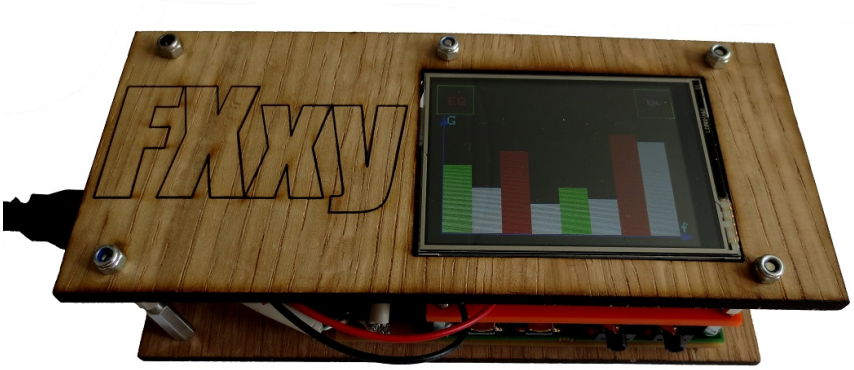
CHALMERS



Digitala musikeffekter med DSP

Design av multieffektenhet med grafiskt gränssnitt och pekskärmsstyrning

ERIK PAYERL
ERIK SANDGREN
GUSTAF JOHANSSON
OMID FASSIHI
RICKARD DAHL

Institutionen för Signaler och system
Chalmers Tekniska Högskola
Kandidatarbete inom civilingenjörsprogrammet
Göteborg, Sverige 2014

Sammanfattning

I rapporten beskrivs framtagningen av en musikeffektsenhet för användning i realtid. Designprocessen har bestått av två huvudsakliga delar: utvecklingen av musikeffekter för en digital signalprocessor samt ett tillhörande användargränssnitt.

För samtliga implementerade musikeffekter redovisas relevant teori, implementationsmetod och utvärdering. Effekterna har till en början utvecklats i MATLAB för att verifiera algoritmena. Därefter har de implementerats i C-kod för realtidsprocessering på utvecklingskortet *SD eZdsp5535* baserat på signalprocessorn *TI TMS320C5535*. De effekter som beskrivs i rapporten är: filter, equalizer, vibrato, tremolo, eko, reverb och distortion.

Effekterna styrs av användaren via ett pekskärmsgränssnitt. Gränssnittet består av pekskärmen *Adafruit 2,8" TFT Touch Shield v2* som drivs av mikrokontrollern *Arduino Uno R3*. Gränssnittet har utvecklats med avseende på användarvänlighet för att kunna användas i en live-situation. Flera kommunikationsprotokoll används för att överföra parametrar från skärmen till signalprocessorn.

Abstract

This report describes the development of a music effect unit for the purpose of realtime usage. The process of designing the product consisted of two main parts: the implementation of music effects on a digital signal processor and a graphical user interface.

All of the implemented music effects are presented with regards to relevant theory, method of implementation and evaluation. The effects were at first developed in MATLAB in order to verify the algorithms. They were subsequently implemented in C-code for the purpose of realtime processing on the development platform *SD eZdsp5535* based on the digital signal processor *TI TMS320C5535*. The following effects are described in the report: filter, equalizer, vibrato, echo, reverb and distortion.

The effects are controlled by the user through a touch screen interface. The interface consists of the touch screen *Adafruit 2,8" TFT Touch Shield v2* driven by the microprocessor *Arduino Uno R3*. The interface was developed with regards to usability in a live situation. Several communication protocols are required for the transmission of parameters from the touch screen to the digital signal processor.

Förord

Vi vill tacka styrelsen och alla medlemmar i Chalmers Robotförening (CRF) för användning av deras lokaler, maskiner, material och goda råd vilket möjliggjorde tillverkning av produktskalet. Vi vill även tacka professor Tomas McKelvey för hans entusiasm och goda handledning i projektets alla stadier. Slutligen vill vi tacka institutionen Signaler och System för deras finansiering av material.

Författarna, Göteborg 2014-05-18

Innehåll

1 Inledning **1**
1.1 Bakgrund . . . 1
1.2 Syfte . . . 1
1.3 Problemformulering . . . 2
1.4 Rapportstruktur . . . 2

2 Teori **3**
2.1 Hörselfysiologi . . . 3
2.2 Musikteori . . . 3
2.3 Rumsakustik . . . 4
2.4 Sampling . . . 5
2.5 Digitala filter . . . 6
2.5.1 Grundläggande frekvenssvar . . . 7
2.5.2 Filterdesign . . . 8
2.6 Digital signalprocessor . . . 9
2.7 Binära talformat . . . 10
2.7.1 Rationella fixtal - Q-format . . . 11
2.7.2 Fixtalsaritmetik . . . 11
2.8 Buffertar . . . 12
2.8.1 Cirkulär buffert . . . 12
2.9 Digital Fördröjning . . . 13
2.9.1 Interpolering . . . 13
2.9.2 Digitala kamfilter . . . 15

3 Musikeffekter **17**
3.1 Designmetodik . . . 17
3.2 Chamberlin state-variable filter . . . 18
3.3 Grafisk equalizer . . . 19
3.4 Vibrato . . . 23
3.4.1 Flanger . . . 25
3.5 Tremolo . . . 25
3.6 Eko . . . 26
3.7 Reverb . . . 27
3.8 Distortion . . . 29

4 Systembeskrivning **32**
4.1 Utvecklingskort - eZdsp5535 . . . 32
4.2 Ljudkodek - TLV320AIC3204 . . . 33
4.3 Arduino Uno . . . 34
4.4 Användargränssnitt . . . 34
4.4.1 Komponentbeskrivning . . . 35
4.5 Produktskal . . . 36
4.5.1 Mekanisk uppbyggnad . . . 36
4.5.2 In- och utgångar . . . 36
4.5.3 Spänningsdelare . . . 37

4.6 Programstruktur 38
4.6.1 Signalprocessorns programvara 38
4.6.2 Parameteruppdatering 41
4.6.3 Gränssnittets programvara 42
4.7 Gränssnittsdesign 43
4.7.1 State-variable filter 44
4.7.2 Equalizer 44
4.7.3 Övriga effekter 45

5 Diskussion **46**

6 Slutsats **50**

A Kravspecifikation **A1**

B Dataprotokoll **B1**
B.1 Avbrott B1
B.2 Inter-Integrated Circuit (I^2C) B1
B.3 Integrated Interchip Sound (I^2S) B2
B.4 Universal Asynchronous Receiver/Transmitter (UART) B3
B.5 Serial Peripheral Interface (SPI) B4
B.6 Direct Memory Access (DMA) B5

C Programkod och MATLAB-kod **C1**

D CAD-ritningar **D1**

Nomenklatur

ACK (acknowledgement)	Bit som bekräftar att kommunikation mellan två olika enheter har skett felfritt. Kallas också för bekräftelsebit.
AD (Analog Digital)	Förkortning som står för analog till digital omvandling.
ALU (Arithmetic Logic Unit)	Enhet som utför matematiska operationer såsom addition, subtraktion, absolutbelopp samt logiska operationer såsom AND och OR.
Arduino	Används som förkortning för microprocessorenheten Arduino Uno R3 i rapporten.
Asynkron	Asynkron kommunikation innebär att sändande och mottagande enheter inte delar på en gemensam klocka. Dessutom skickar sändaren bara data när den har något att meddela.
Baudhastighet	Måttenhet för hastigheten symboler skickas i kommunikation mellan två olika enheter.
Buffert	Ett minnessegment som används för att tillfälligt spara gamla sampel som kan användas vid beräkning.
CSL (Chip Support Library)	Ett mjukvarubibliotek från Texas Instruments med funktionsanrop för olika periferienheter på deras utvecklingskort.
DA (Digital Analog)	Förkortning som står för digital till analog omvandling.
DMA (Direct Memory Access)	Metod för periferienheter att direkt kommunicera med minnesenheter istället för att använda sig utav CPU:n.
DSP (Digital signalprocessor)	Processor optimerad för matematiska beräkningar med syftet att utföra effektiv signalbehandling.

DSPLIB	Mjukvarubibliotek från Texas Instrument speciellt optimerat för matematiska beräkningar i digitala signalprocessorer av typen C55x.
FFT (Fast Fourier Transform)	Vanlig algoritm som åstadkommer snabb fouriertransformering.
FIR (Finite Impulse Response)	Filtertyp med ändligt impulssvar.
Fixtal	Metod för att i datorer representera heltal.
Flyttal	Metod för att i datorer approximativt representera reella tal.
Godhetstal/Q-värde	Parameter som för musikaliska filter brukar avse resonansen.
IIR (Infinite Impulse Response)	Filtertyp med oändligt impulssvar.
Interpolering	Metod för att konstruera en kontinuerlig signal ur en diskret signal.
LFO (Low Frequency Oscillator)	Enhet som genererar en periodisk vågform med låg frekvens. Vågformen är vanligtvis sinus-, triangel- eller fyrkantsvåg.
MAC (Multiply-accumulate Unit)	Enhet som används för multiplikation.
Makro	Makron används för att förenkla eller förkorta en sekvens kodinstruktioner som används återkommande i en programvara.
Master	Master är enheten vars klocka alla andra kommunicerande enheter synkroniseras till vid synkron kommunikation. I många fall är mastern den huvudsakliga sändande enheten.
NACK (negative acknowledgement)	Bit som visar att det skett ett fel när data skickats mellan två enheter. Kallas också för negativ-bekräftelsebit.

Pingpongbuffert	Metod för bufferthantering där arbetet med att lagra och hämta data delegeras till två olika buffertar.
Q-format	Ett sätt att hantera bråktal i fixtalsprocessorer och används för att spara beräkningskraft.
RX	Betecknar mottagare (receiver) i kommunikationssammanhang.
Slav	Slaven är en kommunicerande enhet som synkroniseras till masterns klocka vid synkron kommunikation. I många fall är slaven huvudsaklighen en mottagare.
Synkron	Synkron kommunikation innebär att sändande och mottagande enheter kommunicerar utifrån en gemensam klocka.
TX	Betecknar sändare (transmitter) i kommunikationssammanhang.
Vågtabell	Look-up tabell som innehåller värdena för en vågform.

1 Inledning

Ända sedan människan upptäckte musik har viljan att skapa nya ljud funnits. Under lång tid uppfanns nya ljud genom skapandet av musikinstrument. Det senaste århundradet har elektroniska instrument som avger elektriska signaler uppkommit. I och med detta uppstår nya möjligheter för att skapa intressanta ljud genom att modifiera dessa elektriska signaler. Att förändra innehållet i en signal på ett önskvärt sätt brukar kallas för signalbehandling.

1.1 Bakgrund

Signalbehandling är ett brett vetenskapligt område med många användningsområden. Signalbehandlingen tar som sin utgångspunkt den matematiska teorin kring linjära system för att förändra och/eller extrahera information ur en signal på ett kontrollerat sätt [1]. Dess användningsområde är bland annat fingeravtrycksigenkänning, satellitkommunikation, autopiloter och talsyntetisering. Området är uppdelat i analog signalbehandling och digital signalbehandling.

I analog signalbehandling används kretskomponenter såsom resistorer, kondensatorer och spolar för att åstadkomma operationer som till exempel filtrering och förstärkning [1]. Innan ankomsten och utbredningen av digital teknik var analog signalbehandling det enda alternativet. Digital signalbehandling använder sig emellertid av sampling och representation av signaler i form av diskreta talserier för att med datorkraft utföra matematiska operationer på dessa diskreta signaler.

Inom musiken används signalbehandling i form av ljudsyntetisering och ljudbehandling för att skapa så kallade musikeffekter [1]. En musikeffekt används för att ge ett ljud nya önskade egenskaper. De digitala musikeffekter som finns på marknaden idag är oftast en digital modell av ett analogt dito som till exempel distortion [2].

Digital signalbehandling för att åstadkomma musikeffekter är till fördel kontra analog signalbehandling eftersom den digitala enheten kan programmeras om när nya funktioner önskas [1]. Önskas ny funktionalitet i analoga signalbehandlingsenheter innebär det att komponenter måste bytas, vilket oftast är mer mödosamt.

I praktiken är det vanligt att digital signalbehandling implementeras på en digital signalprocessor (DSP) [1]. Dessa processorer är speciellt anpassade för att snabbt kunna utföra matematiska operationer som ofta används vid signalbehandling. Även digital signalbehandling av ljud kan utföras på en DSP [2]. Det finns flera fördelar med en DSP gentemot en vanlig dator: processeringstiden är snabb och förutsägbar, processorkraft används framförallt till signalbehandling, enheten är normalt fysiskt liten och förbrukar mindre elektrisk energi [1].

1.2 Syfte

Arbetets syfte har varit att utveckla en musikeffektsenhet avsedd för uppträdande musiker. Denna enhet implementerar musikeffekter med hjälp av en di-

gital signalprocessor (DSP) som sköter ljudbehandlingen. Enheten styrs via en pekskärm och har därtill ett grafiskt användargränssnitt. Vid utvecklingen av enheten har fokus legat på att den ska vara lättstyrd och utföra ljudbehandling i realtid för att kunna användas i livesammanhang.

1.3 Problemformulering

Produktens och projektets huvuduppgift är att erbjuda lättstyrda musikeffekter i realtid för musiker i livesammanhang. För att realisera detta är det nödvändigt att först identifiera de delproblem som existerar.

I grunden för produkten finns musikeffekterna. Dessa implementeras på en DSP och styrs efter behov av ett antal parametrar. Hur dessa musikeffektsalgoritmer är uppbyggda och hur de implementeras måste därför undersökas.

Utformningen av gränssnittet behöver också redas ut. Problem som finns är att ta reda hur DSP:n kan kommunicera med externa enheter och hur detta implementeras. Gränssnittet ska vara utformat så att det för användaren är enkelt och intuitivt att använda produkten.

Den ingående hårdvaran ska på något sätt skyddas fysiskt, vilket skapar ett behov av ett skyddande skal. Hur skalet utformas med hänsyn till funktion och estetik måste därför utredas.

Utgående från de delproblem som formulerats har en kravspecifikation satts upp för att konkretisera produktens eftersträvade egenskaper. Efter att produkten är realiserad ska den utvärderas gentemot denna kravspecifikation som redovisas i tabell 4 i appendix A Kravspecifikation. Utöver kravspecifikationen har även önskemål för vidare egenskaper satts upp. Önskemålen finns redovisade i tabell 5 i appendix A Kravspecifikation och är viktade för att ge en indikation av prioritet bland dessa.

1.4 Rapportstruktur

Rapporten är indelad i tre huvudavsnitt: teori, musikeffekter och systembeskrivning. I kapitel 2 beskrivs generell teori som syftar till att ge läsaren en grundläggande förståelse för ämnesområdena som projektet berör. Kapitel 3 förklarar hur musikeffekter används, teorin bakom, hur dessa implementerats i projektet samt resultat. Kapitel 4 syftar till att ge läsaren en förståelse för produktens hårdvaru- och mjukvaruimplementation. I slutet av rapporten ges en diskussion och slutsats kring arbetet.

2 Teori

Teorikapitlet syftar till att introducera den grundläggande teori som behövs för att genomföra projektet.

Kapitlet består av flera olika delavsnitt med olika inriktning. De tre första delavsnitten (Hörselfysiologi, Musikteori och Rumsakustik) beskriver hur ljud och musik uppfattas. Dessa avsnitt är nödvändiga för att förstå vad som sker när ljud manipuleras.

Resterande delavsnitt i detta kapitel är mer tekniskt inriktade på hur en elektrisk ljudsignal kan digitaliseras för att sedan förändras och hanteras med digital signalbehandling.

Rapporten innehåller mer teori än det som avhandlas i detta kapitel. Specifik teori för olika musikeffekter hittas i kapitel 3 Musikeffekter och specifik teori för den hårdvara och mjukvara som används finns i kapitel 4 Systembeskrivning.

2.1 Hörselfysiologi

En viktig aspekt för att förstå hur människans hörselsinne och uppfattning av ljud fungerar är frekvensuppfattning. Unga människor med ett friskt öra kan höra ljud inom ett frekvensområde på ungefär 20 Hz till 20 kHz [3]. Den maximalt hörbara frekvensen avtar dock med åldern och ligger runt 18 kHz för äldre människor. Inom frekvensområdet uppfattas högre frekvenser som ljusare och lägre frekvenser som mörkare. På grund av det begränsade frekvensområdet kommer dessutom färre övertoner att höras vid högre frekvenser. Detta medför att högfrekventa ljud uppfattas som tunna medan lågfrekventa ljud upplevs som mer fylliga och ibland grumliga.

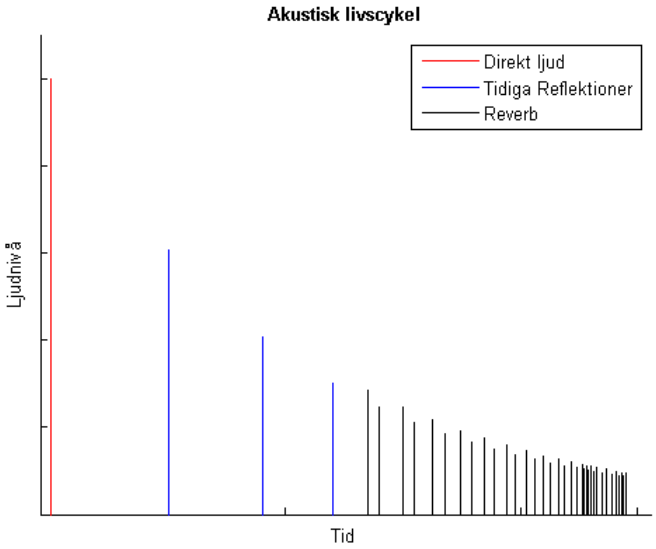
Örat uppfattar det inkommande ljudet logaritmiskt snarare än linjärt [3]. Detta innebär till exempel att om frekvensen ändras från 100 Hz till 120 Hz kommer människan uppfatta det som en mycket större ändring än om frekvensen ändras från 1000 Hz till 1020 Hz, trots att frekvensdifferensen är samma i båda fallen. Om en lika stor ändring i ljudets relativa tonläge (samma intervall) ska uppfattas måste förhållandet mellan två frekvenser för olika frekvensområden vara lika [3], enligt

$$\frac{f_{a1}}{f_{a2}} = \frac{f_{b1}}{f_{b2}} \tag{1}$$

där f_{a2} och f_{b2} är referensfrekvenser.

2.2 Musikteori

En egenskap som ligger till grunden för många musiksystem är att toner som är separerade med ett antal oktaver (ett antal dubbleringar eller halveringar av en frekvens) uppfattas som nästan lika [4]. Exempelvis uppfattas ljud med referensfrekvens på 440 Hz ungefär lika som ljud med en dubbelt så hög frekvens på 880 Hz. I västerländsk musik används det kromatiska systemet där varje



Figur 1: Ljud som emitteras i ett rum reflekteras och avtar i styrka.

oktav delas upp i tolv lika stora logaritmiska steg (semitoner) vars frekvenser ges av

$$f_n = \sqrt[12]{2^n} f_0 \tag{2}$$

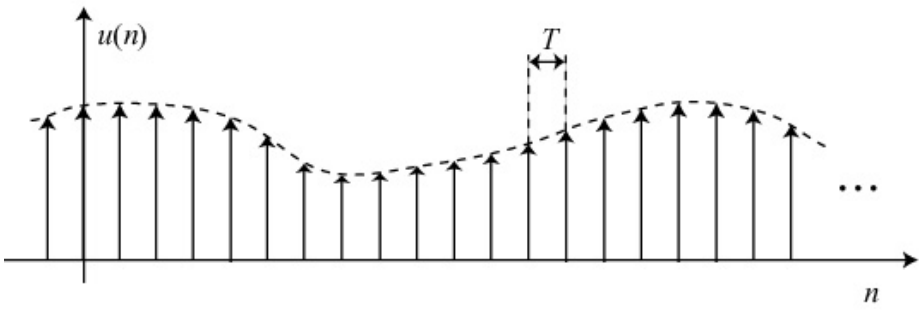
där n är antalet semitoner från referensfrekvensen f_0.

2.3 Rumsakustik

Ljud som emitteras i en instängd miljö såsom ett rum består av två delar: direkt ljud och indirekt ljud [5]. Det direkta ljudet når mottagaren utan att stöta på hinder som väggar och förmedlar därmed egenskaper som ljudets styrka och riktningen till ljudkällan. Hur lång tid det tar beror på avståndet mellan ljudkällan och mottagaren, och är ofta $(20 - 200)$ ms i rumssammanhang.

Indirekt ljud når däremot mottagaren efter att ha stött på hinder och kan delas upp i två delar: tidiga reflektioner och reverb [5]. De tidiga reflektionerna når mottagaren inom 30 ms och bidrar till ljudets fyllighet och styrka. Dessutom ger de hörselsinnet en uppfattning av rummets storlek. Reverb resulteras däremot från att de tidiga reflektionerna reflekteras vidare och avtar i styrka. Ju längre tid det går från att initialljudet emitterats desto tätare blir reflektionerna. Reverb fyller ut ljudbilden, bidrar till ljudstyrkan och skapar en akustisk rymlighet och utsträckning. En stor del av ljudets totala energi blir till reverb. Tiden det tar för det emitterade ljudet att avta med 60 dB definieras som *reverbation time* och är ett mått på ett rums akustiska egenskaper. I figur 1 visas en akustisk livscykel för ljud emitterat i ett rum.

Eko är till skillnad från reverb en distinkt reflektion av det initiala ljudet [5]. En fördröjning på minst 35 ms brukar uppfattas som ett eko.



Figur 2: Den streckade linjen representerar den kontinuerliga (analoga) signalen, medan pilarna representerar den diskreta (digitala) signalens värden vid diskreta tidpunkter. Tiden mellan två sampel av den analoga signalen är T, samplingstiden.

2.4 Sampling

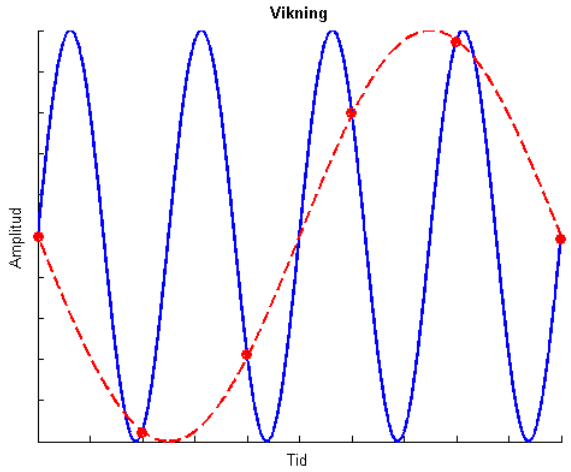
I många fall är det önskvärt att representera en analog signal i digital form. Överförandet från analog signal till digital signal görs genom vad som kallas för sampling. Den huvudsakliga anledningen till att digitalisera en signal är att det ofta är mer flexibelt att behandla diskreta signaler än analoga. Detta beror till stor del på den långt utvecklade digitalteknik som finns idag [6]. Samplingen utförs genom att vid jämnt spridda tidspunkter mäta och notera värdet av en signal, figur 2 visar hur detta kan gå till. I praktiken är det inte bara i tid en signal diskretiseras vid sampling. Även den samplade signalens amplitud kommer enbart att kunna anta diskreta värden [8]. Detta beror på att varje datapunkt representeras av ett begränsat antal bitar. Om exempelvis varje sampel representeras av 16 bitar finns det $2^{16} - 1 = 65535$ möjliga amplitudnivåer för signalen.

En bra sampling av en analog signal har egenskapen att den analoga signalen kan återskapas felfritt ur dess sampel [8]. Ett mycket viktigt teorem inom digital signalbehandling, *samplingsteoremet*, avgör huruvida detta är möjligt för en godtycklig signal. Teoremet säger att om en signal $x(t)$ är bandbegränsad, det till säga om $\mathcal{F}(x(t)) = X(j\omega) = 0$ för något $|\omega| > \omega_m$ är $x(t)$ återskapningsbar ur dess sampel $x(nT)$ där samplingshastigheten $\omega_s > 2\omega_m$. Samplingshastigheten måste alltså vara dubbelt så stor som den största förekommande frekvenskomponenten i $x(t)$ för att samplingsteoremet ska vara uppfyllt [6].

Om en analog signal samplas för långsamt, det vill säga att samplingsfrekvensen är lägre än dubbelt så stor som den högsta frekvensen i signalen och samplingsteoremet därmed bryts, uppstår fenomenet vikning. Följden av vikning är att att den samplade signalen inte kan återskapas felfritt ur sina sampel [9]. Figur 3 visar ett exempel på konsekvensen av undersampling.

I praktiken används AD-omvandlare för att sampla en analog signal. Återskapandet av en digital signal till analog görs då av en DA-omvandlare .

Olika tillämpningar kräver olika samplingshastigheter. Vid val av samplings-



Figur 3: Den blå signalen är den signal som önskas samplas. Samplingspunkterna är de röda punkterna. Här är frekvensen av den blå signalen $f = 0{,}8f_s$. Vilket bryter mot samplingskriteriet. Då signalen ska återskapas ur sampel ges istället den rödsträckade signalen, som har lägre frekvens $0{,}2f_s$.

frekvens måste ett avvägande mellan kvalitet på den samplade signalen och datahastighet göras [8]. Högre samplingshastighet betyder också högre datahastighet. Om exempelvis varje sampel har en datastorlek av 16 bitar och samplingsfrekvensen är 10 kHz, ges en datahastighet av $16 * 10000 = 160$ kbit/s. För musik brukar vanligtvis en samplingsfrekvens av 44,1 kHz användas, vilket ger tillräckligt hög kvalitet för de flesta öron [7].

2.5 Digitala filter

Digitala filter kan vara linjära eller olinjära och tidsvarianta eller tidsinvarianta. Eftersom många av de vanliga musikeffekterna byggs upp av linjära tidsinvarianta filter och teorin om linjära system utgör en av grundpelarna i signalbehandling, följer här en kort beskrivning av linjära tidsinvarianta filter och dess design.

Till linjära system hör till exempel eko, reverb och vågrörelser som ljud och elektromagnetiska vågor. Även matematiska operationer såsom faltning och rekursion är linjära system, därav är grundläggande digitala filter linjära då de bygger på dessa operationer. Det ska dock nämnas att vissa musikeffekter så som vibrato och distortion är olinjära, men förståelsen för linjära system är till hjälp även när olinjära system ska analyseras.

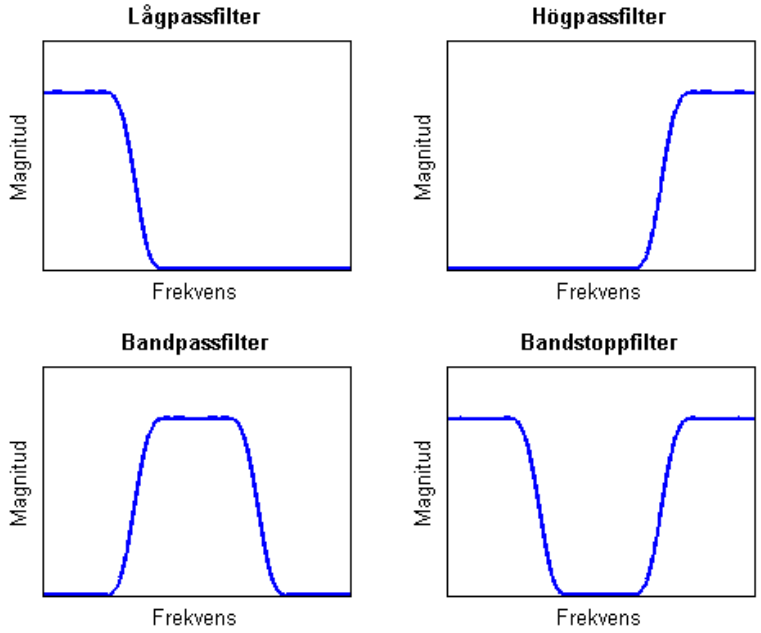
Varje linjärt tidsinvariant filter har ett impuls-, steg- och frekvenssvar som var för sig innehåller en fullständig beskrivning av filtret [11]. I audiosammanhang är frekvenssvaret, $H(e^{j\omega})$ av störst intresse. Eftersom frekvenssvaret är Fouriertransformen av impulssvaret är det i allmänhet en komplexvärd funktion och

representeras vanligtvis på polär form med en magnitud $A(\omega)$ och en fas $\phi(\omega)$.

$$H(e^{j\omega}) = A(\omega)e^{j\phi(\omega)} \tag{3}$$

2.5.1 Grundläggande frekvenssvar

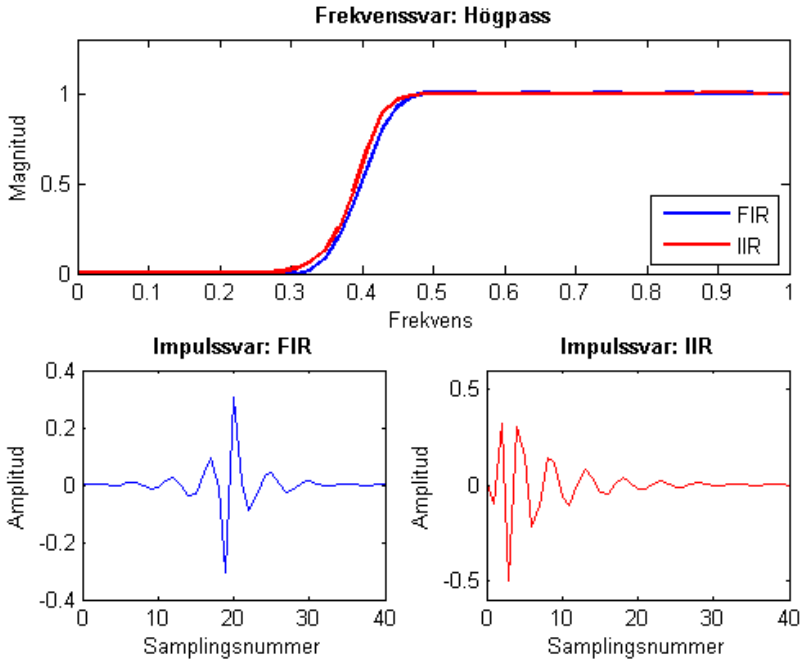
I figur 4 visas exempel på de fyra grundläggande frekvenssvaren, det vill säga lågpass- och högpassfilter samt bandpass- och bandstoppfilter. De används för att blockera vissa frekvenser och låta andra passera obehindrat. Parametrar att ta hänsyn till vid design av dessa filter är brantheten i övergången mellan passbandet och stoppbandet (roll-off), rippel i passbandet och attenuering (dämpning) i stoppbandet.



Figur 4: Exempel på grundläggande frekvenssvar.

I audiosammanhang brukar mindre hänsyn tas till frekvenssvarets fasfunktion eftersom det mänskliga örat är mycket okänsligt för variationer i fas. Detta kan enkelt förstås genom följande exempel. Antag att en åhörare lyssnar på en person som pratar i ett mindre rum. Mycket av ljudet som når åhöraren har först reflekterats från väggar, golv och tak. Hur ljud reflekteras beror på dess frekvens och olika frekvenser kommer att gå olika vägar innan det når åhörarens öra. Det innebär att den relativa fasen för varje frekvens kommer att ändras när åhöraren rör sig i rummet. Trots det uppfattar örat rösten som talar oförändrad.

Då fasen är intressant kan så kallade allpassfilter skapas. Denna typen av filter släpper igenom alla frekvenser med oförändrad magnitud, men ändrar fasen. Det går även att skapa filter med perfekt fassvar, det vill säga som lämnar fasen oförändrad.



Figur 5: Frekvenssvaret för ett högpass IIR- och FIR-filter kan vara mycket lika varandra men impulssvaret skiljer sig åt väsentligt.

Det finns två sätt att skapa linjära, tidsinvarianta digitala filter. De kan ha ett ändligt långt impulssvar, så kallade FIR-filter (Finite-duration Impulse Response) eller ha teoretiskt oändligt långt impulssvar, så kallade IIR-filter (Infinite-duration Impulse Response), se figur 5 för exempel. FIR-filter har fördelar såsom att de alltid är stabila och är mycket mer okänsliga för avrundningsfel än IIR-filter [10]. IIR-filter är däremot beräkningsmässigt mycket effektivare och går generellt en storleksordning snabbare att beräkna än ett FIR-filter med motsvarande prestanda.

2.5.2 Filterdesign

FIR-filter FIR-filter har per definition en ändlig längd N som också kallas filtrets ordningstal. Utsignalen $y(n)$ kan skrivas som en diskret faltning av insignalen $x(n)$ och filtrets N långa impulssvar $h(n)$: ($h(n) = 0$ då $n > N$.)

$$y(n) = \sum_{m=0}^{N-1} h(m)x(n-m) \tag{4}$$

FIR-filtrets överföringsfunktion $H(z)$ ges av z-transfomen av impulssvaret:

$$H(z) = \sum_{n=0}^{N-1} h(n) \cdot z^{-n} \tag{5}$$

Filtrets frekvenssvar fås genom att substituera $z = e^{\mathrm{j}\omega}$.

Det finns många olika metoder för att designa FIR-filter. Tre vanligt förekommande metoder är *Window design method*, *Frequency Sampling Method* och *Parks-McClellan method*.

Window design method: Metoden går ut på att utgå från ett idealt (oändligt) impulssvar och trunkera det till ändlig längd genom att multiplicera impulssvatet med en fönsterfunktion [12]. Utseendet på fönsterfunktionen påverkar rippel i pass- och stoppband samt övergångshastigheten mellan banden.

Frequency Sampling method: Ett filter med godtyckligt frekvenssvar kan konstrueras genom att det önskade frekvenssvaret samplas [13]. Med invers diskret Fouriertransform fås därefter impulssvaret och därmed filterkoefficienterna. Resultatet blir ett filter vars frekvenssvar exakt överensstämmer med det önskade frekvenssvaret i de samplade punkterna. Däremot så fås ingen explicit kontroll över filtret mellan sampelpunkterna.

Parks-McClellan method: En effektiv iterativ metod för att hitta ett optimalt filter som minimerar rippel i både pass- och stoppband [12].

IIR-filter IIR-filter bygger på återkoppling och kallas därför också för rekursiva filter. Utsignalen $y(n)$ ges av följande differensekvation:

$$y(n) = \sum_{m=0}^{M-1} b(m)x(n-m) - \sum_{k=1}^{N} a(k)y(n-k) \tag{6}$$

Den första summeringen är precis samma som för FIR-filter. Skillnaden består av den andra summanden som är en viktad summation av N tidigare utsignaler. Det är den återkopplade eller rekursiva delen av summan som gör att IIR-filter teoretiskt kan ha ett oändligt långt impulssvar och att IIR-filter kan bli instabila.

IIR-filter har följande överföringsfunktion:

$$H(z) = \frac{\sum_{n=0}^{M-1} b(n) \cdot z^{-n}}{1 + \sum_{k=1}^{N} a(k) \cdot z^{-k}} \tag{7}$$

IIR-filter kan exempelvis utformas genom att utgå från kända analoga designprinciper som Butterworth-, Chebyshev- och elliptiska filtertyper. Det analoga filtret kan sedan konverteras till ett digital filter genom t ex så kallad bilinjär transform [14].

2.6 Digital signalprocessor

En signal är en bärare av information från en sändare till en eller flera mottagare. Ordet signalprocessor antyder en enhet som mottar och behandlar information från en sändare. I praktiken utförs detta av en mikroprocessor, som med hjälp av ingångar och perfiferikretsar på ett kretskort samplar signaler och utför beräkningar på data från signalen.

Digitala signalprocessorer skiljer sig från vanliga processorer, som är optimerade för att sköta ett operativsystem och sysslor som ordbehandling och lagring och sortering av data i minne. Digitala signalprocessorer är istället optimerade för att utföra en viss sekvens av matematiska beräkningar och är istället sämre på att hämta och spara data i sitt minne [15].

Med hjälp av en Arithmetic Logic Unit (ALU) utför en signalprocessor precis som traditionella processorer matematiska operationer såsom addition, subtraktion, absolutbelopp, samt logiska operation som AND, OR med mera. En separat multiplikationsenhet används för att multiplicera ihop två värden ifrån två olika register och spara i ett annat register. Alternativt så används en MAC-enhet (Multiply-accumulate unit, MAC) för att utföra en multiplikation och addition samtidigt. En skiftare används bland annat för att utföra bitskift, genom att ta emot ett antal bitar och sedan flytta dem fram eller bak till andra värdespositioner. Att bitskifta ett tal n gånger motsvarar multiplikation med 2^n, där n är ett heltal som kan vara både positivt och negativt. En skiftare kan också extrahera ett önskat antal bitar ur ett bitfält.

Signalprocessorer använder ofta också AD- och DA-omvandlare för att omvandla signaler mottagna på ingångarna till data, samt omvandla processerad data till analoga signaler som kan skickas via utgångar. Dessa periferienheter utgör en arkitektur hos digitala signalprocessorer som gör att den kan både utföra flera olika matematiska operationer och parallellt sköta mottagandet, omvandlandet och presenterandet av signaler vilket drastiskt minskar beräkningstiden [15].

2.7 Binära talformat

Alla digitala tal sparas binärt med ettor och nollor. Vilket värde dessa ettor och nollor representerar är dock inte unikt utan beror på hur det binära talet tolkas. De två vanligaste binära talformaten följer nedan.

Flyttal består av två delar, värdesiffra s och exponent e. Tal sparas på formen $x = s \cdot b^e$ där b är basen. I ett 32-bitars flyttal representeras vanligen värdesiffrorna av 24 bitar och exponenten av 8 bitar [16]. Exponenten gör det möjligt att spara mycket stora och mycket små tal, det vill säga att ett stort dynamiskt omfång möjliggörs, på bekostnad av antalet värdesiffror som kan sparas.

Fixtal har bara en del där alla bitar representerar värdesiffror. Talet tolkas ofta som ett heltal. I tvåkomplementsform med N bitar blir värdet:

$$x = -2^{N-1}b_{N-1} + \sum_{i=0}^{N-2} 2^i b_i, \qquad b_i = \{0,1\} \tag{8}$$

Digitala signalprocessorer kan delas in i två kategorier: de som arbetar med flyttal och de som jobbar med fixtal. Flyttalsprocessorerna har bättre precision och större dynamiskt omfång än fixtalsprocessorer och dessutom ofta kortare utvecklingstid för programvara då det ofta är lättare att skriva algoritmer för flyttal än fixtal [15]. Fixtalsprocessorerna är dock ofta billigare att tillverka och lämpar sig därför till produkter med stora produktionsvolymer.

2.7.1 Rationella fixtal - Q-format

I digital signalbehandling behövs ofta användning av andra tal än heltal. Ett sätt att hantera decimaltal med hjälp av fixtal är att skriva dem som ett bråk, till exempel:

$$0{,}125 = \frac{1}{8} = \frac{2}{16} = \frac{4}{32} = \dots$$

Tal kan alltså sparas på formatet L/M men endast täljaren L sparas i minnet som ett binärt heltal. Värdet på nämnaren M får programmeraren själv komma ihåg varje gång värdet L ska användas. Av ett fixtal med N bitar används då n bitar till att spara decimaldelen av talet och m bitar till heltalsdelen. n ges av nämnaren $M = 2^n$ och $m = N - n - 1$ då en teckenbit används. Att spara tal på detta sätt kallas för Q-format och noteras vanligen Q$m.n$ eller Qn om $m = 0$. Fler heltalsbitar m ger möjlighet att representera större tal. Fler decimalbitar n ger däremot bättre upplösning [17].

Omfång: $x_{max} = 2^m - 2^{-n}, \qquad x_{min} = -2^m$

Upplösning: 2^{-n}

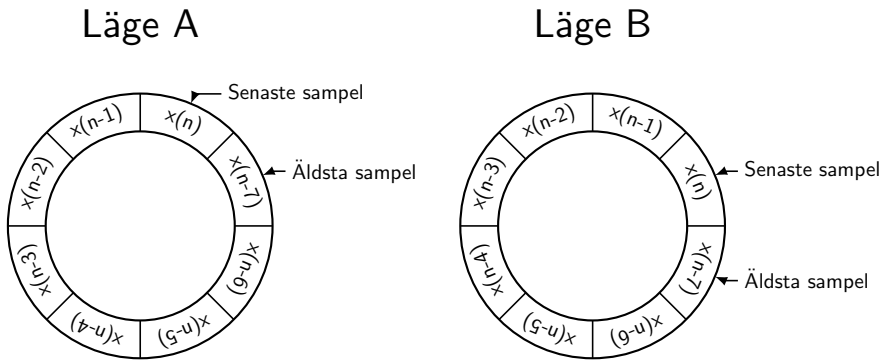
2.7.2 Fixtalsaritmetik

Addition och subtraktion För att addera och subtrahera två tal på Q-format måste först decimalkommats placering justeras så att den överensstämmer, det vill säga att båda talen måste vara på samma Q$m.n$-format.

Multiplikation Om a och b är tal på Q15-format, det vill säga $a = \frac{\hat{a}}{2^{-15}}$ och $b = \frac{\hat{b}}{2^{-15}}$ blir produkten $c = a \cdot b = \frac{\hat{a} \cdot \hat{b}}{2^{-30}}$, det vill säga ett tal på Q30-format. Generellt blir produkten av Q$a.b \times$ Q$c.d =$ Q$(a + c).(b + d)$.

Division På motsvarande sätt som multiplikation av två Q15 tal resulterar i ett Q30 tal är resultatet av division Q30/Q15 ett Q15 tal. Om både täljaren och nämnaren är av samma längd N, gäller följande för antal heltalsbitar respektive decimalbitar i kvoten. Antag att täljaren är på formatet Q$m_T.n_T$ och nämnaren Q$m_N.n_N$. Det resulterande kvoten är på formatet Q$m_K.n_K$. Då gäller $m_K = m_T + n_N$ och $n_K = n_T - n_N$. För att kvoten ska vara riktig krävs även att $n_T \geq n_N$ och att $m_T + n_N < N$. Eftersom division kräver fler beräkningscykler än multiplikation kan det vara en fördel att vid division med ett känt tal c, istället först räkna ut kvoten $1/c$ och sedan utföra multiplikation istället.

Look-up tabeller Många funktioner som exempelvis $\sin x$, $\log x$ och e^x är transcendenta funktioner och beräkningsmässigt krävande när de approximeras med talserier såsom maclaurinutvecklingen. Exempelvis $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots$. För att spara beräkningskraft kan det vara lämpligt att spara färdigberäknade funktionsvärden i tabeller, så kallade look-up tables.



Figur 6: Två figurer som tillsammans visar hur en ny sampel skrivs in i en cirkulär buffert och hur indexeringen uppdateras. Läge B visar hur bufferten ser ut efter att en sampel ny sampel skrivs in i Läge A.

2.8 Buffertar

Digitala filter implementeras genom att addera fördröjda och skalade in- och eller utsampel av en signal för att på så sätt skapa en ny signal [18]. För att i praktiken lyckas med detta är det viktigt att på något sätt lagra tidigare sampel, så att dessa kan användas för att bygga utsignalen. Detta åstadkoms vanligtvis genom en buffert i minne där gamla sampel lagras. Om ett FIR-filter implementeras räcker det att spara gamla sampel av insignalen, medan om ett IIR-filter implementeras måste både in- och utsignal sparas. Det vill säga en skild buffert för insignal och en för utsignal.

I signalbehandlingssammanhang skiljs det på realtidsprocessering och off-line processering. Vid off-line processering finns hela signalen som ska behandlas (insignalen) i minnet samtidigt, utsignalen kan då skapas när man vill i ett svep. Vid realtidsprocessering ska insignalen avläsas kontinuerligt samtidigt som utsignalen skapas [15]. Arbetsgången vid realtidsprocessering börjar med att en insampel läses av och skickas till signalbehandlingsalgoritmen som med den nya sampeln samt minnet av tidigare avlästa sampel skapar en utsignal.

2.8.1 Cirkulär buffert

I realtidsapplikationer av signalbehandling där nya sampel oavbrutet läses av och ett minne av ett antal tidigare sampel krävs för att skapa utsignalen är det inte längre lönsamt att använda sig av en vanlig buffert. I en vanlig buffert skulle vid varje ny insampel alla element i bufferten behöva skiftas. Detta är oekonomiskt då det använder onödigt många klockcykler [19]. Istället kan en cirkulär buffert implementeras, där det räcker att skriva in det nya samplet i bufferten, samt uppdatera en pekare för indexering.

I figur 6 illustreras hur en cirkulär buffert kan fungera när ett nytt sampel läses av och skrivs in i minnet. Det äldsta samplet i bufferten skrivs över av det nya och indexeringen uppdateras. Indexeringen kan lösas genom att ha en pekare på den senaste sampeln som skrivits in i bufferten, samt vetskapen om

hur många element som får plats i bufferten. Varje gång en ny sampel skrivs in inkrementeras pekaren som alltså rör sig runt cirkeln. Detta medför att inga andra sampel behöver flyttas runt i bufferten när en ny sampel skrivs in.

2.9 Digital Fördröjning

Att digitalt fördröja en signal är inte svårt, detta görs genom att helt enkelt spara signalen i en FIFO (First In First Out)-buffert med en längd som motsvarar den önskade fördröjningstiden [20]. Denna enkla implementation av fördröjning är en av de stora fördelarna för digital signalbehandling kontra analog signalbehandling. Fördröjningen är viktig för många olika digitala musikeffekter som phasing, vibrato, chorus, flanging och reverb, men även inom andra signalbehandlingsområden.

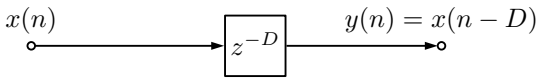
För att skapa en mer formell diskussion kring fördröjning, låt L_D utgöra en operator som fördröjer en signal med D sampel, fördröjning kan då matematiskt uttryckas som:

$$y(n) = L_D(x(n)) = x(n - D) \tag{9}$$

En fördröjning av en signal $x(n)$ kan alltså uttryckas som $x(n - D)$, där fördröjningen är DT s lång, om T är samplingstiden. En fördröjning inte endast skrivas i tidsplanet, i z-planet kan en överföringsfunktion för ovanstående operation betecknas som:

$$H_{L_D} = \frac{Y(z)}{X(z)} = \frac{z^{-D}X(z)}{X(z)} = z^{-D} \tag{10}$$

I blockscheman representeras fördröjningar av ett block med z^{-D}. Ett blockschema för fördröjningen D av signalen $x(n)$ visas i figur 7.

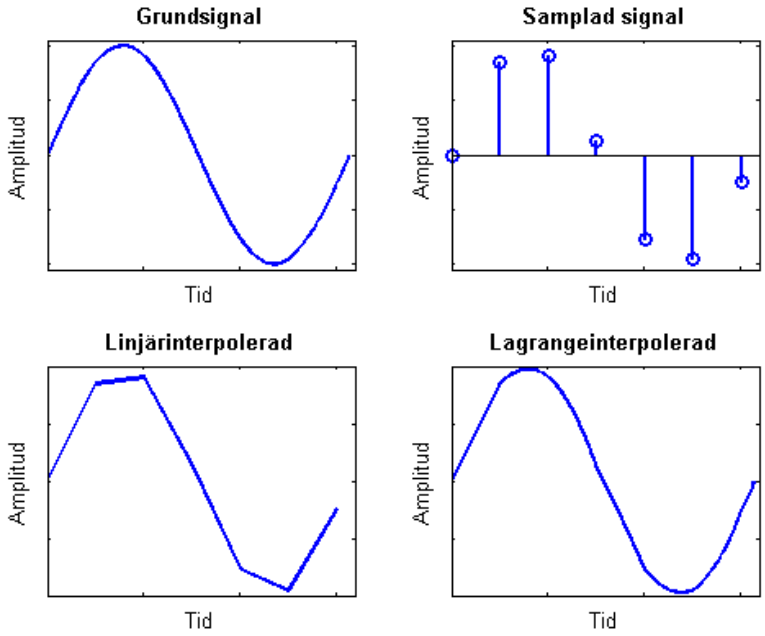


Figur 7: Illustration av hur en enkel fördröjning representeras i blockschema

Tekniken att spara ett sampel i bufferten den tid som krävs för önskad fördröjning fungerar felfritt så länge den önskade fördröjningen är en multipel av signalens samplingstid T, det vill säga att signalen fördröjs ett helt antal sampel [20]. Problem uppstår när en fördröjning av ett icke-heltal antal sampel önskas, det vill säga då D i ovanstående ekvationer och figur inte är ett heltal. Problemet är att en diskret signal inte är definierad för alla tänkbara tidpunkter, vilket omöjliggör direkt fördröjning av godtycklig längd. För att kringgå detta problem på ett bra sätt kan *interpolering* användas för att approximera vad signalen rimligtsvis har för värden mellan de diskreta tidpunkterna för vilka den är definierad. Interpolering avhandlas i nästa delavsnitt.

2.9.1 Interpolering

Interpolering är konstruktionen av en kontinuerlig signal ur en sekvens av sampel (en diskret signal) [6]. Detta görs exempelvis vid DA-omvandling. Som det är



Figur 8: Illustration av hur grundsignalen (en sinusvåg) samplas och sedan återskapas med två olika interpoleringar.

beskrivet i samplingsavsnittet kan en signal återskapas perfekt om och endast om samplingen uppfyller samplingsteoremet. En annan situation då interpolering kan användas är vid behandling av en digital signal. Om en fördröjning av ett icke-heltal antal sampel för en diskret signal önskas krävs interpolering. I detta fall behöver ett värde för signalen mellan två påföljande sampel approximeras, eftersom en digital signal är odefinierad mellan sina diskreta samplingspunkter. Det finns olika typer av interpolering (olika tillvägagångssätt för att uppskatta ett värde). En av de matematiskt enklaste formerna av interpolering är *linjär interpolering* [21].

Om det antas att samplingsperioden för en kontinuerlig signal $x(t)$ är T s ges den diskreta signalen $x(nT)$, där $n = 0,1,2,...$ via sampling. Fortsättningsvis betecknas $x(nT) = x(n)$. Om nu värdet $x(n+h)$ av den diskreta signalen $x(n)$ ska hittas, där $0 < h < 1$, krävs interpolering. Vid linjär interpolering används ekvationen:

$$x(n+h) = (1-h)x(n) + x(n+1)h \tag{11}$$

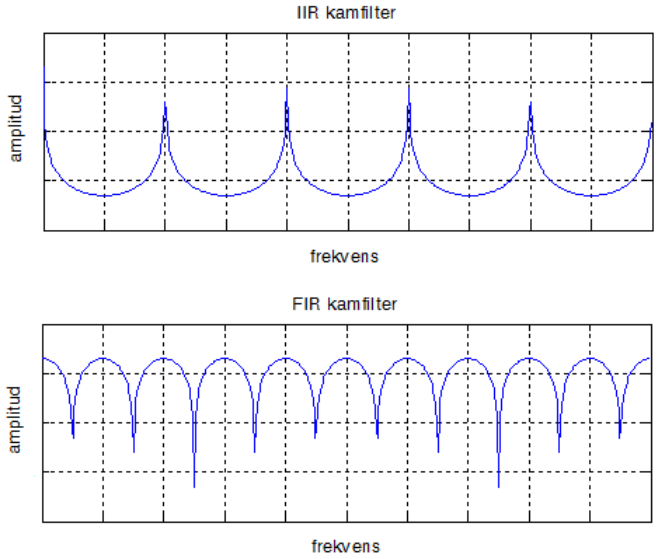
Önskas finare interpolering kan en högre ordningens interpolering användas, till exempel Lagrangeinterpolering. Vid Lagrangeinterpolering av andra ordningen [21] och med samma förutsättningar som tidigare används ekvationen:

$$x(n+h) = \frac{h(h-1)}{2}x(n-1) + (1-h^2)x(n) + \frac{h(h+1)}{2}x(n+1) \tag{12}$$

Denna interpolering ger en mindre kantig kontinuerlig signal än linjär interpolering. I figur 8 jämförs dessa två interpoleringsmetoder, där har en signal samplats, från samplingen har sedan en kontinuerlig signal återskapats med hjälp av de två interpoleringsmetoderna. Det är givetvis önskvärt att de interpolerade signalerna liknar grundsignalen.

2.9.2 Digitala kamfilter

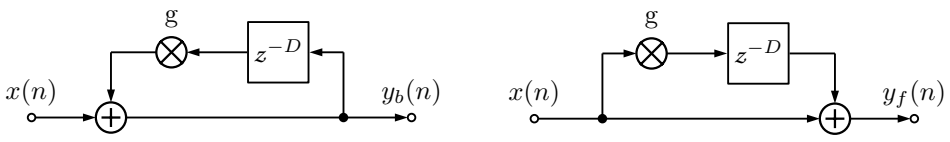
Ett digitalt kamfilter är ett system som adderar en signal fördröjd D sampel till sig själv. Detta ger upphov till interferens vilket kan ses i filtrets amplitudsvar (Figur 9) som karakteriseras av jämt fördelade spikar respektive dalar.



Figur 9: Amplitudsvar för återkopplat (ovan) och framåtmatat (under) kamfilter. Skalningsfaktor g > 0

Kamfilter kan användas för att implementera ljudeffekter som modellerar eko eller reverb. Detta genom att låta kamfiltret efterlikna akustiska stående vågor.

Det finns två typer av kamfilter - återkopplat och framkopplat, dessa illustreras i figur 10a och 10b. Bådas struktur är, utöver den variabla fördröjningen D, identiska med IIR- och FIR-filter av första ordningen.



(a) Återkopplat kamfilter. (b) Framåtmatat kamfilter.

Figur 10: Två typer av kamfilter.

Utifrån figurerna kan differensekvationerna för de två kamfilterna härledas:

$$y_b(n) = x(n) + gy(n - D) \tag{13}$$

$$y_f(n) = x(n) + gx(n - D) \tag{14}$$

Där D är fördröjningen mätt i antal sampel och g är skalningsfaktorn för den fördröjda signalen. Genom att z-transformera dessa differensekvationer fås deras frekvensfunktioner:

$$H_b(z) = \frac{Y(z)}{X(z)} = \frac{z^D}{z^D - g} \tag{15}$$

$$H_f(z) = \frac{Y(z)}{X(z)} = \frac{z^D + g}{z^D} \tag{16}$$

För att kunna studera frekvenssvaren för ekvationerna 15 och 16 i frekvensdomänen så utförs substitutionen $z = e^{j\omega}$. Därefter används Eulers formel för att uttrycka frekvenssvaret med trigonometriska funktioner. När absolutbeloppet tas på detta uttryck ges för de båda ekvationerna:

$$|H_b(e^{j\omega})| = \frac{1}{\sqrt{(1 + g^2) - 2g\cos(\omega D)}} \tag{17}$$

$$|H_f(e^{j\omega})| = \sqrt{(1 + g^2) + 2g\cos(\omega D)} \tag{18}$$

För båda ekvationerna ses här att dalarna i amplitudsvaret uppkommer vid $\omega = \frac{\pi}{D}, \frac{3\pi}{D}, \frac{5\pi}{D} \ldots$

3 Musikeffekter

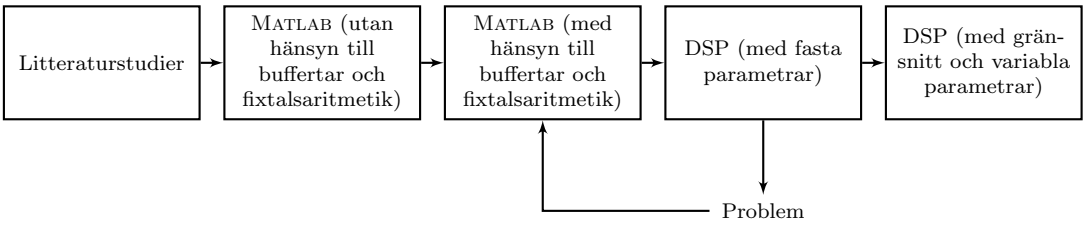
I detta avsnitt avhandlas de musikeffekter som har implementerats i projektet. Det första delavsnittet (3.1 Designmetodik) innehåller en förklaring av metodiken med vilken effekterna har utvecklats och är gemensam för alla effekterna.

Beskrivningen av varje effekt innehåller en introduktion till vad effekten är samt när och hur den kan användas. Vidare ges en teoretisk framställning av effekten och hur den kan implementeras digitalt.

Avslutningsvis ges resultatet av hur effekten har implementerats i detta projekt, det vill säga på en digital signalprocessor med hänsyn till exempelvis fixtalsaritmetik.

3.1 Designmetodik

Implementeringen av musikeffekterna följde ett gemensamt arbetsflöde, som illustreras i figur 11. Processen delades upp i flera olika arbetssteg för att systematiskt lösa problemen.



Figur 11: Arbetsflödet vid implementeringen av musikeffekterna.

Musikeffekterna på DSP:n grundades på redan existerande strukturer, verktyg och metoder. Dessa studerades och valdes ut baserat på beräkningsbelastning, minneskrav, komplexitet och önskade specifikationer.

Med hjälp av Matlab testades och utvärderades effekten först i dess grundligaste utförande, det vill säga utan att simulera DSP:ns uppbyggnad och funktion. I detta tidiga skede låg fokus istället på att finna lämpliga ekvationer för effekten, vilka parametrar som ska styras från gränssnittet, samt effektens ljudkvalitet.

Därefter implementeras effekten i Matlab med DSP:ns struktur i åtanke. Gemensamt för alla effekter var att samplingshanteringen och fixtalsanpassningen behövde ses över. Vissa effekter kräve olika implementationsmetoder som cirkulära buffertar och look-up tabeller.

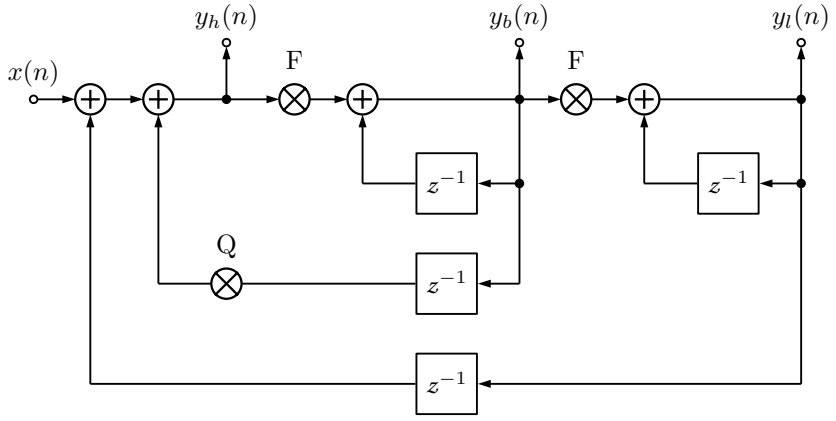
Matlab-koden översättes till C-kod och överfördes till DSP:n. Om arbetet i förra arbetssteget gjordes grundligt och noggrant var översättningen från Matlab till C-kod en relativt enkel process. Effekten implementerades med fasta parametrar för att enkelt och snabbt kunna verifiera att effekten fungerar på DSP:n. Problem som kunde uppstå här är till exempel att effekten var för beräkningstung eller felanpassad till fixtalsaritmetik.

Sist anpassades effekten på DSP:n så att dess parametrar gick att styra från användargränsnittet. Detta innebar att DSP:n behövde ha tillräckligt med beräkningskraft för att processera sampel samtidigt som den tog emot nya parametervärden från gränssnittet. När detta bekräftades fungera blev designprocessen för effekten klar.

3.2 Chamberlin state-variable filter

Olika typer av filter används på flera sätt av musiker och DJs. I syntar utgör filterdelen, tillsammans med oscillator och signalförstärkning, grunden för signalkedjan. Bland DJs används filter ofta för att enklare skapa en övergång mellan två låtar. Genom att skära bort de låga frekvenserna på ena låten och behålla dem på den andra blir övergången mer subtil.

Analoga state-variable filter används flitigt i elektriska musikinstrument. Fördelen med state-variable filter är att högpassad, lågpassad, bandpassad och bandstoppad signal kan erhållas samtidigt. Utöver inparametern brytfrekvens har den även ett, från brytfrekvensen oberoende, godhetstal Q vilket öppnar upp fler möjligheter för användaren. Vid högt Q-värde kommer filtret att självoscillera, vilket skapar intressanta ljudeffekter.



Figur 12: Chamberlin state-variable filter.

I figur 12 illusteras ett diskret state-variable filter beskrivet av Hal Chamberlin [22]. Precis som dess analoga motsvarighet har filtret en branthet på 12 dB/oktav. Chamberlin-filtrets överföringsfunktion för de olika utsignalerna ges av:

$$H_l(z) = \frac{F^2}{1 + (F^2 - (1 - FQ) - 1)z^{-1} + qz^{-2}} \tag{19}$$

$$H_b(z) = \frac{F(1 - z^{-1})}{1 + (F^2 - (1 - FQ) - 1)z^{-1} + qz^{-2}} \tag{20}$$

$$H_h(z) = \frac{(1 - z^{-1})^2}{1 + (F^2 - (1 - FQ) - 1)z^{-1} + qz^{-2}} \tag{21}$$

där F är frekvenskontrollkoefficienten och Q är godhetstalkoefficienten. Dessa ges av:

$$F = 2 \sin \frac{\pi f_c}{f_s} \tag{22}$$

$$Q = \frac{1}{q} \tag{23}$$

Där f_c är filtrets brytfrekvens, f_s är samplingsfrekvensen och q är inparameter för godhetstalet. Parametern q kan väljas från 0.5 och uppåt.

Implementation av effekten Utifrån figur 12 kan filtrets differensekvationer extraheras:

$$y_l(n) = F y_b(n) + y_l(n-1) \tag{24}$$

$$y_b(n) = F y_h(n) + y_b(n-1) \tag{25}$$

$$y_h(n) = x(n) - y_l(n-1) - Q y_b(n-1) \tag{26}$$

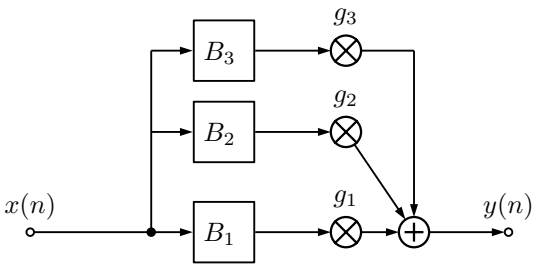
Ekvationerna 22 till 26 används för att implementera denna effekt. Genom att sätta en fast samplingsfrekvens f_s och bestämma ett intervall för brytfrekvensen f_c kan en look-up tabell genereras på förväg och på så sätt spara beräkningskraft under ljudbehandlingen. Nackdelen är att stora tabeller, som allokerar mycket minne, krävs för att erhålla god upplösning.

Resultat Effekten har implementerats på ovan beskrivna sätt både i MATLAB och på DSP:n, och fungerar felfritt i båda fallen. Beräkningsmässigt tar det ca 120 klockcykler för DSP:n att bearbeta en ljudsampel. Det motsvarar ungefär 5,8 % av DSP:ns totala beräkningskapacitet på 2083 klockcykler per sampel vid samplingsfrekvensen 48 kHz.

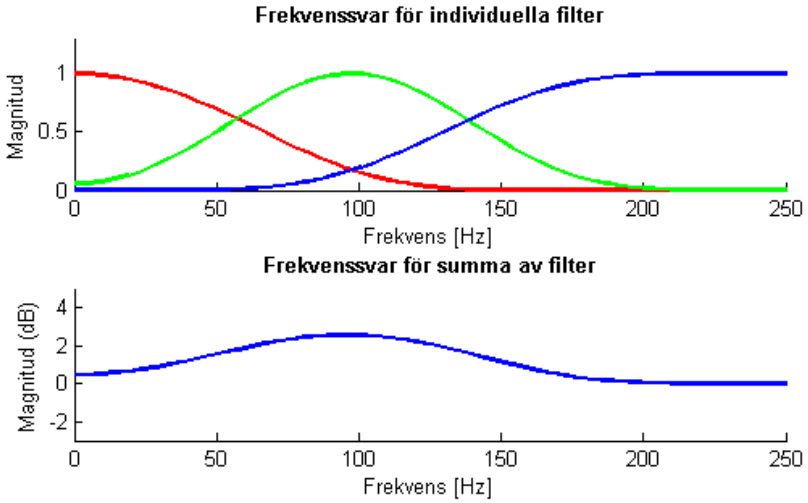
3.3 Grafisk equalizer

En equalizer, eller EQ, är ett filter som låter dig justera volymnivån av en frekvens eller ett band av frekvenser i en audiosignal. I sin enklaste form låter en equalizer användaren justera bas och diskant, medan tillämpningar inom till exempel ljudinspelning och live-konserter ofta kräver en mycket precisare kontroll av ljudbilden. En vanlig equalizervariant är den grafiska equalizern som låter användaren kontrollera volymen för ett antal fixa frekvensband. Antalet band varierar normalt mellan 7 till 31 stycken och delar in frekvensspektrumet i hela oktaver, (begreppet oktav introduceras i avsnitt 2.2 Musikteori) 2/3 oktav eller 1/3 oktav i fallet med 31 band.

En grafisk equalizer konstrueras vanligtvis med hjälp av en så kallad filterbank. Ljudet går parallellt genom flera bandpassfilter som vardera filtrerar ut ett av equalizerns frekvensband. De multipliceras vardera med en skalfaktor innan signalen adderas ihop igen. En schematisk bild för att visa principen visas i figur 13.



Figur 13: Blockshema för grafisk equalizer konstuerad med filterbank om tre band som var skalas med faktorn g_1, g_2 respektive g_3.



Figur 14: Exempel på grafisk EQ implementerad med filterbank om tre FIR-filter av vardera ordning 1024. Detta är uppenbart en för låg ordning för att ge ett summerat flackt frekvenssvar, vilket ses i den nedre figuren.

För att banden inte ska överlappa varandra krävs att bandpassfiltren är mycket branta och därmed har en hög filterordning. Därför används FIR-filter då IIR-filter lätt blir instabila vid högre ordningstal. Ett exempel på en equalizer med tre band/filter ses i figur 14. Den är implementerad med FIR-filter av ordning 1024. I figuren ser vi tydligt att filtren har en för låg ordning för att de summerat ska ge ett flackt frekvenssvar.

Design av equalizerns band och filter I projektet har en grafisk equalizer med åtta band implementerats (se tabell 1). Det första bandet täcker subkontra- och kontraoktaven samt stora oktaven. Banden täcker sedan varsin oktav fram till sjunde bandet där till exempel den ljusaste tonen på ett piano eller en piccolaflöjt befinner sig. Det sista åttonde bandet täcker resten av frekvensspektrat till och med 16640 Hz. Amplituden för övriga frekvenser upp till Nyquistfrekvensen, dvs halva samplingsfrekvensen $fs/2 = 24000$ Hz sätts till noll då dessa höga frekvenser ger ett instabilt beteende vid filtrets implementering på DSP:n.

Tabell 1: Frekvensband för den grafiska equalizern.

Band	Från [Hz]	Till [Hz]
1	0	130
2	130	260
3	260	520
4	520	1040
5	1040	2080
6	2080	4160
7	4160	8320
8	8320	16640

Då en filterbank kräver filter av hög ordning (flera tusen) och då detta är mycket beräkningskrävande valdes en annan metod för att utforma equlizern än att använda en filterbank. Alla banden i equalizern implementerades istället med ett och samma FIR-filter som designas med metoden *Frequency sampling*. Metoden går ut på att utifrån ett givet, önskat frekvenssvar $H = Ae^{\mathrm{j}\phi}$, sampla detta i N punkter, där funktionen A i detta fallet utgörs av amplituden hos equalizerns olika band.

$$H_d(e^{\mathrm{j}\omega_k}) = H(k), \qquad \omega_k = \frac{2\pi k}{N}, \qquad k = 0, 1, \ldots, N-1 \tag{27}$$

Filterkooefficienterna h ges då av den inversa diskreta fouriertransformen:

$$h(n) = \mathrm{IDFT}[H(k)], \qquad n = 0,1,\ldots,N-1 \tag{28}$$

För att filterkoefficienterna $h(n)$ ska vara reella, något som inte garanteras av fouriertransformen, krävs utan närmare matematisk motivering följande:

1. $H(0)$ måste vara ett reellt tal
2. $H(N-k) = H^*(k)$, dvs komplexkonjugerad
3. Om N är ett jämt tal så krävs att $H(N/2) = 0$.

Filtret h som skapas kommer att ha ett frekvenssvar som exakt motsvarar det önskade frekvenssvaret i de N samplade punkterna. Mellan sampelpunkterna finns dock i detta fall ingen explicit kontroll över filtrets frekvenssvar. För att filtret ska bete sig bra mellan punkterna måste filtrets fasfunktion ϕ vara konsistent med filtrets längd N. Ett kausalt filter måste ha en gruppfördröjning på minst $g = N/2$. För att filtret ska vara fritt från fasdistortion ska det ha ett linjärt fassvar, vilket motsvarar en konstant gruppfördröjning. Därför väljs ett fassvar ϕ samplat i N punkter:

$$\phi(\omega_k) = \frac{-g \cdot 2\pi k}{N} = -\pi k, \qquad k = 0,1,\ldots,N-1 \tag{29}$$

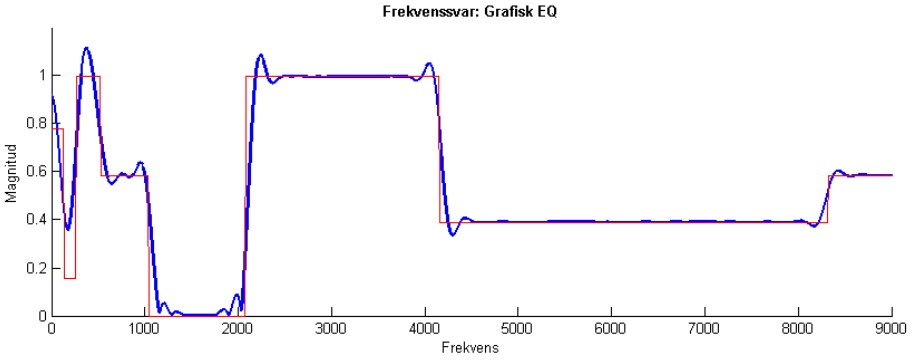
Implementation på DSP:n Equalizerns filterkoefficienter räknades ut på DSP:n med följande steg. För att kunna använda snabb fouriertransform (FFT) krävs att N är en tvåpotens och N valdes till 512. Detta värde är en avvägning mellan beräkningshastighet och filterprestanda.

1. Det önskade frekvenssvarets magnitud samplas i $N/2$ punkter från 0 Hz till $fs/2$ och sparas i vektorn H.
2. Magnituden multipliceras med fasen e^{ϕ}, vilket med vald fas innebär att värdena i H multipliceras omväxlande med 1 och -1. Produkten sparas i H.
3. Resten av H, samplingspunkter $N/2+1$ till N, skapas genom att sätta $H(N-k-1) = H(k+1)$, $k = 0,1,\ldots,N-1$.
4. $H(N/2) = 0$ då N väljs till en jämn tvåpotens för att i nästa steg kunna använda den snabba fouriertransformen (FFT).
5. Filterkoefficienterna $h(n)$ beräknas ur H med hjälp av co-processorns hårdvaruaccelererade snabba inversa fouriertransform.
6. $h(n)$ multipliceras med en fönsterfunktion för att minska rippel i pass- och stoppband.

FIR-filtret implementerades sedan på DSP:n i tidsdomänen med hjälp av assemblerkod för då c-kods implementering var för långsam.

Resultat I sin första utformning i MATLAB beräknades den grafiska equlizerns filterkoefficienter med Parks-McClellans metod. Då denna metod visade sig vara svår att anpassa till fixtalsaritmetik ändrades designen till ovan beskrivda *Frequency sampling method* som med framgång fixtalsanpassades och implementerades på DSP:n.

Att beräkna FIR-filtrets koefficienter för filterlängd 512 på DSP:n tar cirka 86000 klockcykler. Att sedan tillämpa FIR-filtrering med assembler-funktionen tar ca 600 klockcykler per sampel. Detta motsvarar ungefär 29 % av DSP:ns totala beräkningskraft per sampel. Ett exempel på hur equalizern kan se ut i frekvensdomänen med olika amplituder på de olika banden ses i figur 15. För att minska ripplet något i stopp- och passband så har filtret multiplicerats med en fönsterfunktion av typen Tukey. Ripplet kan närmast helt filtreras bort med hjälp av andra fönsterfunktioner till priset av en långsammare övergång mellan stopp- och passband.



Figur 15: Exempel hur amplituden på de olika banden i den grafiska equalizern kan vara inställd. I rött visas det ideala fallet och i blått det faktiska frekvenssvaret från DSP:n, filterlängd 512, multiplicerats med fönsterfunktion av typen Tukey.

3.4 Vibrato

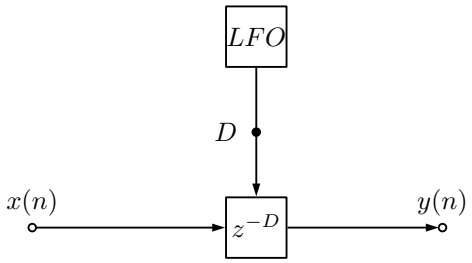
Vibrato är en utbredd musikeffekt som länge använts i många typer av musik, såsom klassisk musik, jazz, folk och popmusik [23]. Effekten innebär en periodisk variation av tonhöjd, dvs tonens frekvens. På ett stråkinstrument åstadkoms detta genom att utföra små periodiska variationer av fingrets position på en sträng när stråken används. En musikeffekt som liknar vibrato och som ofta används tillsammans med vibrato är tremolo [24], se nästa avsnitt.

Detta kan liknas vid det fysikaliska fenomenet dopplereffekten. Då en polisbil med sirener igång kör förbi en åhörare kan åhöraren höra en tonhöjdsförändring i ljudet. Det här beror på förändringen i avstånd mellan åhöraren och bilen, det vill säga en förändring i avstånd mellan ljudkällan och åhöraren kommer att införa en förändring i fördröjningen mellan det att ljudet "spelas" och att du hör det. Större avstånd betyder större fördröjning på grund av ljudets utbredningshastighet, vilket upplevs som en förändring i tonhöjd [25].

Implementation av effekten Digitalt kan ett vibrato skapas på likande sätt som polisbilen i exemplet ovan producerar en tonhöjdsförändring, det vill säga genom att förändra fördröjningen. Fördröjning varieras periodiskt enligt en så kallade Low Frequency Oscillator (LFO). LFO:n har typiskt för vibrato en frekvens av $5 - 14\,\text{Hz}$ [25]. Bredden för fördröjningen, det vill säga den maximala fördröjningen under en period är vanligtvis $5 - 10\,\text{ms}$.

Implementation av ett digitalt vibrato kräver en LFO som bestämmer hur variationen av fördröjningen ska ske, samt en digital fördöjningslinje (ett element som kan fördröja en signal ett önskat antal gånger). LFO:n kan variera enligt exempelvis en sinus- eller triangelvåg. På en signalprocessor kan detta implementeras på olika sätt, till exempel genom att varje gång beräkna ett nytt värde för LFO:n enligt:

$$D = 1 + d + \sin(2\pi f n)d \tag{30}$$



Figur 16: En förenklad vibrato-effekt, LFO-blocket genererar en periodiskt varierande fördröjning och fördröjnings-blocket fördröjer insignalen därefter.

I ekvationen ger D längden av fördröjningen i antal sampel, d är bredden av vibraton (det vill säga den bestämmer vilka fördröjningar som är möjliga för vibraton) och f är frekvensen av LFO:n i Hz. Med dessa förutsättningar kan utsignalen för vibraton $y(n)$ uttryckas som:

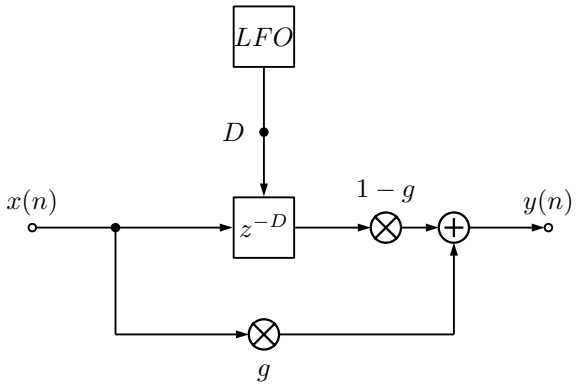
$$y(n) = x(n - D) \tag{31}$$

En annan möjlighet för att implementera en LFO är genom att använda sig av vågtabeller, där värdena för LFO:n är sparade och kan hämtas vid behov.

För att kunna använda sig av fördröjningar på DSP:n krävs att tidigare insampel sparas i en buffert. Som beskrivs i 2.8.1 Cirkulär buffert görs detta bäst för realtidsprocessering med hjälp av just en cirkulär buffert. När bufferten konstrueras är det speciellt viktigt att se till att den lagrar tillräckligt många sampel för den längsta fördröjning som kan användas för vibraton. Sampel för utsignalen med varierande fördröjning relativt senaste insampel kan sedan hämtas ur denna buffert och skapa den nya utsignalen. Ett blockschema för en enkel vibrato-effekt visas i figur 16.

För att skapa en mjukare utsignal kan det vara viktigt att inte enbart kunna fördröja signalen ett heltal antal sampel [26]. Detta kan göras genom interpolering av närliggande insampel (se 2.9.1 Interpolering).

Resultat I detta projekt har ett vibrato med många av de egenskaper som beskrivs ovan implementerats. Implementationen av vibrato använder sig av en LFO som varierar enligt en sinusvåg. Dess värden hämtas ur en vågtabell sparad i minnet, tabellen innehåller 2^9 element, upptar 2^{13} bitar och har en upplösning 2^{-8}. En cirkulär buffert används för att möjliggöra fördröjning vid realtidsprocessering och linjär interpolering har använts för att skapa en mjukare utsignal. Effekten har implementerats både i MATLAB och på DSP:n med framgång. Beräkningsmässigt tar det ca 180 klockcykler för DSP:n att bearbeta en ljudsampel med vibratoeffekten. Det motsvarar ungefär 8,6 % av DSP:ns totala beräkningskapacitet på 2083 klockcykler per sampel vid samplingsfrekvensen 48 kHz.



Figur 17: En förenklad flanger-effekt, LFO-blocket genererar en periodiskt varierande fördröjning och fördröjnings-blocket fördröjer insignalen därefter.

3.4.1 Flanger

En effekt som är väldigt lik vibrato och som enkelt kan implementeras då ett vibrato tagits fram är flanger. Denna effekt ger en känsla av att fler instrument spelar samtidigt, vilket beror på att flanger är en viktad summa av den insignalen och utsignalen av ett vibrato. Detta visas i figur 17, där g är en skalfaktor med värden inom intrevallet $0 - 1$ som avgör blandningen mellan icke-processerat och processerat ljud för utsignalen av flanger.

Resultat En flanger-effekt som bygger på det tidigare beskrivna vibratot har implementerats i projektet och fungerar både på DSP:n och i MATLAB. Beräkningsbelastningen för flangern är ca 190 klockcykler per sampel på DSP:n och detta motsvarar ungefär 9,1 % av DSP:ns totala beräkningskrafft.

3.5 Tremolo

Tremolo är en typ av amplitudmodulation som ger en dramatisk och mer intensiv känsla av musiken. Beroende på avsikt kan effekten användas försiktigt eller mer tydligt [27]. Instrument som tremolo ofta används för är elgitarr och orgel, men effekten kan användas för i princip vilket instrument som helst.

På samma sätt som vibratoeffekten använder sig tremolot av en LFO för periodisk modulering, men nu av signalens amplitud istället för dess tonhöjd. LFO:n för tremolo kan variera enligt sinus- triangel- eller fyrkantsvåg [27]. Det normala frekvensområdet för ett tremolo LFO är 1-10 Hz [28].

De parametrar som behövs för att på ett nöjaktigt sätt styra tremoloeffekten är modulationsfrekvensen (LFO-frekvensen) samt effektens djup. Med djupet för en tremolo avses hur mycket insignalens amplitud ska förändras som mest. Djupet kan anges i procent $0 - 100\,\%$, där $0\,\%$ betyder ingen effekt, medan $100\,\%$ betyder att amplituden för utsignalen kommer att ha högsta amplitudmodulering två gånger insignalen och lägsta modulering noll gånger insignal [27].

Matematiskt kan tremoloeffekten beskrivas med:

$$y(n) = x(n)(1-d) + x(n)\sin(2\pi f n)d \tag{32}$$

där x är insignal och y är utsignal, d är djupet och f är frekvensen.

Implementation av effekten Tremolot som implementerats i detta projekt arbetar enligt ekvation 32 ovan. Sinusvågen har implementerats med en 2^9 element lång vågtabell med en upplösning på 2^{-8}. Nedan visas hur effekten har implementerats med C-kod och fixtalsaritmetik på DSP:n.

```
1 sinus = LFO(frekvens);

3 modulator = (2^13-djup*2^8) + djup*sinus;

5 y = (Int32) x * modulator;

7 return (Int16) (y >> 13);
```

C-kod 1: Kodutsnitt för implementering av tremoloeffekten.

`djup` och `frekvens` är inparametrar till tremolofunktionen. Djupet anges på Q10.5-format, det vill säga ett heltal i intervallet $0 - 2^5$ och representerar ett decimaltal $0-1$. Funktionen `LFO(frekvens)` ger det aktuella värdet för sinusvågen ur vågtabellen, värdena i vågtabellen representeras på Q7.8-format. På rad 3 beräknas värdet av moduleringen för insignalen, dvs $(1-d)+\sin(2\pi f n)d$. På denna rad (3) är `modulator` ett Q2.13 tal. På rad 5 sker multiplikationen mellan insignalen och modulatorn, som sparas i en 32-bitars integer för att förhindra overflow. På sjunde raden sker en nedskiftning med 13-bitar och utsignalen returneras som en 16-bitars integer på formatet Q15.

Resultat Effekten har implementerats på ovan beskrivna sätt både i MATLAB och på DSP:n, och fungerar felfritt i båda fallen. Tremoloeffekten tar cirka 120 klockcykler per sampel att beräkna vilket motsvarar ungefär 5,8 % av DSP:ns totala beräkningskraft på 2083 klockcykler per sampel.

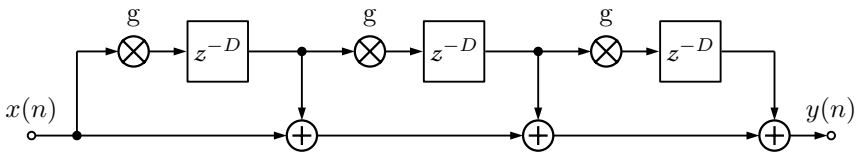
3.6 Eko

Ett eko är en upprepning av ett originalljud efter en viss tid och med en förändring i amplitud. Ljudeffekten eko imiterar ljudfenomenet då akustiska ljudvågor studsar mot tillräckligt stora föremål långt borta och återhörs. Mer om detta finns i avsnitt 2.3 Rumsakustik.

Inom ljudbehandlingen har eko-effekten återskapats på många sätt. De vanligaste analoga eko-effekterna använder sig av band som lagrar ljud och spelar

upp ljudet efter en viss tid. Genom att justera bandlängden och förstärkningen justeras längden och volymen på ekot [29].

Inom den digitala ljudbehandlingen är ekot en simpel effekt som kan vara uppbyggt av ett återkopplat kamfilter (figur 10a). Fördelen med denna uppbyggnad är att en enkel implementation ger eko som återkommer flera gånger. Nackdelen är att effekten blir svårstyrd på grund av att antalet studsningar bara går att styra med hjälp av skalningsfaktorn g.



Figur 18: Tre stycken framåtmatade kamfilter i serie.

Implementation av effekten Ett annat alternativ är att använda ett framkopplat kamfilter (figur 10b). I denna form kommer ljudet att upprepas en gång skalat av g. Genom att koppla flera kamfilter i serie kan antalet upprepningar väljas exakt oavsett skalningsfaktorn g. Denna struktur med tre stycken framåtmatade kamfilter i serie illustreras i figur 18 och dess differensekvation blir:

$$y(n) = x(n) + gx(n - D) + g^2x(n - 2D) + g^3x(n - 3D) \tag{33}$$

Nackdelen med denna typ av implementation är att varje steg av filter behöver en egen buffert för att fördröja signalen. Vid långa fördröjningar av signalen behöver dessa buffertar vara mycket stora och kräver därför mycket minne.

Maximal fördröjning sattes till 12000 sampel vilket motsvarar en fjärdedels sekunds fördröjning då $f_s = 48$ kHz. Den totala buffertstorleken blir då $3 * 12000 * 2 = 72kB$. Det sista ekot uppkommer 0,75 sekunder efter sitt ursprung. Fördröjningen D är en utav två inparametrar till effekten.

Vid implementationen kan skalningsfaktorerna för de tre filterna vara tre separata variabler för individuell styrning men valdes istället att sättas till en gemensam variabel. Detta ger en exponentiell dämpning av ljudet som uppfattas som naturlig. Det innebär även att effekten behöver färre inparametrar vilket leder till ett enklare användargränsnitt.

Resultat Effekten har implementerats framgångsrikt både i MATLAB och på DSP:n. Ursprungligen försöktes effekten implementeras med 24000 sampel fördröjning, vilket ger en fördröjning på 0,5 sekunder per kamfilter. Detta fungerade inte på grund av minnesbrist. För att processera ljudet på DSP:n använder ekot ca 150 klockcykler per sampel. Det motsvarar ungefär 7,2 % av DSP:ns totala beräkningskraft på 2083 klockcykler per sampel.

3.7 Reverb

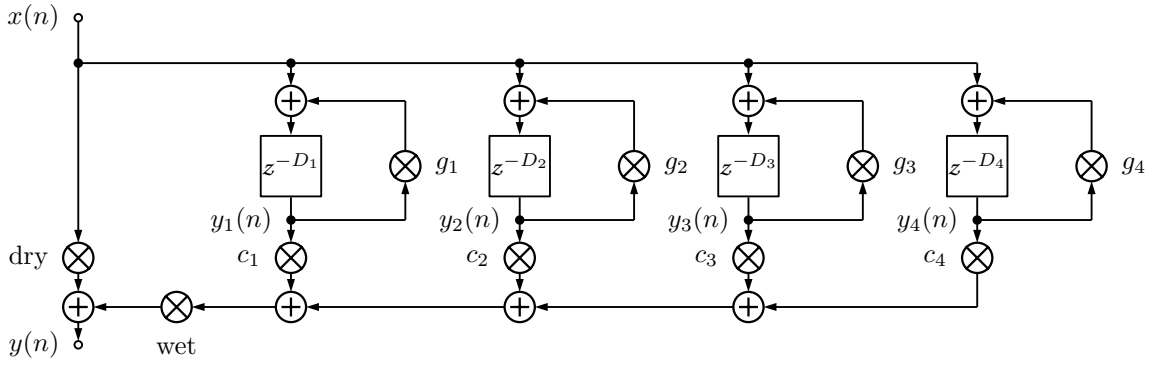
Reverb är en ljudeffekt vars uppgift är att efterlikna ett rums akustik, som togs upp i avsnitt 2.3 Rumsakustik. Musiker använder ofta denna effekt vid live eller

studioinspelningar när naturligt rumsklang inte finns att tillgå.

Digitala reverb använder sig utav matematiska algoritmer som modellerar det naturliga fenomenet. Enkla reverb-algoritmer består i huvudsak av flera återkopplade kamfilter för att skapa ett stort antal korta avtagande ekon. Schroeder tog 1961 fram ett system bestående utav parallellkopplade kamfilter i serie med allpass filter för att ytterliggare sprida ut ekona [30]. Sen dess har otaliga system av filter tagits fram för att försöka efterlikna naturligt rumsklang [31].

Ett helt annat sätt att digitalt skapa rumsklang är så kallade faltningsreverb. Dessa reverb använder sig utav impulssvar inspelade från den plats vars rumsklang användaren vill simulera. En ljudsignal faltas med detta impulssvar för att applicera det rummets klang på signalen [31].

Ett reverb i dess enklaste form kan realiseras genom att använda endast ett återkopplat kamfilter (figur 10a) med lämpliga värden för fördröjningen D och skalningsfaktorn g. Detta återspeglar dock inte ett verkligt rumsklang då ljudet studsar mot flera objekt tillbaka till lyssnaren.



Figur 19: System av parallellkopplade kamfilter.

Implementation av effekten Genom att parallellkoppla flera kamfilter fås ett mer verklighetstroget resultat. Varje kamfilter har olika inställningar för fördröjningen D och skalningsfaktorn g. Därefter adderas dessa signaler ihop till en utsignal, vilket visas i figur 19.

Differensekvationen för detta system blir:

$$y(n) = x(n)\text{dry} + \text{wet}(c_1 y_1(n) + c_2 y_2(n) + c_3 y_3(n) + c_4 y_4(n)) \tag{34}$$

Där dry = 1 - wet, wet är skalningsfaktor för den reverberade utsignalen. $c_{1,2,3,4}$ är skalningsfaktorer för varje kamfilters utsignal. $y_{1,2,3,4}$ ges av:

$$y_i(n) = x(n - D_i) + g_i y_i(n - D_i) \tag{35}$$

Skalningsfaktorerna c_i kan vara variabla så att användaren själv kan mixa mellan de olika kamfilterna. Detta ökar dock antalet inparametrar till effekten. Istället valdes dessa till $c_i = 0{,}25$ för att erhålla en jämn balans mellan kamfilterna. Vidare kan även wet, som i sin tur används av dry, vara en inparameter men även

denna valdes att vara ett konstant värde för att hålla nere antalet parametrar att styra. Denna sattes till wet = 0,5.

Lämpliga värdena för skalningsfaktorerna g_i och antalet fördröjda sampel D_i togs fram genom tester och lämpliga värden uppfattades för D_i ligga mellan 50 och 100 ms och för g_i mellan 0,7 till 0,8. För att förenkla gränsnittet för effekten anpassas styrningen av parametrarna så att endast det första kamfiltrets parametrar (D_1 och g_1) sätts av användaren. Utifrån dessa parametrar beräknas övriga kamfilters parametrar med fasta differenser, vilket visas i tabell 2.

Tabell 2: Differenser för de två parametrarna g_i och D_i. Framtagna med f_s satt till 48 kHz.

Kamfilter i	g_i [-]	D_i [antal sampel]
1	0	0
2	+0,03	-144
3	-0,02	+720
4	-0,04	+1104

Resultat Reverbeffekten beskrivet ovan har implementerats framgångsrikt både i MATLAB och på DSP:n. En skillnad mellan dessa är att på DSP:n sätts skalfaktorerna g_i till samma värde. Detta på grund av tidsbrist vid implementeringen på DSP:n. Det på DSP:n implementerade reverbet använder cirka 260 klockcykler för att beräkna ett sampel. Detta motsvarar ungefär 12 % av DSP:ns totala beräkningskraft på 2083 klockcykler per sampel vid samplingshastigheten 48 kHz.

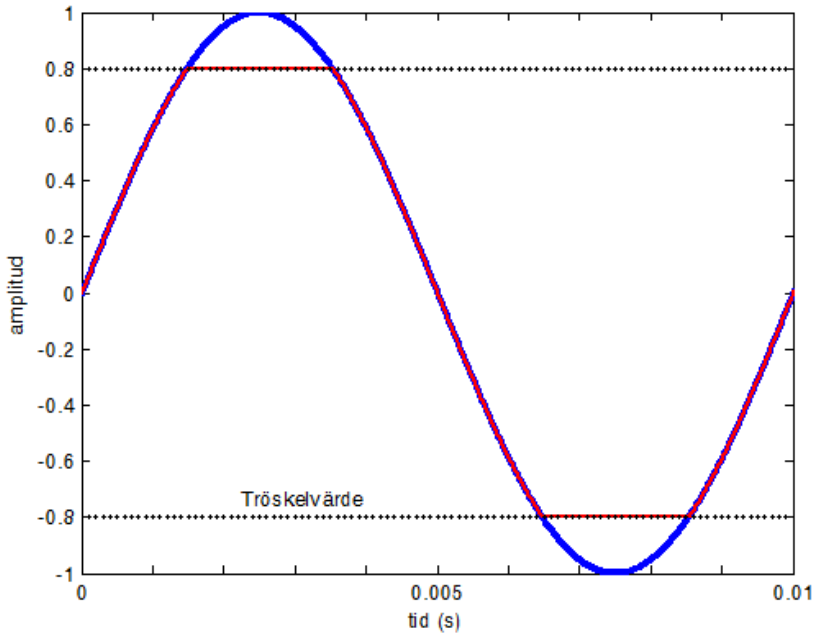
3.8 Distortion

Inom rockmusiken är den distorderade gitarren en central del. Effekten uppkom genom att musiker höjde volymen på deras förstärkare över den gräns som förstärkaren var designad för. Signalens dynamik överstiger då förstärkarens förmåga att förstärka signalen utan att förvränga den. Denna form av distortion kallas för klippning [32].

Musiker har sedan dess återskapat denna effekt på ett mer kontrollerat sätt genom att använda sig av analoga kretsar designade just för att klippa signalen. Fördelen är att flera parametrar kan styras och på så sätt kan musikern utforma ljudet efter smak [32].

Klippning delas normalt upp i två delar: mjuk och hård klippning. Vid hård klippning klipps signalen av vid ett visst tröskelvärde för amplituden. Detta illustreras i figur 20. Vid mjuk klippning dämpas signalen ovanför ett visst tröskelvärde och ger då inte en lika stor förändring på signalen som hård klippning. För en sinuston som klipps innebär det att utsignalen får flera övertoner adderade till grundtonen [28].

Inom den digitala signalbehandlingen uppstår distortion då ett sampel begränsas av den tillämpade datatypen. Till exempel för en 16-bit *signed integer* är det största positiva värdet som kan representeras 32767. Om samplets värde



Figur 20: Blåa linjen visar insignalen. Den röda visar den klippta utsignalen.

skulle överstiga 32767 sparas värdet trots det som 32767 (förutsatt att overflow förhindras) vilket resulterar i en förvanskning av signalen då information går förlorad. Detta är i normala fall en oönskad effekt men används med fördel för att skapa en distortionseffekt.

Implementation av effekten Hård klippning, enligt figur 20, kan, för ett insampel x och utsampel $f(x)$, beskrivas med :

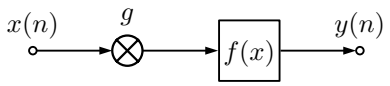
$$f(x) = \begin{cases} p & \text{om } x \geq p \\ x & \text{om } p > x > -p \\ -p & \text{om } x \leq -p \end{cases} \tag{36}$$

där p är tröskelvärdet.

I C-kod implementeras detta enkelt med en if-sats, vilket visas följande kodexempel.

```
if (x > p)
    x = p;
else if (x < -p)
    x = -p;
return x;
```

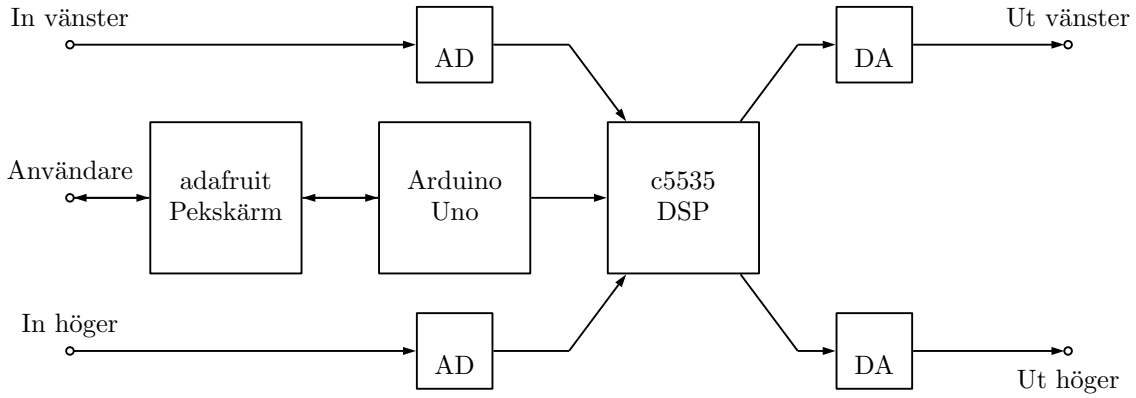
C-kod 2: Implementering av distortioneffekten.



Figur 21: Distortioneffektens struktur.

Genom att multiplicera varje sampel med en faktor $g > 1$ innan klippningen utförs kan en starkare effekt skapas eftersom fler sampel kommer att vara större än tröskelvärdet `p`. Effektens struktur visas i figur 21.

Resultat Effekten har implementerats med framgång både i MATLAB och på DSP:n. Att beräkna en ljudsampel på DSP:n tar cirka 110 klockcykler och det motsvarar ungefär 5,3 % av DSP:ns totala beräkningskraft på 2083 klockcykler per sampel.



Figur 22: Flödesschema för produkten.

4 Systembeskrivning

Detta kapitel syftar till att beskriva den produkt som tagits fram i projektet. Genom att beskriva hur dess olika delar fungerar och hur de kopplas ihop ges en helhetsbild. I figur 22 visas ett flödesschema över produktens huvudkomponenter.

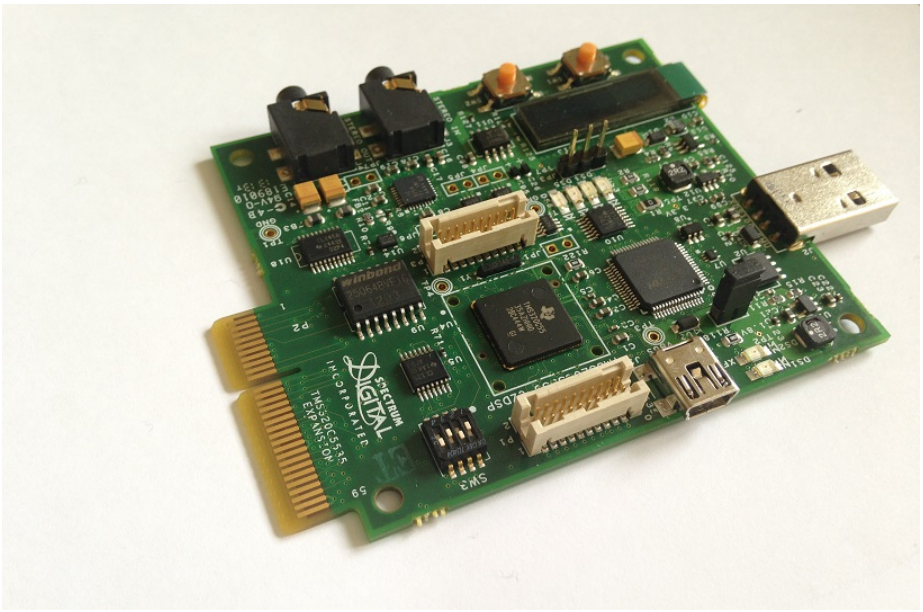
Inledningsvis i detta avsnitt beskrivs de tre huvudsakliga hårdvarukomponenternas (DSP, Arduino, pekskärm) egenskaper var för sig. Vidare beskrivs grundläggande hur enheterna sammanfogas med hjälp av ett produktskal samt hur de kopplas ihop och kommunicerar med varandra. Slutligen beskrivs den mjukvarumässiga programstrukturen för både DSP:n och gränssnittet samt gränssnittets utformning.

En ingående redogörelse för hur olika data- och kommunikationsprotokoll (Avrbott, I^2C, I^2S, UART, SPI och DMA) som använts fungerar finns i appendix B Dataprotokoll.

4.1 Utvecklingskort - eZdsp5535

Ljudeffekterna som behandlas i detta projekt har implementerats på ett utvecklingskort för signalbehandling från Spectrum Digital. Utvecklingskortet heter *eZdsp5535* och visas i figur 23. Detta är en enhet med måtten 8,5 x 6,7 cm. På utvecklingskortet sitter en signalprocessor (DSP) *TMS320C5535* från Texas Instruments. Kortet ansluts till en dator för programmering i utvecklingsmiljön *Code Composer Studio*. Anslutning sker via USB-kontakt och det är även via denna kontakt som kortet strömförsörjs. Kortet har en inbyggd *XDS100-JTAG* emulator som möjliggör styrning av exekveringen och felsökningen av kod via utvecklingsmiljön. I *Code Composer Studio* kallas detta debug-läget.

Utvecklingskortet erbjuder goda möjligheter för behandling av ljud genom en



Figur 23: Utvecklingskortet *eZdsp5535*.

ljudkodek som kommer att beskrivas mer ingående i 4.2 Ljudkodek - TLV320AIC3204. Anslutning till kortet för ljudkällor och högtalare eller hörlurar sker via 3,5 mm-stereokontaker. Minneskortsläsare för microSD kort finns på kortets undersida och kan användas för att lagra program utan strömtillförsel. På kanten av utvecklingskortet finns expansionsanslutningar för inkoppling av periferienheter via olika kommunikationsprotokoll, exempelvis UART, SPI och GPIO.

TMS320C5535 som sitter på kortet är en energisnål fixtalsprocessor för signalbehandling. Vid en spänning av 1.3 V har processorn klockfrekvensen 100 MHz. Processorn har två stycken ALU:er och två stycken MAC-enheter. Utöver detta har den en hjälpprocessor för hårdvaruaccelererad beräkning av Fast Fourier Transform (FFT) i upp till 1024 punkter.

4.2 Ljudkodek - TLV320AIC3204

En ljudkodek är en enhet som kan omvandla analoga ljudsignaler till digital form (AD) för bearbetning i till exempel en DSP och samtidigt omvandla digitala ljudsignalerna till analog form (DA) [33]. Det finns många parametrar som definierar en specifik ljudkodek men två viktiga är upplösning och samplingsfrekvens. Upplösningen bestämmer i hur många diskreta punkter en analog inspänning kan delas upp i och ges av 2^n där n är antalet bitar ljudkodeken arbetar med. Samplingsfrekvensen brukar vara över 40 kHz eftersom hörbart ljud brukar ligga på upp till 20 kHz som tidigare nämnts. Dock kan högre samplingsfrekvenser vara användbara för brusreduktion eller för att utnyttja specifika filterdesigner. Högre samplingsfrekvenser blir mer krävande för ljudprocesseringen. För att minska resursanvändning kan därför lägre samplingsfrekvenser användas. Detta är användbart under förutsättning att det samplade ljudet inte

innehåller för höga frekvenser.

Ljudkodeken som används i *eZdsp5535* (*TLV320AIC3204*) har upp till 32-bitars upplösning [34] och arbetar med samplingsfrekvenser på 8 kHz mono upp till 192 kHz stereo både för inspelning (AD-omvandling) och uppspelning (DA-omvandling) [35]. Den har även två programmerbara ingångsförstärkare vars förstärkning kan varieras mellan 0 dB och 47,5 dB. Även om *eZdsp5535* bara har ingångar för 3,5 mm ljudkablar finns det möjlighet att koppla in 6,25 mm ljudkablar via adapter [36]. Det går att direkt koppla in en mikrofon eller gitarr till DSP:n under förutsättning att ingångsförstärkningen ställs in på rätt sätt.

Kodeken initeras och styrs av DSP:n genom att använda kommunikationsprotokollen SPI eller I^2C [35]. För ljudöverföring mellan kodeken och resten av DSP:n finns det också flera överföringsmetoder att välja mellan: R (Right-Justified), L (Left-Justified), DSP och I^2S. I produkten används I^2C för att sköta initering och styrning av kodeken medan I^2S används för ljudöverföring (båda protokollen beskrivs i detalj i Appendix B Dataprotokoll).

4.3 Arduino Uno

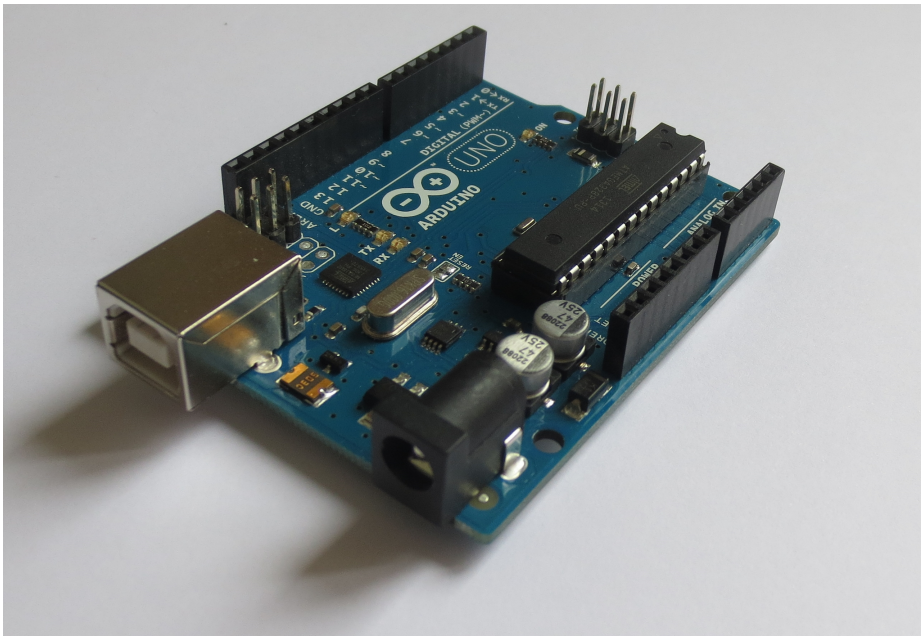
I produkten används mikroprocessorenheten *Arduino Uno R3* (se figur 24) som en länk mellan signalprocessorn och pekskärmen. Mikroprocessorenheten bygger på ATmega328 som är en 8-bitars mikroprocessor med klockfrekvens upp till 20 MHz och inbyggt flashminne på 32 kByte [37]. På själva mikroprocessorsenheten finns det sex analoga ingångar, 14 digitala in- och utgångar, en USB-ingång, en strömingång och en keramisk resonator på 16 MHz [38]. Enheten kräver en matningsspänning på 7-12 V via USB, strömadapter eller batteri och dess klockhastighet är 16 MHz. Kommunikation mellan *Arduino Uno* och andra enheter kan åstadkommas genom protokoll såsom UART, USB, SPI och I^2C.

För att kompilera och ladda upp kod på *Arduino Uno* krävs det ett kompileringsprogram som Eclipse [39] eller Arduinos egna utvecklingsmiljö [38]. Kod som laddas upp till mikroprocessorsenheten bränns automatiskt ner till dess flashminne. Detta innebär att det senast uppladdade programmet kommer starta varje gång enheten är spänningssatt.

4.4 Användargränssnitt

Ett användargränssnitt är en förbindelse mellan en mänsklig användare och ett system, där användaren kan ge kommandon som anger vilka uppgifter som systemet ska utföra, och systemet kan via gränssnittet i sin tur återkoppla sitt tillstånd till användaren. Produkten presenterar musikeffekter tillgängliga till användaren samt tar emot effektparametrar via ett grafiskt användargränsnitt i form av en pekskärm.

Ett grafiskt användargränsnitt återkopplar ett systems tillstånd genom visuell åskådliggörning. En pekskärm är en skärm som samtidigt som den kan återge visuell information, kan översätta en användares tryck på skärmen till koordinater som ett datorsystem kan använda för att tolka inkommandon. Detta görs antingen med resistiva eller kapacitiva skärmar.



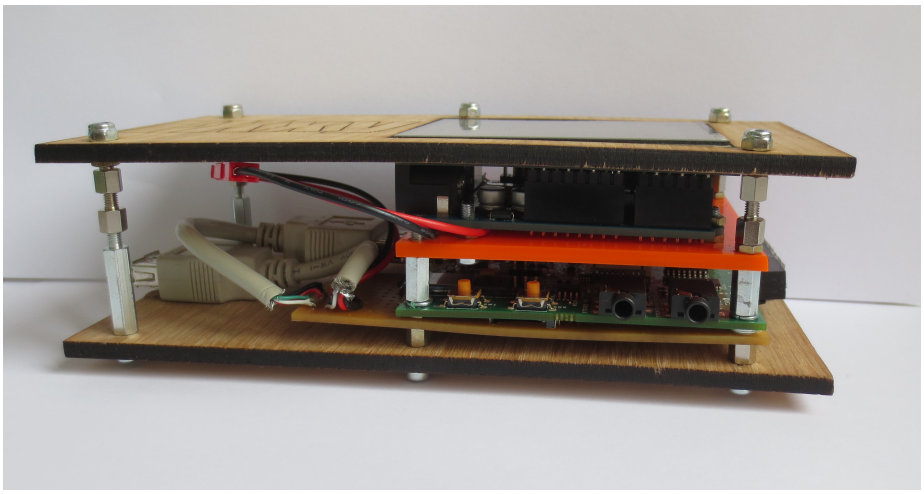
Figur 24: Mikroprocessorsenheten *Arduino Uno R3*.

4.4.1 Komponentbeskrivning

Resistiva skärmar fungerar så att två lager av rutnät med resistanser fästs ovanpå skärmen med ett litet mellanrum [40]. När en användare trycker ihop dessa lager så erhålls en viss ledningsresistans beroende på vilken resistans på bredden och höjden på skärmen som trycks ihop. Detta används sedan för att bestämma vart användaren har tryckt.

I slutprodukten används en resistiv pekskärm vid namn *2.8" TFT Touch Shield v2* av komponenttillverkaren Adafruit [41]. Skärmen har en diagonal storlek på 2,8 tum och en pixelupplösning på 240x340. Den har en inbyggd pekskärmskontroller som sköter lagring av användarinmatningar för att avlasta Arduinon. Skärmen har dessutom en ingång för ett microSD kort som kan användas för att visa bilder på skärmen. Den valdes eftersom den tillsammans med Arduinon ger en god helhetslösning för gränssnittet, med färdiga kodbibliotek för programmeringen av gränssnittet.

Kommunikationen mellan Arduinon och pekskärmen sker via SPI protokollet. Det krävs fem stycken SPI ledningar mellan *Arduino Uno* och displayen [41]. Därutöver krävs det en extra ledning var för pekskärmskontrollern och microSD kortet eftersom pekskärmens enheter delar på MOSI, MISO och SCK ledningarna. Kommunikationen sköts av färdig programvara som medföljer i produkten och har inte modifierats av projektgruppen.



Figur 25: Produktskal - Sidovy.

4.5 Produktskal

Produktskalet som visas i figur 25 (och även på framsidan) skapades för att sammanbinda de tre hårdvaruenheterna (*eZdsp5535*, *Arduino Uno* och *Adafruit 2.8" TFT Touch Shield v2*) och används för att få en fungerande musikeffektsenhet. Skalet skyddar hårdvaruenheterna, minskar behovet av kablar och ger en portabel produkt med estetiskt tilltalande design. I nedanstående delar beskrivs produktens ingående mekaniska uppbyggnad och därefter dess elektriska sammankoppling.

4.5.1 Mekanisk uppbyggnad

Som figur 25 visar har produktskalet en enkel konstruktion bestående av flera lager fästa med distansskruvar och muttrar. Topp- och bottenplattorna är gjorda i trä och har måtten enligt CAD-ritningarna i figur D.2 och D.3 (Appendix D). Mellan *eZdsp5535* och *Arduino Uno* används en platta av orangemålad akrylplast med måtten enligt CAD-ritningen i figur D.1 (Appendix D). Dessutom finns det ett experimentkort mellan bottenplattan och *eZdsp5535* som används för de elektriska sammankopplingarna. För ökad stabilitet har gummifötter fästs under bottenplattan.

4.5.2 In- och utgångar

Designen med det öppna produktskalet ger möjlighet till att utnyttja en stor del av in- och utgångarna på både *eZdsp5535* och *Arduino Uno*. För produktens syfte behövs dock bara några av dessa.

För strömsättning av produkten samt programöverföring till *eZdsp5535* används en USB-anslutning monterad på bottenplattan. En USB-kabel har fyra olika ledningar: röd +5 V, svart GND, grön Data+ och vit Data-. För att

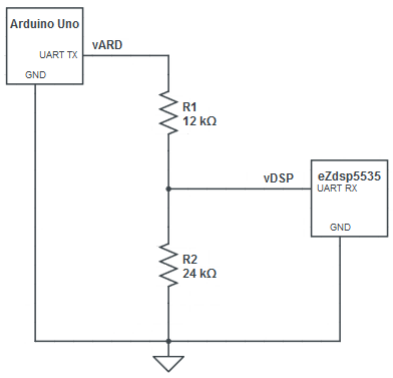
spänningssätta en enhet behövs bara den röda och den svarta ledningen. För att möjliggöra dataöverföring krävs dessutom den gröna och den vita ledningen. *Arduino Uno* och *eZdsp5535* spänningssätts parallellt men dataöverföringen går bara till *eZdsp5535*. Önskas istället programöverföring till *Arduino Uno* kan Arduinons egna USB-kontakt nås.

Program som laddats över på *Arduino Uno* sparas på det inbyggda flashminnet och laddas direkt vid spänningssättning. På *eZdsp5535* behövs däremot ett microSD kort där program kan sparas för att startas vid spänningssättning av enheten.

För ljudöverföring används stereo in respektive stereo ut från en ljudkälla till *eZdsp5535* respektive från *eZdsp5535* till en åhörare via hörlurar eller högtalare. Om behovet finns kan 6,25 mm kablar kopplas in via en adapter.

4.5.3 Spänningsdelare

För att kommunikationen mellan två mikroprocessorer ska fungera på ett korrekt sätt bör kommunikationssignalerna som enheterna sänder till varandra ligga inom varandras logiknivåer (spänningsnivåer för en logiska ettor och nollor). Detta är speciellt viktigt att beakta om data som skickas från den ena enheten har högre logiknivå än vad som rekommenderas för den andra enheten. Konsekvensen av detta är ökad strömgenomförsel och medföljande värmeutveckling vilket kan leda till skador av komponenter i enheten med lägre logiknivå. Alla enheter bör också vara inkopplade till en gemensam jord.



Figur 26: Spänningsdelningskretsen.

I produkten har ettor från mikrocontrollern *Arduino Uno* en spänningsnivå på 5 V och spänningsnivån för ettor från signalprocessorn *eZdsp5535* är 3,3 V. Att mikrokontrollerns spänningsnivå är högre än signalprocessorns innebär att det finns risk att en hög signal till mikrokontrollern misstolkas då den inte når upp till mikrokontrollerns logiknivå. Samt att en hög signal från mikrokontrollern till signalprocessorn kan skada signalprocessorns komponenter. För att undvika

en eventuell skada används en spänningsdelningskrets enligt figur 26 mellan *Arduino Uno* och *eZdsp5535*.

Spänningen in till *eZdsp5535* ges då av

$$V_{dsp} = \frac{R_2}{R_1 + R_2} V_{ard} \tag{37}$$

där $V_{ard} = 5\,\text{V}$, $R_1 = 12\,\text{k}\Omega$ och $R_2 = 24\,\text{k}\Omega$.

För att få kommunikationen mellan enheterna att fungera behöver också position 2 på switchen på *eZdsp5535* vara OFF.

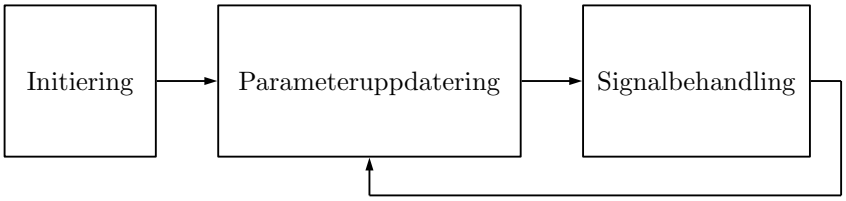
4.6 Programstruktur

I detta avsnitt presenteras programstrukturen för produkten. Först presenteras programstrukutren för det program som körs av DSP:n. Sedan redovisas strukturen för det program som körs av Arduinon och utgör gränssnittet.

4.6.1 Signalprocessorns programvara

Programvaran i produkten som används på signalprocessorn togs fram med hjälp av kodbiblioteket Chip Support Library (CSL). Kodbiblioteket innehåller datastrukturer för styr- och statusregisterna på periferienheter som bland annat sköter I^2S- och UART-kommunikation [42]. Datastrukturerna används av standardiserade funktioner eller drivrutiner för att initiera samtliga periferienheter . Dessa drivrutiner översätter med avseende på implementeringen lämpliga driftinställningar för periferienheterna. Dessutom skriver drivrutinerna erhållna värden i rätt form till rätt styrregister. För att exemplifiera en initiering av en periferienhet redogörs i nedanstående stycken i detalj hur initiering av UART-modulen sker.

En bild över programvarans flöde på signalprocessorn visas i figur 27. I stora drag kan programmet delas upp i tre delar: initiering, parameteruppdatering och signalbehandling. I initieringen allokeras minne för parametrar varefter kommunikationsprotokollen I^2C, I^2S och UART startas och konfigureras. När initieringen är färdig påbörjas en evighetsloop i programmet. I denna loop växlar programmet mellan parameteruppdatering och signalbehandling. Signalbehandlingen som sker i realtid processerar ett sampel i taget . Detta avsnitt syftar till att förklara de tre delarna av programmet i detalj.



Figur 27: Programflödet på signalprocessorn.

Initieringsfunktioner och makros Initiering av en periferienhet innebär att enheten startas upp genom att lämpliga värden skrivs till periferienhetens styrregister. När kodbiblioteket CSL används görs detta genom att i programmet fylla i fördefinierade datastrukturer som sedan ges som inparametrar till funktioner i CSL. CSL-funktionerna fyller därefter styrregister med värden från dessa datastrukturer. Dessa styrregister representeras i sin tur av datastrukturer som utgörs av variabler med samma minnesstorlek och som läggs i samma ordning som registerna är adresserade. I nedanstående kodexempel (C-kod 3) redovisas datastrukturen som redogör för alla styr- och statusregister som UART-modulen besitter. Den totala minnesarea som en periferienhets register adresseras med delas upp i variabler om 16-bitar och i vissa fall fält av 16-bitars variabler. Minnesarealer som benämns `RSVD` (**Reserved**) i olika numerationer används för att benämnda reserverade eller oskrivbara delar av minnesarean.

Med hjälp av så kallade *defines* binds dessutom flera bitvärden eller fält i dessa register till symboliska värden. Dessa defines representerar numeriska värden, ofta binära tal, med symboler på samma sätt som en variabel. Symboler används både för att representera värden i register som ger en viss konfiguration vilket beskrivs i nästa stycke. Defines används också för att skapa makron som används för att förenkla i koden återkommande instruktioner. Makron som används i produktens programvara beskrivs närmare i ett senare stycke.

```
typedef struct  {
    volatile Uint16 THR;
    volatile Uint16 RSVD0;
    volatile Uint16 IER;
    volatile Uint16 RSVD1;
    volatile Uint16 FCR;
    volatile Uint16 RSVD2;
    volatile Uint16 LCR;
    volatile Uint16 RSVD3;
    volatile Uint16 MCR;
    volatile Uint16 RSVD4;
    volatile Uint16 LSR;
    volatile Uint16 RSVD5[3];
    volatile Uint16 SCR;
    volatile Uint16 RSVD6;
    volatile Uint16 DLL;
    volatile Uint16 RSVD7;
    volatile Uint16 DLH;
    volatile Uint16 RSVD8[5];
    volatile Uint16 PWREMU_MGMT;
} CSL_UartRegs;
```

C-kod 3: UART-modulens registersdatastruktur från cslr_uart.h

CSL-funktionerna som till exempel initierar UART-modulen använder sedan `CSL_UartRegs` som en pekare för att nå de olika registerna inom strukturen. När detta görs används en konfigureringsdatastruktur som redovisas i nästa kodexempel (C-kod 4). Under initieringsförloppet i programmet skapas en egen instans av nedanstående struct-variabel. Instansen fylls sedan i med antingen förbestämda symboliska värden eller rena numeriska värden som specificerar hur UART-kommunikationen kommer att fungera. Datastrukturen ges därefter som inparameter till en funktion som kallas `UART_setup` som med hjälp av värdena i

denna struktur använder makron som fyller styrregisterna med lämpliga värden. Detta ger UART-modulen en implementationsberoende drift.

```
typedef struct
{
  /* Klockfrekvens i Hz */
    unsigned long   clkInput;
  /* Baud rate  */
    unsigned long   baud;
  /* Ordlängdsval
     Valid values - 5,6,7, or 8bits */
    unsigned short  wordLength;
  /* Antal stopbitar
     Giltiga värden - 0 or 1 */
    unsigned short  stopBits;
  /* Parititetsbitar */
    unsigned short  parity;
  /* Fifo-kontrollregister */
    unsigned short  fifoControl;
  /* Loopbackläge */
    unsigned short  loopBackEnable;
  /* AFE (Auto flow enable) */
    unsigned short  afeEnable;
  /* RTS */
    unsigned short  rtsEnable;
} CSL_UartSetup;
```

C-kod 4: Datastruktur för inparameter till `UART_setup()` från csl_uart.h

Dessa makron använder sig av de datastrukturer och fältdefinitioner som beskriver registerna som finns i CSL för att på ett systematiskt sätt hantera dataöverföring mellan registerna och CPU:n kontra att explicit ange register- och bitadresser. Till makrot ges ett par inparametrar för att utföra detta. Pekaren till datastrukturen samt registrets angivna namn används för att adressera registret. Fältdefinitioner anger vilka bitar som avses i registret och därmed vilket värde hela registervärdet ska maskas med (en AND-operation för att isolera vissa bitvärden). I fältdefinitionerna ingår dessutom hur många gånger skiftning ska utföras på ett registervärde för att få önskade värdesbitar på rätt värdepositioner. Dessa makros används bland annat för att hämta värden från statusregister för att kunna styra händelser med programvaran.

Datainsamling För att åstadkomma digitala musikeffekter behövs ljudet i analog form föras från DSP:ns ingång till kodeken för AD-omvandling. För att kodeken ska kunna utföra AD-omvandlingen måste den först initieras. Detta sker genom att först initiera I^2C-bussen med hjälp av en CSL-rutin. Ytterligare en CSL-rutin tar som inparameter önskade parametrar såsom samplingsfrekvens och förstärkning. Drivrutinen laddar sedan upp värden som motsvarar önskad funktion till kodekens styrregister via I^2C-bussen. Även I^2S-bussen initieras och via den läser och skriver DSP:n ljudsampel till och från kodek.

Signalbehandling När ett insampel lästs av från kodek ska samplet matas till en funktion som beräknar och returnerar ett nytt sampelvärde enligt en musikeffektsalgoritm. Dessa funktioner anropas i huvudloopen genom att ett element i ett fält av pekare till funktioner hämtas ur fältet med hjälp av ett

index som användaren bestämmer via mikrokontrollern. Då funktionerna som åstadkommer effekterna ska huseras i samma fält krävs det att de har samma antal inparametrar av samma typer. Varje effekt tillåts inparametrarna: två bytes för insampel, två bytes för effektparametrar samt en byte som anger operationsläge. Operationsläget bestämmer vilken specifik sort av effekten som ska köras, till exempel för state-variable filtret bestämmer detta vilken filterkaraktär filtret ska ge.

4.6.2 Parameteruppdatering

För att tillgodose parameteröverföringen från mikrokontrollern *Arduino Uno* till signalprocessorn C5535 måste deras UART-moduler initieras i början av enheternas program. För signalprocessorn beskrevs detta i detalj i förra kapitlets avsnitt om initiering. På mikrokontrollern görs detta med en initieringsfunktion från biblioteket Serial som heter start där baudhastigheten anges. Hos signalprocessorn konfigureras dess UART-modul för att ta emot data med en överföringshastighet på 9600 baud, utan stopp- eller paritetsbitar, och en ordlängd på 8 bitar det vill säga 1 byte.

Mellan processering av ljudsampel på DSP:n sker en kontroll över huruvida någon parameteruppdatering från gränssnittet har gjorts. Detta görs med hjälp av ett CSL-makro som hämtar statusbit från UART-modulens statusregister som antyder om en byte har mottagits i mottagarregistret. Om statusbiten är satt har en byte mottagits och finns i mottagarregistret. Den mottagna byten läses från mottagarregistret och sparas i ett fält där positionen i fältet som värdet sparas bestäms med hjälp av en indexvariabel. Indexvariabeln inkrementeras för varje byte som avläses.

Varje meddelande består av en sändning av fem bytes. Meddelandet påbörjas med att en byte som kallas `START_OF_MESSAGE` skickas från Arduinon.
`START_OF_MESSAGE` representeras på både Arduinon och DSP:n av heltalsvärdet 255, eller en byte som utgörs exklusivt av ettor. Ett villkor för att programmet ska fortsätta avläsningen av ett meddelande är att första byten som avläses ska motsvara detta protokollbelagda värde. Detta hjälper för att urskilja mellan meddelandena, vilket är av vikt vid användning av asynkrona kommunikationsprotokoll.

Därefter skickas en byte som kallas effektindex (`CHOSEN_EFFECT`), som på DSP:n används som index i vektorn, vilken innehåller funktionspekare till samtliga effekter. Det är denna byte i gränssnittet som används för att styra vilken sorts musikeffekt som utförs. Efter detta skickas tre bytes som utgör parametrar till effekten.

Eftersom pekskärmskontrollern matar Arduinon med användarinput i en takt som inte alltid är lämplig för parameteröverföring mellan gränssnittet och DSP:n så införs en relativt stor fördröjning mellan påbörjandet av sändning av varje meddelande.

```
if(timeN == 0 || millis() > timeN + 50)
{
    Serial.write(START_OF_MESSAGE);
    Serial.write(CHOSEN_EFFECT);
    Serial.write(p2);
    Serial.write(p1);
    Serial.write(opmode);

    timeN = millis();
}
```

C-kod 5: Kodexempel för parameteröverföring på Arduino Uno. Funktionen `millis()` anger tiden i millisekunder efter att Arduinon startats upp och lagras sedan i variabeln timeN. Funktionen write från biblioteket Serial skickar bytes bit för bit.

4.6.3 Gränssnittets programvara

Programvaran för mikrokontrollern *Arduino Uno R3* som driver pekskärmen har utvecklats med hjälp av kodbibliotek från Arduino och Adafruit. Biblioteket från Arduinon som används främst är Serial. Serial-biblioteket innehåller funktioner som möjliggör kommunikation via Arduinons UART-modul som används för kommunikation med DSP:n. Adafruits bibliotek innehåller färdigskrivna funktioner för att måla rektangulära, cirkulära och triangulära former och rita linjer på skärmen. Biblioteket innehåller även funktioner för pekskärmskontrollern [41]. Funktionerna för pekskärmskontrollern används bland annat för att starta pekskärmen och returnera koordinater för vart användaren har tryckt på skärmen.

Styrning med hjälp av gränssnittet görs på olika sätt beroende på vilken vy användaren betraktar. Hemvyn eller hemskärmen utgörs av knappar som leder till de olika effektskärmarna, se figur 28. I denna skärm presenteras produktens namn samt åtta rektangulära knappar för att välja effekt, varje knapp märks med namnet på effekten den leder till. Rektanglarnas sidors konstanta längd används tillsammans med koordinaterna för en användares tryck för att bestämma vilken knapp användaren tryckt på. Detta sker i huvudloopen genom en av flera if-satser som styr händelseförlopppet beroende på vart användaren har tryckt på skärmen.

Figur 28: Gränssnittets hemskärm

If-satserna som styr knapparnas funktionalitet kontrollerar om koordinaterna för en användarinmatning för det första ligger i höjd med knapparna. Sedan är skärmens bredd och höjd uppdelad i intervall, som utgörs av multipler av rektanglarnas sidlängd. Beroende på i vilket intervall på bredden och höjd som koordinaten ligger bestäms det vilken knapp som har tryckts. En variabel lagrar vilken vy som visas för användaren i nuläget. Vilken knapp som har tryckts på samt vilken vy som visas för användaren bestämmer vilken vy som ska ritas upp.

Pekare till de funktioner som ritar upp gränssnitt anpassade för de enskilda effekterna lagras i ett fält. När det har bestämts vilken vy som ska ritas upp lagras ett värde som motsvarar den plats i fältet där en funktionspekare för den valda vyn är sparad. Denna pekare används i sin tur för att köra funktionen som ritar den valda vyn.

Funktionerna som ritar upp de grafiska gränssnitten för de olika effekterna använder sig i sin tur av en enda funktion (bortsett från filter och EQ som har egna gränssnitt) som utnyttjar användargränssnittets standardiserade utseende för att rita upp effektvyer. Den gör detta genom att ta som inparametrar en specifikation som anger hur gränssnittet ska se ut. Denna specifikation visas i koden nedan.

```
1 void drawUI(char* rectText[], int rectNr, bool axis, float
2 textSize, char* axisText[])
```

C-kod 6: Funktion för att rita grafiskt gränssnitt för effekterna eko, reverb, tremolo och distortion. `rectText[]` är ett fält som anger textsträngarna med teckenstorleken, `textsize` som ska skrivas inom `rectNr` stycken kvadrater. `axis` anger om interfacet ska rita upp koordinataxlar där `axisText[]` är ett fält som anger texten på koordinataxlarna.

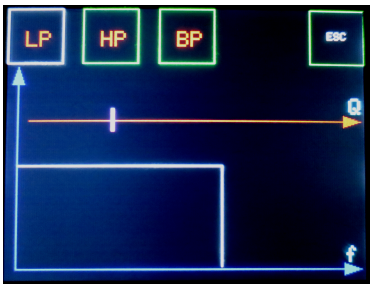
I området på skärmen under knapparna sker parameterstyrning efter det att en effekt har valts. Ytterligare en if-sats i huvudloopen kontrollerar om en användare har tryckt inom detta område. I sådana fall anropas, beroende på vilken vy som visas i nuläget, en funktion som tar som inparametrar koordinater för användarens inmatning och använder dem för att bestämma nya parametervärden. Dessa återkopplas grafiskt till användaren samt skickas med hjälp av UART-modulen till signalprocessorn. Styrningen och presentationen av parametrarna sker på olika sätt för dem olika effekterna.

4.7 Gränssnittsdesign

I denna del beskrivs gränssnittets utseende och funktionalitet. Totalt utgörs gränssnittet av fyra olika sorters vyer eller skärmar. Hemvyns utseende kan ses i figur 28 ovan, medan resterande vyer hittas nedan.

4.7.1 State-variable filter

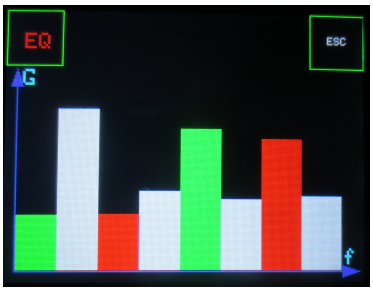
För state-variablefiltrets gränssnitt, se figur 29, ritas ett kordinatsystem upp längs skärmens botten och vänstra sida som visar filtrets ideala frekvenssvar (se avsnitt 2.5.1 Grundläggande frekvenssvar) med filtrets gränsfrekvens på den horisontella axeln. I mitten av skärmen finns ett horisontal reglage för filtrets Q-värde. Tre knappar ritas längst upp i vänstra hörnet som gör att användaren kan välja mellan lågpass-, högpass- och bandpassfiltrering. Längst upp till höger finns det dessutom en knapp för att återgå till hemvyn. Användaren kan ändra både brytfrekvensen och Q-värdet genom att trycka inom området för parameterstyrning där det önskade värdet ligger



Figur 29: Gränssnittet för state-variable filtret här inställt som lågpassfilter.

4.7.2 Equalizer

Hur en signals frekvensinnehåll påverkas av equalizern bestäms av användaren genom att ändra höjden på åtta staplar på pekskärmen, se figur 30. Dessa åtta staplars höjd sparas i ett fält och anger ett läge kring ett mittläge som representerar bandens förstärkning och dämpning. Kvoten mellan var längs frekvensaxeln användaren har tryckt och en stapelbredd används för att bestämma inom vilket band användaren har tryckt och vilket element i fältet som ska ändras. Detta element används sedan för att rita angiven höjd på stapeln.



Figur 30: Gränssnittet för equalizer.

4.7.3 Övriga effekter

De övriga effekternas (vibrato, flanger, tremolo, eko, reverb och distortion) gränssnitt utgörs av ett koordinatsystem under knappen för hemskärmen med två vinkelräta axlar, se figur 31. En punkt i koordinatsystemet ritas där användaren trycker och anger genom sin utbredning längs axlarna storleken för två parametrar. Dessa parametrar skickas till DSP:n via UART-modulen på Arduinon enligt procedur beskrivet i avsnitt 4.6.2.

Figur 31: Gränssnittet för eko, reverb, distorsion och tremolo.

5 Diskussion

Syftet med arbetet var att utveckla en musikeffektsenhet avsedd för realtidsanvändning. I början av projektet gjordes därför en kravspecifikation över önskade egenskaper för denna produkt, kravspecifikationen hittas i Appendix A. I diskussionen utvärderas den färdiga produkten och de resultat som presenterats i kapitel 3 och 4 mot denna kravspecifikation. Diskussionen innehåller även ett resonemang kring svårigheter i att uppnå dessa mål samt möjligheter för vidareutveckling.

Musikeffekterna Vissa effekter var svårare än andra att implementera på DSP:n. Detta kunde bero på effektens komplexitet och begränsningar i hårdvara. I de flesta fall visade sig problemen i form av hackande ljud av olika karaktär eller brus. I kravspecifikationen finns ett krav om vilka effekter som ska implementeras till produkten. Detta krav uppfylldes då alla effekter implementerades framgångsrikt på DSP:n och styrs framgångsrikt i slutprodukten.

Vibrato-effekten är den musikeffekt vi enskilt har lagt mest tid på att felsöka. Det har visat sig att problemet inte låg i vibratot utan snarare i den underliggande programstrukturen. Detta har påvisats då effekten fungerade utan problem med en ny programstruktur. Då ett fungerande vibrato implementerats var det enkelt att även implementera flanger.

I och med att en LFO implementerades för vibrato-effekten var det enkelt att realisera en tremolo-effekt. Även om tremolo i dess struktur är en simpel musikeffekt anser vi att den ger en påtaglig och högkvalitativ effekt. Möjlig förbättring för tremolot vore att erbjuda flera olika vågformer för LFO:n. Detta skulle visserligen innebära att mer minne används för att spara vågtabeller.

Även om state-variable filtret fungerar felfritt är det i sin struktur endast ett filter med brantheten 12 dB/oktav, vilket uppfattas av oss som för lågt för denna typ av tillämpning. Filtrets struktur är däremot flexibelt på så sätt att det samtidigt filtrerar enligt flera olika filterkaraktärer (lågpass, högpass, bandpass) och användaren kan välja vilken karaktär som önskas. Detta har underlättat implementeringen med avseende på kommunikationen. En möjlighet till vidareutveckling hade varit att seriekoppla två stycken state-variable filter för att erhålla 24 dB/oktav branthet.

Distortion-effekten var den effekten som tog minst tid att implementera, då den är väldigt simpel i sin uppbyggnad. Vilken typ av distortion som önskas beror på tillämpning och preferens. En enkel förbättring av effekten vore att ge användaren en möjlighet att välja mellan hård och mjuk klippning, då dessa ger olika karaktär på ljudet.

Då en filterbank med god prestanda antogs vara beräkningsmässigt krävande för DSP:n, implementerades den grafiska equalizern istället med hjälp av frequency sampling. Det potentiella problemet att filterbanden i en filtebank överlappar varandra om de har för lågt ordningstal undgås helt med frequency sampling. Frequency sampling kan alltså prestera ett mycket flackt frekvenssvar med en förhållandevis låg filterordning jämfört med om samma frekvenssvar skulle implementerats med en filterbank. Frequency sampling tappar dock i prestanda i

Tabell 3: Beräkningsbelastning per sampel för de olika musikeffekterna

Musikeffekt	*Klockcykler per sampel*	*Procent av DSP:ns beräkningskraft*
Grafisk equalizer	600	29
Reverb	260	12
Flanger	190	9,1
Vibrato	180	8,6
Eko	150	7,2
Tremolo	120	5,8
Chamberlin state-variable filter	120	5,8
Distortion	110	5,3

jämförelse med en filterbank om skillnaden i amplitud mellan närliggande band är stor. Den lägre filterordningen ger nämligen långsammare övergångar mellan stopp- och passband, speciellt för låga frekvenser.

Prestanda Då all ljudbehanling sker sampel per sampel är borde fördröjningen i produkten endast vara $1/fs \approx 21\,\mu$s då samplingsfrekvensen är 48 kHz. Detta uppfyller väl kravspecifikationen på maximal fördröjning om 20 ms. På $21\,\mu$s hinner signalprocessorn på 100 MHz utföra 2083 klockcykler. I tabell 3 ser vi sammanställt hur många klockcykler de olika musikeffekterna behöver för att processa en ljudsampel och vad det motsvarar i procent av processorn totala kapacitet på 2083 klockcykler per sampel.

Equalizern är den effekt som är beräkningsmässigt klart mest krävande då den behöver ca 600 klockcykler per sampel. Equalizern är dock ineffektivt implementerad då den filtrerar i tidsdomänen. Den hade kunnat effektiviseras betydligt om FIR-filtreringen flyttats till frekvensdomänen. En annan förbättring hade varit att utnyttja de båda MAC-enheterna parallellt istället för som nu bara använda en av dem. Trots detta så har equlizern ändå betydlig marginal till vad processorn beräkningsmässigt borde klara av per sampel.

Analys har gjorts av hur mycket av processorns kraft hanteringen av parameteröverföringen kräver, vilket givetvis påverkar hur mycket som blir över till musikeffekterna. Slutsatsen av denna analys är att parameteröverföringen tar upp förhållandevis väldigt få klockcykler.

Prestandaanalys har endast gjort genom att räkna klockcykler för olika delar av programmet. Prestandaanalysen hade kunnat vidareutvecklas genom att även titta på aspekter som minnesanvändning och databussbelastning.

Användarvänlighet Användarvänlighet var ett av de stora kraven för produkten. Pekskärmsimplementationen ger ett användarvänligt gränssnitt som uppfyller de uppsatta kraven. Med pekskärmen är det enkelt att styra musikeffekterna med en hand, utan behov av övriga reglage. Skärmen visualiserar musikeffektsstyrningen på ett enkelt och intiutivt sätt. Ett problem vi observerat med användargränssnittet är dock att det i vissa fall kan bli synliga fördröjningar när olika delar ritas upp.

Produktskal Mycket lite tid har lagts på att skapa ett produktskal. Trots detta uppfyller det färdiga skalet målen i kravspecifikationen, till exempel vikt, mått och nätanslutning. Det är däremot svårt för oss att garantera en livslängd för produkten.

Svårigheter med utvecklingen De största svårigheterna i detta arbete har legat i programmering av DSP:n då detta var något helt nytt för hela gruppen och något vi har fått lära oss från grunden. Speciellt svårt har det varit på grund av att information kring programmering av signalprocessorn har varit knapp. Det fanns exempelvis inga exempel där UART-modulen och DMA-enheten initieras samtidigt och används parallellt.

Just implementeringen av flera periferienheter (såsom UART och DMA) i drift samtidigt för DSP:n har varit problematiskt och tagit mycket tid i projektet. Problemet löstets slutligen genom att helt frångå att använda DMA och låta DSP hämta och skriva varje sampel direkt från/till kodek.

Problemet försvåras av att signalprocesseringen i projektet sker i realtid, vilket betyder att timing är av högsta vikt. Det finns risk att processeringen blir osynkroniserad, vilket i sin tur kommer att ge upphov till ett hackande eller förvrängt ljud.

Kodbiblioteken som användes i programvaran för DSP:n och speciellt Arduinon för gränssnittet medförde med sina färdiga funktioner en högre abstraktionsnivå på vissa delar av resulterande kod. Detta gör att programvaran blir mer översiktlig, men samtidigt förlorar den en detaljnivå som gör att det ibland blir svårt att förstå och kontrollera vad programmet utför och undvika konflikter. Ett exempel på detta är CSL:s funktion som initierar UART-modulen. Den satte även I^2S-bussen i reset-läge och detta ledde till stopp i ljudflödet då I^2S-bussen användes för att hämta ljudsampels från kodex till DSP:n

Vidareutveckling Det finns goda möjligheter till vidareutveckling av många av produktens delar tack vare det öppna produktskalet. Nya musikeffekter kan enkelt läggas till så länge som längde de använder samma antal parametrar som de redan implementerade effekterna. Nya effekter kräver också nyttjandet utav de standardiserade interfacen eller att nya typer av interface skapas. Om så önskas kan en ny och mer passande interfacestruktur skapas då pekskärmen erbjuder stor flexiblitet i utformning av användargränssnitt. För effekter såsom vibrato, eko och reverb skulle mer utveckling och felsökning behövas för att lyckas med att implementera dem på det önskade sätten som beskrivs i kapitel 3.

Det som begränsar vidareutvecklingen är framför allt hårdvaran. Arbetsminnet för både *Ardunio Uno* och *eZdsp5535* är självklart begränsat och sätter en gräns för hur många ljudeffekter som kan sparas. Detta kan lösas med en mer optimerad minneshantering och eller användning av periferienheter som microSD-kort för lagring. Sådan vidareutvekling kräver dock större förståelse för hårdvaruarkitektur av vad som funnits i projektgruppen. Nya effekter kan också vara begränsade av processeringskraften hos DSP:n. Det finns dock möjlighet att

öka effekternas prestanda betydligt genom att programmera i assembler i stället för C.

6 Slutsats

En fungerande musikeffektsenhet har tagits fram i projektet. Enheten erbjuder åtta olika musikeffekter i realtid: filter, equalizer, vibrato, flanger, tremolo, eko, reverb och distortion. Musikeffekterna har implementerats i C-kod för att köras på DSP:n *TMS320C5535* som sitter på utvecklingskortet *eZdsp5535*.

Styrning av enheten har implementerats med hjälp av en pekskärm Adafruit *2,8" TFT Touch Shield v2* som är en utbyggnadsenhet för en Arduino. Via pekskärmen ges indata till Arduinon som skickar information till DSP:n. Till de tre enheterna (DSP, skärm och Arduino) har ett produktskal byggts för att sammanbinda dem till en enhetlig produkt.

Den färdiga produkten uppfyller alla utom ett av de kraven som specificerades i början av projektet. Det krav som inte uppfylls är att vi inte kan garantera en viss livslängd för produkten.

Projektets mest utmanande del har varit att programmera en DSP eftersom detta var något helt nytt för gruppen.

Referenser

[1] B. Mulgrew *et al.*, *Digital Signal Processing Concepts and Applications*, 2a uppl. Hampshire, United Kingdom: Palgrave Macmillan, 2003 .

[2] R. Brice, *Music Engineering*, 2:a uppl. Oxford, United Kingdom: Newnes, 2001

[3] M. Wölfel och J. McDonough, "Acoustics" i *Distant Speech Recognition*, 1:a uppl. Hoboken: Wiley, 2009, kap. 2, del 3, ss.42-43.

[4] P. Guillaume, "Sounds" i *Music and Acoustics: From Instrument to Computer*, 1:a uppl. London: ISTE Ltd, 2006, kap. 1, del 2.

[5] S.R. Alten, "Behavior of Sound" i *Working with Audio*, 1:a uppl. Boston: Course Technology, 2012, kap. 1.

[6] A.V Oppenheim et al., "Sampling" i *Signals & Systems*, 2:a uppl. Harlow, UK: Prentice Hall, 1997

[7] S.W. Smith, "Audio Processing" i *The scientist and engineer's guide to digital signal processing*, 1:a uppl. San Diego: California Technical Publ., 1997, kap. 22, ss. 351-372.

[8] S.W. Smith, "ADC and DAC" i *The scientist and engineer's guide to digital signal processing*, 1:a uppl. San Diego: California Technical Publ., 1997, kap. 3., ss. 35-66.

[9] F.J. Taylor, *Digital Filters: Principles and Applications with MATLAB*, 1:a uppl. Hoboken: John Wiley & Sons, 2012

[10] S.W. Smith, "Introduction to Digital Filters" i *The scientist and engineer's guide to digital signal processing*, 1:a uppl. San Diego: California Technical Publ., 1997, kap. 14, ss. 261-276.

[11] S.W. Smith, "Linear Systems" i *The scientist and engineer's guide to digital signal processing*, 1:a uppl. San Diego: California Technical Publ., 1997, kap. 5, ss. 87-106.

[12] L. Thede, "Finite Impulse Response Digital Filter Design" i *Practical Analog and Digital Filter Design*, Norwood, MA: Artech House, 2004, kap. 7, ss. 161-185.

[13] C.S. Burrus. (2009, Nov 2) *FIR Filter Design by Frequency Sampling or Interpolation* [Online]. Tillgänglig: `http://cnx.org/content/m16891`

[14] L. Thede, "Infinite Impluse Response Digital Filter Design" i *Practical Analog and Digital Filter Design*, Norwood, MA: Artech House, 2004,kap. 6, ss. 141-160.

[15] S.W. Smith, "Digital Signal Processors" i *The scientist and engineer's guide to digital signal processing*, 1:a uppl. San Diego: California Technical Publ., 1997, kap. 28, ss. 503-534.

[16] *Standard for Floating-Point Arithmetic* IEEE Standard 754, 2008

[17] R. Yates, (2013, Jan 2), *Fixed-Point Arithmetic: An Introduction* [Online]. Tillgängligt: `http://www.digitalsignallabs.com/fp.pdf`.

[18] S.T Karris, "Analog and Digital Filters" i *Signals and Systems with MATLAB Computing and Simulink Modeling*, 4:e uppl. Orchard Publications, 2008

[19] N. Kehtarnavaz, *Real-Time Digital Signal Processing: Based on the TMS320C6000*, 1:a uppl. Oxford, UK: Elsevier Inc., 2005

[20] T.I. Laakso et al., "Splitting the Unit Delay" i *IEEE Signal Processing Mag.*, vol. 13, ss. 30-60, Jan., 1996.

[21] E. Meijering, Ä chronology of interpolation:from ancienct astronomy to moderen signal and image processing"i *Proceedings of the IEEE*, vol. 90, ss. 319-342, Mar., 2002.

[22] H. Chamberlin, *Musical Applications of Microprocessors*, 2:a uppl. New Jersey: Hayden Books, 1985

[23] N. Amir *et al.*, "Automated evaluation of singers' vibrato through time and frequency analysis of the pitch contour using the DSK6713", i *16th International Conference Digital Signal Processing*, Santorini-Hellas, 2009

[24] J.O. Smith, "Time Varying Delay-Effects" i *Physical Audio Signal Processing: For Virtual Musical Instruments and Audio Effects*, 2:a uppl. W3K Publishing, 2010, Kap. 5.

[25] U. Zölzer, Delay Based Audio Effects"i "*DAFX: Digital Audio Effects*", 2:a uppl. Chichester, UK: John Wiley & Sons, 2011, Kap. 2, Del 6.

[26] D. Marshall & K.Sidorov, "CM3106 Chapter 7:Digital Audio Effects", Cardiff University, Cardiff

[27] R. Izhaki, *Mixing Audio: Concepts, Practices and Tools*, 1:a uppl. Oxford, UK: Focal Press, 2008, ss. 406-407.

[28] D.J. Dailey, "Tremolo" i *Electronics for Guitarists*, 2:a uppl., New York:Springer, 2013, kap. 5, ss. 230.

[29] W. Haas, (2007, Aug), *Tape Delay In Your DAW* [Online]. Tillgängligt: `http://www.soundonsound.com/sos/aug07/articles/tapeecho.htm`.

[30] M. Schroeder, *Colorless artificial reverberation*, New Jersey: Bell Telephone Laboratories, 1961.

[31] P. White, (2006, Mar), *Choosing The Right Reverb* [Online]. Tillgängligt: `http://www.soundonsound.com/sos/mar06/articles/usingreverb.htm`.

[32] M. Ross *Getting Great Guitar Sounds*, 2:a uppl., Hal Leonard, 1998, s. 39.

[33] P.P. Chu, "Audio Codec Controller" i *Embedded SoPC Design with NIOS II Processor and VHDL Examples*, 1:a uppl. Hoboken: John Wiley & Sons, 2011, kap. 18, del 1, ss. 511-512.

[34] *Audio Converter – Audio CODEC – TLV320AIC3204 – TI.com* [Online]. Tillgängligt: `http://www.ti.com/product/tlv320aic3204`

[35] *TLV320AIC3204 Application Reference Guide*, TI, Dallas, TX, 2012, ss. 2-3, 11, 16, 67, 78.

[36] R. Sikora. (2010) *C500 Teaching ROM – Educators – Wiki – Educators – TI E2E Community* [Download]. Tillgängligt: `http://e2e.ti.com/group/universityprogram/educators/w/wiki/2040.c5000-teaching-rom.aspx`

[37] ATmega328 [Online]. Tillgängligt: `http://www.atmel.com/devices/atmega328.aspx`.

[38] Arduino – ArduinoBoardUno [Online]. Tillgängligt: `http://arduino.cc/en/Main/arduinoBoardUno`.

[39] Arduino Playground – Eclipse [Online]. Tillgängligt: `http://playground.arduino.cc/Code/Eclipse`.

[40] Cirque Corp. (2011, Sep 15). *Resistive Touch (Touch Sensing Technologies - Part 3)* [Online]. Tillgänglig: `http://www.touchadvance.com/search/label/Resistive`.

[41] *Adafruit 2.8" TFT Touch Shield v2*, Adafruit Corp., New York, NY, 2014.

[42] *TMS320C55x Chip Support Library API Reference Guide*, Texas Instruments Inc., Dallas, TX, ss. 3.

[43] R. Scott. (1995, May). *Interrupts may seem basic, but many programmers avoid them*, [Online]. Tillgänglig: `http://www.sltf.com/articles/pein/pein9505.htm`.

[44] *C55x v3.x CPU - Reference Guide*, Texas Instruments Inc., Dallas, TX, ss. 2-25.

[45] D.A. Patterson och J.L. Hennesy, "Exceptions"i *Computer Organization and Design*, 3:e uppl., Burlington: Morgan Kaufmann Publishers, 2006/2007, kap. 5, del 6, ss. 340.

[46] D.A. Patterson och J.L. Hennesy, Storage, Networks, and Other Peripherals"i *Computer Organization and Design: The Hardware/Software Interface*, 3:e uppl., Burlington: Morgan Kaufmann Publishers, 2006/2007, kap. 8 del 5, ss. 591-593.

[47] *UM10204* I^2C*-bus specification and user manual*, v.5, NXP, 2012, ss. 1-10.

[48] I^2S *bus specification*, 1:a uppl. (revised), Philips Semiconductors, 1996.

[49] *Inter-IC Sound Bus (I2S)*, 2:a uppl., Cypress Semiconductor Corporation, 2010.

[50] gbmhunter. (2011, Sep 12). *UART | Clandenstine Laboratories* [Online]. Tillgänglig: Tillgänglig: `http://cladlab.com/electronics/circuit-design/communication-protocols/uart-protocol`

[51] *KeyStone Architecture Universal Asynchronous Receiver/Transmitter (UART) Users Guide*, 1:a uppl., Texas Instruments, 2010, ss. 14.

[52] *Serial Communications*, Silicon Labs

[53] J. Catsoulis, “Adding Peripherals Using SPI” i *Designing Embedded Hardware*, 2:a uppl. O’Reilly: Sebastopol, 2005, kap. 7, ss. 160-161.

[54] R. Toulson och T. Wilmshurst, “Starting with Serial Communication” i *Fast and Effective Embedded Systems Design: Applying the ARM mbed*, 1:a uppl. Elsevier Ltd: Waltham, 2012, kap. 7, del 2.

[55] *Using the Serial Peripheral Interface to Communicate Between Multiple Microcomputers*, 1:a rev., freescale semiconductor, 2002, ss.2.

[56] *SPI Interface in Embedded Systems* [Online]. Tillgängligt: `http://www.eeherald.com/section/design-guide/esmod12.html`.

[57] M. Abd-El-Barr och H. El-Rewini, ”Input-Output Design and Organization”, i *Fundamentals of Computer Organization and Architecture*, 1:a uppl. Hoboken: Wiley, 2005, kap. 8, del 4, ss. 175-177.

A Kravspecifikation

Tabell 4: *Krav för produkten*

Nr.	Beskrivning
1	Minst 5 musikeffekter kända för marknaden
1.1	*Equalizer*
1.2	*Filter (HP,BP,LP)*
1.3	*Reverb*
1.4	*Vibrato*
1.5	*Distortion*
2	Portabel
2.1	*Maximala mått 15x20x20 cm (höjd, bredd, djup)*
2.2	*Maximal vikt 1 kg exklusive ev. batterier*
3	Användarvänlighet
3.1	*Musikeffekterna ska kunna styras med en hand*
3.2	*Beskrivande text till olika inputs*
3.3	*Samtliga reglage på produktens ovansida*
4	Maximal fördröjning mellan in- och utsignal på 20 ms
5	Inte innehålla fläktar
6	Kunna garantera en viss livslängd
7	Ha nätanslutning
8	6,33 mm Tele in- och utgång
9	Insignalen skall vara normaliserad

Tabell 5: *Önskemål för produkten med viktning där 1 är låg prioritet och 5 är hög.*

Nr.	Beskrivning	Vikt
1	Portabel	-
1.1	*Batteridriven*	3
1.2	*Batteridriven och batteritiden ska vara minst 3 timmar*	2
1.3	*Maximala mått 10x15x15 cm*	3
1.4	*Maximal vikt 0,5 kg exklusive batterier*	3
1.5	*Produkten bör ha låg energiförbrukning*	3
2	Användarvänlighet	-
2.1	*Grafisk feedback i form av grafisk visualisering av ljudet*	3
2.2	*Inga menyer, alla funktioner nås direkt med ett knapptryck*	5
3	Sampler	-
3.1	*Inspelningsmöjligheter*	4
3.2	*Möjlighet att ladda över ljudfiler via USB*	3
3.3	*Möjlighet att sequenca dessa samplingar*	4
4	Estetiskt tilltalande	-
4.1	Fysisk utformning	4
4.2	Virtuell utformning	4
5	Minst 10 musikeffekter kända för marknaden	4
6	Återuppladdningsbart batteri	2
7	Möjlighet till inkoppling av externt klaviatur	1
8	Möjlighet att styra eller synkronisera extern utrustning via MIDI	3
9	Phono stereo in- och utgång	3
10	Kunna använda två effekter samtidigt (serie eller parallellt)	3
11	Ha gummifötter	3

B Dataprotokoll

B.1 Avbrott

Istället för att en processor i ett datorsystem skall behöva kontrollera att en händelse har skett, till exempel genom att ständigt jämföra två värden i olika register så kan en mekanism som kallas *avbrott* eller *interrupt* användas [43]. Avbrott fungerar genom att en periferienhet skickar en signal till processorn som antyder att en händelse behöver processorns uppmärksamhet. Därefter sätts en bit i Interrupt Flag Registrer (IFR) som motsvarar en viss typ av händelse [44]. Processorn sparar då alla sina registervärden, därmed också programräknarregistrets värde vars funktion är att visa vilken adress i programminnet processorn höll på att exekvera ifrån. Därefter laddar processorn in en avbrottsvektor ifrån en förbestämd plats i minnet in i programräknaren så att processorn börjar exekvera en så kallad avbrottshanteringsrutin. Avbrottsvektorn är alltså en minnesadress där adressen till avbrottshanteringsrutinen lagras.

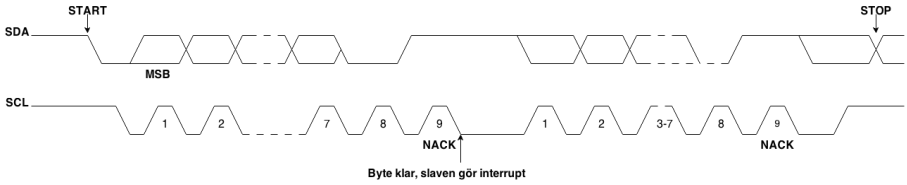
Det finns två huvudtyper av avbrott: hårdvaruavbrott och mjukvaruavbrott (även kallat undantag eller *exception*) [45]. Ett hårdvaruavbrott sker när processorn mottar en extern signal som initierar avbrottet, medan ett mjukvaruavbrott initieras genom exceptionella händelser i exekveringen av kod. Ett exempel på ett mjukvaruavbrott är i en funktion som till exempel utför division genom att ta täljare och nämnare som inparameter och returnerar kvoten. Om nämnaren då är lika med 0 går divisionen ej att utföra och funktionen kan utlösa ett mjukvaruavbrott som avbryter funktionen och istället returnerar ett felmeddelande.

Avbrott kan i också delas upp i ytterligare två grupper, maskbara och omaskbara avbrott [46]. Maskbara avbrott är avbrott som går att avaktivera vid önskemål genom att ändra ett av värdena i ett register som oftast kallas Interrupt Mask Register (IMR) eller Interrupt Enable Register (IER), medan omaskbara avbrott är oftast av sådan hög prioritet att de inte går att ignorera. Mjukvaruavbrott är av typen omaskbara avbrott.

B.2 Inter-Integrated Circuit (I^2C)

Inter-Integrated Curcuit (I^2C) är ett seriellt och synkront datorkommunikationsprotokoll utvecklat av Philips Semiconductor [47]. Kommunikationen sker genom två ledare (busstrådar); en för datasignalen (Serial Data Line, SDA) och en för klocksignalen (Serial Clock Line, SCL). Alla inkopplade enheter på bussen använder sig av unika adresser och kan fungera både som master och slav beroende på sammanhanget (funktionaliteten ändras efter behov). För att konflikter mellan olika masters på bussen ska undvikas använder protokollet sig utav kollisonsdetektering och medling (*arbitration*) som bestämmer vilken master som ska få kommunicera under en viss cykel. Den främsta begräsningen för antalet kommunicerande enheter på en I^2C buss är busskapacitansen (ofta är den begränsad till 400 pF).

Figur B.1 visar hur en typisk ram för I^2C ser ut. Kommunikationen startar när startvillkoret är uppfyllt, det vill säga då SDA ändrar sitt värde från högt till lågt samtidigt som SCL är hög [47]. Slutet av ramen signaleras av stopvillkoret vilket



Figur B.1: Kommunikation via I2C.

innebär att SDA ändrar sitt värde från lågt till högt samtidigt som SCL är hög. Däremellan kan ett obegränsat antal bytes (8-bitarsblock) skickas. Efter varje byte som tagits emot skickar mottagaren en bekräftelsebit (acknowledgement, ACK). Om sändningen däremot har misslyckats fås en negativ-bekräftelsebit (negative acknowledgement, NACK), varefter den aktiva mastern kan avbryta eller starta en ny sändning. ACK respektive NACK signaleras av att SDA är låg respektive hög när den nionde klockpulsen skickas. Slaven kan också tvinga fram mastern i ett vänteläge (genom att göra SCL låg) om den måste utföra någon uppgift, till exempel ett avbrott, innan den kan ta emot eller skicka en ny byte. SCL kommer att släppas först när slaven är redo att återgå till kommunikationen.

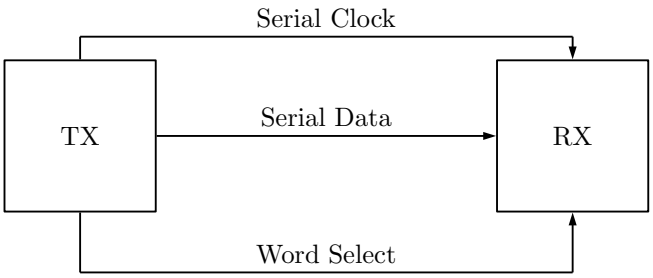
B.3 Integrated Interchip Sound (I^2S)

Inter-Interchip Sound (I^2S) är ett kommunikationsprotokoll utvecklat av Philips Semiconductors, som utgör en standard vid överföring av digitalt ljud [48]. I^2S använder seriell kommunikation och överför enbart ljud.

Minst tre ledningar krävs mellan två enheter som kommunicerar med varandra via I^2S [48]. Dessa ledningar kallas för serial data (SD), word select (WS) och continuous serial clock (SCK). SD används för att överföra ljud som seriell data. Genom tidsmulitplexing kan samma ledning användas för att överföra ljudet för både höger och vänster ljudkanal. WS avgör vilken av kanalernas ljud som överförs. Om WS är hög är det höger kanal som skickar och om den är låg så är det vänster. SCK är en gemensam kontinuerlig seriell klocka för sändare och mottagare. Eftersom sändare och mottagare delar klocka är I^2S ett synkront kommunikationsprotokoll. I^2S är ett master/slav protokoll där den enhet som genererar den gemensamma klockan är master. En bild över hur en uppkoppling mellan sändare och mottagare kan se ut hittas i figur B.2.

För flexibilitet skickas den mest signifikanta biten först i I^2S [48]. Av detta följer att mottagare inte behöver känna till hur många bitar som sändaren har tänkt att skicka. Inte heller behöver sändaren veta hur många bitar som mottagaren klarar av att ta emot. Om mottagaren får fler bitar än den egentligen klarar av kommer den att ignorera alla bitar som kommer efter den minst signifikanta biten. Om mottagaren istället får färre bitar än vad den normalt behandlar, sätts resterande bitar till 0. Sändaren i I^2S skickar alltid första biten i nästa ord en klockcykel efter att word select (WS) har uppdaterats.

Vilken klockfrekvens som SCK bör ha kan beräknas ur ljudets samplingsfrekvens och den ord längd som I^2S busen använder sig av, enligt ekvation 38 [49].



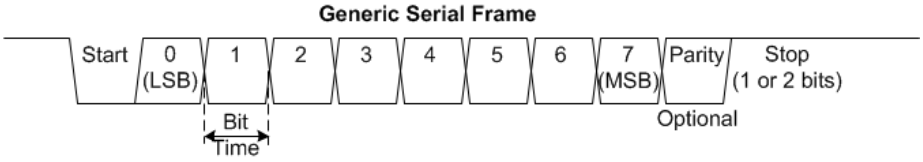
Figur B.2: Två enheter som kommunicerar med I^2S, här är sändaren master.

$$\text{Seriell klockfrekvekens} = \text{Samplingsfrekvens} * \text{Ordlängd} * 2 \tag{38}$$

Det vill säga för varje ljudsampel ska I^{2}S hinna överföra två ord (höger och vänster sampel) av önskad ordlängd.

B.4 Universal Asynchronous Receiver/Transmitter (UART)

Universal Asynchronouse Receiver/Transmitter eller UART är ett protokoll för digital kommunikation [50]. Protokollet är universellt i det avseende att parametrar så som spänning, timing, felupptäckning och flödeskontroll är inställningsbart. Det är asynkront i det avseende att sändaren (TX) bara sänder då den har något att meddela. Att protokollet är asynkront innebär också att sändaren och mottagaren (RX) inte har någon gemensam klocka. På grund av detta måste start- och stoppbitar användas för att indikera början och slut av en skickad ram, så att mottagaren ska kunna upptäcka denna. UART överför information seriellt, det vill säga en bit i taget över ett medium (en ledning). Figur B.3 visar hur en typisk seriell ram ser ut. Bitarna 0-7 är nyttobitar, alltså de bitar som faktiskt ska överföras, möjlighet till felupptäckning i form av paritetskontroll är valbar medan start- och stoppbitar är ett krav.



Figur B.3: Utseendet för en typisk UART-ram. Startbit i början, paritetsbit och stopbit i slutet, däremellan meddelandet.

I och med att endast en bit överförs i taget räcker det att representera en bit med en spänningsnivå, för UART gäller det att hög spänning motsvarar en etta och låg spänning motsvarar en nolla. Enligt dagens standarder är UART ett långsamt överföringsprotokoll. För att kunna överföra data i olika takt med UART krävs en möjlighet till att variera *baudklockan*, som avgör hur ofta symboler skickas, vilket är dess *baudhastighet* [51]. Baudklockan är ett antal gånger snabbare

än baudhastigheten, det vill säga att varje skickad eller mottagen symbol varar i ett visst antal *baud-klockcykler*, typiskt åtta. Signalen samplas efter halva sin varaktighet, typiskt fyra baud-klockcykler. Baudklockan åstadkoms genom att man från enheten får en input klocka, oftast i form av enhetens processorklocka. Sedan delas denna klockfrekvens, så att önskvärd överföringstakt fås fram. Delaren ges enligt ekvation 39 [51].

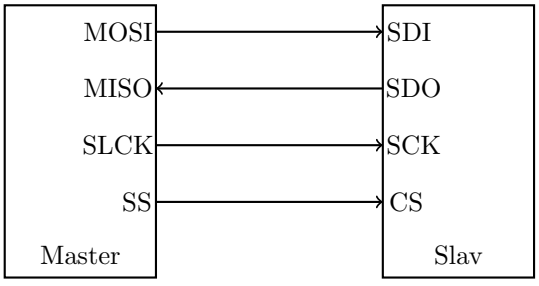
$$\text{Delare} = \frac{\text{UART input klockfrekvens}}{\text{Avsedd baudhastighet*8}} \tag{39}$$

För två enheter som kommunicerar med varandra via UART sker denna klockgenerering separat för varje enhet. För att tillförlitlig informationsöverföring ska uppnås är det därför viktigt att baudklockorna som genereras är noggranna. En felaktig klocka kan medföra att en symbol samplas vid fel tidpunkt vilket medför att överförd data kan blir korrumperad [52].

Vanliga överföringshastigheter som UART stödjer är till exempel 4800, 9600, 14400, 56000, 115200, 256000 Baud [Symboler/Sekund]. I fallet UART är en baud ekvivalent med en bit per sekund, detta på grund av att varje symbol motsvarar endast en bit [50]. Generellt gäller det att högre överföringshastighet medför större sannolikhet för fel, delvis på grund av att synkronisering av klockor blir mer känsligt.

B.5 Serial Peripheral Interface (SPI)

Serial Perephial Interface (SPI) är ett seriellt och synkront protokoll för datorkommunikation utvecklat av Motorola [53]. Kommunikationen sker på minst fyra ledare (busstrådar) [53], [54] som visas i figur B.4; en för datasändning från mastern till slaven (Master Out Slave In, MOSI), en för att skicka data från slaven till mastern (Master In Slave Out MISO), den tredje för klockan (Serial Clock, SCK) och slutligen N stycken ledare (en för varje slav) för att välja vilken slav som ska få delta i kommunikationen (Slave Select, SS). Protokollet är anpassat för kommunikation mellan en master och ett godtyckligt antal slavar [55], [56]. Det finns dock möjlighet att använda fler masters förutsatt att bara en master kommunicerar på bussen åt gången.



Figur B.4: Kommunikation mellan två SPI enheter. SDI, SDO, SCK och CS är namnen på slavens motsvarigheter till masterns in- och utgångar som beskrivs ovan.

I SPI kommunikation använder sig både mastern och slaven utav skiftregister [53]. Under varje klockcykel skiftar mastern ett steg i skiftregistret och skickar en bit till slaven [53], [54]. Under samma klockcykel skiftar slaven ett steg i skiftregistret och skickar en bit till mastern. Detta innebär alltså att data kan både skickas och läsas av samtidigt [53]. Mastern kan också användas för att bara skriva till slaven genom att ignorera mottagna bytes eller bara läsa av data från slaven genom att skicka dummy bytes till slaven som sätter igång kommunikationen.

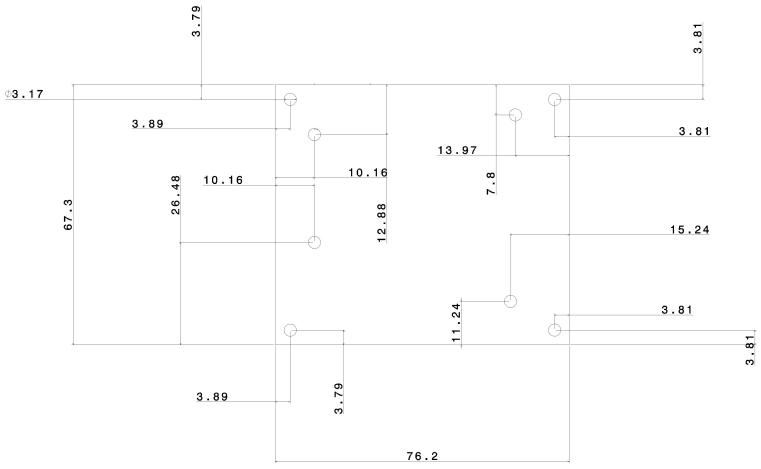
B.6 Direct Memory Access (DMA)

Direct Memory Access (DMA) är ett sätt för periferienheter att minska belastningen på centralprocessorn (CPU) genom direkt åtkomst av minnesenheter via minnesbussen [57]. En DMA-enhet är generellt uppbyggd av flera så kallade DMA-kanaler som enskilda periferienheter använder. Minnesbussen kan endast nås av en enhet (till exempel CPU:n eller DMA-kanalerna) åt gången. DMA-kanalerna initieras av CPU:n genom att i deras konfigureringsregister skriva vilka minnesadresser dem får tillgång till, storleken på meddelandena som ska överföras samt vilket överföringsläge som ska användas. De olika överföringslägena kallas *burst* eller *single word*. I burst-läget så får DMA-kanalen genomföra överföring av hela meddelandet innan den överlåter kontroll av minnesbussen till CPU:n igen. I *single word*-läget däremot så överförs endast en ordlängd data innan kontroll överlåts.

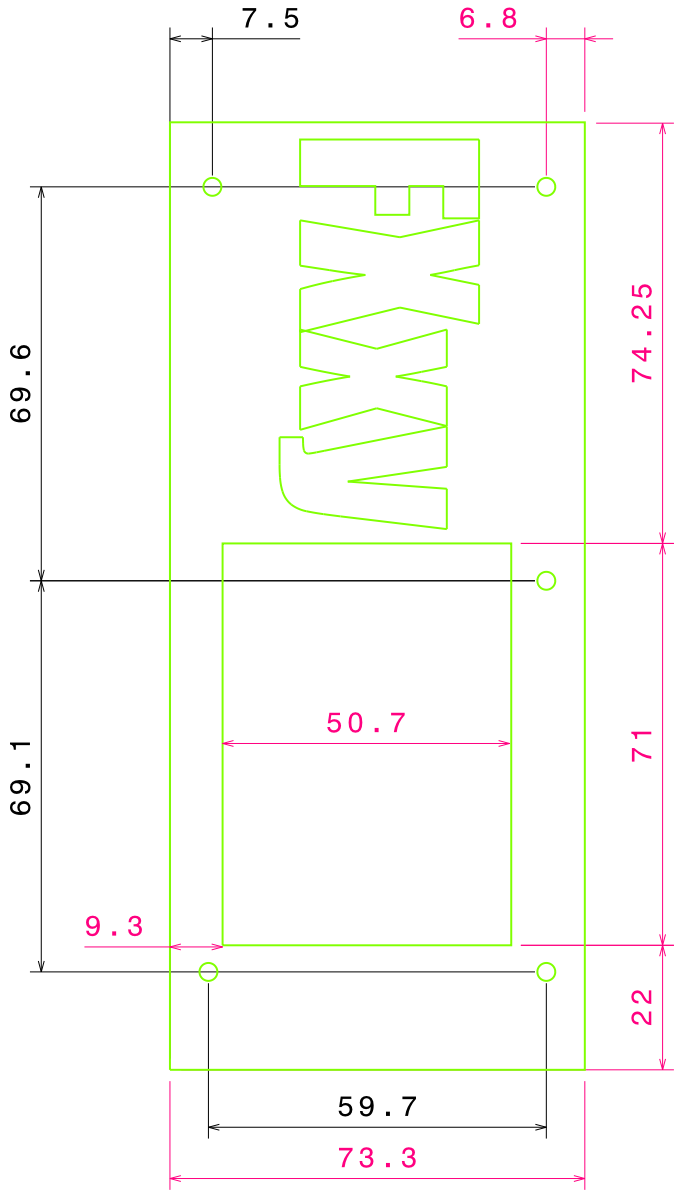
C Programkod och MATLAB-kod

All programkod och MATLAB-kod finns tillgänglig online på Github under: `https://github.com/erikpayerl/Digital-musical-effects-with-DSP`.

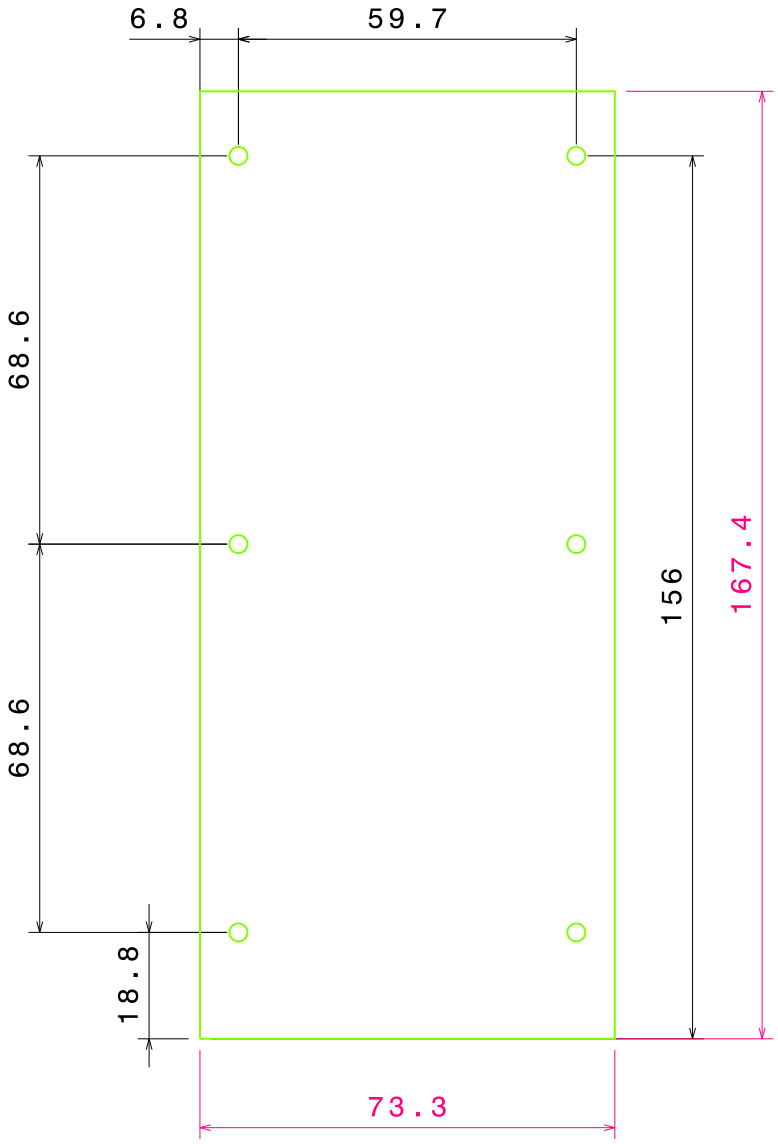
D CAD-ritningar



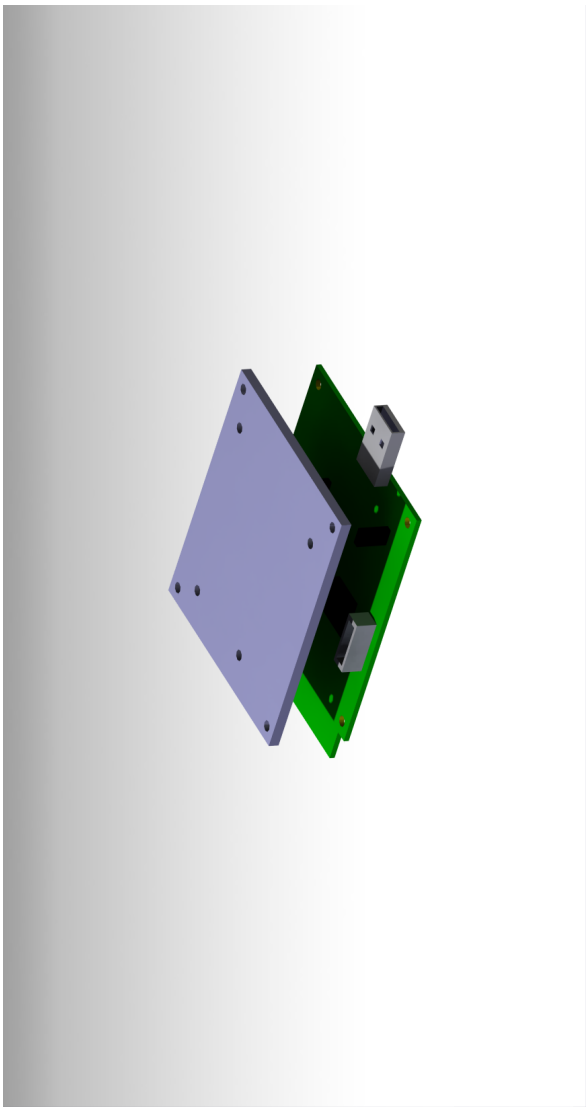
Figur D.1: Produktens mittplatta sedd ovanfrån



Figur D.2: Produktens ovansida sedd ovanfrån



Figur D.3: Produktens undersida sedd ovanfrån



Figur D.4: Produktens mittplatta och DSP i fler dimensioner