The Sound Of Acid

Virtual Analog Emulation der TB-303 in Max/Msp

Diplomarbeit

Ausgeführt zum Zweck der Erlangung des akademischen Grades
Dipl.-Ing. für technisch-wissenschaftliche Berufe

am Masterstudiengang Digitale Medientechnologien an der
Fachhochschule St. Pölten, **Masterklasse Audiodesign**

von:
Max Kernmayer, BSc
dm171533

Betreuer/in und Erstbegutachter/in: Dipl.-Ing. Patrik Lechner, BSc
Zweitbegutachter/in: FH-Prof. Dipl.-Ing. Andreas Büchele

Wien, 25.04.2020

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.

- ich dieses Thema bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Diese Arbeit stimmt mit der vom Begutachter bzw. der Begutachterin beurteilten Arbeit überein.

....

Ort, Datum Unterschrift

Kurzfassung

Das Ziel dieser Diplomarbeit war es, eine Virtual Analog Emulation der TB-303 in Max/Msp umzusetzen. Durch diesen Prozess soll gezeigt werden wie, und ob es überhaupt möglich ist eine gut klingende Software Emulation in Max/Msp umzusetzen, und ob aktuelle Virtual Analog Modelle in der Lage sind den charakteristischen Klang der TB-303 korrekt zu emulieren. In der Arbeit wurde zunächst die Funktionsweise der analogen TB-303 analysiert, und danach die Implementierung in Max/Msp beschrieben. Mithilfe von Messungen und einer qualitativen Befragung wurde das Ergebnis evaluiert. Damit konnte gezeigt werden, dass die Max/Msp Emulation insgesamt klanglich recht nah an die analoge TB-303 herankommt, jedoch vorallem bei hohen Cutoff Frequenz und Resonanz Einstellungen des Filters an seine Grenzen stößt. Außerdem konnte gezeigt werden, dass Max/Msp nicht die ideale Entwicklungsumgebung für eine solche Emulation darstellt.

Abstract

The aim of this masterthesis was to implement a virtual analog emulation of the TB-303 in Max/Msp. This process is intended to show how and whether it is even possible to implement a good sounding software emulation in Max/Msp, and whether recent virtual analog models are able to correctly emulate the characteristic sound of the TB-303. In the thesis, the functionality of the analog TB-303 was first analyzed, and then the implementation in Max/Msp was described. The result was evaluated using measurements and a qualitative survey. It could be shown that the Max/Msp emulation as a whole comes very close to the analogue TB-303, but especially at high cutoff frequency and resonance settings of the filter reaches its limits. It was also shown that Max / Msp is not the ideal development environment for such an emulation.

Inhaltsverzeichnis

Ehrenwörtliche Erklärung II

Kurzfassung III

Abstract IV

Inhaltsverzeichnis V

1 Einleitung 1

2 TB-303 4

2.1 Die Geschichte der TB-303 & Acid House 4

2.1.1 Rolands kommerzieller Fehlschlag 4

2.1.2 Die Entstehung von Acid House & der Aufstieg der TB-303 6

2.2 Grundlegende Analyse und Funktionsweise der TB-303 8

2.2.1 Überblick & Signalflow 9

2.2.2 Oscillator 10

2.2.3 Filter 11

2.2.4 Envelope Generatoren 14

2.2.5 Accent 15

2.2.6 Slide 16

3 DSP Theorie für Synthesizer 19

3.1 Oszillatoren 19

3.1.1 Generischer digitaler Oszillator und Definition 20

3.1.2 Analog vs. Digital und das Aliasing Problem 27

3.1.3 Quasi bandbegrenzte Oszillator Algorithmen 34

3.2 Filter 39

3.2.1 Simpler Lowpass Filter 39

3.2.2 FIR vs. IIR Filter 41

3.2.3 Analog vs. Digital und die Bilineare Transformation 41

3.3 Nichtlineare Funktionen 42

3.3.1 Tanh 43

3.3.2 Cubic Nonlinear Distortion 44

4 TB-303 Emulations Projekt 46

4.1 Diode Ladder Filter 46

4.1.1 Vom Blockdiagramm zum Filter 46

4.1.2 Python Implementation 59

4.1.3 Gen Implementation 66

4.2 Filter Envelope Modulation 72
4.2.1 Filter Envelope Generator Subpatch (FilterEnvelope) 72
4.2.2 Filter Envelope Modulation Subpatch (FilterEnvMess) 75
4.3 Volume Envelope Modulation 76
4.3.1 Volume Envelope Generator Subpatch (VolEnvelope) 76
4.3.2 Volume Envelope Modulation 77
4.4 Oszillator 78
4.4.1 polyBLEP Sägezahn Oszillator 79
4.4.2 Rechteckwellenform 80
4.5 Accent 82
4.5.1 Accent Sweep Circuit Emulation 84
4.5.2 Accent Envelope 86
4.5.3 Accent VCA Drive 87
4.6 Slide und MIDI 88
4.7 GUI 90
5 Evaluation 92
5.1 Messungen 92
5.1.1 Filter Messungen 92
5.1.2 Oszillator Messungen 97
5.2 Hörtest 99
5.2.1 Testaufbau 100
5.2.2 Testergebnisse 102
6 Fazit 110
Literaturverzeichnis 112
Abbildungsverzeichnis 114
Listingverzeichnis 119

1 Einleitung

Computer mit DAWs, inklusive diverser Plugins, sind heutzutage der Standard in der professionellen Audio Produktion. Durch diese Technologien wurde es auch jungen Produzenten ermöglicht mit einzig und allein einem Laptop und Kopfhörern beziehungsweise Lautsprechern professionelle Musik zu produzieren, ohne sehr viel Geld für diverse Hardware Synthesizern, Drum Machines und Effektgeräten investieren zu müssen, wie es früher der Fall war. Sehr viele klassische Synthesizer wurden mittlerweile im Plugin Format emuliert, um somit jedem den besonderen Klang dieser Geräte für vergleichsweise wenig Geld in digitaler Form zugänglich zu machen. Zusätzlich bietet die digitale Welt zahlreiche Möglichkeiten, die in der damaligen analogen Welt noch nicht verfügbar waren.

Mithilfe von Max/Msp ist es für programmieraffine Produzenten möglich eigene Synthesizer und Audio Effekte zu kreieren, um sie in ihren Produktionen einzusetzen. Jedoch gibt es bisher nur sehr wenige Emulationen analoger Synthesizer die in Max/Msp umgesetzt wurden. Das Ziel dieser Arbeit ist es die TB-303, ein Synth der den Sound elektronischer Musik wie kaum ein anderer geprägt hat, in Max/Msp zu emulieren und einen möglichst originalgetreuen Klang zu erzeugen. Der fertige Synthesizer wird als Max4Live Instrument veröffentlicht um ihn für alle Ableton Nutzer sehr einfach benutzbar zu machen.

Die Arbeit soll während diesem Prozess mehrere Forschungsfragen beantworten. Einerseits soll gezeigt werden, mit welchen Strategien die TB-303 in Max/Msp emuliert werden kann. Mit dem Ergebnis dieser Arbeit soll geklärt werden, ob es überhaupt möglich ist den charakteristischen Klang der TB-303 in Max/Msp originalgetreu zu realisieren (beziehungsweise ob sich Max/Msp für ein solches Projekt als Entwicklungsumgebung eignet), und ob die aktuellen Virtual Analog Modelle in der Lage sind die analoge Hardware korrekt zu emulieren. Weiters soll gezeigt werden welche klanglichen Unterschiede sich zwischen der digitalen Emulation und dem analogen Vorbild feststellen lassen. Außerdem wird der Frage nachgegangen ob lediglich der Filter für den charakteristischen Klang der TB-303 verantwortlich ist.

Um diese Fragen zu beantworten wurde zunächst eine grundlegende Literatur- und Internetrecherche zu den Gebieten Digital Sound Processing (DSP) und Virtual Analog Algorithmen durchgeführt. Weiters wird die TB-303 untersucht, um ihre Funktionsweise genauer zu verstehen, und mit diesem Wissen die Emulation in Max/Msp realisieren zu können.

Danach wird durch die eigenständige Programmierung einer Virtual Analog Emulation gezeigt, wie die TB-303 als digitale Emulation in Max/Msp umgesetzt werden kann, und wie nah die Emulation klanglich an das analoge Original herankommt. Nach der Programmierung wurden Messungen wie Filter Frequency Response und Impulse Response durchgeführt, um das Ergebnis technisch zu verifizieren.

Als letzten Schritt wurde eine qualitative Befragung in Form eines Hörtest durchgeführt um die klanglichen Unterschiede zwischen der analogen TB-303 und der digitalen Max/Msp Emulation herauszufinden.

In Kapitel 2 („TB-303“) der Arbeit wird zunächst die interessante Geschichte der TB-303 erläutert, und wie das Acid Genre entstanden ist. Dieser Abschnitt soll auch als Motivation verstanden werden, warum genau dieser Synthesizer für diese Arbeit ausgewählt wurde. Weiters wird in diesem Kapitel die Funktionsweise der TB-303 analysiert, um einen guten Ausgangspunkt für die Entwicklung der Emulation zu bekommen.

Kapitel 3 („DSP Theory für Synthesizer“) beschäftigt sich mit den DSP Grundlagen für Synthesizer um theoretische Hintergründe, vor allem für Oszillatoren und Filter, zu klären.

In Kapitel 4 („TB-303 Emulations Projekt“) befindet sich eine vollständige Dokumentation der, für diese Arbeit entwickelten, Max/Msp Emulation. Ein großer Teil davon beschäftigt sich mit dem sogenannten Diode Ladder Filter der TB-303, und wie dieser digital emuliert werden kann. Basierend auf Zavalishin’s Modellen aus dem Buch „The Art Of VA Filter Design“ wurde zunächst ein Prototyp des Filter Algorithmus in Python implementiert, da Python bessere Möglichkeiten bietet die Implementation des Filters zu testen und zu analysieren. Danach wurde der Python Code in Max/Msp innerhalb des gen~ Objekts portiert. Auch der Sägezahn Oszillator wurde in gen~ implementiert. Die weiteren Bestandteile des Synthesizers (Rechteckwelle, Filter und Volume Envelopes, Accent und Slide) wurden direkt in Max/Msp umgesetzt.

In Kapitel 5 („Evaluation“) finden sich die Impulse- und Frequency Response Messungen des Filters, sowie Messungen des Oszillators im Zeit- und

Frequenzbereich. Außerdem wird in diesem Kapitel die Durchführung der qualitativen Befragung in Form eines Hörtest beschrieben, sowie die Ergebnisse des Tests bekanntgegeben und interpretiert.

Im letzten Kapitel, Kapitel 6 („Fazit“) werden letztendlich die eingangs gestellten Forschungsfragen aufgrund der Erkenntnisse der Arbeit beantwortet und die Ergebnisse der Arbeit zusammengefasst.

2 TB-303

Die TB-303 ist ein Synthesizer, der den Klang elektronischer Musik wie kaum ein anderes Instrument geprägt hat. Dabei wurde der Synth eigentlich für einen ganz anderen Zweck entwickelt.

In diesem Kapitel wird zunächst auf die interessante Geschichte des großen Aufstiegs des Synthesizers sowie die, durch den Synth erst möglich gemachte, Entstehung des Acid Genres eingegangen, um zu zeigen welchen Einfluss das Gerät auf die Elektronische Musik Welt hatte, beziehungsweise auch heute immer noch hat. Dieser Abschnitt soll auch als Motivation verstanden werden, warum genau dieser Synthesizer für diese Arbeit ausgewählt wurde. Im weiteren Verlauf des Kapitels wird die grundlegende Funktionsweise des analogen Synthesizers analysiert und beschrieben, um ein erstes Verständnis dafür zu bekommen wie die TB-303 eigentlich funktioniert und wie sie ihren charakteristischen Klang erzeugt.

2.1 Die Geschichte der TB-303 & Acid House

Bei der TB-303 handelt es sich um einen monophonen analogen Synthesizer, der 1982 von der Firma Roland auf den Markt gebracht wurde. Die Intention des damaligen Roland Entwicklers Tadao Kikumoto (der „Erfinder“ der TB-303) war es eigentlich mit dem Gerät einen Bassisten imitieren zu können. Zeitgleich mit der TB-303 ist auch die TR-606 Drum Machine auf den Markt gekommen, die mithilfe eines Sync-Kabels mit der TB-303 verbunden und synchronisiert werden konnte. Somit konnte ein Bandfundament, bestehend aus Schlagzeug und Bass, erzeugt werden, zu dem Musiker (Gitarristen, Keyboarder, usw.) zu Hause mit ihrem Instrument proben können. Dies war zumindest die Intention des Herstellers der beiden Geräte.

2.1.1 Rolands kommerzieller Fehlschlag

Eine tolle Idee, könnte man meinen. Dem war aber nicht so. Bereits 1984, also schon zwei Jahre nach Start der Serienproduktion, wurde die Produktion der

beiden Geräte aufgrund schlechter Verkaufszahlen eingestellt. Die TB-303 und TR-606 waren also zunächst ein kommerzieller Flop. (Brinkmann, 2018)

Abbildung 1 Oscar Peterson (Jazzmusiker) mit Roland PianoPlus, TB-303 und TR-606. Das Bild stammt aus einer alten Roland Broschüre von 1982. Dieses Bild zeigt sehr gut, wofür Roland die TB-303 und TR-606 entwickelt hat, und als was die beiden vermarktet wurden.

Dabei gab es einige Gründe warum niemand die TB-303 zur Zeit ihres Erscheinens haben wollte. Allen voran klang das Gerät einfach nicht wie eine Bassgitarre. Auch die wenigen Klangbearbeitungsmöglichkeiten machten den Synthesizer schon damals zu einem vergleichsweise sehr limitierten Gerät. Außerdem konnte die TB-303 nicht, wie man es von einer Bassgitarre, oder auch von jedem anderen Synthesizer oder Instrument dieser Zeit, gewohnt war gespielt werden. Es gab kein Keyboard, sondern lediglich einen etwas kompliziert zu bedienenden Sequencer. Da während der Eingabe der einzelnen Noten die Sequenz nicht abgespielt werden konnte, mussten auf den einzelnen 16 Steps des Sequencers die Noten manuell ausgewählt werden, ohne der Möglichkeit die gerade hinzugefügten Noten anhören zu können. Dies gab der TB-303, vor allem wegen ihres Sequencers, den Ruf nicht wirklich benutzbar zu sein, da oft auf mysteriöse Weise die Noten Eingaben des Benutzers vom Gerät in eine seltsame Sequenz übersetzt wurde. (Heermans, 2016)

Es gab aber noch weitere Einflüsse des damaligen Marktes weswegen der Synthesizer kommerziell keinen Erfolg bringen konnte. 1983, also ein Jahr nach Veröffentlichung der TB-303 wurde auf der NAMM-Show die MIDI-Schnittstelle von Sequential Circuits und Roland selbst vorgestellt. Ein neuer Standard, der quasi über Nacht alle Synthesizer ohne MIDI veraltet aussehen ließ. Außerdem wollten Keyboard Spieler Mitte der 80er Jahre polyphone Synthesizer, mit vielen

Presets die möglichst realistisch klangen. Beides Eigenschaften mit denen die TB-303 nicht dienen konnte. (Van Dijk, 2019)

Somit wurde letzten Endes der, bereits von Anfang an moderat preisige, Synthesizer zu sehr günstigen Preisen abverkauft, oder ist in Second Hand Shops gelandet.

2.1.2 Die Entstehung von Acid House & der Aufstieg der TB-303

Erst dadurch, dass die TB-303 Mitte der 80er Jahre in Second Hand Shops für sehr wenig Geld verfügbar war, fiel sie in die Hände der aufstrebenden House & Techno Produzenten Szene. Bei diesen Produzenten handelte es sich, vor allem zu Beginn der Entstehungsgeschichte des Musik Genres, weitgehend um Amateurmusiker ohne klassischen Musiktheoretischen Training. Sie verwendeten zum produzieren ihrer Musik also das Equipment, dass für sie günstig zur Verfügung stand. Dabei wurden, aus den dadurch entstehenden Limitationen, natürlich manche Geräte auch zweckentfremdet. (Wegscheider, 2008)

Eines Tages fand Pierre Jones (alias DJ Pierre) eine so gut wie neue TB-303 in einem dieser Second Hand Shops, und kaufte sie um 40 US$. Der Chicagoer DJ & Produzent und sein Freund Earl „Spanky“ Smith (zusammen später bekannt unter dem Pseudonym Phuture), steckten den Synthesizer mit den Geräten die sie bereits besitzten zusammen, und fingen an damit zu jamen. Keiner von beiden wusste wie man den Sequencer bedient, also ließen sie den Synthesizer einfach die bereits eingespeicherten Sequenzen abspielen. Während Spanky auf einer Drum Machine einen Drum Loop programmierte, fing Pierre damit an, an den verschiedenen Reglern der TB-303 zu drehen, um die Möglichkeiten der Soundgestaltung des Geräts auszutesten. Während Pierre an den Reglern drehte, hörte Spanky plötzlich seltsame quietschende alienhafte Klänge, die er noch nie zuvor gehört hatte, und ermutigte Pierre dazu weiter mit den Reglern zu experimentieren. Pierre entdeckte also in diesem Moment wie noch nie zuvor gehörte Acid Klänge (eine Kombination aus hoher Cutoff Frequenz, hoher Resonanz und Env Mod Einstellung des Filters), abseits des nicht besonders realistisch klingenden Bass Sounds, mit der TB-303 erzeugt werden können. Er drehte für weitere 3-4 Stunden spielerisch an den Reglern des Synthesizers und modulierte somit den Sound in Echtzeit. Eine solche Spielweise war für das Gerät nie vorgesehen. Aus dieser langen Jam Session entstand das erste Stück des Acid House Genres: „Acid Trax“. (Heermans, 2016)

Damals war es üblich, dass House Produzenten ihre neuen Stücke auf Kassette aufnahmen und dem DJ eines Clubs gaben. Wenn dieser dann das Stück spielte,

konnten der Produzent, wie auch der DJ, die Reaktion des Publikums beobachten, und daraus ihre Schlüsse ziehen ob das Stück gut ankommt, oder weitere Anpassungen vorgenommen werden müssen. Diese Kultur startete damals die Karrieren vieler Künstler. Die damalige Szenegröße Ron Hardy, DJ des Clubs Music Box in Chicago, war der erste der „Acid Trax“ von DJ Pierre auf einer Kassette überreicht bekommen hatte. Er spielte das Stück an diesem Abend sogar mehrere Male. Zu Beginn waren die Leute scheinbar mit dem neuen Sound überfordert, und verließen mit Aussprüchen wie „What the fuck is this?!“ die Tanzfläche. Schlussendlich reagierte das Publikum extrem positiv auf den unbekannten neuen Sound. Acid Trax erschien 1985 auf Schallplatte, wurde aber laut DJ Pierre selbst davor bereits regelmäßig in den Clubs gespielt. (Wegscheider, 2008)

DJ Pierre, der also als einer der Erfinder des Acid Sounds gilt, beschreibt den eben geschilderten Vorgang folgendermaßen:

> „Für die Music Box machte ich meinen allerersten Track, ‚Acid Trax’. Ein Tape davon gab ich Ron Hardy, und als er es spielte, drehten die Leute völlig durch. In derselben Nacht brachte er es noch ein paar Mal, und jedes Mal wurde es wahnsinniger. Marshall Jefferson hat das Stück dann an das Label Trax vermittelt.“ (Nieswandt, 2007)

Zunächst wurde irrtümlicherweise Ron Hardy dieses erste Acid Stück zugeschrieben. Die Club Besucher benannten den neuen Sound „Ron Hardy’s Acid Trax“, was auch für die letztendliche Veröffentlichung des Stückes namensgebend war. Dabei soll der Name des Stückes, laut der Urheber, aber nichts mit der Droge LSD (die auch Acid genannt wird) zu tun haben. (Wegscheider, 2008)

Acid Trax wurde letztendlich zu einem großen Hit im Underground, der mit seiner Veröffentlichung den Grundstein für das, heute immer noch hoch relevante, Acid Genre legte.

> „Der Hype um die 303 war nun nicht mehr zu bremsen, und es entwickelte sich ein neuer Musikstil, der auf Drum Loops im Chicagoer Housestyle, kombiniert mit experimentellen Sequencer-Loops aus der 303 teilweise sogar unter Zugabe eines aufgedrehten Gitarrenverzerrers setzte.“ (Brinkmann, 2018)

Mit dem großen Hype um den neuen Acid Sound, der auch kurz darauf nach Europe überschwappte, wurde die TB-303 (die zu diesem Zeitpunkt schon nicht mehr produziert wurde) zu einem viel gesuchtem Gerät auf dem Gebrauchtmarkt.

Die große Nachfrage lies den Preis immer weiter steigen, und es wurde immer schwerer überhaupt eine zum Verkauf stehende TB-303 zu finden. Auch heute, über drei Jahrzehnte später, hat sich an dieser Situation am Gebrauchtmarkt kaum etwas geändert.

Wegen der großen Nachfrage wurden über die Jahre auch unzählige analoge Klone von verschiedenen Firmen entwickelt. Zu den bekanntesten zählen: x0xb0x, Cyclone TT-303 Bass Bot, Dyn Sync RE-303, und seit 2020: Behringer TD-3. Darüber, wie gut die verschiedenen Klone den Klang des Originals emulieren lässt sich natürlich streiten, und ist ein häufiges Diskussionsthema in der Synthesizer Liebhaber Szene. Auch einige Modifikationen der TB-303, die das Gerät mit zusätzlichen Funktionen ausstatteten, waren über die Jahre im Umlauf. Zu den bekanntesten dieser Modifikationen zählt der Devilfish Mod.

All dies zeigt wie besonders der charakteristische Klang der originalen TB-303 ist. Es stellt sich natürlich die Frage wie dieser charakteristische Klang entsteht, und warum nur das originale analoge Gerät diesen Klang erzeugen kann. Daher kommt auch die Motivation eine Virtual Analog Emulation dieses Synthesizers herzustellen, um während dieses Prozesses herauszufinden wie dieser Acid Klang entsteht.

2.2 Grundlegende Analyse und Funktionsweise der TB-303

Abbildung 2 Roland TB-303

Um in weiterer Folge dieser Arbeit eine Virtual Analog Emulation der TB-303 herstellen zu können, muss als erstes die grundlegende Funktionsweise der

Klangerzeugung des Synthesizers analysiert und beschrieben werden. Danach werden die einzelnen Bestandteile der TB-303 genauer beschrieben, um eine gute Grundlage für die Programmierung der Emulation eben dieser zu schaffen.

2.2.1 Überblick & Signalflow

Die TB-303 ist, wie bereits erwähnt, ein monophoner Synthesizer, daher kann immer lediglich nur eine Note zur selben Zeit gespielt werden. Der Oszillator kann wahlweise entweder eine Sägezahn- oder Rechteckwellenform erzeugen. Das vom Oszillator erzeugte Signal kann dann mit einem Filter bearbeitet werden. Um die Parameter des Filters zu steuern stehen folgende Drehregler im oberen Teil des Geräts (siehe Abbildung 2) zur Verfügung: Cutoff Freq, Resonance, Env Mod und Decay.

Mithilfe des Sequencers im unteren Teil des Geräts können verschiedene Patterns programmiert werden. Die Patterns können auch im Gerät gespeichert und wieder geladen werden. Im Sequencer können auch die, ebenfalls für den Klang der TB-303 sehr charakteristischen, Slide und Accent Features auf einzelne Noten angewendet werden. Zur Steuerung der Stärke des Accent gibt es ebenfalls einen Drehregler im oberen Teil des Geräts.

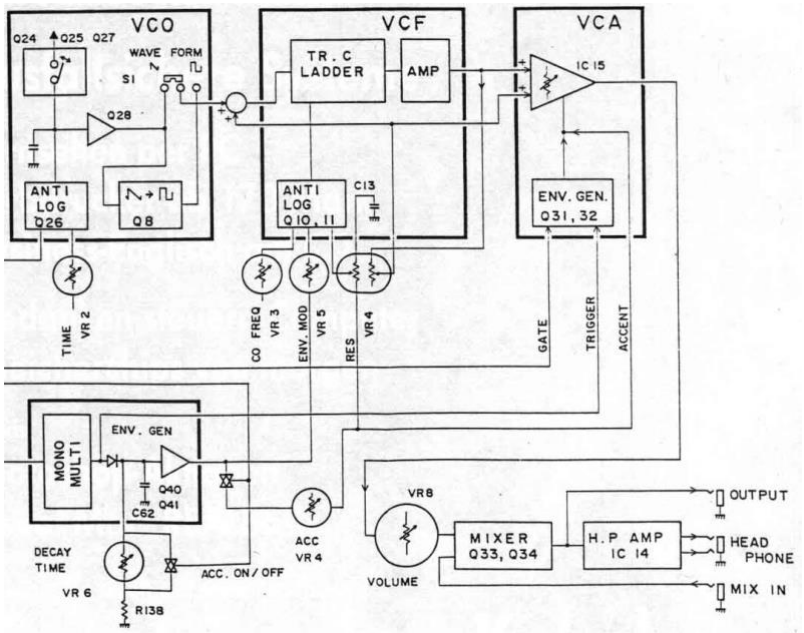

Abbildung 3 TB-303 Block Diagramm aus dem Service Manual

In Abbildung 3 ist ein Block Diagramm der TB-303 aus dem Service Manual zu sehen. Das Block Diagramm gibt einen guten Eindruck zum Signalflow des Geräts. Das vom Oszillator erzeugte Signal fließt direkt in den Diode Ladder Filter und danach in einen Verstärker (VCA), der durch einem Envelope Generator moduliert wird. Ein zweiter Envelope Generator steht zur Modulierung der Cutoff Frequenz des Filters zur Verfügung. Durch den Decay Drehregler kann der Decay des Filter Envelopes verändert werden. Der erzeugte Envelope fließt durch den Env Mod Drehregler, daher kann mit dem Regler die Tiefe der Modulation verändert werden. Besonders interessant ist es den Signalflow der Accent Funktion genauer anzusehen. Die mit dem Accent Drehregler eingestellte Stärke wirkt sich auf die Envelope Modulation, die Resonanz und auch den Verstärker am Ende aus. Eine accented Note bewirkt also Veränderung an fast jedem Parameter. In Abbildung 4 ist eine vereinfachte Darstellung des Signalflows zu sehen.

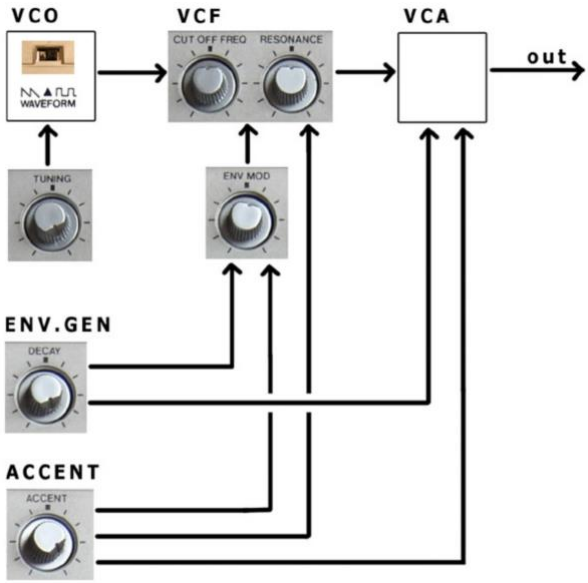

Abbildung 4 Vereinfachte Darstellung des Signal Flow der TB-303

2.2.2 Oscillator

Die TB-303 enthält lediglich einen Oszillator. Die Wellenform des Oszillators kann an der originalen analogen Hardware über einen Schiebeschalter an der Rückseite des Gehäuses umgeschalten werden. Sägezahn- und Rechteckwellenformen stehen zur Auswahl.

Bei der Sägezahnwellenform handelt es sich um eine, in den meisten analogen Synthesizern, übliche Art der Wellenform (siehe Abbildung 5).

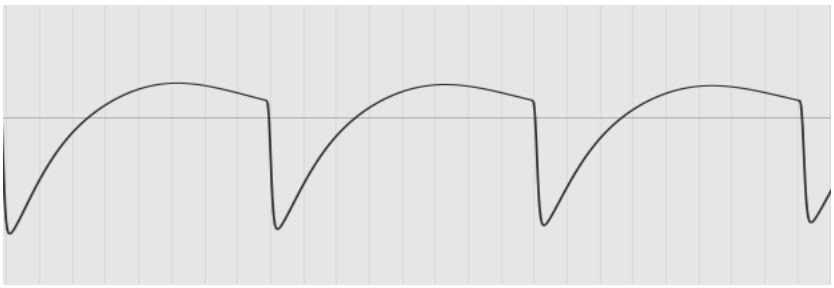

Abbildung 5 Sägezahnwellenform der TB-303 (drei cycles).

Die Rechteckwellenform hingegen ist sehr unüblich, und weit weg von einem idealen Rechteck (siehe Abbildung 6). In der analogen Hardware wird die Rechteckwelle mithilfe eines Waveshapers aus der Sägezahnwelle erzeugt. Die genauen Eigenschaften des Waveshapers sind jedoch nicht bekannt.

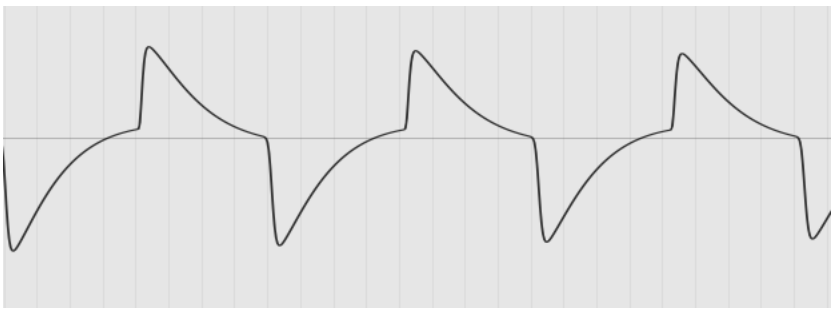

Abbildung 6 Rechteckwellenform der TB-303 (drei cycles).

Wichtig anzumerken ist, dass es sich bei den Wellenformen in Abbildung 5 & 6 um eine Messung am finalen Output der TB-303 handelt. Somit hat auch der Filter (der für diese Messungen auf die höchstmögliche Cutoff Frequenz und die niedrigste Resonanz Einstellung gesetzt wurde) einen Einfluss auf ihre Form. Es ist leider nicht möglich eine Messung direkt am Output des Oszillators durchzuführen.

2.2.3 Filter

Der Filter der TB-303 ist wohl der meistdiskutierteste Bestandteil des Synthesizers, und ein großer Teil dieser Arbeit beschäftigt sich damit. Die wenigen Drehregler die am Gehäuse der TB-303 zu finden sind widmen sich, wie vorher schon beschrieben, fast alle der Modulation des Filters. Es handelt sich hierbei um einen 4-pole Diode Ladder Filter mit einer Flankensteilheit von 24dB/Oktave. Achtung: oft wird der Filter fälschlicherweise als 3-pole 18dB/Oktave Filter beschrieben! Der Filter besteht aus vier in Reihe geschaltenen Lowpass Filtern, daher kommt auch der Name „Ladder“ Filter.

Dabei sollte der Diode Ladder Filter nicht mit dem Transistor Ladder Filter, der in Moog Synthesizern zu finden ist, verwechselt werden. Die beiden Filter sind zwar sehr ähnlich in ihrem Aufbau, es gibt aber einige entscheidende Unterschiede. Im nächsten Abschnitt wird die Funktionsweise des Diode Ladder Filters anhand eines Vergleichs mit dem Moog Transistor Ladder Filter beschrieben.

2.2.3.1 Transistor Ladder Filter im Vergleich mit Diode Ladder Filter

Abbildung 7 VCS-3 Synthesizer der Firma EMS. Einer der ersten Synthesizer mit Diode Ladder Filter.

Der Diode Ladder Filter wurde ursprünglich von der Firma EMS entwickelt (und als erstes in ihren eigenen Synthesizer Modellen verbaut, z.B. im VCS-3 (siehe Abbildung 7), um das Patent der Firma Moog für den Transistor Ladder Filter zu umgehen. Daher wurden zum Bau des Filters Dioden statt Transistoren verwendet, da sie durch ihre elektrotechnischen Eigenschaften eine ähnliche Funktionsweise aufweisen. Die Diode Ladder funktioniert daher auch nach einem ähnlichen Prinzip wie die Moog Transistor Ladder. Die Diode erlaubt oder begrenzt unter dem Einfluss einer angelegten Steuerspannung den Stromfluss zu einem Filterkondensator, so dass sich die Cutoff Frequenz des Filters ändert, wenn die Steuerspannung variiert wird. Der große Unterschied der beiden Schaltungen besteht darin, dass in der Transistor Ladder Schaltung die einzelnen Poles, oder Filter Stufen, voneinander elektrisch isoliert sind, während in der Diode Ladder Schaltung dies nicht der Fall ist. Hier interagieren die einzelnen Filter Stufen elektrisch miteinander. Dieser Umstand macht es weitaus schwieriger den Diode Ladder Filter mathematisch zu analysieren, da unter

anderem mit erhöhter Resonanzeinstellung das Verzerrungsverhalten der Schaltung nichtlinearer und schwerer vorauszusagen wird.

„Performers often say that the diode ladder filter has a "dirtier" sound, which is probably the effect of the less predictable distortion behavior.“ (Electronic Music Wiki)

In Abbildung 8 ist ein simples Block Diagramm des Moog Transistor Ladder Filters zu sehen. Die Transistor Ladder Filter Schaltung besteht aus vier, in Reihe geschaltenen, Lowpass Filtern. Um die Ladder herum befindet sich ein globaler Zero Delay Feedback Loop. Die einzelnen Filter Stufen bekommen lediglich das Signal des vorhergehenden Filters, und sind ansonsten, wie vorher beschrieben, voneinander abgeschirmt.

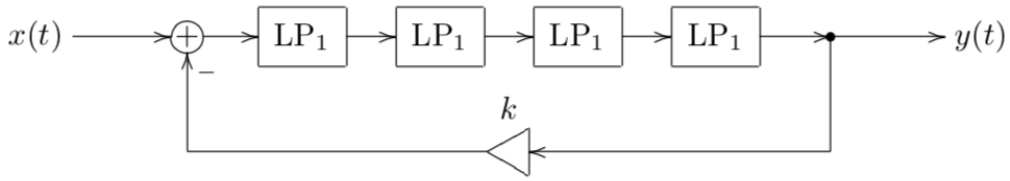

Abbildung 8 Simples Block Diagramm des Moog Transistor Ladder Filter

Die Diode Ladder Schaltung besteht ebenfalls aus vier in Reihe geschaltenen Lowpass Filtern. Hier gibt es aber zwischen den einzelnen Filter Stufen zusätzliche Zero Delay Feedback Loops, somit interagieren die einzelnen Filter Stufen miteinander. Dabei wird, wie im Block Diagramm in Abbildung 9 zu sehen ist, das Output Signal einer Filter Stufe als Feedback in den jeweils davor liegenden Filter eingespeist. Diese komplizierten Verbindungen zwischen den einzelnen Lowpass Filtern sind für den charakteristischen „dreckigen“ Diode Ladder Filter Sound verantwortlich. Deswegen ist aber auch das Verhalten der Diode Ladder Schaltung schwerer vorauszusagen als bei der Transistor Ladder Schaltung. Außerdem befindet sich um die gesamte Ladder, wie auch bei der Transistor Ladder Schaltung, ebenfalls ein globaler Zero Delay Feedback Loop.

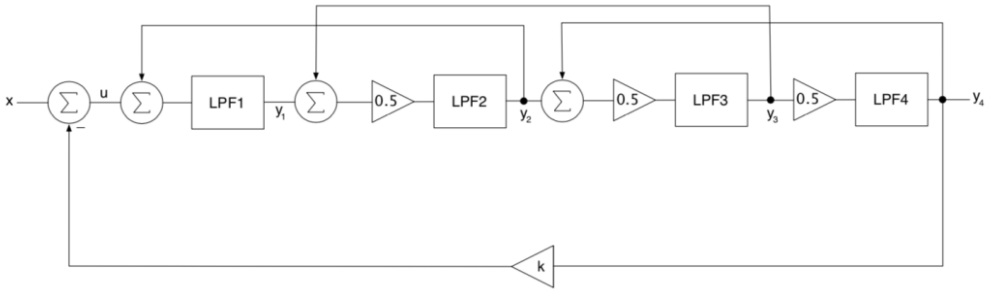

Abbildung 9 Simples Block Diagramm des Diode Ladder Filter

2.2.4 Envelope Generatoren

Die TB-303 hat (wie im Block Diagramm in Abbildung 3 zu sehen ist) insgesamt zwei, voneinander unabhängige, Envelope Generatoren eingebaut. Beide Generatoren generieren einen AD (Attack und Decay) Envelope, mit exponentiell abfallenden Decay. Einer der Envelope Generatoren ist für die Modulierung des Filters zuständig, der andere moduliert den Verstärker am Ende, und somit die Amplitude des Outputs.

2.2.4.1 Volume Envelope Modulation

Der Envelope Generator, der den Verstärker moduliert, hat eine schnelle Attack Zeit und eine eher längeren Decay Zeit, die auf diesen Wert fixiert ist. Der Envelope wird für nichts anderes verwendet, er ist nur für die Modulation der Amplitude zuständig. (Whittle, 1999)

2.2.4.2 Filter Envelope Modulation (EnvMod)

Der zweite Envelope Generator moduliert die Cutoff Frequenz des Filters. Er hat ebenfalls eine schnelle Attack Zeit, aber eine variablen Decay Zeit, die über den Decay Drehregler steuerbar ist. Die Modulationstiefe, also wie stark sich die Envelope Modulation auf die Cutoff Frequenz auswirkt, kann über den Env Mod Drehregler verändert werde, wobei die Modulation bei der originalen analogen Hardware jedoch nicht komplett ausgeschaltet werden kann. (Whittle, 1999)

Daher ist auch die Filter Envelope Modulation ein wichtiger Bestandteil des Sounds der TB-303, bzw., des Diode Ladder Filter Sounds. Ohne der Modulation würde der Diode Ladder Filter alleine nicht den gewollten Acid Sound erzeugen können.

2.2.5 Accent

Die Accent Funktion ist recht einzigartig und lässt sich, in der Form wie sie in der TB-303 eingebaut ist, in nur wenigen anderen Synthesizer finden. Accent macht es möglich ausgewählte Noten quasi stärker zu betonen. In der originalen Hardware kann im Sequencer bei jeder beliebigen Note die Accent Funktion eingeschalten werden. Ein Irrglaube ist es, dass accented Noten einfach nur lauter sind als die anderen Noten. Es passiert innerhalb der Schaltung, wie in Abbildung 3 und 4 zu sehen ist, viel mehr als nur eine Anhebung der Lautstärke. Eine accented Note hat Auswirkung auf die Filter Envelope Modulation, die Resonanz und den Verstärker.

Während einer accented Note wird beim Filter Envelope Generator die Steuerung des Decay Parameter (durch den Decay Drehregler) kurzgeschlossen und die Decay Zeit auf einen recht kurzen Wert gesetzt (genau wie, wenn der Decay Drehregler auf seiner Null Position wäre). Weiters wird im analogen Gerät dem Verstärker mehr Steuerspannung zugeführt. Dadurch wird die Note einerseits lauter, andererseits wird durch eine spezielle Schaltung (RC Network) auch die Attack des Signals ein wenig länger. Außerdem gibt es noch eine weitere Schaltung durch die sich accented Noten auf den Filter auswirken: der „Accent Sweep Cuircuit“. Im Grunde liefert die Schaltung eine etwas erhöhte Steuerspannung an den Filter, wodurch die Cutoff Frequenz während accented Noten ein wenig erhöht wird. Dieses Verhalten ist, wie auch die Erhöhung der Lautstärke bei accented Noten im Verstärker, mit dem Accent Drehregler steuerbar. (Whittle, 1999)

2.2.5.1 Accent Sweep Circuit

Richtig interessant wird es aber erst, wenn man sich genauer ansieht was dieser Accent Sweep Cuircuit noch bewirkt. Im analogen Original Gerät passiert nämlich folgendes:

Der Accent Drehregler beeinflusst die Ladung eines Kondensators. Dieser ist in der Schaltung mit einem der beiden Summierknoten der Filter Frequenz verbunden. Wenn der Accent Drehregler in seiner Null Position steht, wird der Filter hauptsächlich durch den Output des Filter Envelope Generator moduliert, der Kondensator des Reglers bekommt aber auch schon eine kleine Ladung, und gibt diese Ladung ab, da er sich immer entladen muss. Das bedeutet, dass der Filter plötzlich eine etwas erhöhte Steuerspannung erhält, und sich dadurch die Cutoff Frequenz erhöht. Es ändert sich in Null Stellung des Accent Drehreglers jedoch nur sehr wenig an der Cutoff Frequenz, da eben nur eine kleine Ladung vom Kondensator abgegeben wird.

Wenn der Accent Drehregler dann weiter aufgedreht wird, erhöht sich natürlich die Ladung des Kondensators, und dadurch auch die Cutoff Frequenz des Filters. Bei höheren Ladungen wird das Signal, dass an den Filter geht in der Schaltung aber geglättet.

Für eine einzelne accented Note bedeutet dies folgendes: der Filter macht „Wow“, und öffnet und schließt sich in einer schnellen und glatten Kurve, die etwa so wie Abbildung 10 aussieht.

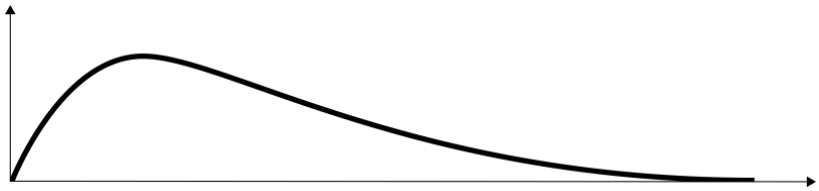

Abbildung 10 Filter Cutoff Modulation bei einer einzelnen accented Note.

Richtig interessant wird dieses Verhalten aber wenn mehrere accented Noten in kurzer Zeit aufeinander folgen. Da sich der Kondensator noch nicht vollständig von der vorherigen Note entladen hat, und somit noch eine Restladung übrig hat, wird bei der zweiten accented Note noch etwas mehr Ladung vom Kondensator abgegeben. Dadurch wird die Cutoff Frequenz des Filters um ein wenig mehr erhöht als bei der vorherigen Note. Durch dieses Verhalten steigt die Cutoff Frequenz weiter mit jeder accented Note, die gespielt wird, bevor sich der Kondensator vollständig entladen konnte. Abbildung 11 illustriert dieses Verhalten. (Whittle, 1999)

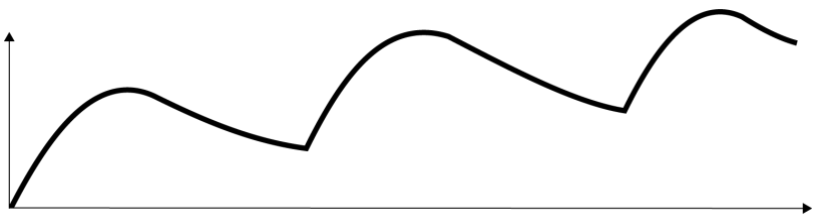

Abbildung 11 Filter Cutoff Modulation bei mehreren accented Noten in Folge.

2.2.6 Slide

Slide ist eine zweite spezielle Funktion der TB-303, um einzelne Noten im Sequencer zu bearbeiten. Mit dieser Funktion kann einprogrammiert werden, dass eine Note in die nächste hineingleitet (auch in vielen Synthesizer als Glide, Legato oder Portamento benannt).

Im Diagramm in Abbildung 12 ist zu sehen wie sich der Sequencer ohne Slide Noten verhält. In diesem Beispiel werden drei Sechszehntelnoten hintereinander gespielt. Jede Note hat eine Dauer von sechs Takten in der internen Clock des Sequencers. Die Note wird aber noch durch ein Gate geschickt. Das Gate öffnet sich an Takt 0 und schließt sich wieder in der Mitte von Takt 3. Bei einer normalen Sechszehntelnote ohne Slide ist das Gate also 3,5 Takte lang geöffnet und 2,5 Takte lang geschlossen.

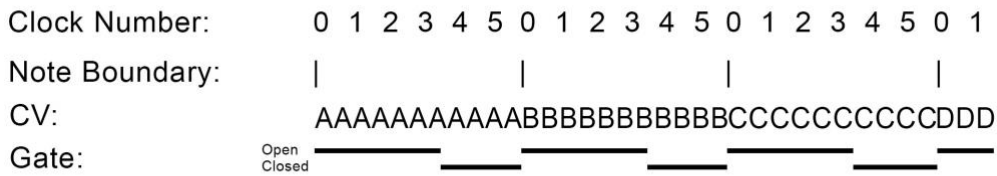

Abbildung 12 Verhalten des Sequencers über die Dauer von drei Sechzehntelnoten, ohne Slide Noten

Wenn Slide auf eine Note programmiert wurde, wird die Slide Schaltung eigentlich auf die darauffolgende Note angewendet. Ein weiterer Effekt der Funktion ist, dass das Gate, bei der Note auf die Slide programmiert ist, sich nicht schließt, und bis zur nächsten Note offen bleibt. In Abbildung 13 ist dies in einem Diagramm dargestellt.

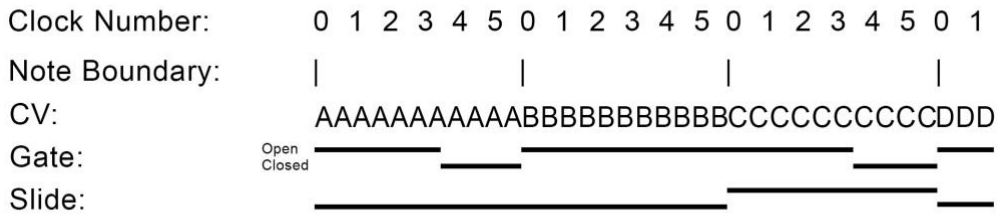

Abbildung 13 Verhalten des Sequencers über die Dauer von drei Sechzehntelnoten, inklusive einer Slide Note (B)

Im Diagramm (Abbildung 13) ist Slide auf der Note B programmiert. Deswegen wird Note C von der Slide Schaltung beeinflusst. Note B wird in ihrer normalen Tonhöhe gespielt, aber hört nicht nach 3,5 Takten der internen Clock auf zu spielen, da das Gate bis zur zur nächsten Note geöffnet bleibt. Note C wird vom

Gate nicht wie eine neue Note behandelt, das Gate bleibt daher einfach weiterhin offen. An der Steuerspannung (CV) die an den Oszillator geht (dadurch wird die Tonhöhe gesteuert) ändert sich aber natürlich etwas, da es sich ja um eine neue Note handelt.

Der Oszillator erhält im Falle einer Slide Note seine Steuerspannung über die Slide Schaltung. Die Steuerspannung steigt von der Tonhöhe von Note B zur Tonhöhe von Note C. Dies passiert während Note C! Das Ende dieses Notenübergangs ist außerdem nicht hörbar, da sich das Gate bei Note C wieder nach 3,5 Takten der internen Clock schließt. (Whittle, 1999)

Somit ist nun die TB-303 und die Funktionsweise ihrer einzelnen Bestandteile analysiert, und wir haben dadurch eine gute Idee wie wir vieles davon in der digitalen Welt emulieren können. Im Kapitel TB-303 Emulations Projekt wird dann beschrieben wie die einzelnen Teile, mithilfe dieser Analyse, emuliert wurden.

3 DSP Theorie für Synthesizer

Um die TB-303 digital emulieren zu können, müssen zunächst einige Grundlagen des Digital Sound Processing geklärt werden, um die Anforderungen und richtigen Strategien für ein gutes Virtual Analog Modell zu finden. In diesem Kapitel wird auf die wichtigsten theoretischen DSP Grundlagen für die Programmierung eines Synthesizers eingegangen. Zunächst werden theoretische Grundlagen zur Erzeugung von digitalen Oszillatoren geklärt. Weiters finden sich Grundlagen zu digitalen Filtern, sowie den nichtlinearen Funktionen, die im weiteren Verlauf der Arbeit verwendet werden.

3.1 Oszillatoren

In diesem Kapitel wird zunächst die Funktionsweise digitaler Oszillatoren anhand der „naiven“ Oszillator Algorithmen, die auf den idealen mathematischen Funktionen der gewählten Wellenformen basieren, beschrieben. Weiters wird geklärt welche Probleme diese naiven digitalen Modelle, im Vergleich mit analogen Oszillatoren, mit sich bringen, und welche besseren Möglichkeiten es für die Generierung einer Virtual Analog Wellenform gibt.

Oszillatoren bilden die Basis für die meisten Arten der Synthese, und sind für die Klangerzeugung zuständig. Ein Oszillator schwingt nach einer sich wiederholenden Wellenform mit einer Grundfrequenz, sowie einer Spitzenamplitude. Neben der Frequenz (oder auch Tonhöhe) und der Amplitude mit denen der Oszillator schwingt, ist auch die Form seiner Wellenform ein wichtiges Merkmal.

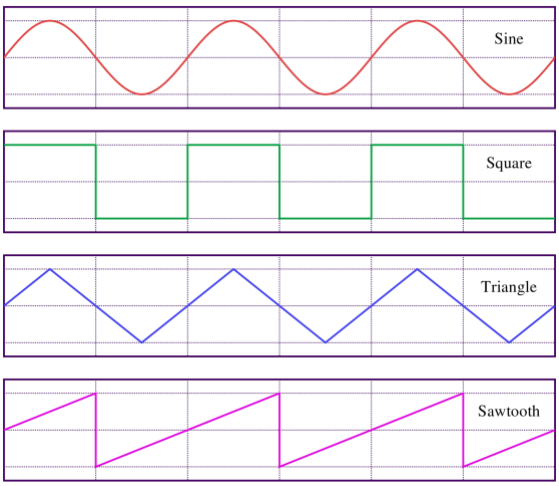

Abbildung 14 Sinus, Rechteck, Dreieck und Sägezahn Wellenformen.

Die Sinus, Rechteck, Dreieck und Sägezahn Wellenformen, die in Abbildung 14 zu sehen sind, sind dabei die am häufigsten verwendeten Wellenformen. Es ist natürlich möglich verschiedenste Varianten von Wellenformen zu erzeugen, diese vier bieten aber eine sehr gute Basis für verschiedenste Synthese Anforderungen, da das Spektrum vom weichen und einfachen Klang einer Sinuswelle, bis zum obertonreichen Brummen einer Sägezahnwelle reicht.

Oszillatoren werden im allgemeinen Fall durch ein Keyboard oder ein anderes MIDI Gerät angesteuert. Ein Tastendruck schickt dabei einen MIDI Notenwert an den Oszillator. Dieser wird zuvor noch in die zum Notenwert zugehörige Frequenz (Hz) umgewandelt. Der Oszillator schwingt nun in seiner Wellenform, mit einer Periodendauer entsprechend der eingegebenen Frequenz. Das Ausgangssignal des Oszillators kann dann in einem Synthesizer auf verschiedenste Weisen bearbeitet werden, um den Sound weiter zu formen. (Karras, 2018)

3.1.1 Generischer digitaler Oszillator und Definition

Die meisten digitalen Oszillatoren haben dieselbe generische Topologie. Ein Phasenakkumulator generiert eine einfache Sägezahnwelle bei der Frequenz und Phase gesteuert werden können und eine frequenzabhängige speicherlose Funktion bildet die Welle in der gewünschten Form ab. Ein optionaler Post-Filter kann am Ende noch zur Anpassung der hohen Frequenzen eingesetzt werden. Abbildung 15 zeigt ein Block Diagramm eines generischen digitalen Oszillators.

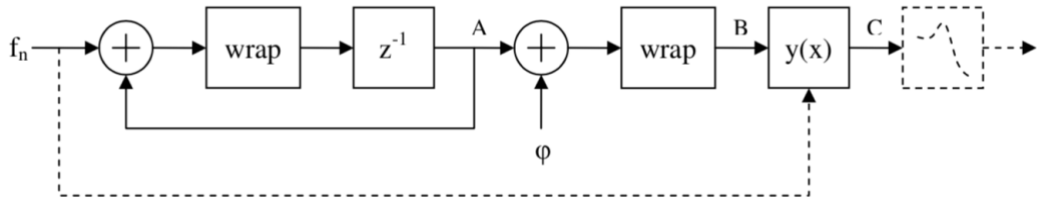

Abbildung 15 Topologie eines generischen digitalen Oszillators

f_n an Beginn des Block Diagramms bezeichnet die normalisierte Grundfrequenz $2Nf_0/f_s$, wobei f_0 für die Grundfrequenz und f_s für die Samplerate steht.

Die wrap(x) Funktion arbeitet wie folgt: wenn x ≥ N: x - 2N, wenn x < -N: x + 2N, andernfalls wird wiederholt, bis das Ergebnis innerhalb von [-N, N] liegt.

Die z^{-1} Operation bedeutet einen Delay um 1T, wobei T = 1/f_s, und somit ein Delay um 1 Sample gemeint ist.

φ steht für die Phasenverschiebung in Radianten: N/π, und wird mit dem bestehenden Signal summiert.

y(x) beschreibt den vollen Zyklus einer Mapping Funktion, hiermit wird das bereits bestehende Signal auf die gewünschte Wellenform abgebildet (z.B. eine Sinuswelle). Dies kann auch mithilfe einer im vorhinein berechneten Lookup Tabelle durchgeführt werden. (Frei, 2019)

3.1.1.1 Naiver digitaler Sinus Oszillator Algorithmus

Der Sinus ist wohl die simpelste und reinste Wellenform. Die Sinus Funktion hat eine Periodendauer von 2π Radiant, und eine Spitzenamplitude von +/- 1, wie in Abbildung 16 zu sehen ist. In einem digitalen System besteht die generierte Welle aus einer Serie einzelner Werte (mit gleichem Abstand voneinander, aufgrund der Samplerate).

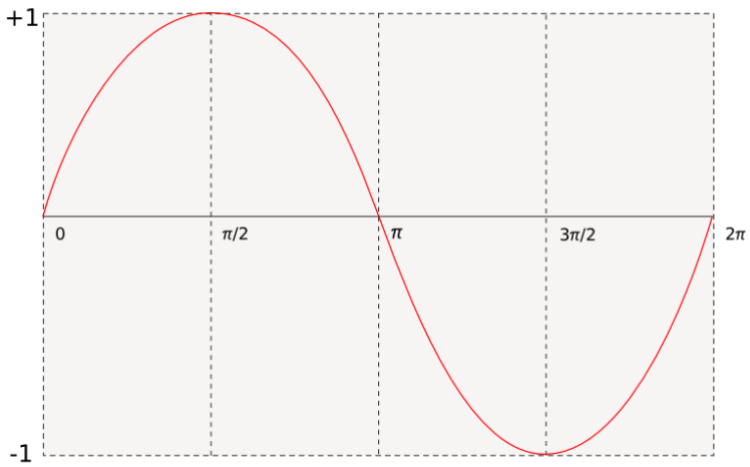

Abbildung 16 Eine Periode einer Sinuswelle mit Phase 2π (Radiant)

Beispielsweise mit einer Samplerate von 44100 Samples/Sekunde, und einer festgesetzten Periodendauer von 1 Sekunde, wird es 44100 Samples dauern, um von 0 zu 2π zu kommen. Somit können wir die Schritte pro Periode S aus der Periodendauer T bestimmen, wobei f_s wieder die Samplerate bezeichnet:

$$S = T * f_s$$

Daher wird jeder Schritt den folgenden Wert (in Radianten) annehmen, wobei f für die Frequenz steht:

$$\frac{2\pi f}{f_s}$$

Eine Sinuswelle kann also erzeugt werden, indem ein Phasenwert wiederholt um einen bestimmten Betrag erhöht wird, um letztendlich auf den gewünschten Wert von 2π innerhalb einer Periodendauer, unter Einflussnahme der Samplerate, zu kommen. Aus dieser Funktion entsteht dann ein Ausgangssignal, das zwischen der festgelegten Amplitude liegt.

All dies kann relativ einfach, mithilfe simpler mathematischer Funktionen, in einem Algorithmus umgesetzt werden, der eine Sinuswelle mit wählbarer Frequenz und Amplitude erzeugt:

Listing 1. Naiver Sinus Oszillator Algorithmus.

```
1.  Input: Peak amplitude (A), Frequency (f)
2.  Output: Amplitude value (y)
3.
4.  y = A * sin(phase)
5.
6.  phase = phase + ((2 * pi * f) / samplerate)
7.
8.  if phase > (2 * pi) then
9.         phase = phase - (2 * pi)
```

Mit der Variable A wird hierbei die Amplitude festgelegt, und mit der Variable f die Frequenz. Vor allem wichtig sind die letzten beiden Zeilen des Algorithmus. Hier wird überprüft ob der Phasenwert 2π übersteigt. Wenn dies der Fall ist, und somit eine ganze Periode zu Ende ist, wird wieder eine ganze Periode (2π) vom Phasenwert abgezogen, damit die Funktion wieder auf die korrekte Position gesetzt (anstatt einfach auf 0 zu reseten) wird, um einen korrekten Übergang in die nächste Periode zu gewährleisten. Wenn nämlich ein Phaseninkrement 2π übersteigt, und dann die Phase einfach wieder auf 0 resetet wird, entstehen unerwünschte Diskontinuitäten, die eine harmonische Verzerrung der Welle zur Folge haben. (Karras, 2018)

Dieser Algorithmus bietet auch die Basis für weitere Algorithmen zur Erzeugung verschiedenster Wellenformen, wie auch die Sägezahn und Rechteck Wellenformen, die für die TB-303 Emulation benötigt werden.

3.1.1.2 Naiver digitaler Rechteck Oszillator Algorithmus

Im Gegensatz zur Sinuswelle, haben Rechteckwellen viele harmonische Obertöne über ihrer Grundfrequenz, und bekommen dadurch einen viel helleren und schärferen Klang.

„After examining a number of different waveforms it will start to become apparent that waveforms with steep edges and/or abrupt changes and discontinuities are usually harmonically rich.„ (Karras, 2018)

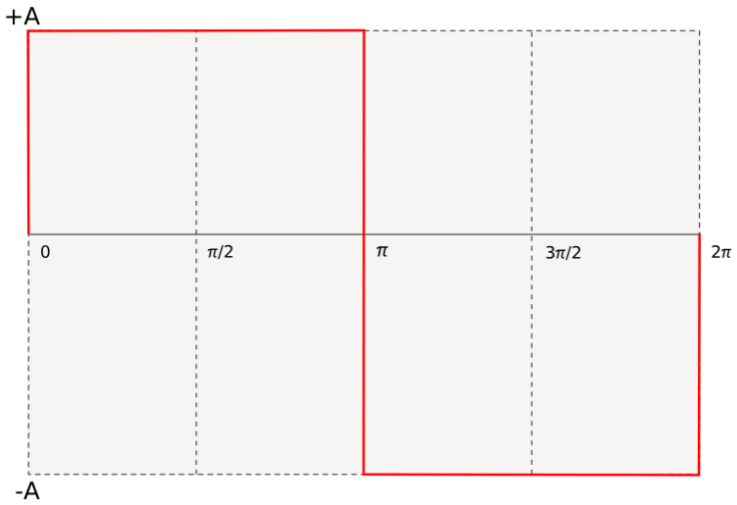

Abbildung 17 Eine Periode einer Rechteckwelle mit Phase 2π (Radiant)

Die Rechteckwelle wird in einer ähnlichen Weiße wie die Sinuswelle erzeugt. Es wird derselbe Ansatz, eine Phase Variable in einem bestimmten Muster zu erhöhen und sie zu reseten sobald 2π überschritten wird, verfolgt.

Listing 2. Naiver Rechteck Oszillator Algorithmus.

```
1.  Input: Peak amplitude (A), Frequency (f)
2.  Output: Amplitude value (y)
3.
4.  if phase < pi then
5.        y = A
6.  else
7.        y = -A
8.
9.  phase = phase + ((2 * pi * f) / samplerate)
10.
11. if phase > (2 * pi) then
12.       phase = phase - (2 * pi)
```

Die Rechteckwelle kann durch einfach Arithmetik definiert werden, da sie im Grunde nur zwischen zwei Werten hin- und herwechselt. Der Algorithmus überprüft lediglich ob der derzeitige Phasenwert einer Sinusschwingung über oder unter π liegt. Wenn die Phase unter π liegt, und wir uns somit in der ersten Hälfte der Periode befinden, wird die positive Spitzenamplitude (A) ausgegeben. Die negative Spitzenamplitude wird sobald der Phasenwert π, und somit die zweite Hälfte der Periode erreicht wird ausgegeben. Sobald die Phase 2π erreicht, werden wieder 2π von der Phase abgezogen, um das ganze wieder von vorne beginnen zu lassen.

Durch eine einfache Modifikation kann dieser Algorithmus auch eine Pulswelle (mit veränderbarer Pulsweite) erzeugen. Dafür muss nur die Stelle an dem von positiver auf negative Amplitude gewechselt wird (in diesem Fall π) durch eine Variable veränderbar sein. (Karras, 2018)

3.1.1.3 Naiver digitaler Sägezahn Oszillator Algorithmus

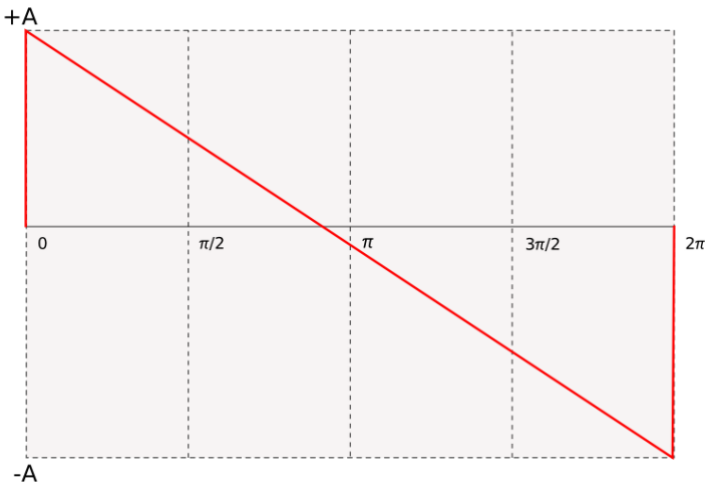

Abbildung 18 Eine Periode einer Sägezahnwelle mit Phase 2π (Radiant)

Die Sägezahnwelle hat noch mehr harmonische Obertöne wie die Rechteckwelle. Da sie aus diagonal abfallenden Liniensegmenten besteht, wird eine Liniengradientengleichung für den Algorithmus benötigt. Die mathematische Gleichung dafür lautet:

$$y = A - \frac{A}{\pi}\,\phi$$

Hierbei repräsentiert A die Amplitude und ϕ die Phase. Die Gleichung kann folgendermaßen in den Algorithmus implementiert werden:

Listing 3. Naiver Sägezahn Oszillator Algorithmus.

```
1.  Input: Peak amplitude (A), Frequency (f)
2.  Output: Amplitude value (y)
3.
4.  y = A - (A / pi * phase)
5.
6.  phase = phase + ((2 * pi * f) / samplerate)
7.
8.  if phase > (2 * pi) then
9.        phase = phase - (2 * pi)
```

Somit wird die Amplitude (A) mit Anstieg der Phase immer kleiner. Sobald 2π erreicht sind wird die Phase wieder resetet. Dadurch steigt die Amplitude am Beginn einer neuen Periode wieder auf ihren Spitzenwert.

3.1.1.4 Naiver Dreieck Oszillator Algorithmus

Die Dreieck Wellenform wird zwar für die Umsetzung der TB-303 Emulation nicht benötigt, der Vollständigkeit halber lohnt es sich aber trotzdem sie zu erwähnen.

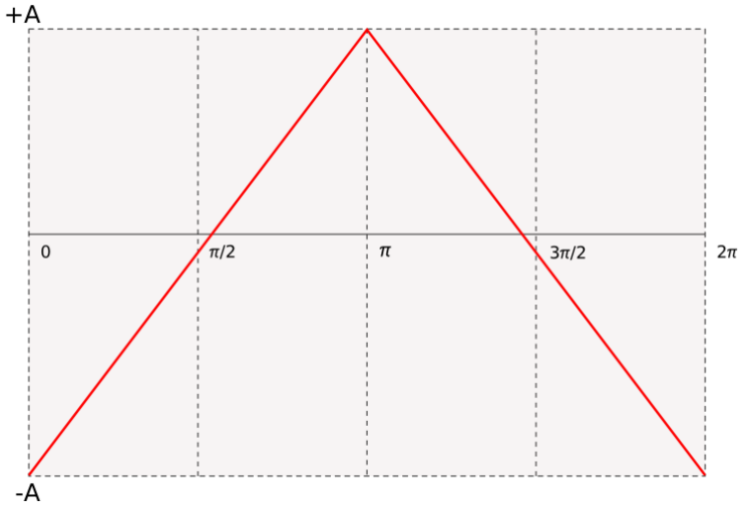

Abbildung 19 Eine Periode einer Dreieckswelle mit Phase 2π (Radiant)

Die Dreieckswelle hat geometrisch gesehen hohe Ähnlichkeit zur Sägezahnwelle, bis auf das sie zwei auf- bzw. absteigende Liniensegmente hat. Dreieckswellen enthalten nur Obertöne an den ungeraden ganzzahligen Vielfachen der Grundfrequenz, und klingen (ähnlich der Sinuswelle) deutlich weicher im Gegensatz zur Sägezahn- oder Rechteckwelle. Die mathematische Gleichung für die beiden Liniensegmente lautet:

Phase = 0 bis π:

$$y = A - \frac{2A}{\pi}\,\phi$$

Phase = π bis 2π:

$$y = 3A - \frac{2A}{\pi}\,\phi$$

Der Algorithmus ähnelt den vorher vorgestellten Algorithmen für die anderen Wellenformen, nur dass die obenstehenden Gradientengleichungen integriert werden:

Listing 4. Naiver Dreieck Oszillator Algorithmus.

```
1.  Input: Peak amplitude (A), Frequency (f)
2.  Output: Amplitude value (y)
3.
4.  if phase < pi then
5.        y = -A + (2 * A / pi) * phase
6.  else
7.        y = 3A - (2 * A / pi) * phase
8.
9.  phase = phase + ((2 * pi * f) / samplerate)
10.
11. if phase > (2 * pi) then
12.       phase = phase - (2 * pi)
```

Somit überprüft der Algorithmus im ersten Schritt ob der Phasenwert unter oder über π ist, um die jeweils richtige der beiden Gradientengleichungen auszuführen. Wenn der Phasenwert 2π erreicht wird, wird wie immer wieder der Phasenwert auf 0 resetet.

Bei den soeben beschriebenen Algorithmen zur Generierung von Rechteck-, Sägezahn-, und Dreieckswellen ist aber zu beachten, dass es sich hierbei um „naive“ Funktionen handelt. Sie entsprechen also dem Abtasten von idealen mathematischen Funktionen, ohne sie in ihrer Bandbreite zu begrenzen. Das Problem mit diesen Algorithmen ist, dass die Obertöne, die über der Nyquist Frequenz liegen, als Aliasing im hörbaren Bereich zum Signal hinzuaddiert werden. Diese Aliasing Obertöne bewegen sich im Frequenzspektrum auf- und ab, und erzeugen dadurch ein dem Sendersuchlauf eines Radios ähnliches Geräusch im Hintergrund. (Karras, 2018)

Im nächsten Abschnitt wird dieses Problem näher veranschaulicht.

3.1.2 Analog vs. Digital und das Aliasing Problem

Analoge Systeme geben ein zeitkontinuierliches Signal aus, wohingegen digitale Systeme in einen diskreten Zeitsystem arbeiten, und daher ein periodisches Spektrum innerhalb der Samplerate f_s haben. Wenn also z.B. eine analoge zeitkontinuierliche Sägezahnwelle abgetastet wird (wie es bei den naiven Oszillator Algorithmen der Fall ist) entstehen weitere ganzzahlige Vielfache von f_s und überlagern das Signal. Eine Komponente an der Grundfrequenz (f_0) im zeitkontinuierlichen Signal führt zu zusätzlichen Obertönen bei $Nf_s \pm f_0$ (wobei N immer ganzzahlig ist). Wenn diese Artefakte in den hörbaren Bereich fallen, tritt wie bereits erwähnt Aliasing auf. Die Aliasing Artefakte können im

Allgemeinen nicht entfernt werden, und werden als unerwünschte Töne wahrgenommen.

In Abbildung 20 ist eine zeitkontinuierliche Sägezahnwelle und ihre zeitdiskrete Repräsentation mit dem entstandenen Aliasing Artefakten zu sehen. (Frei, 2019)

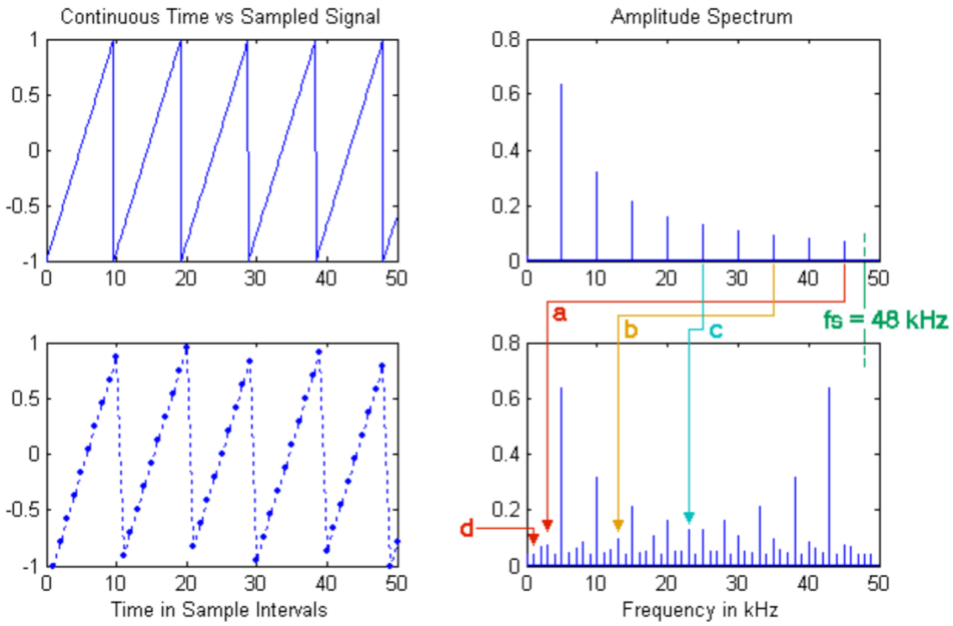

Abbildung 20 Zeitkontinuierliche Sägezahnwelle (f_0 = 5kHz), mit f_s = 48kHz abgetastet.

In der Praxis hat aber auch die Psychoakustik einen Einfluss auf dieses Phänomen. Manche durch Aliasing entstandene Frequenzteile sind durch ihre schwache Amplitude und auditiven Maskierungseffekten für uns Menschen nicht hörbar. Leisere Töne bzw. Frequenzen, die nah über einer sehr lauten Frequenz liegen, werden unter bestimmten Umständen maskiert. Bei Tönen, die unter dem lauten Ton liegen nimmt dieser Effekt jedoch schneller ab, und wird in Intervallen von mehr als einer Oktav vernachlässigbar. Abbildung 21 zeigt wie sich die Hörschwelle des menschlichen Gehörs für die Erkennung leiserer Töne, um ein lautes schmalbandiges 1 kHz Signal herum verhält. (Zwicker & Feldtkeller, 1967)

Eine grobe Analyse der naiven digitalen Sägezahnwelle aus Psychoakustischer Sicht ergibt (Abbildung 20):

a) Durch Aliasing entstandene 9. Harmonische: $\mathrm{f_s} - 9f_0 = 3\ kHz$. Ist hörbar.
b) Durch Aliasing entstandene 7. Harmonische: $\mathrm{f_s} - 7f_0 = 13\ kHz$. Teilweise maskiert durch 2. Harmonische. Außerdem wird die Tonhöhen-

Wahrnehmung des menschlichen Gehörs bei hohen Frequenzen unpräzise. Somit wird diese Komponente möglicherweise vom Gehör lediglich als Teil der generellen Höhen des Signals interpretiert. Dies kann aber von Ohr zu Ohr unterschiedlich sein.

c) Durch Aliasing entstandene 5. Harmonische: : $\mathrm{f_s} - 5f_0 = 23\ kHz$. Nicht hörbar, da die Frequenz außerhalb des hörbaren Bereichs liegt.
d) Durch Aliasing entstandene 19. Harmonische: : $2\mathrm{f_s} - 19f_0 = 1\ kHz$. Ist hörbar. Die Frequenz müsste, laut Abbildung 21, mindestens 80 dB (relativ zum Originalsignal) leiser sein, um unhörbar zu werden.

Daher lässt sich sagen, dass Spektralkomponenten eines zeitkontinuierlichen anlogen Signals, die nahe an ganzzahligen Vielfachen der Samplerate $\mathrm{f_s}$ liegen, während der Abtastung zu unerwünschten hörbaren Aliasing im hörbaren Frequenzbereich führen. Der Effekt ist am deutlichsten bei Aliasing Komponenten, die unter der Grundfrequenz f_0 liegen. (Frei, 2019)

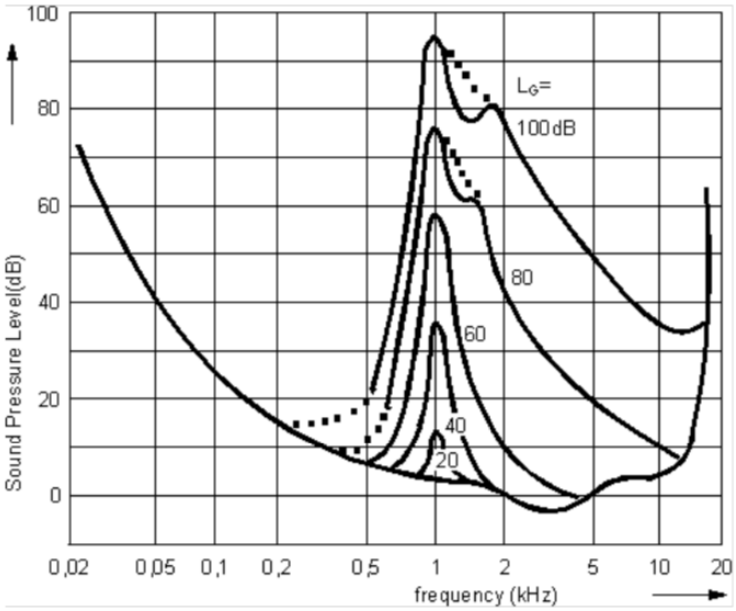

Abbildung 21 Hörschwelle für leisere Töne um ein lauteres schmalbandiges Signal bei 1 kHz.

Aus praktischer Sicht lässt sich sagen das ein Oszillator eines Synthesizers in der Lage sein muss, Töne mit einer Grundfrequenz von bis zu 4-5 kHz in exzellenter Qualität erzeugen zu können. Höherfrequente Signale werden meist nur als weniger bedeutsame Modulatoren oder als eine Quelle für weitere Obertöne genutzt. Daher können laut Frei (2019) folgende Design Regeln für die Abbildung eines zeitkontinuierliches Prototypsignals festgelegt werden:

- Das Frequenzspektrum des generierten Signals soll 20 kHz nicht überschreiten.
- Der Roll-Off nach 20 kHz soll so steil wie möglich unter den gegebenen Bedingungen erfolgen:
  - Bei $f_s - 2\,kHz$, unter -80 dB
  - Bei $f_s - 1\,kHz$, unter -85 bis -90 dB
  - Nach $f_s - 1\,kHz$ bis unendlich soll der Roll-Off mit mindestens $-20\ \mathrm{dB/Dekade}$ (~ 6 dB/Oktave) erfolgen.

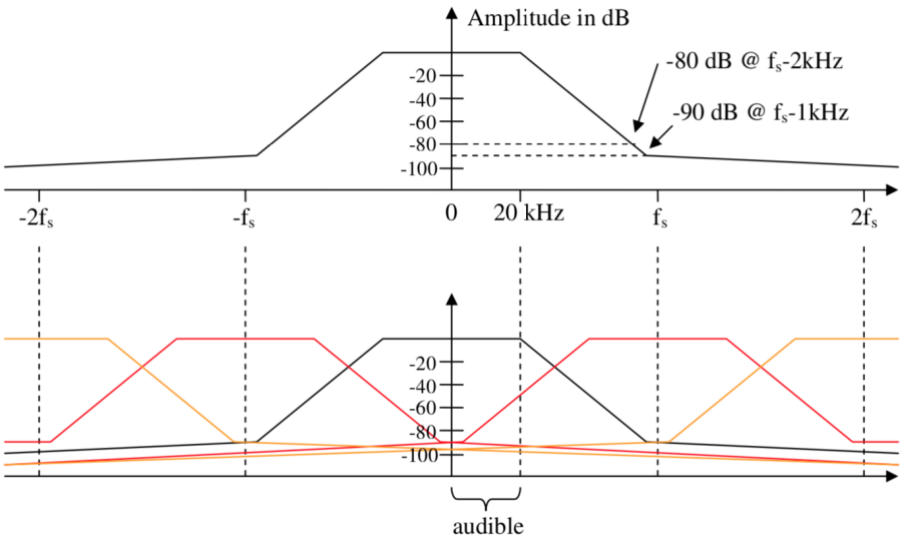

Abbildung 22 Spektrum eines zeitkontinuierlichen Prototypsignal und seine zeitdiskrete Repräsentation mit Aliasing Komponenten 1. und 2. Ordnung (Rot, Orange).

Somit ist geklärt wie sich die Effekte von Aliasing im Frequenzbereich bemerkbar machen, aber auch im Zeitbereich kann unerwünschtes Aliasing erkannt werden. Im Zeitbereich können Aliasing Komponenten bei den Diskontinuitäten an den Ecken der synthetisierten Wellenform erkannt werden. Daher treten bei den naiven Oszillator Algorithmen in den Momenten Probleme auf, in denen sich die Amplitude schnell verändert. Je höher die Grundfrequenz bzw. die Tonhöhe des Oszillators ist, desto schneller wechselt der Amplitudenwert, wodurch bei höheren Tonhöhen mehr Diskontinuitäten an den Ecken der Wellenform feststellbar sind. Abbildung 23 zeigt die Repräsentation einer naiven, nicht bandbegrenzten Rechteckwellenform im Zeitbereich, dabei sind die Probleme bei den Diskontinuitäten an den Ecken zu beachten die eben durch Aliasing entstehen. (Kane, 2019)

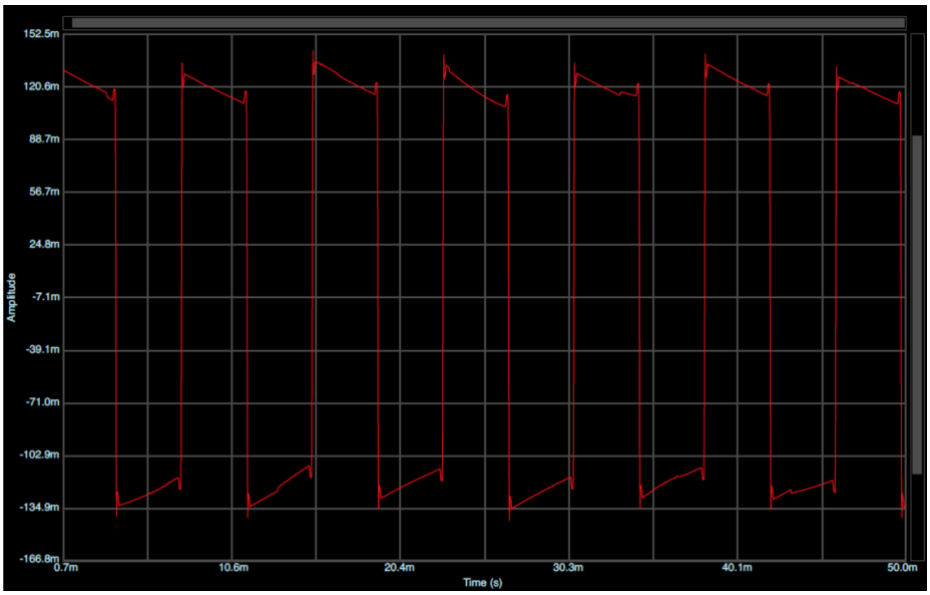

Abbildung 23 Darstellung einer naiven, nicht bandbegrenzten Rechteckwelle im Zeitbereich.

Zum Vergleich ist In Abbildung 24 eine bandbegrenzte Variante der Rechteckwelle zu sehen. Hier haben sich die Probleme bei den Diskontinuitäten an den Ecken der Wellenform deutlich verringert.

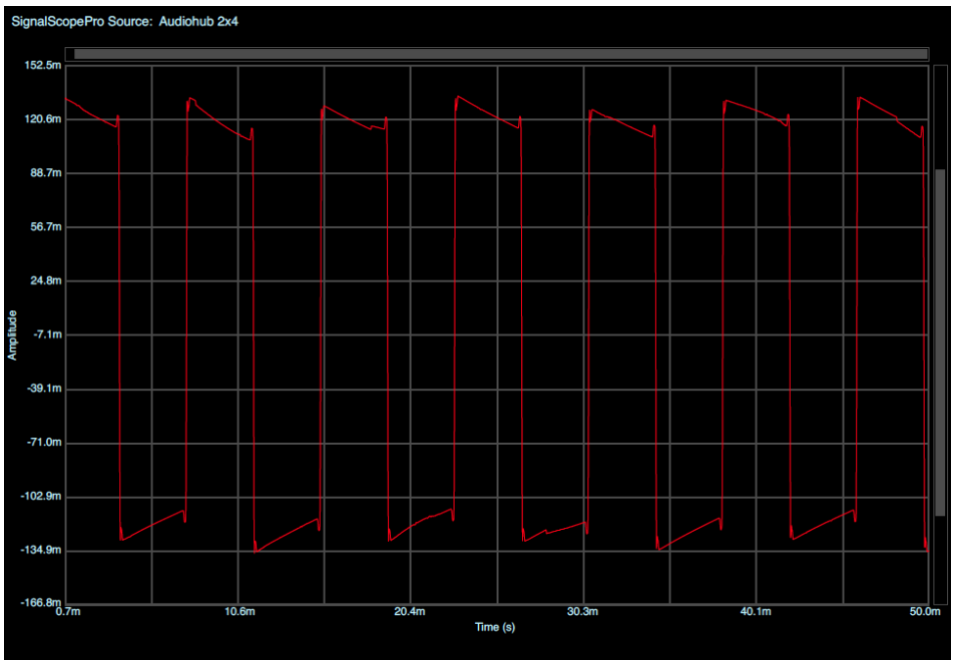

Abbildung 24 Darstellung einer bandbegrenzten Variante der Rechteckwelle im Zeitbereich.

Die naiven Sägezahn und Rechteck Oszillator Algorithmen haben also ein großes Problem mit Aliasing. Wichtig zu erwähnen ist jedoch, dass der naive Dreieckswellen Algorithmus kaum Probleme bei den Diskontinuitäten in der erzeugten Wellenform aufweist und somit nur sehr wenig Aliasing erzeugt. Daher kann die Dreieckswelle, im Gegensatz zu den anderen Wellenformen, recht unproblematisch mithilfe des naiven Algorithmus implementiert werden.

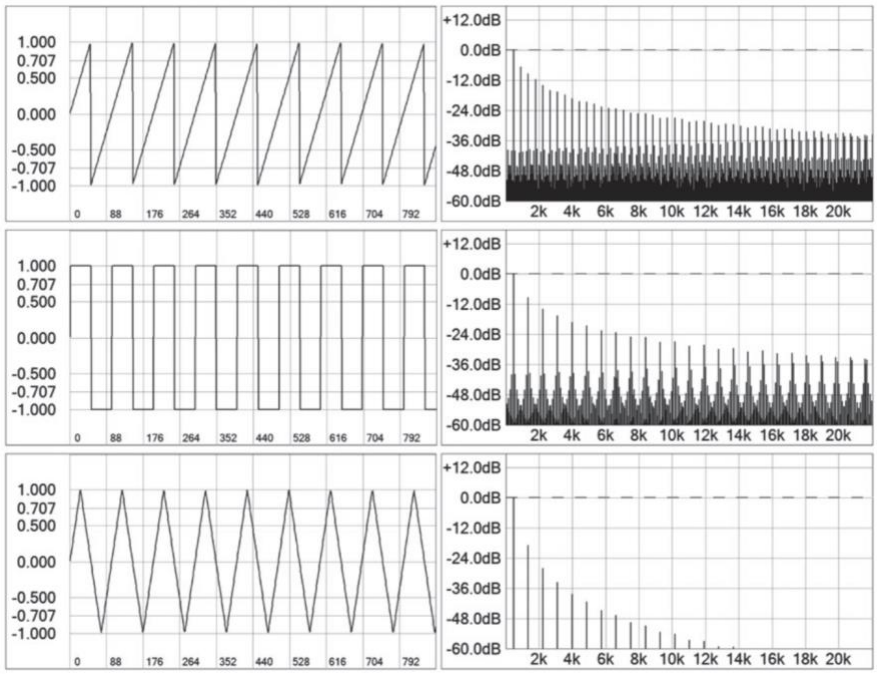

Abbildung 25 Wellenformen und Frequenzspektren von naiven Sägezahn (oben), Rechteck (mitte) und Dreieck (unten) Implementationen. f = 440 Hz

Abbildung 25 zeigt die Wellenformen und Frequenzspektren der drei naiven Oszillator Algorithmen, die mit einer Grundfrequenz f = 440 Hz schwingen. Das Sägezahnwellen Frequenzspektrum sollte eigentlich nur Spitzen an den ganzzahligen Vielfachen der Grundfrequenz (also 440 Hz, 880 Hz, 1320 Hz, usw.), und das Rechteckwellen Frequenzspektrum Spitzen an den ungeraden ganzzahligen Vielfachen (440 Hz, 1320 Hz, 2200 Hz, usw.) zeigen. Alle anderen im Signal enthaltenen Frequenzen sind Aliasing Komponenten. Bei der naiven Dreieckswelle hingegen entstehen, wie im Frequenzspektrum im unteren Teil von Abbildung 25 zu sehen ist, kaum unerwünschte Aliasing Komponenten. Auch die Darstellung der Wellenform im Zeitbereich zeigt kaum Probleme bei den Diskontinuitäten, wie es bei den anderen beiden Wellenformen der Fall ist. (Pirkle, 2014)

Es gibt einige verschiedene Lösungsansätze für diese Aliasing Probleme. Da die naiven Oszillator Algorithmen eben diese Probleme aufweisen ist die

naheliegenste Herangehensweise natürlich ganz andere Techniken für die Generierung von Oszillatoren anzuwenden, um somit die Probleme gleich von vornherein zu umgehen. Dafür gäbe es viele Möglichkeiten: z.B. könnte eine frequenzbegrenzte Fourierreihendarstellung dafür genutzt werden ein Aliasing freies Signal zu erzeugen. Auch additive Synthese Techniken, bei denen die gewünschte Wellenform mithilfe von einer Vielzahl an Sinuswellen mit den jeweiligen passenden Frequenzen und Amplituden erzeugt wird, können ein ähnliches Ergebnis erzeugen. Diese beiden Varianten sind aber sehr rechenintensiv. Obwohl die Berechnung von Trigonometrischen Funktionen wie sin(), cos() oder tan() heutigen Computern keinerlei Probleme bereiten muss bedacht werden, dass eine tiefe Note wie z.B. A0 (die tiefste Note eines Klaviers), eine Grundfrequenz von 27,5 Hz, sowie 800 Obertöne bis zur Nyquist Frequenz hat. Somit müssten bei der Verwendung von additiven Synthese Techniken 801 sin() Funktionen ausgeführt werden, was eben sehr rechenintensiv ist.

Um dies zu umgehen können die mathematischen Gleichungen für die Wellenformen auch vorberechnet und gespeichert werden, um dann mithilfe eines Look-Up Table Algorithmus die einzelnen Werte auszulesen und somit die Wellenform zu erzeugen. Diese Technik wird auch Wavetable Oszillator genannt. Es ist bei dieser Technik aber wiederum sehr schwierig den Wavetable Oszillator bandbegrenzt und auch frei von einigen anderen Verzerrungsarten (z.B. Quantisierungs- und Interpolationsverzerrung) zu implementieren. (Pirkle, 2014)

Welcher Lösungsansatz macht also nun den meisten Sinn? Die weitverbreitetste Technik ist, im Gegensatz zu den bereits genannten Herangehensweisen, ein Kompromiss. Es wird bei diesen Techniken bewusst ein wenig Aliasing zugelassen, solange die Aliasing Komponenten weit von der Grundfrequenz entfernt sind, und eine niedrige Amplitude haben. Wenn wir also z.B. einen bandbegrenzten Oszillator mit einer Grundfrequenz von 2 KHz schwingen lassen, werden Aliasing Komponenten mit einer Amplitude um -60 dB herum bei etwa 10 kHz im Frequenzspektrum auftauchen. Wenn man in Betracht zieht, dass das menschliche Gehör einen Dynamikbereich von 130 dB aufweist, kann man sagen das Aliasing Komponenten mit einer Amplitude von -60 dB schwer bis garnicht hörbar sind. Diese modernen Ansätze entfernen die entstehenden Aliasing Komponenten also nicht komplett, sie werden nur soweit unterdrückt, dass sie für uns nicht mehr hörbar sind. Deswegen werden diese Algorithmen auch quasi bandbegrenzte Oszillatoren genannt. (Kane, 2019)

3.1.3 Quasi bandbegrenzte Oszillator Algorithmen

Über die Jahre wurden einige moderne Oszillator Algorithmen vorgestellt die versuchen, die Aliasing Komponenten möglichst unter den hörbaren Bereich zu bringen. Die Algorithmen fokussieren sich großteils auf die bereits erwähnten Diskontinuitäten an den Ecken der synthetisierten Wellenformen. Zu den gängigsten bandbegrenzten (Virtual Analog) Oszillator Algorithmen zählen:

- Band-limited Impulse Trains (BLIT) (Stilson & Smith, 1996)
- Band-limited Step Functions (BLEP) (Brandt, 2001/Leary & Bright, 2009)
- MiniBLEP (Minimum Phase Lowpass Filter vor der BLEP Implementierung) (Brandt, 2001)
- Differentiated Parabolic Waveform Oscillators (DPW) (Välimäki, 2005)
- Polynomial Band-limited Step Functions (Poly BLEP) (Välimäki, Pekonen, Nam, 2010)

Die meisten aktuellen Software Synthesizer (abgesehen natürlich von Wavetable Synthesizern) verwenden für ihre Klangerzeugung einen der aufgezählten Algorithmen. Es wird z.B. auch angenommen das die standardmäßig verfügbaren Oszillatoren in Max/Msp entweder einen BLIT oder BLEP Algorithmus nutzen. (Kane, 2019)

In den folgenden Unterpunkten werden die BLIT, BLEP und polyBLEP Oszillator Algorithmen beschrieben.

3.1.3.1 Band-limited Impulse Trains (BLIT)

Der BLIT Algorithmus war der erste Versuch eines Quasi bandbegrenzten Oszillators von Stilson & Smith, Ende der 90er Jahre. Die Idee dahinter ist es einen Impuls mit einen Lowpass Filter zu filtern, um dadurch eine abgerundete Diskontinuität zu erzeugen. Aus diesen gefilterten Impulsen wird ein Stream (oder die „Impulse Train“) gebildet, wodurch im Endeffekt ein Stream aus sinc() Funktionen entsteht. Die Impulse Response eines idealen Lowpass Filter (oder auch die sinc() Funktion), wird genutzt da sie von Natur aus bandbegrenzt ist.

Die mathematische Definition der sinc() Funktion lautet (M = Anzahl der Obertöne):

$$sinc(x) = \frac{\sin(\pi x)}{Msin(\frac{\pi x}{M})}$$

Diese Train aus sinc() Funktionen wird dann mit sich selbst integriert, um Sägezahn- oder Rechteckwellen zu erzeugen. Dabei spielt die Polarität der

Impulse Train eine große Rolle (Abbildung 26 zeigt eine solche Impulse Train in unipolarer und bipolarer Form). Wird eine unipolare BLIT integriert, entsteht eine Sägezahnwelle mit abgerundeten Diskontinuitäten. Wird aber eine bipolare BLIT integriert, entsteht eine Rechteckwelle, und wenn sie doppelt integriert wird entsteht eine Dreieckswelle. (Stilson & Smith, 1999)

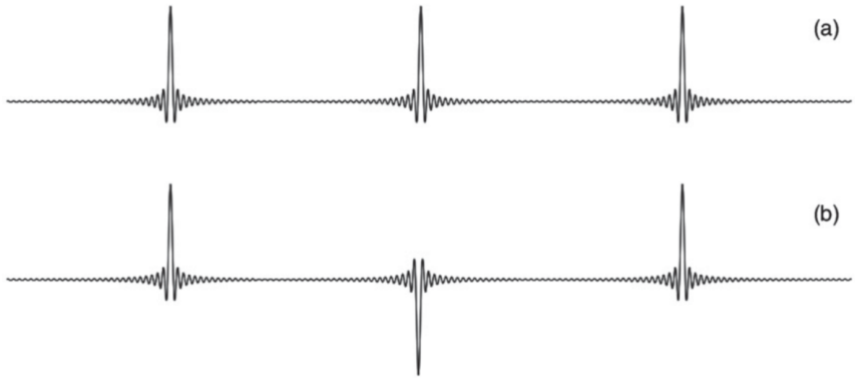

Abbildung 26 Bandbegrenzte Impulse Train, unipolar (a) und bipolar (b).

In Abbildung 27 sind Blockdiagramme für die Erzeugung von Sägezahn, Rechteck, und Dreieckswellen mit dem BLIT Algorithmus zu sehen.

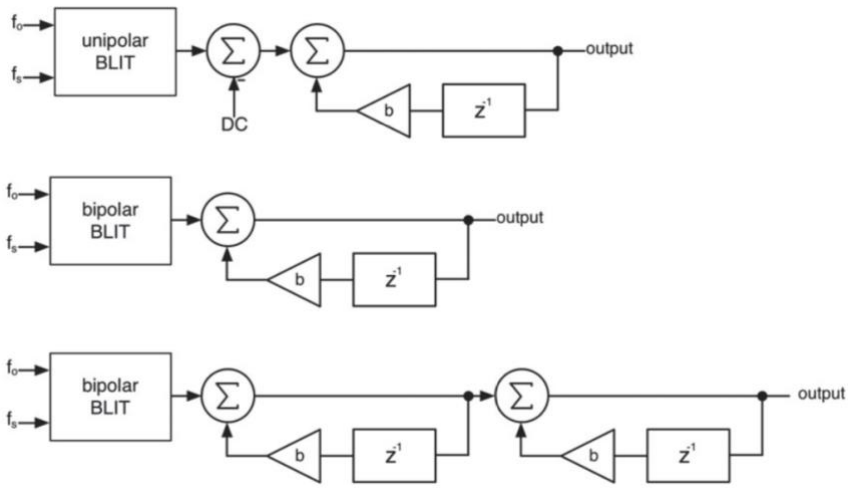

Abbildung 27 Blockdiagramme der BLIT Algorithmen für die Erzeugung von Sägezahn (oben), Rechteck (mitte) und Dreieckswellen (unten).

Der Output der BLIT Oszillatoren weist ein leichtes „Klingeln“ an den Bandkanten auf. Aliasing wird zwar reduziert, aber nicht vollständig entfernt. Auch die Berechnung von BLITs in Echtzeit kann sehr rechenintensiv sein, daher kann die Summe der sinc() Funktionen auch vorberechnet und in einer Tabelle gespeichert werden. Dies ist notwendig da die Periode der gewünschten

Wellenform möglicherweise kein ganzzahliges Vielfaches der Samplerate ist. Die Tabellen müssen dann während der Ausführung des Algorithmus interpoliert werden. Diese Version des Algorithmus wird auch BLIT-SWS (Sum of Windowed Sincs) genannt. (Pirkle, 2014)

Neben einigen weiteren Problemen mit dem Algorithmus, wurde von Andy Leary (2009) auch gezeigt, dass durch die Integrationschritte BLIT und BLIT-SWS ein ungewolltes DC-Offset erzeugen, wenn der Oszillator zu schwingen beginnt.

3.1.3.2 Band-limited Step Functions (BLEP)

Die Idee des BLIT Algorithmus wurde von Brandt weiterentwickelt. Seine Idee war es, eine einzelne sinc() Funktion zu integrieren und das Ergebnis in einer Tabelle zu speichern. Somit muss die Integration nicht in Echtzeit geschehen, und es entsteht eine bandbegrenzte Step Funktion, bzw. ein BLEP. In Abbildung 28 ist die BLEP Funktion dargestellt.

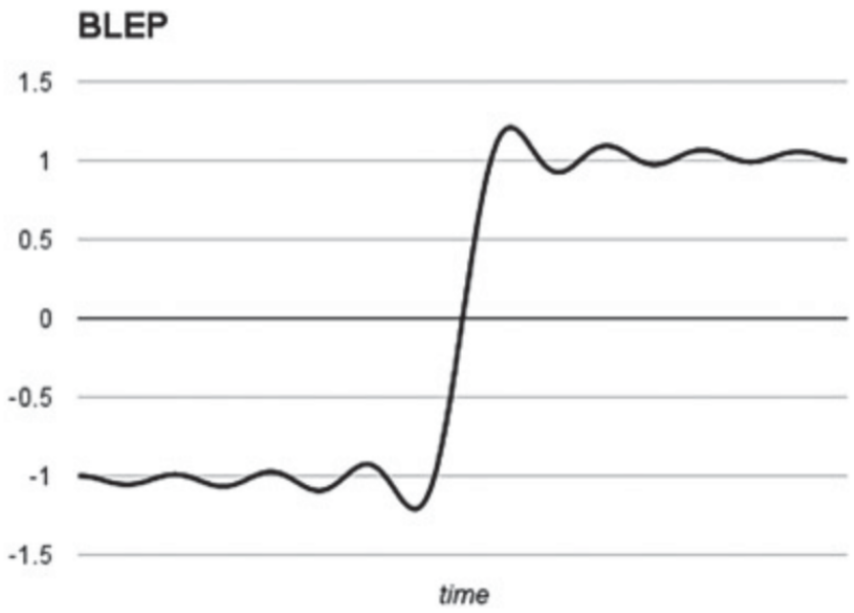

Abbildung 28 BLEP Funktion, die aus dem integrieren eines bandbegrenzten Impuls entsteht.

Die Idee hinter dem BLEP Oszillator Algorithmus ist es, zuerst eine naive Wellenform (die Diskontinuitäten enthält) zu erzeugen, und diese mit der BLEP Funktion zusammenzuführen, um die Diskontinuitäten zu glätten. Das bedeutet, dass eine bestimmte Anzahl an Samples während diesem Prozess modifiziert werden. Bei der Wahl der Anzahl der Samples, die modifiziert werden sollen, spielt die Leistung des Rechners eine Rolle (CPU, Speicher, usw.). In der Praxis

muss aber nur eine kleine Anzahl an Samples modifiziert werden, um bereits eine drastische Verringerung von Aliasing zu erreichen. (Brandt, 2001)

Andy Leary und Charlie Bright konnten 2009 ein US-Patent für ihre Arbeit „Bandlimited Digital Synthesis of Analog Waveforms“ erlangen. Das Patent verwendet BLEPs für die Reduzierung von Aliasing. Hierbei wird wieder von der sinc() Funktion ausgegangen, welche integriert wird und in seine bipolare Form konvertiert wird. Die naive Wellenform wird mit der BLEP Funktion durch einen Restwert zusammengeführt. Der Restwert entsteht durch das Subtrahieren des idealen Impulses von der vorher berechneten BLEP Funktion. Für eine aufsteigende Kante wird der Restwert addiert, für eine abfallende Kante wird er subtrahiert. (Leary & Bright, 2009)

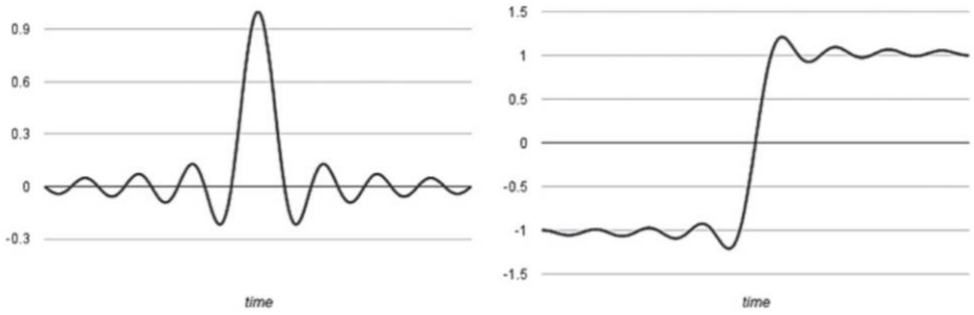

Abbildung 29 Die Impuls Response eines idealen Lowpass Filter (links) und das in bipolare Form konvertierte Step Signal das aus der Integration der Impulse Response entsteht (rechts).

Die BLEP Technik ist sehr flexibel. Z.B. können statt der sinc() Funktion jegliche Lowpass Filter Impulse Responses zu Beginn verwendet werden. Auch die vorher erwähnte Anzahl an Samples an denen die naive Waveform korrigiert wird ist variabel. All diese Parameter können zu verschiedenen Ergebnissen in Bezug auf Aliasing führen, und es gibt daher sehr viele verschiedene Möglichkeiten den BLEP Oszillator Algorithmus umzusetzen. (Pirkle, 2014)

3.1.3.3 Polynomial BLEP (PolyBLEP)

Välimäki hat 2010 eine Weiterentwicklung des BLEP Algorithmus vorgestellt. Den PolyBLEP Algorithmus. Seine Idee ist es, die vorher verwendete sinc() Funktion durch ein Polynom anzunähern. Verschiedenste Polynome können dafür verwendet werden, eine einfache Möglichkeit wäre es aber einen unipolaren Dreiecksimpuls zu wählen, da es sich hierbei um die lineare Annäherung eines windowed sinc() Impulses handelt. In Abbildung 30 ist der Dreiecksimpuls zu sehen (a), dieser wird integriert und erzeugt die Kurve (b). Da es sich um einen

geschlossenen Ausdruck handelt wird bei dieser Methode keine Tabelle für die Speicherung benötigt. (Välimäki et. al., 2010)

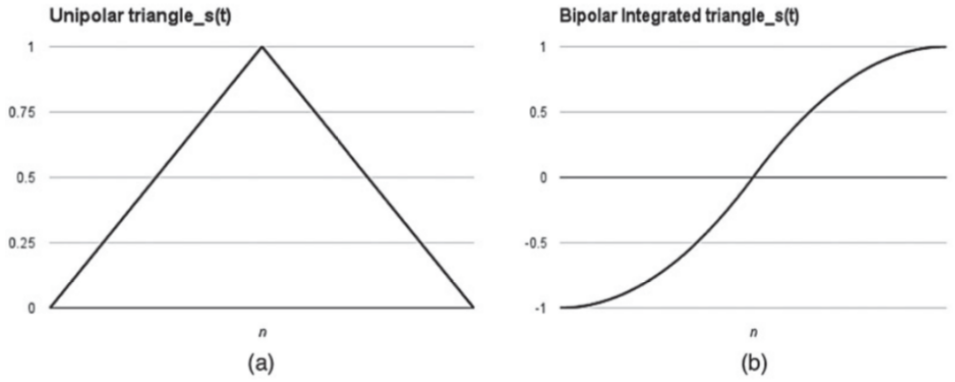

Abbildung 30 Ein unipolarer Dreiecksimpuls der statt sinc() verwendet wird (a), und das Ergebnis der Integration in bipolare Form konvertiert (b)

Der integrierte Dreiecksimpuls kann mathematisch mit folgender Gleichung ausgedrückt werden:

$$S_{tri(t)} = \begin{cases} 0 & t < -1 \\ \frac{t^2}{2} + t + \frac{1}{2} & -1 \leq t \leq 0 \\ t - \frac{t^2}{2} + \frac{1}{2} & 0 < t \leq \\ 1 & t > 1 \end{cases}$$

In der Gleichung steht t für die Distanz zur Diskontinuität (welche bei t = 0,0 zentriert ist). Der zweiteilige polyBLEP Restwert wir dann berechnet, indem der Step vom bipolaren integrierten Dreiecksimpuls subtrahiert wird, und erneut in bipolare Form umgewandelt wird. Wie auch im BLEP Algorithmus wird dann der Restwert mit der naiven Wellenform zusammengeführt, um einzelne Samples rund um die Diskontinuität rauf oder runter zu schieben, um dadurch eine geglättete Wellenform zu produzieren.

Somit werden zusammengefasst folgende Schritte ausgeführt, um die endgültige Wellenform zu erzeugen:

- Zunächst wird bestimmt an welchen Punkt die naive Wellenform sich im Übergangsbereich um die Diskontinuität befindet.
- Danach wird bestimmt ob der Punkt sich auf der linken oder rechten Seite der Diskontinuität befindet.
- Die Distanz des Punkts bis zur Diskontinuität wird gemessen.

- Der gemessene Wert wird verwendet, um den Restwert der Polynomgleichung zu berechnen.

(Pirkle, 2014)

Für die Software Emulation der TB-303 wurde letztendlich der polyBLEP Algorithmus für die Implementierung des Oszillators gewählt. Im Kapitel „TB-303 Emulations Projekt“ (Abschnitt „Oszillator“) wird die Implementierung des polyBLEP Algorithmus in gen~ beschrieben.

3.2 Filter

Neben dem Oszillator zählt der Filter zu den wichtigsten Bestandteilen eines Synthesizers. In diesem Kapitel wird zunächst die grundlegende Theorie hinter einem einfachen digitalen Lowpass Filters beschrieben. Außerdem wird geklärt welche Art von Filter (IIR oder FIR) sich am besten für die Entwicklung von Synthesizern eignet, und wie ein analoger Filter in die digitale Domäne transformiert werden kann.

Die einfachste Art einen Filter zu beschreiben ist die Impulse Response. Ein Input Signal x(n), dass durch das System h(n) geschickt wird ergibt das Output Signal y(n) (siehe Abbildung 31).

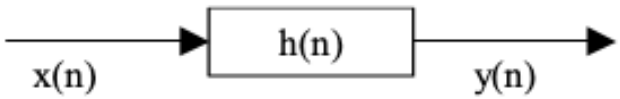

Abbildung 31 Simples Filter Block Diagramm.

Mathematisch wird dies wie folgt ausgedrückt:

$$y(n) = x(n) * h(n)$$

3.2.1 Simpler Lowpass Filter

Der am einfachsten zu beschreibende Filter ist ein IIR Filter. Dieser wird hier zunächst als linear-time-invariant (LTI) System dargestellt. Dies bedeutet das sich die Koeffizienten des Filters über die Zeit nicht verändern und die Auswirkungen des Filters auf das Signal somit konstant bleiben.

Die Transferfunktion eines solchen Filter Systems 1. Ordnung lässt sich wie folgt beschreiben:

$$H_1(z) = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}} = b_0 \frac{1 - z_1 z^{-1}}{1 - p_1 z^{-1}}$$

Wobei b und p für die Filter Koeffizienten, und p und z für die komplexen Pole und Zero Koeffizienten stehen.

Für ein Filter System 2. Ordnung wird die Gleichung wie folgt umgewandelt:

$$H_1(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} = b_0 \frac{(1 - z_1 z^{-1})(1 - z_2 z^{-1})}{(1 - p_1 z^{-1})(1 - p_2 z^{-1})}$$

Dieser Filter ist auch als Biquad Filter bekannt und kann als Baustein für die Entwicklung von Filtern höherer Ordnung genutzt werden. Abbildung 32 zeigt ein Block Diagramm des Biquad Filters. (Nielsen, 2000)

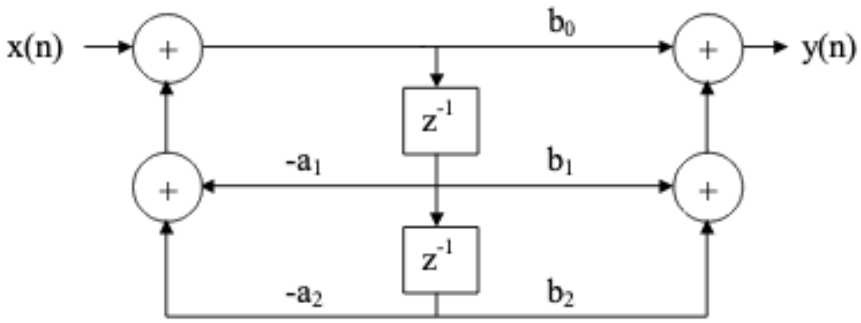

Abbildung 32 Block Diagramm eines Biquad Filters.

3.2.1.1 One Pole Filter

Die Transferfunktion eines simplen (und auch verstellbaren) One Pole Lowpass Filter lautet wie folgt:

$$H_1(z) = \frac{1 - a}{1 - a z^{-1}}$$

Da es sich um einen One Pole Filter handelt fällt die Amplitude nach der Cutoff Frequenz mit 6dB/Oktave ab. Dieser Filter kann mit folgender Gleichung implementiert werden:

$$y(n) = (1 - a)x(n) + a y(n - 1)$$

Zu beachten ist, dass der Filter bei a = 0 keinen Effekt hat. Daher sind die Koeffizienten bei einem System 1. Ordnung $b_0 = 1 - a$, $b_1 = 0$, und $a_1 = a$. Abbildung 33 zeigt das Block Diagramm des Filters. (Nielsen, 2000)

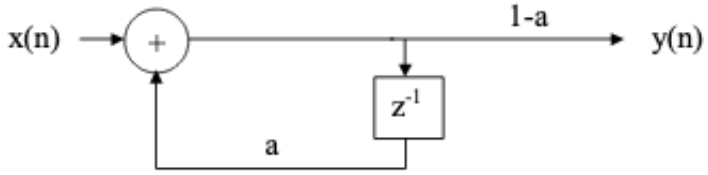

Abbildung 33 Block Diagramm eines One Pole Lowpass Filters.

Da der Diode Ladder Filter der TB-303 aus vier in Reihe geschaltener One Pole Lowpass Filtern besteht, bietet dies eine gute Grundlage dafür. Bei der Berechnung des Diode Ladder Filters wurde jedoch auf eine von Zavalishin (2015) beschriebene Methode mithilfe von sogenannten TPT-Modulen zurückgegriffen, wobei die einzelnen Lowpass Filter in der Ladder mit den TPT Gleichungen modelliert werden. Diese Methode wird im Kapitel „TB-303 Emulations Projekt“ ausführlich beschrieben.

3.2.2 FIR vs. IIR Filter

Bei den vorhin beschriebenen Filtern handelt es sich um Infinite Impulse Response (IIR) Filter. Dabei handelt es sich um Filter, die mit Feedback arbeiten. Finite Response Filter (FIR) arbeiten hingegen ohne Feedback. FIR Filter werden weitläufig für DSP Anwendungen eingesetzt, da sie einfach zu kontrollieren sind, und sie sich gut dafür eignen Filter höherer Ordnung zu bauen, die eine starke Dämpfung unerwünschter Frequenzen ermöglichen. FIR Filter benötigen aber meist mehr Rechenzeit als äquivalente IIR Filter, da sie mehrere Koeffizienten benötigen. Außerdem können FIR Filter in der Praxis nicht so gut interaktiv gestimmt werden wie es bei IIR Filtern der Fall ist. FIR Filter sind im Gegensatz zu IIR Filtern in der Lage eine lineare Phase Response zu erreichen. In einem musikalischen Kontext ist die Phase Response jedoch zu vernachlässigen, da das menschliche Ohr kaum in der Lage ist solche kurzen Verzögerungen im Signal überhaupt wahrzunehmen. Daher ist die Frequency Response weitaus wichtiger als die Phase Response bei einem Filter eines Synthesizers. Aus diesen Gründen gelten IIR Filter für den Einsatz in Synthesizern als praktikabler. (Nielsen, 2000)

3.2.3 Analog vs. Digital und die Bilineare Transformation

Es ist zwar möglich, wie auch vorher gezeigt wurde, Filter direkt in der digitalen z-Domäne zu entwerfen, jedoch gilt dieses Vorgehen nicht als die bevorzugte Methode um digitale Filter zu erstellen. Besonders die Reaktion von IIR Filter lässt sich in der digitalen Domäne schwer voraussagen, und ist auch schwer zu

kontrollieren. In der analogen Domäne ist dies aber nicht der Fall, deswegen gilt der Vorgang einen Filter zunächst in der analogen Domäne zu entwerfen und dann in die digitale Domäne zu transformieren als vorteilhaft. (Dodge & Jerse, 1997)

Eine Möglichkeit einen analogen Filter in die digitale Domäne zu transformieren bietet die Bilineare Transformation. Dabei wird die s-Domäne (in der wir uns bei analogen Filtern befinden) mit der digitalen z-Domäne in Relation gesetzt.

$$s = \frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right)$$

Wobei T für das Sampling Intervall steht. Ein großer Vorteil dieser Transformation ist, dass Aliasing vermieden wird. (Nielsen, 2000)

Die Frequenzrelation zwischen der s- und z-Domäne lautet wie folgt:

$$\omega = 2\arctan\left(\frac{\Omega T}{2}\right)$$

Jedoch muss erwähnt werden, dass die Bilineare Transformation lediglich eine Annäherung des analogen Filters bietet. Daher sind zwischen dem analogen und dem entsprechenden digitalen Filter leichte Unterschiede in der Frequenzgang Charakteristik festzustellen. (Dodge & Jerse, 1997)

3.3 Nichtlineare Funktionen

Analoge Filter Schaltungen verhalten sich im Gegensatz zu digitalen Filtern nicht linear. Um dieses nichtlineare Verhalten zu modellieren können verschiedenste mathematische Funktionen eingesetzt werden. Folgend werden zwei der meistgenutzten nichtlinearen Funktionen beschrieben: die Tangens Hyperbolicus (tanh) und Cubic Nonlinear Distortion Funktionen. Diese beiden Funktionen kommen auch bei der Umsetzung der TB-303 Emulation zum Einsatz.

3.3.1 Tanh

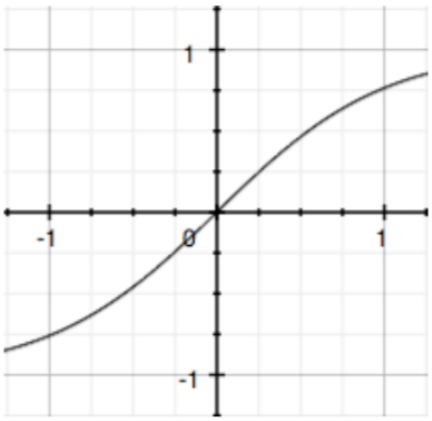

Abbildung 34 tanh Funktion.

Die tanh Funktion verwandelt ein Input Signal x = [-1..+1] in ein Output Signal y mit weniger als [-1..+1]. Abbildung 34 zeigt die tanh Funktion. In diesem Fall wird z.B. y = 0,8 wenn x = 1.

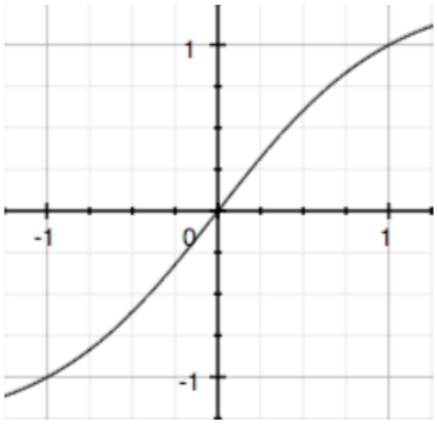

Abbildung 35 Normalisierte tanh Funktion.

Weiters kann die Funktion normalisiert werden (wie es in der Implementation der Emulation der Fall ist). Dadurch bleibt y = +/-1, wenn x = +/-1 ist. Abbildung 35 zeigt wie sich die Kurve dadurch sichtbar verändert. Die Formal hierfür lautet:

$$y = \frac{1}{\tanh(1)} \tanh(x)$$

Mit der normalisierten Version kann noch weiter experimentiert werden, indem eine Saturation Variable zur Gleichung hinzugefügt wird, mit der sich die Steilheit der Kurve steuern lässt: (Pirkle, 2013)

$$y = \frac{1}{\tanh\,(sat)} \tanh\,((sat)x)$$

Abbildung 36 zeigt die neue Funktion mit sat = 3.

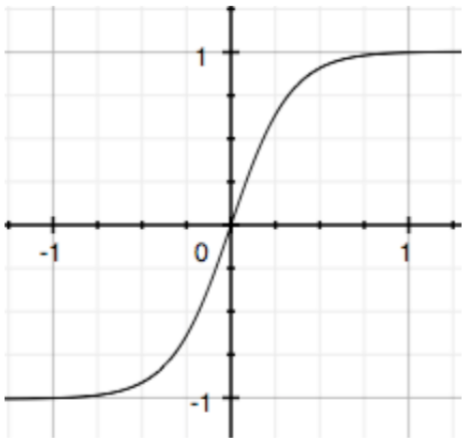

Abbildung 36 Normalisierte tanh Funktion mit sat = 3

3.3.2 Cubic Nonlinear Distortion

Eine weitere oft genutzte nichtlineare Funktion ist die Cubic Distortion. Die Formal hierfür lautet:

$$f(x) = \begin{cases} -\frac{2}{3} & x \leq -1 \\ x - \frac{x^3}{3} & -1 < x < 1 \\ \frac{2}{3} & x \geq 1 \end{cases}$$

Die daraus resultierende Funktion ist in Abbildung 37 zu sehen:

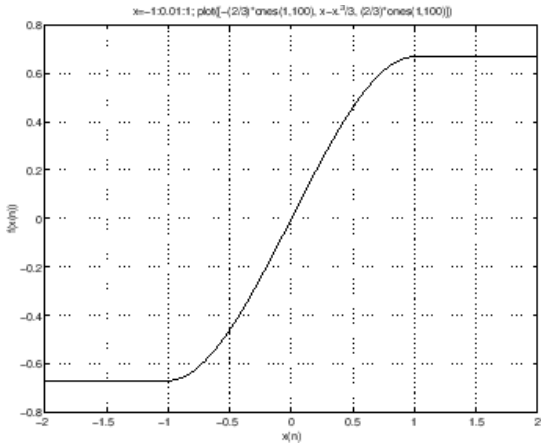

Abbildung 37 Cubic Nonlinear Distortion Funktion

Bei dieser Funktion kann durch den Input Gain die gewünschte Stärke der Distortion eingestellt werden. (Smith, 2013)

4 TB-303 Emulations Projekt

In diesem Kapitel wird die Max/Msp Emulation der TB-303, und wie diese umgesetzt wurde, beschrieben. Als erstes wird die Funktionsweise des Diode Ladder Filter mithilfe eines Block Diagramm und basierend auf Zavalishin's Ansatz der TPT Module synthetisiert und analysiert. Danach wird ein Prototyp des Filters in Python implementiert. Auf Basis des Python Code wird der Filter letztendlich in GenExpr portiert, um ihn in Max/Msp nutzen zu können. Der Sägezahn Oszillator wird ebenfalls in Gen implementiert. Die weiteren Bestandteile des Synthesizers (Midi, Rechteckwelle, Volume und Filter Envelope Modulation, Slide, Accent) werden dann in Max/Msp umgesetzt.

4.1 Diode Ladder Filter

Der Filter der TB-303 ist der wichtigste Bestandteil des Synthesizers, da dieser großteils für den speziellen Klang des Synthesizers verantwortlich ist. In diesem Abschnitt wird beschrieben wie der analoge Filter als Virtual Analog Algorithmus umgesetzt wurde.

4.1.1 Vom Blockdiagramm zum Filter

Beim Filter handelt es sich um einen Lowpass Filter 4. Ordnung. Dieser setzt sich, wie auch der Moog Ladder Filter, aus vier, in Reihe geschaltenen, synchron gestimmten, Lowpass Filtern 1. Ordnung (LPFs), die in einen globalen Feedback Loop eingebettet sind, zusammen. Der negative Feedback Loop hat eine Verstärkung k, die nur an der Cutoff-Frequenz eine positive Rückkopplung erzeugt, da ein Filter 4. Ordnung an der Cutoff-Frequenz eine Phasenverschiebung von 180 Grad erzeugt. Der Unterschied zum Moog Ladder Filter ist jedoch, dass sich beim Diode Ladder Filter mehrere Feedback Loops zwischen den einzelnen Filtern befinden (LPF2 speist in LPF1 zurück, LPF3 speist in LPF2 zurück, LPF4 speist in LPF3 zurück). Siehe Abbildung 38.

Im Filter sind mehrere Nichtlinearitäten enthalten, die typischerweise durch nonlinearity processing (NLP) Blöcke modelliert werden. Diese NLP Blöcke

können unter anderem aus der tanh Funktion bestehen. Es ist zunächst aber einfacher eine lineare Version des Filters aufzubauen, und danach Nichtlinearitäten hinzuzufügen. (Pirkle, 2013)

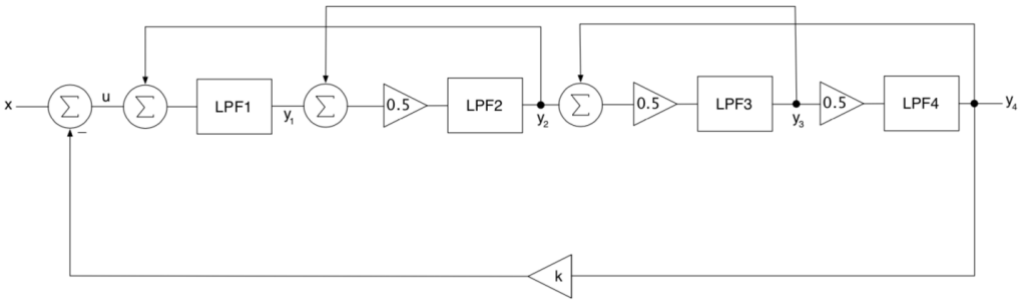

Abbildung 38 Blockdiagramm des linearen Diode Ladder Filter

4.1.1.1 Linear Model

In Abbildung 38 ist das Blockschaltbild der linearen Version zu sehen. Die nichtlineare Version könnte z.B. zusätzlich NLP-Blöcke im globalen Feedback Pfad, oder auch in den lokalen Feedback Pfaden eingebaut haben. (siehe Abschnitt nonlinear Model)

Zusätzlich zu den Feedback Pfaden wird das Signal an den Eingängen der letzten drei LPFs abgeschwächt, da der Filter sonst instabil werden würde.

Der Input des Filters ist mit x, und der Output mit y4 gekennzeichnet, während u den Input des ersten Summierer bezeichnet. Jeder LPF-Output ist mit y1 bis y4 gekennzeichnet.

Zunächst soll der Wert für u errechnet werden (Zavalishin, 2015).

$$u = x - ky_4$$

Also muss, um u zu lösen, die Beziehung zwischen y4 und x gefunden werden. Dies wird durch einen wiederholten Vorgang erreicht der in Zavalishin's „The Art of VA Filter Design" beschrieben wurde.

Zunächst wird die Diode Ladder isoliert, und unabhängig von ihrem globalen Feedback Loop betrachtet.

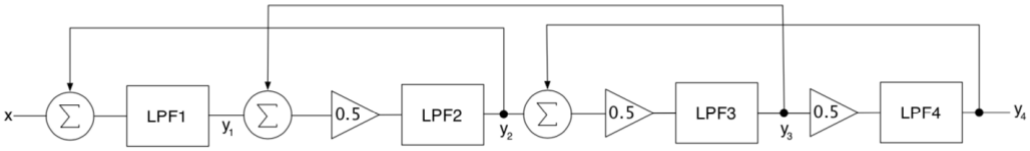

Abbildung 39 Isolierte Diode Ladder Struktur, ohne globalen Feedback Loop

Jeder einzelne LPF wird laut Zavalishin als ein TPT Modul 1. Ordnung mit seinem eigenen lokalen Werten für G und S dargestellt. Somit gilt für jeden Filter (Zavalishin, 2015):

$$y_n = Gx_N + S$$

$$G = \frac{g}{1+g}$$

$$S = \frac{S_N}{1+g}$$

Nun müssen diese Gleichungen auf die einzelnen LPFs angewendet werden, indem Eingänge, Ausgänge, Feedback und Dämpfung in Beziehung gesetzt werden. Dies führt zu neuen G- und S-Variablen, die sich auf die Feedback Pfade zwischen den einzelnen Filtern beziehen.

Danach muss noch eine Gleichung geformt werden, die den globalen Feedback Pfad miteinbezieht, um die gesamte Struktur (Abbildung 38) zu lösen.

Um die Gleichungen der einzelnen LPFs inklusive ihrer Input + Feedback Kombinationen zu formen wird von hinten, also mit LPF4, angefangen.

LPF4

Der letzte Filter hat eine Input Dämpfung a0 = 0,5 und bekommt kein Feedback eingespeist. Somit ist LPF4 am einfachsten zu berechnen.

Die Berechnung läuft wie folgt ab (Zavalishin, 2015):

$$y_4 = g(x - y_4) + s_4$$

$$= g(0{,}5y_3 - y_4) + s_4$$

$$y_4 + gy_4 = 0{,}5gy_3 + s_4$$

$$y_4(1+g) = 0{,}5gy_3 + s_4$$

$$y_4 = \frac{0{,}5gy_3 + s_4}{1+g}$$

Somit erhalten wir:

$$y_4 = G4y_3 + S4$$

$$G4 = \frac{0{,}5g}{1+g}$$

$$S4 = \frac{s_4}{1+g}$$

LPF3

Dieser Filter hat eine Input Dämpfung a0 = 0,5 und bekommt Feedback von LPF4 (y4) eingespeist. Daher ist der Input des Filters $0{,}5y_2$ + $0{,}5y_4$. Die Berechnung wird auf die gleiche Art wie bei LPF4 durchgeführt. Die Gleichung wird aber unabhängig vom y4 Feedback Input geformt (dies wird später hinzugefügt), und ist somit nur von y2 als Input abhängig.

$$y_3 = g(x - y_3) + s_3$$

$$= g(0{,}5y_2 \; + \; 0{,}5y_4 - y_3) + s_3$$

$$= g(0{,}5y_2 \; + \; 0{,}5(G4y_3 + S4) - y_3) + s_3$$

$$0{,}5gy_2 + 0{,}5gG4y_3 + 0{,}5gS4 - gy_3 + s_3$$

$$y_3 + gy_3 - 0{,}5gG4y_3 = 0{,}5gy_2 + 0{,}5gS4 + s_3$$

$$y_3(1 + g - 0{,}5gG4) = 0{,}5gy_2 + 0{,}5gS4 + s_3$$

$$y_3 = \frac{0{,}5gy_2 + 0{,}5gS4 + s_3}{1 + g - 0{,}5gG4}$$

Somit erhalten wir:

$$y_3 = G3y_2 + S3$$

$$G3 = \frac{0{,}5g}{1 + g - 0{,}5gG4}$$

$$S4 = \frac{0{,}5gS4 + s_3}{1 + g - 0{,}5gG4}$$

LPF2

Dieser Filter hat eine identische Topologie wie LPF3, somit ist die Berechnung fast identisch, mit Ausnahme der Input Variablen. Die Gleichung ist wie zuvor unabhängig von y3 (Feedback von LPF3).

$$y_2 = g(x - y_2) + s_2$$

$$= g(0{,}5y_1 + 0{,}5y_3 - y_2) + s_2$$

$$= g(0{,}5y_1 + 0{,}5(G3y_2 + S3) - y_2) + s_2$$

$$0{,}5gy_1 + 0{,}5gG3y_2 + 0{,}5gS3 - gy_2 + s_2$$

$$y_2 + gy_2 - 0{,}5gG3y_2 = 0{,}5gy_2 + 0{,}5gS3 + s_2$$

$$y_2(1 + g - 0{,}5gG3) = 0{,}5gy_1 + 0{,}5gS3 + s_2$$

$$y_2 = \frac{0{,}5gy_1 + 0{,}5gS3 + s_2}{1 + g - 0{,}5gG3}$$

Somit erhalten wir:

$$y_2 = G2y_1 + S2$$

$$G2 = \frac{0{,}5g}{1 + g - 0{,}5gG3}$$

$$S2 = \frac{0{,}5gS3 + s_2}{1 + g - 0{,}5gG3}$$

LPF1

Dieser Filter hat keine Input Dämpfung (daher a0 = 1), bekommt aber Feedback von y2 eingespeist. Die Berechnung ist wieder fast identisch mit den Berechnungen zuvor.

$$y_1 = g(x - y_1) + s_1$$

$$= g(\mathrm{x} + y_2 - y_1) + s_1$$

$$= g(\mathrm{x} + \mathrm{G2}y_1 + S2 - y_1) + s_2$$

$$= gx + gG2y_1 + gS2 - gy_1 + s_1$$

$$y_1 + gy_1 - gG2y_1 = gx + gS2 + s_1$$

$$y_1(1 + g - gG2) = gx + gS2 + s_1$$

$$y_1 = \frac{gx + gS2 + s_1}{1 + g - gG2}$$

Somit erhalten wir:

$$y_1 = G1x + S1$$

$$G1 = \frac{g}{1 + g - gG2}$$

$$S1 = \frac{gS2 + s_1}{1 + g - gG2}$$

Als nächstes schauen wir uns die einzelnen Filter genauer an. Es müssen die SN Feedback Variablen der einzelnen Filter Stufen erzeugt werden um sie in den vorherigen Filter zurückzuspeisen. Es wird wieder bei LPF4 begonnen.

LPF4

LPF4 ist soweit komplett da es bei dieser Filter Stufe keinen Feedback Input gibt. Es muss nur ein Feedback Output angedacht werden, um einen Feedback Pfad zum vorherigen Filter (LPF3) zu erzeugen.

LPF3,2,1

Die restlichen drei Filter haben alle einen Feedback Input der vom lokalen delay-less Feedback Pfad vom darauffolgenden Filter gespeist wird. Dieser Feedback Input wird mit dem „normalen" Input summiert. Abbildung 40 veranschaulicht dies in einem Block Diagramm anhand des LPF3 & LPF4 Filter Paar.

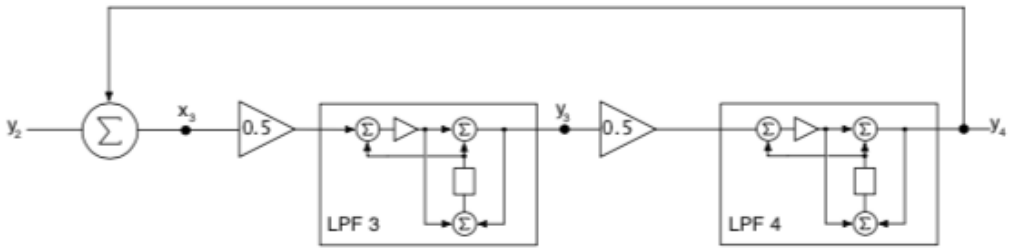

Abbildung 40 LPF3/LPF4 Filter Paar

Es gibt also einen weiteren delay-less Feedback Loop um LPF3 und LPF4. In der Gleichung von LPF3 wird dieser Feedback Loop auf die gleiche Weise wie die (lokale TPT-) delay-less Loop Gleichung beschrieben, wobei die Quelle des Feedbacks (in diesem Fall y4) berücksichtigt wird und die Gleichung nur aufgrund des non-Feedback Inputs geformt wurde. Somit ist der Großteil der Berechnungen bereits erledigt. Es muss aber noch herausgefunden werden wie sich die Input Variable nach dem Feedback Summierer berechnet (in diesem Fall x3, siehe Abbildung 40). Da bei 3 der 4 LPF Stufen eine 0,5 Dämpfung am Input eingesetzt wird, kann der Koeffizient so ins Block Diagramm eingesetzt werden.

Berechnung von x3:

$$x_3 = y_2 + y_4$$

$$= y_2 + G4y_4 + S4$$

$$= y_2 + G4(G3y_2 + S3) + S4$$

$$= y_2 + G3G4y_2 + G4S3) + S4$$

$$= y_2(1 + G3G4) + G4S3 + S4$$

Die G4 und S4 Variablen repräsentieren die Feedback Komponenten. Abbildung 41 zeigt diese Struktur isoliert als Block Diagramm.

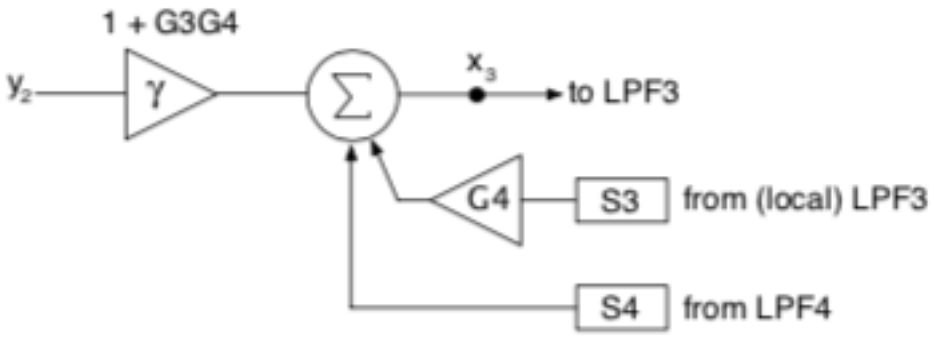

Abbildung 41 Input und Feedback Summierung

S4 ist das Feedback von LPF4. Der Wert G4 basiert auf LPF4, wird aber zuvor berechnet und muss sich nur ändern, wenn sich die Cutoff Frequenz ändert.

$$G4 = \frac{0{,}5g}{1 + g}$$

$$S4 = \frac{s_4}{1 + g}$$

Die S3 Variable kommt von LPF3 selbst, und beinhaltet daher das lokale Feedback von LPF3, aber auch S4 (Feedback von LPF4) ist enthalten.

$$S3 = \frac{0{,}5gS4 + s_3}{1 + g - 0{,}5gG4}$$

Die Struktur der einzelnen Diode Ladder Filter Stufen muss modifiziert werden um die Input-Dämpfung, Feedback-In und Feedback-Out Knoten, sowie die benötigten Koeffizienten zur Berechnung von S zu inkludieren. Der verallgemeinerte TPT Block (Abbildung 41) hat nun sowohl die ursprüngliche Form des TPT Blocks 1. Ordnung als auch die zusätzlichen benötigten Komponenten für die Feedback Pfade.

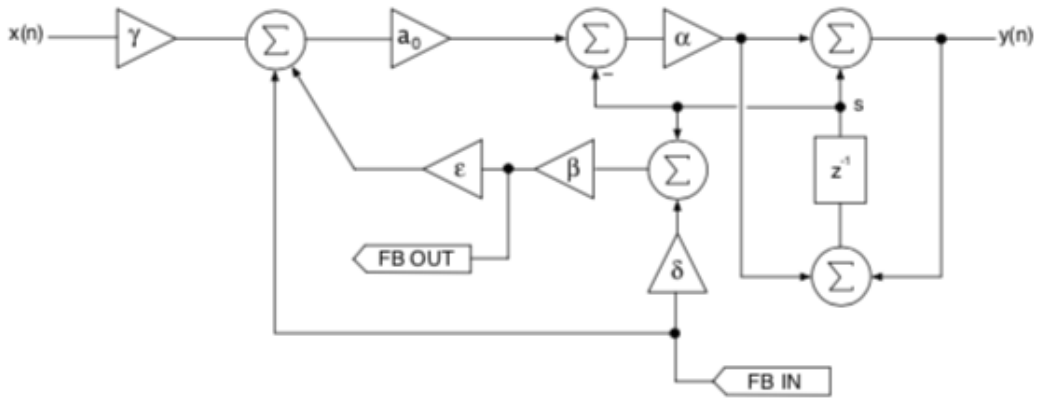

Abbildung 42 modifizierter TPT Lowpass Filter mit den benötigten Feedback Inputs und Outputs

In Abbildung 42 ist das finale TPT Lowpass Filter Modul (1. Ordnung) als Blockdiagramm zu sehen. Rechts befindet sich die „Standard“ TPT Schaltung, links sind eine Reihe an neuen Koeffizienten für die Umsetzung der Feedback Inputs und Outputs, und die Summierung des Inputs mit dem Feedback zu sehen. Es ist somit möglich ein Feedback Signal (FB IN) in das Modul einzuspeisen, sowie ein Feedback Signal aus dem Modul zu extrahieren (FB OUT). Die Funktionalität dieses Blockdiagramm ist im Python Prototyp in der Klasse „LP“ umgesetzt. (siehe Kapitel Python Implementation)

Zur weiteren Veranschaulichung der Feedback Beziehung zwischen den einzelnen Modulen nehmen wir wieder die LPF3 und LPF4 Kombination her.

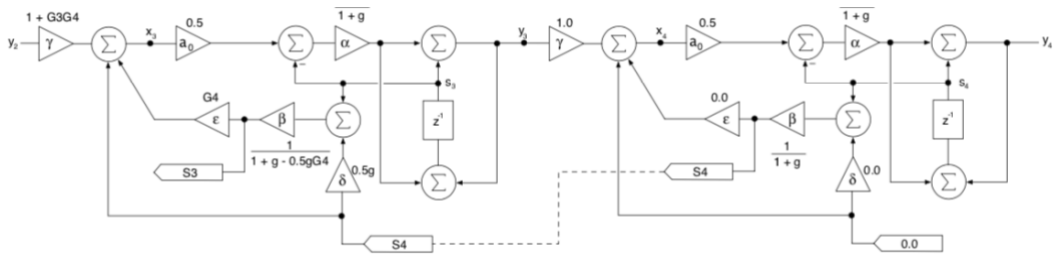

Abbildung 43 LPF3/LPF4 Filter Paar mit allen Feedback Paths

Der Feedback Output Port von LPF4 wird als S4 bezeichnet. Die strichlierte Linie zeigt die Verbindung zwischen den Feedback Output und Input Ports, und somit wie das Feedback von LPF4 in LPF3 eingespeist wird. Auf dieselbe Art wird auch S3 in LPF2 eingespeist. Somit lassen sich anhand des Block Diagramms (Abbildung 43) folgende Aussagen treffen:

$$x_4 = y_3$$

$$x_3 = y_2(1 + G3G4) + G4S3 + S4$$

$$S4 = \frac{S_4}{1 + g}$$

$$S3 = \frac{0{,}5gS_4 + S_3}{1 + g - 0{,}5gG4}$$

Der gesamte Diode Ladder Filter besteht aus allen 4 LPFs in Serie geschalten, sowie einen globalen delay-less Feedback Path durch den das Resonanzverhalten des Filters entsteht. Wir können nun also die vorhergehenden Berechnungen der 4 LPFs implementieren und sie in derselben Weise wie beim vorherigen Beispiel miteinander verbinden.

Daraus entsteht die komplette Diode Ladder (ohne globalen Feedback Pfad), wie in Abbildung 44 zu sehen ist. Wichtig sind die Feedback Pfade zwischen den einzelnen Filter Stufen. Mithilfe der SN Feedback In- und Output Knoten wird z.B. S4 in LPF3 eingespeist, S3 in LPF2, usw.

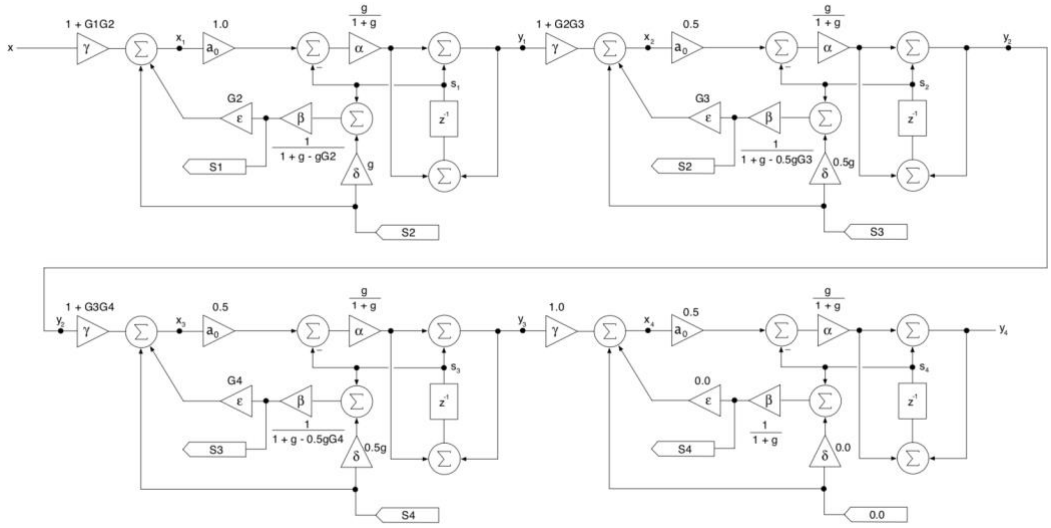

Abbildung 44 Die komplette Diode Ladder (ohne globalen Feedback Pfad)

Um nun die ganze Serie in einen globalen Feedback Loop einzuspeisen benötigen wir eine weitere Gleichung. Pirkle (2013) nennt diese Gleichung „TPT delay-less k-Loop Equation“. Die Gleichung besagt, wenn ein Element, bestehend aus TPT Blöcken, in einen delay-less Feedback Loop mit einer Feedback Variable k gespeist wird, kann der Output des Elements in folgender Form beschrieben werden:

$$y = G_x + S$$

Demnach verhält sich der Input in das Element (u) folgendermaßen (entweder Summierung oder Differenzierung des Feedback Path):

Positives Feedback:

$$u = \frac{x + kS}{1 - kG}$$

Negatives Feedback:

$$u = \frac{x - kS}{1 + kG}$$

Dies zeigt auch einen Vorteil der Herangehensweise Zavalishin's mit TPT Blöcken auf, und ist der Grund warum delay-less Feedback Loops als lineare Gleichungen, anstatt von differenzial Gleichungen, beschrieben werden können.

Weiters muss der finale Output y4 der gesamten Diode Ladder berechnet werden. Dies kann durch ineinander einsetzen der vorher berechneten Gleichungen der einzelnen Filter Stufen gelöst werden:

$$y_4 = G4y_3 + S4$$

$$= G4(G3y_2 + S3) + S4$$

$$= G4G3(G2y_1 + S2) + (G4S3 + S4)$$

$$= G4G3G2y_1 + (G4G3S2 + G4S3 + S4)$$

$$= G4G3G2(G1x + S1) + (G4G3S2 + G4S3 + S4)$$

$$= G4G3G2G1x + (G4G3G2S1 + G4G3S2 + G4S3 + S4)$$

$$= G_x + S$$

(Zavalishin, 2015)

Nun lässt sich diese Gleichung in die Berechnung des globalen Feedback Loops einsetzen. Zur Vereinfachung, und um Verwirrung mit den anderen G und S Variablen zu vermeiden werden wir das Ergebnis der obigen Gleichung mit folgenden griechischen Buchstaben beschreiben:

$$= \Gamma_x + \Sigma$$

$$\Gamma = G4G3G2G1$$

$$\Sigma = G4G3G2S1 + G4G3S2 + G4S3 + S4$$

Und somit:

$$u = \frac{x - k\Sigma}{1 + k\Gamma}$$

Wir können beobachten, dass 1/1+kΓ ein Skalarwert ist, und somit der finale Input in den Filter durch die Differenz von x und $k\Sigma$, multipliziert mit 1/1+kΓ (Σ beschreibt die Summe aller S), errechnet wird.

Abbildung 45 zeigt das vollständige Block Diagramm:

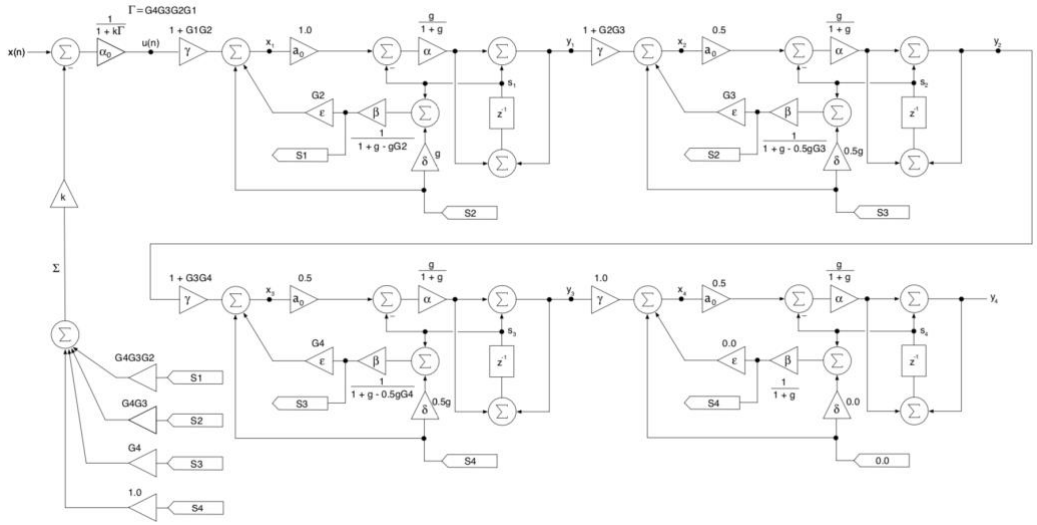

Abbildung 45 Vollständiges Block Diagramm des Diode Ladder Filter (inklusive globalen Feedback Path)

4.1.1.2 Nonlinear Model

Um Nichtlinearitäten zum Filter System hinzuzufügen gibt es verschiedenste Ansätze. Die NLP Blöcke können an verschiedensten Stellen im Block Diagramm eingefügt werden, und es können verschiedenste nichtlineare Funktionen eingesetzt werden (siehe Kapitel Nichtlinearitäten).

Dabei gibt es laut Zavalishin (2015) „simple“ und „advanced“ Modelle für die Einführung von Nichtliniearitäten.

4.1.1.2.1 Simple nonlinear Model

Für die einfachste Implementierung von Nichtliniearitäten in einem Ladder Filter Modell, wird von Zavalishin (2015) vorgeschlagen lediglich einen NLP Block zu verwenden. Dabei kommt die tanh Funktion zum Einsatz. Der Block kann aber an verschiedene Positionen in der Struktur gesetzt werden:

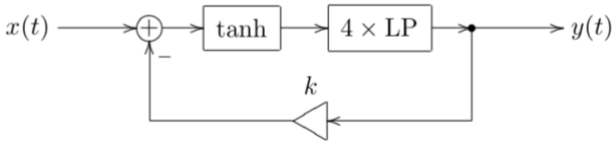

Abbildung 46 Simple nonlinear Model mit einem NLP Block am Input in die Ladder

Man könnte den NLP Block am Input in die Ladder einsetzen. In dieser Position beeinflusst der Block nicht nur das reine Input Signal, sondern auch das Feedback Signal, und gibt somit dem gesamten Signal, dass in die Ladder Struktur geht, eine schöne Overdrive Saturation.

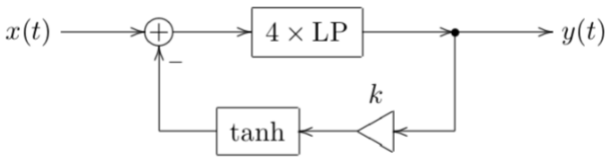

Abbildung 47 Simple nonlinear Model mit einem NLP Block im globalen Feedback Pfad

Eine weitere Möglichkeit wäre den NLP Block in den globalen Feedback Pfad einzusetzen. Dies hat einen eher transparenteren Effekt, da hier das reine Input Signal nicht beeinflusst wird, sondern nur das globale Feedback Signal. Weiters könnte auch der NLP Block mit dem Feedback Gain (Resonanz) k vertauscht werden, um die tanh Funktion unabhängig vom eingestellten k Wert zu machen.

Zavalishin bezeichnet diese Ansätze auch als die „cheap method“. Diese Ansätze können Probleme bei hohen Cutoff Einstellungen mit sich bringen. Außerdem ist, auch generell bei der Einführung von Nichtlinearitäten, zu beachten, dass immer, im Regelfall eine unendliche Anzahl, Obertöne zum Signal hinzugefügt werden. Dies wiederum erzeugt Aliasing. Deshalb wird Oversampling wichtig. Je stärkere Nichtliniearitäten in das System eingefügt werden, desto mehr muss oversampled werden. (Zavalishin, 2015)

Pirkle (2013) schlägt mit seiner sogenannten „budget“ Version eine weitere Variante, die einfach zu implementieren ist, vor. Dabei werden die NLP Blöcke im Feedback Pfad ignoriert und lediglich ein NLP Block am Input (außerhalb des globalen Feedback Loops) eingefügt. Abbildung 48 zeigt die budget NLP Version des Diode Ladder Filters.

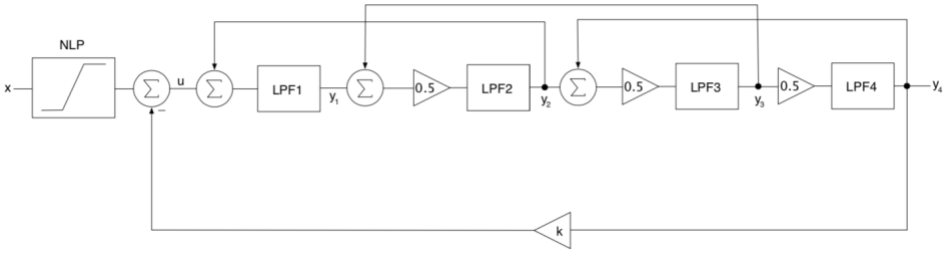

Abbildung 48 Budget NLP Version des Diode Ladder Filters mit NLP Block am Input

4.1.1.2.2 Advanced nonlinear Model

Die simplen nichtlinearen Ansätze kommen schon sehr nah an das Verhalten des analogen Hardware Filters heran. Trotzdem sind diese Ansätze in ihrer Funktionsweise doch noch etwas entfernt von der Weise wie Nichtliniearitäten im analogen Diode Ladder Filter wirklich funktionieren.

Die advanced nichtlineare Version des Filters ist schwierig zu synthetisieren. Sie enthält NLP Blöcke in den lokalen Feedback Pfaden der einzelnen Filter Stufen, sowie einen NLP Block am Input in die Ladder Struktur. (Pirkle, 2013)

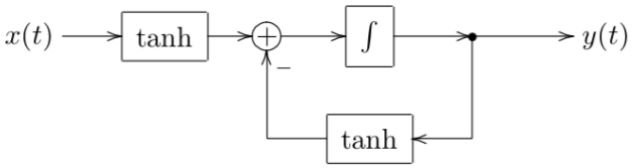

Abbildung 49 1-Pole Lowpass Filter Stufe mit NLP Blöcken am Input sowie im lokalen Feedback Pfad

Somit muss damit angefangen werden Nichtliniearitäten in den einzelnen 1-Pole Lowpass Filter Stufen der Diode Ladder einzufügen. In Abbildung 49 ist ein vereinfachtes Block Diagramm einer einzelnen Lowpass Filter Stufe mit NLP Blöcken im lokalen Feedback Pfad, sowie am Input in die Filterstufe, zu sehen.

Dies hat den Effekt das der Filter seinen eigenen Status langsamer updated, und somit näher am Verhalten seines analogen Vorbilds ist. (Zavalishin, 2015)

Nun können wir wiederum dieses einzelne Nichtlinearen Lowpass Filter Modelle viermal in Reihe schalten, um das Advanced nonlinear Diode Ladder Filter Modell zu erhalten. In Abbildung 50 ist das Modell als Block Diagramm dargestellt.

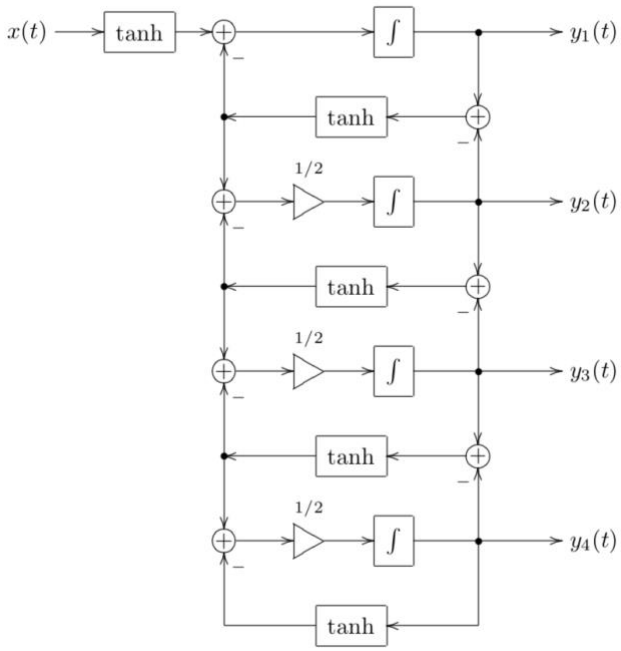

Abbildung 50 Advanced nonlinear Model

Dieses Model könnte nun noch mit weiteren NLP Blöcken im globalen Feedback Pfad ausgestattet werden. Dabei kann auf die Strategien des simplen nichtlinearen Modells zurückgegriffen werden (Abbildung 46 & 47), und somit ein NLP Block entweder nach der Summierung des Input Signal mit dem globalen Feedback Signal, oder im globalen Feedback Pfad selbst (entweder vor oder nach k). Im Kapitel Nonlinear Implementation wird beschrieben wie diese Strategien zur Einführung von Nichtliniearitäten praktisch auf den Filter angewendet werden.

4.1.2 Python Implementation

Nachdem das Blockdiagramm des Filters erfolgreich synthetisiert wurde, geht es daran das Modell in Python zu implementieren. Python eignet sich gut, um den Filter zu testen und zu analysieren, weswegen sich diese Programmiersprache gut für die Programmierung eines Prototyps eignet. Weiters liefert der folgende Python Code die Basis für die darauffolgende GenExpr Implementation des Diode Ladder Filters, um diesen in Max/Msp nutzen zu können.

4.1.2.1 Lineare Implementation

Zuerst wird, wie zuvor bei der Berechnung des Block Diagramms, ein lineares Modell des Filters implementiert. Das lineare Modell ist zunächst einfacher und weniger fehleranfälliger zu realisieren, und kann nach der erfolgreichen

Fertigstellung problemlos mit NLP Blöcken zu einem nichtlinearen Modell erweitert werden.

Als erstes wurde die Klasse LP erstellt. In dieser Klasse wird die Funktionalität unseres TPT Moduls (Abbildung 42) bzw. einer einzelnen Filter Stufe in der Ladder umgesetzt.

Listing 5. LP Klasse

```
1.  class LP ():
2.      def __init__(self):
3.          self._Z1 = 0;             # z-1 storage variable
4.
5.          self.Alpha = 1.0;         # Feed-Forward coeff
6.          self.Beta = -1.0;         # Feed-Back coeff from s + FB_In
7.          self.Gamma = 1.0;         # Pre-Gain
8.          self.Delta = 0.0;         # FB_In coeff
9.          self.Epsilon = 1.0;
10.         self.Feedback = 0.0;      # Feed-Back storage register
11.         self.a0 = 1.0;            # filter gain
12.
13.     def getFeedbackOutput(self):
14.         return self.Beta*(self._Z1 + self.Feedback*self.Delta);
15.
16.     def setFeedback(self, fb_in):
17.         self.Feedback = fb_in;
18.
19.     def doFilter(self, xn):
20.         self.x_in = (xn * self.Gamma + self.Feedback + self.Epsilon * se
    lf.getFeedbackOutput());
21.         self.vn = (self.a0 * self.x_in - self._Z1) * self.Alpha;
22.         self.out = self.vn + self._Z1;
23.         self._Z1 = self.vn + self.out;
24.
25.         return self.out;
```

Z1 ist hier eine private Variable, da nur vom jeweiligen (Filter) Objekt auf die Variable zugegriffen werden darf. In Python gibt es ansich keine Funktion, um eine Variable auf private zu setzen, die Variablen werden aber meist mit einem „“ vor dem Variablennamen geschrieben um sie als private zu kennzeichnen.

Alle weiteren Koeffizienten werden zunächst in der Klasse mit Standardwerten initialisiert, und später nach Berechnung der jeweiligen Koeffizienten für die jeweilige Filter Stufe überschrieben.

Weiters besitzt die Klasse 3 Funktionen:

getFeedbackOutput realisiert die Ausgabe des lokalen Feedbacks der jeweiligen Filter Stufe (z.B. bei LPF4 gibt die Funktion das Feedback S4 aus, siehe Abbildung 45).

setFeedback ermöglicht das einspeisen von Feedback in die jeweilige Filter Stufe (z.B. LPF3 bekommt S4 von LPF4 eingespeist). Dieses Feedback wird wiederum in der getFeedbackOutput Funktion eingespeist, womit der Feedback Output der Filter Stufe auch vom Feedback der darauffolgenden Filter Stufe abhängig ist.

doFilter berechnet den Output der jeweiligen Filter Stufe (z.B. y2 von LPF2). In dieser Funktion ist also der im Block Diagramm (Abbildung 42) beschriebene Prozess umgesetzt.

Listing 6. Initialisierung der 4 LPF Objekte

```
# init 4 lpf objects
lpf1 = LP()
lpf2 = LP()
lpf3 = LP()
lpf4 = LP()
```

Im nächsten Schritt werden zunächst die 4 LPFs, die die Diode Ladder bilden, als Objekte der Klasse LP initialisiert.

Listing 7. Setzung der Parameter

```
Fc = 2000              # cutoff
K = 16                 # resonance
SampleRate = sr        # Sample Rate
```

In diesem Abschnitt werden die Parameter des Filters gesetzt. Die Variable Fc enthält die Cutoff Frequenz. K enthält den Resonanz Wert.

Listing 8. Berechnung von GN und Gamma

```
# calculate alphas
wd = 2 * np.pi * Fc;
T  = 1 / SampleRate;
wa = (2 / T) * np.tan(wd * T / 2);
g = wa * T / 2;

# Big G's
G4 = (0.5 * g) / (1 + g);
G3 = (0.5 * g) / (1 + g - 0.5 * g * G4);
G2 = (0.5 * g) / (1 + g - 0.5 * g * G3);
G1 = g / (1 + g - g * G2);

# big G value Gamma
GAMMA = G4*G3*G2*G1;
SG1 = G4*G3*G2;
SG2 = G4*G3;
SG3 = G4;
SG4 = 1.0;
```

Nun geht es an die Berechnung der Koeffizienten der einzelnen Filter Stufen. Zunächst wird g unter Einflussnahme der Cutoff Frequenz und der Samplerate berechnet. Danach werden die GN und Gamma Werte (die für die Berechnung des Feedbacks benötigt werden) berechnet.

Listing 9. Berechnung der Koeffizienten für LPF1-4

```
# set lpf1 coeffs
lpf1.Alpha = g/(1.0 + g);
lpf1.Beta = 1.0/(1.0 + g - g*G2);
lpf1.Gamma = 1.0 + G1*G2;
lpf1.Delta = g;
lpf1.Epsilon = G2;
lpf1.a0 = 1;

# set lpf2 coeffs
lpf2.Alpha = g/(1.0 + g);
lpf2.Beta = 1.0/(1.0 + g - 0.5*g*G3);
lpf2.Gamma = 1.0 + G2*G3;
lpf2.Delta = 0.5*g;
lpf2.Epsilon = G3
lpf2.a0 = 0.5;

# set lpf3 coeffs
lpf3.Alpha = g/(1.0 + g);
lpf3.Beta = 1.0/(1.0 + g - 0.5*g*G4);
lpf3.Gamma = 1.0 + G3*G4;
lpf3.Delta = 0.5*g;
lpf3.Epsilon = G4
lpf3.a0 = 0.5;

# set lpf4 coeffs
lpf4.Alpha = g/(1.0 + g);
lpf4.Beta = 1.0/(1.0 + g);
lpf4.Gamma = 1.0;
lpf4.Delta = 0.0;
lpf4.Epsilon = 0.0;
lpf4.a0 = 0.5;
```

Nachdem alle GN Werte berechnet sind können die endgültigen Koeffizienten der einzelnen 4 Filter Stufen berechnet werden. Dafür werden die zuvor berechneten Gleichungen der einzelnen LPFs verwendet. Bei jedem der 4 LPF Objekten werden die Variablen des jeweiligen Objekts mit den neu erechneten Koeffizienten überschrieben.

Listing 10. doFilter Funktion

```
def doFilter(xn):

    un = np.zeros_like(xn)
    Y = np.zeros_like(un)
    SIGMA = np.zeros_like(un)

    for i in range (len(xn)):
        # feedback paths inbetween lpfs
```

```
9.          lpf4.setFeedback(0.0);
10.         lpf3.setFeedback(lpf4.getFeedbackOutput());
11.         lpf2.setFeedback(lpf3.getFeedbackOutput());
12.         lpf1.setFeedback(lpf2.getFeedbackOutput());
13.
14.         # calc global feedback path
15.         SIGMA[i] = SG1*lpf1.getFeedbackOutput() + SG2*lpf2.getFeedbackOu
    tput() + SG3*lpf3.getFeedbackOutput() + SG4*lpf4.getFeedbackOutput();
16.
17.         # input + global feedback path
18.         un[i] = (xn[i] - K*SIGMA[i])/(1 + K*GAMMA);
19.
20.         # do filter
21.         Y[i] = lpf4.doFilter(lpf3.doFilter(lpf2.doFilter(lpf1.doFilter(u
    n[i]))))
22.
23.     return Y
```

Als letztes wird eine doFilter Funktion definiert. Die Funktion führt den gesamten Diode Ladder Filter auf einem Eingangssignal aus, und gibt den finalen Output des Filters aus. Hier kommen sozusagen alle vorherigen Berechnungen zusammen.

Da der Algorithmus die Samples des Eingangssignal nacheinander einzeln bearbeitet, läuft die Verarbeitung innerhalb einer for-Schleife (mit Länge des Eingangsarrays) ab. Zunächst werden die lokalen Feedback Pfade zwischen den einzelnen Filterstufen gelöst (SN, siehe Abbildung 45). Dazu werden die zuvor in der LP Klasse definierten Funktionen setFeedback und getFeedbackOutput verwendet. Danach wird der Sigma Wert für die Berechnung des globalen Feedback Pfad mithilfe der zuvor berechneten GN Variablen und den Feedback Outputs der einzelnen Filter Stufen berechnet. Darauf wird der globale Feedback Pfad mit dem Eingangssignal summiert.

Als letzter Schritt werden die doFilter Funktionen der einzelnen Filter Stufen kaskadiert aufgerufen, um letztendlich die gesamte Diode Ladder auf das Eingangssignal anzuwenden.

Um nun den gesamten Diode Ladder Filter auf ein Eingangssignal anzuwenden, muss lediglich die doFilter Funktion aufgerufen werden.

Listing 11. Aufruf des Diode Ladder Filter

```
1.  output = doFilter(x)
```

4.1.2.2 Nonlinear Implementation

Verschiedenste Strategien wie Nichtlinearitäten zum Filter hinzugefügt werden können wurden bereits im Kapitel Nonlinear Model beschrieben. Doch wie können die NLP Blöcke in der Python Implementation eingefügt werden?

Zunächst wird, aufgrund seiner Einfachheit, das Budget NLP Modell implementiert. Hierbei wird lediglich ein NLP Block am Input (vor der Summierung mit dem globalen Feedback) eingesetzt. Als nichtlineare Funktion für diesen Block habe ich mich für tanh entschieden. Die Funktionalität des NLP Blocks wurde in der doFilter Funktion integriert. Weiters wurde ein NLP Block im globalen Feedback Pfad (unabhängig von k) eingefügt Die neue doFilter Funktion sieht folgendermaßen aus:

Listing 12. Implementation des Budget Nonlinear Model in der doFilter Funktion

```
1.  def doFilter(xn):
2.
3.      un = np.zeros_like(xn)
4.      Y = np.zeros_like(un)
5.      SIGMA = np.zeros_like(un)
6.      Saturation = 1
7.
8.      for i in range (len(xn)):
9.          # feedback paths inbetween lpfs
10.         lpf4.setFeedback(0.0);
11.         lpf3.setFeedback(lpf4.getFeedbackOutput());
12.         lpf2.setFeedback(lpf3.getFeedbackOutput());
13.         lpf1.setFeedback(lpf2.getFeedbackOutput());
14.
15.         # calc global feedback path
16.         SIGMA[i] = SG1*lpf1.getFeedbackOutput() + SG2*lpf2.getFeedbackOu
    tput() + SG3*lpf3.getFeedbackOutput() + SG4*lpf4.getFeedbackOutput();
17.         # global feedback path NLP (normalized tanh sat)
18.         SIGMA[i] = (1.0/np.tanh(1))*np.tanh(1*(0.5*SIGMA[i]));
19.
20.         # input NLP (normalized tanh sat)
21.         xn[i] = (1.0/np.tanh(Saturation))*np.tanh(Saturation*xn[i]);
22.
23.         # input + global feedback path
24.         un[i] = (xn[i] - K*SIGMA[i])/(1 + K*GAMMA);
25.
26.         # do filter
27.         Y[i] = lpf4.doFilter(lpf3.doFilter(lpf2.doFilter(lpf1.doFilter(u
    n[i]))))
28.
29.     return Y
```

Da der erste NLP Block auf den Input wirkt wird innerhalb der doFilter Funktion das xn Array durch die tanh Funktion bearbeitet. Mithilfe der Variable „Saturation“ kann die Steilheit der tanh Kurve, und somit die Stärke der Saturation, beeinflusst werden. Hierfür ist der Standardwert auf 1 gesetzt, und kann später im fertigen Synthesizer mithilfe eines Reglers verstellt werden, um mit verschieden steilen tanh Kurven zu experimentieren zu können. Da die NLP Blöcke innerhalb er Struktur auch wie ein Gain Element wirken, gestaltet sich der zweite NLP Block, der sich im Feedback Pfad befindet, als etwas schwieriger zu implementieren. Um die Nichtlinearität im globalen Feedback Pfad unabhängig von k zu

implementieren wird das SIGMA Array, bevor es mit k multipliziert wird, mit der tanh Funktion bearbeitet. Wenn die tanh Funktion allerdings genauso wie beim ersten NLP Block am Input implementiert wird entstehen einige sehr starke Obertöne oberhalb der Cutoff Frequenz, die natürlich nicht gewollt sind und den Filter zerstören. Um dies zu verhindern wird das Signal bevor die tanh Funktion angewandt wird abgeschwächt (in diesem Fall um die Hälfte). Außerdem ist die Steilheit der Kurve auf 1 festgesetzt und kann nicht verändert werden, wie es beim NLP Block am Input der Fall ist.

Auf dasselbe Problem stoßen wir beim Versuch einen tanh NLP Block in die Feedback Pfade zwischen den einzelnen Filter Stufen einzubauen. Deswegen ist es ratsam hierbei auf andere Arten von nichtlinearen Funktionen zurückzugreifen, die weniger aggressiv auf das Signal wirken. Dafür bietet sich zum Beispiel die „Cubic Distortion“ Funktion an. Um die Cubic Distortion in Python zu implementieren wird zunächst die Funktion „cubicdist“ definiert, um die den NLP Block einfach an der gewünschten Stelle einbauen zu können.

Listing 13. cubicdist Funktion

```
def cubicdist(xin):
    if xin <= -1:
        return -2/3
    if xin >= 1:
        return 2/3
    else:
        return xin - xin**3/3
```

Somit kann die Funktion einfach durch den Befehl cubicdist() im Feedback Pfad eingebaut werden. Die Funktionalität der Feedback Pfade zwischen den einzelnen Filter Stufen ist in der Klasse LP implementiert, daher findet die neue cubicdist Funktion dort ihren Einsatz. Die neue LP Klasse sieht folgendermaßen aus:

Listing 14. Neue LP Klasse

```
class LP ():
    def __init__(self):
        self._Z1 = 0;            # z-1 storage variable

        self.Alpha = 1.0;        # Feed-Forward coeff
        self.Beta = -1.0;        # Feed-Back coeff from s + FB_In
        self.Gamma = 1.0;        # Pre-Gain
        self.Delta = 0.0;        # FB_In coeff
        self.Epsilon = 1.0;
        self.Feedback = 0.0;     # Feed-Back storage register
        self.a0 = 1.0;           # filter gain

    def getFeedbackOutput(self):
```

```
14.         return cubicdist(self.Beta*(self._dZ1 + self.Feedback*self.Delta
    ));
15.
16.     def setFeedback(self, fb_in):
17.         self.Feedback = fb_in;
18.
19.     def doFilter(self, xn):
20.         self.x_in = (xn * self.Gamma + self.Feedback + self.Epsilon * se
    lf.getFeedbackOutput());
21.         self.vn = (self.a0 * self.x_in - self._Z1) * self.Alpha;
22.         self.out = self.vn + self._dZ1;
23.         self._dZ1 = self.vn + self.out;
24.
25.         return self.out;
```

Mithilfe der getFeedbackOutput Funktion der Klasse bekommen die jeweils vorherigen Filter Stufen in der Ladder ihr Feedback vom nachfolgenden Filter, daher wurde an dieser Stelle die neue cubicdist Funktion, und somit eine neue Nichtlinearität in den Feedback Pfaden zwischen den einzelnen Lowpass Filter Stufen, implementiert.

Somit besitzt das nichtlineare Diode Ladder Filter Modell nun einen NLP Block am Input, einen NLP Block im globalen Feedback Pfad (unabhängig von k) sowie einen NLP Block innerhalb der lokalen Feedback Loops der einzelnen Filter Stufen selbst, und innerhalb der Feedback Loops zwischen den einzelnen Filter Stufen.

4.1.3 Gen Implementation

Nachdem der Diode Ladder Filter in Python erfolgreich implementiert wurde, wurde der Code in GenExpr portiert, um den Filter in Max/Msp nutzen zu können.

GenExpr ist die Programmiersprache innerhalb des gen~ Objekts. Damit ist es möglich in Max/Msp eigene Objekte in Code zu schreiben, um sie danach mit anderen Max Objekten in gewohnter Art und Weise zu verbinden. Gen hat durch seine Funktionsweise einige Vorteile gegenüber normalen Max Objekten, wobei einer dieser Vorteile essentiell für die Implementation eines Filters ist.

Bei GenExpr handelt es sich um eine sehr ähnliche Sprache wie C. Der in die Gen Codebox geschriebene Code wird Sample für Sample ausgeführt, im Gegensatz zu normalen Max Objekten, die über Blöcke von mehreren Samples operieren. Durch diese Funktionsweise wird überhaupt die Programmierung von Filtern möglich gemacht, da es daher in Max nicht möglich ist 1 Sample Delays zu erzeugen, die essentiell für die Umsetzung der meisten Filter sind (da die lokalen Feedback Pfade innerhalb eines Samples durchgeführt werden müssen, wie es natürlich auch beim Diode Ladder Filter der Fall ist). Durch diese

Funktionsweise können Objekte auch effizienter gestaltet werden. GenExpr bietet natürlich auch die Möglichkeit Standard Befehle der C Sprache wie zum Beispiel Verzweigungen (If), Schleifen (for, while) und Funktionen zu verwenden. Somit ist es auch relativ simpel den Python Code, oder auch andere Programmiersprachen, in GenExpr zu portieren. Außerdem können Algorithmen, die in GenExpr geschrieben wurden als C++ Code exportiert werden, und somit relativ einfach in verschiedene andere Audio Applikationen wie zum Beispiel VST oder Audio Unit Plugins integriert werden. Des Weiteren kann in Gen nicht nur Code geschrieben werden, sondern auch im klassischen Max Style Objekte miteinander verbunden werden. Diese Patches können wiederum sofort in Code umgewandelt werden, und somit für eine Vielzahl an Anwendungen verwendet werden.

Da GenExpr quasi den gesamten Code für jedes Sample ausführt, muss der Python Code des Filters nur etwas umgedacht werden. GenExpr bietet leider auch keine Klassen bzw. objektorientierte Programmierung, somit kann auch die LP Klasse mit ihren 4 LP Objekten nicht in derselben Weise verwendet werden.

Listing 15. Cubicdist Funktion in Gen

```
1.  cubicdist(xin) {
2.          dist = xin;
3.          if(xin <= -1) {
4.              dist = -2/3;
5.              }
6.          else if(xin >= 1) {
7.              dist = 2/3;
8.              }
9.          else {
10.             dist = xin - (xin*xin*xin)/3;
11.             }
12.         return dist;
13.     }
```

In Gen müssen Funktionen immer zu Beginn der Codebox definiert werden. Somit wird als erstes die cubicdist Funktion umgesetzt. Dabei ändert sich kaum etwas zur Python Version.

Als nächstes werden die für den Filter essentiellen Zn Variablen definiert. Diese werden benötigt um die 1 Sample Delays (Z^{-1} Operation) für die lokalen Feedback Loops der einzelnen Filter Stufen zu erzeugen.

Listing 9. Definition von Zn Variablen in Gen

```
1.  History _Z1(0.), _Z2(0.), _Z3(0.), _Z4(0.);
```

Hierfür gibt es in Gen den History Operator. Dieser muss ebenso zu Beginn der Codebox (gleich nach den Funktions Definitionen) deklariert werden. Für jede der 4 Filter Stufen gibt es eine eigene Z Variable, die mit Wert 0 initialisiert wird.

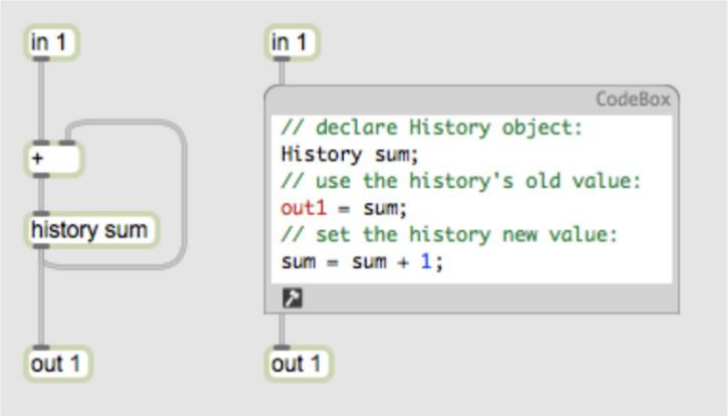

Abbildung 51 History Operator in Gen

Wie in Abbildung 51 zu sehen ist, wird dadurch vom Inlet in den History Operator der Outlet Wert für das nächste Sample gesetzt (beziehungsweise der Outlet Wert des History Operators ist der Inlet Wert vom vorhergehenden Sample). Es kann damit also der Wert einer Variable nach einem Durchgang des Algorithmus (also nach einem einzelnen Sample) für den nächsten Durchgang (bzw. für das nächste Sample), gespeichert werden. Dies wäre mit einer normalen Variable nicht möglich, da diese bei jedem Sample neu initialisiert wird, und somit nichts vom vorherigen Sample weiß. Daher ist es durch den History Operator, sowie der generellen auf einzelnen Samples basierten Funktionsweise von Gen, im Gegensatz zu normalen Max Objekten möglich 1 Sample Delays, bzw. Z^{-1} Operationen durchzuführen. Dies wird benötigt um die lokalen um 1 Sample Delayed Feedback Loops der einzelnen Filter Stufen zu lösen.

Um den Filter einerseits ein Eingangsignal einzuspeisen, und auch andererseits seine Parameter (Cutoff Frequenz, Resonanz, Input Drive) zu steuern hat der Gen Patch 4 Inlets. Die einzelnen Werte der Inlets werden zunächst in der Codebox in Variablen gespeichert, und natürlich ebenfalls nach jedem Sample geupdatet.

Listing 16. Definition der Parameter durch Inlets

```
1.  x = in1;
2.  cutoff = in2;
3.  K = in3;
4.  Saturation = in4;
```

Genauso wie im Python Code werden als nächstes die GN Werte und der Gamma Wert berechnet.

Listing 17. Berechnung von GN und Gamma

```
// calculate alphas
wd = 2 * PI * cutoff;
T  = 1 / samplerate;
wa = (2 / T) * tan(wd * T / 2);
g = wa * T / 2;

// Big G's
G4 = (0.5 * g) / (1 + g);
G3 = (0.5 * g) / (1 + g - 0.5 * g * G4);
G2 = (0.5 * g) / (1 + g - 0.5 * g * G3);
G1 = g / (1 + g - g * G2);

// big G value Gamma
GAMMA = G4*G3*G2*G1;
SG1 = G4*G3*G2;
SG2 = G4*G3;
SG3 = G4;
SG4 = 1.0;
```

Da GenExpr wie Eingangs schon erwähnt keine Objektorientierte Programmierung ermöglicht, musste bei der Klassen Struktur der einzelnen Low Pass Filter Stufen aus dem Python Code umgedacht werden. Daher gibt es in der Gen Implementation keine LP Objekte, sondern lediglich einige Variablen für jeden einzelnen Filter. Ebenso gibt es in der Gen Implementation keine setFeedback und getFeedback Funktion. Diese Funktionalität wurde ebenfalls mit lediglich einer Feedback Variable (um den Feedback Input der jeweiligen Filter Stufe zu speichern) sowie mit Einsetzen der Berechnung des Feedback Outputs (anstatt dem Funktionsaufruf) gelöst.

Listing 18. Berechnung der Koeffizienten von LPF1-4 in Gen

```
// set lpf1 coeffs
lpf1_Alpha = g/(1.0 + g);
lpf1_Beta = 1.0/(1.0 + g - g*G2);
lpf1_Gamma = 1.0 + G1*G2;
lpf1_Delta = g;
lpf1_Epsilon = G2;
lpf1_a0 = 1;

// set lpf2 coeffs
lpf2_Alpha = g/(1.0 + g);
lpf2_Beta = 1.0/(1.0 + g - 0.5*g*G3);
lpf2_Gamma = 1.0 + G2*G3;
lpf2_Delta = 0.5*g;
lpf2_Epsilon = G3;
lpf2_a0 = 0.5;

// set lpf3 coeffs
lpf3_Alpha = g/(1.0 + g);
```

```
lpf3_Beta = 1.0/(1.0 + g - 0.5*g*G4);
lpf3_Gamma = 1.0 + G3*G4;
lpf3_Delta = 0.5*g;
lpf3_Epsilon = G4;
lpf3_a0 = 0.5;

// set lpf4 coeffs
lpf4_Alpha = g/(1.0 + g);
lpf4_Beta = 1.0/(1.0 + g);
lpf4_Gamma = 1.0;
lpf4_Delta = 0.0;
lpf4_Epsilon = 0.0;
lpf4_a0 = 0.5;
```

Die Feedback Pfade zwischen den einzelnen Filter Stufen werden daher wie gerade beschrieben gelöst. Der globale Feedback Pfad wird wie gehabt berechnet, nur dass hier ebenfalls die Berechnungen der Feedback Outputs statt der Funktion eingesetzt wurden. Danach wird SIGMA durch die Cubic Distortion Funktion als NLP Block bearbeitet. Weiters wird das Eingangssignal mit einem tanh NLP Block (mit variabler Stärke mithilfe der Saturation Variable, die sich später einfach durch einen Drehregler einstellen lässt) bearbeitet. Der folgende Code entspricht der doFilter Funktion der Python Version.

Listing 19. Berechnung der Feedback Pfade zwischen den Filtern, des globalen Feedback Pfad, Input NLP Block und Summierung des Inputs und globalen Feedback Pfad

```
// feedback paths inbetween lpfs
lpf4_Feedback = 0.;
lpf3_Feedback = cubicdist(lpf4_Beta*(_Z4 + lpf4_Feedback*lpf4_Delta));
lpf2_Feedback = cubicdist(lpf3_Beta*(_Z3 + lpf3_Feedback*lpf3_Delta));
lpf1_Feedback = cubicdist(lpf2_Beta*(_Z2 + lpf2_Feedback*lpf2_Delta));

// calc global feedback path
SIGMA = SG1*cubicdist(lpf1_Beta*(_Z1 + lpf1_Feedback*lpf1_Delta)) + SG2*
cubicdist(lpf2_Beta*(_Z2 + lpf2_Feedback*lpf2_Delta)) + SG3*cubicdist(lp
f3_Beta*(_Z3 + lpf3_Feedback*lpf3_Delta)) + SG4*cubicdist(lpf4_Beta*(_Z4
 + lpf4_Feedback*lpf4_Delta));
SIGMA = cubicdist(SIGMA);

// input NLP (normalized tanh sat)
x = (1.0/tanh(Saturation))*tanh(Saturation*x);

// input + global feedback path
un = (x - K*SIGMA)/(1 + K*GAMMA);
```

Als letzten Schritt müssen nur noch alle 4 Lowpass Filter Stufen in Reihe ausgeführt werden. Dabei wird auf den Code der doFilter Funktion der LP Klasse aus der Python Version zurückgegriffen und für jeden der 4 LPFs einzeln ausgeführt (die Berechnung der Feedback Outputs wird wieder statt der Funktion eingesetzt). Der Output der ersten Filter Stufe wird dabei als Input für die zweite Filter Stufe verwendet, und so weiter, bis am Ende in der Variable output4 das

Endergebnis vorliegt. Am Ende wird das Ergebnis über out1 (repräsentiert in diesem Fall das erste Outlet des Gen Objekts) ausgegeben.

Listing 20. Ausführung von LPF1-4 (doFilter)

```
1.  // doFilter lpf1
2.  x_in = (un * lpf1_Gamma + lpf1_Feedback + lpf1_Epsilon * cubicdist(lpf1_
    Beta*(_Z1 + lpf1_Feedback*lpf1_Delta)));
3.  vn = (lpf1_a0 * x_in - _Z1) * lpf1_Alpha;
4.  output1 = vn + _Z1;
5.  _Z1 = vn + output1;
6.
7.  // doFilter lpf2
8.
9.  x_in2 = (output1 * lpf2_Gamma + lpf2_Feedback + lpf2_Epsilon * cubicdist
    (lpf2_Beta*(_Z2 + lpf2_Feedback*lpf2_Delta)));
10. vn2 = (lpf2_a0 * x_in2 - _Z2) * lpf2_Alpha;
11. output2 = vn2 + _Z2;
12. _Z2 = vn2 + output2;
13.
14. // doFilter lpf3
15.
16. x_in3 = (output2 * lpf3_Gamma + lpf3_Feedback + lpf3_Epsilon * cubicdist
    (lpf3_Beta*(_Z3 + lpf3_Feedback*lpf3_Delta)));
17. vn3 = (lpf3_a0 * x_in3 - _Z3) * lpf3_Alpha;
18. output3 = vn3 + _Z3;
19. _Z3 = vn3 + output3;
20.
21. // doFilter lpf4
22.
23. x_in4 = (output3 * lpf4_Gamma + lpf4_Feedback + lpf4_Epsilon * cubicdist
    (lpf4_Beta*(_Z4 + lpf4_Feedback*lpf4_Delta)));
24. vn4 = (lpf4_a0 * x_in4 - _Z4) * lpf4_Alpha;
25. output4 = vn4 + _Z4;
26. _Z4 = vn4 + output4;
27.
28.
29. out1 = output4;
```

Nun haben wir ein gen~ Nonlinear Diode Ladder Filter Objekt (mit vier Inlets und einem Outlet), das sich in einem Max Patch wie gewohnt einbauen lässt.

4.1.3.1 Oversampling

Da es aber, wie schon erwähnt, wichtig ist beim Einfügen von Nichtlinearitäten (da dies zu Aliasing führen kann) Oversampling zu betreiben wurde das gen~ Objekt in ein poly~ Objekt verbaut, um damit 4x Oversampling für den Filter Algorithmus zu erreichen (Abbildung 52). Im Abschnitt „Filter Messungen“ im Kapitel „Evaluation“ wird auch gezeigt warum Oversampling Sinn macht.

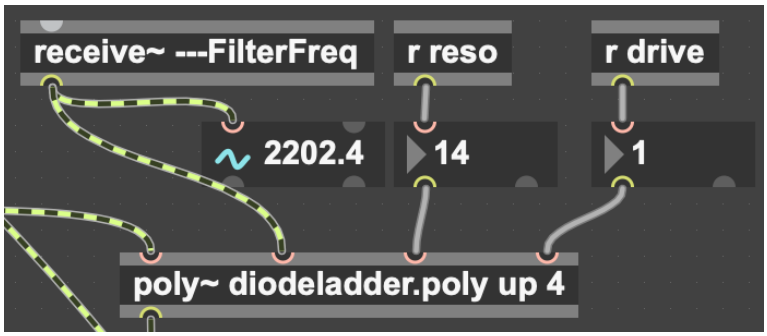

Abbildung 52 Diode Ladder Filter gen~ Implementation innerhalb des übergeordneten Synthesizer Max Patch

4.2 Filter Envelope Modulation

In der originalen analogen Hardware sind, wie bereits erwähnt, insgesamt zwei Envelope Generatoren verbaut. Um den Diode Ladder Filter zu vervollständigen wird zunächst die Filter Envelope Modulation implementiert.

Die Envelope Modulation des Diode Ladder Filter ist ein weiterer wichtiger Bestandteil des Sounds der TB-303. So kann zum Beispiel erst durch die Kombination von einer hohen Resonanz Einstellung sowie der Modulation der Cutoff Frequenz der charakteristische quietschende Acid Sound des Synthesizers erzeugt werden.

4.2.1 Filter Envelope Generator Subpatch (FilterEnvelope)

Beim Filter Envelope Generator handelt es sich um einen AD Envelope, mit Exponentiell Abfallenden Decay. Lediglich der Decay Parameter ist variabel, Attack, Sustain und Release sind fixiert.

Zunächst wurden Versuche mit dem in Max standardmäßig enthaltenen adsr~ Objekt durchgeführt. Beim adsr~ Objekt handelt es sich aber um einen linearen Envelope Generator, somit konnte mit diesem Objekt nicht dasselbe Modulationsverhalten wie beim analogen Vorbild erreicht werden. Nach vielen weiteren Versuchen mit unterschiedlichsten Envelope Generatoren, wurde letztendlich für dieses Projekt der Analog-style ADSR Envelope Generator in Gen von Peter McCulloch (2015) verwendet, da dieser in seinem Verhalten dem der TB-303 sehr ähnelt. Der Envelope Generator befindet sich im vollständigen Max Patch im FilterEnvelope Subpatch (Abbildung 53).

Das Envelope gen~ Objekt besitzt sechs Inlets: Gate, Trigger, Attack, Decay, Sustain, Release. In diesem Fall muss nur der Decay Parameter variabel sein,

deshalb bekommt dieses Inlet seinen Wert über ein Receive Objekt, das seinen Wert wiederum vom Decay Drehregler im GUI bekommt. Die anderen drei Envelope Parameter werden mit einer loadmessage auf 0 gesetzt. Außerdem wird natürlich ein Trigger Signal an das Inlet des Subpatches gesendet, um den Envelope Generator bei jedem NoteOn Signal zu starten. Am Outlet des Subpatch wird dann der Envelope ausgegeben. Ein scope~ Objekt steht zusätzlich zur Kontrolle der erzeugten Envelope Kurven innerhalb des Subpatches zur Verfügung.

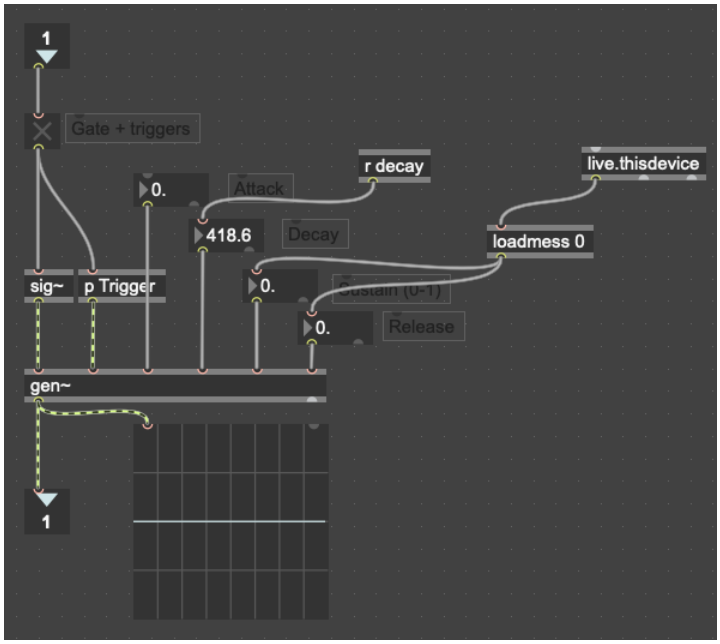

Abbildung 53 FilterEnvelope Subpatch - Filter Envelope Generator

Innerhalb des gen~ Objekts findet sich folgender Code in der Codebox:

Listing 21. Analog-style ADSR Envelope Generator in Gen (McCulloch, 2015), part 1

```
History ad(0);
History asr(0);
History atk_pulse(0);
History eoa(0);
History triggered(0);
History gate_hist(0);
History sustain(0.5);
History dcy(1000);
Param attack_lockout(0);    // If attack_lockout is on, the attack segment MUST complete
                            // before the release or decay can be triggered
```

Zu Beginn kommt, wie schon in der Filter Implementation, der History Operator zum Einsatz, um die einzelnen Parameter Werte zu speichern.

Daraufhin wird dann der Envelope mit folgendem Code generiert, und bei out1 ausgegeben.

Listing 22. Analog-style ADSR Envelope Generator in Gen (McCulloch, 2015), part 2

```
iter = 1 (1/e);
// 0.63212055882856

// The fractional term is the remaining distance for each iteration (1 t
he first time, 1/e the second...)
high_target = 1./(iter*(1/1) + iter*(1/e) + iter*(1/(e*e)));        // G
oing up... (~1.05)
low_target = 1.-
high_target;                                          // Going down... (~-
0.05)

smoothing = 1 -(1/mstosamps(20.));

do_trig = (delta(in2>0) > 0);  // Read from the trigger inlet

gate_in = in1 > 0;

// If triggered, or attack hasn't ended and gate still active or in lock
out mode
atk_pulse = (do_trig || !eoa) && (gate_in || attack_lockout);

attack = mstosamps(in3)*0.33333333333333;
release = mstosamps(in6)*0.33333333333333;
dcy = gate_in*mstosamps(in4)*0.33333333333333 + (!gate_in)*release;  //
Short-circuits so that it uses release time when gate is low
sustain = mix(in5,sustain,smoothing);

puls = atk_pulse > 0 ? high_target : low_target;

asr_gate = gate_in || atk_pulse ? high_target : low_target;
// If you ONLY
// asr_gate = (in1 > 0) ? high_target : low_target;

// Clamp so that on overshoot values hit min/max
ad = clamp(slide(puls,attack,dcy),0.,1.);
asr = clamp(slide(asr_gate,attack,release),0.,1.);     // in1 acts as a
gate signal

eoa = !atk_pulse || (ad >= 1.);


out1 = mix(ad,asr,sustain); // blend the two envelopes based on the sust
ain value
out2 = eoa;
```

4.2.2 Filter Envelope Modulation Subpatch (FilterEnvMess)

Nun wird ein Envelope mit den passenden Eigenschaften generiert und aus dem FilterEnvelope Subpatch ausgegeben. Als nächstes muss die Modulation (in diesem Fall die Modulation der Cutoff Frequenz des Filters) durch den Envelope umgesetzt werden. Die Stärke der Modulation soll am Ende, wie beim analogen Vorbild, ebenfalls durch einen Drehregler im GUI gesteuert werden können.

Dies kann mit ganz normalen Max Objekten gelöst werden. Die Modulation der Cutoff Frequenz wird im FilterEnvMess Subpatch (Abbildung 54) durchgeführt.

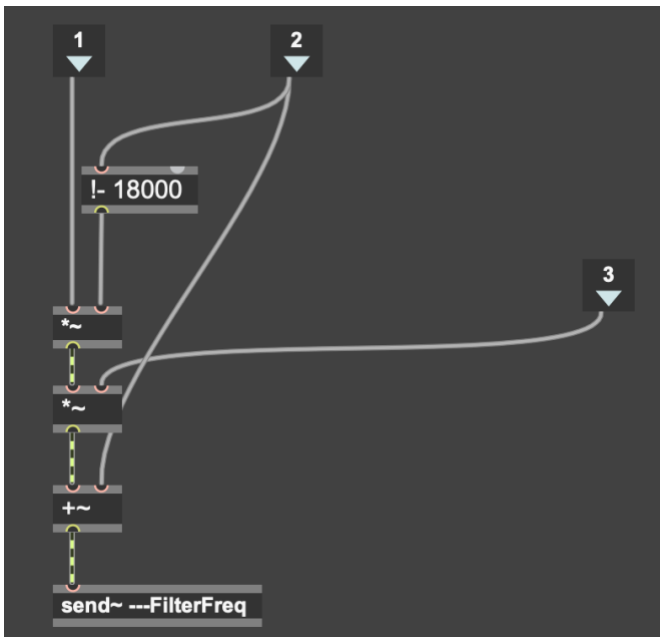

Abbildung 54 FilterEnvMess Subpatch – Modulierung der Cutoff Frequenz

Der Subpatch erhält 3 Inlets. In Inlet 1 wird der vorher im FilterEnvelope Subpatch generierte Envelope gesendet. Inlet 2 bekommt die, mit einem Drehregler im GUI, ausgewählte Cutoff Frequenz. Inlet 3 bekommt den Parameter für die Tiefe der Cutoff Modulation, der ebenfalls mithilfe eines Drehreglers im GUI eingestellt werden kann.

Als erstes wird die Cutoff Frequenz (von Inlet 2) in ein !- 18000 Objekt geschickt. Das !- Objekt funktioniert im Grunde wie das – Objekt, nur werden die Funktionen der Inlets vertauscht. Daher wird der Inlet Wert vom im !- Objekt (in diesem Fall 18000) spezifizierten Wert subtrahiert. Daraufhin wird der bearbeitete Cutoff Wert mit dem von Inlet 1 kommenden Envelope multipliziert. Der daraus entstehende Wert wird dann mit dem Parameter für die Tiefe der Modulation (Inlet 3) ebenfalls multipliziert. Wichtig anzumerken ist das der Wert des Tiefen Parameters, bevor

er in den Subpatch kommt, skaliert wird (Abbildung 55). Am Ende wird nochmal die eingestellte Cutoff Frequenz zu unserem moduliertem Wert hinzuaddiert, um den letztendlich modulierten Cutoff Frequenz Wert wieder in die richtige Dimension zu bekommen. Um den übergeordneten Patch übersichtlicher zu gestalten wird der endgültige modulierte Cutoff Frequenz Wert mit einem send~ Objekt aus dem Subpatch geschickt, und von einem receive~ Objekt (siehe Abbildung 52) empfangen, um den Parameter direkt an den Filter zu übergeben. Somit ist die Filter Envelope Modulation vollständig implementiert.

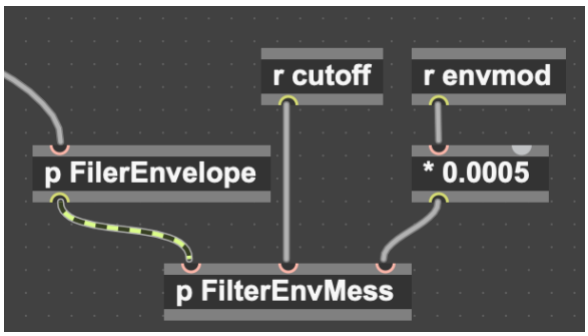

Abbildung 55 FilterEnvMess Subpatch im übergeordneten Patch

4.3 Volume Envelope Modulation

Der zweite der beiden Envelope Generatoren, die in der TB-303 verbaut sind, ist für die Modulation der Amplitude des gesamten Sounds zuständig. In der analogen Hardware sind die Parameter des Envelopes fixiert, und können nicht verändert werden. Ansonsten handelt es sich hierbei um einen Envelope Generator mit fast denselben Eigenschaften wie beim Filter Envelope.

4.3.1 Volume Envelope Generator Subpatch (VolEnvelope)

Es wurden, wie schon beim Filter Envelope, verschiedenste Arten von Envelope Generatoren ausprobiert. Wegen der Ähnlichkeit der beiden Generatoren im analogen Vorbild, wurde für dieses Projekt ebenfalls derselbe Analog-style ADSR Envelope Generator von Peter McCulloch, wie auch beim Filter Envelope, verwendet. Der Envelope Generator selbst wurde bereits im Abschnitt Filter Envelope Generator Subpatch beschrieben.

Da der Volume Envelope aber unabhängig vom Filter Envelope agiert, wurde für den Volume Envelope Generator ein neuer Subpatch erstellt (VolEnvelope). Die beiden Envelope Subpatches unterscheiden sich aber nur in der Wahl der einzelnen Parameter Werte. Der Wert des Decay Parameter ist in diesem Fall

nicht variabel und wird, annäherungsweise an die analoge TB303, auf 800ms fixiert. Der Sustain Parameter wird wieder auf 0 fixiert, um bei kurzen Filter Envelope Decay Zeiten den knackigen Sound des Originals zu ermöglichen. Da jeweils am Anfang und Ende einer Note Knackgeräusche entstanden sind wurde der Attack Parameter auf 3ms, und der Release Parameter auf 10ms fixiert. Mit diesen Einstellungen konnten die Knackser eliminiert werden. Alle Parameter Werte werden beim Laden des Synthesizers mit einer Loadmessage initialisiert siehe Abbildung 56). Um gegenüber dem analogen Original eine kleine Verbesserung einzubauen, könnten zum Beispiel auch die Parameter des Volume Envelopes über Drehregler im GUI gesteuert werden, um dadurch noch ein wenig mehr Möglichkeiten in der Soundgestaltung zu bekommen. Dies würde sich sehr einfach umsetzen lassen.

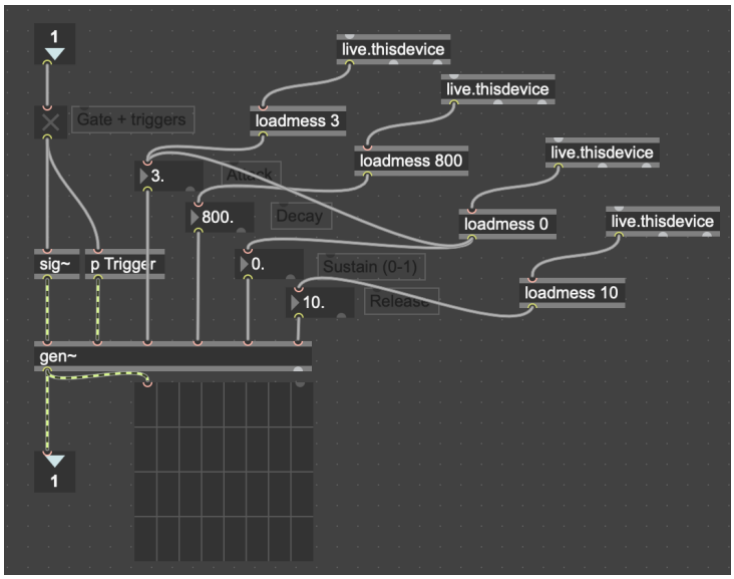

Abbildung 56 VolEnvelope Subpatch - Volume Envelope Generator

4.3.2 Volume Envelope Modulation

Um nun die Amplitude zu modulieren muss das Signal des Envelope Generators lediglich mit dem zu modulierenden Signal multipliziert werden. In diesem Fall wird der Envelope mit dem Output Signal des Filters multipliziert. Somit ist nun auch der zweite der beiden Envelope Generatoren des Synthesizers vollständig implementiert. In Abbildung 57 ist die Multiplikation des Volume Envelopes mit dem Filter Output Signal mithilfe des *~ Objekts zu sehen.

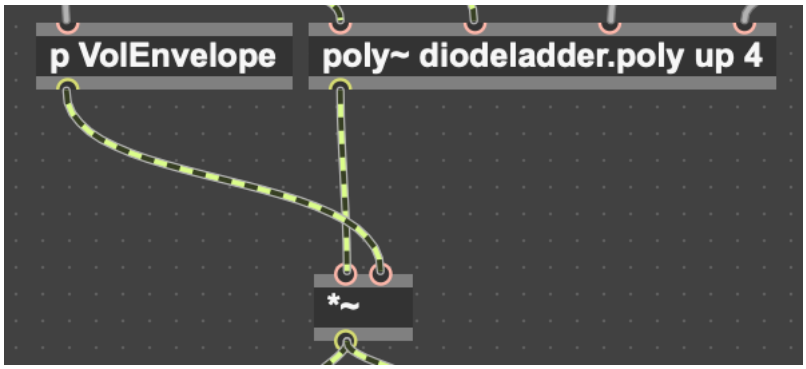

Abbildung 57 Volume Envelope Modulation im übergeordneten Max Patch.

4.4 Oszillator

Die TB-303 beinhaltet einen Oszillator, der entweder eine Sägezahn- oder eine Rechteckwelle erzeugen kann. Im analogen Original entsteht die Rechteckwelle mithilfe eines Waveshapers (dessen Eigenschaften leider unbekannt sind) aus der Sägezahnwelle. Dieselbe Strategie wurde auch in der Software Emulation angewendet.

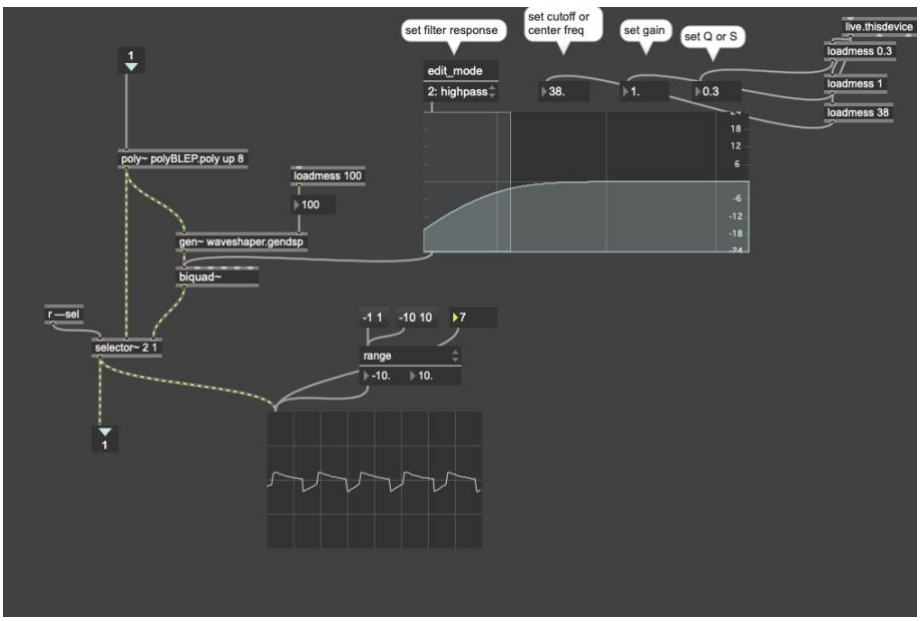

Abbildung 58 Osc Subpatch.

Für die Erzeugung der Sägezahnwelle kommt der polyBLEP Algorithmus zum Einsatz. Dieser wurde innerhalb einer gen~ Codebox implementiert. Das gen~ Objekt befindet sich wiederum in einem poly Objekt, um die Berechnung des Oszillators mit 8x Oversampling durchzuführen.

4.4.1 polyBLEP Sägezahn Oszillator

Wie der polyBLEP Algorithmus in der Theorie funktioniert wurde im Kapitel „DSP Theory für Synthesizer“ beschrieben. In diesem Abschnitt wird beschrieben wie der Algorithmus in gen~ implementiert wurde.

Listing 23. polyBLEP Sägezahn Oszillator Algorithmus in gen~, part 1

```
1.  polyblep(phase, inc){
2.      if(phase < inc){
3.          p = phase / inc;
4.          return p+p – p*p – 1.0;
5.      }
6.      if(phase > (1.0-inc)){
7.          p = (phase-1.0) / inc;
8.          return p+p + p*p + 1.0;
9.      }
10.     else{
11.         return 0;
12.     }
13. }
14.
15. History phase(0.);
```

Zu Beginn der Codebox wird eine polyblep Funktion definiert, die den aktuellen Phasen- und Inkrementwert erhält. Mit dieser Funktion wird die naive Sägezahnwelle angepasst, um die Samples rund um die Diskontinuität zu glätten und daraus eine polyBLEP Sägezahnwelle zu berechnen. Nach der Funktionsdefinition kommt wieder der History Operator zum Einsatz, da der Wert der Variable phase logischerweise bis zum nächsten Sample gespeichert werden muss.

Listing 24. polyBLEP Sägezahn Oszillator Algorithmus in gen~, part 2

```
1.  freq = in1;
2.  sat = 1.5;
3.
4.  ph_inc = freq / samplerate;
5.
6.  naiveSaw = 2 * (1.0/tanh(sat))*tanh(sat*phase) - 1;
7.  y = naiveSaw - polyblep(phase, ph_inc);
8.
9.  phase = phase + ph_inc;
10. if (phase > 1){
11.         phase = phase - 1.0;
12.     }
13.
14. out1 = y;
15. out2 = naiveSaw;
```

Nun müssen zunächst ein paar wichtige Variablen definiert werden. Die Grundfrequenz, mit der der Oszillator schwingen soll (freq) wird über Inlet 1

gesteuert. Weiters wird die sat Variable mit 1,5 initialisiert (dazu kommen wir gleich). Die Variable ph_inc dient dazu den Anstieg der phase Variable, nach jedem Sample, je nach gewählter Grundfrequenz zu steuern.

Danach wird eine naive Sägezahnwelle erzeugt und in der Variable naiveSaw gespeichert. Hier kommt auch die vorher erwähnte sat Variable zum Einsatz: es wird nämlich die tanh Funktion verwendet um einen gebogenen anstatt eines linearen Anstiegs in der Wellenform zu erhalten (wie es beim TB-303 Sägezahn, und auch bei vielen anderen analogen Synthesizern der Fall ist). Durch Veränderung des Wertes der sat Variable kann die Stärke der Biegung in der Wellenform bestimmt werden.

Als nächstes wird der Wert, den die polyblep Funktion für das aktuelle Sample berechnet von der naiven Sägezahnwelle subtrahiert und die endgültige polyBLEP Wellenform in der Variable y gespeichert. Als letzter Schritt wird die phase Variable nach allen Berechnungen für das aktuelle Sample erhöht. Sobald phase 1 übersteigt wird die Variable wieder zurückgesetzt.

Outlet 1 gibt die polyBLEP Sägezahnwellenform aus. Zusätzlich wird in diesem Fall über Outlet 2 auch die naive Sägezahnwelle ausgegeben, um beide Wellenformen einfach vergleichen zu können.

4.4.2 Rechteckwellenform

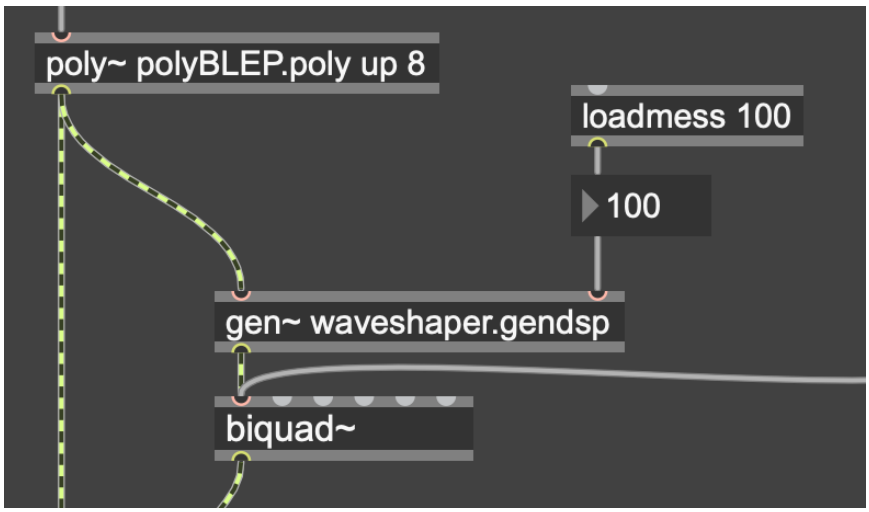

Abbildung 59 Erzeugung der Rechteckwellenform im Osc Subpatch.

Für die Erzeugung der Rechteckwellenform gibt keinen separaten Oszillator, da diese aus der Sägezahnwellenform erzeugt wird. Wie in Abbildung 59 zu sehen

ist wird der Output des polyBLEP Sägezahn Oszillator durch einen Waveshaper bearbeitet und danach mit einem biquad~ Objekt gefiltert.

Der Waveshaper wurde wieder in einer gen~ Codebox implementiert:

Listing 25. Waveshaper in gen~.

```
1.  x = in1;
2.  a = in2;
3.
4.  out1 = x*(abs(x) + a)/(x*x + (a-1)*abs(x) + 1);
```

Inlet 1 ist hierbei das Signal, dass bearbeitet werden soll, und Inlet 2 steuert die Stärke des Waveshapers (in diesem Fall wird dies auf den Wert 100 initialisiert, siehe Abbildung 59). In Abbildung 60 ist der Output des Waveshapers zusehen.

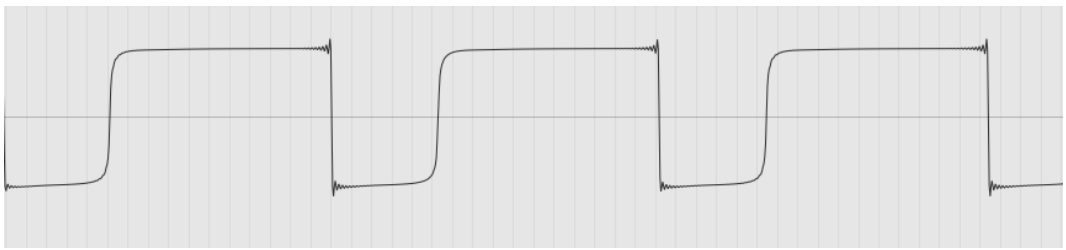

Abbildung 60 Output des Waveshapers.

Mithilfe eines Highpass Filters kann nun dieses Signal weiter bearbeitet werden um eine ähnliche Wellenform wie beim analogen Original zu erhalten. Hierfür wurde ein biquad~ Objekt verwendet. (siehe Abbildung 61)

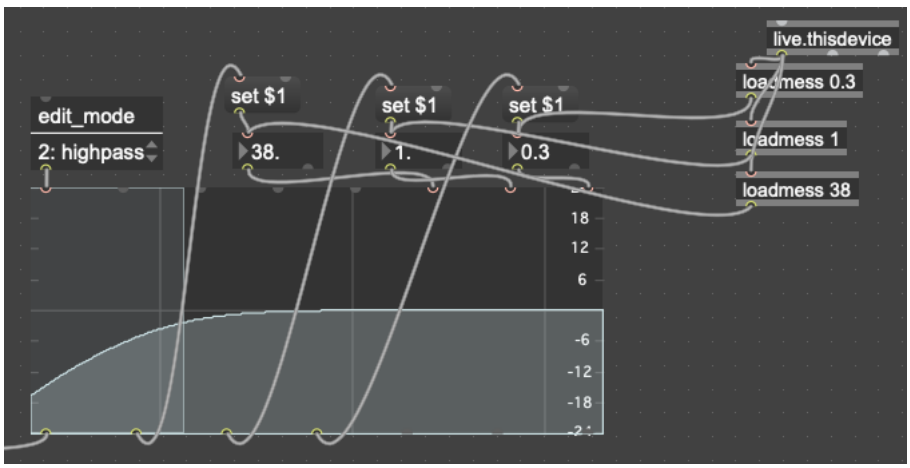

Abbildung 61 biquad~ Filter.

Das beste Ergebnis konnte mit folgenden Einstellungen des Highpass Filters erhalten werden (Abbildung 61): Cutoff-Frequenz = 38 Hz, Filter Gain = 1, Q = 0,3.

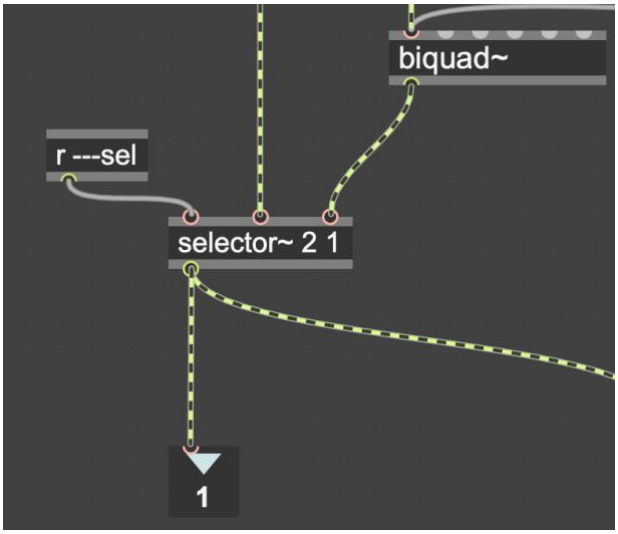

Abbildung 62 selector~ Objekt für die Auswahl der gewünschten Wellenform.

Am Ende des Osc Subpatches befindet sich ein selector~ Objekt, um die vom User gewünschte Wellenform auszuwählen und letztendlich in das Outlet des Subpatches zu schicken (Abbildung 62). Für die Auswahl der Wellenform steht im GUI ein live.tab Button zur Verfügung, dessen Auswahl an das receive ---sel Objekt gesendet wird.

4.5 Accent

Welche Auswirkungen eine accented Note auf verschiedene Bestandteile beziehungsweise Parameter der TB-303 hat wurde im Kapitel „Grundlegende Analyse und Funktionsweise der TB-303“ bereits analysiert. In diesem Abschnitt wird beschrieben wie dieses Verhalten in Max/Msp implementiert wurde.

Zusammengefasst bewirkt eine accented Note folgendes:

- Der Decay Parameter des Filter Envelopes wird für die Dauer der accented Note auf eine sehr kurze Zeit gesetzt. Die Modulation ist aber immer noch von der Einstellung des Envmod Reglers abhängig.
- Die accented Note wird im VCA verstärkt und ist somit lauter (abhängig von der Einstellung des Accent Parameters).

- Durch den Accent Sweep Circuit wird die Cutoff Frequenz bei einer accented Note erhöht.
- Bei mehreren aufeinander folgenden accented Noten wird die Cutoff Frequenz mit jeder der aufeinander folgenden accented Noten noch etwas höher.

Zunächst muss definiert werden wie eine accented Note programmiert wird, bzw. wie diese erkannt wird. Dies wurde mithilfe des Velocity Parameters der Noten gelöst. Eine Note mit einer Velocity von 100 oder mehr gilt als eine accented Note, Noten mit einer Velocity unter 100 gelten als „normale“ Noten. Ansonsten hat der Velocitywert keine weiteren Auswirkungen auf die Noten.

Um in Max/Msp zwischen diesen zwei Arten an Noten unterscheiden zu können wurde der accent? Subpatch erstellt.

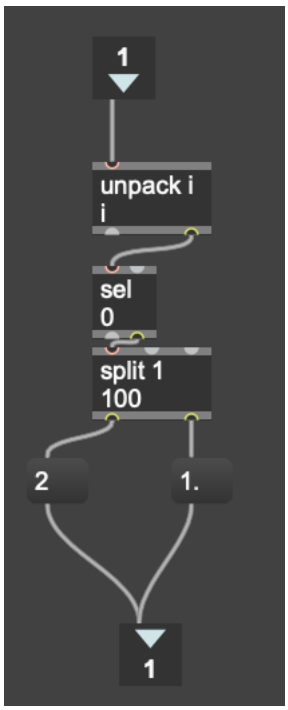

Abbildung 63 accent? Subpatch für die Unterscheidung zwischen normalen und accented Noten.

Der Subpatch (Abbildung 63) bekommt über sein Inlet die eingehende Midi Note und überprüft deren Velocity Wert mithilfe eines split Objekts. Bei einer Velocity von 100 oder mehr, also einer accented Note, wird 1 in das Outlet des Subpatches gesendet. Bei normalen Noten mit einer Velocity unter 100 wird 2 ins Outlet gesendet. Das Ergebnis der Überprüfung auf Accents wird dann über das Accent send Objekt an mehrere Orte im Patch gesendet. (siehe Abbildung 64)

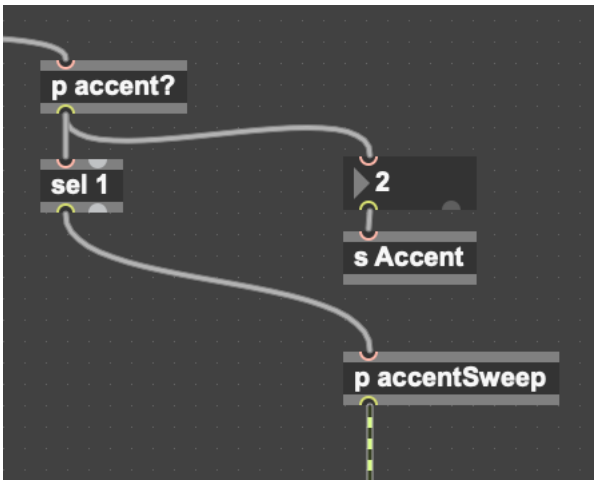

Abbildung 64 accent? und accentSweep Subpatches im übergeordneten Patch.

Weiters wird bei jeder erkannten accented Note ein Bang in den accentSweep Subpatch geschickt. In diesem Subpatch wird das Verhalten des Accent Sweep Circuit emuliert, der sich auf die Cutoff Frequenz des Filters auswirkt.

4.5.1 Accent Sweep Circuit Emulation

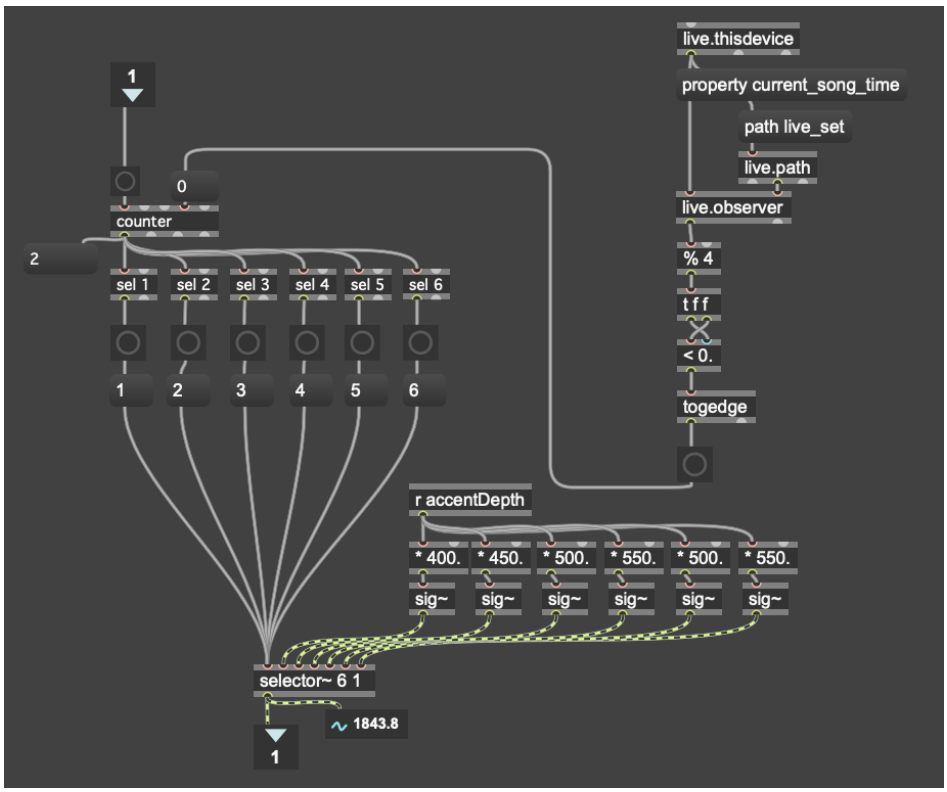

Abbildung 65 accentSweep Subpatch.

Der accentSweep Subpatch (Abbildung 65) setzt zwei Funktionen um: einerseits wird die Erhöhung der Cutoff Frequenz bei einer accented Note berechnet, andererseits wird gezählt wie viele accented Noten aufeinander folgen, und die Cutoff Frequenz dementsprechend angepasst.

Um wie viel die Cutoff Frequenz erhöht wird ist abhängig von der Einstellung des Accent Reglers im GUI. Über das accentDepth receive Objekt gelangt dieser Wert in den Subpatch, und wird mit einem entsprechenden Wert (je nachdem die wievielte accented Note in Reihe erkannt wurde) multipliziert, um die endgültige Erhöhung der Cutoff Frequenz zu berechnen.

Mithilfe eines counter Objekts werden die ankommenden Bangs gezählt, um zu erkennen wie viele accented Noten aufeinander folgen und bei der wievielten accented Note wir uns gerade befinden. Dafür wird aus Ableton Live (welches als Sequencer benutzt wird) ausgelesen wann ein neuer Takt beginnt, damit das counter Objekt am Beginn eines neuen Takts resetet werden kann. Somit wird gezählt wie viele accented Noten innerhalb eines Taktes vorkommen.

Durch das selector~ Objekt vor dem Outlet des Subpatches wird der jeweilige Wert für die Erhöhung der Cutoff Frequenz ausgewählt, je nachdem bei der wievielten accented Note innerhalb eines Taktes wir uns gerade befinden.

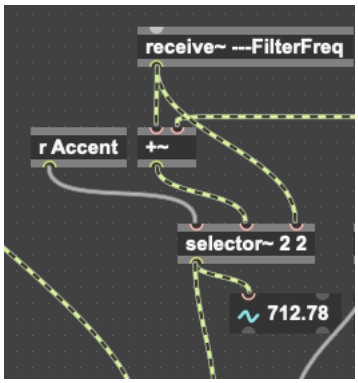

Abbildung 66 Erhöhung der Cutoff Frequenz bei einer accented Note im übergeordneten Max Patch.

Der Wert, der dann aus dem Outlet des accentSweep Subpatch kommt, wird mit der aktuellen Cutoff Frequenz (nachdem diese durch den Filter Envelope moduliert wurde) addiert. Ein selector~ Objekt entscheidet dann ob der addierte Wert oder der originale Wert der Cutoff Frequenz benutzt wird (Abbildung 66). Das Accent receive Objekt erhält bei einer erkannten accented Note den Wert 1, und lässt somit den addierten Cutoff Frequenz Wert durch das selector~ Objekt.

4.5.2 Accent Envelope

Um die verkürzte Decay Zeit des Filter Envelopes im Falle einer accented Note zu realisieren, wurde ein zweiter Envelope Generator im FilterEnvelope Subpatch hinzugefügt. Hierfür wurde, wie auch für die anderen Envelopes der Emulation, wieder der Analog-style ADSR Envelope Generator in Gen von Peter McCulloch (2015) verwendet (der bereits im Abschnitt „Filter Envelope Generator Subpatch“ beschrieben wurde). Dieser kommt dem Verhalten des Envelopes der analogen TB-303 sehr nahe, da es sich hierbei um einen RC Envelope handelt, dessen besondere Eigenschaft ein konvexer Anstieg während der Attack Zeit ist.

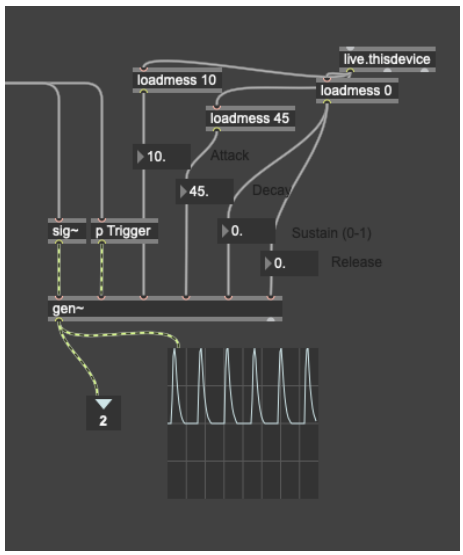

Abbildung 67 Zweiter Envelope Generator für accented Noten im FilterEnvelope Subpatch.

Die Parameter des Accent Envelope Generators wurden auf folgende Werte festgesetzt (Abbildung 67): Attack = 10ms, Decay = 45ms, Sustain = 0, Release = 0ms. Der generierte Accent Envelope wird dann über Outlet 2 des FilterEnvelope Subpatch ausgegeben.

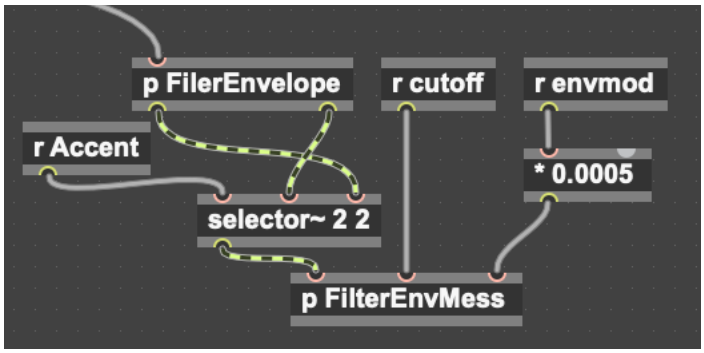

Abbildung 68 selector~ Objekt für die Auswahl des gewünschten Filter Envelope.

Durch ein selector~ Objekt wird wieder entschieden welcher der beiden Envelopes für die Modulation der Cutoff Frequenz verwendet werden soll. Erhält das Accent Receive Objekt den Wert 1 wird der Accent Envelope für die Modulation verwendet. Der Envelope wird dann, wie auch im Falle einer normalen Note, in den FilterEnvMess Subpatch geschickt, um auch die Stärke der Modulation der Cutoff Frequenz durch den Accent Envelope von der Einstellung des Envmod Reglers abhängig zu machen. (Abbildung 68)

4.5.3 Accent VCA Drive

Nun fehlt von den vorher genannten Effekten einer accented Note nur noch die Verstärkung der Note im VCA.

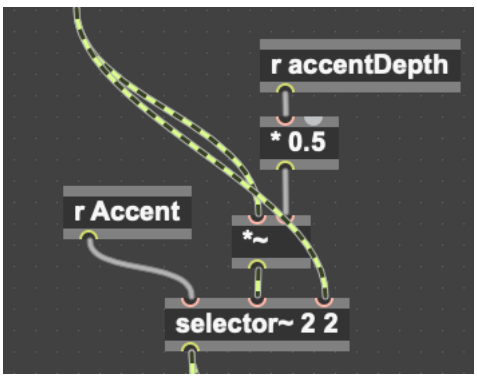

Abbildung 69 Verstärkung von accented Noten.

Hierfür wird das Output Signal des Synthesizers während einer accented Note (die Unterscheidung wird wieder mithilfe eines weiteren selector~ Objekts durchgeführt) mit einen vom Accent Regler abhängigen Wert multipliziert (Abbildung 69).

4.6 Slide und MIDI

Die letzte fehlende Funktion ist das Slide Feature. Wenn Slide auf eine Note programmiert wird, gleitet die Tonhöhe der ausgewählten Note in die Tonhöhe der nächsten Note. Eine Slide Note kann programmiert werden indem sich zwei Midi Noten im Ableton Sequencer überlappen.

Abbildung 70 zeigt die Implementation von Midi und Slide innerhalb des übergeordneten Patch. Die in Ableton programmierten Midi Noten werden zunächst durch das midiin Objekt in Max/Msp ausgelesen. Das darauffolgende midiparse Objekt gibt über sein erstes Outlet die Note-on und Note-off (inklusive Tonhöhe und Velocity) Nachrichten aus. Diese Daten werden auch als Input für den accent? Subpatch verwendet, um accented Noten zu erkennen.

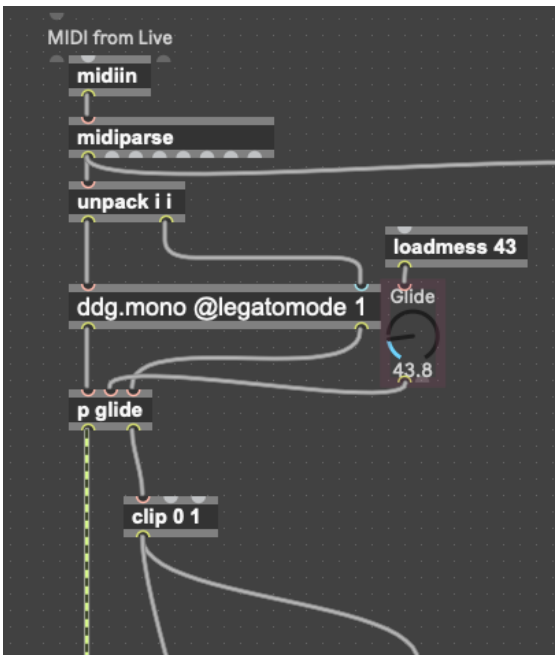

Abbildung 70 Midiin und glide Subpatch im übergeordneten Patch.

Im unpack Objekt werden Tonhöhe und Velocity der Midi Note auf jeweils ein Outlet aufgeteilt, und dann in das ddg.mono Objekt geschickt. Dieses Objekt wird genutzt, um die ankommenden Noten für einen monophonen Synthesizer richtig zu verarbeiten. Dies bedeutet das nur eine einzelne Note, und nicht mehrere Noten gleichzeitig, vom Synthesizer gespielt werden kann. Die vom ddg.mono Objekt ausgegebenen Tonhöhe (erstes Outlet) und Velocity (zweites Outlet) Daten werden danach im glide Subpatch weiterverarbeitet.

In Abbildung 71 ist der glide Subpatch zu sehen. Der Subpatch hat drei Inlets: Inlet 1 erhält die Tonhöhe, und Inlet 3 die Velocity der ankommenden Midi Note.

Über Inlet 2 kann die Glide Time gesteuert werden. Um Slides erkennen zu können, muss überprüft werden ob eine neue Note-on Nachricht, vor der Note-off Nachricht der vorigen Note ankommt.

Auf der rechten Seite des Subpatches wird zuerst überprüft ob die ankommende Velocity ungleich 0 ist, um Note-on Nachrichten zu erkennen. Diese werden dann mithilfe des pipe Objekts um 5ms verzögert. Während diesen 5ms wird überprüft ob eine weitere ankommende Note-on Nachricht erkannt wird. Dies wird mithilfe des gate und des darauffolgenden pack Objekt umgesetzt. Das line~ Objekt ermöglicht letztendlich das Gleiten von einer Note in die nächste. Das mtof~ Objekt wandelt danach die Midi Noten in einen Frequenzwert um, welcher an den Osc Subpatch geschickt wird, um den Oszillator mit der entsprechenden Grundfrequenz schwingen zu lassen.

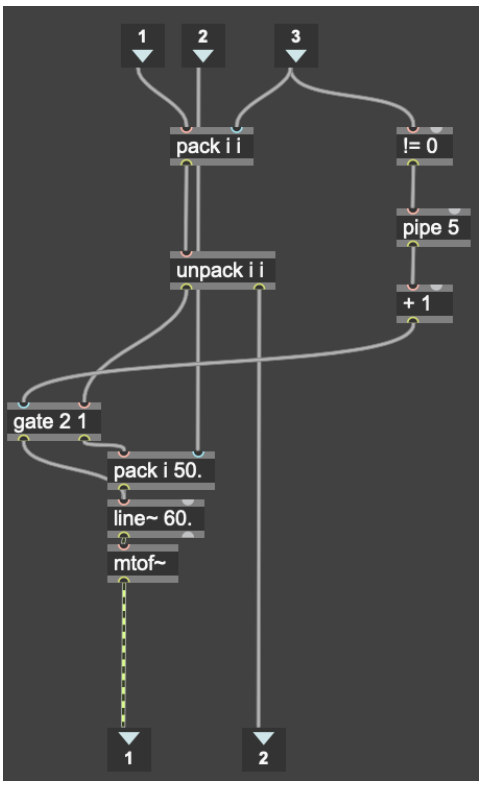

Abbildung 71 glide Subpatch.

Outlet 2 des glide Subpatch gibt den Velocity Wert der Note aus. Der Velocity Wert wird mithilfe eines clip Objekts in die Werte 0 (Note-off) oder 1 (Note-on) umgewandelt (Abbildung 70). Dieser Wert wird wiederum verwendet um den Filter und Volume Envelope Generator bei einer Note-on Nachricht zu triggern.

4.7 GUI

Abbildung 72 GUI der MK303 Software Emulation.

Das GUI der Software Emulation (Abbildung 72) ist dem Aufbau der Regler an der originalen TB-303 nachempfunden, beinhaltet aber auch drei zusätzliche Funktionen. Es stehen 5 große Regler für Cutoff Frequenz, Resonanz, Filter Envelope Modulation, Decay (Filter Envelope) und Accent zur Verfügung. Mit einem kleinen zusätzlichen Drive Regler lässt sich mit der Stärke der Nichtlinearität am Input des Filters experimentieren. Die Wellenform des Oszillators lässt sich durch einen Tab Button auswählen. Daneben kann die Glide Time frei gewählt werden. Neben dem Volume Regler befindet sich ein weiterer (ein- und auschaltbarer) Drive Regler. Mit diesem lässt sich eine weitere Nichtlinearität am finalen Output des Synthesizers steuern (siehe Abbildung 73).

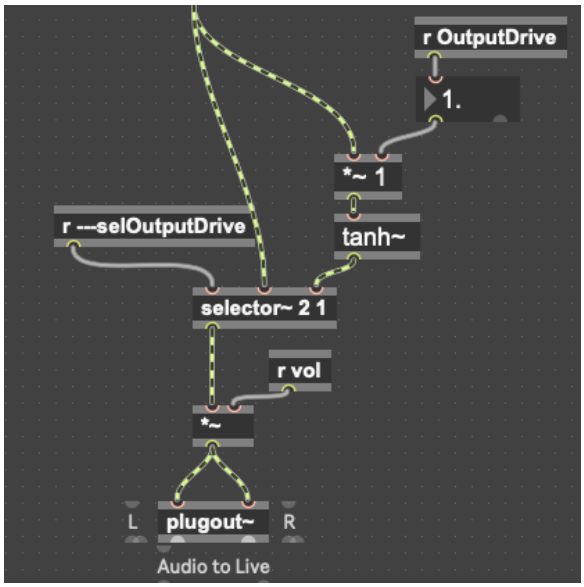

Abbildung 73 Nichtlinearität (Drive) am finalen Output des Synthesizers mithilfe eines tanh~ Objekts.

5 Evaluation

In diesem Kapitel wird die fertige TB-303 Software Emulation evaluiert. Zunächst wurden Messungen des Filters und des Oszillators durchgeführt. Die Ergebnisse finden sich im Abschnitt „Messungen“. Weiters wurde eine qualitative Befragung durchgeführt. Diese wird im Abschnitt „Hörtest“ beschrieben und ausgewertet.

5.1 Messungen

5.1.1 Filter Messungen

Um einerseits die Funktion des Filters zu überprüfen, und um andererseits zu erkennen ob es zwischen der Gen~ und der Python Implementation Unterschiede gibt, wurden Messungen der beiden Filter Implementationen durchgeführt. Leider ist es nicht möglich an der analogen Hardware dieselben Messungen durchzuführen, da weder die originale TB-303, noch die zahlreichen Klone des Geräts eine Möglichkeit bieten den Filter über einen externen Input mit einem anderen Signal als den eingebauten Oszillatoren direkt anzusteuern.

5.1.1.1 Python Implementation Messungen

Zunächst wurden die Impulse und Frequency Response der Python Implementation des Diode Ladder Filters gemessen:

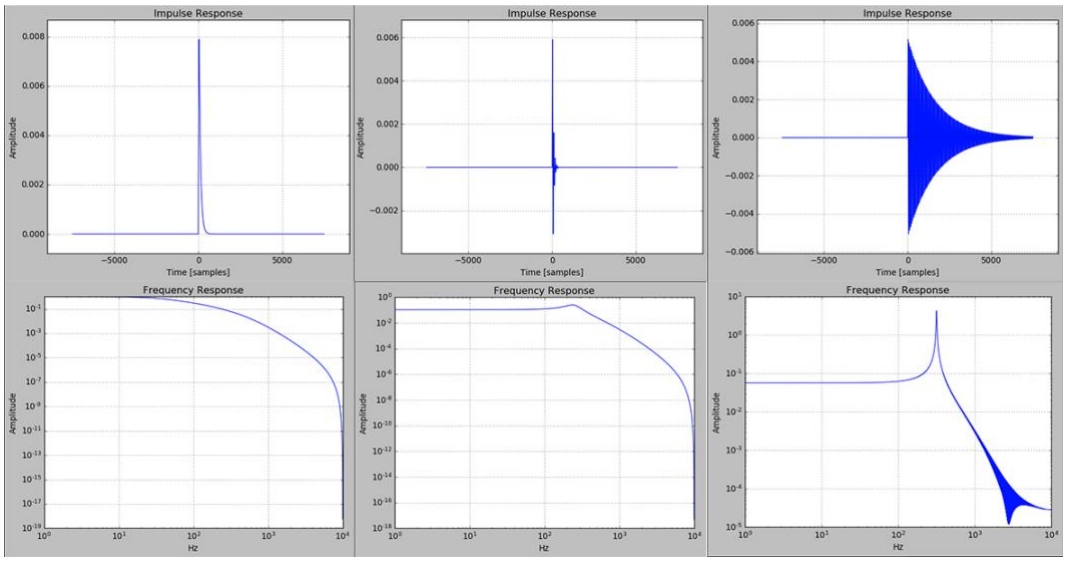

Abbildung 74 Impulse und Frequency Response Messungen der Diode Ladder Filter Implementation in Python. Links: Cutoff = 1 kHz, Resonance = 0, Mitte: Cutoff = 1 kHz, Resonance = 8, Rechts: Cutoff = 1 kHz, Resonance = 16.

Abbildung 74 zeigt die Impulse und Frequency Response Messungen bei einer Cutoff Frequenz von 1 kHz, sowie mit Resonanzwerten von 0 (links), 8 (mitte) und 16 (rechts). Auffällig ist vor allem der Gain Verlust im Durchlassbereich wenn die Resonanz erhöht wird. Dieser Effekt ist bei Ladder Filtern jedoch zu erwarten. Außerdem fällt auf, dass die Spitze der Resonanz sich nicht an der Cutoff Frequenz (1 kHz) befindet. Dies ist ebenfalls korrekt, da sich die Spitze um den Faktor $\frac{fc}{\sqrt{2}}$ verschieben soll. (Zavalishin, 2015) Bei einer Cutoff Frequenz von 1000 Hz ist die Spitze der Resonanz also bei 707 Hz zu erwarten, so wie es in den Messungen auch der Fall ist.

Während weiteren Praxistests konnte festgestellt werden, dass der Filter ab einem Resonanzwert von 17 selbst zu oszillieren beginnt, er bleibt dabei aber stabil. Daher macht es Sinn, wenn der Filter in einen Synthesizer implementiert wird, die Resonanz auf 16,9 zu limitieren, falls dieser Effekt nicht gewünscht ist.

Weiters wurde die Performance des Filters bei verschiedenen Cutoff Frequenzen gemessen:

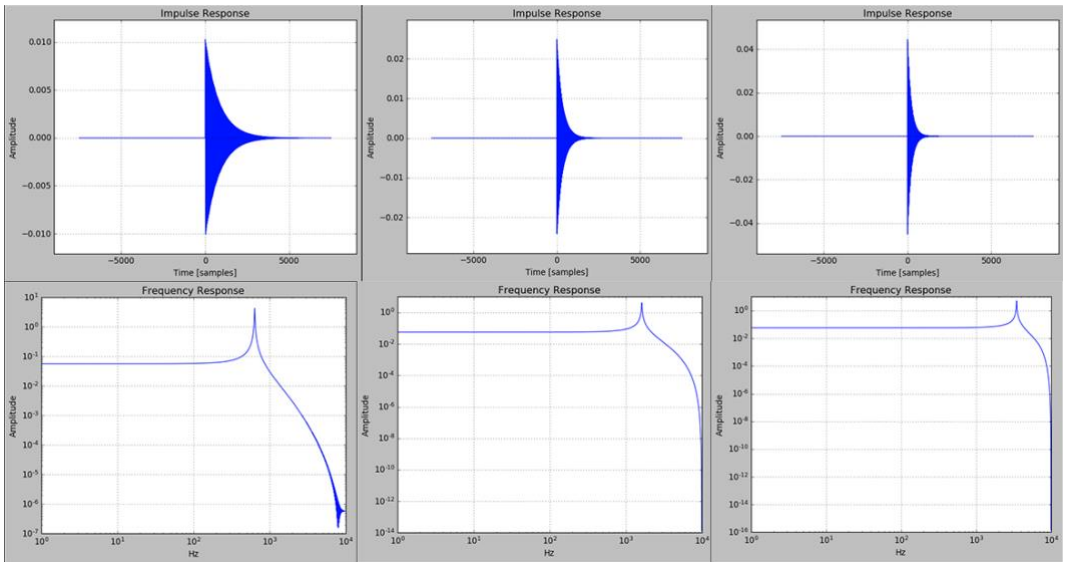

Abbildung 75 Impulse und Frequency Response Messungen der Diode Ladder Filter Implementation in Python. Links: Cutoff = 2 kHz, Resonance = 16, Mitte: Cutoff = 5 kHz, Resonance = 16, Rechts: Cutoff = 10 kHz, Resonance = 16.

Abbildung 75 zeigt die Impulse und Frequency Response Messungen bei einer Cutoff Frequenz von 2 kHz (links), 5 kHz (mitte) und 10 kHz (rechts), sowie einer Resonanz von 16. Der Filter bleibt bei diesen Parametern stabil.

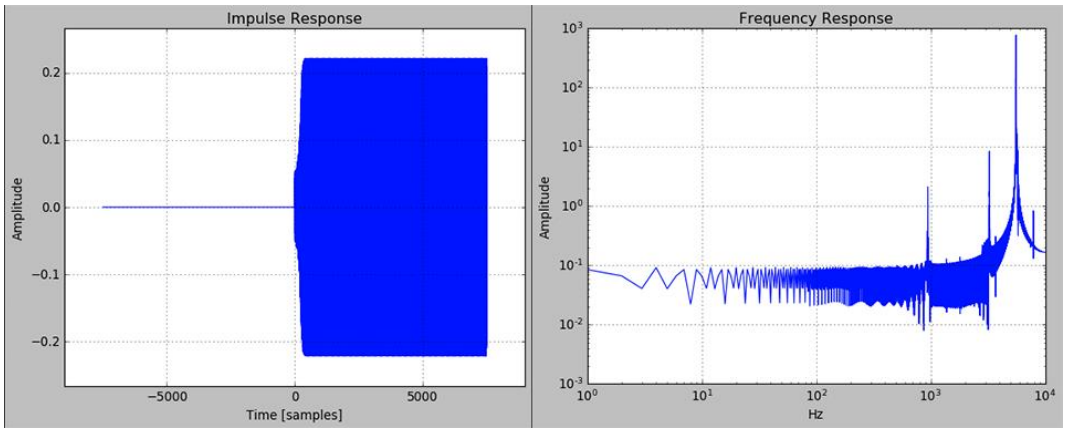

Abbildung 76 Impulse und Frequency Response Messungen der Diode Ladder Filter Implementation in Python. Cutoff = 15 kHz, Resonance = 16.

Ab einer Cutoff Frequenz von 15 kHz (bei einer Resonanz von 16) wird der Filter instabil, wie in Abbildung 76 zu sehen ist. Wird die Resonanz jedoch auf 15 reduziert, bleibt der Filter bis zu einer Cutoff Frequenz von 18 kHz stabil (siehe Abbildung 77).

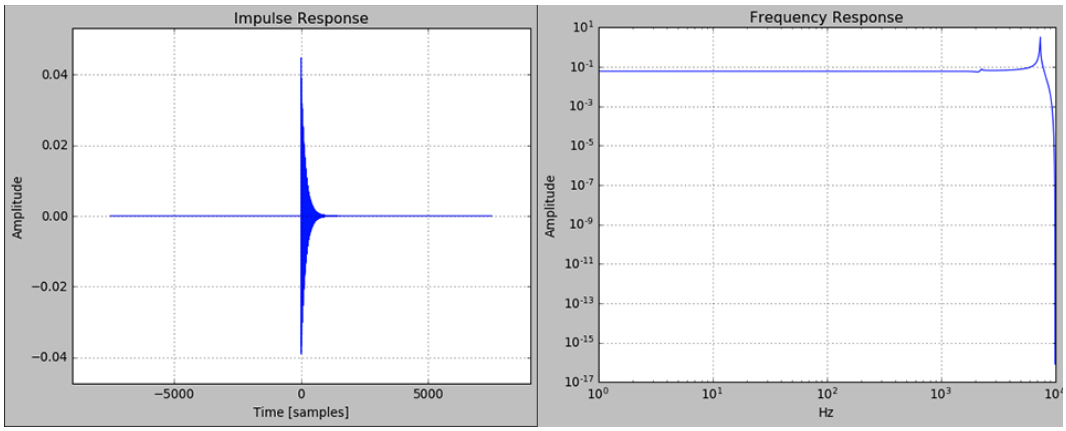

Abbildung 77 Impulse und Frequency Response Messungen der Diode Ladder Filter Implementation in Python. Cutoff = 18 kHz, Resonance = 15.

5.1.1.2 Gen Implementation Messungen

Als nächstes wurden die Impulse und Frequency Response der Gen Implementation des Filters gemessen:

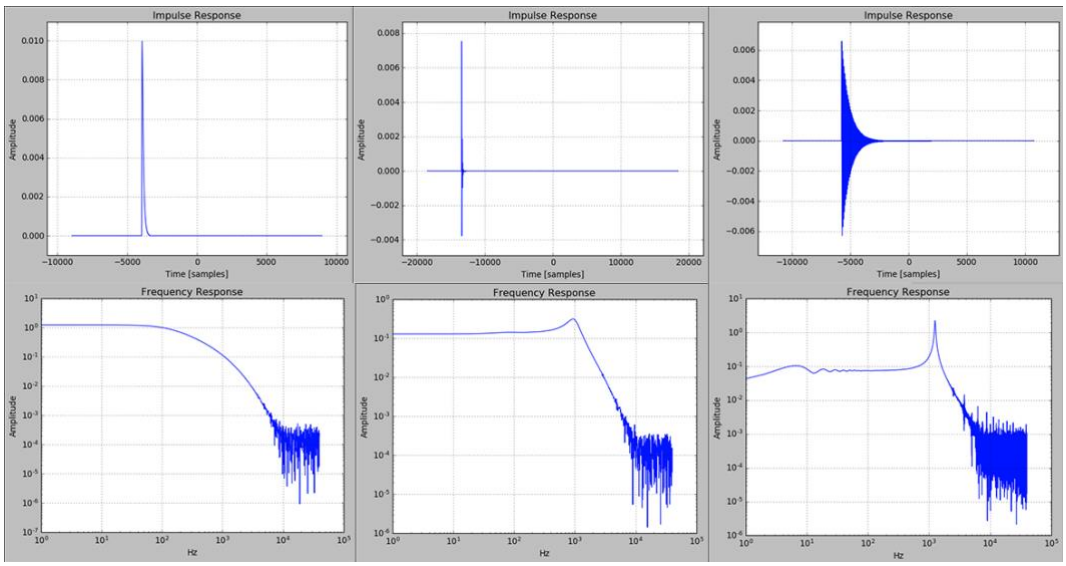

Abbildung 78 Impulse und Frequency Response Messungen der Diode Ladder Filter Implementation in Gen (ohne Oversampling). Links: Cutoff = 1 kHz, Resonance = 0, Mitte: Cutoff = 1 kHz, Resonance = 8, Rechts: Cutoff = 1 kHz, Resonance = 16.

Abbildung 78 zeigt die Messungen der Gen Implementation (ohne Oversampling) wieder bei einer Cutoff Frequenz von 1 kHz, sowie einem Resonanzwert von 0 (links), 8 (mitte) und 16 (rechts). Sofort fällt in der Frequency Response auf, dass Probleme im Abfall des Filters auftreten. Je höher der Resonanzwert ist, desto stärker treten diese Probleme im Frequenzgang auf. Auch die Impulse Response

scheint kürzer zu sein als bei der Python Implementation. Abgesehen davon gleicht die Frequency Response der Gen Implementation der Python Implementation.

Der Filter wird, wie bereits erwähnt, im finalen Max/Msp Patch mit 4x Oversampling betrieben, um Aliasing zu reduzieren. Daher wurden zusätzliche Messungen mit aktiviertem Oversampling durchgeführt:

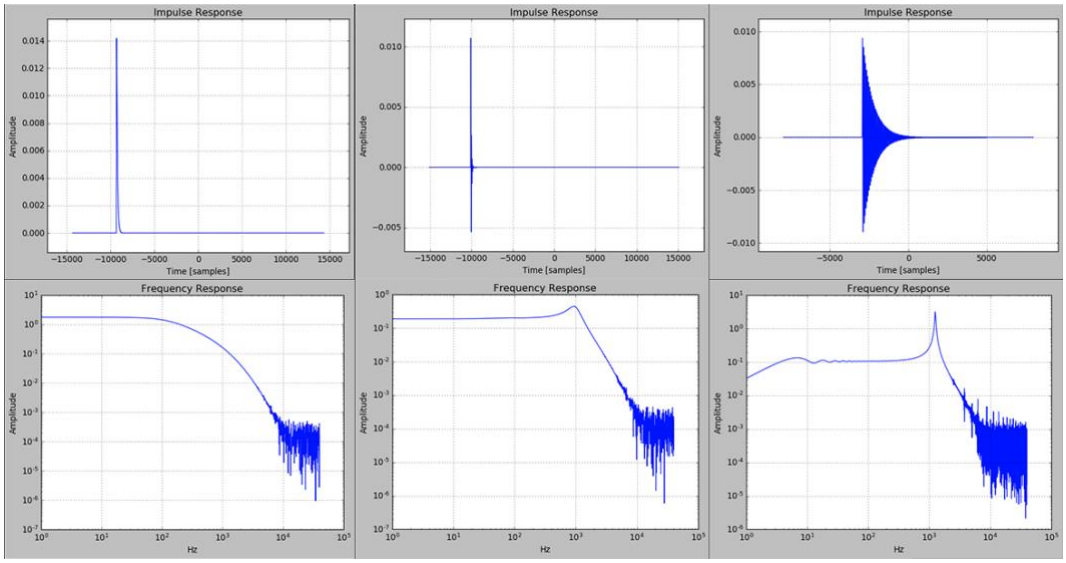

Abbildung 79 Impulse und Frequency Response Messungen der Diode Ladder Filter Implementation in Gen (mit 4x Oversampling). Links: Cutoff = 1 kHz, Resonance = 0, Mitte: Cutoff = 1 kHz, Resonance = 8, Rechts: Cutoff = 1 kHz, Resonance = 16.

Wie in Abbildung 79 zu sehen ist, können die Probleme im Frequenzgang mit aktiviertem Oversampling geringfügig verringert werden, jedoch bei weitem nicht eliminiert werden.

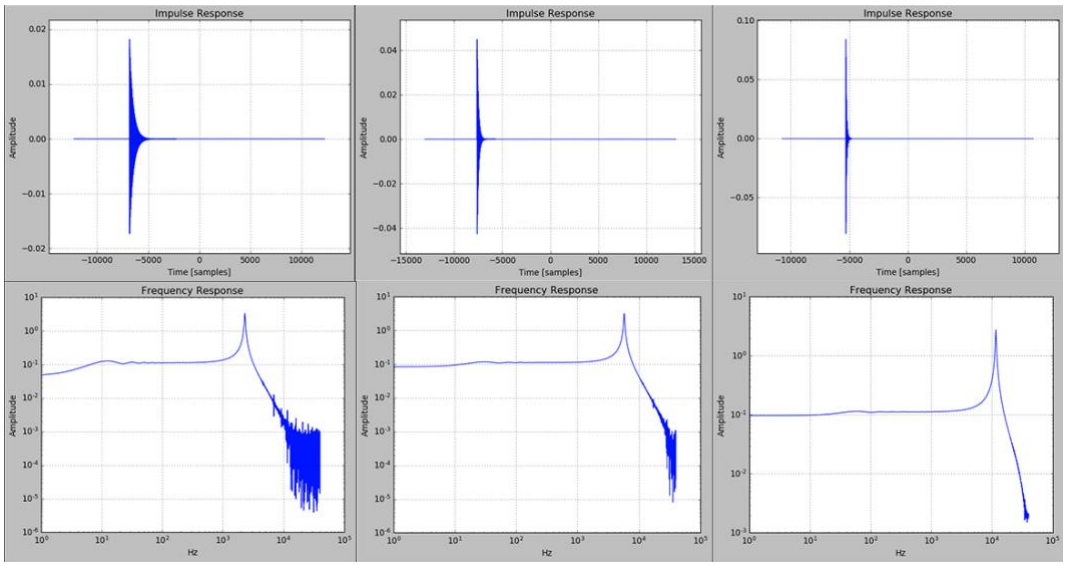

Abbildung 80 Impulse und Frequency Response Messungen der Diode Ladder Filter Implementation in Gen (mit 4x Oversampling). Links: Cutoff = 2 kHz, Resonance = 16, Mitte: Cutoff = 5 kHz, Resonance = 16, Rechts: Cutoff = 10 kHz, Resonance = 16.

Weiters wurde die Gen Implementation des Filters ebenfalls mit den Cutoff Frequenzen 2 kHz (links), 5 kHz (mitte) und 10 kHz (rechts) mit einer Resonanz von 16 gemessen (Abbildung 80). Auch bei diesen Messungen bestehen weiterhin Probleme im Abfall des Filters. Daraus lässt sich schließen, dass es in Gen, und somit in Max/Msp, nicht möglich ist einen solchen Virtual Analog Filter korrekt zu synthetisieren.

5.1.2 Oszillator Messungen

Um zu zeigen wie sehr sich die digital erzeugten Wellenformen denen des analogen Originals ähneln wurde eine Messung durchgeführt. Dabei ist jedoch zu beachten, dass die Messungen der TB-303 am finalen Output des Geräts vorgenommen wurden, da es leider keine Möglichkeit gibt direkt am Output des Oszillators zu messen. Somit hat auch der Filter (dieser wurde auf die höchstmögliche Cutoff-Frequenz sowie die niedrigste Resonanz eingestellt) einen Einfluss auf die gemessene Wellenform.

Zunächst wurde die Sägezahnwellenform gemessen (Abbildung 81):

Abbildung 81 Sägezahnwellenform der TB-303 (links) und die Wellenform des polyBLEP Oszillators (rechts)

Durch die tanh Saturation in der Berechnung der naiven Wellenform (siehe Abschnitt „polyBLEP Sägezahn Oszillator“) konnte die digitale Wellenform (rechts) in eine sehr ähnliche Form wie beim analogen Vorbild (links) gebracht werden. Die ähnlichste Wellenform konnte mit einem sat Wert von 1,5 (in der tanh Funktion) erreicht werden.

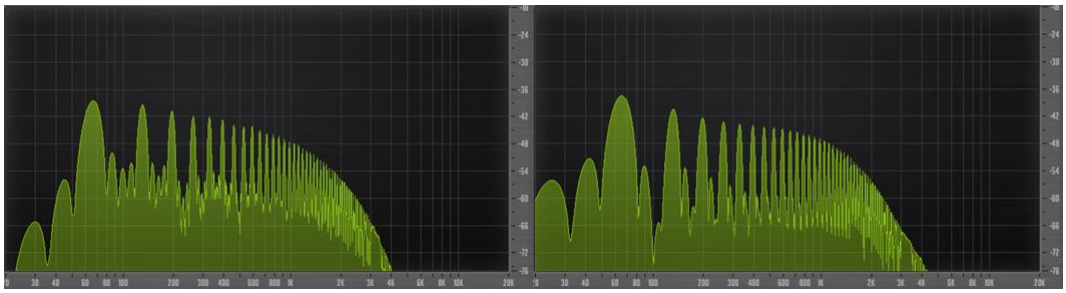

Abbildung 82 Frequenzspektren der TB-303 Sägezahnwelle (links) und der digital emulierten Sägezahnwelle (rechts).

Abbildung 82 zeigt die Frequenzspektren der beiden Sägezahnwellen. Die Spektren ähneln sich sehr stark. Die Sägezahnwelle der TB-303 (links) hat jedoch etwas weniger Energie im Subbass Bereich unter 50 Hz. Außerdem enthält sie teilweise zusätzliche Obertöne, im Spektrum ist dies vor allem zwischen 80 und 150 Hz zu erkennen. Die digital emulierte Sägezahnwelle kommt also recht nah an das analoge Vorbild heran.

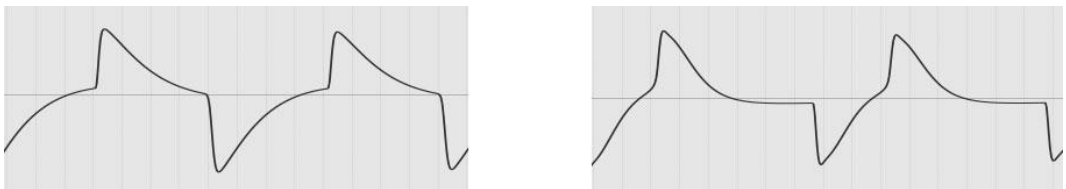

Abbildung 83 Rechteckwellenform der TB-303 (links) und die digital emulierte Wellenform (rechts)

In Abbildung 83 sind die Messungen der Rechteckwelle zu sehen. Durch den Einsatz des Highpass Filter, nachdem die Sägezahnwellenform durch den

Waveshaper zu einer Rechteckwelle gemacht wurde, konnte ebenfalls eine Wellenform erzeugt werden (rechts) die der des analogen Originals (links) sehr nahekommt. Da die Eigenschaften des Waveshapers in der analogen TB-303 nicht bekannt sind, wirft dies die Vermutung auf das möglicherweise auch ein Highpass Filter Teil des mysteriösen Waveshapers ist.

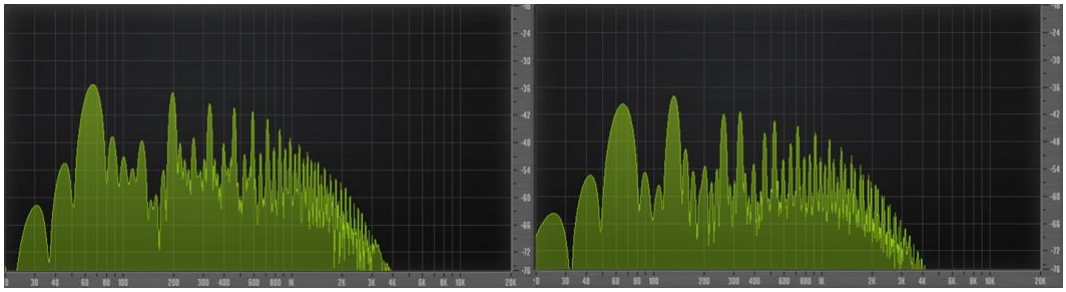

Abbildung 84 Frequenzspektren der TB-303 Rechteckwelle (links) und der digital emulierten Rechteckwelle (rechts).

Abbildung 84 zeigt die Frequenzspektren der beiden Rechteckwellen. Hier ist bei der digitalen Emulation (rechts) vor allem der ausgeprägtere Oberton bei etwa 150 Hz zu erkennen, dafür enthält die die Rechteckwelle der TB-303 bei 200 Hz einen deutlich stärkeren Oberton als die Emulation. Auch bei der Rechteckwelle lässt sich etwas mehr Energie im Subbass Bereich unter 50 Hz feststellen. Die digital erzeugte Rechteckwelle kommt der des analogen Vorbilds ebenfalls recht nah. Die vorher erwähnten abweichenden Obertöne im Frequenzspektrum, sowie die etwas weitere Pulsweite der Wellenform (Abbildung 83), machen jedoch einen eindeutig hörbaren klanglichen Unterschied zwischen den analogen und digitalen Wellenformen aus.

5.2 Hörtest

Um eine Aussage zu den Forschungsfragen wie nah die Software Emulation klanglich an die analoge TB-303 herankommt, und wie gut die eingesetzten Virtual Analog Modelle das analoge Original emulieren können, treffen zu können hat ein Hörtest stattgefunden. Der Hörtest wurde in Form eines Online Fragebogens durchgeführt. Es wurde auf eine qualitative Forschung gesetzt, indem es sich bei allen ProbandInnen um Musik Produzenten (wovon ein großer Teil bereits oft mit der TB-303 gearbeitet hat), sowie audiotechnisch versierte Menschen handelt.

5.2.1 Testaufbau

Der Hörtest besteht zunächst aus 5 verschiedenen Audiobeispielen, bei denen der Klang der Software Emulation (MK303) mit dem Klang der analogen TB-303 verglichen wird. Die Beispiele wurden so gewählt, dass ein möglichst breites Spektrum der verschiedenen möglichen Klänge der TB-303 abgedeckt wurde. Für die Beispiele wurden verschiedene Patterns und verschiedene Parametereinstellungen gewählt:

1. Sägezahnwelle, Pattern ohne Accents und Slides, mittlere Cutoff Frequenz, Resonanz und Envelope Modulation, langer Decay.
2. Sägezahnwelle, Pattern ohne Accents und Slides, tiefere Cutoff Frequenz, mittlere Resonanz und niedrige Envelope Modulation, langer Decay.
3. Sägezahnwelle, Pattern mit Accents und Slides, mittlere Cutoff Frequenz, Resonanz und Envelope Modulation, langer Decay.
4. Sägezahnwelle, Pattern mit Accents und Slides, hohe Cutoff Frequenz, Resonanz und Envelope Modulation, kurzer Decay.
5. Rechteckwelle, Pattern ohne Accents und Slides, mittlere Cutoff Frequenz, Resonanz, Envelope Modulation und Decay.

Abbildung 85 Erstes der fünf Beispiele des Hörtest.

Abbildung 85 zeigt den Aufbau des Onlinetests anhand des ersten Audiobeispiels. Die ProbandInnen bekamen die Audiobeispiele in einem Video zu hören. Die Audiobeispiele werden mehrmals abwechselnd nacheinander abgespielt, mit einer Anzeige ob gerade die TB-303 oder die Emulation zu hören ist. Die ProbandInnen sollten die Ähnlichkeit des Klangs zwischen den beiden Beispielen anhand einer Skala von 0 bis 10 bewerten. Wobei 0 für überhaupt keine Ähnlichkeit steht und 10 ausgewählt werden soll, wenn beide Beispiele exakt gleich klingen. Die ProbandInnen wurden zu Beginn des Tests über das Bewertungsschema aufgeklärt, sowie das unbedingt gute Kopfhörer oder Studio Monitore für den Test verwendet werden sollen.

Nachdem die fünf Beispiele bewertet wurden, war es den ProbandInnen möglich ihren Eindruck zum Vergleich in einem Freitextfeld auch noch in Worten ausdrücken. Das Textfeld war jedoch optional.

Um zu erfahren ob auch blind ein Unterschied zwischen der analogen TB-303 und der digitalen Emulation festzustellen ist, beziehungsweise um zu testen ob der analoge Sound von den ProbandInnen erkannt werden kann, gab es danach noch zwei Beispiele als Blind Test zu hören. Hier sollten die ProbandInnen herausfinden welches der beiden zu hörenden Beispiele die analoge TB-303 ist, und welches die Emulation ist. Wie bei den ersten fünf Beispielen sind hier wieder abwechselnd die TB-303 und die Emulation zu hören, jedoch sind diese lediglich mit 1 und 2 gekennzeichnet. Die ProbandInnen sollten auswählen ob 1 oder 2 die analoge TB-303 ist. (siehe Abbildung 86)

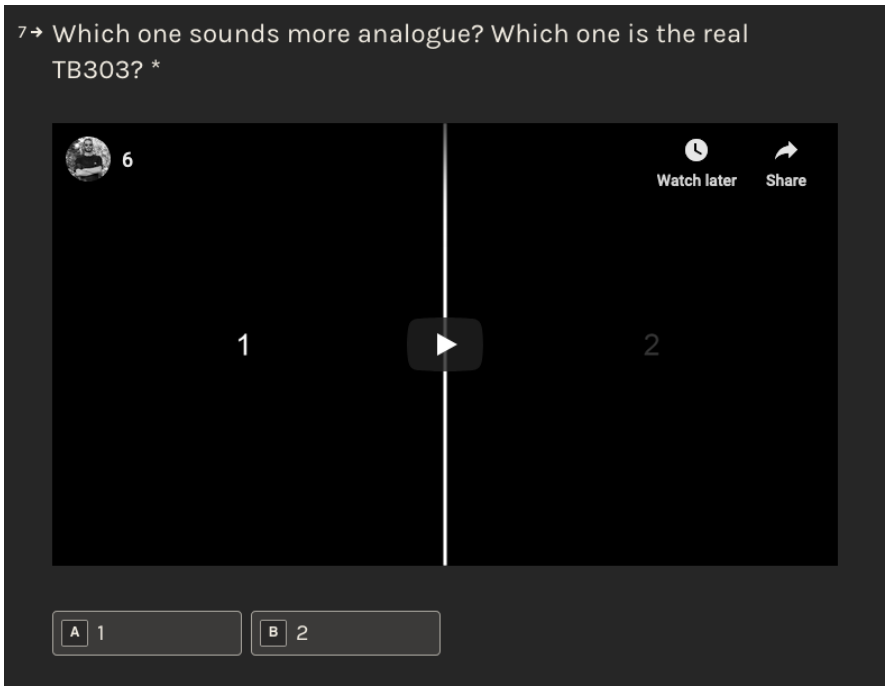

Abbildung 86 Blindtest am Ende des Hörtest.

Die Durchführung des Hörtest dauerte etwa 4 bis 5 Minuten. Der Test wurde bewusst möglichst kurzgehalten, um einer Ermüdung der Ohren bei den ProbandInnen (die das Testergebnis verfälschen könnte) entgegenzuwirken.

5.2.2 Testergebnisse

Insgesamt haben 51 ProbandInnen am Hörtest teilgenommen. Folgend werden die Befragungsergebnisse der einzelnen Audiobeispiele sowie des Blindtests gezeigt und interpretiert.

Audiobeispiel 1

Sägezahnwelle, Pattern ohne Accents und Slides, mittlere Cutoff Frequenz, Resonanz und Envelope Modulation, langer Decay.

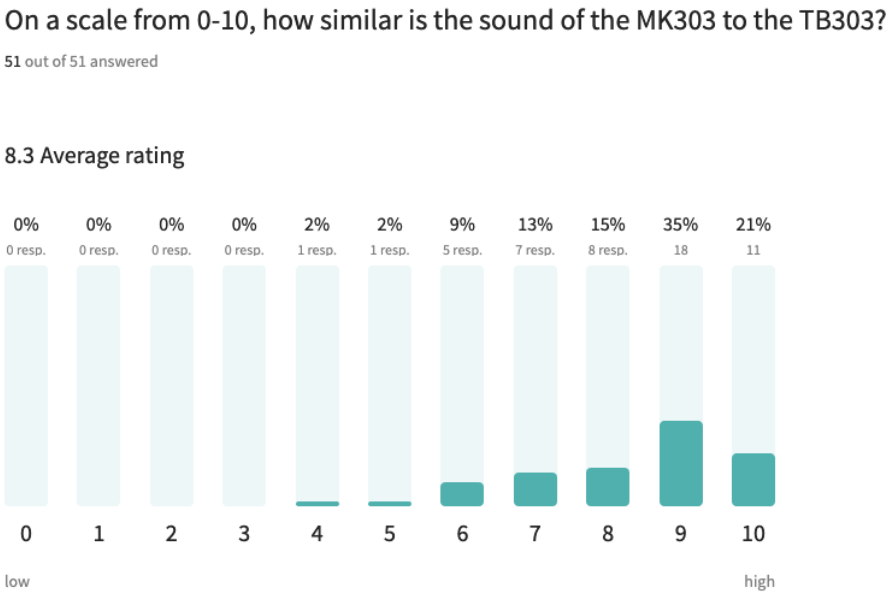

Abbildung 87 Befragungsergebnis des 1. Audiobeispiel.

Der größte Teil der ProbandInnen hat die Ähnlichkeit der digitalen Emulation mit der analogen TB-303 beim 1. Audiobeispiel mit 9 (35%, 18 ProbandInnen) und 10 (21%, 11 ProbandInnen) bewertet (Abbildung 87). Im Durchschnitt wurde das Beispiel mit 8,3 bewertet. Daraus lässt sich schließen, dass der Klang der Emulation in diesem Beispiel sehr nah an den analogen Klang der TB-303 herankommt.

Audiobeispiel 2

Sägezahnwelle, Pattern ohne Accents und Slides, tiefere Cutoff Frequenz, mittlere Resonanz und niedrige Envelope Modulation, langer Decay.

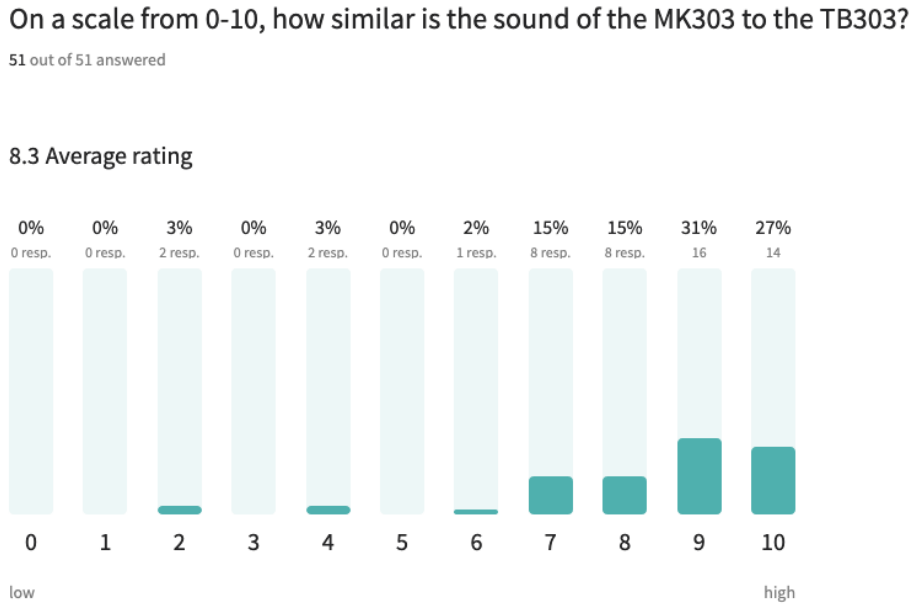

Abbildung 88 Befragungsergebnis des 2. Audiobeispiel.

Das Befragungsergebnis des 2. Audiobeispiels ähnelt dem des 1. Beispiels. Die meisten ProbandInnen bewerteten die Ähnlichkeit mit 9 (31%, 16 ProbandInnen) und 10 (27%, 14 ProbandInnen) (Abbildung 88). Im Durchschnitt wurde das Beispiel, wie auch Beispiel 1, mit 8,3 bewertet. Laut diesen Ergebnissen ist auch das 2. Audiobeispiel sehr nah am Klang der analogen TB-303.

Audiobeispiel 3

Sägezahnwelle, Pattern mit Accents und Slides, mittlere Cutoff Frequenz, Resonanz und Envelope Modulation, langer Decay.

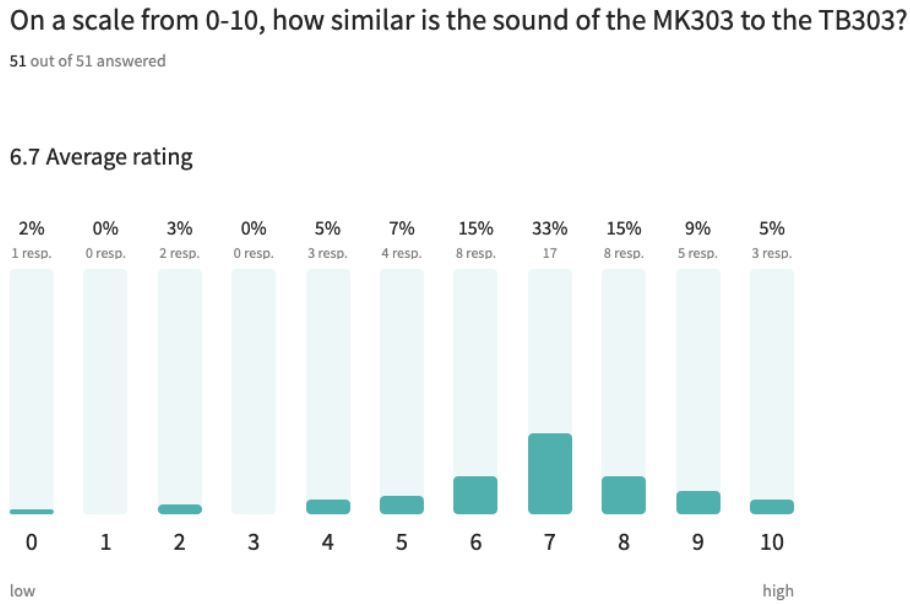

Abbildung 89 Befragungsergebnis des 3. Audiobeispiel.

Das 3. Audiobeispiel haben die meisten ProbandInnen mit 7 (33%, 17 ProbandInnen) sowie 6 und 8 (jeweils 15%, 8 ProbandInnen) bewertet (Abbildung 89). Der Durchschnitt der Antworten beträgt 6,7. Der Klang der Emulation hat bei diesem Beispiel also weniger Ähnlichkeit mit dem Klang der TB-303 als bei den ersten beiden Beispielen. Da in diesem Beispiel (im Gegensatz zu den vorangegangenen Beispielen) ein Pattern mit Accents und Slides verwendet wurde, lässt sich daraus schließen, dass die Implementation dieser beiden Features in der Emulation klanglich nicht das gleiche Ergebnis bringt wie bei der analogen TB-303.

Audiobeispiel 4

Sägezahnwelle, Pattern mit Accents und Slides, hohe Cutoff Frequenz, Resonanz und Envelope Modulation, kurzer Decay.

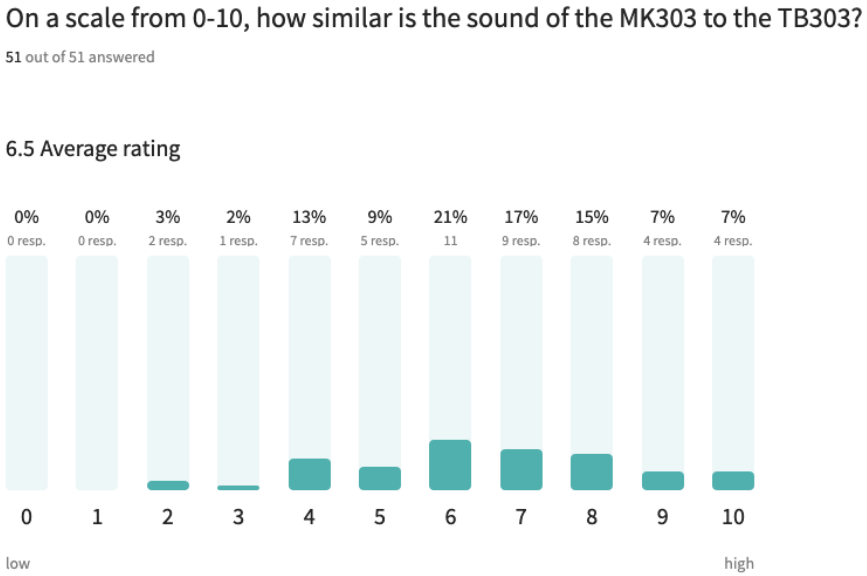

Abbildung 90 Befragungsergebnis des 4. Audiobeispiel.

Beim 4. Audiobeispiel ist das Befragungsergebnis weniger eindeutig als bei den vorangegangen Beispielen. Die meisten ProbandInnen haben dieses Beispiel mit 6 (21%, 11 ProbandInnen), 7 (17%, 9 ProbandInnen) und 8 (15%, 8 ProbandInnen) bewertet (Abbildung 90). Der Durchschnitt liegt bei 6,5. Einerseits zeigt dies, wie schon in Beispiel 3, dass die Accent und Glide Features der Emulation nicht denselben Klang ermöglichen wie bei der analogen TB-303. Außerdem scheint die Emulation des Filters bei einer Kombination aus hoher Cutoff Frequenz und Resonanz nicht exakt das gewünschte Ergebnis zu bringen. Trotzdem ähnelt der Klang der Emulation bei diesem Beispiel laut der Befragungsergebnisse der analogen TB-303 bis zu einem gewissen Grad.

Audiobeispiel 5

Rechteckwelle, Pattern ohne Accents und Slides, mittlere Cutoff Frequenz, Resonanz, Envelope Modulation und Decay.

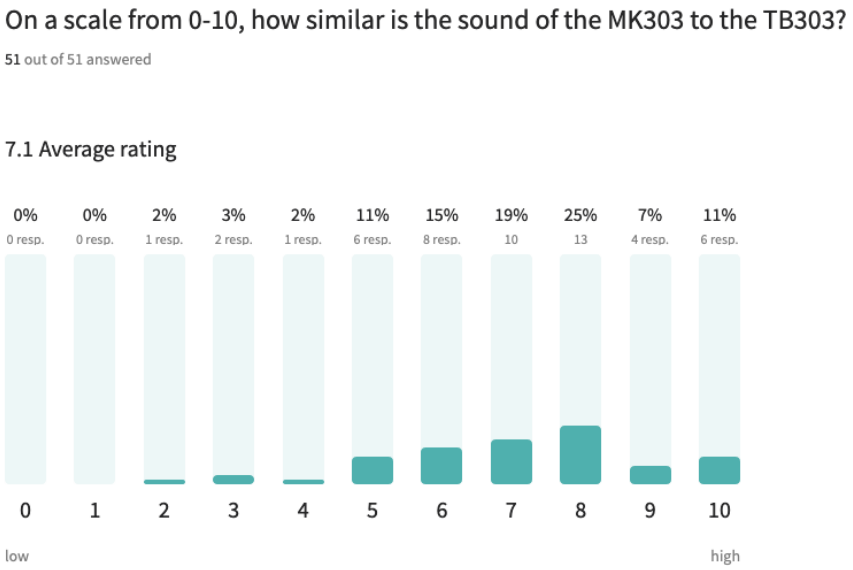

Abbildung 91 Befragungsergebnis des 5. Audiobeispiel.

Das letzte der 5 Audiobeispielen wurde von den meisten ProbandInnen mit 8 (25%, 13 ProbandInnen) und 7 (19%, 10 ProbandInnen) bewertet (Abbildung 91). Im Durchschnitt wurde das 5. Beispiel mit 7,1 bewertet. Bei diesem Beispiel lag der Fokus am Klang der Rechteckwelle. Diese scheint laut den Befragungsergebnis nah an den Klang der Rechteckwelle der analogen TB-303 heranzukommen, jedoch ist weiterhin ein Unterschied zu hören.

Blindtest 1

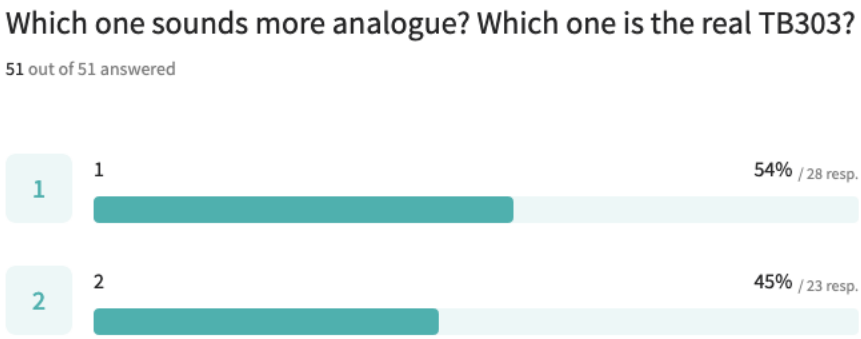

Abbildung 92 Befragungsergebnis des 1. Blindtest.

Beim ersten der beiden Blindtests konnten 54% (28 ProbandInnen) der ProbandInnen die analoge TB-303 erkennen (Abbildung 92). 45% (23 ProbandInnen) glaubten die Emulation sei die TB-303. Somit konnten lediglich etwas mehr als die Hälfte der ProbandInnen den analogen Sound von der digitalen Emulation unterscheiden.

Blindtest 2

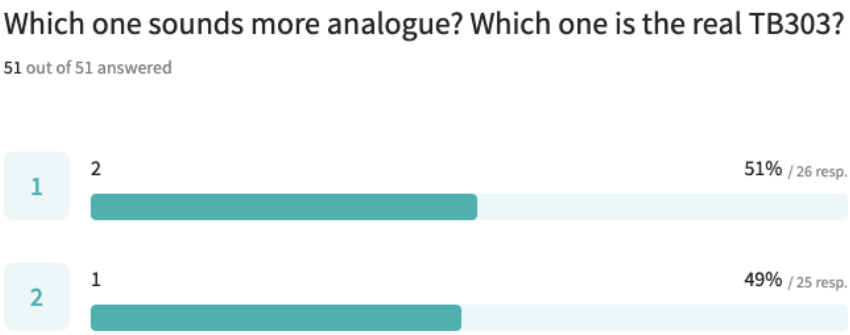

Abbildung 93 Befragungsergebnis des 2. Blindtest.

Beim zweiten Blindtest konnten nur 51% (26 ProbandInnen) der ProbandInnen die analoge TB-303 erkennen (Abbildung 93). 49% (25 ProbandInnen) hielten die Emulation für die analoge TB-303. Es konnten somit nur die Hälfte der ProbandInnen den analogen Sound von der digitalen Emulation unterscheiden.

Auch das optionale Freitextfeld wurde von einigen ProbandInnen genutzt, und beinhaltet einige interessante Erkenntnisse:

Einige ProbandInnen gaben zusätzlich an, dass für sie der Klang der Emulation insgesamt sehr nah am analogen Vorbild liegt. Mehrere ProbandInnen gaben an, dass vor allem die Slide Funktion der Emulation sich von der analogen TB-303 unterscheidet. Weiters gaben mehrere ProbandInnen an, dass die Tiefen Frequenzen bei der Emulation etwas voller wirken, und bei der TB-303 die hohen Frequenzen etwas anders wirken. Vor allem bei hohen Resonanzeinstellungen konnten mehrere ProbandInnen bei der analogen TB-303 mehr Bewegung in den hohen Frequenzen feststellen, wohingegen die Emulation etwas statischer wirkte. Jedoch gaben einige ProbandInnen an, dass vor allem die Mittleren Frequenzen annähernd gleich wie die der TB-303 klingen. Zwei ProbandInnen gaben sogar an, dass ihnen der Klang der Emulation besser gefiel als die analoge TB-303.

Zusammengefasst lassen sich die Befragungsergebnisse wie folgt interpretieren:

Für die meisten ProbandInnen war der Klang der Emulation vor allem ohne Accents und Slides, sowie mit niedrigen bis mittleren Cutoff Frequenz und Resonanz Einstellungen sehr ähnlich dem Klang der analogen TB-303. Mit Accents und Slides, und vor allem bei höheren Resonanz Einstellungen konnte ein größerer Unterschied zwischen den beiden festgestellt werden. Daraus lässt sich schließen, dass einerseits das digitale Diode Ladder Filter Modell bei hohen

Cutoff und Resonanz Einstellungen nicht wie gewünscht funktioniert, und andererseits, dass die Implementationen der Slide und Accent Funktionen nicht ganz das gewollte Ergebnis bringen. Auch die feinen Unterschiede zwischen der emulierten und der analogen Sägezahnwellenform könnten hierbei eine Rolle spielen. Insgesamt kommt der Klang der Emulation dem Klang der analogen TB-303 aber relativ nahe. Fast die Hälfte der ProbandInnen konnte die Emulation im Blindtest nicht von der TB-303 unterscheiden.

6 Fazit

In diesem Kapitel werden die Erkenntnisse der Arbeit zusammengefasst und die zu Beginn in der Einleitung aufgestellten Forschungsfragen beantwortet.

Ein möglicher Weg, wie die TB-303 in Max/Msp emuliert werden kann, und welche Strategien dabei angewendet wurden, wurde in Kapitel 4 („TB-303 Emulations Projekt“) ausführlich beschrieben. Es ist dabei ein sehr gut spielbares Max4live Instrument entstanden.

Im Laufe der Arbeit ist schnell klar geworden, dass nicht nur der Diode Ladder Filter für den charakteristischen Klang der TB-303 verantwortlich ist, sondern das Zusammenspiel zwischen allen Bestandteilen (vor allem Oszillator und Filter Envelope Modulation) sehr wichtig ist.

Klanglich ist die Software Emulation, laut den Ergebnissen des Hörtest, insgesamt recht nah am Klang der analogen Hardware dran. Es konnten trotzdem von den ProbandInnen während des Vergleichs einige Unterschiede bezüglich des Klanges festgestellt werden:

- Bei hohen Cutoff Frequenz und Resonanz Einstellungen konnten die größten klanglichen Unterschiede festgestellt werden.
- Vor allem bei hohen Resonanz Einstellungen konnte bei der analogen Hardware etwas mehr Bewegung in den hohen Frequenzen festgestellt werden. Die Software Emulation wirkt etwas statischer.
- Die tiefen Frequenzen wirken bei der Software Emulation etwas ausgeprägter.
- Die Implementation der Accent und Slide Funktionen in der Software Emulation bringen nicht dasselbe klangliche Ergebnis wie die analoge Hardware.
- Die emulierte Rechteckwelle kommt dem Klang der analogen Hardware zwar nahe, lässt sich aber recht eindeutig unterscheiden.

Die Messungen des Oszillators zeigten, dass die emulierte Sägezahnwelle der analogen sehr ähnlich ist. Bei der emulierten Rechteckwelle konnten jedoch auch bei den Messungen größere Unterschiede erkannt werden.

Die Messungen des Filters zeigten, dass die Gen~ bzw. Max/Msp Implementation Probleme im Abfall des Filters aufweist. In der Python Implementation treten diese Probleme jedoch nicht auf. Vor allem aus diesem Grund lässt sich sagen, dass Max/Msp nicht die ideale Entwicklungsumgebung für eine solche Virtual Analog Emulation ist, und es daher nicht möglich ist den exakten originalgetreuen Klang der analogen Hardware in Max/Msp zu realisieren. Der Hörtest zeigt jedoch, dass die verwendeten Virtual Analog Modelle, auch trotz der Probleme in Max/Msp, klanglich nah an das analoge Vorbild herankommen. Vor allem im Blindtest konnte nur etwas mehr als die Hälfte der ProbandInnen die analoge TB-303 von der Software Emulation unterscheiden. Außerdem findet sich im Internet auch keine andere in Max/Msp realisierte TB-303 Emulation, deren Klang so nah am Original liegt.

Die aus dieser Arbeit resultierende Max/Msp Emulation würde sich daher als ein sehr guter Ausgangspunkt oder Prototyp für eine Implementation in z.B. C++ eignen, um ein VST Plugin zu erstellen. Hierbei könnten die Probleme die der Diode Ladder Filters in Max/Msp mit sich bringt vermieden werden. Weiters müsste vor allem an der Implementation der Rechteckwelle und der Accent und Slide Funktionen gearbeitet werden. Ich glaube zwar, dass auf diesen Weg eine noch bessere Emulation entstehen kann, jedoch halte ich es für unwahrscheinlich, dass eine verbesserte Software Emulation exakt gleich wie die analoge Hardware klingen wird. Dies ist auch der billigen Bauweise und des hohen Alters (Produktion wurde 1984 eingestellt) der originalen TB-303 geschuldet. Kaum eine TB-303 klingt exakt gleich wie eine Zweite.

Literaturverzeichnis

Brandt, E. (2001). Hard Sync without Aliasing

Brinkmann, H. (2018). Blue Box: Roland TB-303 - The whole Story! Retrieved April 21, 2020, from https://www.amazona.de/blue-box-roland-tb-303-the-whole-story/

Dodge, C., Jerse, T. (1997). Computer Music.

Electronic Music Wiki. Retrieved April 21, 2020, from https://electronicmusic.fandom.com/wiki/Diode_ladder

Frei, B. (2019). Digital Sound Generation

Heermanns, A. (2016). Rhetorical Synthesis: The Story of the TB-303. Retrieved April 21, 2020, from https://www.dwrl.utexas.edu/2016/11/22/rhetorical-synthesis-the-story-of-the-roland-tb-303/

Kane, P. (2019). All About Digital Oscillators Part 1. Retrieved April 21, 2020, from http://metafunction.co.uk/all-about-digital-oscillators-part-1-aliasing-foldover/

Kane, P. (2019). All About Digital Oscillators Part 2. Retrieved April 21, 2020, from http://metafunction.co.uk/all-about-digital-oscillators-part-2-blits-bleps/

Karras, D. (2018). Sound Synthesis Theory.

Leary, A., Bright, C. (2009). Band limited digital synthesis of analog waveforms.

Nielsen, K. (2000). Digital Filters for Music Synthesis.

Nieswandt, H. (2007). plus minus acht. DJ Tage DJ Nächte.

Pirkle, W. (2013). Virtual Analog (VA) Diode Ladder Filter.

Pirkle, W. (2013). Virtual Analog (VA) FIlter Implementations and Comparisons.

Pirkle, W. (2014). Designing Software Synthesizer Plug-Ins in C++ For RackAFX, VST3, and Audio Units.

Smith, J. (2013). Making Virtual Electric Guitars and Associated Effects Using Faust

Stilson, T., Smith, J. (1999). Alias-Free Digital Synthesis of Classic Analog Waveforms

Välimäki, V., Pekonen, J., Nam, J. (2010). Perceptually informed synthesis of band limited classical waveforms using integrated polynomial interpolation.

Van Dijk, M. (2019). Eine Liebeserklärung an die Roland TB-303. Retrieved April 21, 2020, from https://www.bonedo.de/artikel/einzelansicht/eine-liebeserklaerung-an-die-roland-tb-303.html

Wegscheider, N. (2008). TB-303 und Acid House. Die Verfügbarkeit von Technologie und der Implikationen für das Aufkommen der Electronic Dance-Music

Whittle, R. (1999). Investigating some unique aspects of TB-303's sound. Retrieved April 21, 2020, from http://www.firstpr.com.au/rwi/dfish/303-unique.html

Whittle, R. (1999). Investigating the Slide function of the TB-303. Retrieved April 21, 2020, from http://www.firstpr.com.au/rwi/dfish/303-slide.html

Zavalishin, V. (2015). The Art Of VA Filter Design.

Zwicker, E., Feldtkeller, R. (1967). Das Ohr als Nachrichtenempfänger.

Abbildungsverzeichnis

Abbildung 1 Oscar Peterson (Jazzmusiker) mit Roland PianoPlus, TB-303 und TR-606. Das Bild stammt aus einer alten Roland Broschüre von 1982. Dieses Bild zeigt sehr gut, wofür Roland die TB-303 und TR-606 entwickelt hat, und als was die beiden vermarktet wurden..... 5

Abbildung 2 Roland TB-303..... 8

Abbildung 3 TB-303 Block Diagramm aus dem Service Manual..... 9

Abbildung 4 Vereinfachte Darstellung des Signal Flow der TB-303..... 10

Abbildung 5 Sägezahnwellenform der TB-303 (drei cycles)..... 11

Abbildung 6 Rechteckwellenform der TB-303 (drei cycles)..... 11

Abbildung 7 VCS-3 Synthesizer der Firma EMS. Einer der ersten Synthesizer mit Diode Ladder Filter..... 12

Abbildung 8 Simples Block Diagramm des Moog Transistor Ladder Filter..... 13

Abbildung 9 Simples Block Diagramm des Diode Ladder Filter..... 14

Abbildung 10 Filter Cutoff Modulation bei einer einzelnen accented Note..... 16

Abbildung 11 Filter Cutoff Modulation bei mehreren accented Noten in Folge... 16

Abbildung 12 Verhalten des Sequencers über die Dauer von drei Sechzehntelnoten, ohne Slide Noten..... 17

Abbildung 13 Verhalten des Sequencers über die Dauer von drei Sechzehntelnoten, inklusive einer Slide Note (B)..... 17

Abbildung 14 Sine, Square, Triangle und Sawtooth Wellenformen..... 20

Abbildung 15 Topologie eines generischen digitalen Oszillator..... 21

Abbildung 16 Eine Periode einer Sinuswelle mit Phase 2π (Radiant)..... 22

Abbildung 17 Eine Periode einer Rechteckwelle mit Phase 2π (Radiant)..... 24

Abbildung 18 Eine Periode einer Sägezahnwelle mit Phase 2π (Radiant)..... 25

Abbildung 19 Eine Periode einer Dreieckswelle mit Phase 2π (Radiant)..... 26

Abbildung 20 Zeitkontinuierliche Sägezahnwelle ($f0$ = 5kHz), mit fs = 48kHz abgetastet. 28

Abbildung 21 Hörschwelle für leisere Töne um ein lauteres schmalbandiges Signal bei 1 kHz. 29

Abbildung 22 Spektrum eines zeitkontinuierlichen Prototypsignal und seine zeitdiskrete Repräsentation mit Aliasing Komponenten 1. und 2. Ordnung (Rot, Orange). 30

Abbildung 23 Darstellung einer naiven, nicht bandbegrenzten Rechteckwelle im Zeitbereich 31

Abbildung 24 Darstellung einer bandbegrenzten Variante der Rechteckwelle im Zeitbereich 31

Abbildung 25 Wellenformen und Frequenzspektren von naiven Sägezahn (oben), Rechteck (mitte) und Dreieck (unten) Implementationen. f = 440 Hz 32

Abbildung 26 Bandbegrenzte Impulse Train, unipolar (a) und bipolar (b). 35

Abbildung 27 Blockdiagramme der BLIT Algorithmen für die Erzeugung von Sägezahn (oben), Rechteck (mitte) und Dreieckswellen (unten). 35

Abbildung 28 BLEP Funktion, die aus dem integrieren eines bandbegrenzten Impuls entsteht. 36

Abbildung 29 Die Impuls Response eines idealen Lowpass Filter (links) und das in bipolare Form konvertierte Step Signal das aus der Integration der Impulse Response entsteht (rechts). 37

Abbildung 30 Ein unipolarer Dreiecksimpuls der statt sinc() verwendet wird (a), und das Ergebnis der Integration in bipolare Form konvertiert (b) 38

Abbildung 31 Simples Filter Block Diagramm. 39

Abbildung 32 Block Diagramm eines Biquad Filters. 40

Abbildung 33 Block Diagramm eines One Pole Lowpass Filters. 41

Abbildung 34 tanh Funktion. 43

Abbildung 35 Normalisierte tanh Funktion. 43

Abbildung 36 Normalisierte tanh Funktion mit sat = 3 44

Abbildung 37 Cubic Nonlinear Distortion Funktion 45

Abbildung 38 Blockdiagramm des linearen Diode Ladder Filter 47

Abbildung 39 Isolierte Diode Ladder Struktur, ohne globalen Feedback Loop ... 48

Abbildung 40 LPF3/LPF4 Filter Paar 51

Abbildung 41 Input und Feedback Summierung 52

Abbildung 42 modifizierter TPT Lowpass Filter mit den benötigten Feedback Inputs und Outputs 53

Abbildung 43 LPF3/LPF4 Filter Paar mit allen Feedback Paths 53

Abbildung 44 Die komplette Diode Ladder (ohne globalen Feedback Pfad) 54

Abbildung 45 Vollständiges Block Diagramm des Diode Ladder Filter (inklusive globalen Feedback Path) 56

Abbildung 46 Simple nonlinear Model mit einem NLP Block am Input in die Ladder 57

Abbildung 47 Simple nonlinear Model mit einem NLP Block im globalen Feedback Pfad 57

Abbildung 48 Budget NLP Version des Diode Ladder Filters mit NLP Block am Input 58

Abbildung 49 1-Pole Lowpass Filter Stufe mit NLP Blöcken am Input sowie im lokalen Feedback Pfad 58

Abbildung 50 Advanced nonlinear Model (ohne globalen Feedback Pfad) 59

Abbildung 51 History Operator in Gen 68

Abbildung 52 Diode Ladder Filter gen~ Implementation innerhalb des übergeordneten Synthesizer Max Patch 72

Abbildung 53 FilterEnvelope Subpatch - Filter Envelope Generator 73

Abbildung 54 FilterEnvMess Subpatch – Modulierung der Cutoff Frequenz 75

Abbildung 55 FilterEnvMess Subpatch im übergeordneten Patch 76

Abbildung 56 VolEnvelope Subpatch - Volume Envelope Generator 77

Abbildung 57 Volume Envelope Modulation im übergeordneten Max Patch. 78

Abbildung 58 Osc Subpatch. 78

Abbildung 59 Erzeugung der Rechteckwellenform im Osc Subpatch. 80

Abbildung 60 Output des Waveshapers. 81

Abbildung 61 biquad~ Filter. 81

Abbildung 62 selector~ Objekt für die Auswahl der gewünschten Wellenform. .. 82

Abbildung 63 accent? Subpatch für die Unterscheidung zwischen normalen und accented Noten. 83

Abbildung 64 accent? und accentSweep Subpatches im übergeordneten Patch. 84

Abbildung 65 accentSweep Subpatch. 84

Abbildung 66 Erhöhung der Cutoff Frequenz bei einer accented Note im übergeordneten Max Patch. 85

Abbildung 67 Zweiter Envelope Generator für accented Noten im FilterEnvelope Subpatch. 86

Abbildung 68 selector~ Objekt für die Auswahl des gewünschten Filter Envelope. 87

Abbildung 69 Verstärkung von accented Noten. 87

Abbildung 70 Midiin und glide Subpatch im übergeordneten Patch. 88

Abbildung 71 glide Subpatch. 89

Abbildung 72 GUI der MK303 Software Emulation. 90

Abbildung 73 Nichtlinearität (Drive) am finalen Output des Synthesizers mithilfe eines tanh~ Objekts. 91

Abbildung 74 Impulse und Frequency Response Messungen der Diode Ladder Filter Implementation in Python. Links: Cutoff = 1 kHz, Resonance = 0, Mitte: Cutoff = 1 kHz, Resonance = 8, Rechts: Cutoff = 1 kHz, Resonance = 16.. 93

Abbildung 75 Impulse und Frequency Response Messungen der Diode Ladder Filter Implementation in Python. Links: Cutoff = 2 kHz, Resonance = 16, Mitte: Cutoff = 5 kHz, Resonance = 16, Rechts: Cutoff = 10 kHz, Resonance = 16. 94

Abbildung 76 Impulse und Frequency Response Messungen der Diode Ladder Filter Implementation in Python. Cutoff = 15 kHz, Resonance = 16. 94

Abbildung 77 Impulse und Frequency Response Messungen der Diode Ladder Filter Implementation in Python. Cutoff = 18 kHz, Resonance = 15. 95

Abbildung 78 Impulse und Frequency Response Messungen der Diode Ladder Filter Implementation in Gen (ohne Oversampling). Links: Cutoff = 1 kHz, Resonance = 0, Mitte: Cutoff = 1 kHz, Resonance = 8, Rechts: Cutoff = 1 kHz, Resonance = 16. 95

Abbildung 79 Impulse und Frequency Response Messungen der Diode Ladder Filter Implementation in Gen (mit 4x Oversampling). Links: Cutoff = 1 kHz, Resonance = 0, Mitte: Cutoff = 1 kHz, Resonance = 8, Rechts: Cutoff = 1 kHz, Resonance = 16. 96

Abbildung 80 Impulse und Frequency Response Messungen der Diode Ladder Filter Implementation in Gen (mit 4x Oversampling). Links: Cutoff = 2 kHz, Resonance = 16, Mitte: Cutoff = 5 kHz, Resonance = 16, Rechts: Cutoff = 10 kHz, Resonance = 16. 97

Abbildung 81 Sägezahnwellenform der TB-303 (links) und die Wellenform des polyBLEP Oszillators (rechts)..... 98

Abbildung 82 Frequenzspektren der TB-303 Sägezahnwelle (links) und der digital emulierten Sägezahnwelle (rechts). 98

Abbildung 83 Rechteckwellenform der TB-303 (links) und die digital emulierte Wellenform (rechts) 98

Abbildung 84 Frequenzspektren der TB-303 Rechteckwelle (links) und der digital emulierten Rechteckwelle (rechts). 99

Abbildung 85 Erstes der fünf Beispiele des Hörtests. 100

Abbildung 86 Blindtest am Ende des Hörtests. 102

Abbildung 87 Befragungsergebnis des 1. Audiobeispiel..... 103

Abbildung 88 Befragungsergebnis des 2. Audiobeispiel..... 104

Abbildung 89 Befragungsergebnis des 3. Audiobeispiel..... 105

Abbildung 90 Befragungsergebnis des 4. Audiobeispiel..... 106

Abbildung 91 Befragungsergebnis des 5. Audiobeispiel..... 107

Abbildung 92 Befragungsergebnis des 1. Blindtest. 107

Abbildung 93 Befragungsergebnis des 2. Blindtest. 108

Listingverzeichnis

Listing 1. Naiver Sinus Oszillator Algorithmus. 23

Listing 2. Naiver Rechteck Oszillator Algorithmus. 24

Listing 3. Naiver Sägezahn Oszillator Algorithmus. 25

Listing 4. Naiver Dreieck Oszillator Algorithmus. 27

Listing 5. LP Klasse 60

Listing 6. Initialisierung der 4 LPF Objekte 61

Listing 7. Setzung der Parameter 61

Listing 4. Berechnung von GN und Gamma 61

Listing 9. Berechnung der Koeffizienten für LPF1-4 62

Listing 10. doFilter Funktion 62

Listing 11. Aufruf des Diode Ladder Filter 63

Listing 12. Implementation des Budget Nonlinear Model in der doFilter Funktion 64

Listing 13. cubicdist Funktion 65

Listing 14. Neue LP Klasse 65

Listing 15. Cubicdist Funktion in Gen 67

Listing 16. Definition der Parameter durch Inlets 68

Listing 17. Berechnung von GN und Gamma 69

Listing 18. Berechnung der Koeffizienten von LPF1-4 in Gen 69

Listing 19. Berechnung der Feedback Pfade zwischen den Filtern, des globalen Feedback Pfad, Input NLP Block und Summierung des Inputs und globalen Feedback Pfad 70

Listing 20. Ausführung von LPF1-4 (doFilter) 71

Listing 21. Analog-style ADSR Envelope Generator in Gen (McCulloch, 2015), part 1 73

Listing 22. Analog-style ADSR Envelope Generator in Gen (McCulloch, 2015), part 2 74

Listing 23. polyBLEP Sägezahn Oszillator Algorithmus in gen~, part 1 79

Listing 24. polyBLEP Sägezahn Oszillator Algorithmus in gen~, part 2 79

Listing 25. Waveshaper in gen~. 81