Hybrid Duophonic Synthesizer

Tom Scherlis, Sam Zeloof, Graham MacFarquhar

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A full featured, hybrid, duophonic synthesizer. The synthesizer will make use of digital oscillators and analog filters and amplifiers. It will feature digitally implemented chorus, pitch shifting, chords, and arpeggiators. The analog path will include tunable low pass filters and an amplifier. Notes will be provided separately with an external midi controller, and the whole synthesizer will be built into a sturdy mechanical housing with a strong focus on user experience.**

***Index Terms*—Duophonic, Synthesizer, Wavetable, FPGA, VFO, VCF, Envelope**

I. Introduction

The classic analog synthesizers of the 1960s and 1970s set precedents for the modern musical instruments that we refer to as synthesizers. Even with the development of high speed digital electronics which are capable of producing similar sounds and effects, true analog instruments are still sought after by many musicians. Improvements in DAC linearity and signal processing have brought digital synthesizers to compete very closely with analog ones; however many musicians still prefer analog synthesizers for a collection of objective and subjective reasons. Their interface is often more natural, direct, and akin to an acoustic instrument but the complexity of implementing a fully-featured synthesizer with analog building blocks means that their material and parts cost is high. For this reason original, reproduction, and modern analog synthesizers are sold for tens of thousands of dollars. Digital synthesizers can be minimally realized with a microcontroller, DAC, and touchscreen interface which is comparatively very cheap although may give an inferior user experience. Some artists believe that the “warm tone” produced by having an analog signal path (due to distortions) cannot be reproduced in a digital synthesizer. Regardless of the merit of these subjective arguments, we aim to fill this large divide of performance, interface intuitiveness, and cost with a hybrid synthesizer. Effects and oscillators will be implemented with digital electronics (FPGA and Linux SoC) and we will still have a physical front panel interface and analog signal path to provide the “analog feel.”

II. Use-Case Requirements

As this project is building a musical instrument, the use case-requirements are naturally split between qualitative and quantitative requirements. Our main goal is to create an inexpensive hybrid synthesizer with a large feature set. After doing market research, we have the following minimum set of features that we believe are essential to our synthesizer:

- 2 software controlled oscillators
- Paraphonic voicing
- Modulatable analog filters (resonance & cutoff)
- Modulatable analog amplification
- Effects
  - Pitch shifting
  - Chords
  - Arpeggiators
- Intuitive front panel with hardware knobs

We also wish to implement additional features, as time permits, once we establish the core functionality.

The synthesizer must support multiple voicing modes, i.e. when multiple notes are played simultaneously each tone should be produced and the articulation/envelope behavior should be selectable. This will add additional complexity in software but will allow for better sounding chords and will give the user more control over specific sound. We require that the synthesizer support paraphonic voicing as well as monophonic and stereo modes. Stereo voicing will require two duplicate analog paths but will open a larger space for effects such as chorus that we will implement given enough time.

We require that the synthesizer have two independent software-controlled oscillators that have selectable wave shapes such as sine, sawtooth, triangle, square, pulse, and noise.

We require that the synthesizer have low noise and low distortion analog signal paths consisting of software-modulatable filters. A “mod matrix” can be used to assign LFOs or envelopes to filter parameters such as low-pass cutoff frequency and resonance amount. Each analog path should have a VCF and VCA. Because the focus of this project is to create a synthesizer capable of making lead electronic sounds rather than brass or drum tracks, high-pass filters are considered to be not necessary and we only require that the synthesizer have tunable low-pass filters as this should provide the user with enough satisfaction.

The synthesizer should also be intuitive to use and offer a degree of portability (smaller than a microwave oven). To create the best possible user experience, consideration should be given to the layout and function of potentiometers/encoders on the front panel.

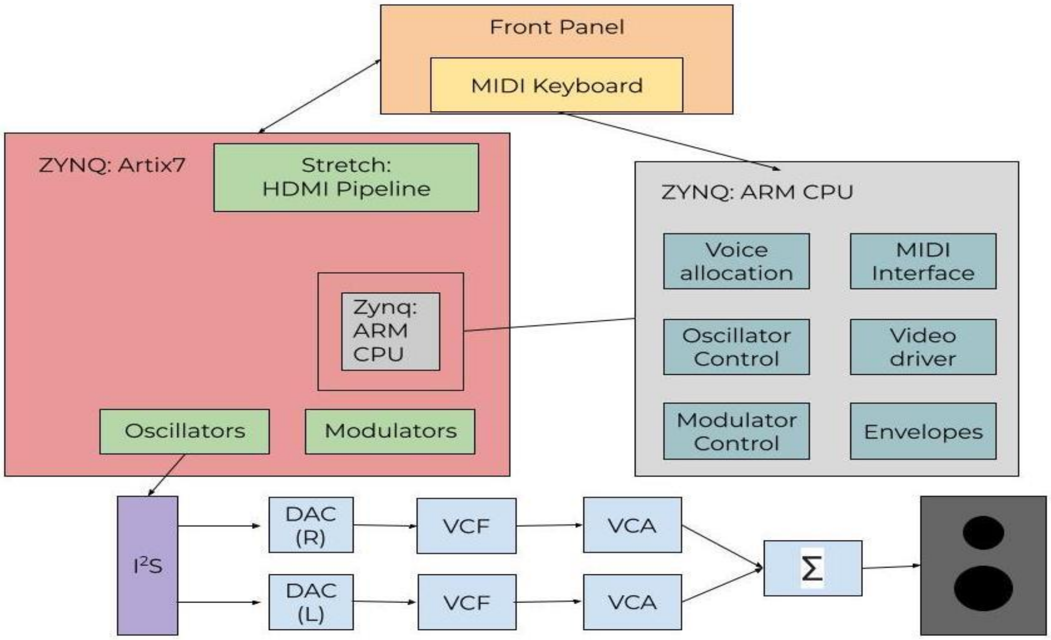


Figure 4. System Block Diagram

III. Architecture and/or Principle of Operation

The principle of operation of a typical hybrid synthesizer is fairly straightforward. The instrument consists of a digital processor and analog filter path with a DAC between them. When the user presses a key on the MIDI input device, the software oscillators are set to the correct frequency and any effects, if selected, are applied. The user may have loaded a "patch" which consists of settings and parameters which describe a specific sound that was previously recorded. The patch may be loaded and modified using the knobs on the front panel. One important feature is the envelopes. These have four parameters: attack, sustain, decay, and release. They may be programmed to modulate some parameter in the analog path such as filter cutoff or amount immediately after the key is pressed on the MIDI controller. The effect is programmable note articulation control.

From the top level, our architecture is simple and consists of a front panel, MIDI input jack, Zynq 7020 FPGA, and analog filter path. The front panel will have potentiometers, encoders, buttons, and a screen to set and display the various sound parameters and effects. The MIDI input will accept any MIDI controller and make the synthesizer extremely flexible. The Zynq FPGA is chosen because it has an integrated dual-core ARM processor. A stretch goal is to include an HDMI pipeline in the FPGA to control a screen on the front panel that can display waveshapes, patch information, and knob positions.

The Zynq FPGA will give us extreme flexibility in implementation because we can choose to put various parts of the software stack in either the FPGA (written in Verilog HDL) or the ARM processor (high level embedded C++) depending on what is easiest or makes the most sense. This will give us the opportunity to implement some effects that would be extremely time consuming with only an FPGA. We hope to maintain the productivity of working in a high language with the real-time parallel capability of an FPGA and having the two integrated on the same chip with shared memory will be helpful. HDMI, oscillators, and modulator control will be in the FPGA while voice allocation, envelopes, MIDI interface, and miscellaneous effects will be implemented in the ARM CPU. The FPGA and CPU will have internal communication by writing to and reading from shared memory and both devices will communicate with other chips via various serial protocols.

The FPGA will drive the analog path via serial. There will be a stereo audio DAC (I2S) and a 8-channel control signal DAC (I2C) to modulate the filters. A paraphonic synthesizer only requires one analog path but we will include a second with switchable summing to support stereo and more complicated voicing modes. Each analog path will be driven from one channel of the stereo audio DAC and consist of a controllable filter (VCF) and controllable amplifier (VCO). Like in a traditional hybrid synthesizer, these filters and amplifiers will be modulated by envelopes or LFOs selectable on the front panel or in software. We also desire the ability to output to two separate right and left line-level outputs (stereo paraphonic) or sum the two analog paths for more complicated voicing modes (duophonic).

A minimal set of cables will be required to use the synthesizer. Power will be provided by a 12V DC adapter, MIDI input will be over a standard 5-pin DIN connector, and audio out will be either via two ¼" right/left jacks or a single ⅛" stereo headphone jack.

IV. Design Requirements

Our design has several qualitative requirements that are essential for a simple synthesizer. These include oscillators, low-pass filters, and amplifiers which will be modulatable through the use of LFOs and ADSR envelope controls. The entire project should fit neatly into an enclosure with a well designed front panel. These are the minimum requirements for a marketable synthesizer today. In addition to these very basic requirements, our synthesizer will implement wavetable synthesis, duophonic voicing, and multiple effects such as pitch shifting, chords and arpeggiators.

Table 1 shows several quantitative requirements that should be achieved to ensure a straightforward and pleasant user experience. We require that the system is correctly tuned to our input frequency. The precision of human pitch perception varies from person to person. However, musicians are typically capable of recognizing an out-of-tune note after a deviation of ±10¢. Therefore, we have settled on an acceptable ±5¢. Also, we expect our off-the-shelf filter and amplifier IC's to maintain an even frequency response so that we retain all desired frequencies. We chose a four-pole low-pass filter chip to ensure greater attenuation than our discrete design would achieve. Additionally, minimizing our harmonic distortion is essential for achieving a pleasant sounding output. The input to output delay should be imperceptible for users. Naturally, the human mind can compensate for upto 12ms of latency. Finally, we would like to have a competitive price point and the entire project should be relatively portable.

Metric	Quantitative Goal
Pitch Correctness	±5¢
Filter Cutoff	<5% off ideal
THD	<1%
Latency	<12ms
Competitive Pricing	<$500
User Enjoyment	>70% +ve feedback
Portability	>= toaster, < microwave

Table 1. Design Requirements

V. Design Trade Studies

A. Analog Filters Design

Through exploring alternative designs for the analog filter architecture we came to the conclusion that implementing analog filters and amplifiers using discrete components and switched capacitor architecture would be too difficult within the time-span required for our project.

As a more efficient alternative, the synthesizer will now implement low-pass filters and amplifiers using VCF and VCA ICs. These ICs are a cheap solution to simplify the design of our analog path and allow us to focus more efforts on FPGA programming and components of sound synthesis.

B. Voicing Complexity

After much thought and a very productive conversation with our faculty mentor, Tom Sullivan, we made the decision to decrease the voicing complexity of our system.

Previously, we desired a 6 voice polyphonic synthesizer, hence the name 'PROGNOSTICATOR-6' however, we soon came to understand that we would need six separate analog paths with 4 control signals for each set of filters. This introduced worries with filter consistency, I/O and general complexity requiring a larger enclosure and more components.

We explored alternatives such as a paraphonic synthesizer, with only one VCF and VCA for all 6 voices. This design was much more feasible, however it fell considerably short of our goals. We have now settled on a duophonic synthesizer with 3 voices per analog path. Each set of voices has modulatable ADSR envelopes, low-pass filter cutoff and amplifier. This design has reduced complexity to an acceptable level that we believe we can handle within our build phase.

C. PYNQ Z2 FPGA

Our choice in FPGA is a PYNQ Z2 FPGA development board. The PYNQ Z2 integrates the Xilinx Zynq 7020 SoC. We will exploit the benefits of programmable logic and microprocessors in Zynq to build a more capable and exciting synthesizer. Essentially, having the SoC provides us with the capability to implement effects programmed in python. There are also several notable I/O elements on the board that will allow us to more easily integrate different components of the project. General purpose GPIO pins on the board make MIDI UART input more simple. An audio output will allow us to test the difference between our sound before and after filtering. Additionally, the HDMI output could allow us to add a screen in the future to visualize waveforms.

There are so many advantages to our choice in FPGA, especially the fact that it is owned by one of our teammates so we will not need to disassemble the project after completion. The very notable downside is that none of us have experience programming such an FPGA using the necessary toolchains. There are many resources online to guide us through these tasks, and we have endeavored to learn from them in the past two weeks.

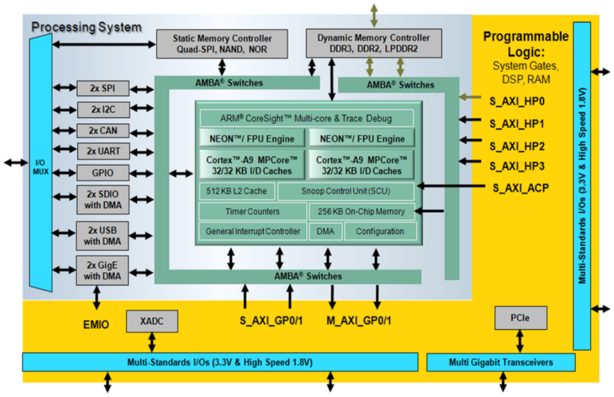

Processing System
Static Memory Controller Quad-SPI, NAND, NOR
Dynamic Memory Controller DDR3, DDR2, LPDDR2
Programmable Logic: System Gates, DSP, RAM
I/O MUX
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
AMBA® Switches
AMBA® Switches
ARM® CoreSight™ Multi-core & Trace Debug
NEON™/ FPU Engine
NEON™/ FPU Engine
Cortex™-A9 MPCore™ 32/32 KB I/D Caches
Cortex™-A9 MPCore™ 32/32 KB I/D Caches
512 KB L2 Cache
Snoop Control Unit (SCU)
Timer Counters
256 KB On-Chip Memory
General Interrupt Controller
DMA
Configuration
AMBA® Switches
S_AXI_HP0
S_AXI_HP1
S_AXI_HP2
S_AXI_HP3
S_AXI_ACP
EMIO
XADC
S_AXI_GP0/1
M_AXI_GP0/1
PCIe
Multi-Standards I/Os (3.3V & High Speed 1.8V)
Multi Gigabit Transceivers
Multi-Standards I/Os (3.3V & High Speed 1.8V)

Figure 2. Zynq Architecture

VI. System Implementation

A. Analog Filters

As described earlier, each of two analog paths will contain a DAC, VCF, and VCA (Figure 2). The DAC (UDA1334A) will be driven by the FPGA via I2S, a streaming serial protocol designed for audio applications. The FPGA will also drive an 8-channel DAC (DAC7578SPQR) via I2C to generate control voltage for the voltage controlled filter and amplifiers.

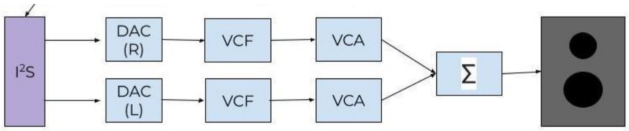


Figure 3: Analog Path Block Diagram

Since the synthesizer is to support duophonic and stereo voicing modes, the two filter paths will be followed by a switchable summing amplifier (controlled by the FPGA/SoC) that will either sum the two VCA outputs or send each to a separate audio line-out.

Figure 3 shows the completed schematic with a stereo audio DAC, two independent VCFs, a single 4-channel VCA (we are only using 2) and the 8-channel control signal DAC. The large text shown in blue delineates the functional blocks within the schematic.

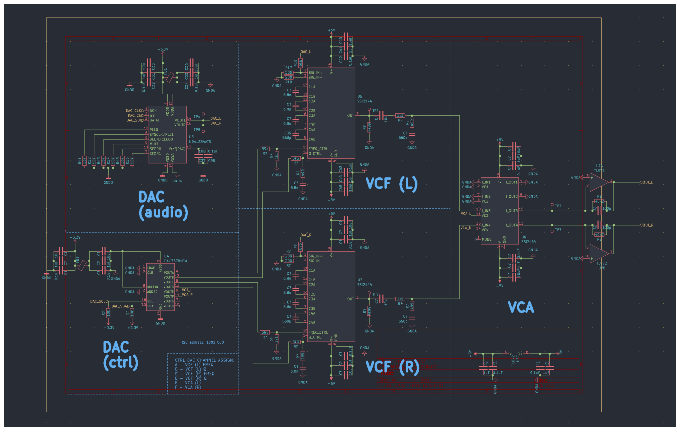


Figure 4. Analog Filter Schematic

The VCF and VCA integrated circuits are the SSI2144 and SSI2164, respectively. They are monolithic filter and amplifier ICs and are manufactured by Sound Semiconductor as reproduction of vintage 70s synthesizer analog paths. We originally planned on building the filters from discrete components and evaluated this idea in simulation but ultimately decided to opt for off-the-shelf filter and amplifier ICs so that we can focus on parts of the project that will require more attention.

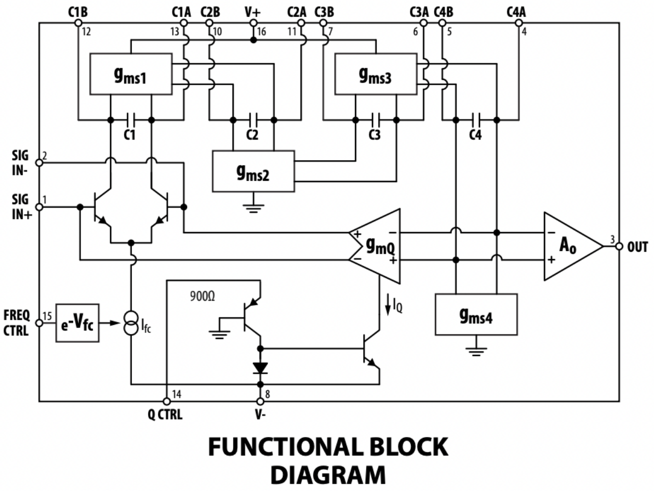


Figure 5. SSI2144 Functional Block Diagram

The SSI2144 VCF uses voltage-controlled transconductance cells to implement a four-pole low pass filter. The chip has a differential audio input, although we are only using the noninverting input and leave the other at ground. It also has a voltage controlled cutoff frequency pin and a current controlled resonance pin. Both are driven from the control signal DAC and a series resistor is used to convert the voltage-proportional signal to a current-proportional signal for the resonance control.

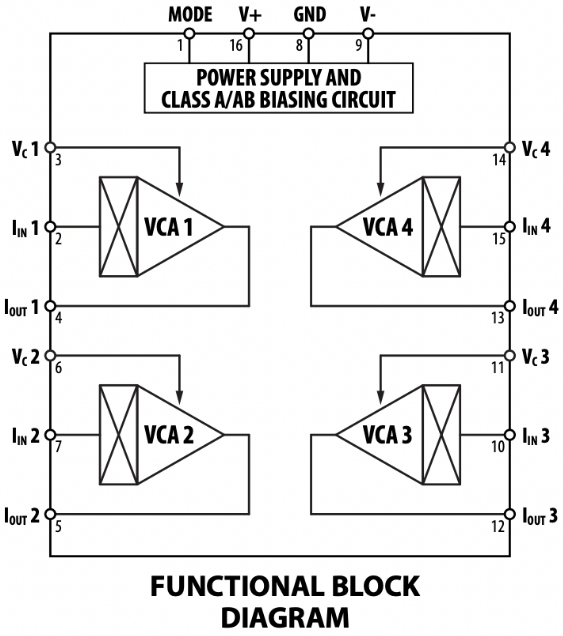


Figure 6. SSI2164 Functional Block Diagram

The SSI2164 VCA chip has four programmable gain amplifiers. For our stereo synthesizer, we only require two channels so the remaining two have grounded inputs to reduce power consumption. The audio input is scaled using a resistor divider to the appropriate range to avoid clipping and distortions. The output of the IC is a current rather than voltage, so the VCA is followed by two operational amplifiers configured as transimpedance amplifiers to convert it to a voltage output. That is scaled with another operational amplifier to line-level before going to the audio output jack.

B. Front Panel

We plan to put considerable thought into the front panel layout to make the synthesizer fun and intuitive to use. Options for typical front panel interfaces include buttons, potentiometers, switches, and encoders. There are many parameters which we wish to control from the front panel, such as attack/sustain/decay/release envelopes, oscillator shape, oscillator octave, voicing modes, LFOs, mod matrix, and save/load patch. We plan to implement all of these features with a carefully assigned set of 26 quadrature rotary encoders, 3 buttons, and 1 potentiometer. The potentiometer will be used for master volume control. The 3 buttons will be for patch control (loading, saving) and navigating menus on the screen, if implemented.

The main challenge with front panel design is how to interface with a large number of rotary encoders. First, we chose to use rotary encoders over potentiometers for the majority of knobs because it would be easier than having 26 analog channels and ADCs. After some research, we found an off-the-shelf board that uses a microcontroller to “convert” the encoder pulses to a I2C interface (fig 6). This will allow us to chain all 26 encoders together on the same 2-wire serial I2C interface and easily read values from all of them on the FPGA and SoC.

Figure 7. I2CEncoder V2.1 - DUPPA on tindie.com

C. MIDI Controller Interface

In addition to encoders and switches on the front panel, we will utilize a MIDI controller in the form of an off-the-shelf keyboard for user input of notes. This controller will interface with the FPGA and notes must be interpreted in the linux SoC.

Our keyboard has 5-pin DIN output and USB output. We will only be using the 5-pin DIN output in order to avoid designing or sourcing a USB controller. The DIN connector is included for MIDI controllers in the official MIDI technical specifications. The DIN connector utilizes one pin for serial output that we will stream directly to the UART input port on the FPGA. The inputs will be sent as bytes to a MIDI decoder written on the SoC.

D. Software

Synthesizer software will be implemented with C++ and Qt. The software will run on the ARM core on the Processing System. (PS) The software will implement all synthesizer behavior aside from the oscillators. This includes midi parsing, voice allocation, ADSR envelopes, modulation via mod matrix, and the user interface.

The software is implemented on top of PetaLinux, Xilinx's linux image for the Zynq PS. PetaLinux includes framebuffer drivers for video out and drivers for I2C to communicate with the encoder knobs and filter control DAC. We will need to implement kernel drivers for the oscillators which are implemented on FPGA fabric on the Programmable Logic (PL).

Qt was chosen as the C++ software framework, since it provides a convenient threading/callback model (Signals/Slots) as well as a well documented and understood GUI framework for visualizers.

E. Voicing

The synthesizer supports true duophony, but there are many ways to voice notes using that. We have decided to implement Mono, Single, Paraphonic, and Duophonic modes. These modes vary based on gatiing, oscillator allocation, and stereo support. Gating refers to how the ADSR filter envelopes are reset.

Single voicing is monophonic and triggers the gate on every key press. Notes can be voiced with one analog voice, two in chorus, or two in stereo.

Mono voicing is monophonic and triggers the gate when no keys are pressed. Notes can be voiced with one analog voice, two in chorus, or two in stereo. Pressing notes in close sequence without lifting the first will not reset the filter envelopes and allows notes to blend using portmanteau and glides.

Paraphonic voicing is similar to monophonic, but allows for multiple tones to be played simultaneously by making use of multiple oscillators. The gate is triggered when all keys are lifted. Chords and polyphonic melodies can be played, but only one filter envelope can be used. Notes can be voiced with one analog voice, two in chorus, or two in stereo.

Duophonic voicing allows the two analog voices to be used independently, allowing true polyphony (both notes get their own filter envelope.) Gating happens on every key press, and if more than two notes are held, the oldest note will not be voiced. This mode will not support stereo since both hardware voices are being used independently.

F. Oscillators

Oscillators will be implemented on the FPGA fabric on the PL. We plan on programming them using Vivado. We will either implement oscillators using wavetable synthesis or pre-defined saw/sin/tri/square wave shapes depending on time and complexity.

Wavetable oscillators present a significant number of challenges for implementation. Wavetables need to be stored in memory, and multiple wavetables need to be generated/blended depending on pitch to prevent aliasing. If this proves too complex to implement, we will implement sine, sawtooth, triangle, and square waves analytically instead. This still gives us significant timbre flexibility, but removes the ability to interpolate between wave shapes or modulate wave shape on the fly.

We will interface with oscillators using the Xilinx AXI Memory-Mapped interface. This allows the linux software on the PS to control the oscillators using register read/writes. Oscillator outputs will be streamed directly off of the FPGA via I2S to the stereo audio DAC.

VII. Test, Verification and Validation

A. Total Harmonic Distortion

Total Harmonic Distortion (THD) is an important metric for describing sound quality. Harmonic distortions are variations in current and voltage due to frequencies within circuitry. THD is an important metric for sound quality. We are aiming for <1% THD with our synthesizer. This is the ideal goal, but we may run into trouble considering how tightly packed and noisy our system may be. We will measure THD using an FFT on an oscilloscope to see what harmonics arise in the frequency domain. Hopefully we can mitigate some of these frequencies by rearranging and separating components of the synthesizer.

B. Analog Filter Attenuation

The analog filters are four-pole low pass filters built into ICs by Sound Semiconductor for the very purpose of replicating analog synthesizer components. We should hopefully have very little trouble verifying the behavior of these filters. Nevertheless, we will construct BODE plots using the oscilloscope to verify their cutoff frequency and attenuation.

C. Pitch Correctness

To Verify our correctness of pitch within ±5¢ we will use a regular instrument tuner. The cent is a logarithmic unit of measure for music intervals. Cents are measured as the interval between a desired note frequency and a generated note frequency. The octave is divided into 12 semitones of 100 cents each.

$$¢ = 1200 \bullet log_2\left(\frac{f_1}{f_2}\right)$$

An app or a physical tuner will do just fine when testing this metric.

D. Latency

Our final metric, latency, can also be measured on the oscilloscope. Using two inputs we can detect an original input through the midi controller or encoder and then observe the time taken for a response to be generated from our audio output. This will suffice for ensuring our latency does not exceed 12ms.

VIII. Project Management

A. Schedule

Our Gantt Chart in Table 2. has been updated recently. After the initial construction, we added time to design the system before beginning our build phase. Our software became more complex and our hardware has become less complex so we shifted responsibilities to focus on the FPGA.

These past two weeks have been highly focused on nailing down an allocation for encoders and focusing on altering the design to fit our capabilities.

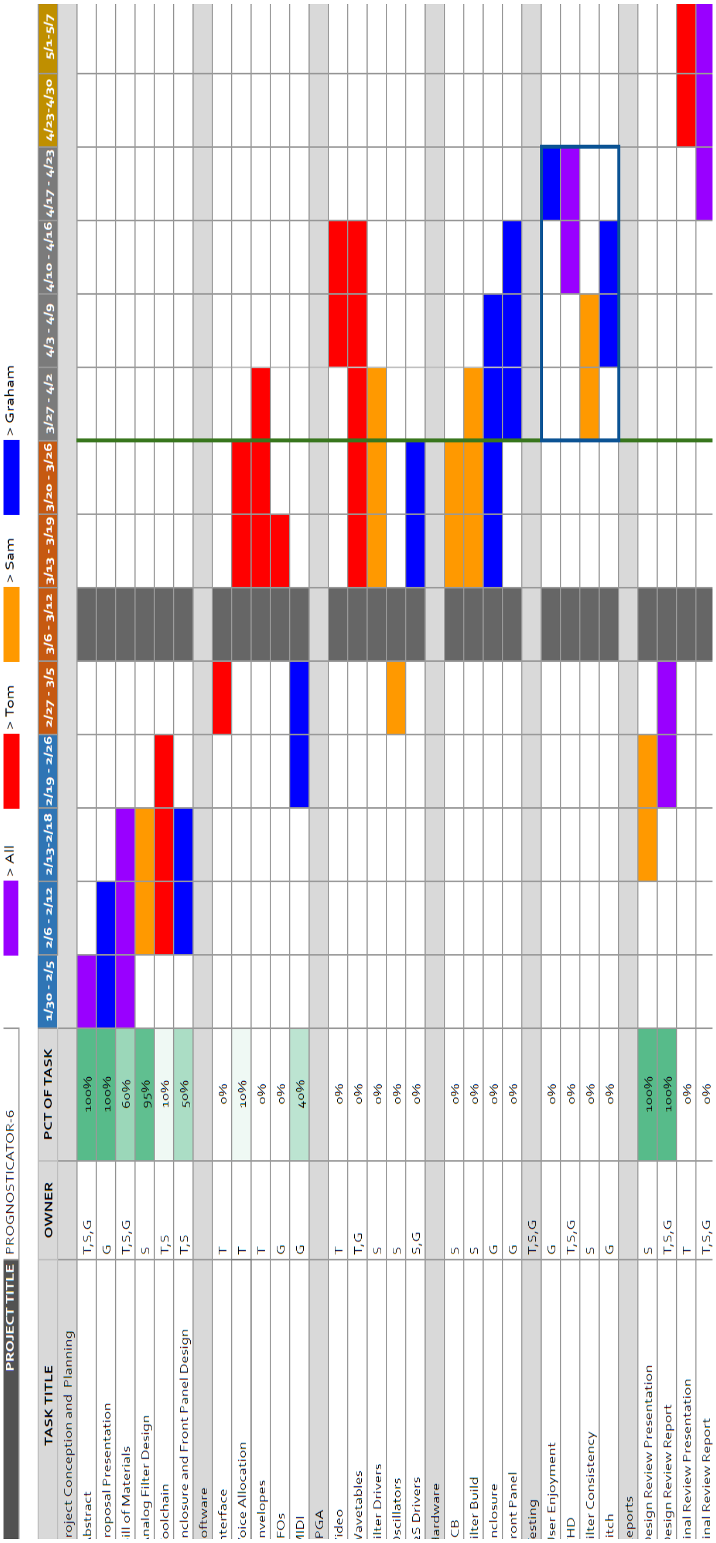
B. Team Member Responsibilities

Shown in Table 2, team member responsibilities are separated between the three teammates based on experience. Lately, Tom has been focused on the FPGA toolchain, Sam is assigned to work on the Analog filter architecture and Graham has created a MIDI interpreter and is designing the Front panel.

In the near future, we will all be working together on the FPGA and doing our best to integrate the analog filters and implement oscillators and software effects on the synth,

Tom Scherlis	Sam Zeloof	Graham MacFarquhar
Software: Voice allocation	Analog Filter Design	FPGA: I^{2}S Drivers
Software: Interface	PCB Design	FPGA: Oscillators
Software: Envelopes	FPGA: Filter Drivers	Software: MIDI
FPGA: Video	FPGA: I^{2}S Drivers	Software: LFOs
FPGA: Oscillators	FPGA: Oscillators	Enclosure design
Toolchain	Toolchain	Front panel design

Table 2. Team Member Responsibilities



PROJECT TITLE PROGNOSTICATOR-6

🟪 > All 🟥 > Tom 🟧 > Sam 🟦 > Graham

TASK TITLE	OWNER	PCT OF TASK	1/30 - 2/5	2/6 - 2/12	2/13-2/18	2/19 - 2/26	2/27 - 3/5	3/6 - 3/12	3/13 - 3/19	3/20 - 3/26	3/27 - 4/2	4/3 - 4/9	4/10 - 4/16	4/17 - 4/23	4/23-4/30	5/1-5/7
roject Conception and Planning																
bstract	T,S,G	100%	🟪					▨								
roposal Presentation	G	100%	🟦	🟦				▨								
ill of Materials	T,S,G	60%	🟪	🟪	🟪			▨								
nalog Filter Design	S	95%		🟧	🟧			▨								
oolchain	T,S	10%		🟥	🟥	🟥		▨								
nclosure and Front Panel Design	T,S	50%		🟦	🟦			▨								
oftware																
nterface	T	0%					🟥	▨								
oice Allocation	T	10%						▨	🟥	🟥						
nvelopes	T	0%						▨	🟥	🟥	🟥					
FOs	G	0%						▨	🟥							
IDI	G	40%				🟦	🟦	▨								
PGA																
ideo	T	0%						▨				🟥	🟥			
Vavetables	T,G	0%						▨	🟥	🟥	🟥	🟥	🟥			
ilter Drivers	S	0%						▨	🟧	🟧	🟧					
scillators	S	0%					🟧	▨								
S Drivers	S,G	0%						▨	🟦	🟦						
ardware																
CB	S	0%						▨	🟧	🟧						
ilter Build	S	0%						▨	🟧	🟧	🟧					
nclosure	G	0%						▨	🟦	🟦	🟦	🟦				
ront Panel	G	0%						▨			🟦	🟦	🟦			
esting	T,S,G															
ser Enjoyment	G	0%						▨						🟦		
HD	T,S,G	0%						▨					🟪	🟪		
ilter Consistency	S	0%						▨			🟧	🟧				
itch	G	0%						▨				🟦	🟦			
eports																
esign Review Presentation	S	100%			🟧	🟧		▨								
esign Review Report	T,S,G	100%				🟪	🟪	▨								
inal Review Presentation	T	0%						▨							🟥	🟥
inal Review Report	T,S,G	0%						▨						🟪	🟪	🟪

Table 3. Gantt Chart

C. Bill of Materials and Budget

We have been able to keep a relatively low budget despite the massive chip shortage resulting from supply chain logistics and perhaps other issues in China. Our low costs are due to our previous ownership of the PYNQ Z2 board and the high quality MIDI keyboard both provided by Tom.

The components we have purchased or are required to purchase include: digital encoders, knobs, a MIDI port, a custom PCB, I2C, I2S, DACs and VCA and VCF ICs. The total cost of these components is currently an estimated $400. We plan to purchase an HDMI display as well as part of our display stretch goal, which is an additional $100-150.

D. Risk Mitigation Plans

We are facing the risk of overdeveloping our user experience and PCB integration and failing to implement the synthesizer wavetables. We may have to settle for wired connections and a simple front panel if we fall behind on the elements of sound synthesis that really make or break our synthesizer.

Furthermore, our decision to use the PYNQ Z2 may be hazardous to some of our effects if we cannot build a Petalinux image on the Zynq. Instead, we may need to rely on the actual Python productivity for Zynq and sacrifice some I/O which will cost us a few encoders, and further simplify the complexity of our system.

IX. Related Work

In our research on the topic of synthesizers, we have discovered that FPGAs are often the way to go for developing DIY synths. We have found many projects with similar architecture: MIDI input, wavetables, etc. However, there seems to be much less information on DIY analog synthesizer architecture. This is certainly due to the availability of analog components and the increased complexity of implementation.

Specifically, some useful resources we found include an FPGA Lab from Berkeley EECS “Introduction to FPGA Development + Creating a Tone Generator”. This lab requires students to create a tone generator using a PYNQ Z1. This is a very useful resource for understanding how Vivado may be used in simulation to analyze our implementation.

We also realized early on that several of our predecessors from 18-500 in 2019 built their own Wavetable synthesizer on an Altera FPGA and included analog filters too. This was remarkably similar to our project, however we have elected to add more complexity to user interface and effects. Their project documentation has been a useful reference as we have learned to prepare for issues with noise in our analog filter architecture.

X. Summary

Our proposed system will meet all of our design requirements. Any musician familiar with synthesizers will be able to understand our interface and manipulate our synthesizer to create rich and interesting sounds. Our main challenges in reaching our design goals and use-case requirements are complexity in the core voicing system and toolchain challenges. We need to implement FPGA oscillators, setup and use PetaLinux, interface between the PS and PL of the Zynq, and design a functional analog filter circuit to even produce a tone. This is a very complex system with a lot of inherent risk that cannot be descoped.

However, we believe we have studied and de-risked the trickiest parts of this system and are confident that we can implement a synthesizer that can produce rich complex sounds. We hope to implement voicing modes, modulators, and filters that sound good and work like a real consumer synthesizer. As musicians ourselves, we are highly motivated to produce an instrument that we actually want to use.

Glossary of Acronyms

ADSR - Attack Decay Sustain Release
DAC - Digital to Analog Converter
FPGA - Field Programmable Gate Array
GUI - Graphical User Interface
IC - Integrated Chip
LFO - Low Frequency Oscillator
MIDI - Musical Instrument Digital Interface
PL - Programmable Logic
PS - Processor System
SoC - System on Chip
THD - Total Harmonic Distortion
VCO - Voltage controlled oscillator
VCA - Voltage controlled amplifier
VCF - Voltage controlled filter

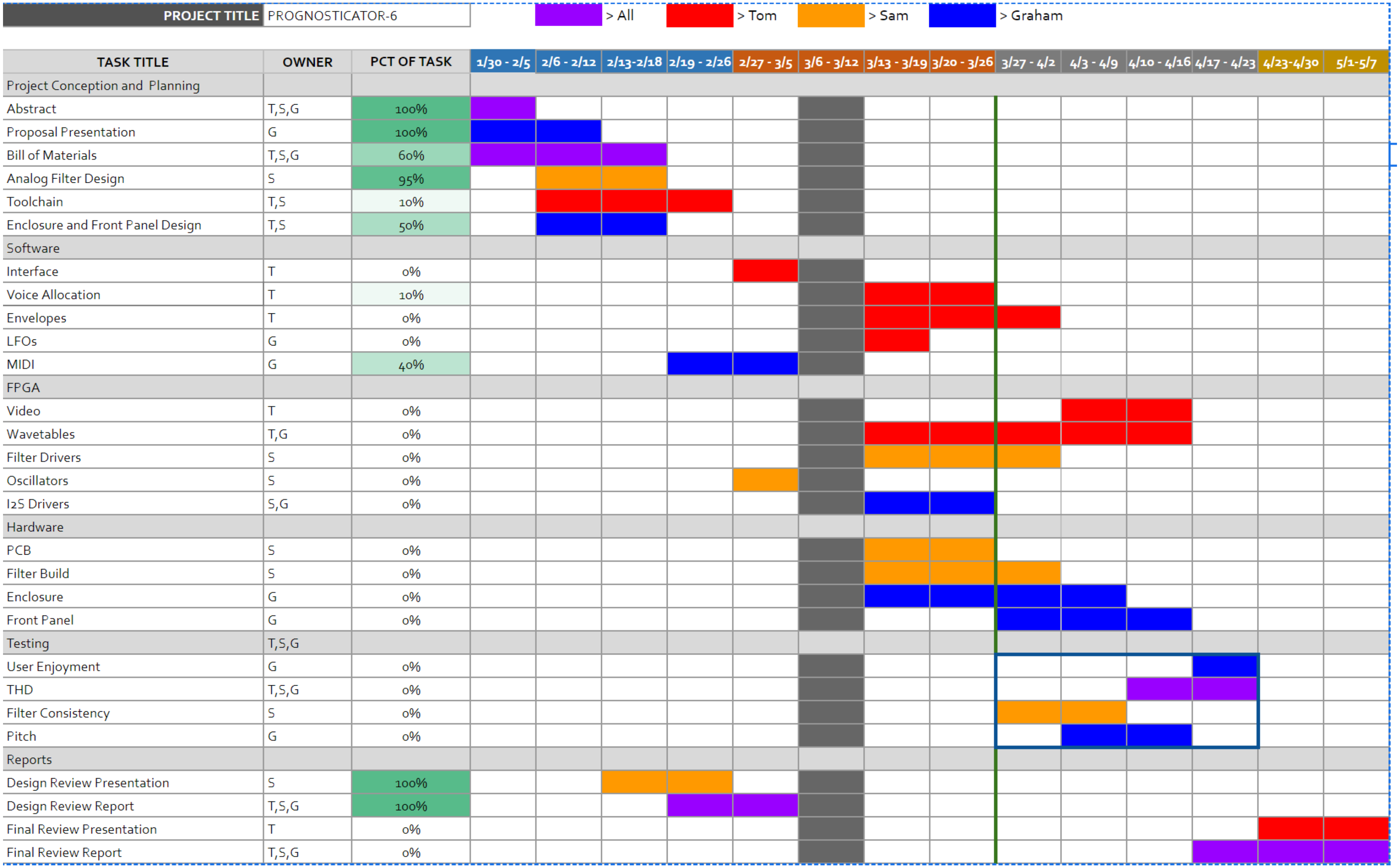
References

[1] MIDI Specifications https://midi.org/specifications

[2] FPGA Lab Spec https://inst.eecs.berkeley.edu/~eecs151/sp18/files/fpga_lab2_spec.pdf

[3] Team A0 ECE Capstone, Spring 2019 http://course.ece.cmu.edu/~ece500/projects/s19-teama0/2019/05/09/final-project-report/

Gantt Chart Section VIII.A



PROJECT TITLE PROGNOSTICATOR-6

🟪 > All 🟥 > Tom 🟧 > Sam 🟦 > Graham

TASK TITLE	OWNER	PCT OF TASK	1/30 - 2/5	2/6 - 2/12	2/13-2/18	2/19 - 2/26	2/27 - 3/5	3/6 - 3/12	3/13 - 3/19	3/20 - 3/26	3/27 - 4/2	4/3 - 4/9	4/10 - 4/16	4/17 - 4/23	4/23-4/30	5/1-5/7
Project Conception and Planning																
Abstract	T,S,G	100%	🟪					⬛								
Proposal Presentation	G	100%	🟦	🟦				⬛								
Bill of Materials	T,S,G	60%	🟪	🟪	🟪			⬛								
Analog Filter Design	S	95%		🟧	🟧			⬛								
Toolchain	T,S	10%		🟥	🟥	🟥		⬛								
Enclosure and Front Panel Design	T,S	50%		🟦	🟦			⬛								
Software																
Interface	T	0%					🟥	⬛								
Voice Allocation	T	10%						⬛	🟥	🟥						
Envelopes	T	0%						⬛	🟥	🟥	🟥					
LFOs	G	0%						⬛	🟥							
MIDI	G	40%				🟦	🟦	⬛								
FPGA																
Video	T	0%						⬛				🟥	🟥			
Wavetables	T,G	0%						⬛	🟥	🟥	🟥	🟥	🟥			
Filter Drivers	S	0%						⬛	🟧	🟧	🟧					
Oscillators	S	0%					🟧	⬛								
I2S Drivers	S,G	0%						⬛	🟦	🟦						
Hardware																
PCB	S	0%						⬛	🟧	🟧						
Filter Build	S	0%						⬛	🟧	🟧	🟧					
Enclosure	G	0%						⬛	🟦	🟦	🟦	🟦				
Front Panel	G	0%						⬛			🟦	🟦	🟦			
Testing	T,S,G															
User Enjoyment	G	0%						⬛						🟦		
THD	T,S,G	0%						⬛					🟪	🟪		
Filter Consistency	S	0%						⬛			🟧	🟧				
Pitch	G	0%						⬛				🟦	🟦			
Reports																
Design Review Presentation	S	100%			🟧	🟧		⬛								
Design Review Report	T,S,G	100%				🟪	🟪	⬛								
Final Review Presentation	T	0%						⬛							🟥	🟥
Final Review Report	T,S,G	0%						⬛						🟪	🟪	🟪