Software Modular Synthesizer

Computer Sound Production Laboratory Assignment Final Report

Faculty of Computer And Information Science, University of Ljubljana

***Abstract*—Modular synthesis is a rather complex and expensive part of the analog synthesizer world. We present a simple and open source digital alternative. Our solution consists of six different modules that can be patched together to create countless different sounds. The user interface is light and beginner-friendly with helpful tooltips. In this report we present our implementation, all of the modules and how a new module can be developed. We show how a sound can be created and present a way of evaluating the modules to ensure they work correctly.**

I. Introduction

Synthesizers are electronic musical instruments that generate audio signals [3]. They do this by producing a waveform, like a sine wave, and then shape it using different techniques. The first proper analog synthesizers were created in the 1960s [2], but they were replaced by digital synthesizers in the 1980s. As of then, they have been widely spread in the computer sound production world. Here we will focus on *modular synthesis* - a popular technique in the analog domain, where one takes a collection of different smaller synthesizers called *modules* and connects them together with cables to route the audio signal. This allows for a lot of creativity and freedom in creating new interesting sounds. There exists many different modules ([1] presents some of them) which makes this field of sound production complex. On top of that, building such a system can be rather expensive. The goal of this project was to design a simpler all-in-one software alternative to a modular synthesizer that can easily be expanded by adding new modules. In this report, we will first look at two standard software modular solutions. Then, in the *Methods* section we will present our implementation, along with all of the modules, and talk about how a new module can be added. Finally, we will demonstrate the creation of a basic sound in *Results* and show how we can visually evaluate our solution within the software.

II. Related work

Some standalone modular synthesizer software solutions already exists. A standard one, being in development for almost 30 years, is *Reaktor* [6] by Native Instruments. It is a professional solution with lots of features, but it is still a paid application and therefore it is not open source. However, it is still way cheaper than an analog modular system. An alternative is *Rack* [7] by VCV. Although there exists a paid version, there is also a free version that is open source and allows for creation of custom modules. A problem with this application might be that the code is relatively big and it might be complex to add a new module, although we haven't tried it ourselves.

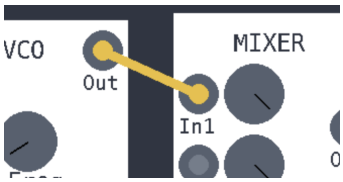

Fig. 1. An example of a connection between modules.

III. Methods

This application was developed in the programming language *Python* with the help of libraries *PYO* [4], a library for digital sound production, and *PyGame* [5], a library for visualizations.

Let us first talk about the core of our application - modules.

A. Modules

Each module has (at least some of) the following components:

- *Inputs*. These can either be a sound that comes into the module and gets processed, or a signal that controls one of the parameters of a module.
- *Potentiometers*. These are rotary knobs that allow us to manually control the parameters of a module.
- *Outputs.* These output a sound/signal.
- *Indicator lights*. These can show the status of some internal parameter of a module.

We make sounds by connecting modules together with "cables" that link outputs to inputs. We do this by simply clicking on one pin and then on another. An example of such a connection is shown in Figure 1. On The same figure we can also observe how a potentiometer looks like. PYO has the connections implemented, which makes our job easier. For example, if we have a Sine oscillator, we can set its frequency parameter to another Sine oscillator to make a connection.

We have designed a *Base module* that handles all of the above mentioned features, plus the visualizations and user interactions. Each module is a child class of this base module. We proceed to present all of the modules we have implemented. Their visual presentation will be shown in the *Results* section.

1) Master out: Master out is the default module that cannot be removed. It has a single input pin and it sends the audio from it to the computer's sound card. It also has a frequency-magnitude diagram displaying the output audio spectrum. This is useful for testing if our modules work correctly and will be mentioned more in the *Evaluation* subsection of the section *Results*.

2) VCO: The Voltage Controlled Oscillator (VCO) is the basic sound generator. It can produce four basic waveforms - sine, saw, square and triangle, each giving a different tonal character. The waveform and the frequency (pitch) can be changed with potentiometers. The module has a single output pin, sending out the produced sound. There are two input pins for controlling parameters - one for amplitude and one for frequency. We can change these with the LFO, presented next, but we can also feed one VCO's output to another's frequency input to step into the world of FM (frequency modulation) synthesis that gives us more metalic sounds.

3) LFO: The Low Frequency Oscillator (LFO) is used to control parameters. It operates the same as a VCO, but at a much lower frequency. When used on input pins of other modules, it can produce movement in sound. We have equipped its output with an indicator light so that we can more easily see what's going on with the signal.

4) VCF: The Voltage Controlled Filter (VCF) is a low pass filter. It is the most commonly used filter in the analog world, as it allows us to make a sound softer by cutting off higher frequencies. PYO has a few filters implemented and we chose their recreation of the Moog low pass filter. This one also has a resonance parameter, allowing us to control the "nasality" of the sound. The filter has an input pin that takes in a signal and an output pin that outputs the filtered signal. We also have two potentiometers for the cutoff frequency and resonance, and an input pin for controlling the cutoff.

5) Mixer: The mixer allows us to mix together multiple sources into a single sound. It has four input pins and an amplitude potentiometer for each one. It has two outputs that both output the same joint signal. The reason we have two instead of one is that we can create a feedback loop. Connecting an output pin to one of the input pins produces a sound similar to when feedback happens with microphones. It allows us to create very interesting and rich sounds. The effect is most prominent with square and saw waveforms and when we actively change the VCO frequency during feedback.

6) Sequencer: The sequencer is a module that allows us to create simple melodies. Since PYO doesn't have a built in sequencer that would allow us to use it in the modular sense, we had to create a custom one. Our sequencer has an internal melody consisting of 8 equal notes. The note frequencies can be changed with potentiometers and there is an extra one for controlling the melody speed in beats per minute (BPM). The sequencer has a single output pin that outputs the frequency that is currently being played. We usually connect the sequencer to a VCO's frequency pin. When a connection is made, we manually update the frequency of the VCO so that the melody plays.

7) Defining a new module: Adding a new module is rather simple and allows for user extensions of our solution. We have to create a child class of the base module class, define the inputs, outputs and potentiometers. We can take one of the existing modules as an example. If our new module has an implementation in PYO, our job is easy. We just have to define what parameter each pin/potentiometer is linked to. Custom modules may require some extra work, but they are possible to implement, like our sequencer.

B. User interface

The user interface of our application is rather simple. Upon starting, it consists of a blank canvas with a master out module. On the top we can see a menu that has the list of all modules. When clicking on one of the buttons, the corresponding module gets added to the workspace. We can move it around by clicking on it and moving the mouse. Connections between modules can be made by clicking on one pin and then on another. Potentiometers can be changed by clicking and moving the mouse up and down. When doing so, a tooltip will appear, showing the parameter value. When hovering over any interactable object in the workspace (pin, potentiometer, button) a tooltip will also appear with a simple text, saying what the object is used for. These tooltips are meant to guide an inexperienced user through the application without steering them away from creativity and exploration too much.

The program is currently not packed in an application format, so we run it through the command line. Refer to the file `README.md` provided with the project to see details about installation and running.

IV. Results

In this section we will first present an example of a sound that can be created with our application and then we will proceed to show how modules can be evaluated.

A. Creating a sound

In Figure 2 we can see how a simple patch looks like in our application. We have deliberately used all of the available modules to show the reader what they look like. In these patch we have two signal chains joined together in a mixer. The first chain uses FM synthesis with two VCO that pass through a VCF. We control the frequency of the VCF with a slow LFO to create a droning sound. The second chain plays a simple melody with a sine wave oscillator. Together we get a soundscape that could be used in a sci-fi movie for example.

B. Evaluation

The nature of our project makes it hard to evaluate the implementation directly, such as with tests, but we still managed to perform evaluation empirically. The first step was just to listen to the sound that the application is producing and see if it makes sense. We personally have some past experience in sound synthesis so this was an easy task for us. On top of this, we also devised a second way. We mentioned before that the master out module contains a frequency spectrum visualization. We can use this to determine if modules are acting correctly. Three examples of this can be observed in Figure 3. In the first picture we are generating a sine wave and the frequency spectrum shows a single peak at the VCO frequency. In the second picture we have a saw wave and here we have multiple peaks with falling amplitudes, which makes

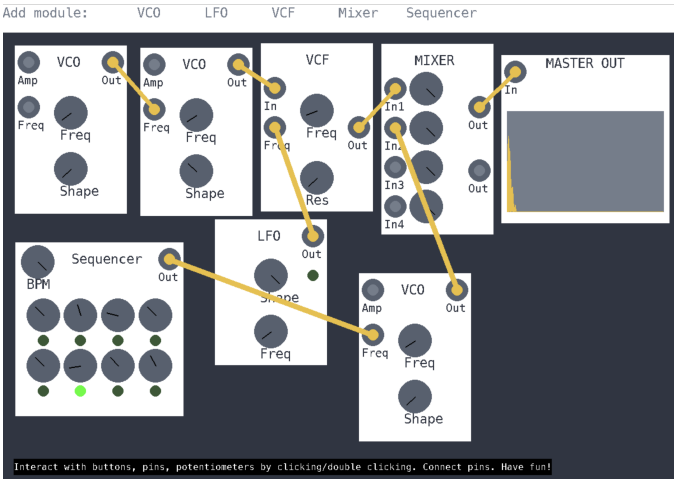

Fig. 2. An example of a patch using all modules.

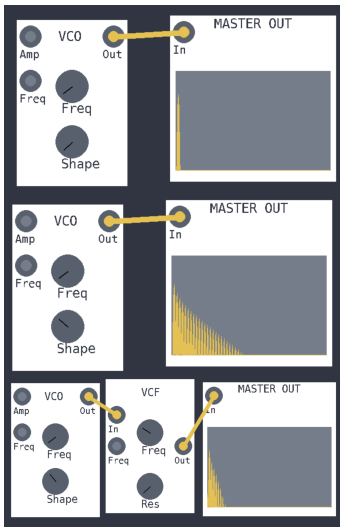

Fig. 3. Frequency spectrums of a sine wave (top), a saw wave (middle) and a filtered saw wave (bottom).

sense. In the third picture we take the same saw wave and filter the top frequencies with the VCF. The effect of this is apparent in the frequency spectrum. We can repeat this process for all of the modules and ensure that they are all producing the desired effect.

V. Discussion

In this report we presented our implementation of an all-in-one simulation of a modular synthesizer system. We described all of the implemented modules and mentioned how a new module can be created. We gave an example of how a sound can be created and showed how we can evaluate modules to ensure they work as intended.

We are happy with how our implementation turned out, as we created a simple modular system simulation that is easy to use and can be simply expanded. An improvement would be the addition of some more control options that we didn't add, like a potentiometer for the VCO amplitude. We could also add some more standard modules, like an envelope generator and a noise generator. Otherwise, our application is a good standalone alternative to other modular synth simulators, that is light, easy to use and open source.

References

[1] James, A., *The Secret World Of Modular Synthesizers*, Sound On Sound, 2013. https://www.soundonsound.com/reviews/secret-world-modular-synthesizers
[2] Analog Synthesizer, Wikipedia, 2024. https://en.wikipedia.org/wiki/Analog_synthesizer
[3] Synthesizer, Wikipedia, 2024. https://en.wikipedia.org/wiki/Synthesizer
[4] PYO, Ajax Sound Studio, 2021. http://ajaxsoundstudio.com/software/pyo/
[5] PyGame, 2024. https://www.pygame.org/
[6] Native Instruments, *Reaktor 6*, 2015. https://www.native-instruments.com/en/products/komplete/synths/reaktor-6/
[7] VCV, *Rack 2*. 2021. https://vcvrack.com/