Check Out Our Soundcloud: An FPGA Wavetable Synthesizer

Jens Ertman, Charles Li, Hailang Liou:

Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of being a low-cost, FPGA-based wavetable synthesizer with digital wave-blending effects and digital effects chain including multi-voice unison, distortions, and reverb. While there are many inexpensive wavetable synthesizers that include basic digital effects, wave-blending effects are only found on expensive full-featured hardware synthesizers or software synthesizers. Our goal is to create a system targeting a platform competitive with other low-cost synthesizers on the market while capable of unique and interesting wave-blending effects not found in this price segment.**

***Index Terms*— FPGA, Music, Synthesizers, Wavetable Synthesis**

I. Introduction

In this project, our team aims to develop an FPGA-based wavetable musical synthesizer that targets wave-blending and wave-shaping effects along with other standard digital effects found in synthesizers at comparable price points. Although we are developing the system on an FPGA, we envision this project as a prototype for evaluation on the track to create a dedicated chip for musical synthesis. From a market perspective, we are targeting the prosumer audio/music market, where people are interested in unique effects and sounds but are not willing to spend upwards of $4000-5000 for a professional synthesizer kit. Other devices in the market range typically either have simple oscillators combined with effects or wavetables that focus on instrument sound reproduction. While software solutions do exist, they are not self-contained and require a digital audio workstation, a MIDI controller, and some other software for the system to work in, meaning that although the wavetable synthesizer may not be extremely expensive, the combined workflow can be both high cost and limiting. Our goals for our synthesizer focus on having a polyphonic synthesizer with unique wavetable synthesis effects in addition to standard effects found on low-cost synthesizers such as distortion, delay, unison, and delay. On a more technical side, we want to have pitch accuracy within 5 cents of standard tunings, minimal harmonic distortion, and even frequency response. Finally, an important goal is to keep the total parts cost of the project as low as possible, to show that these effects can be had at a low cost.

II. Design Requirements

The requirements of the project will be split into two parts: features and audio fidelity. Other metrics we will be measuring, but without a hard requirement, are FPGA usage (chip area) and price. Power consumption is of a lesser concern because the system will be designed to plug into a wall outlet.

For features, we will first discuss traits of the synthesizer. The synthesizer will support four note polyphony, where four notes can be played through the synthesizer simultaneously. Polyphony is a feature found on some, but not all low-cost synthesizers, but we believe it to be a valuable feature to have in order to support being able to play chords. The synthesizer should have at least four different wave shapes stored in the wavetables; we anticipate the shapes to be two simple ones, such as a sine and a sawtooth, and two complex waveforms which can make interesting sounds when blended with others. The synthesizer can support user-controlled blending any two wave shapes together. For user-controlled synthesizer effects, we plan on implementing distortion (sample reduction and bit-depth reduction), delay, and reverb. the analog side, we will have an analog equalizer doing a final filtering step. For the digital synthesis and effects components, we will test them using Verilog testbenches, and for the analog filtering, we will generate frequency response plots by passing in noise.

For audio fidelity, we have determined several technical requirements that the project should meet. As a musical instrument, we want our synthesizer to be in tune, so we want our synthesizer to produce sounds within five cents of standard tunings. This will be tested with a regular instrument tuner and the synthesizer outputting an undistorted sine. We want the total harmonic distortion to be less than 5%, a property we will test by generating sine waves of different frequencies, running an FFT on the output, and calculating how much distortion exists at higher harmonics. Similarly, we want the frequency response to be even across all levels (<5%), and make sure no effect or filter behaves differently at different frequencies. This will be tested by observing output response over a range of frequencies and looking for deviations.

Our soft metrics, area and power, will be measured using the FPGA synthesis tools and measured for the analog components. Price will be measured by the total cost of components that we purchase, as well as the cost of the FPGA board, which is provided by the university. We aim to minimize these metrics overall but are not working with a hard requirement.

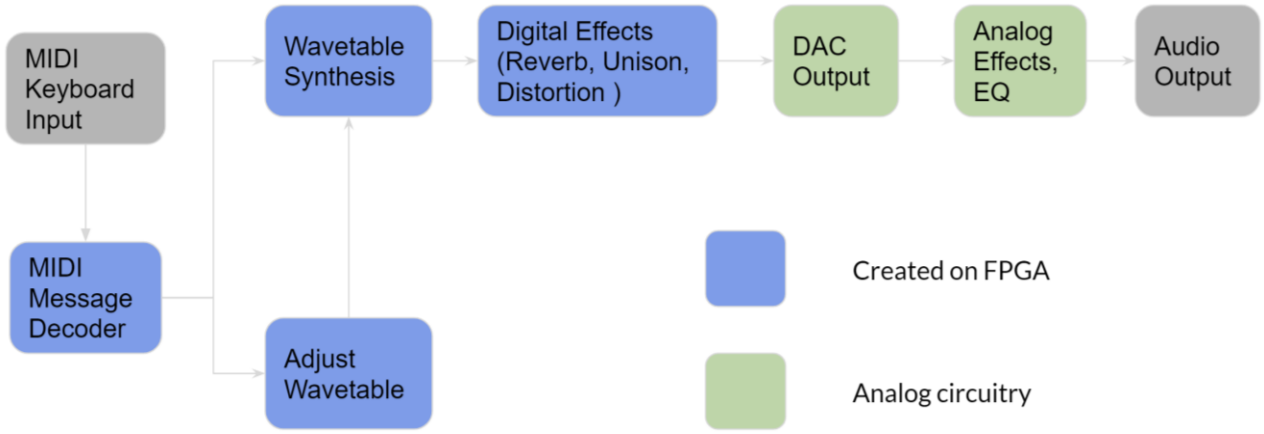


Fig. 1. System Block Diagram

III. Architecture and/or Principle of Operation

The way that the overall architecture is designed is that first the player interacts with the MIDI controller that we have purchased. This controller includes the keyboard for playing notes and each of the control knobs. The control knobs each have two functions controlled by a function key on the keyboard. Additionally, active wavetable selection will be controlled by the switches on the FPGA board.

From the MIDI controller, a MIDI control message is sent to the FPGA via serial UART and decoded into a format that the rest of our architecture can read. The MIDI control messages can either be a control knob, which controls the strength of different effects, or a keypress message, which signals that a key was pressed or released, as well as how fast the key was pressed.

The first stage is the wavetable synthesis stage which takes the MIDI control signals for the notes and converts them into samples to feed through the rest of the pipeline. It also takes in the controls to choose which two active wavetables to fetch samples from. The last input control that this stage takes in is the unison control knob which controls the distance from in-tune each of the unison voices is. The samples are generated from traversing the wavetable, containing multiple different wave shapes, and the stride of the traversal determines the frequency or pitch of the sound. These samples are mixed together according to the desired amount of wave blending and then fed through into the effects chain. The number of samples mixed together is dependent on the amount of wave-blending, the number of voices from the polyphony, as well as the degree of unison effect.

Next is the stage of digital effects applied to the samples. These begin with the distortion module. The distortion comes in two flavors, one that reduces bit depth (bit crushing) and another that reduces the sample rate. The bit crushing effect uses a control knob value that adjusts between a value from 0 to 15 that determine how many of the bits will be truncated to zeros from the sample. The sample rate reduction module is similarly controlled by a control knob and discards some number of samples to reduce the effective sample rate. Following the distortion effect is the delay effect. This operates by taking a control knob value to determine the length of the delay and another control knob value to determine the loudness of the repeated sound and then plays back all samples a second time that delay length later. Lastly in the effects chain is the reverb effect. This effect is very similar in operation to delay however, the control knob for reverb controls how strong the reverb is, adjusting the amount of attenuation between “wall reflections”. Once the digital effects have been applied the sample is adjusted to be compatible with the DAC interface and then converted to an analog signal.

After the sample has been converted to analog through the DAC support circuitry, it is sent to a bank of filters that act as an 8-band equalizer with high pass and low pass filters as the outside bands. Each of these bands can be mixed at a different level using potentiometers and the overall amplification level can also be controlled. Finally, this signal goes through one final stage of amplification in preparation for being fed into a speaker.

The wavetable synthesis itself will all be done on-board the FPGA, with the analog filtering done only as a final step. This is quite different than most comparable synthesizers, which do very little in the digital space and do most of the effects processing using analog components. While many music-makers place a lot of value on the idea of the “analog sound,” we seek to give our synthesizer output some of these qualities by having the final output stage be fully analog.

The FPGA board that we have chosen for the project is a Terasic DE0-CV board with an Altera Cyclone V FPGA chip on-board. The full details of why this board was chosen will be expanded on in the later sections of this paper, but the key benefits of this platform were its relatively low cost as well as the large amount of on-board block ram to facilitate the easy storage and retrieval of the wavetables themselves, as well as making the design of the delay and reverb effects much simpler. While we ultimately see this project as a potential prototype for a production model using a dedicated chip, the benefits of using a cheap FPGA platform allow the option of going into production with the FPGA platform in the future while staying cost-competitive.

IV. System Description

A. *MIDI controller to FPGA interface*

The first stage of the synthesizer system is the MIDI keyboard to FPGA interface. This stage consists of the MIDI keyboard, the support circuitry to convert the MIDI serial out to UART, the UART receiver module on the FPGA, and finally a UART decoder to convert the raw bytes into signals for the synthesis and effects pipeline.

The MIDI keyboard we decided to use for the project, the Stage Right by Monoprice 49-Key MIDI controller, was chosen for several reasons, the three most important reasons being the availability of rotary encoders, MIDI serial output, and low price. First was the number of rotary encoder knobs on the keyboard. The effects that we have on the synthesizer are all adjustable, for example, like the degree of blending between two wave shapes, the decay of the reverb, or the length of the delay, and we decided that leveraging on-keyboard rotary dials to control the effects would be the simplest method of control. Second was the availability of a MIDI out port. While the official MIDI standard specifies a 5-pin DIN connector as the primary connector, almost all modern MIDI controllers use a USB output instead, allowing the keyboard to connect easily to the desktop computer-based digital audio workstations instead. We wanted the MIDI controller we used to have the MIDI DIN output, since the signal sent out can easily converted into UART with some simple support circuitry, and we wanted to avoid falling into a trap of either designing a hardware USB controller from scratch or having to use a pre-made USB controller IP block, which could bring its own set of compatibility and integration issues. Unfortunately, this constraint severely restricted the pool of MIDI controllers we could use, since lower-end controllers only supported USB output and eschewed the legacy DIN connector. Finally, price was a major factor as well. Obviously, we need to stay under the $600 project budget, and one of the goals with our project is the keep the total parts cost as low as possible to stay in the same price/cost range as similar synthesizers. Many higher-priced MIDI controllers come with lots of unnecessary bells and whistles as well, and for our synthesizer, we only needed the most basic features along with the other requirements above. In the end, the Monoprice MIDI controller was the one of the only keyboards under $100 that satisfied the other requirements and did not have atrocious reviews on Amazon.

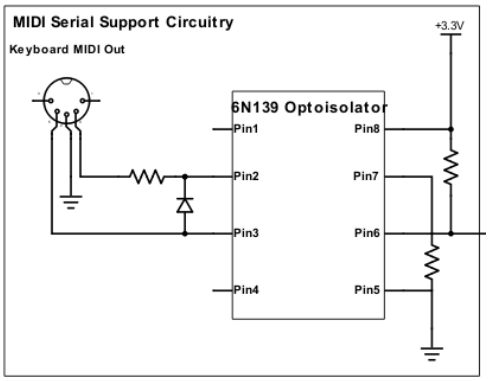


Fig. 2. MIDI Support Circuitry

The next component of the chain is the MIDI support circuitry. The DIN connector outputs a serial data stream using current on and off to represent zeros and ones, so support circuitry is required to convert this into a voltage-based signal. Luckily, specific circuitry for this is detailed in the MIDI specification, requiring a specific opto-isolator chip with some other passive components. Although the exact opto-isolator chip used in the specification is no longer produced, equivalent circuitry was easily available online, using an alternative model of opto-isolator chip (6N139) that was mentioned in the MIDI specification. The details for the circuitry are shown in the block diagram. The output of this support circuitry is a UART input line that is connected to the GPIO pins on the FPGA board.

B. *MIDI Message Decoder*

From here on out, the “components” of the synthesis and effects pipeline will be Verilog modules synthesized on board the FPGA until the DAC interface with the final analog filtering component of the pipeline.

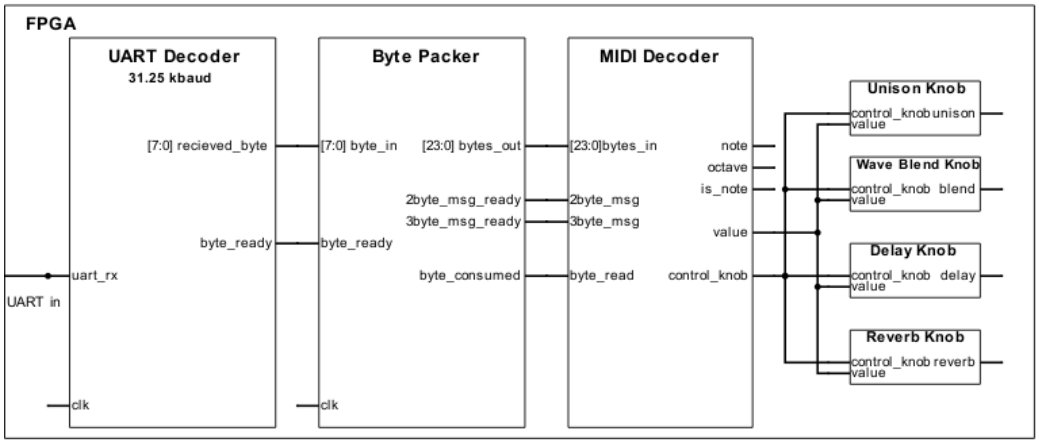


Fig. 3. MIDI Message Decoding Diagram

The first module of the pipeline is the UART receiver module. This module is fairly straightforward: it takes in the raw 1-bit UART serial signal as input and outputs a 8-bit data byte along with a 1-bit byte data ready signal. The baud rate of the MIDI transmission is specified in the MIDI specification as 31.25 kbaud, very slow compared to the system clock frequency we will be running the FPGA on. This module consists of a state machine to detect when the UART line drops low, signaling the beginning of the start bit, waiting half of a UART bit to start sampling in the middle of each bit, and then proceeding to sample again every UART bit. We anticipate to be running the system clock at 44.1 MHz in order to simplify the interface with the DAC later in the pipeline, so each UART bit will be slightly more than 1411 clock cycles. There is no worry of drifting out of sync due to the very large number of clock cycles per bit and that there will only be 10 bit per message before resynchronizing. The state machine will then proceed to read the 8 data bits, the one stop bit, and then either return to an idle state or detect a new incoming byte. The bits are pushed through an 8-bit serial in, parallel out shift register with the output connected to the output of the module. The format specified in the MIDI specification does not contain any parity bits, just one start and one stop bit per 8-bit data byte. Although MIDI messages are never just a single byte, this module is format agnostic and will only process a single byte through UART, asserting a ready signal after every byte. We will see in the next module how the full message is constructed.

The next module of the pipeline assembles the MIDI message from the individual bytes received by the first module. MIDI messages that we will care about for this project will come in two flavors, a two-byte message for sending rotary encoder information and a three-byte message for sending keypress data. Because the goal of this module is to assemble the whole MIDI message to pass on to the full MIDI message decoder, this module will need to decode the first byte of the message, the header byte, to determine which type of message is being received and how many bytes the message is. Each byte coming into the assembly module from the receiver is pushed through another serial in, parallel out 32-bit shift register which shifts in 8-bits at a time. The inputs to this module are the byte output and byte ready of the previous module, while the outputs of this assembly module are a 32-bit MIDI message bit-vector as well as a two-bit one-hot signal that signals both that the message is ready and how many bytes the message contains. A value of 2'b00 will mean that the output is invalid, 2'b01 means the output is a valid two-byte message, and 2'b10 means the output is a valid three-byte message, with 2'b11 being an unused, illegal output. The 32-bit MIDI message bit-vector will have the bottom 8 bits be zeroes if the message is only two bytes long. This assembly module will also take an acknowledgement signal from downstream modules signaling that the MIDI message on the output has been consumed.

The next phase of MIDI message decoding is translating this 3- or 2-byte signal into a more readable format for the purpose of digital synthesis. This decoder module ingests a MIDI message and translates it out into note name, octave, note control, control knob value, and velocity. Note name is the name of the note desired using the standard musical note naming scheme of A to G. All half-steps between notes will be named as sharps. The octave output is the octave that the note falls under ranging from 0 to 7. If the MIDI message that is being decoded is not a note message the default values for these two signals are C, and 0 respectively. Note control is a two-bit signal where the high bit represents if the message is a note control message, and the low bit is a 0 if the message is a note off message and is 1 if the message is a note on message. Control knob value is an enum encoding of the names of the various control knobs that control effects on the synthesizer. There is a bank of registers that represent the control knobs if the control knob value matched the name of the control knob velocity is stored in that register and it represents the level of that knob. For a note message velocity represents the volume of the note.

C. Digital Synthesis System

The second major subsystem of the synthesizer pipeline is the wavetable digital synthesis subsystem. This is the portion of the pipeline which takes in the control data sent by the MIDI messages and outputs a stream of 16-bit samples that are to be run through the effects chain. This MIDI control data comes into this portion of the pipeline in a format which has taken the 3-byte or 2-byte MIDI signal and translated it into the more readable format generated at the end of the MIDI control subsystem.

The beginning of this chain is the polyphony control module. This module takes in a note name, from the conventional note naming scheme of A to G with all half-steps named as sharps, the octave number for the note, the velocity value for the note, and whether the signal is a valid note, a note on, or a note off message. Using this information, the polyphony control module registers the note name, octave level, and velocity value for up to four simultaneous notes. These values are held for the duration between when a note on message is sent for that note until when a note off signal that matches the note name and octave of one of the notes being held is seen. Four note polyphony was chosen because it was a middle ground between the single voice note playback that many synthesizers utilize and the complexity that comes with many more voices. With four playable notes a standard triad chord can be played with a single voice melody line being played on top. Should a fifth note be pressed while all four voices of polyphony are playing notes the note that has been played the longest will be replaced by the new note. When this control module then receives the note off signal for the note that has already been removed from the set of held notes the state of the polyphony control module does not change.
From the polyphony control module each of the four notes is sent to its own incrementor module. This incrementor module uses the name of the note and its octave value to determine a pair of wavetable memory addresses to access samples from. These addresses start at zero and then are incremented by the value that will traverse the wavetables at the rate desired for the note being played. This is achieved by using the name of the note to determine the incrementation value for that note at octave zero. Since a note of the same name played an octave higher is twice the frequency, from the base incrementation value at octave level zero this value can be doubled for the

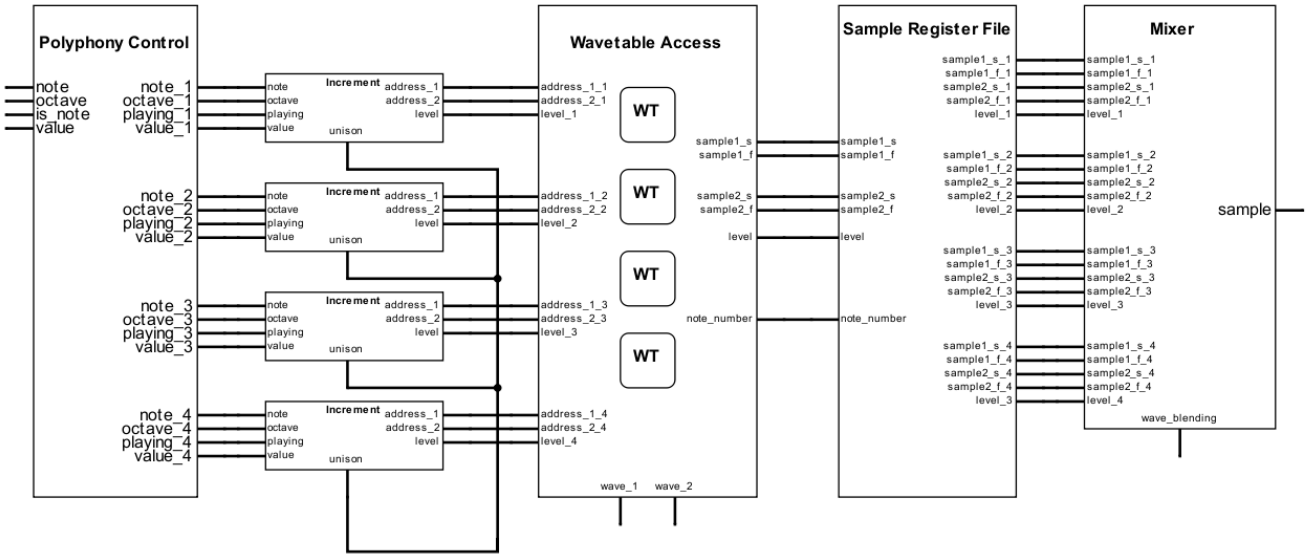


Fig. 4. Digital Synthesis Diagram

incrementation value at octave one or quadrupled for the value at octave 2 and so on. The base incrementation value is computed based on the wavetables containing a single period of each waveform being read at the system clock. Two addresses are generated for the purpose of a unison effect. This is an effect where instead of the desired note being played at the standard frequency, two notes are played, one at a slightly higher frequency and one at a slightly lower one.

These addresses are then sent to the wavetable access module. This module takes each of these addresses and sends them to M10k block-RAM containing each waveform. Each note has a copy of each of the four waveforms to access samples from. The M10k block-RAM is configured as a 2-ported ROM so that each module can have a port dedicated to each of the two unison addresses. All samples in the wavetables are 12- bit sample being read as a 16-bit sample. The remaining 4-bit are reserved for overhead involved in mixing the samples and applying effects. There are four possible waveforms available, a saw wave, a sine wave, and two more complex shapes generated for the user. For any given configuration of the synthesizer two wavetables will be active at a time. These active waveforms are selected by the player using the switched on the FPGA itself. Because of this every clock cycle 16 samples are fetched from the wavetables. This breaks down as 8 samples per active waveform and 2 samples for each of the 4 notes being held.

Once the 16 samples are fetched from the wavetable memory, they are sent to the mixer module. This module takes in each sample and the velocity value for its respective note. It then weights each notes sample by its velocity level and adds together all the samples for each of the two waveforms. Then taking the value provided by the blending control knob adds the resultant samples for each of the waveforms. The blending control knob determines how much of each of the two active waveforms is desired. At a value of 0 only active waveform one is played, at a value of 127 only active waveform two is played. The output of the mixer is buffered and updated on the sample clock as opposed to the system clock. This is done so that the effects chain which involves a delay effect and a reverb effect that both need to use memory must store the minimum number of samples to achieve their effects.

D. Digital to Analog Conversion

Our digital signal now needs to be converted into an analog signal so that it can go through our equalizer filters and become and actual sound through the speakers. In order to do this, we chose a digital/analog converter stereo DAC: the PCM 1793. The stereo portion of the DAC is not made use of as our design currently does not support stereo noise. Furthermore, the voltage lines are powered from the FPGA board, which removes the need for an outside power source in addition to the one needed for the FPGA. Shown in Fig. 3. below is the pinout for the FPGA. In Fig. 4 we have the connections needed for the pinout DAC to work with our system. In green are the FPGA connections, in yellow are the power sources, which will also come from the FPGA. The gray pins denote ground, while the red and blue pins are the two sound outputs. As mentioned before, the DAC supports stereo, but our circuitry currently does not, and as such, we only use one of the two output pins.

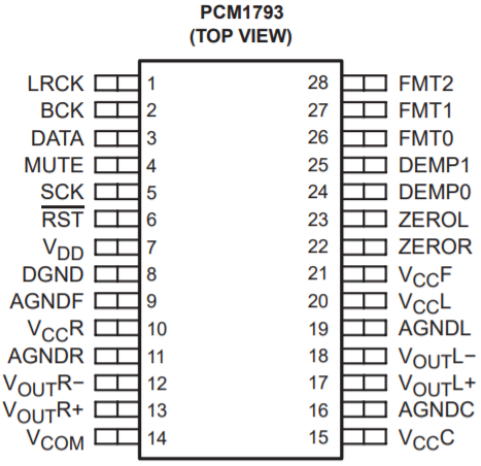


Fig. 5. DAC DIP chip pinout

Pin Number	Pin Name	Connection	Pin Number	Pin Name	Connection
1	LRCK	FPGA GPIO	15	VCC_C	5V FPGA
2	BCK	FPGA GPIO	16	AGNDC	GND
3	DATA	FPGA GPIO	17	V_OUT L+	OUT
4	MUTE	FPGA GPIO	18	V_OUT L-	OUT
5	SCK	FPGA GPIO	19	AGNDL	GND
6	RST_N	FPGA GPIO	20	V_CC L	5V FPGA
7	V_DD	3.3V FPGA	21	V_CC F	5V FPGA
8	DGND	GND	22	ZEROR	FPGA GPIO
9	AGNDF	GND	23	ZEROL	FPGA GPIO
10	V_CC R	5V FPGA	24	DEMP0	FPGA GPIO
11	AGNDR	GND	25	DEMP1	FPGA GPIO
12	V_OUT R-	NC	26	FMT0	FPGA GPIO
13	V_OUT R+	NC	27	FMT1	FPGA GPIO
14	V_COM	GND	28	FMT2	FPGA GPIO

Fig. 6. FPGA value table

According to the specification for the PCM1793, many of the signals coming from the FPGA are relatively simple to generate. Almost all of them are clocks, which are much slower than the FPGA's clock, or held signals which are set to the same value always when received by the DAC. The DAC will communicate over I2S specification, which will also be controlled by the FPGA to the data pin on the DAC. Verification for this part will be by sending a known note to the DAC and measuring the output waveform to ensure that it is the note specified.

E. Filters and Equalizer

After we have converted our digital signal into an analog one, we wanted to create a simple equalizer to give the user greater control over the sound output. This equalizer also has a volume control tacked onto the end. As a baseline target, we wanted to be able for the user to control sounds from a note referred to as A0, to a note referred to as C8. These notes are denoted as the tone followed by an octave, and the range that we chose gives us a range equivalent to that of a piano. The equalizer has eight different ranges to mirror the number of equalizers in a normal synthesizer. Shown in Table Fig. 7. below are the ranges of each filter. Regions 1 and 8 are the final low and high pass filters that we wanted to use. Currently, they are fixed filters, but the design allows for us to potentially control where their cutoffs are and make them variable cutoff filters, which is a feature that many on-the-market synthesizers have.

Region Cutoffs	Low (Hz)	High (Hz)	Low (rad/s)	High (rad/s)	Lowest Note in Range	Highest Note in Range
Region 1	0	51.53959	0	323.8328324	-	G1
Region 2	51.53959	96.59381	323.83283	606.9168495	G#1	F#2
Region 3	96.59381	181.0329	606.91684	1137.463609	G2	F3
Region 4	181.0329	339.2860	1137.4636	2131.796905	F#3	E4
Region 5	339.2860	635.8786	2131.7969	3995.343682	F4	D#5
Region 6	635.8786	1191.742	3995.3436	7487.941794	E5	D6
Region 7	1191.742	2233.525	7487.9417	14033.65437	D#6	C#7
Region 8	2233.525	4186	14033.654	26301.4137	D7	-

Fig. 7. Filter Ranges

In order to create these, our circuitry implements two-stage filters for each frequency range. The reason for this is that it is simpler to create a low-pass combined with a high-pass that have sharp cut-off frequencies than to create a bandpass with less sharp cutoffs. Furthermore, it allows us to separately design each filter, which makes testing and fixing any errors that have been made much easier. The overall circuitry is shown in Figure Fig. 8., but with only two band-pass filters shown rather than the 6 that we intend to have when our design is completed.

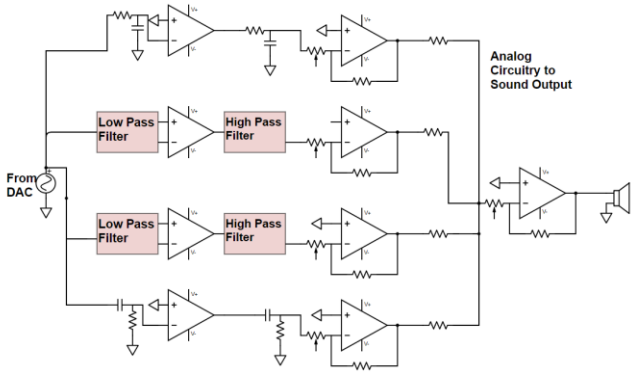


Fig. 8. Equalizer design

The low-pass and high-pass filters are in Cauer topologies and have a third-order pole at the given cutoff frequency. This pole allows us to achieve 60dB/decade roll-off, which is important in ensuring that sound frequencies that we do not want to pass through are not passing through. The topologies for the third-order low-pass and high-pass filters are relatively simple in terms of design and implementation. Shown in figures Fig. 9. and Fig. 10. are the third-order filters.

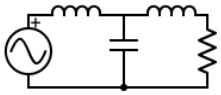


Fig. 9. Low Pass

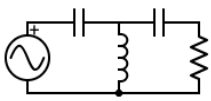


Fig.10. High Pass

In terms of overall design, we wanted to add as many layers of buffers as possible in order to ensure that there will be minimal loading between all the stages of filters that we have

designed. This also allows us to test each filter independently and then combined in order to help smooth integration. After the two stages of filters that we have, we have an effectively variable gain op-amp whose gain is controlled by the potentiometers. These allow us to create an equalizer which can be used to boost or attenuate certain frequencies and allow the user to create unique sounds. Finally, we combine all our signals via a summing amplifier, which also has variable gain, and send the output to a speaker. The speaker is yet to be determined, but the chosen speaker will determine what amount of gain is needed at minimum on the summing amplifier at the end.

F. Power

In terms of overall power use, we expect to be easily within the parameters of the FPGA where the power for all our analog components will be drawn from. The relevant active components currently are the DAC and all op-amps. In all, we use 17 op-amps in our design. According to the specification for the op-amp that we have chosen, the OPA 1692, the maximum current that it should draw from its power rails is 975µA. Our power rail for the op-amps will be 5V and ground coming from the FPGA. For the DAC, the 5V rail can draw up to 8mA according to the specification while the 3.3V rail can draw up to 16mA. According to the specification on the FPGA that is being used, any given power pin can supply up to 5W of power. That means that on the 5V rail, we can draw up to 1A and on the 3.3V rail, we can draw up to 1.515A. Our design currently draws 17*.975mA + 8mA = 24.575mA from the 5V rail at worst, and 16mA from the 3.3V rail at worst. Even if our design ends up having a lot of loss between the power source and the op-amps/DAC, we have more than enough clearance on the power use of our power to support all of our circuitry.

V. Project Management

A. Schedule

Our breakdown of work is relatively simple. We wanted to ensure that everyone was scheduled in their comfort zone where they would be able to do their best work. Furthermore, we wanted to make sure that everyone had enough slack available so that they would have some extra time to do their work if necessary. Our schedule can be seen in the figure below. It may be a bit difficult to read, but in yellow are tasks that require everyone to pitch in to complete, in green are tasks for Hailang, blue tasks for Jens, and red tasks for Charles. Additionally, a larger version of the schedule can be seen after the references section. Each person should have roughly two weeks of slack for themselves, while the overall project where everyone might need slack has around another added week of slack.

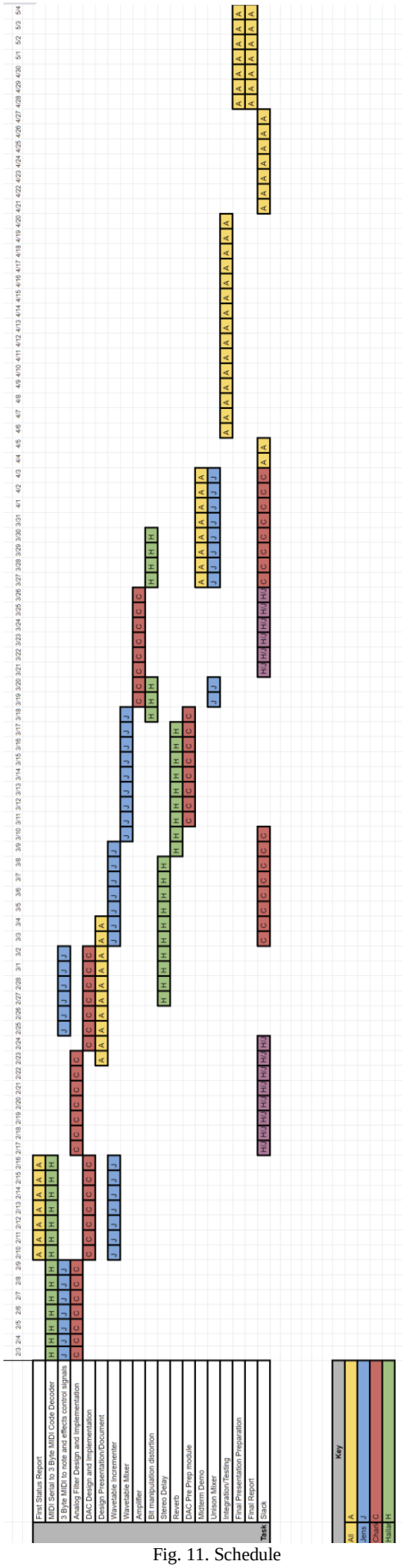


Fig. 11. Schedule

B. Team Member Responsibilities

In terms of work breakdown, Jens has a work focus on converting signals into digital signals. His focus is on determining what needs to be read from the wavetables which store digital sample values in the distributed block ram.

Hailang's focus is on digital logic manipulation, taking a sample value and applying different effects to that to get a final digital value. Charles' work focuses on the analog side of the synthesizer. His work involves converting the digital value through a DAC to get to an analog signal and filtering that signal to reach the speaker and an end sound.

As such, we broke down our project into several parts, which are distributed as described above. The distribution of work can be seen in the table below.

Jens	**Charles**	**Hailang**
MIDI message decoder	Analog filter design	MIDI to FPGA interface
Wavetable Incrementor	DAC support circuitry	Delay effect
Wavetable blending and mixing	FPGA to DAC interface	Reverb effect
Unison mixing	Amplifier design	Bit distortion effect

Fig. 12. Work Distribution Table

C. Budget

Our budget will mostly comprise of analog pieces. The primary non-analog part bought is the keyboard that we will be using to transmit our MIDI signal to our FPGA. All other parts will be analog parts. The op-amps chosen are just over a dollar in cost and impact our budget relatively little. All passive components should sum to a total of no more than $50, though the specific components have not been bought yet. This is because passive components are generally extremely cheap in cost. As such, our total overall cost is budgeted to be lower than $100 currently.

Item	**Price**
Monoprice MIDI Keyboard w/ serial output	$50
Terasic DE0-CV FPGA board	$100*
PCM1793 DAC	$6.40
6N139 Opto-isolator	$1.80
Various Passive Analog Components	$30**

*Is not subtracted from $600 budget
**Components not chosen, price estimated

Fig. 13. Estimated Bill of Materials

VI. Related Work

In researching similar work to the synthesizer that we wished to create we used three specific works for reference. Those were: Serum, the Waldorf Blofeld, and the Waldorf Quantum. These three synthesizers were chosen as they all use wavetable synthesis to produce sound but are very different in the markets they target and the features they provide.

Serum is a software wavetable synthesizer plugin made by Xfer Records and is the industry standard wavetable synthesizer for electronic music production. It is built on the core feature of being able to blend and manipulate waveforms. Additionally, it comes in at a relatively price point of 300 dollars. However, it is locked into a software environment that requires a significant amount of additional software and compute power to run to its full potential.

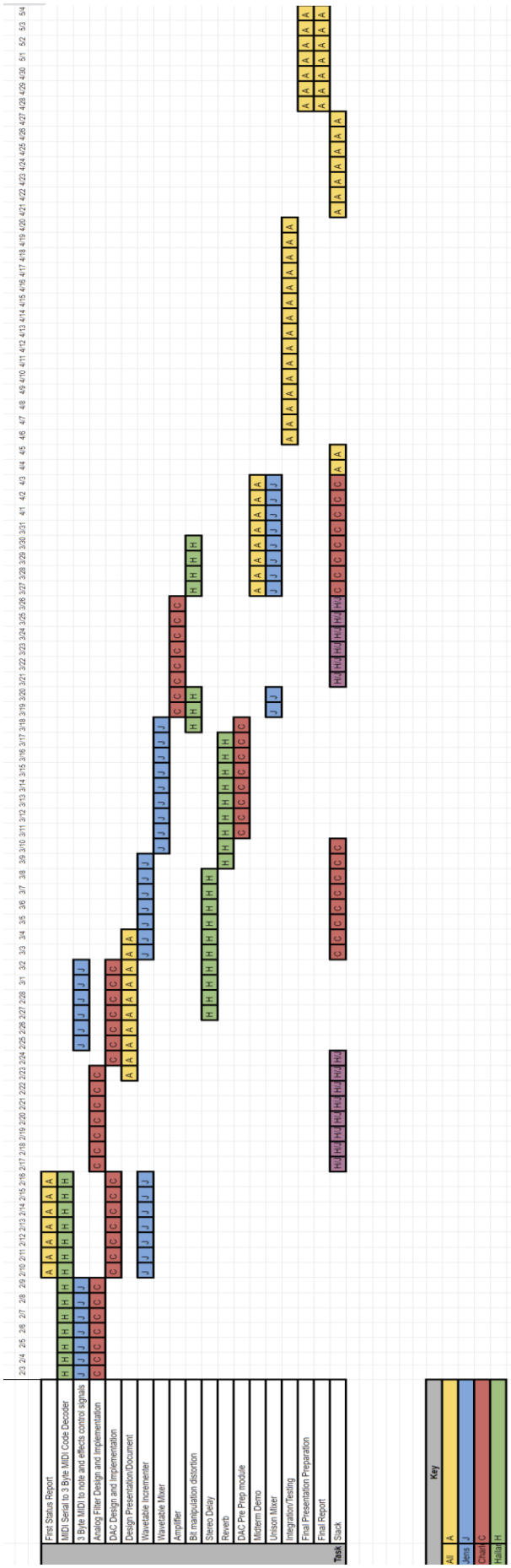
The first of the hardware synthesizers that we compared to was the Waldorf Blofeld. This synthesizer was chosen for its low price for a hardware synthesizer of 400 dollars. However, the core feature that our synthesizer targets of waveform blending is completely absent from this synthesizer. This lead us to attempt to find an example of a hardware synthesizer that did include this core feature.

Moving up Waldorf's product stack we arrived at the Waldorf Quantum, a wavetable synthesizer with an over 4000-dollar price tag. This synthesizer did include wavetable blending effects but also included copious other effects to justify the extreme price. This confirmed that there was a vacancy in the market for a hardware wavetable synthesizer that focused on wave manipulation effects while achieving a lower budget price target.

References

[1] Xfer Records xferrecords.com.
[2] Waldorf www.waldorfmusic.com/en/
[3] MIDI Spec www.midi.org/specifications

APPENDIX A. WORK SCHEDULE



APPENDIX B. MODULE DIAGRAMS