The Exploration of Hybrid Synthesizer Architecture and Implementation

Collin Champagne, Alex Scarlatos, Margaret Schedel (faculty adviser)
Department of Music, College of Arts and Sciences, Stony Brook University, Stony Brook, NY

Overview:
The synthesizer has risen in the age of computer music as a fundamental tool for producers and editors in the sound studio, as well as for hobbyists and musicians alike. The late 19th century saw the advent of the first analog synthesizers – with digital synthesizers dominating the modern market. Our concept is to explore the necessary construction for the synthesis of these two categories of musical instruments through the application of computer science, electrical engineering, and music. This project uses a microprocessor to receive input from an external keyboard as well as a physical user interface integrated into a custom designed enclosure. The microprocessor then uses this data to generate a MIDI signal that is then converted into an audio signal to be processed by various analog modules. These modules include various filters and oscillators to produce a sound that is similar to that of purely analog synthesizer. The objective of this dichotomous relationship is to simplify the implementation of an analog synthesizer while maintaining an authentic sound and providing a higher degree of customization and precision in sound design.

Why We Chose Hybrid:

Digital technology has changed the world immensely. Manipulating bits at unimaginable speeds it just too convenient, and digital tech has invaded every aspect of our lives, from our phones to our coffeemakers.

Digital synthesizers have all but replaced their analog predecessors. They can store virtual instruments and modify a huge number of audio parameters, all in one box without additional hardware needed for each new set of features. Many purists swear by their analog synths, hoping to avoid digital downfalls like wave approximation (the fact that a digital processor cannot create a perfectly smooth wave). But our reason for combining the two was simply that it was most practical.

While a digital synth would be the most powerful option, we discovered through our endeavors that it would be harder to implement than expected. A tiny microprocessor is not an all-powerful modern computer, and is limited by memory and speed. It couldn't handle all the responsibilities of a synth on its own, so we decided to take the best of both worlds.

We gave the digital component the responsibility of generating and enveloping audio, and passing a digital signal off to an analog chain. The chain could take care of cleaning up and effecting the signal in any way it was then designed to. This seemed to be the perfect compromise, and provided the perfect opportunity for a computer science/electrical engineering collaboration.

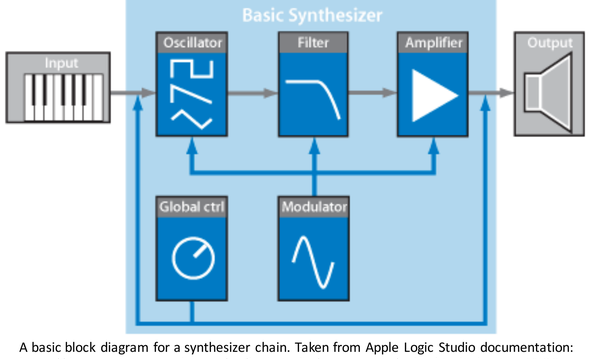
Instructions

The knob on the far left controls the volume.

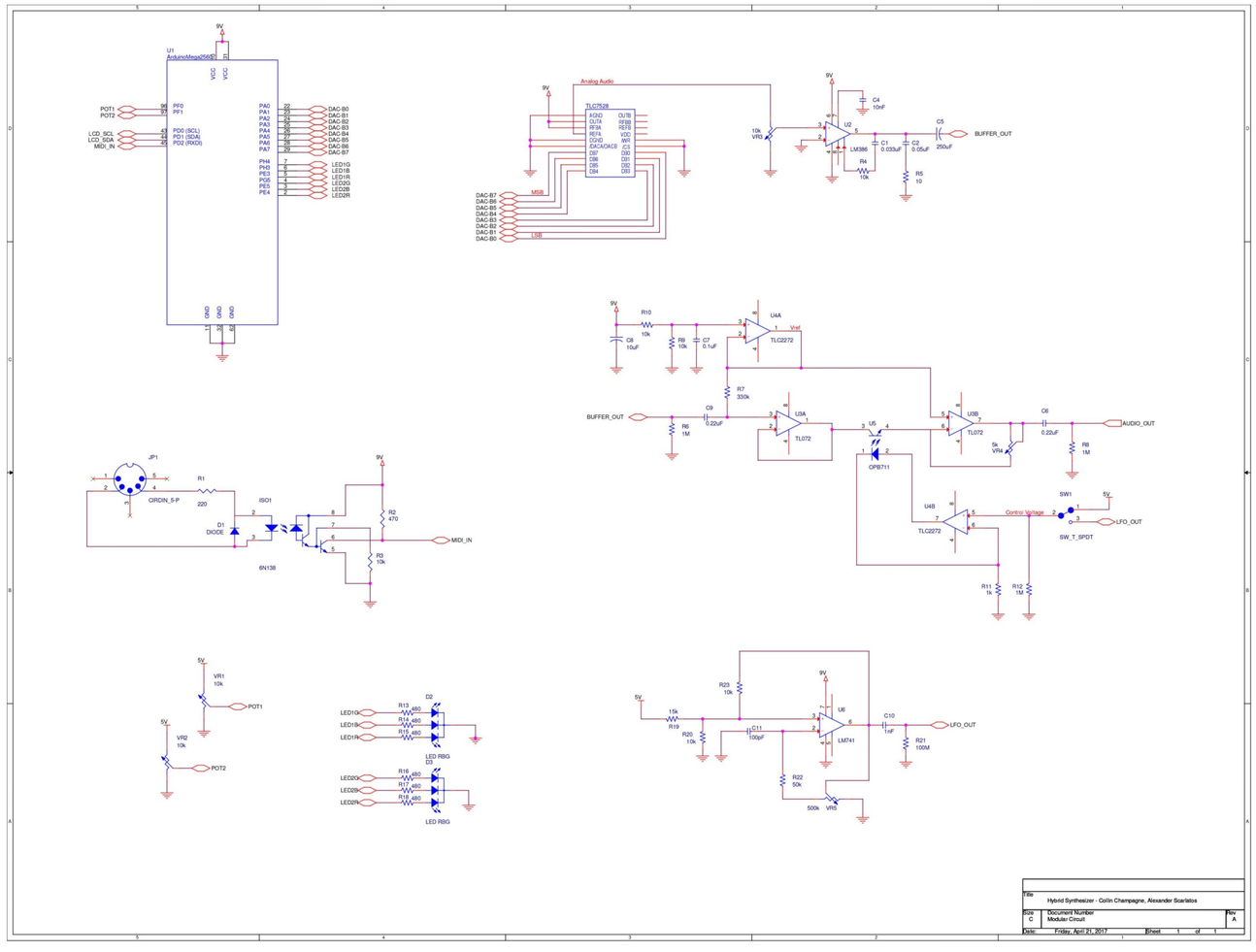
The two knobs under the LCD display control the digital settings:

- Left: Controls the editing mode of the audio generator
  - Off – no sound will be generated
  - Preset – select from a group of precompiled settings
  - Wave Type – choose the waveform that will be generated
  - Attack – control the attack of the envelope (in ms)
  - Decay – control the decay of the envelope (in ms)
  - Sustain – control the sustain of the enevelope (min: 2, max: 20)
  - Release – control the release of the envelope (in ms)
- Right: Controls the value of the selected mode
  - Ex: If Wave Type is selected with the left knob, use the right knob to select between Sine, Square, Triangle and Sawtooth.

The knob on the far right controls the frequency of an LFO and the switch bypasses it.



A basic block diagram for a synthesizer chain. Taken from Apple Logic Studio documentation: https://documentation.apple.com/en/logicstudio/instruments/index.html#chapter=A%26section=3%26tasks=true



Audio Generation:

Our synthesizer generates audio using digital waves originating from an Arduino ATMEGA2560.
With an audio loop running at 40kHz, an 8-bit amplitude signal is sent to a DAC (Digital to Analog Converter) in the circuit every cycle.

When the microprocessor receives a MIDI note, it is converted into a frequency, and then further processed depending on the waveform.

- Sine – The ATMEGA2560 is powerful for its size, but is not capable of performing 40k sine operations per second. So we generated a list of byte values for a sine wave centered around 2.5V, and every step that list is queried at an index relative to the frequency of the current pitch. We got the idea for this method from an Instructable by Amanda Ghassaei (http://www.amandaghassaei.com).
- Square – The simplest and most accurate waveform. A square wave can be generated by returning high voltage for half of the period and low voltage for the other half. A simple number of cycles after which to switch voltages will handle this with high accuracy. We expect this is why old games using MIDI soundtrack tended to use square waves. So if you know the Super Mario theme, play it on this setting!
- Triangle/Sawtooth – A sawtooth wave is just comprised of a linearly rising amplitude throughout its period, and when it reaches max amplitude resets to 0. A triangle wave is almost the same, except that it reaches max amplitude at half its period, and then decreases for the second half of its period to 0. We generated these by storing an amount to increment the amplitude by every cycle.

A basic ADSR envelope fires whenever a note is hit, and every frame determines a cutoff amplitude for the audio output.