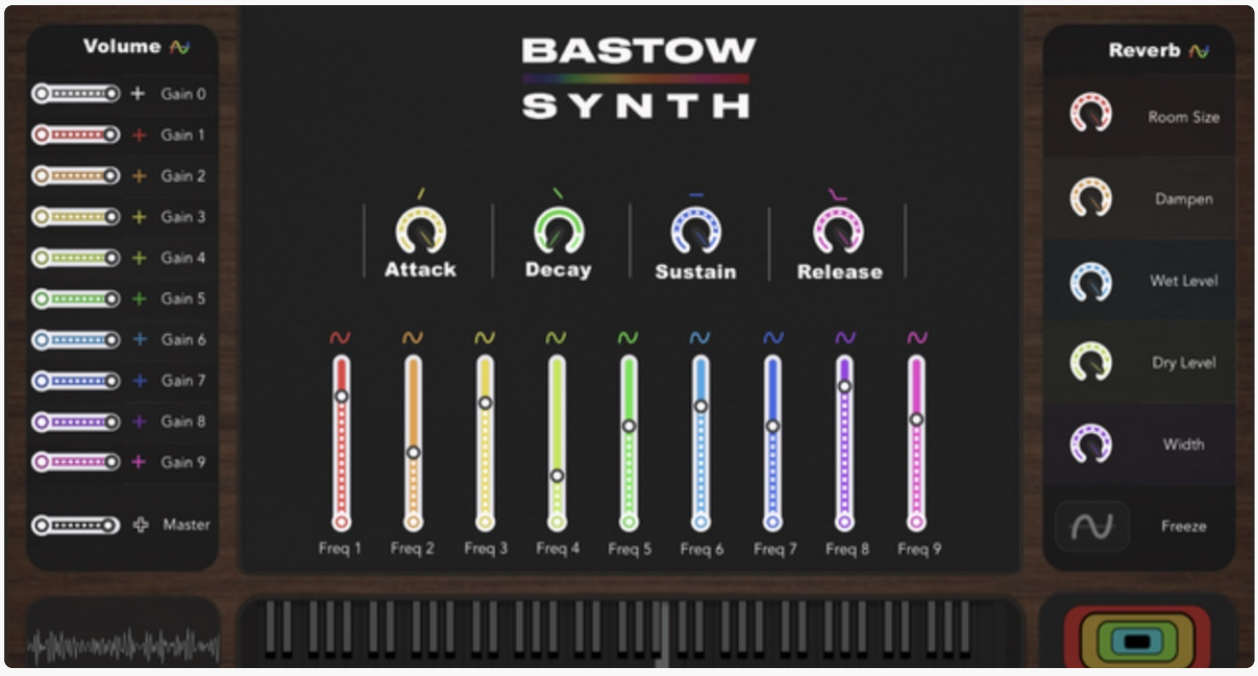
Assessment Report 3

Introduction

For my undergraduate project, I developed an additive synthesiser in C++ using the JUCE framework. At that time, my understanding of audio programming was limited, having issues with clicks and pops.

This practical assessment attempts to expand on prior attempts, going further by-constructing a fm multi-feedback bandlimited additive synthesiser.

Figure 1: Bastow Synth (2022)



Link to video

Assessment 3 Additive fm multi feedback Synth
Queen Mary Masters Student
https://youtu.be/yr4RxDwaD60

Synth Building Blocks: Section 1

Section 1 goes through the important features of my synthesiser

---

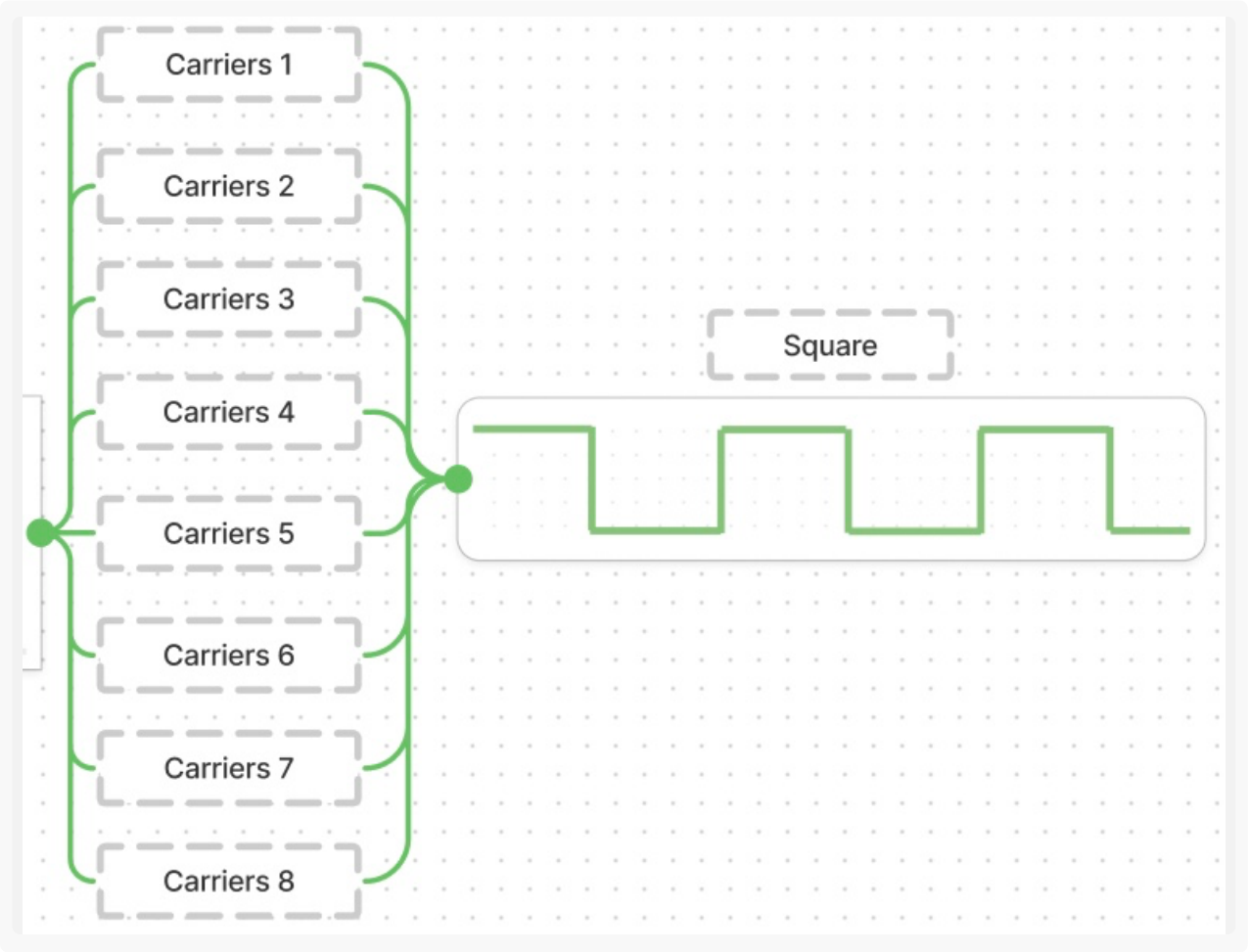
My synthesiser consists of **8 Carriers** and **4 modulators**.

```
wavetable =  generation->prompt_Harmonics(3);
```

The `wave-type` can be changed by altering the number in `prompt_Harmonics()`.

1. Sine Wave
2. Triangle Wave
3. Square Wave
4. Sawtooth Wave

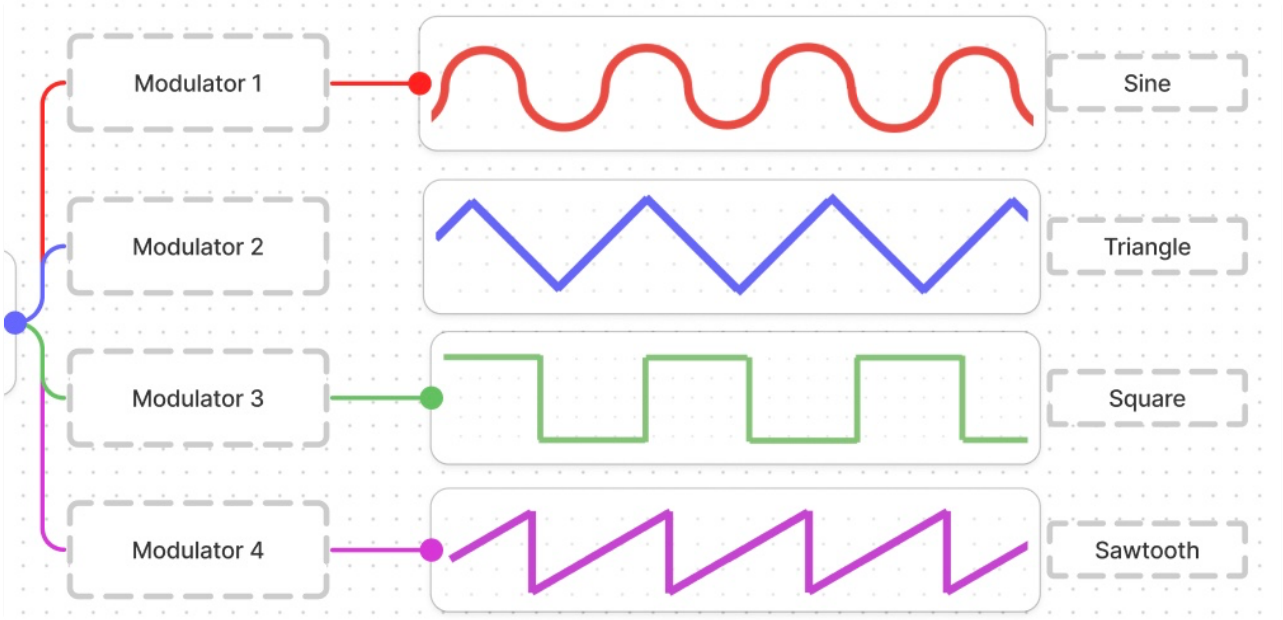
Figure 2: Carriers



- All the modulators are assigned a specific wave form before generating.

```
for(int n = 0; n < kNumModulators; n++)
    {
        unsigned int waveNumber = n < 1 ? n+1 : (n < 2 ? n+2 : (n < 3 ?
n+3 : n+4));
        gModulators[n] = generation->prompt_Modulator(gModulators[n],
waveNumber);
    }
```

Figure 3: Modulators

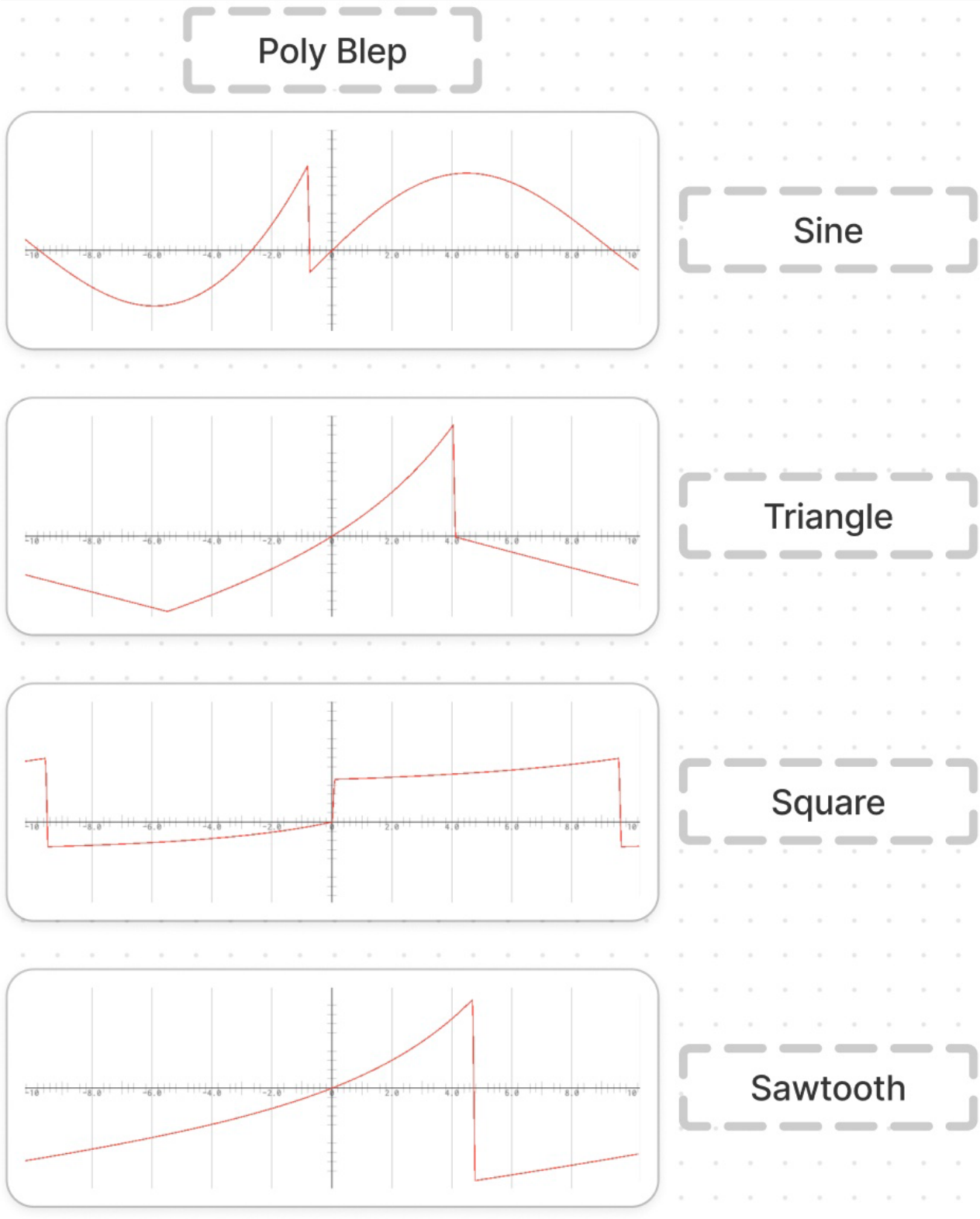


I got my **inspiration** for making the wavetable oscillator code from **week 3** , the additive synthesis lecture and the way that those oscillators were called and initialised.

Polynomial bandlimited step is then applied for **reducing aliasing**, I have used the technique provided by Martin Finke (Finkle, M.(no date)).

Below in figure 3, are all of the waves from my synthesiser that are using the poly blep technique.

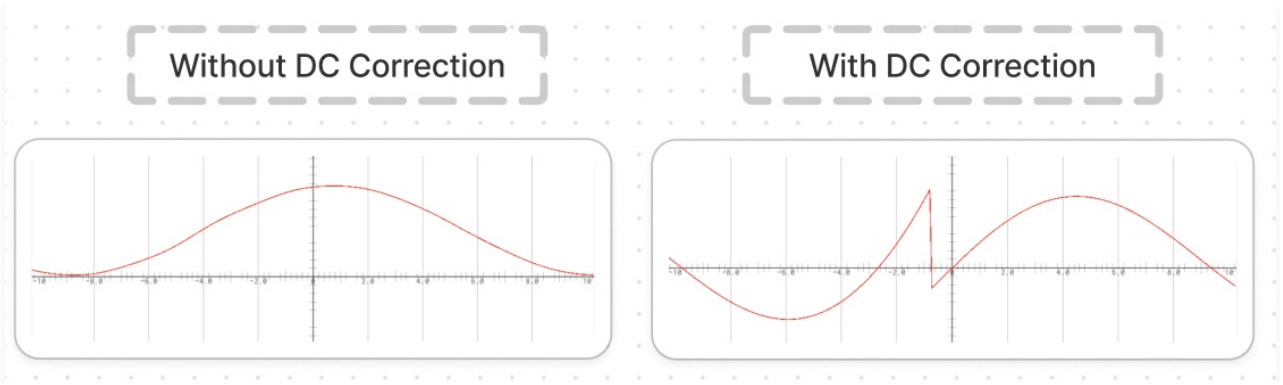
Figure 4: Polynomial Bandlimited Step



Before DC correction my synthesiser would sound a little weirder and the **scope would be upside down.**

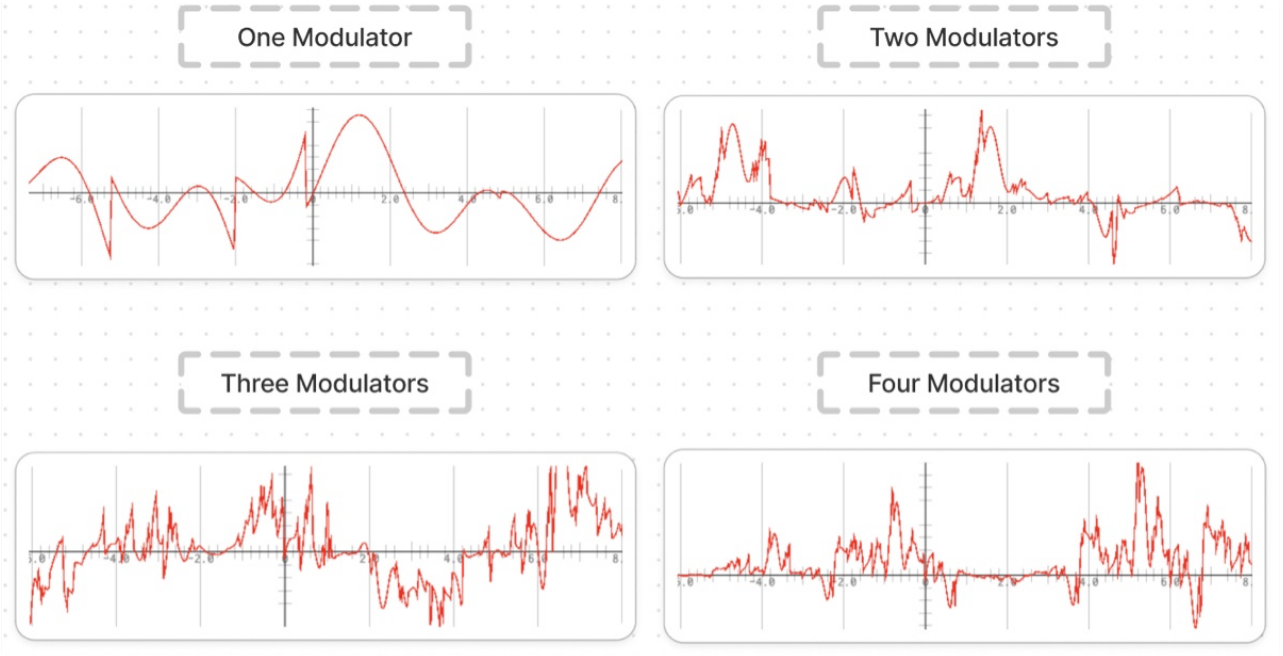
Equation provided by (Allnor, R., 2009)

Figure 5: DC correction



We now have the main building blocks for my synthesiser, in *figure 5* you can see what the wave form looks like with different amounts of modulators.

Figure 6: Modulator Number



The next is the feedback algorithms.

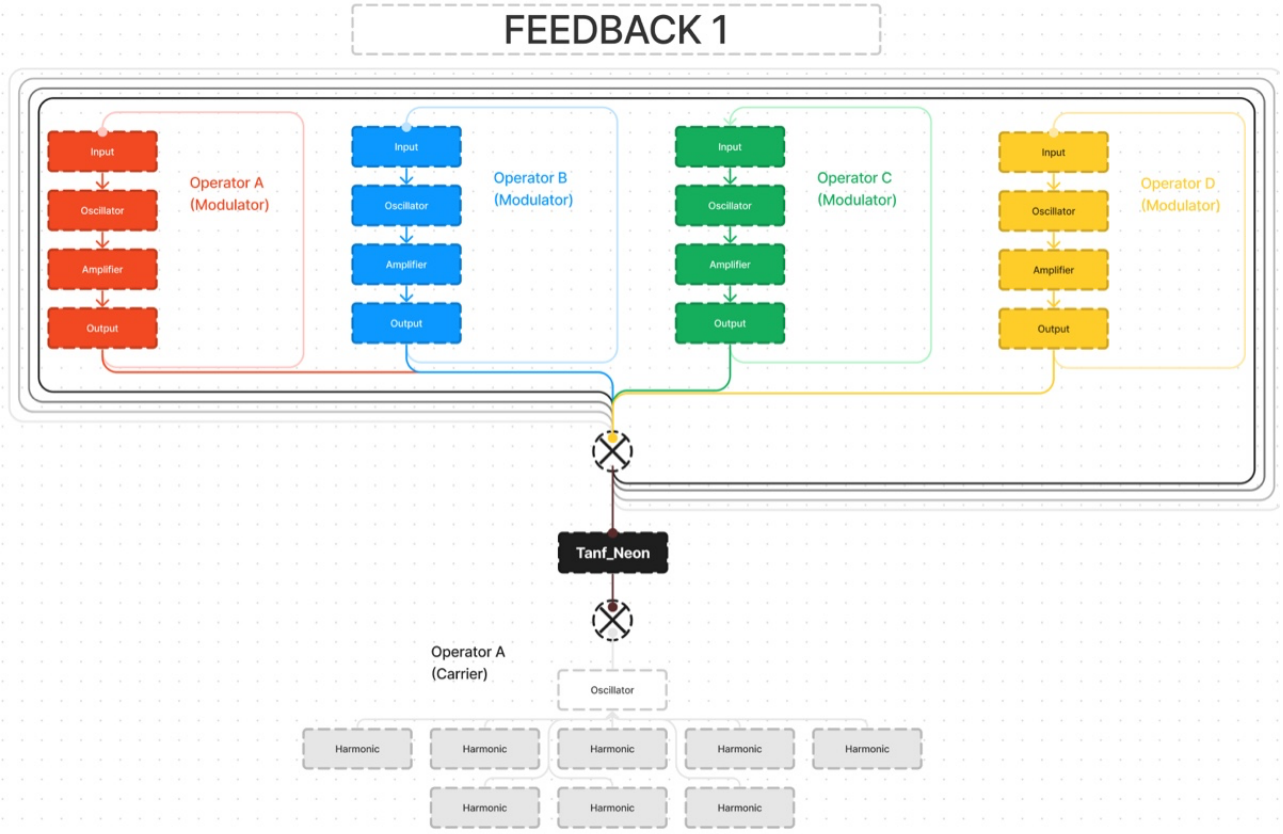
- **All** modulators have **their** *own feedback loops*.
- These modulators are then **added together**, which is then **looped** multiple times before being **added** to the **carrier**.

- Attempts were made to do feedback with the carrier; due to the multiple partials there was only noise, I decided to pass on this.
- I had to experiment a lot here, due to certain algorithms causing instant distortion and crackling.

- Modulators loop individually

1. Modulators are added together and looped
2. Feedback is the same as feedback 1
3. Feedback is the same as feedback 2
4. Feedback is the same as feedback 3 with `tanf_neon()` applied

Figure 7: Feedback 1

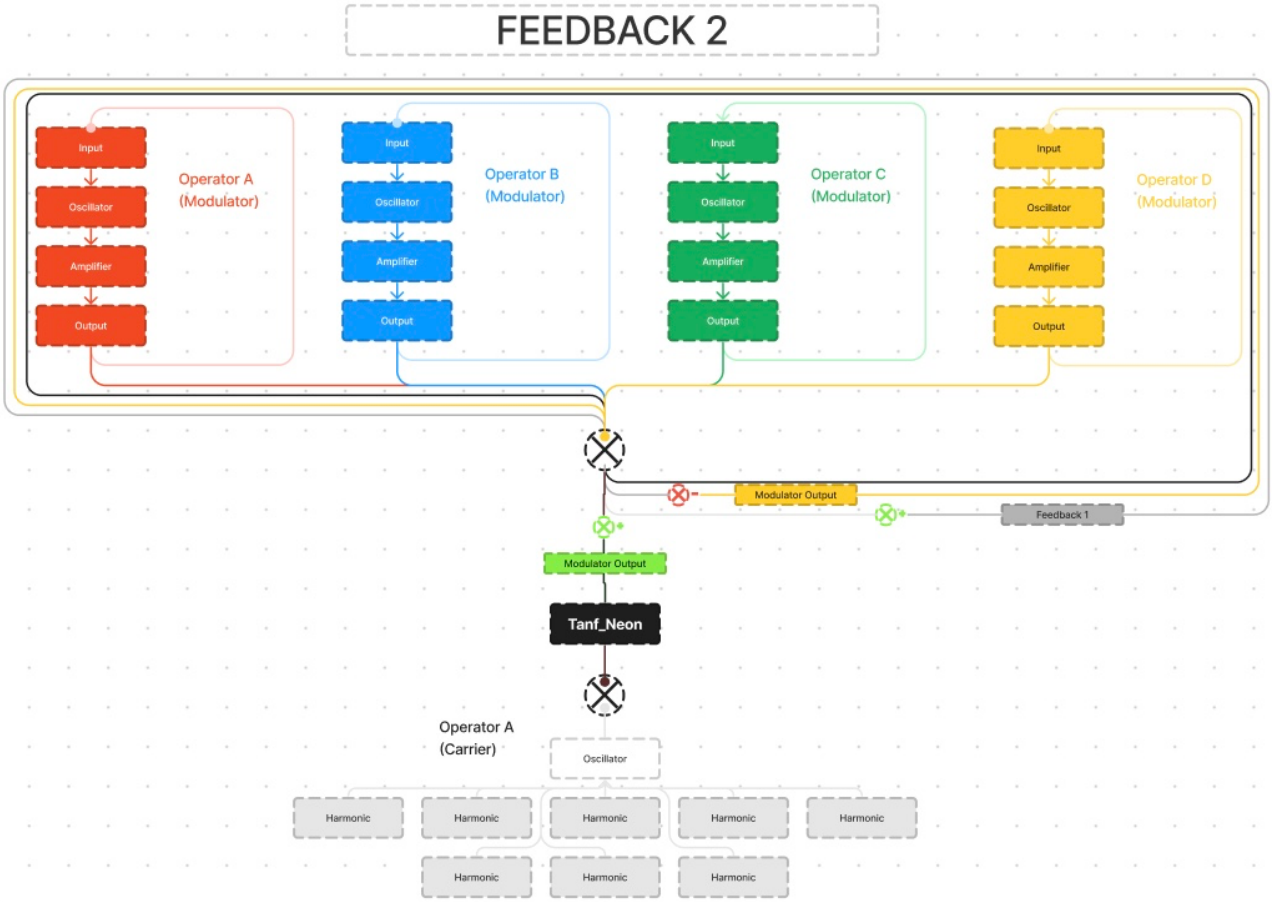


- Modulators loop individually

1. Modulators are added together and looped
2. The added modulator output is subtracted from the second main feedback loop
3. The first feedback loop is added to the third feedback loop

4. The final feedback loop adds the added modulators and applies `tanf_neon()`

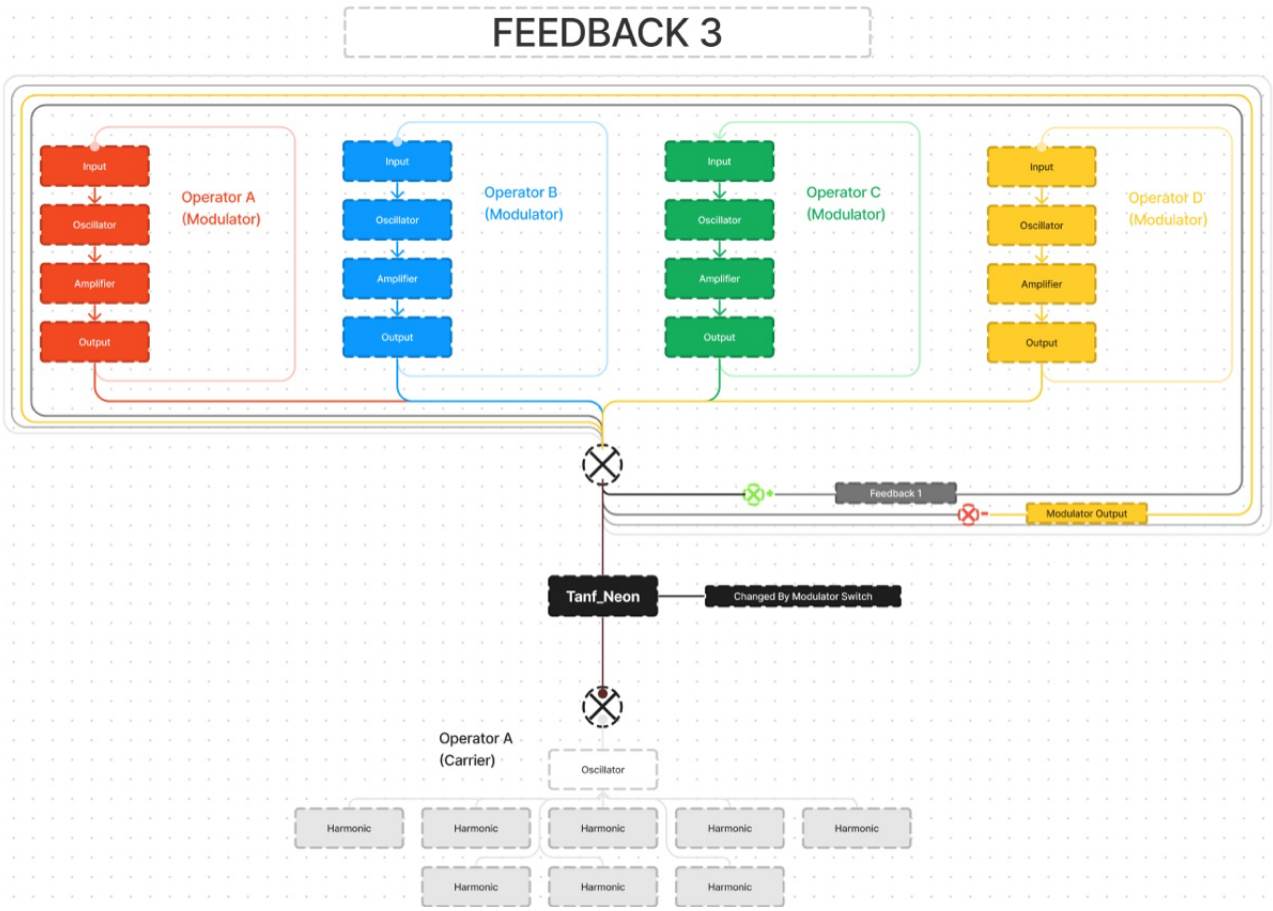
Figure 8: Feedback 2



- Modulators loop individually

1. Modulators are added together and looped
2. The first feedback-loop is added to the second main feedback loop
3. The first added modulators are subtracted to the third feedback loop
4. The final feedback loop applies `tanf_neon()`

Figure 9: Feedback 3



- Modulators loop individually

1. Modulators are added together and looped
2. The third feedback-loop is added to the second main feedback loop
3. The first added modulators are subtracted to the third feedback loop
4. The final feedback loop applies `tanf_neon()`

Figure 10 : Feedback 4

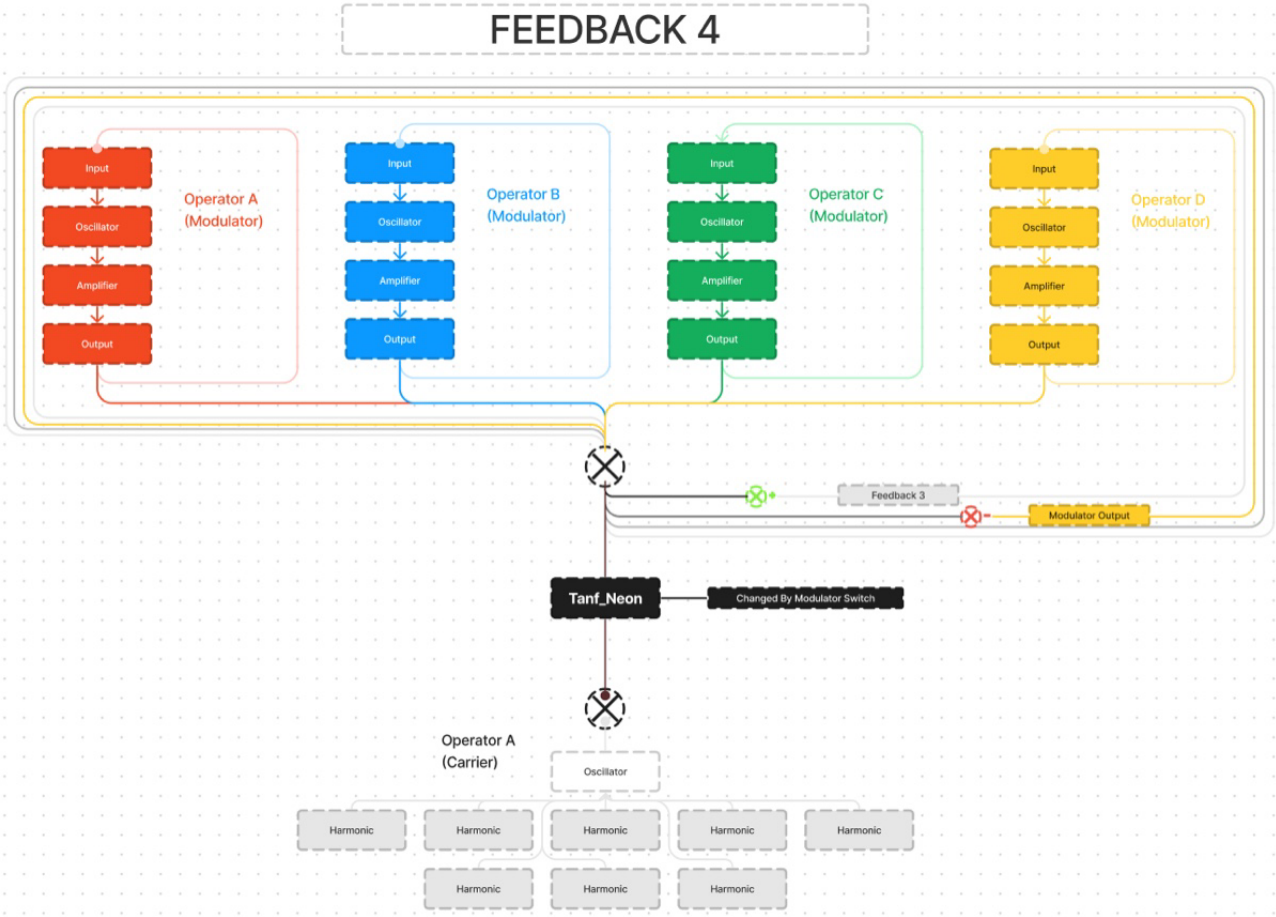
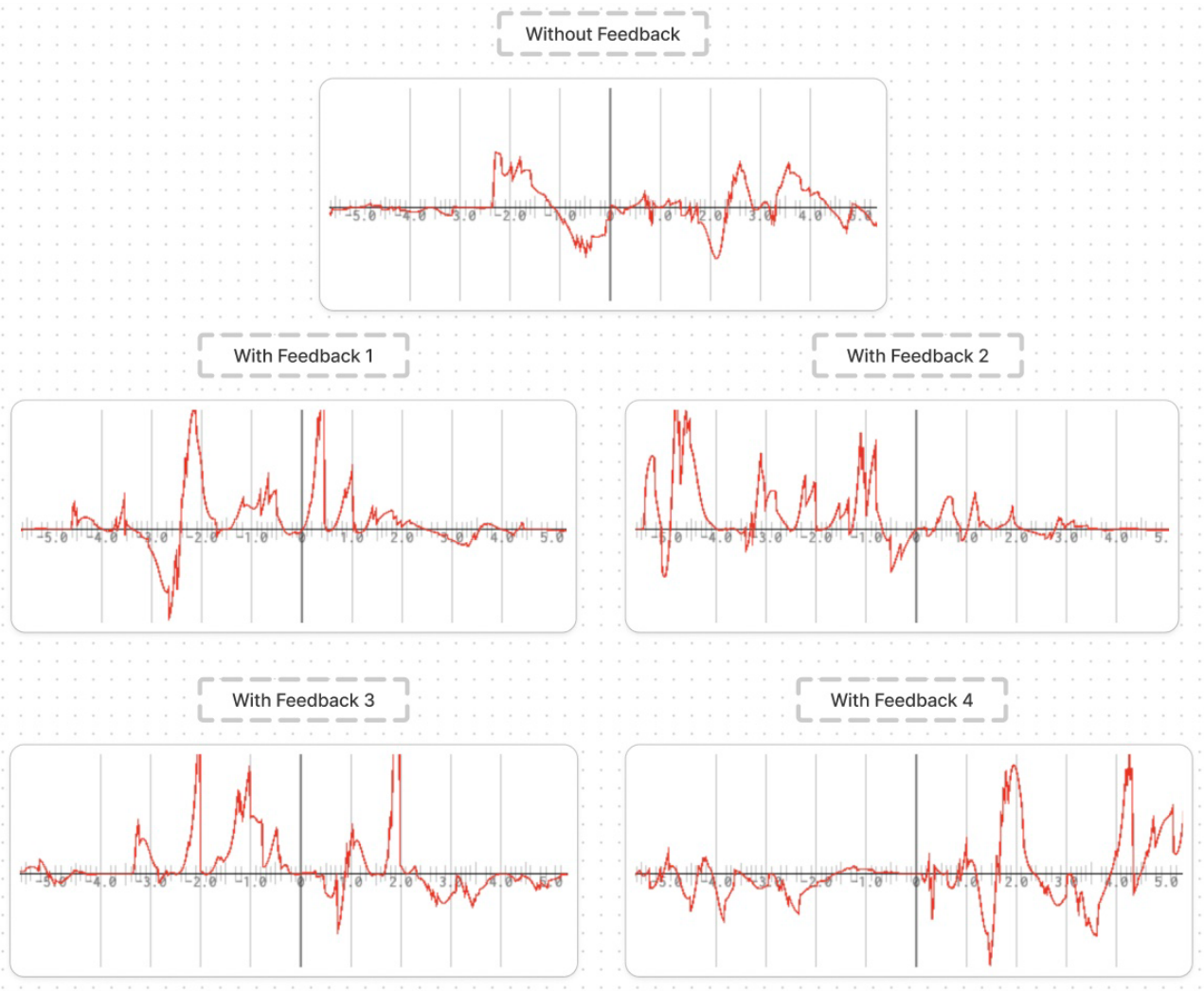


Figure 11 : Feedback Visual



From the very start I wanted to have some breadboard interaction.

Figure 12 : breadboard design.

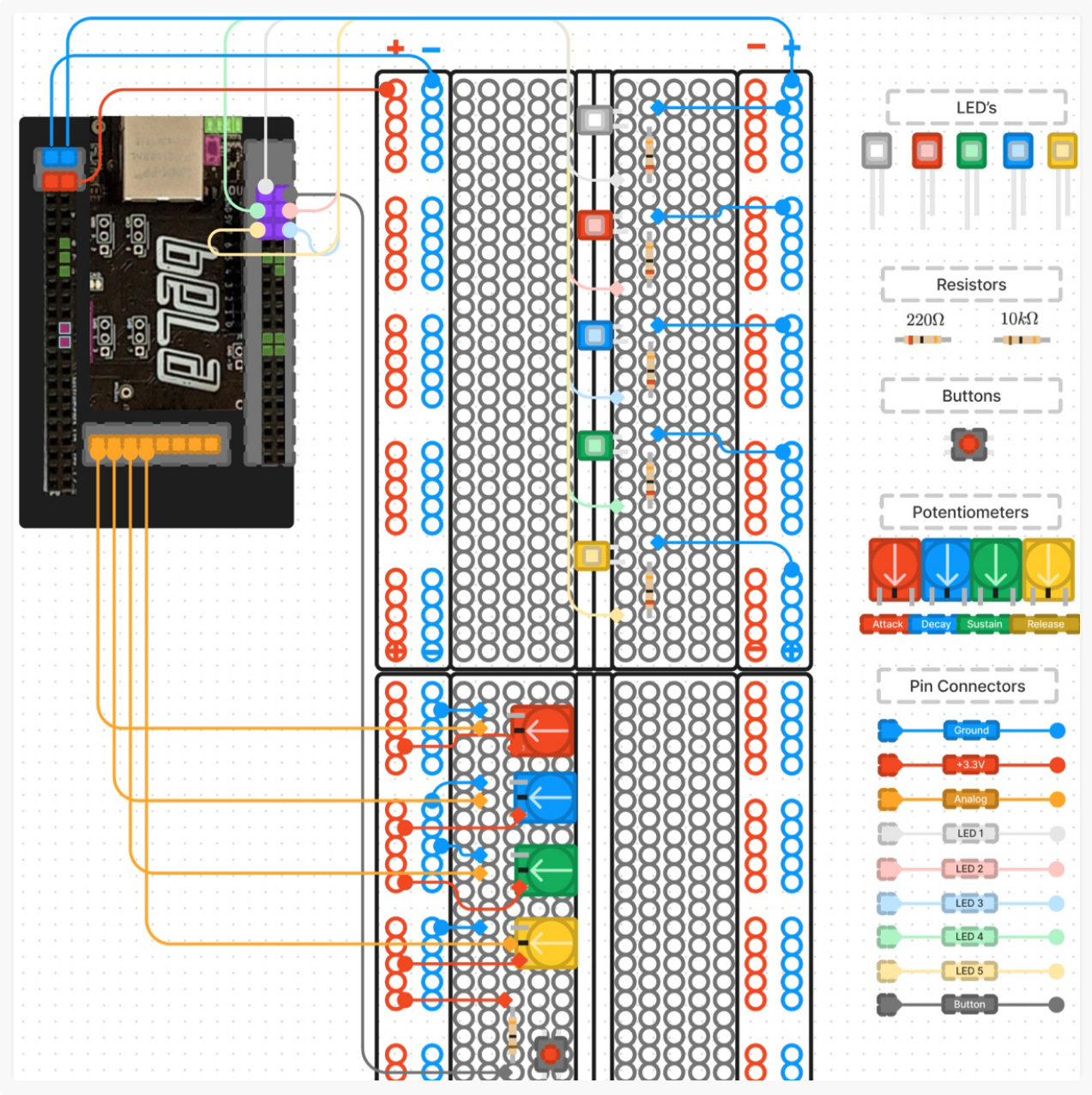


Figure 13 : 4th modulator on && note off

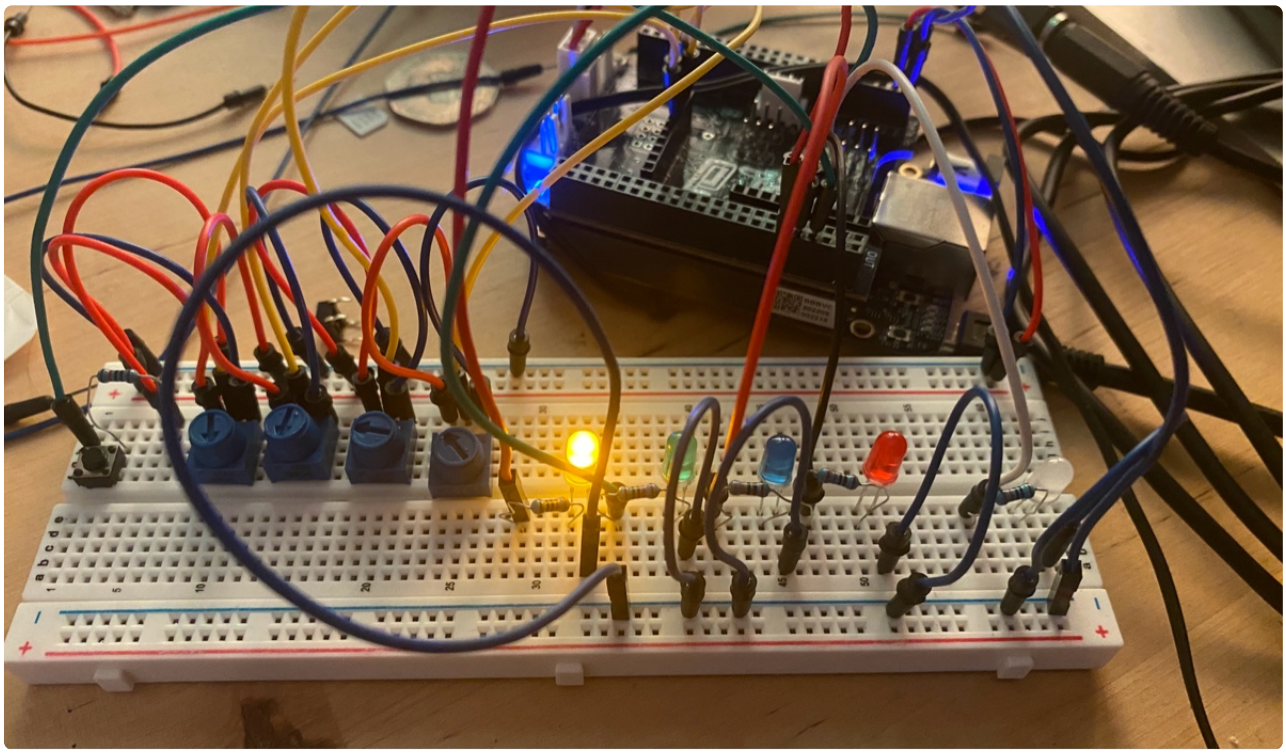


Figure 14 : 4th modulator on && note on

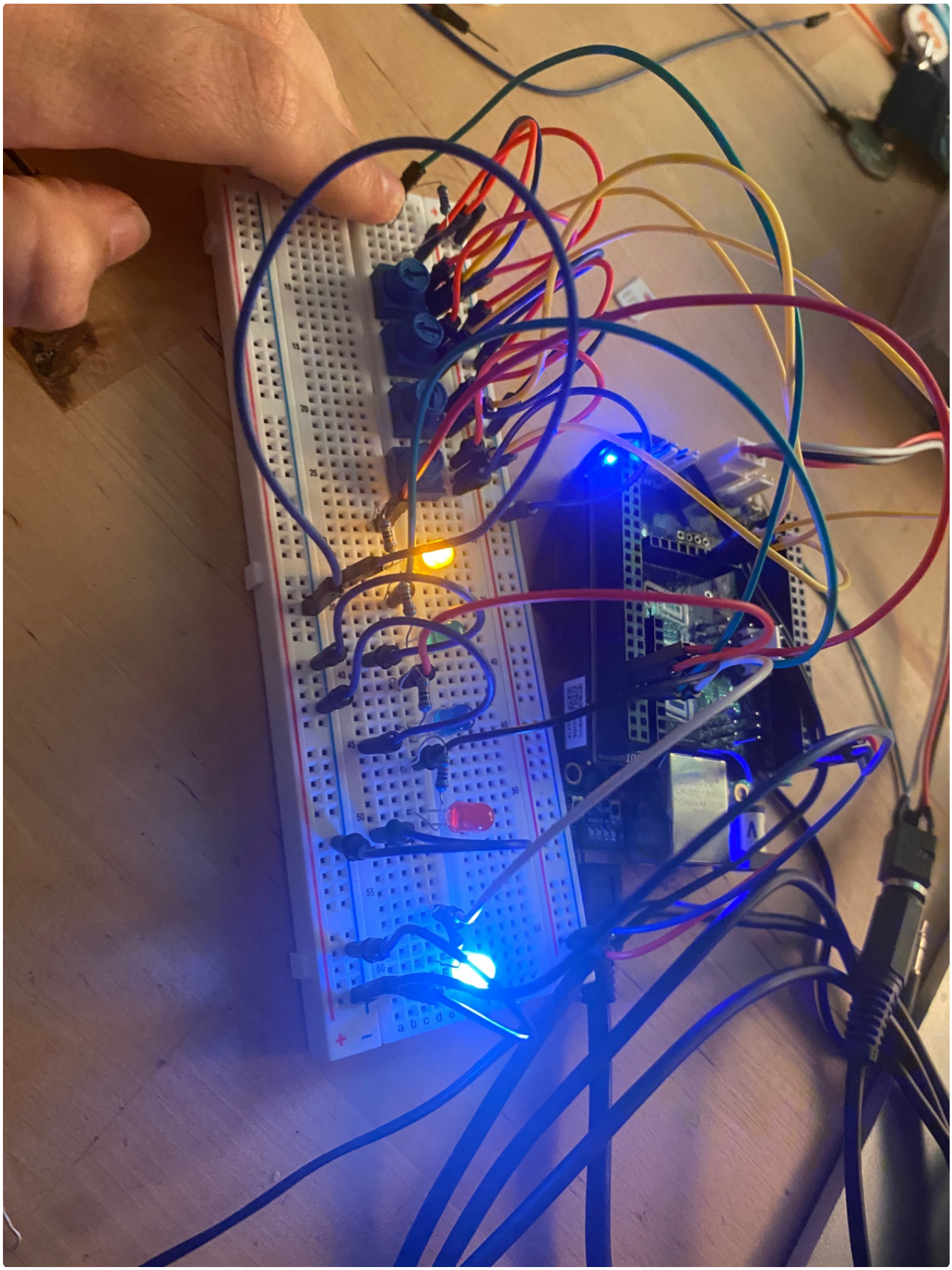
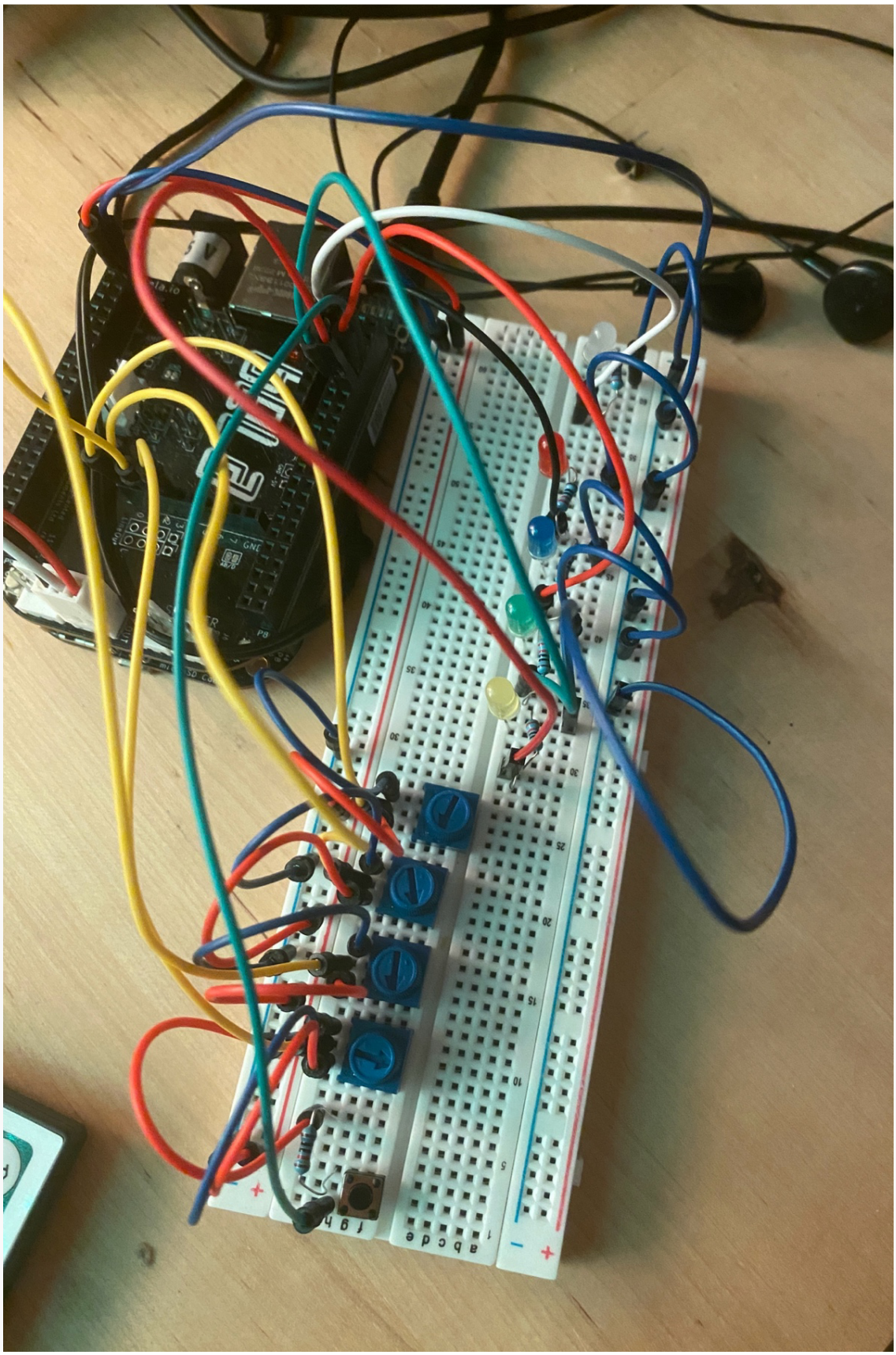


Figure 15: General look at the breadboard



When the **button is pressed the audio starts** and the *white* l*ed turns on;* when the **modulator switch changes modulator**, the led changes from red to yellow. The potentiometers change the filter ADSR components.

Evaluation: Section 2

Section 2 looks at the evaluation of my synthesiser, what more I would have liked to achieve and the draw backs of implementing what I have implemented.

Without any audio my synthesiser **starts** at around 36.8% in CPU, although this doesn't seem too bad, it limits the amount of partials I am able to add. (Explained further down)

Figure 16 : CPU without audio

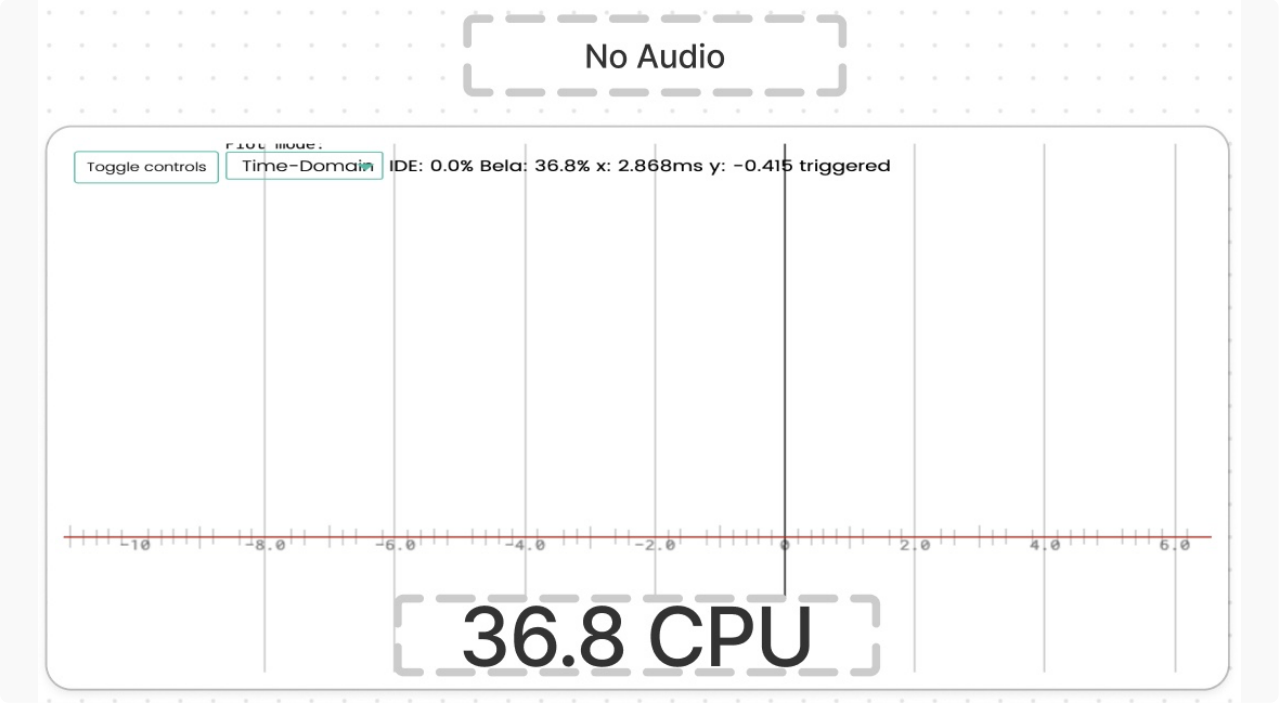


Figure 17 : Testing modulators affect on CPU (one Harmonic)

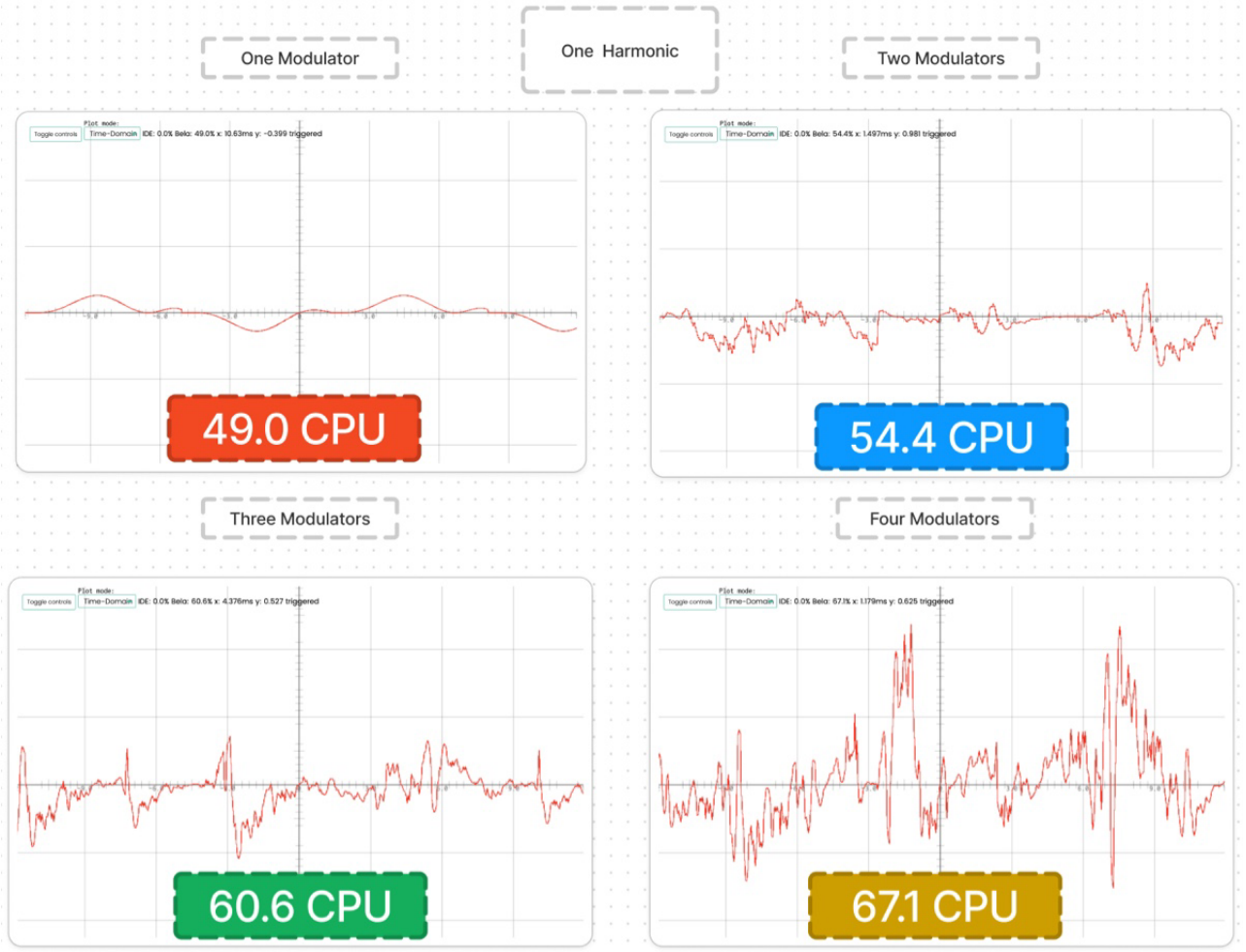


Figure 18 : Testing modulators affect on CPU (Eight Harmonics)

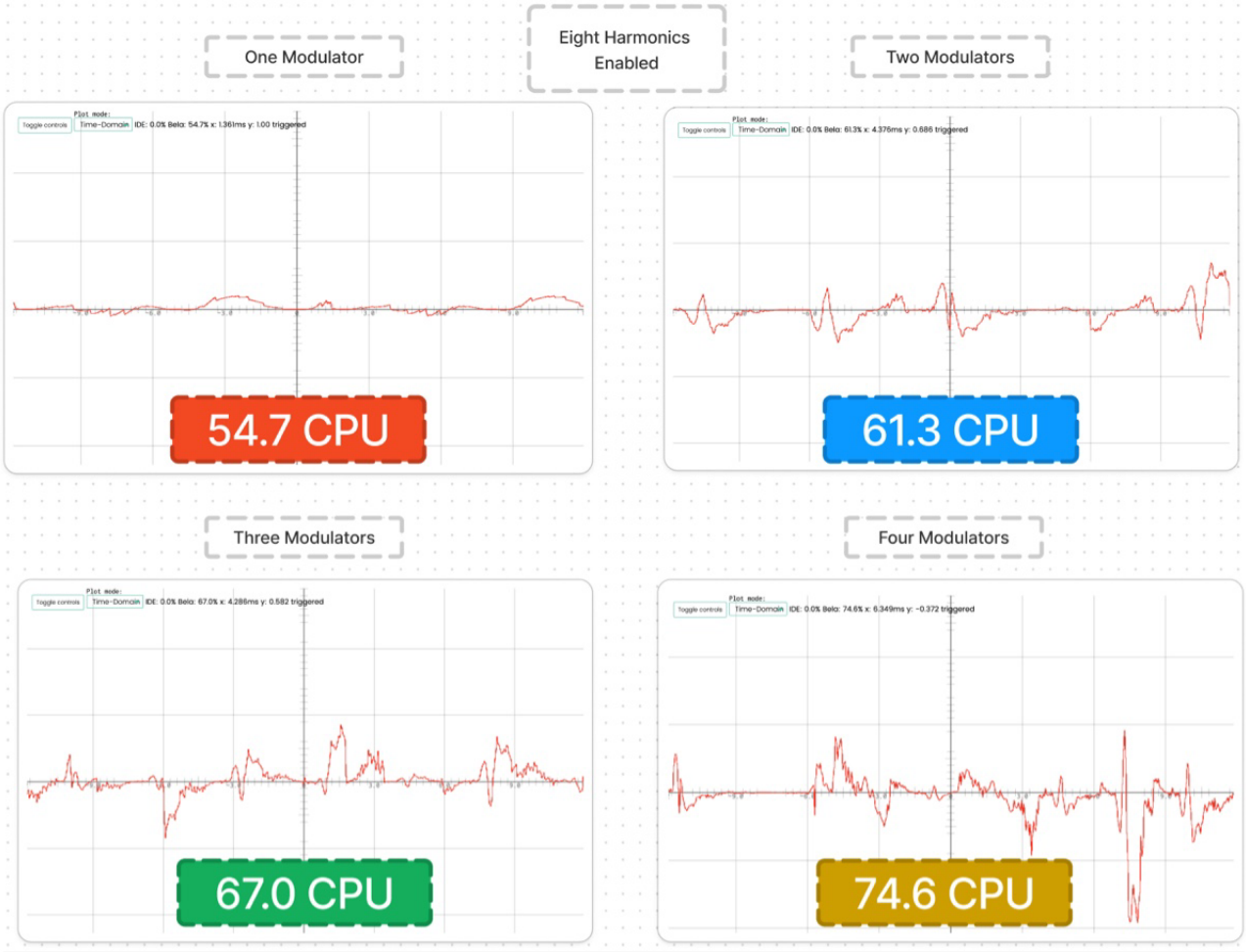


Figure 19 : Testing Feedback 1 affect on CPU (One Harmonics)

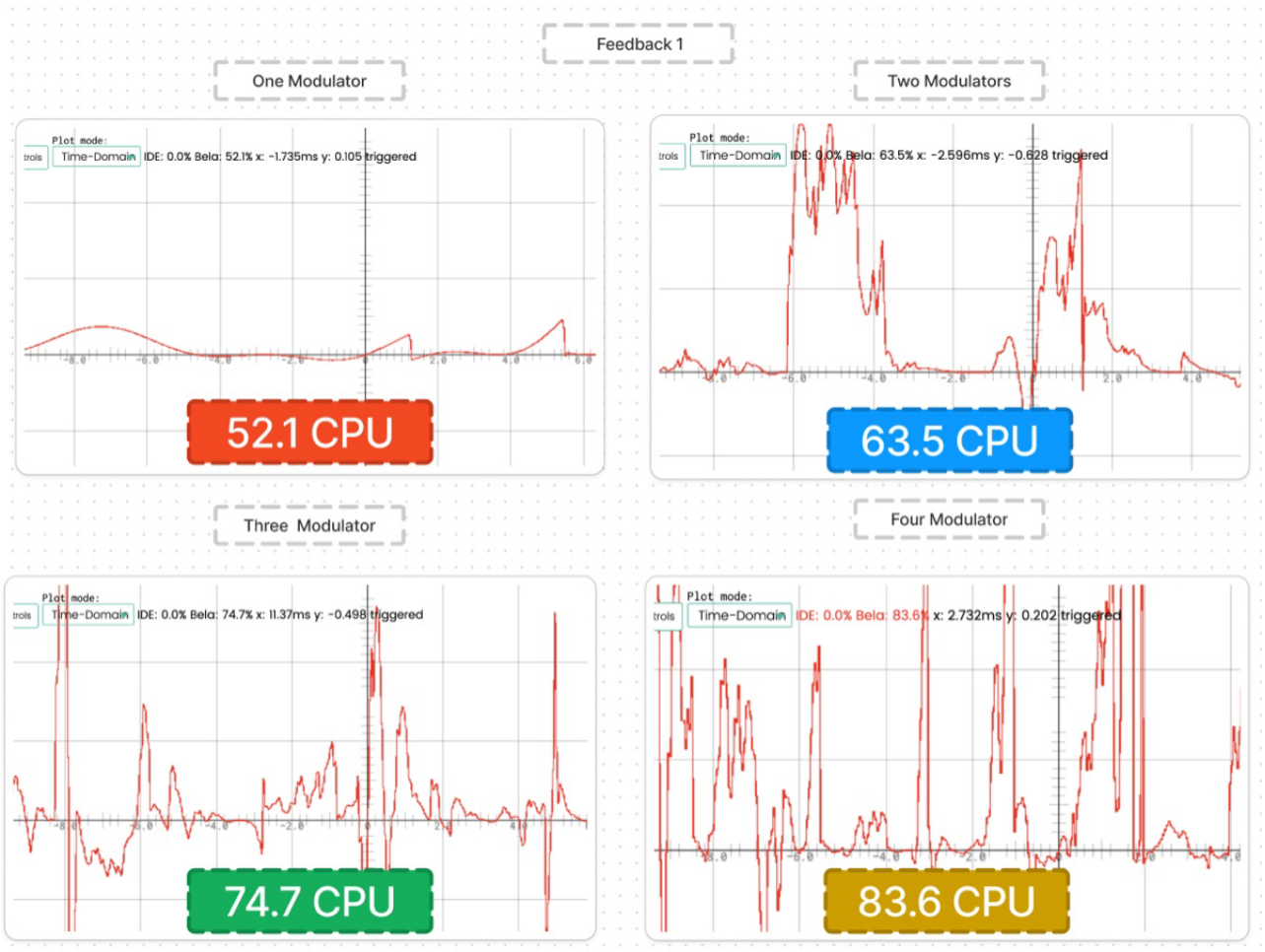


Figure 20 : Testing Feedback 2 affect on CPU (One Harmonics)

Figure 21 : Testing Feedback 3 affect on CPU (One Harmonics)

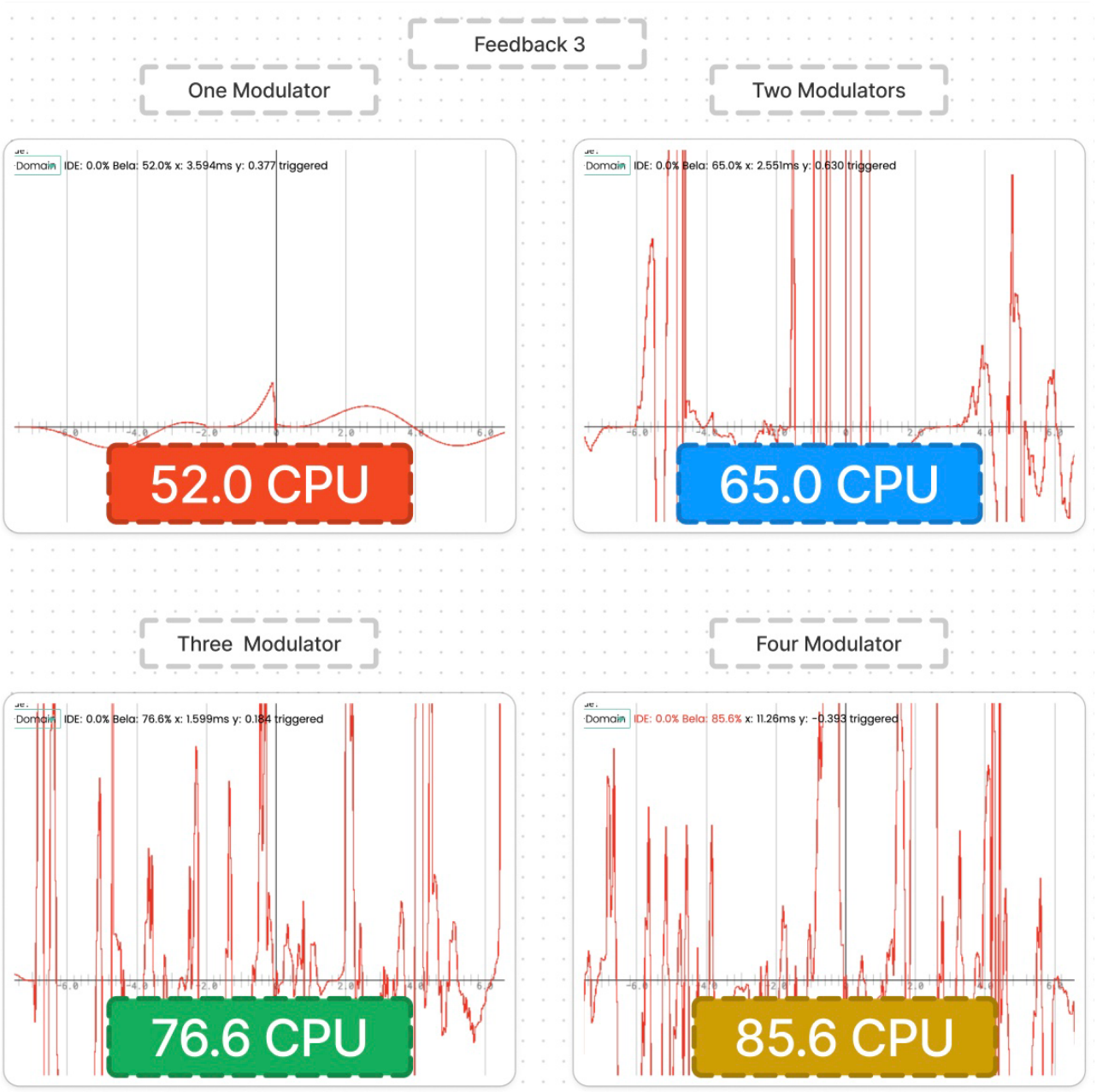
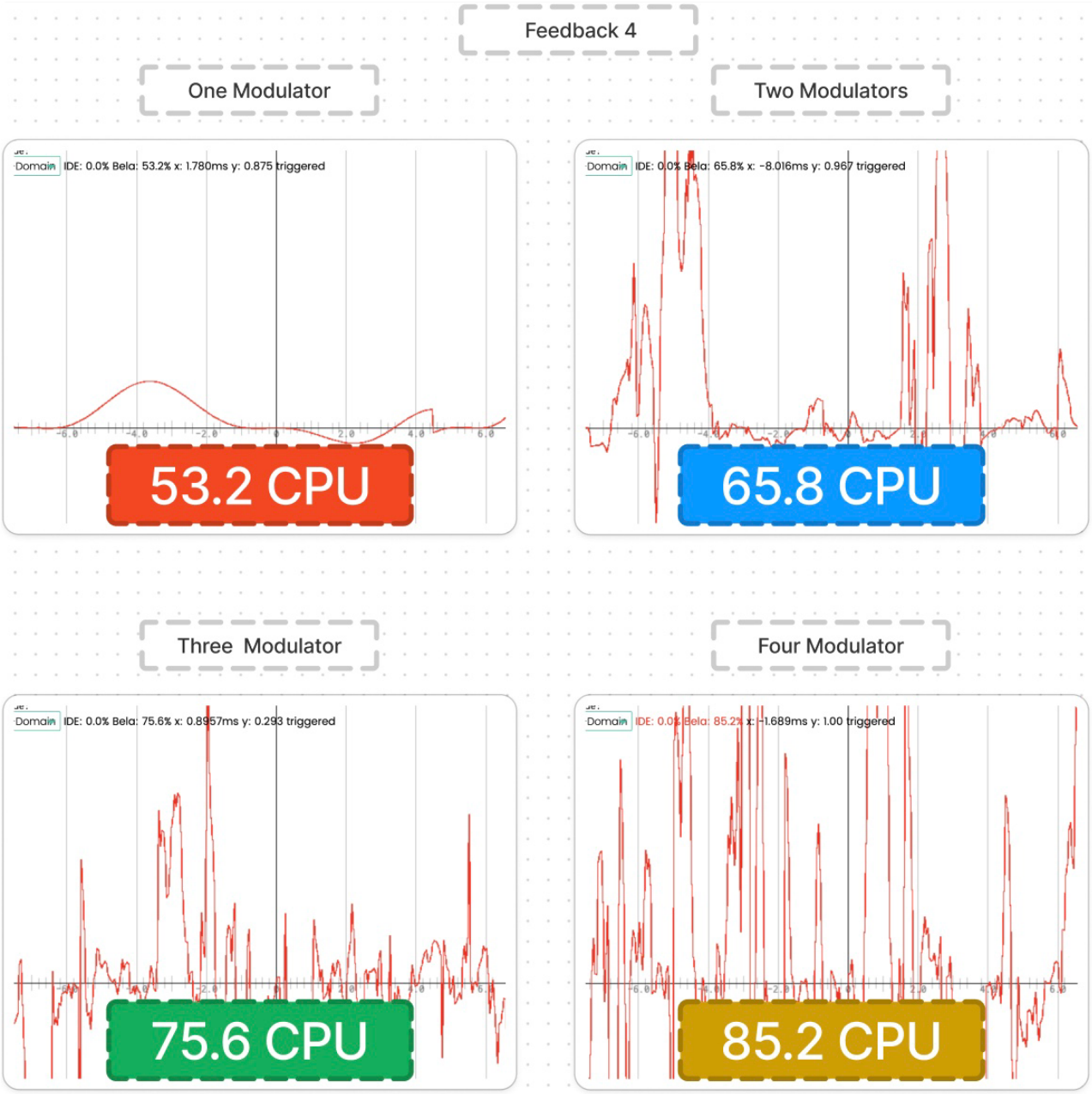


Figure 22 : Testing Feedback 4 affect on CPU (One Harmonics)



With all of the figures displayed, I decided against adding more partials, as you can see the **fourth modulator** when **applied** with **feedback** already **goes above** the **CPU safety** of 80%. This *only occurred* when the oscillators were *held* for more than $3-4$ *seconds*. On algorithms 3 and 4 I had to hold longer for the cpu to go above 80%. All of these feedback tests **only used one carrier oscillator** opposed to eight and if we compare *figure 17 to figure 18* we see an average increase of 6.95%. This would mean that if we used eight partials we could see the cpu go to 91.97% and even more if I had added more harmonics.

In general, I don't think that the current state is too bad, as you have to hold for a couple of seconds before it starts to underrun or have memory problems.

If I had more time I would have liked to **modify the ADSR section** and try to *implement multi segment ADSR*, I would have wanted to *add a button to change the main wave type* of the carriers and oscillators and find out **which class is taking up the cpu** and refactor the code.

Overall I am happy with my implementation I got most of what I wanted to do complete; from the breadboard interactions to the multiple modulators.

Conclusion

I have managed what I have wanted to achieve from this project and have surpassed my previous additive synthesiser attempt on the audio side of the project. For future improvement I would work on CPU improvements and adding more buttons and led interactions for the synthesiser.

Referencing

Allnor, R. (2009) *DC offset (finding it, not removing it!)*. *Available at: https://www.dsprelated.com/showthread/comp.dsp/108402-1.php* (Accessed: 11 May 2023).

Finkle, M. (no date) *Making Audio Plugins Part 18: PolyBLEP Oscillator.* Available at: https://www.martin-finke.de/articles/audio-plugins-018-polyblep-oscillator/ (Accessed: 11 May 2023).