0.1 Arbejdsblad

Hej Søren

Vi kunne godt tænke os at få respons på beskrivelse af accepttest, test af delmoduler og accepttesten. Af den grund har vi også vedlagt problemforulering/afgrænsning og kravspecificationen.

Hvis du keder dig må du også gerne læse design af kontrolmodulet og design af sequencer, men på grund af de mange antal sider, og det faktum at der ikke er tid til at lave mange ændringer, forventer vi ikke respons på det. Kapitel 5, hvor designafsnittet står skrevet, kan findes efter appendiks.

Indholdsfortegnelse

Problemformulering 1

1.1 Projektafgrænsning

En trommemaskine 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.

Der vælges at alt rytmik forgår i taktarten $frac44$, da denne er den mest anvendte inden for musik.

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"prototype samt at gøre det nemmere at fejlfinde og viderebygge projektet, vælges det at implementere projektet på konfigurerbart hardware, navnligt en FPGA som beskrevet i **??**[1].

I en trommemaskine vil det være nødvendigt at konvertere digitale signaler til analoge hvilket kræver en D/A-konverter, men da projektets fokus ligger på digital design 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 trommelyde og sekvenser der kan gemmes, i betragtning af hvad en reel gangbar trommemaskine bør have.

Istedet vil fokuset 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 lydsamples.

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 trommelyde frem for at fremstille dem selv, grundet at de anvendte lyde ikke har stor indflydelse på resten af projektet. Der kunne benyttes hvilke som helst lyde, uden at noget betydelig effekt på resten af trommemaskinens funktioner.

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

[1] FPGA'en vælges frem for PSoC, da PSoC'en virker til primært at være designet til brug som microcontroller, hvor fokusset ofte ligger på dens microprocessor med lidt konfigurerbar hardware ved siden af. Da det ønskes at lave så meget som muligt på parralelt hardware vælges Papilio Duo'en istedet.

1.2 Problemformulering

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

Hvordan designes og konstrueres en digital trommemaskine, hvorpå man kan indpille en rytme, som kan afspilles via en sequencer?

Kravspecifikation 2

2.1 Funktionelle krav

1. Skal gemme en rytme i en sequncer, med inputs fra en sampler og en step-sequencer.
2. Skal kunne tage imod input fra en sampler uden af gemme dette i sequnceren.
3. Skal have 4 forskellige trommelyde der kan afspilles.
4. Skal kunne afspille 4 trommelyde på samme tid.
5. Skal kunne indstille tempoet mellem 50 BPM og 200 BPM inden for ± 1 BPM.
6. Skal kunne gemme og genfinde 4 sekvenser i hukommelse.
7. Skal kunne afspille sequencerens rytme som et audio-output

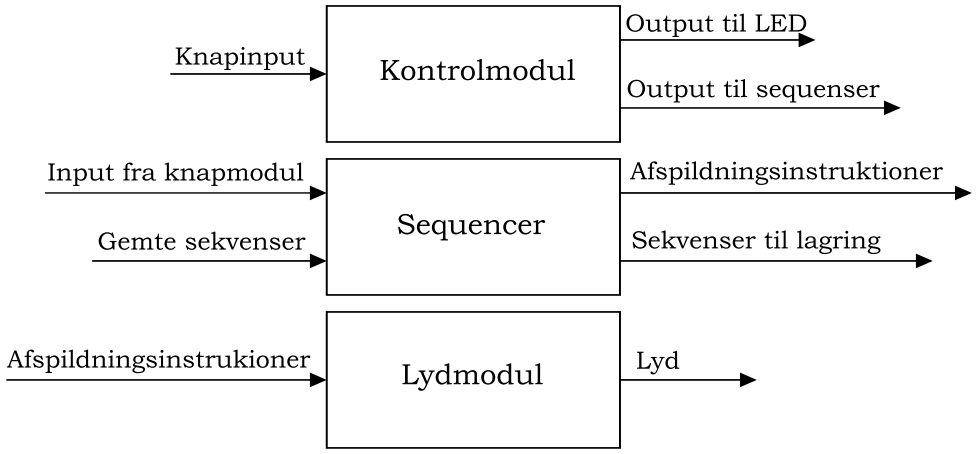
2.2 Krav til delmoduler

Ud fra de funktionelle krav, er funktionaliteterne blevet inddelt i tre overmoduler med hver deres inputs og outputs. Dette er gjort for at kategoriserer disse funktionaliteterne, således de bliver mere overskueligt. De er inddelt i følgende:

Kontrolmodul Er input delen til den fysiske verden, altså vil dette være mellemleddet mellem brugen og trommemaskinen. Det vil primært bestå af knapper og LED'er, som skal gøre det muligt for brugeren at intrigerer med trommemaskinen, samtidig med give brugeren indikation af trommemaskinens status.

Lydmodul I modsatte ende vil lydmodulet være. Dette modul skal stå for alt hvad der har at gøre med lyd og har til opgave at producerer lyd, på baggrund af instruktioner fra sequenceren.

Sequencer Mellem disse to moduler skal sequenceren være, og kommer til at fungerer som bindeled. Den skal tage i mod inputs fra kontrolmodulet, og alt efter inputtet stå for, at gennem sekvenser af lyde, vælge hvilke lyde der skal afspilles og give instruktioner til lydmodulet.



Figur 2.1: Moduldiagram over input og output til de forskellige moduler

Krav til Kontrolmodul

1. Skal tage imod brugerinput til:
    a) Indstilling af BPM
    b) Valg af trommesample
    c) Valg af, gemme, hente 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.

Krav til sequencer

1. Skal kunne blive programmet ved input fra kontrolmodulet
2. Skal kunne indspillede trommeslag via sampler, skal ligge inden for 1,6[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.

Krav til lydmodul

1. Skal kunne afspille trommelyde med en samplingfrekvens på mindst 44,1 kHz
2. Skal konvertere lyden fra et digitalt signal i 16 bits opløsning til et analogt outputsignal.
3. Skal afspille op til 4 forskellige lyde samtidig.
4. Skal kunne repræsentere summen af fire trommelyde i 16 bits opløsning.

2.3 Beskrivelse af accepttest

Beskrivelse af accepttest til funktionelle krav 1

Denne test udføres ved at indkode en specifik sekvens i både sampleren og i step-sequenceren. Hvis afspilningsinstruktionerne til lydmodulet er magen til den indkodet sekvens og gentages hver sekvens, må sequenceren kunne gemme en sekvens. Sekvensen indkodes således at outputet kendes med sikkerhed. Afspilningsintruktionerne måles på et oscilloskop hvorefter timingen analysers. Disse skal gentages periodisk og må ikke afvige mere end ét millisekund.

Beskrivelse af accepttest til funktionelle krav 2

For at teste dette krav ses der på, om der kan afspilles en trommelyd når der trykkes på en realtidsknap, uden at denne bliver gemt i en sekvens og afspillet igen. Hvis dette er tilfældet anses kravet som værende opfyldt.

[1] Denne værdi er valgt da den er en faktor ti mindre end den gennemsnitlige afvigelse målt i [1]

Beskrivelse af accepttest til funktionelle krav 3

Her tændes der for hver trommelyd enkeltvis, og afspilningsinstruktionerne aflæses. Hvis der måles fire forskellige signaler må den kunne indeholde 4 trommelyde der er uafhængige af hinanden, og derfor forskellige. Afspilningsintruktionerne måles på et oscilloskop hvorefter timingen analyseres. Her må ingen af signalerne være tændt på samme tid, da dette kan indikere at en tromme optræder to gange.

Beskrivelse af accepttest til funktionelle krav 4

Her afspilles alle lyde på samme tid, under antagelsen at funktionelle krav nummer 3 er bestået. Afspilningsintruktionerne måles igen på et oscilloskop hvorefter timingen analysers, men denne gang skal alle signalerne være tændt, og de må ikke afvige med mere end et millisekund fra hinanden.

Beskrivelse af accepttest til funktionelle krav 5

Da sequenceren spiller i $\frac{4}{4}$ betyder dette at der optræder 4 taktslag ved hver sekvensgennemgang. Testen udføres ved at indkode en sekvens bestående af et enkelt slag. Længden af en sekvens kan derefter udregnes som afstanden mellem trommelydens transient, og transienten af den gentagede trommelyd på den efterfølgende takt. Denne tid bruges til at omregne til en sekvens' tempo i BPM, og denne værdi sammenlignes med den BPM-værdi, der står på brugerinterfacets BPM-indikator. Dette gøres både ved BPM-værdien 50 og 200.

Beskrivelse af accepttest til funktionelle krav 6

Beskrivelse af accepttest til funktionelle krav 7

Først programmeres trommemaskinen til af afspille trommelyde i en specifik rytme. Herefter aflæses udgangen på D/A-konverteren med et oscilloskop, og afstanden mellem transienter aflæses. Det aflæste på oscilloskop skal være ens med den indkodet sekvens for at kravet kan anses som bestået.

Test af delmoduler 3

I dette kapitel vil der redegøres for de tests der er udført for at se om delmodulerne overholder deres individuelle krav som de står i afsnit 2.2.

3.1 Test af kontrolmodul

Kontrolmodulet har til formål at håndtere input fra brugeren, og indikere de forskellige indstillinger. I den forbindelse skal der testes, hvorvidt de forskellige signaler, der styrer disse funktionaliteter, er sat korrekt op. For at teste dette, ses der blot på sammenhænget mellem de input der bliver givet og de resulterende output. Der startes med at se på taktindikatoren, da denne ikke har brug for input til at styre hvilke LED'er der skal lyse. Her ses det at alle LED'erne kan lyse individuelt ud fra et signal, som opdateres med hensyn til BPM'en. For at opdatere BPM'en kan der bruges to knapper, for henholdsvis at skrue op eller ned for hastigheden. Når en af knapperne holdes nede, bliver ændres hastigheden, som taktindikatorne blinker med, og derfor vides det at processen virker. Dette kan også ses på de tre 7-segmentsdisplay, som viser den beregnede BMP. Denne passer også overens med beregningerne, da den kan gå fra 50 til 200 og starter i 100.

Da det vides at processen, der opdatere LED'erne virker korrekt, ses der på om diverse input kan bruges til at opdatere signaler hertil. Her ses der først på step-sequencerens knapper og LED'er, da hver knap har en tilhørende LED. Det kan observeres at tryk på de individuelle knapper får den dertilhørende LED til at lyse, og derfor virker efter hensigten. Til sidst set der på knapperne, der håndtere valg af trommer og sekvenser. Her skal LED'erne opdateres, så kun en kan være tændt at gangen, da der ikke skal opdateres flere signaler af gangen. Når der trykkes på en knap her kan det ses at kun den LED, som høre til denne knap lyser op, og virker derfor også efter hensigten. Til sidse kan det ses at alle LED'erne for step-sequenceren slukker, når der trykkes på slet knappen.

Med hensyn til kravet om at sende instruktioner videre til sequencer-moduler, kan dette nemt løses. Det skyldes at begge delmoduler, er styret af det samme VHDL-modul, og har derfor har adgang til samme interne signaler.

Opsummering

For at give et bedre overblik over hvilke krav der er opfyldt, kan der ses på tabel 3.1.

Krav nr.	Beskrivelse	✓/ ÷
1	Skal kunne tage imod diverse brugerinput	✓
2	Skal kunne vise trommemaskinens status på en række LED'er	✓
3	Skal kunne sende instruktioner til sequenceren	✓

Tabel 3.1: Tabel over delkrav til kontrolmodulet

3.2 Test af sequnecer

Test af krav 2 kan findes i appendiks C.

Krav 1. - Skal kunne blive programmet ved input fra kontrolmodulet. På baggrund af koden og observationer, kan det konstateres at indgangssignalerne resulterer i det ønskede respons. Eksempelvis når step-sequenceren programmeres til at afspille lyde i en sekvens, afspilles disse også her.

Krav 2 - Skal kunne indspillede trommeslag via sampler, skal ligge inden for 1,6 ms fra brugerens reelle taktslag. Resultaterne af testen for dette viser, at gennemsnittet ved ti målinger, var en tid på 685,1 µs og den største tid der blev målt var 1137 µs.

Krav 3. - Skal sende instrukser om hvilke lyde der skal afspilles og hvornår. På baggrund af koden og observationer ved undersøgelse af krav 2. konstateres det at instrukserne er korrekte både for tid og lyd.

Krav 4. - Skal kunne gemme og hente en sekvens fra et stykke hukommelse. På baggrund af koden og observationer af kontrolmodulet konstateres det, at resultaterne er som ønsket. Når man eksempelvis trykker på knapper beregnet til at styre sekvenserne, lyser LED'erne til at vise skevens og step-sequenceren korrekt i forhold til det ønskede.

Krav 5. - Skal kunne ændre den tid det tager at køre gennem en sekvens i henhold til indtastede BPM. På baggrund af koden og observationer af kontrolmodulets LED'er, samt 7-segments displayet, konstateres det at instruktioner fra kontrolmodulets knapper, giver mulighed for at ændre tempoet og dermed tiden det tager, at kører en sekvens igennem. Ligeledes kan det ses at gennemsnitstiden i C.2 for 100 BPM også er næsten halvdelen af tiden for 50 BPM.

Opsummering

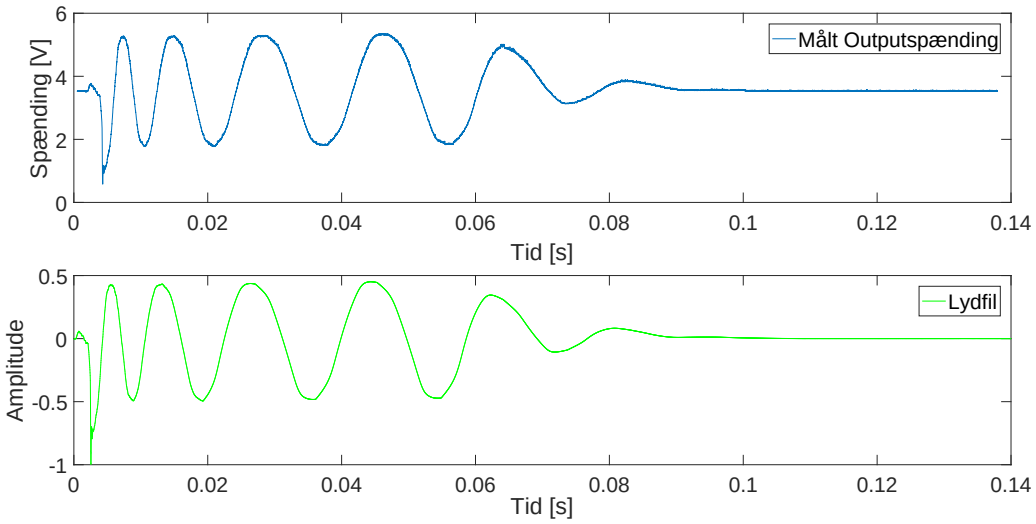
Krav nr.	Beskrivelse	✓/ ÷
1	Programmet ved input fra kontrolmodulet	✓
2	Trommeslag skal ligge indenfor 1,6 ms	✓
3	Instrukser om hvilke lyde der skal afspilles og hvornår	✓
4	Gemme og hente en sekvens fra et stykke hukommelse	✓
5	Ændre den tid det på en sekvens i henhold til indtastede BPM	✓

Tabel 3.2: Tabel over delkrav til sequencer

3.3 Test af lydmodul

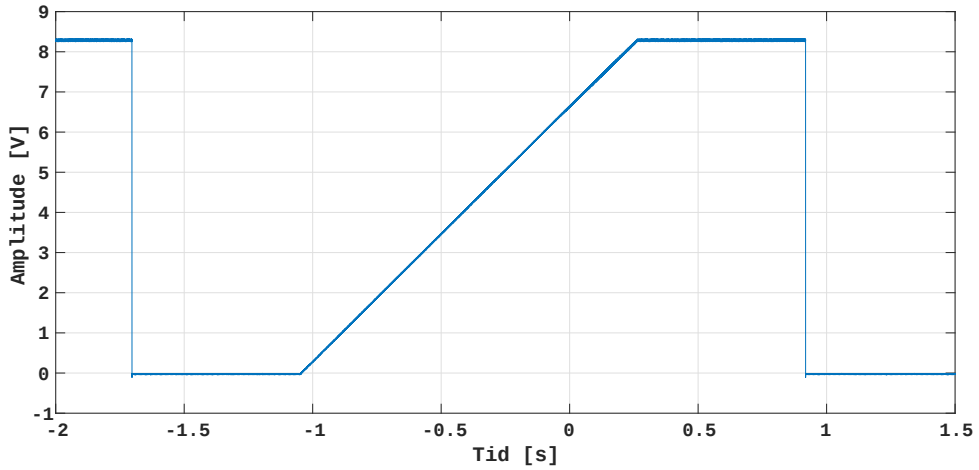
De fulde tests udført på lydmodulet kan findes i appendiks A og B.

Krav 1. Dette krav anses som at være opfyldt på baggrund af testen beskrevet i appendiks B. Da de originale trommelyde er samplet med 50 kHz, og de målte trommelyde bliver afspillet ligeså hurtigt (illustreret på figur 3.1), må lydmodulets output blive afspillet med samme samplingsfrekvens.



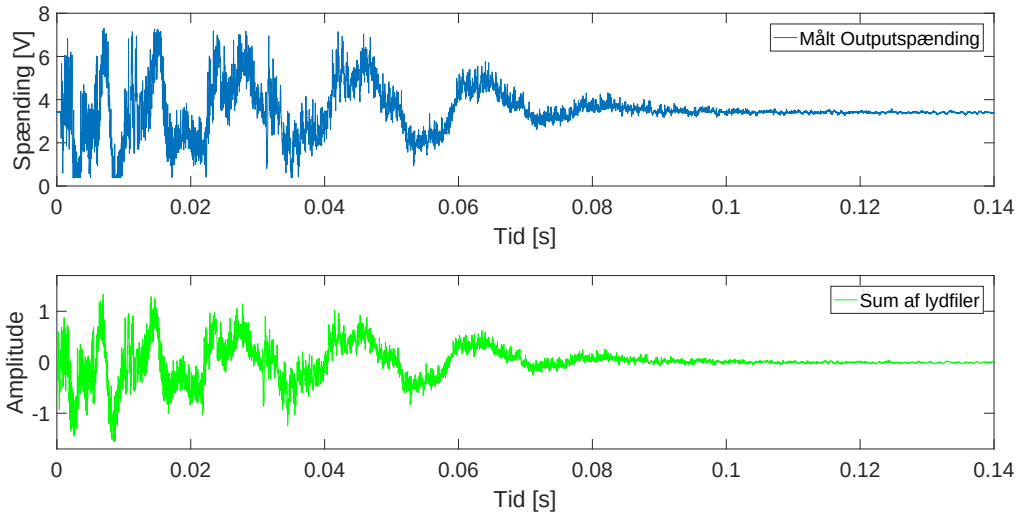
Figur 3.1: Måling af stortromme samt den originale lydfil. Uddrag af fra appendiks B

Krav 2. Kravet vises at være opfyldt i målerapporten i appendiks A, da det ses at alle 16-bits værdier, sendt ind i D/A-konverteren med I²S, har et tilsvarende analogt output givet som en spænding. Resultatet af denne test kan ses på figur 3.2.



Figur 3.2: Måledata over spændingsniveauerne fra D/A-konverteren.

Krav 3. Kravet anses som at være opfyldt igen på baggrund appendiks B. I appendikset kan det ses at lydmodulet kan afspille 4 forskellige lydfiler. Det kan også ses at er i stand til at outputte alle fire lyde samtidigt hvilket kan ses på figur 3.3.



Figur 3.3: Måling af alle fire lyde afspillet samtidigt samt summen af de fire lydfiler

Krav 4. Summen af lydfilerne, vist nederst i figur 3.3, er normeret med hensyn til den størst mulige 16-bits værdi. Det kan ses øverst på figuren at det målte signal klipper når den skal afspille værdier der ikke kan repræsenteres som 16-bits tal. Dette er også den måde lydmodulet er designet som det kan ses i afsnit **??**. På grund af dette anses dette krav som at være opfyldt.

Opsummering

Igen vises en tabel over hvilke krav der er opfyldt på en overskuelig måde, denne gang på tabel 3.3.

Krav nr.	**Beskrivelse**	✓/ ÷
1	Afspilning med korrekt samplingsfrekvens	✓
2	D/A-konvertering	✓
3	Polyfoni	✓
4	Håndtering af over/underflow	✓

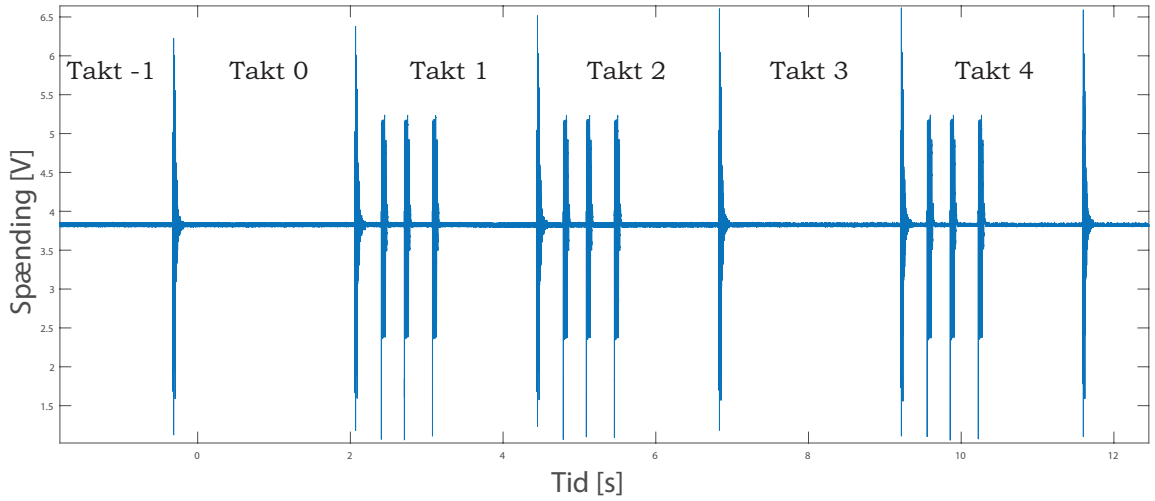
Tabel 3.3: Tabel over delkrav til kontrolmodulet

Eftersom alle delmoduler fungerer efter deres individuelle specifikationer, vil der nu kigges på testene udført på den samlede trommemaskine.

Accepttest 4

I dette kapitel vil der redegøres for accepttestene, der er udført på trommemaskinen for at påvise, at trommemaskinen overholder dens funktionelle krav. De udførte test tager udgangspunkt i testbeskrivelserne i afsnit 2.3. Målerapporterne kan til disse tests kan findes i appendiks D.

Krav 1. *Skal gemme en rytme i en sequncer, med inputs fra en sampler og en step-sequencer.* Dette krav anses som at være opfyldt på baggrund af test D.1. I testen vises det at trommemaskinen er i stand til at tage imod inputs fra sampleren. Da inputtet bliver gentaget i næste sekvens antages det at inputtet er blevet gemt i sequenceren. Dette kan ses på figur 4.1

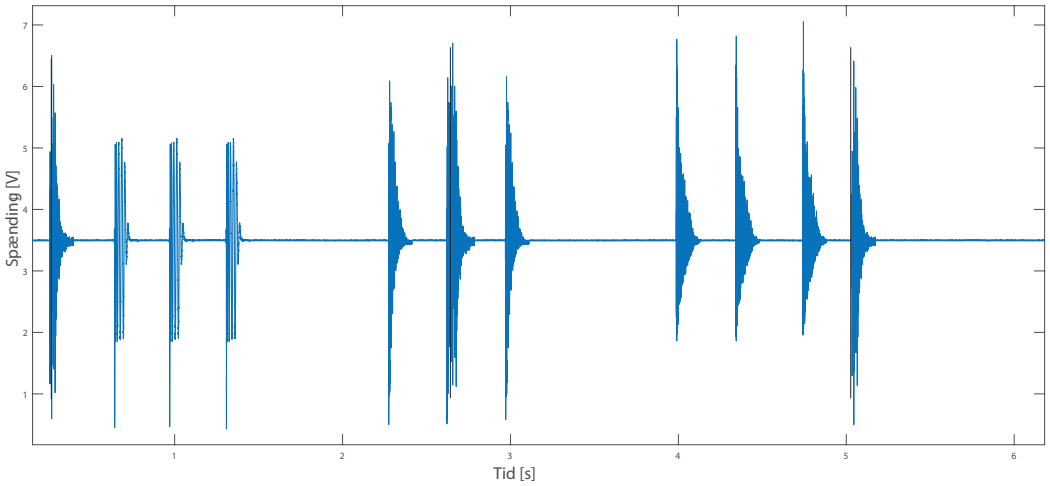


Figur 4.1: Graf over udgangssignalet fra trommemaskinen

Det kan også ses at trommemaskinen kan tage imod input fra step-sequenceren da denne benyttes til at markere starten af en sekvens. Derudover skiftes der mellem forskellige sekvenser, hvilket medfører at udgangssignalet, mellem indikationen, ikke er ens.

Krav 2. *Skal kunne tage imod input fra en sampler uden af gemme dette i sequnceren.*

Kravet til fri tromning er specifikt blevet testet i testen D.2. Heri vises det at trommemaskinen er i stand til at tage imod input fra en sampler og afspille trommelyde herpå, uden at gemme inputtet i sequenceren. Dette kan ses på figur 4.2. Her ses det at udgangssignalet ikke bliver gentaget sekventielt.



Figur 4.2: Tromning uden gentagelse af samplerinputtet

Krav 3. *Skal have 4 forskellige trommelyde der kan afspilles.*

I testen af lydmodulet i appendiks B, vises det at lydmodulet er i stand til at afspille de fire trommelyde, der er lagt ind på trommemaskinen, og dette krav er derfor opfyldt.

Dette krav er også indirekte vist i testen til krav 2, da det kan ses på figur 4.2

Krav 4. *Skal kunne afspille 4 trommelyde på samme tid.*

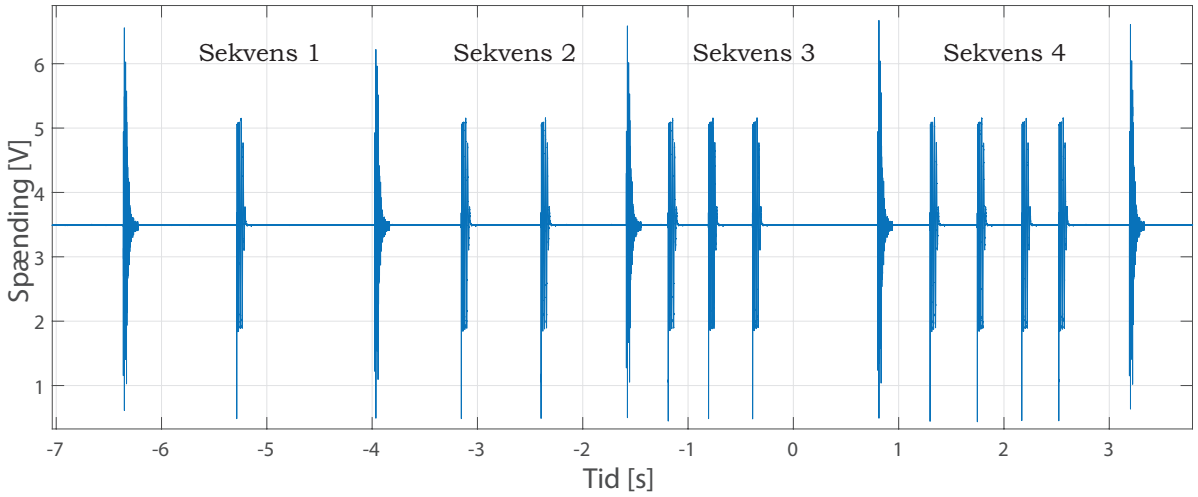
Dette krav er også vist at være overholdt i appendiks B. Der er ikke ændret på måden, hvorpå lydmodulet afspiller lyd efter at trommemaskinen er samlet, og inputsignalerne der tilgås i testen, er de samme som sequenceren bruger til at afspille lyd i virkeligheden. Derfor antages det at tilføjelsen af sequenceren og kontrolmodulet ikke har haft en effekt på lydmodulets polyfoni, og kravet anses derfor som opfyldt.

Krav 5. *Skal kunne indstille tempoet mellem 50 BPM og 200 BPM inden for ± 1 BPM.*

Testen til dette krav står beskrevet i afsnit D.3. Ved indstilling til både 50, 100 og 200 BPM ligger den målte værdi indenfor ±1 BPM. Den målte BPM er også altid større end den viste hvilket stemmer overens med måde BPM-værdien udregnes på FPGA'en, hvor den virkelige BPM-værdi vil ligge sig inden for $\text{BPM}_{målt} \leq \text{BPM}_{rel} < \text{BPM}_{målt} + 1$.

Krav 6. *Skal kunne gemme og genfinde 4 sekvenser i hukommelse.*

Testen til dette krav kan findes i afsnit D.4. I målerapporten ses det at trommemaskinen er i stand til at have fire forskellige indkodede sekvenser, som gemt illustreret på figur D.6.



Figur 4.3: Fire forskellige sekvenser gemt på trommemaskinen

Krav 7. *Skal kunne afspille sequencerens rytme som et audio-output.*

Da alle tests i appendiks D er blevet udført ved at måle på trommemaskinens audio-output, er dette krav indirekte blevet påvist at være opfyldt. At hver trommelyd bliver afspillet korrekt som et analogt signal er vist i dybden i målerapporten i appendiks B.

Opsummering af accepttest

Krav nr.	**Beskrivelse**	✓ / ÷
1	Inputs fra bruger til sequencer	✓
2	Fri tromning	✓
3	Lydfiler	✓
4	Polyfoni	✓
5	Tempokontrol	✓
6	Sekvenslagring	✓
7	audio-output	✓

Tabel 4.1: Tabel over delkrav til kontrolmodulet

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, 3

FPGA Field Programable Logic Array. Integreret kredsløb af konfigurærbar logic, bygget af logic-blokke eller "celler" . 1

I²S Inter-IC Sound. Seriel bus til interfacing mellem digitalt lydudstyr. . 8

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

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

ROM Read only memory. Hukommelse der kun kan læses fra.. *se* RAM

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

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

Test af D/A-konverter og I2S A

Formål

Formålet med dette forsøg er at undersøge hvorvidt Papilio DUOen kan kommunikere med den valgt D/A-konverter via I2S, og om D/A-konverteren kan repræsentere alle værdier i et 16 bits ord som en spænding.

Udstyr

På tabel B.1 ses en tabel over det anvendte forsøgsudstyr.

Instrument	Model	AAU-nummer
FPGA	Papilio-duo 2MB	
Logik analysator	Digilent Analog Discovery 2	2179-04
D/A-konverter	TD1541A	
Strømforsygning	HAMEG HM7042	B1-101-N-7 B1-101-O-6

Tabel A.1: Tabel over anvendt forsøgsudstyr.

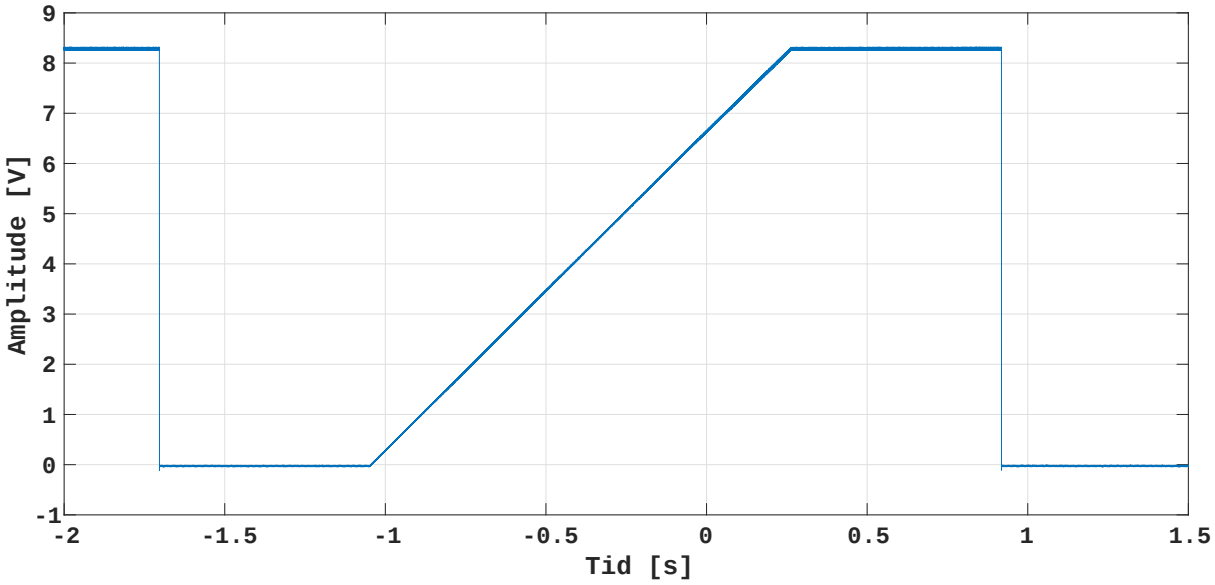
Metode

Der er kodet et VHDL-modul, som skal kommunikere med D/A-konverteren via I2S. Dataet der sendes til D/A-konverteren, skal falde inden for grænserne for et 16 bits ord i 2's kompliment. I decimaltal svarer dette til, at den sendte serielle data skal gå fra -32768 til 32767. Der er i kravspecifikation vedtaget, at outputtet skal gå i klipning hvis værdien overskriver disse grænser. For at teste hvorvidt dette overholdes, tælles en variable op fra -33000 til 33000 hver gang der skiftes fra venstre til højre lydkanal. Denne øvre og nedre grænse er valgt, da den ligger uden for et 16 bits ord i 2's kompliment, hvilket vil medfører over- eller underflow. Hver gang der er blevet sendt en specifik værdi på hver kanal, vil den næste værdi være én størrer. Når variablen når sin øvre grænse på 30000, sættes den igen til -30000.

D/A-konverteren opkobles jævnfor dens datablad[4], hvor dens input bliver danne af Papilioen. I databladet ses det at D/A-konverterens output skal direkte i en operationsforstærker med predefineret tilbagekobling. Forstærkningsgraden, og derved den maksimale udgangsspænding, er derfor ukendt. Der benyttes intet DC-ofset i operationsforstærkeren, så negative spændinger forventes ikke. Operationsforstærkeren medtages som en del af D/A-konverteren, og dette udgangssignal måles og gemmes af logikanalysatoren. Hvis outputdatavariablen overskider grænserne for et 16 bits ord forventes det at ouputtet klipper, og hvis variablen ligger inden for grænsen forvæntes en lineær stignen i D/A-konverterens output.

Resultater

På figur A.1 ses forsøgets måledata. Der ses her at udgangsspændingen klipper ved 0 V og lige over 8 V. Det ses også at stignen i spændingen er lineær mellem den øvre og nedre grænse for klipning.



Figur A.1: Måledata over spændingsniveauerne fra D/A-konverteren.

Konklusion

Det konkluderes at Papilioen kan kommunikere med D/A-konverteren via I2S, og at alle værdier for et 16 bit ord kan repræsenteres som en spænding. Hvis ikke de alle var repræsenteret, ville stigningen på figur A.1 have små ujævnhedere hvor værdien ikke kunne repræsenteres. Det konkluderes også at klipning håndteres som forventet.

Test af lydmodul B

Formål

Denne test går ud på at undersøge om lydmodulet afspiller alle fire lydfiler korrekt, samt at lydmodulet er i stand til at afspille flere filer på samme tid.

Udstyr

Det anvendte forsøgsudstyr i testen kan ses på tabel 2.1

Instrument	Model	AAU-nummer
FPGA	Papilio-duo 2MB	
Digitalt oscilloskop	Digilent Analog Discovery 2	2179-04
D/A-konverter	TD1541A	
Strømforsygning	HAMEG HM7042	B1-101-N-7 B1-101-O-6
Pulldown-modstand	Hjemmelavet	

Tabel B.1: Tabel over anvendt forsøgsudstyr.

Metode

I denne test er FPGA'en programmeret med hele lydmodulet, hvis design er beskrevet i afsnit **??**, ud over at inputtetsignalet fra sequenceren er forbundet til en pulldown-knap, lavet på et brødbræt.

FPGA'ens I²S-output er forbundet til D/A-konverteren TDA1541, der er forbundet som det beskrives i afsnit **??**.

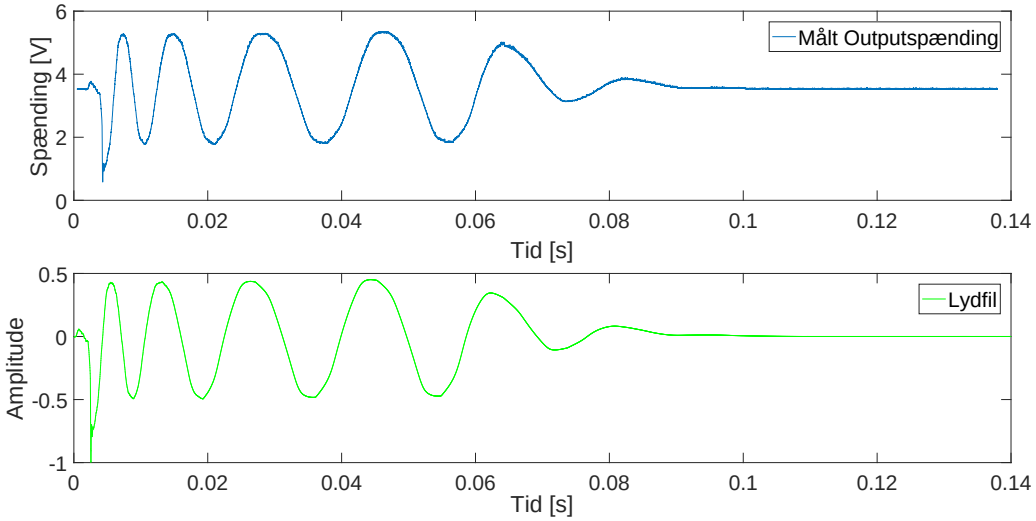
På udgangen af den ene operationsforstærker på D/A-konverteren tilkobles et oscilloskop.

Ganske simpelt går testen ud på at trykke på knappen og måle spændingen på udgangen, over et stykke tid. Da hver lydfil vare 140 ms, måles der i et stykke længere end dette.

Testen udføres fem gange. En gang for hver tromme hvor trommelyden afspilles individuelt og en gang hvor alle fire trommelyde afspilles samtidigt. Andre kombinationer af trommer antages at være unødvendige, da lydmodulet ikke gør forskel på hvilke trommer der spilles, da den i princippet lægger alle fire trommers data sammen hver gang. link til i2s kodeeksempel. Derudover er det når alle fire trommer skal spilles at lydmodulet bruger længst tid på at hente data fra SRAM'ene.

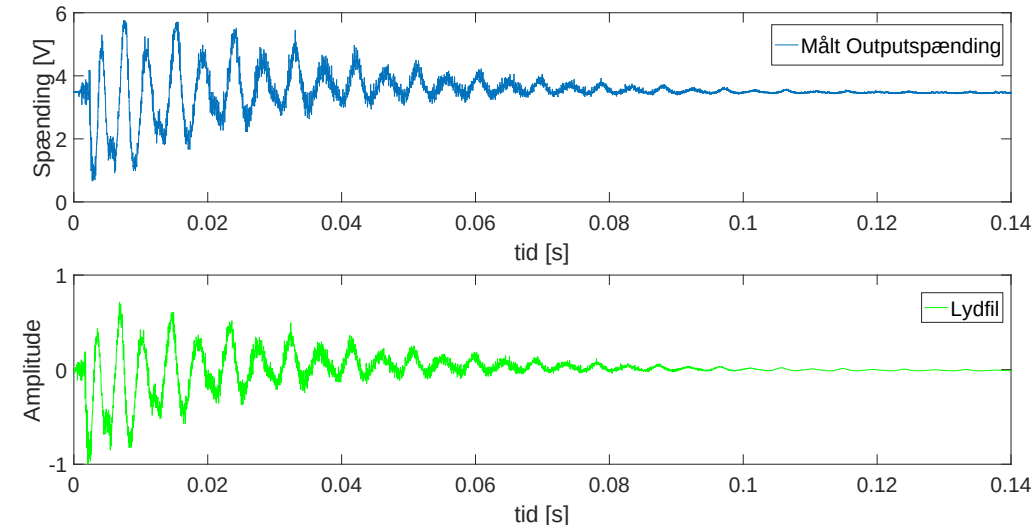
Resultater

De målte outputspændinger er plottet ind i MatLab. Desuden er de originale lydfiler læst ind i Matlab og plottet, for at kunne sammeligne outputputtet på D/A-konverteren med hvad det burde være[1].



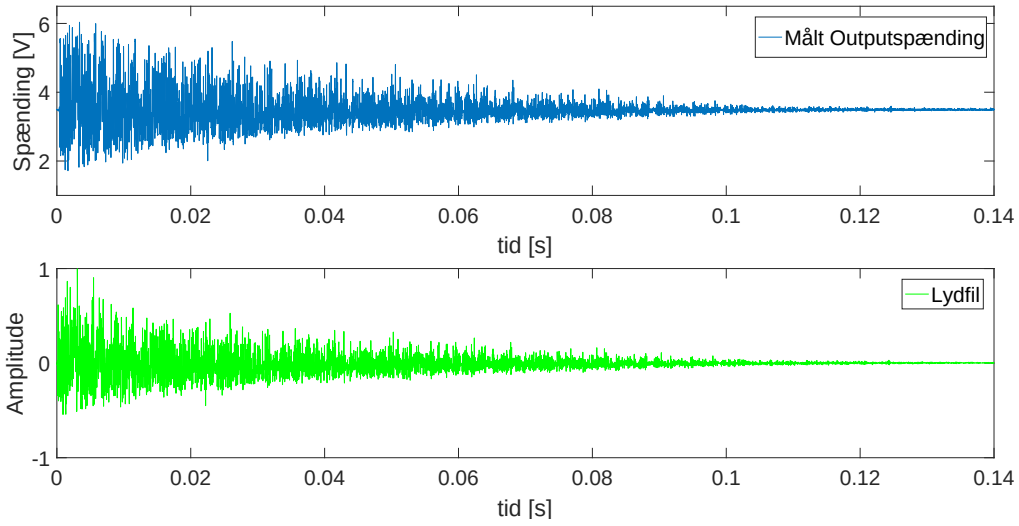
Figur B.1: Måling af stortromme samt den originale lydfil.

Ud over en smule støj ligner det målte signal på figur B.1 meget den originale lydfil.

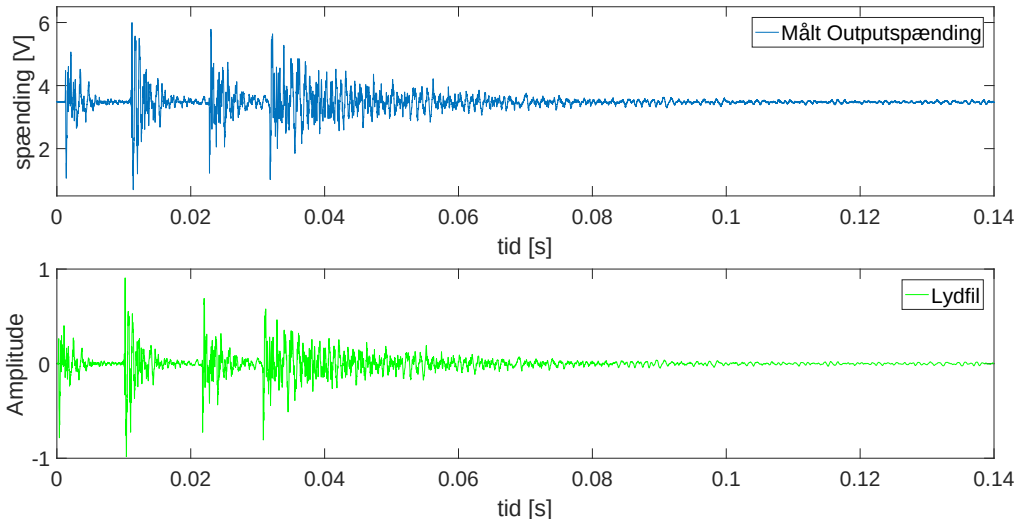


Figur B.2: Måling af stortromme samt den originale lydfil.

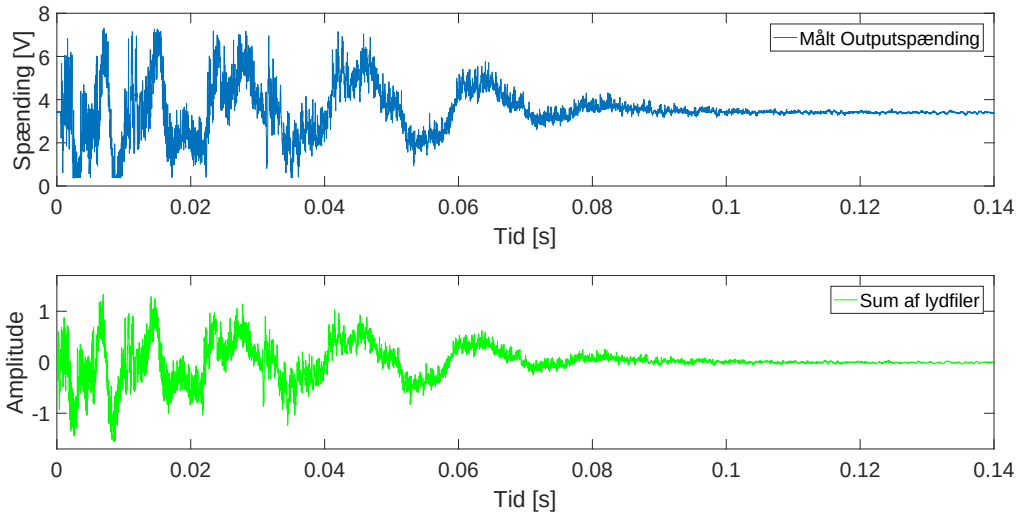
[1] Plottet af lydfilen er normeret fordi det er sådan matlabs "audioread"funktion læser en wav-fil.



Figur B.3: Måling af hi-hat samt den originale lydfil.



Figur B.4: Måling af klap samt den originale lydfil



Figur B.5: Måling af alle fire lyde afspillet samtidigt samt summen af de fire lydfiler

Lydfilerne er normeret sådan at amplituden 1 og -1 svarer til den henholdstvist største og mindste mulige 16-bits værdi. På grund af den måde lydmodulet er designet på burde outputsignalet "klippe"hvis et sample der ligger udenfor dette interval skal afspilles, hvilket også understøttes af testen i appendiks A. Dette virker ud fra figur B.5 også til at være tilfældet.

Konklusion

Ud fra resultaterne kan det konluderes at lydmodulet er i stand til at afspille de fire lydfiler korrekt, hvis dets input aktiveres. Derudover kan det ses at lydmodulet er i stand til at afspille alle lydfiler samtidi hvis dette skulle være nødtvendigt.

Test af sequencer C

Formål

Denne test går ud på at undersøge om del-kravet til sequenceren, om at kunne sample et trommeslag inden for 1,6 ms, er overholdt. Dette krav kan findes i afsnit 2.2.

Udstyr

Det anvendte forsøgsudstyr i testen kan ses på tabel C.1

Instrument	Model	AAU-nummer
FPGA	Papilio-duo 2MB	
Digitalt oscilloskop	Digilent Analog Discovery 2	2179-04
WaveForms (software)	2015 version 3.5.4	

Tabel C.1: Tabel over anvendt forsøgsudstyr.

Metode

Eftersom det kun er krav 2, som er præcist specificeret, hvor resten er funktionelle krav, er det kun dette krav der vil blive testet vha. udstyr. De resterende krav er observeret og bliver her efter beskrevet i 3.2. Kravet om afspilning efter 1,6 ms, er blevet undersøgt vha. det digitale oscilloskop "Digilent Analog Discovery 2". Her er der set på, hvornår der er blevet trykke på en knap til et trommeslag. Herefter ses der på hvor lag tid det tager før et signalet til lydmodulet indtræffer. Dette er blevet gjort ved at tage 10 målinger ved hhv. 100 og 50 BPM, eftersom 100 BPM er det beregningerne tager udgangspunkt i, og ved 50 BPM fordi det er her, sequenceren er langsomst og derfor har den længste responstid.

Resultater

De målte tider for sequencerens opdateringshastighed er blevet noteret og der er fundet et gennemsnit af disse tider, som kan ses i tabel C.2.

Måling	Tid ved 100 BPM [µs]	Tid ved 50 BPM [µs]
1	300	75
2	300	540
3	502	873
4	503	635
5	550	804
6	700	1137
7	400	1097
8	150	732
9	327	30
10	140	928
Gennemsnit:	387,2	685,1

Tabel C.2: Tabel over anvendt tids resultater.

Konklusion

Ud fra resultaterne konkluderes det, at kravet er opfyldt, da gennemsnitstiden 685,1 µs og den længste tid er 1137 µs, ved en hastighed på 50 BPM.

Accepttest D

Indledning

I denne målerapport vil der redegøres for de test der er udført på den samlede trommemaskine. Testene er udført for at teste om trommemaskinen er i stand til at overholde kravspecifikationen i afsnit 2.1.

Udstyr

Alle testene beskrevet i dette appendiks anvender samme forsøgsudstyr. Det anvendte forsøgsudstyr i testen kan ses på tabel D.1.

Instrument	Model	AAU-nummer
Trommemaskine (DUT)	Papilio-duo 2MB	
Digitalt oscilloskop	Digilent Analog Discovery 2	2179-04
Strømforsygning	HAMEG HM7042	B1-101-N-7 B1-101-O-6

Tabel D.1: Tabel over anvendt forsøgsudstyr.

D.1 Test af sequencerinkodning

Denne test går ud på at teste hvorvidt trommemaskinen kan gemme rytmer i en sequencer. Med andre ord laves denne test for at se om trommemaskinen opfylder det første funktionelle krav.

Metode

For at kunne måle hvornår der er gået en sekvens sættes første slag i alle fire sekvenser til at være et klap ved hjælp af stepsequenceren. Denne lyd er valgt da denne har en meget "skarp"transient i starten, og derfor atntages det at denne er forholdvis nem at måle efter.

Ud over dette ene slag tømmes alle sekvenser i sequenceren for slag. Kontrolsignalet "modeselect"sættes til at være 420, sådan at inputs fra realtidstrommer bliver gemt i sequenceren. Derudover sættes brugerinputtet til valg af sekvens til at være sekvens nummer 1.

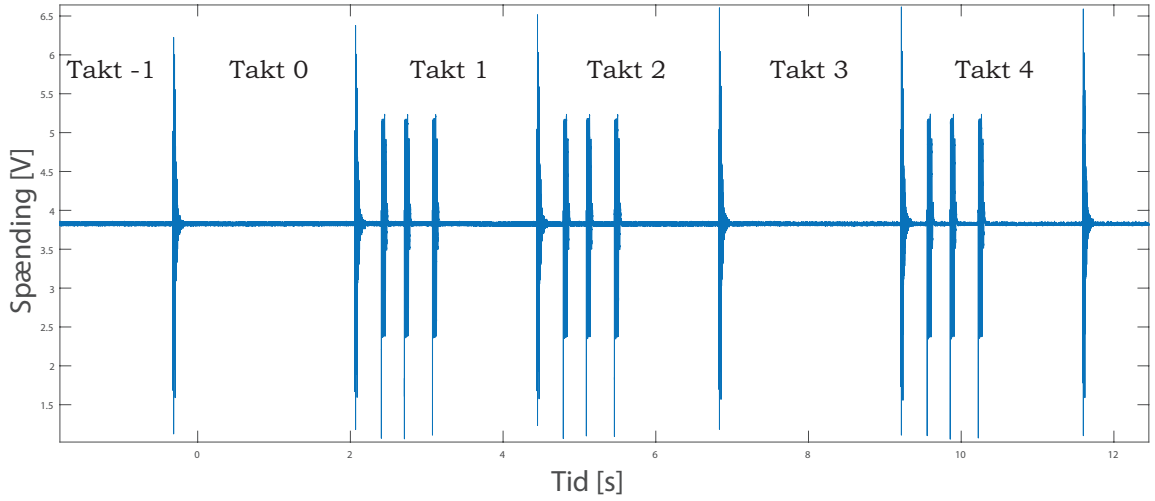
Trommemaskinen tilsluttes et oscilloskop, der sættes til at måle trommemaskinens output over et stykke tid.

Efter at have kørt et par takter igennem påføres sampleren et input (i dette tilfælde tre stortrommeslag i træk). Efter begyndelsen af en ny takt burde det samme output kunne læses, som da realtidstrommerne blev påført input.

Efter dette skiftes sequenceren over til at afspille sekvens nummer 2, der her kun burde være det inkodede klap i starten af sekvensen, over en takts tid. Derefter skiftes der tilbage til at afspille sekvens nummer 1, der stadig burde have den gemte sekvens.

Resultater

Et udsnit af den målte data kan ses på figur D.1. De målte takter er blevet numeret for bedre at kunne beskrive hvilket skridt i testen der er tale om.



Figur D.1: så det..

Indtil og med "takt 0"ses der ikke andet en klappet i starten af hver sekvens.

På takt 1, ses det at der afspilles en lyd tre gange. Dette svarer til den takt hvor sequenceren bliver programmeret fra realtidstrommerne. Den samme sekvens bliver gentaget på takt 2, hvilket tyder på at sekvensen der blev indkodet i takt 1 er blevet gemt.

På Takt 3, skiftes der over til sekvens nummer 2, hvor intet andet end klappet burde være gemt. Dette ser også ud til at være tilfældet.

Til sidst vendes der tilbage til sekvens nummer 1 på takt fire, og det kan ses at samme sekvens der blev afspillet i takt og 2 bliver afspillet her.

Alt i alt forløber hele testen som den burde.

D.2 Test af fri tromning

Denne test har til formål at teste trommemaskinens 2. krav, navnligt at tromme maskinen skal være i stand til at tage imod input fra en sampler og afspille det, uden at slagene gemmes i sequenceren når "modeselect"er ovr 9000.

Metode

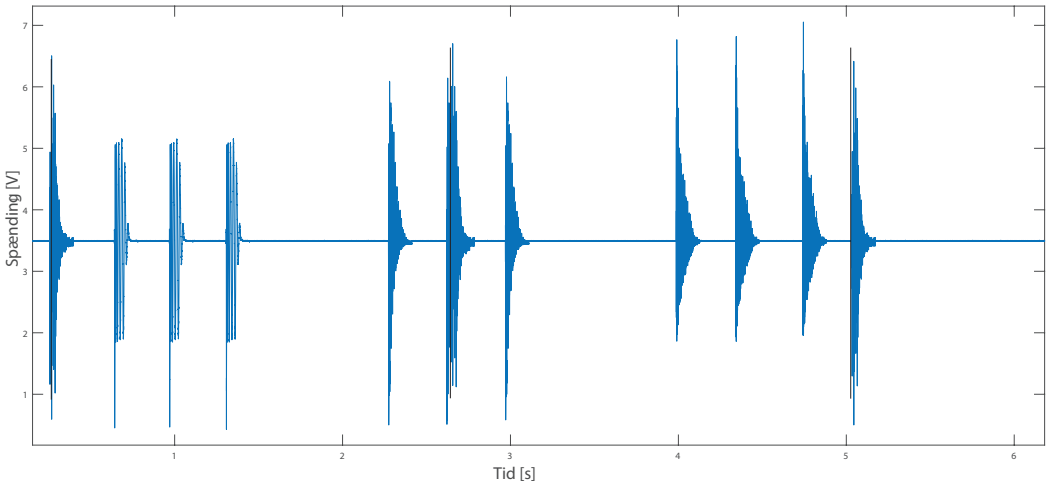
For at kunne måle hvornår der er gået en sekvens sættes første slag i sekvensen igen til at være et klap ved hjælp af stepsequenceren. Kontrolsignalet "modeselect"sættes til at være 69,

hvilket burde sørge for at inputs fra realtidstrommeknapperne kun foresager en afspilning af trommelyden, men ingen lagring af inputtet i sequenceren.

Trommemaskinen tilsluttes et oscilloskop, og der trommes først tre gange på stortrommen, derefter tre gange på lilletrommen og til sidst tre gange på hi-hatten. Hvis alt fungerer som det skal burde ingen af disse slag gentages efter at sequenceren har kørt en sekvens igennem (hvilket kan ses på klappet).

Resultater

Oscilloskoppets spænding er målt over en periode på ca. 3 sekvenser, og plottet med Matlab. Starten på hver sekvens er markeret med en sort lodret linje.



Figur D.2: mulle

Det kan ses at ingen af de indspillede inputs bliver gentaget den efterfølgende sekvens, og at ingen trommer umiddelbart bliver afspillet efter de ni inputs er blevet afspillet.

D.3 Test af hastighedsinstilling

Krav 5 i kravspecifikationen omhandler indstilling af trommemaskinens tempo. I denne test ses det om trommemaskinen kan indstille sit tempo fra 50 BPM til 200 BPM, samt om det den målte BPM stemmer overens med den BPM-værdi brugeren kan læse på 7-segmentsdisplayet.

Metode

En sekvens længde i BPM måles i denne test ved i første omgang at måle længden af en sekvens.

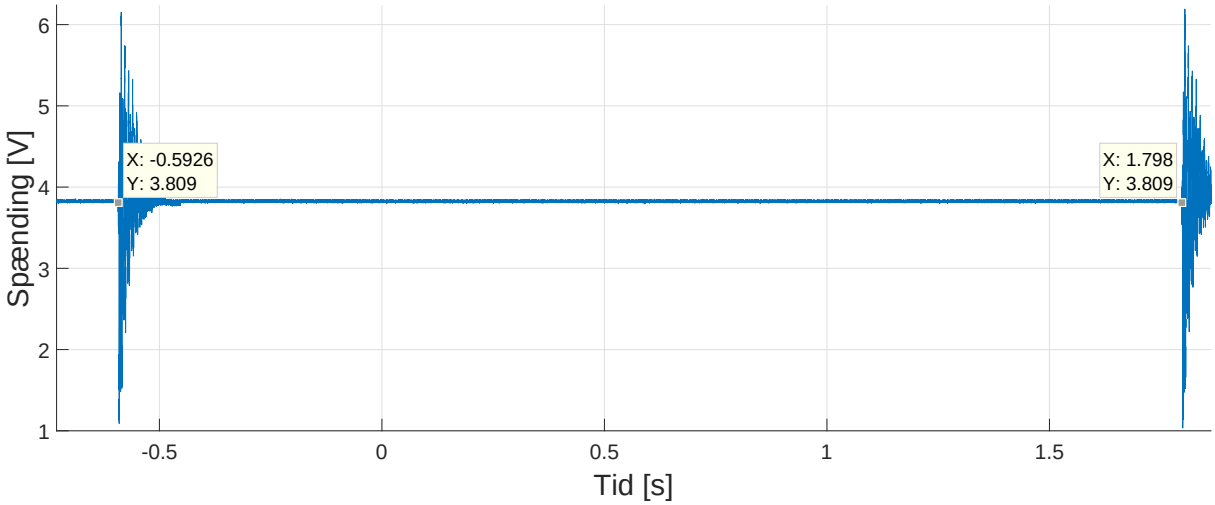
Trommemaskinen er endnu en gang tilsluttet et oscilloskop der måler trommemaskinens output over et stykke tid.

I stepsequenceren indkodes der et enkelt slag i stepsequenceren. Denne ene trommelyd vil derefter blive afspillet en gang per sekvens. Længden af en sekvens kan derefter måles som længden mellem to trommelydes første transient.

Testen påbegyndes ved at Trommemaskinens tempo indstilles sådan at 7-segments displayet viser hhv. 100, 50 og 200 BPM.

Resultater

De målte data ved 100 BPM kan ses på figur D.3.



Figur D.3: the graph below me is an dumb

På figur D.3 er starten er hver transient markeret.

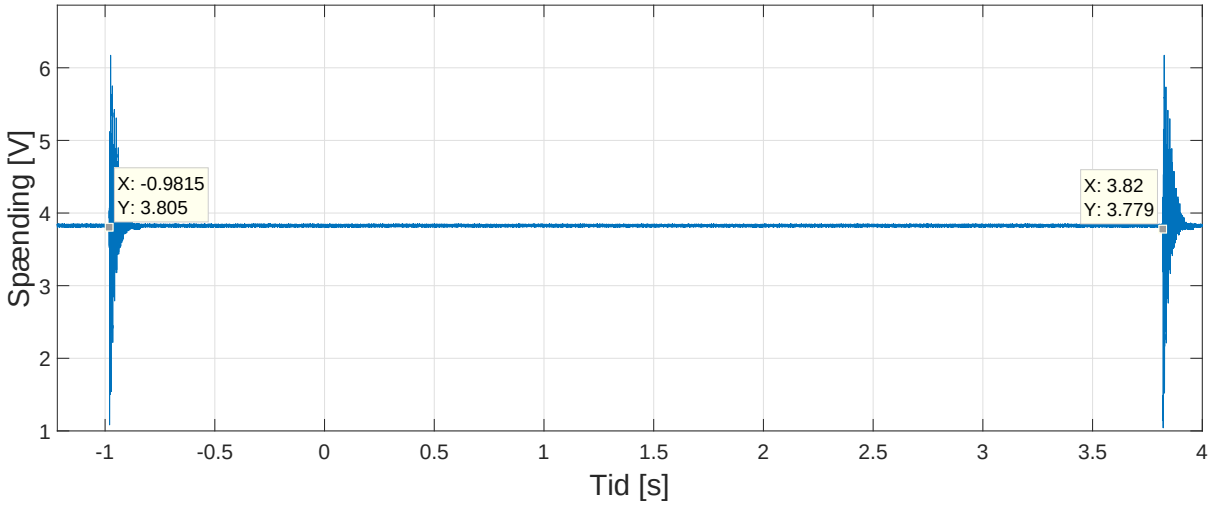
Antallet af sekunder per sekvens T kan udregnes som forskellen på punkternes x-værdi.

$$T = 1,798 \text{ s} - (-0,593) \text{ s} = 2,391 \text{ s} \tag{D.3.1}$$

Der går som beskrevet tidligere fire takslag på en sekvens og der går 60 sekunder på et minut. Da vi nu ved antallet af sekunder per sekvens kan BPM-værdien udregnes som:

$$\text{BPM} = 4 \, \frac{\text{Taktslag}}{\text{Sekvens}} \cdot \frac{1}{2,39} \, \frac{\text{Sekvens}}{\text{s}} \cdot 60 \frac{\text{s}}{\text{min}} = 100,38 \, \frac{\text{Taktslag}}{\text{min}} \, (\text{BPM}) \tag{D.3.2}$$

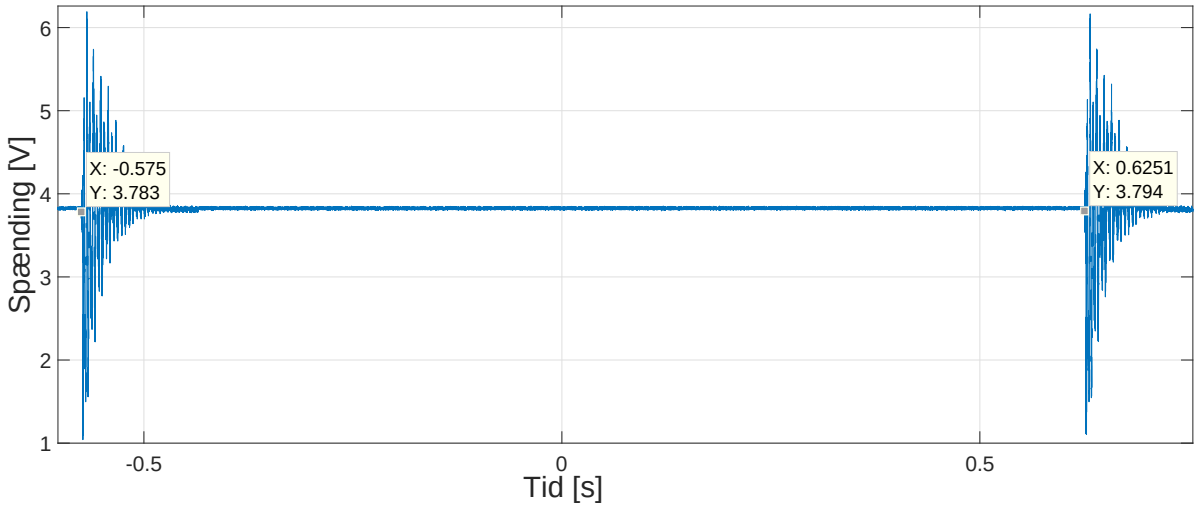
Denne målte værdi ligger inden for $\pm 0,5$ af værdien på 7-segmentsdisplayet, hvilket vil sige at 7-segmentet ikke er i stand til at vise et tal der ligger tættere på.



Figur D.4: khg

Samme udregning udføres for testen ved 50 BPM.

$$\text{BPM} = 4\ \frac{\text{Taktslag}}{\text{Sekvens}} \cdot \frac{1}{3.779 - (-0.982)}\ \frac{\text{Sekvens}}{\text{s}} \cdot 60\frac{\text{s}}{\text{min}} = 50,4\ \frac{\text{Taktslag}}{\text{min}}\ (\text{BPM}) \quad \text{(D.3.3)}$$



Figur D.5: this

$$\text{BPM} = 4\ \frac{\text{Taktslag}}{\text{Sekvens}} \cdot \frac{1}{0,625 - (-0,575)}\ \frac{\text{Sekvens}}{\text{s}} \cdot 60\frac{\text{s}}{\text{min}} = 200,0\ \frac{\text{Taktslag}}{\text{min}}\ (\text{BPM}) \quad \text{(D.3.4)}$$

Da måden hvorpå BPM-signalet bliver udregnet på FPGA'en altid sker ved at tallet bliver rundet ned til nærmeste heltal. Derfor kan tallet 200 kun læses når tempoet har præcis denne værdi.

D.4 Test af sekvenslagring

Denne test har til formål at vise at tromme maskinen kan lagre 4 forskellige sekvenser, hvilket er sat som krav 6 i kravspecifikationen.

Metode

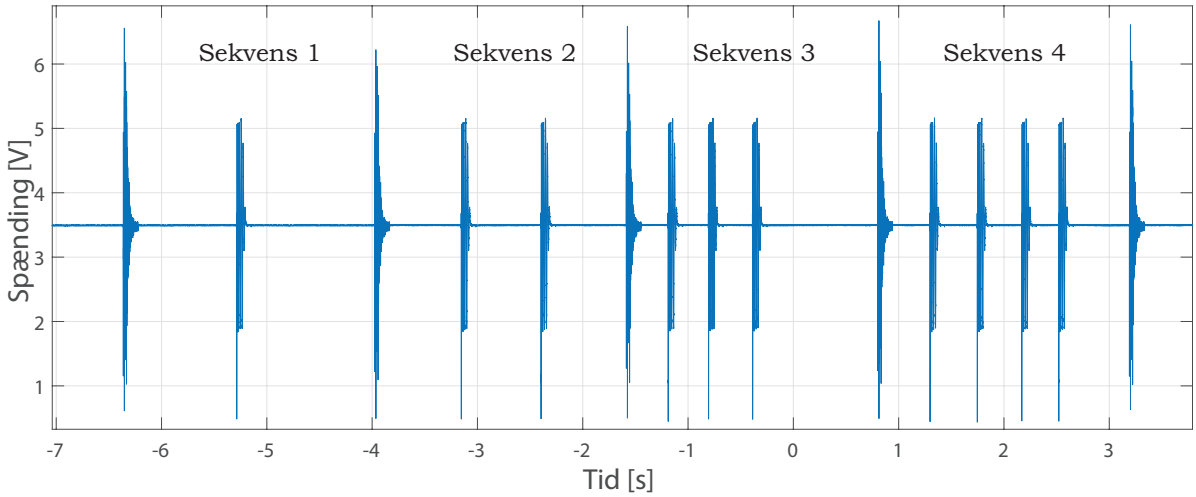
Endnu engang sættes der et klap for at markere starten af en sekvens med stepsequenceren. Dette gøres for alle fire sekvenser.

Derefter indspilles der fire forskellige rytmer i de fire mulige sekvenser. For at gøre det let at skelne mellem sekvenserne, sættes første sekvens til at være et slag på stortrommen, anden sekvens til to slag på stortrommen o.s.v.

Trommemaskinen tilsluttes igen et oscilloskop. Sekvensene afspilles herefter en gang hver i træk.

Resultater

Resultaterne af måligen kan ses på figur D.6.



Figur D.6: Fire forskellige sekvenser

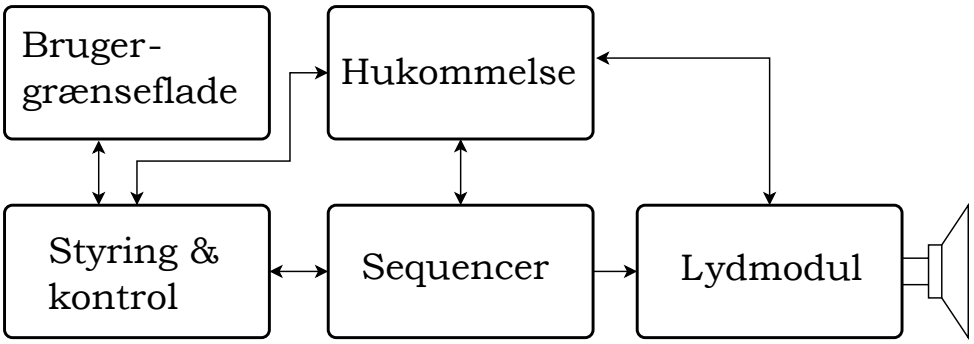
Af figuren kan det ses at alle sekvenserne er forskellige, og at de spiller den sekvens de burde.

Konklusion

Igennem alle disse fire tests har trommemaskinen konsekvent opført sig som formodet. Trommemaskinen er både i stand til at blive programmeret med stepsequencer og sampler, samt at blive trommet uden at gemme sekvensen. Der kan gemmes 4 sekvenser, og man kan indstille tempoet mellem 50 og 200 BPM. På baggrund af dette vil de testede krav, hermed anses som at være opfyldte.

Design 5

I dette kapitel beskrives trommemaskinens design og implementering. Designet er opdelt med udgangspunkt i tre delmodulers kravspecifikationer (kontrolmodul, sequencer og lydmodul) som det er beskrevet i afsnit 2.2.



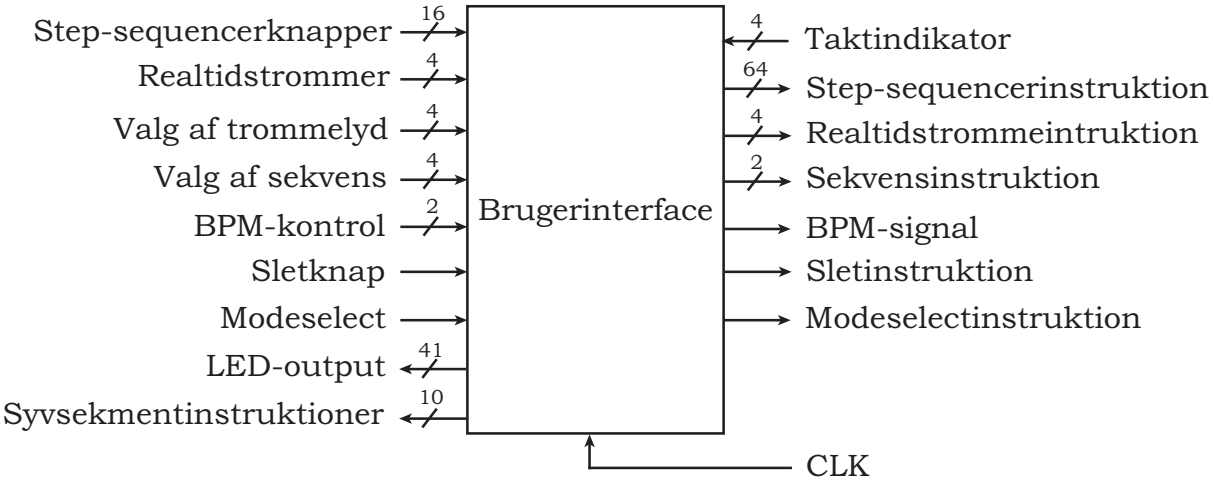
Figur 5.1: Blokdiagram over trommemaskinen som helhed

5.1 Design af kontrolmodul

Kontrolmodulet er en grænseflade mellem brugeren og resten af systemet. Den har til formål at bearbejde brugerens indtastede instruktioner beskrevet i afsnit 2.2, og informere brugeren vedrørende valg af tromme, indkodet sekvens, lokation i sekvens mm.

Dette modul er opbygget af to dele, hvor den første er selve hardwaren som brugeren skal interagere med, som fremadrettet kaldes brugerinterfacet. I denne forbindelse vælges der er fremstille et stykke hardware (eller et "shield") til Papilio Duo'en, som håndterer de ønskede funktionaliteter. Den anden del er behandlingen af signalerne mellem topmodulet og resten af system, hvad end det er inputs eller outputs.

For at opbygge en forståelse for hvilke signaler der skal arbejdes med, vises et moduldiagram for kontrolmodulet som kan ses på figur 5.2. Her er den venstre side af diagrammet signaler brugeren kan interagere med, altså brugerinterfacet, og den højre side er signaler til eller fra resten af systemet.

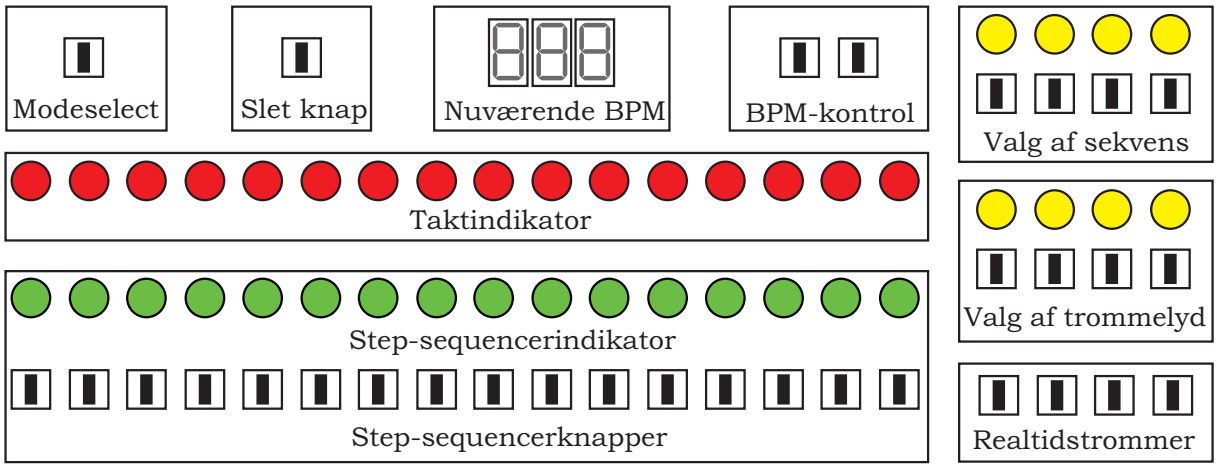


Figur 5.2: Moduldiagram over signaler for kontrolmoduler

5.1.1 Opbygning af brugerinterface

Der er valgt at gøre brugerinterfacet så let anvendeligt som muligt. Derfor er der valgt at de forskellige funktionaliteter skal have sin egen knap, hvoraf mange af disse også vil have en indikator for om signalet er tændt eller ej. Funktionerne opdeles i ti kategorier: step-sequencerknapper, step-sequencerindikator, taktindikator, sekvensvælger, trommevælger, realtids-trommeknapper, tromme-mode, slet-knap, BPM-kontrol og BPM-indikator.

For at overskueliggøre dette interface er der fremstillet en skitse der kan ses på figur 5.3.



Figur 5.3: Illustration af brugerinterfacet.

Hver kategori vil herunder blive beskrevet.

Step-sequencerknapperne består af 16 trykknapper, der benyttes til at vælge om der skal afspilles en trommelyd på et specifikt tidspunkt i de 16 sekstenedele af en takt.

Step-sequencerindikatorne består af 16 LED'er, som tændes eller slukkes når man trykker på tilsvarende step-sequencer knap. Disse LED'er har til formål at vise brugeren, om der er valgt at afspille en trommelyd, på en specifik taktdel. Her betyder en tændt LED at lyden skal afspilles.

Taktindikatoren består af 16 LED'er som tænder og slukker en efter en, som viser i hvilken af de 16 sekstenedele af en takt man er på. Altså vil lyd eksempelvis spille når der er lys i både den

specifikke taktindikator-LED samt den tilsvarende step-sequencer-LED.

Sekvensvælgeren består af fire tryk-knapper, som skal gøre det muligt at skifte mellem forskellige sekvenser. De skal kunne vælge mellem forskellige sekvenser, så der muligt at opdatere disse signaler. Det må kun være muligt at have én af de fire sekvenser at aktiveret af gangen, og der benyttes fire LED'er til at indikere hvilken sekvens der benyttes.

Trommevælgeren er fire trykknapper, som skal bruges at fortælle sequenceren hvilken trommelyd der programmeres med hensyn til, når step-sequenceren indkodes. Her kan kun én af trommelydene vælges af gangen, og der benyttes igen fire LED'er til at indikere hvilken trommelyd der benyttes.

Realtids-trommeknapper er også fire trykknapper, men disse skal programmere sequenceren i realtid for deres respektive trommelyd.

Tromme-mode er en switch der afgør om inputs fra realtids-trommeknapperne skal afspilles og gemmes i den nuværende sekvens, eller om de kun skal afspille uden at blive gemt.

Slet-knappen har til formål at slette dataen for en bestemt sekvens og tromme for step-sequenceren, og slette alt dataen for en bestemt sekvens for alle realtids-trommerne.

BPM kontrollen består af to tryk-knapper, som kan skrue op eller ned for BPM i et interval mellem 50 og 200.

BPM-indikator skal vise brugeren hvilket sequencerens tempo. Da trommemaskinens BPM kan have værdier fra 50 til 200, vælges BPM'en at repræsenteres på et display, dette bliver beskrevet yderligere i afsnit 5.1.5

Shift-registeres forunderlige verden

Det ses på figur 5.2 at brugerinterfacet sammenlagt kræver 80 I/O-porte, hvilket er mere end de 54 porte Papilioen er udstyret med hvis hver knap/led skal forbindes direkte. Af denne grund vælges der at bruge shift-registre som et mellemled mellem FPGA'en og step-sequencerknapperne samt outputs til LED'erne, da disse ikke er specielt tidsfølsomme. Med shift-registre er det muligt at konvertere mellem serielle og parallelle signaler, hvorved der kan spares mange porte på FPGA'en, da denne derved hovedsagligt skal sende og modtage serielt data.

Først og fremmest ønskes der at adskille ind- og outputs således at disse bruger forskellige shift-registre for at overskueliggøre og adskille data. Derudover ønskes der at adskille LED-udgangssignalerne i to kategorier, navnligt LED'er der styres af knapper, og LED'er der styres af taktlokationen. Dette gøres da der kun af én af taktlokations-LED'erne der skal være tændt af gangen, hvorimod de andre LED'er styres af deres respektive knapper.

Med denne opdeling vil den maksimale mængde parallelle LED'er der skal styres serielt være 24, hvilket betyder LED'erne potentielt kan styres 24 gange langsommere end clock'en der styre shift-registerene. Der tages her udgangspunkt i det 8-bits 74HC595 shift-registrer[2], som er serielt til parallelt. Det er muligt at implementere denne type shift-register i daisy-chain, hvilket betyder at der kan sættes tre i serie, for at opnå et 24-bits shift-register.

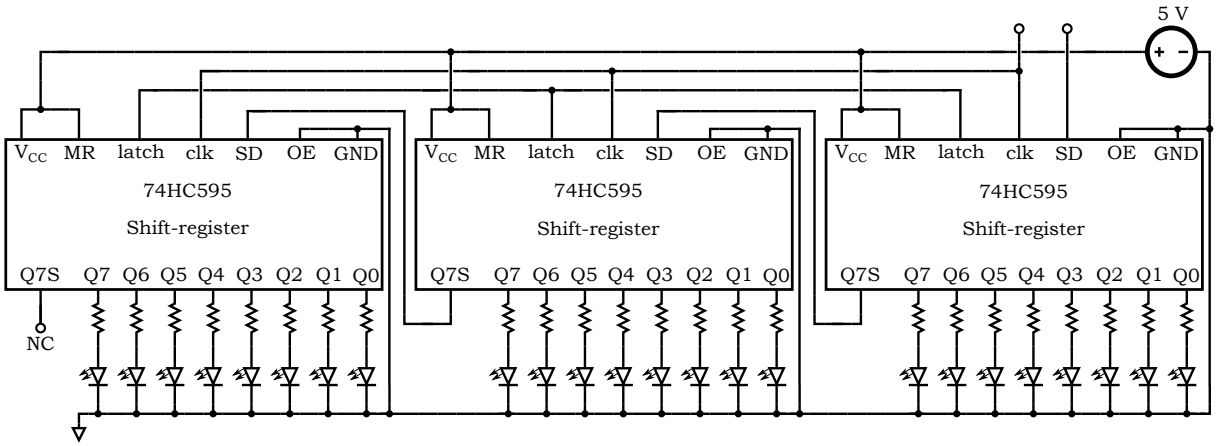
Da alle tre bliver drevet af de samme inputsignaler, undersøges der om denne implementering kan lade sig gøre tidsmæssigt. I følge databladet er den maksimale clockfrekvens for shift-registret

4 MHz, hvilket svare til en periodetid på 250 ns. Opdatiringshastigheden for de 24 LED'er kan nu udregnes som:

$$T_{LED-opdatering} = 24 \cdot 250\,\text{ns} = 6\,\mu\text{s} \tag{5.1.1}$$

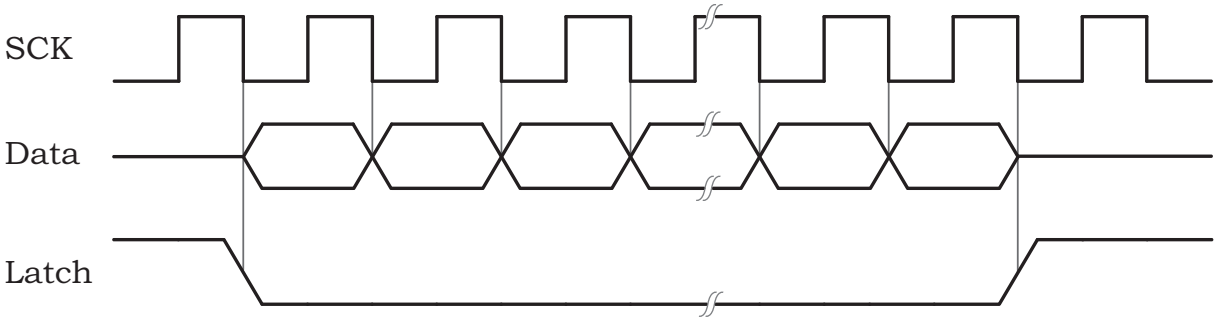
Denne opdateringshastighed anses for at være mere end tilstrækkelig hurtig for at kunne opdatere LED'erne. Dette medfører også at der kan benyttes shift-registre til opdatering af signalstatus for step-sequencerknapperne, da den øvre tidsgrænse for systemet er 1 ms. Signalstatus for disse knapper ville derudover kunne opdateres endnu hurtigere, da der kun er 16 knapper, hvilket medfører at opdateringshastigheden kun er 16 gange langsommere end den anvendte clock, frem for det 24 som er gældende for LED'erne.

På figur 5.4 ses kredsløbsdiagrammet for shift-registrene til de 24 LED'er. Det ses her at de alle sammen benytter samme latch, clock og serielle datasignal(SD). Dette medføre at der, ved brug at tre I/O-porte på FPGA'en, kan tændes og slukkes for 24 LED'er.



Figur 5.4: Komponenttegning af 3 seriel til parallel i daisy-chain.

Med denne sammenkobling fyldes shift-registrene én af gangen. Når den ene er fyldt, hvilket bestemmes af clk-signalet, påbegyndes fyldning af det næste register automatisk. Timingsdiagrammet for disse shift-registre kan ses på figur 5.5. Det ses her at der kun skrives data når latchen er 0, hvilket medfører at der ingen ændring i data er, når latchen er 1.



Figur 5.5: Timingsdiagram for shift-registrene.

Til kontrollering af LED'ernes stadie, er der fremstillet en tilstandsmaskine. Denne ses i kodeeksempel 5.1.

Kodeeksempel 5.1: Opdatering af data på shift-registre til LED'er

```
if rising_edge(shiftClk) then
  case controlLEDstate is
    when resetting =>
      controlLEDstate <= shifting;
      controlLEDLatch <= '0';
    when shifting =>
      shift_counter2 := shift_counter2 + 1;
      controlLEDData <= controlLEDSignal(shift_counter2-1);
      if (shift_counter2 >= 26) then
        controlLEDstate <= resetting;
        controlLEDLatch <= '1';
        shift_counter2 := 0;
      end if;
  end case;
end if;
```

Denne paragraf skal renskrives. Det ses at der skiftes mellem to tilstande, som er henholdsvis resetting og shifting. I resetting sættes shift-registrenes latch til lav, så dataen herpå kan opdateres. Samtidig skiftes tilstanden til shifting. I denne tilstand startes en counter, så de enkelte bit i en tilhørende bitstreng kan placeres på deres retmæssige udgangsben. I dette tilfælde er det signalet "countLEDSignal", der er tale om. Da den tilhørende counter går en op, samtidig med at der skal sendes data, bliver der trukket en fra hvad?, når pladsen i bitstrengen vælges. For hvert clocktick stiger counteren med en og den næste bit sendes til shift-registrene. Når counteren når til 26, nulstilles counteren, samtidig med at latchen åbnes og tilstanden kommer tilbage til nulstillingstilstanden. Dette sker først når counteren når 26, da man mister det første step i counteren når der startes. Ligeledes skrives der ikke til shift-registrener når counteren rammer 26 da latchen åbnes her. Dette vil sige at der sendes 24 bit til shift-registrene med denne opstilling. Processen kører på klokken shiftClk, som også bliver sendt til clock-benet på shift-registrene. På den måde er man sikret at det hele kører parallelt måske nærmere synkront?. Det kan yderligere ses at det ikke er muligt at stoppe med at skrive til shift-registrene. Dette skyldes at brugeren ikke nødvendigvis trykker på en knap, i takt med en bestemt clock. bestemt clock? bruges der ikke kun en her Derfor opdateres der konstant, for at tage højde for ændringer i hvad. For shift-registrene der styrer knapperne, gennemgår samme skridt. Her bliver dataen blot sat ind i et array i stedet for at blive shiftet ud. <– her brydes den røde tråd meget synes jeg. Det giver mere mening at have det her samen med beskrivelse af knapshiftophookning

det skader ikke med et kodeeksempel + eldiagram til dette

Ud over de 24 omtalte LED'er, indeholder brugerinterfacet yderligere 16 LED'er til at vise den nuværende lokation i sekvensen. Disse er implementeret på næsten samme måde. Forskellen er at der kun benyttes to shift-registre i daisy-chain, og at der kun må være én LED tændt af gangen. Det serielle datainput bliver dannet af instruktioner fra sequenceren, hvor dens lokation i sekvensen afgør hvilken LED der tændes.

Shift-registeret, 74HC595, har sin parallelle til serielle modpart ved navn HEF4021B. [3] Disse fungerer på stort set samme måde, hvad angår timing og opkobling som shiftregistrene til LED'erne. Forskellen er at HEF4021B giver seriel data, frem for at modtage. Dette benyttes til de 16 step-sequencerknapper, hvor de indkommende data håndteres på næsten samme måde som i kodeeksempel 5.1. Her bliver FPGA'ens signaler dog opdateret på baggrund af af brugerens inputs.

Det er kun de 16 step-sequencerknapper der vælges at læses med shift-regsitrer. De resterende 15 knapper har hver deres I/O-port. Dette er valgt for ikke at gøre afkodning af indkommende serielt data unødvendigt kompliceret. Derudover gøres det nemmere at ændre på FPGA'ens variabler og signaler på baggrund af input fra disse knapper.

5.1.2 Udradering af prel

Et problem med de fysiske knapper er at de har brug for at få fjernet prel. Prel er et udtryk for stadiet ved den stigende signalkant, hvor det ikke er helt sikkert om signalet er 1 eller 0. Hvis ikke dette fjernes er knappens stadie usikkert når der skiftes fra lav til høj, hvilket giver problemer fremadrettet da det kan kalde processer flere gange. For eksempel kunne man ved tromning i realtid, afspille en tromme mange gange ved et tryk. I kodeeksempel 5.2 ses der hvordan dette håndteres for én enkelt knap:

Kodeeksempel 5.2: Kode til at fjerne prel fra knapper

```
process(clk)
begin
  if (clk'event and clk = '1') then
    Q1 <= button_in;
    Q2 <= Q1;
    Q3 <= Q2;
  end if;
end process;
debounced_button <= Q1 and Q2 and (not Q3);
```

Ved hvert stigende signalkant på en valgfri clock, der i dette tilfælde er den på 32 MHz, gemmes det nuværende knapsignal. Derudover roteres signalbuffers således at de ønskede knapsignal først anses som værende 1, når det har været 1 i to efterfølgende clockcyklusser, hvor det fåregående har været 0. Det antages at hvis dette er sandt vil knapsignalet være tilpas stabilt til at benyttes i fremadrettede operationer. Dette gøres for de 16 stepsequencerknapper, så signalet kun bliver inverteret en gang. Samme proces gennemgås for realtidstrommeknapperne. Her ses dog på knappernes stadie i længere tid. Dette kan ses i kodeeksempel 5.3.

Kodeeksempel 5.3: Kode til at fjerne prel fra realtidsknapper

```
counter_size : integer := 9;
signal flipflops : STD_LOGIC_VECTOR(1 DOWNTO 0);
signal counter_set : STD_LOGIC;
signal counter_out : STD_LOGIC_VECTOR(counter_size DOWNTO 0) := (others => '0');
counter_set <= flipflops(0) xor flipflops(1);
process(clk)
begin
  if rising_edge(clk) then
    flipflops(0) <= button;
    flipflops(1) <= flipflops(0);
    if counter_set = '1' then
      counter_out <= (others => '0');
    elsif counter_out(counter_size) = '0' then
      counter_out <= counter_out + 1;
    else
      result <= flipflops(1);
    end if;
  end if;
```

```
19 end process;
```

Som nævnt tidligere er fremgangsmåden for at fjerne prel, den samme for realtidstrommeknapperne. Der ses først på om der er sket en ændring. Hvis der sker en ændring nulstilles counteren counter_out. Hvis der ikke er sket ændringer begynder denne counter at tælle op, indtil den når den tiende bit i count_out-arrayet. Det vi sige at signalet for knapperne skal være stabilt i 32 µs før resten af systemet antager at det er højt, hvilket antages at være tilstrækkeligt kort tid til ikke at have nogen stor effekt på systemet. Hvis der ikke er sket ændringer og counteren når sit mål, sendes signalet videre til resten af systemet. På denne måde er prel fjernet fra knapperne og vil derved ikke skabe problemer for de mere tidssensitive processer.

5.1.3 Behandling af data til og fra brugerinterface

For at bestemme hvilke trommer der skal afspilles i step-sequenceren, skal det være muligt at gemme instruktionerne. Step-sequencerinstruktionerne bliver gemt i et array af instruktioner, som bliver opdateret af step-sequencerknapperne. For hver af de fire trommer, er der et separat array, som det ses i kodeeksempel 5.4.

Kodeeksempel 5.4: Data struktur for sequencerinstruktionerne

```
1 TYPE seqData is array (15 downto 0) of STD_LOGIC; --Incoded by step-sequencer.
2 TYPE drum is array (3 downto 0) of seqData;  --Choice of drum sound.
3 TYPE seq is array (3 downto 0) of drum;   --Chouse of sequence.
4 shared variable data : seq;
5 signal seqVar, drumVar : integer := 0;
```

For at tilgå og ændre data benyttes syntaksen: data(seqVar)(drumVar)(seqData), hvor seqVar bliver bestemt af "valg af sekvens-inputtet, drumVar bliver bestemt af "valg af trommelyd-inputter og seqData bliver indkodet af step-sequencerknapperne. Der er valgt at benytte denne datastruktur da det er let at ændre og læse specifikke sekvenser med få tryk på knapperne. Step-sequencerinstruktionerne på figur 5.2 bliver bestemt alt efter hvilket af de fire seqVar signaler der er høje.

For at opdatere signalet for en step-sequencerinstruktion, køres en proces på baggrund af knapperne hertil. Først sendes knappernes signal gennem en debounce process, for at sikre at dataen, som høre hertil, kun bliver opdateret en gang. Herefter kan der køres en process som set i kodeeksempel 5.5.

Kodeeksempel 5.5: Opdatering af sequencerinstruktionerne

```
1  begin
2    if deleteButton = '1' then
3      data(seqVar)(drumVar)(15 downto 0) := (others => '0');
4    else
5      if rising_edge(dbButtonDataSignal(0)) then
6        data(seqVar)(drumVar)(7) := not data(seqVar)(drumVar)(7);
7      end if;
8      if rising_edge(dbButtonDataSignal(1)) then
9        data(seqVar)(drumVar)(6) := not data(seqVar)(drumVar)(6);
10     end if;
11     ...
12     if rising_edge(dbButtonDataSignal(15)) then
```

```
13      data(seqVar)(drumVar)(8) := not data(seqVar)(drumVar)(8);
14    end if;
15  end if;
16 end process;
```

I overstående kodeeksemple bliver der gennemgået, hvordan dataen for step-sequenceren opdateret. Først i processen ses der på hvorvidt delete-knappener trykket ind. Hvis dette er tilfældet, skal alt dataen for den valgte sekvens og tromme sættes til 0. Hvis dette ikke er tilfældet, ses der på om der er en stigende signalkant, for en af knappers signaler. Den tilhørende plads i data-array'et inverteres så knapperne kan bruges til at tænde og slukke for en trommelyd. På den måde kan brugeren holde styr på hvilke tromme der skal afspilles på et bestemt tidspunkt.

5.1.4 Ændring af BPM

Brugeren skal også have mulighed for at ændre tempoet af trommemaskinen. Dette implementeres som nævnt tidligere ved hjælp af to knapper.

Ved tryk på BPM-kontrolknapperne skal det være muligt at forøge eller mindske den nuværende BPM så længe denne er mellem 50 og 200 BPM jævnfør funktionelle krav nr. 5.

Beregningen af den nuværende BPM tager udgangspunkt i 100 BPM som vælges som "starttempo", hvorved den reelle BPM kan beskrives som følgende forhold:

$$BPM = \frac{speed_0 \cdot 100}{speed} \tag{5.1.2}$$

hvor:

$speed_0$	=	Grundstørrelsen for BPM-signalet	$[\cdot]$
$speed$	=	Den nuværende størrelse for BPM-signalet	$[\cdot]$

Før tempoet kan variere mellem 50 og 200 BPM, skal *speed* variere mellem halv og dobbelt størrelse af $speed_0$. Denne øvre og nedre grænse er afhængigt af tiden det tager at gennemløbe en sekvens og sekvensens opløsning i forhold til Papilioens clock på 32 MHz. Dette kan findes beskrevet og udregnet i sammenfald med designet af sequenceren i afsnit **??**.Derfor tages der her kun udgangspunkt i funktionalitet.

I kodeeksempel 5.6 ses styringen af den nuværende BPM.

Kodeeksempel 5.6: Ændring af BPM

```
1  if rising_edge(spdClk) then
2    if spdUp = '1' then
3      if (speedSignal > minimalValue) then  --equivalent to 200 BPM
4        speedSignal <= speedSignal - 1;  --decreases the sequence time by one clocktick (ca 31
             ns) per sequence tick
5      end if;
6    elsif spdDown = '1' then --equivalent to 50 BPM
7      if (speedSignal < maxvalue) then
8        speedSignal <= speedSignal + 1;  --increases the sequence time by one clocktick (ca 31
             ns) per sequence tick
9      end if;
10   end if;
11 end if;
```

Denne proces revideres på en clock kaldet "spdCLK"som har en periodetid på 0,5 ms. Denne periode tid er valgt ud fra hvad føles naturligt når der ændres i BPM'en.

På den stigende signalkant checkes der først om tempoet skal forøges, hvilket indikeres ved brug af BPM-kontrolknapperne. Der vælges først at checke om BPM'en skal forøges for at undgå tilfældet at begge BPM-kontrolknapperne trykkes ned samtidig, og derved skruer op og ned for BPM'en. Hvis der skues op for BPM'en og den nuværende hastighed er mindre end den maksimale, mindskes hastighedsvariablen hvilket formindsker antallet af clockperioder pr. sekvensskridt. Da en clockperiode kun varer 31,25 ns skal processen køres igennem mange gange før der sker en mærkbar ændring i tempoet.

Hvis tempoet skal formindskes gøres det modsatte.

5.1.5 Display til BPM

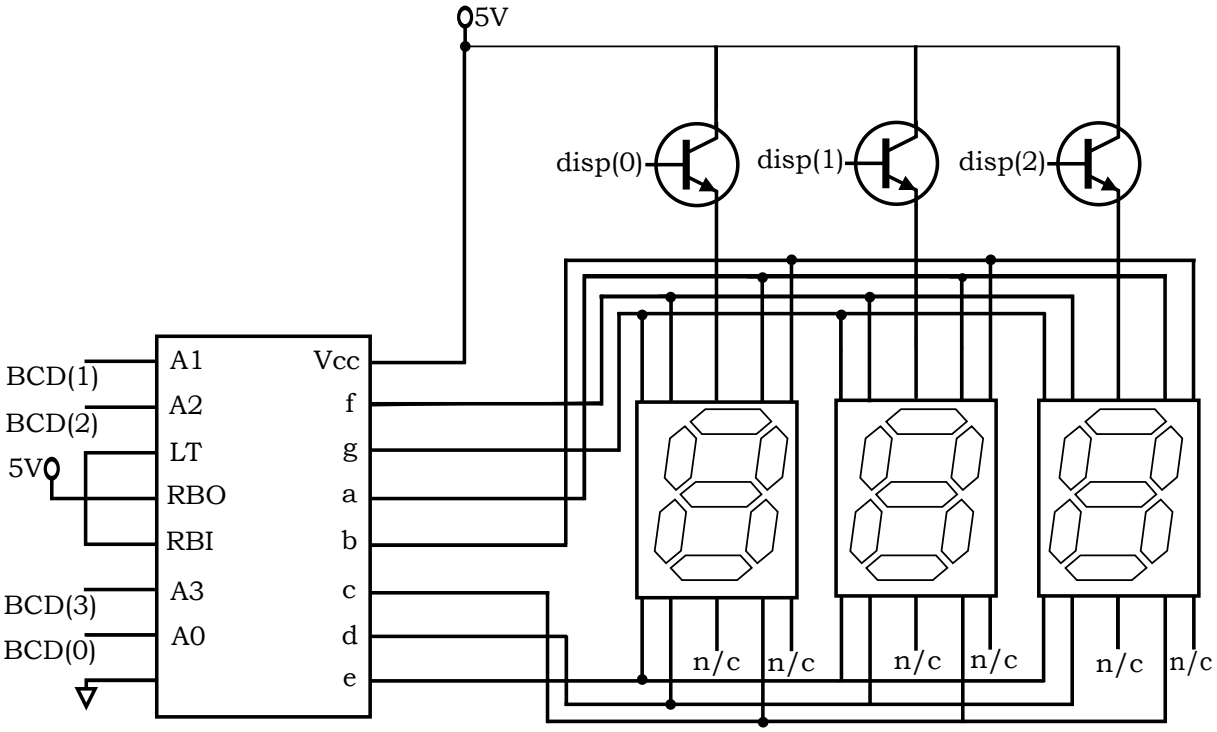
Der er meget vanskeligt at bestemme den nøjagtige BPM-værdi ud fra taktindikatoren. Der er derfor valgt at implementere et display til at indikere den nuværende BPM. Til dette benyttes der tre 7-segmentsdisplay da det er nødvendigt at vise hastigheder til og med 200 BPM jævnfør funktionelle krav nr. 5. i afsnit 2.2.

Afspilningshastigheden bliver på selve FPGA'en repræsenteret som en binært tal hvilket ikke er optimalt, det ikke er sikkert at en bruger kender til dette talsystem, og derudover ønskes der ikke konstant manuel omregning til decimaltal. For at letteregøre denne omregning vælges der at benytte en DM74LS47 BCD (binary coded decimal) [5]. Denne IC benytter en nibble som input, hvorefter dette input konverteres til hvilke af 7-segmentets dioder der skal tændes for at representere tallet decimalt. Nibble'en repræsenteres på en logisk måde således at 0000 bliver til et 0 på displayet, og 1001 bliver til 9. Hvis inputtet til BCD'er 1111 vil ingen af 7-segmentets LED'er være tændt, og de resterende binære kombinationsmuligheder benyttes ikke.

De sidste tre ben kan bruges til at bruge nogle yderligere funktionaliteter på 7-segmentet der ikke ønskes at anvendes i dette projekt. Da disse ben er aktiv lav er benene til "LTRBO"og "RBI"forbundet til forsyningsspændingen.

Da det er nødvedigt at bruge tre 7-segmentsdisplay, men der ikke ønskes at bruge mere end én BCD, er alle segmenterne sammenkoblet parallelt på nær tre kontrolsignaler. Disse kontrolsignaler afgør om et display skal være tændt eller slukket, hvorved det er muligt at skrive til segmenterne én af gangen. Hvis Kontrolsignalerne blinker hurtigt nok kan det skabe en illusion om at alle segmenterne er tændt på samme tid. I databladet er propagation delayet opgivet til 100 ns hvilket er den mindste tid BCD'en skal påføres et specifikt signal, før at udgangssignalet er stabilt. Der vælges derfor at benytte clock'en til opdatering af andre LED'er på brugerinterface. Denne har en periodetid på 250 ns, hvilket vurderes at være hurtigt nok til at opdatere de tre 7-segmenter.

På figur 5.6 ses kredsløbsdiagrammet for BCD'en og 7-seqmentsdisplay'ene. Det er ikke muligt for Papilio'ens I/O-porte at levere tilstrækkeligt strøm til LED'erne i 7-segmentsdisplay'ene, så der er derfor valgt at benytte transistorer til at tænde for segmenterne. Til dette formål bruges slavetransistoren BC547B. Her sidder collectoren til Papilions 5 V's forsygning og emitteren til segmentets anode. Transistorens baseben styres af Papilioens I/O-pins, der derved bestemmer om segmenterne skal tændes.



Figur 5.6: Kredsløbsdiagram over BCD og 7-segmentsdisplay

Den reelle BPM skal vises på tre 7-segmenter, der henholdsvis skal vise hundredere, tiere og ettere. Som det ses i kodeeksempel 5.7 gøres dette ved brug af modulo-operatoren(mod), som returnerer restmængden ved division. For at forklare logikken bag opperationerne tages der udgangspunkt i en eksempel med en BPM-værdi på 193 BPM. Her udregnes først decimaltallet for 7-segmentet der viser hundredere, hvilket kan ses på formel 5.1.3. For en god ordens skyld gøres der opmærksom på at 193 mod 100 er lig 93, hvor dette gemmes i variablen, A, i kodeeksempel 5.7.

$$BPM_{hundrede} = \frac{193 - (193 \mod 100)}{100} = \frac{193 - 93}{100} = 1 \tag{5.1.3}$$

Dernæst udregnes tierne som følgende:

$$BPM_{tiere} = \frac{93 - (93 \mod 10)}{10} = \frac{93 - 3}{10} = 9 \tag{5.1.4}$$

Resultatet for 93 mod 10 gemmes i variablen, B, hvilket svarer til værdien for det tredje og sidste 7-segmentsdisplay.

$$BPM_{enere} = 93 mod 10 = 3 \tag{5.1.5}$$

Som det ses i kodeeksempel 5.7 på linje 8, tælles en counter op for hver gennemgang af processen. Denne bruges til at indikere hvilket af de tre displays der skrives til.

Kodeeksempel 5.7: Visning af BPM på 7-segmentsdisplay

```
1 process (updateCLK)
```

```
variable dispCounter: integer range 0 to 2 := 0;
variable BPM: integer range 50 to 200;
variable A, B: integer;
variable hundreds, tens, ones: integer range 0 to 9;
begin
  if (rising_edge(updateClk)) then
    dispCounter := dispCounter + 1;
    BPM := ((9375*100) / speedSignal);
    A := BPM mod 100;
    hundreds := ((BPM - A) / 100);
    B := A mod 10;
    tens := ((A - B) / 10);
    ones := B;
    case dispCounter is
      when 0 =>
        disp <= "100";
        BCD <= STD_LOGIC_VECTOR(to_unsigned(hundreds,4));

      when 1 =>
        disp <= "010";
        BCD <= STD_LOGIC_VECTOR(to_unsigned(tens,4));

      when 2 =>
        disp <= "001";
        BCD <= STD_LOGIC_VECTOR(to_unsigned(ones,4));
    end case;
  end if;
end process;
```

Ved denne udregning er der ikke mulighed for nøjagtigt at vise BPM'en da der kunne regnes med heltal. Der vil her altid rundes ned til nærmeste heltal, men da det funktionelle krav nr. 5 dikterer en nøjagtighed inder for $\pm$ 1 BPM, antages denne implementering at være tilstrækkelig.

5.1.6 Opsumering af kontrolmodul

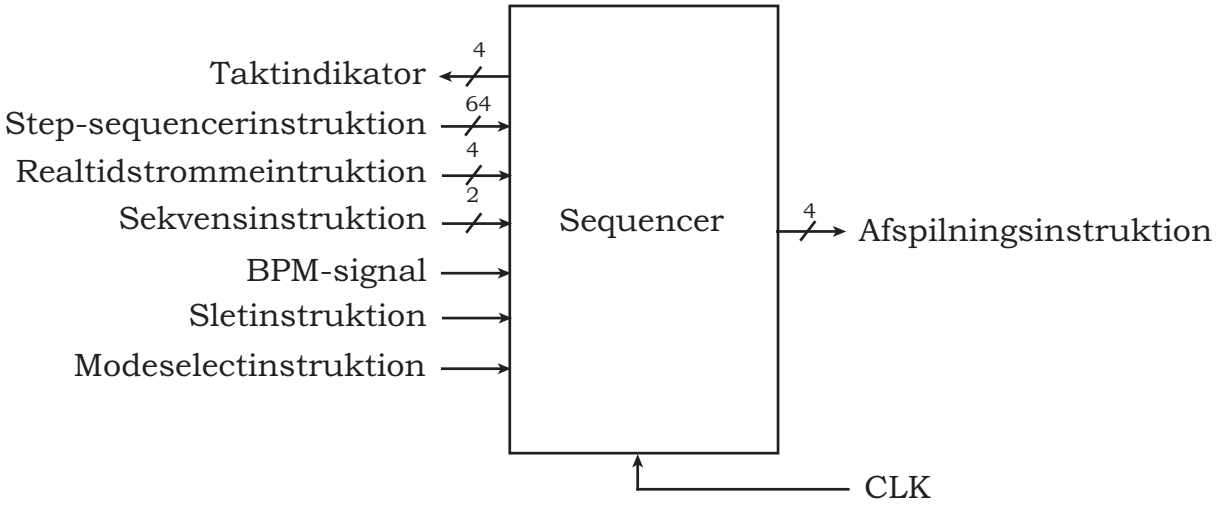
Kontrolmodulet stå nu designet færdigt. Dette modul giver bruger mulighed for at interagere med trommemaskinen, gennem en række af knapper. Nogle af disse læses af shift-registre og andre er direkte er koblet til FPGA'en, hvoraf en del af dem er sikret mod pral. Disse knapper giver mulighed for at programmere trommemaskinen, og tænder en række LED'er for at indikere diverse brugerinput. Disse LED'er er styret af shift-registre. For yderligere at øge brugervenligheder er der designet et 7-segment display, som viser brugeren den aktuelle BPM vha. BCD. Da de forskellige input til trommemaskinen nu er redegjort, vil der blive undersøgt, hvordan disse instruktioner skal behandles af sequenceren.

5.2 Design af sequencer

Sequenceren har til formål at kunne afspille en brugerprogrammeret rytme i et brugerspecificeret tempo. Disse instruktioner for skal sequenceren tage som input fra kontrolmodulet og afgive instruktioner til lydmodulet når en trommelyd skal afspilles. Blandt instruktionerne fra kontrolmodulet befinder der sig signaler fra realtidstrommeknapperne, hvilket bliver behandlet af en sampler og indsat i sequenceren.

5.2.1 Sequencerens grænseflader og signalernes funktionalitet

Sequenceren er det eneste af de tre delmoduler uden eksterne grænseflader. På figur 5.7 ses moduldiagrammet for sequenceren, hvor signalerne på højre side kommer fra kontrolmodulet, og signalerne på venstre side leder til lydmodulet.



Figur 5.7: Moduldiagram for sequencerens med dens indgangs- og udgangssignaler.

Det ses at alle indgangssignalerne stammer fra kontrolmodulet. Her skal sequenceren programmeres ud fra dataet i step-sequenceres dataarray lave lækker label. Den skal derudover læse signalerne fra realtidstrommeknapperne, og gemme/afspille disse på det rigtige tidspunkt.

Sequenceren modtager også 3 kontrolsignaler som inddeles i tre funktionaliteter: BPM-signal, slettefunktion og modeselect.

Ved ændring i BPM-signalet skal sequenceren enten øge aller mindske tiden for gennemgang af én sekvens. Dette signal bliver dannet og beskrevet i underafsnit endnu et flot label.

Med stimuli fra slettesignalet skal det være muligt at fjerne alt programmeret data fra den valgte af de fire sekvenser.

Modeselectsignalet giver brugeren mulighed for "fri leg". Den afgør om brug af realtidstrommeknapper skal gemmes i den nuværende sekvens samtidig med afspilning af lyden, eller om det blot skal afspilles.

Sequencerens nuværende lokation i en sekvens sendes, som signal, til kontrolmodulet. Dette bruges til taktlokations-LED'erne.

Derudover er sequencerens udgangssignaler instruktioner til lydmodulet. Som det ses på figur 5.7 består dette af fire signaler der hver repræsenterer én trommelyd. Hvor gang der ønskes at afspille en trommelyd skal disse signaler kortvarigt sættes høje som beskrevet i afsnit label til qwerTheSecond eller opsummering af lydmodul.

5.2.2 Sekvensopbygning

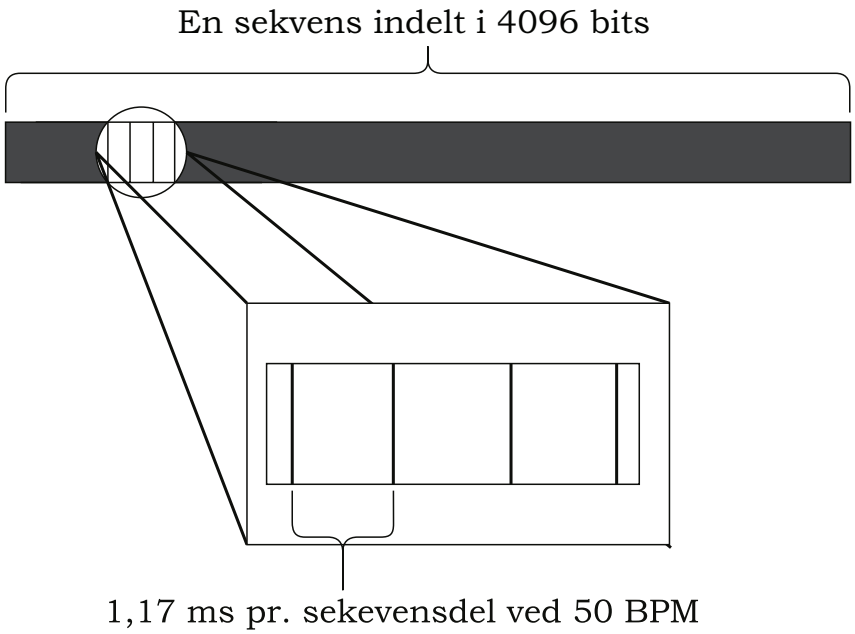
En sekvens består af et antal pladser hvor det er muligt at indkode lydafspilninger der består af information om hvorvidt hver er de fire trommer skal afspilles på denne plads. Antallet af pladser kaldes for sekvensens opløsning. For at finde denne opløsning udregnes der først hvor lang tid det skal tage at gennemgå én sekvens. Da der rytmisk set er valgt at spille i $\frac{4}{4}$ medfører dette 4 taktslag for én sekvensgennemgang. For at udregne gennemløbningstiden for én sekvens, tages der igen udgangspunkt i 100 BPM. Dette giver at:

$$T_{100BPM} = \frac{60\ \frac{\text{s}}{\text{minut}} \cdot 4\ \frac{\text{taktslag}}{\text{takt}}}{100\ \text{BPM}} = 2{,}4\,\text{s} \tag{5.2.1}$$

Ved en BPM på 50, hvilket er den nedre hastighedsgrænse, medfører dette en gennemløbstid på 4,8 s. Kravet til sequencerens opløsning er at indkodning fra realtidstrommerne maksimalt må komme 1,6 ms før eller efter brugerens reélle taktslag. Der forsøges at findes den lavest mulige værdi for denne opløsning således at kravet overholdes, men da opløsningen implementeres binært er skal opløsningen gå op i en to'er potens. Her vælges der en opløsning på 4096 hvilket giver en præcision på:

$$T_{prcision} = \frac{4,8\ \text{s}}{4096} = 1,17\ \text{ms} \tag{5.2.2}$$

Det ses at præcisionen er inden for kravet, og derfor arbejdes der videre med denne opløsning. På figur 5.8 ses den tidsmæssige struktur for sekvensopbygningen. Det er her muligt at placere en lydafspilning mellem hver af de indikerede streger.



Figur 5.8: Visualisering af en sekvens ved 50 BPM.

Sekvensens pladser skal gennemløbes i et specifikt jævnt tempo som bestemmes af BPM'en. Derfor fremstilles der en variable clock som justeres af BPM-signalerne fra kontrolmodulet. Da denne clock fremstilles ud fra Papiliones 32 MHz oscillator, skal der findes et forhold mellem oscillatoren, sekvensopløsningen og gennemløbstiden for en sekvens, som ændres ved ændring af BPM. Her tages der igen udgangspunkt i 100 BPM, hvorved forholdet mellem oscillatoren og den ønskede clock udregnes til:

$$\frac{32\ \text{MHz} \cdot 2,4\ \text{s}}{4096} = 18750 \tag{5.2.3}$$

I kodeeksempel 5.8 ses der hvordan denne clock implementeres. Her er "speedSignal"en variable der ændres af BPM-signalerne fra kontrolmodulet, således at clockens frekvens ændres.

For at opnå dette sammenhæng, tages følgende kode i brug:

Kodeeksempel 5.8: Clocken der holder rytmen

```
1 if rising_edge(clk) then
2   count := count + 1;
3   -- See if count needs to be reset
4   if count >= speedSignal then
5     sequencerClk <= not sequencerClk;
6     count := 0;
7   end if;
8 end if;
```

Det ses på linje 5 i kodeeksemplet at udgangssignalet, "sequencerClk", kun inverteres hver gang der bliver talt op til variablen, "speedSignal". Derfor skal forholdet fra formel 5.2.3 halveres for at kunne danne clocken ved 100 BPM.

Det huskes fra underafsnit 5.1.4 på formel 5.1.2 at BPM'en kan udregnes som:

$$BPM = \frac{speed_0 \cdot 100}{speed} \tag{5.2.4}$$

Med udregning af clockforholdet ved 100 BPM, er det nu muligt at bestemme de øvre og nedre grænser for BPM'en der skal bruges i kodeeksempel 5.6. Ved 100 BPM skal der tælles 9375 clockcyklusser på Papilioens 32 MHz clock. Dette er $speed_0$. Den nedre grænse på 50 BPM kan derved udregnes til det dobbelte som er 18750, og 200 BPM kan udregnes til 4687,5. Da dette ikke er et heltal vælges der at runde ned, som giver en øvre grænse på 4687. Da dette ikke er en nøjagtig BPM på 200, undersøges der om den maksimale BPM ligger inden for kravet om en maksimal afvigelse på ± 1 BPM. Den maksimale BPM udregnes til:

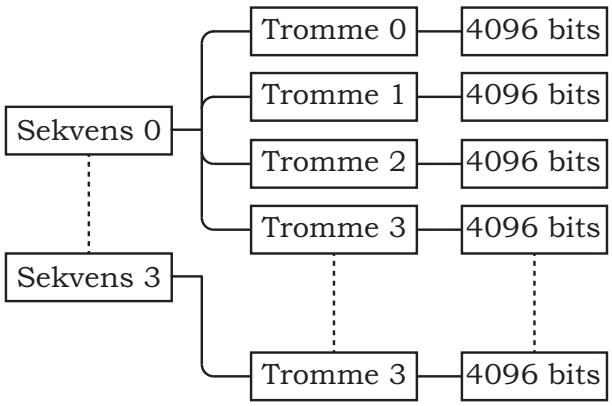
$$BPM_{max} = \frac{9375 \cdot 100}{4687} = 200,02 \text{ BPM} \tag{5.2.5}$$

Det ses at dette er inden for kravet, så der undersøges ikke yderlige løsninger.

5.2.3 Datastruktur

Ved ikke om følgende er skrevet før: Sequenceren har til opgave at holde styr på de enkelte sekvenser for de forskellige trommelyde. For at holde styr på dette skal der brugen en række input fra kontrolmodulet. For det første skal der bruges et input der fortæller hvilken sekvens der er valgt. Derudover skal der bruges input, der fortæller hvilke dele af en sekvens der skal spille. Hvis der spilles i realtid skal også der bruges et input, der fortæller hvornår, der bliver trykket på en trommeknap. For at bestemme hvilken plads i arrayet der skal opdateres bruges den tidligere beregnede clock. Clock'en er designet til at styre en counter, som går fra 0 til 4095, bruges denne counter til at bestemme hvilken plads i dataarrayet det skal tilgås. Da der bruges fire knapper, en for hver tromme, styrer de tilhørende signaler hvilken tromme man tilgår. Ligesom tidligere vælges sekvensen med sit eget signal.

På figur 5.9 ses en illustration for sequencerens opdeling af data.



Figur 5.9: Diagram over datastrukturen for blok-RAM

Det ses her at på grund af de fire sekvenser, som hver inderholder data fra de fire trommer, skal der kunne gemmes 16 forskellige sekvenser. Den nødvendige mængde lagerplads til sequencerens data regnes til at være:

$$Size_{sequencerData} = 16 \cdot 4096 = 65536 \text{ b} = 8192 \text{ B} \tag{5.2.6}$$

Denne mængde data kan ikke implementeres som flip-flops, så der vælges et benytte Papilioens blok-RAM til at gemme dataen.

Da det nu er bestemt hvordan dataen skal struktureres, kan det nu sættes op, hvordan blok-RAM'ene skal tilgås. Dette kan gøres som set på kodeeksempel 5.9.

Kodeeksempel 5.9: Opsætning af blok-RAM

```
1  TYPE ram_t is array (0 to 4095) of STD_LOGIC;
2  signal ram : ram_t := (others => '0');
3  process(clk)
4  begin
5    if rising_edge(clk) then
6      if we='1' then
7        ram(address) <= dataIn;
8      end if;
9      dataOut <= ram(address);
10   end if;
11 end process;
```

Der startes med at definere strukturen, som RAM'ene skal tilpasses. Dette gøres ved at definere den type RAM der ønskes at bruge, som her er et array af bits, som er 4059 langt. Herefter oprettes selve signalet. Med disse linjer kode burger ISE kodeeksempler til at oprette forbindelse til de interne blok-RAM under syntese processen. For at sætte dette op skal der bruges en række input til RAM'ene. For det første skal de vide hvilken adresse der tages i brug. Dette bestemmes ud den tidligere nævnte counter. Derudover har blok-RAM'ene brug for at et input signal, i dette tilfælde dataIn, der fortæller hvad, der skal stå på de enkelte adresser. For at bestemme om der skal skrives til blok-RAM'ene eller ej bruges signalet we. Hvis denne er høj bliver den ønskede data indsat på adressen. Dataen på RAM'ene bliver dog altid læst. Dette er for at sikre, at man kan læse om der skal afspilles en lyd på et hvilket som helt givet tidspunkt. Strukturen er skrevet i et modul for sig selv, så det kan tilgås flere gange for at oprette forbindelse til alle de nødvendige RAM-blokke. Dette kan ses i kodeeksempel 5.10.

Kodeeksempel 5.10: Oprettelse af en bestem RAM-blok

```
1  ram00: entity work.RAM_module
2    port map (
3      clk=>clk, address=>address0, we=>we0(0), dataIn=>dataIn(0), dataOut=>dataOut0(0)
4    );
```

I overstående kodeeksempel ses det hvordan man opretter en bestemt RAM-blok. Adressen der bliver sat ind, er bestemt af clock'en der holder styr på hvor langt i sekvensen man er nået. we0 referere til en bestemt tromme. Dette vil sige at man kan ændre we0 til høj hvis man vil skrive til den første tromme. Dette signal er lavet som en STD_LOGIC_VECTOR fra tre ned til nul, så inputtet, der bestemmer den valgte sekvens, kan bruges til at bestemme hvilken RAM-blok man gerne vil skrive til. Det vil sige at we0(1) kan ændres til høj for at skrive til til den anden sekvens. Det betyder også at we2 bruges til at skrive til den tredje tromme, så man ligeledes kan kontrollere dataen her. dataIn er også lavet som en STD_LOGIC_VECTOR fra tre ned til nul. Her høre dataIn(0) til den første tromme. Dette bruges så man sender den samme data til alle RAM-blokke, der høre til den samme tromme, men der åbnes kun for we, for den sekvens man gerne vil sende til. dataOut er lavet på samme måde som we, så dataOut1 høre til RAM-blokken for den anden tromme. Når dataen så skal behandles, kan sekvens signalet igen bruges til at tilgå outputtet for en bestem RAM-blok. For at demonstrere hvordan der skrives til blok-RAM kan der ses på kodeeksempel 5.12.

Kodeeksempel 5.11: Signalopsætning for at skrive til en RAM-blok

```
if rising_edge(clk) then
  if yesRun(0) = '1' then
      case realState0 is
        when 0 =>
          dataIn(0) <= '1';
          tickCount0 := 0;
          we0(seqVar) <= '1';
          realState0 := realState0 + 1;
        when 1 =>
          tickCount0 := tickCount0 + 1;
          if tickCount0 = 4 then
            realState0 := realState0 + 1;
          end if;
        when 2 =>
          we0(seqVar) <= '0';
          dataIn(0) <= '0';
          realState0 := realState0 + 1;
        when 3 =>
          yesRun(0) <= '0';
          realState0 := 0;
    end case;
  end if;
end if;
```

For at skrive til blok-RAM'ende er der valgt at tage udgangspunkt i en tilstandsmaskine. Først ventes der på at signalet yesRun bliver sat til høj så de forskellige tilstade kan gennemgås. Signalet yesRun bliver sat til høj når der modtaget et input fra en af realtidstrommeknapperne. Når dette sker starter tilstandsmaskinen med den første tilstand. Her bliver we og dataen sat til høj, så blok-RAM'ene kan opdateres. I anden tilstand ventes der på at en counter. Dette gøres for at give RAM-modulet lidt tid til at opdatere dataen på blok-RAM'enen. Herefter sættes dataIn og we til lav, da der ikke længere ønskes at skrive til RAM'ene. Til sidst sættes tilstanden tilbage til start og signalet der sætter processen i gang sættes til lav, så der kan ventes på et nyt input fra realtidstrommeknapperne. Denne funktionalitet, kombineret med de tidligere nævnte funktioner, gør det muligt at skrive til blok-RAM'ene og derved holde styr på real tids trommerne.

Den sidste funktionalitet der ønskes af blok-RAM'ene, er at kunne rydde signaler. Opsætningen for dette kan ses i understående kodeeksempel:

Kodeeksempel 5.12: Signalopsætning for at skrive til en RAM-blok

```
if rising_edge(clk) then
  tickCount1 := tickCount1 + 1;
  if tickCount1 = 1 then
    we1(seqVar) <= '1';
  elsif tickCount1 < 4098 then
    address1 <= (tickCount1 - 2);
  else
    we1(seqVar) <= '0';
    yesRun(1) <= '0';
    realState1 := 0;
    tickCount1 := 0;
  end if;
end if;
```

For at slette data på blok-RAM'ene skal alle de forskellige adresser sættes til nul. For at kontrollerer dette anvendes en counter, som bruges til at tilgå de enkelte adresser. Ligesom i kodeeksempel 5.12, sættes we til høj for at der kan skrives til blok-RAM'ene. Da dataen, der skal sættes ind, altid er nul når der ikke indsættes et signal, behøves denne variable ikke at blive opdateret. Herefter gennemgås adresserne en efter en, så der kan blive sat nuller ind her. Til sidst nulstilles de forskellige variabler, så processen er klar til at blive brugt igen.

5.2.4 Implementering af sequnecer

Da datastrukturen og kontrol af dette nu er beskrevet, vil der blive gennemgået hvordan de forskellige signaler bliver behandlet. Dette vil også inkludere, hvordan det bestemmes hvad der skal spille på hvilket tidspunkt, og hvordan der bliver givet besked til lydmodulet om dette.

Realtidstromme uden RAM interface

Da der er opsat et krav om at det skal være muligt at tromme i realtid, uden at dette bliver gentaget i sekvensen, skal der bruges et signal, til at fortælle om dataen skal gemmes eller ej. Dette signal kommer, ligesom de andre, fra kontrolmodulet. Der er her tale om modeselect. Hvis dette signal er sat til høj, kan brugeren tromme på knapperne, og blot afspille trommelyden med det samme. Dette styres som i følgende kodeeksempel:

Kodeeksempel 5.13: Behandling af input fra realtidstrommeknapper

```
1  if rising_edge(clk) then
2    if dbRealTimeButton(0) = '1' then
3      beforeButton(0) <= '1';
4    else
5      beforeButton(0) <= '0';
6    end if;
7    if (dbRealTimeButton(0) and (not beforeButton(0)) and (not modeButton)) = '1' then
8      goNow(0) <= '1';
9    elsif (dbRealTimeButton(0) and (not beforeButton(0)) and modebutton) = '1' then
10     freeDrum(0) <= '1';
11   else
12     goNow(0) <= '0';
13     freeDrum(0) <= '0';
14   end if;
15 end if;
```

Da processen, som skriver til blok-RAM'ene, også køre på grundklokken, skal signalet, som starter processen, sendes som en puls, så den kun igangsættes en gang. Derfor ses der først på tilstanden for inputtet fra realtidstrommeknapperne. Her skal der kun sendes et signal videre hvis dbRealTimeButton går fra lav til høj. Derfor gemmes den forrige tilstand for knapsignalet i en variable, som kan bruges til at sammenligne med. For at sammenligne signalerne, ses der på knapsignalets nuværende tilstand, knapsignalets tidligere tilstand og modeselect signalet. Hvis den knapsignalet er høj i den nuværende tilstand og lav i den forrige, ved man at knappen er blevet trykket ned. Hvis modeButton er lav opdateres goNow, som opdatere signalet på blok-RAM'ene. Hvis modeButton er høj, sendes et signal til en process, som afspiller en trommelyd med det samme, uden at gemme dataen.

Output signal til lydmodulet

Det sidste led i sequenceren, er at sende et signal til lydmodulet, så det kan afspille en trommelyd. I den forbindelse skal der sammenlignes tre forskellige signaler. Et for step-sequneceren og et for realtidstrommeknapperne og et fra blok-RAM'ene. Først sammenlignes blok-RAM'enes signal og step-sequencerens signal, da begge disse skal sendes med udgangspunkt i en clock. Dette gøres på følgende måde:

Kodeeksempel 5.14: Sammenligning af blok-RAM og stepsequencer

```
1  if rising_edge(sequencerClk) then
2    case timeNow is
3      when 0 =>
4        sequencerOutput(0) <= (data(seqVar)(0)(0) or dataOut0(seqVar));
5        sequencerOutput(1) <= (data(seqVar)(1)(0) or dataOut1(seqVar));
6        sequencerOutput(2) <= (data(seqVar)(2)(0) or dataOut2(seqVar));
7        sequencerOutput(3) <= (data(seqVar)(3)(0) or dataOut3(seqVar));
8      when 256 =>
9        sequencerOutput(0) <= (data(seqVar)(0)(1) or dataOut0(seqVar));
10       -- Repeat for every 256. value of timeNow
11     when others =>
12       sequencerOutput(0) <= dataOut0(seqVar);
13       sequencerOutput(1) <= dataOut1(seqVar);
14       sequencerOutput(2) <= dataOut2(seqVar);
15       sequencerOutput(3) <= dataOut3(seqVar);
16   end case;
17 end if;
```

Det kan ses i kodeeksempel 5.14 at der skal laves et signal for hver tromme. Disse er repræsenteret med sequencerOutput, som fortæller om der skal afspilles en tromme. Der ses på om enten step-sequenceren eller blok-RAM'ene er høj, for en givet sekvens på et givet tidspunkt. Disse sammenlignes kun for hver 256. tidspunkt i timeNow, hvilket er et signal, som er en integer, der går op til 4095. Det det første punkt i timeNow er nul. Her ses der på den første del af step-sequencerens signal. Ligeledes ses der på den andet punkt ved 256. For blok-RAM'ene bruges timeNow også til at tilgå en specifik adresse. Dette medføre at dataOut bliver opdateret synkront med denne process og derfor kan sendes direkte videre, hvis en plads er høj. Hvis timeNow ikke er på en af de 256. værdier, behøves der ikke blive sammenlignet med step-sequnecer-signalet og signalet fra blok-RAM'ene sendes direkte videre. Signalet sequencerOutput kan så blive behandlet og sendt videre til lydmodulet, som set i kodeeksempel 5.15.

Kodeeksempel 5.15: Opsætning for signalet til lydmodul

```
1  if rising_edge(clk) then
2    current <= sequencerOutput;
3    previous <= current;
4    if (((not previous(0)) and current(0)) or freeDrum(0)) = '1' then
5      sequencerInput(0) <= '1';
6    else
7      sequencerInput(0) <= '0';
8    end if;
9  end if;
```

For at sikre at lydmodulet kun kalder afspilningsprocessen en gang, ønskes der her at signalet der igangsætter processen, kommer som en puls. Dette håndteres, ligesom i kodeeksempel 5.13,

ved at se på det nuværende og det forrige stadie for signalet. Hvis det går fra lav til høj, sendes en puls til lydmodulet så der kan blive afspillet en trommelyd. Det er også i denne process at der tages højde for realtidstrommen, hvis der ikke skal gemmes på blok-RAM'ene. Da signalet i forvejen kommer som en puls, kan der blot ses på om det er højt eller lavt. Hvis ikke der skal afspilles en trommelyd, holdes signalet bare lavt.

5.2.5 Opsummering af sequenceren

Med de tidligere beskrevne funktioner, er sequenceren nu færdigdesignet og klar til at blive implementeret. Den modtager en række signaler fra kontrolmodulet, der fortæller hvilke trommer der skal afspilles på hvilket tidspunkt. Nogle af disse trommer følger en bestemt rytme, hvor andre er styret af brugeren selv. Sequenceren behandler så disse signaler og gemmer den nødvendige data. Herefter sendes et signal til lydmodulet, der giver besked om at en trommelyd skal afspilles.

5.3 Opsumering af design

Med disse tre delmoduler designet og implementeret, burde alle funktionaliteter i trommemaskinen kunne realiseres.

Alt VHDL er samlet under et VHDL-modul, og implementeret på Papilio Duo'en. Det samlede VHDL-kode benytter kun 14% "slice registre"(1.683 ud af 11.400) hvor de fleste blot anvendes som normale flip-flops. Med hensyn til slice'enes look-up-tables bliver 60% udnyttet (3.468 ud af 5.720). Langt de fleste af disse LUT'er bliver brugt til at implementere logik.

På kontrolmodulet benyttes 36 af spartan-6'ens I/O-porte til brugerinterfacet, og lydmodulet bruger 3 porte til at sende I^2S til D/A-konverteren. Dette vil sige at, ud af Papilio'ens 54 anvendelige I/O-ben, er der 15 porte der stadig er frie.

Med designet af trommemaskinen på plads, vil der nu kigges på, om trommemaskinen er i stand til at opfylde opstillede krav.