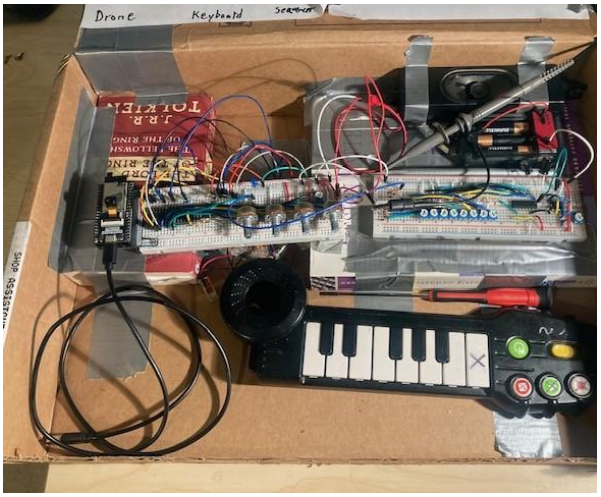
Digital Wrover-ESP32 Synthesizer with Analog 555 & 4017 8-step Sequencer

Jake Simonds
CS7980 Embedded Systems
Northeastern University – Roux Institute
Portland, ME
simonds.j@northeastern.edu



I. Intro

This paper describes the technical details and motivations for building a synthesizer as a beginner-friendly embedded systems project for a software-experienced person learning hardware systems. The synthesizer incorporates Integrated Circuits (ICs) (555 & 4017), an ESP32-wrover executing digital logic & periphery components, including a keyboard harvested from a toy musical instrument. Code and more information is available on GitHub.

Keywords—555, 4017, ESP32, digital/analog, synthesizer, frequency, pitch, education.

II. Design

Synthesizers can be entirely digitally, entirely analog, or hybrid & attain similar qualities of sound. This specific system is digital with analog inputs.

One early consequential design decision was to use an ESP32-Wrover instead of the Raspberry Pi I had initially planned to use since I am more familiar with that ecosystem. I made the switch because the ESP32 natively supports analog reads and writes, and so that allowed me to make waves beyond square waves and also take in analog inputs. Another major design decision was to build modularly. A secondary goal of my project is to build things that work both together as one big system, but also potentially could be broken apart and combined with off-the-shelf synthesizer equipment, particularly from the euro-rack world (where there's lot of documentation and an ethos of finding ways to make systems talk to each other). As a result, my final project contains three modules: the ESP32-Wrover, the Sequencer, and the Keyboard.

Two final smaller but still worth mentioning design decisions were to use the ESP32's native power as a source and to harvest my keyboard from a children's toy. The decision to use the ESP32-Wrover's 3.3 volt power was not made lightly as its a pretty major diversion from Eurorack-style synthesizers, which as an ecosystem was a major influence and source of technical details. I made the decision to stick with my existing power supply once I determined that there was more than enough to explore with a limited power supply, and exploring those areas (as seen later in this project) were of more interest and in closer alignment with my coursework than bootstrapping a power supply from +12 to –12 volts (as is standard for eurorack).

Using the keyboard from the toy was a choice that satisfied just my own curiosity in reverse engineering, but had a net-negative effect on the quality of my final project. My reverse engineering is imperfect and introduced bugs which I had to find software ways of hiding (by turning the keyboard off basically in most modes of operation).

III. Build Process

What follows is a brief description of each of the three modules of my system: the ESP32-Wrover, the Sequencer, and the Keyboard

A. ESP32-Wrover Module

The 'brain' of this project is an ESP32-Wrover Module with an ESP32 chip. This device is compatible with Arduino IDE software, and all code for this synthesizer was C-like and written in '.ino' files which were then flashed to the device. The ESP32 module can be thought of as a central command unit that takes in analog (from the Sequencer and potentiometer) and digital (from the Keyboard) inputs and then uses that information to alter the sound wave being generated.

The sound wave is generated entirely digitally, and then output via an analog pin on the hardware device from 0 to 3.3 volts. That current goes through a volume control potentiometer and then straight to the speaker circuit.

B. Keyboard

The keyboard for this synthesizer comes from an electronic music toy. The original toy was dismantled & parts with identifying marks were disposed of unfortunately before I realized how central the keyboard would be to this project--part of what I liked about it, compared to other keyboards from other toys, was that the keyboard itself was entirely modular from the rest of the toy and had a simple 8-wire ribbon connector. Noticing that the keypresses for the keyboard were simply completing a circuit, I then experimentally found a system that, while imperfect allowed me to digitally read keypresses. The system I devised works like this:

DigitallyReadKeypress:

```
GPIO_OUT_SIGNAL = {a,b,c}
GPIO_IN_SIGNAL = {d,e,f,g}
KEY_DICTIONARY = {}

For GPIO_OUT_SIGNAL in GPIO_OUT_PINS:

    Send HIGH signal on GPIO_OUT_SIGNAL

        For GPIO_IN_SIGNAL in GPIO_IN_PINS:

   read GPIO_IN_SIGNAL

        if READING:

                write {GPIO_OUT_SIGNAL,
                GPIO_IN_SIGNAL}        to
                KEY_DICTIONARY
```

The dictionary is then interpreted and matched against known patterns for known key presses. The keyboard also has a switch and a button and an LED light which I was able to similarly read in as digital signals. This was never a perfect process, and took a lot of inelegant code adjustments to avoid stray unintentional signals.

C. Sequencer

The sequencer module is a stand-alone analog device that converts a 4.5 volt input voltage from three AAA batteries into a < 3.3 volt variable 8-step sequence of voltages via a 555 chip (to turn the initial voltage into a square wave), a 4017 decimal counter chip (to create the 8-step sequence), 9 potentiometers (8 one-to-one with each step of the sequence and 1 to control the tempo of the 555-output square wave), and a considerable number of diodes used to keep voltage flowing only in the direction intended (there's lots of places where, without diodes, you get voltage flowing unpredictably). The final step is a two-resistor voltage divider which I included as GitHub a precaution since I'm starting with 4.5 volts and am uncertain how much voltage I am or am not taking out in the process of the sequencer, but I do know that my ESP32 has a maximum analog input of 3.3 volts.

The basic 'story' of this sequencer is:

1: We start with a steady 4.5 volt flow from three AAA batteries (this was chosen arbitrarily because it was what I had available for a power supply)

2: A 555 chip takes this flow and turns it into a square wave, variable in tempo based on a potentiometer.

3: The 4017 chip takes in this "clock" signal and outputs a signal off of its 1-8 legs in sequence, finally with the 8th step outputting also to the 'reset' leg starting the sequence again.

3a: Each of these 1-8 output 'legs' is connected to its own potentiometer, which can variably add resistance. Through diodes the outputs then all come back together to then leave the sequencer through the same channel.

4: The back-together-again stepped sequence goes through the two-resistor voltage divider, and then proceeds as an analog signal to be read by the ESP32 on an analog input pin.

IV. Experimentation

A. Things Attempted But Not Included in Final Device

The first iteration of this project was a simple c-file that wrote a variable square wave to a GPIO pin on a raspberry pi, which was connected to a toy speaker. In terms of bang-for-your-buck this was an excellent experiment, but I quickly discovered that without analog the square wave was all I could produce.

I also used a 555 chip to generate a tone. This was interesting, and numerous ways to do this can be found on YouTube. I did not pursue this as for my skill level doing the sound generation digitally allowed a lot more flexibility of ways to intentionally alter that sound generation.

I dismantled lots of toy keyboards before finding the one I used for this project. This was educational and fun, but now having worked extensively with this found item I'd discourage myself in the future or anyone trying a similar project like this from relying so much on something undocumented and with known bugs.

B. Overview of What I Finally Tried to Build

This project had three goals:

I. As a system it is a self-contained musical instrument 'friendly' enough to be played by people who might not know much about synthesizers (specifically thinking of my nieces aged 4 & 2).

II. At least one module of this system can interface with existing musical equipment (either the ESP32 taking as input some pre-existing module that outputs an analog signal or the Sequencer or Keyboard interfacing with an existing professional piece of equipment), and the entire system can output a signal on a

3.5 mm jack (so it could be a real live instrument for a musician).

III. It is a 'real synth' in that a skilled electronic musician could use it to make complex sounds.

V. Results of Experimentation

Evaluation of the three goals:

I. This goal was partially achieved in that the synthesizer can be played by keyboard, but the keyboard is buggy enough that I personally find it frustrating to use. If this project were to be continued improving the keyboard (or replacing the entire component) would improve the device greatly.

II. This goal was achieved by successfully interfacing my sequencer as a stand-alone device with a Behringer Cat analog synthesizer (Demo). After determining the voltage levels of my sequencer were safe for the device, I attached the sequencer via 3.5 mm to the input channels of the analog synth.

III. This goal was achieved, and wasn't that hard in retrospect. If anything, an increased focus on quality over quantity of sound options may have resulted in a better instrument.

VI. Conclusions and Future Work

I recommend digital-analog hybrid synthesizers on Arduino-like hardware as a fun, educational project for software engineers interested in learning more about hardware. Future work could include circuit bending, replacing the keyboard of this project with something more reliable, adding some sort of microphone/audio input, and for me personally learning more about power supplies for eurorack set ups.

References

[1] Moritz Klein, "Designing a simple 5-step sequencer from scratch,"**https://www.youtube.com/watch?v=vHNQQ6yUGyo&list=WL&index=43&t=654s**

[2] Ideas TV, "LED Chaser Using 4017 and 555 IC (Breadboard Tutorial) | Electronics," https://www.youtube.com/watch?v=wwfvmEk83K4.

[3] Beginner's Guide to Eurorack https://www.synthmode.com/content/beginners-guide-eurorack

[4] Element14presents, "How 555 timers Work – The Learning Circuit," https://www.youtube.com/watch?v=oZzjmAbyyIQ

[5] ESP32-Wrover Module github, https://github.com/Freenove/Freenove_ESP32_WROVER_Board.