

An Analysis and the Process of Creating a Synthesizer in Native Instruments' Reaktor

Vanja Pantelic
MA Commercial Music Producer
CPO0122
Tileyard Education,
July 2022.

Table of Contents :

Introduction	3
The Process	4
Creating an Oscillator	
Introducing Math	5
Adding ADSR Envelope	6
Adding More Features to The Synthesizer	7
Oscillator Balance	
Filter	
Making the Synth More Interesting	8
Introducing LFO modulation	
Vibrato	
Tremolo	
Delay	9
Conclusion	10
Appendix	11
Appendix	12
Bibliography :	13

Introduction

This essay will represent a brief walkthrough of the process it took me to make the fully functional virtual instrument and the things I learned during the process. For this project I have decided to build a synthesizer from scratch in Native Instruments' Reaktor synthesizer. This decision came about as I have realized that I have little to no experience in this field, and that doing something like this for the first time would be beneficial for me in many ways. I have already had the chance to tweak the existing presets in virtual synthesizers such as Serum, Korg and Arturia Bundles, but I have never felt comfortable making something from scratch. As someone who has mainly been focused on the music making process, it is fair to say that I am totally new to the Sound Design game. I could also say that I am experienced in layering sounds and making them work as a whole and in context, but the approach of making something from the ground up is something I have never been proficient at, so it is safe to say that this was quite a challenging and time consuming project.

On the other hand, I have never found a way to understand the coding and programming in their full form, thus Pure Data, Max DSP and HISE Audio did not seem as a good option for this short period of time that I have had to complete this project. Even though I am always up for learning something new, this seemed way too far-fetched to me. The idea behind this project was not to create a game changing instrument, but instead to make a simple, hands on working synthesizer that could be understood by anyone.

Even though I have had a chance to use Reaktor as a host to Monark and Razor and other ensembles Native Instruments has to offer, I have never had a chance to explore its "Build" function, so this seemed as the perfect opportunity to try it out.

As described on the Native Instruments website, Reaktor is a creative toolkit that specializes in sound exploration and it presents a creative sandbox with limitless options for exploring and diving into the whole new universe of sounds, by simply putting things together. This way something that doesn't already exist can be made, and this synthesizer is a perfect tool for bringing ideas to life. (Native Instruments, n.d.)

This powerful instrument also has the option called "Blocks", where by putting together premade units that they call "blocks", users could build something potentially very useful and practical. This is not what I was going for, or what this project requested, but it was useful and good to learn about this option in breaks from creating the actual synth.

The Process

Creating an Oscillator

After I have watched the video material that was provided to us on this topic and done some research on my own in the form of articles and videos, I pretty much understood the essence of how Reaktor works.

I have started with a very logical move which is programming a sound source by building a first oscillator in order to have something coming out as a signal. Most of the components that ended up in the final version of the synthesizer are based on macros that I have separately built and organized. To further simplify this, an oscillator included in Reaktor is a basic voltage circuit that produces alternating waveforms. Some of the most popular ones are sine wave, sawtooth, pulse wave (also called square wave), triangle, etc... (Electrical4U, 2020)

By creating a new ensemble in Reaktor, I was ready to start giving the instructions to the matrix and create a simple signal path with a first oscillator. By default, Reaktor provides an input section that, from the get go, I did not need, so I deleted it. The reason for this is that I had the idea of making a simple monophonic synth, which did not need to be triggered by a separate macro that alters voices, and I could just start building the synth straight to the panel. I decided that it would be best to create a separate oscillator macro and then load the Built-In Module oscillators inside of it. This way I could have more control over the whole oscillator section and I could add more interesting stuff to it. I will be addressing this in the later phase of the process. Further on, inserting a simple sine wave oscillator I recognized that there are two parameters : P that stands for Pitch, and A that stands for Amplitude. That is when I created two inputs that matched the oscillators and labeled them as Pitch and Amplitude and connected it to the oscillator's inputs. On the other side of the oscillator macro I have created the output, so the oscillator would be able to communicate with other modules and eventually produce the sound. It's important to say that the oscillator is controlled by the note pitch of the incoming MIDI notes. This is why I added the "Note Pitch" and "Gate" options straight from the MIDI section in Reaktor's main panel. The MIDI section was located under the Built-In Module function in Reaktor. In order to make that control work I had to connect the "Note Pitch" function to the P input of the Sine Wave oscillator I had previously made, and "Gate" function to the A input of the oscillator. This action was fairly simple, as Reaktor is based on the system of wires that are connected from module to module.(ADSR, n.d) At this point, Reaktor's indicator was showing that everything was connected well and working, which I have later on proved by actually hearing the sine wave play after I triggered it with the MIDI keyboard. At that moment, I had a pretty much very simple and basic sine wave based oscillator that was working. It is fair to say that this was only the beginning of the process, but it gave me a certain amount of satisfaction to proceed developing it. What I wanted to do next in order to visually develop my synth is make a selectable choice of waveforms. That is where I have added a "switch" control to the oscillator section, and put the minimal amount of ports to 5 as I have decided to have 5 waveform options. Following the signal path, this switch was the last put right after the sine wave oscillator, so it goes directly into the output of the oscillator macro. This way, after I have added more oscillators, I would be able to choose what exact waveform I would like to select.

This is where I started to see the visual changes as the actual five switches appeared on the main Reaktor panel. And this is where the appearance factor of the synthesizer started to take place, triggering my creative side and inspiring me to visually design it. In order to be able to choose multiple oscillators, I first had to create them. I have decided to have Sine, Sawtooth, Pulse, Triangle and White Noise oscillator as this was something I thought could be used creatively. This is where I have connected the oscillators to the inputs of the switch module and routed it back to the output of the oscillator macro. I have run a couple of tests of the waveforms, and after I was sure they were each working, I was ready to proceed with building my synth. Now at this moment, I came across something I have never heard of before. Furthermore, I noticed that when I created the pulse wave, besides the usual “P” and “A” initials, there was a “W” initial as well. After doing some research on the topic, I have learned that “W” stands for Width that is characteristic to the pulse wave. Pulse Wave Modulation or PWM is a control that could be useful when there is an idea to make the sound a bit more interesting and it is based on a concept of duty circuit. (SparkFun, n.d)

So when I learned that I could actually control that, I immediately created a knob for that purpose, and tested it accordingly. After I was sure that PWM control worked, I was curious to make the oscillator section a bit more up to date with the modern synthesis, and I wanted to have more functions to it.

Introducing Math

At this point I decided to add another “Switch” control that would actually determine the octave that I want my synth to play. In order to keep the design of the synth simple and hands on, I have decided to have one octave up and one octave down from the root. In terms of the signal flow, I wanted the switch to be activated before it reaches the actual oscillator. That way, I could alter the source that will trigger the oscillator(s). So after placing the “Octave” switch control right after the “Pitch” input, I had to actually feed it with information in order for this function to work. This is where I was first introduced to the math section of Reaktor. While I am usually frustrated when any kind of math is involved, after doing some research, I found out that this function is not as complicated as one would think. I realized that these equations are not at all complex and that I just needed basic understanding of math to comprehend what is going on inside the Reaktor’s panel. In this phase and for this particular action I did not need to use any more operations other than addition and subtraction. As I have added the addition module called “Add”, I created a constant with a value of 12 in order for the synth to add twelve semitones when engaged, resulting in the reproduction of the higher octave sound. On the opposite, I added a subtraction module - “Subtract” with the same value of 12. This way when engaged, it would subtract twelve semitones from the root, playing a lower octave sound. After wiring and connecting everything, I was able to test the switch control that I had previously created, and after making sure that everything was working, I was ready to proceed developing the functionality of this oscillator, which eventually turned out to be the foundation of the synth. (ADSR, n.d.)

At this moment, when thinking of what else could be added there, I remembered that Serum, a VST Instrument by XFER, has a fine tune feature, and I thought I could use that as a reference for my synth. For the fine tune function, I have created a separate knob to control the value. The

idea behind it was to have one semitone/one hundred cents up and down from the default value, which I set to zero. By setting a minimal value to -1 and maximum value to 1 in the range window of the knob, I was able to have the default value in the middle, while I could alter the knob to left and right according to what I wanted to achieve. In order to make this control work, I had to introduce another mathematical operation that needed to add the values that I had previously set in the range window. After connecting that up to the oscillators that were previously already hooked to the output, everything was working fine and I pretty much had my basic oscillator done. At this point, as the oscillator was done, and I wanted to have at least three oscillator synthesizer, I duplicated this oscillator macro twice, and theoretically, I had my three oscillator synthesizer done. (Rajan, 2016) After patching everything up and making sure it was working, I added Reaktor's own master level fader that would control its main output. After I had done this, I was ready to move onto the next phase.

Adding ADSR Envelope

For this action, traditionally, I also had to create a separate macro, and build the envelope inside it. After locating the ADSR module in the LFO section of Reaktor, I needed to set up a couple of things in order to make this feature work. First things first, I needed to enable an initial communication of this macro with other modules. I have created a gate control and output port in order to enable the passing of the signal through the ADSR macro. In order for the ADSR controls to appear on the actual panel, I needed to create controls for attack, decay, sustain and release inside the macro itself. I have decided to leave the controls in the form of knobs, as I was looking for more of an analog looking synthesizer with a minimal design.

Speaking of which, I have come to realize that there is an option for the envelope to be visible, but for the aforementioned reason, I did not want to include that option. This is where the "Gate" MIDI setting from the beginning of the process came to play. I patched that gate to the gate input that I have created in the ADSR macro, and the ADSR section was ready to go. (Electric Trumpet, n.d.)

Adding More Features to The Synthesizer

Oscillator Balance

At this point, I was satisfied with what I had so far, having in mind that I was doing something like this for the first time, I started to feel inspired and brainstorm about the next feature I could add to this synthesizer. This is when I turned back to the Virus A synthesizer by Access company that I had used as a reference for this project and started checking out the features that it had. Once I had realized Virus A has a “Mixer” section that combines the volumes of all three oscillators, I decided to incorporate this into my project as well (Vintage Synth Explorer, n.d). I created a new macro, and loaded three inputs for the number of oscillators that I had, in order for them to communicate with other modules and get the information from the oscillators. As in every macro, I also had to create an output port in order for the signal to be undisturbed. After that I have inserted an Amp/Mixer module with a minimal number of three inputs in order for each oscillators’ signal to be patched with it. For the sake of visibility of this module, I had to create control for the level unit on the Amp/Mixer. This is where three level knobs appeared and after naming them accordingly and patching them to the output section of this macro, they were ready to go. This feature was fairly simple and logical to implement, and it made sense in context and was visually corresponding to my reference.

Filter

After all this, I wanted to add a filter to my synthesizer, in order to make it more creative and inspiring and to make it as functional as possible. I have tried to program a filter on my own, and eventually succeeded, but personally it looked very simple and had limited functions and was not visually appealing to me. I have done this through Ladder Filter where I have just created controls for cut off and resonance. The filter was initially working, but as stated above, I personally was not fond of it. Due to the lack of knowledge on how to create a more complex filter, I decided to load a premade Linear Filter and put it inside of a new filter macro that I have created. I did this because I wanted to address the aforementioned visual aspect and to make the synth a bit more functional. This filter had the basic cut off and resonance controls, but it also had a lot of extra features. This included lowpass, highpass, bandpass and notch filter and also different cutoff slopes such as 6dB, 12dB and 24dB per octave. It also did include the visual representation of what the filter does, so I found that way more appealing and decided to keep this filter. (ADSR, n.d)

Making the Synth More Interesting

Introducing LFO modulation

At this point, I already had a proper basic synthesizer that was working and could be used to create different sounds. But for the sake of learning something new, and making the most out of this project, I wanted to add some more features in order to make this synth a bit more interesting. I was thinking of doing something like a portamento, but after doing some research I realized that as I lacked experience in this field, and that this feature might be too complicated to program, so I gave up on that idea.

Vibrato

Instead, I came across an interesting option of incorporating pitch modulation in the form of vibrato. In my experience with synthesizers I learned that vibrato can add certain character and uniqueness to the sound. I created a new macro, and logically placed it right after the “Note Pitch” module, so the signal could be altered before it reaches the oscillator. After learning that this was done with LFO, I inserted an LFO module inside the existing macro that I have named “Vibrato”. After creating controls for Frequency and Amplitude, I renamed them into “Rate” that defines the speed of pitch variation and “Depth” that represents the amount of pitch variation. (Owens, n.d.) This is where the math function came into play once again, but this time it was even simpler. This time there was no need to create a constant value, as this module only had a purpose of adding frequency modulation that was earlier programmed. After wiring the vibrato module to the pitch inputs of the oscillators, I ran a couple of tests and tweaked the knobs, and when I was sure this feature was working, I moved onto the next one. (Hutchinson, 2020)

Tremolo

On the other hand, even though both tremolo and vibrato are based on modulating the LFO of the sound, tremolo was a bit harder to make, as the math behind it is a bit more complex.

Tremolo in its basis is volume/amplitude modulation, that gives the sound that wavy modern touch, even though it has existed for hundreds of years. It is frequently heard in radio friendly tracks that are influenced by the genres such as Future Bass, Dubstep, Riddim, etc..

When engaged, this function allows the element to have a more rhythmic and creative touch to it, making the track sound more interesting in context. (Owens, n.d.) I duplicated the macro that I had for the vibrato effect, as the foundation is pretty much the same, but I had to add some more operations to the equation. This is where multiplication was first introduced in this project. In order to make this function work, I had to somehow alter the amplitude from the minimum value of -1 to the maximum value of 1. After doing a bit of research, it made sense to me to introduce multiplication and the addition, one after the other.

Bearing in mind that the lowest value is -1, I created a multiplication module and introduced the constant of 0.5 and then included addition with the same value of 0.5. This equaled 0 and represented the lowest point for the tremolo.

On the other hand, the highest value was 1, and going through the same operation, this equaled 1, and represented the highest value of the tremolo. After doing this simple equation, all I had to

do was to create another multiplication module in the Reaktor's panel that would feed the signal to the amplitude of the oscillators and the tremolo was ready to go. (ADSR, n.d.)

Delay

At this point I was pretty satisfied with what I had built, but I wanted to take the synth one step further and have a tiny FX section. I was thinking of reverb at first, but I decided to try and have my take on delay as it could also be used for making a manual chorus or flanger. This module was placed at the end of the signal flow path, right before the signal reaches the output. That was the only option that made sense to me, as this was the final destination of the signal before reaching the output and eventually producing the sound. Creating a very simple and basic delay was really easy and straightforward as Reaktor has pre-programmed delay units. I have created a delay macro and inserted a module called "Single Delay" and patched the input and output ports into it. At this moment, I have created a delay time control and pretty much had the delay effect, but not exactly the one I was going for. This effect delayed the initial signal for the time that was set in milliseconds. This is where I realized that I needed to create dry and wet knobs in order to control when the delay is occurring and to protect the dry signal from being altered. I thought that it would be best to do this through the Amp/Mixer module with two level controls and two inputs. After patching the initial input of the delay macro and Single Delay's output to it, I created these two controls that I named "Dry" and "Wet". At this point, the delay was working solidly and I was happy to have accomplished what I was going for. But as curious as I was, I wanted to see I was able to make the delay even more functional. This is where I decided to try and make the feedback section in order to try and control the number of repeats. After doing some research, I have found that, logically, the delayed signal needs to be fed to the input. This is where I added addition module to which I patched the input signal and then fed the Single Delay back into it. At this moment, the delay was still working properly and the feedback was active and it was not stopping, which is when I realized I needed to create control for it. This is where it started to go downwards, as for this control to be created I needed to introduce another math module to multiply the delay signal and then revert it back to the input. As I said, at this point, due to the lack of my knowledge and experience, the delay was only partially working and I was not able to figure out where I had gone wrong. Nevertheless, for the final step of this synthesizer, I wanted to have a filter cutoff that responds to the delay. I have simply added the 1-Pole Filter and then patched the delay into it and followed through to the Amp/Mixer and then to the output unit. I also patched it into the aforementioned addition module, so the delay recognizes that the filter is reacting towards the feedback signal.

I only wanted a lowpass filter, so this is the one that I pursued and created the control for. After testing it and making sure that it is cutting the high end of the feedback signal, I was sure that my synth was done. (Computer Music, 2021)

Conclusion

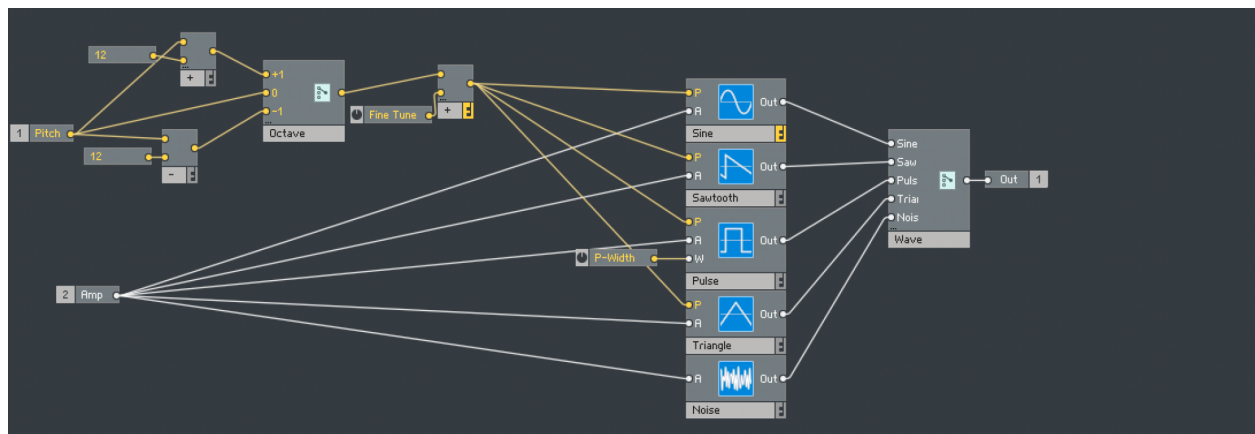
To conclude this essay, I could only say that this project was super productive for me. As someone with little to no experience in synthesis and programming something from scratch, I could definitely conclude that I have learned a lot throughout this process that was nothing close to ordinary. To prove the point, I have even made a demo track to prove that this synth does not lack the ability to work in context and that it could be as good as any other synth if used properly. What made this process a bit more complicated and has definitely helped me learn the basics a bit better, is that I have previously made a synth very similar to this one, but failed to save it properly. Even though it was a painful process to remake it, I really am convinced that it helped me get a better understanding of how Reaktor and the basic synthesis works. To prove that, I have even started implementing techniques and features that I have incorporated in this project to my personal projects. As for what I would improve and add to this synthesizer apart from making it a bit more visually appealing is that I would like to try and make it polyphonic. It does sound good and it works, but is still very limited in terms of musical expression. I would also try and add more effects to it, which would come after doing thorough research on the topic and finding out what I did wrong with the delay section.

Overall, I think I really did learn a lot from this project and I am looking forward to incorporating these techniques on a day to day basis.

Appendix

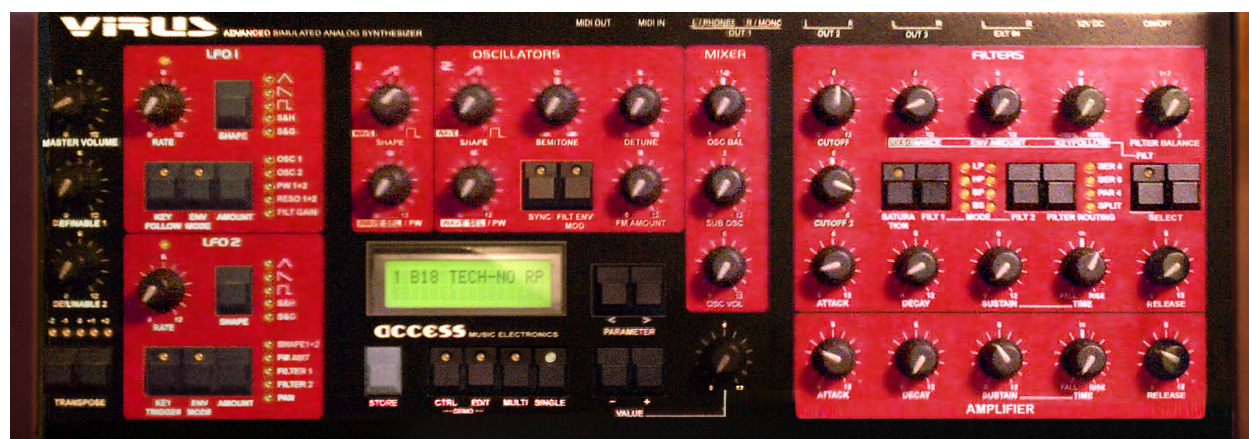


Appendix A : XFER's Serum Oscillator used as a reference for the Fine Tune function



Appendix B : Oscillator Macro

Appendix



Appendix C : Access Virus A - a synthesizer used as a reference for the design of my synthesizer



Appendix D : The final result

Bibliography :

1. Native Instruments (No Date), '*Synths : Reaktor 6 | Komplete - Native Instruments*', Native Instruments, July. Available at :
<https://www.native-instruments.com/en/products/komplete/synths/reaktor-6/>
2. Electrical4U (2020), '*Oscillators : What Are They? (Definition, Types & Application)*', Electrical4U, July. Available at :
<https://www.electrical4u.com/what-is-an-oscillator/>
3. ADSR (No Date), '*How to Build a Synth In Reaktor - A beginners guide*', ADSR, July. Available at :
<https://www.adsrsounds.com/reaktor-tutorials/how-to-build-a-synth-in-reaktor-a-beginners-guide/>
4. SparkFun (No Date). '*Pulse Width Modulation*', SparkFun, July. Available at :
<https://learn.sparkfun.com/tutorials/pulse-width-modulation/all>
5. ADSR (No Date), '*Implementing Math in Reaktor*', ADSR, July. Available at :
<https://www.adsrsounds.com/reaktor-tutorials/implementing-math-in-reaktor/>
6. Electric Trumpet (No Date), '*Reaktor Tutorial - Simple Synth : Envelopes*', Electric Trumpet, July. Available at :
<http://electrictrumpet.com/SympleSynth/sstutorialenv.htm>
7. Rajan, R. (2016), '*A Guide to Building Synths in Native Instruments Reaktor : Part 2*', Ask Audio, July. Available at :
<https://ask.audio/articles/a-guide-to-building-synths-in-native-instruments-reaktor-part-2>
8. Vintage Synth Explorer (No Date), '*Access Virus A*', Vintage Synth Explorer, July. Available at :
<https://www.vintagesynth.com/misc/virusa.php>
9. ADSR (No Date), '*Creating a Custom Reaktor Filter with Modulation*', ADSR, July. Available at :
<https://www.adsrsounds.com/reaktor-tutorials/creating-a-custom-reaktor-filter/>
10. Hutchinson, S. (2020), '*Reaktor Primary 5 : LFO (Low Frequency Oscillator) Tutorial | Simon Hutchinson* [Video]', Simon Hutchinson, July. Available at :
<https://www.youtube.com/watch?v=Hn3dWS60aRY>
11. Owens, J (No Date), '*Pitch or Volume? The Difference Between Tremolo and Vibrato*', Fender, July. Available at :
<https://www.fender.com/articles/tech-talk/pitch-or-volume-the-difference-between-tremolo-and-vibrato>
12. ADSR (No Date), '*How to Build a Synth in Reaktor (Part V) - Tremolo and Glide*', ADSR, July. Available at:

<https://www.adsrsounds.com/reaktor-tutorials/how-to-build-a-synth-in-reaktor-part-v-tremolo-and-glide/>

13. Computer Music (2021). '*How to build a delay effect in Reaktor*', *Music Radar*, July.
Available at :

<https://www.musicradar.com/news/how-to-build-delay-reaktor>