NTNU
Kunnskap for en bedre verden

Simulering av ultralydstransdusere ved bruk av en FPGA.

Vegard Broch
Gunnar K. Hagen
Vegar Sande

Bachelor i Elektro
Innlevert: mai 2021
Hovedveileder: Knut Wold

Norges teknisk-naturvitenskapelige universitet
Institutt for elektroniske systemer

NTNU
Kunnskap for en bedre verden

Oppgavens tittel:	Dato: 19.05.2021			
«Simulering av ultralydstransdusere ved bruk av en FPGA»	Antall sider: 57			
	Masteroppgave:		Bacheloroppgave	X
Navn: Vegard Broch, Gunnar K. Hagen og Vegar Sande				
Veileder: Knut Wold				
Eventuelle eksterne faglige kontakter/ veiledere: Didier Siboniyo				

Kort sammendrag:
Denne oppgaven går ut på å utvikle og programmere et testinstrument for firmaet Dolphitech. Dette testinstrumentet skal brukes for å redusere mengden skadede komponenter ved prøvemontering. Instrumentet konstrueres for å etterligne en ultralydstransduser, og kobles til Dolphitechs eksisterende elektronikk for å verifisere at denne gir ut og mottar signalene den er designet for. Bakgrunnen av dette prosjektet kommer av at en rekke transdusere og elektronikk har blitt ødelagt under testing tidligere, og dette er kostbart. Ved bruk av denne programvaren koblet til elektronikken, vil elektronikken kunne verifiseres som fullt fungerende eller ikke. Ved feil etter montering av funksjonell elektronikk og en transduser, vil det være naturlig å anta at feilen ligger i ultralydstransduseren.

Det har blitt benyttet en FPGA, og denne er programmert i VHDL. Selve programmet er detaljert godt i oppgaven, og burde gi Dolphitech gode forutsetninger for å videreutvikle systemet. Under testing fungerer systemet godt, og er en god tilnærming til en transduser for testingen. Tilkoblingskretsen ble ikke realisert fysisk, men er detaljert teoretisk og med forslag til utførelse i oppgaven.

Stikkord:

Signalbehandling
Skadebegrensning
Ultralydstransduser
FPGA
VHDL
Testbenk

_____

(sign.)

Forord

Med denne oppgaven konkluderes vårt treårige bachelorstudium i Elektronikk ved Norges teknisk-naturvitenskapelige universitet (NTNU). Oppgaven er utført ved institutt for elektroniske systemer gjennom vårsemesteret 2021, og krediteres med 20 studiepoeng. Bachelorgruppen består av Vegar Sande, Gunnar Kvesetberget Hagen og Vegard Broch.

Denne bacheloroppgaven baserer seg på banebrytende teknologi innenfor NDT (Non Destructive Testing) og ultralyd. Grunnet dette har gruppen holdt seg motivert og arbeidet målrettet igjennom perioden, for å få et så godt sluttresultat som mulig.

Oppgaven er gitt av Dolphitech AS. Bedriften har også stått for FPGA og finansieringen av prosjektet. Oppgaven skal løse et problem ved testmontering av ultralydstransdusere på en spesiallaget elektronikk. Det er her rom for feilmontering og skader. Gruppen skal derfor forsøke å programmere en FPGA til å ligne på en transduser, slik at elektronikken kan verifiseres og testes alene.

Vi ønsker å rette en takk til Dolphitech, og bedriftskontakt Didier Siboniyo som har gitt oppgaven, og et godt samarbeid underveis i prosessen. Knut Wold, som har vært gruppas veileder fra NTNU, har hjulpet med spørsmål som har kommet opp under arbeidet.

Sammendrag

Oppgaven er gitt av Dolphitech AS, og baserer seg på et problem firmaet har opplevd. Problemet Dolphitech har opplevd går ut på at de ikke har noe testinstrument som kan benyttes på den eksisterende elektronikken eller ultralydstransduseren. Dette fører til at funksjonstesting av de forskjellige komponentene først skjer ved montering, og her ligger det en moderat mekanisk risiko for å ødelegge begge komponenter.

Formålet med oppgaven er å utvikle og programmere et testinstrument som etterligner en transduser. Dette vil kunne benyttes til å verifisere TRM elektronikken. Testinstrumentet vil gå igjennom TX- og RX-linjene og rapportere tilbake om de fungerer som tiltenkt eller har en feil. Dette vil kunne gjøres før monteringen av transduseren, og vil føre til færre ødelagte transdusere ved av- og påmontering ved testing.

Denne oppgaven er delt opp i tre hoveddeler. Den første delen omfatter teorien som ligger i bakgrunnen for hele prosjektet. Den andre delen tar for seg metodene som er benyttet igjennom prosjektperioden, og er en dokumentasjon på gruppas tankegang og fremgangsmåter. Den siste delen omhandler resultater, analyse og diskusjon av prosjektet, og presenterer gruppas funn igjennom prosessene.

Hovedtyngden av oppgaven ligger på programvaren som er implementert i FPGA-en, og hvordan denne fungerer. I korte trekk vil TRM-en sende ut pulser til FPGA-en. Deretter behandles signalene i programvaren, og FPGA-en sender tilbake signaler som imiterer transduserens respons til TRM-elektronikken.

Tilkoblingskretsen ble ikke realisert fysisk, og er derfor å anse som forslag til utførelse. Det er nødvendig å redusere spenningen ut fra TRM-en slik at FPGA-en ikke tar skade. Samt redusere spenningen ut fra FPGA-en slik at mottakerelektronikken ikke tar skade. Dette er planlagt gjennomført med et tilkoblingskort. Kretskortet er planlagt med påkobling av TRM-ens fleksfilm, hvor hver linje går igjennom en spenningsdeler for å oppnå korrekte spenningsnivåer.

Det konkluderes med at problemstillingen er oppnådd delvis teoretisk i form av tilkoblingskretsen, og delvis fysisk med tanke på programvaren. Dette vil gi et godt grunnlag for videre utvikling av Dolphitech.

Abstract

The assignment is given by Dolphitech and is based on a problem the company has experienced. Dolphitech has experienced issues stemming from the fact that they do not have a test instrument that can be used on the existing electronics or the ultrasonic transducer. This means that function testing of the various components takes place during assembly, and there is a moderate mechanical risk of destroying both components.

The purpose of the thesis is to develop and program a test instrument that mimics a transducer. This can be used to verify the TRM electronics. The test instrument will go through the TX and RX lines and report back if they work as intended or have an error. This can be done before mounting the transducer and will lead to fewer damaged transducers during installation and removal during testing.

This thesis is divided into three main parts. The first part covers the theory behind the whole project. The second part describes the methods used throughout the project and is a documentation of the group's thought process and procedures. The last part describes the results, analysis, and discussion of the project, and presents the group's findings through the process.

The main focus of the assignment is on the software implemented in the FPGA and how it works. In short, the TRM will send out pulses to the FPGA. The signals are then processed in the software, and the FPGA returns signals that mimic the transducer's response to the TRM electronics.

The connection circuit was not physically realized and is therefore to be regarded as a proposal for execution. It is necessary to reduce the voltage from the TRM so that the FPGA is not damaged. As well as reducing the voltage from the FPGA so that the receiver electronics are not damaged. This is planned to be done with a connection card. The circuit board is planned with the connection of the TRM's flex film, where each line passes through a voltage divider to achieve correct voltage levels.

It is concluded that the problem has been achieved in part theoretically in the form of the connection circuit, and in part physically with regards to the software. This will provide a good basis for further development for Dolphitech.

Innholdsfortegnelse

1 Introduksjon 1
1.1 Bakgrunn 1
1.2 Oppgavebeskrivelse 2
1.3 Problemstilling 3
1.4 Avgrensninger 3
1.5 Rapportens oppbygning 4
2 Teori og Bakgrunn 5
2.1 Resistanser 5
2.2 Transdusere 6
2.3 Dolphicam2 6
2.3.1 TRM 6
2.3.2 Dolphitechs Transdusere 7
2.4 FPGA 8
2.4.1 Altera Cyclone V 8
2.5 VHDL 9
2.5.1 Virkemåte 9
2.5.2 Quartus Prime Lite Edition 10
2.5.3 ModelSim 10
2.5.4 Testbenk 11
3 Metode 12
3.1 Kretsoppbygning 12
3.1.1 Spenningsdeling 12
3.1.2 Tilkoblingskrets 14
3.2 Quartus Prime 15
3.3 Testing og feilsøking 16
3.3.1 Testbenk 16
3.3.2 Simulering 17
3.3.3 ModelSim 18
4 Design og implementasjon 19
4.1 Programvarens funksjon 19

4.1.1 Programvaren 19
4.1.2 Testbenk implementasjon 23
5 Programvaretesting og simulering 27
5.1 FPGA resultater 27
5.2 Simulasjon 33
5.2.1 Simulasjon – Resultater 33
6 Diskusjon 35
6.1 Teorien brukt i prosjektet 35
6.2 Refleksjon 36
6.2.1 Utvikling av programvaren 36
6.2.2 Utvikling av testbenken 36
6.3 Vurdering av Metoder og løsninger 37
6.4 Fremtidig arbeid 38
6.4.1 Kretsutvikling 38
6.4.2 Programvare 39
7 Konklusjon 40
Kilder 41
Vedlegg 42

Figurliste

Figur 2. 1 Bilde av Altera Cyclone V brukt i prosjektet. Bilde tatt av Vegar Sande (14.04.2021) 8

Figur 3. 1 Blokk skjema av komplett krets 38
Figur 3. 2 Knapper på FPGA-en 38
Figur 3. 3 Bryterne brukt til å endre frekvenser under testing 39
Figur 3. 4 Figuren viser når Key1 og Key0 er aktiv og inaktiv 40
Figur 3. 5 Figuren viser når SW2, SW1 og SW0 er aktiv eller og inaktiv 41

Figur 4. 1 Sample viser hvordan klokke og puls generatoren fungerer. Utklipp fra programvaren 21
Figur 4. 2 Utklipp av programvaren, prosess p_BB 22
Figur 4. 3 Visuell representasjon av FPGA-ens programvare 22
Figur 4. 4: Oppsettet på testbenken for å koble den opp mot hovedprogramvaren 23
Figur 4. 5: Prosessen som genererer klokke signalet 24
Figur 4. 6: Prosessen som setter reset signalet 24
Figur 4. 7: Prosedyren som initialiserer signalene i simulasjonen 25
Figur 4. 8: Prosedyre som med tidsmellomrom stiller inn c_Sample til alle mulige tilstander 25
Figur 4. 9: Prosedyre for aktivering av knapp 26
Figur 4. 10: Testbenkens arkitektur 26

Figur 5.1 Utgang signal uten Start signal. Key1 = 1 27
Figur 5.2 Utgang signal for 1.5MHz med start signal. SW2, SW1 og SW0 = «000» 28
Figur 5. 3 Utgang signal for 2.5MHz med start-signal. SW2, SW1 og SW0 = «001» 29
Figur 5.4 Utgang signal for 3.5MHz med start-signal. SW2, SW1 og SW0 = «010» 29
Figur 5.5 Utgang signal for 5MHz med start-signal. SW2, SW1 og SW0 = «011» 30
Figur 5.6 Utgang signal for 6MHz med start-signal. SW2, SW1 og SW0 = «100» 30
Figur 5.7 Utgang signal for 8MHz med start-signal. SW2, SW1 og SW0 = «101» 31
Figur 5.8 Utgang signal for 10MHz med start-signal. SW2, SW1 og SW0 = «110» 31
Figur 5.9 Utgang signal når en bryter kombinasjon som ikke er angitt blir valgt. SW2, SW1 og SW0 = «111» 32
Figur 5.10 Utgang signal når reset og start knappen er ned trykket. For å teste nullstillings funksjonen til programvaren Key0 = 0 og Key1 = 0 32
Figur 5. 11: Simuleringsresultat fra testbenken viser hvordan frekvensen kan endres 33
Figur 5. 12: Hvordan c_Sample endres demonstrert 34
Figur 5. 13: Simulasjon av programvaren viser knappens funksjonalitet 34

Tabeller

Tabell 4. 1 Tabell over verdiene p_Speed opererer med..... 20

Ordliste	
Testbenk	En programvare som blir brukt for simulering av VHDL programvarer. Tester spesifikke deler av programvaren og er designet for testing av bestemte programvarer.
Kompilere	Å oversette programmeringsspråk til objektivkode.[1]
Hardware	Maskinvare, elektrisk krets.
Tx	Transmitter (Sender).
Rx	Receiver (Mottaker).
BlackBox	Dolphitechs navn på en elektrisk enhet som behandler signaler, mellom skjermen og TRM.
TRM	Dolphitechs forkortelse for «**TR**ansduser **M**odule». Dette er den håndholdte proben som brukes under skanning.
NDT	Forkortelse for «**N**on-**D**estructive **T**esting», er et samlebegrep for overflatetesting av materialer som ikke skader materialet i seg selv.
VHDL	«**V**HSIC **H**ardware **D**escription **L**anguage» - Programmeringsspråk beregnet for å beskrive struktur og oppførsel til logiske kretser.
VHSIC	**V**ery **H**igh-Speed **I**ntegrated **C**ircuit
FPGA	Field Programmable Gate Array
Logiske elementer	En del av FPGA-ens design som utfører en arbeids oppgave. Disse delene blir lagt sammen for å utføre kalkulasjoner eller andre arbeids oppgaver.
Lavt (Signal)	Definerer at et signal er inaktive eller ca. lik 0V
Høyt (Signal)	Definerer at et signal er aktive eller ca. lik 3.3V
Up pull	Up pull gjør at pinner som er udefinerte får en viss spennings verdi.
IDE	Integrated Development Environment. En applikasjon som kan brukes til å utvikle programvare.

1 Introduksjon

1.1 Bakgrunn

Oppgaven er gitt av Dolphitech AS og gjennomføres i tett samarbeid med bedriften. Dolphitech ble startet i 2009 med en idé om å benytte ultralyd for å lese merking og dokumentasjon som var tildekt av lakk eller maling. Etter deres suksess innenfor ultralydteknologi skjønte de at teknologien de hadde kunne bli benyttet til andre formål. Dolphitech begynte å benytte den samme teknologien som ble benyttet til å lese strekkoder, til å visualisere forskjellige materialers interne kvalitet. Denne teknologien ga Dolpitech muligheten til å oppdage interne skader i materialene. Dolphicam2 er deres nyeste produkt og brukes blant annet til å kontrollere fly, båter og biler etter interne skader ned til 1mm i størrelse.

Dolphitech AS er et selskap som arbeider med NDT (Non-Destructive Testing) av forskjellige materialer ved bruk av ultralyd. Bedriften har utviklet et system hvor det benyttes transdusere med 128 Tx-linjer, og 128 Rx-linjer i en matrise. Dette utgjør 16384 transduserelementer på en 30mm x 30mm flate, som er en ekstrem økning i forhold til eksisterende marked. Transduseren er montert i en håndholdt enhet (TRM, eller Transducer Module), sammen med et egendesignet kretskort.

Bakgrunn for oppgaven er å sjekke etter feil i elektronikken som sender og mottar pulser fra transduseren. Dolphitechs nåværende metode for å sjekke etter feil er å tilkoble transduseren til elektronikken, noe som er ømfintlig og kan forårsake skader om prosessen ikke gjøres forsiktig.

1.2 Oppgavebeskrivelse

Oppgaven går ut på å utvikle et produkt til bruk i kvalitetstesting av den eksisterende elektronikken i Dolphitechs håndholdte ultralydstransduser. TRM-en består av en transduser som produseres hos Dolphitech, og et intrikat kretskort for behandling av transduserens analoge signaler. Per dags dato testes dette systemet ved å montere en transduser på elektronikken, koble TRM-en til en blackbox, og en datamaskin. Deretter utføres en funksjonstest av systemet ved å skanne en referanseblokk og resultatene kommer i sanntid i en test-programvare på datamaskinen. Her kommer det tydelig frem om Tx eller Rx linjer er kortsluttet eller brutt, men ikke om problemet ligger i transduseren eller elektronikken. Det er også en moderat risk involvert i monterings og demonteringsprosessen, da dette utføres for hånd.

Målet med oppgaven er å konstruere et testinstrument som kan verifisere at elektronikkens Tx linjer sender signaler, og at Rx linjene mottar disse. Dette skal gjøres uten å montere en transduser for å minimere skader på disse. Om linjene i elektronikken er verifisert og det oppstår brutte eller kortsluttede linjer i test-programvare, kan en konkludere med at feilen må ligge i transduseren.

Resultatmål

- Verifisere et satt antall linjer, i både Tx og Rx delen av elektronikken.
- Ha en løsning som er skånsom ved montering, for å unngå ødeleggelser av elektronikk.
- Bidra til fremtidige kostnadsbesparelser i form av tid benyttet til testing, og ødeleggelse av elektronikk.

Effektmål

- Færre ødelagte deler ved prøvemontering.
- Enkel verifisering av elektronikk, slik at produksjonsfeil fra leverandør kan avdekkes og sendes tilbake.

1.3 Problemstilling

Oppdragsgiver ønsker en komponent som har muligheten til å koble til elektronikkdelen av TRM for å sjekke etter feil i Tx og Rx linjene, slik at den ømfintlige transduseren ikke må tilkobles før bruk. Gruppens mål er å løse problemstillingen når prosjektet er slutt eller kunne ha lagt et godt grunnlag for videre utvikling av oppdragsgiver. Gruppen tar utgangspunkt i å verifisere et utvalg av linjene i både Tx og Rx, og programmerer på en slik måte at dette kan skaleres opp om prinsippet fungerer. Dette gjøres for å ha et bedre grunnlag for å fullføre oppgaven, da den kan deles i mindre stykker.

Med dette som bakgrunn for prosjektet er følgende problemstilling valgt:

«Hvordan kan vi verifisere TRM-elektronikkens funksjonalitet uten å montere en transduser, på en måte som unngår å skade eksisterende elektronikk?»

1.4 Avgrensninger

Vi har satt følgende avgrensninger for prosjektet:

- Det er kun Tx og Rx linjene i elektronikken som skal testes, ingen andre deler av elektronikken.
- Prosjektet skal teste 1 Tx-linje og 1 Rx-linje, men programvaren skrives på en måte som lar oss skalere opp til et høyere antall linjer.
- Ingen endringer på eksisterende elektronikk, systemet burde være «plug and play».

1.5 Rapportens oppbygning

Rapporten følger standard IMRAD metode for å representere arbeidet gjort i dette prosjektet. Formålet med **I**ntroduksjon kapittelet er å etablere hva oppgaven er, avgrensninger gruppen har gjort og en grunnleggende beskrivelse av hvor oppgaven kommer fra.

I Kapittel 2 brukes problemstillingen og avgrensningen definert i kapittel 1 til å lage en ramme for informasjon om komponenter og programvarer brukt, med mer spesifikk forklaring i kapittel 3 **M**etode.

Kapittel 4 og 5 består av **R**esultat og **A**nalyse, hvor metodene forklart i tidligere kapitler blir testet og analysert. Kapittelet består av forklaringen av programvaren designet for å løse problemstillingen og hvilke resultater den oppnår ved bruk av en FPGA.

I kapittel 6 blir resultatene fra kapittel 4 og 5 vurdert og **D**iskutert for å forklare framgangen i prosjektet, om løsningen prosjektet har oppnådd løser problemstillingen og hva arbeidsgiver kan videreutvikle.

Rapporten avsluttes med en konklusjon av hele prosjektet i kapittel 7.

2 Teori og Bakgrunn

Dolphitech er ledende i utviklingen av NDT-utstyr. NDT er en forkortelse, som står for «Non-Destructive Testing»[2], og benyttes om utstyr til å måle feil i materialer, uten å demontere eller ødelegge testemnet.

2.1 Resistanser

Resistans er et begrep innen elektroteknikk, som beskriver elektrisk motstand. Resistans har vanligvis bokstaven R som symbol, og Ohm (Ω) som måleenhet [3]. Resistanser er elektriske komponenter som benyttes i de aller fleste elektriske kretser. De kan for eksempel benyttes til å endre spenningsnivåer ved å bruke to eller flere motstander i serie som en spenningsdeler. Resistanser finnes i varierende størrelser og utforminger til å passe enhver krets.

For spenningsdeling ved bruk av motstander gjelder følgende formel:

$$V_{ut} = V_{inn} * \frac{R_2}{R_1 + R_2} \quad (1)$$

Ved bruk av denne formelen kan spenningen ut av spenningsdeleren, ved en gitt inngangsspenning bestemmes ut ifra størrelsen på motstandene R_1 og R_2.

2.2 Transdusere

Sitatet omhandler transdusere fra store norske leksikon: «Transduser er et elektrisk, elektronisk eller elektromekanisk apparat som omformer én type energi til en annen.» [4]

Transdusere har derfor svært mange bruksområder, som for eksempel i: mikrofoner, høyttalere, sensorer eller og antenner. [4]

Dolphitech bruker transdusere for å gjøre om elektrisk energi, til ultralyd. Ultralyd benyttes fordi den beveger seg meget raskt igjennom materialer. Den vil reflekteres da den treffer bakveggen i materialet, en luftlomme eller et annet materiale som ligger innkapslet. Dolphitechs egne system måler tiden ultralydspulsene bruker før de reflekteres tilbake, og det kan da visualiseres et bilde av dybde, og feil i materialet.

2.3 Dolphicam2

Dolphicam2 er maskinvare-plattformen til Dolphitech. Den inneholder TRM-en, en BlackBox, og en Panasonic Toughpad. Dolphicam2 er en videreutvikling av Dolphicam, og er vesentlig mer kompakt og brukervennlig.

2.3.1 TRM

Transduser modulen (TRM-en) er enheten som digitaliserer den analoge dataen sendt fra transduseren. Denne digitale dataen blir videresendt til BlackBox-en der den blir analysert og resultatet presentert på en Panasonic Toughpad gjennom, Dolphitechs egenutviklede GUI (Graphical User Interface) i sann-tid.

TRM-en er en håndholdt gjenstand som brukeren kan dra over overflaten til et objekt for å skanne etter riper, hakk, innvendige skader og luftlommer. Før TRM-en tas i bruk trenger ultralyden et transmisjonsmiddel mellom transduseren og overflaten. Mens vann er optimalt rent akustisk, er ikke vann alene så mye brukt, da det kan være vrient og sikre at vannet fester seg til overflaten. Dolphitech anbefaler heller en gel blandet sammen med vann som smøres på overflaten før bruk.

2.3.2 Dolphitechs Transdusere

Transduseren er enheten som er ansvarlig for sendingen og mottakelsen av ultralyd-signaler via en sender-mottaker-matrise på 128x128 elementer som former totalt 16384 transduserelementer, og sitter montert i front av TRM-en. For å lage et puls-ekko-signal sendes en 70V puls på sender-elektrodene som danner et lyd-felt langs alle sender- og mottakerelektroder. Mellom mottaker- og sender linjene ligger det et piezoelektrisk materiale som genererer ultralyden.

Dolphicam 2 ser skader når ultralyden treffer et materiale med annerledes impedans enn materialet som skannes og reflekteres tilbake til transduseren. Dette kan for eksempel være en luftboble eller delaminering, som er den mest vanlige skaden på kompositttmaterialer. Når ultralydspulsen treffer en delaminering vil den reflekteres tilbake til transduseren på en kortere tid enn forventet. Dette visualiseres i «Time of Flight» bildet med et fargespekter. Det er her lett å tyde på hvilken dybde feilen ligger på og omfanget av skaden.

Dolphitechs egenproduserte transdusere kommer i et bredt spekter av frekvenser, for å kunne benyttes på mange forskjellige materialer. Lavere frekvenser vil kunne se igjennom tykkere materialer og kompositter, mens de aller høyeste frekvensene benyttes hovedsakelig til relativt tynne metaller. Dette kommer av materialets egenskaper for overføring av lyder. Det er også viktig for oppgaven og presisere at frekvensene som sendes fra BlackBox-en til TRM-en ikke trenger å være lik merkefrekvensen på TRM-en. Kravet den derimot må oppfylle er at frekvensen som sendes til TRM-en må være lavere enn den maksimale frekvensen TRM-en kan motta.

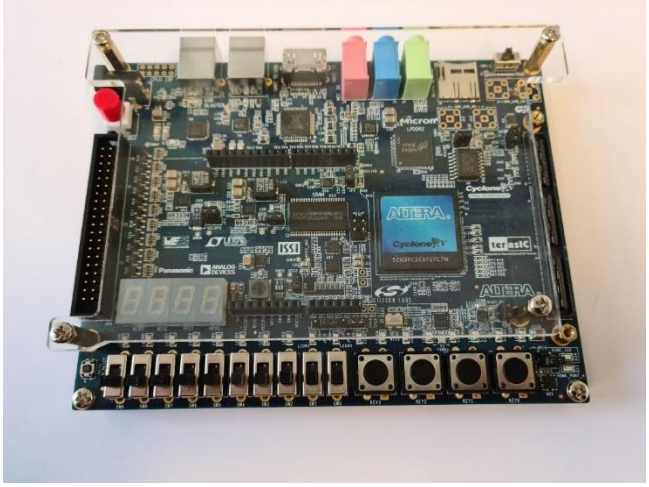
2.4 FPGA

En FPGA (Field Programmable Gate Array) er et kretskort bestående av mange eksterne komponenter som kan bli programmert til å utføre arbeidsoppgaver. Komponenten varierer fra FPGA til FPGA, men er vanlig oppbygd av brytere, dioder og tilkoblings pinner. For å bruke disse komponenten må logiske elementer programmeres ved hjelp av et program eller programmeringsspråk.[5]

FPGA-ens kjennetegn er at den er oppbygd av en matrise bestående av mange logiske elementer. Denne oppbygningen er kjent som «field programmable», eller felt-programmerbar, som betyr at den kan kjapt og effektivt reprogrammeres. For å bruke FPGA-en må funksjonene beskrives i en IDE (Integrated Development Environment) med et hardwarebeskrivende språk, som for eksempel VHDL eller Verilog.

2.4.1 Altera Cyclone V

FPGA-en brukt i dette prosjektet er gitt av Dolphitech og er en Altera Cyclone V GX C5, heretter kalt Altera Cyclone V. Altera Cyclone V er den femte i Altera sin Cyclone serie og kommer i fem forskjellige versjoner, C3, C4, C5, C7 og C9. Disse versjonen tilsvarer hvor kraftig FPGA-en er og hvor kjapt den kan utføre arbeidsoppgaver. C9 er den sterkeste med 301 000 logiske elementer (LE) og er den dyreste versjonen. [6]



Figur 2. 1 Bilde av Altera Cyclone V brukt i prosjektet. Bilde tatt av Vegar Sande (14.04.2021)

Altera Cyclone V C5 ble lansert i 2011 og har 77 000 LE, 336 I/O med muligheter for 3.0V – 3.3V in- og output, 1.2V – 3.3V in- og output, PCI, HSTL osv.[7] FPGA-ens interne klokkefrekvenser er 156.25MHz 1.5V PCML, 125MHz 2.5V og 50MHz 3.3V. Se Figur 2. 1 for visuell representasjon av FPGA-en. [8]

2.5 VHDL

Very high speed integrated circuit - Hardware Description Language (VHSIC - HDL)[9] også kjent som VHDL er et programmeringsspråk som benyttes for å beskrive hardware. VHDL blir brukt til å programmere FPGA-er og ASIC-er (Application Specific Integrated Circuit). Programmeringsspråket er designet for å beskrive hvordan fysiske komponenter skal fungere.[10]

Programmeringsspråket VHDL kan simuleres og lages syntese av ved bruk av programmer som ModelSim og Quartus Prime 20.1 Lite Edition (Quartus Prime). Simuleringen vil gi en visuell beskrivelse over hvordan kretsen vil reagere på forskjellige signaler den mottar eller sender. Syntesen vil lage en digital representasjon av kretsen ut ifra programvaren.

2.5.1 Virkemåte

En VHDL programvare består av tre hoveddeler – entitet, arkitektur og prosesser. Hver del ufører forskjellige roller som bygger på hverandre for å oppnå en funksjonell programvare. Delene fungere som byggeklosser, hvor delene til sammen bygger et tårn, med Entiteten i bunn og prosessen på topp. Hver brikke er like viktig og utfører hver sin arbeids oppgave.

1. I starten av alle VHDL programvarer blir en **entitet** startet for å kunne definere alle portene FPGA-en skal bruke. Portene blir definert som innganger, utganger eller som både inn- og utganger. Portene får en bestemt datatype som er lik datatypen som ønskes mottatt eller sendt på denne porten. De definerte portene kan deretter bli brukt i resten av programvaren til å lese eller sende data.

2. Alle signaler som kommer igjennom entiteten, blir prosessert og brukt i **arkitekturen**. Arkitekturen lager interne signaler for å bestemme hvordan data blir brukt internt og eksternt i programvaren. Ved å bruke interne signaler blir de eksterne signalene fordelt til forskjellige prosesserer, funksjoner og prosedyrer for å utføre bestemte arbeidsoppgaver.

3. **Prosessene** i arkitekturen blir satt opp for å aktivere hver gang et spesifikt signal blir mottatt. Når det definerte signalet er gitt, utføres arbeidsoppgaven programmert i prosessen. Ut ifra hva som blir gjort i hver enkel prosess, kan signalene påvirke andre prosesser i programvaren eller bli sendt ut på en port igjennom entiteten.

2.5.2 Quartus Prime Lite Edition

Intel Quartus Prime er Intel sitt eget FPGA programmerings applikasjon. Applikasjonen gir brukerne muligheten til å skape egne programmer, syntetisere programmet, optimalisere nye og eksisterende programmer ved å søke igjennom programmene etter feil. Quartus Lite edition støtter programmeringsspråk som VHDL, Verilog, SystemC og andre «hardware description language» (HDL). Quartus Lite edition er gratisversjonen Intel tilbyr og støtter FPGA-er som Cyclone® IV, Cyclone® V, Intel® Cyclone 10 LP og MAX® series. Denne versjonen har komplette biblioteker for generiske og spesifikke kommandoer som kan bli brukt til å konstruere forskjellige programvarer. [11]

2.5.3 ModelSim

ModelSim er et simuleringsverktøy, brukt til å simulere kretser programmert i VHDL, Verilog eller SystemC. Simuleringen gir en oversiktlig grafisk fremstilling over hvordan kretsen kommer til å fungere uten bruk av fysiske komponenter. ModelSim gir muligheten til å teste selve programvaren og overvåke alle signaler for å lokalisere feil. Grafene programmet skaper har mulighet til å vise hvordan programvaren fungerer i sanntid, som en digital representasjon av hvordan kretsen oppfører seg. Under simuleringer kan programvaren påtrykkes en rekke signaler i en eller flere sykluser, for å teste programvaren på nøyaktige tidspunkter. Denne prosessen gir en bedre oversikt over hvordan kretsen ville oppføre seg i forskjellige situasjoner.[12]

2.5.4 Testbenk

En testbenk i sammenheng med en HDL-kode refererer til en ekstern kode som har i hensikt å teste programvaren. Når et førsteutkast av koden er klargjort, kan det være lønnsomt og gjøre tester på kodens funksjonalitet. Dette gjøres for å sikre at alle egenskaper og funksjoner fungerer som de skal. Dette kan gjøres på flere måter, der en av dem er å skrive en testbenk tilpasset HDL koden for testing. Testbenken tar plassen til kretskortet i FPGA-en og simulerer signaler som kommer fra brytere, porter og andre eksterne signaler som påvirker programvaren. Ved å bruke simulerings verktøy kan små endringer gjøres og testes raskere og mer effektivt. Når testbenken er skrevet ferdig, finnes det flere muligheter for tredjeparts kodesimulerings programmer. To eksempler på disse er Riviera-PRO og ModelSim.

3 Metode

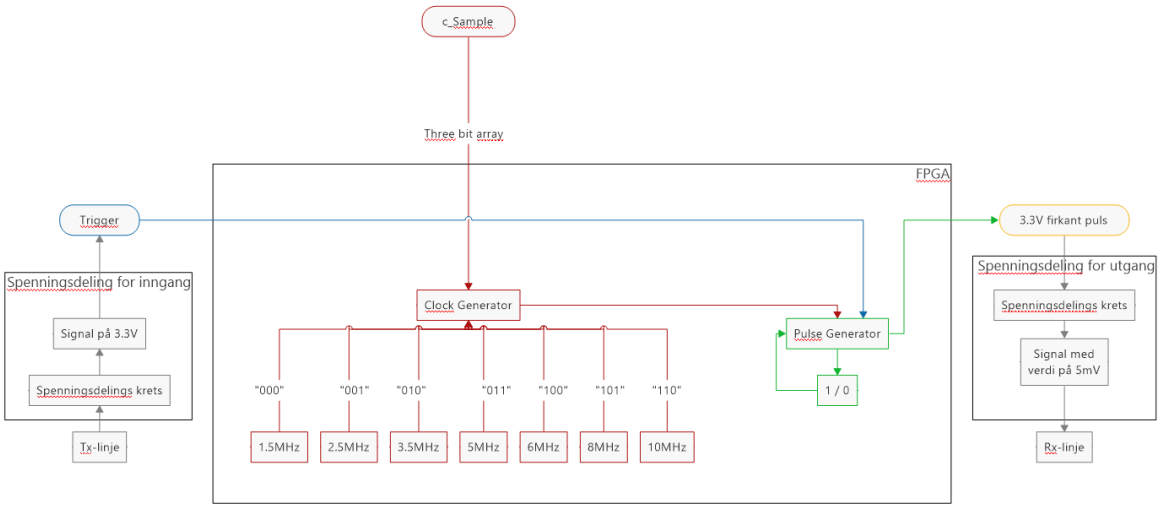
Dette kapittelet vil omhandle fremgangsmåten og metodene benyttet for å utforme programvaren, samt planleggingen av den fysiske realiseringen av kretsen.

3.1 Kretsoppbygning

Den planlagte kretsen ble ikke realisert fysisk, men vi vil her detaljere forslag til utforming av denne. Vi kommer til å gå igjennom bruken av spenningsdelere og tilkoblingsmuligheter for systemet.

3.1.1 Spenningsdeling

Her vil spenningsdelerne planlagt til FPGA-ens innganger og utganger detaljeres. Under er Figur 3. 1 som gir et bilde av hvordan kretsen kan fungere.



Figur 3. 1 Blokk skjema av komplett krets

Inngang

Dolphicam2 sender ut høyfrekvente signaler på 70V fra TRM elektronikken til Transduseren igjennom en fleksfilm fastmontert på transduseren. Dette er et for høyt spenningsnivå for FPGA-en, så den må reduseres. I dette tilfellet kan en enkel spenningsdeler konstruert av motstander benyttes. Ønsket spenningsnivå inn på FPGA-en er 3,3V.

Ved å bruke formel (1) fra kapittel 2.1:

$$V_{ut} = V_{inn} * \frac{R_2}{R_1 + R_2} \quad (1)$$

Kan motstandsverdiene for R_2 bestemmes ved å sette: $V_{inn} = 70V$, $V_{ut} = 3{,}3V$ og velge motstand R_1 til 10kΩ:

$$3{,}3V = 70V * \frac{R_2}{10k\Omega + R_2} \quad (1)$$

Ved å snu om på formelen og sette inn de valgte verdiene kan R_2 beregnes:

$$R_2 = R_1 * \frac{1}{\left(\frac{V_{inn}}{V_{ut}} - 1\right)} = 10k\Omega * \frac{1}{\left(\frac{70V}{3{,}3V} - 1\right)} = 494{,}75\Omega \quad (1)$$

I dette tilfellet kan det da benyttes en spenningsdeler med $R_1 = 10k\Omega$ og $R_2 \leq 494{,}75\Omega$ for å komme under 3,3V inn i FPGA. Denne spenningsdeleren kan kun behandle en av de 128 TX linjene, så det er nødvendig å lage et kretskort med 128 av denne spenningsdeleren for å dekke alle linjene. Her vil det være hensiktsmessig å benytte seg av SMD resistanser, da disse er vesentlig mindre enn tradisjonelle motstander.

Utgang

Det er også nødvendig å ha en spenningsdeler på hver utgang fra FPGA-en. Elektronikken får i vanlig operasjon en spenning på opptil 5mV tilbake fra transduseren. Utgangen på FPGA-en gir ut en spenning på 3,3V og må derfor reduseres betraktelig.

Samme formel benyttes som tidligere, men velger en større verdi for R_1 siden en mindre del av spenningen skal ligge over R_2.

$$R_2 = R_1 * \frac{1}{\left(\frac{V_{inn}}{V_{ut}} - 1\right)} = 100k\Omega * \frac{1}{\left(\frac{3{,}3V}{5mV} - 1\right)} = 151{,}745\Omega \quad (1)$$

For å oppnå en spenning lavere enn 5mV, kan det brukes R_1 = 100k**Ω** og $R_2 \leq$ 151**Ω**. Da forskjellen imellom de to motstandene er så store, er det mulig dette vil skape problemer. Det kan argumenteres for at det er verdt et forsøk før man forsøker å benytte andre, dyrere komponenter.

3.1.2 Tilkoblingskrets

Tilkoblingen av TRM elektronikken til FPGA-en er ikke realisert, så her detaljeres forslag til løsninger. Grunnet antallet separate linjer som skal verifiseres i det ferdige produktet, blir dette et relativt stort kretskort. I denne oppgaven er det spesifisert at det kun tar utgangspunkt i å sjekke en TX- og en RX-linje, så kretsen kan derfor testes med enkle komponenter.

Elektronikken kobles sammen med transduseren ved bruk av en 128 linjer bred fleksfilm. Denne sitter fastmontert på transduseren, men kan også benyttes frittstående, med koblinger i begge ender. Det anses derfor som naturlig å bruke dette i tilkoblingskretsen. Kretsen, sett fra TRM elektronikken, starter med en koblingsklemme for å montere fleksfilmen. Deretter må hver linje deles igjennom to motstander for å benytte spenningsdelingsprinsippet detaljert tidligere i dette kapittelet. Dette omformer spenningen på 70V fra TRM elektronikken, til en 3,3V spenning til FPGA. Spenningsdelingen er detaljert under 3.1.1 – Inngang.

Per nå benyttes et utviklerkort (FPGA), som har vesentlig flere muligheter og funksjoner enn det som er nødvendig. Dette kan skaleres ned, ved å designe en egen FPGA, hvor kun de nødvendige komponentene for tilkobling og signalbehandling inkluderes. Dette har ikke blitt utført som en del av oppgaven, og er kun et forslag til videre utvikling hos Dolphitech.

På utgangen av FPGA-en må det igjen foretas en spenningsdeling, da transduseren sender en vesentlig lavere spenning fra Rx-linjene og tilbake til TRM elektronikken enn FPGA-en gjør. Utgangsspenningen fra spenningsdeleren føres så tilbake til Rx i TRM elektronikken. Spenningsdelingen er detaljert under 3.1.1 - Utgang.

3.2 Quartus Prime

For å oppnå problemstillingen bruker gruppen Quartus Prime til å programmere en FPGA for å kunne bli brukt til å teste Dolphicam2 sin matrise. Prosjektet bruker FPGA-en Altera Cyclone V C5 som nevnt i Kapittel 2.4.1.

Versjonen av Quartus Prime som blir brukt i dette prosjektet er Quartus Prime 20.1 Lite Edition og ikke Quartus Prime 20.1.1 Lite Edition. Dette er fordi gruppen har tidligere erfaring med Quartus Prime 20.1 Lite Edition, og valgte å holde seg til kjente verktøy.

Programvaren blir skrevet på programmeringsspråket VHDL og vil bruke eksisterende biblioteker for at programvaren kan bli installert direkte i Altera Cyclone V. Målet med programvaren er å kunne motta et signal fra TRM og returnere en kjent sekvens av signaler tilbake. Signalet som kommer fra TRM vil bestemme når den kjente sekvensen blir returnert til TRM-en. Programvaren skrevet i VHDL blir utviklet med disse målene som fokus.

Programvaren er delt i 3 hoved prosesser. Den første prosessen skal skape de bestemte verdiene brukt til å styre frekvensen utgangssignalet har. Denne prosessen bruker FPGA-ens interne klokke som divisor for å oppnå en kvotient som tilsvarer transduserens frekvenser.

$$\frac{Intern\ klokke\ frekvens}{Transduser\ frekvens} = Div \qquad (2)$$

For å kunne bruke «Div» må tallverdien være et heltall, så ved divisjoner hvor «Div» verdien inneholder desimaler blir tallverdien redusert til nærmeste heltall. Ut ifra formelen blir alle verdiene manuelt skrevet inn i programvaren og valgt ut ifra brytere på FPGA-en. Eksempel av formel 2:

$$\frac{25MHz}{2.5MHz} = 10 \qquad (2)$$

Den andre prosessen skal lage en pulsgenerator som endrer signalet fra 1 til 0 og tilbake igjen. Dette skaper et alternerende signal som har en kjent rekkefølge av 1 og 0. Denne kjente sekvensen kan derfor brukes til å oppdage feil på elektronikkens matrise, da det skal aldri være flere 0 eller 1 etter hverandre. Pulsgeneratoren endrer verdi hver gang telleren, som økes

med én per klokke puls, oppnår samme verdi som «Div» ble satt til i den første prosessen. Når telleren og «Div» er like endres verdien fra 1 til 0 og omvendt.

Den siste prosessen venter på et aktiverings signal for å sende pulsgeneratorens signal på utgangen. Utgangen skal åpnes når FPGA-en mottar signal fra Tx-linjen eller når den mottar et signal som simulerer Tx.

3.3 Testing og feilsøking

Testing og feilsøking er sentrale prosesser i utviklingen av program- og maskinvare. En strukturert og oversiktlig feilsøkings metode er et ekstremt viktig redskap for å fullføre og optimalisere et prosjekt.

Programvaren blir konstruert for tre forskjellige tester. Først blir programvaren designet til å testes mot en testbenk med målet om å utelukke mulige feil i programvaren. Målet med testen er å bekrefte at programvaren vil fungere før den blir implementert i FPGA-en.

Test nummer to blir å teste programvaren med FPGA-en. Knappene og bryterne på FPGA-en blir benyttet som manuelle måter å aktivere programvaren og til endring av frekvensene.

Den siste testen er å teste den med en ekstern spenningskilde som aktiveringssignal for å simulere signal fra Tx etter spenningsdelingskretsen.

3.3.1 Testbenk

For testing og feilsøking av programvaren er det skrevet en testbenk. Testbenkens oppgave er å simulere et kretskort som programvaren kan kjøre på. Dette gjøres ved å påtrykke signaler som programvaren prosesserer, for så å sjekke responsen på utgangssignalene. I tillegg til disse må testbenken også generere kretskortets «reset» og klokke funksjon.

Programvarens testbenk starter først med å initialisere c_Sample til «000» og s_Trigger til «0». s_Trigger er knappen som må være aktiv slik at TRM mottar frekvensene og c_Sample velger frekvensen. Deretter setter testbenken s_Trigger aktiv for hver forskjellige c_Sample for å teste om alle fungerer som de skal. Bilder av denne simulasjonen kan finnes i kapittel 5.2.1, Figur 5. 11 til Figur 5. 13.

3.3.2 Simulering

For simulering og testing av program- og maskinvare i sann-tid utnyttes ModelSim. Hovedkoden kjøres i lag med en testbenk som påtrykker signaler på portene. Hvordan programvaren håndterer disse signalene vises på skjermen i formen av firkantpulser.

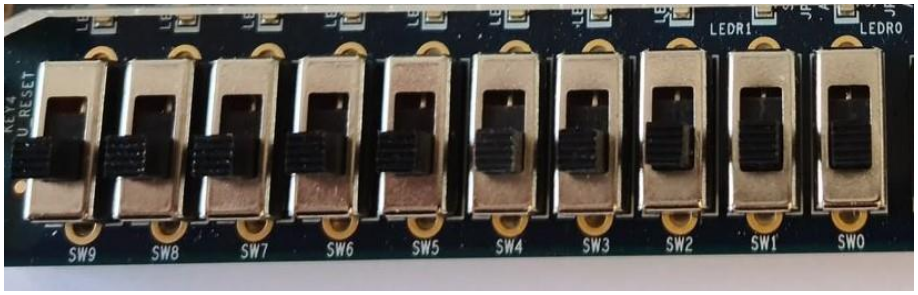
Under testing av maskinvaren blir brytere på FPGA-en brukt som en manuell måte å starte simuleringen. Figur 3. 2 til Figur 3. 5 viser hvilke deler av FPGA-en som blir brukt under simuleringen og hvilke signaler FPGA-en leser av de.

Figur 3. 2 viser knapper på FPGA-en. I dette prosjektet brukes KEY0 og KEY1 som måter å manuelt påtrykke signaler på kretsen under testing. KEY0 blir brukt til å nullstille alle verdier i programvaren. For å garantere at resultatene ikke er påvirket av andre tester blir denne knappen brukt etter hver test. KEY1 er knappen som blir brukt for å simulere start signalet fra elektronikken fra Dolphicam2. KEY2 og KEY3 har ingen funksjon og blir ikke brukt i dette prosjektet.

Figur 3. 2 Knapper på FPGA-en

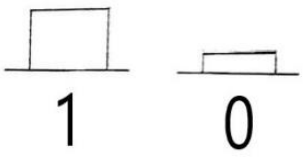
Figur 3. 3 viser brytere på FPGA-en som brukes til å simulere de forskjellige frekvensene transduseren har. FPGA-en har ti brytere. Bryter SW9 til SW3 blir ikke benyttet, så de har ingen funksjon. Bryter SW2, SW1 og SW0 blir brukt til å simulere de forskjellige frekvensene FPGA-en skal bruke. Bryterne har åtte forskjellige kombinasjoner som brukes til

de syv forskjellige transduser frekvensene. SW2, SW1 og SW0 representeres som «xxx», hvor x-ene er en kombinasjon av 1 eller 0. Figur 3. 3 er stilt til «000».

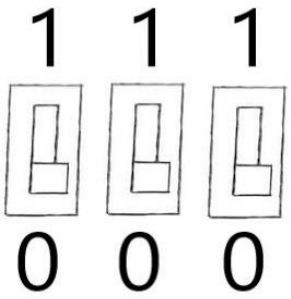


Figur 3. 3 Bryterne brukt til å endre frekvenser under testing

Figur 3. 4 viser hvilke signaler knappene på FPGA-en påtrykker programvaren. Knappene er høy når den ikke blir bruk og lav under bruk. Figur 3. 5 viser beskrivelse av hvilket signal programvaren får fra bryternes posisjoner.



Figur 3. 4 Figuren viser når Key1 og Key0 er aktiv og inaktiv



Figur 3. 5 Figuren viser når SW2, SW1 og SW0 er aktiv eller og inaktiv

3.3.3 ModelSim

Ved konstruksjon av en programvare er det viktig å feilsøke programvaren og lage en syntese for å oppdage feil eller svakheter. ModelSim er simuleringsverktøyet som blir brukt for dette prosjektet. ModelSim er et program gruppen er kjent med og har blitt brukt for testing av programmer skrevet i VHDL fra Quartus Prime. ModelSim brukes til å skape en grafisk representasjon av hvordan programvaren skal oppføre seg som om den var FPGA-en. ModelSim gir en god oversikt over funksjonaliteten til HDL-koden, og er derfor et viktig verktøy i feilsøkingen og verifiseringen av kretsdesignet.

4 Design og implementasjon

4.1 Programvarens funksjon

Problemstillingen baserer seg på å teste funksjonaliteten til Dolphicam2 sin elektronikk. Prosjektet bruker derfor FPGA-en til å simulere signaler som kan bli brukt til å teste elektronikkens matrise for å utelukke mulige feil. FPGA-en blir brukt til å etterligne signalene transduseren returnerer til elektronikken. Programvaren til FPGA-en er delt opp i de tre måtene forklart i kapittel 2.5.1 med forskjellige prosesser som ufører hver sin oppgave.

4.1.1 Programvaren

Programvaren er designet for å kunne teste Dolphicam2 sin elektronikk uten bruk av transdusere. Programvaren er konstruert slik at testen kan repeteres for alle transdusere Dolphicam2 er kompatibel med, og utgangssignalet vil være kjent for alle punktene på matrisen. Som nevnt i kapittel 1.4 blir ikke hele matrisen til Dolphicam2 simulert av FPGA-en. Programvaren er designet med 5 punkter på en linje med muligheten til å legge til flere linjer. TRM_2 er et ekstra signal som kan legges til på samme steder som TRM for å kunne simulere fler enn en linje. Dette signalet vil sende akkurat de samme verdiene som TRM for at alle resultatene elektronikken mottar kan sammenlignes.

Som nevnt tidligere er programvaren bygd opp av entitet, arkitektur og prosesser. Denne programvaren har arbeidsoppgavene delt opp i tre prosesser: p_speed, p_Sample og p_BB.

For å kunne simulere flere transdusere med forskjellige hastigheter er **p_Speed** designet med en mulighet til å stille inn for hver av Dolphicam2 sine transdusere. Programvaren bruker kombinasjonen av signalene fra SW2, SW1 og SW0 til å bestemme mellom de forskjellige hastighetene. For enhver kombinasjon av disse bryterposisjonene (se Figur 3. 3), blir en kombinasjon av 1 og 0 gitt. Denne kombinasjonen blir brukt til å velge en tallverdi som brukes til å dele opp FPGA-ens interne klokkefrekvens. FPGA-ens interne klokke er satt til 50MHz og kan derfor deles opp med de forskjellige verdiene til å oppnå frekvenser som tilsvarer transduserne. Se Tabell 4. 1 for alle kombinasjonene.

Tabell 4. 1 Tabell over verdiene p_Speed opererer med

p_Speed frekvens oppdeling		
c_Sample («xxx»)	div verdi	Frekvens verdier (MHz)
«000»	18	1.5
«001»	10	2.5
«010»	8	3.5
«011»	5	5
«100»	5	6
«101»	4	8
«110»	3	10
«111»	18	1.5

For å skape nye frekvenser bruker **p_Sample** verdien til div som divisor for å skape en kvotient som tilsvarer transduserens frekvenser. Frekvensene som p_Sample genererer er ikke nøyaktig lik de transduserne har, men Dolphicam2 kan motta signalene, så lenge signalene er lavere enn transduserens frekvens. FPGA-en øker en teller for hver høy puls fra den interne klokka til tallverdien er lik eller større enn verdien til div. Denne prosessen fortsetter uansett hvilken frekvens FPGA-en er stilt inn til å etterligne. Når teller verdien blir lik div nullstilles telleren og s_tick blir satt høy for én puls. s_tick blir brukt for å signalisere når puls generatoren skal endre fra høy til lav og lav til høy. For å unngå problemer ved endring av

frekvenser så er denne telleren designet til å nullstille verdien om teller verdien er høyere enn div. Dette kan oppstå ved endring av c_sample fra en div verdi til en lavere. Se vedlegg 1.

```
p_Sample: Process(clk, resetN) is
begin

if(resetN = '0') then

    countBaud<= 1;
    s_tick      <= '0';
    f_Send      <= '0';

elsif (clk'event and clk = '1') then

    if countBaud >= div then
        countBaud <= 1;
        s_tick <= '1';
    else
        countBaud <= countBaud + 1;
        s_tick <= '0';
    end if;

    if s_tick = '1'then
        f_Send <= not f_Send;
    end if;
end if;
end process p_Sample;
```

Figur 4. 1 Sample viser hvordan klokke og puls generatoren fungerer. Utklipp fra programvaren

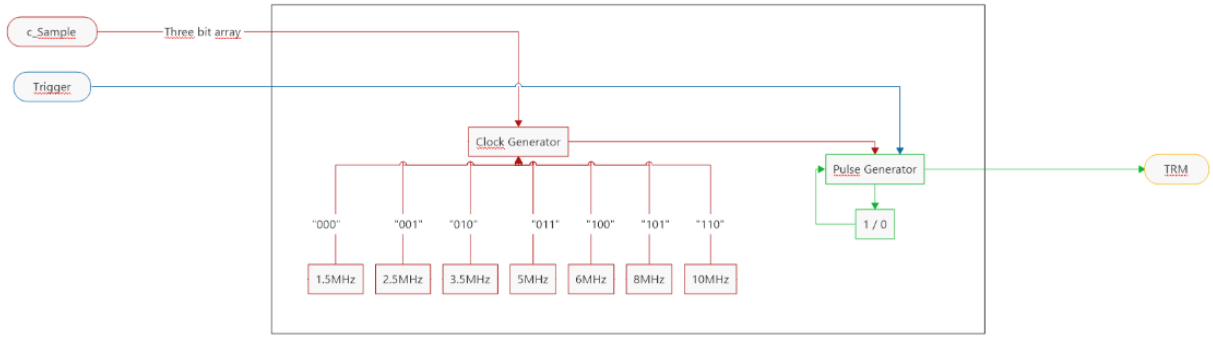
Figur 4. 1 er et utklipp av programvaren som utfører både klokkefrekvensen og endringene på pulsgeneratoren. f_Send er selve puls generatoren og endrer verdi hver gang s_tick blir høy. Dette signalet blir brukt i p_BB som signal på utgangen. Utgangen er lav til FPGA-en mottar start signalet, da blir f_Send påtrykt på utgangen og signalet kan leses av TRM-en.

Figur 4. 2 er et utklipp av programvaren. Utklippet viser prosessen p_BB og dens arbeidsoppgaver. Denne delen av programvaren har to forskjellige versjoner for de forskjellige måtene den ble testet på i Kapittel 5. Forskjellen er verdien s_Trigger skal sammenlignes med. Som nevnt i Kapittel 3.3.2 så gir knappen fra seg et høyt signal når den ikke er i bruk. Dette vil føre til at utgangen vil sende signaler helt til knappen blir trykket inn. For å gjøre simuleringen så nøyaktig som mulig, er verdien s_Trigger sammenlignes med satt til '0' på den første testen og '1' på den andre for å oppnå et nøyaktig resultat.

```
p_BB: Process(Clk, resetN) is
begin
if(resetN = '0') then
elsif (rising_edge(clk)) then
    if (s_Trigger = '1') then
        TRM <= f_Send;
        else
        TRM <= '0';
    end if;
end if;
end process p_BB;
```

Figur 4. 2 Utklipp av programvaren, prosess p_BB

Figur 4. 3 er representasjon av hvordan FPGA-en er programmert. Funksjonen er som forklart tidligere, men gir en bedre oversikt over FPGA-ens funksjon. De røde linjene er signalene som p_Speed bruker, de blå signalene er for p_BB og de grønne er signaler i og fra p_Sample.



Figur 4. 3 Visuell representasjon av FPGA-ens programvare

4.1.2 Testbenk implementasjon

Testbenk er en digital løsning for å teste en programvare uten å måtte bruke noen fysiske komponenter. Dette er en fordel for konstruksjon av programvarer som har mange forskjellige arbeidsoppgaver hvor flere feil kan oppstå. Testbenken fungerer som en ramme rundt Figur 4. 3 hvor c_Sample, Trigger og TRM blir styrt av testbenkens programvare. Denne prosessen fremhever hvor eventuelle problemer kan oppstå slik at de kan løses før programvaren blir implementert i FPGA-en.

Testbenkens oppgave for programvaren er å teste signalene s_Trigger og c_Sample fra programvaren. Testbenken i sin helhet er vedlagt og kan finnes i Vedlegg 2.

```
architecture tb_main of tb_dolphitech is

        constant c_ClkPer      : time := 40 ns;

        component dolphitech

                port (

                        s_Trigger       :       in      std_logic;
                        TRM             :       out     std_logic;
                        c_Sample        :       in      std_logic_vector(2 downto 0);
                        clk, resetN     :       in      std_logic

                        );

        end component dolphitech;


signal clk, resetN              :       std_logic;
signal s_Trigger, TRM           :       std_logic;
signal c_Sample                 :       std_logic_vector(2 downto 0);

begin


        test : component dolphitech

port map (

        clk             =>      clk,
        resetN          =>      resetN,
        s_Trigger       =>      s_Trigger,
        TRM             =>      TRM,
        c_Sample        =>      c_Sample

        );
```

Figur 4. 4: Oppsettet på testbenken for å koble den opp mot hovedprogramvaren

Testbenken har en tom entitet, fordi den skal utnytte entiteten til hovedprogramvaren. Testbenken lager heller en komponent med identiske signaler som den kobler opp mot hovedprogramvaren, som vist i Figur 4. 4. Denne koblingen skjer i dannelsen av komponenten med navn «test».

Deretter er testbenken skrevet med tre prosesser totalt: En prosess for generering av klokke signal, en prosess for reset signal og en hoved prosess for selve testingen av hovedprogramvaren.

```
p_clk : process is
begin

        clk <= '0';
        wait for c_ClkPer;
        clk <= '1';
        wait for 10 ns;

end process p_clk;
```

Figur 4. 5: Prosessen som genererer klokke signalet

Klokkeprosessen fra Figur 4. 5 skal simulere et klokkesignal fra en tilkoblet krets. Dette gjør den ved hjelp av en loop, der den setter seg selv lav og høy med 40 nanosekunders mellomrom.

```
p_resetN : process is

begin

        resetN <= '0';
        wait for c_ClkPer;
        resetN <= '1';
        wait;

end process p_resetN;
```

Figur 4. 6: Prosessen som setter reset signalet

Neste prosess, se Figur 4. 6, er reset prosessen. Denne sørger for at simulasjonen startes en resett slik at alt blir initialisert korrekt ved oppstart. Deretter setter den seg selv høy og venter til simulasjon slutt. I denne testbenken utnyttes den ikke mer enn én gang etter oppstart, men det kan være hensiktsmessig og teste reset funksjonen dypere i andre situasjoner.

```
p_main : process is

    procedure tb_init is
    begin

        c_Sample      <= (others => '0');
        s_Trigger     <= '0';

    end procedure tb_init;
```

Figur 4. 7: Prosedyren som initialiserer signalene i simulasjonen

I Figur 4. 7 er den første av tre prosedyrer i hovedprosessen. Hensikten med denne prosessen er bare å få en verdi på signalene i oppstarten slik at de ikke er uinitialiserte. Dette gjør den enkelt ved å sette dem til 0 ved oppstart.

```
procedure tb_test is
begin

    wait until rising_edge(clk);
    wait for c_ClkPer;

    c_Sample <= "000";
    wait for c_ClkPer;
    -- Sett frekvens til 1,5MHz

    c_Sample <= "001";
    wait for c_ClkPer;
    -- Sett frekvens til 2,5MHz

    c_Sample <= "010";
    wait for c_ClkPer;
    -- Sett frekvens til 3,5MHz

    c_Sample <= "011";
    wait for c_ClkPer;
    -- Sett frekvens til 5MHz
    c_Sample <= "100";
    wait for c_ClkPer;
    -- Sett frekvens til 6MHz

    c_Sample <= "101";
    wait for c_ClkPer;
    -- Sett frekvens til 8MHz

    c_Sample <= "110";
    wait for c_ClkPer;
    -- Sett frekvens til 10MHz

end procedure tb_test;
```

Figur 4. 8: Prosedyre som med tidsmellomrom stiller inn c_Sample til alle mulige tilstander

Den andre prosedyren i hovedprosessen (Figur 4. 8) stiller inn c_Sample for å teste alle mulige tilstander. Den stiller inn c_Sample til en verdi, venter en klokke periode og repeterer seg for alle mulige verdier av c_Sample.

```
procedure tb_trig is
begin

        s_Trigger <= '1';
        wait for 5000 ns;
        s_Trigger <= '0';
        wait for 5000 ns;

end procedure tb_trig;
```

Figur 4. 9: Prosedyre for aktivering av knapp

Den tredje og siste prosedyren (Figur 4. 9) aktiverer knappen ved å sette s_Trigger høy. Den venter med et 5000 nanosekunders mellomrom for å vise frekvens forskjellene tydeligere. Disse prosedyrene utnyttes ulikt i arkitekturen i Figur 4. 10:

```
begin

        tb_init;
        c_Sample <= "000";
        tb_trig;
        c_Sample <= "001";
        tb_trig;
        c_Sample <= "010";
        tb_trig;
        c_Sample <= "011";
        tb_trig;
        c_Sample <= "100";
        tb_trig;
        c_Sample <= "101";
        tb_trig;
        c_Sample <= "110";
        tb_trig;

wait for 3 ms;

assert false report "Pog" severity failure;

        end process p_main;

end architecture tb_main;
```

Figur 4. 10: Testbenkens arkitektur

Arkitekturen bruker først «tb_init» prosedyren for å initialisere signalene, så blir prosedyren «tb_trig» brukt for å aktivere knappen for alle de ulike frekvensene. Slik blir programvaren oversiktlig demonstrert og testet når det simuleres (se Figur 5. 11 til Figur 5. 13).

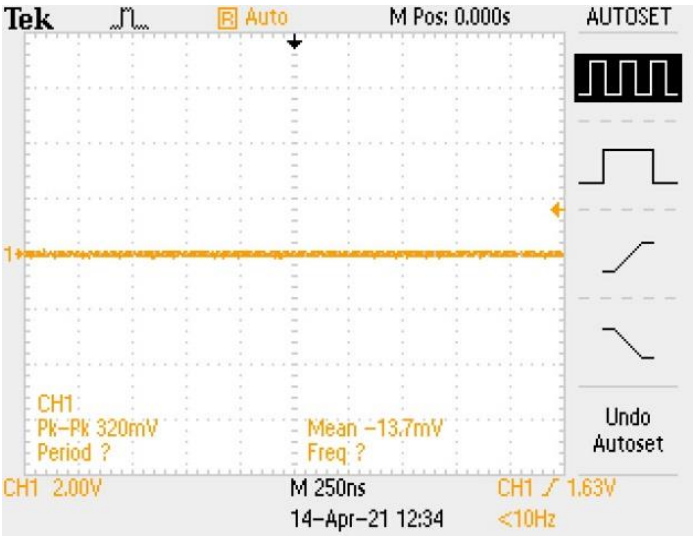
5 Programvaretesting og simulering

I dette kapittelet skal resultatene fra simuleringen og testen presenteres. Resultatene er delt opp i de forskjellige måtene de ble testet på for å gi en bedre oversikt over hvert enkelt resultat.

5.1 FPGA resultater

Som beskrevet i kapittel 4.1 skal FPGA-en kun utføre arbeids oppgaven når den mottar start-signalet og kunne tilpasses til å fungere som de forskjellige transduserne Dolphicam2 kan tilkobles til. Dolphicam2 kan tilkobles syv forskjellige transdusere, hvor hver av de har forskjellige frekvenser. De forskjellige hastighetene er vist i Figur 5.2 til Figur 5.8. For å kunne velge mellom de forskjellige frekvensene brukes tre brytere på FPGA-en til å velge mellom de forskjellige transduserfrekvensene. Se Figur 5.1.

For å oppnå et godt standpunkt blir FPGA-en testet uten eksterne signaler og bruker en bryter på FPGA-en til å simulere aktiverings signalet fra Dolphicam2. Figur 5.1 er standpunktet for testen hvor FPGA-en er på, men mottar ingen signaler. Selv om bryterne Figur 3. 3 blir endret vil utgang signalet forbli lavt til et aktiverings signal blir gitt.

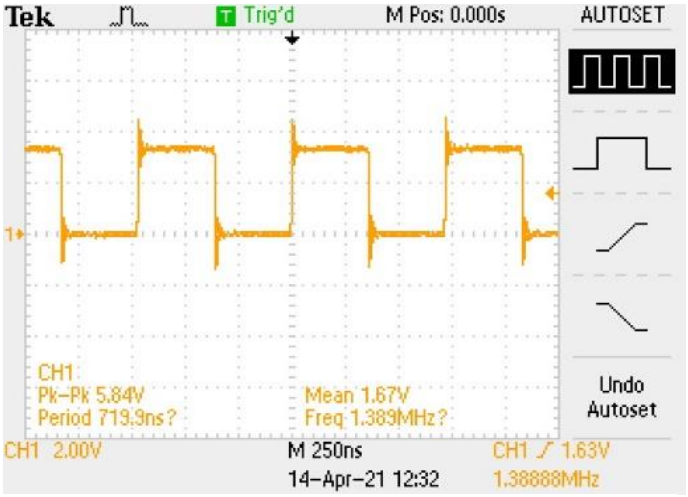


Figur 5.1 Utgang signal uten Start signal.
Key1 = 1

Den første testen på FPGA-en ble gjort med en bryter som aktiveringssignal. Key1 ble brukt for aktivering. Dette er for å teste programvarens funksjonalitet før flere elementer blir introdusert til krets. Ved å teste programvaren uten ekstra elementer blir resultatene sikrere og eventuelle feil som oppstår kan kun komme fra programvaren.

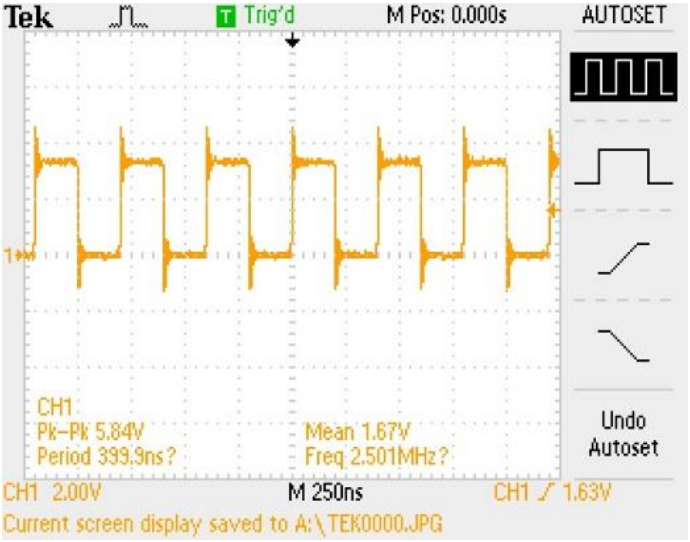
I den første testen er programvaren skrevet slik at start signalet må være lavt for at utgangen begynner å sende signaler. Key1 er lav når den blir trykket in, som vist i Figur 3. 2 så Key1 blir brukt til å simulere signalet fra Dolphicam2. SW2, SW1 og SW0 er alle lave for å sette frekvensen til den laveste hastighet, se Figur 3. 3. Utgangen til FPGA-en blir koblet opp mot et oscilloskop for å kunne se endringen i frekvensen.

Når Key1 blir presset ned blir signalet inn på FPGA-en lav og utgangen begynner å sende signalet vist i Figur 5.2. Kombinasjonen «000» av SW2, SW1 og SW0 produserer FPGA-en en frekvens på 1.389MHz, se Figur 5.2 Målet med denne kombinasjonen var 1.5MHz som sett i Tabell 4. 1.



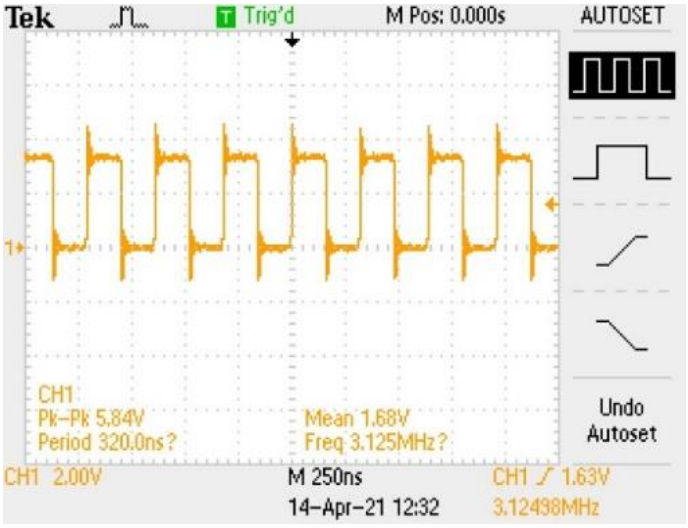
Figur 5.2 Utgang signal for 1.5MHz med start signal.
SW2, SW1 og SW0 = «000»

For kombinasjonen «001» av SW2, SW1 og SW0 produserer FPGA-en en frekvens på 2.5MHz, se Figur 5. 3 Målet med denne kombinasjonen var 2.5MHz som sett i Tabell 4. 1.



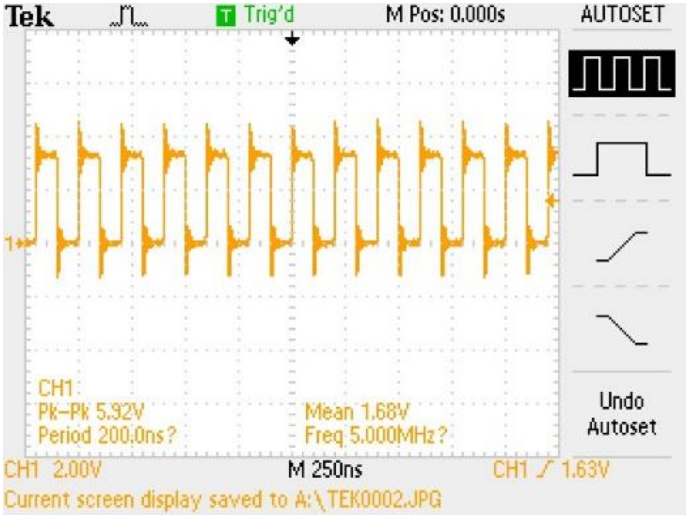
Figur 5. 3 Utgang signal for 2.5MHz med start-signal.
SW2, SW1 og SW0 = «001»

For kombinasjonen «010» av SW2, SW1 og SW0 produserer FPGA-en en frekvens på 3.125MHz, se Figur 5.4 Målet med denne kombinasjonen var 3.5MHz som sett i Tabell 4. 1.



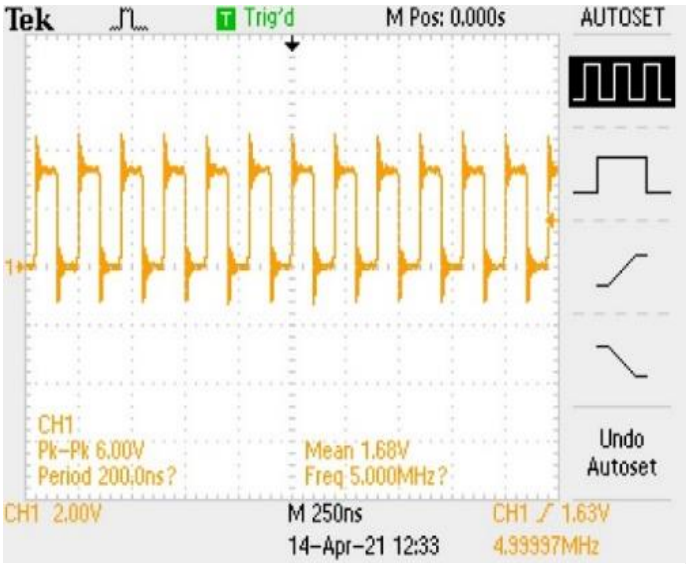
Figur 5.4 Utgang signal for 3.5MHz med start-signal.
SW2, SW1 og SW0 = «010»

For kombinasjonen «011» av SW2, SW1 og SW0 produserer FPGA-en en frekvens på 5MHz, se Figur 5.5 Målet med denne kombinasjonen var 5MHz som sett i Tabell 4. 1.



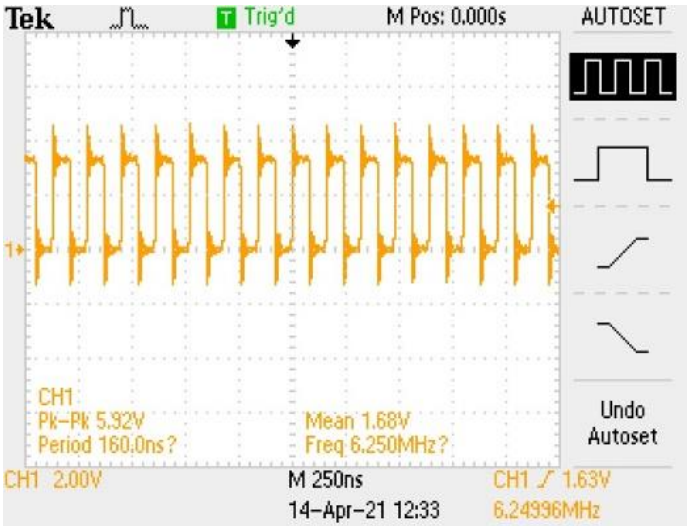
Figur 5.5 Utgang signal for 5MHz med start-signal.
SW2, SW1 og SW0 = «011»

For kombinasjonen «100» av SW2, SW1 og SW0 produserer FPGA-en en frekvens på 5MHz, se Figur 5.6 Målet med denne kombinasjonen var 6MHz som sett i Tabell 4. 1.



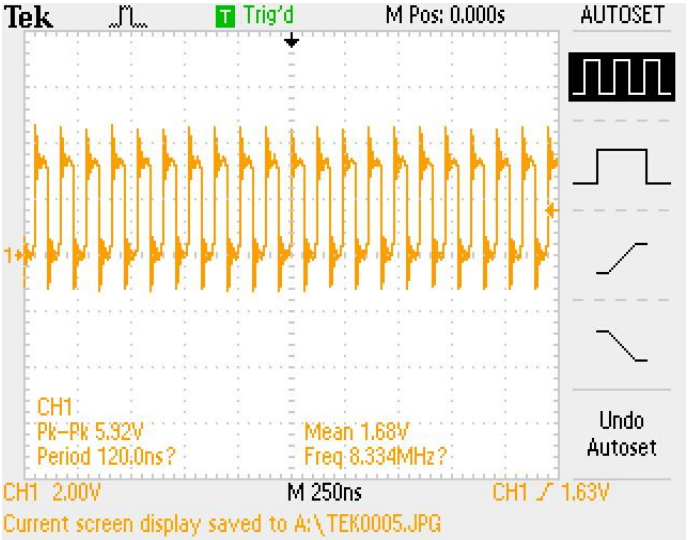
Figur 5.6 Utgang signal for 6MHz med start-signal.
SW2, SW1 og SW0 = «100»

For kombinasjonen «101» av SW2, SW1 og SW0 produserer FPGA-en en frekvens på 6.25MHz, se Figur 5.7 Målet med denne kombinasjonen var 8MHz som sett i Tabell 4. 1.



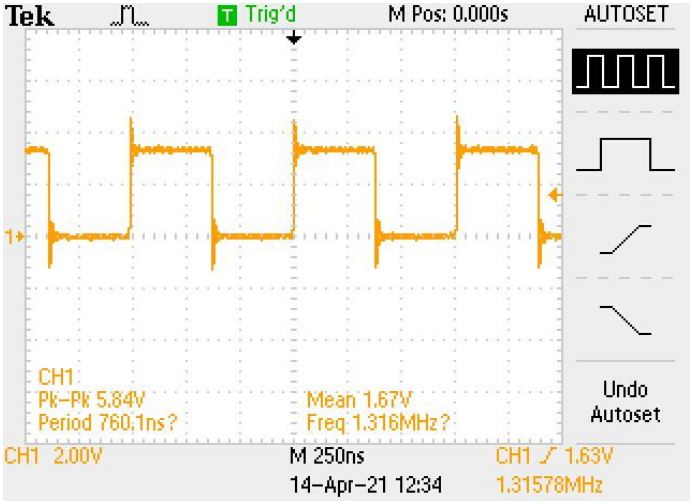
Figur 5.7 Utgang signal for 8MHz med start-signal.
SW2, SW1 og SW0 = «101»

For kombinasjonen «110» av SW2, SW1 og SW0 produserer FPGA-en en frekvens på 8.334MHz, se Figur 5.8 Målet med denne kombinasjonen var 10MHz som sett i Tabell 4. 1.



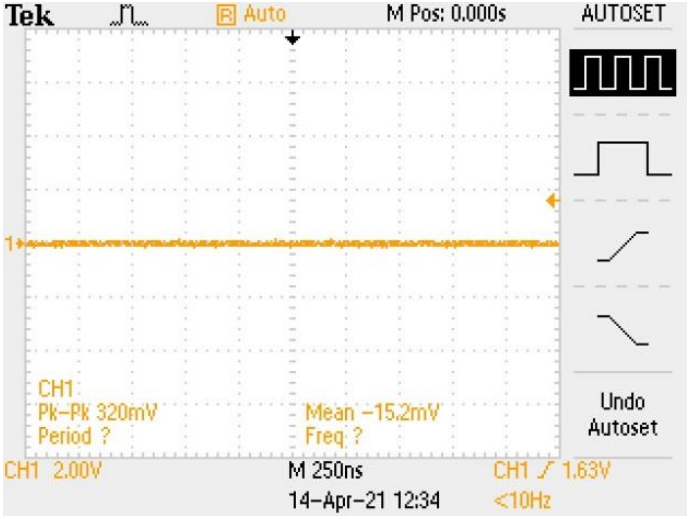
Figur 5.8 Utgang signal for 10MHz med start-signal.
SW2, SW1 og SW0 = «110»

For kombinasjonen «111» av SW2, SW1 og SW0 produserer FPGA-en en frekvens på 1.316MHz, se Figur 5.9. Målet med denne kombinasjonen var 1.5MHz som sett i Tabell 4. 1.



Figur 5.9 Utgang signal når en bryter kombinasjon som ikke er angitt blir valgt.
SW2, SW1 og SW0 = «111»

For hver kombinasjon av SW2, SW1 og SW0 skal frekvensen oppnå den nærmeste mulige frekvensen uten å over stige den maksimale frekvensen transduseren prøver å simulere. I Figur 5.10 vises hvordan FPGA-en reagerer når reset knappen blir brukt. Så lenge reset knappen er i bruk vil alle teller verdier og signaler stilles tilbake til sin startverdi.



Figur 5.10 Utgang signal når reset og start knappen er ned trykket.
For å teste nullstillings funksjonen til programvaren
Key0 = 0 og Key1 = 0

Etter at den første testen bekrefter at programvaren gjør det den er designet for, kan kretsen testes mot et simulert signal fra en spenningskilde. Spenningskilden blir koblet opp mot FPGA-en med en liten motstand mot jord. Denne motstanden er for å fjerne en svak «Up pull» pinnen på FPGA-en har. Spenningskilden er stilt til å gi 3.3V for å simulere signalet som FPGA-en skulle mottatt fra Dolphicam2 etter omformingen.

Med denne testen blir bryteren fra den første testen byttet ut med spenningskilde. Spenningskilden blir skrudd av og på for å simulere start signalet fra elektronikken og FPGA-en oppfører seg likt som første test. Målingene på utgangen er lik resultatene fra den første testen. Se Figur 5.2 til Figur 5.10. Vi kan da si at triggersignalet fungerer som det skal.

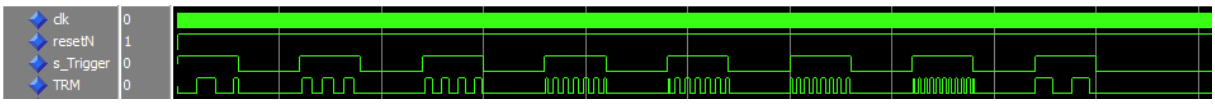
5.2 Simulasjon

Under utviklingen av en programvare er det kritisk å teste den regelmessig underveis for å kontrollere at den fungerer riktig. Denne testen kan utføres ved å simulere programvaren på tredjepartsapplikasjoner. En simulasjon vil gi en veldig god oversikt over hvilke deler av koden som fungerer og ikke fungerer. I tillegg vil en simulasjon gi et mye bedre bilde av akkurat hva det er som er galt med koden, istedenfor en generisk feilmelding i IDE-en. Ved å simulere programvaren kan det også lett demonstreres hvordan programvaren virker.

5.2.1 Simulasjon – Resultater

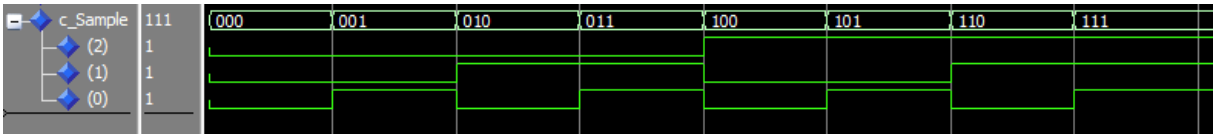
Nedenfor er det tre figurer som viser simulasjons resultatene fra testbenken. På dem er det totalt fem ulike signaler: clk, resetN, s_Trigger, TRM og c_Sample. Av disse er clk og resetN generert av testbenken, mens s_Trigger, TRM og c_Sample hentes fra programvaren. Hvordan testbenken er skrevet finnes i vedlegg 2.

Frekvensen kan endres, som demonstrert i Figur 5. 11. Hvilken frekvens det er, styres av signalet c_Sample, som er avbildet på Figur 5. 12. Tabell 4.1 viser relevante frekvenser.



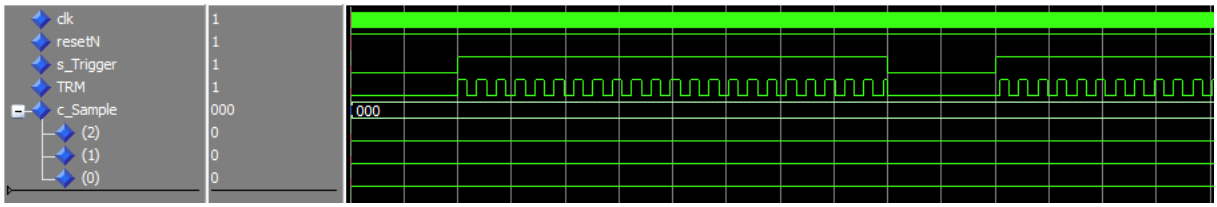
Figur 5. 11: Simuleringsresultat fra testbenken viser hvordan frekvensen kan endres

Figur 5. 12 viser hvordan c_Sample endres bitvis for å stille inn hvilken frekvens TRM-en skal motta og varierer binært fra «000» til «110». Særtilfellet er hvis den blir stilt til den siste ubrukte tilstanden, «111», siden det bare er bruk for 8 forskjellige frekvenser. I et slikt tilfellet er det blitt avgjort at frekvensen nullstilles til 1,5MHz.



Figur 5. 12: Hvordan c_Sample endres demonstrert

Som vist i Figur 5. 13 så mottar TRM-en signalet kun når s_Trigger er aktiv. Her er c_Sample satt til «000», altså er frekvensen i Figur 5. 13 1,5MHz. s_Trigger er aktiv høy.



Figur 5. 13: Simulasjon av programvaren viser knappens funksjonalitet

6 Diskusjon

6.1 Teorien brukt i prosjektet

Informasjonen i teori kapittelet er basert på skolelitteratur, artikler og produsentens egne datablader og brukermanualer. Alle kildene brukt i denne oppgaven brukes til å best representere utstyret, applikasjoner og metoder brukt i prosjektet. Kildene er utvalgt for å gi en bred dekning og solide referanser for stoffet oppgaven dekker.

Kapittel 2 handler om introduksjon av Dolpicam2, FPGA og applikasjonene brukt for å løse problemstillingen satt for prosjektet. Informasjonen presentert i dette kapittelet er grunnlaget for simuleringene gjort i senere kapitler.

Kapittel 3 gir et grunnlag for bruk av de forskjellige programvarene og planen for hvordan prosjektets simuleringer blir utført. Dette er et viktig kapittel for forståelse av hvordan arbeidet skal utførers for å oppnå resultatet problemstillingen er ute etter. Dette kapittelet definerer også hvordan prosjektet er planlagt å løses ved bruk av FPGA og VHDL programmering.

Kapittel 4 handler om designet til kretsen og programvaren. Designets oppbygning viser hvordan kretsen kan kobles til Dolphicam2 og beskriver kretsens funksjon. Kretsen er designet for å transformere spenninger fra Dolphicam2 ned til spenningsnivåer FPGA-en tåler. Dette signalet blir så håndtert av FPGA-en og sendt ut av FPGA-en for deretter å bli transformert ned igjen til et spenningsnivå Dolphicam2 elektronikken tåler. Disse prinsippene er så testet og simulert i kapittel 5.

Kapittel 5 skal undersøke om det er mulig å simulere signaler som Dolphicam2 elektronikken mottar fra transdusere ved bruk av FPGA. Etter testing av FPGA-en og simuleringen av programvaren er det avdekket at det er mulig å bruke en FPGA til å simulere transduserne Dolphicam2 tilkobles. Ved å bruke FPGA-en til å simulere en kjent sekvens av signaler på Dolphicam2 sin matrise kan punktene på matrisen testes uten bruk av transdusere. Dette gjør at Dolphicam2 kan testes mot alle transdusere før tilkobling av hver enkel transduser, som utelukker muligheter for feil på elektronikken.

6.2 Refleksjon

I underkapittelet Refleksjon blir utførsel av oppgaven vurdert med forklaring av hvorfor forskjellige valg ble tatt. Kapittelet tar også for forslag arbeidsgiver har kommet med underveis.

6.2.1 Utvikling av programvaren

Problemstillingen prosjektet baserer seg på er å finne en måte å teste elektronikken til Dolphicam2 uten bruk av transdusere. For å oppnå dette konstruerte gruppen programvaren i Vedlegg 1. Denne programvaren er designet for å sende en kjent rekke med signaler til elektronikken i en hastighet som tilsvarer en av transduser hastighetene som kan tilkobles elektronikken. Programvaren var originalt designet til å sende en bestemt sekvens av høye og lave signaler, men ble endret til designet i Vedlegg 1.

Fordelen med den tidligere versjonen av programvaren var at elektronikken mottok en bestemt rekke som alltid starter likt. Dette gjorde at signalet kunne repeteres uten endringer på rekken elektronikken fikk, men dette hadde sine limitasjoner. Limitasjonen er at programvaren måtte kalibreres for antall punkter i matrisen som skal bli testet. Dette gjorde at programvaren ikke var ideell for prototype testing og arbeidsgiver kom med et forslag om å endre programvaren. Programvaren ble endret til å bruke en simplere signal sekvens som ikke hadde noen limitasjoner for antall punkter i matrisen som kunne testes.

Den endelige programvaren bruker en pulsgenerator som signal og sender derfor enten et høyt eller lavt signal på utgangen. Dette gjør at signalet vil være kjent og kan derfor brukes til å sjekke hvert punkt siden hvert punkt i en linje skal variere fra 1 til 0. Dette betyr at signalet kan variere siden den kan starte enten høyt eller lavt, men resten av sekvensen er kjent uansett start signalet.

6.2.2 Utvikling av testbenken

Idéen bak testbenken var å oppnå en oversiktlig simulasjon av programvaren ikke bare for testing og feilsøking, men også som en ekstra metode å demonstrere programvarens funksjonalitet i tillegg til den faktiske testen på FPGA-en. Det ble da valgt å utvikle denne

testbenken i Quartus Prime og simuleres via tredjepartsapplikasjonen ModelSim, den første en IDE for VHDL og verilog kodespråk, og den siste et avansert simuleringsprogram. Det var naturlig å velge disse to programmene ovenfor andre muligheter da gruppen allerede har tidligere erfaring med bruk av begge fra andre fag.

Det er tre signaler som testbenken må teste: TRM, c_Sample og s_Trigger. TRM skal motta frekvensen, c_Sample bestemmer hvor stor frekvensen er og s_Trigger er knappen som slipper frekvensen gjennom til TRM-en. Hvordan testbenken tester disse signalene er å sette s_Trigger høy for 3000 nanosekunder for alle mulige tilstander av c_Sample. Dermed er det lett å se programvarens funksjonalitet, og i tillegg se at alt fungerer som det skal.

Testbenkens struktur er bygget opp av tre prosesser og tre prosedyrer. Den første prosessen genererer klokkesignalet, den andre prosessen danner reset-signalet og den siste er hovedprosessen der signalene blir påtrykt verdier.
Den første prosedyren initialiserer simuleringen ved å gi signalene en start verdi. Den andre prosedyren stiller inn c_Sample til de forskjellige frekvensverdiene og den siste prosedyren setter knappen s_Trigger høy i 3000 nanosekunder, så lav i 3000 nanosekunder.

Prosedyren tb_test endte opp med og ikke bli tatt i bruk, da en annen, lettere metode ble oppdaget. Prosedyren er likevel beholdt i testbenken siden den presenterer en annen, litt mer optimal metode for hvordan programvaren kan testes, selv om litt modifisering er nødvendig.

6.3 Vurdering av Metoder og løsninger

Prosjektets oppgave oppsto fordi Dolphitech ønsket å skape en komponent som kan utføre en sjekk av matrisen før bruk. Denne komponenten vil da ha muligheten til å lokalisere feilen før tilkobling av transduser om feilen ligger på matrisen. Ved å teste elektronikken først kan det forhindres skade på transduserne ved tilkobling da denne prosessen kan skade transduseren om den blir utført feil.

Ved bruk av FPGA-en og kretsen beskrevet i metode kapittelet, er det mulig å teste funksjonaliteten til matrisen uten transduseren. Kretsen er kun et konsept og er antatt å fungere i praksis, men komplikasjoner på grunn av Covid-19 gjorde at denne delen aldri ble

fullstendig realisert. Programvaren er testet teoretisk og praktisk med bruk av en testbenk i simulerings programmet ModelSim og testet ved bruk av FPGA og simulerte signaler.

Ut ifra resultatene oppnådd i kapittel 5 utfører FPGA-en arbeidsoppgavene gruppa ønsker at den skal utføre. Programvaren aktiveres ved mottak av et signal som enten simulerer eller tilsvarer signalet fra TX linja. Svakheten med programvaren er at den er ikke designet med alle punktene i matrisen aktiv på samme tid. Dette er tenkt på med at programvaren kan skaleres opp til ønsket antall punkter. Den eneste limiterende faktoren når det kommer til programvaren er FPGA-en den blir installert på. For at FPGA-en skal kunne teste alle punktene trenger det nok eksterne punkter. FPGA-en har 36 punkter som kan tilkobles direkte eller små USB, HDMI eller HSMC brukt til å koble til ekstra punkter. En måte å jobbe rundt dette er å sjekke elektronikken del for del for å sjekke alle punktene i matrisen. Se side 51 i kilde [8].

6.4 Fremtidig arbeid

Se Figur 3. 1 for et skjema over kretsen. Denne figuren viser hvilke deler som er utarbeidet, og hvilke deler som trenger videre utforskning. Disse delene inkluderer blant annet en fysisk realisering av kretsen og spenningsdelere.

6.4.1 Kretsutvikling

Det er nødvendig å utarbeide en fungerende tilkoblingskrets mellom TRM elektronikken og FPGA-en. Dette er hovedsakelig detaljert under kapittel «3.1 Kretsoppbygging». Grunnlaget for sammenkoblingskretsen er et eksisterende kretskort hos Dolphitech. Dette kortet har tilkobling for fleksfilm i ene enden, og leder deretter ut enhver linje til et loddepunkt. Det trengs derfor fire av disse kortene for å dekke alle linjene fra TRM elektronikken, da det her fremkommer fire fleksfilmer. To er Tx-linjer, og to er Rx-linjer.

En mulig fremgangsmåte her er å designe et nytt kort, med utgangspunkt i det eksisterende kortet. Målet er at de fire separate delene blir et enkelt kort. Det er også ansett som nødvendig å utvide disse kortene med spenningsdelere både på innganger og utganger, slik at alle signaler er innenfor maksgrensene til komponentene. Dette er henholdsvis 3,3V for signalene som går inn i FPGA-en, og opptil 5mV for signalene som går tilbake til TRM elektronikken.

Det er mulig det blir problematisk å benytte spenningsdeling på utgangene, siden det er en stor forskjell i spenningene. Det påtrykkes 3,3V ut fra utgangen, og dette må senkes til 5mV. Dette er en såpass stor endring, at motstandene kan bli for ulike, og at resultatet ikke blir stabilt over alle de forskjellige linjene. Om det ikke fungerer med spenningsdeling, kan det benyttes svært nøyaktige omformere, for å få en konstant spenning på 5mV, hvor også større deler av strømmen bevares.

6.4.2 Programvare

Ut ifra arbeidet som er blitt gjort i dette prosjektet er Vedlegg 1 og Vedlegg 2 grunnlaget for videre utvikling. Programvaren er designet for å repetere et kjent signal, og sende det til utgangen når FPGA-en mottar aktiveringssignalet. Programvaren er nå stilt til å konstant variere mellom 1 og 0. Dette kan forårsake et problem om Dolphicam2 sin elektronikk forventer et start-signal, som vanlig vis er representert som et 1.

Om elektronikken ikke trenger et start-signal må programvaren utvides til å kunne kobles på alle Tx- og Rx-linjene for å teste aktiveringen og mottaket av signaler. Programvaren er designet for en Tx-linje og en Rx-linje, men designet har en Rx-linje til klar for implementasjon i programvaren. For å aktivere flere Rx linjer må TRM2 tilkobles en utgang på FPGA-en og bli påtrykt samme signal som TRM i prosess p_BB. Figur 6. 1 viser et eksempel på hvordan videre utviklingen kan gjøres.

```
p_BB: Process(Clk, resetN) is
begin
if(resetN = '0') then
elsif (rising_edge(clk)) then
    if (s_Trigger = '1') then
        TRM <= f_Send;
        TRM2 <= f_Send;
        TRM3 <= f_Send;
        …
        else
        TRM <= '0';
        TRM2 <= '0';
        TRM3 <= '0';
        …
    end if;
end if;
end process p_BB;
```

Figur 6. 1 Eksempel på programvare videre utvikling

7 Konklusjon

Dette prosjektet har kommet langt på vei mot en fungerende prototyp av produktet. Hoveddelen av oppgaven omhandler VHDL-programvaren som simulerer responsen fra Dolphitechs transdusere. Tilkoblingskretsen er ikke realisert fysisk. Forslag til oppbygningen av kretsene er detaljert kapittel 3.1. Forslaget går i grove trekk ut på å benytte et eksisterende kretskort hos Dolphitech, som har en tilkoblingsklemme for transduserens fleksfilm. Dette gjør det mulig å føre signalene ut ifra, og tilbake til TRM-elektronikken.

FPGA-en mottar signaler fra TRM-elektronikken. Programvaren behandler disse signalene, og modulerer de til å likne transduserfrekvensene. De forskjellige frekvensene kan endres av brukeren, og velges ved å bruke bryterne på FPGA-en. FPGA-en sender de simulerte transdusersignalene tilbake til RX-linjene.

Simuleringen av programvaren viser lovende resultater for et konseptbevis som Dolphitech kan jobbe videre med. Programvaren fungerer godt under testing, og er forventet å fungere under videre testing med reelle komponenter.

Tilkoblingskretsen ble som nevnt ikke realisert fysisk, dette skjedde i all hovedsak på grunn av tidsmangel og en særdeles spesiell arbeidssituasjon med tanke på den pågående Covid-19 pandemien. Under bedre omstendigheter hadde gruppa kunnet teste programvaren og tilkoblingskretsen på reelle komponenter fra Dolphitech. Da dette ikke er mulig legges arbeidet frem som det står.

Gruppa kan derfor konkludere med at problemstillingen er oppnådd delvis teoretisk i form av tilkoblingskretsen, og delvis fysisk med tanke på programvaren. Med dette vil vi kunne presentere et godt utgangspunkt for videre utvikling hos Dolphitech, og forhåpentligvis en god løsning på problemet.

Kilder

[1] E. H. Vihovde. "Kompilere." Store Norske Leksikon https://snl.no/kompilere_-_IT (accessed 22.04, 2021).

[2] ndt.org. "What is NDT? Introduction to NDT." https://www.ndt.org/link.asp?ObjectID=59686 (accessed 11.03.2021, 2021).

[3] K. Hofstad. "Resistans." Store Norske Leksikon. https://snl.no/resistans (accessed 28.04, 2021).

[4] Ø. Grøn. "transduser." https://snl.no/transduser (accessed 20.04, 2021).

[5] B. B. Larsen. "SNL FPGA." snl.no/FPGA (accessed 26.02.2021, 2021).

[6] Intel. "Cyclone® V GX FPGA." https://www.intel.com/content/www/us/en/products/details/fpga/cyclone/v/gx.html (accessed 15.04, 2021).

[7] Intel. "Cyclone® V 5CGXC5 FPGA Specifications." Intel Corporation. https://www.intel.com/content/www/us/en/products/sku/210453/cyclone-v-5cgxc5-fpga/specifications.html (accessed 15.04, 2021).

[8] T. Technologies, *Cyclone V GX Starter Kit*, V1.02 ed.: Terasic Technologies, 07.04.2014, p. 103. [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=830&FID=17219f04ba333c8a2ee2066deab991e5.

[9] B. B. Larsen. "VHDL." https://snl.no/VHDL (accessed 03.22, 2021).

[10] S. Yalamanchili, *VHDL A Starter's Guide*, Second edition ed. Pearson Education, Inc., 2005.

[11] Intel. "Intel® Quartus® Prime Software Suite." https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html (accessed 02.10, 2021).

[12] S. Technology. "ModelSim." https://www.saros.co.uk/products/eda/tools/modelsim (accessed 03.05, 2021).

Vedlegg

Vedlegg 1

```
library ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;
Entity Dolphitech is
Port
    (
    -- Signals to and for Blackbox:

    -- Signal for line 1 on Tx
    TRM        : out std_logic;
    -- Signal for line 2 on Tx (Extra)
    -- TRM_2 : out std_logic;
    -- Signal form BB
    s_Trigger  : in std_logic;
    -- To choose which frequency the TRM is set too or what it's base frequency is;
    c_Sample  : in std_logic_vector(2 downto 0);

    -- Standard clock and reset for the board. Reset triggers low
    clk, resetN: in std_logic
    );
End Dolphitech;

architecture brain of Dolphitech is
--Signal for clock generator
    signal div  : integer range 0 to 1000000000;
    signal baudRate    : std_logic;
    signal countBaud : integer range 1 to 1000000000 :=1;
--Signal for different clk frequencies
    signal s_tick        : std_logic;
--Signal for the pulse generator
    signal f_Send       : std_logic := '0';
begin
```

```
p_speed: Process(c_sample) is
    begin

case c_Sample is
    when "000" => div <= 18;
    when "001" => div <= 10;
    when "010" => div <= 8;
    when "011" => div <= 5;
    when "100" => div <= 5;
    when "101" => div <= 4;
    when "110" => div <= 3;
    when others => div <= 18;
end case;
end process p_speed;

p_Sample: Process(clk, resetN) is
begin

if(resetN = '0') then

    countBaud<= 1;
    s_tick      <= '0';
    f_Send      <= '0';

elsif (clk'event and clk = '1') then

    if countBaud >= div then
        countBaud <= 1;
        s_tick <= '1';
    else
        countBaud <= countBaud + 1;
        s_tick <= '0';
    end if;

    if s_tick = '1'then
        f_Send <= not f_Send;
    end if;
end if;
end process p_Sample;

p_BB: Process(Clk, resetN) is
begin
if(resetN = '0') then
elsif (rising_edge(clk)) then
```

```
    if (s_Trigger = '1') then
        TRM <= f_Send;
        else
        TRM <= '0';
    end if;
end if;
end process p_BB;


end achitecture brain;
```

Vedlegg 2

```
library ieee;

use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity tb_dolphitech is

end entity tb_dolphitech;

architecture tb_main of tb_dolphitech is

        constant c_ClkPer      : time := 40 ns;

        component dolphitech

                port (

                        s_Trigger       :       in      std_logic;
                        TRM             :       out     std_logic;
                        c_Sample        :       in      std_logic_vector(2 downto 0);
                        clk, resetN     :       in      std_logic

                        );

        end component dolphitech;


signal clk, resetN              :       std_logic;
signal s_Trigger, TRM           :       std_logic;
signal c_Sample                 :       std_logic_vector(2 downto 0);

begin


        test : component dolphitech

                port map (

                        clk             =>      clk,
                        resetN          =>      resetN,
                        s_Trigger       =>      s_Trigger,
                        TRM             =>      TRM,
                        c_Sample        =>      c_Sample

                        );
```

```
p_clk : process is
begin

        clk <= '0';
        wait for c_ClkPer;
        clk <= '1';
        wait for 10 ns;

end process p_clk;

p_resetN : process is

begin

        resetN <= '0';
        wait for c_ClkPer;
        resetN <= '1';
        wait;

end process p_resetN;

p_main : process is

        procedure tb_init is
        begin

                c_Sample        <= (others => '0');
                s_Trigger       <= '0';

        end procedure tb_init;

        procedure tb_test is
        begin

                wait until rising_edge(clk);
                wait for c_ClkPer;

                c_Sample <= "000";
                wait for c_ClkPer;
                -- Sett frekvens til 1,5MHz

                c_Sample <= "001";
                wait for c_ClkPer;
                -- Sett frekvens til 2,5MHz

                c_Sample <= "010";
                wait for c_ClkPer;
                -- Sett frekvens til 3,5MHz

                c_Sample <= "011";
```

```
                wait for c_ClkPer;
                -- Sett frekvens til 5MHz
                c_Sample <= "100";
                wait for c_ClkPer;
                -- Sett frekvens til 6MHz

                c_Sample <= "101";
                wait for c_ClkPer;
                -- Sett frekvens til 8MHz

                c_Sample <= "110";
                wait for c_ClkPer;
                -- Sett frekvens til 10MHz

        end procedure tb_test;

        procedure tb_trig is
        begin

                s_Trigger <= '1';
                wait for 5000 ns;
                s_Trigger <= '0';
                wait for 5000 ns;

        end procedure tb_trig;
    begin

        tb_init;
        c_Sample <= "000";
        tb_trig;
        c_Sample <= "001";
        tb_trig;
        c_Sample <= "010";
        tb_trig;
        c_Sample <= "011";
        tb_trig;
        c_Sample <= "100";
        tb_trig;
        c_Sample <= "101";
        tb_trig;
        c_Sample <= "110";
        tb_trig;

wait for 3 ms;

assert false report "Pog" severity failure;

    end process p_main;

end architecture tb_main;
```