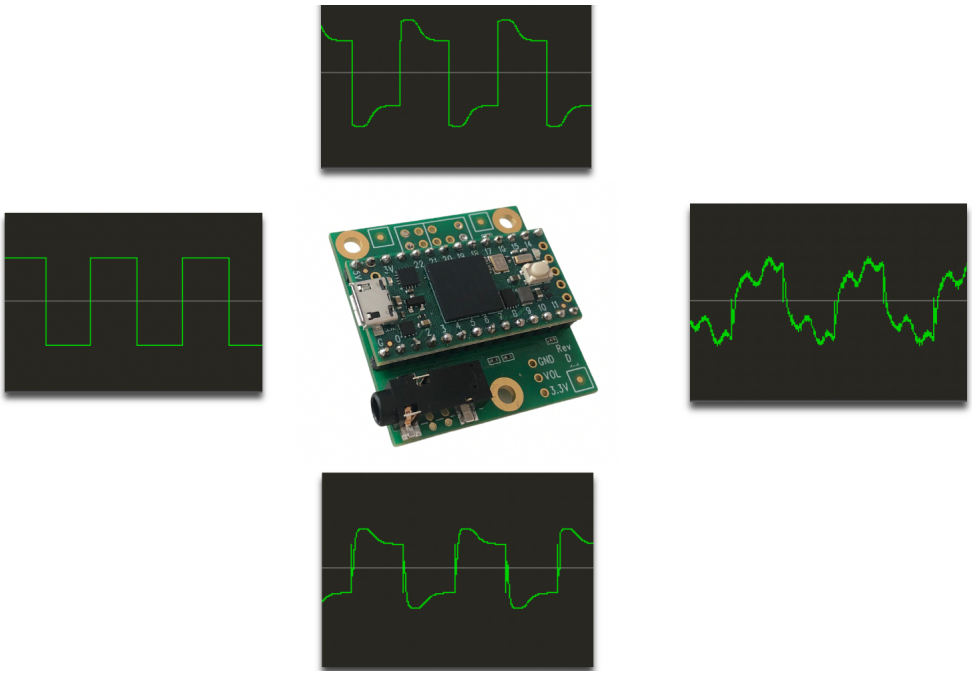
CHALMERS



Utveckling av en analog och digital effektenhet för musikalisk signalbehandling

Examensarbete inom Högskoleingenjörsprogrammet Elektroteknik

Lucas Olsson
Victor Simonsson

INSTITUTIONEN FÖR RYMD-, GEO- OCH MILJÖVETENSKAP

CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2021
www.chalmers.se

Examensarbete 2021

Utveckling av en analog och digital effektenhet för musikalisk signalbehandling

Lucas Olsson

Victor Simonsson

CHALMERS

Institutionen för rymd-, geo- och miljövetenskap
Chalmers Tekniska Högskola
Göteborg, Sverige 2021

Utveckling av en analog och digital effektenhet för musikalisk signalbehandling

© Lucas Olsson, Victor Simonsson 2021.

Handledare: Björn Bergholm, Broccoli Engineering AB
Examinator: Arto Heikkilä, Institutionen för rymd-, geo- och miljövetenskap

Institutionen för rymd-, geo- och miljövetenskap
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslag: Visualisering av signalbehandling.

Skrivet i LaTeX

Göteborg, Sverige 2021

Development of an analog and digital musical effects processing unit
Lucas Olsson
Victor Simonsson
Department of Space, Earth and Environment
Chalmers University of Technology Gothenburg, Sweden 2021
Bachelor´s thesis

Abstract

In the music world, the ability to shape sound is at least as important as the sound source. A guitar doesn't get far without its distortion, a synth without its filter or a voice without its reverberation. A multi effect unit combines these effects compactly and makes it applicable to many different instruments. In this project, carried out at Broccoli Engineering AB we have designed a multi effect unit consisting of three signal processing functions, Distortion, Filters and Reverberation. We have studied both analogue and digital signal processing in order develop an effect unit that is compatible with different instruments. The effect unit shall take the form of a traditional effect pedal and be large enough to accommodate all electronics. A multi effect device can usually contain a greater number of musical effects than the device in this project. To maximize the chances of a finished product, we chose to implement only three effects where the digital effect is programmed through Arduino IDE with the PJRC audio library. The analogue electronics should be smooth and efficient, therefore we have considered the implementation of circuits with a smaller amount of components but that meet sufficient power. The goal was to have a ready-made protype for direct application in musical signaling pathways. The result of the project was not as expected. After cumbersome troubleshooting, we first managed to make each effect in the device work on its face. Due to the misdesign of the overall circuit, the steps did not work together, but only one by one. The project stopped at testing the effetcs unit as there was no time to complete the entire unit with mounting on the metal box. Since the project relates to a DIY aspect, the result was positive. With minimal experience in the field, it was possible to study electronics and similar commercial and non-commercial effects units to form a theoretical understanding of the electronics and signal processing behind the effects.

The report is written in Swedish.
Keywords: Electroacoustics, DSP, Distortion, Audio effects, Acoustics, Pseudoacoustics, DIY, Filter

Sammanfattning

Inom musikvärlden är möjligheten att forma ljudet minst lika viktigt som ljudkällan. En gitarr kommer inte långt utan sin distorsion, en synth utan sitt filter eller en röst utan sin efterklang. En multieffektenhet kombinerar dessa effekter kompakt och gör det tillämpbart på många olika instrument. I det här projektet har en multieffektenhet designats bestående av tre signalbehandlingsfunktioner: Distorsion, filter och efterklang. Arbetet utfördes hos Broccoli Engineering AB. Vi ska genom att studera både analog och digital signalbehandling utveckla en effektenhet som är kompatibel för olika instrument. Effektenheten ska ha formen av en traditionell effektpedal samt vara tillräckligt stor för att få plats med all elektronik. En multieffektenhet kan i vanliga fall innehålla ett större antal musikaliska effekter än enheten i detta projekt. För att maximera chanserna för en färdig produkt valde vi att endast implementera tre effekter där den digitala effekten drivs av ett arduinokompatibelt kort och programmeras genom Arduino IDE med hjälp av PJRCs audio bibliotek. Den analoga elektroniken ska vara smidig och effektiv, därför har vi tagit i hänsyn till att implementera kretsar med en mindre mängd komponenter men som uppfyller tillräcklig effekt. Målet är att ha en färdig protyp för direkt applicering i musikaliska signalvägar. Resultatet av projektet blev inte som förväntat. Efter omständliga felsökningar lyckades vi först få varje effekt i enheten att funka var för sig. På grund av feldesign i den övergripande kretsen fungerade inte stegen tillsammans utan bara en och en. Projektet stannade vid test av effektenheten då tid inte fanns för att färdigställa hela enheten med montering på metallådan. Då projektet förhåller sig till en DIY-aspekt så var resultatet positivt. Med minimal erfarenhet inom området gick det att studera bland annat elektronik och liknande kommersiella samt icke-kommersiella effektenheter för att bilda en teoretisk förståelse för elektroniken bakom effekterna.

Nyckelord: Elektroakustik, DSP, Distorsion, Ljudeffekter, Akustik, Pseudoakustik, DIY, Filter

Förord

Det följande arbetet har utförts som examensarbete för Elektroingenjör, på Chalmers tekniska högskola. Arbetet har utförts hos Broccoli Engineering AB, på distans hemifrån samt i Campus Lindholmens laborationssalar. Ett enormt tack till Björn Bergholm som agerat handledare för oss under projektets gång. Björn och dom andra anställda vi träffat på Broccoli har gett oss ett varmt välkomnande till kontoret där vi gjort den större delen av vårt arbete. Vi vill också tacka Sakib Sistek som har gett oss tillträde till laborationssalar och ordnat relevant utrustning. Sist men inte minst vill vi självklart ge ett stort tack till vår examinator Arto Heikkilä som hjälpt oss strukturera vårt arbete och nå dit vi är idag.

Lucas Olsson och Victor Simonsson, Göteborg

Innehåll

Figurer

Terminologi

Effektpedal: Ett elektroniskt verktyg som förvränger en ljudsignal från ett elektroniskt musikinstrument.

DSP: Digital Signal Processing. Att med hjälp av digitala medel behandla en signal för att ändra dess egenskaper och karaktäristik

Efterklang: Fenomenet som uppstår när en ljudsignal inte bara går direkt till mottagaren utan också via väggar golv och tak i ett rum.

Delay: En musikalisk effekt som efterliknar ett akustiskt eko.

DIY: Do It Yourself. Att producera en produkt själv, utan proffesionell hjälp eller någon form av expertis i området

DC: Direct Current, Likspänning.

IC: Integrated Circuit, integrerad krets.

EQ: Equalizer, ett ljudtekniskt verktyg för att justera volymer i specifika frekvensområden.

Kompression: Ett ljudtekniskt verktyg för att komprimera ljudsignalers dynamiska omfång. Brett använt i inspelningssammanhang.

1
Inledning

Den här rapporten kommer innehålla vår ideskapning, utveckling och testning av en tredelad effektenhet för musikalisk signalbehandling.

1.1 Bakgrund

Musikalisk signalbehandling tar en otalig mängd former och funktioner i syfte att förändra karaktären för en viss ursprungssignal. Marknaden för så kallade effektpedaler för instrument såsom elgitarr, elbas och synthesizers är enorm och inkluderar stora företag med lång erfarenhet och stora resurser samt mindre producenter som utvecklar, designar och producerar allt hemifrån. I den här rapporten kommer vi, Lucas Olsson och Victor Simonsson, presentera vårt examnsarbete i elektroteknik, utfört hos Broccoli Engineering AB, där vi utvecklar en multieffektenhet för användning med elektroniska musikaliska instrument och studioutrustning.

1.2 Syfte

Syftet med det här arbetet är att utveckla en effektenhet med ett enkelt och intuitivt användargränssnitt samt en effektiv design och smart komponentval. Arbetet kommer kretsa till stor del kring hur utvecklingen av en sådan produkt går till, hur billigare hårdvara kan användas med mål att konkurrera med de stora företagen samt hur man utvecklar mer avancerade effekter med högnivåprogrammering och utan tidigare erfarenhet av realtids DSP-programmering.

1.3 Avgränsningar

Musikaliska multieffekter kommer i många skepnader med olika antal effekter och diverse formfaktorer. För att hålla oss fokuserade landade vi på tre effekter som ofta förekommer i effektkedjor, distorsion, filter och efterklang. Den digitala utvecklingen kommer ske i Arduinos IDE med syftet att hålla det på en programmeringsnivå som passar oss och kan vara lättare att greppa än lågnivåprogrammering i C++.
För de två analoga kretsarna vill vi att komponentmängden ska vara liten som möjligt. För att åstadkomma detta vill vi fokusera de analoga kretsarnas funktion kring operationsförstärkare. Den färdiga produkten ska konstrueras i en fabriksbyggd metalllåda. Detta för att efterlikna de produkter som redan finns på marknaden samt att begränsa storleken på prototypkortet.

1.4 Precisering av frågeställning

Målet med arbetet är att utforska hur utvecklingen av en hybrideffekt kan se ut och hur en enklare lösning på signalväg, kretsstruktur och algoritmer kan konkurrera med de kommersiella produkter som finns på marknaden idag.

Vi undersöker hur en ljudsignal utbreder sig genom analog elektronik för att förstå hur elektriska småsignaler förvrängs och formas för att åstadkomma musikaliska ljudeffekter.

Vi ska också studera vilka tekniska aspekter, t.ex. spänningsamplituder, likspänningsoffset och impedansnivåer som är relevanta för vårt projekt.

Spänningsmatning kommer att ske genom vägguttaget. Vi kommer då behöva studera tillämpningar för att hantera problemen kring att spänningsmata all elektronik.

Vi kommer utveckla en högnivåalgoritm för efterklang i Arduino för att se hur man kan effektivisera vissa delar av en redan etablerad algoritm för att kunna utveckla den vidare utan djup kunskap i lågnivå-DSP.

2
Teori

I det här kapitlet kommer vi gå igenom kort bakgrund, historia och teorin bakom de effekter och metoder som kommer användas i projektet.

2.1 Musikaliska ljudeffekter

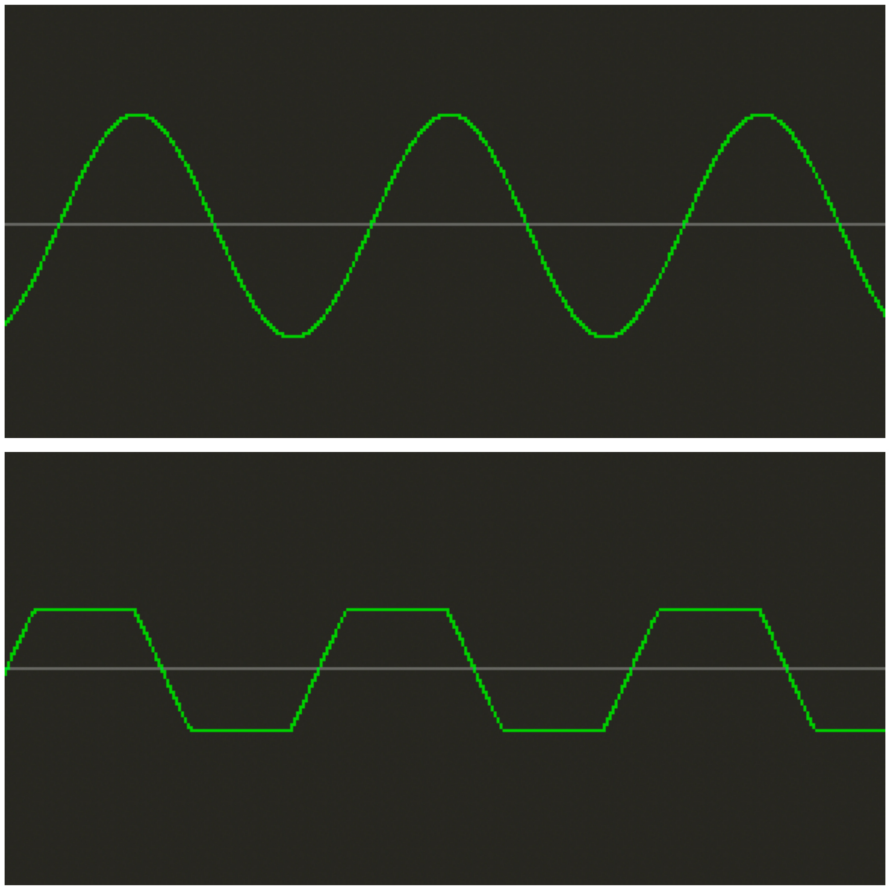
Musiken som den ser ut idag beror väldigt mycket, om inte helt på ljudeffekter. Det finns oerhört många effekter att använda för att göra musiken mer spännande. Vissa effekter brukar appliceras oftare på specifika instrument, t.ex distorsion, som vanligtvis appliceras på elgitarrer och elbasar. En så kallad effektpedal är, simpelt sagt, en elektronisk apparat som tar in en ljudsignal, behandlar den för att förändra karaktären på ljudet. Signalen skickas sedan till ett högtalarsystem, in i ett inspelningssystem eller vidare till andra effektpedaler för att vidare behandlas och förvrängas. Namnet effektpedal grundas i att enheterna oftast använts av gitarrister som använder fötterna för att styra dessa effekter då de spelar.
Andra effekter så som kompression, EQ eller efterklang appliceras i en mer detaljerad mån i slutproduktion av musik. Idag kan man nästan inte ens gå på en konsert utan att höra historiens påverkan av signalbehandling. Även där hör man alla signaler påverkade av dessa tekniker.
I det här projektet implementeras tre ljudeffekter i en enhet, distorsion, filter och efterklang. Dessa tre effekter är väldigt utbredda och vanligt använda inom musiken och alla dess genrer. Att implementera de här effekterna ger oss en bra balans av både analoga och digitala funktioner.

2.1.1 Distorsion

Distorsion är bland de mest använda signalbehandlingsmetoderna i hela världen men som tidigare sagt appliceras det oftast på elgitarrer. Första gången man någonsin började använda sig av distortion som en ljudeffekt var redan på 1940-talet då blues gitarrister började märka av en annorlunda extrem ton när de skruvade upp deras förstärkare till väldigt höga volymer [10]. Av ingenjörer på den tiden var den här effekten inte önskvärd och var något som man skulle hålla sig borta ifrån. Istället revolutionerade effekten musikvärlden.

Distorsion innebär i grunden att ljudsignalen förvrängs då den ökas över sin maximala signalnivå. En ren ton, det vill säga en ren sinusvåg förvrängs och börjar mer och mer efterlikna en fyrkantsvåg då den tvingas mer och mer mot sin övre och

undre amplitudtröskel. En fyrkantsvåg innehåller mer övertoner av högre ordning jämfört med en ren sinusvåg. Att öka inflytandet av övertoner av högre ordning resulterar i ett mycket skarpare och komplexare ljud. Detta låter inte önskvärt men distorsionen ger ljudet mer karaktär och gör en banal ton lite mer spännande. Den är som nämnt tidigare en av de mest använda effekterna inom gitarrer och baser men också väldigt stor inom synthvärlden. Figur 2.1 illustrerar en sinusvåg och en klippt sinusvåg.



Figur 2.1: En förenklad plot av hur en sinusvåg, som funktion av tiden, klipps vid en viss tröskel [9]

För att uppnå en musikalisk distorsion behövs det då öka inflytandet av övertoner. Att forma om en sinusvåg till att efterlikna en fyrkantsvåg så ökar man gradvis inflytandet av övertoner. Det finns flera sätt att uppnå distorsion. Ett av sätten är att uppnå det via användning av dioder i samspel med förstärkningen. En diod är en passiv och ickelinjär komponent som främst används för att sätta ett tröskelvärde på signalamplituden. Förstärkningen uppnås med hjälp av en återkopplad operationsförstärkare där en potentiometer bestämmer förstärkningen. Man brukar skilja på mjuk och hård klippning. Det beskriver hur hårt man begränsar signalen. Amplitudtröskeln bestäms av analoga komponenter, närmare sagt, dioder. Med hjälp av en varierbar förstärkning kan man uppnå hur mycket signalen formas mot en fyrkantsvåg. Med en varierbar förstärkning ökar man alltså indirekt inverkan av

övertoner i signalen. Det är alltså viktigt att notera att signalen inte nödvändigtvis förstärks i amplitudnivå.

En komplex vågform går att bryta upp i många delar, en fundamental frekvens och relevanta övertoner. Övertoner är sinus- och cosinusvågor som är direkt relaterade till den fundamentala frekvensen f genom en heltalsmultiplikator, till exempel: $2f$, $3f$, $4f$. Denna matematiska grundprincip kallas Fourierserie och upptäcktes av Jean-Baptiste Joseph Fourier.

Fourierserie går att applicera på många vetenskapsområden. Inom musik går det också appliceras då ljudvågor är en del av vågteori. I väldigt många fall inom musiken förekommer även komplexa vågformer såsom fyrkantsvågor eller sågtandsvågor. Fourierserien beskriver att en komplex våg går att beskrivas av vågens fundamentala frekvens plus en mängd övertoner.

$$y(t) = \frac{A_0}{2} + \sum_{k=1}^{\infty} \Big(A_k sin(k\omega_0 t) + B_k cos(k\omega_0 t) \Big) \tag{2.1}$$

Ekvation 2.1 beskriver att funktionen $y(t)$ är lika med en summa av ett antal sinus/cosinus vågor, k, med vinkelfrekvens $k\omega_0$, där ω_0 är grundtonens vinkelfrekvens. A_0 är en DC-komponent av grundsignalen det vill säga, medelvärdet av signalen $y(t)$. Är signalen centrerad kring noll så är A_0 lika med noll. A_n och B_n är amplituder för varje övertonkomponent $k\omega_0$. Övertoner tenderar att bli svagare när de är av högre ordning.

En fyrkantsvåg centrerad kring noll är en udda funktion, där definitionen för en udda funktion är:

$$-y(x) = y(-x) \tag{2.2}$$

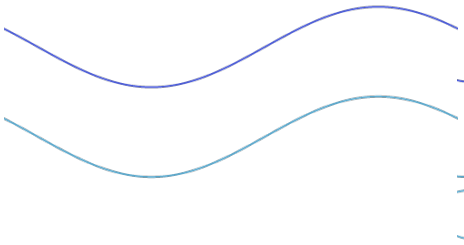
Det vill säga att övertonerna också kommer bestå av udda funktioner. En sinusfunktion är en udda funktion och en cosinusfunktion är en jämn funktion. Därför kommer fyrkantsvågens övertoner inte bestå av några cosinusövertoner. B_k i ekvation 2.1 är därför noll.

En fyrkantsvåg går då att approximera med k antal sinusvågor. Uttrycket ser då ut som följande:

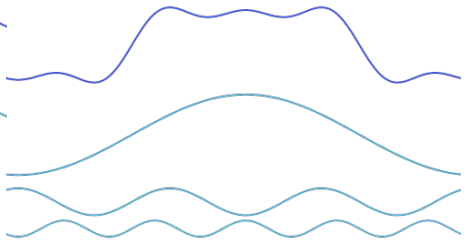
$$y(t) \approx Asin(\omega t) + \frac{A}{k} sin(k\omega t), \;\; k = 1, 3, 5, 7, ...\infty \tag{2.3}$$

A i ekvation 2.3 innebär att amplituden kan skilja från våg till våg.
Figurerna 2.2, 2.3 och 2.4 visualiserar hur en sinusvåg formas om till en fyrkantsvåg när inflytandet av övertoner ökar. En vanlig sinusvåg med sin fundamentala frekvens syns i figur 2.2. Samma sinusvåg fast med 2 till övertoner syns i figur 2.3. I figur 2.4 syns den fundamentala sinusvågen fast med åtta adderade övertoner. Det går tydligt att se skillnaden i form när den vanliga sinusvågen formas mot en fyrkantsvåg när mer övertoner adderas.

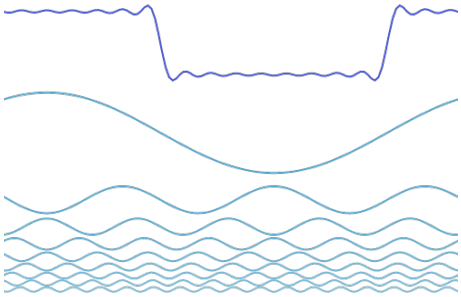
Vi använde [7] för att få en inblick i hur förändringen i vågen sker vid ökat inflytande av övertoner. Den fundamentala frekvensen är den första turkosa vågen och under den syns de resterande övertoner som adderas. X-axeln representerar tid och y-axeln representerar amplitud.



Figur 2.2: Sinusvåg i den fundamentala frekvensen



Figur 2.3: Den fundamentala vågen och fyra övertoner



Figur 2.4: Fundamentala frekvensen och 8 övertoner

Det är värt att notera att varje adderad överton har k gånger frekvensen, där k bara antar udda tal. Varje överton där k är lika med ett jämnt tal har värdet 0. Det vill säga att:

$$B_k = 0, \quad k = 0, 2, 4, 6.....\infty \tag{2.4}$$

2.1.2 Filter

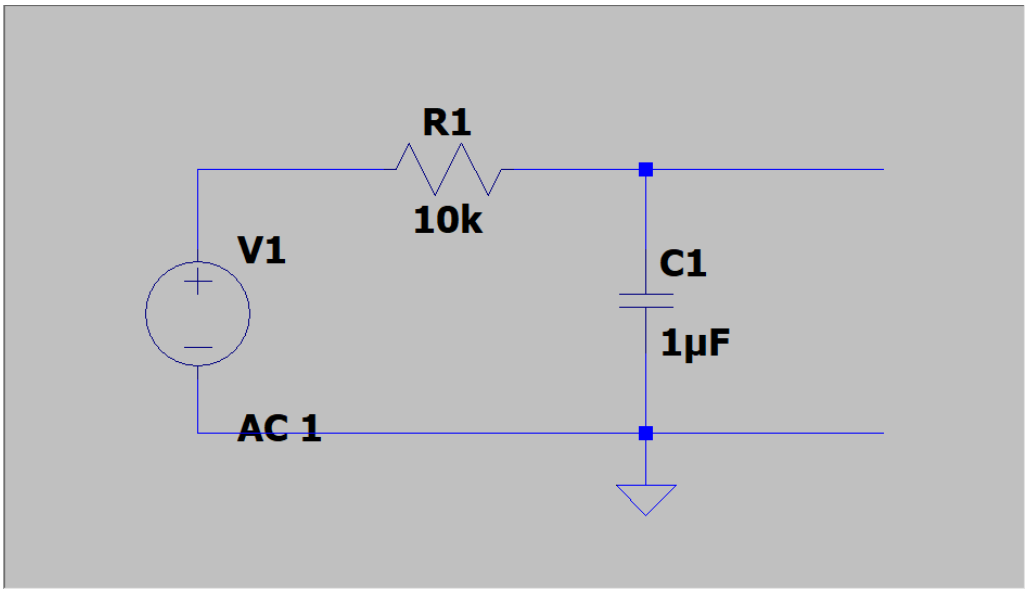
Analoga filter har använts i musikaliska sammanhang sedan 60-talet då Dr. Robert Moog och andra pionjärer började utveckla analoga synthar för konsumentmarknaden. Filtren användes för att forma en komplex signal till något simplare och enklare att implementera i ett större musikaliskt kontext. Genom att forma de komplexa signalerna så kan man välja att framhäva samt dämpa frekvenser i signalens spektrum. På synthesizers och andra instrument har filter således brukats för att designa och skapa ljud. Lågpassfilter kan till exempel användas för att forma ett basljud genom att filtrera bort de högre frekvenserna.

De vanligaste filterna som används i musikaliska sammanhang är lågpass, högpass samt bandpass. Bandspärr används inte lika ofta som ett musikaliskt filter. Fokuset i det här projektet är lågpass- och högpassfilter.

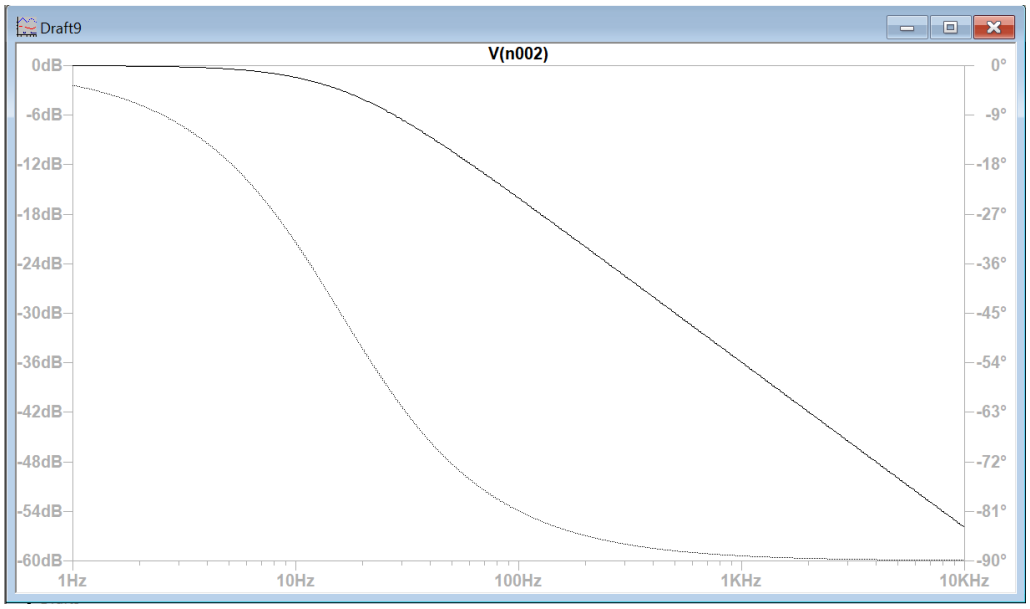
Analoga filter består mestadels av passiva komponenter såsom resistorer, kondensatorer och induktorer. Kondensatorn är den viktigaste komponenten i vår filterkrets och är nyckeln till filtereffekten. Kondensatorer är så kallade 'spänningströga' komponenter [21]. Appliceras en spänning över en kondensator kommer det ta en viss tid innan den givna spänningen stabiliserats. Som alla andra passiva komponenter har kondensatorn en impedans. Impedansen är en beteckning för elektriskt motstånd och kondensatorns impedans definieras av formeln:

$$Z = \frac{1}{j\omega C}, \qquad \omega = 2\pi f \tag{2.5}$$

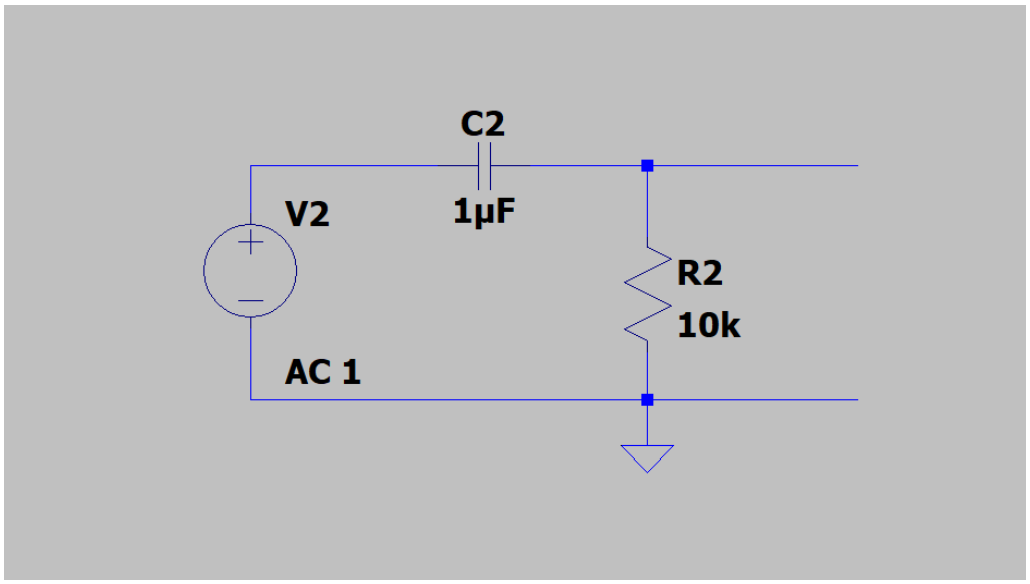
Kondensatorns impedans är därför frekvensberoende. Det som direkt går att se i formeln för kondensators impedans är att när ω är väldigt låg, i princip 0, så är impedansen oändligt hög. Placeringen av frekvensberoende komponenter är viktig för att åstadkomma önskad effekt.



Figur 2.5: En passiv lågpasskrets



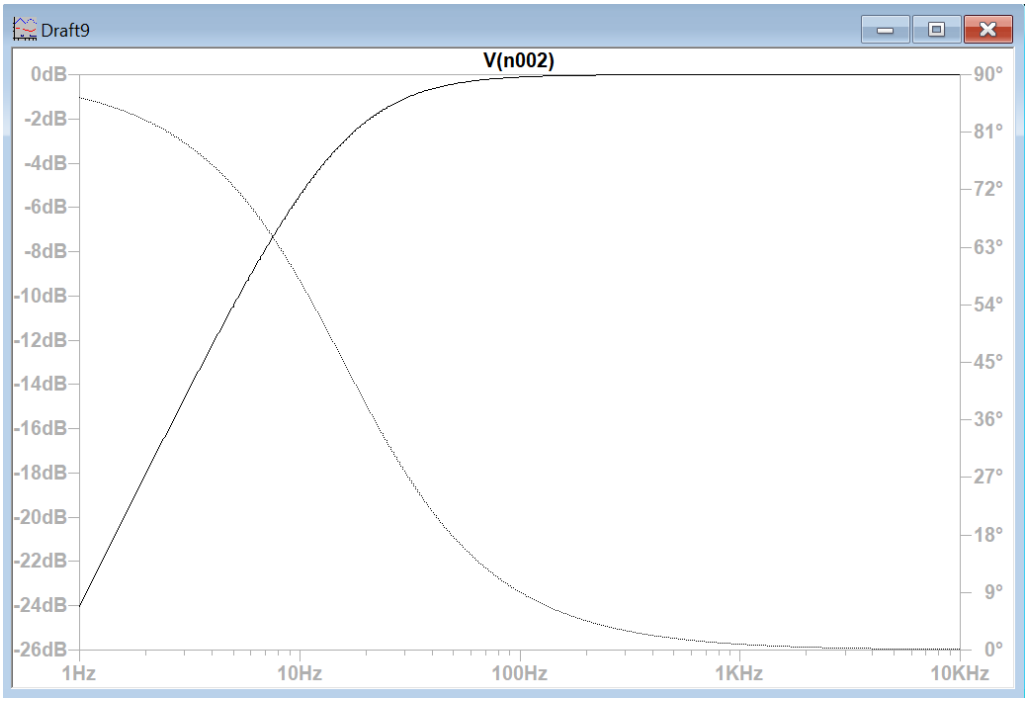
Figur 2.6: Lågpasskretsens bodeplot



Figur 2.7: En passiv högpasskrets

I figurerna 2.5 och 2.7 är två exempel på de enklaste filterkretsarna. Figur 2.5 är ett lågpassfilter och figur 2.7 är ett högpassfilter. Det syns i figurerna att komponenternas placeringar spelar stor roll. Samma komponenter, en resistor och en kondensator, åstadkommer motsatta filtereffekter. Frekvensresponserna i figur 2.6 och 2.8 illustrerar även detta.

Kretsarnas överföringsfunktioner går att ta fram med hjälp av enkel kretsanalys.



Figur 2.8: Högpasskretsens bodeplot

Överföringsfunktion definieras som en relation av utspänningen V_o och inspänningen V_i och betecknas som H.

$$H = \frac{V_o}{V_i} \tag{2.6}$$

Den totala överföringsfunktion går att ta fram genom spänningsdelning. Utspänningen blir då spänningen över kondensatorn C1 som går att se på figur 2.5. Spänningsdelningen blir då inspänningen multiplicerat med kvoten av komponentens impedans och kretsens totala impedans:

$$Z_{total} = R + \frac{1}{j\omega C} \tag{2.7}$$

$$V_o = \frac{\frac{1}{j\omega C}}{R + \frac{1}{j\omega C}} V_i, \;\; V_o = \frac{1}{Rj\omega C + 1} V_i \tag{2.8}$$

Detta blir den allmänna överföringsfunktionen från filtret. Amplitudresponsen är det enda intressanta då fasresponsen från enheten inte är relevant. Den resulterande amplitudresponsen blir då:

$$|H| = \frac{1^2}{\sqrt{(Rj\omega C)^2 + 1^2}} \tag{2.9}$$

Brytfrekvensen i ett filter definieras som punkten där signalen dämpats med 3 dB, det motsvarar med en faktor roten ur 2. Det vill säga att $|H| = |\frac{V_o}{V_i}| = \frac{1}{\sqrt{2}}$. Ekvation

2.9 går då att utveckla till:

$$\omega = \frac{1}{RC}, \quad f = \frac{1}{2\pi RC} \tag{2.10}$$

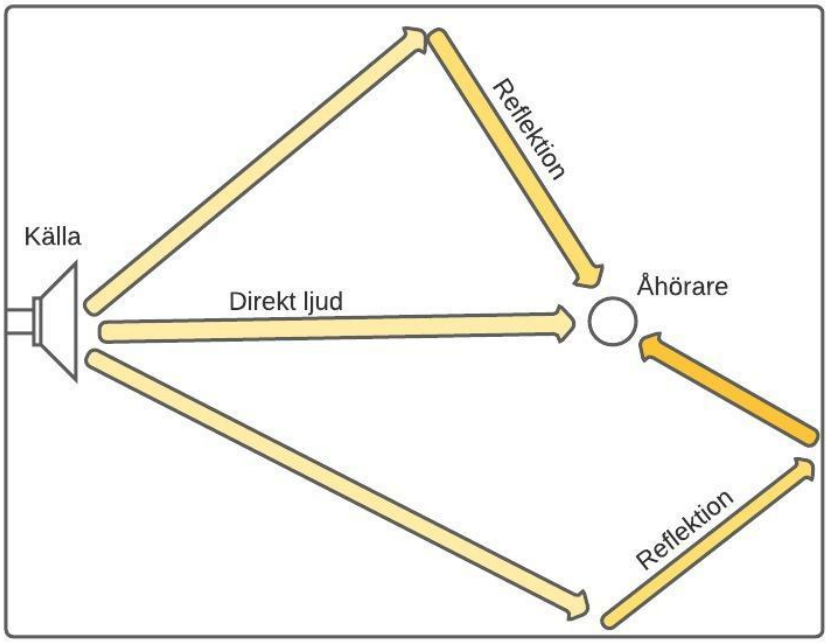
Brytfrekvensen är vad som främst används vid design av filtret. Med tanke på att alla signaler ska omvandlas till ljudvågor är alla frekvenser som hamnar utanför 20-20 kHz området inte nödvändiga. Anledningen till det är att människan inte kan höra frekvenser under 20 Hz eller över 20 kHz. Ekvation 2.10 används då för att designa övre och undre gränsfrekvenser så att de passar frekvenser för ljudvågor.

I kretsen behövs även filter med tekniska funktioner istället för musikaliska. En DC-offset kan förekomma i signalkedjan och förstöra ljudkvaliteten genom att förskjuta växelspänningens amplitud. Detta kan leda till obalans i ljudet vilket inte är önskvärt. För att lösa detta problemet har vi implementerat högpassfilter på flera ställen med en brytfrekvens strax under 20 Hz-gränsen för att filtrera bort DC. DC kan betraktas som en växelspänning med frekvens $f = 0$ Hz.

2.1.3 Efterklang

Efterklang, Reverb på engelska, är produkten av de reflektioner som uppstår i ett stängt utrymme efter att en ljudvåg exciteras [19]. I alla rum som inte är aggressivt ljudbehandlade har dessa reflektioner. I rum av mindre storlek, såsom vardagsrum eller sovrum är dessa reflektioner små men ändå hörbara. Det som oftast nämns dock när man pratar efterklang är det ljud som uppstår när ljud i större rum såsom kyrkhallar eller konsertlokaler. Sedan tidigt 1930-tal har artister och ingenjörer försökt efterlikna akustiken i dessa lokaler genom olika artificiella metoder [1]. De första artificiella efterklangen gjordes med hjälp av mekaniska medel, så som fjädrar eller metallplattor som signalen skickades igenom. Antingen genom att få strömmen genom materialet att inducera ett magnetiskt fält eller att använda kontaktmikrofoner och högtalarelement mot ytorna för att skapa oregelbundna vibrationer i materialet som efterliknar klangen av ett rums efterklang. Dessa mekaniska metoder används fortfarande men mer i syfte att få en väldigt specifik karaktär på klangen. Idag har digitala artificiella efterklangsalgoritmer blivit normen då de enkelt kan implementeras på de kraftfulla datorer vi har tillgängliga.

Efterklangen i ett rum kan representeras av blandningen av de olika signaler som uppstår. När en ljudvåg skickas genom ett rum kommer den sträckas åt alla håll i rummet genom reflektioner. Dessa reflektioner kan i de flesta fall påverka hur signalen låter i slutändan. Detta är beroende av de material som vägg, tak och golvytor är gjorda av och hur dessa är formade. Figur 2.9 nedan är en förenklad bild av hur ett ljud reflekteras mot olika ytor och kommer tillbaks till åhöraren som en summa av de olika signalerna. En ljudvåg är dock inte diskret och kan delas in i separata signaler som är antingen reflektioner eller direkta. De reflektioner som uppstår är inte förlustfria vilket medför att signalens styrka kommer avta drastiskt.



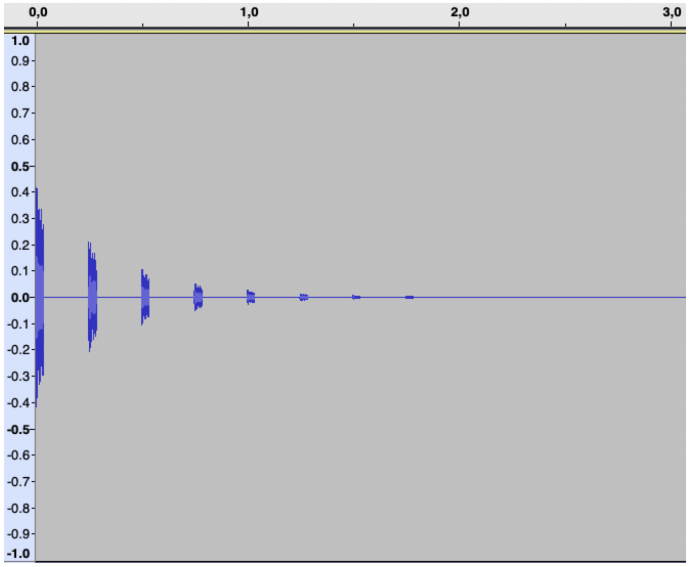
Figur 2.9: En förenklad visualisering av hur reflektioner agerar i ett stängt utrymme.

De flesta har nog någon gång upplevt ett eko i tillexempel en bergdal eller vid en tät skog. Vid så stora områden kommer ljudet ta så lång tid på sig att reflektera mot den motstående ytan att åhöraren kan uppfatta en nästintill identisk reflektion några sekunder efter excitation. Detta så kallade eko upplevs inte på samma sätt i ett rum eller inbyggt utrymme. På grund av ljudhastigheten i luft som är ca 340 m/s kommer dessa reflektioner skapa egna reflektioner otaligt många gånger för att ge en hög så kallad ekodensitet efter excitation av ursprungsljudet. Det medför att individuella transienter överlappar och signalens efterklang blir en mer kontinuerlig signal.

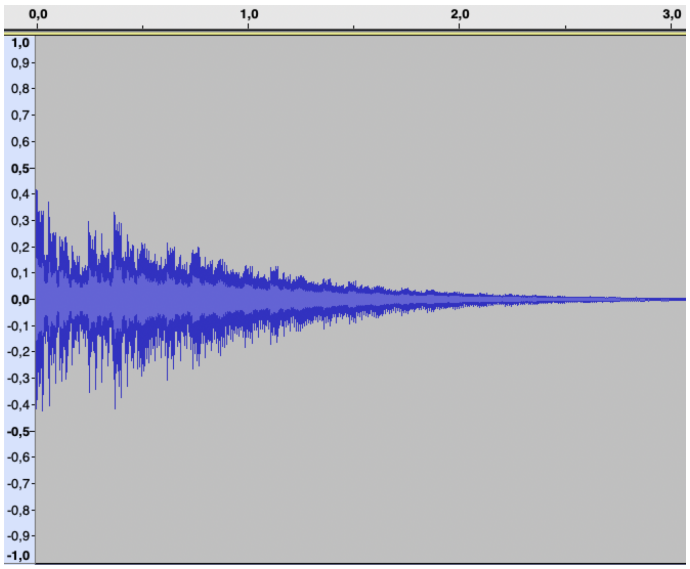
Efterklang och eko kan verka som ekvivalenta i slutändan då båda är en summering av reflektioner över tid. För att urskilja vad som är efterklang och vad som är eko sattes ett test upp i musikproduktionsprogrammet Audacity för att visa impulsresponsen över tid. I figur 2.10 visas impulsresponsen när en signal exciteras och återkopplas med -6 dB dämpning och 250 ms fördröjning under 3 sekunders tid.

Genom att seriekoppla ett större antal av dessa ekoeffekter uppnås en högre ekodensitet snabbt. Figur 2.11 visar samma impuls spelad genom sex seriekopplade fördröjningssteg. Fördröjningstiderna är slumpmässigt valda på ett spann mellan 1 ms och 250 ms.

Som man ser i figur 2.11 skapar en enkel seriekoppling av sex stycken fördröjningsfunktioner en signal som liknar en efterklang. Fastän man ser de toppar som återkommer i signalen så faller de olika reflektionerna samman till en kontinuerlig signal. Detta är basfunktionen av många efterklangsalgoritmer och det som skapar den essentiella skillnaden mellan efterklang och eko.



Figur 2.10: Impulsrespons över tre sekunder med 250 ms fördröjning

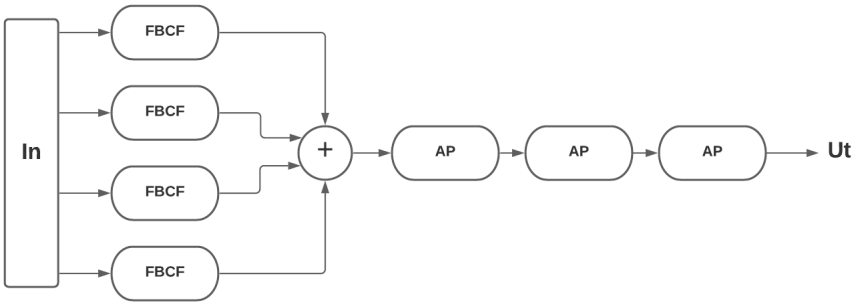


Figur 2.11: Impulsrespons av sex seriekopplade ekofunktioner med fördröjningstider mellan 1 ms och 250 ms

2.1.4 Algoritmisk efterklang

Algoritmiskt efterklang kommer i många olika former och funktioner. I artikeln 'Colorless Artificial Reverberation' från 1961 [11] redogör Manfred Schroeder och Ben Logan för den teknik som var grunden för en av de första, realistiska efterklangsalgoritmerna. Strukturen var väldigt simpel men används ofta som grund för mer komplicerade algoritmer idag. Den grundas i fyra parallellt återkopplade kamfilter (FBCF) som summeras och förs genom tre allpassfilter (AP), se figur 2.12. Återkopplingen är grunden till efterklangseffekten då den låter signalen 'reflekteras' tillbaka för att skapa illusionen av reflektioner i ett rum. [11]. Tidskonstanterna är satta så att de aldrig överlappar varandra och därav fortsätter fylla ut signalen utan att

dubblera och skapa resonanser. Figur 2.12 visar blockschemat för en enkel schroeder reverbarator utvecklad av Prof. John Chowning på Stanford [14].

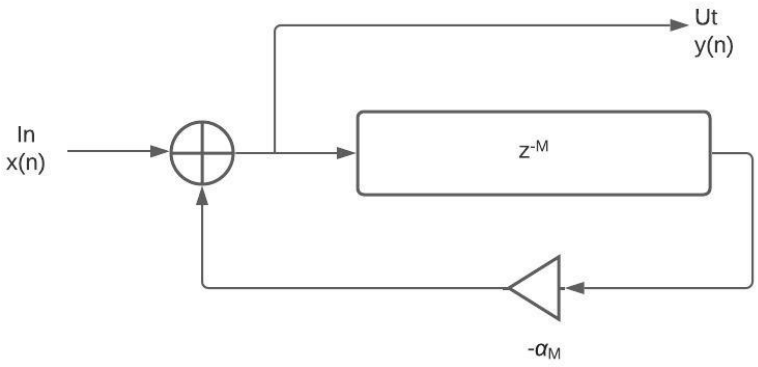


Figur 2.12: Blockdiagram av Schroederalgoritmen [15]

För att förstå vad varje block i Schroeders algoritm gör behöver dess inre struktur undersökas.

2.1.4.1 Kamfilter

Kamfilter kommer i två varianter, Framkopplade och återkopplade. Den mest klassiska varianten är det återkopplade kamfiltret [2] vilket Schroeder nyttjade för att efterlikna summeringen av fasförskjutna signaler som avtar i amplitud.



Figur 2.13: Blockschema för de återkopplade kamfiltrerna

I figur 2.13 tar det återkopplade kamfiltret (FBCF) in signalen och återkopplar en fördröjd version av signalen. Summeringen återkopplade signalen och den direkta skickas ut. För att uttrycka detta matematiskt erhålls en enkel funktion x(n) beroende av införstärkningen b_0, återkopplingskoeffecienten a_M och fördröjningstiden M. Se ekvation 2.11.

$$y(n) = x(n) - \alpha_M y(n - M) \tag{2.11}$$

Med detta kan in- och utsignalen (x(n) och y(n)) separeras och z-transformen tas för att få ut överföringsfunktionen.

$$(1 - \alpha_M z^M)Y(z) = X(z) \tag{2.12}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 - \alpha_M z^M} \tag{2.13}$$

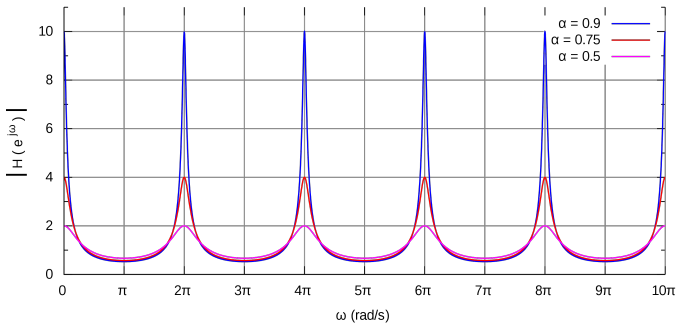
Genom att sätta $z = e^{j\Omega}$ erhålls ekvation 2.14.

$$H(e^{j\Omega}) = \frac{1}{1 - \alpha_M e^{Mj\Omega}} \tag{2.14}$$

Med absolutbeloppet av ekvation 2.14 ges ekvation 2.15

$$|H(e^{j\Omega})| = \frac{1}{\sqrt{1 + \alpha_M^2 - 2\alpha_M cos(\Omega M)}} \tag{2.15}$$

Detta plottas för att visa figur 2.14.

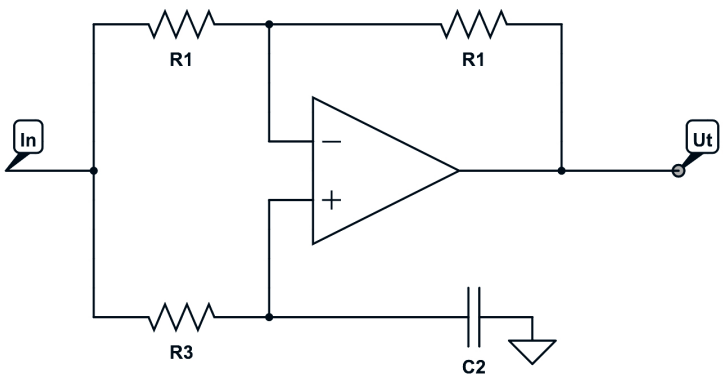


Figur 2.14: Frekvensrespons hos ett återkopplat kamfilter [3]. Figuren är använd under licensen Creative Commons, CC BY-SA 3.0 [4].

Trots namnet är kamfilter egentligen inte ett vanligt filter utan bara en biprodukt av den fasförskjutningen som sker när återkopplingen summeras med insignalen. Vid observation av hur amplitudresponsen ser ut för överföringsfunktionen visas filtrets karaktäristiska respons som gett den sitt namn. På grund av den fasförskjutningen mellan de två summerade signalerna kommer de interferera vid vissa periodiska frekvenser. Om man gör ett svep över frekvensspektrat uppstår följande form, där av 'kamfilter'.

2.1.4.2 Allpassfilter

Ett allpassfilter är speciellt i världen av filter då dess funktion inte är att filtrera ut valda frekvenser ur en signal utan att fasförskjuta och fördröja en signal [20]. Allpassfiltret är likt kamfiltret i struktur och konstrueras ofta genom att kombinera ett återkopplat och ett framåtkopplat kamfilter. Detta kan byggas analogt med en operationsförstärkarkrets som vi ser i figur 2.15.



Figur 2.15: Analog krets för ett Första ordningens allpassfilter [20]

Den analoga kretsen ger följande överföringsfunktion, amplitudrespons och fasrespons. Se ekvation 2.16, 2.17 respektive 2.18.

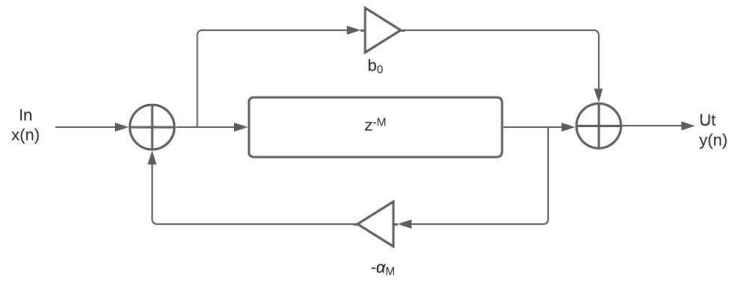
$$H(j\omega) = \frac{1 - j\omega R_3 C}{1 + j\omega R_3 C} \tag{2.16}$$

$$|H(j\omega)| = 1 \tag{2.17}$$

$$\angle H(j\omega) = -2arctan(\omega R_3 C) \tag{2.18}$$

Som visat med amplitudresponsen $|H(j\omega)|$ och fasresponsen $\angle H(j\omega)$ är allpassfilter konstanta i amplitud över hela frekvensspektrat men varierar i fas. Med ekvation 2.18 kan vi se, baserat på att $arctan(x)$ har en värdemängd mellan $-\pi$ och π och att ω R_3 och C är positiva tal, att fasvinkeln där av går mellan 0 och -180°.

Om vi tar en närmre titt på blockschemat i figur 2.16 ser vi dess funktion.



Figur 2.16: Signalschema för ett allpassfilter

Med de tre seriekopplade allpassfiltrena används för att utöka längden på efterklangen. På grund av att varje allpasssteg är en sumering av tre signaler; den opåverkade insignalen, återkopplingen från allpassteget och kommer detta ge högre ekodensitet på slutsteget. Med olika fördröjningar på varje allpassfilters fördröjningsledning skapar det hög komplexitet och ekodensitet för varje konsekutivt seriekopplat allpassteg.

2.1.4.3 Val av fördröjningskoefficienter

För att åstadkomma en stabil efterklang behöver tidskoefficienterna så sällan som möjligt överlappa. Om två reflektioner överlappar exakt kommer deras amplituder att interferera och skapa en spets i vågformen. För att undvika detta kan man använda det som i talteori kallas för relativa primtal. Det är tal som inte har någon gemensam faktor, förutom 1, kallas för relativa primtal. De behöver inte vara primtal i sig självt men som grupp tal är de primtal [8]. Som exempel kan vi titta på talen 21 och 22. 21 är delbart med 3 och 22 är delbart med 2 och 11. Dessa två tal är då inte primtal men då de inte delar någon gemensam faktor är de primtal relativt varandra. Enligt Schroeder maximerar detta tiden innan impulsresponsen repeterar [13]. Om tidskoeffecienterna är delbara med varandra kommer resonanser skapas väldigt snabbt.

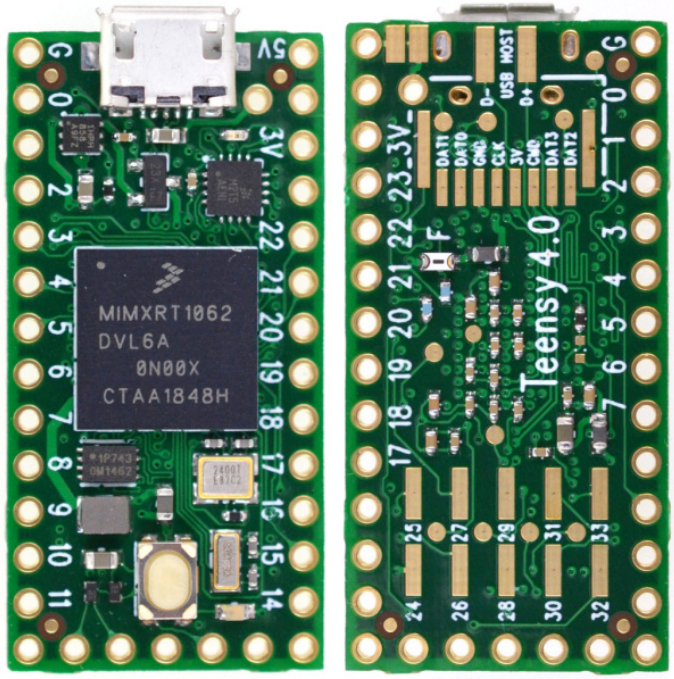
2.2 Hårdvara

Hårdvaran som valdes för det här projektet är ett Teensykort producerat av PJRC.com [18]. PJRC:s serie med arduinokompatibla mikrokontrollers har blivit mer och mer populära bland hobbyister då de, i sitt smidiga paket ändå ofta har mer beräkningskraft än arduinos egna kort. Med Teensyduinobiblioteken adderas många spännande funktioner såsom realtids DSP.

2.2.1 Teensy 4.0

All kod kommer laddas upp på och köras av en Teensy 4.0 med tillhörande Teensy audio shield. Figur 2.17 visar Teensy 4.0 fram och baksida. Kortet är utrustat med en

micro-USBkontakt för kommunikation och spänningsmatning, samt en resetknapp för att återkalla programmet som är uppladdat på kortet. Kortet kan drivas av två källor, 5 V via USB eller via 5 V från extern källa. Detta konfigureras med hjälp av att bryta en koppling mellan VUSB och VIN.



Figur 2.17: Teensy 4.0 [18]

Se tabell 2.1 för Teensy 4.0 specifikationer.

Tabell 2.1: Tekniska specifikationer hos Teensy 4.0

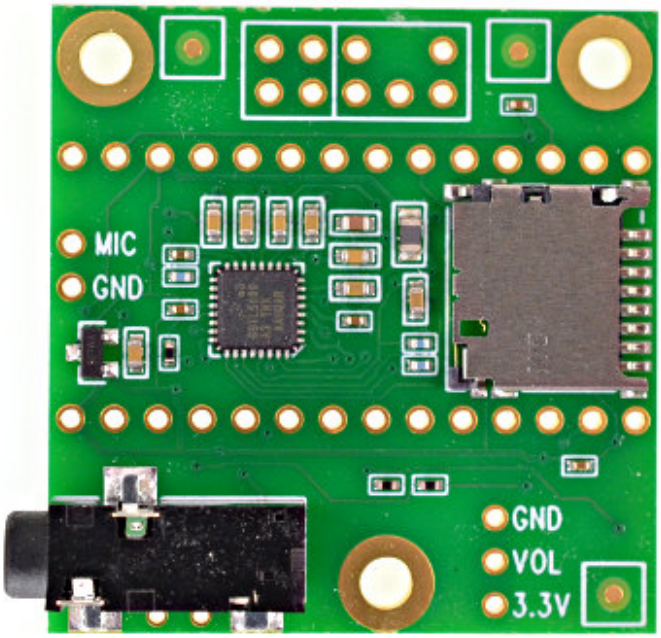
Processor	Kärna	Clockhastighet	Flashminne	Ramminne
IMXRT1062DVL6A	Cortex-M7	600MHz	1984 kbytes	1024 kbytes

2.2.2 Teensy audio shield

För att få ljud in och ut ur teensy 4.0 används teensys Audio shield. Teensy audio shield är ett enkelt kretskort som lägger till en 16-bitars, 44.1 kHz DAC (Digital to Analog Converter) till teensys familj av kort. Kortet är även utrustat med läsare för microSDkort, 3.5 mm stereo hörlursutag och lödpunkter för en mikrofon, stereo ljud in och stereo ljud ut. Figur 2.18 visar en bild av ljudkortet.

2.2.3 Teensy audio GUI

Teensys system design verkryg används för att beskriva hela signalflödet för programmet. Miljön har en samling av många vanliga och ovanliga funktioner till ljudbehandling och nedan kommer en enkel beskrivning av de vi använt oss av samt dess grundläggande funktionalitet.



Figur 2.18: Teensy Audio Shield [18]

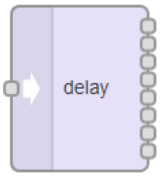
Nedan visas blocken för I^2S in och ut. I^2S är ett protokoll likt I^2C i funktionalitet i form av att de båda skickat och tar emot digitala signaler. I^2S skilljer sig mot sin enklare motpart i att I^2S-bussen endast innehåller ljudsignaldata [12]. I^2S är på samma sätt som I^2C standardiserad och används ofta i digitala transmissionssystem. Figurerna 2.19 och 2.20 visar hur dessa portar deklareras i Teensys Audio GUI.

Figur 2.19: I2S inport

Figur 2.20: I2S utport

Delayblocket har en enkel funktion i grunden där den tar in en signal och med delay.delay(i,t) ställer in vilken fördröjningstid i millisekunder (t) som utport (i) ska ha. Varje individuell utgång kallas för en tap vilket gör det här blocket till en 8 tap delay. När en tidsperiod deklareras för varje tap sparar funktionen den längden av ljud i teensys dynamiska minne och skickar sedan vidare det.

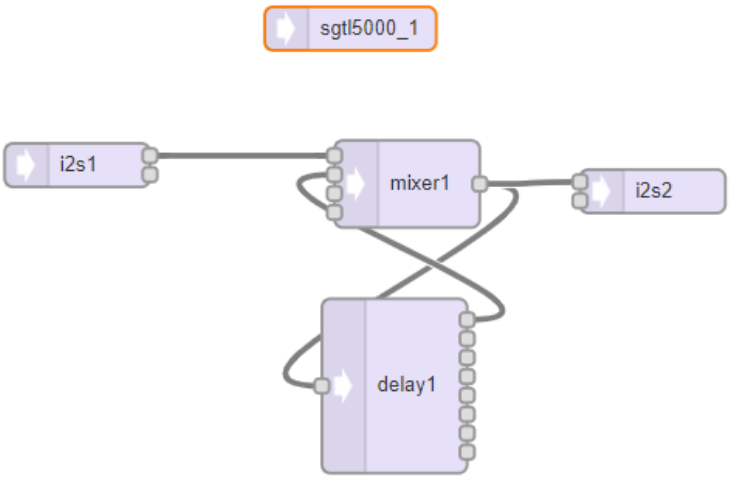
Mixern är en enkel summeringsmixer där de fyra inportarnas signaler till vänster summeras till utgången till höger. Mixerns nivåer sätts med mixer.gain(i,g); där i är valt port och g är signalstyrkan ut. För att signalen ska hållas ren bör summan av utsignalernas magnitud vara max 1. Om summan av insignalernas amplitud är större än 1 kommer signalen distorderas. Se figur 2.21 och 2.22 för respektive blocks representation i teensy audio GUI.



Figur 2.21: Symbolen för Delayfunktionen i Teensy Audio GUI [16]

Figur 2.22: Symbolen för mixerfunktionen i Teensy Audio GUI

Figur 2.23 visar ett enkelt blockschema som genererar en ekoeffekt genom ett fördröjningsblock och en mixer. Blocket med namn sgtl5000_1 är deklareringen av ljudkortet som används. Detta för att mikrokontrollern ska veta var var den tar in och ut ljudet.



Figur 2.23: En enkel återkoppling med en nyttjad fördröjningstapp.

När blockschemat exporteras genereras en bit kod som beskriver de block som finns samt hur dessa block är sammankopplade.

```
#include <Audio.h>
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <SerialFlash.h>

// GUItool: begin automatically generated code
AudioInputI2S            i2s1;
AudioEffectDelay         delay1;
AudioMixer4              mixer1;
```

```
AudioOutputI2S           i2s2;
AudioConnection          patchCord1(i2s1, 0, mixer1, 0);
AudioConnection          patchCord2(delay1, 0, mixer1, 1);
AudioConnection          patchCord3(mixer1, delay1);
AudioConnection          patchCord4(mixer1, 0, i2s2, 0);
AudioControlSGTL5000     sgtl5000_1;
// GUItool: end automatically generated code
```

Detta kan sedan importeras i Teensy Audio GUI för att återgenerera blockschemat och göra korregeringar utan att behöva leta i kodhuvudet.

2.2.4 Line- eller instrumentnivå

I musikutrusting stöter man ofta på två ord, Line och Instrument. Dessa dyker ofta upp i form av en switch vid en ingång på en mixer eller ett ljudinterface. Dessa ord innefattar vilken ljudkälla du använder och vilken nivå dessa signaler har. Line syftar till aktiva ljudkällor, det vill säga synthar, DJ utrustning eller andra enheter med intern aktiv nivåstyrning. Instrument och andra sidan syftar på passiva instrument så som elgitarrer eller elbasar där elektroniken är passiv i källan.

För effektpedaler är nivån vanligtvis densamma in och ut det vill säga att förstärkningen genom enheten är 0 dB. Dock kan olika nivåer på insignal påverka hur signalen formas av effekterna.

2.2.5 Texas instruments TL074

I det här projektet valde vi att bygga de analoga kretsarna huvudsakligen kring operationsförstärkare. Distorsionskretsar går också att bygga upp med hjälp av transistorer istället för operationsförstärkare. Operationsförstärkare var det självklara valet i det här fallet. Motivationen till det valet är både en tidsbegränsning som hindrar från att utveckla tre stycken avancerade effektsteg, men också att tanken med detta projektet är att det ska vara DIY. Poängen med DIY är att utan professionell bakgrund eller hjälp utföra ett givet projekt. Det vill säga att projekten i fråga oftast bruka vara relativt simpla. Att implementera en operationsförstärkarkrets i distorsions- samt filtersyfte är generellt sett simpelt.

En operationsförstärkare är en elektronisk förstärkarkrets med två ingångar och en utgång. Operationsförstärkaren är en differentiell förstärkare, det vill säga att den förstärker skillnaden i spänningen mellan de två ingångarna. Grunden till operations-

förstärkarens funktioner ligger i dess tekniska egenskaper. De tekniska egenskaperna kan delas upp i ideala och icke ideala. De ideala tillämpas vid teoretiska beräkningar och de icke ideala egenskaperna tillämpas vid praktiska implementeringar.

De ideala egenskaperna:

- Oändligt hög inimpedans, strömmarna in är noll.
- Noll utimpedans, oändligt hög ström ut.
- Obegränsad utspänningsnivå.
- Noll brus.
- Oändlig slew rate
- Oändlig bandbredd

De ideala egenskaperna är bara teoretiska. I verkligheten är det annorlunda. De praktiska egenskaperna liknar mer:

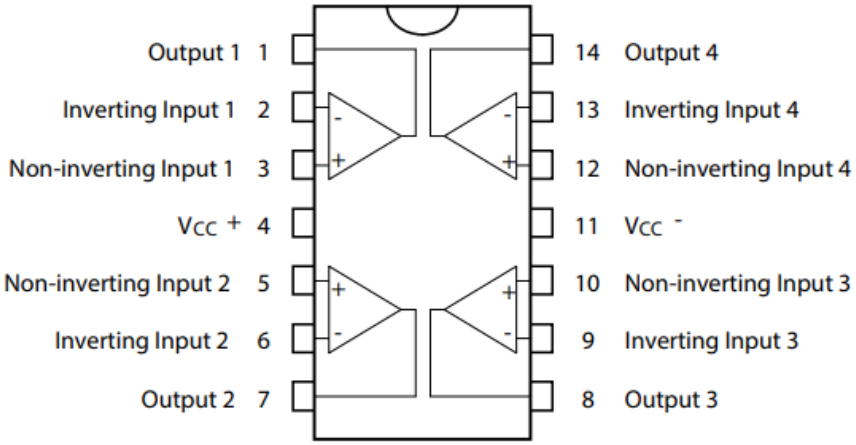
- Väldigt hög inimpedans, strömmarna in är minimala.
- Väldigt låg utimpedans, hög ström ut.
- Begränsad utspänningsnivå.
- Existerande brus.
- Begränsad slew rate
- Begränsad bandbredd

Operationsförstärkaren är en aktiv komponent. Det vill säga att den kräver en extern matningsspänning för att fungera. Utspänningen är begränsad till den matningsspänning man försörjer förstärkaren med.

Det finns många implementeringar för en operationsförstärkare, med hjälp av passiva komponenter kan man bland annat bygga kretsar som utför matematiska operationer, inverterar spänningar och förstärker spänningar. I det här projektet tillämpar vi operationsförstärkaren som en spänningsförstärkare och en spänningsföljare.
I distorsionsdelen fungerar operationsförstärkaren som en ren förstärkning. I samspel med dioderna åstadkommer vi distorsion. I filtret däremot används operationsförstärkaren som en spänningsföljare. En spänningsföljare är en operationsförstärkare som är direkt återkopplad till sin negativa ingång. Det man åstadkommer med en återkoppling är en bufferkrets. De olika stegen i kretsen separeras alltså med en spänningsföljare så att de inte påverkar varandras funktioner. Anledningen till att spänningsföljaren fungerar är tack vare operationsförstärkarens höga inimpedans och låga utimpedans. Det är särskilt viktigt att ha det i filterkretsarna då vi har två steg av RC-kretsar i respektive filter. Att separera de två RC-kretsarna är väsentligt. Om inte RC-kretsarna separeras funkar inte filtret, då påverkar de varandra för mycket. Spänningsföljarens utgång följer alltså spänningen på den positiva ingången. 5 V på den positiva ingången och 0V på utgången hade resulterat i att utgången fått en spänning som är $A(V_+ - V_-)$,då utgången är noll är skillnaden 5 V-0 V. Utgången kommer då drivas mot 5 V.

Det finns väldigt mycket IC-kretsar på marknaden. Från vår forskning märkte vi att TL074an används i många effektenheter och synthar. TL074 är en integrerad krets eller IC (Integrated Circuit) med 14 ben som består av 4 inbyggda operationsförstärkare och två ben för spänningsmatning. TL074 tillverkas av Texas Instruments. Den passar vår effektenhet perfekt sett till funktion, pris och implementering på

prototypkortet. All spänningsmatning sker med 9 V DC, enligt TL074s datablad rekommenderas en spänningsmatning på mellan 5-15 V DC, det vill säga att 9V passar perfekt in som rekommenderad matningsspänning. Utöver matningspänningen hanterar den småsignaler väldigt bra då den är en 'general-purpose' operationsförstärkare med låg brusfaktor.



Figur 2.24: TL074 Pin-diagram

Från figur 2.24 visas TL074s interna kopplingar. Den innehåller fyra individuella operationsförstärkare samt två ben för matningsspänning V_{cc}^{+} och V_{cc}^{-}. IC-kretsen TL074 är då en stor anledning att vår krets är så kompakt men effektiv.

2.2.6 Kretsdesign

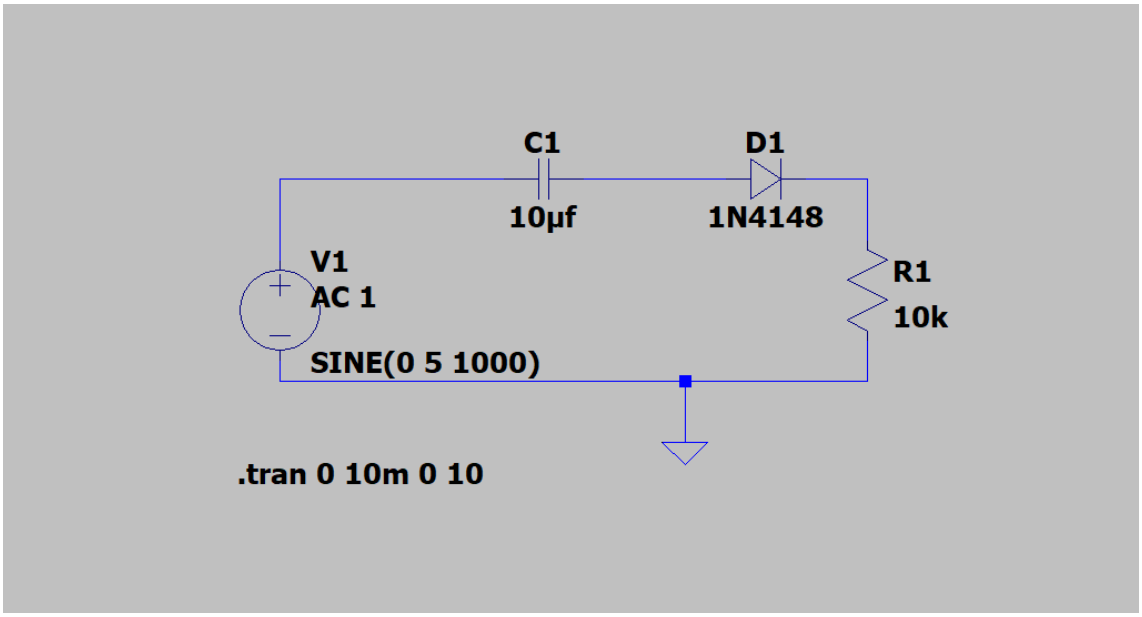
Programvarorna som användes i det här projektet var LTspice och DIYLC. LTspice är en kretssimulator gjord av Linear Technology och Analog Devices och är en av de största SPICE programvarorna som finns. DIYLC är en open-source programvara som används för att designa struktur på prototypkort. Båda programvarorna är helt gratis och var därför optimala val för det här projektet. LTspice har vi båda erfarenhet sedan tidigare i utbildningen vilket gjorde valet mycket mer självklart.

I det här projektet används först LTspice för att studera kretsar. Att lära sig funktionaliteten bakom liknande effektenheter är viktigt för att få förståelse för tekniken bakom effekterna.

Vid simulering av distorsionskretsar är det främst intressant att analysera signalen i tidsdomänen, det vill säga att signalen ritas upp som funktion av tiden. I LTspice är detta en 'Transient analysis'. Det är konkret att se hur mycket och på vilket sätt signalen distorderas när man kollar i tidsdomänen. Genom en sådan här analys kan vi då se hur signalen förvrängs med alla olika förstärkningsvärden. Att rita upp signalen i frekvensdomänen är också intressant då vi är intresserade av inflytandet

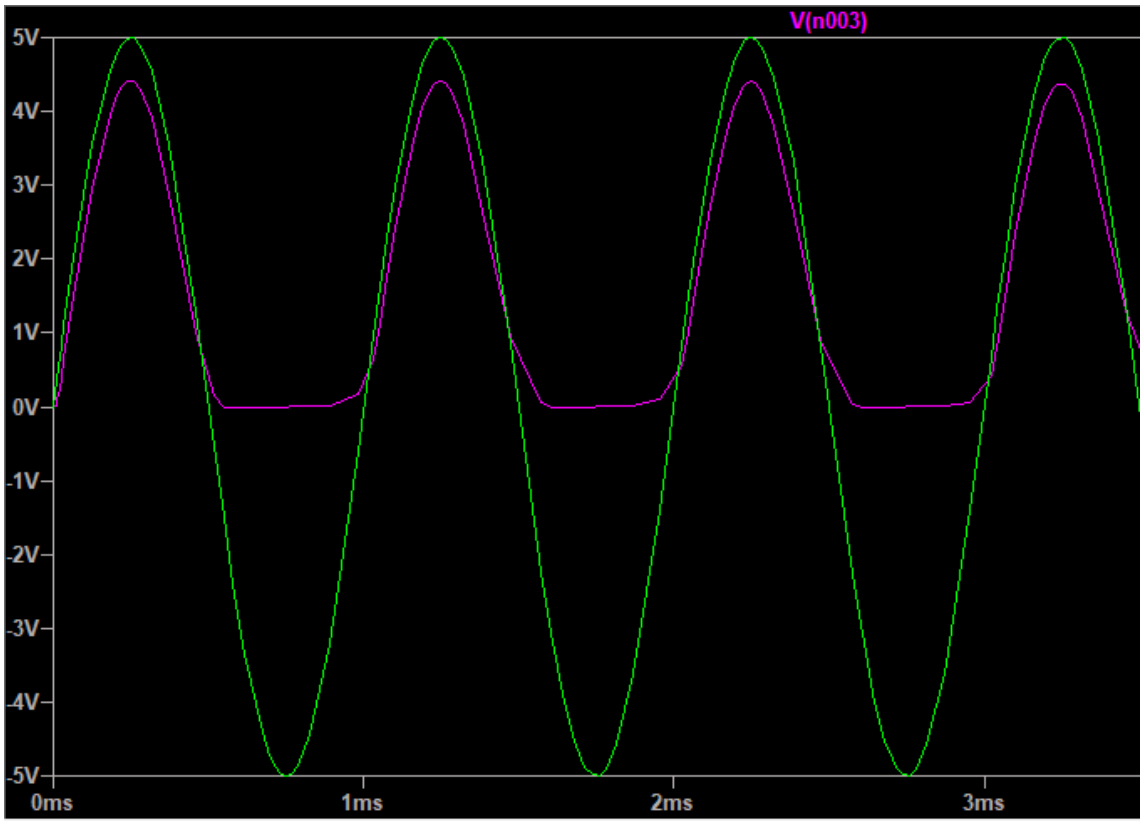
av övertoner. Detta definieras som en 'AC analysis' i LTspice. Frekvensdomänen visar hur starka övertonerna blir som funktion av förstärkningen.

Vid simulering av filterkretsar är det främst frekvensdomänen som är intressant. Det är genom frekvensdomänen vi kan se hur filtrets form ser ut. Parametrar som är intressanta är brytfrekvens, sluttning och eventuell resonans. Filtret kan formas smidigt med hjälp av en AC analysis. Design och komponentval kan smidigt simuleras fram tills filtret får önskad form. Det kan vara intressant att plotta signalen i tidsdomänen för att säkerställa att filtret fungerar som det ska. Filtret formar om signalen beroende på frekvens hos signalen och form på filtret, Hög- eller lågpass. En fyrkantsvåg formas till exempel om av ett lågpassfilter för att filtret blockerar snabba förändringar i signalen.
Figurerna 2.25, 2.26 och 2.27 visar exempel på simuleringar i LTSpice samt design i DIYLC. Dessa exempel har inget att göra med multieffektenheten.

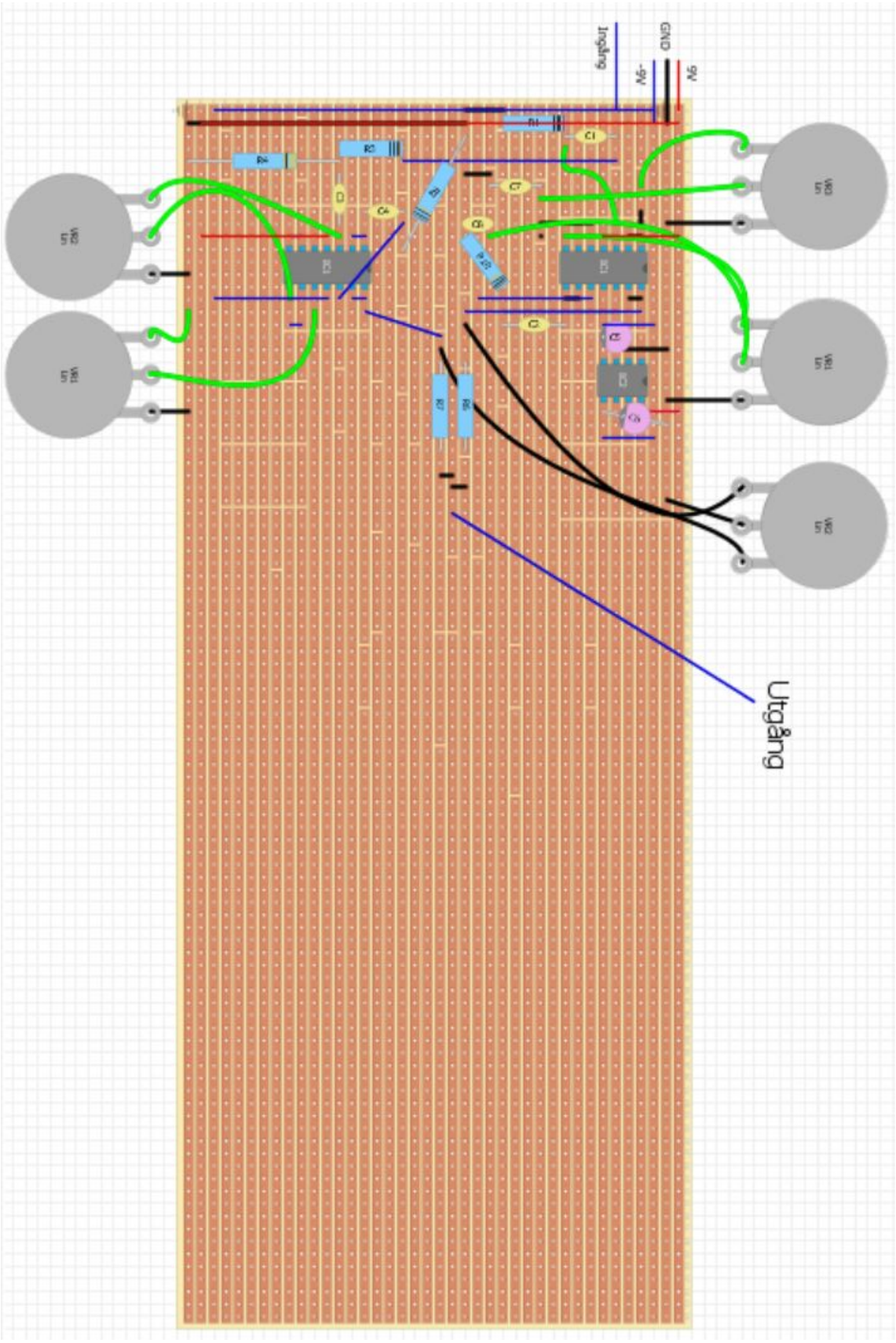


Figur 2.25: Ett exempel på en enkel krets

DIYLC är inte ett simuleringsprogram och används därför bara för att skissa upp prototypkortets design. DIYLC är ett smidigt och simpelt program. I det här projektet används DIYLC för att rita upp en mall att följa under lödningsprocessen. Att följa en mall gör processen smidigare och säkrare för eventuella fel. Det tillåter även för en logisk placering där det är enkelt att följa signalprocessen vid eventuell felsökning. Prototypkortet måste också förhålla sig till storleken på lådan. Med skisser kan vi säkerställa att den faktiskt gör det.



Figur 2.26: Simuleringsexempel från kretsen



Figur 2.27: Bild av filterkretsen i DIYLC

3
Metod

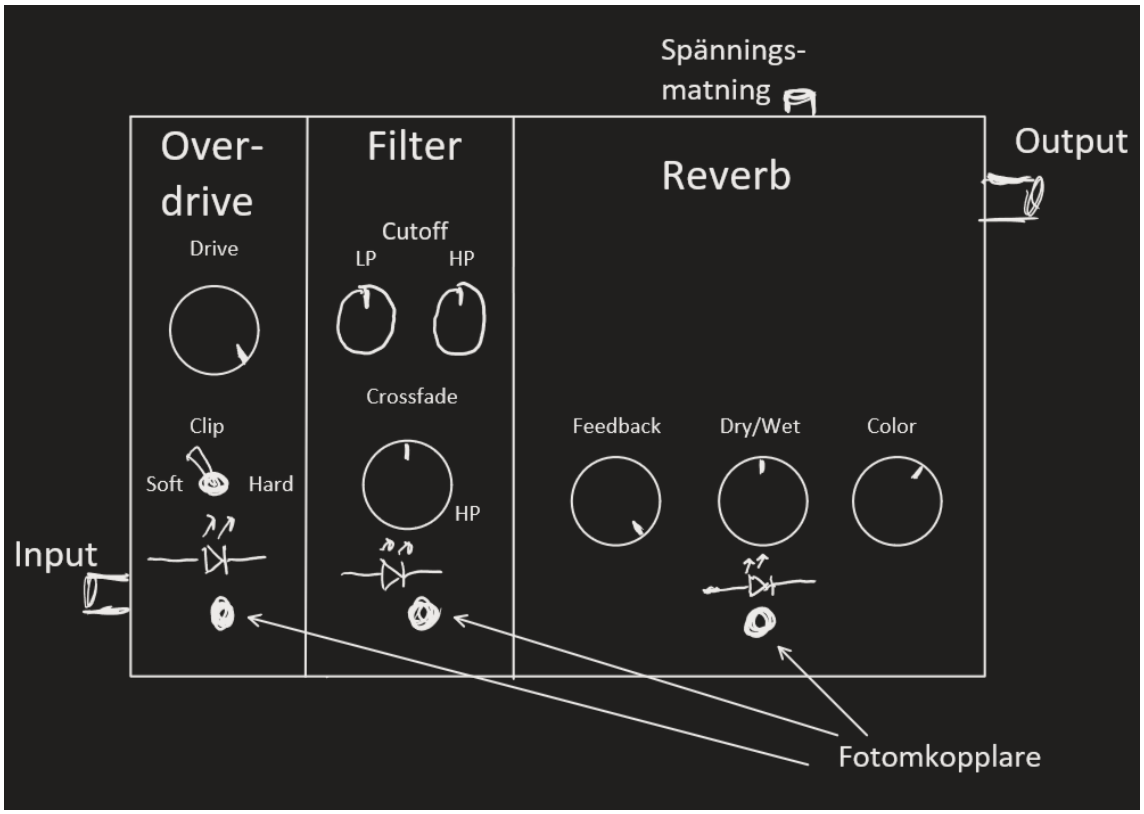
I följande kapitel presenteras tillvägagånssättet som använts under arbetets gång. Kapitlet är indelat i fyra delar; planering och förundersökning, analog effekter, efterklang- och DSPutveckling samt konstruktion och felsökning. Varje del presenterar en av de övergripande aspekterna av projektet.

3.1 Planering och förundersökning

Arbetet började med att dela upp enheten i de individuella effekterna och bestämma tillvägagångsätt för varje del. Samtidigt gjordes en tidsplan för att logiskt planera arbetet och dela upp det så mycket som möjligt. Enheten delades upp i tre delar, två analoga delar och en digital. En mängd virtuella tester gjordes i musikproduktionsmjukvaror såsom Logic Pro X, Reaktor 6 och Audacity. Detta för att se hur signalkedjan skulle se ut för att vara så välljudande och tillämpbar som möjligt. Med mängden gratis mjukvaror hjälpte det att få en idé om hur slutprodukten kan låta.
All förundersökning och litteratursamling skedde via google scholar och Chalmers digitala bibliotek Även med ett flertal elektronikkurser i ryggen behövdes många hål i kunskapen fyllas. För att få en bredare grund i hur elektroniken nyttjades Denton J. Daileys bok 'Electronics for Guitarists' [5] samt undersökning av en mängd kretsar från andra effektpedaler.

En skiss ritades upp för att planera hur strukturen skulle se ut på prototyplådan. Det viktigaste är att placera effekterna för att kunna få en bild av signalkedjan. På skissen placeras även andra relevanta delar ut så som ingång/utgång, DC-jack, potentiometrar, dioder och fotomkopplare.

Figur 3.1 är då skissen som används för utgångspunkt för effektenheten. Signalkedjan går från vänster till höger på figuren. Overdrive/distorsion är det första steget, efter det kommer filtret och till slut efterklangen.



Figur 3.1: Skiss för prototyplådan

3.2 Analoga effekter

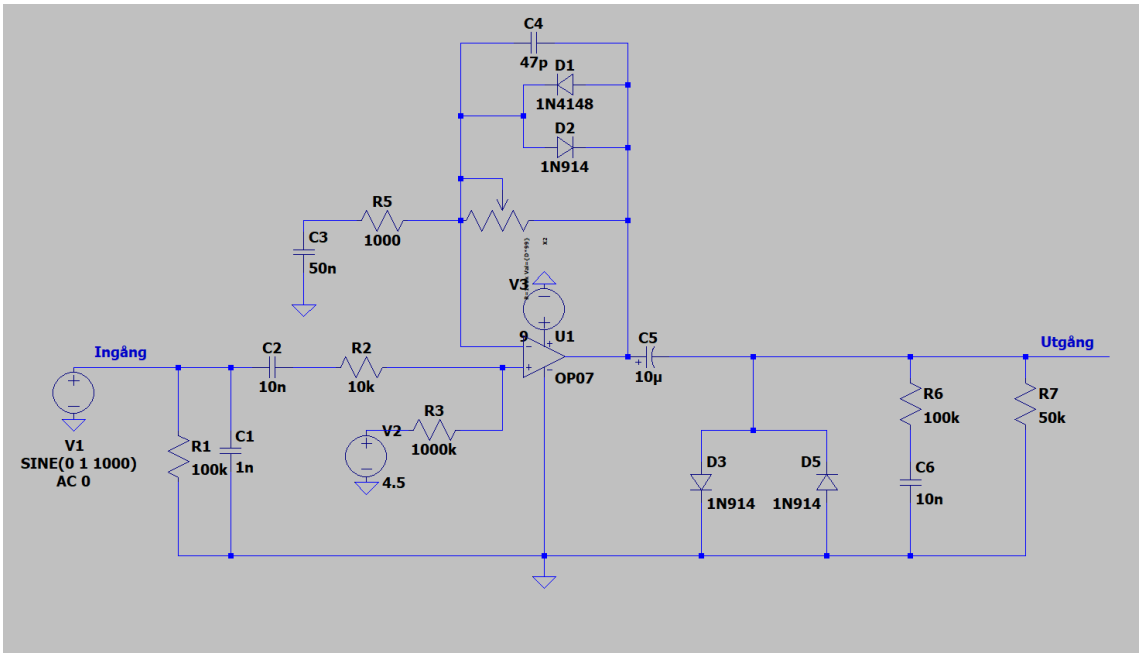
De två första stegen i enheten, filtret och distorsionen, är analoga. Studerandet av en mängd filter- och distorsionskretsar sker löpande med projektets gång. Det innebär att hela effektenheten eventuellt kan ändras när som beroende på nya idéer eller hinder. Att bestämma komponenter var således en viktig del i processen. Komponenterna bestämdes utifrån en fast kretsdesign, för att möjliggöra för eventuella förändringar i kretsdesignen beställdes en större antal komponenter än vad kretsen krävde för att ge rum för förändringar i design under teststadiet.

Det absolut första steget i utvecklingsprocessen för den analoga delen är att rita upp och simulera kretsar genom LTspice. Simuleringarna är väldigt konkreta och ger en tillräckligt bra bild över hur kretsarna fungerar i verkligheten. Dock så skiljer sig självklart simuleringar från verklighet på så sätt att simuleringar antar ideala förhållanden. Med hjälp av simuleringarna kan relevanta komponenter beställas in. Den större mängden komponenter beställs via stora och välkända distributörer som RS Components och Electrokit.

3.2.1 Distorsion

De två stora valen att välja mellan är transistordistorsion eller operationsförstärkardistorsion. Motivationen bakom att välja operationsförstärkare som distorsion var

främst för att kunna minimera mängden olika komponenter i systemet. Med insikten att spänningsföljare var kritiska för att isolera de individuella kretsarna och att filtrerna kunde dra nytta av operationsförstärkare i sin design. Inspiration för kretsen kommer främst från liknande kommersiella distorsionsenheter, men även från tidigare erfarenheter. Distorsionen från en transistor jämfört med en operationsförstärkare skiljer sig alltså inte märkbart och är i detta fallet subjektivt. Designen ritas upp i LTspice för att simuleras. Kretsen simuleras främst för testa distorsionens funktion. Flera olika kretsar simuleras då inspiration har tagits från flera källor och sambandet mellan kretsarna blir konkret. De essentiella delarna från varje testad krets sammanfogas till slut i en resulterande design:



Figur 3.2: Kretsschema för distorsionen

Kretsen är uppbyggd kring en operationsförstärkare. Komponenterna vid ingången av kretsen, $R1, C1, C2, R2$, bildar tillsammans ett hög- och lågpassfilter. De är där för att filtrera ut DC-offset och filtrera bort högre frekvenser som inte är nödvändiga.

Plusingången på operationsförstärkaren är kopplad till en spänningskälla på 4.5 V genom en 1 $M\Omega$ resistor. Detta är essentiellt för kretsen för att operationsförstärkaren är enkelmatad med endast $+9V$. En enkelmatad operationsförstärkare kan inte förstärka signaler under 0 V. Spänningskällan fungerar då som en DC-offset för att temporärt skjuta upp signalen så att den är centrerad runt 4.5 V. Distorsionen av signalen sker alltså när signalen är förskjuten i y-led.

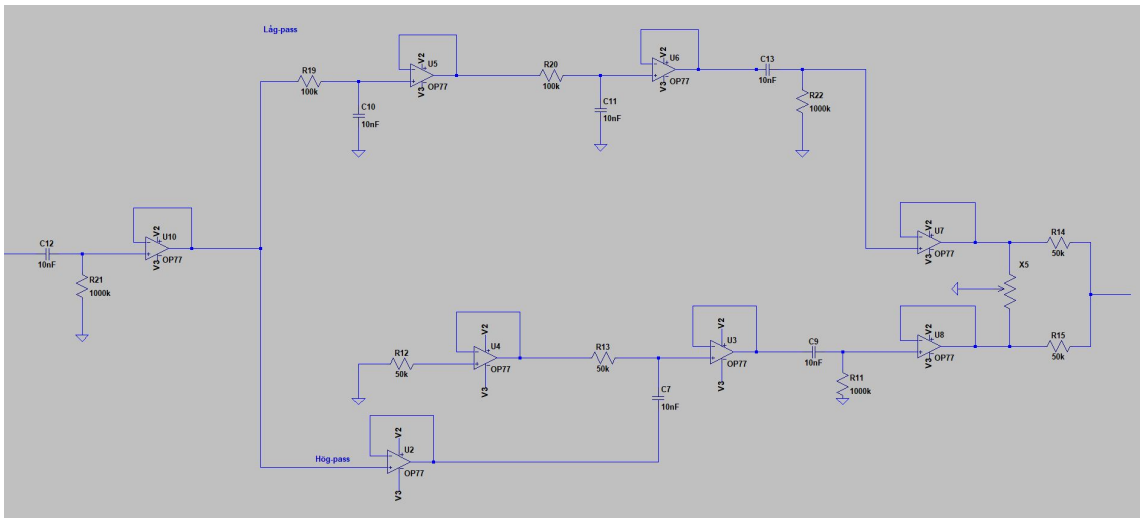
Det återkopplade steget fungerar som en förstärkning och ett mjukt klippsteg. Potentiometern fungerar främst som förstärkning för signalen. Tillsammans med $C3, R5$ och $C4$ fungerar den dock också som ett hög- och lågpassfilter. Dioderna $D1$ och

$D2$ utgör det mjuka klippsteget.

Vid utgången på operationsförstärkaren sitter en polariserad kondensator $C5$. Uppgiften för kondensatorn är att filtrera bort DC, det innebär att den filtrerar bort den 4.5 V likspänningen som adderades till signalen innan operationsförstärkaren. $D3$ och $D5$ är ett hårt klippsteg. Vid utgången på kretsen utgör $R6$ och $C6$ ett ytterliggare lågpassfilter. Resistorn $R7$ är i verkliga fallet ersatt med en potentiometer som utgör volymkontrollen för kretsen.

3.2.2 Filter

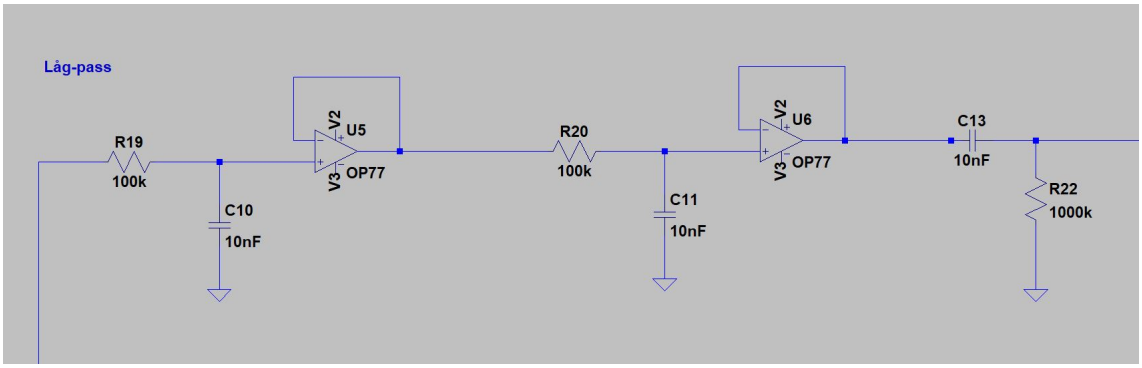
Filterdesignen var klurigare då tidigare kunskap inte var lika stor som för förstärkningen. Med hjälp av boken 'Linear circuit design' [21] utvidgades kunskapen om analoga filter i stort och designprocessen började. Möjligheterna var väldigt många och att hitta en filterdesign var det absolut första steget. Från början verkade 'state variable filter' som en bra implementering då det är brett använt i musikvärlden och ger mycket möjligheter. Hur som helst så ändrades designen kort efter. Istället valdes ett andra ordningens aktivt låg- och högpassfilter. Ett andra ordningens filter är ett simpelt men väldigt effektivt filter. En av Anledningarna till ändringen av design för filtret var för att avgränsa funktionen. Ett hög- och lågpassfilter räcker för enheten. Det innebär att state variable filtret är en överdriven implementering. Ett andra ordningens låg- och högpassfilter var då en mer passande krets för enheten. Likt distorsionskretsen så simuleras först kretsarna för att bestämma en färdig design med bestämda komponenter. Sedan testas kretsen på ett breadboard för att verifiera funktionen.



Figur 3.3: Komplett kretsschema för filtret

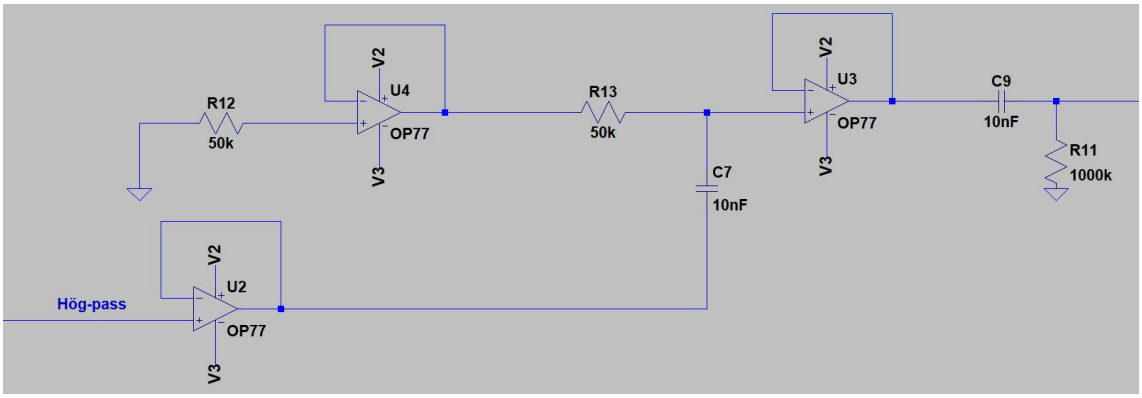
Insignalen kommer antingen direkt från distorsionskretsen eller från fotomkopplaren. För att isolera signalen från distorsionskretsen placeras en spänningsföljare och precis innan spänningsföljaren sitter ett högpassfilter för att filtrera bort eventuell

DC. Tanken är sedan att signalen delas för att skickas in i både hög- och lågpassfiltret samtidigt. Operationsförstärkarna i LTspice simuleringen är OP77. I den verkliga implementeringen används TL074.



Figur 3.4: Kretsschema för lågpassteget

Lågpassfiltret består av två identiska lågpass-steg isolerade från varandra med hjälp av spänningsföljare. Utan en spänningsföljare hade det andra lågpassteget påverkat det första steget och kretsen hade inte fungerat. Resistorerna $R19$ och $R20$ i figur 3.4 är i verkliga fallet en stereo potentiometer, det vill säga två potentiometrar inbyggd i en. Potentiometerns resistans är mellan 1-100 $k\Omega$ och kondensatorerna är båda 10 nF. Filtret formades enligt ekvation 2.10. Det kommer då att resultera i två gränsvärden för brytfrekvensen, en brytfrekvens när $R = 1\ \Omega$ och en brytfrekvens när $R = 100\ k\Omega$. Filtret är helt stängt när $R = 100\ k\Omega$. Absolut sist i lågpassfiltret sitter ett till högpassfilter för att filtrera bort DC.



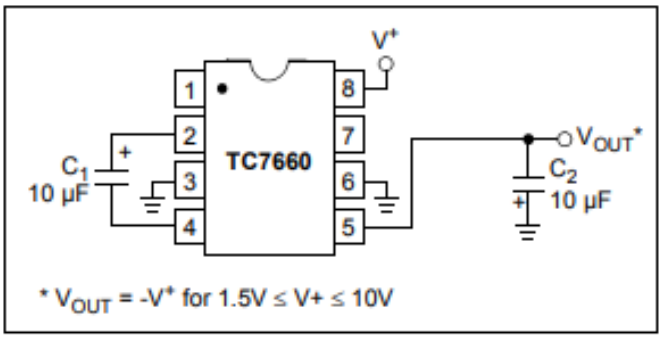
Figur 3.5: Kretsschema för högpass-steget

I högpassteget är även här $R12$ och $R13$ i verkliga fallet en stereo-potentiometer. Liknande princip gäller i detta fallet som i lågpassteget. Resistorerna bestämmer brytfrekvensen i samband med kondensatorn. $C7$ är nyckelkomponenten i den här kretsen och seriekopplas i signalflödets väg för att åstadkomma högpass effekten. $C9$ och $R11$ utgör även här ett DC-filter.

3.2.3 Spänningsmatning

För att elektronik ska fungera krävs spänningsmatning. Spänningsmatningen beror på kretsen och dess aktiva komponenter. Effektenheten består av flera olika delar, både analoga och digitala. Det vill säga att i detta fallet krävs olika spänningsnivåer för att mata olika delar av enheten. En stor del av projektet var att åstadkomma spänningsmatning genom vägguttag för att åstadkomma en mer verklig applicering av enheten. Den primära spänningen är 9 V, enheten ansluts därför genom en 9 V AC adapter till vägguttaget. Genom en DC-kontakt löddes två 9 V- och jordbussar på både över- och undersida av prototypkortet för att smidigt nå bussen oavsett komponent på kortet.

Operationsförstärkarna för filtrerna matas med ± 9 V och distorsionskretsen matas bara med 9 V. Således behövs ett sätt att omvandla 9 V till -9 V. Lösningen till detta problem är en IC-kretsen TC7660CPA. TC7660CPA är en spänningsomvandlare som kan användas för att invertera en spänning. En buss tvärs över prototypkortet skapas även för -9 V.

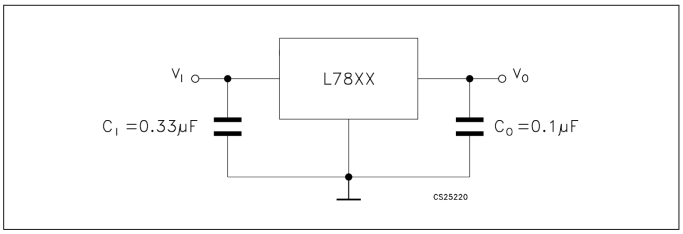


Figur 3.6: Kopplingsschema för inverterarkrets

Figur 3.6 illustrerar hur två polariserade kondensatorer på 10μF kan omvandla 9 V till -9 V.
Att styra den digitala delen med 9 V är inte möjligt. Enligt PJRCs datablad klarar Teensy endast av spänningar mellan 3.6-5.5 V. För att smidigt få ner spänningen från 9 V till 5 V användes spänningsregulatorn LC7805CV. En buss skapas för 5 V på den digitala delen av enheten. Det vill säga att bussen inte sträcker sig över hela kortet. Teensyn och lysdioderna är de enda som använder sig av den bussen, därför finns det inget behov av en 5 V-buss överallt. Via en 10 $k\Omega$ resistor kopplades 5 V mot lysdioderna och vidare till fotomkopplarna. Detta i syfte att tända dioderna då fotomkopplaren var i effektläge. I det fallet kopplades katoden på respektive LED mot jord och slöt kretsen för att tändas.

3.3 Efterklang och DSP

Den digitala delen av enheten styrs av en Teensy 4.0 och dess tillhörande ljudkort. Med hjälp av teensys audio GUI verktyg började efterklangsalgorithmen prototy-

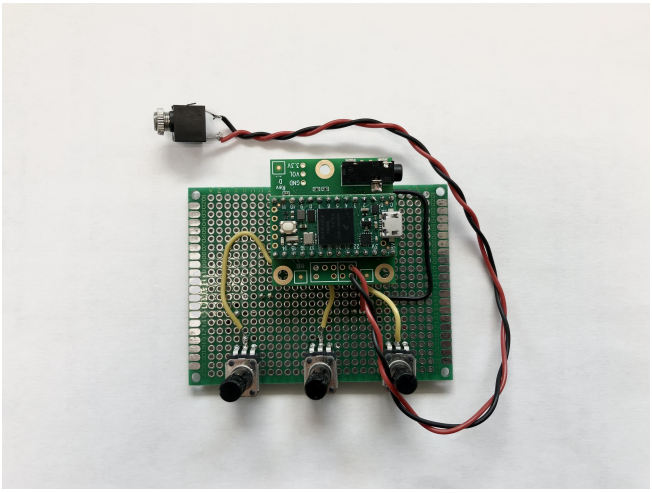


Figur 3.7: Kopplingsschema för en 5 V spänningsregulator

pas. Med hjälp av detta verktyg kunde hela signalflödet exporteras som en textfil som laddades in i arduinos IDE där alla initieringar och block funktion deklarerades. En mall till ett testskript skrevs för att göra implementeringen av olika varianter av algorithmen så smidig som möjligt. De variabla parametrar som skulle finnas tillgängliga på enheten var klara till ett antal på tre stycken och därav sattes delkareringen av tre portar för analogRead(); Se kod nedan.

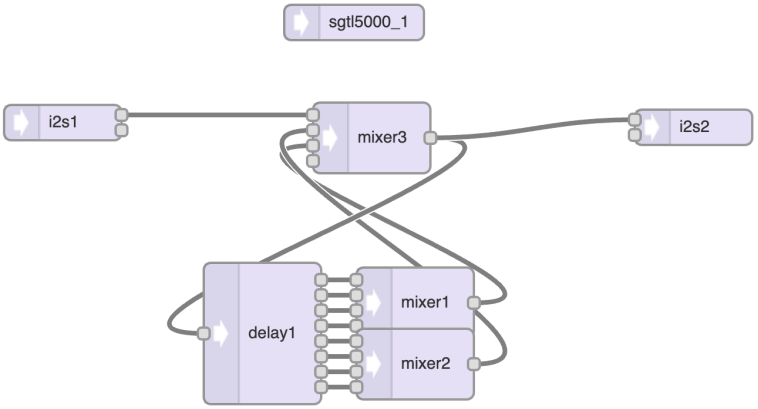
```
#define pot1            A1
#define pot2            A2
#define pot3            A3

float knob1 = (float)analogRead(pot1) / 1023.0;
float knob2 = (float)analogRead(pot2) / 1023.0;
float knob3 = (float)analogRead(pot3) / 1023.0;
```

I och med att analogRead() läser ett 10bitars tal delas det med 1023 för att få ut en float mellan 0 och 1. Detta gjorde implementeringen av potentiometernarnas värden lättare i koden senare.
Med detta byggdes ett provisoriskt testkort för att underlätta utvecklingen av efterklangsaloritmen. Se figur 3.8



Figur 3.8: Provisoriskt testkort för efterklangsprogrammeringen

När prototypriggen var monterad användes grundprogrammet upp för att börja testa olika varianter av algorithmen. Med inspirationen av Manfred Schroeders algoritm byggdes en enkel ekobaserad struktur upp i GUIn. Se figur 3.9.



Figur 3.9: Blockschema för ett efterklangssteg med 8 tappar använda

Att kortet var kraftfullt var känt men hur den skulle klara av en större mängd fördröjningstappar var okänt. PJRC definierar deras olika modellers maximala fördröjningsbuffer men inte Teensy 4.0 vilket användes i det här projektet. För att få ut en så realistisk efterklangseffekt som möjligt var den övre gränsen av kortets prestanda tvungen att testas och användas.

För att deklarera plats i minnet för en ljudbuffer kallas funktionen AudioBlocks(n); där n är antalet minnesblock som reserverasför fördröjningsbuffrarna. Antalet testades för att hitta en maxgräns. Detta på grund av att PJRC inte anger vad deras kort klarar av. En stabil maxgräns hittades vid n = 1300. Då varje block tillåter ungefär 2.9 ms av fördröjning ger det en total fördröjning på 3.77 sekunder. Detta blev dock snabbt för lite då många separata kanaler behövde spara ljud samtidigt. Ett alternativ var att börja med att lägga till en extern RAM-enhet för att utöka minneskapaciten av enheten. PJRC förser med ett kort "23LC1024 RAM memory chipsom tillför 128k RAM vilket Paul Stoffregren menar ger 1.48 sekunder extra fördröjningsbuffer [17]. Då detta kort inte fanns tillgängligt vid planering var en annan lösning tvungen att utvecklats.
Då varje fördröjningstap deklareras självständigt fanns stor möjlighet att skapa det myller av reflektioner som uppkommer i riktiga utrymmen genom att sätta upp villkor som slumpar varje taps tid i millisekunder. Med två delayblock och 8 taps vardera deklareras dom som nedan:

```
for(int i = 0; i < 8; i++){
    delay1.delay(i,random(100, 200));
    delay2.delay(i,random(50,100));
delay(20);
}
```

Denna idé med två parallellkopplade fördröjningsblock med 8 taps vardera som skapade ett fördröjningssteg skalades uppåt genom att seriekoppla dessa steg och summera i slutet. Detta gav större möjlighet till långa efterklanger då ett stegs

efterklang sedan skickades vidare till ett nästa steg för att dra ut på klangen. För varje steg ökades fördröjningstiderna något. Detta resulterade i en hög ekodensitet och en lång efterklang.
För att utvinna en varm och lång efterklang användes lågpassfilter efter varje steg för att successivt ändra frekvensresponsen för varje steg i följd. För varje steg blir efterklangssvansen längre så för att efterlikna ett varmt reverberande rum filtreras mer och mer av signalens höga frekvenser ut för varje steg. Detta hjälper också med minimeringen av flutter och transienter i efterklangssignalen.

Med detta skalades algoritmen genom att seriekoppla successiva efterklangs steg. Fördröjningskoeffecientdeklareringen optimerades för att få så lite transienter i signalen och få så lång efterklang som möjligt. För att undvika att tidskoeffecienterna är samma så deklarerades dom genom att slumpa flyttal inom ett spann, relativt mot utportens index.

```
    for(int i = 1; i < 9; i++){
    delay2.delay(i-1,(float)random(0.10,3*i));
    delay3.delay(i-1,(float)random(0.30,2*i));
    delay4.delay(i-1,(float)random(0.50,1.014*i));
    delay5.delay(i-1,(float)random(0.12,10*i));
    delay6.delay(i-1,(float)random(20,50*i));
    delay7.delay(i-1,knob1*400*i/5);
    delay8.delay(i-1,knob1*100*i/3.4);
    delay(10);
}
```

3.4 Konstruktion och felsökning

En stor del av slutstadiet i projektet är konstruktionen av prototypkortet. Den slutgiltiga designen ritas först upp i DIYLC. Kretsarna ritades upp i separata DIY-filer för att ha separerade mallar för varje krets. När respektive design var klar importerades ena designen in till den andra för att färdigställa helheten. Det viktiga var att planera rum för distorsionskretsen och filterkretsen. Då mikrokontrollern kräver lite plats på prototypkortet behövdes ingen egen mall. Mikrokontrollern samt spänningsmatningen löddes på fri hand.
Att löda på alla komponenter på prototypkortet är ett relativt enkelt men tidskrävande jobb som utfördes på Broccoli Engineering AB. Med noggrannhet i fokus löddes de tre effekterna samt spänningsmatning på. Då prototypkort ger stor risk för överlödningar kontrollerades alla lödpunkter för kortslutningar så att kretsen skulle ha önskad funktion. När prototypkortet var klart borrades hål för potentiometrar, lysdioder, matningsplug samt in och utgångar i lådan. Kortet monterades inte i lådan förrän hela systemet var bevisat att fungera.
Efter veckor hos på Broccolis kontor med lödande var mätningarna näst på tur. Mätutrustning sattes upp i en laborationssal på campus Lindholmen för att göra de slutgiltiga mätningarna. Fastän konstruktionen var noggrant planerad spenderades timmar på att felsöka då många funktionsfel upptäcktes. Överlödningar och felkopplingar undersöktes med multimeter och eliminerades därefter. Då fotomkopplarna

var inte helt pålitliga så varje del var tvungen att testas individuellt. Mätningarna separerades till varje individuell enhet och med nya mätpunkter på prototypkortet kopplades en signalgenerator och ett oscilloskop till kretskortet för att göra mätningar. Bilder togs av varje relevant mätpunkt.

Då vissa komponentplaceringar inte tog hänsyn till fotomkopplarnas djup föll idéen med att kunna stänga lådan helt. Designprocessen fokuserades mest kring prototypkortet vilket resulterade i att chassit var för litet för dess innehåll. På grund av denna felbedömning och tidsbrist avslutades arbetet med att göra fler mätningar och ljudinspelningar för att kunna analysera signalernas egenskaper i Audacity.

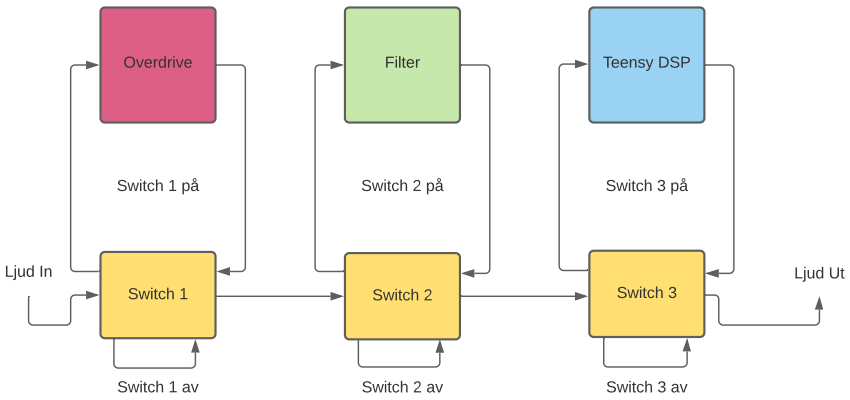
4

Resultat

I följande kapitel behandlas de resultat som arbetet slutade i. Alla övergripande delar får

4.1 Signalkedja

Efter virtuella tester av olika signalkedjor med de tre valda effekttyperna sattes denna som slutgiltig. Se figur 4.1.



Figur 4.1: Visualisering av signalväg mellan de olika blocken

Tanken med effektenheten var att den färdiga prototypen skulle innefatta prototypkortet monterat på metallådan. På metallådan skulle förutom prototypkortet, ingång och utgång för ljudsignalen, DC-kontakt, fotomkopplare och lysdioder, monteras. Ljudsignalen kommer in från vänstersida av enheten genom en 6,3 mm telekabel. Varje gul ruta i figur 4.1 representerar en fotomkopplare. Signalvägen genom effektenheten är på så sätt väldigt logisk och enkel att följa. Varje effekt aktiveras med ett enkelt knapptryck och en diod lyses upp för att indikera att en effekt är aktiv. Om vald effekt är inaktiv kopplas signalen förbi och dioden är släckt. Ljudsignalen är till slut tänkt att vidarekopplas från utgången till en högtalare eller gitarr/bas-förstärkare.

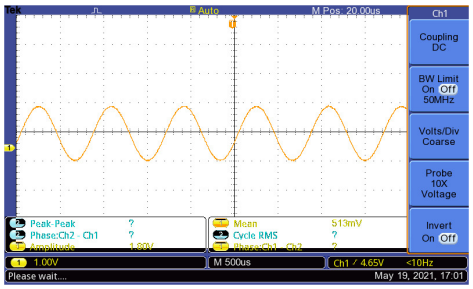
På grund av omfattande felsökning och tidsbrist kunde ej hela lådan monteras klart. Prototypkortet och effekterna fungerade både enskilt och tillsammans, Att färdigställa lådan skulle ta för lång tid. Mätdata och inspelningar var möjliga med hjälp

av diverse labb- och musikutrustning. På grund av fel i kretsens konstruktion testades aldrig alla tre effekter tillsammans utan bara en i taget. I följande mätdata testas vågor med frekvens $f = 1\ kHz$. Det är värt att notera att skalan på y-axeln, amplituden, skiljer sig på viss mätdata. På lågpassfiltret och efterklanget användes 500 mV som skala, på resterande mätdata användes 1 V.

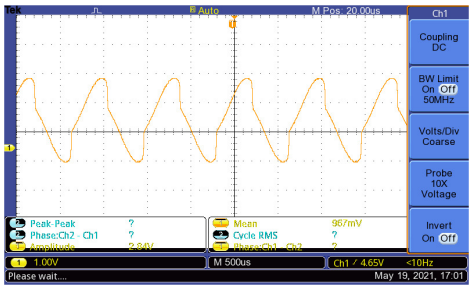
4.2 Distorsion

Distorsionseffekten är det absolut första steget i signalkedjan. Signalen ska distorderas, inflytandet av övertoner ökar och ljudet ska då bli skarpare. Detta åstadkommer vi genom att klippa signalen. Målet var att öka inflytandet av övertoner i signalen så mycket som möjligt. Exempelvis att komma så nära en fyrkantsvåg som möjligt, givet att insignalen är en sinusvåg.

Resultatet av distorsionskretsen blev ungefär planerat. Som tidigare nämnt är vår referens för att distordera en signal att forma om en sinusvåg till en fyrkantsvåg. Simuleringarna av den givna kretsen lyckades att distordera en sinusvåg till en fyrkantsvåg. Den verkliga implementeringen lyckades dock inte distordera signalen lika mycket som simuleringen. I figur 4.2 visas insignal till distorsionskretsen och i figur 4.3 visas utsignal. Signalen är förvrängd men inte som tänkt från början. Notera också att den distorderade signalen är förskjuten i y-led, det vill säga att den har fått en DC-offset. Detta är inte önskvärt då en DC-offset i ljudsignaler leder till oljud och kan medföra skada till utrustning.
Även fast kretsen inte fungerade exakt som i simuleringarna så gör det ändå det jobb den ska göra. Detta uppenbarar sig mycket tydligare i ljudtesterna av kretsen. Ökad förstärkning i kretsen leder till en distorderad signal.

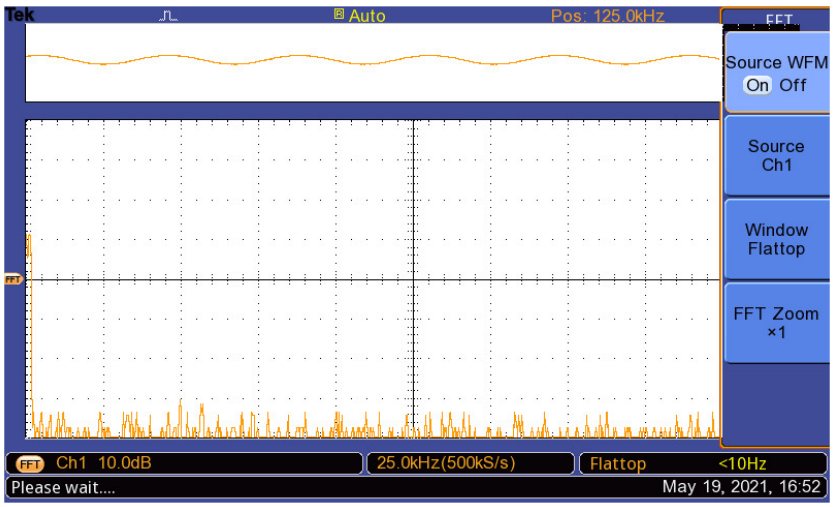


Figur 4.2: En ren sinusvåg

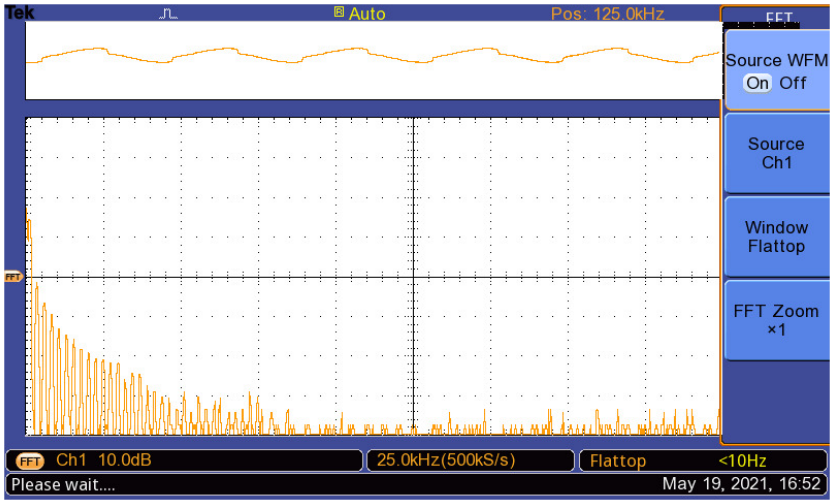


Figur 4.3: Den distorderade sinusvåg

I den här rapporten har vi tidigare betonat vikten av övertoner i musikalisk sammanhang. I figurerna 4.4 och 4.5 syns det även där att kretsen lyckats distordera signalen. Amplitudspektrat plottar en signals frekvenskomponenters amplitud. X-axeln är frekvensen och y-axeln är amplituden. I figur 4.4 är den fundamentala frekvensen f_0 den enda frekvensen av signifikans. I figur 4.5 syns det däremot att inverkan av övertoner ökat signifikant. Figurerna har även en mindre ruta högst upp där tidsfunktionen av signalen visas.



Figur 4.4: Frekvensrespons för en ren sinusvåg



Figur 4.5: Frekvensrespons för en Distorderad sinusvåg

4.3 Filter

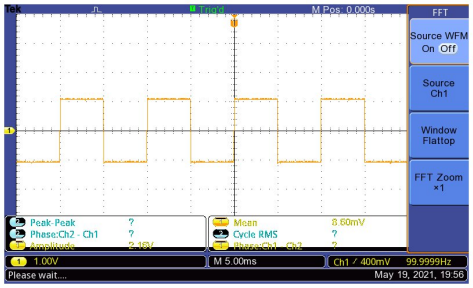
Efter virtuella tester och simuleringar resulterade designen i två separata filter, ett högpass och ett lågpass. Simpla kretsar med bra kontroll och skarp nog lutning. Ytterligare två passiva steg på varje filter hade möjligen adderats och på så sätt få en lutning på -24 dB/oktav, dock så räckte det med -12 dB/oktav för det här projektets tillämpning.

Tanken från början var att signalvägen skulle matas in till båda filtren samtidigt. Genom en potentiometer skulle det då finnas möjligheten att på utgången svepa

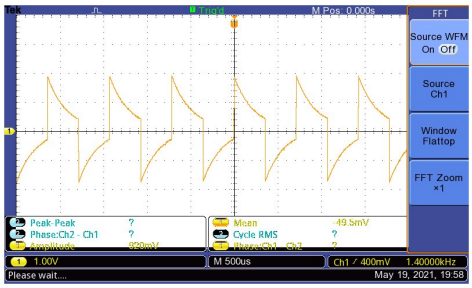
mellan vilket filter som ska vara aktivt. Det visade sig att vara svårare att implementera än förväntat. Filtrerna fungerade inte som tänkt och signalen försvann i princip helt. Under testningen fixades detta problem genom separera filterkretsarna och mäta på respektive utgång.

4.3.1 Högpassfilter

För att kunna visualiera effekten av ett filter är det en bra idé att testa en fyrkantsvåg. Anledningen till detta är för att det är väldigt konkret att kunna se förändringen i form när filtrets effekt ökar. I 4.6 visas en oscilloskop bild på en fyrkantsvåg tagen direkt från en funktionsgenerator. 'Amplitude' på fyrkantsvågen är dess topp-till-topp värde och ligger på 2.15 V. Topp-till-topp värdet på efterliknas en vanlig 'line-level' signal så mycket som möjligt. En line-levelsignal har amplitud på omkring 1 V. Definitionen av amplitud är halva topp-till-topp värdet och ska därför inte förvirras med oscilloskopets 'Amplitude'. Halveras då fyrkantsvågens topp-till-topp värde $2.15V/2 = 1,075V$, det innebär att amplituden är i princip den av en line-level signal.



Figur 4.6: En fyrkantsvåg



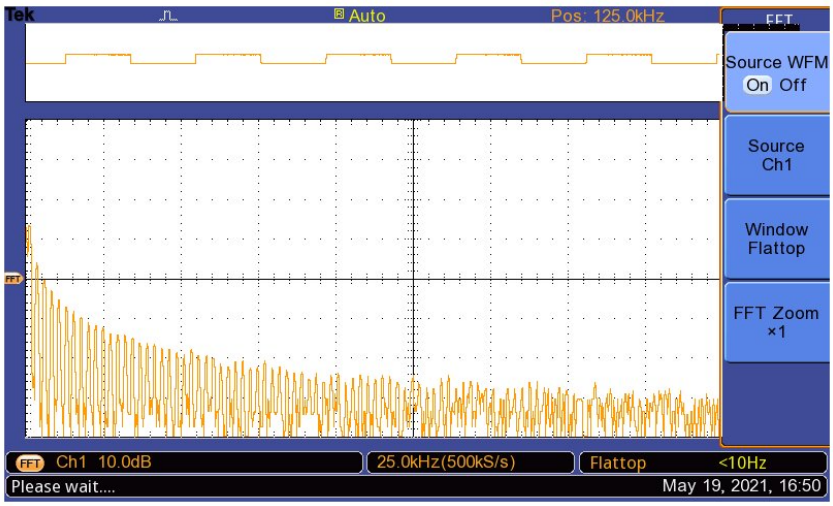
Figur 4.7: En högpass filtrerad fyrkantsvåg

I 4.7 syns inverkan av filtret på fyrkantsvågen. Jämfört med 4.6 så syns det att transienterna, det vill säga de snabba förändringarna i vågen, sticker ut mycket mer. En fyrkantsvåg skiftar från värde direkt och håller amplituden konstant den tiden den antingen är på sitt högsta eller lägsta värde. I den högpassfiltrerade fyrkantsvågen syns det att vågen istället tappar amplitud under den tiden då den vanligtvis är konstant. Anledningen till detta är att ett högpass filter alltid kommer blockera likspänning. Kondensatorer har en stor inverkan på effekten. Under tiden vågen förändras hinner kondensatorn laddas upp och när vågen sen blockeras laddas kondensatorn ur.

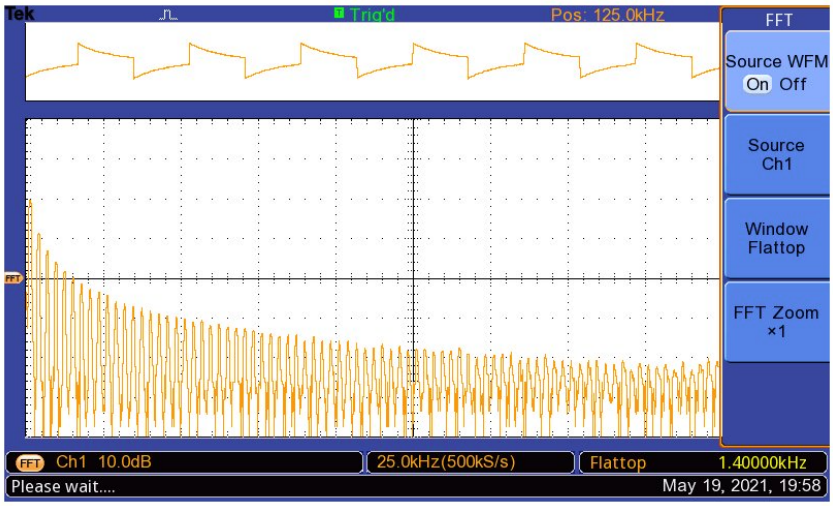
Med hjälpa av ett par aktiva högtalare testade vi även ljudkvaliteten av filtret. Filtret fungerade som förväntat då den gav effekten av ett högpass filter. Vi stötte dock på problem med stereo-potentiometern som styr brytfrekvensen hos filtret. När potentiometern vreds ner till dess lägsta värde så kortslöts kretsen vilket resulterade i störningar och oväsen. När potentiometern sveptes från minimum till maximum märke vi att resistansvärdet där filtrets effekt var maximerat låg på 50%, eller då 50 $k\Omega$. Ökade vi resistansen över 50 $k\Omega$ återgick signalen till sin ursprungliga form innan

den till slut kortslöts. Anledningen till detta problemet är oklart. Potentiometern kunde dock hållas i det effektiva omfånget och att lösa detta problemet var då inte nödvändigt för att kunna utföra lämpliga mätningar.

När man studerar filter är det då också relevant att kolla på amplitudspektrat av signalen för att kunna se filtrets påverkan på signalens frekvenskomponenter.



Figur 4.8: Övertoner av en fyrkantsvåg



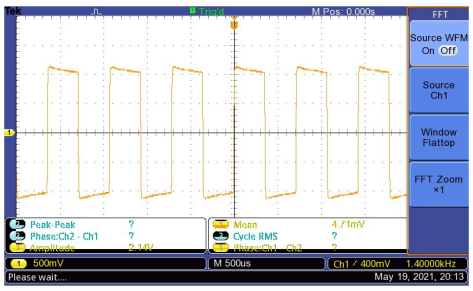
Figur 4.9: Övertoner av en högpassfiltrerad fyrkantsvåg

Figurerna 4.8 och 4.9 visar amplitudspektrat på en fyrkantsvåg och en högpass filtrerad fyrkantsvåg. En fyrkantsvåg består av många övertoner, det syns på figur

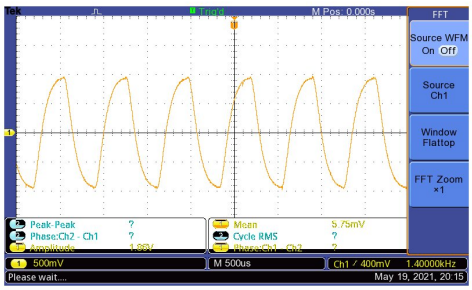
4.8. Jämför man figurerna syns det att figur 4.9 innehåller mer frekvenser av högre ordning.

4.3.2 Lågpassfilter

Lågpassfiltret testades också med en fyrkantsvåg. Som vi ser i figurerna 4.6 och 4.11 formas fyrkantsvågen också om signifikant. Skillnaden jämfört med högpassfiltret är att lågpassfiltret filtrerar bort de högre frekvenserna. I figur 4.11 syns det även i detta fall på förändringarna, främst när vågen skiftar från maxvärdet till minvärdet, eller vice versa. De snabba förändringarna i vågens struktur är bortfiltrerade och det resulterar i en våg som är mer lik en sinusvåg.

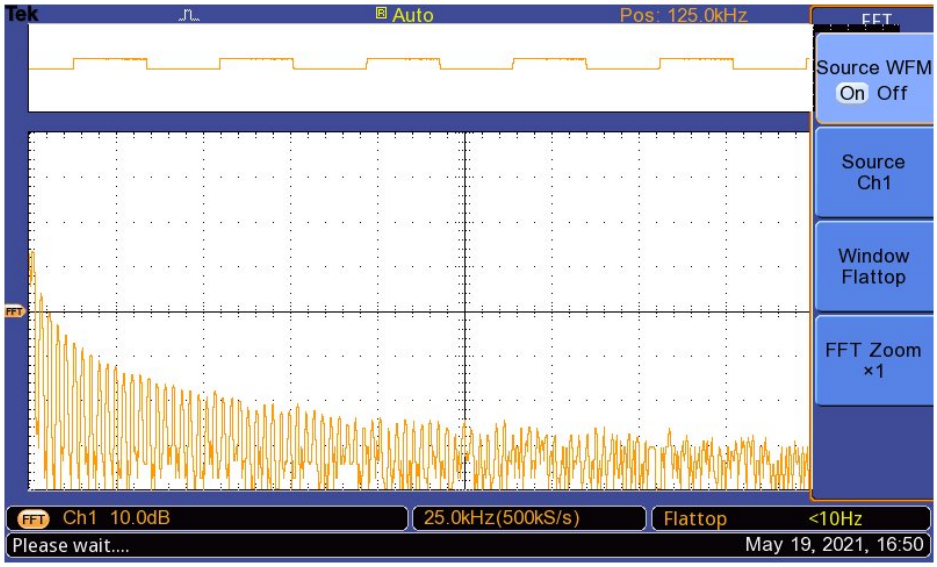


Figur 4.10: En fyrkantsvåg

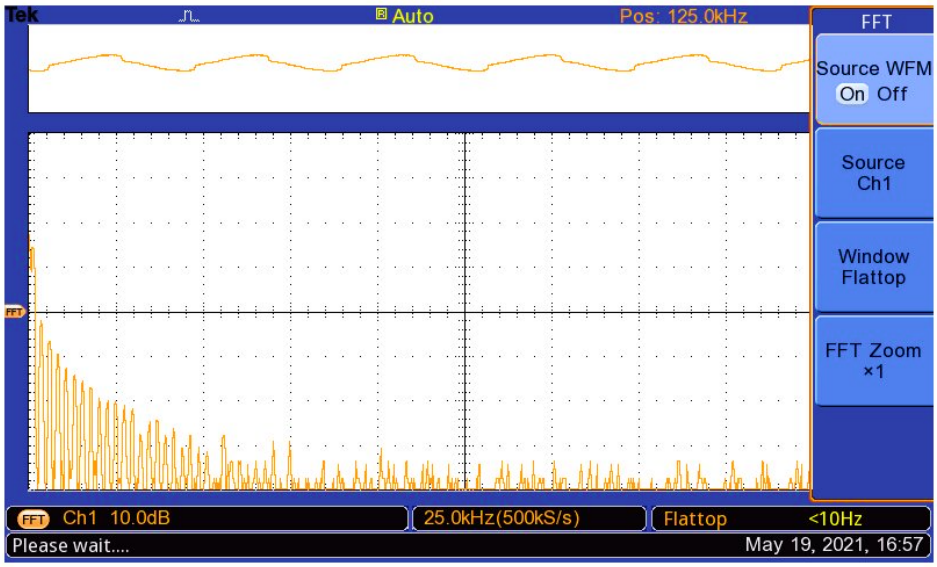


Figur 4.11: En lågpassfiltrerad fyrkantsvåg

Likt testerna av högpassfiltret är det också relevant att analysera amplitudspektrat vid tester av lågpassfiltret. Vid jämförelse av figurerna 4.12 och 4.13 syns skillnaden i övertoner hos signalen. Till skillnad från högpassfiltret syns det tydligt att lågpassfiltret istället dämpar frekvenser av högre ordning och släpper igenom frekvenser av lägre ordning.
Likt högpassfiltret stötte vi på problem med stereo-potentiometern då den kortsluts vid sitt lägsta värde, utöver det fungerade kretsen.



Figur 4.12: Övertoner av en fyrkantsvåg

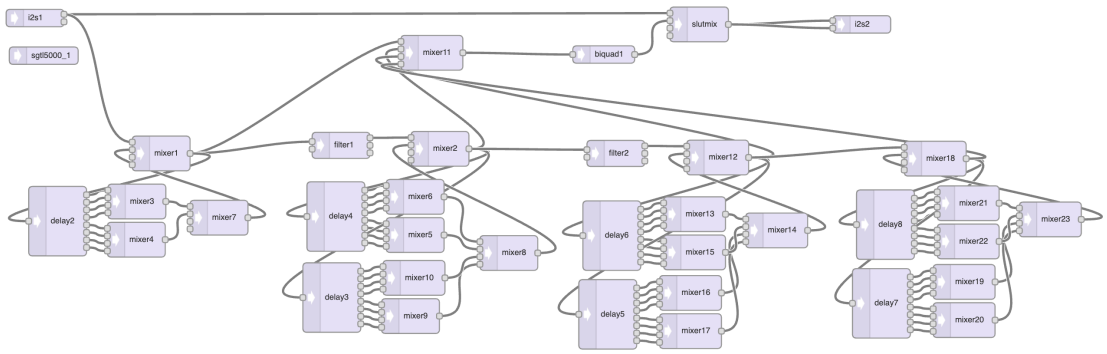


Figur 4.13: Övertoner av en lågpassfiltrerad fyrkantsvåg

4.4 Efterklang

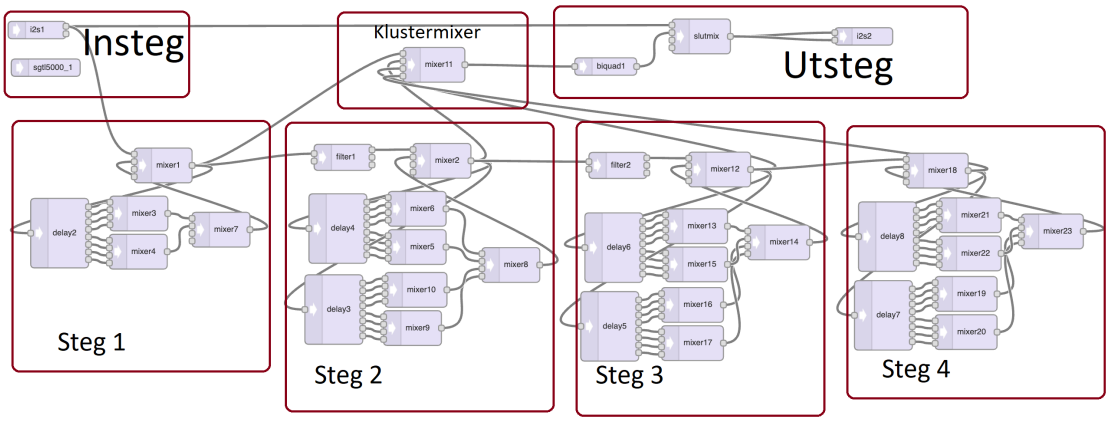
Efterklangsalgorithmen resulterade i en bekant struktur bestående av fyra seriekopplade fördröjnings steg där varje steg är en summering och återkoppling av 8 eller 16 tappar. Efter steg 1 och 2 filtreras en del av signalen för att minska tydligheten av signalens transienter och därav skapa en mer sammansatt efterklang. Detta efterliknar effekten av att ha tusentals aktiva reflektioner istället för ett par

hundra vilket är fallet i vårt system.



Figur 4.14: Slutgiltig signalschema för Efterklangsalgorithmen

Nedan är algoritmen indelad i de övergripande stegen för att visa hur signalflödet fungerar mer tydligt. Varje steg består av ett eller två fördröjningsblock vars signaler summeras och återkopplas för att skapa en loop. En potentiometer styr hur mycket signalen dämpas i återkopplingsingången och där av hur snabbt efterklangen försvinner.



Figur 4.15: Stegindelning av algorithmen

För att få koden att laddas upp och köras korrekt behövs vissa initeringar. Dessa görs i void setup() .

```
void setup() {

  Serial.begin(9600); \\kommunikation med kortet via USB

  AudioMemory(1300);     \\deklaration av ljudblock för
                         \\sparande av signal
```

```
  sgtl5000_1.enable();  \\Sätter igång ljudkortet
  sgtl5000_1.volume(0.7);   \\Sätter ljudnivån på utgången

  AudioInterrupts(); \\Tillåter avbrott i flödet av ljudsignalen
                     \\ifall kortet blir överbelastat eller
                     \\fallerar på något sätt.

}
```

Fördröjningstiderna i varje block ökar för varje successivt steg och kontrolleras i steg fyra med hjälp av en potentiometer. Detta för att kunna variera illusionen av rumsstorlek. Då en serie relativa primtal på så stora tidskoeffecienter som behövdes användes slumpade flyttal för att generera koefficienterna. Alla koefficienter skalas med dess index, i, för så att varje output på varje block ska ha en unik fördröjningstid.

```
for(int i = 1; i < 9; i++){
    delay2.delay(i-1,(float)random(0.10,3*i));
    delay3.delay(i-1,(float)random(0.30,2*i));
    delay4.delay(i-1,(float)random(0.50,1.014*i));
    delay5.delay(i-1,(float)random(0.12,10*i));
    delay6.delay(i-1,(float)random(20,50*i));
    delay7.delay(i-1,knob1*400 *i/5);
    delay8.delay(i-1,knob1*100*i/3.4);
    delay(10);
}
```

Återkopplingsmagnituden styrs av en potentiometer vars värde varierar den skalär som signalen minskas med varje reflektion. Nedan ser vi kod för hur ett av de fyra stegen summeras och återkopplas. Input 1 i mixer 2 bestäms då att ha en amplitud mellan 0.15 och 0.95. Detta för att signalen aldrig ska nå ett och skapa en förlustfri rundgång.

```
mixer2.gain(0, 1); // dry
mixer2.gain(1, 0.15 + (knob3 * 0.8)); // feedback

mixer8.gain(0,0.25);
mixer8.gain(1,0.25);
mixer8.gain(2,0.25);
mixer8.gain(3,0.25);
```

De två lågpassfiltrena kontrolleras med samma potentiometer som rumsstorleken. Brytfrekvenserna kan kontrolleras mellan 5000 Hz och 15000 Hz respektive 1000 Hz och 5000 Hz. Detta för att ge den längre delen av efterklangen mer värme och mindre skarpa reflektioner. Ju längre efterklang algoritmen tar desto mer av signalen kommer filtreras ut.

```
filter1.frequency(10000*knob1+5000);
filter1.resonance(1);

filter2.frequency((1-knob1)*4000+1000);
filter2.resonance(1);
```

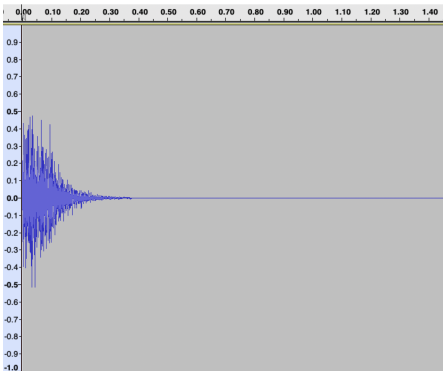
För att blanda mellan den rena signalen (Dry) och den behandlade (Wet) implementerades slutmixern, se figur 4.15. Med deklareringen nedan skapas en överbländning mellan den rena och behandlade signalen som efterliknar hur en ljudsignal uppfattas i ett rum med både den direkta och den reflekterade ljudsignalen. Detta så att om knob2 är på 0 kommer bara den behandlade signalen höras och om den går över till 1 så hörs bara den rena signalen.

```
//-----------Dry/Wet-Kontroll--------
    slutmix.gain(0,1-knob2);
    slutmix.gain(1,knob2);
```

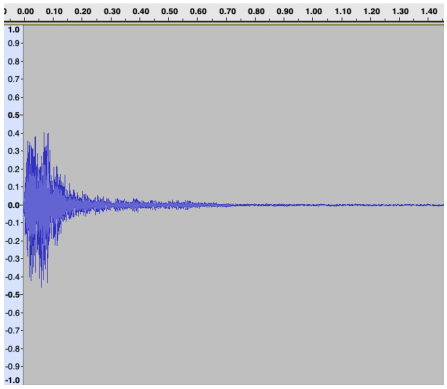
För att sammansätta alla signaler används teensys audio mixer. Följande block förenklar summeringsmixrarnas delkarering så att utsignalen ur alla är 1. Detta ger att alla utsignaler dämpas när dom blandas med andra för att aldrig förstärka signalen i sammankopplingarna.

```
float delaymix = 0.25;

for(int i = 0; i < 4; i++){
  mixer3.gain(i,delaymix);
  mixer4.gain(i,delaymix);
  mixer5.gain(i,delaymix);
  mixer6.gain(i,delaymix);
  mixer9.gain(i,delaymix);
  mixer10.gain(i,delaymix);
  mixer13.gain(i,delaymix);
  mixer15.gain(i,delaymix);
  mixer16.gain(i,delaymix);
  mixer17.gain(i,delaymix);
  mixer22.gain(i,delaymix);
  mixer21.gain(i,delaymix);
  mixer20.gain(i,delaymix);
  mixer19.gain(i,delaymix);
}
```

Genom att spela in ett ljudklipp och skicka det genom efterklangskretsen kan vi se hur vågformen ser ut. Ett ackord spelades på ett piano och i figur 4.16 representeras den vågform som genereras över 1.5 sekunder efter ljudets början. I figur 4.17 ser vi samma ljudsignal genom efterklangseffekten. Skillnaden är tydlig i figur 4.17 då man ser att signalens amplitud håller sig över en mycket längre tid. De senare efterklangstegen, steg 3 och 4, en signal som ekar kvar i nästan en sekund.
För komplett kod, Se bilaga A.1



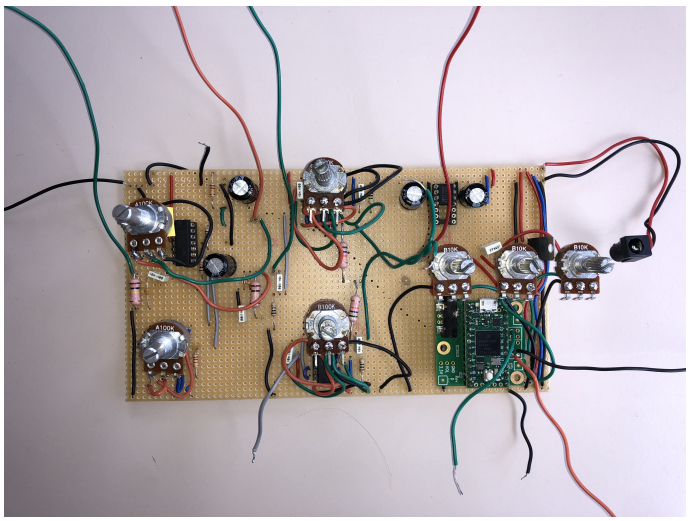
Figur 4.16: Vågform för en ren signal



Figur 4.17: Vågform för en behandlad signal

4.5 Konstruktion

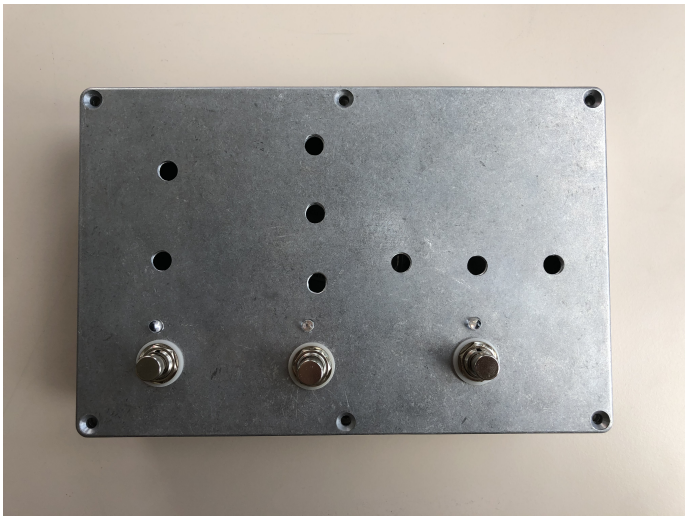
På ett prototypkort med kanaler byggdes kretsen efter de designer som gjordes i DIYLC. I figur 4.18 visas kretskortet i helhet. De lösa kopplingstrådarna i figur 4.18 är bestämda mätpunkter eller menade att lödas på mot panelen.



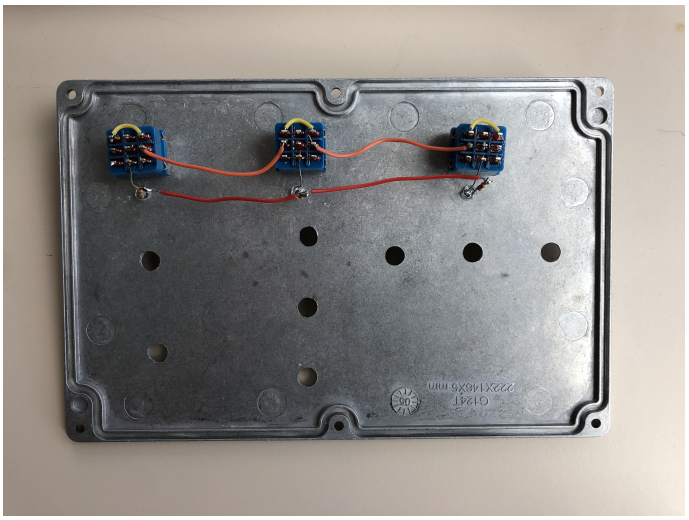
Figur 4.18: Bild av prototypkortet i färdig konstruktion

Figur 4.19 visar panelen till lådan köptes för att montera kretskortet i. Hål borrades för att placera potentiometrar i. Fotomkopplare och lysdioder är påmonterade I figur

4.20 ser vi baksidan av panelen. Där ser vi hur lysdiodernas katod, seriekopplade med varsin 10 kΩ resistor mot 5 V. Med detta lyser dioderna när effekten är påkopplad av fotomkopplaren.



Figur 4.19: Framsidan av lådan med fotomkopplare och aktivitets-LEDs monterade.



Figur 4.20: Baksidan av panelen med lysdioderna kopplade mot 5 V.

5
Slutsatser och diskussion

I detta kapitel kommer vi redogöra för resultatens kvalité samt hurvida arbetets syfte uppnåddes.

5.1 Slutsatser och förbättringsområden

Under det här projektet har vi lärt oss extremt mycket. Att få full kontroll över ett projekt och att få utforska tekniska lösningar med ett kreativt sinne har varit otroligt utvecklande och lärorikt. Syftet att designa en funktionell prototyp av en analog och digital multieffekt för musikinstrument uppfylldes inte i sin helhet. I planeringen la vi nog på för mycket olika saker som skulle byggas för att få den helt funktionell och med saker som den fysiska konstruktionen tänkte vi nog inte på hur klurigt det kunde vara. Under arbetets slutskede har vi diskuterat möjliga förbättringsområden på produkten. I den här delen kommer vi diskutera hur man kan utveckla enheten för det bättre. Nedan följer indelade förbättringspunkter på olika områden.

5.1.1 Efterklang

Teensyn är kraftig och resultatet är ändå tillfredsställande med tanke på omständigheterna. Dock finns stora möjligheter för att utveckla efterklangsalgoritmen.
På grund av att det inte finns jättemycket dokumentation eller litteratur kring signalarkitekturen i många efterklangsalgorithmer så var utvecklingen av vår algoritm tvungen att vara självdesignad i grunden. Och trots att Teensy 4.0 är ett kraftfullt kort behövs mer RAM-minne för att uppnå en efterklangseffekt av samma kvalité som de kommersiella produkter som finns på marknaden. Arduino som utvecklingsmiljö har stor potential men att bygga komplexa och väl-ljudande realtidseffekter är det inte det bästa för. Fastän man kan skriva egna DSP-bibliotek och köra dessa så kräver det en större mängd kunskap om programmering, datateknik och signalbehandling. Vår undersökning har fokuserat på hurvida man kan utveckla mer avancerade effekter med hjälp av en högnivåmiljö som Teensy Audio GUI. Svaret är ju att det självklart går. I detaljerna ser man dock att det är långt ifrån optimalt. Med arduino har utvecklaren inte den exakta kontroll över tidsparametrar som du behöver och i vårt fall blev detta en ekobaserad efterklang då tidsparametrarna var stora nog att kunna höra transienterna. För att komma åt den riktiga karaktäristiken behöver man kunna skapa många fler aktiva reflektioner simultant och ha möjligheten till att spara ljudet under längre för att få illusionen av en större

efterklang.
Fast än de analoga kretsarna är permanenta gör den öppna designen av den digitala effekten att en tredje part kan utveckla produkten efter dess behov. Med hjälp av Teensy Audio GUI och vår testkod är det enkelt att implementera vilket DSPsystem som helst. Genom micro-usb går det smidigt att programmera om Teensyn och de tre kontrollpotentiometrarna. Detta öppnar för skapa nya varianter där alla kan implementera de effekter och funktioner de vill för att få ut så mycket som möjligt av enheten.
En sak som diskuterades inledningsvis men inte fanns tid att implementera är en enkel display. De analoga parametrarna är intuitiva i grunden men när man pratar efterklang kunde en display som visar parametrarnas namn och värden. Detta hade varit simpelt med hjälp av Arduinos LCD-bibliotek. Att definiera efterklangens längd i sekunder och printa detta på en display hade varit effektivt för att få efterklangen att passa bra i en mix. Dock på grund av vår algoritms oförutsägbara natur är det svårt. Med vidare optimering av tidskoeffecienterna kan detta nog göras men detta var inte en prioritet i vårt arbete.

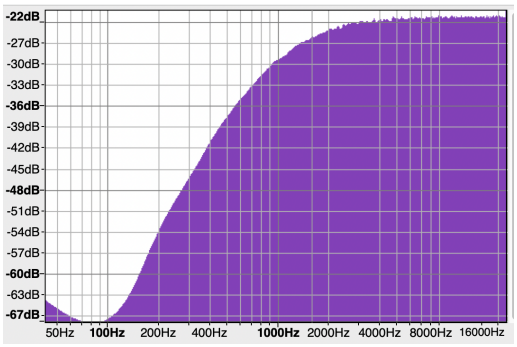
5.1.2 Distorsion

Distorsionskretsars syfte är i grund och botten att 'förstöra' signalen på ett tillfredsställande sätt. Med detta så finns många tekniker, kretsar och specifika komponentval som kan påverka. I vårt fall med den kretsen vi designat används dioder som klippingsteg. I efterhand inser vi att valet av diod, 1N914, inte var det mest effektiva. Genom att byta ut en eller ett par av de dioderna mot något som en germaniumdiod som ofta används i de effekter som klipper signalen hårdare och skapar en skarpare, ännu mer distorderad signal. Genom att sätta germaniumdioder i efter operationsförstärkaren för att klippa signalen hårdare och implementera en enkel brytare mellan signalvägen och klippingsteget skulle detta ge mer interagering mellan användaren och enheten.
Transistorsteg och vakuumrör är två alternativa förstärkningsmetoder med väldigt intressanta effekter. Transistorförstärkare och distorsionskretsar är mycket vanliga och hade varit ett lika bra alternativ ur ljudperspektivet men att bygga hela den analoga kretsen kring en och samma operationsförstärkare var en spännande utmaning. Vakuumrör är kända för att ge en väldigt tillfredsställande klang när dom drivs över sin gräns. Vakuumrören är dock dyra och inget man kan köpa på alla elektronikdistributörer.

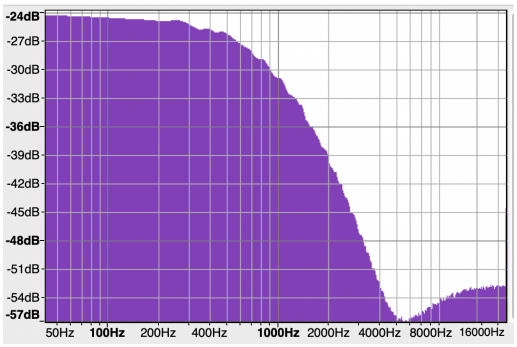
5.1.3 Filter

De två olika filterstegen var bra men var inte utan sina problem. För att få ut ett så intressant filter som möjligt separerades signalen och skickades till ett lågpass- och ett högpassfilter. Sedan summerades dessa signaler i en enkel överbländningskrets kontrollerade av en potentiometer. Detta visade sig inte fungera då sammanslutningen av filtrernas utsignaler påverkade dess funktion negativt. Då vi såg detta för sent valde vi att separera högpass filtret och lågpassfiltrets utgångar och mäta dom individuellt. Idén med överbländaren var en spartansk liten krets som kortslöt den

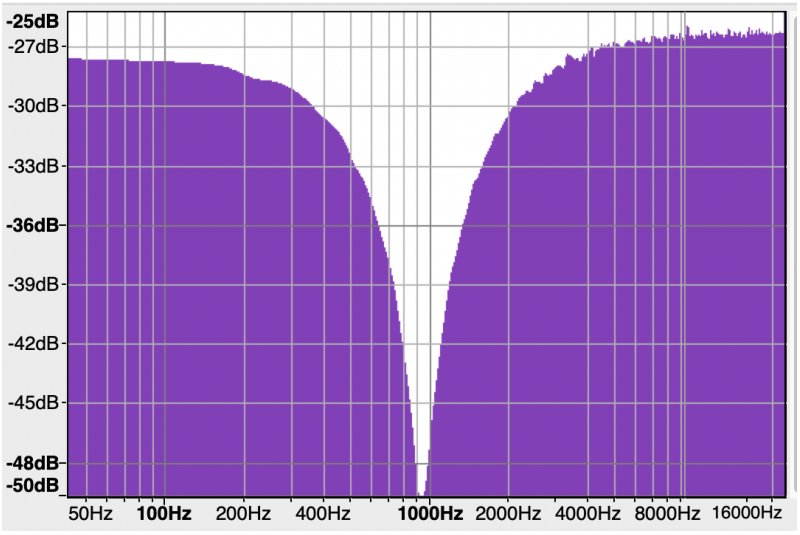
ena ingången eller den andra beroende av potentiometerns position. Om mer tid hade funnits hade vi kunnat applicera en mer pålitlig krets som uppfyllt det syfte som var menat. Annars, för att göra användarvänligheten bättre, hade en enkel switch mellan de två olika filtrernas utsignal kunnat implementeras enkelt. Detta hade inte gett samma bredd i klang men hade varit en rimlig uppoffring då de flesta multifilter man hittar i musikutrustning är diskreta, dvs att de är antingen högpass eller lågpass.
För att undersöka hur den här designen skulle kunnat se ut i en spektrumanalysator byggdes en virtuell version av filtersteget i Native Instruments Reaktor 6. Tre olika ljudfiler spelades in och analyserades sedan i Audacity.



Figur 5.1: Frekvensrespons för ett 2-poligt högpassfilter



Figur 5.2: Frekvensrespons för ett 2-poligt lågpassfilter



Figur 5.3: Summering av ett 2-poligt högpass- och lågpassfilter vid brytfrekvens 1000 Hz

Dessa simuleringar i figur 5.1 och 5.2 visar hur ett högpass och ett lågpass ser ut individuellt och 5.3 visar dess summering. Att ha möjligheten att blanda de två filtrerna är en spännande idé som tyvärr inte dyker upp i de flesta syntar men kan vara otroligt kraftfullt ur ett ljuddesignperspektiv.

5.1.4 Konstruktion

Den slutgiltiga produkten blev tyvärr inte komplett. Med fokus på kretsdesignen och funktionstestning insåg vi att den kretsen vi designat, med alla periferier såsom potentiometrar och kontakter, inte fick plats i lådan vi köpt. Chassistorleken var en rimlig begränsning då den hade samma höjd som de flesta effektpedaler har på marknaden. Därav skulle den inte sticka ut rent fysiskt. Detta blev ett problem när slutprodukten monterades i och vi insåg att det inte fanns plats nog för att kunna stänga lådan helt. Många metoder hade hjälpt för att undvika det problemet så som att använda ett CAD-program för att designa en PCB (Printed Circuit Board) och beställa den för att montera komponenter själva. Detta hade varit mer pålitligt och mer platseffektivt. Att designa en egen PCB hade dock varit dyrt med en beställning på endast ett kort och tagit tid att få levererat. Med syftet att jobba mot DIY och hobbyister kändes det mer rätt i slutändan att göra den slutgiltiga konstruktionen på prototypkort. För att konstruktionen ska vara komplett behövs såklart förändringar i komponentplaceringar göras, alternativt byta till en djupare låda då den största platsbristen kan undkommas med mer utrymme i z-led.

Tack vare spänningsregulatorn, LC7805CV, kunde vi enkelt få ut en stabil 5 V källa för att driva teensyn. Dock upptäcktes att denna utsöndrade en hög värme när strömmen var inkopplad. Denna värmen är helt normal men inget åtråvärt. Med hjälp av att till exempel addera en kylfläns eller limma komponentens kropp mot chassit hade vi enklare skingrat värmen bort från kretskortet och minimerat risken för att kopplinsgstrådar brinner eller att värmen i lådan blir för hög. En kylfläns hade också gynnat Teensyns processor chip. Vid körning av våra mest krävande versioner av algoritmen märkte vi att den blev ytters varm väldigt snabbt. Fastän detta inte påverkade kortets funktion så kan detta skapa problem i längden om värmen är hög för länge.

5.2 DIY

Under de senaste 10 åren har kulturen kring DIY växt bland musiker och producenter och därav har fler och fler helt open source produkter kommit till internets alla hörn. Att tillföra till en marknad där konsumenter får ett alternativ mellan inköp av en produkt eller möjligheten att köpa delarna själv och bygga den är något som vi ser som en bra och nyttig utveckling av marknaden. Många stora företag, så som Behringer och Korg, har gjort steg för att sänka priset på sina produkter men hålla kvar kvaliteten. Detta är såklart en bra sak men dessa produkter är fortfarande stängda och de begränsade resurser som finns för att lära sig musikalisk DSP som hobbyist gör det svårt för musiker att förstå vad som händer i bakgrunderna när dessa maskiner låter. Ett företag som jobbat passionerat för att göra intressanta instrument tillgängliga för massorna är Mutable Instruments [6] vars katalog av euroracksynthmoduler är bland de mest kreativa och fascinerande på marknaden. Fastän deras moduler kostar mellan 1000 och 6000 kr släpper dom alla produkter helt open source för att folk ska kunna bygga dom själva om de inte har pengarna att köpa de färdiga eller bara vill lära sig och spara en slant på vägen.

Med en effektpedal som vår kan man utforska Teensys intuitiva bibliotek och lägga till vilken sorts effekt man vill ha och lära sig DSP på ett mer lättillgängligt sätt. Mängder av resurser finns för enklare utveckling med Teensy samt mer kraftfulla och djupgående utvecklingsmiljöer såsom Mozzi, Faust eller Supercollider.

5.3 Miljö och etik

Då effektenheten i det här projektet jobbar med småsignaler och minimala strömmar är energiförbrukningen väldigt låg. Den främsta miljöpåverkan kommer då från materialvalet och beställningar.
Med open source decentraliseras marknaden genom att fler lär sig att producera utrustning, i större eller mindre skala, samtidigt som konsumenterna får ett val mellan att köpa en färdig produkt och stödja företagen eller göra det själv. Genom att göra det själv minskas industriella utsläpp genom att komponenter går direkt till konsument. Då produktionen av effektenheter decentraliseras och konsumenter börjar bygga sina egna enheter innebär det att konsumenterna i större utsträckning kommer vända sig till lokala distributörer av komponenter. Det innebär att effektenheter kommer att bli mestadels tillverkade av komponenter och material som kommer direkt från lokala distributörers lager. Det innebär att massproduktionen minskar och då också företagens stora beställningar av PCB-kort och annat material som skeppas världen över.

I vårt fall kan man smidigt programmera om den digitala delen, Teensyn, genom att ansluta den genom USB till en dator. Detta öppnar upp oändliga möjligheter att ändra effekt beroende på tillämpning. Det innebär att Teensyn i princip kan innehålla alla sorters effekter och kan ändras snabbt och smidigt, detta leder i sin tur till en minskning i industriproduktion och miljöpåverkan då konsumenten inte behöver köpa en enhet för varje önskad effekt. Och fastän den ofta tydliga skilladen i den ljudmässiga kvaliteten mellan digitala och analoga effekter så sker utveckligen i en sådan fart att de två metoderna kommer vara nästintill identiska. På grund av att digitala ljudeffekter har fördelen att kunna spara inställningar och vara helt baserade i mjukvara är de rent praktiskt sett att föredra.
Inte bara är det bra för miljön. Med fler mindre aktörer som utvecklar sina egna effekter sprids kunskapen och intresset på ett helt nytt sätt. Det bidrar till en mer spännande och utvecklande marknad.

Litteratur

[1] G. Brown. *A History of Reverb in Music Production*. URL: https://www.izotope.com/en/learn/a-history-of-reverb-in-music-production.html. (Åtkommen: 01.05.2021).

[2] M.G Christensen. *Introduction to Audio Processing*. Springer, New York, 2019. ISBN: 978-3-030-11780-1.

[3] *Comb Filter*. URL: https://en.wikipedia.org/wiki/Comb_filter. (Åtkommen: 04.06.2021).

[4] Creative Commons. *Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0)*. URL: https://creativecommons.org/licenses/by-sa/3.0/. (Åtkommen: 22.05.2021).

[5] D.J. Dailey. *Electronics for Guitarists*. Springer, New York, 2013. ISBN: 978-1-4614-4087-1.

[6] Mutable Instruments. *Mutable Instruments*. URL: https://mutable-instruments.net/about/. (Åtkommen: 24.05.2021).

[7] Jezzamon.com. *An Interactive Introduction to Fourier Transforms*. URL: https://www.jezzamon.com/fourier/. (Åtkommen: 17.05.2021).

[8] B. Kevius. *Delbarhet*. URL: http://matmin.kevius.com/delbar.php. (Åtkommen: 12.05.2021).

[9] Mackie.com. *What is Clipping?* URL: https://mackie.com/blog/what-clipping. (Åtkommen: 27.05.2021).

[10] Sam Ash Music. *Distorsions-historia*. URL: https://www.samash.com/spotlight/the-history-of-guitar-distortion-how-the-most-important-guitar-effect-came-to-be/. (Åtkommen: 22.06.2021).

[11] M.R. Schroeder och B.F. Logan. "Colorless artificial reverberation". I: *IRE Transactions on Audio* Vol. 9 Issue 6 (1961), p209–214.

[12] Philips Semiconductors. *I2S bus Specification*. URL: https://www.sparkfun.com/datasheets/BreakoutBoards/I2SBUS.pdf. (Åtkommen: 11.05.2021).

[13] J.O. Smith. *Choice of Delay Lenghts*. URL: https://ccrma.stanford.edu/~jos/pasp/Choice_Delay_Lengths.html. (Åtkommen: 07.05.2021).

[14] J.O. Smith. *Schroeder Reverberators*. URL: https://ccrma.stanford.edu/~jos/pasp/Example_Schroeder_Reverberators.html. (Åtkommen: 22.05.2021).

[15] J.O. Smith. *Schroeder Reverberators*. URL: https://ccrma.stanford.edu/~jos/pasp/Schroeder_Reverberators.html. (Åtkommen: 01.05.2021).

[16] P Stoffregen. *AudioEffectDelay*. URL: https://www.pjrc.com/teensy/gui/?info=AudioEffectDelay. (Åtkommen: 01.05.2021).

[17] P Stoffregen. *Maximum Delay for Teensy 4.0*. URL: https://forum.pjrc.com/threads/58295-Maximum-delay-for-Teensy-4-0. (Åtkommen: 04.04.2021).

[18] P Stoffregen. *Teensy*. URL: https://www.pjrc.com/teensy/. (Åtkommen: 01.05.2021).

[19] Izotope Educational Team. *What is Reverb?* URL: https://www.izotope.com/en/learn/reflecting-on-reverb-what-it-is-and-how-to-use-it.html. (Åtkommen: 23.05.2021).

[20] U. Tietze och C. Schenk. *Electronic circuits: handbook for design and applications: with 1771 figures and CD-ROM*. Springer, 2008. ISBN: 978-3540004295.

[21] Hank Zumbahlen, utg. *Linear Circuit Design Handbook, 2008*. URL: https://www.analog.com/en/education/education-library/linear-circuit-design-handbook.html. (Åtkommen: 20.05.2021).

A

Bilagor

A.1 Arduinokod

```
#include <Audio.h>
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <SerialFlash.h>

// GUItool: begin automatically generated code
AudioInputI2S            i2s1;           //xy=1028.000005722046,252.99999570846558
AudioEffectDelay         delay1;         //xy=1094,603.9999923706055
AudioMixer4              mixer4;         //xy=1230,635.9999923706055
AudioMixer4              mixer3;         //xy=1231,569.9999923706055
AudioMixer4              mixer1;         //xy=1273,486.99999237060547
AudioMixer4              mixer7;         //xy=1373,597.9999923706055
AudioFilterStateVariable filter2;        //xy=1506,472.99999237060547
AudioEffectDelay         delay3;         //xy=1541,735.9999923706055
AudioEffectDelay         delay2;         //xy=1548,595.9999923706055
AudioMixer4              mixer9;         //xy=1677,767.9999923706055
AudioMixer4              mixer10;        //xy=1683,701.9999923706055
AudioMixer4              mixer6;         //xy=1685,561.9999923706055
AudioMixer4              mixer5;         //xy=1685,627.9999923706055
AudioMixer4              mixer11;        //xy=1714.000015258789,312.99999618530273
AudioMixer4              mixer2;         //xy=1728,478.99999237060547
AudioMixer4              mixer8;         //xy=1847,658.9999923706055
AudioFilterStateVariable filter1;        //xy=1921.000015258789,486.99999618530273
AudioAmplifier           amp1;           //xy=1938.000015258789,309.99999618530273
AudioEffectDelay         delay5;         //xy=2018.000015258789,878
AudioEffectDelay         delay4;         //xy=2025.000015258789,742
AudioMixer4              mixer12;        //xy=2115.000015258789,527.9999961853027
AudioMixer4              mixer16;        //xy=2161.000015258789,841
AudioMixer4              mixer17;        //xy=2161.000015258789,907
AudioMixer4              mixer13;        //xy=2168.000015258789,705
AudioMixer4              mixer15;        //xy=2168.000015258789,771
AudioMixer4              slutmix;        //xy=2176.000015258789,263.99999618530273
AudioMixer4              mixer14;        //xy=2310.000015258789,733
```

```
AudioFilterBiquad       biquad1;        //xy=2315.000015258789,263.99999618530273
AudioOutputI2S          i2s2;           //xy=2457.000015258789,263.99999618530273
AudioEffectDelay        delay6;         //xy=2491,745.9999923706055
AudioEffectDelay        delay7;         //xy=2498,609.9999923706055
AudioMixer4             mixer18;        //xy=2584,495.99999237060547
AudioMixer4             mixer19;        //xy=2634,708.9999923706055
AudioMixer4             mixer20;        //xy=2634,774.9999923706055
AudioMixer4             mixer21;        //xy=2641,572.9999923706055
AudioMixer4             mixer22;        //xy=2641,638.9999923706055
AudioMixer4             mixer23;        //xy=2783,600.9999923706055
AudioConnection         patchCord1(i2s1, 0, slutmix, 0);
AudioConnection         patchCord2(i2s1, 0, mixer1, 0);
AudioConnection         patchCord3(delay1, 0, mixer3, 0);
AudioConnection         patchCord4(delay1, 1, mixer3, 1);
AudioConnection         patchCord5(delay1, 2, mixer3, 2);
AudioConnection         patchCord6(delay1, 3, mixer3, 3);
AudioConnection         patchCord7(delay1, 4, mixer4, 0);
AudioConnection         patchCord8(delay1, 5, mixer4, 1);
AudioConnection         patchCord9(delay1, 6, mixer4, 2);
AudioConnection         patchCord10(delay1, 7, mixer4, 3);
AudioConnection         patchCord11(mixer4, 0, mixer7, 1);
AudioConnection         patchCord12(mixer3, 0, mixer7, 0);
AudioConnection         patchCord13(mixer1, delay1);
AudioConnection         patchCord14(mixer1, 0, mixer11, 0);
AudioConnection         patchCord15(mixer1, 0, filter2, 0);
AudioConnection         patchCord16(mixer7, 0, mixer1, 1);
AudioConnection         patchCord17(filter2, 0, mixer2, 0);
AudioConnection         patchCord18(delay3, 0, mixer10, 0);
AudioConnection         patchCord19(delay3, 1, mixer10, 1);
AudioConnection         patchCord20(delay3, 2, mixer10, 2);
AudioConnection         patchCord21(delay3, 3, mixer10, 3);
AudioConnection         patchCord22(delay3, 4, mixer9, 0);
AudioConnection         patchCord23(delay3, 5, mixer9, 1);
AudioConnection         patchCord24(delay3, 6, mixer9, 2);
AudioConnection         patchCord25(delay3, 7, mixer9, 3);
AudioConnection         patchCord26(delay2, 0, mixer6, 0);
AudioConnection         patchCord27(delay2, 1, mixer6, 1);
AudioConnection         patchCord28(delay2, 2, mixer6, 2);
AudioConnection         patchCord29(delay2, 3, mixer6, 3);
AudioConnection         patchCord30(delay2, 4, mixer5, 0);
AudioConnection         patchCord31(delay2, 5, mixer5, 1);
AudioConnection         patchCord32(delay2, 6, mixer5, 2);
AudioConnection         patchCord33(delay2, 7, mixer5, 3);
AudioConnection         patchCord34(mixer9, 0, mixer8, 3);
AudioConnection         patchCord35(mixer10, 0, mixer8, 2);
AudioConnection         patchCord36(mixer6, 0, mixer8, 0);
```

```
AudioConnection          patchCord37(mixer5, 0, mixer8, 1);
AudioConnection          patchCord38(mixer11, amp1);
AudioConnection          patchCord39(mixer2, delay2);
AudioConnection          patchCord40(mixer2, delay3);
AudioConnection          patchCord41(mixer2, 0, mixer11, 1);
AudioConnection          patchCord42(mixer2, 0, filter1, 0);
AudioConnection          patchCord43(mixer8, 0, mixer2, 1);
AudioConnection          patchCord44(filter1, 0, mixer12, 0);
AudioConnection          patchCord45(amp1, 0, slutmix, 1);
AudioConnection          patchCord46(delay5, 0, mixer16, 0);
AudioConnection          patchCord47(delay5, 1, mixer16, 1);
AudioConnection          patchCord48(delay5, 2, mixer16, 2);
AudioConnection          patchCord49(delay5, 3, mixer16, 3);
AudioConnection          patchCord50(delay5, 4, mixer17, 0);
AudioConnection          patchCord51(delay5, 5, mixer17, 1);
AudioConnection          patchCord52(delay5, 6, mixer17, 2);
AudioConnection          patchCord53(delay5, 7, mixer17, 3);
AudioConnection          patchCord54(delay4, 0, mixer13, 0);
AudioConnection          patchCord55(delay4, 1, mixer13, 1);
AudioConnection          patchCord56(delay4, 2, mixer13, 2);
AudioConnection          patchCord57(delay4, 3, mixer13, 3);
AudioConnection          patchCord58(delay4, 4, mixer15, 0);
AudioConnection          patchCord59(delay4, 5, mixer15, 1);
AudioConnection          patchCord60(delay4, 6, mixer15, 2);
AudioConnection          patchCord61(delay4, 7, mixer15, 3);
AudioConnection          patchCord62(mixer12, delay4);
AudioConnection          patchCord63(mixer12, 0, mixer11, 2);
AudioConnection          patchCord64(mixer12, delay5);
AudioConnection          patchCord65(mixer12, 0, mixer18, 0);
AudioConnection          patchCord66(mixer16, 0, mixer14, 2);
AudioConnection          patchCord67(mixer17, 0, mixer14, 3);
AudioConnection          patchCord68(mixer13, 0, mixer14, 0);
AudioConnection          patchCord69(mixer15, 0, mixer14, 1);
AudioConnection          patchCord70(slutmix, biquad1);
AudioConnection          patchCord71(mixer14, 0, mixer12, 1);
AudioConnection          patchCord72(biquad1, 0, i2s2, 0);
AudioConnection          patchCord73(biquad1, 0, i2s2, 1);
AudioConnection          patchCord74(delay6, 0, mixer19, 0);
AudioConnection          patchCord75(delay6, 1, mixer19, 1);
AudioConnection          patchCord76(delay6, 2, mixer19, 2);
AudioConnection          patchCord77(delay6, 3, mixer19, 3);
AudioConnection          patchCord78(delay6, 4, mixer20, 0);
AudioConnection          patchCord79(delay6, 5, mixer20, 1);
AudioConnection          patchCord80(delay6, 6, mixer20, 2);
AudioConnection          patchCord81(delay6, 7, mixer20, 3);
AudioConnection          patchCord82(delay7, 0, mixer21, 0);
```

```
AudioConnection          patchCord83(delay7, 1, mixer21, 1);
AudioConnection          patchCord84(delay7, 2, mixer21, 2);
AudioConnection          patchCord85(delay7, 3, mixer21, 3);
AudioConnection          patchCord86(delay7, 4, mixer22, 0);
AudioConnection          patchCord87(delay7, 5, mixer22, 1);
AudioConnection          patchCord88(delay7, 6, mixer22, 2);
AudioConnection          patchCord89(delay7, 7, mixer22, 3);
AudioConnection          patchCord90(mixer18, delay7);
AudioConnection          patchCord91(mixer18, delay6);
AudioConnection          patchCord92(mixer18, 0, mixer11, 3);
AudioConnection          patchCord93(mixer19, 0, mixer23, 2);
AudioConnection          patchCord94(mixer20, 0, mixer23, 3);
AudioConnection          patchCord95(mixer21, 0, mixer23, 0);
AudioConnection          patchCord96(mixer22, 0, mixer23, 1);
AudioConnection          patchCord97(mixer23, 0, mixer18, 1);
AudioControlSGTL5000     sgtl5000_1;     //xy=1295.0000038146973,297.9999942779541
// GUItool: end automatically generated code


void setup() {


  Serial.begin(9600);

  AudioMemory(1300);

  sgtl5000_1.enable();
  sgtl5000_1.volume(0.7);

  AudioInterrupts();

}

#define pot1            A0
#define pot2            A1
#define pot3            A2

float delaymix = 0.25;

void loop() {


float knob1 = (float)analogRead(pot1) / 1023.0;  // Predelay
float knob2 = (float)analogRead(pot2) / 1023.0;  // crossfade
float knob3 = (float)analogRead(pot3) / 1023.0;  // feedback
```

```
filter1.frequency(knob1*10000+5000);
filter1.resonance(1);

filter2.frequency(knob1*4000+1000);
filter2.resonance(1);



//-----------Dry/Wet-Kontroll--------
slutmix.gain(0,1-knob2);
slutmix.gain(1,knob2);



//-----------TAP MIXERS-----------

//Sätter alla summeringsmixrar till samma amplitud

for(int i = 0; i < 4; i++){
  mixer3.gain(i,delaymix);
  mixer4.gain(i,delaymix);
  mixer5.gain(i,delaymix);
  mixer6.gain(i,delaymix);
  mixer9.gain(i,delaymix);
  mixer10.gain(i,delaymix);
  mixer13.gain(i,delaymix);
  mixer15.gain(i,delaymix);
  mixer16.gain(i,delaymix);
  mixer17.gain(i,delaymix);
  mixer22.gain(i,delaymix);
  mixer21.gain(i,delaymix);
  mixer20.gain(i,delaymix);
  mixer19.gain(i,delaymix);
}


mixer7.gain(0,delaymix);
mixer7.gain(1,delaymix);


//--------------Steg 1-------------
mixer1.gain(0, 1); // dry
mixer1.gain(1, 0.15 + (knob3 * 0.8)); // feedback
```

```
mixer7.gain(0,0.25);
mixer7.gain(1,0.25);


//--------------Steg 2--------------
mixer2.gain(0, 1); // dry

mixer2.gain(1, 0.15 + (knob3 * 0.7)); // feedback

mixer8.gain(0,0.25);
mixer8.gain(1,0.25);
mixer8.gain(2,0.25);
mixer8.gain(3,0.25);


//--------------Steg 3--------------
mixer12.gain(0,1);
mixer12.gain(1,0.2 + (knob3 * 0.7)); //Feedback

mixer14.gain(0,0.25);
mixer14.gain(1,0.25);
mixer14.gain(2,0.25);
mixer14.gain(3,0.25);


//--------------Steg 4--------------
mixer18.gain(0,1);
mixer18.gain(1,0.2 + (knob3 * 0.78)); //Feedback

mixer23.gain(0,0.25);
mixer23.gain(1,0.25);
mixer23.gain(2,0.25);
mixer23.gain(3,0.25);


//-------------Klustermixer-----------
mixer11.gain(0,0.25);
mixer11.gain(1,0.25);
mixer11.gain(2,0.25);
mixer11.gain(3,0.25);

for(int i = 1; i < 9; i++){
    delay2.delay(i-1,(float)random(0.10,3*i));
    delay3.delay(i-1,(float)random(0.30,2*i));
    delay4.delay(i-1,(float)random(0.50,1.014*i));
    delay5.delay(i-1,(float)random(0.12,10*i));
```

```
    delay6.delay(i-1,(float)random(20,50*i));
    delay7.delay(i-1,knob1*400 *i/5);
    delay8.delay(i-1,knob1*100*i/3.4);
    delay(10);
}

}
```

INSTITUTIONEN FÖR RYMD-, GEO- OCH MILJÖVETENSKAP
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

CHALMERS