    ESP_ERROR_CHECK(uart_param_config(ECHO_UART_PORT_NUM, &uart_config));
    ESP_ERROR_CHECK(uart_set_pin(ECHO_UART_PORT_NUM, ECHO_TEST_TXD,
ECHO_TEST_RXD, ECHO_TEST_RTS, ECHO_TEST_CTS));
}
```

FIGURE 22: INITIALIZATION OF ESP32S2 UART FOR RECEIVING MIDI DATA

The MIDI must be parsed into controls for the synth as well as pitch data. The next code snippet shows the runtime loop that reads available data from the UART, checks if it is an appropriate length, checks if it is a “note on” or “note off” command, and adds or removes notes from the notes structure.

```
// Read data from the MIDI UART and handle adding/removing notes
        int len = uart_read_bytes(ECHO_UART_PORT_NUM, data, BUF_SIZE, 20 /
portTICK_RATE_MS);
        if(len >= 3){
            for(int byte_index = 0; byte_index <= len - 3; byte_index++){

                if((data[byte_index] & 0xF0) == 0x90){
                    if(data[byte_index + 2] == 0){
                        remove_note((int)data[byte_index + 1]);
                    }
                    else{
                        add_note((int)data[byte_index + 1]);
                    }
                }
                else if((data[byte_index] & 0xF0) == 0x80){
                    remove_note((int)data[byte_index + 1]);
                }
            }
        }
```

FIGURE 23: READING AND PARSING MIDI DATA FROM THE UART BUFFER

4.9 DISPLAY MULTIPLEXING

Instead of implementing an LCD or multicharacter screen, I choose the PSA08-115RWA 16 segment LED display to show the waveform and synth state. The PSA08-115RWA contains no logic or LED drive circuitry and therefore requires external components to control it with the ESP32 [16].

To drive the display, I implemented the TLC59283 LED drive IC. This IC allows the control of 16 common anode LED's at a constant current set by a single external resistor [17]. This resistor was chosen to provide a minimum legible brightness to conserve current draw. The rest of the connections are identical to the application example in the datasheet. The current is sunk through the IC from each LED channel. The only interesting design decision here is that the LED common anode is driven from the unregulated supply to prevent any switching current variations from generating noise in the analog path. The TLC59283 is powered via the 3.3V logic net as it is controlled via 3.3V logic levels.

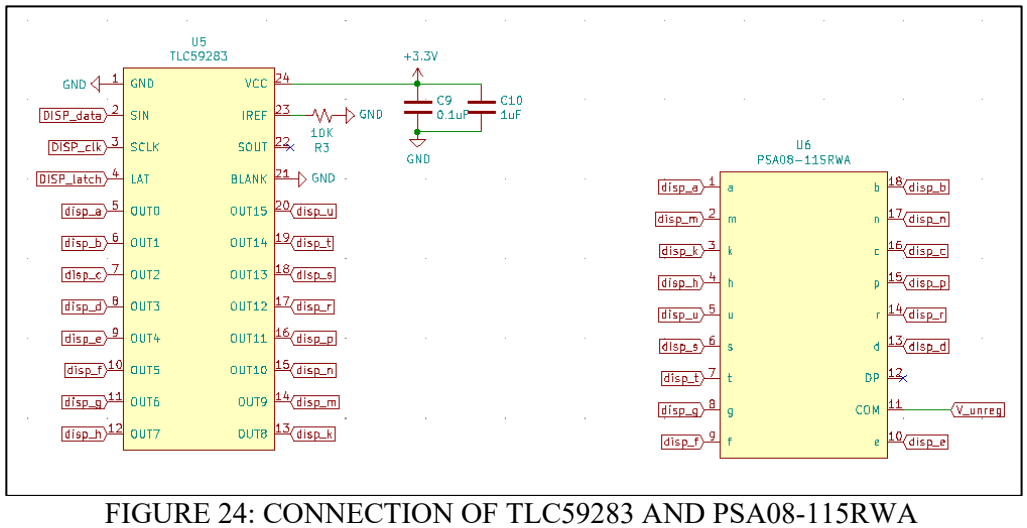


FIGURE 24: CONNECTION OF TLC59283 AND PSA08-115RWA

The three-wire bus code is very similar to the code that controls the DCA. First the I/O is initialized.

```
    // 3 WIRE BUS FOR DISPLAY
    gpio_reset_pin(DIS_DAT);
    gpio_reset_pin(DIS_CLK);
    gpio_reset_pin(DIS_LAT);
    gpio_set_direction(DIS_DAT, GPIO_MODE_OUTPUT);
    gpio_set_direction(DIS_CLK, GPIO_MODE_OUTPUT);
    gpio_set_direction(DIS_LAT, GPIO_MODE_OUTPUT);
    gpio_set_level(DIS_CLK, 0);
    gpio_set_level(DIS_LAT, 0);
```

FIGURE 25: INITIALIZATION OF DISPLAY 3-WIRE SERIAL I/O

Then the bus can be driven via the same “bit banged” three wire bus code. This presents a bit masked bit of the two bytes on the data line, toggles the clock, and latches the data in.

```
//// UPDATES DISPLAY OVER 3WIRE BUS ////
void update_display(uint16_t data_bytes){

    gpio_set_level(DIS_LAT, 1); //pulse latch
    gpio_set_level(DIS_LAT, 0);

    for(int bit=0; bit<=15; bit++){
        //check if cur bit is high
        if(((0x8000 >> bit) & data_bytes) > 0){
            gpio_set_level(DIS_DAT, 1);
        }
        else{
            gpio_set_level(DIS_DAT, 0);
        }

        //pulse clk line
        gpio_set_level(DIS_CLK, 1);
        gpio_set_level(DIS_CLK, 0);
    }

    gpio_set_level(DIS_LAT, 1); //pulse latch
    gpio_set_level(DIS_LAT, 0);
    gpio_set_level(DIS_DAT, 0);
}
```

FIGURE 26: IMPLEMENTATION OF 3-WIRE SERIAL FOR DISPLAY DRIVER

4.10 USER INPUT

Because almost all of the synth's parameters are controlled by the ESP32S2 microcontroller, it is necessary for the user to provide an input that the ESP32S2 can read. The circuit is simply a filtered voltage divider that provides a voltage output between zero and 2.6 volts.

The 2.6 volt maximum arises from a quirk using the ESP32S2's internal ADC's in which even at the highest attenuation mode, the 13bit value overflows past an input of 2.6 volts [18]. The filtering provided by the 0.1uf capacitor ensures that no high frequency noise on the 3V3 rail will be coupled into the analog voltage at the ADC pin. This stabilizes the reading significantly.

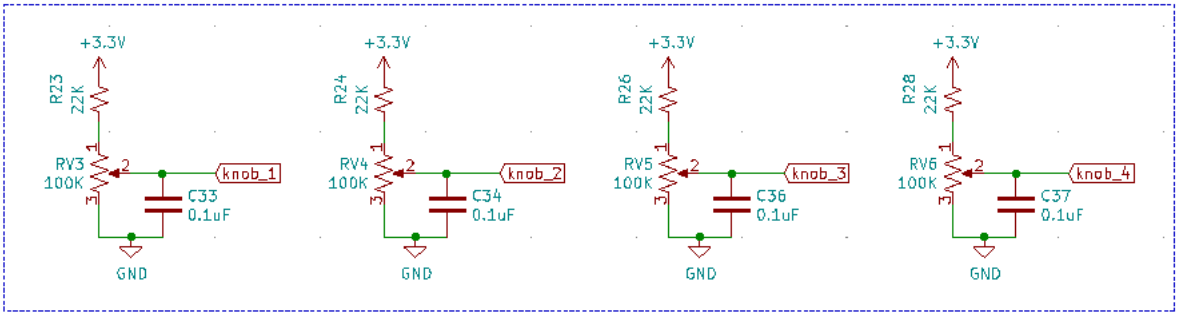


FIGURE 27: USER INPUT POTENTIOMETERS AND FILTERING

The following code snipped shows the initialization of the ADC's with the attenuation settings described previously.

```
// ADC CONFIG for KNOBS
adc1_config_width(ADC_WIDTH_BIT_13);
adc1_config_channel_atten(ADC1_CHANNEL_0,ADC_ATTEN_DB_11);
adc1_config_channel_atten(ADC1_CHANNEL_1,ADC_ATTEN_DB_11);
adc1_config_channel_atten(ADC1_CHANNEL_2,ADC_ATTEN_DB_11);
adc1_config_channel_atten(ADC1_CHANNEL_3,ADC_ATTEN_DB_11);
```

FIGURE 28: CONFIGURATION OF ESP32S2 ADC's

Unfortunately, even with the capacitor filtering, the ADC readings still vary 10 or 15% due to noise. To stabilize the ADC reading, I implemented a multisampling system in which the ADC is polled a number of times to average the reading across a longer time span. The

ADC_POS is an accumulation of these reads, to extract the KNOB_POS the ADC_POS value is divided by the number of reads.

```
        //clear last vals
        int ADC_POS[] = {0,0,0,0};

        //check knob positions
        for (int i = 0; i < MULTISAMP; i++) {
            ADC_POS[0] += adc1_get_raw(ADC1_CHANNEL_0);
            ADC_POS[1] += adc1_get_raw(ADC1_CHANNEL_1);
            ADC_POS[2] += adc1_get_raw(ADC1_CHANNEL_2);
            ADC_POS[3] += adc1_get_raw(ADC1_CHANNEL_3);
        }
```

FIGURE 29: READING AND AVERAGING POTENTIOMETER ADC VALUES

The other user input that is required is the mode button. This is a simple circuit consisting of a momentary single pole NO switch that is pulled up to 3V3. When depressed, a falling edge is generated on the I/O pin of the ESP32S2. This falling edge is debounced and toggles a state flag that can be read by the other functions.

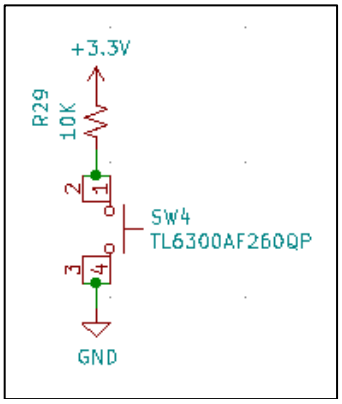


FIGURE 30: MODE SELECTION BUTTON

SECTION 5 - FINAL DEVELOPMENT PLATFORM

5.1 CONSTRUCTION

Initially, testing of each circuit was done on a breadboard. This allowed quick modification and troubleshooting of the circuits as well as for the circuit to be fully tested and verified as functional before the final schematics were completed and the circuit was moved to a more permanent perf board construction.

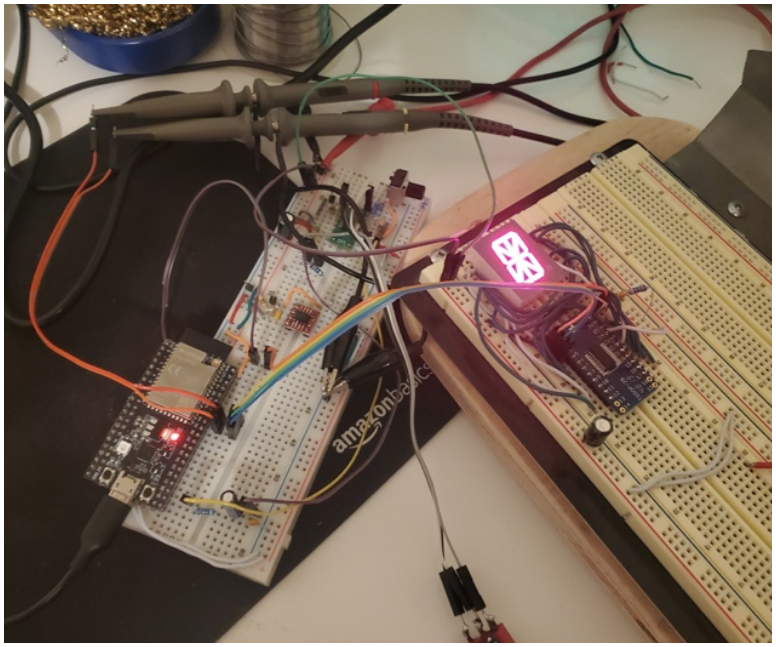


FIGURE 31: INITIAL BREADBOARD TESTING

After the whole circuit was verified and minor modifications to the design noted in the final schematics, the circuit was moved to a perf board setup. The perf board allows components to be soldered into a semi-permanent development platform that is much more stable than the initial tests on breadboards. The perf board also made it easier to structurally mount potentiometers, connectors, and bulkier components like the LED display.

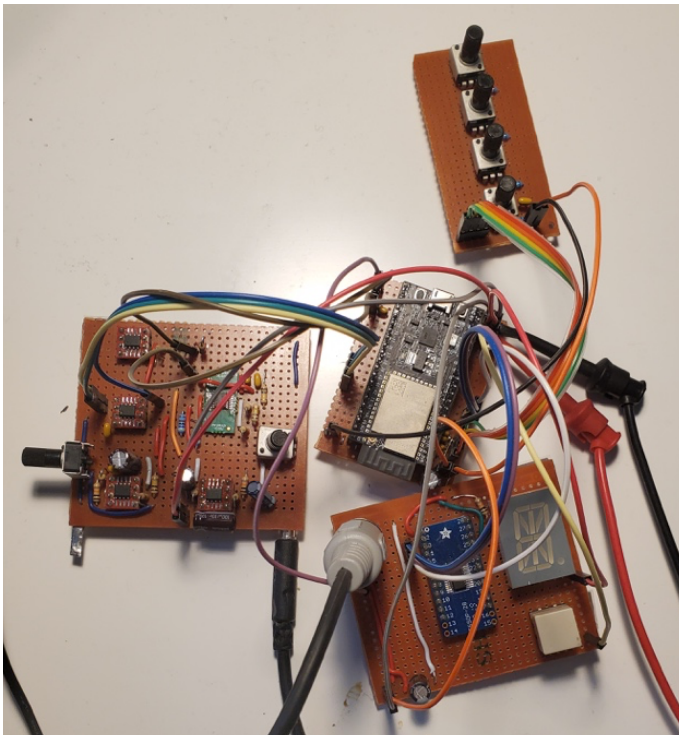


FIGURE 32: FINAL PERF BOARD ASSEMBLY

The figures below show the wiring on the bottom of the perf board. Most connections can be made directly with the preconnected copper features that run along the rows. These can be cut with a small drill bit to separate them into smaller lengths. 20-gauge stranded wire was used where connections needed to be made between these strips. On the display board, a ribbon cable was split out to make the connections between the display and driver IC as the smaller gauge wire was perfect for the more dense connections.

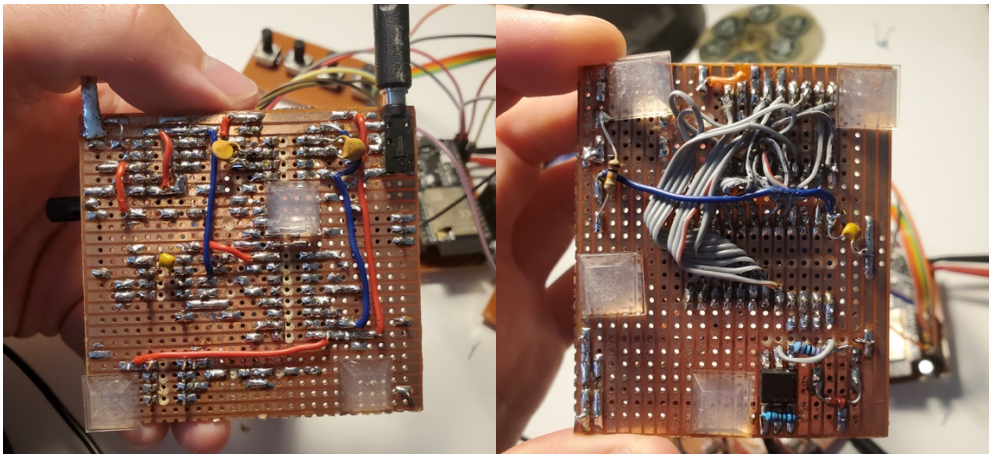


FIGURE 33: BOTTOM DETAIL OF FINAL PERF BOARD ASSEMBLY

5.2 SOFTWARE FLOW DIAGRAMS

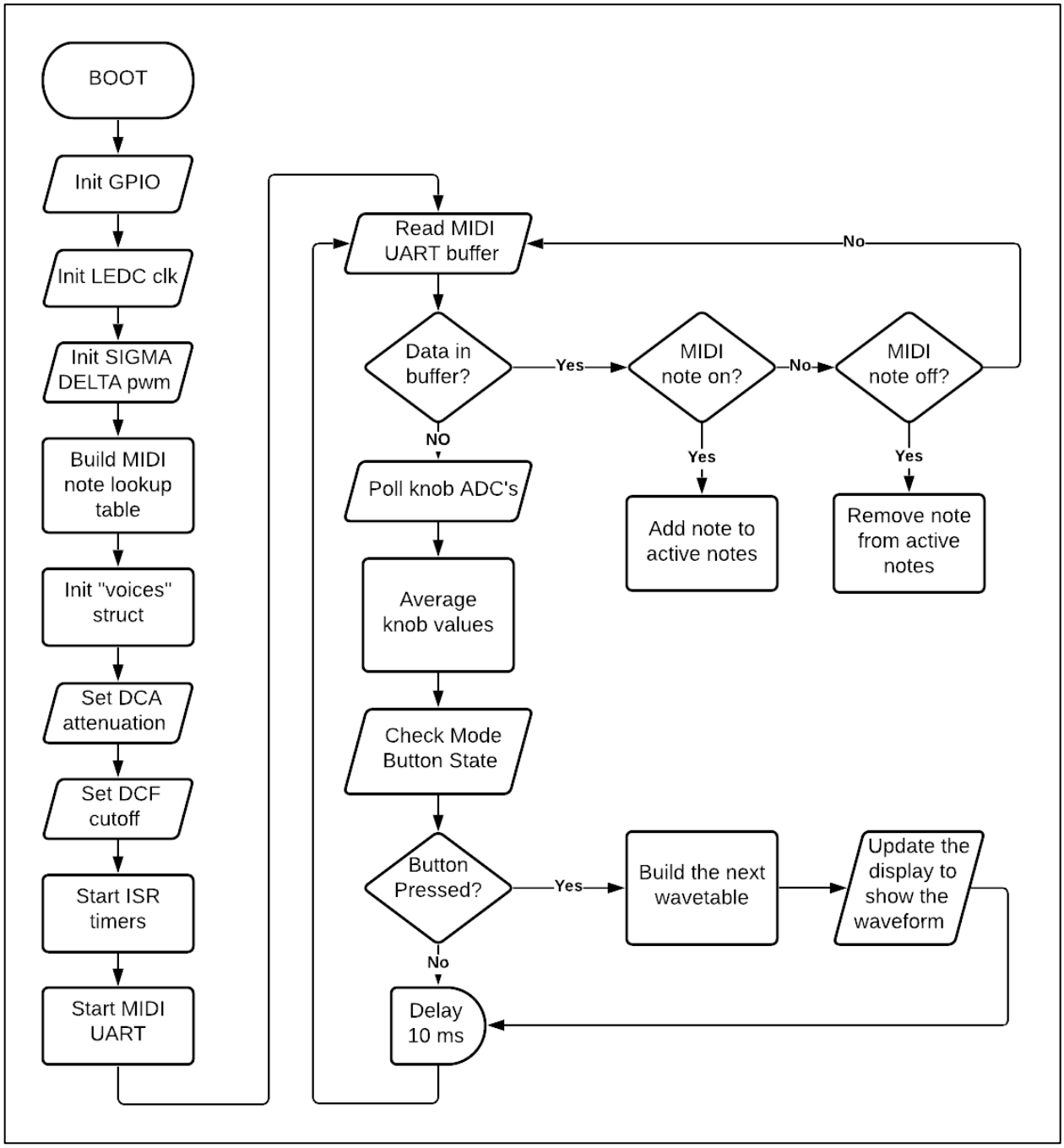


FIGURE 34: SOFTWARE FLOW DIAGRAM OF MAIN LOOP

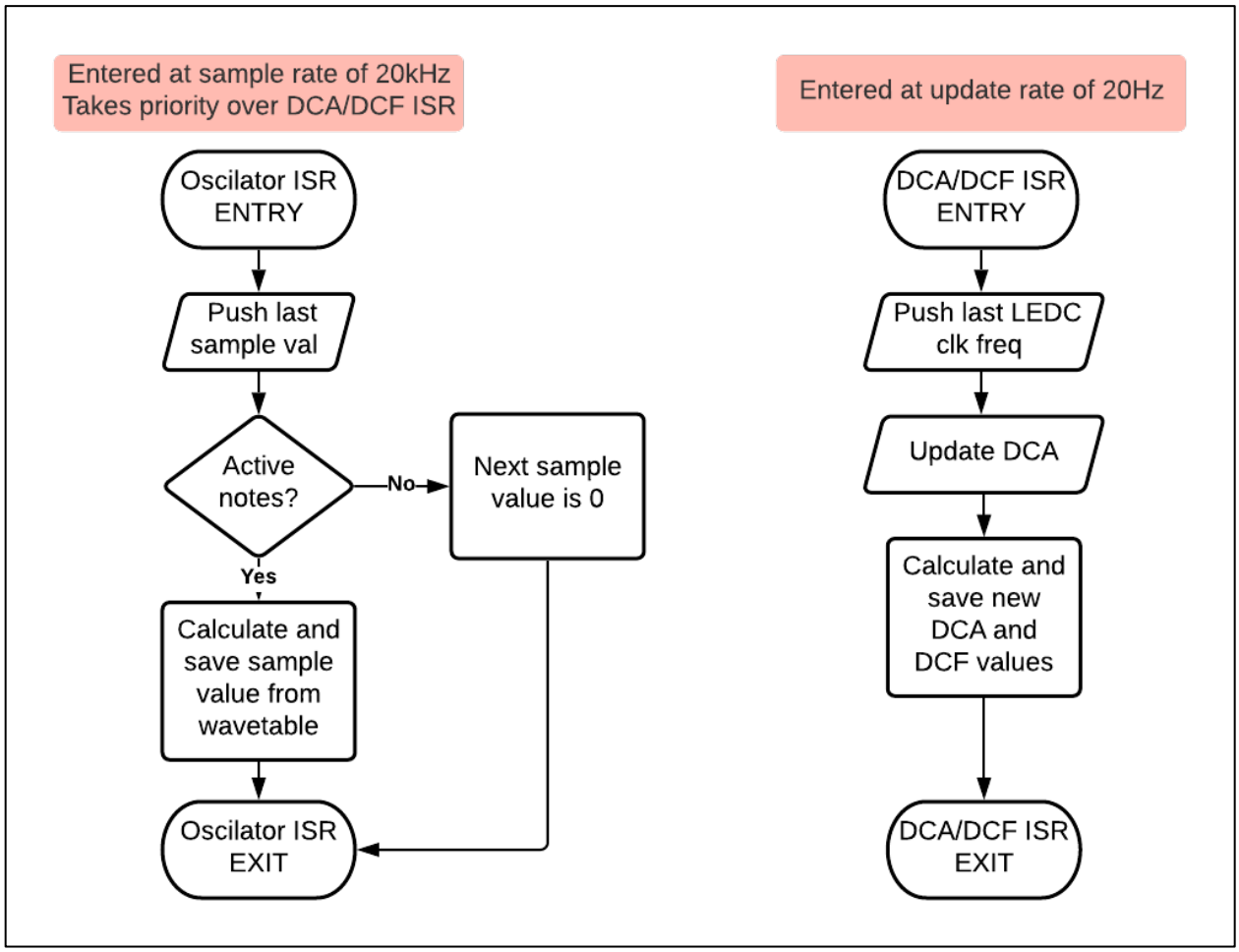


FIGURE 35: SOFTWARE FLOW DIAGRAM OF OSCILLATOR AND DCA/DCF ISR's

SECTION 6 – TESTING

The following chapter outline the testing and verification of the finalized circuitry assembled as the final development platform.

6.1 POWER CONSUMPTION

To verify that the synthesizer meets the power draw engineering specification of less than 150mA (see table I) the device-under-test [DUT] was connected to a lab power supply though a mA resolution digital multimeter. The device was powered, and various waveform modes were toggled while measurements were taken. The DUT was also measured while notes were being played as well as while notes were not being played.

FIGURE 36: BATTERY POWER DRAW TESTING

Design Specification: <150mA nominal current draw *Measured: 60mA and 74mA current draw* ***DUT PASSES***

6.2 LINE LEVEL OUTPUT

Verifying the synthesizer's "line" level output consisted of measuring the voltage at the "line output" connector with the HP54645A Oscilloscope. The test was performed with a single note being played in "square wave" mode.

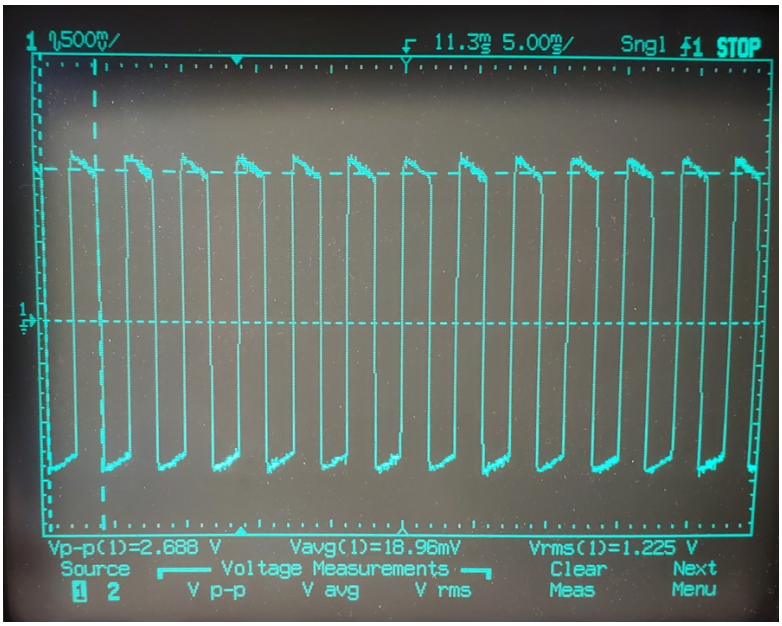


FIGURE 37: MEASUREMENT OF LINE OUTPUT AMPLITUDE

Design Specification: +4dBu (1.228 Vrms)
Measured: 1.225Vrms
DUT FAILS

Despite the DUT failing this test, the output was deemed acceptable as it is well above the standard "consumer" line level of 0.36 Vrms.

6.3 HEADPHONE DRIVE CAPABILITY

The synthesizer also provides the user with a “headphone” output. This output consists of identical left and right amplifier’s which are both driven from the “line” output signal. It is important for this output to provide the appropriate power into a headphone load. To test the output power, a dummy load was constructed to have a nominal resistance close to the 32ohm impedance specified in the design requirements.

FIGURE 38: MEASUREMENT AND SETUP OF HEADPHONE “DUMMY LOAD”

The left channel was connected to the resistor load and a middle C (262Hz) was played in sine wave mode. The voltage potential across the load was measured with the HP54645A Oscilloscope and the RMS voltage noted.

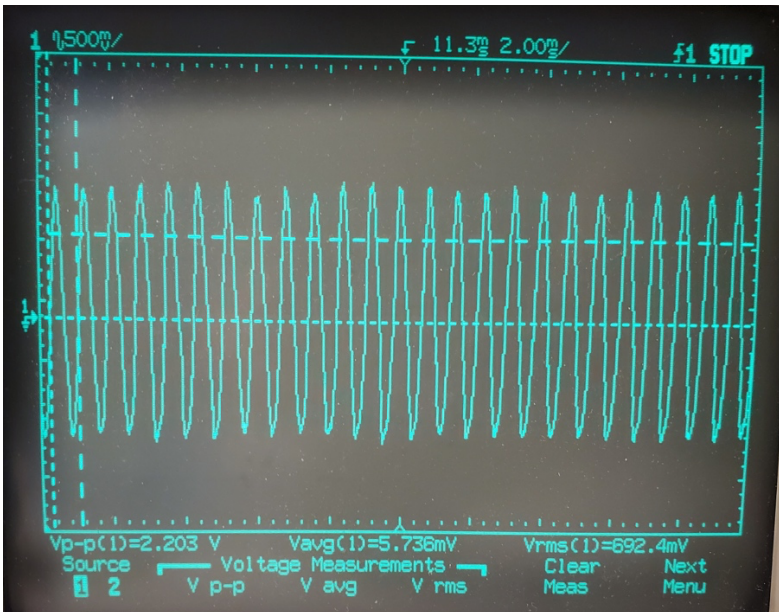


FIGURE 39: MEASUREMENT OF HEADPHONE POWER VIA LOAD VOLTAGE

$$P = \frac{V^2}{R}$$

$$\frac{0.692\ Vrms^2}{30.4\ Ohms} = 15.7mW$$

Design Specification: 10-20mW power per channel into 32ohm nominal load *Measured: 15.7mW* ***DUT PASSES***

6.4 MIDI LATENCY

As the synthesizer is controlled via MIDI note data, it is important to characterize the latency between the receiving of a MIDI command and the production of the requested audio at the output. Channel-1 of the HP54645A Oscilloscope was used to probe the serial data at the Schmidt triggered output of the optoisolator. Channel-2 was used to probe the output audio waveform at the “line” output.

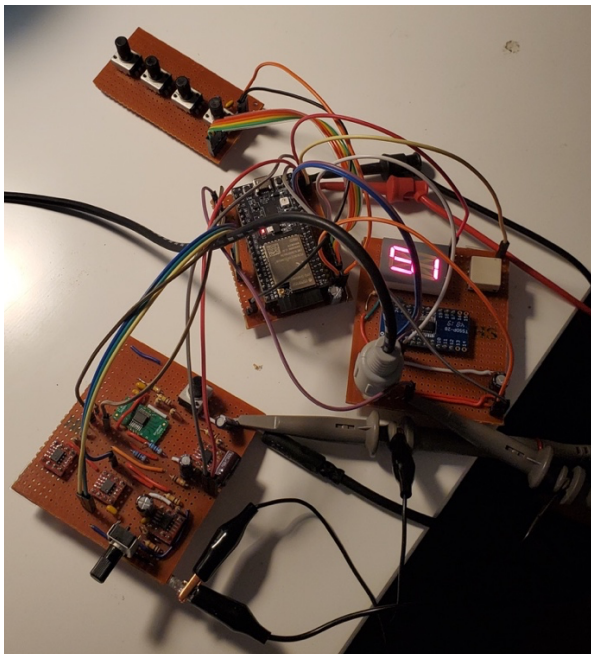


FIGURE 40: SETUP FOR MEASURING MIDI INPUT AND AUDIO OUTPUT

The horizontal markers were used to find the delta time between the end of the MIDI packet and the start of the output waveform. 17.2ms was recorded as the latency.

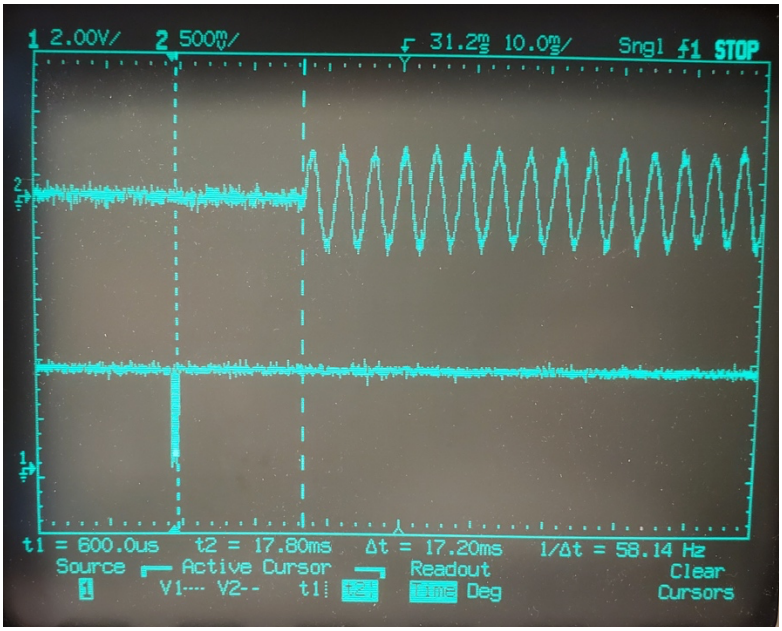


FIGURE 41: MEASUREMENT OF MIDI TO AUDIO LATENCY

Design Specification: MIDI "note on" command to audio output latency less than 100ms
Measured: 17.2ms
DUT PASSES

6.5 SIGNAL TO NOISE RATIO

To measure the signal-to-noise-ratio [SNR] of the synthesizer, voltage measurements of the “headphone” output were made with a note being played and without.

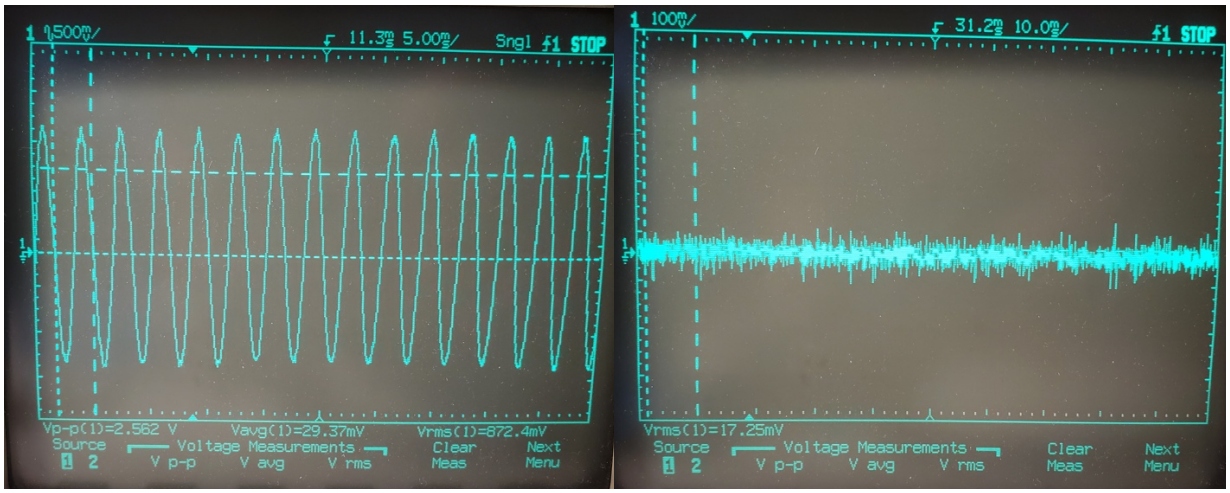


FIGURE 42: MEASUREMENT OF NOTE ON AND NOTE OFF VOLTAGE AMPLITUDE

The signal to noise ratio was then calculated as a ratio between these measurements and converted to log voltage.

$$V_{note\ on} = 872.4\ mVrms$$

$$V_{note\ off} = 17.25\ mVrms$$

$$SNR = 10\ log\left[\frac{signal}{noise}\right]$$

$$10\ log\left[\frac{872.4\ mVrms}{17.25\ mVrms}\right] = 17\ dB$$

Unfortunately, this measurement is due to the noise floor of the HP54645A Oscilloscope and not the noise floor of the DUT. Measuring the noise of the testing setup directly by connecting the probe to the ground net of the DUT showed the presence of 20.5mVrms of noise. This indicates that the Oscilloscope does not have a low enough internal noise floor to measure the signal to noise ratio of the DUT. Due to Covid 19 protocols I was unable to access an audio signal analyzer to obtain a valid signal to noise ratio.

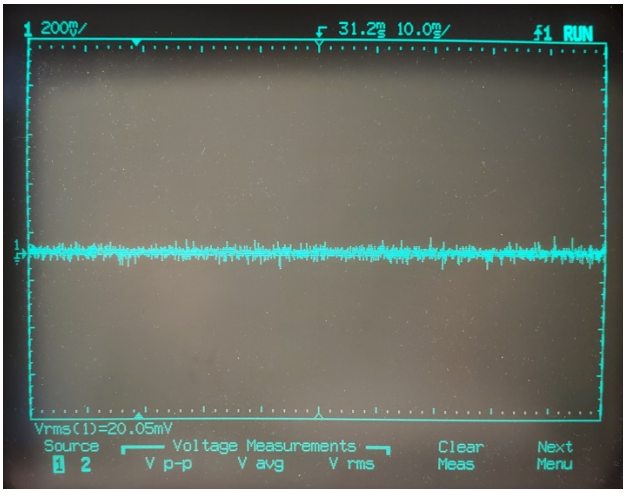


FIGURE 43: HP54645A OSCILLOSCOPE NOISE WITH INPUT GROUNDED

Design Specification: Noise at headphone output at least 60dB less than "note on" amplitude
Measured: NO MEASUREMENT

SECTION 7 - CONCLUSIONS and FUTURE WORK

Completing this senior project over the last three quarters has widened my familiarity with design in multiple ways. Working through the project from a framework of designing for a customer helped to structure and focus my design before I even specified the electronics. Without these steps I do not believe the final design would be as cost effective and efficient as it is in final form. Looking at the entire lifecycle from construction to end of life considerations allowed a clearer view of the project's impact on its user and the environment.

As a learning experience, this project presented a plethora of electrical and computer engineering challenge. On a more technical level, it provided an opportunity to become more familiar with low voltage analog electronics engineering as well as design for cost and manufacturing. Designing a device with analog and digital systems presented real world hybrid electronic design problems. Building a consumer synthesizer required simultaneous consideration of noise, sensitive analog systems, power draw requirements, and an overarching cost issues.

Moving forward with this project would see the polishing of the product into a sale ready unit. The architecture and software are essentially complete, so PCB layout and component sourcing would be the next step. If the year was not riddled with component supply and PCB manufacturing issues from the COVID-19 pandemic, a more final version of the synthesizer would have likely been possible. It also feels pertinent to mention that in the future the Electrical Engineering department would benefit from acquiring an audio analyzer for these types of projects. I was unable to test the noise floor nor obtain accurate spectrum analysis of my synthesizers output due to a lack of any audio range lab equipment.

SECTION 8 – AUDIO DEMONSTRATION

This project is at its base an instrument. It would be a loss if this report did not include an example of the synthesizer doing what it was designed and constricted to do, play music! The permanent link below leads to an uploaded recording of the synthesizer reciting a MIDI transcription of Claud Debussy's Clair de Lune. The MIDI file was sent to the synthesizer via a USB to MIDI interface. Other than the addition of outboard reverb, all the audio in this demonstration was generated in real time by the hybrid audio synthesizer.

https://gymprofessor.bandcamp.com/track/senior-project-demo-clair-de-lune

SECTION 9 - REFERENCES

[1] B. John, *The Synthesizer*. Oxford: Oxford University Press, 1988.

[2] A. J. Evans, *Making Sense of Sound: The Basics of Audio Electronics and Technology*. Rev. 1st ed. Indianapolis, IN: PROMPT Publications, 1992.

[3] MIDI Manufacturers Association, *The Complete MIDI 1.0 Detailed Specification Incorporating all Recommended Practices,* MIDI Manufacturers Association, 1995 [3rd Revision 2014].

[4] D. P. Rossum, "Digital sampling instrument employing cache memory," United States Patent 5698803A, Feb. 28, 1994.

[5] B. Wright and S. Sukittanon, "Integer-based wavetable synthesis for low-computational embedded systems,"

[6]"teenage engineering," *pocket operators*. [Online]. Available: https://teenage.engineering/products/po. [Accessed: 03-Nov-2020].

[7] M. S. Ansari and M. Z. Khan, "Digitally programmable first-order current mode continuous-time filters," *2012 2nd International Conference on Power, Control and Embedded Systems*, Allahabad, 2012, pp. 1-6, doi: 10.1109/ICPCES.2012.6508039.

[8] U. Gaume Echeverría, F. E. Guerrero Castro and J. M. D. Báez López, "Comparison between a Hardware and a software synthesizer," 2010 20th International Conference on Electronics Communications and Computers (CONIELECOMP), Cholula, 2010, pp. 311-314, doi: 10.1109/CONIELECOMP.2010.5440747.

[9] Espressif Systems,, "ESP32 Series datasheet," ESP32 datasheet, 2020.

[10] Microchip, "8-/10-/12-Bit Single/Dual Voltage Output Volatile Digital-to-Analog Converters with SPI Interface," MCP48FVBXX datasheet, 2015.

[11] Jithendra, R. Santos, D. French, J. Young, idan Ben-Moshe, E. Gongora, Michael, K. S, S. Santos, Franklin, Kevin, RobinBlood, I. Gheorghe, M. Hortal, Saber, Omemanti, Safalya, T. kumar, D. Tuttle, M. McKinney, T. Hollifield, K. Jha, Shimul, T. Chen, and Bob, "Getting Started with the ESP32 Development Board," *Random Nerd Tutorials*, 03-Jun-2020. [Online]. Available: https://randomnerdtutorials.com/getting-started-with-esp32/. [Accessed: 06-Nov-2020].

[12] "Prototype PCB - Online PCB Quote - Full feature custom PCB prototype service at low cost - PCBWay," *Custom PCB Prototype Manufacturer*. [Online]. Available: https://www.pcbway.com/orderonline.aspx. [Accessed: 06-Nov-2020].

[13] *Get Started*. 2020, docs.espressif.com/projects/esp-idf/en/latest/esp32s2/get-started/.

[14] Maxim Integrated, “Dual Universal Switched-Capacitor Filters,” 19-1768, 2009.

[15] Texas Instruments, “LM1971 OvertureTMAudio Attenuator Series Digitally Controlled 62 dB Audio Attenuator with/Mute,” SNAS104B, 1995.

[16] Kingbright, “PSA08-11SRWA 20.32mm (0.8inch) 16 Segment Single Digit Alphanumeric Display,” DSAP8309 / 1311000038, 2020.

[17] Texas Instruments, “TLC59283 16-Channel, Constant-Current LED Driver with Pre-Charge FET,” SBVS199B, 2012.

[18] “Analog to Digital Converter.” *ESP*, Espressif, 2016, docs.espressif.com/projects/esp-idf/en/latest/esp32s2/api-reference/peripherals/adc.html#_CPPv411adc_atten_t.

[19] Texas Instruments, “LM2623General-Purpose, Gated-Oscillator-BasedDC-DCBoost Converter,” SNVS188I, REVISED 2017.

[20] Texas Instruments, “OPA168x SoundPlus 36-V, Single-Supply, 10-MHz, Rail-to-Rail Output Operational Amplifiers,” SBOS724, 2015

APPENDIX A. ABET SENIOR ANALYSIS

Project Title: Low Cost Hybrid Music Synthesizer

Student's Name: Spencer Drewry

Student's Signature:

1. Summary of Functional Requirements

The 'Low Cost Hybrid Music Synthesizer' is an inexpensive and versitile music synthesizer module. Building on a history of affordable pro audio equipment, the project will provide a flexible and easy to use interface to a powerful hybrid paraphonic synthesizer in a low cost, portable formfactor. The device provides standard line level, headphone, and MIDI connectivity and an easy to understand layout and interface.

2. Primary Constraints

This project has posed several challenges so far. Initially, I ran into the challenge of technically defining a consumer musical instrument effectively. Price is a limiting factor as the end user is in the consumer electronics market and expects devices to be cheap. This meant that the design has needed to be economically efficient with the way it satisfies its design requirements. Is there a simpler way to provide the same experience for the user for less cost/parts?

3. Economic

Gantt Chart and Price/Labor estimates are included in appendix B.

The economic impacts of the project are small in comparison to other consumer or industrial electronics projects. The quantity I plan to produce is less than 1000 units, the methods of which the device will be produced are standard, and the economic profits less than $20,000 a year. Regardless, attention must be taken to the companies and people providing both the components, PCB assembly, and production abilities. Their economic, human, and environmental impacts are directly driven, at least partially, by employing their services to manufacture this project.

As mentioned, the project has a relatively small impact as the reach is constrained effectively. If the device were to be produced it would be produced by myself solely and sold in relatively small quantities. Following the previously calculated profit estimations, it is clear that the net income the device would generate is modest even if collected a single person.

The device is purely artistic in nature, provides no real use beyond it's value to an artist. Therefore, the economic impact of its price, production, etc, is minimal on the end consumer. The benefits are defined by the end user. This also means that the projects value is accrued most

entirely in the design phase. The devices functionality and design determine its value as a musical instrument. Not necessarily the device solving a problem.

With regards to timing, the device is entering a relatively untapped market, and thusly should be expedited. I would expect that as a real consumer product it would likely be well within its window if released before the end of 2021. This feels like a viable timeline for the project as the manufacturing takes advantage of already well established and well-defined processes. One the design is completed; devices could be ready for sale within a month.

After the project ends, I hope the project not only satisfies the ABET requirements of our senior project, but the device is completed enough to prove my merit as both a practical and artistically minded electrical engineer. I hope to use the project to "sell" myself to companies that see merit in the creative side of circuits and electrical engineering.

4. If manufactured on a commercial basis:

The manufacturing of the device on a commercial basis is directly linked to PCB manufacturing services and lead times. As the device is in essence a raw PCB, other manufacturing processes for an enclosure or other parts are not required. This leads to the only manufacturing bottleneck being associated with the board house and availability of components.

At relatively low quantities (100 or less), the PCB manufacturer should have no problem supplying the product at standard lead times. The components are all standard and supplied in large quantities, and the PCB's can be panelized and produced at a high rate.

Estimated Number of Devices Sold (Annually)	Estimated Manufacturing Cost (Per Device)	Estimated Purchase Price	Estimated Profit Per Year
500	$60.00	$100.00	$20,000

If the device were to reach sales above the manufacturing ability of one board house or IC supplier, it would be very simple to employ other companies to supply PCB's in parallel. In this case, quality control may become an issue if different standards are followed by each manufacturer. This could be mitigated by ensuring standard PCB substrates and solder masks are used by each company.

5. Environmental

The environmental impact of the project is that of a consumer electronic device. The components, PCB, and manufacturing all carry environmental issues. The standard electronic components require the mining of heavy metals including lead, cadmium and nickel and use significant amounts of both water and energy in their production [1]. The PCB manufacturing and assembly process also uses nonrenewable resources and consumes both water and energy. There are also harsh chemicals and industrial waste associated with PCB manufacturing. Etching PCB substrates requires caustic compounds like copper sulfate and sulfuric acid [1].

The device also requires batteries to function. If these cells are non-rechargeable they will add to the waste generated by the devices life. One method of reducing this waste flow would be

to ship the device with rechargeable cells and provide a way of recharging aforementioned cells with the device.

Initial low quantity manufacturing will likely be completed with overseas board houses as their turnaround times and cost points are significantly better than US board houses. Once a production run is reached, the device will be manufactured in the United States as large-scale production overseas leverages relaxed environmental standards. Despite the increased cost associated with local manufacturing, an increased production quantity should decrease the per unit cost to a price similar to low quantity production overseas.

6. Manufacturability

The main challenge for the manufacturing is related to the devices lack of enclosure. To keep the price of the device low, and the use of material low as well, the PCB will be the main frame, circuit, and interface of the device. This means that the PCB must carry all the physical load as well as all electronic functionality. To accomplish this, special measures must be taken in layout and design.

It is also important to ensure the components, processes, and final product are lead and heavy metal free. The entire process must be ROHS compliant as the end user will be in physical contact with the PCB [2].

7. Sustainability

The device poses one main issue when it comes to sustainability. Because it is a consumer device, and is built to a price point, it is important to consider the inevitably short lifecycle of the device. The components and manufacturing techniques sacrifice some component and process quality for a decrease in per device cost. The device uses preexisting user electronic pathways and resources in its manufacturing and distribution, and therefore buys into the sustainability issues that already plague user electronics. The device is environmentally costly to manufacture, and environmentally costly to dispose of.

Some of these impacts could be mitigated by redesigning the device and the process in which it is manufactured. For example, the lifetime of the product could be increased by investing in higher quality components. Especially with buttons, potentiometers, and connectors an increase in quality would lead to a longer device lifetime.

Unfortunately, as mentioned before it is imperative that the project is priced competitively, or it does not fit in the market. Upgrading some components may be viable, but a complete redesign to include all of the highest quality components would surely push the devices cost far above the market price point.

8. Ethical

Because the device is of a consumer, artistic, nature the ethical implications of its use are relatively unproblematic. The use of the device is entirely determined by the user.

The main ethical framework I believe impacts this project is captured in the first IEEE ethics statement:

> 1. To hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, to protect the privacy of others, and to disclose promptly factors that might endanger the public or the environment

It's paramount that the device is safe to the user without and enclosure. As mentioned previously, the PCB must be safe physically for the user to be in contact with. All the components must be ROHS compliant and the PCB must be free of any materials that would be unsafe to the end user. It is my responsibility as the designer to ensure that the manufacturing of the product adheres to these standards.

IEEE ethical framework 1 also outlines the importance of minimizing the ecological impact of the device. It is my responsibility as the designer to ensure that the components and PCB processes used in producing the product adhere to modern standards. The components must be ethically sourced, and the PCB manufacturing process must be as responsible or more responsible ecologically.

The second IEEE ethics statement I found connected with the project is number five:

> 5. to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, to be honest and realistic in stating claims or estimates based on available data, and to credit properly the contributions of others

It is very common in audio devices to see the uncredited use or adaptation of historic circuits. Following the IEEE ethics code, I am working towards the goal of crediting effectively all the inspirations of the final device. Despite the commonplace nature of mimicking and cloning filter and amplifier circuits, I pledge to appropriately modify and attribute any designs I take inspiration from.

9. Health and Safety

The health and safety of the manufacturers is ensured by following standard practice with design and by using trustworthy and transparent manufactures with vetted policy for their workers and processes.

As mentioned previously, the device MUST be safe for the end user. Electrically, no voltages are present above 6Vdc, and therefore do not create a safety risk for the end user [3]. Any misuse of the device is unlikely to lead to harm as it is artistic in nature and a failure does not lead to a dangerous situation.

10. Social and Political

The social and political impacts of the project once again mirror those of the consumer electronics market. The manufacturing of the PCB's, integrated circuits, and other electronic components impacts the economies and structures built into the overseas electronics market. It is extremely important to ensure that educated choices in manufacturers and suppliers are made to ensure the indirect exploitation of workers or communities is avoided.

11. Development

The development of this project has been a learning experience for me as an engineer. I have implemented many new tools and processes throughout the design process so far. Managing a project as a system has improved the scope of my planning skills. Managing design requirements, specifications, and cost and time estimations has allowed me to be much more confident in the scope and process required to complete a project.

I think one of the most standout design techniques I have independently learned has been the use of the PCB as the entire structure of the product. This process allows a significant cut in manufacturing cost as no extra tooling, parts, or labor is required in the assembly of the device. The method is still relatively small in the consumer market due to some pitfalls [4]. The use of PCB manufacturing processes to create non formal designs, often mechanical or purely aesthetic parts is a relatively unexplored field that I would like to include in the project.

REFERENCES

[1] “Guides to Pollution Prevention The Printed Circuit Board Manufacturing Industry.” Risk Reduction Engineering Laboratory Center for Environmental Research Information, Cincinnati, OH, Jun-1990. https://www.osti.gov/biblio/6535725-guides-pollution-prevention-printed-circuit-board-manufacturing-industry

[2] “RoHS Guide,” *RoHS 2 vs RoHS 3 (EU 2015/863)*, 23-Nov-2020. [Online]. Available: https://www.rohsguide.com/rohs3.htm.

[3] T. Galassi, “Standard Interpretations / Guarding requirements for 50 volts or more DC,” *Occupational Safety and Health Administration*. [Online]. Available: https://www.osha.gov/laws-regs/standardinterpretations/2015-09-04. [Accessed: 23-Nov-2020].

[4] L. Day, “The Benefits And Pitfalls Of Using PCBs As An Enclosure,” *Hackaday*, 16-Sep-2019. [Online]. Available: https://hackaday.com/2019/09/16/the-benefits-and-pitfalls-of-using-pcbs-as-an-enclosure/. [Accessed: 23-Nov-2020].

APPENDIX B. PROJECT PLANNING

ANALYSIS

The timeline of this project relies on viable iteration of the circuit design. As the system is a combination of analog and digital signals, or mixed signal, meeting noise and stability requirements will require careful physical and electrical design that will likely require a significant iteration. This is reflected in both the Gantt chart (figure 3) and the cost estimates (table 4).

The Gantt chart assumes the use of 2-week turnaround PCB manufacturing as well as worst case project layout estimates. It is also important to mention that this project does have a significant amount of embedded programming required. As there is a large codebase for the proposed microcontroller, the ESP32, the programming timeline can be shortened to a manageable timeline for a single year development [11].

The project cost is based on estimated component, assembly, and manufacturing costs. I determined the manufacturing costs assuming the use of PCBway's fast turnaround PCB and turnkey population services [12]. This also assumes that the most expensive components include the microcontroller, DAC's, and OP amps.

GANTT CHART

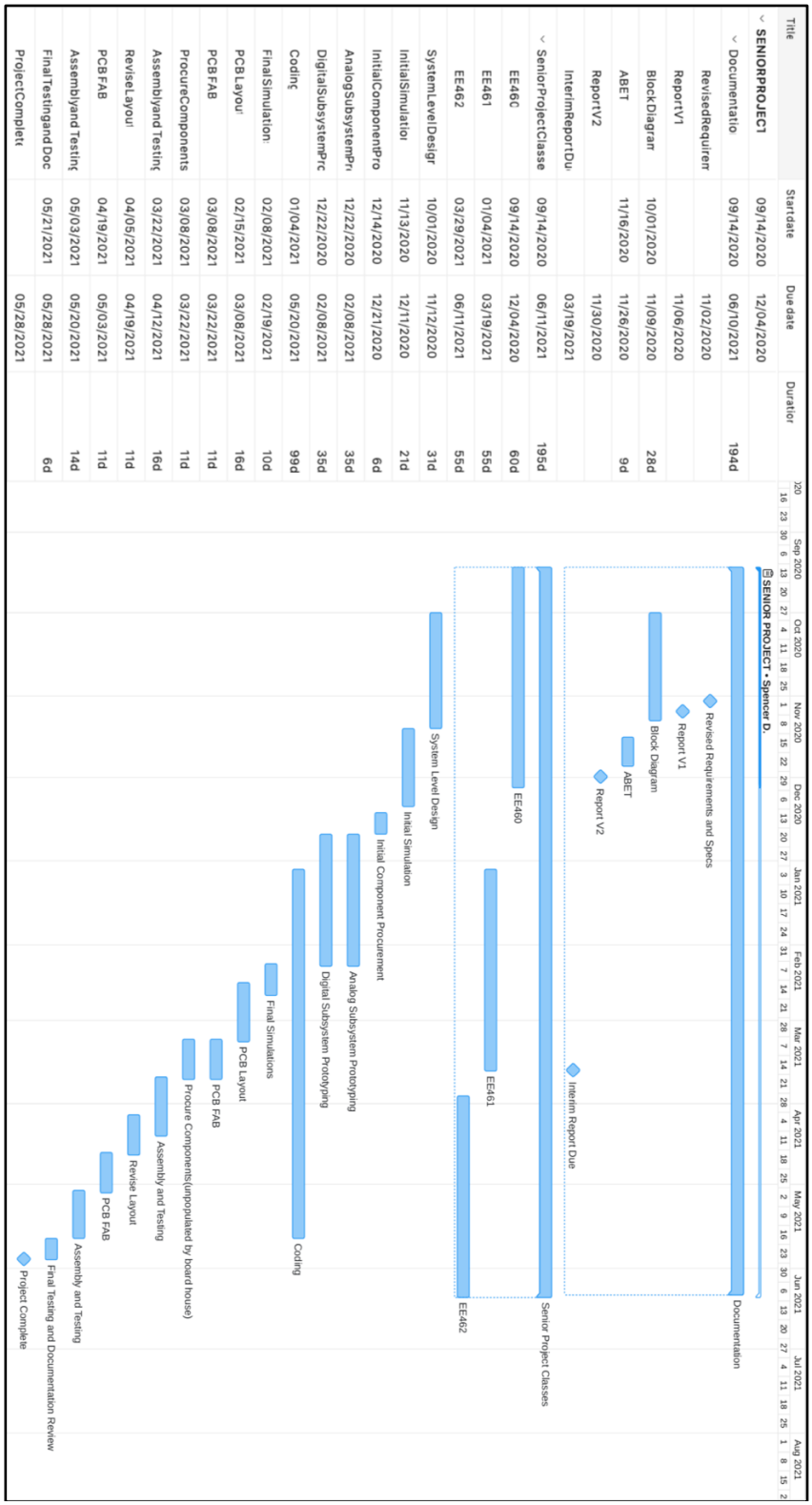


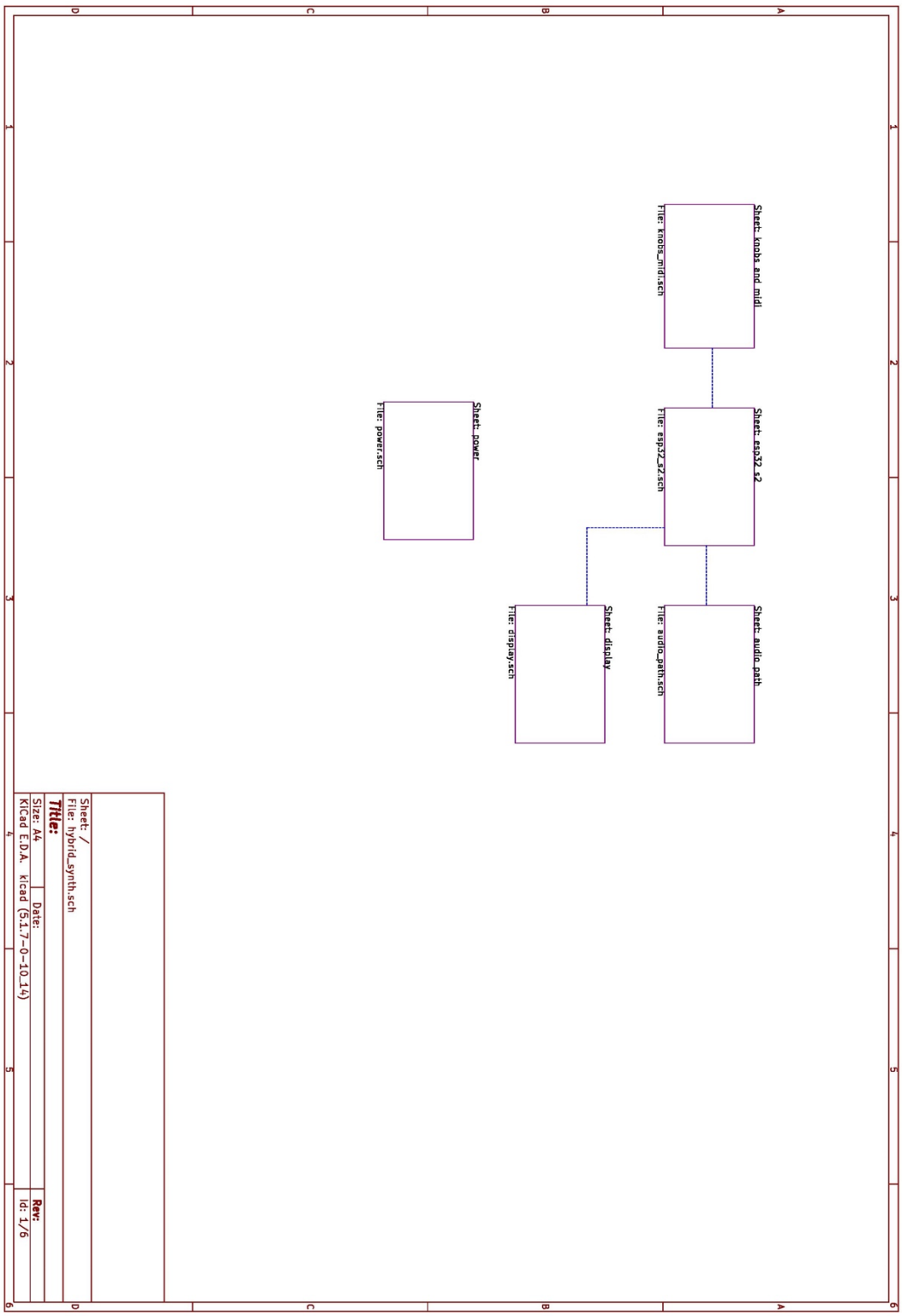
FIGURE 44: PROJECT TIMELINE GANTT CHART

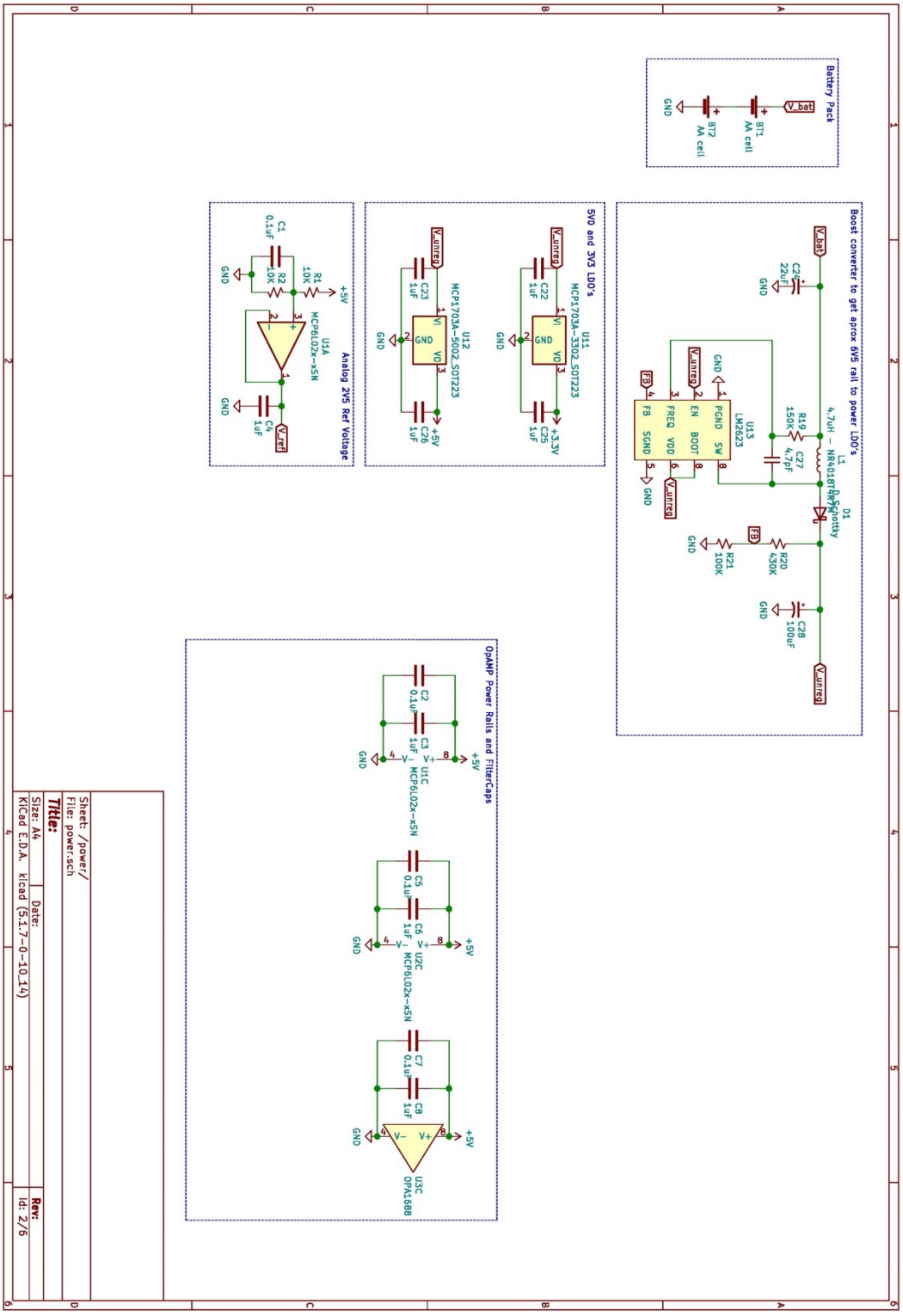
COST ESTIMATES

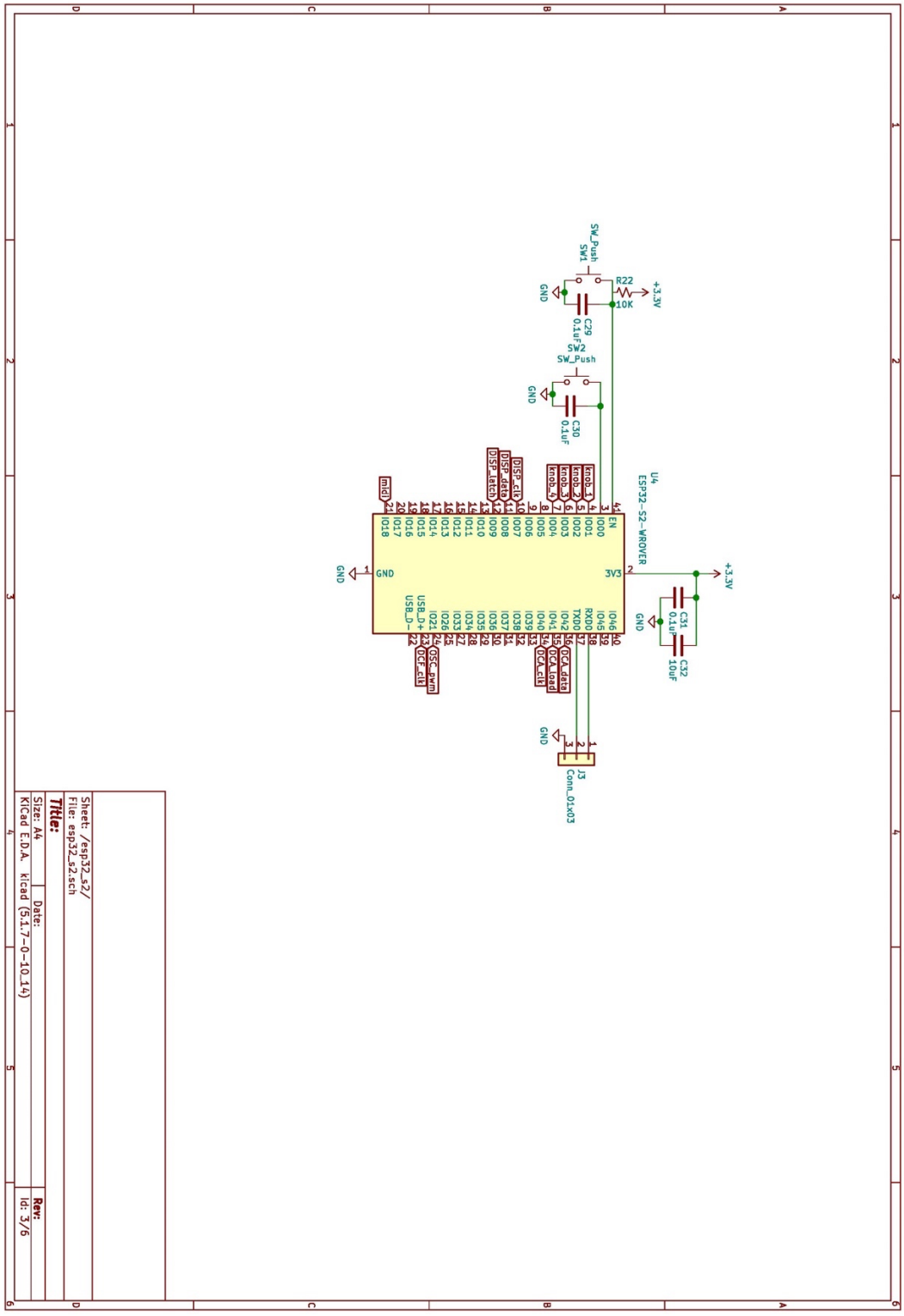
TABLE IV
COST ESTIMATES

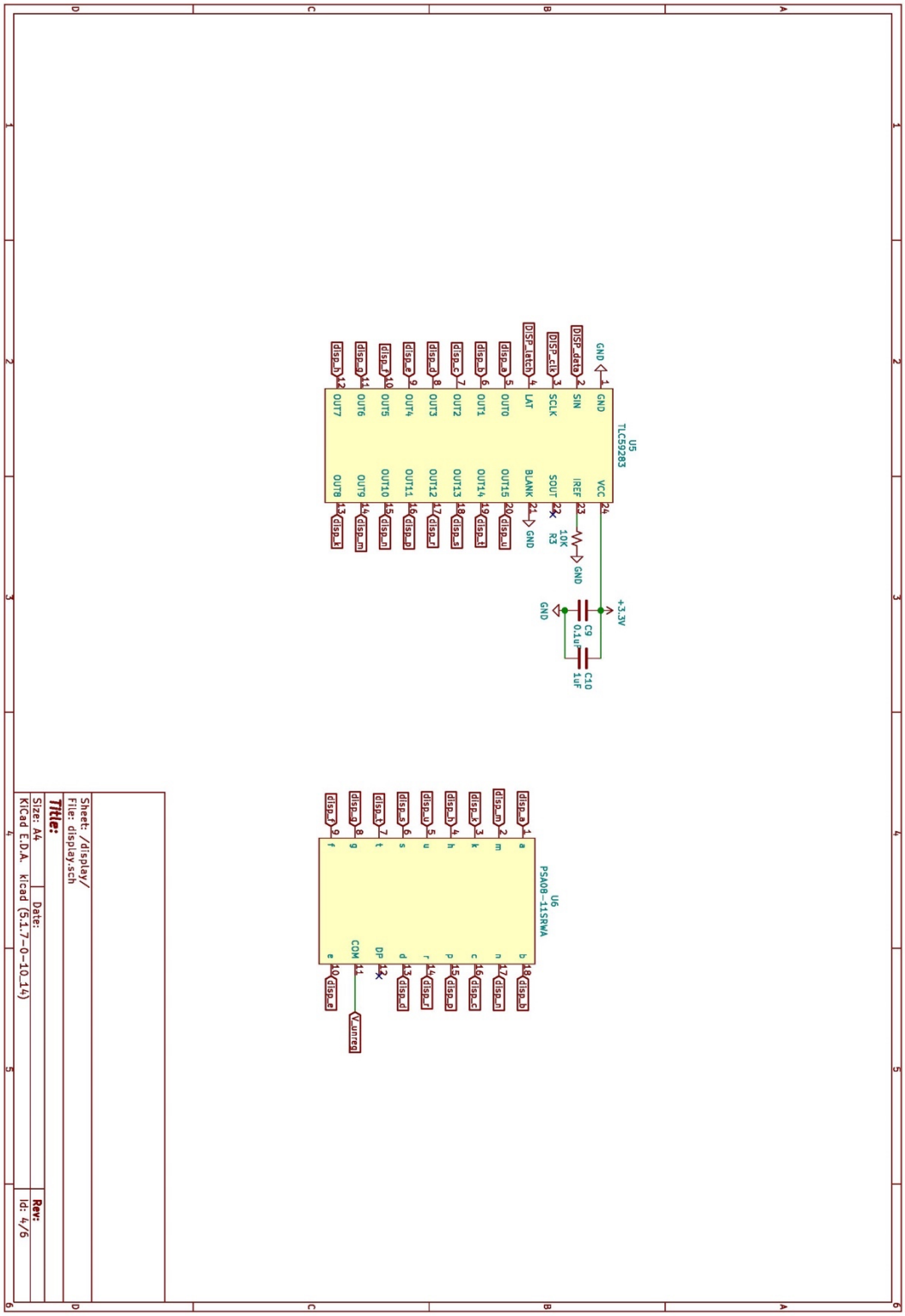
Task	Best Case (hrs)	Likely Case (hrs)	Worst Case (hrs)	Justification	Hours	Approx. Cost
Initial Design and Schematic	20	25	50	Analog design pretty simple after block diagram etc, initial simulations are done	28.33	$567
Simulation and Revision	30	40	100	This takes more time, revising and doing sensitivity analysis on initial circuit	48.33	$967
Prototyping	20	50	100	This could go fast if the initial design works as intended, or could require a redesign of some portion	53.33	$1,067
PCB Layout	50	80	150	I have quite a bit of layout practice, but this will take a long time due to the PCB being integral to the devices structure as well as mixed signal routing	86.67	$1,733
Assembly/Testing	50	60	100	This once again could be a few days or a few weeks of work depending on what issues arise	65.00	$1,300
					Total:	$5,633
Item	**Best Case (USD)**	**Likely Case (USD)**	**Worst Case (USD)**			**Approx. Cost**
Initial Components for Prototyping	50	80	150	This is determined from adding the more expensive component costs as well as a dev kits for esp32		$113
PCB Manufacturing/Population V1	40	60	200	Standard low quantity order from PCB way		$100
PCB Manufacturing/Population V2	40	60	200	Standard low quantity order from PCB way		$100
PCB Components for Turnkey	100	225	300	Estimated component cost for 4 populated PCB's assuming digikey and standard passive component prices		$292
					Total:	$605

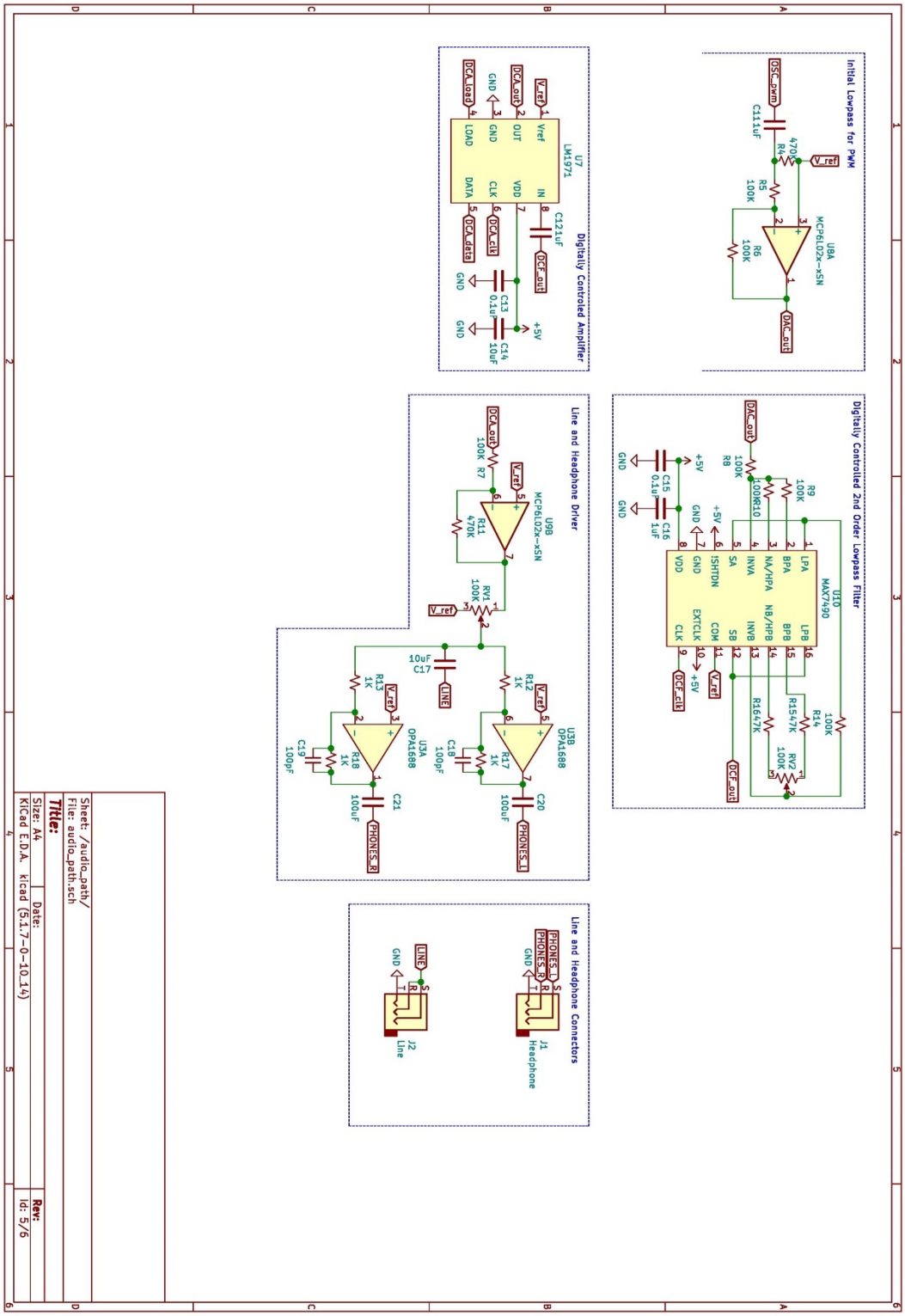
APPENDIX C. SCHEMATICS

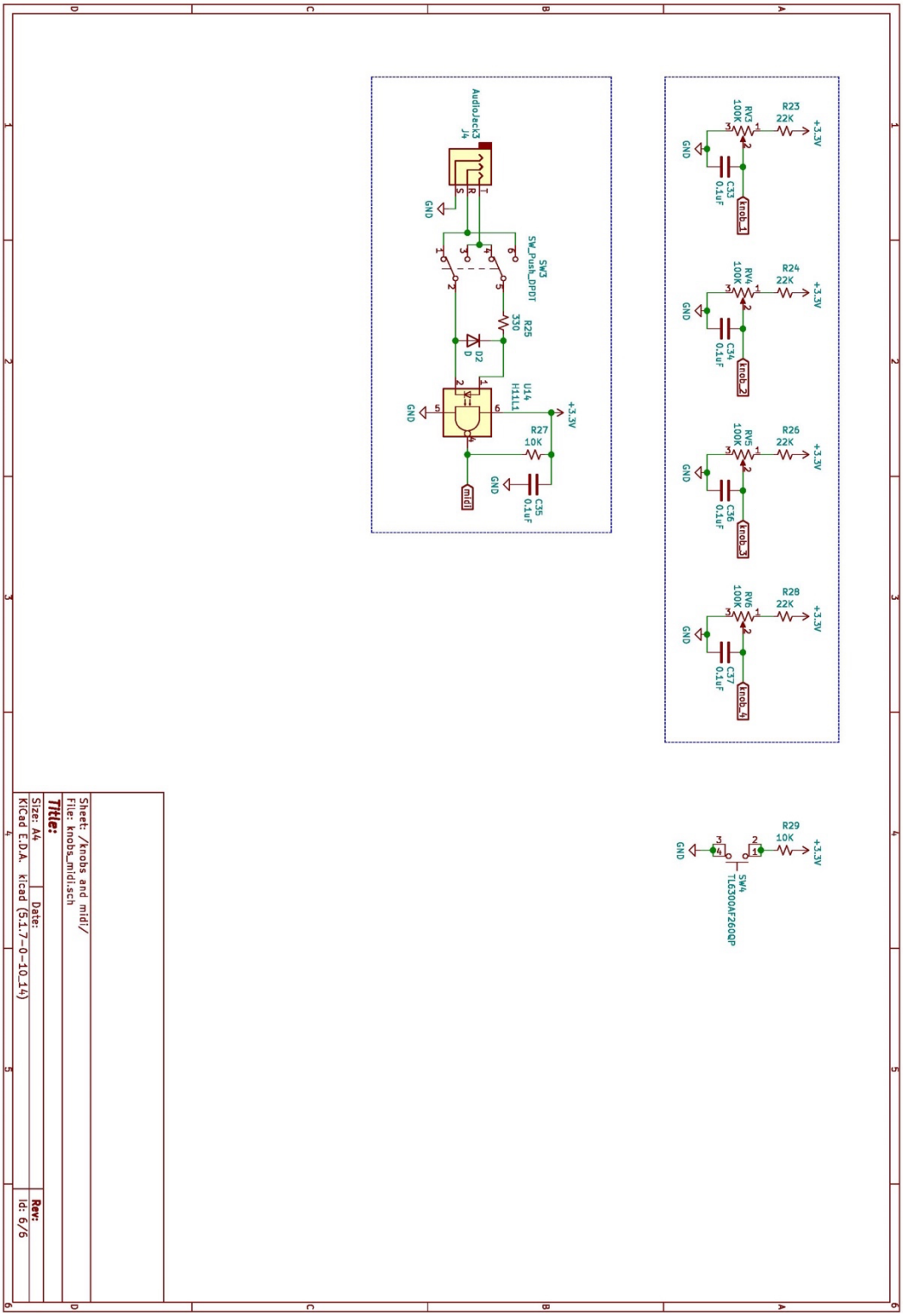



Battery Pack
V_bat
BT1
AA cell
BT2
AA cell
GND
Boost converter to get aprox 6V5 rail to power LDO's
V_bat
C24
22uF
GND
4.7uH – NR4018T4R7N
L1
D1
PGND
SW
EN
BOOT
FREQ
VDD
FB
SGND
U13
LM2623
V_unreg
R19
150K
C27
4.7pF
R20
430K
R21
100K
C28
100uF
5V0 and 3V3 LDO's
U11
MCP1703A-3302_SOT223
C22
1uF
C25
1uF
+3.3V
U12
MCP1703A-5002_SOT223
C23
1uF
C26
1uF
+5V
Analog 2V5 Ref Voltage
U1A
MCP6L02x-xSN
C1
0.1uF
R1
10K
R2
10K
C4
1uF
V_ref
+5V
OpAMP Power Rails and FilterCaps
U1C
MCP6L02x-xSN
C2
0.1uF
C3
1uF
U2C
MCP6L02x-xSN
C5
0.1uF
C6
1uF
U3C
OPA1688
C7
0.1uF
C8
1uF
Sheet: /power/
File: power.sch
Title:
Size: A4
Date:
KiCad E.D.A. kicad (5.1.7-0-10_14)
Rev:
Id: 2/6


SW_Push
SW1
R22
10K
+3.3V
GND
C29
0.1uF
SW2
SW_Push
C30
0.1uF
U4
ESP32-S2-WROVER
EN
GND
3V3
+3.3V
C31
0.1uF
C32
10uF
J3
Conn_01x03
Sheet: /esp32_s2/
File: esp32_s2.sch
Title:
Size: A4
Date:
KiCad E.D.A. KiCad (5.1.7-0-10_14)
Rev:
Id: 3/6


U5
TLC59283
GND
DISP_data
DISP_clk
DISP_latch
SIN
SCLK
LAT
OUT0
OUT1
OUT2
OUT3
OUT4
OUT5
OUT6
OUT7
VCC
IREF
SOUT
BLANK
OUT15
OUT14
OUT13
OUT12
OUT11
OUT10
OUT9
OUT8
10K
R3
C9
0.1uF
C10
1uF
+3.3V
U6
PSA08-11SRWA
COM
DP
V_unreg
Sheet: /display/
File: display.sch
Title:
Size: A4
Date:
KiCad E.D.A. kicad (5.1.7-0-10_14)
Rev:
Id: 4/6


Initial Lowpass for PWM
OSC_pwm
C111uF
470K R4
V_ref
R5 100K
R6 100K
U8A MCP6L02x-xSN
DAC_out
Digitally Controled Amplifier
U7 LM1971
DCA_out
DCA_load
DCA_clk
DCA_data
DCF_out
C121uF
C13 0.1uF
C14 10uF
+5V
GND
Digitally Controlled 2nd Order Lowpass Filter
U10 MAX7490
DAC_out
100K R8
100K R10
R9 100K
C15 0.1uF
C16 1uF
LPA
BPA
MA/HPA
INVA
SA
!SHTDN
GND
VDD
NB/HPB
INVB
SB
COM
EXTCLK
CLK
R14 100K
R15 47K
R16 47K
RV2 100K
DCF_clk
Line and Headphone Driver
U9B MCP6L02x-xSN
R7 100K
R14 470K
RV1 100K
C17 10uF
LINE
R13 1K
R18 1K
C19 100pF
U3A OPA1688
C21 100uF
PHONES_R
R12 1K
R17 1K
C18 100pF
U3B OPA1688
C20 100uF
PHONES_L
Line and Headphone Connectors
PHONES_L
PHONES_R
J1 Headphone
J2 Line
Sheet: /audio_path/
File: audio_path.sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (5.1.7-0-10.14)
Id: 5/6


RV3
100K
R23
22K
+3.3V
GND
C33
0.1uF
knob_1
RV4
100K
R24
22K
+3.3V
GND
C34
0.1uF
knob_2
RV5
100K
R26
22K
+3.3V
GND
C36
0.1uF
knob_3
RV6
100K
R28
22K
+3.3V
GND
C37
0.1uF
knob_4
AudioJack3
J4
GND
SW3
SW_Push_DPDT
R25
330
D2
D
U14
H11L1
+3.3V
GND
R27
10K
GND
C35
0.1uF
midi
R29
10K
+3.3V
GND
SW4
TL6300AF260QP
Sheet: /knobs and midi/
File: knobs_midi.sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (5.1.7-0-10_14)
Id: 6/6

APPENDIX D. PARTS LISTING

Manufacturer Part Number	Manufacturer	Digi-Key Part Number	Quantity	Unit Price	Extended Price	Description
PTV09A-4020U-B104	Bourns Inc.	PTV09A-4020U-B104-ND	6	0.83	$4.98	POT 100K OHM 1/20W CARBON LINEAR
H11L1S(TA)	Everlight Electronics Co Ltd	1080-1201-1-ND	1	0.76	$0.76	OPTOISO 5KV OPEN COLLECTOR 6SMD
TLC59283DBQ	Texas Instruments	TLC59283DBQ-ND	1	0.76	$0.76	IC LED DRIVER LINEAR 45MA 24SSOP
PSA08-11SRWA	Kingbright	754-1675-5-ND	1	3.38	$3.38	DISPLAY 16SEG 0.8" SGL RED 18DIP
LM1971MX/NOPB	Texas Instruments	LM1971MX/NOPBCT-ND	1	2.06	$2.06	IC VOLUME CONTROL 8SOIC
OPA1688IDR	Texas Instruments	296-47272-1-ND	1	1.65	$1.65	IC OPAMP GP 2 CIRCUIT 8SOIC
MAX7490EEE+	Maxim Integrated	MAX7490EEE+-ND	1	7.88	$7.88	IC FILTER 40KHZ SWITCHED 16QSOP
TL6300AF260QP	E-Switch	EG6117-ND	1	0.34	$0.34	TACT SWITCH THRU HOLE 12MM X 12M
B32-1300	Omron Electronics Inc-EMC Div	SW454-ND	1	0.32	$0.32	CAP TACTILE SQUARE IVORY
ESP32-S2-WROVER	Espressif Systems	1965-ESP32-S2-WROVERCT-ND	1	2.4	$2.40	RX TXRX MOD WIFI SURFACE MOUNT
MCP6L02T-E/SN	Microchip Technology	MCP6L02T-E/SNCT-ND	3	0.31	$0.93	IC OPAMP GP 2 CIRCUIT 8SOIC
CL10A105KA8NNNC	Samsung Electro-Mechanics	1276-1102-1-ND	10	0.036	$0.36	CAP CER 1UF 25V X5R 0603
RR0816P-104-D	Susumu	RR08P100KDCT-ND	25	0.085	$2.12	RES SMD 100K OHM 0.5% 1/16W 0603
CL32A107MPVNNNE	Samsung Electro-Mechanics	1276-3364-1-ND	2	1.19	$2.38	CAP CER 100UF 10V X5R 1210
SJ1-3533	CUI Devices	CP1-3533-ND	3	1.6	$4.80	CONN JACK STEREO 3.5MM R/A
LM2623AMM/NOPB	Texas Instruments	LM2623AMM/NOPBCT-ND	1	1.69	$1.69	IC REG BOOST ADJ 2.2A 8VSSOP
2460	Keystone Electronics	36-2460-ND	2	1.24	$2.48	BATTERY HOLDER AA PC PIN
MCP1703-3302E/DB	Microchip Technology	MCP1703-3302E/DB-ND	1	0.65	$0.65	IC REG LIN 3.3V 250MA SOT223-3
MCP1703T-5002E/DB	Microchip Technology	MCP1703T-5002E/DBCT-ND	1	0.65	$0.65	IC REG LINEAR 5V 250MA SOT223-3
JS202011JAQN	C&K	CKN10722CT-ND	1	0.58	$0.58	SWITCH SLIDE DPDT 300MA 6V
NR4018T4R7M	Taiyo Yuden	587-1671-1-ND	1	0.41	$0.41	FIXED IND 4.7UH 1.2A 108 MOHM
FSMSM	TE Connectivity ALCOSWITCH Switches	450-1140-ND	1	0.28	$0.28	SWITCH TACTILE SPST-NO 0.05A 24V
SD0805S020S1R0	AVX Corporation	478-7800-1-ND	1	0.38	$0.38	DIODE SCHOTTKY 20V 1A 0805
SD0805S020S1R0	AVX Corporation	478-7800-1-ND	1	0.38	$0.38	DIODE SCHOTTKY 20V 1A 0805

FINAL PER UNIT PART COST: $42.61 (at low order volume)

APPENDIX E. FULL SOFTWARE LISTING

```
#include <stddef.h>
#include <math.h>
#include <driver/adc.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_intr_alloc.h"
#include "esp_attr.h"
#include "driver/timer.h"
#include "driver/gpio.h"
#include "driver/ledc.h"
#include "esp_err.h"
#include "driver/sigmadelta.h"
#include "driver/uart.h"

// IO for waveform mode button
#define MODE_BUTTON 17

// UART for midi
#define ECHO_TEST_TXD 19
#define ECHO_TEST_RXD 18
#define ECHO_TEST_RTS UART_PIN_NO_CHANGE
#define ECHO_TEST_CTS UART_PIN_NO_CHANGE

#define ECHO_UART_PORT_NUM      UART_NUM_1
#define ECHO_UART_BAUD_RATE     31250

#define BUF_SIZE 128

#define PI 3.14159265
#define TIMER_PIN_1 5   //pin for timing isr duration
#define TIMER_PIN_2 6   //pin for timing isr duration

#define DCA_DAT 42      //pins for dca 3wire interface
#define DCA_CLK 41
#define DCA_LAT 40

#define DIS_DAT 6
#define DIS_CLK 7
#define DIS_LAT 8

#define FLTR_CLK_PIN 20 //clock output for DCF

#define TMR_TICK_PER_US 0.25    //calulate and place the TMR tick duration in us here (required
for acurate smple timing)
```

```
#define SAMPLE_PER_US 50        //sample rate period (sets the sample rate when timer reg setup
happens)

#define MAX_ACTIVE_NOTES 10     //max notes before rolover (this must be set to not write over
DAC ISR period)

#define MULTISAMP 50

// note struct to easily manage polyphony
struct note {
    int note_active;            //note on flag
    int samples_played;         //samples played of note, used to calc wavetable position
accurately without freq error
    float samples_per_cycle;    //samples per cycle, used to determine wavetabel position
accurately
    int current_sample_index;   //sample index
};
static struct note active_notes[MAX_ACTIVE_NOTES];  //global array for note structs to manage
polyphony
static int num_act_notes = 0;                       //counter to manage number of active notes

static float midi_freq_lookup[128]; //lookup table for decoding midi vals to float freq
(populated by build_midi_lookup)

//ASR ENVELOPE CALIBRATION STUFF
#define FILTER_BIAS 2000
#define AMP_BIAS 5

//// GLOBAL VARS FOR WAVETABLE AND SYNTH SETTINGS ////
static int ENV_counter = 0;
static int KNOB_POS[4] = {0};

int cur_wavetable = 0;

/// GLOBAL LOOKUP TABLES -- WAVETABLE ///
#define WAVETABLE_SIZE 100  //number of samples per cycle in wavetable (independant of output
freq)
static uint8_t active_wave_table[WAVETABLE_SIZE];

static int DCA_aten = 0x3F; //global DCA attenuation

static intr_handle_t s_timer_handle;    //inherit the s timer struct for our intrs

//// DISPLAY TABLE ////
static uint16_t disp_patts[] = { 0x82B6, 0x4900, 0x2133, 0x8844, 0xFF00};

// config the clock devision and routing that generates ticks for the osc and filter ISR timers
timer_config_t timer_config = {
```

```
        .alarm_en = true,
        .counter_en = false,
        .intr_type = TIMER_INTR_LEVEL,
        .counter_dir = TIMER_COUNT_UP,
        .auto_reload = true,
        .divider = 20   /* .025us per tick */
    };

ledc_channel_config_t ledc_channel_fltr_clk = {
        .channel    = LEDC_CHANNEL_1,
        .duty       = 0,
        .gpio_num   = FLTR_CLK_PIN,
        .speed_mode = LEDC_LOW_SPEED_MODE,
        .hpoint     = 0,
        .timer_sel  = LEDC_TIMER_2
};

ledc_timer_config_t ledc_timer_fltr_clk = {
        .duty_resolution = LEDC_TIMER_1_BIT, // resolution of PWM duty
        .freq_hz = 200000,                      // frequency of PWM signal
        .speed_mode = LEDC_LOW_SPEED_MODE,    // timer mode
        .timer_num = LEDC_TIMER_2,            // timer index
        .clk_cfg = LEDC_AUTO_CLK,             // Auto select the source clock
};

//// UPDATES DCA OVER 3WIRE BUS ////
void update_DCA(int volume){
    gpio_set_level(DCA_CLK, 0); //clk low
    gpio_set_level(DCA_LAT, 0); //load low
    uint16_t data_bytes = 0x0000 | volume;

    for(int bit=0; bit<=15; bit++){
        gpio_set_level(DCA_CLK, 0); //falling edge clk line

        //check if cur bit is high
        if(((0x8000 >> bit) & data_bytes) > 0){gpio_set_level(DCA_DAT, 1);}
        else{gpio_set_level(DCA_DAT, 0);}

        gpio_set_level(DCA_CLK, 1);//rising edge clk line
    }

    gpio_set_level(DCA_LAT, 1); //latch high
    gpio_set_level(DCA_DAT, 1);
}

//// UPDATES DISPLAY OVER 3WIRE BUS ////
void update_display(uint16_t data_bytes){
```

```
    gpio_set_level(DIS_LAT, 1); //pulse latch
    gpio_set_level(DIS_LAT, 0);

    for(int bit=0; bit<=15; bit++){
        //check if cur bit is high
        if(((0x8000 >> bit) & data_bytes) > 0){
            gpio_set_level(DIS_DAT, 1);
        }
        else{
            gpio_set_level(DIS_DAT, 0);
        }

        //pulse clk line
        gpio_set_level(DIS_CLK, 1);
        gpio_set_level(DIS_CLK, 0);
    }

    gpio_set_level(DIS_LAT, 1); //pulse latch
    gpio_set_level(DIS_LAT, 0);
    gpio_set_level(DIS_DAT, 0);
}

//// SIGMA DELTA DRIVER INIT FOR OSC ////
static void sigmadelta_init(void)
{
    sigmadelta_config_t sigmadelta_cfg = {
        .channel = SIGMADELTA_CHANNEL_0,
        .sigmadelta_prescale = 1,
        .sigmadelta_duty = 0,
        .sigmadelta_gpio = GPIO_NUM_21,
    };
    sigmadelta_config(&sigmadelta_cfg);
}

//// UART DRIVER INIT FOR MIDI ////
void midi_uart_init(void){
    /* Configure parameters of an UART driver,
     * communication pins and install the driver */
    uart_config_t uart_config = {
        .baud_rate =  ECHO_UART_BAUD_RATE,
        .data_bits = UART_DATA_8_BITS,
        .parity    = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
        .source_clk = UART_SCLK_APB,
    };
    int intr_alloc_flags = 0;
```

```
    #if CONFIG_UART_ISR_IN_IRAM
    intr_alloc_flags = ESP_INTR_FLAG_IRAM;
    #endif

    ESP_ERROR_CHECK(uart_driver_install(ECHO_UART_PORT_NUM, BUF_SIZE * 2, 0, 0, NULL,
intr_alloc_flags));
    ESP_ERROR_CHECK(uart_param_config(ECHO_UART_PORT_NUM, &uart_config));
    ESP_ERROR_CHECK(uart_set_pin(ECHO_UART_PORT_NUM, ECHO_TEST_TXD, ECHO_TEST_RXD, ECHO_TEST_RTS,
ECHO_TEST_CTS));
}

//// INITIALIZE GPIO ////
void init_io(void){
    //TIMING PINS
    gpio_reset_pin(TIMER_PIN_1);
    gpio_set_direction(TIMER_PIN_1, GPIO_MODE_OUTPUT);

    gpio_reset_pin(TIMER_PIN_2);
    gpio_set_direction(TIMER_PIN_2, GPIO_MODE_OUTPUT);

    // 3 WIRE BUS FOR DCA
    gpio_reset_pin(DCA_DAT);
    gpio_reset_pin(DCA_CLK);
    gpio_reset_pin(DCA_LAT);
    gpio_set_direction(DCA_DAT, GPIO_MODE_OUTPUT);
    gpio_set_direction(DCA_CLK, GPIO_MODE_OUTPUT);
    gpio_set_direction(DCA_LAT, GPIO_MODE_OUTPUT);
    gpio_set_level(DCA_CLK, 1); //init clk high
    gpio_set_level(DCA_LAT, 1); //init lat high

    // 3 WIRE BUS FOR DISPLAY
    gpio_reset_pin(DIS_DAT);
    gpio_reset_pin(DIS_CLK);
    gpio_reset_pin(DIS_LAT);
    gpio_set_direction(DIS_DAT, GPIO_MODE_OUTPUT);
    gpio_set_direction(DIS_CLK, GPIO_MODE_OUTPUT);
    gpio_set_direction(DIS_LAT, GPIO_MODE_OUTPUT);
    gpio_set_level(DIS_CLK, 0);
    gpio_set_level(DIS_LAT, 0);

    // GPIO for MODE BUTTON
    gpio_reset_pin(MODE_BUTTON);
    gpio_set_direction(MODE_BUTTON, GPIO_MODE_INPUT);

    // ADC CONFIG for KNOBS
    adc1_config_width(ADC_WIDTH_BIT_13);
    adc1_config_channel_atten(ADC1_CHANNEL_0,ADC_ATTEN_DB_11);
    adc1_config_channel_atten(ADC1_CHANNEL_1,ADC_ATTEN_DB_11);
```

```
    adc1_config_channel_atten(ADC1_CHANNEL_2,ADC_ATTEN_DB_11);
    adc1_config_channel_atten(ADC1_CHANNEL_3,ADC_ATTEN_DB_11);

}

//// OSCILATOR SAMPLE UPDATE ISR ////
static void osc_timer_isr(void* arg){
    gpio_set_level(TIMER_PIN_1, 1); //set the timing output high
    //clear isr
    TIMERG0.int_clr.t0 = 1;
    TIMERG0.hw_timer[0].config.alarm_en = 1;

    static int8_t current_sample = 0;

    sigmadelta_set_duty(SIGMADELTA_CHANNEL_0, current_sample);

    if(num_act_notes > 0){
        int sample_value_sum = 0;
        int i;
        for ( i = 0; i < MAX_ACTIVE_NOTES; i++ ){
            if(active_notes[i].note_active == 1){ //if the note is set as active
                float pos = active_notes[i].samples_played/active_notes[i].samples_per_cycle;
                sample_value_sum += active_wave_table[(int)(WAVETABLE_SIZE * (pos - (int)pos))];
                active_notes[i].samples_played++;
            }
        }
        current_sample = (sample_value_sum/num_act_notes)-127;
    }
    else{
        current_sample = 0;
    }
    gpio_set_level(TIMER_PIN_1, 0); //set the timing output low
}

//// FILTER SAMPLE UPDATE ISR ////
static void fltr_timer_isr(void* arg){
    gpio_set_level(TIMER_PIN_2, 1); //set the timing output high
    TIMERG0.int_clr.t1 = 1; //clear ISR
    TIMERG0.hw_timer[1].config.alarm_en = 1;

    //push last val to pwm reg, do this first for timing reasons
    ledc_timer_config(&ledc_timer_fltr_clk);
    update_DCA(DCA_aten);

    //increment the env count before calculating the next vals
    ENV_counter++;

    ledc_timer_fltr_clk.freq_hz = ((KNOB_POS[0] * KNOB_POS[0]) / 32) + FILTER_BIAS;
```

```
    // if(ENV_counter <= fltr_ENV[0]){
    //     ledc_timer_fltr_clk.freq_hz =
100*(((FILTER_RANGE*ENV_counter*ENV_counter)/(fltr_ENV[0]*fltr_ENV[0]))+FILTER_BIAS);
    // }
    // else if(ENV_counter < (fltr_ENV[0]+fltr_ENV[1])){
    //     ledc_timer_fltr_clk.freq_hz =
100*(((FILTER_RANGE/(fltr_ENV[1]*fltr_ENV[1]))*(ENV_counter-
(fltr_ENV[0]+fltr_ENV[1]))*(ENV_counter-(fltr_ENV[0]+fltr_ENV[1])))+FILTER_BIAS);
    // }

    // //calculate the new amp value from asr settings
    // if(ENV_counter <= amp_ENV[0]){
    //     DCA_aten = (63 - ((63 * ENV_counter) / amp_ENV[0])) + AMP_BIAS;
    // }
    // else if(ENV_counter < (amp_ENV[0]+amp_ENV[1])){
    //     DCA_aten = ((63 * (ENV_counter - amp_ENV[0])) / amp_ENV[1]) + AMP_BIAS;
    // }

    gpio_set_level(TIMER_PIN_2, 0); //set the timing output low
}

//// CONFIGS THE LEDC PWM OUTPUTS ////
void init_ledc_pwm(){
    // config the LEDC PWM for filt
    ledc_timer_config(&ledc_timer_fltr_clk);
    ledc_channel_config(&ledc_channel_fltr_clk);
    ledc_set_duty(ledc_channel_fltr_clk.speed_mode, ledc_channel_fltr_clk.channel, 1);
    ledc_update_duty(ledc_channel_fltr_clk.speed_mode, ledc_channel_fltr_clk.channel);
}

/// CONFIGURES THE TIMERS TO TRIGGER OSC AND FILTER ISR'S ////
void init_timers(int osc_timer_period_ticks, int fltr_timer_period_ticks){

    timer_init(TIMER_GROUP_0, TIMER_0, &timer_config);
    timer_set_counter_value(TIMER_GROUP_0, TIMER_0, 0);
    timer_set_alarm_value(TIMER_GROUP_0, TIMER_0, osc_timer_period_ticks);
    timer_enable_intr(TIMER_GROUP_0, TIMER_0);
    timer_isr_register(TIMER_GROUP_0, TIMER_0, &osc_timer_isr, NULL, 0, &s_timer_handle);

    timer_init(TIMER_GROUP_0, TIMER_1, &timer_config);
    timer_set_counter_value(TIMER_GROUP_0, TIMER_1, 0);
    timer_set_alarm_value(TIMER_GROUP_0, TIMER_1, fltr_timer_period_ticks);
    timer_enable_intr(TIMER_GROUP_0, TIMER_1);
    timer_isr_register(TIMER_GROUP_0, TIMER_1, &fltr_timer_isr, NULL, 0, &s_timer_handle);

    timer_start(TIMER_GROUP_0, TIMER_0);
```

```
    timer_start(TIMER_GROUP_0, TIMER_1);
}

//// ADDS AN ACTIVE NOTE TO THE NOTES LIST AT MIDI NOTE ID, PLAYING IT ////
void add_note(int note_id){
    if(num_act_notes <= 1){//NO CLUE why this needs to be 1 not zero
        ENV_counter = 0;
    }

    for ( int i = 0; i < MAX_ACTIVE_NOTES; i++ ){
        if(active_notes[i].note_active == 0){
            active_notes[i].note_active = 1;    // set the note to active status

            //update the notes sample playback parameters
            float note_period_us = 1000000/midi_freq_lookup[note_id];
            active_notes[i].samples_per_cycle = note_period_us/SAMPLE_PER_US;
            num_act_notes++; //inc the active notes counter
            break;
        }
        //if we haven't found an open note and we have traversed the entire array, then we just
highjack the last note
        if(i == MAX_ACTIVE_NOTES - 1){
            float note_period_us = 1000000/midi_freq_lookup[note_id];
            active_notes[i].samples_per_cycle = note_period_us/SAMPLE_PER_US;
        }
    }
}

void remove_note(int note_id){
    //update the notes sample playback parameters
    float note_period_us = 1000000/midi_freq_lookup[note_id];
    float samples_per_cycle = note_period_us/SAMPLE_PER_US;

    for ( int i = 0; i < MAX_ACTIVE_NOTES; i++ ){
        if((active_notes[i].note_active == 1) & (active_notes[i].samples_per_cycle ==
samples_per_cycle)){
            active_notes[i].note_active = 0;    // set the note to inactive status
            num_act_notes--; //subtract a note from the amount of active notes
            break;
        }
    }
}

//// RESETS A THE VOICE SPECD BY NOTE ID ////
void reset_voice(int note_id){
    active_notes[note_id].note_active = 0;
    active_notes[note_id].samples_played = 0;
    active_notes[note_id].samples_per_cycle = 100;
```

```
    active_notes[note_id].current_sample_index = 0;
}

void initialize_voices(){
    int i;
    // initialize all the note structs by resetting them
    for ( i = 0; i < MAX_ACTIVE_NOTES; i++ ){
        reset_voice(i);
    }
    num_act_notes = 0;
}

//// BUILDS FUNCTIONS INTO WAVETABLE ////
void build_wavetable(int waveshape_id){

    //build functions into the wavetables
    for ( int i = 0; i < WAVETABLE_SIZE; i++ ) {

        //tbl 1 - sin
        if(waveshape_id == 0){
            double wav_samp = (sin((i*2*PI)/WAVETABLE_SIZE))+1;
            active_wave_table[ i ] = (int)(wav_samp*64);
        }

        //tbl 2 - tri
        if(waveshape_id == 1){
            if( i < (WAVETABLE_SIZE/2)){ active_wave_table[ i ] = (int)(i*128)/WAVETABLE_SIZE;}
            else{active_wave_table[ i ] = (int) 128 - ((i*128)/WAVETABLE_SIZE); }
        }

        //tbl 3 - inv saw
        if(waveshape_id == 2){
            active_wave_table[ i ] = (int)(i*64)/WAVETABLE_SIZE;
        }

        //tbl 4 - square
        if(waveshape_id == 3){
            if( i < (WAVETABLE_SIZE/2)){ active_wave_table[ i ] = 0;}
            else{active_wave_table[ i ] = 128;}
        }

        //tbl 5 - junk
        if(waveshape_id == 4){
            active_wave_table[ i ] = rand() % (65);
        }
    }
}
```

```
//// BUILDS VALUES INTO MIDI NOTE LOOKUP TABLE ////
void build_midi_lookup(){
    int i;
    for ( i = 0; i < 127; i++ ){
        midi_freq_lookup[i] = (float)(pow(2, ((float)i - 69)/12))*400;
    }
}

void app_main()
{
    sigmadelta_init(); //initialize the sigma delta pwm output for the osc
    init_ledc_pwm(); //initialize and start the ledc pwm channel for the fltr lock
    init_io();  //initialize the io for timing

    build_midi_lookup(); //populate the lookup table for the midi notes
    initialize_voices(); //initialize the notes class objects in the notes list

    DCA_aten = 0x05; //open the DCA fully
    ledc_timer_fltr_clk.freq_hz = 20000 * 100; //open the DCF completely
    build_wavetable(cur_wavetable); //build fn into wavetable
    update_display(disp_patts[cur_wavetable]);

    // initialize the sample timer and interrupts, starting the oscilator and envelope update
timer
    float osc_ticks_required = SAMPLE_PER_US/TMR_TICK_PER_US;
    init_timers((int)osc_ticks_required, 100000);

    add_note(midi_freq_lookup[55]);

    midi_uart_init();//initialize the midi uart
    uint8_t *data = (uint8_t *) malloc(BUF_SIZE);// Configure a temporary buffer for the incoming
data

    int last_state = 0; //flag for button debounce

    //THIS ROUTINE ONLY WORKS IF ISR ROUTINE OCCURS LESS THAN 1300 times per sec
    //ensures a 3 byte message doesnt get truncated by to frequent a read
    while (1) {
        // Read data from the MIDI UART and handle adding/removing notes
        int len = uart_read_bytes(ECHO_UART_PORT_NUM, data, BUF_SIZE, 20 / portTICK_RATE_MS);
        if(len >= 3){
            for(int byte_index = 0; byte_index <= len - 3; byte_index++){

                if((data[byte_index] & 0xF0) == 0x90){
                    if(data[byte_index + 2] == 0){
                        remove_note((int)data[byte_index + 1]);
                    }
                    else{
```

```
                    add_note((int)data[byte_index + 1]);
                }
            }
            else if((data[byte_index] & 0xF0) == 0x80){
                remove_note((int)data[byte_index + 1]);
            }
        }
    }

    //clear last vals
    int ADC_POS[] = {0,0,0,0};

    //check knob positions
    for (int i = 0; i < MULTISAMP; i++) {
        ADC_POS[0] += adc1_get_raw(ADC1_CHANNEL_0);
        ADC_POS[1] += adc1_get_raw(ADC1_CHANNEL_1);
        ADC_POS[2] += adc1_get_raw(ADC1_CHANNEL_2);
        ADC_POS[3] += adc1_get_raw(ADC1_CHANNEL_3);
    }

    // Only update envelope parameters if no notes are currently active
    if (num_act_notes < 1){
        for (int i = 0; i < 3; i++) {
            KNOB_POS[i] = ADC_POS[i]/MULTISAMP;
        }
    }

    // Continually update the cuttoff knob
    KNOB_POS[0] = ADC_POS[0]/MULTISAMP;


    //check mode button
    if(gpio_get_level(MODE_BUTTON) == 0 && last_state == 0){
        if(cur_wavetable < 4){ cur_wavetable += 1;}
        else{cur_wavetable = 0;}
        build_wavetable(cur_wavetable);
        update_display(disp_patts[cur_wavetable]);
        last_state = 1;
    }
    //mode button debounce
    if(gpio_get_level(MODE_BUTTON) == 1) {
        last_state = 0;
    }

    //loop timer
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}
```