0.1 Arbejdsblad

Hej Søren

Generelt vil vi gerne have respons på hvad der foreløbigt står i kapitel 5.

Vi vil gerne høre hvordan vi skal strukturere og beskrive kode i rapporten, da vi mener at lange kodeeksempler vil blive tørre og ensformige at læse. Der vil dog stadig være brug for at forklare nogle ting og vi vil derfor gerne vide i hvilket omfang det skal gennemgås.

Indholdsfortegnelse

Introduktion 1

1.1 Hvad er en trommemaskine?

En trommemaskine er et redskab der benyttes til let at fremstille rytmer. Dette gøres ved at maskinen kører rundt i et uendeligt loop, hvor brugeren kan indsætte eller fjerne instrumentlyde på de ønskede dele af takter. Et enkelt af disse loops kan kaldes for en sekvens, hvilket svare til én takt. De fleste trommemaskiner har delt denne takt op i 16 dele, hvor man kan indsætte hver enkelt instrumentlyd på hver 16. del takt. Trommemaskinen består af flere forskellige trommelyde så som stortromme, lilletromme og high-hat. På denne måde kan man fremstille en trommerytme ved at indsætte forskellige lyde på udvalgte steder i cyklussen.

Trommemaskinens lyde er forskellige alt efter trommemaskinens producent. Nogle trommemaskiner anvender digitale lydsamples, hvor andre benytter sig af analoge kredsløb til at imitere trommelyde, eller en blanding heraf. Begge metoder har sine fordele og ulemper. For eksempel kræver digitale trommemaskiner konvertering mellem analoge- og digitale signaler, men de har til gengæld større mulighed indspille personlige lydklip. Fælles for begge typer af trommemaskiner, er at de kan afspille flere instrumentlyde på samme brøkdel af takten, hvilket betyder at de begge har en metode hvorpå de sammenlægger lyde.

Trommemaskinen var med til at skabe den kendte lyd fra 80'erne. Blandt de mest kendte trommemaskiner finder man Roland TR-808, som blev introduceret i 1980, og dens efterfølger, Roland TR-909 fra 1984. TR-808 imiterede diverse trommelyde ved hjælp af analoge kredsløb, hvilket betød at den havde den velkendte "robotagtgie" lyd i form af kliklyde fra hardwaren. Opfindelsen af trommemaskinen gjorde det muligt for enhver at fremstille rytmer uden at kunne spille trommer eller have anden musikalsk kendskab, hvilket også lægger til grund for dens popularitet.

1.2 Initierende problemformulering

Da dette er et 4. semesters projekt, som omhandler digital design, tages der udgangspunkt i en digital trommemaskine. Derfor opstilles en initierende problemformulering således at teorien bag trommemaskinens opbygning og elementer kan undersøges. Den initierende problemformulering lyder som følgende:

Hvilke elementer indgår i en trommemaskine der fremstilles digitalt?

Teori 2

I dette kapitel beskrives teorien bag elementer i en digital trommemaskine og hvordan disse fungerer.

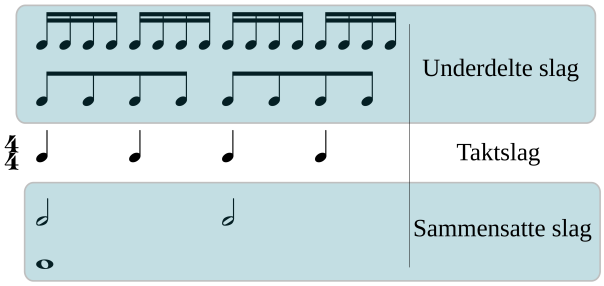
2.1 Musik og rytmik

En trommemaskines primære funktion er at kunne programmere rytmer og afspille disse. Derfor er det nødvendigt at redegøre for hvordan en rytme kan beskrives.

Musik inddeles i takter som udgør grundrytmen. Disse takter opdeles i taktslag. En taktart noteres som en brøk f.eks. $\frac{4}{4}$. Tallet over brøkstregen bestemmer hvor mange taktslag der går på en takt, og tallet i nævneren bestemmer nodeværdien af taktslagende.

Den mest anvendte taktart i vestlig musik er $\frac{4}{4}$ der dermed består af fire taktslag af fjerdedelsnoder. [10]

Taktslagene i en takt kan inddeles på mange måder. Eksempelvis kan en fjerdedelsnode erstattes med to ottendedelsnoder, som derefter kan opdeles i to sekstendedelsnoder, o.s.v. På modsatvis kan to fjerdedelsnoder erstattes af en halvnode, hvormed to halvnoder kan erstattes med en helnode, der fylder en hel takt. Alle disse nodeværdier kan også erstattes med en pause af samme længde.



Figur 2.1: $\frac{4}{4}$ inddelt i serier af forskellige nodeværdier how do i poot dis?. Illustration fra [15]

I musik bliver tempoet bestemt af antallet af taktslag pr. minut også kaldet BPM (beats per minute). Normalvis er tempoet mellem 80 til 160 BPM.[9]. Dette vil sige at en sang med et tempo på 100 BPM i $\frac{4}{4}$ vil have 25 takter i minuttet.

For at kunne holde en konstant stabil rytme bruger musikkere ofte et apparat kaldet en metronom. En metronom er oftest konstrueret på en sådan måde at brugeren vælger et givent tempo i BPM, hvorefter metronomen spiller en puls i det givne tempo, ofte i en given taktart med markering på første slag.

Metronomer findes i et utal af størrelser lige fra mekaniske metronomer bestående af et pendul, til apps til smartphones.

Det viser sig at selv trænede musikkere ikke ramme hver enkelt takslag helt perfekt. Musik spillet eller indspillet af mennesker har typisk en gennemsnitlig afvigelse fra takstslagene på 10-20 ms.

Dette er dog ikke noget problem, faktisk tværtimod. Computergenereret musik der falder præcis på slagene lyder i menneskelige ører som meget unaturligt og følelsesløst, hvilket gør at man ofte tilføjer små afvigelser i computergenereret musik for at gøre det mere "menneskeligt"[7].

2.2 Sequencer

En sequencer har til formål, at styre hvornår bestemte lyde bliver afpillet i trommemaskinen. Dette gøres ved at afspille en sekvens som er opdelt i mindre dele. Alt efter hvilke funktionalitetter der søges af sequneceren, kan antallet af dele i en sekvens ændres. Det samme gælder for tiden det tager at afspille en sekvens. Eksempelvis kan der være 16 dele for en trommemaskine, hvilket giver rig mulighed for at repræsenterer en takt. Disse giver mulighed for at afspille en bestemt sekvens, som indeholder række lyde. Denne sekvens kan så gentages eller en anden sekvens kan efterfølgende afspilles.

Sequenceren har den fordel at tiden det tager at afspille en bestemt sekvens er forudbestemt. Det er derved muligt at få sequenceren til at holde en bestemt rytme. Derudover er det muligt at ændre tiden det tager af afspille en sekvens, så man derved kan hæve eller sænke tempoet der afspilles i.

De første sequencere der blev opfundet var byggede af analoge komponenter. Disse bestod af en "clock", som gennemgår en række steps og derefter starter forfra. Disse sequencere blev koblet til en synthesizer styret af flere række potentiometre, oftes med otte per række. Hver række blev så brugt til at styre en kanal for synthesizer. Et eksempel på dette kan ses på figur 2.2, hvor sequenceren, Moog 960, ses midterst til højre. Derved kunne man skrue på potentiometrene for at få en bestemt lyd ud og vælge om den skulle afspilles eller ej. Nogle af de første musikere, som lavede elektronisk musik, ville indstille disse potentiometre live på scenen når de optrådte.

Figur 2.2: På billedet ses en Moog 960, som er en analog sequencer og synthesizer

Moderne sequencere er nemme at styre ved hjælp af software. Ud over at gøre det nemmere at sætte op, har det også givet mulighed for at tilføje en række funktioner. Eksempelvis kan de ønskes at sequenceren kan afspille i reeltid. Her vil mængden af steps per sekvens blive øget

betydeligt, så det er nemmere at indpasse hvornår et bestemt step skal afspilles i forhold til realtid.

Der findes mange måde at programmere en sequencer til at afspille en sekvens. En meget anvendt metode er med en række knapper der kan rykkes ned. 16 knapper kan f.eks. svare til en takt inddelt i 16 dele som nævnt tidligere.

En måde er at lade en bruger slå på nogle 'trommer' i en bestemt rytme, der derefter gemmes og afspilles. Til dette formål kan man anvende en såkaldt sampler.

2.2.1 Sampler

En sampler er en form for musikinstrument der har til formål at afspille nogle bestemte lyder eller "samples", når der gives et input. Brugerfladen kan for eksempel være store knapper, der kan trommes på, eller tangenter på et keyboard.

Sampleren bruges som en erstatning til et virkeligt instrument f.eks. til et trommesæt ved at lade den kunne spille samples fra forskellige trommer på et trommesæt, eller et sample fra et klaver der kan afspilles i forskellige toner. Det er vel ikke sampleren der spiller musikken, det er vel en kombi af sampler og sequencer?

Ofte er knapperne trykfølsomme sådan at der kan kendes forskel på om en knap berøres let eller den trykkes langt ned, så man både kan spille 'blødt' og 'hårdt'.

I daglig tale benyttes sampler også som en fællesbetegnelse for et apparat der både indeholder sampling og en sequencer, som for eksempel sampleren vist på figur 2.3

Figur 2.3: Sampleren Akai MPC2000 fra 1997. Billede fra [14]

Sampleren kan eventuel også bruges til at programmere en sequencer i realtid. Dette gør at den indspillede input kan blive gentaget og eksporteret.

skal der stå mere?

2.3 Digital lyd

Dette afsnit er skrevet af !Mathias Hvorfor er der behov for at gemme lyd digitalt?

Når to personer snakker sammen, foregår det ved at taleren producerer lydbølger, som lytteren hører. Lydstyrken der opleves af lytteren, afhænger af hvor højt taleren snakker samt afstanden mellem dem. Fastholdes lydstyrken mens afstanden mellem dem øges, vil lytteren opleve en lavere lydstyrke, og efter en hvis afstand kan lytteren ikke længere kunne høre taleren. Der er derfor et behov for en bedre måde at transportere lyden fra taleren til lytteren.

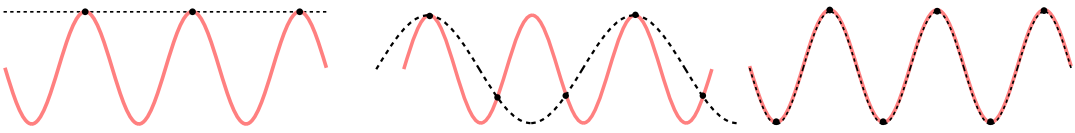
Ved at placere en transducer (f.eks. en mikrofon) ved lydkilden, kan lydbølgerne omdannes til elektriske signaler som kan videregives via ledninger. Placeres endnu en transducer (f.eks. en højtaler eller hovedtelefoner) ved lytteren, kan det elektroniske signal igen omdannes til hørbare lydbølger. På denne måde kan afstanden mellem lydkilden og lytteren blive væsentligt længere. Det elektroniske signal kan være enten analogt eller digitalt. Den store fordel ved gøre systemet digitalt er, det er mere hårdfør overfor støj. Derudover kan det gemmes til senere brug.

Analog til digital

Mikrofonen ved taleren omdanner lydbølgerne til et analogt elektrisk signal. Når dette analoge signal skal konverteres til et digitalt signal, tages en analog-til-digital-konverter i brug. Denne kaldes fremover for en ADC. En ADC konverterer det analoge signal til en sekvens af digitale bits, som repræsenterer spændingen. Denne måling kaldes en sample. Modtager den en DC-spænding på indputtet vil den altid give det samme på outputtet. Sendes en AC-spænding ind, vil udgangssignalet blive omdammet til en digital værdi for spændingen, på det tidspunkt samplen er taget.

2.3.1 Sampleratens indflydelse på genskabelsen af det originale signal

For at kunne genskabe AC-signalet er der behov for et antal samples og når der snakkes om AC-signaler snakkes der også en en frekvens derfor angives antallet af samples pr. tid, som en sample-frekvens. Ifølge Nyquist skal samplefrekvensen være over dobbelt så høj som den højeste frekvens man vil kunne genskabe.



(a) Ens sample-rate og frekvens.

(b) Sample-raten er 1,5 gange frekvens.

(c) Sample-rate er den dobbelte frekvens.

Figur 2.4: Nyquist læresætning for samplerate. Den røde er signalet og den sorte er den bedste genskabelse af signalet.

Som det kan ses på figur 2.4 skal samplingfrekvensen være minimun den doppelte frekvens for at opfange signalet ordenligt. Da hørbar lyd ligger i frekvensområdet mellem 20 Hz og 20 kHz skal samplingfrekvensen være over 40 kHz. Samplefrekvensen for CD'er er valgt til 44,1 kHz, jævnfør IEC 60908KILDE, da denne ligger over de tidligere nævnte 40 kHz. Aliasering er et udtryk for, når to eller flere signaler ikke kan kendes fra hinanden og vil i praksis betyde at CD-systemet ikke kan kende forskel på det ønskede signal og et forkert signal. KILDER! For at højere frekvenser ikke kan ødelægge signalet grundet aliasering, er det valgt at filtrere signaler over 20 kHz fra. Dette filter er et lavpas filter, der er implementeret således at frekvenser, som kan skabe problemer for sampleren, bliver filtreret fra. Dette er også kendt som "anti-aliasering". Når sampleraten er valgt til 44,1 kHz er det muligt at genskabe frekvenser under 22,05 kHz, og

alle frekvenser herover skal derfor filtreres fra. Dette betyder også at alle de hørbare frekvenser kan opfanges.

Da nogle af problematikkerne ved genskabelsen af det originale signal, i forhold til sampleraten af perfekte samples, er undersøgt, er det interessant at undersøge hvilken indflydelse præcisionen af disse samples har.

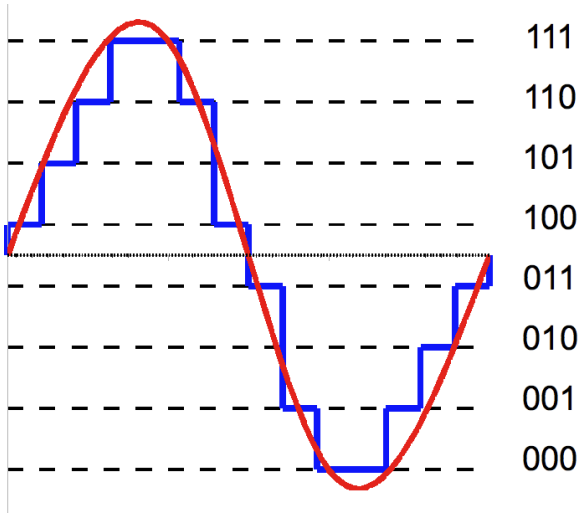
2.3.2 Nødvendig præcision for analog-digital konvertering

indsæt sejt billede med ADC Opløsning

Når et analogt signal skal lagres digitalt, skal det først konverteres til et digitalt. I de tidligere afsnit er behovet for konverteringen beskrevet og sampleraten er undersøgt. I dette afsnit undersøges den nødvendige bit-dybde til optagelse af analoge signaler.

Når det analoge signal skal konverteres, tages en Analog til Digital Konverter i brug (ADC). Denne virker ved at måle spændingen på et given tidspunkt, bestemt af sampleraten. Spændingen repræsenteres så med en digital værdi af et antal bits bestemt af opløsningen af ADC'en. Dette er illustreret på figur 2.5 herunder.

SQNR - Wiki



Figur 2.5: Sammenligning mellem en 3-bit repræsentation (blå) af et analogt signal (rød)

På figur 2.5 ses en periode af et sinusoidalt signal (den røde) og en 3-bit ADC (den blå). Som det kan ses er den digitale repræsentation ikke helt ens. Dette skyldes ADC'ens opløsning, da denne definerer de mindste værdier signalet kan inddeles i. En 3-bit ADC har maksimalt $2^3 = 8$ værdier hvorimod en f.eks. 16-bit ADC har $2^{16} = 65.536$, hvilket medfører flere bit giver en bedre repræsentation, da afrundingsfejlen bliver mindsket. Størrelsen af afrundingsfejlen kaldes "Signal-to-Quantization-Noise Ratio" (SQNR). Antages at ADC'en er en ideal komponent hvor fejlmarginen for afrundingsfejl er $\pm^1/_2$ LSB og afrundingsfejlen ikke afhænger af området der måles i, kan SQNR udregnes som:

$$\text{SQNR} = 20 \cdot \log_{10}(2^Q) \quad [\text{dB}] \tag{2.3.1}$$

hvor:

Q = ADC bit-dybde [·]

Ud fra ligning (2.3.1) kan ses en sammenhæng mellem antallet af bits i ADC'en og fejlmargenen. Eksempelvis er SQNR for en 6 bit ADC $\approx 36,1$ dB. Sammenlignes denne med en 8 bit ADC ($\approx 48,1$ dB) er forskellen ≈ 12 dB.

I praksis anvender lydstudier ofte 24 bit, eller derover, når der samples lyd, da dette giver en mere præcis AD-konvertering. Typisk er den software der arbejdes med, udviklet til at kunne håndtere endnu flere bit når der mixes, som derefter nedskaleres til eksempelvis 16-bit CD kvalitet.

2.3.3 Lagring af digital lyd

Efter konverteringen til digital lyd, skal dataene kunne gemmes, således at den kan tilgås efter ønske. Der er tre overordnede måder at gøre dette på. Det første metode er at gemme dataet Til SKO - hvordan bøjes data i rå format. Det vil sige at der ikke sker nogen komprimering af data, hvilket betyder at denne metode kræver meget lagerplads. Til gen1gæld er lydkvalitet på sit højeste, da man på ingen måde behandler dataen og at lyden kan genskabes hvis ellers nyquist er k. En anden fordel ved denne datatype er at der ikke behøves nogen speciel form for læsning af dataen, hvilket vil gøre afspilning lettere. Filtyper der gemmes som rå data kan for eksempel være .WAV-formatet (Waveform Audio File Format). Denne filtype inkluderer dog en header der indeholder information om filen, og derudover er den rå data delt op i output til skiftevis højre og ventre audiokanal.

En anden måde at gemme sin data på er med lossless komprimering, hvor det er muligt at genskabe den originale lyd, uden at tabe data. Dette er filtyper som .flac (Free Lossless Audio Codec) eller .alac (Apple Lossless Audio Codec). Fordelen ved disse filtyper er at komprimeringen kan reducere filens størrelse med op til 50 % uden tap af data. Dette fungere ved at beskrive et antal bit med et mindre antal bit.[6] Kravet for at afspille en sådan fil, er at afspilningsenheden skal vide hvordan den skal håndtere filen, hvilket også kaldes audiocodecs.

Den sidste metode er at gemme sin lyd i et lossy format. Ved denne metode vil der være tab af data, som ikke kan genskabes under afspilning af lyden. Til gengæld er filens størrelse meget lille i forhold til den rå data. I stedet for at gemme den nøjagtige data, gemmes der i stedet for data der minder om den originale.[6] Der er mange forskellige måder at komprimere sin data på, og de har alle deres fordele og ulemper. To af disse metoder kan være ved brug af A-law og µ-law, hvor komprimeringen tager udgangspunkt i den menneskelige hørelse. Da den menneskelige hørelse er logaritmisk er den også mere sensitiv ved små signaler frem for store. Der afsættes derfor flere bits til små signaler hvilket giver dem en højere opløsning.[16]

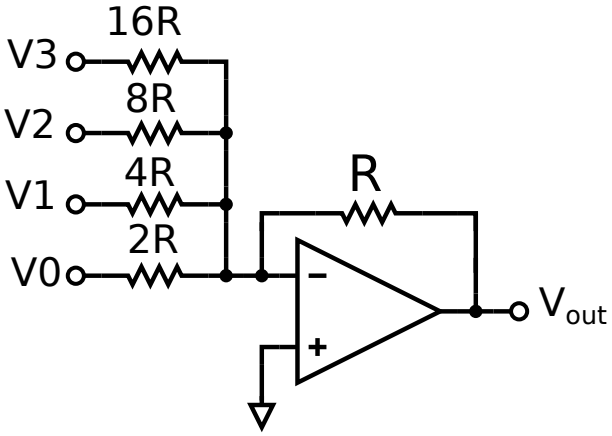
Ligesom filer med lossless komprimering kræver filer med lossy komprimering også kendskab til hvordan filen skal afspilles.

2.4 Digital-til-analog-konvertering

For at en trommemaskine skal kunne afspille lyd eller give et audiosignal som output, er det nødvendigt at kunne konverterer digitale lydsignaler til analoge. Dette kan gøres på forskellige måder. En række metoder til digital-til-analog-konvertering vil blive beskrevet i dette afsnit.

2.4.1 Binær vægtet D/A-konverter

En simpel måde at konvertere fra et tal repræsenteret som binære cifre eller bits, er ved en såkaldt binær vægtet D/A-konverter.



Figur 2.6: 4 bit D/A-konverter med binært vægtede modstande

Kredsløbet som kan ses på figur 2.6 er baseret på en summerende operationsforstærker. For dette kredsløb kan opstilles sammenhænget:

$$V_{out} = V_0 \cdot \frac{R}{2 \cdot R} + V_1 \cdot \frac{R}{4 \cdot R} + V_2 \cdot \frac{R}{8 \cdot R} + V_3 \cdot \frac{R}{16 \cdot R} = V_0 \cdot \frac{1}{2} + V_1 \cdot \frac{1}{4} + V_2 \cdot \frac{1}{8} + V_3 \cdot \frac{1}{16} \quad [\text{V}] \quad (2.4.1)$$

Hver inputspænding skal have værdien 1 eller 0 svarende til en spænding på V_{IO} og 0 V. Det kan ses i formel 2.4.1 at inputspændingernes forstærkning halveres, med MSB ved V_0. På denne måde bliver spændingen V_{out} en sum af spændingerne på inputtet, hvilket resulterer i den digitale til analoge konvertering med inkrementer af $\frac{1}{16}$ i intervallet 0 V - $\frac{15}{16} \cdot V_{IO}$. Det kan ligeledes ses at jo flere paralelle inputs, jo højere opløsning kan der opnås.

fordele

simpel

kan laves selv

ulemper

krav til mange præcise modstandsværdier

?????

P.g.a. mange komponentværdier...

2.4.2 R-2R-netværk

En R-2R-D/A-konverter virker på mange måder som den binære vægtede operationsforstærker som beskrevet i afsnit 2.4.1. fordelen ved en R-2R-D/A-konverter er at den udelukkende anvender to komponentværdier. Modstande med størrelsen R og modstande med størrelsen $2 \cdot R$.

I appendiks **??** vises det at R-2R-konverteren er ækvivalent med den binært vægtede D/A-konverter.

For at tilføje yderligere inputspændinger skal der blot tilføjes to modstande med størrelsen R og $2R$.

bruger flere komponenter men har færre komponentværdier

2.4.3 Pulsbredde-modulation

En yderligere simpel måde at omdanne fra et digital signal til et analogt er ved at omdanne det digitale signal til et pulsbreddemoduleret signal[1]. mere om pwm kommer senere.... Dette signal kan derefter omdannes til et analogt signal, ved at lade det passere et analogt lavpasfilter med betydeligt lavere knækfrekvens end frekvensen på PWM-signalet. Dette vil medføre at outputtet vil fremkomme som et gennemsnit af inputsignalet, hvilket vil give en analog spænding.

Kvaliteten uddyb, din scrublord af D/A-konverteren kommer i høj grad til at afhænge af det analoge filter man vælger at anvende [1].

1

2.4.4 Oversamplende D/A-konverter

En meget anvendt måde at lave D/A-konvertere er ved at....

bruger en samplingfrekvens en faktor større end nødtvendigt, til at opnå en større opløsning.

dette vil uddybes når vi engang fatter hvordan det virker.

2.5 Relevante interfaces

2.5.1 Musical Instrument Digital Interface (MIDI)

For at få digitale musikinterfaces til at kommunikere med hinanden anvendes ofte protokollen "Musical Instrument Digital Interface"i daglig tale kaldet MIDI. MIDI bruges til at sende informationer om musik ved at beskrive hvornår der skal spilles, hvilke toner der skal spilles og hvor hårdt/hvor længe en node skal anslås. Derudover sendes der en række kontrolsignaler til at beskrive parametre der er sat til et apparat. Dette kan f.eks. være information om volumen, rumklang og andre effekter. MIDI sender udelukkende informationer om musikken der skal fortolkes af et andet apparat, og er altså ikke en måde at sende ren lyd på. Eksempelvis anvendes MIDI, når man vil spille musik på et digitalt keyboard og indkode musikken i en sequencer, eller afspille det via en synthesizer.

Trommemaskiner anvender ofte MIDI til at interface en rytmesekvens med en computer, og til at bruge trommemaskinen til at udløse lyde fra et andet instrument. [2]

Det er muligt for MIDI at sende informationer om op til 16 kanaler svarende til 16 toner der kan spilles samtidigt.[2]. Dog er MIDI standarden lavet på en sådan måde at der kun sendes data på en kanal hvis der sker noget.

[1] Ofte kaldet PWM
[2] Antallet af toner der bliver spillet på samme tid kaldes polyfoni

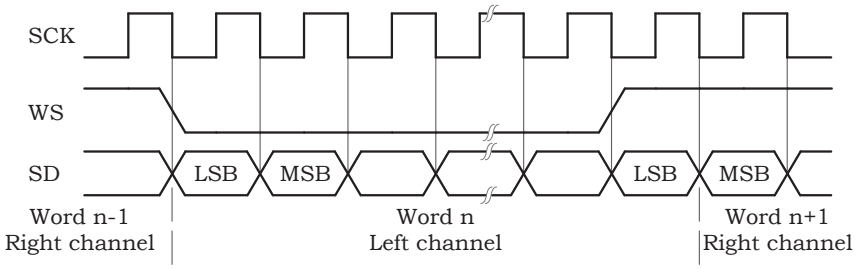
MIDI bliver sendt som en 8 bit besked der sendes serielt på en baud rate på 31,25 Hz. MIDI sender sit data asynkront og beskeden er derfor indkapslet i et start- og stop-bit. [3]

2.5.2 Inter-IC Sound (I^2S)

Det er også hensigtsmæssigt at have en stadard til hvordan digital lyd kan sendes imellem forkkellige moduler. Til dette formål findes interfacet I^2S. En I^2S-bus består af tre signaler: En seriel clock kaldet SCK, selve det serielle data SD, samt en word-select WS, der har til formål at styre hvornår der sendes på højre og venstre kanal, samt at styre bit-dybden af signalet[12].

Når der kommer en ændring i Word-select, modtages der en enkelt bit på den nuværende kanal inden der skiftes kanal. Derefter sendes næste MSB på den nye kanal. Dette er for at give tid til modtageregen til at gemme den sidste bit og blive klar til en ny MSB. WS = 0 svarer til at der sendes på venstre kanal og WS = svarer til højre.

Timingen af et I^2S-interface kan ses på figur 2.7



Figur 2.7: sum txt...muh HiRez Billede fra [12]

Hvis man vil afspille lyd i stereo, optaget ved 44,1 kHz med en bitdybde på 16 bit, skal man clock'en have en frekvens på:

$$f_{SCK} = 16 \cdot 2 \cdot 44,1\text{kHz} = 1,411\text{MHz} \tag{2.5.1}$$

2.6 Hardware i projektet

I dette projekt er det blevet valgt, at bruge en FPGA til at lave trommemaskinen. måske lidt tidligt at tage valg i projektet i teori delen... jeg kan ikke se noget godt alternativ men overvej at omformulere det. måske asic vs fpga vinklen? Dette er blevet valgt, da der på forhånd er blevet givet projekt gruppen en FPGA i form af en Papilio Duo, samtidig med at undervisningen tager udgangspunkt i denne.

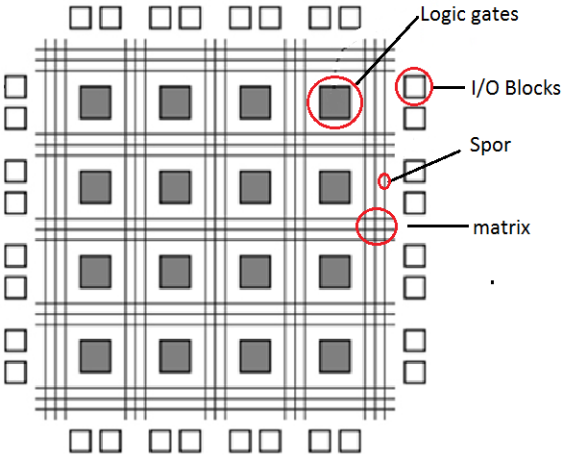
Fordelen ved denne er, at den giver indsigt i hvordan bl.a. logic gates fungere sammen med andre digitale komponenter. Samtidige med at FPGA'en er et ekstremt ageilt værktøj, i den forstand at det ikke er nødvendigt at lave nyt print, finde nye komponenter og lignede hvis man har lavet fejl eller har brug for at implementere nye funktioner, FPGA'en kan blot omprogrammeres, selvfølgelig inden for FPGA'ens egen begrænsninger. Alternativet ville være, at man selv skulle sætte passende logic gates sammen. Men eftersom der utvivltsomt ville komme ændringer, som følger af ny viden eller fejl, er FPGA'en igen et godt valg. Brugen af FPGA'en vil også betyde,

at man langt hurtigere kan starte designfasen, da eventuelle fejl ikke vil give et tilageslag, eller stort spildtid, men blot en omprogrammering af FPGA'en kan løse fejlen.

ER ASIC ÉN CHIP ELLER ER DET OGSÅ EN PASSENDE BESKRIVELSE AF PRINT SOM VI SELV LAVER? Når en færdig og fuldt funktionelt trommemaskine er lavet, kan man eventuelt lave den som en ASIC (application-specific integrated circuit), altså et anvendelses specifik integreret kredsløbs. ASIC er modparten til FPGA'en, som navnet antyder bliver en ASIC brugt til én specefik opgave. Det kan ikke omprogrammeres, men har en bedere performance og strøm forbrug.

2.6.1 Teori om FPGA

Felt Programmerbar Port Table som på engelsk hedder Field-Programmable Gate Array (FPGA), er i alt sin enkelthed et gate array som er? som er programmertbart, som det også fremgår af navnet. FPGA'en består af configurable logic blocks (CLB). disse CLB'er består af loop-up table(s) (LUT), Full adders (FA) og Flip Flops (FF). vi skal lige finde ud af om vi bruger engelske eller danske udtryk



Figur 2.8: General arkitektur i en FPGA

Disse CLB er forbundet med hinanden vha. forbindende spor, som gennem en forbindelsesmatrix giver mulighed for at forbinde CLB'sne på den måde man ønsker sig. Yderligere er dette også forbundet til programmerbar Input Output blokke (IOB) giver mulighed for at skabe forskellige in- og outputs fra eksterne kilder.

2.6.2 Papilio duo

Papilio duoen består af en Spartan 6 (XC6SLX9) FPGA fra Xilinx, som har 9.152 logic cells, 11.440 flip-flops og maksimalt 200 bruger I/O. Yderligere har den en AVR ATmega32U4 Microcontroller. Dette er en 8 bit microcontroller med en maksimal clock frekvens på 8/16 MHz (alt efter forsnyningsspænding), denne microcontroller er bl.a. brug på arduino leonardo.

Skriv løbende når ny indformation opdages

Nu er det muligt at afgrænse/inddele i moduler/lave problemformulering kom frisk

Problemformulering 3

3.1 Projektafgrænsning

En trommemaskine er et redskab der kan se ud på mange måder og der kan derfor tilføjes mange elementer, og endnu flere fordybelsesområder. Der foretages derfor en projektafgrænsning for at mindske projektet omfang, således at et realistisk mål for projektet kan opnåes.

På grund af projektets omfang og emne vil der ikke lægges stort fokus på at lave det hele "fra bunden". For at kunne fremstille en "spillende"demo samt at gøre det nemmere at fejlfinde og viderebygge projektet, vælges det at implementere projektet på konfigurerbar hardware, navnligt en FPGA som beskrevet i 2.6.

I en trommemaskine vil det være nødtvendigt at omdanne digitale signaler til analoge hvilket kræver en D/A-konverter, men da projektets fokus ligger på groove og lækre rytmer vælges der at benytte en allerede eksisterende D/A-konverter.

Da der ikke opnås stor teoretisk indsigt i emnet ved at gøre det samme flere gange, vil projektet også begrænses i forhold til antallet af lydsamples og gembare sekvenser, i betragtning af hvad en reel gangbar trommemaskine bør have.

Istedet vil fokuset i første omgang? ligge på at fremstille de essentielle dele i en trommemaskine, såsom at gøre det muligt at programmere rytmer i en sequencer, lagre disse og afspille disse rytmer med trommesamples.

Programmeringen af sequenceren vælges at implementeres både som sampler og en step-sequencer. Der vælges at arbejde med en sampler da denne anses som en udfordrende opgave, da dette er et forholdsvist tidsfølsomt system.

Dertil vil der laves en stepsequencer da denne vil gøre det nemt at programmere en sekvens i sequenceren, og derved gøre det lettere at arbejde med trommemaskines resterende moduler.

Derudover vælges der at bruge allerede eksisterende lydsamples frem for at fremstille dem selv, grundet at de anvendte lyde ikke har stor indflydelse på resten af projektet. Der kunne i princippet benyttes lyde af hunde der gør, uden at noget betydelig effekt på resten af trommemaskinens funktioner.

flere ting følger efterhånden som vi finder på flere ting at lave

3.2 Problemformulering

På baggrund af analysen i kapitel 2 er det nu mulige at opstille en problemformulering til dette projekt.

I accidentely the whole drum machine. what do now??

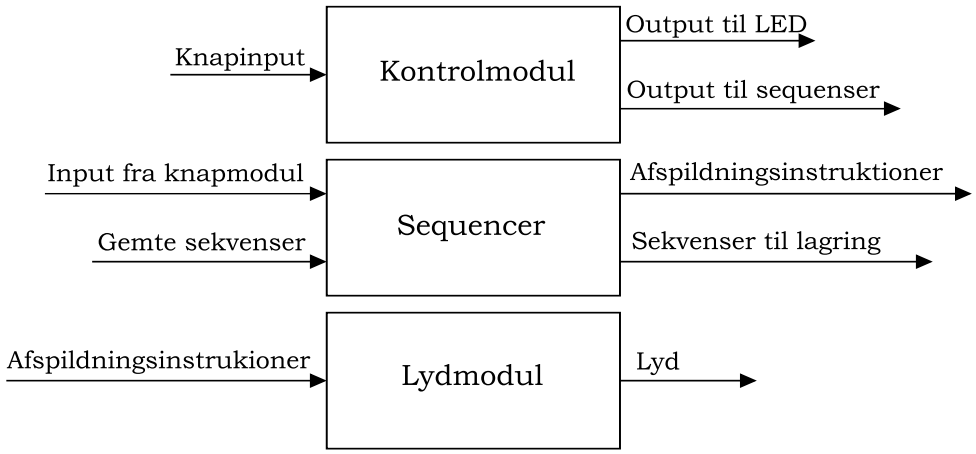
Kravspecifikation 4

4.1 Funktionelle krav

1. Skal gemme en rytme i en sequncer, med inputs fra en sampler og en step-sequencer
2. Skal have TBD forskellige kanaler at indspille til svarende til forskellige trommesamples.
3. Skal kunne afspille TBD samples på samme tid.
4. Indstille tempo mellem 50 BPM og 200 BPM.
5. Skal kunne gemme og genfinde sekvenser i hukommelse
6. skal kunne afspille sequencerens rytme som et audio-output

4.2 Krav til delmoduler

Ud fra de funktionelle krav, er funktionalliketerne blevet inddelt i tre overmoduler med hver deres inputs og outputs.



Figur 4.1: Flot modulting

4.2.1 Krav til Kontrolmodul

1. Skal tage imod brugerinput til:
   a) Indstilling af BPM
   b) Valg af trommesample
   c) Valg af, saving, loading og sletning af sekvenser
   d) Valg af samplingsmetode (step-sequencer eller fri sampling)
   e) Programmering af stepsequencer
   f) Tromning via sampler
2. Skal vise trommemaskinens status med en antal LED'er (hvordan step-sequenceren er programmeret, valg af trommesample, BPM og hvilket saveslot der benyttes)
3. Skal sende instrukser til hvad sequenceren skal gøre.

4.2.2 Krav til sequencer

1. Skal kunne blive programmet ved input fra kontrolmodulet
2. Indspillede trommeslag via sampler, skal ligge inden for 1 ms fra brugerens reelle taktslag.
3. Skal sende instrukser om hvilke lyde der skal afspilles og hvornår.
4. Skal kunne gemme og hente en sekvens fra et stykke hukommelse.
5. Skal kunne ændre den tid det tager at køre gennem en sekvens i henhold til indtastede BPM.

4.2.3 Krav til lydmodul

1. Skal kunne afspille trommesamples med en samplingfrekvens på 44,1 kHz i 16 bits opløsning.
2. Skal afspille op til 4 forskellige lydsamples samtidig.
3. Skal konvertere lyden fra et digitalt signal til et analogt outputsignal.

4.3 Beskrivelse af accepttest

if(spiller == True){return "fed... fyraften"} else {rubberDuckDebug(tålmodighed);}

Design 5

I dette kapitel vil der beskrives hvordan trommemaskinen designes. Designet er opdelt med udgangspunkt de tre delmodulers kravspecifikationer (kontrolmodul, sequencer og lydmodul) som det er beskrevet i **??**

5.1 Design af lydmoduler

En af de pprimære opgaver for en trommemaskine en at være i stand lyde som et trommesæt ved at afspille lyde. .....

5.1.1 Valg af samples

hvilke samples?

hvor mange byte fylder der?

5.1.2 modulindeling

5.1.3 Hurtig Tilgang til Lydfiler

Når lydfilerne skal afspilles skal filerne være klar til at blive tilgået med kort varsel. Derfor er det nødtvendigt at have lydfilerne gemt et sted hvor de hurtigt kan læses.

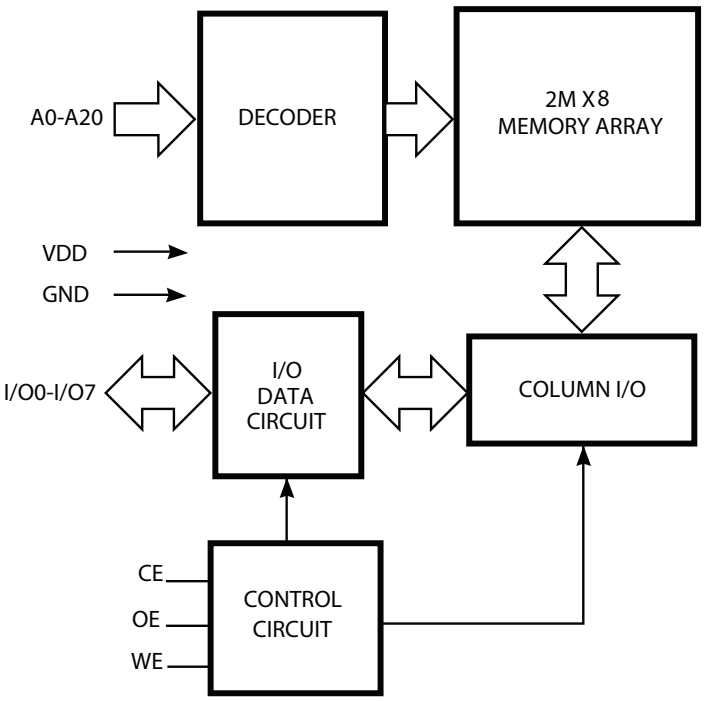
Dette kan gøres på flere forskellige måder. Det er muligt benytte selve FPGA'en som RAM, på to forskellige måder. Enten som distribueret RAM eller blok-RAM. forklaring eller ordlise følger. Distribueret RAM består af blokke af hukommelse lavet ud af FPGA'ens logikceller. Distribueret RAM er meget nemt at bruge siden det kan forbindes direkte til systemets øvrige logik. Dette giver også en klar øvre grænse til hvor meget hukommelse der kan bruges. Det er kun muligt at få nogle få kilobytes hukommelse med distribueret RAM, hvilket gør denne type hukommelse uhensigtsmæssig til at lagre lydfiler.

En anden måde at lagre data på FPGA'en er ved brug af dens blok-RAM[1]. Blok-RAM er en lille modul af hukommelse på en FPGA, der sidder separat fra FPGA'ens logikceller. På en Spartan-6 består hver blok-RAM af 18 kb hukommelse. På den anvendte Papilio Duo er der 32 af disse blok-RAM, hvilket svarer til 576 kb eller 72 kB. Dette er stadig ret lavt i forhold til de valgte wav-filer på i alt TBD.

På grund af manglende hukommelse inb4 rasmus jokes vælges det at filerne skal lagres på eksternt hukommelse. Papilio Duo'en er udstyret med chippen IS61WV20488BLL[8], der er udstyret med 2 mB Statisk RAM[2].

Et blokdiagram over SRAM-chippen kan ses på figur 5.1

[1] Nogen gange benævnt som BRAM
[2] SRAM



Figur 5.1: blokdiagram over SRAM-chip

Chippen kontrolleres ved hjælp af fem signaler. signalet CE står for chip enable og styrer om chippen skal være tændt eller slukket. OE (output enable) sættes til lav, hvis der ønskes at læses fra chippen. WE (write enable) sættes til lav hvis der ønskes at skrive. Det læste eller skrevne data sendes over en in/out-port på 8 bit, og nummeret på den adresse man vil tilgå, skrives over en 21 bits adresseport.

Den anvendte chip har en forholdsvis hurtigt access time på højest 20 ns. Da SRAM-chippen er asynkron kan der potentielt opstå problemer, når der skal udføres hukommelsesoperationer eksempelvis på baggrund af en clock fra resten af systemet. Adresse- og kontrolbit'ene skal holdes fast i et specifikt stykke tid før in- og outputdata kan sendes korrekt.

Til dette er der lavet et simpelt interface, til kommunikation med SRAM-chippen fra FPGA'en, i VHDL. Interfacet er baseret på et kodeeksempel fra [5], modificeret til chippen IS61WV20488BLL.

Et blokdiagram over interfacet mellem FPGA'en og SRAM-chippen kan ses på figur

SRAM-interfacet er lavet som en tilstandsmaskine. Signalet "mem"sættes højt når en hukommelsesoperation skal initieres. Hvorvidt denne operation skal læse eller skrive specificeres af "rw"(read/write). Tilstandsmakinen bruger derefter to clockcyclusser på først at sætte de relevante adresse- og kontrolbits til det de skal være, og derefter at sende/modtage den ønskede data og sætte dette på et register. Da en hukommelsesoperation tager mere end en clockcyclus, kigges der på signalet "ready"om hvorvidt SRAM-interfacet er i "idle-tilstand og dermed er klar til at tage imod nye instrukser. Et diagram over tilstandsmakinen kan ses på figur

5.1.4 Lagring af lydfiler

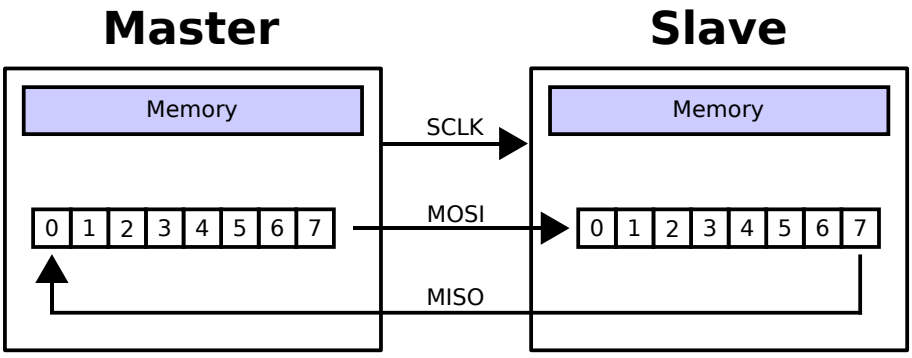
Der er nu styr på, hvordan FPGA'en kan skrive filer til, og læse filer fra SRAM-chippen. SRAM'et har dog det problem, som alt andet volatilt hukommelse, at det forsvinder når der slukkes for strømmen.

Derfor ønskers der noget hukommelse som ikke er volatilt, hvor lydfilerne kan gemmes permanent. Tanken er at lave en opstartssekvens, hvor lydfilerne flyttes fra det permanente hukommelse, ind i SRAM'ene. Umiddelbart kigges der på to løsninger: Enten bruges er det eksternt SD-kort og SD-kortlæser, som enten kobles op til FPGA'en eller arduinoen på papilio'en. Eller også bruges der noget af det flash hukommelse som der i forvejen ligger på papilio'en.

Umiddelbart falder valget på papilio'ens flash hukommelse, da dette allerede er integreret sammen med FGPA'en. Papilio'ens flash hukommelse er på 8 MB. Bitfiler (VHDL kode) fylder hver 33 kB, altså vil der være mulighed for at lagerer flere bitfiler på hukommelsen, men da vi antager at det ikke er nødvendigt at ligge op 23 bitfiler (i følge papilio) på hukommelsen, vil der være overskydende plads på hukommelsen. Lydfilerne som man ønsker at ligger på flash hukommelsen er en størrelse på TBD, og da der ikke ønskes mere end 4-8 lydfiler, bør dette kunne lader sig at gøre. Valget falder derfor på papilio'ens flash hukommelse, således det ikke er nødvendigt at bruge tid på at integrer en SD-kortlæser.

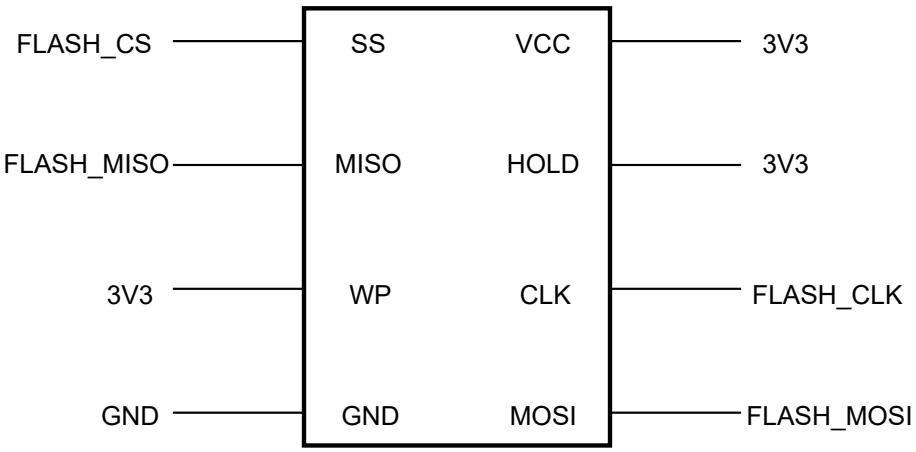
Papilio'ens flash hukommelse er en "MX25L6445E"8-PIN SOP som bruger SPI til at kommuniker med. SPI står for "Serial Perpheral Interface bus". Det bygger på et master-slave forhold, hvor masteren kan tilgå flere slaves, dog kræver dette et dedikeret ben per slave. SPI kommunikation består typisk af 4 ben - nogle gange flere, f.eks. kan MX25L6445E bruge optil 6 ben. De 4 typiske ben består af: SCLK (clock), SS/CS (Slave Select/Chip Select), MOSI (Master Out Slave In) og MISO (Master In Slave Out). Når masteren skal kommunikerer med slaven, sættes CS typisk lav, herefter kan masteren sende data med MOSI benet og modtage data med MISO benet. Ved dataoverførsel skriver masteren typisk dens MSB til slavens LSB over MOSI, samtidig skriver slaven dens MSB til masterens LSB. Altså skrives og læses der samtidig, også selv om man blot ønsker at overføre data til slaven.

Opdater figur, +1 register



Figur 5.2: SPI kommunikation med 8 bits - billede fra [13]

Selv om MX25L6445E chippen ligger op til at det er muligt bruge 4 data-ben på samme tid, er det dog ikke en mulighed, da 2 af disse på papilio'en er sat til VCC. Dermed er det kun muligt at bruge chippen på typisk vis. Forbindelserne af flash chippen kan ses på figur 5.3.



Figur 5.3: Flash chippens forbindelser på papilio duo'en

NOGET OM SPI-FLASH LOADER VHA. ARDUINO - det skal en tur med groft sandpapir

For at kunne hente lydfilerne fra flash hukommelsen over i SRAM, skal filerne først ligge på flash hukommelsen. Til at skrive disse filer til hukommelsen, tages en ekstern Arduino Due i brug, da denne har 3.3V logik og tilstrækkelig plads til at opbevare lydfilerne under programmering. For denne kan få forbindelse til flash hukommelsen udarbejdes et program til FPGA'en, så denne giver forbindelse mellem flash hukommelsen og de eksterne ben.

I flash hukommelsens datablad findes et diagram over flowet når data skal skrives til hukommelsen. Dette kan også findes i appendiks **??**. Der tages udgangspunkt i dette flow, når lydfilerne skal skrives til hukommelsen.

Hvordan beskrives programmerings-arduinoen??

For, med succes, at kunne skrive data til flash hukommelsen

Lydfilerne som skal ligge på flash hukommelsen, ønsker vi at overføre vha. en arduino fordi en rigtigt god begrundelse. Dette gøres ved at programmere enten arduino-duo eller on-board arduinoen, og forbinde denne vha. af FPGA'en til flash hukommelsen. Således at den skriver direkte til hukommelsen, hvorefter man forhåbentligt kan gøre dette skrive-beskyttet. Herefter skal det så være muligt at lave en opstartssekvens, som tidligere nævnt overføre lydfilerne fra flash hukommelsen til SRAM'ene.

noget med spi/flash

5.1.5 Ved ikke hvad dette afsnit skal hedde men det handler om picoBlaze

Nu mangler der bare en måde at få filerne på Flash-hukommelsen over på FPGA'en. Det logiske valg er at lade lydfilerne blive på FPGA'en så længe trommemaskinen er tændt, og da FPGA'ens hukommelse er statisk er det kun nødvendigt at overføre filerne én gang.

skal dette skrives om, da vi muligvis også kommer til at bruge picoBlaze'eren til andet? Da denne handling kun skal udføres én gang og at filerne kun kan sendes sekventielt på baggrund af at SPI-Flash og FPGA'ens interfaces, vælges der at denne handling skal laves som noget der kan udføre disse sekventielle instruktioner én gang, og derefter ikke gøre noget.

Derfor vælges det at anvende en mikroprocesser til at lægge lydfilerne fra flash-hukommelsen over til FPGA'en, når trommemaskinen tændes.

Der er to oplagte valg til en mikroprocesser givet det hardware, der er tilgængelig i projektet. For det første kunne man anvende arduino'en eller skal jeg skrive AVR?, Der er en 8-bits mikroprocesser, med en maksimal clock-frekvens på 16 MHz er dette rigtigt?. Fordelen ved at bruge arduino'en er at den er meget simpel at programmere, og at den allerede ligger klar på papilio-boardet.

Derudover kunne kan en processor implementeres af FPGA'ens logikceller (en såkaldt soft-processor). Dette har den fordel at der ikke skal interfaces flere eksterne moduler, og at koblingen mellem SRAM og Flash-hukommelse bliver nemt da disse i forvejen er forbundet til FPGA'en. Derudover kan soft-processorens clock-frekvens sættes til at være clock'en på papilio'en, som har frekvensen 32 MHz. Dette er en dobbelt eller hvad? så hurtigt som clock-frekvensen på Arduionen.

Xilinx har lavet to mikroprocessore, der frit kan anvendes på Xilinx's FPGA'er (såsom Spartan-6). Disse kaldes henholdsvist microBlaze og picoBlaze. MicroBlaze er en 32-bit processor, og en del mere kompleks end en picoBlaze der kun er en 8-bits processor. En MicroBlaze er designet til at blive programmeret i c, hvorimod picoBlaze er lavet til a køre mindre programmer skrevet i assembly. Dog fylder en microBlaze en stor del af en FPGA, og da mikroprocessoren blot skal udføre et meget simpelt arbejde, en enkelt gang under systemets opstart vælges der at anvende en picoBlaze.

et eller andet med hvorfor vi har valgt picoblaze'eren over microBlaz'eren og arduinoen Fordelen ved at implementerer en soft-processor på FPGA'en, er at det ikke vil være nødvendigt, at implementerer fysiske I/O's på FPGA'en, men der kan dette gøres internet i FPGA'en, hvilket også betyder at det er muligt, at give soft-processoren lagt flere I/O, hvis dette skulle vise sig nødvendigt.

En picoBlaze er implemeteret som en fuldt udbygget microcontroller med program-hukommelse og en smule RAM til at gemme variabler.

Den specifikke udgave af picoBlaze skrevet til en spartan-6 FPGA, kaldes KCPSM6[3]. KCPSM6 kan eksekvere et program med op til 4096 instruktioner, hvor hver instruktion er defineret med en 18-bits vektor. Hver instruktion udføres på 2 clock-cyclusser hvilket i dette projekts tilfælde giver 16 Mips. Dog kan en reduceret clockfrekvens opstå med et program med over 2048 instruktioner på en Spartan-6 FPGA.

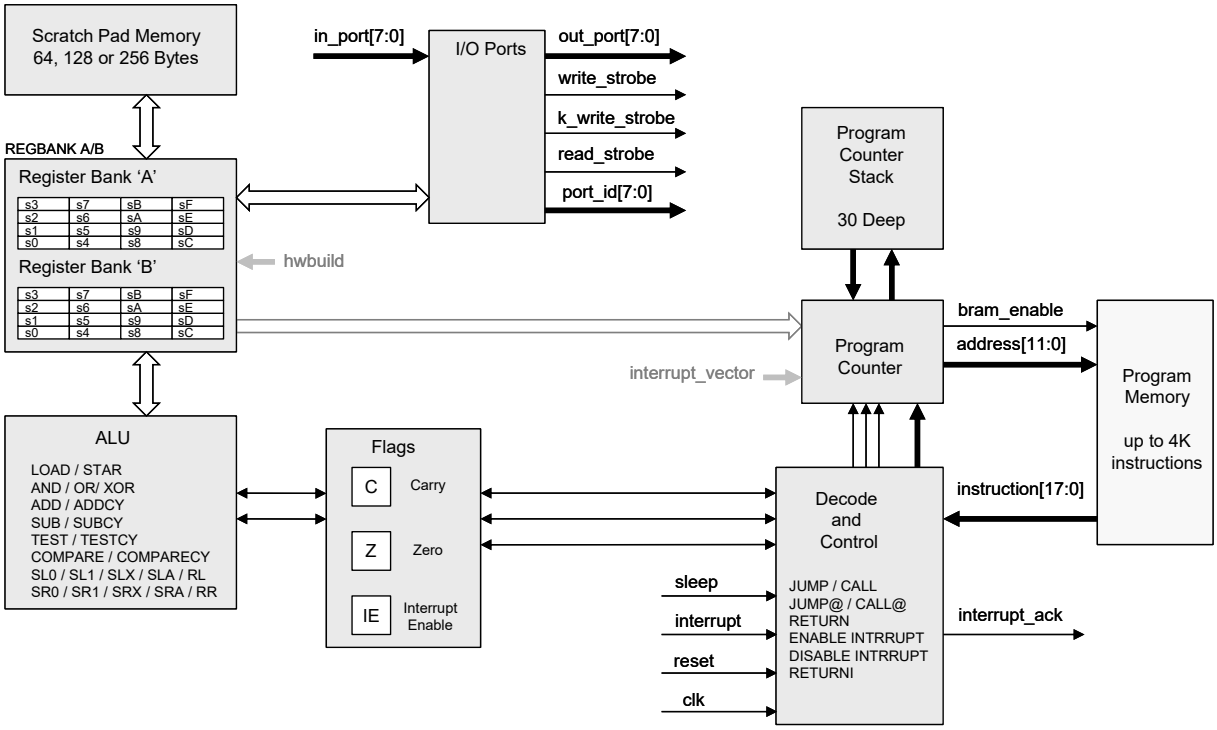
KCPSM6 har to registerbanke, A og B, af 16 registre til generelt brug. Registrerne benævnes "s1,s2,...,sF"[4].

picoBlaze'ens input og output styres ved hjælp af tre kontrolsignaler: write_strope, k_write_strope og read_strope, der hver viser om processeren har taget imod et input/sendt et output. Desuden bruges den 8-bit lange vektor port_id, til at styre hvor der skal læses fra/skrives til.

Det er også muligt at bruge "interrupts"og "sleeps"i forbindelse med picoBlaze'en men da dette ikke nvende i dette projekt vil disse ikke blive beskrevet videre.

[3]KCPSM står for "Constant(K) Coded Programmable State Machine"(førhen "Ken Chapman's PSM")

Et diagram over KCPSM6's arkitektur kan ses på figur 5.4



Figur 5.4: Blokdiagram over en picoBlaze Mikrokontroller. Billede fra [4]

Implementering af en PicoBlaze er relativt simpelt. Der kræves ikke andet en at tilføje to VHDL-moduler: Selve KCPSM6'en samt et modul til programmet's ROM. Sammen med KCPSM6'en tilbyder Xilinx et program der automatisk læser en Assemply-fil og oversætter det til et stykke program-hukkomelse til picoBlaze'en.

Da picoBlaze er en RISC-processor[4], kan den kun eksevere en begrænset mængde maskininstruktioner. måske i et appendix?

picoBlaze som postbud

Xilinx har lavet et reference design til kommunikation med flash hukommelsen vha. SPI kaldt "KC705_KCPSM6_SPI_Flash_reference_design"<- for meget?. I dette reference design findes "N25Q128_SPI_routines", som er assembly rutiner, som kan kaldes i det kode som man selv skriver til picoBlaze'eren. Reference designet til VHDL koden, er umiddelbart ikke brugbar, da visse dele af modulet ikke fungere på en Spartan-6. Derfor programmere vi istedet selv de 4 ben, som skal bruges som interface mellem flash og picoBlaze, på samme måde reference designet umiddelbart selv ligger optil, således assembly rutinerne kun skal ændres minimalt.

Rasmus'es beskrivelse af hvordan SRAM spiller sammen med picoBlaze'eren + billede?.

5.1.6 Design af D/A-konverter

Rasmus skal huske at skrive at der ikke er fokus på DAC i afgrænsning

[4]Reduced instruction set computer

For at trommemaskinen skal kunne afspille de valgte waw-filer i en højtaler er det nødvendigt at omdanne waw-filerne til et analogt signal. Da der anvendes lydfiler i waw-formatet er det ikke nødtvendigt med nogen form for dekodning af filerne, og der kan blot nøjes med en D/A-konverter. Da dette projekt ikke lægger specielt meget op til analog elektronik, og da der ønskes en forhold realistisk men hurtig gengivelse af de digitale lydfiler, vælges der at anvende en allerede eksisterende D/A-konverter i lydmodulet.

Da lydfilerne er samplet med en bitdybde på 16-bit, og da det er lydfiler der arbejdes med, falder det logiske valg på D/A-konverteren TDA1541A [11], givet de D/A-konvertere der er til rådighed. En TDA1541A er en ofte anvendt D/A-konverter i digitalt Hi-Fi-udstyr og egner sig derfor godt til dette projekt.

TDA1541A'en har tre måder at operere på. Fælles for dem alle er at det digitale input læses serielt, og at afspilningsfrekvensen bliver styret af en bitclock "BCK". Derudover er det altid muligt at afspille to kanaler (stereo), i en bitdybde på max 16 bit. D/A-konverterens måde at operere på styres ved hjælp af et enkelt ben på IC'en "OB/TWC", ved at sætte det til hhv. 5 V, 0 V eller -5 V.

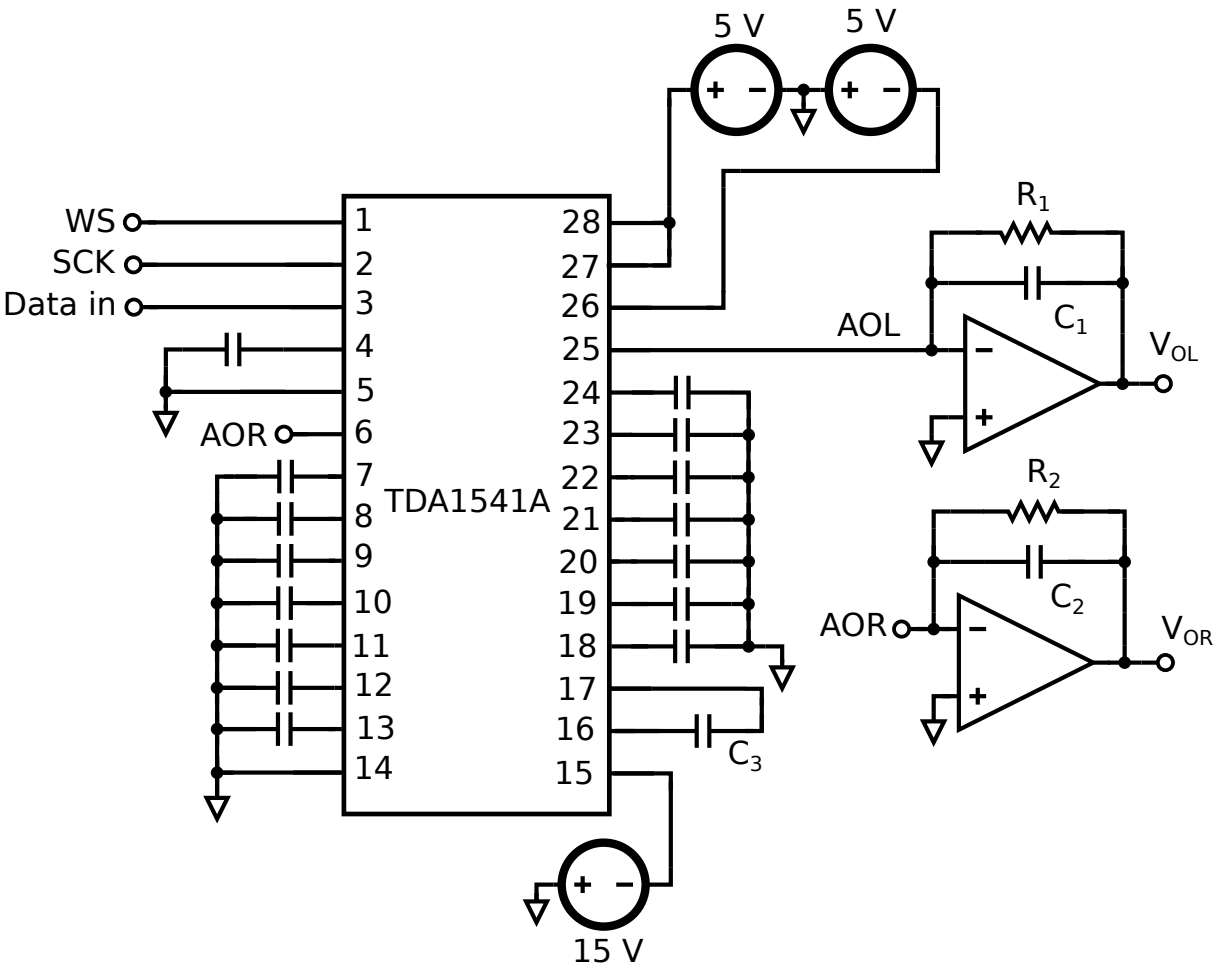
Første mulighed er at styre starten på et nyt "ord", ved at benytte et ben som "latch enable". "Latch enable"sættes høj i en clockcyclus for at markere starten på et nyt "ord". I denne tilstand sendes hver stereokanal på to separate ben. Dog kan D/A-konverteren i denne tilstand kun behandle data representeret som forskudt binær[5].

Anden og tredje mulighed er ens på nær hvordan input-data representeres binært (enten forskudt eller i to's kompliment). I disse tilstande bliver input-data læst på enkelt ben, styret med et "word select-signal. "Word select"sættes lav når data sendes på venstre kanal og høj når data sendes på højre. Med data representeret som to's kompliment, er D/A-konverterens input ækvivalent med kommunikationsstandarden I^2S.[6] Denne standard er beskrevet nærmere i afsnit 2.5.2.

Da WAV-filer i forvejen er i to's kompliment, vælges D/A-konverteren til at styres med "word select"i to's komplimet, d.v.s. I^2S.

For at anvende D/A-konverteren i denne tilstand skal TDA1541Aen opkobles som vist på figur 5.5 jævnfør databladet [11]

[5] En måde at representere binære tal på, hvor 00...0 svarer til den største negative værdi.
[6] Her svarer bitclocken "BCK"til "Serial Clock"SCK.



Figur 5.5: TDA1541A opkoblet til I²S-format

Da de fleste ben på IC'en enten går til DC eller er afkoblet, vil de individuelle ben ikke beskrives i dybden.

På begge udgange er der, som det kan ses på figur 5.5, placeret et aktivt førsteordens lavpasfilter, til at fjerne frekvenser over det hørbare frekvensområde. Modstandene R_1 og R_2 er ii databladet specficeret til 1,8 kΩ og kondensatorne C_1 og C_2 er specificeret til 2,2 nF.

Dette giver lavpasfiltret en knækfrekvens på:

$$f_c = \frac{1}{2\pi R_1 C_1} = \frac{1}{2\pi \cdot 1,8\ \text{k}\Omega \cdot 2,2\ \text{nF}} = 40,2\ \text{kHz} \tag{5.1.1}$$

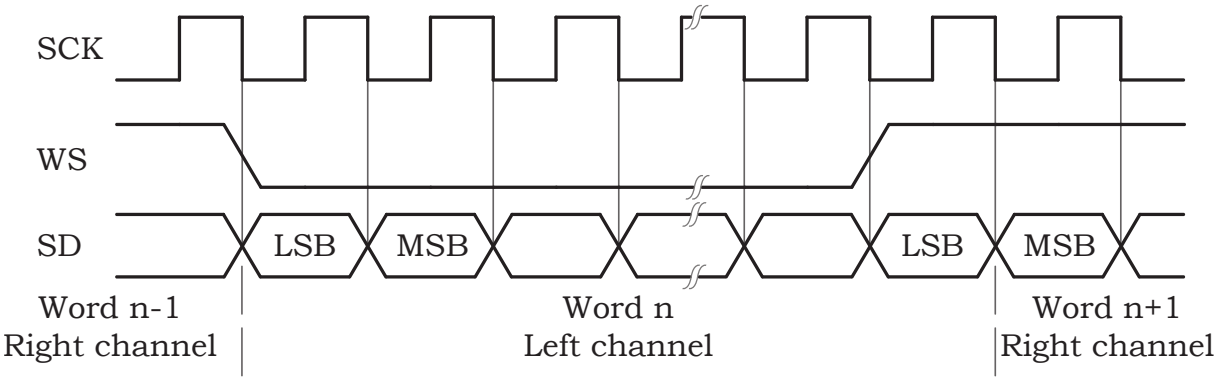
Dette virker meget rimeligt da 40 kHz anses for at ligge tilpas meget over 20 kHz til at filtret ikke har en arkant effekt på det ønskede frekvensområde.

Kondensatoren C_3 er specificeret til 470 pF og de 15 resterende afkoblingskondensatore er sat til 100 nF.

5.1.7 i2sOut

Eventuelt meget flot moduldiagrammer af I2S

Med valget af D/A-konverter medfører dette at udgangstrinet skal overholde I²S-standarden[7]. På figur 5.6 genses opbygningen af I²S-standarden.



Figur 5.6: Opbygning af I²S-standard

For at overholde standarden er det nødvendigt at fremstille de to kloksignaler, SCK(Serial Clock) og WS(Word Select).

På en periode af WS-signalet sendes der ét ord på begge kanaler. Dette medfører at WS skal have samme frekvens som trommelydene er samplet med for at trommelydene afspilles korrekt. Igennem en WS-periode skal der sendes et 16-bits ord på både venstre og højre kanal. hvilket giver S CK en frekvens på

$$f_{SCK} = 2 \cdot 16 \cdot f_{WS} = 2 \cdot 16 \cdot 44,1\text{kHz} = 1,41 \text{ MHz} \tag{5.1.2}$$

tidsperioden for dette clock-signal vil derfor være:

$$T_{SCK} = \frac{1}{f_{SCK}} = \frac{1}{44,1\text{kHz}} = 708,62 \text{ ns} \tag{5.1.3}$$

Clock-signalet fra Papilioens oscilator har en frekvens på 32 MHz, hvilket giver den en tidsperiode på $\frac{1}{32\text{MHz}} = 31{,}25$ ns.

Da forholdet mellem disse to clock-signaler giver $\frac{708{,}62 \text{ ns}}{31{,}25\text{ns}} = 22,67$ er det ikke muligt at danne den ønskede SCK ud fra et antal clock-cyclusser af clock-signalet fra Papilioen.

Derfor er der valgt at resample Trommelydene med en samplingsfrekvens på 50 kHz. Dette giver BCK en frekvens på

$$f_{BCK} = 2 \cdot 16 \cdot 50\text{kHz} = 1{,}6\,\text{MHz} \tag{5.1.4}$$

hvilket giver en tidsperiode på

$$T_{BCK} = \frac{1}{1{,}6\,\text{MHz}} = 625\text{ns} \tag{5.1.5}$$

[7]Se afsnit 2.5.2 for dybere information.

Det nye forhold bliver derved

$$\frac{625 \text{ ns}}{31,25 \text{ ns}} = 20 \tag{5.1.6}$$

Faktoren mellem disse to clock-signaler er 20, hvilket betyder at den eksisterende occilator skal geares ned med en faktor 20. Hver gang der optræder en stigende kant på occilatorsignalet lægges der én til en variable. Når denne variable opnår en predefineret værdi nulstilles den og SCK-signalet inverteres, hvilket medfører at den predefineret værdi skal være 10, fordi I^2S-standarden læser bits på stigende signalkanter så SCK-signalet skal inverteres to gange. Samtidig benyttes en anden variable til at holde styr på hvornår der skal skiftes kanal. Denne tælles op til 16 gange SCK-signalet, hvorefter WS-signalet inverteres og variablen nulstilles.

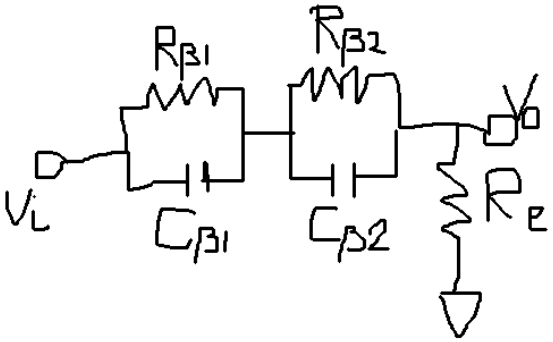
Som det ses på figur 5.6 er den første bit i hvert ord den mindst betydningsfulde bit fra det foregående ord. Da dataen bliver leveret parallelt til modulet benyttes der en buffer til midlertidig opbevaring af ordet der afspilles. Inden denne buffer opdateres til et nyt ord gemmes den mindst betydningsfylde bit i en variable så den er klar til skift i WS og derved det nye ord. Der benyttes herefter en 16:1 MUX til at konvertere den parallele data til seriel data. Denne MUX er drevet af SCK-variablen.

5.1.8 Kobling mellem SRAM og D/A-konverter

Fodring af data Selvom der ikke kan sættes en finger på I^2S-undgangen, er trommemaskinen en smule kedelig uden at have data der kan afspilles. De fire valgt trommelyde ligger på Papilioens SRAM, men fordi SRAMet kun kan gemme én byte pr. adresse, er det nødvendigt at opdele hvert trommesample i to. En af trommemaskinens funktioner er at kunne afspille én trommelyd men den skal også kunne afspille flere trommelyde på samme tid. Da der er tale om lyde, betyder dette at lydenes amplityder skal adderes. Når dette gøres binært giver dette mulighed for enten overflow eller underflow, hvis summen af samplene ligger uden for et 16 bits ord.

5.2 Sequencer

Som nævnt tidligere, er det nødvendigt for den endelige trommemaskine, at kunne holde en rytme og afspille lyde på et bestemt tidspunkt. For at styrer dette er der brug for nogle input og output. Med hensyn til input, skal dette delmodul styres af en række knapper, med forskellige funktioner. For det første skal det være muligt at vælge mellem forskellige sekvenser, som man gerne vil afspille. Derudover skal det være muligt at vælge mellem forskellige trommelyde, da det ellers vil være en forholdsvis kedelig trommemaskine. Yderligere skal der være knapper til at styre hastigheden, som en sekvens afspilles med. Til sidst skal der være knapper for de enkelte dele af sekvensen, så det kan bestemmes hvornår trommelyde skal afspilles.



Figur 5.7: Placeholder for blokdiagram

Some math:

Da der sidder en intern clock i Papilio'en bruges denne til at holde tempoet. Denne clock har en frekvens på 32 MHz og skal derfor sænkes i hastighed for at holde rytmen for trommemaskinen. Til at starte med tages der udgangspunkt i 100 BPM. Først omregnes det til beats per sekund, hvorefter det divideres med fire da der er fire beats på en sekvens. Derefter sammenlignes dette med Papilioens clockfrekvens, for at se forholdet mellem dem.

$$\frac{32\,\text{MHz}}{\frac{100}{60 \cdot 4}} = 76,8 \cdot 10^6 \tag{5.2.1}$$

Dette medføre at Papilioens clock skal tælle til $76,8 \cdot 10^6$ for at komme igennem en sekvens. Da det ønskes at sequenceren kan opfange lyd i realtid er der blevet undersøgt hvor præcist et signal skal værre, for at det passer med præcisionen for en trommeslager. Det er her blevet besluttet at 1 ms er passende præcist for dette. Det er derfor nødvendigt at beregne hvor lang tid det tager at afspille en sekvens.

$$\frac{76,8 \cdot 10^6}{32\,\text{MHz}} = 2{,}4\,\text{s} \tag{5.2.2}$$

Da det ønskes at trommemaskinen skal kunne gå ned til en hastighed på omkring 50 BPM udføres de følgende beregninger på 4,8 s. For at gøre det nemmere at programmere i VHDL, er der taget udgangspunkt i den nærmeste to'er potens, som i dette tilfælde er 4096.

$$\frac{4,8\ \text{s}}{4096} = 1,17\ \text{ms} \tag{5.2.3}$$

Det vil sige at når hastigheden for sequenceren er 50 BPM vil der være 1,17 ms mellem sekvensens dele, og dette antages at være tilstrækkeligt. For 60 BPM er dette krav opfyldt da der er 0,97 ms mellem de enkelte dele i sekvensen, hvilket medføre at højere hastigheder også vil opfylde dette krav.

For et få en clock til at passe med de ønskede BPM, laves en tæller, som skifter et signal når der nås til at bestemt tal. Dette signal bruges så som clock i en anden process. Ud fra de tidligere

oplysninger kan det nu beregnes hvad der skal tælles til, ved en hastighed på 100 BPM. Dette medføre naturligvis at der skal tælles til det dobbelte for 50 BPM og det halve for 200 BPM.

$$\frac{76,8 \cdot 10^6}{4096} = 18750 \tag{5.2.4}$$

For at opsummere strukturen for sequenceren tages der først udgangspunkt i den interne clock på 32 MHz, som tæller til 18750 hvorefter der stiftes et signal. Dette signal bruges som clock'en for en process, der skal hold styr på realtid, hvis man gerne vil holde rytmen selv. Processen der holder styr på realtid har igen en tæller, til at holde styr på hvor langt i processen man er. Når denne tæller når 4096 vil der være gennemgået en fuld sekvens og tælleren nulstilles. Hvis der ikke ønskes at bruge realtid kan tælleren for realtid stagid tages i brug, men der ses kun på de 4 MSB, da disse vil være en 16. del af sekvensen.

Accepttest 6

7 Diskussion

Konklusion 8

Perspektivering 9

Ordliste

BPM Beats per minute. generelt andvent måleenhed for musiktempo. Antallet af taktslag per minut. Eksemplelvis antallet af fjerdedele på et minut i taktarten $\frac{4}{4}$.. *se* takt, 2

D/A-konverter Elektronisk enhed der har til formål at omdanne et digitalt signal til et analogt.. 7

FPGA Field Programable Logic Array. Integreret kredsløb af konfigurærbar logic, bygget af logic-blokke eller "celler" . 10

I²S Inter-IC Sound. Seriel bus til interfacing mellem digitalt lydudstyr. . 10

KCPSM6 Processer til en picoBlaze lavet specifikt til Spartan-6 og lign. FPGA'er. *se* picoBlaze, 19

picoBlaze 8-bits soft mikrocontroller designet til FPGA'er fra Xilinx. *se* FPGA, 18

RAM Random Access Memory. hukommelse der både kan læses fra og skrives til. 15

Sequencer Elektronis apparat, der bruges til indspilning/afspilning af musik, ved at inkode lydinformation ind i en sekvens opdelt i mindre dele. En essentiel del af en trommemaskine, som bruges til at bestemme hvornår en bestemt tromme skal afspilles.. 3

SRAM Static Random Access Memory. Statisk RAM. . *se* RAM, 15

Takt Musik opdeles normaltvist i takter, hvis længde er givet ud fra en taktart, eksempelvis $\frac{4}{4}$. . 2

TDA1541A 16 bits stereo D/A-konverter designet til brug i Hi-Fi-aparater. Dette er D/A-konverteren anvendt i dette projekt.. *se* D/A-konverter, 21

WAV / **Wave** Waveform Audio File Format. Filformat til lyd, der foruden en header består af ukomprimeret data.. 7

Bibliografi

[1] David M. Alter. *sing PWM Output as a Digital-to-Analog Converter on a TMS320F280x Digital Signal Controller*. Tek. rap. url: http://www.ti.com/lit/an/spraa88a/spraa88a.pdf. Texas Instruments, 2008.

[2] MIDI Manufacturers Associatio. *An Introduction to MI*. Set d. 18/3/17. URL: https://www.midi.org/images/easyblog_articles/43/intromid_20160320-111631_1.pdf.

[3] Dr. Sebastian Anthony Birch. *The MIDI Physical Layer*. Set d. 18/3/17. URL: http://www.personal.kent.edu/~sbirch/Music_Production/MP-II/MIDI/midi_physical_layer.htm.

[4] Ken Chapman. *PicoBlaze for Spartan-6, Virtex-6, 7-series, Zynq and UltraScale Devices (KCPSM6)*. Xilinx. 2014.

[5] Pong P. Chu. *FPGA programming by VHDL examples*. Wiley, 2008.

[6] Date-compression. *Theory*. Sidst set: 22/02/17. URL: http://www.data-compression.com/theory.html.

[7] Holger Hennig, Ragnar Fleischmann, og Theo Geisel. „Musical rhythms: The science of being slightly off". I: *Physics Today* (2012). URL: http://physicstoday.scitation.org/doi/full/10.1063/PT.3.16

[8] ISSI. *IS61WV2048ALL IS61/64WV2048BLL 2M X 8 HIGH-SPEED CMOS STATIC RAM*. Set d. 1/4/17. 2014.

[9] Musikpedia. *Puls og tempo*. Sidst set: 17/02/17. URL: http://www.musikipedia.dk/puls-og-tempo.

[10] Musikpedia. *Takt og underdeling*. Sidst set: 17/02/17. URL: http://www.musikipedia.dk/takt-og-underdeling.

[11] Philips. *TDA1541A datasheet*. Sidst set: 14/04/2017. 1991. URL: http://pdf.datasheetcatalog.com/datasheet/philips/TDA1541A.pdf.

[12] Philips Semiconduct. *I2S bus specification*. Set d. 18/3/17. 1996. URL: https://web.archive.org/web/20060702004954/http://www.semiconductors.philips.com/acrobat_download/various/I2SBUS.pdf.

[13] wikipedia. *A typical hardware setup using two shift registers to form an inter-chip circular buffer*. Sidst set: 26/4/2017. URL: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#/media/File:SPI_8-bit_circular_transfer.svg.

[14] Wikipedia. *Akai MPC2000*. Sidst set: 01/03/17. URL: https://commons.wikimedia.org/wiki/File:Akai_MPC2000.jpg.

[15] Wikipedia. *Metre (music)*. Sidst set: 25/02/17. URL: https://en.wikipedia.org/wiki/Metre_(music).

[16] Young-Engineering. *A-law and μ-law companding*. Sidst set: 22/02/17. URL: http://www.young-engineering.com/docs/YoungEngineering_ALaw_and_MuLaw_Companding.pdf.