NTNU
Det skapende universitet

3D object tracking using 2D multibeam sonar

Bjørnar Leithe

Elektronikk
Innlevert: Mai 2013
Hovedveileder: Ulf R Kristiansen, IET
Medveileder: Arne Solstad, Norbit

Norges teknisk-naturvitenskapelige universitet
Institutt for elektronikk og telekommunikasjon

Rapport Masteroppgave
3D object tracking using 2D active multibeam sonar

Bjørnar Leithe og Stian Sønderland

30. mai 2013

Sammendrag

I dette prosjektet har vi laget en prototype for en styrbar sender som kan brukes som utvidelse av en 2D-sonar utviklet av Norbit Subsea AS. Prototypen skal sammen med eksisterende sonar-løsning kunne gi data fra et mål i tre dimensjoner. Data i tre dimensjoner vil gi et bedre grunnlag for en algoritme som kan brukes for tracking av objekter i vann. Prototypen består av et styrekort med 16 kanaler som kan styres individuelt, et utviklerkort for FPGA, en transducer med 16 kanaler og tilhørende software. Spesifikasjoner for protypen er utarbeidet fra krav gitt av Norbit.

Data i to dimensjoner oppnås gjennom bruk av eksisterende beamforming i mottak, noe som gir horisontal vinkel og avstand til målet. Den tredje dimensjonen oppnås ved å sende delvis overlappende styrte ortogonale pulser som i mottak skilles ved bruk av matched filter. Signalstyrken på refleksjoner fra de ortogonale pulsene vi gi et estimat av den vertikale vinkelen.

Vi har vurdert forskjellige pulstyper for å møte spesifikasjonene og funnet ut at pulstypen chirp passer best til vårt formål. Kretskortet er designet med tanke på å få god nok sendestyrke for transduceren med 16 styrbare kanaler, og med tanke på å enkelt kunne gjøre tilpassinger. For å få en styrbar sender har vi bygget vår egen transducer som består av 32 parivs parallellkoblede keramiske elementer, slik at vi har 16 kanaler. Styresignalene blir generert med en FPGA som står på et utvikerkort med nødvendige grensesnitt. Prototypen har mulighet for retningsstyring og defokusering av de utsendte pulsene gjennom et brukergrensesnitt. For å analysere de utsendte signalene har vi tatt opp rådata fra en sonar og prosessert de med Matlab i etterkant.

Testene viser at prototypen fungerer for å gjøre posisjonsbestemmelse i tre dimensjoner av et kjent mål i vann. Vi har også verifisert at brukergrensesnittet gjør at de utsendte pulsene kan tilpasses til miljøet det gjøres målinger i.

Abstract

In this project we have made a prototype for a steerable acoustic transducer which can be used as an extension for a 2D sonar developed by Norbit Subsea AS. Combined with the 2D sonar, the prototype will expand the functionality of the sonar system so that it will be able to give a targets position in three dimensions. Data in three dimensions will give a better basis for an algorithm that can be used to track moving objects in water. The prototype consists of a transmitter PCB with 16 channels that can be controlled individually, a development kit for FPGA's, a transducer with 16 channels and software for signal control and reception analysis. Specifications for the prototype are made from requirements given by Norbit.

Data in two dimensions are obtained by using 2D beamforming in reception, which means that we get data with respect to distance and horizontal angle. Data from the third dimension is achieved by transmitting two partially overlapping orthogonal pulses that are separated with matched filtering in the receiver. Signal strength from reflections of the orthogonal pulses will give an estimate of a targets vertical angle.

We have evaluated different pulse types for the transmitted signal and concluded that chirp will best suited for this application. The PCB is designed to give enough power to drive 16 channels and with possibility of making adjustments as we go. To get a steerable transducer we have made our own, consisting of 32 ceramic elements that are parallel coupled in pairs to give 16 channels. The control signals are generated in the FPGA on a development board with necessary interfaces. The prototype has a user interface which allows for changing of the steering angle and defocusing of the signal to get a wider detection area. To analyse the recorded data we used raw data from the 2D sonar and post processing in Matlab.

We have done measurements in water that shows that the prototype can expand the sonar system to give a known targets position in three dimensions. We have also verified that the user interface lets the user adapt the system to the surrounding environment.

Innhold

Figurer

Tabeller

Vedlegg

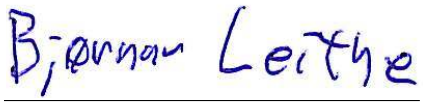
Forord

Denne rapporten er utarbeidet av Bjørnar Leithe og Stian Sønderland ved NTNU, på oppdrag fra Norbit Subsea AS. Rapporten dekker masteroppgaven i det avsluttende året for 2-årig masterstudium, hovedprofil signalbehandling akustikk og media. Masteroppgaven utgjør 30 studiepoeng og er det største enkeltprosjektet i studiet. Temaet er valgt i samråd med Norbit Subsea og faglærer basert på behov spesifisert fra oppdragsgiver og er en videreføring av fordypningsprosjektet høsten 2012. [5]

Vi har valgt et praktisk retning på oppgaven fordi vi ønsket å lage noe fysisk samt at vi tror dette vi forberede oss på jobben som ingeniører etter oppgaven er ferdig. Med dette som utgangspunkt har vi satt oss inn i relevant teori, gjort simuleringer, laget en prototype og gjort tester av denne for å verifisere at løsningene vi har valgt møter kravene fra oppdragsgiver.

Vi vil takke Ulf Kristiansen, Arild Søraunet og Arne Solstad for veiledning, samt Norbit Subsea for å stille med arbeidslokaler og testutstyr. Vi vil også rette en spesiell takk til Hans Rechsteiner hos Norbit for bistand ved bygging av transducer, og Svenn-Inge Sellesbakk og Tormod Vaule hos Norbit for bistand med skjemategning og utlegg.

Trondheim, 30. mai 2013



Bjørnar Leithe

Stian Sønderland

1 Introduksjon

Denne rapporten beskriver hvordan vi har undersøkt muligheten for å implementere en ny funksjonalitet i et eksisterende produkt. Rapporten inneholder en teoridel, en beskrivelse av hvilke krav løsningen må oppfylle og en spesifikasjonsdel som beskriver hvordan vi tenker å møte kravene. Videre presenteres komponentberegninger, design av kretskort og funksjonalitet for software til styresystemet. Vi gir beskrivelser av det praktiske arbeidet som har bestått av å bygge vår egen transducer, lodding av kretskort, designtester og test av prototype i vann. Til slutt har vi skrevet om muligheter for videre arbeid og vi gir vår konklusjon på oppgaven.

1.1 Eksisterende løsning

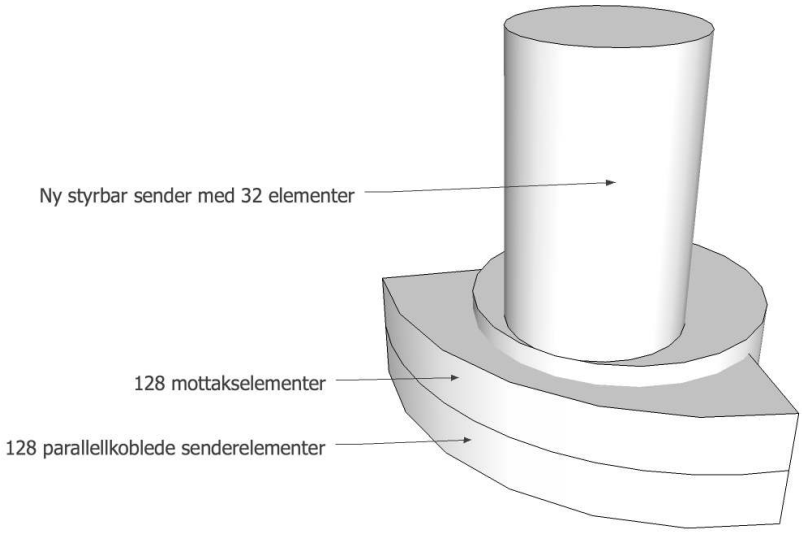
Norbit Subsea har utviklet en multistråle-sonar som har 128 elementer for sending og 128 elementer for mottak. Elementene er arrangert i et todimensjonalt array og de er jevnt spredt over en bue på 90°. Elementene for sending er parallellkoblet og sender det samme signalet. Dette gjør at man ikke har mulighet for retningsstyring av utsendt signal. For mottak bruker sonaren beamforming på de 128 mottakselementene, slik at man kan bestemme et måls lokasjon i to dimensjoner. Dette blir beregnet ut fra tidsforskjellen fra når en refleksjon fra et mål treffer hvert av mottakselementene og tiden det tar fra et signal blir sendt ut til det registreres en refleksjon. Databladet for sonaren finnes i vedlegg I. Senderdelen består av en kondensatorbank som blir ladet opp av en ladekrets til $\pm$100V. Denne spenningen brukes av et utgangstrinn som består av MOSFET-transistorer som blir styrt med et PWM-signal av softwaren i sonaren.

1.2 Vår oppgave

Vår oppgave går ut på å lage en prototype for en styrt sender som i kombinasjon med dagens løsning skal kunne gi data i tre dimensjoner. Dette skal brukes for tracking av objekter. For å lettere kunne klassifisere et mål er det ønskelig med data i tre dimensjoner.

Den nye styrte senderen vil ha 32 elementer som står vertikalt opp fra det eksisterende horisontale arrayet for sending og mottak. Vi planlegger å sende to delvis overlappende pulser, den ene styrt skrått nedover og den andre styrt skrått oppover. Ved å analysere de mottatte pulsene vil man kunne si noe om hvor stor andel av oppoverpuls og nedoverpuls som har blitt reflektert, og dermed gi et estimat av den vertikale posisjonen til målet. Ved analyse av det mottatte signalet må man kunne skille de to pulsene. Dette skal løses ved å bruke ortogonale pulser og matched filtrering.

Elektronikken som skal styre senderen skal ha 16 kanaler som styres individuelt. En modell av sonaren med den nye senderen vises i figur 1.



Figur 1: Wideband Multibeam Sonar med ny sender

2 Teori

2.1 Nærfelt og fjernfelt

Så lenge avstanden til en transducer er stor nok så vil ikke frekvensrespons og direktivitet variere med endringer i avstanden. Dette kalles fjernfelt. Når avstanden blir liten så vil det medføre at bølger fra ulike deler av transduceren ikke adderes på samme måte som de gjør i fjernfelt. Direktiviten og eventuelt frekvensresponsen blir da ikke den samme. Dette kalles nærfelt. Ved måling av direktiviteten til en transducer er det derfor viktig at avstanden mellom mottakeren og senderen er stor nok. Avstanden som sikrer at vi er i fjernfeltet, og dermed kan anta at alle signalene når hydrofonen med samme fase, kan beregnes som to ganger Rayleigh avstanden $\frac{L^2}{2\lambda}$ [3]{Ch-12.7.1}:

$$R \geq 2 \cdot \frac{L^2}{2\lambda} \tag{1}$$

Hvor L er lengden for arrayet i meter.

2.2 Strålestyring

For å ha mulighet til å styre retningen på et utsendt signal brukes strålestyring / beamstearing. Lobemønsteret fra et array av senderelementer kan styres ved å gi en tidsforsinkelse eller faseforskyvning mellom hvert av elementene. Dette kan enklest forklares ved å beskrive det inverse, altså i mottak. Når en reflektert planbølge fra et mål plassert skrått fremfor arrayet kommer frem, vil den nå den ene enden av arrayet først. Etterhvert vil den nå den andre enden og ved hjelp av å se på tidsforskjellen når bølgen når hvert element vil man kunne estimere vinkelen refleksjonen kommer fra. Databehandlingen i sonaren inneholder en beamformer som omdanner data fra de 128 kanalene til 128 stråler. Da sending og mottak er inverse operasjoner vil det samme prinsippet gjelde for sending. Formel 2 beskriver lobemønsteret for et array [3]{EQ-A1316}, og formel 3 beskriver med styrt lobemønster ved hjelp av tidsforsinkelse [2]{EQ-9.99}. N er antall elementer i arrayet, L er lengden på hvert element, s er senter senter separasjon mellom elementene, c er lydhastigheten, t_d er tidsforsnkelsen og λ er bølgelengden. Formel 4 viser hva tidsforsinkelsen mellom hvert element må være for å oppnå en gitt vinkel på utsendt puls.[1]

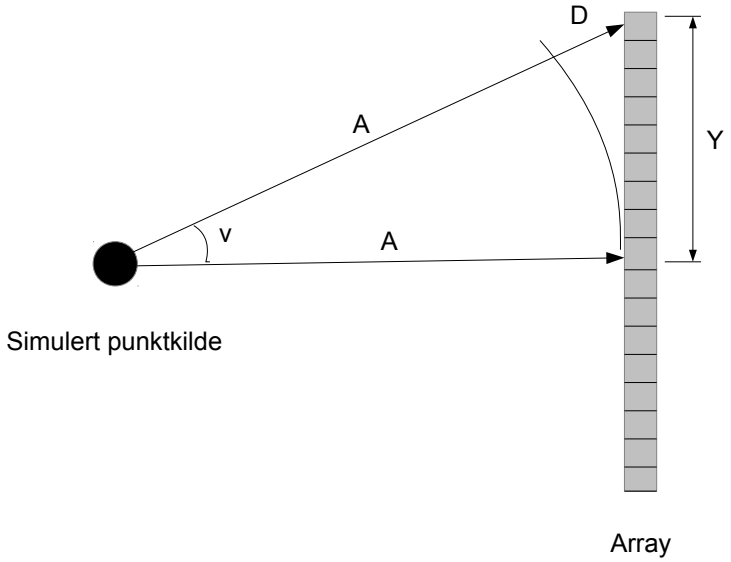
$$P(\theta) = \frac{\left(\frac{\sin(\pi \cdot L/\lambda \cdot \sin(\theta))}{\pi \cdot L/\lambda \cdot \sin(\theta)}\right) \cdot sin\left(N \cdot \pi \cdot s/\lambda\right)}{N \cdot \sin\left(\pi \cdot s/\lambda \cdot \sin(\theta)\right)} \tag{2}$$

$$P(\theta)_s = \frac{\left(\frac{\sin(\pi \cdot L/\lambda \cdot (sin(\theta) - t_d \cdot c/s))}{\pi \cdot L/\lambda \cdot (\sin(\theta) - t_d \cdot c/s)}\right) \cdot \sin\left(N \cdot \pi \cdot s/\lambda\right)}{N \cdot \sin\left(\pi \cdot s/\lambda \cdot (\sin(\theta) - t_d \cdot c/s)\right)} \tag{3}$$

$$\tau = \frac{\lambda}{c}\left(\cos\left(\theta_c\right) - \cos\left(\theta_s\right)\right) \tag{4}$$

2.3 Defokusering av array

Direktiviteten til et array vil normalt sett være gitt av størrelsen og det totale antallet elementer. Når vi har mulighet til å individuelt endre fasen på de 16 kanalene så kan vi også velge å defokusere arrayet slik at loben blir bredere enn det geometriene i seg selv tilsier. Måten vi gjør det på er å simulere at vi har en punktkilde som er plassert med en avstand A bak arrayet. Gangveien for lyden frem til elementene på midten blir da A. Ut mot kanten blir gangveien lengre, differansen D utgjør det delayet vi kan konfigurere i senderen. Avstanden fra senterelementet til elementet vi skal kalkulere delay for er kalt Y. Dette er vist i figur 2. Ved bruk av Pytagoras' læresetning har vi kommet frem til formel 6 for å beregne delay for hver kanal ved defokusering.



Figur 2: Illustrasjon av defokusering

$$(A+D)^2 = A^2 + Y^2 \quad (5)$$

$$D = \sqrt{A^2+Y^2} - A \quad (6)$$

2.4 PWM

Pulse Width Modulation / Puls-bredde modulasjon er variering av påtiden i et firkant-pulstog uten å endre grunnfrekvensen. Ved å endre på hvor lang påtiden i en puls er, vil effektivverdien av spenningen endres. Dette brukes som en digital til analog konvertering av binære verdier. Teknikken brukes for å tilnærne en sinus ut fra digitale kretser som har bare to spenningsnivåer, av og på. Det gir høy virkningsgrad fordi transistorene ikke må jobbe i det aktive området.

2.5 Harmonisk forvrengning

Harmonisk forvregning skjer når et signal føres gjennom en ikke-ideell komponent. På grunn av ulinearitet i komponenten så blir det lagt til overharmoniske frekvenskomponenter i signalet. Klirrfaktor / THD er en måte å måle harmonisk forvregning på. Klirrfaktoren bestemmes av forholdet mellom amplituden på grunnfrekvensen og summen av amplitudenivåene for de overharmoniske frekvensene, når man har en ren sinus på inngangen til komponenten [10].

$$THD = \frac{\sqrt{V_2^2 + V_3^2 + V_4^2 + ...V_{\text{inf}}^2}}{V_1} \tag{7}$$

2.6 Chirp

Et chirp er en puls som er basert på en sinus hvor frekvensen varierer i løpet av pulsens varighet. Et lineært chirp er gitt av formel 8 hvor t er den løpende variabelen, f_0 er nedre frekvens, f_1 er øvre frekvens og T er varigheten. Ved å spille av chirpet baklengs får man en ortogonal puls, det vil si at de to pulsene kan skilles ved bruk av matched filtrering. [5]

$$Y = \sin\left(2\pi \cdot t \cdot \left(f_0 + t \cdot \frac{f_1 - f_0}{T}\right)\right) \tag{8}$$

2.7 Barker-sekvenser

Barker-sekvenser er sekvenser av -1 og +1 som har ideelle autokorrelasjonsegenskaper, det vil si at alle autokorrelasjonskoeffisientene utenom toppen er i området -1 til +1. Ved modulasjon av disse sekvensene må man bruke symboler som kansellerer hverandre. Fasemodulasjon (BPSK) er derfor mye brukt.

Barker-sekvenser som er kjent har lengde fra 2 til 13. For lengdene 2 og 4 finnes det to komplementære sekvenser. Det er antatt at det ikke finnes flere sekvenser enn de som er kjent [9].

2.8 Kasami-sekvenser

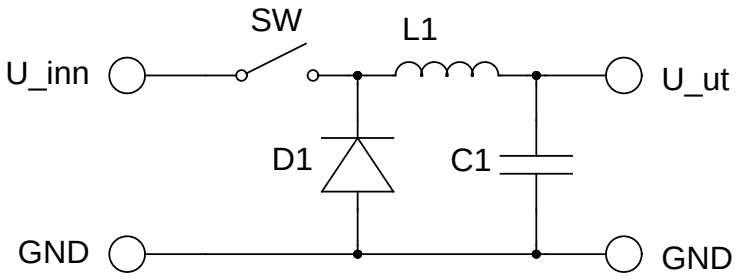
Kasami-sekvenser er sekvenser av -1 og +1 som har gunstige autokorrelasjonsegenskaper, men ikke ideelle som Barker-sekvenser. Sekvensene er generert ved å ta utgangspunkt i en maksimum-lengde-sekvens og gjøre en bitvis *XOR* av den med seg selv med ulike forskyvninger [7]. Man genererer da et sett av Kasami-sekvenser. Sekvensene kan genereres med lengder gitt av $L = 2^{2N} - 1$, som er lengdene til maksimum-lengde-sekvenser.

2.9 FPGA

En FPGA (Field Programmable Gate Array) er et en integrert krets som kan konfigureres til å fylle ulike funksjoner. I motsetning til en mikrokontroller som kjører et program, instruksjon for instruksjon, så programmeres en FPGA ved at den inneholder logiske elementer som man kan koble sammen i ønsket konfigurasjon. Ut i fra størrelsen på FPGA-en og hvordan man programmerer den så kan man få alt fra helt enkle funksjoner som *OG* og *ELLER* til komplekse elementer som mikroprosessorer. FPGA-er konfigureres typisk ved hjelp RTL-kode i språk som VHDL eller Verilog. Hvis man konfigurerer en FPGA til å inneholde en mikroprosessor så får man ekstra fleksibilitet ved at man kan skrive programkode til mikroprosessoren uavhengig av FPGA-konfigurasjonen. Til det bruker man konvensjonelle programmeringsspråk som for eksempel C.

2.10 Buck converter

En buck converter er en enkel topologi for switch-mode DC/DC-convertere som kan brukes til å generere en utgangsspenning som er lavere enn inngangsspenningen, med høy effektivitet. Figur 3 viser hvordan en buck-converter er bygd opp.

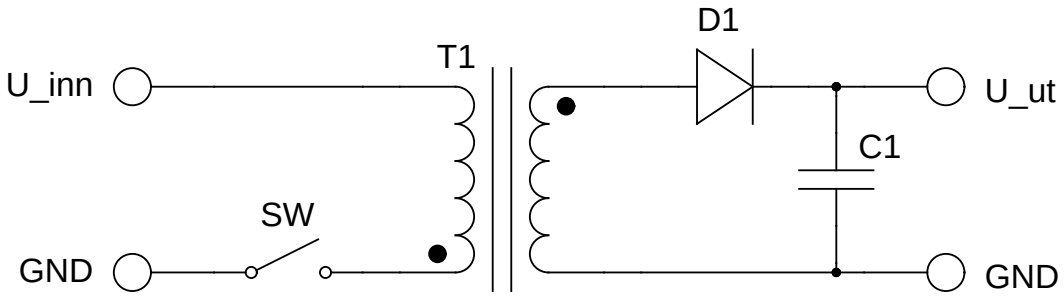


Figur 3: Oppbygning av buck converter

Bryteren SW er normalt sett en MOSFET som styres av en kontroller for å regulere utgangsspenningen. Dioden D1 leder normalt sett når SW ikke leder, slik at det leveres en kontinuerlig strøm gjennom L1 til lasten.

2.11 Flyback converter

En flyback converter er en topologi for switch-mode DC/DC-convertere. Det spesielle med denne topologien er at den mellomlagrer energien i en transformator før den overføres til utgangssiden. På grunn av at man har en transformator med mulighet for å velge viklingsforhold mellom inn- og utgangsside så er den spesielt godt egnet når det er stort forhold mellom inngangs- og utgangsspenning, hvor man ville fått ugunstig høy eller lav duty-cycle med enklere topologier uten transformator. Transformatoren gjør det også mulig å ha galvanisk skille mellom inngang og utgang. Figur 4 viser hvordan en flyback-converter er bygd opp. I praktiske implementasjoner vil ofte SW være en MOSFET-transistor som styres av en kontroller. Kontrolleren overvåker ofte spenning og strøm på inn- eller utgangsside og styrer transistoren basert på det, for eksempel for å oppnå en forhåndsbestemt utgangsspenning.



Figur 4: Oppbygning av flyback converter

Operasjonen til en flyback converter består av to faser. I oppladingskretsen er bryteren SW lukket. Det fører til at inngangspenningen blir påtrykket på inngangssiden av transformatoren. Spenningen på utgangssiden blir da negativ, slik at D1 ikke leder. Strømmen i transformatoren vil øke etter hvert som det bygger seg opp et magnetfelt i transformatoren. Magnetfeltet utgjør en opplagret energi. Hvor fort strømmen og magnetfeltet vil bygge seg opp avhenger av størrelsen på inngangsspeningen og induktansen til transformatoren. I utladningsfasen blir SW åpnet. På grunn av det lagrede magnetfeltet i transformatoren så vil det føre til at spenningen på inngangssiden av transformatoren blir negativ, og spenningen på utgangssiden blir positiv. D1 vil da begynne å lede, og energien vil overføres til kondensatoren C1. Spenningen på inngangssiden av transformatoren vil i løpet av denne fasen være gitt av utgangsspenningen og viklingsforholdet i transformatoren, så det er mulig for en kontroller å regulere spenningen på utgangen bare ved å se på spenningen på inngangsiden av transformatoren. Når all energien i transformatoren er overført til C1 så vil D1 slutte å lede. SW må da lukkes for å starte en ny oppladingsfase. Kontrolleren kan eventuelt velge å vente dersom det ikke er ønskelig å overføre mer energi til utgangssiden, f.eks. fordi ønsket spenning er oppnådd.

2.12 Keramisk transducer

Akustiske transducere finnes i flere varianter og ofte blir elektroakustiske transducere basert på piezo-elektrisitet brukt i undervannsakustikk. Et piezo-elektrisk materiale vi generere et elektrisk felt når det blir utsatt for lydtrykk, og motsatt vil det generere et lydtrykk når det blir utsatt for et elektrisk felt. Dette gjør at materialet kan brukes i både sendere og mottakere. Det finnes to hovedtyper av piezo-elektriske transducere. Den ene blir tykkere eller tynnere avhengig av polariteten på det elektriske feltet. Deformasjonen av transduceren skjer i aksial retning og kalles 33-mode. Den andre typen piezo-elektrisk transducer blir deformert normalt på det elektriske feltet og kalles 31-mode. Resonansfrekvensen for en transducer som opererer i 31-mode kan beregnes med følgende formel.[4]

$$f_{resonance} = \frac{N_{31}}{L} \tag{9}$$

Hvor L er tykkelsen på elementet og N_{31} er et frekvensfaktor basert på typen piezomateriale.

2.13 PCB

PCB er en engelsk fellesbetegnelse på det vi på norsk kaller mønsterkort og kretskort. Mønsterkort er et kort av isolerende materiale som er belagt med et ledende materiale. Kortet brukes for å samle elektroniske komponenter og ved å benytte det ledende laget kan man lage forbindelser mellom komponentene. Dette gjøres ved å først tegne et skjema som viser komponentene og forbindelsene mellom dem. Så lages et utlegg av hvor komponentene skal plasseres og komponentene får loddeland for innfesting og signalføring. Signalbanene mellom komponentene tegnes også inn i utlegget.

I utgangspunktet er hele kortet belagt med ledende materiale, og for å få loddeland og ledende baner som man har tegnet fjernes uønsket ledende materiale. Utlegget trykkes ved hjelp av en fotolitografisk prosess på mønsterkortet slik at man får et avtrykk der hvor det skal være komponenter og baner. Neste steg er å legge kortet i en spesiell syre som etser bort det ledende laget der hvor det ikke er avtrykk for komponenter og baner. Mønsterkort kan lages i flere lag og man kan lage forbindelser mellom lagene med viahull.

Etter at mønsterkortet er etset kan man montere på komponentene. Dette gjøres ved å lodde de fast til loddelandene. Når kortet har komponenter kalles det kretskort.

3 Krav og spesifikasjoner

Under oppstart av oppgaven hadde vi møte med oppdragsgiver hvor krav til systemet ble presentert. Kravene er utarbeidet fra et markedsmessig ståsted for å kunne møte konkurrerende løsninger. Vi har delt opp kravene i tre kategorier og nummerert de for å lettere kunne referere til dem. I kapittel 3.1 har vi listet opp kravene og i kapittel 3.2 presenterer vi spesifikasjoner på våre løsninger som skal møte kravene.

3.1 Krav fra oppdragsgiver

3.1.1 Krav til utsendt puls

P.1 Tidsoppløsning
Hvis pulsene som sendes ut er repetive vil matched filteret i mottak detektere flere ekko innenfor samme puls. Kravet er at mellom 30µs og 100µs før og etter en peak, skal det være minst 20dB nivåforskjell mellom en peak og fantomrefleksjoner. Mer enn 100µs før og etter skal det være minst 30dB nivåforskjell.

P.2 Isolasjon mellom to pulser
Isolasjonen mellom de to ortogonale pulsene skal være så god som mulig når de andre kravene er oppfylt (helst -25dB).

P.3 Pulslengde
Pulslengden skal være justerbar mellom 100µs og 1ms.

P.4 Båndbredde
Båndbredde for den utsendte pulsen skal utnytte seg av hele området som transduceren har god nok sendestyrke. Det vil si innenfor -3dB i forhold til resonansfrekvens.

3.1.2 Krav til sender

S.1 Interface
Prototypen bør være konfigurerbar fra en PC via ethernet.

S.2 Brukergrensesnitt
Det er ønskelig med et brukergrensesnitt som gjør at prototypen kan konfigureres for å tilpasse til ulike formål.

S.3 Retningsstyring
Det skal være mulighet for å endre retning på det utsendte signalet relativt fort gjennom brukergrensesittet.

S.4 Antall kanaler
Minst 8 helst 16 kanaler for tilkobling til transducer.

S.5 Prototype
Det skal lages en prototype. Her bør man vektlegge funksjonalitet og mulighet for endringer fremfor fysisk størrelse. Prototypen skal testes og resultatene skal kunne gi grunnlag for implementering i eksisterende sonar.

S.6 Forvregningskrav
-20dB (10% THD) for 400kHz ved full amplitude.

S.7 Båndbredde
Båndbredde på det utsendte signalet skal være på ca 100kHz - 1.5MHz.

S.8 Oppløsning på retningsstyring
Oppløsningen på retningsstyringen skal være 0.25°, ved lamda spacing mellom elementer og 400kHz.

S.9 Sidelobenivå
Pulsen som sendes ut bør ha et sidelobenivå på -25dB i beste konfigurasjon.

S.10 Kondensatorbank
Energien i pulsen bør kunne være 4 ganger det den er i dagens løsning. Dette kan implementeres med eksisterende kondensatorbank og i tillegg elektrolyttkondensatorer som eventuelt settes på eget kretskort.

S.11 Ladekrets
Ladekretsen som er i dagens løsning for å generere ±100V må kontrollregnes for å se om den er stor nok til å takle den nye kondensatorbanken innenfor fornuftig oppladningstid.

S.12 Utgangstrinn
Utgangstrinnene må tilpasses slik at de tåler 4 ganger effekt, samt at de må være individuelt styrbare.

3.1.3 Krav til mottaksanalyse

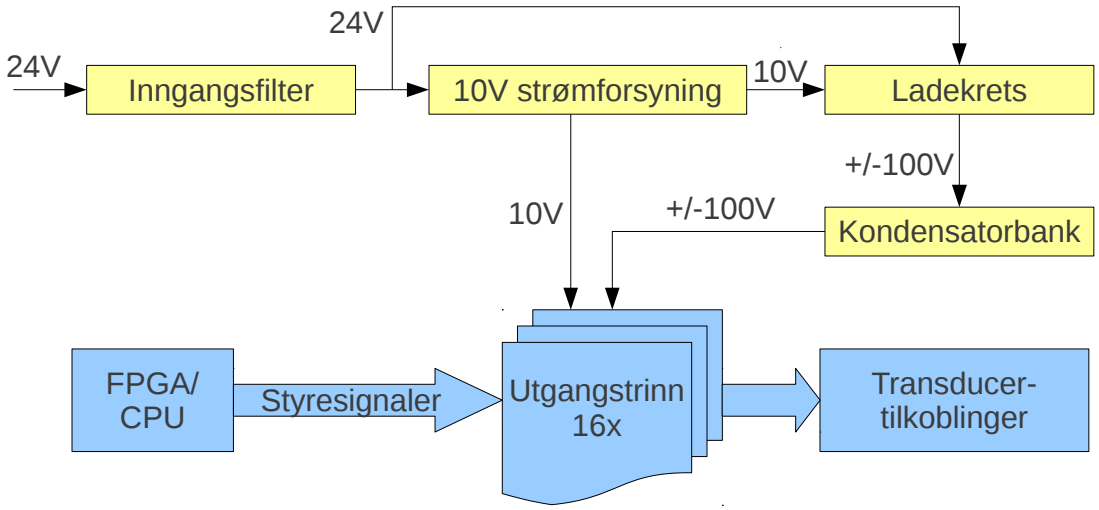
M.1 Analysemetode
Det finnes mulighet for å hente ut rådata fra beamformeren i sonaren, men det er vanskelig å få ut alle 128 kanalene samtidig. I utgangspunktet er det nok med rådata fra én kanal som prosesseres med Matlab i etterkant.

M.2 Sanntidsanalyse
Om det blir tid til overs kan det undersøkes mulighet for implementering av mottaksanalyse i sonarens FPGA. Dermed åpnes muligheten for analyse av flere kanaler samtidig og i sanntid.

M.3 Objekt-tracking
Det kan vurderes å legge inn en algoritme for deteksjon av objekter og tracking av disse i sonarens FPGA. Dette forutsetter at man har implementert krav M.2.

3.2 Spesifikasjoner

Ut fra kravene fra oppdragsgiver har vi utarbeidet spesifikasjoner. Vi vil forsøke å verifisere at alle kravene blir oppfylt, både ved beregninger og designtester. Vi har utarbeidet et enkelt blokkskjema for den styrte senderen. Skjemaet vises i figur 5.



Figur 5: Blokkskjema

3.2.1 Spesifikasjoner for utsendt puls

P.1 Tidsoppløsning
For å undersøke kravet til tidsoppløsning analyserte vi pulstypene som ble designet i høstprosjektet [5]. I høstprosjektet ble det ikke tatt hensyn til tidsoppløsningen. Denne analysen ble gjort i Matlab.

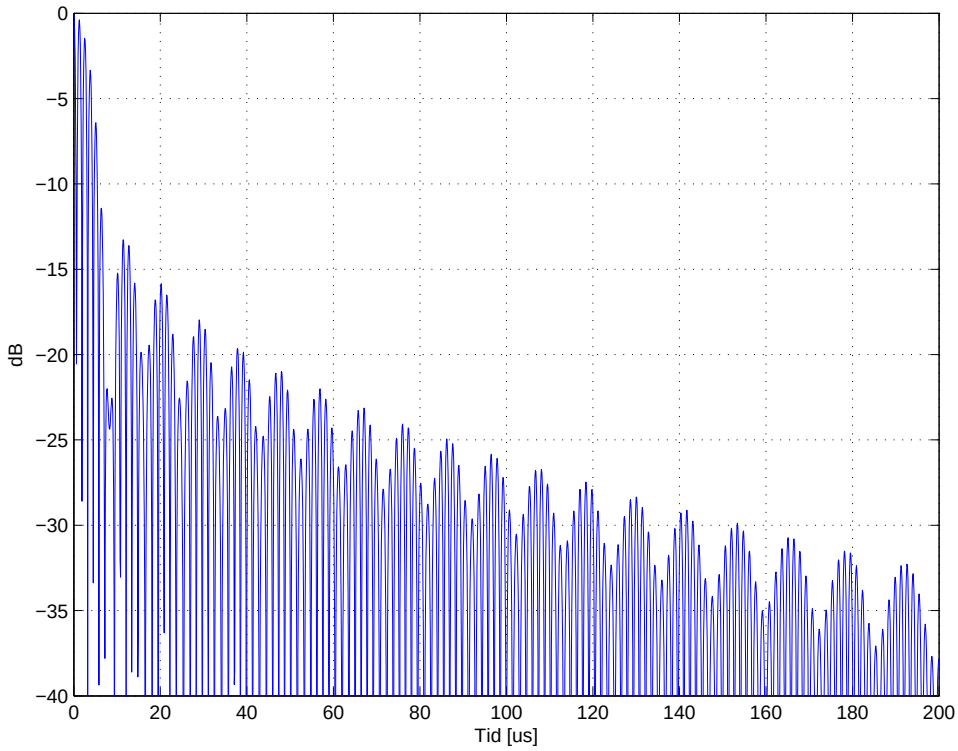
Chirp (kvadratisk konkav): Ved en pulslengde på 1ms fikk vi etter 30µs 18.6dB og etter 100µs hadde vi 26.7dB nivåforskjell mellom peak og fantomrefleksjoner, se figur 6. Kvadratisk konkav chirp med 1ms pulslengde oppfyller dermed ikke kravet til tidsoppløsning. For å få til nok tidsoppløsning måtte vi redusere pulslengden til ca 250µs, men da blir isolasjonen mellom to ortogonale pulser sterkt redusert.

Chirp (lineært): Ved en pulslengde på 1ms fikk vi etter 30µs -23.7dB og etter 100µs hadde vi -33.7dB nivåforskjell mellom peak og fantomrefleksjoner, se figur 7. Dette oppfyller kravet til tidsoppløsning, men isolasjonen mellom to ortogonale pulser er bare 20.5dB.

Kampuls: Ved analyse av kampulsen ser vi klart at denne pulstypen er repetiv, se figur 8. Denne vil derfor ikke oppfylle kravene til tidsoppløsning.

Komplementære sekvenser For å prøve å bedre møte kravet til tidsoppløsning har vi også undersøkt ulike former for komplementære sekvenser. Dette er binære sekvenser som fasemoduleres på bærebølgen. For oss er det viktig både at autokorrelasjonen til disse sekvensene gir god tidsoppløsning og at krysskorrelasjonen er lav slik at vi får god isolasjon mellom pulsene.

Barker-sekvenser har ideelle autokorrelasjonsegenskaper, men de eksisterer bare med lengde opp til 13, og komplementære par finnes bare med lengde på 2 og 4. Isolasjonen til disse vil være lik lengden, altså $20 \log 4 = 12$dB for sekvensene med lengde på 4. Det er for lite for oss.



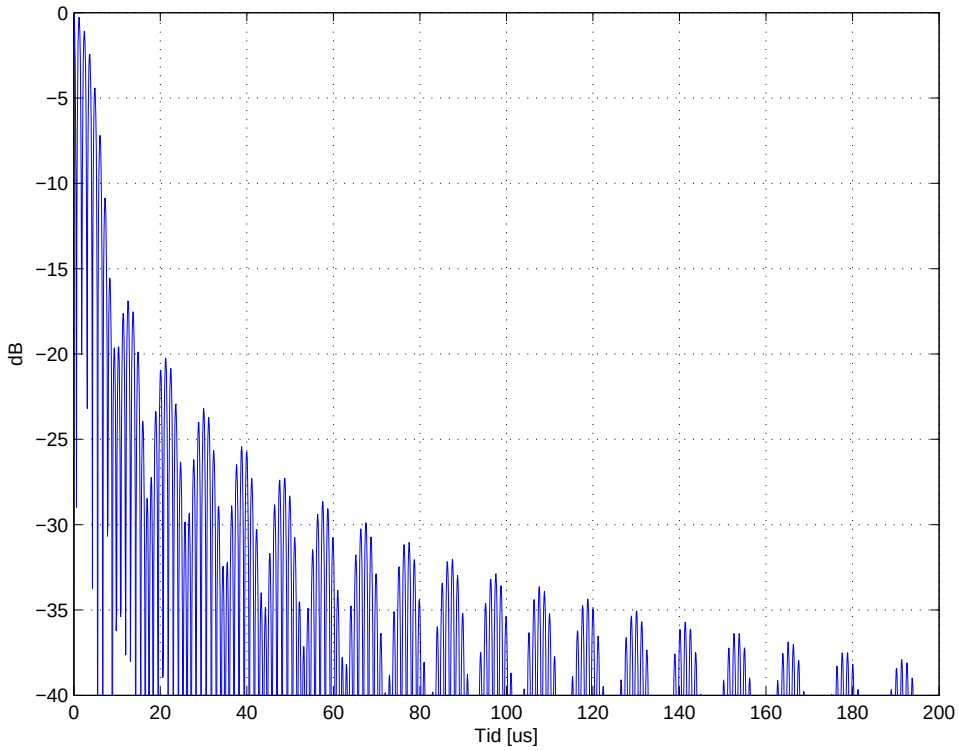
Figur 6: Tidsoppløsning for kvadratisk konkav chirp med en pulslengde på 1ms

Vi har derfor forsøkt med Kasami-sekvenser, som kan genereres i ønsket lengde. Lengden vi kan bruke er begrenset av båndbredden i transduceren. Ut i fra en båndbredde på 120kHz og pulsvarighet på 1ms så vil optimal lengde på sekvensen være 120 elementer. Kasami-sekvenser kan genereres i lengder på $L = 2^{2N} - 1$, altså er det 63 og 255 som er de nærmeste. Vi har testet ut Kasami-pulser med disse to lengdene på sekvensene.

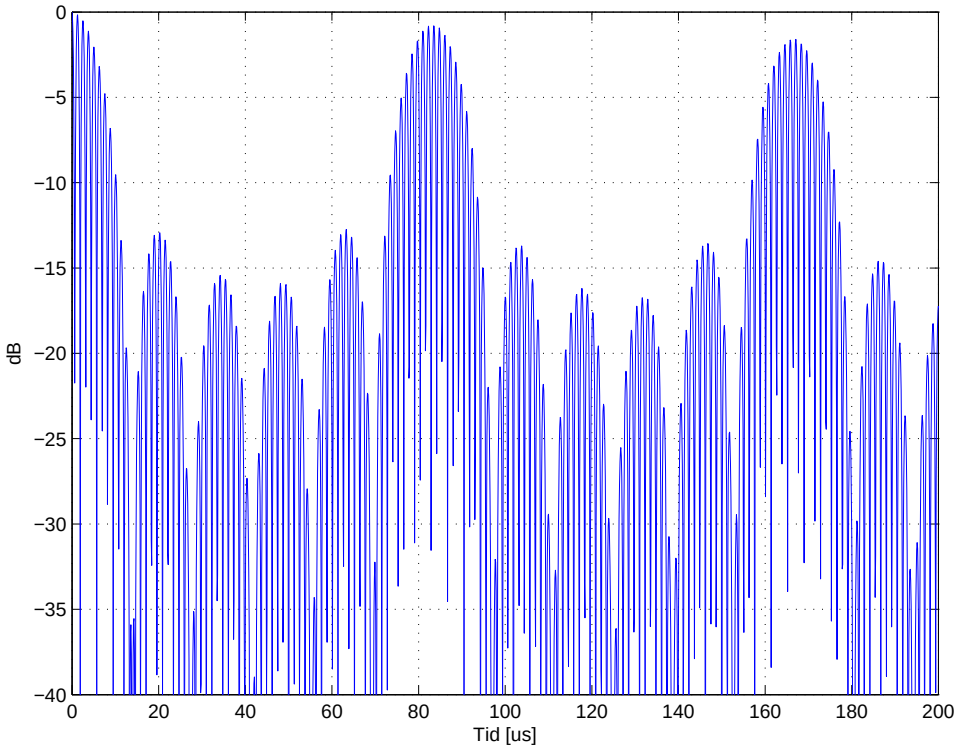
Med sekvenslengde på 63 får vi -12.5dB isolasjon og sidelobenivå i tidsoppløsningen på -22.5dB. Dette er omtrent upåvirket av filtreringen til transduceren. Med en sekvenslengde på 255 får signalet i utganspunktet for stor båndbredde til transduceren, så det taper seg veldig gjennom transduceren. Vi får da en isolasjon på 20.5dB før og 16.4dB etter transducer. Sidelobenivået er -27dB før og -23dB etter transducer. Dette er som forventet når halve båndbredden i signalet i praksis filtreres bort. Figur 9 viser autokorrelasjonen til en slik puls etter transduceren.

Vi har også forsøkt pulser med L=1023. Da får vi ca. -30dB sidelobenivå og -25dB isolasjon før transduceren, men etter transduceren er nivåene som forventet omtrent som for pulsen med L=255.

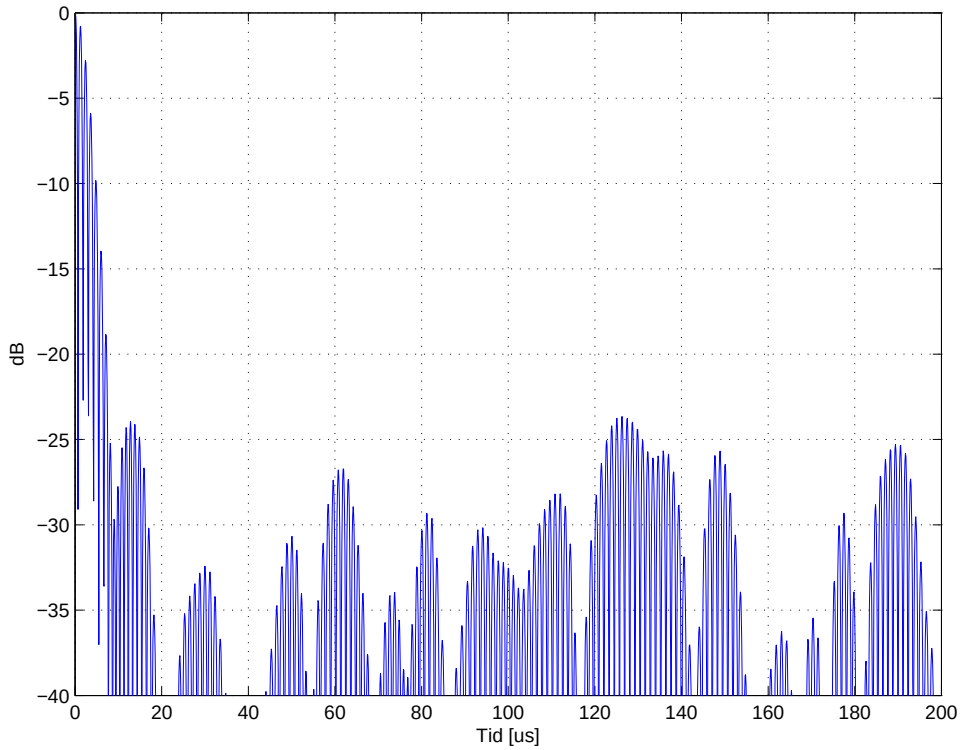
Av disse simuleringene ser vi at ingen av pulstypene oppfyller både kravet til tidsoppløsning og isolasjon. Vårt valg av pulstype blir lineært chirp da dette er nærmest å oppfylle kravene, samt at det er den samme pulstypen som benyttes i dagens sonar.



Figur 7: Tidsoppløsning for lineært chirp



Figur 8: Tidsoppløsning for kampuls

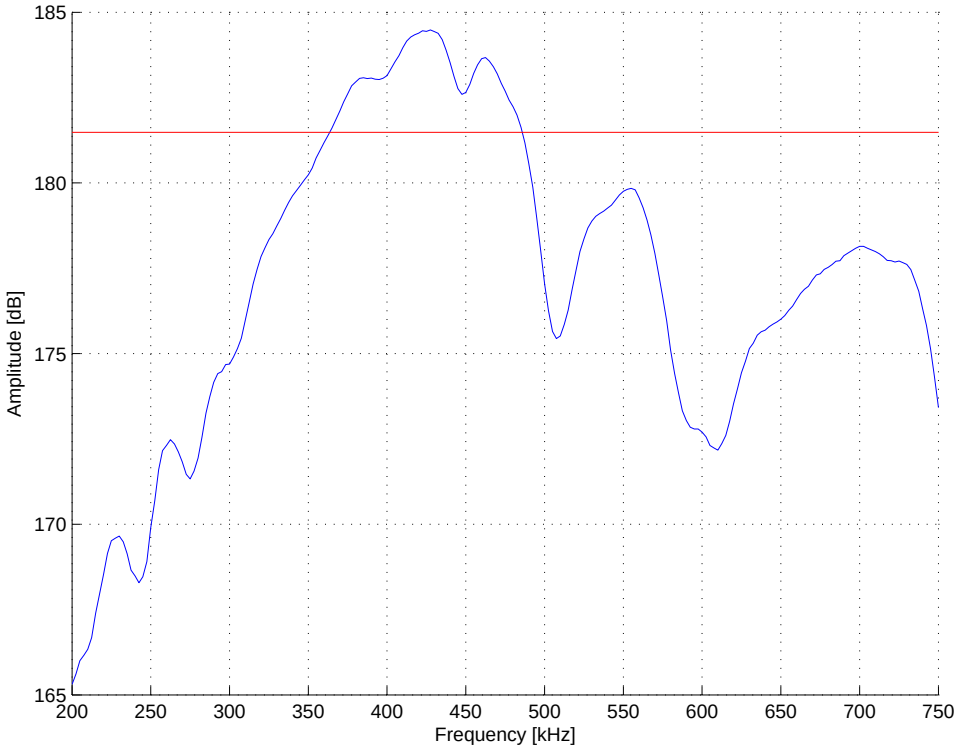


Figur 9: Tidsoppløsning for kasami-puls med L=255

P.2 Isolasjon mellom to pulser
Lineært chirp som er valgt ut i fra en avveing av alle kravene klarer ikke å møte kravet på 25dB isolasjon. Isolasjonen mellom de ortogonale pulsene blir 20.5dB.[5]

P.3 Pulslengde
Kravet til justerbar pulslengde blir oppfylt gjennom å ta dette med i programvaren for FPGA-en.

P.4 Båndbredde
Av figur 10, som viser frekvensresponsen til en transducer med lignende oppbygning, ser vi at en grense på -3dB gir en båndbredde på 122kHz. Kravet til båndbredde gir da et frekvensområde fra 364kHz til 486kHz for den utsendte pulsen.



Figur 10: Frekvensrespons til transduceren

3.2.2 Spesifikasjoner for sender

Vi vil lage et kretskort for den styrbare senderen som består av en strømforsyning, ladekretser for ±100V, kondensatorbanker og 16 individuelt styrbare utgangstrinn.

S.1 Interface
Kretskortet vi skal designe vil være avhengig av eksterne styresignaler for å fungere. Utgangstrinnene skal styres av 16 pwm-utganger. For å generere så mange uavhengige høyfrekvente signaler har vi funnet det mest hensiktsmessig å bruke en FPGA. Vi har valgt å bruke et utviklerkort for FPGA som har en rekke grensesnitt. Kortet vi bruker er Altera Cyclone II FPGA Starter Board. Dette kortet ble valgt fordi oppdragsgiver hadde det tilgjengelig, og det har nødvendige kapasitet og nødvendig antall I/O tilgjengelig på

connector. Kortet er utstyrt med en Cyclone II EP2C20F484C7N FPGA, 512KB SRAM, 4 siffer 7-segment display, 10 av/på-brytere, 4 trykk-knapper og 18 LEDs, i tillegg til en del komponenter som vi ikke benytter oss av. For å styre vårt prototypekort bruker vi parallell I/O. Kortet har muligheter for kommunikasjon mot PC, blant annet SD-kort og RS-232. Vårt kort har ikke ethernet, men det finnes tilsvarende kort som har det.

S.2 Brukergrensesnitt
Vi velger å lage et grensesnitt som benytter knapper, brytere og display på utviklerkortet.

S.3 Retningsstyring
Vi vil lage et styresystem som er forberedt for hurtig endring av retning gjennom brukergrensesnittet.

S.4 Antall kanaler
Vi velger å lage en prototype med 16 kanaler. Vi kan deretter gjøre tester hvor vi reduserer antallet kanaler for å se hvordan dette påvirker ytelsen til systemet. Antallet kanaler kan bli justert på grunn av plassmangel når designet skal implementeres i sonaren, og det endelige antall kanaler bestemmes etter testing.

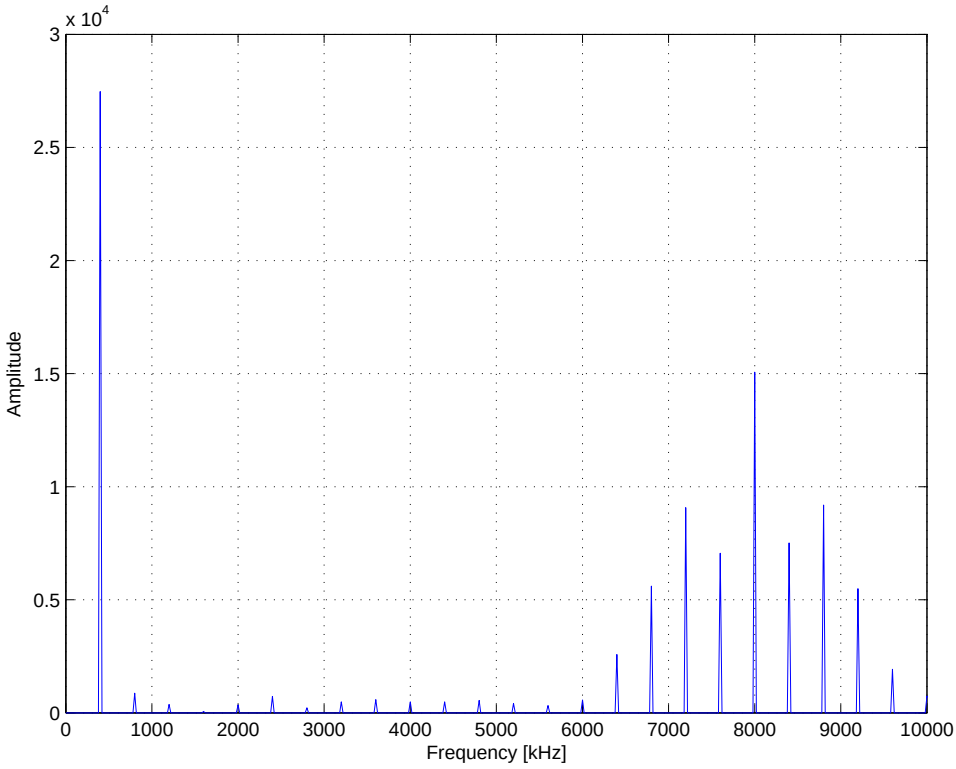
S.5 Prototype
Vi velger å lage en prototype hvor vi ikke tar hensyn til fysiske størrelser, og heller fokuserer på funksjonaliteten. Prototypen vil bestå av et utviklerkort med FPGA, en transducer med 32 elementer og et egendesignet kretskort med strømforsyninger, ladekrets, kondensatorbanker, 16 utgangstrinn og nødvendige tilkoblinger mot FPGA og transducer.

S.6 Forvrengningskrav
For å oppfylle krav til forvrengning bruker vi 8MHz PWM med 15 trinn, hvilket gir en tellerfrekvens på 120MHz. Vi har gjort simuleringer i Matlab og lest av amplitudeverdier på de overharmoniske for grunnfrekvensen på 400kHz fra figur 11. De høye amplitudeverdiene rundt 8MHz i figuren skyldes PWM-frekvensen og dens harmoniske, og tas ikke med i beregningene. Simuleringene er gjort med 120MHz tellerfrekvens, 8MHz PWM-frekvens og 400kHz som grunnfrekvens. Matlab-koden for å generere figuren finnes i vedlegg F.1. Vi ønsker at styresystemet skal kunne sende pulser med opp til 2ms lengde, altså 16000 samples. Vi trenger å kunne sende to ulike pulser med ulike delay- og amplitudeprofiler raskt etter hverandre. Det gir et totalt lagringsbehov på 32000 samples. THD er beregnet fra formel 7.

$$
\begin{aligned}
THD &= \frac{\sqrt{V_{800}^2 + V_{1200}^2 + V_{1600}^2 + V_{2000}^2 + V_{2400}^2 + V_{2800}^2 + V_{3200}^2 + V_{3600}^2}}{V_{400}} \\
THD &= \frac{\sqrt{869^2 + 373^2 + 63^2 + 349^2 + 728^2 + 223^2 + 488^2 + 583^2}}{2.747 \cdot 10^4} \\
THD &= 0.0537 = -25.4d\text{B}
\end{aligned}
$$

S.7 Båndbredde
Båndbredde på 100kHz - 1.5MHz oppnås med PWM-oppløsning på 8MHz.

S.8 Oppløsning på retningsstyring
For å kunne nå kravet om $\theta = 0.25°$ styringsvinkel må vi vite hvor lang tidsforsinkelse



Figur 11: Amplituder for PWM-signal

dette tilsvarer. Tidsforsinkelsen vil gi krav til hastigheten for FPGA-en som skal styre senderen. Ut i fra formel **??** kan vi beregne størrelsen på trinnene som er nødvendig for å få 0.25° vinkeloppløsning.

$$\begin{aligned} \tau &= \frac{1}{400k\text{Hz}} \left(cos\left(90\right) - cos\left(90.25\right)\right) \\ \tau &= 10.9ns \end{aligned}$$

Dette gir et krav om minimum $\frac{1}{10.9ns} = 91.7$MHz klokkefrekvens for FPGA-en. Vi har valgt å bruke 120MHz, fordi det går opp med 8MHz PWM-frekvens og 15 trinn.

I tillegg til retningsstyringen velger vi å legge til mulighet for defokusering av arrayet for å kunne øke deteksjonsområdet for de overlappende pulsene.

S.9 Sidelobenivå

For å få et sidelobenivå som er bedre enn -25dB har vi forsøkt med forskjellige vindusfunksjoner. Det er mulig å oppfylle kravet, men vinduet påvirker bredden på hovedloben og vi får mindre utsendt effekt som følger av at amplitudenivået på de ytterste elementene reduseres. Vi lager mulighet for å aktivere / deaktivere vindusfunksjonen gjennom brukergrensesnittet.

S.10 Kondensatorbank

Vi planlegger å gjenbruke designet i dagens løsning og må gjøre tilpassinger for å få nok effekt ut på senderen. De 32 senderelementene har hver en impedans på 100Ω og kondensatorbanken må dimensjoneres slik at alle elementene kan sende i parallell samtidig.

Impedansen for parallellkoblingen blir:

$$Z = \frac{100\Omega}{32} = 3.125\Omega$$

Elementene får 100V, dette gir en peak-strøm på:

$$\begin{aligned} I &= \frac{U}{Z} \\ I &= \frac{100\text{V}}{3.125\Omega} \\ I &= 32\text{A} \end{aligned}$$

Dette gir en gjennomsnittsstrøm på:

$$\begin{aligned} I_{avg} &= I \int_0^{\frac{1}{2}} sin(2\pi t), \mathrm{d}t \\ I_{avg} &= 32\text{A} \left[\frac{1}{2\pi} cos(2\pi t) \right]_0^{\frac{1}{2}} \\ I_{avg} &= \frac{32\text{A}}{\pi} \end{aligned}$$

Pulslengden er 1ms og dette gjør at kondensatorbanken må ha en kapasitet på:

$$\begin{aligned} Q &= I_{avg} \cdot t_{pulse} \\ Q &= \frac{32\text{A}}{pi} \cdot 1m\text{s} \\ Q &\approx 10.2m\text{C} \end{aligned}$$

Vi tillater et spenningsfall på 25V under sending noe som resulterer i følgende kapasitans:

$$\begin{aligned} C &= \frac{Q}{U_{drop}} \\ C &= \frac{10.2m\text{C}}{25\text{V}} \\ C &\approx 0.4m\text{F} \end{aligned}$$

Denne kapasitansen trenger vi i hver av de to kondensatorbankene for $\pm$ 100V.

S.11 Ladekrets

Ladekretsen må kunne lade opp kondensatorene fra 75V til 100V i tillegg må vi ta hensyn til blødeutladingen i kretsen. I verste fall vil pingraten være 1Hz, noe som gir blødeutladning i 1s. Vi ønsker en viss blødeutladning i kretsen for å forhindre at det står spennning på kondenastorbanken lenge etter at tilførselsspenningen er frakoblet. Vi velger en tidskonstant på 6 sekund. For å få til dette må vi ha en motstand over kondensatorbanken:

$$\begin{aligned} R &= \frac{\tau}{C} \\ R &= \frac{6\text{s}}{0.4m\text{F}} \\ R &= 15k\Omega \end{aligned}$$

Det totale spennngsfallet blir:

$$\begin{aligned}
\Delta U &= U_{full} - U_{bleed} \\
\Delta U &= 100\text{V} - 75\text{V} \cdot e^{-\left(\frac{t_{ping_{min}}}{R \cdot C}\right)} \\
\Delta U &= 100\text{V} - 75\text{V} \cdot e^{-\left(\frac{1\text{s}}{15k\Omega \cdot 0.4m\text{F}}\right)} \\
\Delta U &= 36.5\text{V}
\end{aligned}$$

Dagens løsning lader opp kondensatorene på 15ms, noe som er godt nok, men når kapasitansen skal økes vil ladetiden bli lengre med samme ladekrets. Vi ønsker å optimalisere dagens løsning for å forsøke å redusere ladetiden med den utvidede kapasitansen. Endelig løsning velges etter testing på prototype.

S.12 Utgangstrinn
Utgangstrinnet som brukes i dagens løsning gjenbrukes. En økning på 4 ganger effekten oppnås ved at det nå blir 16 utgangstrinn versus 4 på dagens løsning.

3.2.3 Spesifikasjoner for mottaksanalyse

M.1 Analysemetode
Norbit har allerede et Python-skript for å hente ut rådata fra sonaren. Vi vil tilpasse dette til vårt formål og bruke Matlab for å analysere dataene.

M.2 Sanntidsanalyse
Dette kravet vil vi se på dersom det blir tid til overs. Vi ser for oss at dette vil ta mye tid da vi må sette oss inn i eksisterende programkode for FPGA.

M.3 Objekt-tracking
Dette kravet vil vi se på dersom det blir tid til overs.

4 Komponentberegninger

Noen av komponentene for denne prototypen er gjenbruk av designet i eksisterende løsning for sonaren. For ordens skyld har vi kontrollregnet verdiene for disse komponentene. Nye komponenter er beregnet fra formler i de respektive komponenters datablader.

4.1 Utgangstrinn

Sendereffekt
Sendereffekten bestemmes av påtrykt spenning og impedansen til parallellkoblingen av transducer-elementene. Impedansen ble beregnet til 3.125Ω i kapittel 3.2.2, punkt S10.

$$\begin{aligned} P_{out} &= \frac{U^2}{Z} \\ P_{out} &= \frac{\left(\frac{100\text{V}}{\sqrt{2}}\right)^2}{3.125\Omega} \\ P_{out} &= 1600\text{W} \end{aligned}$$

Transistorer og drivere
Vi har valgt å bruke samme komponenter for å styre ±100V som i eksisterende løsning. I dagens løsning er alle transducer-elementene koblet i parallell og utgangstrinnet består av fire parallellkoblede MOSFET-trinn. Løsningen består av to dobble MOSFET-drivere som hver driver to MOSFET-par. Utgangene fra MOSFET-parene er koblet i parallell til utgangsklemmene. For å kunne håndtere den økede effekten, samt å ha mulighet for retningsstyring av utgangssignalet, har vi designet en løsning med 16 utgangstrinn som hver driver to parallellkoblede transducer-elementer. Vi har brukt komponenter som foreslått i databladene for Supertex MD1210 [13] (driver) og Supertex TC6320TG [12] (transistor).

Utgangsbeskyttelse
Det har vært problemer med at transistorene på utgangstrinnet har røket. I utgangstransistorene er det en ikke-ideell body-diode som kan ryke om reversspenningen blir for høy. Vi har lagt inn dioder på utgangen som et forsøk på å utbedre problemet. Diodene må tåle 200V i reversspening, og en strøm som er bestemt av utgangsspenningen og to transducer-element i parallell:

$$\begin{aligned} I_{out(TC6320TG)} &= \frac{V_{out}}{Z_{2xTransducer}} \\ I_{out(TC6320TG)} &= \frac{100\text{V}}{50\Omega} \\ I_{out(TC6320TG)} &= 2\text{A} \end{aligned}$$

Vi har valgt en diode fra Diodes Incorporated [16], som tåler 400V i reversspenning, 2A strøm og har en reverse recovery time på 35ns.

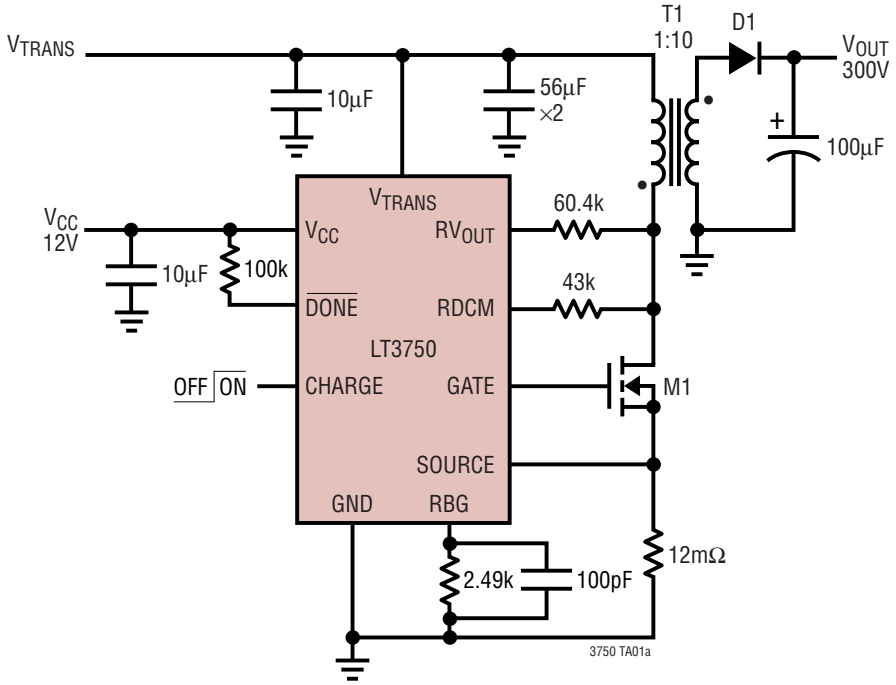
4.2 Kondensatorbank

Fra utregningen i kapittel 3.2.2 punkt S10, vet vi at det trengs 0.4mF kapasitans i kondensatorbanken. Vi velger å bruke $20 \cdot 4.7\mu F$ som keramiske kondensatorer og én 330µF som elektrolyttkondensator. Elektrolytt tar mindre plass, men kan ikke utsettes for trykk. Dette går bra da full sendestyrke kun trengs når sonaren er nær vannoverflaten. Denne konfigurasjonen gir en kapasitans på 424µF for hver av de to kondensatorbankene.

4.3 Ladekrets

Den eksisterende ladekretsen består av en Linear Technologies LT3751 med tilhørende komponenter, en dobbel selvprodusert transformator som genererer ±100V og en kondensatorbank. Vi ønsket å øke effekten for å kunne lade opp den utvidede kondensatorbanken raskere enn om vi hadde brukt eksisterende ladekrets, men etter beregninger og tester på lab har vi kommet frem til at det ikke lar seg gjøre.

Vi har valgt å dele opp ladekretsen i to deler, hvor den ene lager +100V og den andre -100V. Dette gjør at vi kan velge transformatorer som er hyllevare og dette er en stor fordel produksjonsmessig. Vi har valgt 2 stk. LT3750 [14] som er en enklere variant av LT3751. Ladekretsen er en flyback converter og for å fungere trenger den noen eksterne komponenter; inngangskondensator, ekstern svitsjetransistor, transformator, glattekondensatorer og noen motstander, se figur 12. Under følger utregninger av verdiene for de nødvendige komponentene.



Figur 12: Typisk bruksområde for LT3750

Valg av transformator
For at ladekretsen skal rekke å regulere strømmen i transformatoren må følgende kriterium overholdes:

$$\begin{aligned} L_{PRI} &\geq \frac{V_{out} \cdot 1\mu s}{N \cdot I_{pk}} \\ L_{PRI} &\geq \frac{100\text{V} \cdot 1\mu s}{10 \cdot 5\text{A}} \\ L_{PRI} &\geq 2\mu\text{H} \end{aligned}$$

Vi valgte en transformator fra Coilcraft [18] på 10µH, som tåler 5A og har et viklingsforhold på N=10.

Oppladingstid
Fra kapittel 3.2.2 punkt S11, har vi at maksimal utladning av kondensatorbanken under drift er 36.5V. Under følger en beregning av oppladningstiden for kondensatorbanken med kapasitans som i dagens løsning. Formel fra databladet til LT3751 [17].

$$\begin{aligned} t_{ch} &= t_{charge(high)} - t_{charge(low)} \\ t_{ch} &= \frac{\left(2 \cdot N \cdot V_{in} + V_{out(max)}\right) \cdot C \cdot V_{out(max)}}{\text{Efficiency} \cdot V_{in} \cdot I_{pk}} - \frac{\left(2 \cdot N \cdot V_{in} + V_{out(min)}\right) \cdot C \cdot V_{out(min)}}{\text{Efficiency} \cdot V_{in} \cdot I_{pk}} \\ t_{ch} &= \frac{580\text{V} \cdot 22\mu\text{F} \cdot 100\text{V}}{0.8 \cdot 24\text{V} \cdot 5\text{A}} - \frac{(480 + (100 - 36.5))\,\text{V} \cdot 22\mu\text{F} \cdot (100 - 36.5)\,\text{V}}{0.8 \cdot 24\text{V} \cdot 5\text{A}} \\ t_{ch} &= 5.4m\text{s} \end{aligned}$$

Med en utvidet kapasitans til 424µF får vi følgende oppladingstid

$$\begin{aligned} t_{ch} &= \frac{580\text{V} \cdot 424\mu\text{F} \cdot 100\text{V}}{0.8 \cdot 24\text{V} \cdot 5\text{A}} - \frac{(480 + (100 - 36.5))\,\text{V} \cdot 424\mu\text{F} \cdot (100 - 36.5)\,\text{V}}{0.8 \cdot 24\text{V} \cdot 5\text{A}} \\ t_{ch} &= 103m\text{s} \end{aligned}$$

Peak-strøm
Peak-strømmen bestemmes av R_{sense} og vi har valgt maksimal strøm som transformatoren er dimensjonert for, det vil si I_{pk}=5A

$$\begin{aligned} R_{sense} &= \frac{U_{ref}}{I_{pk}} \\ R_{sense} &= \frac{78m\text{V}}{5\text{A}} \\ R_{sense} &= 15.6m\Omega \end{aligned}$$

Vi har valgt nærmeste tilgjengelige motstand, som er 15mΩ. Dette vil gjøre at strømmen øker noe. Vi må også passe på at motstanden vi velger tåler effekten som skal gå gjennom den.

$$\begin{aligned} P_{sense} &= \frac{I_{pk}^2 \cdot R_{sense}}{3} \cdot \left(\frac{V_{out(pk)}}{V_{out(pk)} + N \cdot V_{in}} \right) \\ P_{sense} &= \frac{(5.2\text{A})^2 \cdot 15m\Omega}{3} \cdot \left(\frac{100\text{V}}{100\text{V} + 10 \cdot 24\text{V}} \right) \\ P_{sense} &= 40m\text{W} \end{aligned}$$

Valg av diode

Dioden må ha tåle en peak-strøm i lederetning som er større enn maksimal peak-strøm i transformatoren:

$$\begin{aligned} I_{pk(diode)} &> \frac{I_{pk}}{N} \\ I_{pk(diode)} &> \frac{5.2\text{A}}{10} \\ I_{pk(diode)} &> 0.52\text{A} \end{aligned}$$

Gjennomsnittsstrømmen i dioden vil være størst når utgangskondensatorene er nesten fullt oppladet. Dioden må minst tåle en kontinuerlig strøm på:

$$\begin{aligned} I_{avg(diode)} &= \frac{I_{pk} \cdot V_{in}}{2\left(V_{out} + N \cdot V_{in}\right)} \\ I_{avg(diode)} &= \frac{5.2\text{A} \cdot 24\text{V}}{2\left(100\text{V} + 10 \cdot 24\text{V}\right)} \\ I_{avg(diode)} &= 0.18\text{A} \end{aligned}$$

Dioden må tåle en reversspenning på:

$$\begin{aligned} V_{pk(reverse)} &> V_{out} + N \cdot V_{in} \\ V_{pk(reverse)} &> 100\text{V} + 10\text{V} \cdot 24\text{V} \\ V_{pk(reverse)} &> 340\text{V} \end{aligned}$$

Vi har sett på dioden DFLU1400 [19] , som brukes i eksisterende løsning, og den oppfyller kravene.

Valg av transistor

Transistoren må tåle en gjennomsnittsstøm som er større enn:

$$\begin{aligned} I_{avg(NMOS)} &= \frac{I_{pk} \cdot V_{out}}{2\left(V_{out} + N \cdot V_{in}\right)} \\ I_{avg(NMOS)} &= \frac{5.2\text{A} \cdot 100\text{V}}{2\left(100\text{V} + 10 \cdot 24\text{V}\right)} \\ I_{avg(NMOS)} &= 0.76\text{A} \end{aligned}$$

Vi valgte samme transistor som i eksisterende løsning, Vishay Si4104DY [20], som tåler en drain-strøm på 4.6A og 100V over drain-source.

Utgangsspenning
Utgangsspenningen blir bestemt av tilbakekoblingsnettverket, R_{BG} og RV_{out}. For å unngå overlading anbefaler databladet maks verdi på R_{BG} til 2.5kΩ. Vi valgte R_{BG}=2.2kΩ og beregner RV_{out}:

$$\begin{aligned} RV_{out} &= R_{BG}\left(\frac{V_{out}+V_{diode}}{1.24\cdot N}\right) \\ RV_{out} &= 2.2k\Omega\left(\frac{100\text{V}+1\text{V}}{1.24\cdot 10}\right) \\ RV_{out} &= 18k\Omega \end{aligned}$$

For å unngå at den interne spenningskomparatoren tripper ved høye strømtransienter valgte vi å sette en kondensator på 100pF i parallell med R_{BG}.

4.4 10V Switch-mode strømforsyning

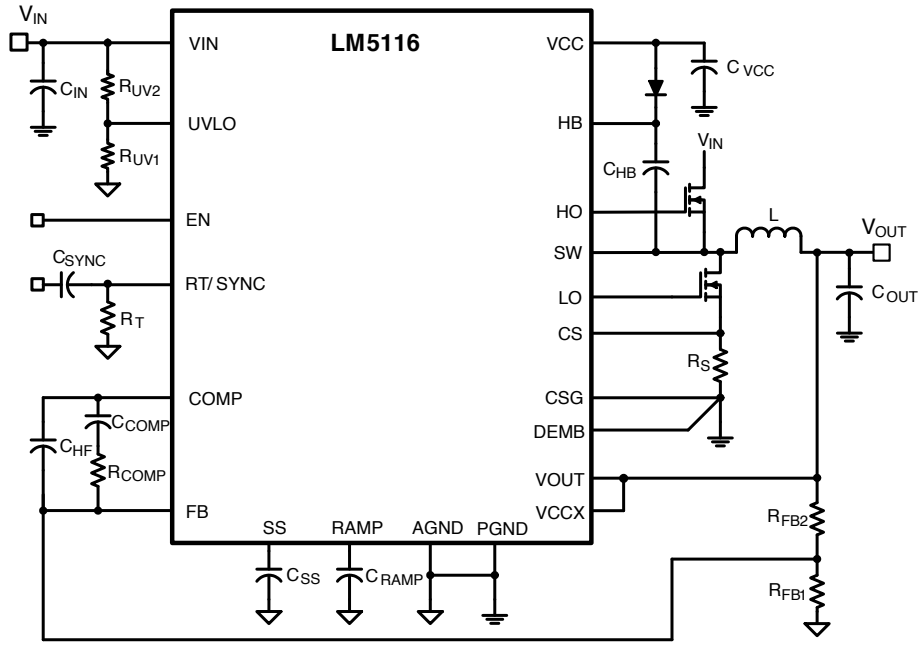
For å generere 10V som skal forsyne ladekretsen og utgangstrinnet har vi valgt en switch-mode-chip fra Texas Instruments. Strømforsyningen er av typen buck-regulator og krever noen eksterne komponenter, se figur 13. Vi har gjort beregninger på komponentene for å få best mulig tilpassing til lasten. Formler for beregninger er tatt fra databladet til LM5116 [11].

Strømtrekk
For å beregne de eksterne komponentene må vi finne ut hvor mye strøm som vil gå i kretsen ved maksimum last.

Mosfet-par i utgangstrinn Supertex TC6320 [12]:
Mosfet-paret har inngangskapasitanser på:
N-kanal: C_{iss}=110pF og P-kanal: C_{iss}=200pF.
Med inngangskapasitansene kan vi regne ut hvor mye ladning og derav hvor stor strøm som går:

$$\begin{aligned} Q &= C\cdot V \\ Q &= 310p\text{F}\cdot 10\text{V} \\ Q &= 3.1n\text{C} \end{aligned}$$

$$\begin{aligned} I_{Mosfet-par} &= Q\cdot f_{sw} \\ I_{Mosfet-par} &= 3.1n\text{C}\cdot 8M\text{Hz} \\ I_{Mosfet-par} &= 24.8m\text{A} \end{aligned}$$



Figur 13: Typisk bruksområde for LM5116

Vi har 16 Mosfet-par, noe som gir en strøm på:

$$\begin{aligned} I_{Mosfet(TOT)} &= I_{Mosfet-par} \cdot 16 \\ I_{Mosfet(TOT)} &= 397m\text{A} \end{aligned}$$

Drivere for Mosfet-par i utgangstrinn Supertex MD1210 [13]:
I_{DD1}=0.88mA ved 5MHz og I_{DD2}=6.6mA ved 5MHz.
Vi har 16 drivere noe som gir en strøm:

$$\begin{aligned} I_{Drivere} &= 16 \cdot \frac{8M\text{Hz}}{5M\text{Hz}} \cdot (0.88m\text{A} + 6.6m\text{A}) \\ I_{Drivere} &= 191m\text{A} \end{aligned}$$

Ladekrets LT3750:[14]
Kretsen trekker 1.6mA i standby, i tillegg kommer switchestrøm. Skal lade opp transistor Si4104dy som har en inngangskapasitans C_{iss}=446pF.

$$\begin{aligned} Q &= C \cdot V \\ Q &= 446p\text{F} \cdot 24\text{V} \\ Q &= 10.7n\text{C} \end{aligned}$$

$$\begin{aligned} I &= Q \cdot f_{sw} \\ I &= 10.7n\text{C} \cdot 200k\text{Hz} \\ I &= 2.14m\text{A} \end{aligned}$$

Vi skal bruke to LT3750 og dette gir en totalstrøm til ladekretsen:

$$\begin{aligned} I_{Ladekrets} &= 2 \cdot (1.6mA + 2.14mA) \\ I_{Ladekrets} &= 6.8mA \end{aligned}$$

Summen av stømforbruket i Mosfet-parene, driverne og ladekretsene gir maks strøm fra 10Vs forsyningen:

$$\begin{aligned} I_{max} &= I_{Mosfet(TOT)} + I_{Drivere} + I_{Ladekrets} \\ I_{max} &= 397mA + 191mA + 6.8mA \\ I_{max} &= 595mA \end{aligned}$$

Vi ønsker en margin og dimensjonerer for et strømtrekk I_{max}=1A.

Under- og overspenningsbeskyttelse

Vi setter den øvre grensen for innganssspenning for kretsen til 30V og nedre grense settes til 18V. Dette gjøres med spenningsdeleren på UVLO-pinnen. Dersom inngangsspenningen er utenfor dette området vil kretsen slå av 10Vs utgangen.

$$\begin{aligned} R_{UV2} &> 500 \cdot V_{in(max)} \\ R_{UV2} &> 500 \cdot 30\text{V} \\ R_{UV2} &> 15k\Omega \end{aligned}$$

Velger R_{UV2}=150kΩ for å begrense strømmen.

$$\begin{aligned} R_{UV1} &= 1.215 \left(\frac{R_{UV2}}{V_{in(min)} + (5\mu\text{A} \cdot R_{UV2}) - 1.215} \right) \\ R_{UV1} &= 1.215 \left(\frac{150k\Omega}{18V + (5\mu\text{A} \cdot 150k\Omega) - 1.215} \right) \\ R_{UV1} &= 10.4k\Omega \end{aligned}$$

Velger 10kΩ i serie med 470Ω.

Svitsjefrekvens

Vi velger å bruke samme svitsjefrekvens som i de andre strømforsyningene i sonaren, det vil si f_{sw}=630kHz. Frekvensen settes med timing-motstanden R_T.

$$\begin{aligned} R_T &= \frac{1/f_sw - 450n\text{s}}{284p\text{F}} \\ R_T &= 4k\Omega \end{aligned}$$

Velger å bruke en 1.8kΩ i serie med en 2.2kΩ.

Spole

Verdien på spolen blir bestemt av laststrømmen, inn- og utgangsspenningen, rippelstrømmen og switche-frekvensen. Vi har valgt å tillate 40% rippelstrøm ved maks belastning.

$$\begin{aligned} L &= \frac{V_{out}}{I_{max} \cdot 0.4 \cdot f_{sw}} \left(1 - \frac{V_{out}}{V_{in(max)}} \right) \\ L &= \frac{10\text{V}}{1\text{A} \cdot 0.4 \cdot 630k\text{Hz}} \left(1 - \frac{10}{30} \right) \\ L &= 26.4\mu\text{H} \end{aligned}$$

Velger en spole fra Coilcraft med nærmeste verdi som er 27µH [15].

Strømbegrenser
Vi har valgt å dimensjonere for 1A maksstrøm. Maksimumverdien settes med R_s:

$$\begin{aligned} R_s &\leq \frac{V_{csth}}{I_{max} + \frac{V_{out}}{2Lf_{sw}}\left(1 + \frac{V_{out}}{V_{in(min)}}\right)} \\ R_s &\leq \frac{0.11\text{V}}{1\text{A} + \frac{10\text{V}}{2 \cdot 27\mu\text{H} \cdot 630k\text{Hz}}\left(1 + \frac{10}{18\text{V}}\right)} \\ R_s &\leq 75.5m\Omega \end{aligned}$$

Velger nærmeste mindre tilgjengelige verdi som er 68mΩ.

Utgangskondensator
For å få minst mulig rippelspenning på utgangen velger kondensatorer med lav ekvivalent seriemotstand (ESR), det vil si 1.14mΩ. Vi har valgt å tillate 5mV rippelspenning på utgangen.

$$\begin{aligned} C_{out} &= \frac{1}{\sqrt{\left(\frac{V_{ripple}}{I_{max} \cdot 0.4}\right)^2 - (ESR)^2} \cdot 8f_{sw}} \\ C_{out} &= \frac{1}{\sqrt{\left(\frac{5m\text{V}}{1\text{A} \cdot 0.4}\right)^2 - (1.14m\Omega)^2} \cdot 8 \cdot 630k\text{Hz}} \\ C_{out} &= 16\mu\text{F} \end{aligned}$$

Velger 20µF, hvor vi setter to 10µF i parallell for å få enda mindre ESR. Dette vil redusere rippelspenningen ytterligere.

Inngangskondensator
For å begrense svitsjestøy på inngangsspenningen trengs det kondensatorer på inngangen. Verdien bestemmes av maksstrømmen, svitsjefrekvensen og hvor mye rippel vi tillater på inngangsspenningen. Vi har valgt en rippelspenning på 1V.

$$\begin{aligned} C_{in} &= \frac{I_{max}}{4 \cdot f_{sw} \cdot \Delta V_{in}} \\ C_{in} &= \frac{1\text{A}}{4 \cdot 630k\text{Hz} \cdot (1\text{V})} \\ C_{in} &= 40\mu\text{F} \end{aligned}$$

Velger å sette to 22µF i parallell.

Ramp-kondensator og ramp-motstand
Ramp-kondensatoren og ramp-motstanden må beregnes fra formler i databladet for at regulering av utgangsspenningen skal fungere som tiltenkt.

$$\begin{aligned} I_{OS} &= \frac{V_{out}}{2} \cdot 10\mu\text{A} \\ I_{OS} &= \frac{10\text{V}}{3} \cdot 10\mu\text{A} \\ I_{OS} &= 33.3\mu\text{A} \end{aligned}$$

$$
\begin{aligned}
T &= \left(\frac{V_{csth}}{R_s} - I_{out}\right) \cdot \frac{L}{V_{out}} \\
T &= \left(\frac{0.11\text{V}}{68m\Omega} - 1A\right) \cdot \frac{27\mu\text{H}}{10\text{V}} \\
T &= 1.67\mu\text{s}
\end{aligned}
$$

$$
\begin{aligned}
C_{RAMP} &= \frac{I_{OS} \cdot L}{V_{out} \cdot A \cdot R_s} \\
C_{RAMP} &= \frac{33.3\mu\text{A} \cdot 27\mu\text{H}}{10\text{V} \cdot 10 \cdot 68m\Omega} \\
C_{RAMP} &= 132p\text{F}
\end{aligned}
$$

Velger nærmeste tilgjengelige verdi som er 150pF.

$$
\begin{aligned}
V_{RAMP} &= \frac{V_{out}}{V_{in}} \cdot \frac{\left(\left(V_{in} - V_{out}\right) g_m + I_{OS}\right) \cdot T}{C_{RAMP}} \\
V_{RAMP} &= \frac{10\text{V}}{24\text{V}} \cdot \frac{\left(\left(24\text{V} - 10\text{V}\right) 5\mu\text{A} + 33.3\mu\text{A}\right) \cdot 1.67\mu\text{s}}{150p\text{F}} \\
V_{RAMP} &= 0.48\text{V} \\
R_{RAMP} &= \frac{V_{cc} - V_{RAMP}}{I_{OS} - 25\mu\text{A}} \\
R_{RAMP} &= \frac{10\text{V} - 0.48\text{V}}{33.3\mu\text{A} - 25\mu\text{A}} \\
R_{RAMP} &= 1.02M\Omega
\end{aligned}
$$

Velger nærmeste tilgjengelige verdi som er 1MΩ.

Bootstrap-kondensator
Velges til 1µF som foreslått i databladet.

Softstart-kondensator
Velger oppstartstiden til 1.2ms som foreslått i databladet.

$$
\begin{aligned}
C_{ss} &= \frac{t_{ss} \cdot 10\mu\text{A}}{1.215\text{V}} \\
C_{ss} &= \frac{1.2m\text{s} \cdot 19\mu\text{A}}{1.215\text{V}} \\
C_{ss} &= 10n\text{F}
\end{aligned}
$$

Spenningsdeler for tilbakekoblingsnettverk
Motstandene R_{FB1} og R_{FB2} i tilbakekoblingen bestemmer utgangsspenningen. Velger R_{FB1}=10kΩ.

$$
\begin{aligned}
R_{FB2} &= \left(\frac{V_{out}}{1.215\text{V}} - 1\right) \cdot R_{FB1} \\
R_{FB2} &= \left(\frac{10\text{V}}{1.215\text{V}} - 1\right) \cdot 10k\Omega \\
R_{FB2} &= 72.3k\Omega
\end{aligned}
$$

Vi velger å seriekoble motstander på 68kΩ og 3.3kΩ for R_{FB2}.

4.5 Inngangsbeskyttelse

For å beskytte mot feil på inngangsspenningen har vi valgt å benytte samme krets som beskyttelsen i eksisterende løsning. Kretsen består av en Zener-diode som står parallelt over inngangen, og sikrer mot for høy inngangsspenning. Videre er det en diode i serie med +24V som sikrer mot feil polaritet på inngangsspenningen. Kretsen har også et lavpassfilter for å filtrere bort støy som kan komme fra Switch-mode-regulatoren og en kondensatorbank som gir et buffer mot spennigsfall ved høye strømtransienter. Skjema for inngangsbeskyttelsen er i vedlegg A

5 Skjemategning og utlegg av mønsterkort

Til prototypen har vi laget et eget kretskort som inneholder strømforsyninger, ladekretser, kondensatorbanker, utgangstrinn og nødvendige interface. For å lage kretskortet har vi tegnet skjema og laget et utlegg for mønsterkortet.

5.1 Skjemategning

Skjemaene er tegnet i Zuken CR5000 som er standardprogram for tegning av skjema hos oppdragsgiver.

Vi valgte å dele opp skjemaet i fem sider. Side 1 inneholder 10V Switch-mode-regulatoren. Side 2 inneholder ladekretsene for ± 100V og kondensatorbankene. Side 3 og 4 inneholder utgangstrinnene. Side 5 inneholder inngangsbeskyttelsen og grensesnitt mot utviklerkort for FPGA.

For å forenkle feilsøking og for å kunne kjøre igang bare deler av systemet, har vi lagt inn 0Ωs-motstander på strategiske steder i kretsene. For å forenkle lesing av skjemaet har vi satt navn ved sentrale komponenter og vi har unngått koblingspunkt hvor fire ledere møtes.

De fleste komponentene var allerede i oppdragsgivers database for komponenter. Arbeidet med å legge inn nye komponenter i databasen ble utført av oppdragsgiver.

Skjemaene er i vedlegg A

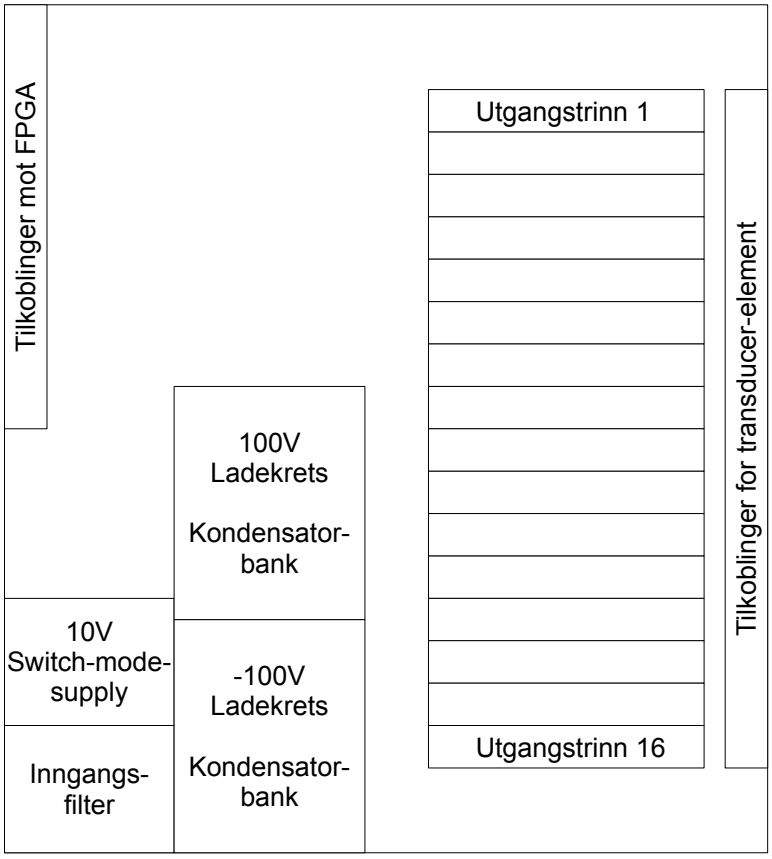
5.2 Utlegg

Utlegget er tegnet med Zuken CR5000. Vi valgte å bruke et firelags mønsterkort med størrelse 18cm x 20cm. Størrelsen ble valgt slik at vi hadde god plass for eventuelle modifikasjoner og for å enkle feilsøking og testing. Når løsningen skal implementeres i sonaren må størrelsen reduseres.

Alle komponenter er plassert på kortets overside og på lag 2 er det jordplan. Lag 3 brukes stedvis til 10Vs-plan og ruting for ±100V. Lag 4 brukes til ruting av signaler og noe ±100V. Der hvor det trengs forbindelse mellom lagene brukes gjennompletterte viahull. For å ha minst mulig induktans der hvor det går store strømmer har vi brukt hele plan eller brede baner. Avkoblingskondensatorer er plassert nærmest mulig komponentene og vi har brukt overflatemonterte komponenter for å redusere banelengder. Bredden på signalbanene er 0.2mm og bredden på forsyningsbaner varierer fra 0.5mm til 2mm, tilpasset hvor stor strøm som går For komponentene som hadde anbefalinger for utlegg i sine datablader har vi fulgt disse.

Mønsterkortet er av materiale FR4, er totalt 1.6mm tykt og har baner av gull som er 35µm tykke. Kortet er bestilt hos GS Nordic, en samarbeidsbedrift for oppdragsgiver, og blir produsert i Kina. Vi har spesifisert at mønsterkortet skal testes elektrisk på fabrikk. Komponenter lodder vi på for hånd.

Vi har planlagt utlegget slik at støyende komponenter er lengst mulig unna støyfølsomme baner. For å forenkle testing har vi lagt inn testpunkter for måleprober på flere steder. Blokkskjema for utlegget vises i figur 14.

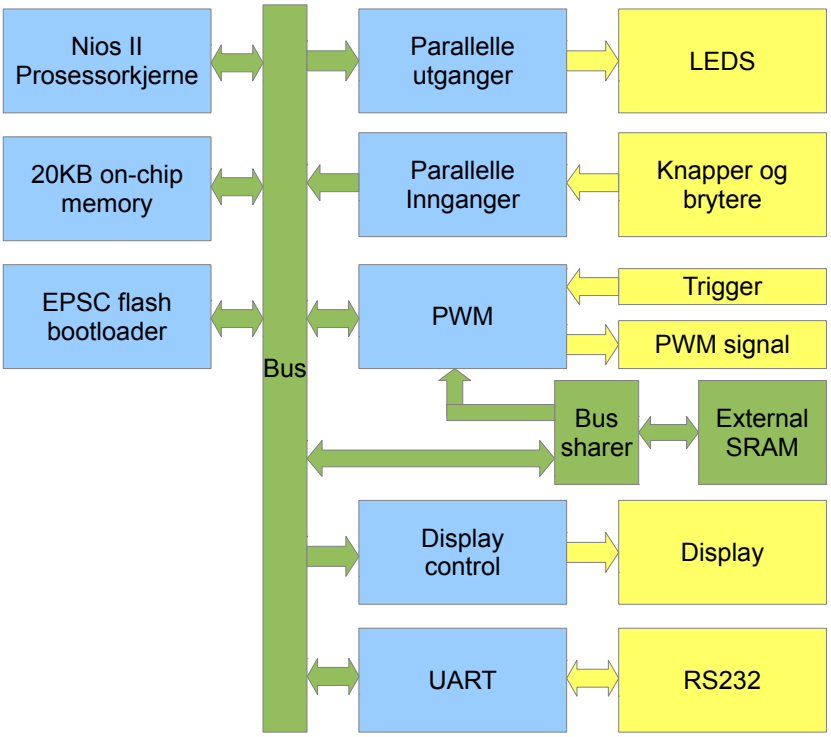


Figur 14: Blokkskjema for utlegget

6 Styresystem

6.1 Systemarkitektur

Systemet er designet i henhold til spesifikasjonene vi har beskrevet i kapittel 3.2.2. For å gi fleksibilitet med tanke på pulstyper og delays, som skal kunne endres ved hjelp av brukergrensesnittet, så er det ønskelig å kunne generere de i programkode fremfor RTL-kode. Det betyr at vi vil integrere en mikrokontroller i designet vårt. For å få til det har vi valgt å bruke systemintegrasjonsverktøyet Qsys som er en del av Altera Quartus. Vår egen RTL-kode har vi laget som Qsys-moduler, som vi ved hjelp av Qsys har integrert i et system bestående av en Nios II kjerne, on-chip RAM for mikrokontrolleren, ekstern SRAM for pulsminne, og I/O for knapper og display. Triggerinngangen har vi valgt å gå utenom Qsys-systemet direkte til PWM-modulen for å få kortest mulig reaksjonstid. Den overordnede strukturen er vist i figur 15. For å lage program til FPGA-en har vi benyttet oss av Altera Quartus II version 12.1. RTL-kode er skrevet i VHDL og software er skrevet i C.



Figur 15: Overordnet blokkskjema for styresystemet som er integrert i FPGA-en

Vi har selv designet PWM-modulen og Display control modulen. De andre modulene er ferdigmoduler levert av Altera. Nios II er en 32-bits prosessorkjerne som kan konfigureres etter ønske. Vi har valgt å bruke den minste varianten for å få plass i FPGA-en, og utstyrt den med flyttallsenhet for å øke hastigheten på beregning av pulsdata.

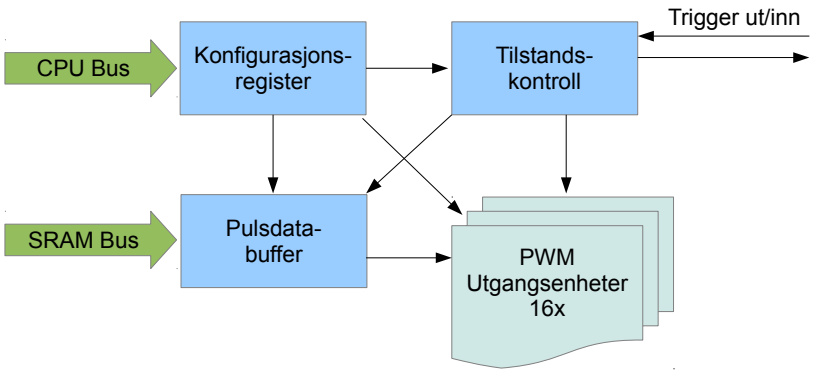
Displaykontrolleren har 4 registre som kan settes av prosessoren, og lager basert på det 4 sett av 7 signaler som styrer de individuelle LED-ene som 7-segments-displayet består av.

UART-en er brukt til å skrive ut informasjon fra programmet. Denne er ikke nødvendig for å bruke systemet slik vi har laget det, men nyttig hjelp under utvikling av C-koden.

EPSC flash bootloaderen har til oppgave å laste inn programmet fra flash-minnet på kortet når det blir skrudd på. Det gjør det mulig å bruke kortet frittstående uten å måtte ha en PC for å laste inn program etter hver omstart.

6.2 Utforming av PWM-modul

PWM-modulen er utformet som en Qsys-kompatibel systemkomponent. For at prosessorkjernen skal kunne konfigurere modulen så er den utstyrt for å være slave på prosessorens databus med et avalon-grensesnitt. Slik gjør den tilgjengelig konfigurasjonsregister slik at prosessoren kan definere delay og amplitude. I tillegg definerer prosessoren hvilken minne-adresse pulsdata skal hentes fra. Den er også utstyrt for å være master på minnebussen mot ekstern SRAM slik at den kan hente inn pulsdata fra spesifisert adresse uavhengig av prosessoren. Modulen eksporterer signaler for triggerutgang og -inngang, i tillegg til 2x16 pwm-signaler og 16 output enable-signaler.



Figur 16: Overordnet blokkskjema for PWM-modulen

Figur 16 viser den overordnede strukturen til PWM-modulen. Konfigurasjonsregistrene er tilgjengelige for skriving fra prosessoren. Layout for registrene er vist i tabell 1. Ved hjelp av disse registrene kan prosessoren sette opp to pulser, og velge om puls A, B eller begge skal sendes ut. Fordi de fleste verdiene er 16 bit så har vi for enkelhets skyld valgt å gjøre denne bus-en 16 bit bred. Startadressene er på 32bit og er derfor delt opp i to registre. Qsys håndterer automatisk tilkobling av 16-bits bus til prosessorens 32-bits bus.

Tilstandskontroll-blokken inneholder en tilstandsmaskin som holder kontroll på hvilken puls som er i ferd med å sendes ut. Basert på konfigurasjonsregistrene vet den hvilken eller hvilke pulser som skal sendes når den får trigger. Den gir da signal til pulsdatabuffer som begynner å hente inn data fra riktig adresse i SRAM.

Utgangsenhetene inneholder en teller som teller fra -7 til +7. For hver gang den starter på nytt så inkrementeres en posisjonsteller som indikerer hvor i bufferet data skal leses fra. Data fra riktig bufferposisjon sammenlignes kontinuerlig med verdien i den første telleren for å gi høyt eller lavt utgangssignal basert på om tellerverdien er høyere eller lavere enn verdien fra bufferet. For hver av de 16 kanalene er det 2 utganger, en for å drive den

Adresse	Størrelse	Funksjon
0x000XXXX	16bit * 16	Delay for puls A i 8.33ns trinn, for hver kanal (16 stk)
0x001XXXX	16bit * 16	Delay for puls B i 8.33ns trinn, for hver kanal (16 stk)
0x010XXXX	8bit * 16	Amplitude for hver kanal for puls A (16 stk)
0x011XXXX	8bit * 16	Amplitude for hver kanal for puls B (16 stk)
0x1000000	16bit	Varighet for pulsene (felles for A og B)
0x1010000	16bit	Adresse for puls A, 16 minst signifikante bit
0x1010001	16bit	Adresse for puls B, 16 mest signifikante bit
0x1010010	16bit	Adresse for puls A, 16 minst signifikante bit
0x1010011	16bit	Adresse for puls B, 16 mest signifikante bit
0x1010100	16bit	Modus for PWM-modulen. 0 for bare puls A, 1 for begge.

Tabell 1: Oversikt over adresseområde for PWM-modulen.

høy og en for å drive den lav, i tillegg til output enable som aktiverer mosfet-driveren. Vi har valgt å gi ut samme signal på de to førstnevnte. Output enable drives høy når posisjonstelleren ligger mellom 0 og lengden på pulsen. Tellerene initialiseres basert på delay-et som er satt opp for den aktuelle kanalen. Det vil si at posisjonstelleren kan starte på en negativ verdi, og i så fall blir den aktuelle utgangen deaktivert i en tidsperiode fra triggertidspunkt før den begynner å sende.

6.3 Utforming av C-kode

C-koden inneholder rutiner for å fylle puls-bufferet og kalkulere delay-verdier. Selv om PWM-modulen i utgangspunktet kan støtte hva som helst av pulstyper så har vi valgt å begrense C-koden til lineære chirp i henhold til formel 8. Konfigurasjonsgrensesnittet er brytere, knapper og display. For amplitudeverdiene har man mulighet til å velge mellom to fastsatte profiler. Den ene er full amplitude på alle kanaler, og den andre er et prekalkulert hanning-vindu.

For delay har man mulighet til å angi både retning og defokuseringsgrad. Basert på dette kalkuleres forsenkningsverdien til hver enkelt kanal. PWM-modulen har mulighet for uavhengig amplitude og delay for de to pulsene, men C-koden er laget slik at de får samme amplitude og speilvendt delay. For retningsstyring angir man hvor mange 8.33ns trinn pulsen skal forsinkes med per element. Defokuseringen kalkuleres basert på formel 6. De to forsinkelsesbidragene beregnes for hver kanal og legges sammen. Etterpå tilpasses verdiene slik at de går fra 0 og oppover.

Selve pulsen kalkuleres i henhold til formelen for chirp. For å spare tid så kalkuleres pulsen en gang og legges samtidig inn i begge bufrene, bare baklengs i det ene.

For å spare kalkuleringstid og programminne er det brukt forenklede funksjoner for kalkulering av sinus og kvadratrot. Disse funksjonene tar opp mindre programminne og har kortere kjøretid enn funksjonene som er inkludert i standardbibliotekene, på bekostning av nøyaktighet. Nøyaktigheten er likevel god nok for vårt formål.

Kvadratrotfunksjonen vi bruker er basert på Q_rsqrt som først ble kjent når kildekoden til PC-spillet Quake III: Arena ble gjort offentlig.[6] Fordi vi skal ha selve kvadratroten

og ikke den inverse, har vi lagt til en multiplikasjon av resultatet og utgangspunktet til slutt. Vi har også tatt i bruk det ekstra steget med Newtons metode som er kommentert ut i Quake III: Arena for å få høyere nøyaktighet. Hastigheten er ikke av stor betydning for kvadratrotfunksjonen, men vi har ikke plass i programminnet slik vi har gjort det til å inkludere standardbibliotekets kvadratrotfunksjon.

Sinuskalkulasjon må gjøres veldig mange ganger i løpet av kalkulasjon av en puls. For å forhindre at det blir tidkrevende å bytte puls så har vi valgt å bruke en lettere implementasjon også av sinusfunksjonen enn det som finnes i standardbiblioteket. Denne tilnærmingen benytter seg av en parabel som er tilnærmet fasongen av en sinus.[8]

6.4 Brukergrensesnitt

De konfigurerbare parametrene for pulsen er retning, defokuseringsgrad, øvre frekvens, nedre frekvens, vindusfunksjon av/på, og pulsens varighet. Grensesnittet for konfigurasjon av parametrene består at man med bryterene velger riktig parameter. Aktuell verdi vil da vises i displayet, og man kan endre den ved hjelp av trykknapper for å øke og senke verdien. Tabell 2 viser hvordan bryterene skal stilles for å aksessere de ulike parametrene. Tabell 3 viser funksjonen til de øvrige bryterene og knappene. Når man har stilt inn alle parametrene slik man ønsker trykker man på en tredje knapp som initierer kalkulering. Displayet vil da telle opp etter hvert som kalkuleringen pågår for å vise fremgangen, og returnere til vanlig visning av parametre når kalkuleringen er fullført.

Parameter	**SW5**	**SW4**	**SW3**
Nedre frekvens	På	Av	Av
Øvre frekvens	Av	På	Av
Varighet	På	På	Av
Retningsstyring	Av	Av	På
Vindusfunksjon	På	Av	På
Defokuseringsgrad	Av	På	På

Tabell 2: Bryterposisjoner for aksessering av parametre

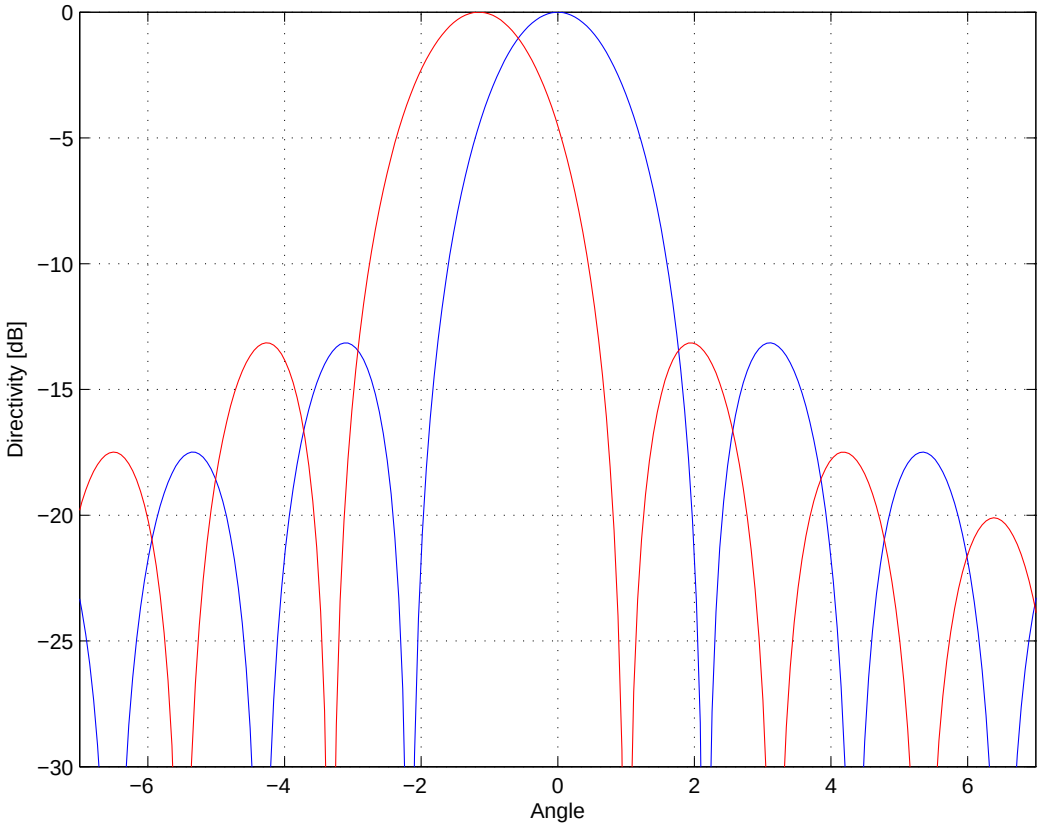
Bryter / Knapp	**Funksjon**
SW9	Av for dobbel puls, på for enkeltpuls
SW8	Når SW9 er på: Av for opp-chirp, På for ned-chirp
SW6	På for automatisk trigging med 10Hz
KEY3	Inkrementer valgt parameter
KEY2	Dekrementer valgt parameter
KEY1	Beregn og ta i bruk ny puls basert på innstilte parametre.
KEY0	Manuell trigger, sender valgt puls med en gang.

Tabell 3: Funksjonalitet til øvrige brytere

7 Bygging av transducer

For å få en styrbar transducer måtte vi bygge den selv. Transduceren er bygget ut i fra et design som Norbit har, men er tilpasset av oss for å få individuell styring av elementene. Arbeidet ble gjort på laboratoriet hos Norbit og under følger en beskrivelse av produksjonsprosessen. På grunn av konkurransehensyn har vi utelatt spesifikke detaljer i denne beskrivelsen.

Transduceren er bygget opp av en alumimniumssylinder som er påmontert 32 keramiske ringer. Transduceren opererer i transversmodus ($31 - mode$) fordi den mekaniske deformasjonen skjer normalt på det elektriske feltet [2]. De keramsike ringene sender lyd både innover mot kjernen og utover. Lyden som blir sendt mot kjernen er uønsket, og derfor har vi brukt et lag av kork mellom aluminiumssylinderen og de keramiske ringene. Korklaget absorberer det meste av lydsignalet som blir sendt mot kjernen av transduceren og tillegg er tykkelsen på korklaget lik en halv bølgelengde. Dette gjør at signaler som blir reflektert fra aluminiumen kommer i fase med lyden som sendes utover. Mellom hver av ringene er det brukt en spesiell type papp med tykkelse tilpasset for å få ønsket lobemønster på det utsendte signalet. Som frontlag brukes en støpemasse av epoxy og tykkelsen er satt til $N \cdot \lambda/4$. Dette gjøres for å minmere refleksjoner som oppstår i overgangen mellom lagene. Ringene som brukes har aktivt område på ca 180° i horisontal retning. De midterste elementene har størst område, og området avtar ut mot endene.



Figur 17: Beregnet vertikal direktivitet (Blå) og strålestyring med 83ns tidsforsinkelse mellom hvert element (Rød)

7.1 Mekanisk sammenstilling og lodding

Vi startet med å file ut et spor i korklaget for isolerte kobbertråder som skulle til innsiden av transducer-elementene. Deretter loddet vi en kobbertråd til innsiden av hvert element og plasserte de på transducer-huset med ett lag papp mellom hvert element. Kobbertrådene ble ført til toppen av huset, limt fast, og ført gjennom et hull inn til hulrommet i aluminiumskjernen.

Neste steg var å lodde fast kobbertråder på utsiden av elementene. Dette ble gjort helt i kanten av aktivt område på ringene. Kobbertrådene ble ført opp på utsiden av aktivt område, til toppen av huset, limt fast og ført inn til hulrommet. Kobbertrådene ble merket underveis for å holde styr på elementene.

For å sjekke at alle loddingene var i orden ble alle 32 elementene målt med kapasitansmeter fra enden på kobbertrådene.

Bilder av prosessen vises i vedlegg D.1.

7.2 Støping med epoxy-belegg

Støping av frontlag ble gjort som en tostegs prosess for å unngå luftbobler. Først penslet vi transduceren med et tynt lag epoxy som fikk herde før neste lag ble støpt. Epoxyen var tyntflytende og for å få et jevnt lag måtte holde transduceren i bevegelse under herdeprosessen. For å få raskere herding brukte vi varmluft. Det neste laget med epoxy ble påført ved å plassere transduceren i en støpeform som ble fylt med støpemasse. Etter herding ble frontlaget dreid ned til ønsket tykkelse i en dreiebenk.

Bilder fra støpeprosessen vises i vedlegg D.2.

Deretter parallellkoblet vi to og to elementer slik at vi fikk 16 kanaler. Dette ble gjort på innsiden av aluminiumssylynderen. Til slutt monterte vi på et deksel i bunnen og tok ut ledninger fra alle kanalene gjennom en slange med nippel mot dekselet. Ledningene er av typen Cat 5 og er tvistet for å minimere krysstale mellom kanalene.

7.3 Test av transducer i vann

For å verifisere at alle 16 kanalene virket gjorde vi en test med signalgenerator som ble koblet til en og en kanal. Utsendt lydsignal ble målt med en hydrofon og signalet ble tatt opp på et oscilloskop. Ved å sammenligne de målte nivåene kunne vi se at alle 16 kanaler hadde omtrent samme styrke.

Type	Fabrikat	Modell	Nummer
Signalgenerator	Agilent	33220A	NI#107
Transducer	Norbit	16 kanals	Egenprodusert
Hydrofon	Reson	TC4034	-
Oscilloskop	Agilent	DSO-X 3014A	NI#255

Tabell 4: Instrumentliste for test av transducer

7.4 Karakterisering av transduceren

For å finne frekvenskarakteristikken og direktiviteten til transduceren måtte vi gjøre karakteriseringer i vanntank.

7.4.1 Båndbredde, resonansfrekvens og horisontal direktivitet

Oppdragsgiver har et ferdig oppsett for slike tester og vi benyttet oss av dette for å måle båndbredde og horisontal direktivitet. Oppsettet er slik at man setter fast transduceren i en testjig som kan rotere, og et Matlab-program tar seg av rotering og måling. Målingene gjøres med en hydrofon som er tilkoblet Matlab via et oscilloskop. Resultatet av målingene vises i figur 18. På grunn av at størrelsen på vanntanken er 1.5 meter måtte vi begrense antall aktive elementer. Hadde vi parallellkoblet mange elementer ville vi fort komme i nærfeltet og da blir målingene dårlige. For vårt tilfelle er hele arrayet med 16 kanaler ca 0.1m langt, noe som vil gi en avstand på 2.6m ved 400kHz før vi er i fjernfeltet. Denne avstanden er beregnet fra formel 1. Med én kanal blir avstanden ca 1cm ved 400kHz. Vi gjorde målinger på én kanal med en avstand på 40cm mellom sender og mottaker, og kan dermed anta at vi er i fjernfeltet.

Båndbredde

Av figur 18a kan vi se at -3dB punktene for båndbredden er på ca 315kHz og 565kHz. Dette gir en båndbredde på 250kHz som er innenfor -3dB i forhold til sterkeste nivå. Dette er større enn hva vi forventet og gikk ut fra i spesifikasjonsdelen. Årsaken til at frekvensresponsen i denne målingen er ulik målingen som ble lagt til grunn i spesifikasjonene, se figur 10, er at transduceren som vi har laget har en litt ulik oppbygging. Kravet fra oppdragsgiver var at vi skulle ha størst mulig båndbredde og dermed kan pulsene som sendes ut tilpasses slik at frekvensområdet utnytter båndbredden i transduceren.

Resonansfrekvens

Av figur 18a kan vi se at resonansfrekvensen er på 430kHz. Vil ville sjekke hvordan dette stemmer med teorien og beregnet en teoretisk verdi på resonansfrekvensen fra formel 9.

$$\begin{aligned} f_{resonance} &= \frac{N_{31}}{L} \\ f_{resonance} &= \frac{1400\text{Hzm}}{0.0031\text{m}} \\ f_{resonance} &= 452k\text{Hz} \end{aligned}$$

Hvor verdier for N_{31} og L er tatt fra databladet for det keramiske materialet. [4] Når vi sammenligner utregningen med det målte resultatet kan vi se et avvik på 22kHz, noe vi mener er som forventet.

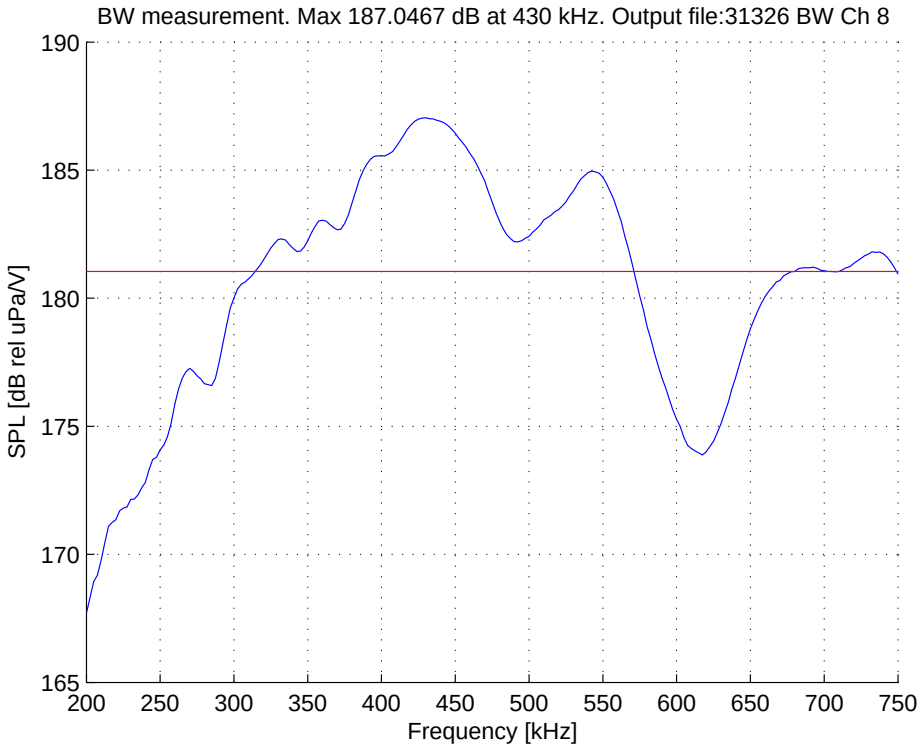
Horisontal direktivitet

Av figur 18b ser vi at direktiviteten er rimelig stabil mellom $\pm 90°$ og går aldri mer enn 3dB ned i dette området. Dette stemmer med at det aktive området for de midterste elementene er ca 180°.

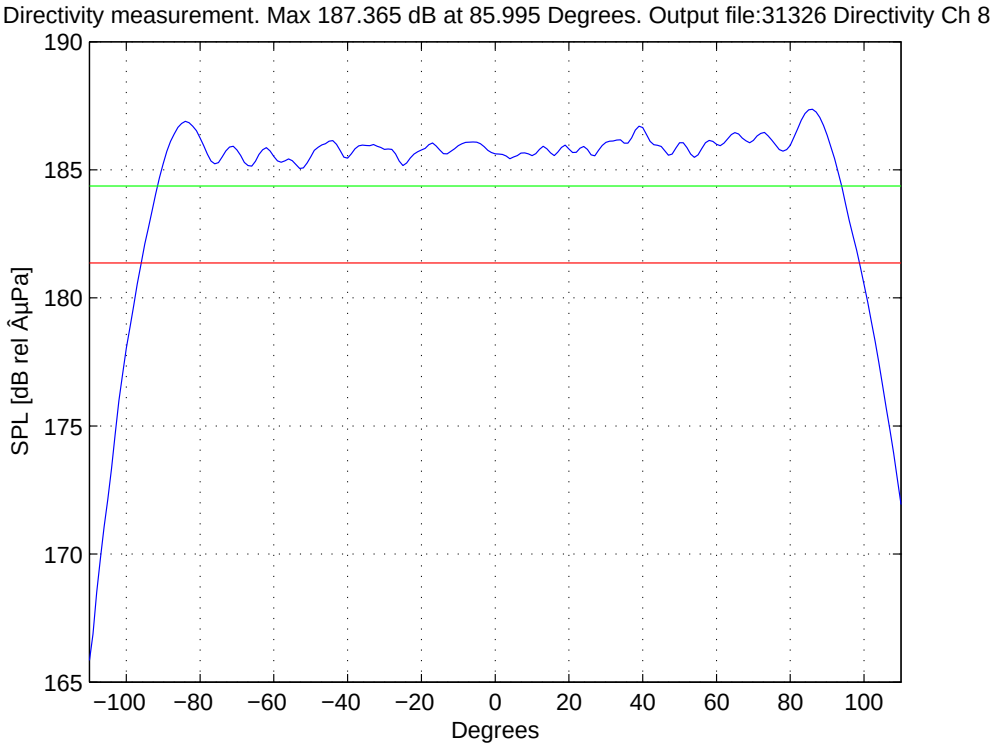
7.4.2 Vertikal direktivitet

Vi har gjort en beregning med Matlab for å se på vertikalt lobemønster med de 32 keramiske ringene. Vi har også beregnet strålestyring med å gi en tidsforsinkelse mellom hvert element. Resultatene vises i figur 17 og vil bli sammenlignet med målte resultater fra test i vanntank. Formel 2 og 3 er brukt i beregningene. Matlab-kode for beregningen finnes i vedlegg F.3.

Vi ønsket også å måle direktiviteten i vertikal retning. Dette ville vi gjøre med alle kanalene samtidig og måtte derfor låne et basseng på Marintek for å holde oss i fjernfeltet. Dette er beskrevet i kapittel 9.2.2.



(a) Båndbredde



(b) Direktivitet ved 400kHz

Figur 18: Karakterisering av transducer

8 Bygging av kretskort

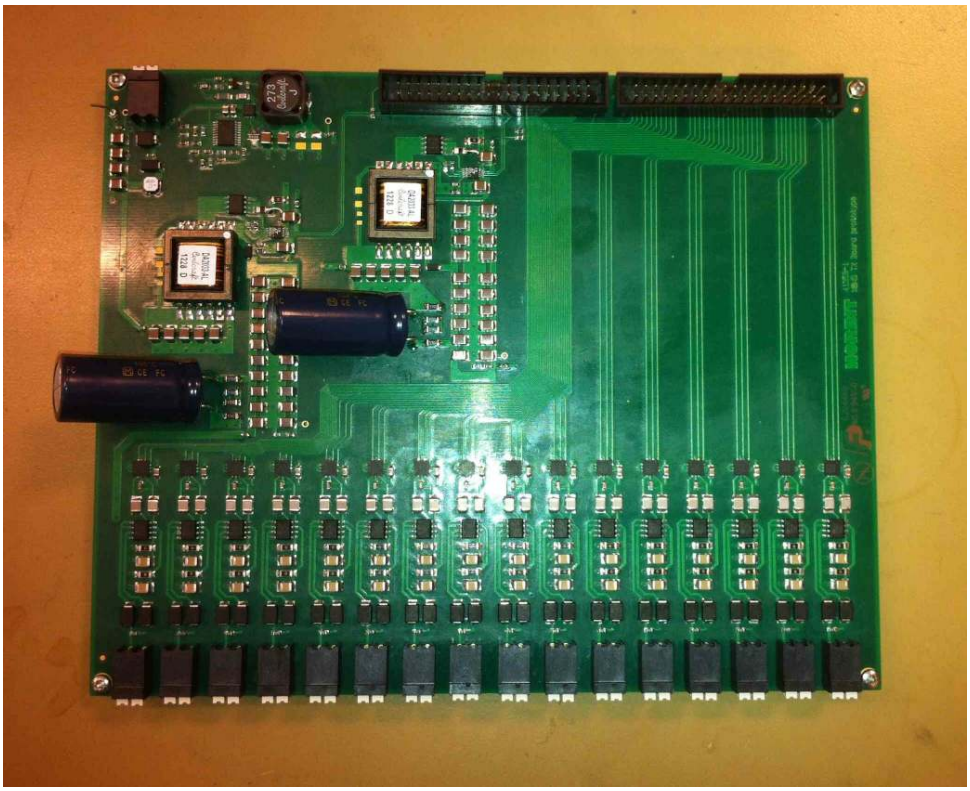
Mønsterkortet vi har designet, se kapittel 5, ble produsert i Kina. Lodding av komponenter gjorde vi selv. Arbeidet ble utført på oppdragsgivers laboratorier, med loddebolt og mikroskop. Under følger en beskrivelse av arbeidet og designtesten vi gjorde.

8.1 Lodding av mønsterkort

Vi startet med komponentene som hadde kjøle-pad på undersiden, fordi disse komponentene var vanskeligst å lodde. Her måtte vi legge ut loddepasta med ei sprøyte og bruke varmluft. Etter hver komponent gjorde vi enkle målinger med multimeter for å sjekke at det ikke ble noen kortsluttninger som følge av for mye loddetinn under komponentene.

Videre forsatte vi med resten av komponentene som ble loddet med loddebolt. Vi ventet med å montere 0Ωs motstandene. Disse ble først montert stegvis i den elektriske designtesten. For å sikre gode loddinger brukte vi loddetinn med bly, flussmiddel, rett størrelse på loddebolt og riktig temperatur.

Til slutt vasket vi kretskortet med rødsprit for å fjerne rester fra flussmiddelet. Bilde av ferdig loddet kretskort vises i figur 19.



Figur 19: Ferdig loddet kretskort

8.2 Elektrisk designtest av kretskort

For å verifisere at kortet virket som det skulle og oppfylte spesifikasjonene i kapittel 3 gjorde vi en elektrisk designtest. Instrumentliste vises i tabell 5.

Type	Fabrikat	Modell	Nummer
Signalgenerator	Agilent	33120A	NI#109
Signalgenerator	Agilent	33220A	NI#107
Signalgenerator	TTi	TG215	NI#58
Multimeter	Fluke	185	NI#177
Strømforsyning	TTi	CPX200	NI#253
Oscilloskop	Agilent	DSO6014A	NI#118

Tabell 5: Instrumentliste for elektrisk designtest av kretskort

Vi startet med å koble 24V til kortet og testet at inngangsfilteret ga beskyttelse mot feil polaritet og for høy spenning. Videre loddet vi inn 0Ωs motstanden R1 slik at 10Vs switch mode regulatoren fikk spenning. Vi målte utgangsspenningen på regulatoren til 9.90V og rippelspenningen var som forventet.

Neste steg var å lodde inn 0Ωs motstandene R17 og R31 slik at ladekretsen for -100V fikk spenning. Videre påtrykte vi et firkantsignal på charge-inngangen for å få kretsen til å lade opp kondensatorbanken. Vi målte spenningen til -100.5V. Samme prosedyre ble gjort for +100Vs ladekretsen, og spenningen på kondensatorbanken ble målt til +100.6V.

8.3 Modifikasjoner på kretskort

Da vi skulle måle oppladingstiden for ladekretsene fikk vi svært forskjellige og altfor lange tider, sammenlignet med hva vi beregnet i kapittel 4. Vi feilsøkte og fant ut at 0Ωs motstandene R24 og R25 som står i serie med sense-motstandene for ladekretsene ikke var nøyaktig 0Ω. Sense-motstandene er bare 15mΩ og når 0Ωs motstandene var mellom 30mΩ og 60mΩ ble det feil verdi på seriekoblingen. 0Ωs motstandene var satt inn for å ha mulighet til å endre verdier og derfor kunne vi bare laske over disse med kobbertråder. Etter dette var gjort målte vi oppladingstiden fra 64.1V til 100V til 72ms. Dette er noe bedre enn hva som ble beregnet i kapittel 4 og skyldes trolig at sense-motstanden som ble valgt var litt forskjellig fra beregnet verdi.

Neste steg i designtesten var å teste alle 16 utgangstrinnene med et påtrykt signal. Vi brukte flere signalgeneratorer, én for charge-signal, én for output enable-signal og én for datasignal. Ved målinger uten last tilkoblet på utgangen gikk alt bra. Da vi koblet til en resistiv last fikk vi problemer med at transistorene gikk i stykker. Etter feilsøking fant vi ut at støy fra utgangstrinnet smittet over på inngangen til transistor-driverne og dette gjorde at driverne begynte å oscillere.

I databladet for driverne fant vi et forslag om å plassere en motstand i serie med kondensatoren mellom driver og transistor, som skulle hjelpe mot oscillering i støyfølsome kretser. Vi prøvde oss frem med forskjellige verdier, for liten verdi gjorde at driverne fortsatt oscillerte, for stor verdi gjorde at signalet ble for tregt (lang tidskonstant). Endelig verdi for seriemotstanden ble 22Ω.

For å sikre mot kobling mellom inngangene på driverne satte vi inn motstander mellom inngangene og jord. Vi prøvde oss frem med forskjellige verdier og endte opp med motstander på 220Ω. Modifikasjonene ble gjort på alle 16 utgangstrinn og vises i figur21.

Vi gjorde også en endring i 10V switch mode-regulatoren slik at den nå leverer 6.2V. Dette ble gjort for å få et lavere nivå ut fra driverne, slik at dette i mindre grad kan smitte over på sin egen inngang.

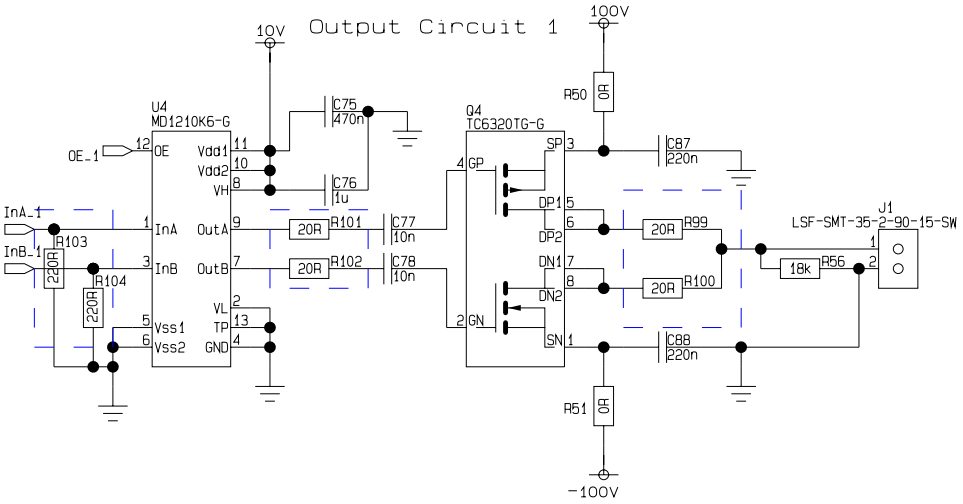
8.4 Elektrisk designtest av modifisert kretskort

Vi fortsatte med designtesten og etter modifikasjonene var gjennomført fikk vi gode resultater på alle 16 kanaler med resistiv last. Bilder fra måling med resistiv last vises i figur 20.

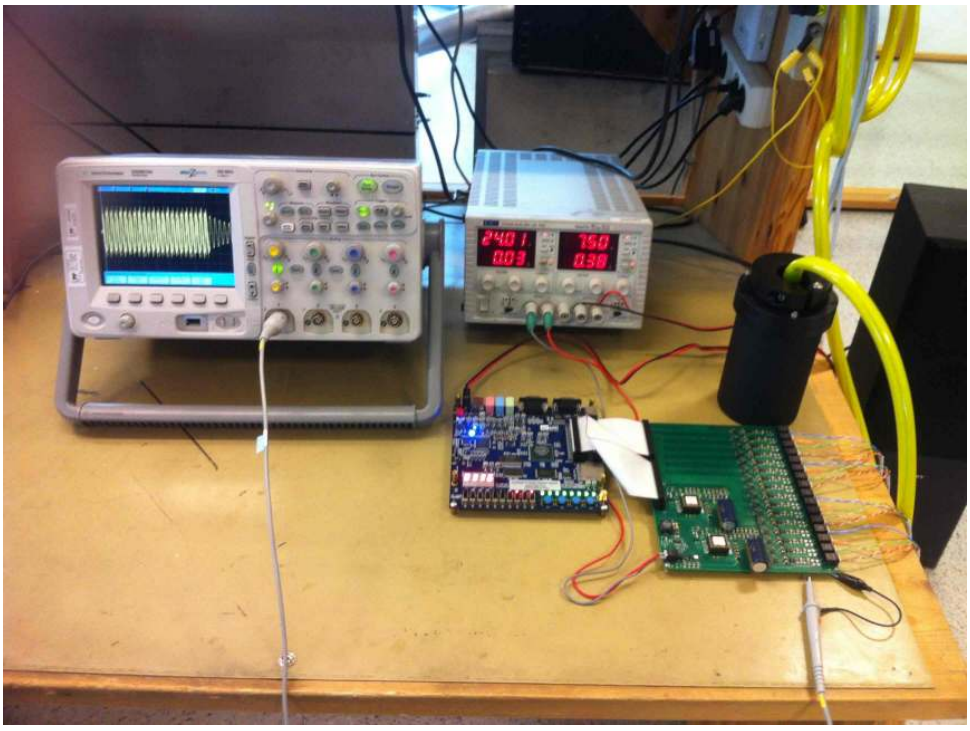
Figur 20: Skopbilde av måling på utgangen med resistiv last

Neste steg var å koble til transduceren. Dette ble gjort og også denne gangen fikk vi problemer med at transistorene røyk. Etter feilsøking fant vi ut at diodene som står i serie med utgangen fra transistorene gjorde at spenningen steg til over 300V målt peak to peak. Dette tåler ikke transistorene vi har valgt. Diodene er plassert etter forslag i databladet for transistorene, men diodene ble erstattet med små motstander. Vi tror at problemet skyldes akustiske refleksjoner som blir transformert til spenning i transduceren fordi den også vil kunne fungere som en mottaker. Ved å bruke motstander som erstatning for diodene vil denne spenningsrefleksjonen "brennes opp" over motstanden. Modifikasjonene ble gjort på alle 16 utgangstrinn og vises i figur 21.

Etter modifikasjonene gjorde vi en ny test og alle 16 kanaler fungerte som forventet. Se bilde av siste test i figur 22.



Figur 21: Modifikasjoner på utgangstrinn



Figur 22: Elektrisk designtest av kretskort

9 Test av prototype

9.1 Enkel test av prototype i vanntank hos Norbit

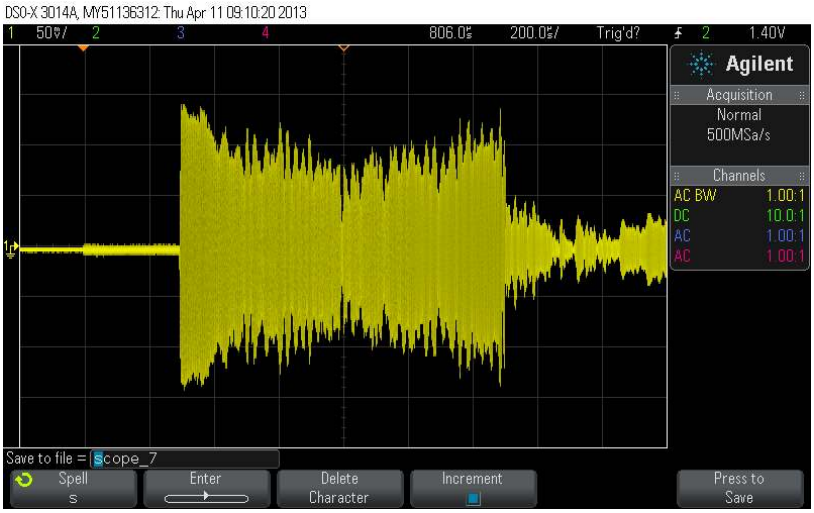
Som en forberedelse til testen vi skulle gjøre av hele systemet på Marintek, gjorde vi noen enkle målinger i vanntanken hos oppdragsgiver. Testene begrenses av størrelsen på vanntanken og av den grunn måtte vi ha mottakeren der hvor et mål normalt sett ville vært. Dette gjorde at vi ikke fikk målt tur/retur for lydsignalene, men testene gir en indikasjon på om prototypen fungerer.

Type	Fabrikat	Modell	Nummer
Senderkort	Norbit	16 kanals	Egenprodusert
Transducer	Norbit	16 kanals	Egenprodusert
Hydrofon	Reson	TC4034	-
Oscilloskop	Agilent	DSO-X 3014A	NI#255
Strømforsyning	TTi	CPX200	NI#253

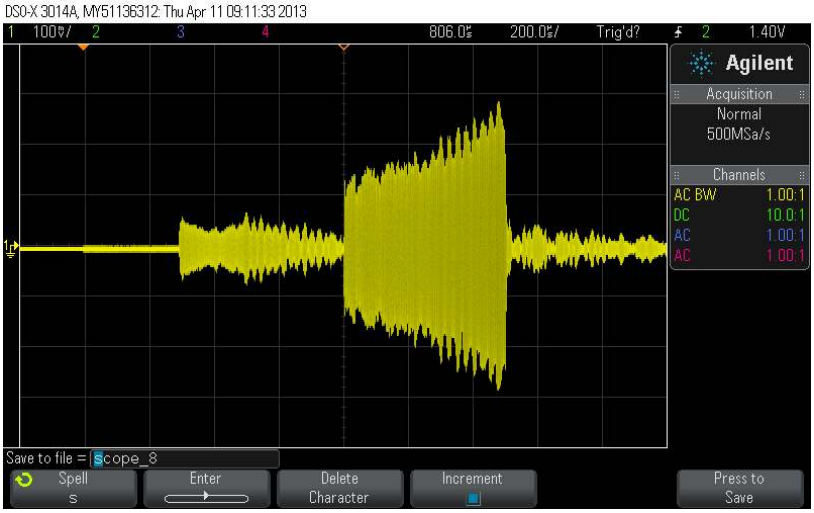
Tabell 6: Instrumentliste for enkel test av prototype

Vi ville teste om senderen klarte å sende ut to ortogonale pulser etter hverandre som hadde en gitt vinkel mellom seg. Vinkelen bestemmes av tidsforsinkelsen mellom hver kanal og justeres via brukergrensesnittet. Vi startet med å sentrere sender og mottaker i tanken, og gjorde målinger med oscilloskopet. Resultatet vises i figur 23a. Av figuren kan vi se at vi har omtrent like mye puls som sendes skrått nedover som skrått oppover. Grunnen til at amplituden endres over tid i hver av pulsene, er at signalstyrken til transduceren varierer med frekvens og signalet som sendes ut er av typen chirp. Videre i figur 23b og 23c, kan vi se resultatet når senderen ble flyttet h.h.v. ned og opp i tanken. Resultatene er som forventet og vi har fått testet at senderen klarer å styre vinkelen på de to utsendte pulsene.

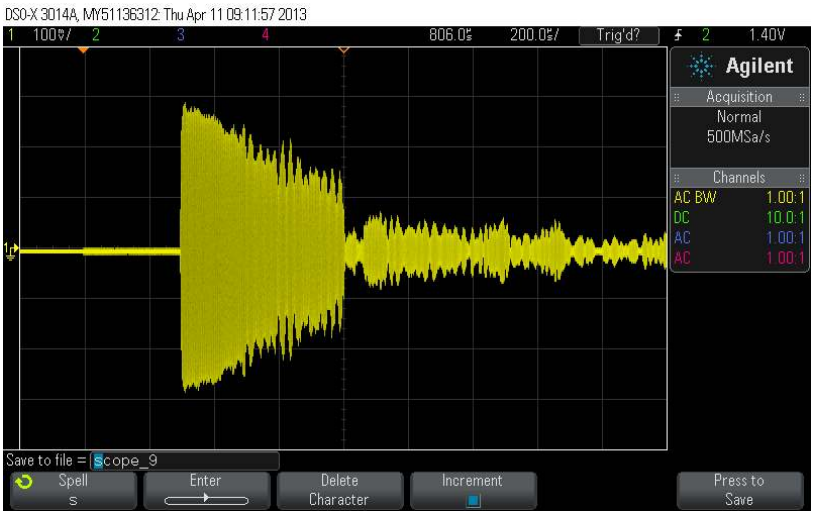
Vi ville også forsøke å analysere dataene fra målingene med ett av testskriptene som vi laget i høstprosjektet.[5] Dette for å se om de to pulsene kunne skilles fra hverandre med matched filtrering og for å se på vinkelen mellom dem. Testskriptet ble tilpasset til den nye båndbredden og til å analysere chirp i stedet for kampulser, og finnes i vedlegg F.2. Resultatet fra analysen viser at med senderen midt i tanken er nivået for de to pulsene ca like. Med senderen enten lavt eller høyt er det en forskjell på ca $\pm$16dB mellom de to pulsene, se figur 24.



(a) Skopbilde med sender midt i tanken

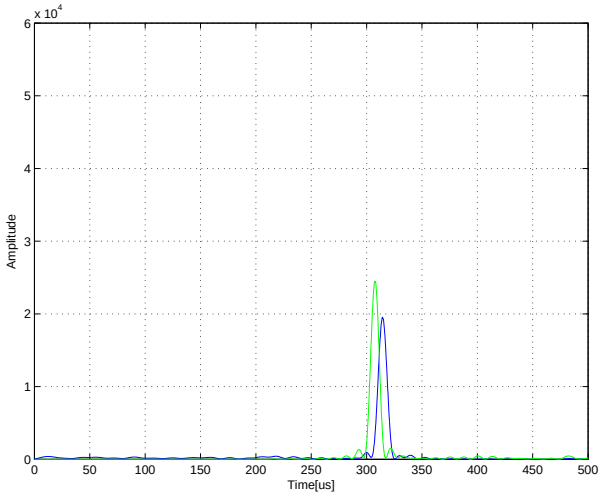


(b) Skopbilde med sender lavt i tanken

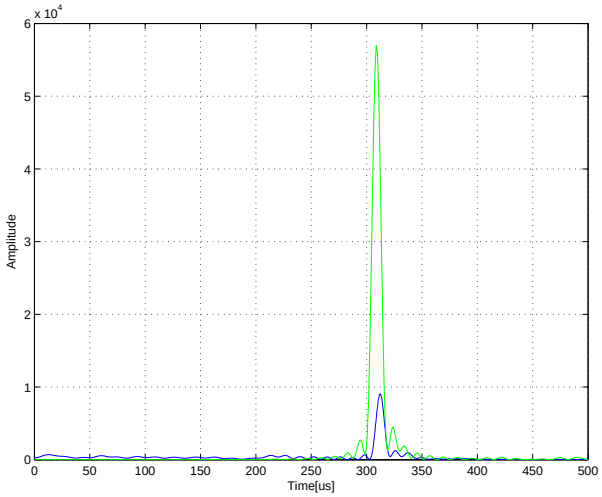


(c) Skopbilde med sender høyt i tanken

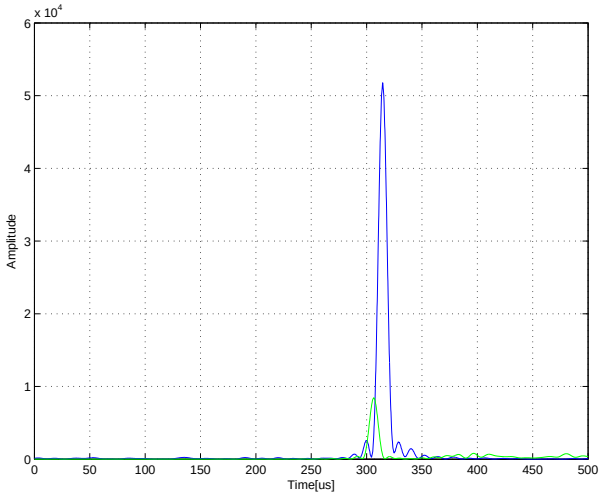
Figur 23: Test av ortogonale pulser i vanntank hos Norbit



(a) Nivåforskjell med sender midt i tanken



(b) Nivåforskjell med sender lavt i tanken



(c) Nivåforskjell med sender høyt i tanken

Figur 24: Matlab-analyse av ortogonale pulser i vanntank hos Norbit

9.2 Test av prototype i vanntank hos Marintek

Etter avtale med Sintef, fikk vi låne lilletanken på Tyholt i hele uke 17. Vi hadde et håp om å få låne et større basseng, men alle var opptatt helt til september.

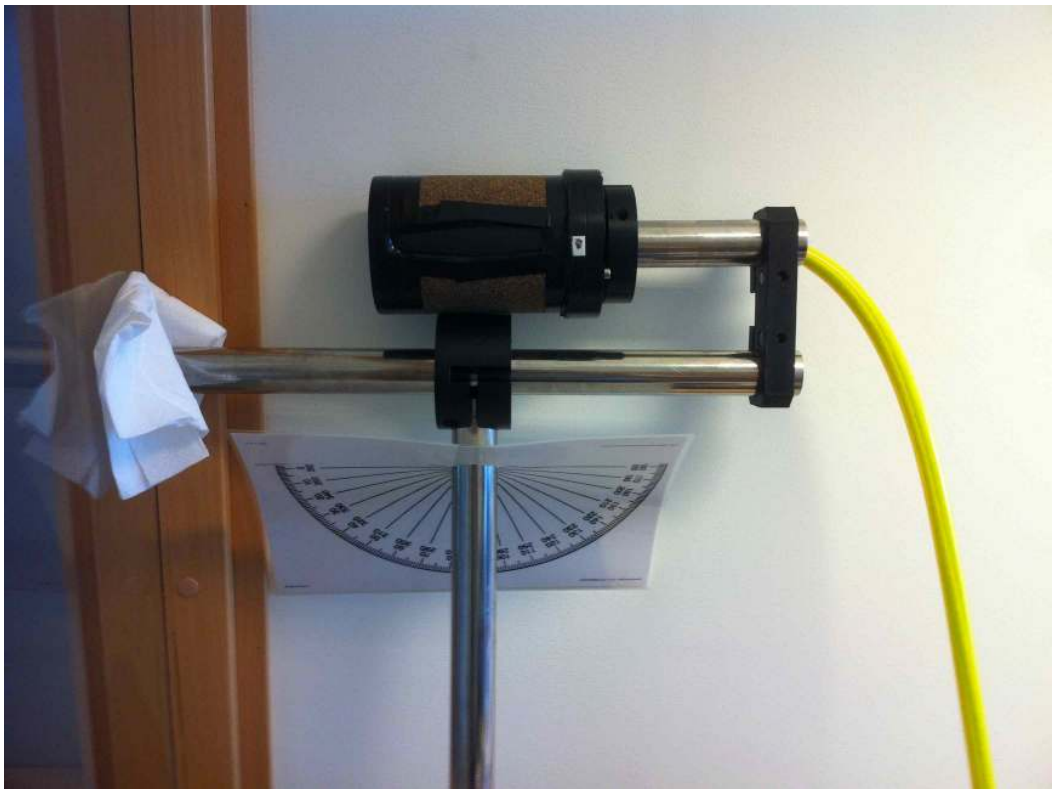
9.2.1 Forberedelser

På grunn av at lilletanken bare er ca 1 meter dyp ville vi gjøre modifikasjoner på transduceren for å begrense refleksjoner fra bassengbunnen og vannoverflaten. Vi lagde en tildekning av korkmateriale til transduceren slik at lobemønsteret i horisontal retning ble smalere. Deretter plasserte vi transduceren med en vinkel på 90° i forhold til normalt, slik at det styrte signalet ble styrt i horisontal retning i stedet for vertikalt. Vi målte den horisontale direktiviteten til den modifiserte transducenen og resultatet vises i figur 25. Om vi sammenligner med det orginale lobemønsteret i figur 18b, kan vi se at åpningsvinkelen er redusert fra ca 180° til ca 10° mellom -3dB punktene.

Figur 25: Horisontal direktivitet med tildekning av transduceren

Om vi plasserer transduceren midt i bassenget og åpningsvinkelen er 5° oppover og nedover vil vi nå bunnen og overflaten etter:

$$\begin{aligned} d &= 0.5\text{m} \cdot \sin(5°) \\ d &= 5.73\text{m} \end{aligned}$$



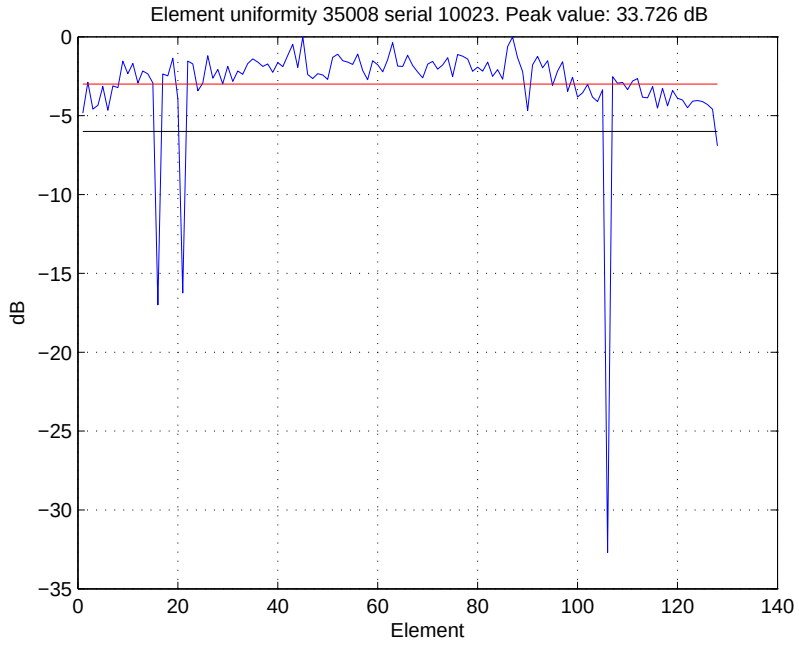
Figur 26: Testjig med tildekt transducer

Tildekningen gjør at vi kan holde oss i fjernfeltet og samtidig begrense støy fra refleksjoner i bunnen og overflaten. Bilde av tildekt transducer og testjig vises i figur 26.

Sonaren vi fikk til rådighet for å bruke som mottaker var en prototype, og tilstanden var noe usikker. Vi ville derfor teste om alle 128 mottakselementene virket. Oppdragsgiver har et ferdig testoppsett for slike tester og vi benyttet oss av dette. Dette er det samme oppsettet som vi brukte under direktivitet- og båndbreddemålingene, bare med en annen software. Testen viser at sonaren mest sannsynlig har 3 døde elementer i mottak, men vi antar at dette ikke vil påvirke måleresultatene i stor grad. Resultatet av elementuniformitetsmålingen vises i figur 27. Tabell 7 viser hvilke instrumenter som ble brukt under testen på Marintek.

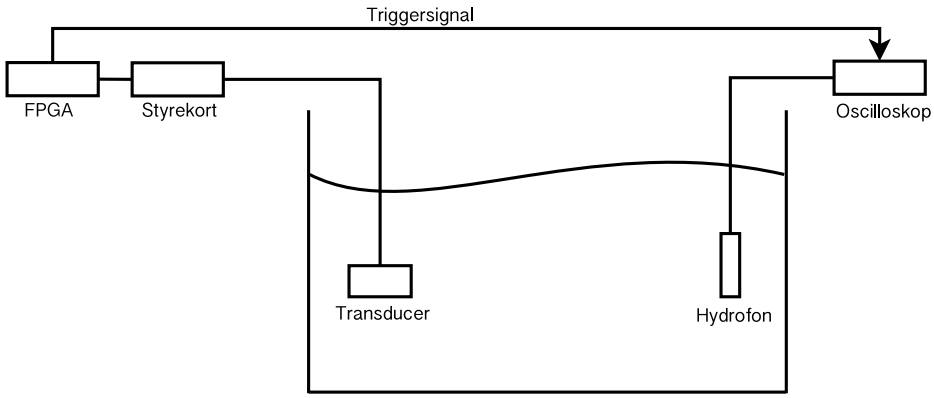
9.2.2 Måling av vertikal direktivitet

Fordi vanntanken hos Norbit er for liten til å gjøre målinger i fjernfelt med alle elementene tilkoblet, måtte vi gjøre måling av vertikal direktivitet på Marintek. Avstanden mellom transduceren og hydrofonen var på 3 meter, noe som sikrer at målingene ble gjort i fjernfeltet. Denne målingen blir ikke like nøyaktig som målingen av horisontal direktivitet, da rotasjon av transduceren gjøres manuelt. Vi forsøkte først å rotere transduceren for hånd ved hjelp av en stor gradskive, men dette ble for unøyaktig. Vi endte opp med å flytte på mottakeren i steg som tilsvarte 0.25° rotasjon og målte nivået fra mottakeren med et oscilloskop. Måleoppsettet vises i figur 28 og en modell av oppsettet i figur 29. Resultatet av direktivitetsmålingen vises i figur 30. Matlab-kode for generering av plot



Figur 27: Elementuniformitet for sonaren som brukes som mottaker

finnes i vedlegg F.3.



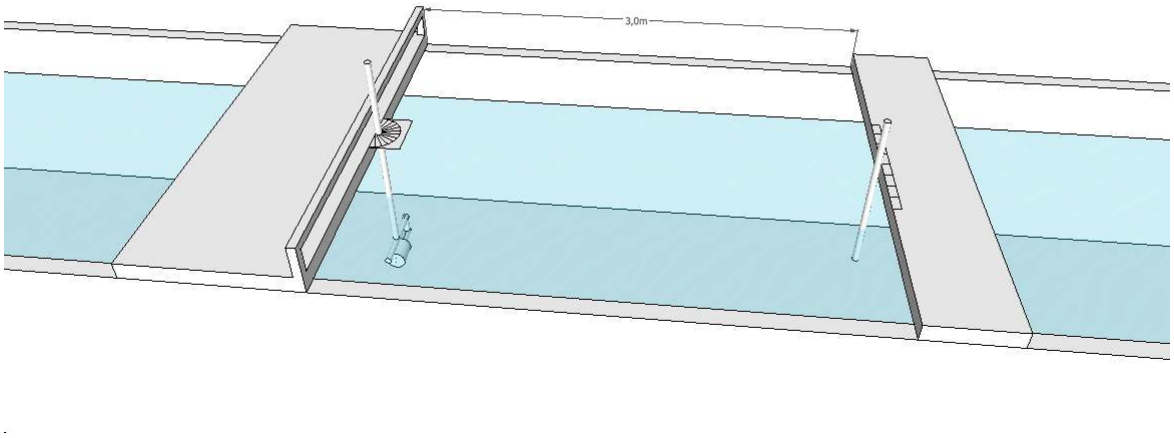
Figur 28: Måleoppsett for måling av vertikal direktivitet

9.2.3 Test av vinkelstyring, defokusering og vindusfunksjon

Etter å ha målt den vertikale direktiviteten fortsatte vi med funksjonstester av softwaren for styring av utsendte pulser. Fremgangsmåten for målingene var den samme som under målingen av vertikal direktivitet. Vi gjorde målinger av direktiviteten med Hanning-vindu. Vi gjorde også en test av retningsstyringen på utsendt signal og vi gjorde en test av defokuseringsfunksjonen. Resultatene av målingene er samlet i figur 31.

Type	Fabrikat	Modell	Nummer
Senderkort	Norbit	16 kanals	Egenprodusert
Transducer	Norbit	16 kanals	Egenprodusert
Hydrofon	Reson	TC4034	-
Oscilloskop	Agilent	DSO-X 3014A	NI#255
Strømforsyning	TTi	CPX200	NI#253
Sonar	Norbit	WBMS	Serial# 23
Hydrofon	Smart Material	045D	02-01
Interface	Norbit	SIU	-
SonarBell	Salt	-	-
Stålkule	-	4cm	-

Tabell 7: Instrumentliste for test av prototype på Marintek

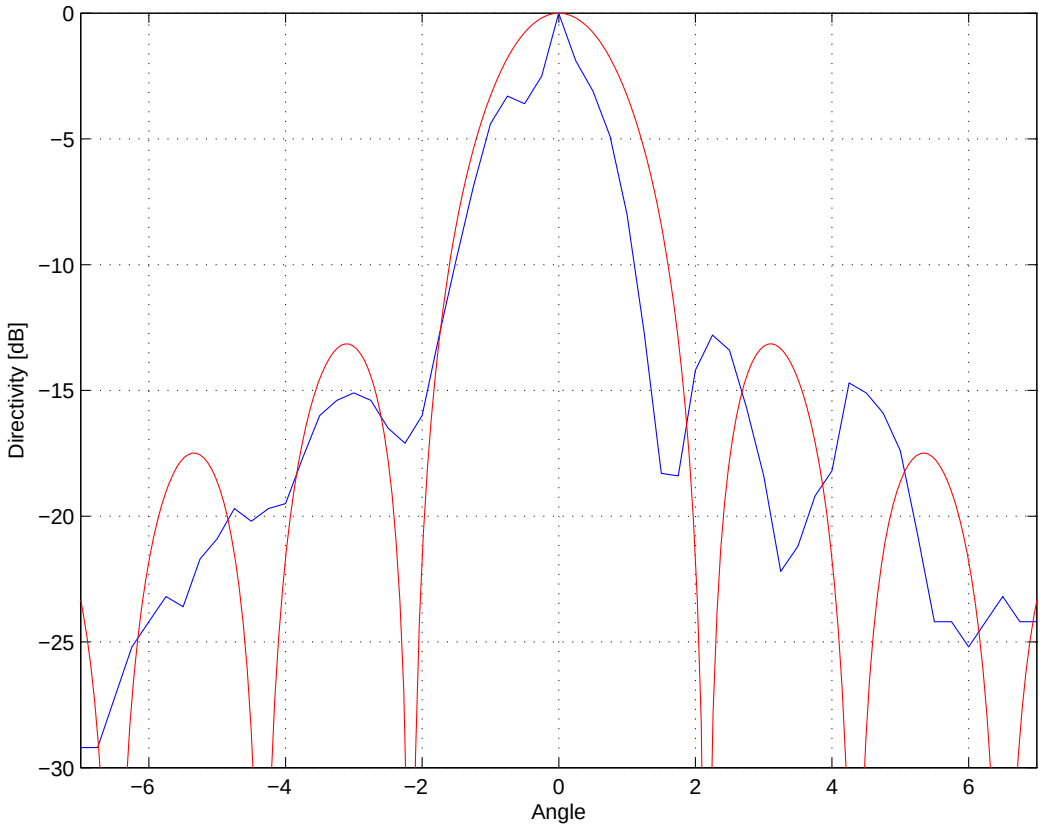


Figur 29: Modell av lilletanken ved måling av vertikal direktivitet

9.2.4 Test av posisjonsbestemmelse

Etter å ha analysert resultatene fra foregående målinger satte vi i gang med å teste måldeteksjon av et kjent mål i vannet. Transduceren ble holdt i samme posisjon, men mottakeren ble erstattet av en sonar og plassert på siden av transduceren. Målet ble plassert i forskjellige avstander fra transduceren. Analyse av mottatte data i sonaren ble gjort ved å hente ut rådata fra en kanal etter beamformeren i sonaren og behandle de i Matlab. Måleoppsettet vises i figur 32 og en modell av oppsettet i figur 33.

Vi forsøkte med forskjellige mål og forskjellige innstillinger for pulslengde, vinkler mellom de to ortogonale pulsene og defokusering. De første målingene ble gjort med en SonarBell som mål og på 3 meters avstand. Etterhvert flyttet vi oss til 6 meters avstand og gjorde målinger. SonarBell er et mål som skal gi høye refleksjoner, men da signalnivået var mer enn godt nok forsøkte vi med ei vanlig stålkule som mål. Vi gjorde målinger med forskjellige innstillinger og på 3 meter og 6 meters avstand. Et utvalg av analyserte data vises i figur 34 og 35. Matlab-kode finnes i vedlegg F.4.



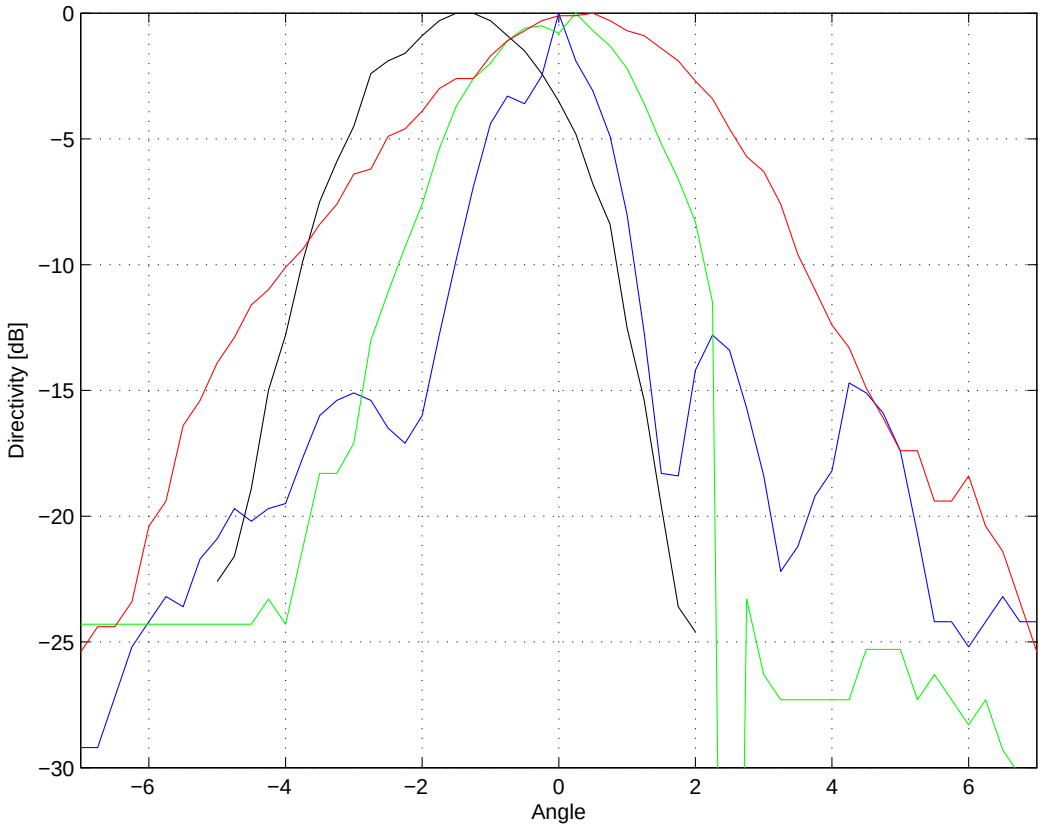
Figur 30: Simulert vertikal direktivitet (Rød) og målt vertikal direktivitet (Blå)

Bilder fra testingen i lilletanken vises i vedlegg E.

9.3 Vurdering av resultater

Under testen opplevde vi noe problematikk på grunn av refleksjoner fra bunnen, men etter noen justeringer gikk det greit. Vi vurderer det slik at tildekningen som vi laget for å begrense lobebredden i horisontal retning fungerte som planlagt og at denne bidro sterkt til å redusere problemer med refleksjoner. Testjig-ene som vi hadde laget fungerte som tiltenkt, og vi fikk plassert og justert utstyret på en grei måte. Videre måtte vi kun gjøre små endringer på softwaren for å hente ut data fra sonaren og totalt sett er vi tilfreds med forberedelsene vi gjorde til testene.

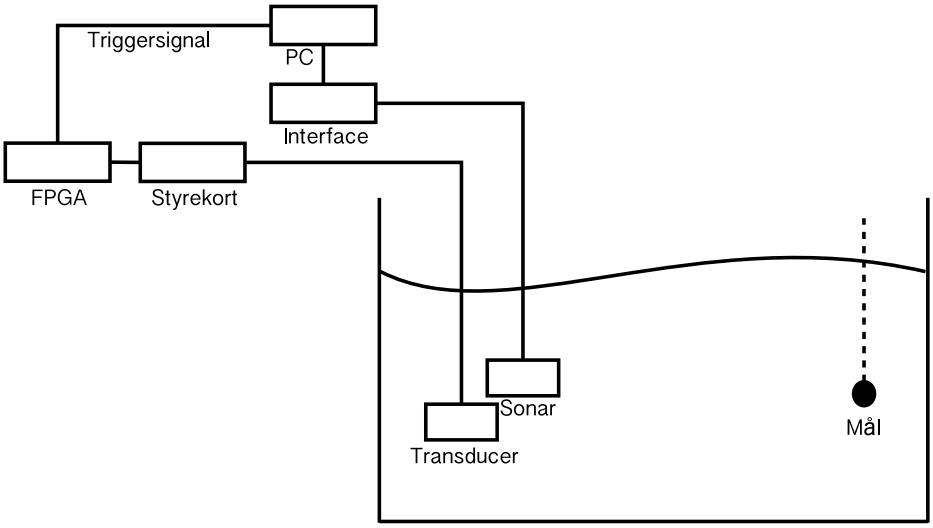
Resultatet fra målingen av vertikal direktivitet for transduceren, som ble målt i horisontal retning fordi vi hadde plassert transduceren med en vinkel på 90°, er presentert i figur 30. Når vi sammenligner simulert og målt kurve i ser vi at det er litt forskjell. Tildekkingen er skråskjært for å begrense sidelober. Det er det ikke tatt hensyn til i simuleringen. Det kan også være andre avvik mellom simulert utforming på transduceren og den faktiske utformingen. Mye av avviket tror vi skyldes målefeil. For å bedre resultatene kunne vi gjort mange målinger og sett på snittet av disse, men det tar lang tid å gjøre hver måling, og på grunn av lite tid fikk vi ikke gjort dette. I hovedsak er kurvene like og vi kan se at teori og praksis stemmer noenlunde.



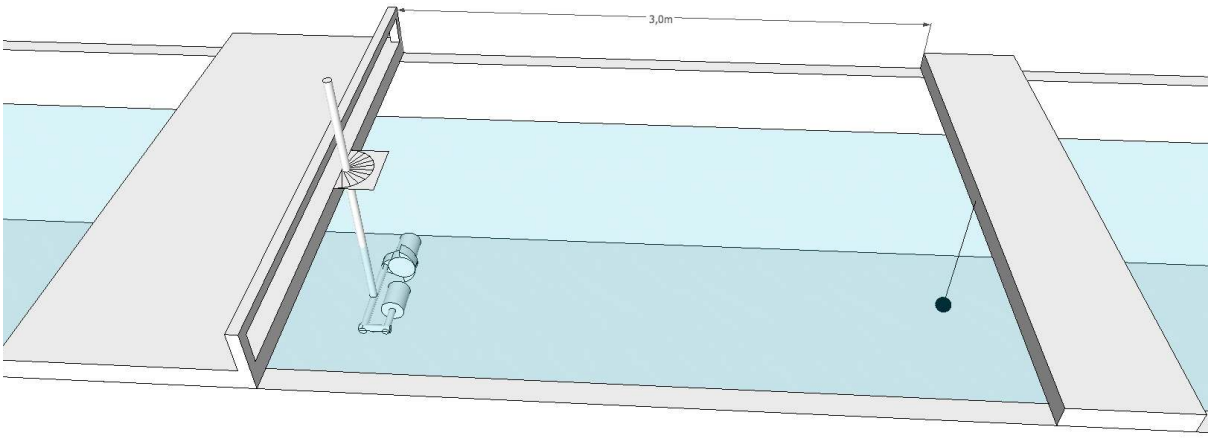
Figur 31: Vertikal direktivitet (Blå), med vindusfunksjon (Grønn), med defokusering (Rød) og med vindusfunksjon + vinkelstyring (Sort)

Resultatet fra testene av funksjonene vinkelstyring, defokusering og vindusfunksjon er presentert i figur 31. Vi kan se at ved bruk av vindusfunksjon er sidelobene redusert fra ca -13dB til under støynivået vi hadde ved måling. Kravet fra oppdragsgiver er -25dB og vårt støynivå ligger i dette området og vi kan anta at kravet er oppfylt. Vi kan også se at hovedloben er en del bredere enn uten vindusfunksjon og dette er som forventet med tanke på at energien smøres utover ved bruk av vindusfunksjoner. Når vi ser på kurven med defokuseringsfunksjon kan vi se at lobemønsteret er mye bredere enn uten denne funksjonen, noe som tilsier at denne fungerer som planlagt. Av figuren kan vi også se at vinkelstyringen fungerer fordi vi har forskjøvet toppen på lobemønsteret til siden.

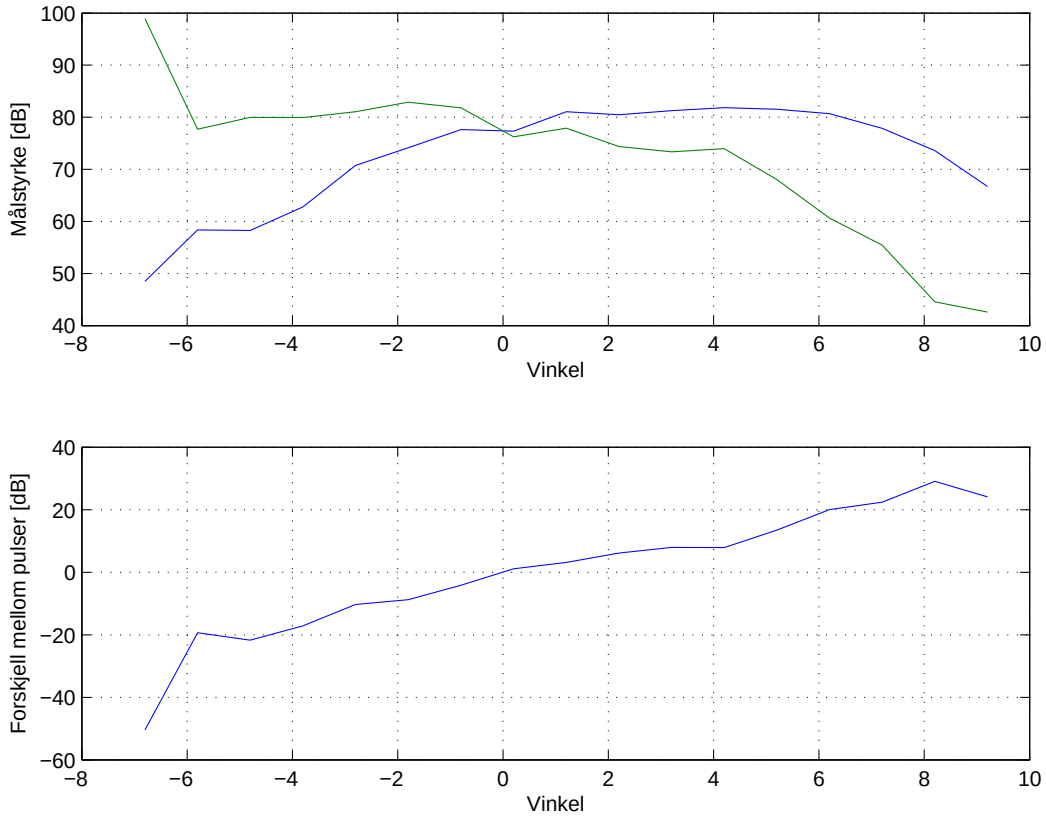
Resultatene fra testing av posisjonsbestemmelse vises i figur 34 og 35. Innstillingene for defokusering og vinkel mellom pulsene ble tilpasset vanntanken vi gjorde målingene i. Dette ble gjort for å begrense refleksjoner fra sideveggene. Hvis vi økte vinkelen mellom de to pulsene fikk vi et dødt område rett frem. Dette ble bedret ved å øke defokuseringen, men om denne ble økt for mye fikk vi refleksjoner fra sideveggene. Målingene med de innstillingene som ga best resultat er presentert i rapporten. Med målet på 3 meters avstand kan vi se at det brukbare deteksjonsområdet er fra ca $-6°$ til $+8°$, og med målet på 6 meters avstand er det brukbare deteksjonsområdet fra ca $-5°$ til $+5°$. Totalt sett er vi fornøyd med målingene og vi ser at systemet enkelt kan tilpasses til miljøet som det gjøres målinger i, gjennom brukergrensesnittet.



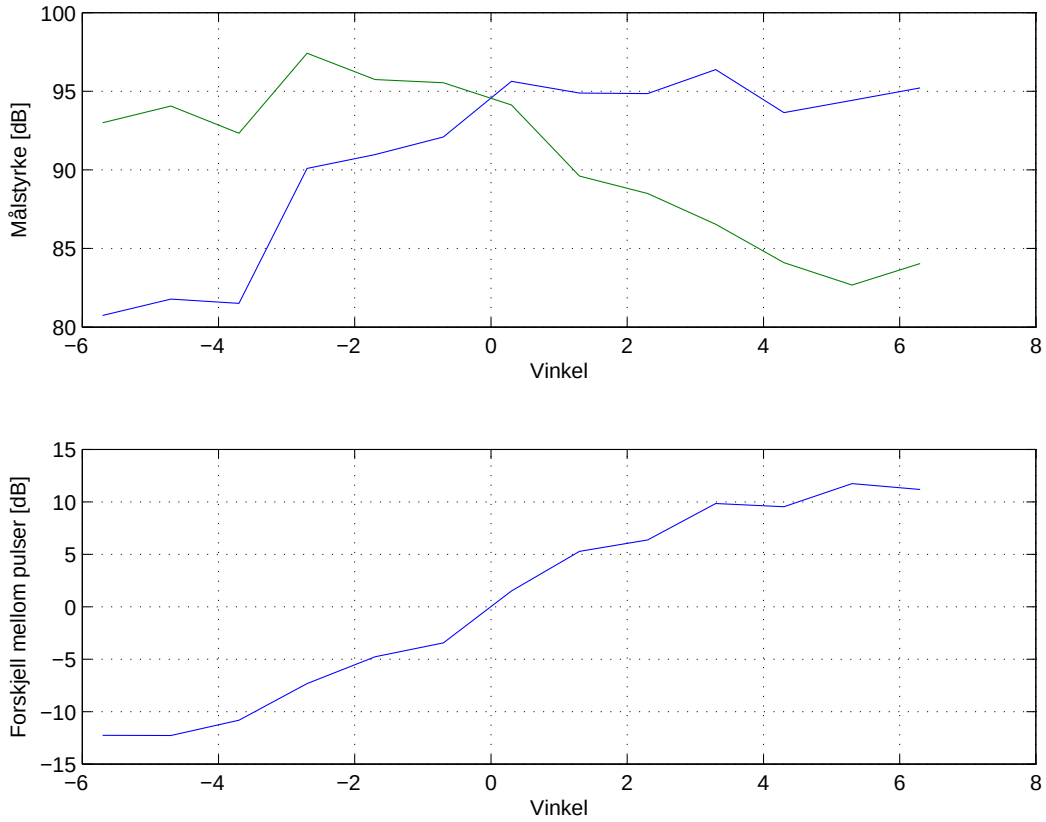
Figur 32: Måleoppsett ved posisjonsbestemmelse av et mål



Figur 33: Modell av lilletanken ved posisjonsbestemmelse av et mål



Figur 34: Stålkule som mål på 3 meters avstand



Figur 35: Stålkule som mål på 6 meters avstand

10 Muligheter for videre arbeid

Våre tester har vært begrenset av størrelsen på vanntankenene vi har hatt til rådighet. På grunn av refleksjoner fra vanntankenes sidevegger og bunn, og vannoverflaten måtte vi tilpasse det utsendte signalet. Det ville vært nyttig å gjøre tilsvarende tester i andre omgivelser for å se på hvilke begrensinger som er i prototypen med tanke både på deteksjonsavstand og med tanke på deteksjonsvinkel når refleksjoner ikke er den begrensende faktoren. Her er det spesielt defokuseringsfunksjonen og vinkelstyringen som ikke har blitt testet fullt ut. Et teststed for slike tester kan være Trondheimsfjorden, men da må man ha tilgang til båt og strømuttak.

Et naturlig neste steg vil være å implementere denne løsningen i en sonar. Senderkortet vi har laget er for stort til å være praktisk å implementere i en sonar. Det betyr at det må miniatyriseres. Effekt og fysisk størrelse henger sammen og man må vurdere hva som trengs opp mot hvor stor plass man har. Vi har brukt elektrolyttkondensatorer for å begrense størrelsen, men disse tåler ikke trykk. Dersom sonaren skal tåle trykk må disse erstattes av keramiske kondensatorer, noe som vil kreve større plass. For å spare plass kan man også se på om det er nødvendig med 16 kanaler. Om man kan klare seg med færre kanaler vil dette være et svært plassbesparende tiltak. Man kan også se på muligheten for å bruke drivertrinn og MOSFET-transistorer i chipper som har flere i én i hver pakke. Driverne MD1210 og transistorene TC6320 finnes også i pakker med h.h.v. 3 driverpar (MD1711) og 6 transistorpar (TC8020). Vår prototype er laget på en slik måte at testing med færre kanaler og mindre eller mer kapasitet i kondensatorbanken enkelt lar seg gjøre. For å ta denne avgjørelsen må man bestemme mer spesifikke krav og se hva som trengs for å oppfylle disse. Det kan også lages flere versjoner av løsningen, slik at man kan velge den beste ut fra bruksområde.

FPGA med nødvendig omkringliggende støttekomponenter må integreres i stedet for å være avhengig av et tilkoblet utviklerkort. Om mulig bør dette integreres i eksisterende FPGA i sonaren for å spare plass og kostnader.

For ha nytte av en styrbar sending er det også nødvendig at de øvrige systemene i sonaren oppgraderes for å utnytte seg av mulighetene det gir. For å kunne bruke ortogonale pulser i henhold til det systemet vi har demonstrert så må mottakssystem oppgraderes med et ekstra sett av alle databehandlingskomponenter fra og med matchfilter.

Man må også undersøke hvordan en tracking-algoritme som leter etter er definert bevegelsesmønster bør lages, og hvordan denne kan implementeres i sonaren. Slik vi ser det kan dette gjøres på to måter: Den ene er å implementere en algoritme direkte i sonaren, den andre er å implementere denne algoritmen i fremvisningsprogramvaren (topside) på en vanlig datamaskin. Fordelen med å implementere dette direkte i sonaren vi være at den kan jobbe i det stille og ta avgjørelser på egenhånd. På denne måten kan man velge at sonaren kun sender beskjed opp når det trengs. Dette er positivt da begrenset kapasitet på kabler og utstyr ofte forekommer på offshore. Ulempen er at dette krever plass i sonaren. Fordelen med å implementere løsningen i software topside er at man har bedre regnekraft og lagringsplass, men denne løsningen krever større båndbredde fra sonaren og opp til topside.

11 Konklusjon

Vi har laget en prototype som demonstrerer hvordan det er mulig å lage en sender som er styrbar med hensyn på retning og direktivitet ved å generere individuelle signaler til hvert element i et array. Vi har også demonstrert at det er mulig å bruke en slik sender i kombinasjon med en sonar som i utgangspunktet er 2D for å posisjonsbestemme mål i tre dimensjoner.

Våre beregninger viser at det er begrensinger på hvor godt et signal kan konstrueres når vi har flere krav å ta hensyn til. Dette fører til at man må inngå et kompromiss hvor man vektlegger hvilke funksjoner som skal få prioritet. Som eksempel kan vi nevne at om man ønsker et lavt sidelobenivå må man bruke en vindusfunksjon, og dette fører til en bredere hovedlobe, noe som kan være negativt om man ønsker høy direktivitet. Det samme gjelder for kravet til isolasjon mellom to pulser, hvor vi må gå ned i isolasjon for å få ønsket tidsoppløsning.

Vi har også sett at ved å først lage en prototype hvor man har fokus på fleksibilitet vi man ha gode muligheter for å gjøre nødvendige modifikasjoner underveis.

Oppgaven har bestått av varierte arbeidsoppgaver og vi har satt oss inn i mye nytt. Her vil vi fremheve design av kretskort og transducer, samt programmering av FPGA som har vært veldig lærerikt og utfordrende. Størstedelen av oppgaven har vært av praktisk art, og vi føler at vi har utviklet noe nyttig. Vi mener oppgaven har passet veldig bra til hovedprofilen på vårt studium, da vi har jobbet med elektronikk-design, akustikk-design, programmering og signalbehandling.

Tidsmessig har prosjektet stort sett gått etter planen. Vi ble litt forsinket i forhold til datoen for når kretskortet skulle vært ferdig testet. Dette skyldes problemene med støy i utgangstrinnene, men forsinkelsen ble tatt inn relativt kjapt.

Masteroppgaven har vært en lærerik prosess samtidig som vi har fått brukt mye av den teoretiske kunnskapen vi har tilegnet oss gjennom studiene Det praktiske arbeidet og verifikasjonstestingen viser at teori og praksis stemmer over ens. Det har vært veldig gøy med en industrinær oppgave, og vi håper at resultatene er til nytte for Norbit.

Referanser

[1] Johnson & Modugno, “IEEE: True angle estimation from a line array using time-delay estimates over a known rotation”, 1987

[2] Hovem, “Marine Acoustics”, 2010

[3] Sherman & Butler, “Transducers and Arrays for Underwater Sound ”, 2007

[4] Ferroperm, “Ferroperm piezoceramics catalogue”, Nedlastet PDF fra `http://www.ferroperm-piezo.com` den 17.4.2013

[5] Bjørnar Leithe, Stian Sønderland “Rapport Høstprosjekt 3D object tracking using 2D active multibeam sonar”, 2012

[6] Matthew Robertson, “A Brief History of InvSqrt”, 2012

[7] T. Kasami, “Weight Distribution Formula for Some Class of Cyclic Codes”, 1966

[8] Nick, “Fast and accurate sine/cosine”, 2006 (Hentet 2013-05-06) `http://devmaster.net/forums/topic/4648-fast-and-accurate-sinecosine/`

[9] Wikipedia, “Barker code”, (Hentet 2013-05-14) `http://en.wikipedia.org/wiki/Barker_code`

[10] APT, “Total Harmonic Distortion and Effects in Electrical Power Systems”, (Hentet 2013-05-07) `http://www.aspowertechnologies.com/resources/pdf/Total%20Harmonic%20Distortion.pdf`

[11] Texas Instruments, “LM5116 Datasheet”, 2011

[12] Supertex, “TC6320 Datasheet”, 2008

[13] Supertex, “MD1210 Datasheet”, 2012

[14] Linear Technologies, “LT3750 Datasheet”, 2005

[15] Coilcraft, “MSS1278 Datasheet”, 2011

[16] Diodes Incorporated, “ES2G Datasheet”, 2005

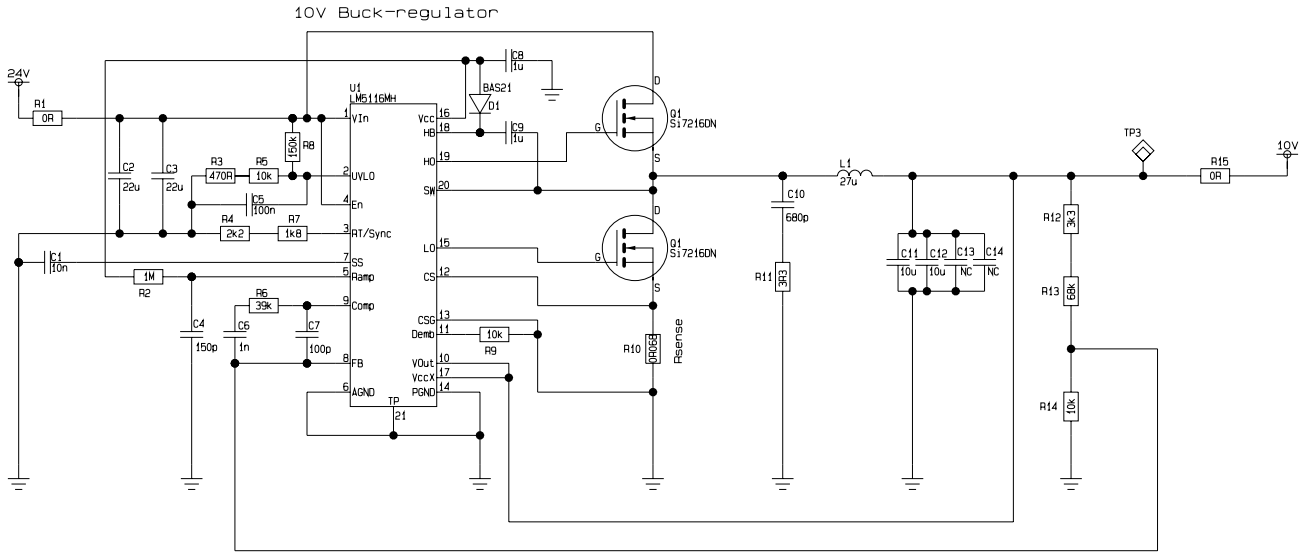
[17] Linear Technologies, “LT3751 Datasheet”, 2008

[18] Coilcraft, “DA2033 Datasheet”, 2011

[19] Diodes Incorporated, “DFLU 1400 Datasheet”

[20] Vishay Siliconix, “Si4104DY Datasheet” 2012

A Skjema



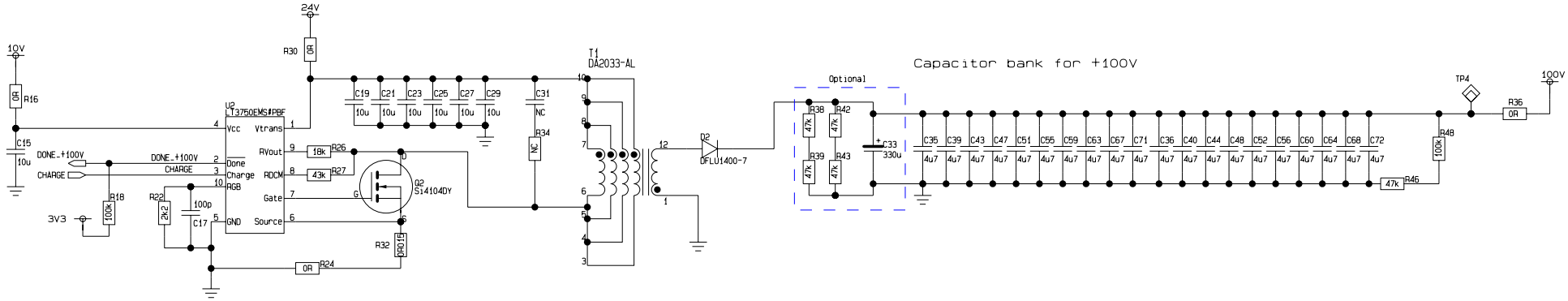
H1 2.7mm H2 2.7mm H3 2.7mm H4 2.7mm

Fid1 Fid2 Fid3

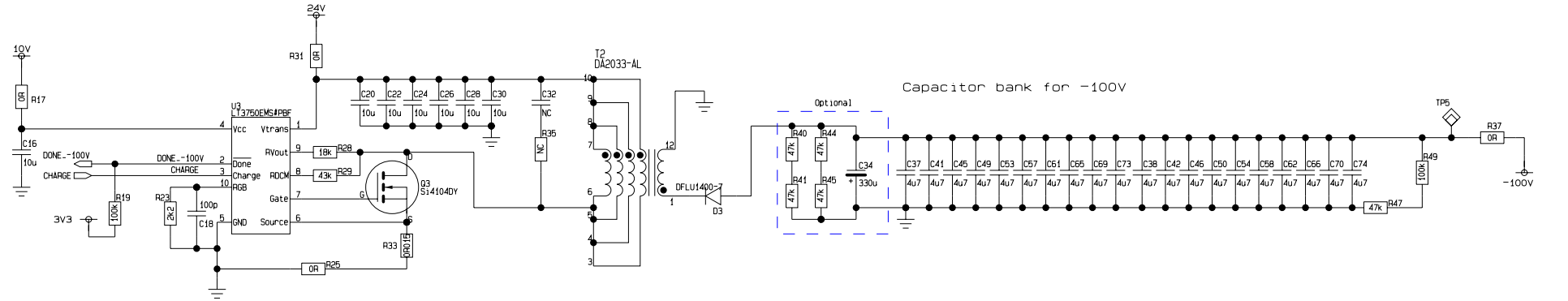
Fiducial Fiducial Fiducial

NORBIT	Title	WBMS_TX_Board_Prototype	
	Part #	31326	
Section	10V Switch-mode supply		
Filename	31326_WBMS_TX_Board_Prototype/A/31326-1.cir/001.sht		
Revision	1	Engineer	SAS/BL
Date		Approved	
			Page 1 of 5

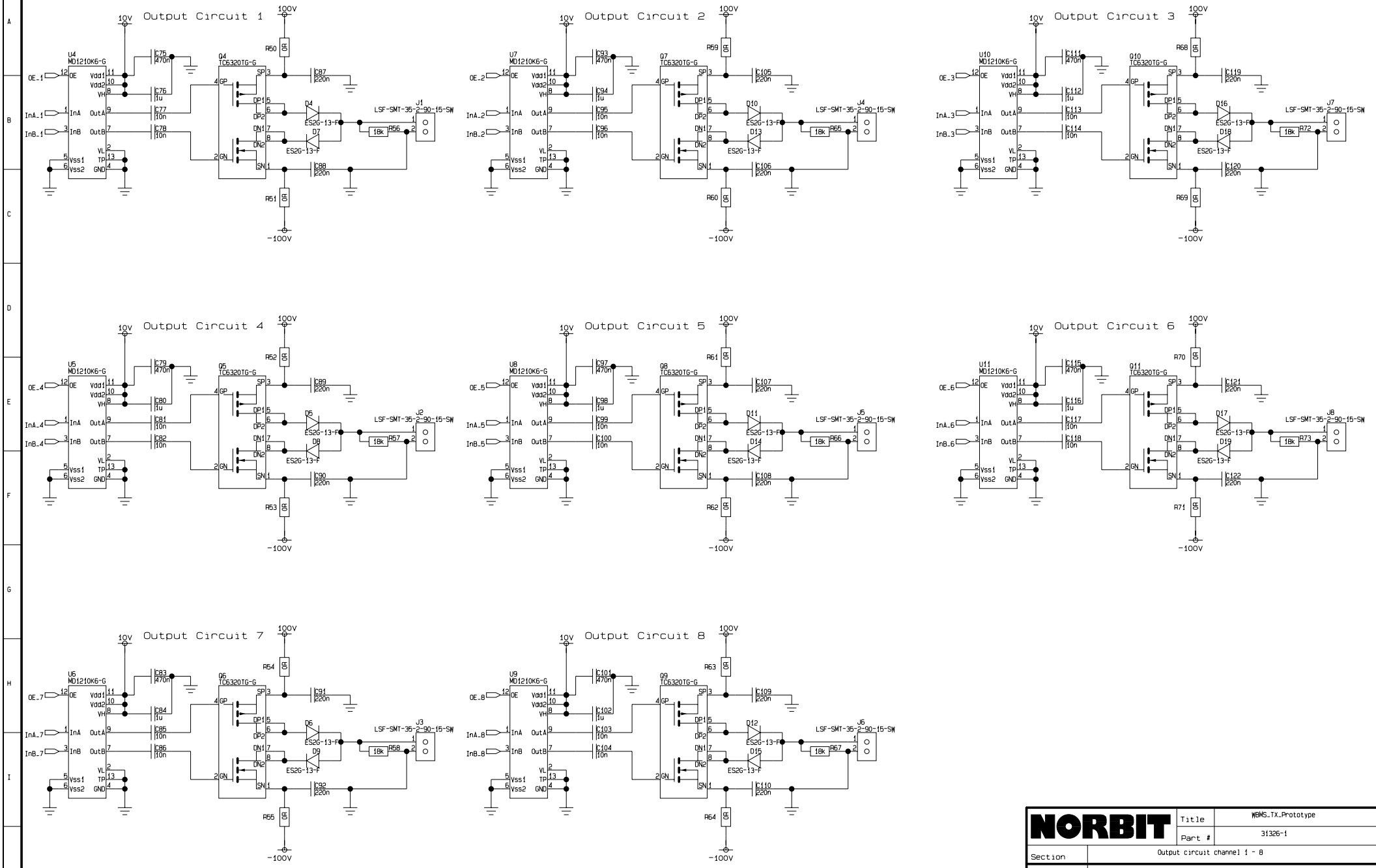
Charger for +100V



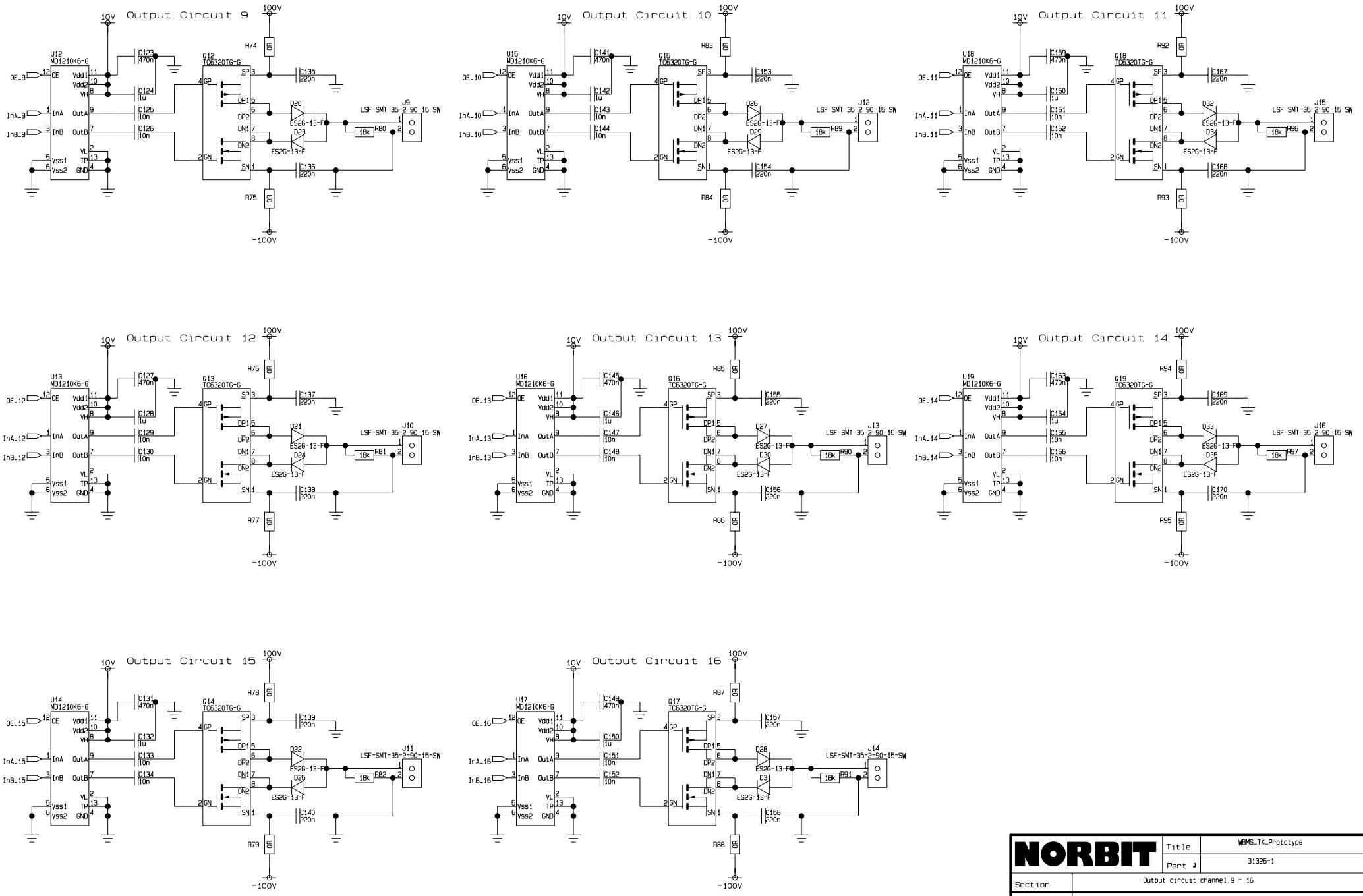
Charger for -100V



NORBIT	Title	WBMS_TX_Board_Prototype		
	Part #	31326-1		
Section	Charger circuit and capacitor bank			
Filename	31326_WBMS_TX_Board_Prototype/A/31326-1.cir/002.sht			
Revision	1	Engineer	BL/SAS	Page 2 of 5
Date		Approved		

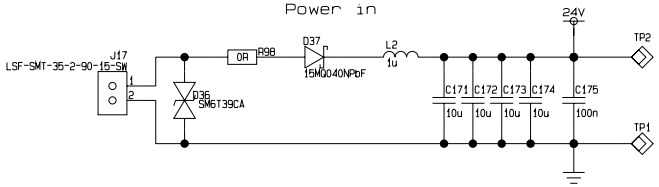


NORBIT	Title	WBMS_TX_Prototype	
	Part #	31326-1	
Section	Output circuit channel 1 - 8		
Filename	31326_WBMS_TX_Board_Prototype/A/31326-1.cir/003.sht		
Revision	1	Engineer	BL/SAS
Date		Approved	

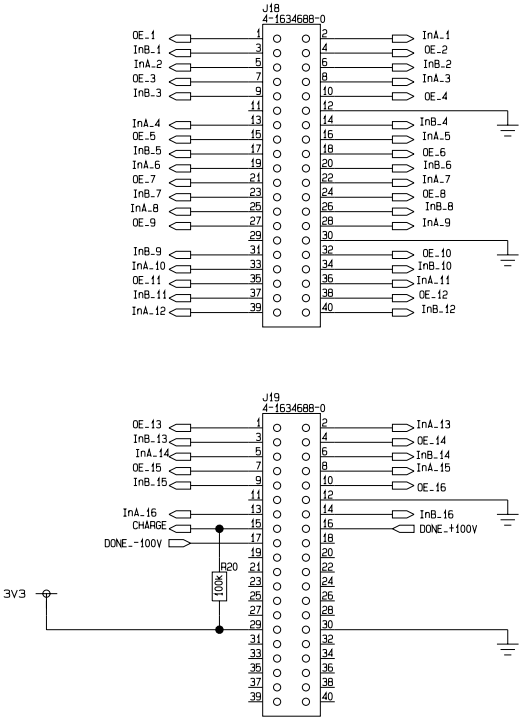


NORBIT	Title	WBMS_TX_Prototype	
	Part #	31326-1	
Section	Output circuit channel 9 - 16		
Filename	31326_WBMS_TX_Board_Prototype/A/31326-1.cir/004.sht		
Revision	1	Engineer	BL/SAS
Date		Approved	

Power in



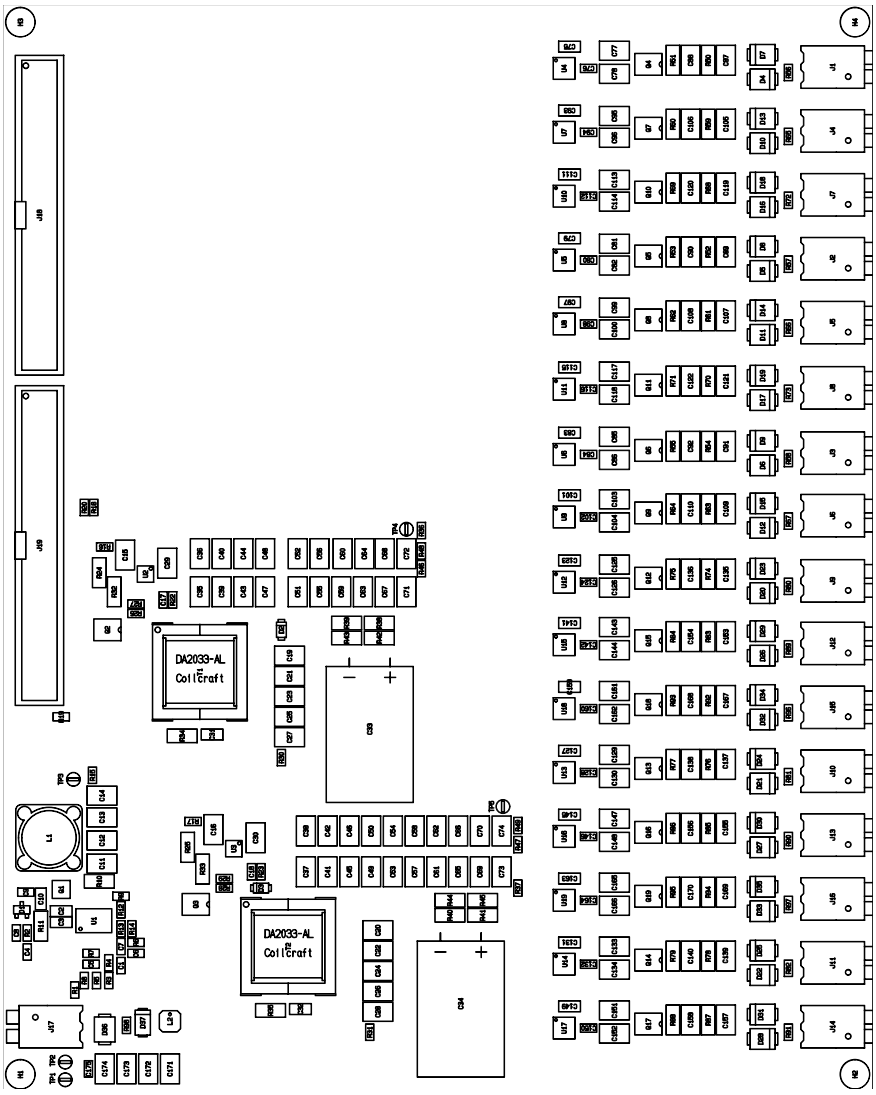
Interface to Altera Cyclone II FPGA Starter Board



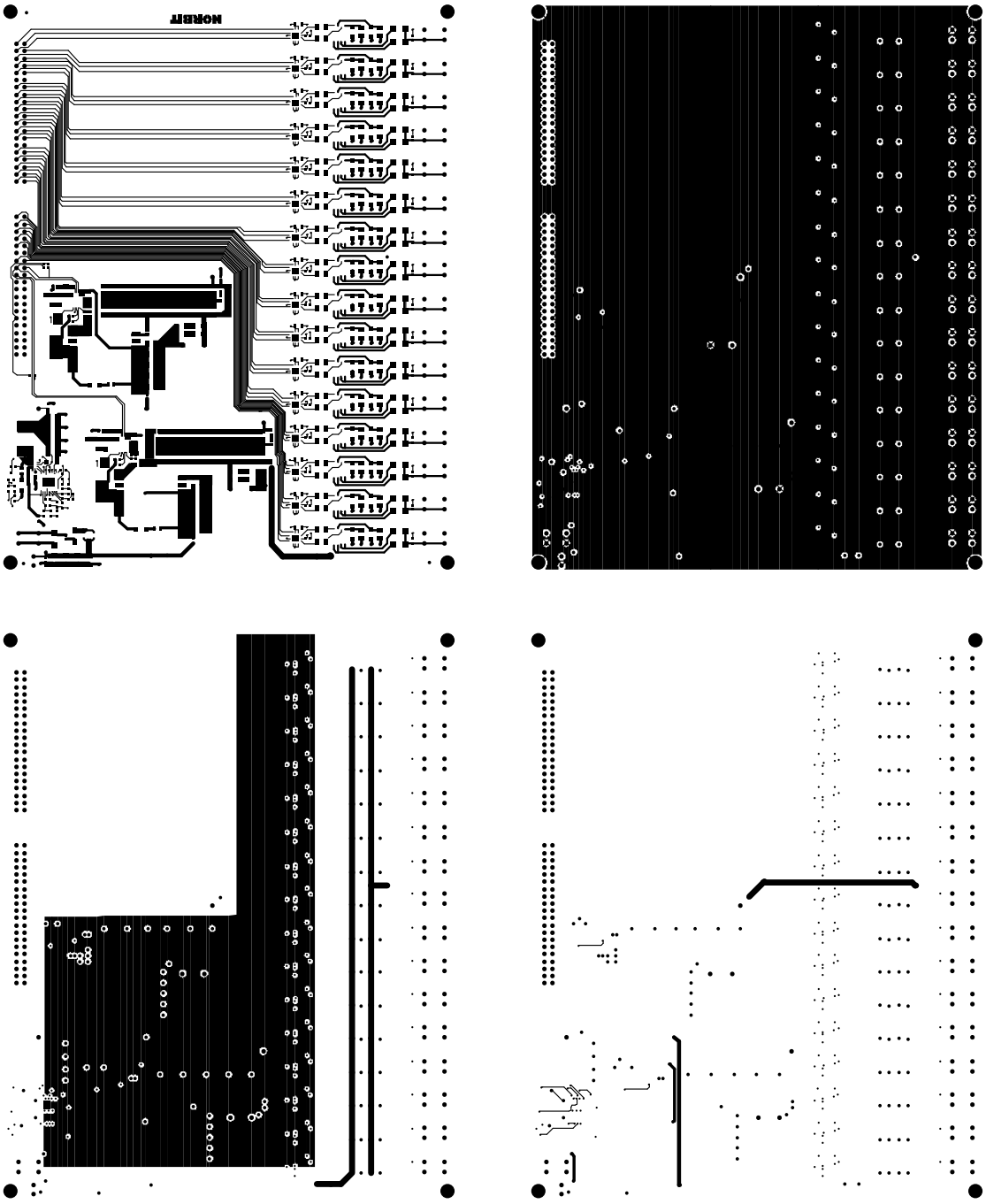
NORBIT	Title	WBMS_TX_Prototype		
	Part #	31326-1		
Section	Interface to FPGA and input filter			
Filename	31326_WBMS_TX_Prototype/A/31326-1.cir/005.sht			
Revision	1	Engineer	BL/SAS	Page 5 of 5
Date		Approved		

B Utlegg

B.1 Komponentplassering



B.2 PCB lag 1 til 4



C Komponentliste

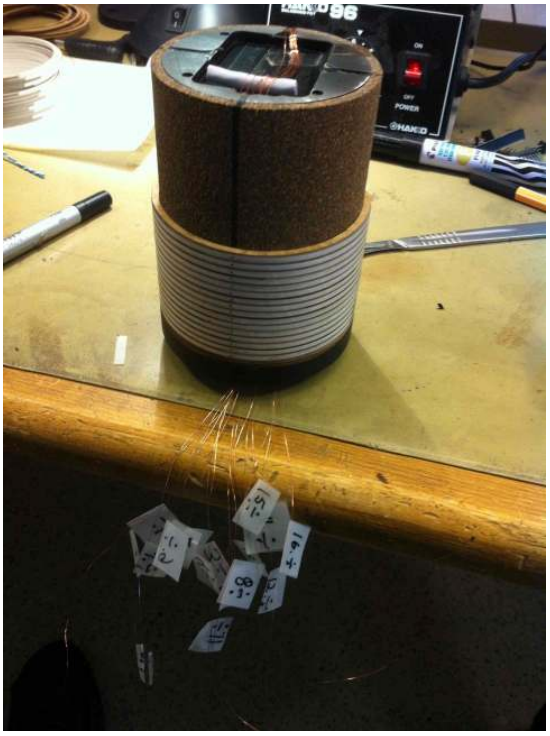
Bill of Materials

Pos	Manufacturer	Partno.	Description	Notes	RoHS	Package	Qty	Ref
1	Norbit	41xxx-x	PCB				1	
2	Phycomp	2E+11	Capacitor, Ceramic 100p/50V, 2%, NP0			603	2	C17, C18
3	Phycomp	2E+11	Capacitor, Ceramic 150p/50V, 2%, NP0			603	1	C4
4	Phycomp	2E+11	Capacitor, Ceramic 680p/50V, 2%, NP0			805	1	C10
5	Phycomp	2E+11	Capacitor, Ceramic 10n/50V, 2%, NP0			1210	32	C77, C78, C81, C82, C85, C86, C95, C96, C99, C100, C103, C104, C113, C114, C117, C118, C125 C126, C129, C130, C133, C134, C143, C144, C147, C148, C151, C152, C161, C162, C165, C166
6	Phycomp	2E+11	Capacitor, Ceramic 100p/50V, 5%, X7R			603	1	C7
7	Phycomp	2E+11	Capacitor, Ceramic 10n/50V, 5%, X7R			603	1	C1
8	Phycomp	2E+11	Capacitor, Capacitor, X7R 0603 100n/50V, 1			603	1	C175
9	Yageo	CC0603KI	Capacitor, Ceramic 1n/50V, 10%, X7R			603	1	C6
10	Taiyo Yuden	QMK325B	Capacitor, Ceramic 220n/250V, 10%, X7R			1210	32	C87, C88, C89, C90, C91, C92, C105, C106, C107, C108, C109, C110, C119, C120, C121, C122 C135, C136, C137, C138, C139, C140, C153, C154, C155, C156, C157, C158, C167, C168, C169, C170
11	AVX Corporati	06033D1(	Capacitor, Ceramic 1u/25V, 1	Yes		603	18	C8, C9, C76, C80, C84, C94, C98, C102, C112, C116, C124, C128, C132, C142, C146, C150, C160, C164
12	Murata	GRM21BR	Capacitor, Ceramic 470n/50V, 10%, X7R			805	16	C75, C79, C83, C93, C97, C101, C111, C115, C123, C127, C131, C141, C145, C149, C159, C163
13	Taiyo Yuden	UMK325B	Capacitor, Ceramic 10u/50V, 20%, X5R			1210	18	C15, C16, C19, C20, C21, C22, C23, C24, C25, C26, C27, C28, C29, C30, C171, C172, C173, C174
14	Yageo	CC0603KI	Capacitor, Ceramic 100n/50V, 10%, X7R			603	1	C5
15	Murata	GRM21BR	Capacitor, Ceramic 22u/6.3V, 20%, X5R			805	2	C2, C3
16	AVX Corporati	12103C1(	Capacitor, Ceramic, automotive, flexiterm	1		1210	2	C11, C12
17	Panasonic	EEU-FC2A	Capacitor, Elyt 330u/100V, 20%,			ELYT-16	2	C33, C34
18	TDK Corporati	C3225X7S	Capacitor, Ceramic 4u7/100V, 10%, X7S			1210	40	C35, C36, C37, C38, C39, C40, C41, C42, C43, C44, C45, C46, C47, C48, C49, C50, C51, C52, C53 C54, C55, C56, C57, C58, C59, C60, C61, C62, C63, C64, C65, C66, C67, C68, C69, C70, C71, C72, C73, C74
19	Phycomp	2E+11	Resistor, 10k, 50V, 0.1W, 1%			603	3	R5, R9, R14
20	Phycomp	2E+11	Resistor, Resistor 0603 100k, 50V, 0.1W, 1			603	5	R18, R19, R20, R48, R49
21	Phycomp	2E+11	Resistor, 1M, 50V, 0.1W, 1%			603	1	R2
22	Phycomp	2E+11	Resistor, 150k, 50V, 0.1W, 1%			603	1	R8
23	Phycomp	2E+11	Resistor, 1k8, 50V, 0.1W, 1%			603	1	R7
24	Phycomp	2E+11	Resistor, 18k, 50V, 0.1W, 1%			603	18	R26, R28, R56, R57, R58, R65, R66, R67, R72, R73, R80, R81, R82, R89, R90, R91, R96, R97
25	Phycomp	2E+11	Resistor, 2k2, 50V, 0.1W, 1%			603	3	R4, R22, R23
26	Phycomp	2E+11	Resistor, 3k3, 50V, 0.1W, 1%			603	1	R12
27	Phycomp	2E+11	Resistor, 39k, 50V, 0.1W, 1%			603	1	R6
28	Phycomp	2E+11	Resistor, 43k, 50V, 0.1W, 1%			603	2	R27, R29
29	Phycomp	2E+11	Resistor, 470R, 50V, 0.1W, 1%			603	1	R3
30	Phycomp	2E+11	Resistor, 47k, 50V, 0.1W, 1%			603	2	R46, R47
31	Phycomp	2E+11	Resistor, 68k, 50V, 0.1W, 1%			603	1	R13
32	Phycomp	2E+11	Resistor, 0R, 50V, 0.1W, 5%			603	9	R1, R15, R16, R17, R30, R31, R36, R37, R98
33	Phycomp	2E+11	Resistor, 3R3, 200V, 1/4W, 1%			1206	1	R11
34	Phycomp	2E+11	Resistor, 47k, 200V, 1/4W, 1%			1206	8	R38, R39, R40, R41, R42, R43, R44, R45
35	Phycomp	2E+11	Resistor, 0R, 200V, 1/4W, 5%			1206	34	R24, R25, R50, R51, R52, R53, R54, R55, R59, R60, R61, R62, R63, R64, R68, R69, R70, R71, R74 R75, R76, R77, R78, R79, R83, R84, R85, R86, R87, R88, R92, R93, R94, R95
36	Panasonic	ERJ8BWFI	Resistor, 0R015, , 1/2W, 1%			1206	2	R32, R33
37	Panasonic	ERJ-8BWF	Resistor, 0R068, , 1/2W, 1%			1206	1	R10
38	Coilcraft	LPS4018-	Inductor, Power 1u, 1.8A/30%			LPS401	1	L2
39	Coilcraft	MSS1278	Inductor, Power 27u, 2.6A/20%			MSS127	1	L1
40	Supertex	MD1210K	Analog, Driver, MOSFET Dual			QFN-12-	16	U4, U5, U6, U7, U8, U9, U10, U11, U12, U13, U14, U15, U16, U17, U18, U19
41	Keystone	5000	Connector, Testpoint TH			THTP-1	4	TP2, TP3, TP4, TP5
42	Keystone	5001	Connector, Testpoint TH			THTP-1	1	TP1
43	Weidmuller	LSF-SMT-	Connector, PCB Terminal, Push in			LSF-2pir	17	J1, J2, J3, J4, J5, J6, J7, J8, J9, J10, J11, J12, J13, J14, J15, J16, J17
44	Tyco	4-163468	Connector, Header, 2x20 2.54mm BOX			2X20 PI	2	J18, J19
45	International I	15MQ040	Diode, Schottky			SMA	1	D37
46	Diodes Incorp	DFLU140(	Diode, Rectifier, Fast			PowerDI	2	D2, D3
47	STMicroelectr	SM6T39C	Diode, Transient Bidirectional			SMB-BI	1	D36
48	NXP	BAS21	Diode, Small Signal			SOT-23	1	D1
49	Diodes Incorp	ES2G-13-	Diode, Rectifier, Fast			SMB	32	D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14, D15, D16, D17, D18, D19, D20, D21, D22 D23, D24, D25, D26, D27, D28, D29, D30, D31, D32, D33, D34, D35
50	National Semi	LM5116M	PwrMan, Controller, Buck, Synchronous	50		TSSOP-2	1	U1
51	Linear Techno	LT3750EM	PwrMan, Capacitor charger controller			MSOP-1(	2	U2, U3
52	Coilcraft	DA2033-A	Transformer, Power, for LT3750/LT3751, SM			DA2033-	2	T1, T2
53	Vishay	Si7216DN	Transistor, NMOS Dual			PowerPA	1	Q1
54	Vishay	Si4104DY	Transistor, Transistor, NMOS			SO-8-Tra	2	Q2, Q3
55	Supertex	TC6320T(	Transistor, NMOS PMOS, dual			SO-8	16	Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12, Q13, Q14, Q15, Q16, Q17, Q18, Q19
		## NOT M	## NOT MOUI	## NOT MOUI	## NOT MOUNTED PARTS ##			
56			Capacitor, Capacitor, NC Capasitor Chip-080			805	2	C31, C32
57			Capacitor, Ceramic NC NC/, ,			1210	2	C13, C14
58			Resistor, Resistor, NC Resistor Chip-1206 N(			1206	2	R34, R35

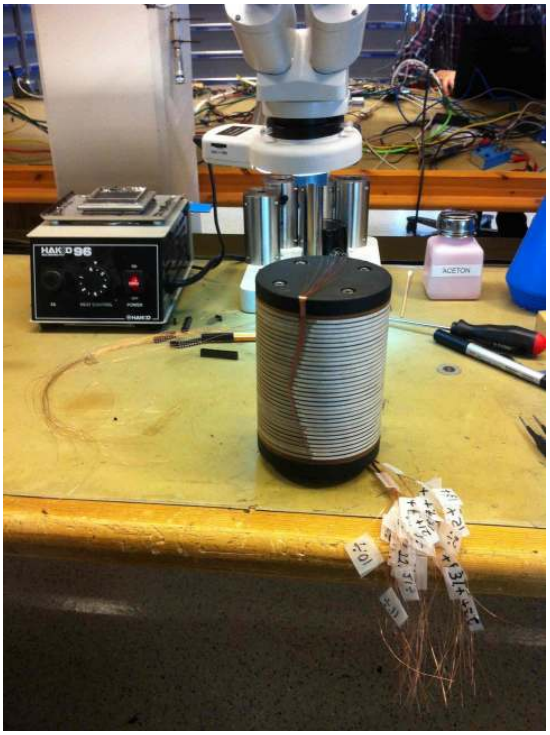
D Bilder fra bygging av transducer

D.1 Bilder fra transducer-bygging

(a) Første keramsike ring



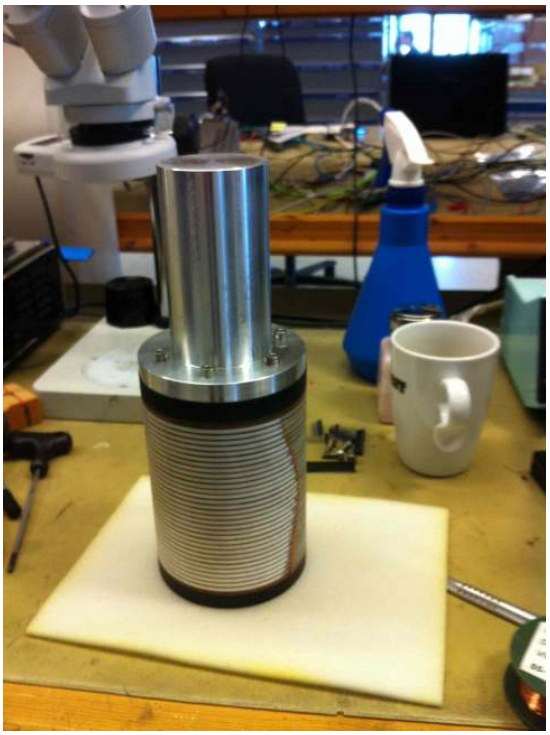
(b) Halvveis



(c) Tilkoblinger på siden

(d) Ferdig med lodding og montering

D.2 Bilder fra støping av transducer



(e) Transducer belagt med første lag epoxy

(f) Etter støping med farget epoxy

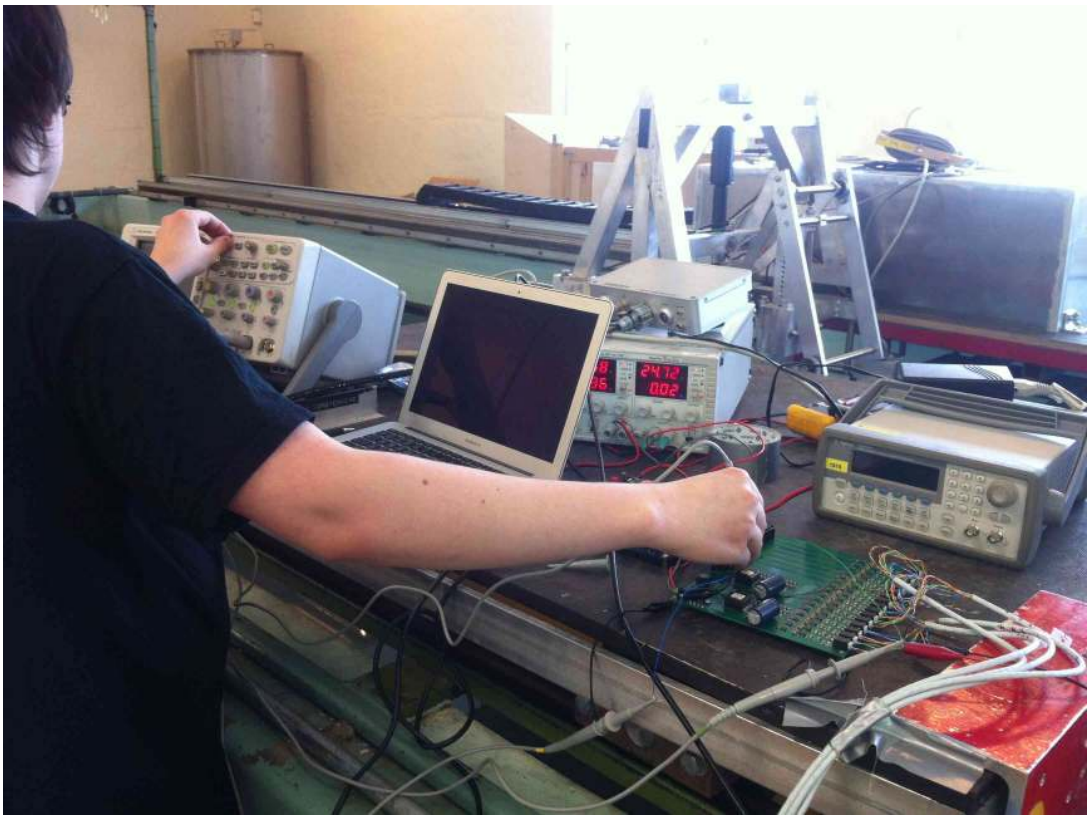
(g) Transducer i dreiebenken

(h) Ferdig transducer klar for test

E Bilder fra test av prototype i lilletanken

(i) Transducer og sonar i testjig

(j) Mål på 6 meters avstand



(k) Styrekort og instrumenter

F Matlab-kode

F.1 Matlab-kode for THD beregning

```
close all
clear all

f = 400e3;
fp = 8e6;
fs = 120e6;
t = 0:1/fs:20/f-1/fs;
x = sin(f*t*2*pi);
xx = (sawtooth(fp*t*2*pi)+0.0667)/0.9334;
y = 2*(x > xx)-1;

Px = sqrt(mean(x.^2))
Py = sqrt(mean(y.^2))

yy = zeros(1,length(y)*10);
for i=0:length(y)-1;
    for j=0:9
        yy(10*i+j+1) = y(i+1);
    end
end
figure;
ff = (0:20:(1200000-20));
plot(ff,abs(fft(yy)));
grid;
axis([0 10e3 0 3e4]);
xlabel('Frequency [kHz]');
ylabel('Amplitude');
print -depsc2 '../figurer/pwm.eps'
```

F.2 Matlab-kode for analyse av ortogonale pulser

```
clear all;
close all;

%load('ofdm.mat') %For bruk med ofdm-pulser

T = 500e-6;

fl = 450e3;
fh = 550e3;

data = csvread('datafiler/scope_center.csv'); %Les inn datafil fra skopet

fs_scope = 1/(data(2,1)-data(1,1))
scope_y = data(:,2);
t = 0:1/fs_scope:T;
t_scope = data(:,1);
zz = 0.*t;
y1 = [chirp(t,fl,T,fh, 'lin', 90) zz];
y2 = [zz chirp(t,fh,T,fl, 'lin', 90)];

figure;
plot(t_scope, scope_y);
grid;

xa = filter(fliplr(y2), 1, scope_y);
xb = filter(fliplr(y1), 1, scope_y);

figure;
```

```
plot(t_scope, xa);
hold on;
plot(t_scope, xb, 'green');
grid

lpf = firpm(100, [0, 0.10, 0.20, 1], [1, 1, 0, 0]);
xxa = filter(lpf, 1, xa.^2);
xxb = filter(lpf, 1, xb.^2);
figure;
plot(t_scope*1e6-1000, xxa);
hold on;
plot(t_scope*1e6-1000, xxb, 'green');
grid
ylabel('Amplitude');
xlabel('Time[us]');
axis([0, 500, 0, 6e4]);
print -depsc2 'matlab_midt.eps'

[maxval, maxind] = max([xxa xxb])
[peakval, channel] = max(maxval);
if channel == 1
   posa = maxind(1)
   [maxvalb, maxindb] = max(xxb(posa-100:posa+100));
   posb = posa - 100 - 1 + maxindb
else
   posb = maxind(2)
   [maxvala, maxinda] = max(xxa(posb-100:posb+100));
   posa = posb - 100 - 1 + maxinda
end
levela = sum(xxa(posa-100:posa+100));
levelb = sum(xxb(posb-100:posb+100));
fprintf('Peak found at %f in channel %d\n', t_scope(maxind(channel)),
    channel);
fprintf('Level A: %f\n', levela);
fprintf('Level B: %f\n', levelb);
ratio = levela/levelb;
fprintf('Ratio: %f / %f dB\n', ratio, 20*log10(ratio));
```

F.3 Matlab-kode for vertikalt lobemønster

```
%Simulering av  lobemnster for egenprodusert transducer
clear all;
close all;

c=1500; % Lydhastighet i vann
f=400e3; % Frekvens
N=16; % Antall elementer
lambda = c/f; %  Blgelengde
deg =(-pi:1/1000:pi);
L=5.2e-3; % Lengde  p hvert element
s=6.2e-3; %CC

%%%%%Uten tidsforsinkelse mellom elementene%%%%%%%%%
x1=(pi*L/lambda).*sin(deg);
x2=(pi*s/lambda).*sin(deg);

p2=(sin(x1)/x1).*(sin(N*x2)./(N.*sin(x2)));
p2 = p2/max(p2);

%%%%%Med forsinkelse (Beamstearing)%%%%%%%%%%%

dt=-83.3e-9; %Tidsforsinkelse mellom elementene

x1_d=(pi*L/lambda).*(sin(deg)-dt*c/s);
x2_d=(pi*s/lambda).*(sin(deg)-dt*c/s);
```

```
p2_d=(sin(x1_d)/x1_d).*(sin(N*x2_d)./(N.*sin(x2_d)));
p2_d = p2_d/max(p2_d);

%%%%%%PLOT%%%%%
figure();
plot(deg*180/pi,20*log10(abs(p2)),deg*180/pi,20*log10(abs(p2_d)),'r');
xlabel ('Angle')
ylabel ('Directivity [dB]')
axis([-7 7 -30 0])
grid

print -depsc2 '../figurer/simulert_lobe_transducer.eps'


%%%%% Mling av vertikalt lobemnster%%%%%%%%%
lille= [-65 -65 -63 -61 -60 -59 -59.4 -57.5 -56.7 -55.5 ...
    -56.0 -55.5 -55.3 -53.5 -51.8 -51.2 -50.9 -51.2 -52.3 ...
    -52.9 -51.8 -48.6 -45.6 -42.7 -40.2 -39.1 -39.4 -38.3 ...
    -35.8 -37.7 -38.9 -40.7 -43.8 -48.5 -54.1 -54.2 -50.0 ...
    -48.6 -49.2 -51.5 -54.2 -58.0 -57.0 -55.0 -54.0 -50.5 ...
    -50.9 -51.7 -53.2 -56.5 -60 -60 -61 -60 -59 -60 -60];
deg2=[-7:0.25:7];

lille = lille-max(lille);

figure();
plot(deg2,lille,deg*180/pi,20*log10(abs(p2)),'r');
xlabel ('Angle')
ylabel ('Directivity [dB]')
axis([-7 7 -30 0])
grid

print -depsc2 '../figurer/Test_Marintek/vertikal_direktivitet.eps'


%%%%% Mling av vertikalt lobemnstermed hanning%%%%%%%%%
lille_vindu_feil=[-65 -63 -62 -58.5 -57 -55.6 -54.2 -54.6 -51.7 -50.8...
    -50 -49.5 -48.4 -48 -48.7 -49.5 -50.3 -52.8 -58 -60 -59 ...
    -53.2 -48.4 -45.7 -44.2 -43.3 -42.8 -42.7 -43.2 -44.1 -46 ...
    -48.2 -52.0 -57.3 -59 -53.4 -51.1 -50 -48.9 -49.0 -49.7 -51 ...
    -52.9 -54.0 -56 -58.8 -62 -62 -64 -64 -65 -64 -64 -62 -62 -63 -61];

lille_vindu=[-63 -63 -63 -63 -63 -63 -63 -63 -63 -63 -63 -62 -63 -60 ...
    -57 -57 -55.8 -51.7 -49.8 -48 -46.3 -44.1 -42.4 -41.3 -40.7...
    -39.8 -39.3 -39.2 -39.5 -38.7 -39.4 -40 -40.9 -42.3 -43.9...
    -45.3 -47 -50.2 -53.3-57.8 -62 -65 -66 -66 -66 -66 -66 ...
    -64 -64 -64 -66 -65 -66 -67 -66 -68 -69 -70];

deg3 = [0:0.25:7];

lille_vindu = lille_vindu-max(lille_vindu);

deg4 = [-5:0.25:2];

% 10x8.3ns delay
lille_83ns=[-61 -60 -57.3 -53.4 -51.2 -48.3 -45.9 -44.3 -42.9 -40.8
   -40.3...
    -40 -39.3 -38.7 -38.4 -38.4 -38.7 -39.3 -39.9 -40.8 -41.9 -43.2
       -45.2...
    -46.8 -50.9 -53.8 -58 -62 -63];

lille_83ns = lille_83ns-max(lille_83ns);

% 9.9 trinn defokus
lille_defokus=[-66 -65 -65 -64 -61 -60 -57 -56 -54.5 -53.5 -52.2 -51.6...
```

```
    -50.7 -50.0 -49 -48.2 -47 -46.8 -45.5 -45.2 -44.5 -43.6 -43.2 -43.2...
    -42.3 -41.7 -41.3 -40.9 -40.7 -40.7 -40.6 -40.9 -41.3 -41.5 -42.0...
    -42.5 -43.3 -44.0 -45.2 -46.3 -46.9 -48.2 -50.2 -51.6 -53 -53.9...
    -55.5 -56.7 -58 -58 -60 -60 -59 -61 -62 -64 -66];

lille_defokus = lille_defokus-max(lille_defokus);

figure();
plot(deg2,lille,'blue',deg2,lille_vindu,'green',deg4,lille_83ns,'black',
   deg2,lille_defokus,'red');
xlabel ('Angle')
ylabel ('Directivity [dB]')
axis([-7 7 -30 0])
grid

print -depsc2 '../figurer/Test_Marintek/funksjonstest_direktivitet.eps'

figure();
plot(deg4,lille_83ns,'black',deg*180/pi,20*log10(abs(p2_d)),'red')
xlabel ('Angle')
ylabel ('Directivity [dB]')
axis([-7 7 -30 0])
grid
```

F.4 Matlab-kode for analyse posisjonsbestemmelse

```
% Test av prototype i basseng 2013-04-24
close all;
clear all;

offset6m=2.3;
offset3m=3.2;


%% Pulslengde 100us, 40x8.33ns delay, hanningvindu, 9,9trinn defokus.
   Sonarbell @3m
diff=[-19.6 -20.3 -13.7 -5.5 -2.5 9.9 17.3 32.6 28];
deg=(-5:1:3);

figure();
plot(deg,diff);
ylabel('Forskjell mellom pulser [dB]')
xlabel('Vinkel')

a=[325.8 334.6 540.2 444.5 601.3 1101.1 1969 2796.2 2906.1];
b=[4303 3453.5 2618.4 839.9 547.9 352.3 269.7 65.9 115.2];

figure();
plot(deg,20*log10(a),deg,20*log10(b));


%% Pulslengde 100us, 50x8.33ns delay, hanningvindu, 25trinn defocus
   Sonarbell @3m
diff2=[-7.2 -9 -9.1 -7.4 -5.4 -0.3 -0.3 3.6 7.6 4.4 3.1];
deg2=(-8:1:2);

figure()
plot(deg2,diff2);
grid;

a2=[438.3 547.1 452.1 556.4 649 1194.5 1194.5 542.6 656.5 1128.8 655.3];
b2=[1004.9 1548.9 1286.2 1300.2 1208 1235.1 1235.1 358.4 274.2 682.6
   460.2];

figure();
```

```
plot(deg2,a2,deg2,b2);
grid;

%% Pulslengde 500us, 40x8.33ns delay, hanningvindu, 10trinn defokus.
    Sonarbell @3m
a3=[3200 4010 2502 9839 13841 19072 36788.4 71114 103486.6 89575];
b3=[83228 123552 90202 66661 30390 15694 8156 6210 2463 8056];
deg3=(-6:1:3);

figure()
plot(deg3,20*log10(a3),deg3,20*log10(b3));
diff3=20*log10(a3./b3);

figure()
plot(deg3,diff3);
grid;

%% Pulslengde 500us, 40x8.33ns delay, hanningvindu, 20trinn defokus.
    Sonarbell @3m

%a4_feil=[8560 7837 13663 21196 34151 29193 28443 59746 52105 59416 45846];
%b4_feil=[58696 39753 48221 49268 56422 43919 23160 38209 20551 13141
    6239];
%diff4_feil=20*log10(a4_feil./b4_feil);
%deg4_feil=(-7:1:3);

%figure()
%plot(deg4_feil,20*log10(a4_feil),deg4_feil,20*log10(b4_feil));

%figure()
%plot(deg4_feil,diff4_feil)

a4=[1053 2628 2463 5515 4483 9660 9907 10181 16161 13499 11599 13445 9441
    8473 2308 3116];
b4=[9563 10238 13573 16599 13932 18628 13403 8529 8239 4004 2957 2318 1203
    437 218 332];
diff4=20*log10(a4./b4);
deg4=(-8:1:7);

figure()
plot(deg4,20*log10(a4),deg4,20*log10(b4));
grid;

figure()
plot(deg4,diff4)
grid;

%% Pulslengde 500us, 40x8.33ns delay, hanningvindu, 25trinn defokus.
    Sonarbell @3m

a5=[2267 9739 2109 7929 12875 6391 9113 3927 2924 4019 6449 3477 5422 2884
    3348 2358];
b5=[7831 21322 5719 14859 20517 9680 10934 2648 4103 4016 2307 1464 2170
    610 449 266];
diff5=20*log10(a5./b5);
deg5=(-8:1:7);

%figure()
%plot(deg5,20*log10(a5),deg5,20*log10(b5));

%figure()
%plot(deg5,diff5)


%% Pulslengde 500us, 40x8.33ns delay, hanningvindu, 20trinn defokus.
    Sonarbell @6m
```

```
a6=[26646 35011 33135 36069 21821 63360 46931 54514 81046];
b6=[60690 52867 60294 38419 53623 22100 16841 47363 18866];
diff6=20*log10(a6./b6);
deg6=(-8:1:0);

%figure()
%plot(deg6,20*log10(a6),deg6,20*log10(b6));

%figure()
%plot(deg6,diff6)

%% Pulslengde 500us, 15x8.33ns delay, hanningvindu, 10trinn defokus.
    Sonarbell @6m
a7=[29934 29969 43110 67839 71769 97035 106469 130912 156827 146879
    133639];
b7=[77776 108395 106714 123777 185804 117767 69555 78316 32509 19849
    232992];
diff7=20*log10(a7./b7);
deg7=(-8:1:2);

%figure()
%plot(deg7,20*log10(a7),deg7,20*log10(b7));

%figure()
%plot(deg7,diff7)

%% Pulslengde 500us, 15x8.33ns delay, hanningvindu, 10trinn defokus.
     Stlkule 5cm diameter @6m
a8=[10890 12270 11892 31949 35361 40217 60451 55472 55281 65913 48073 52593
     57639];
b8=[44654 50435 41356 74284 61254 59860 50871 30210 26584 21224 16034 13604
     15918];
diff8=20*log10(a8./b8);
deg8=(-8:1:4)+offset6m;

figure()
subplot(2,1,1)
plot(deg8,20*log10(a8),deg8,20*log10(b8));
grid;
ylabel('Mlstyrke [dB]')
xlabel('Vinkel')
subplot(2,1,2)
plot(deg8,diff8)
grid;
ylabel('Forskjell mellom pulser [dB]')
xlabel('Vinkel')

print -depsc2 '../figurer/Test_Marintek/target6m.eps'

%% Pulslengde 500us, 40x8.33ns delay, hanningvindu, 20trinn defokus.
     Stlkule 5cm diameter @3m
a9=[266 829 818 1378 3442 5089 7614 7358 11280 10579 11579 12378 11929
    10808 7827 4790 2162];
b9=[88526 7687 9956 9939 11310 13953 12284 6484 7851 5226 4647 4986 2526
    1075 592 169 135];
diff9=20*log10(a9./b9);
deg9=(-10:1:6)+offset3m;

figure()
subplot(2,1,1)
plot(deg9,20*log10(a9),deg9,20*log10(b9));
grid;
ylabel('Mlstyrke [dB]')
xlabel('Vinkel')
subplot(2,1,2)
plot(deg9,diff9)
```

```
grid;
ylabel('Forskjell mellom pulser [dB]')
xlabel('Vinkel')

print -depsc2 '../figurer/Test_Marintek/target3m.eps'
```

G VHDL-kode

G.1 VHDL-kode for PWM-modulen

```
-- PWM-modul laget av Bjørnar Leithe og Stian Sønderland i forbindelse med
    hovedoppgave våren 2013

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity pwm_generator is
  generic (
    AUTO_CLOCK_CLOCK_RATE : string := "-1";
    BUFFER_LENGTH      : integer := 90
  );
  port (
    avs_s0_address       : in  std_logic_vector(6 downto 0)  := (others =>
        '0');
    avs_s0_writedata     : in  std_logic_vector(15 downto 0) := (others =>
        '0');
    avs_s0_write         : in  std_logic             := '0';
    clk                  : in  std_logic             := '0';
    reset                : in  std_logic             := '0';
    avm_m0_read          : out std_logic;
    avm_m0_waitrequest   : in  std_logic             := '0';
    avm_m0_readdata      : in  std_logic_vector(7 downto 0) := (others =>
        '0');
    avm_m0_readdatavalid : in  std_logic             := '0';
    avm_m0_address       : out std_logic_vector(31 downto 0);
    trigger              : in  std_logic             := '0';
    pwm_active           : out std_logic;
    pwm_ready            : out std_logic;
    pwm_buffer_load      : out std_logic;
    pwm_h                : out std_logic_vector(15 downto 0);
    pwm_l                : out std_logic_vector(15 downto 0);
    pwm_oe               : out std_logic_vector(15 downto 0)
  );
end entity pwm_generator;

architecture rtl of pwm_generator is

type signed8_16 is array (15 downto 0) of signed(7 downto 0);
type signed16_16 is array (15 downto 0) of signed(15 downto 0);
type unsigned8_16 is array (15 downto 0) of unsigned(7 downto 0);
type signed4_16 is array (15 downto 0) of signed(3 downto 0);
type signed8_buffer is array (BUFFER_LENGTH-1 downto 0) of signed(7 downto
    0);
type boolean16 is array (15 downto 0) of boolean;
type active_pulse_type is (a, b);
type pwm_state_type is (s_start_reload, s_reloading, s_ready, s_active,
    s_switch_pulse);

signal current_state, next_state: pwm_state_type;
signal active_pulse: active_pulse_type;
signal buffer_full: boolean;
signal data_valid: boolean;
signal must_reload: boolean;
signal done: boolean;

signal delays_a, delays_b, delays: signed16_16;
signal amplitudes_a, amplitudes_b, amplitudes: unsigned8_16;
signal start_address_a, start_address_b, start_address: unsigned(31 downto
    0);
signal mode: std_logic_vector(15 downto 0);
```

```
signal duration: signed(15 downto 0);
signal counters, counters_d, counters_d_d, counters_d_d_d: signed4_16;
signal buffer_pos_d: signed8_16;
signal values_d_d: signed8_16;
signal values_d_d_d: signed4_16;
signal active_d, active_d_d, active_d_d_d: boolean16;
signal buffer_fulls: boolean16;
signal positions: signed16_16;
signal pwm_buffer, pwm_buffer_d: signed8_buffer;
signal pwm_buffer_offset: signed(15 downto 0);
signal data_read: std_logic_vector(7 downto 0);
signal next_read_address: unsigned(31 downto 0);
signal buffer_read: std_logic;

begin

  -- Konfigurasjonsregistre på avs_s0 for delay, amplitude, varighet,
     startadresse, osv.
  process (clk, reset)
  begin
    if (reset = '1') then
      -- Initialiser til 0 ved reset
      delays_a <= (others => (others => '0'));
      delays_b <= (others => (others => '0'));
      amplitudes_a <= (others => (others => '0'));
      amplitudes_b <= (others => (others => '0'));
      start_address_a <= (others => '0');
      start_address_b <= (others => '0');
      duration <= (others => '0');
      must_reload <= false;
    elsif rising_edge(clk) then
      if (avs_s0_write = '1') then
        must_reload <= true;
        if (avs_s0_address(6 downto 4) = "000") then
          delays_a(to_integer(unsigned(avs_s0_address(3 downto 0)))) <=
              signed(avs_s0_writedata);
        elsif (avs_s0_address(6 downto 4) = "001") then
          delays_b(to_integer(unsigned(avs_s0_address(3 downto 0)))) <=
              signed(avs_s0_writedata);
        elsif (avs_s0_address(6 downto 4) = "010") then
          amplitudes_a(to_integer(unsigned(avs_s0_address(3 downto 0)))) <=
               unsigned(avs_s0_writedata(7 downto 0));
        elsif (avs_s0_address(6 downto 4) = "011") then
          amplitudes_b(to_integer(unsigned(avs_s0_address(3 downto 0)))) <=
               unsigned(avs_s0_writedata(7 downto 0));
        elsif (avs_s0_address(6 downto 4) = "100") then
          duration <= signed(avs_s0_writedata);
        elsif (avs_s0_address(6 downto 4) = "101") then
          if (avs_s0_address(3 downto 0) = "0000") then
            start_address_a(15 downto 0) <= unsigned(avs_s0_writedata);
          elsif (avs_s0_address(3 downto 0) = "0001") then
            start_address_a(31 downto 16) <= unsigned(avs_s0_writedata);
          elsif (avs_s0_address(3 downto 0) = "0010") then
            start_address_b(15 downto 0) <= unsigned(avs_s0_writedata);
          elsif (avs_s0_address(3 downto 0) = "0011") then
            start_address_b(31 downto 16) <= unsigned(avs_s0_writedata);
          elsif (avs_s0_address(3 downto 0) = "0100") then
            mode <= avs_s0_writedata;
          else
            -- Ubrukt adresseområde
          end if;
        else
          -- Ubrukt adresseområde
        end if;
      else
        must_reload <= false;
```

```
    end if;
  end if;
end process;

-- PWM-register
process (clk, reset)
begin
  if (reset = '1') then
    -- Initialiser til 0 ved reset
    counters <= (others => to_signed(0, 4));
    positions <= (others => to_signed(0, 16));
  elsif rising_edge(clk) then
    -- To tellere for hver kanal
    for i in 0 to 15 loop
      if (current_state /= s_active) then
        counters(i) <= -delays(i)(3 downto 0) - 7;
        positions(i) <= shift_right(-delays(i),4);
      elsif counters(i) >= 7 then
        counters(i) <= to_signed(-7, 4);
        positions(i) <= positions(i) + to_signed(1, 16);
      else
        counters(i) <= counters(i) + 1;
        positions(i) <= positions(i);
      end if;
    end loop;
  end if;
end process;

-- Utgangsregistre
process (clk, reset)
  -- Midlertidige variabler for mellomlagring av data ved konvertering
     mellom ulike lengder og formater
  variable temp1: signed(8 downto 0);
  variable temp2: signed(16 downto 0);
  variable temp3: signed(15 downto 0);
begin
  if (reset = '1') then
    pwm_h <= (others => '0');
    pwm_l <= (others => '0');
    pwm_oe <= (others => '1');
    active_d <= (others => false);
    active_d_d <= (others => false);
    active_d_d_d <= (others => false);
    counters_d <= (others => (others => '0'));
    counters_d_d <= (others => (others => '0'));
    counters_d_d_d <= (others => (others => '0'));
    buffer_pos_d <= (others => (others => '0'));
    values_d_d <= (others => (others => '0'));
    values_d_d_d <= (others => (others => '0'));
  elsif rising_edge(clk) then
    counters_d <= counters;
    counters_d_d <= counters_d;
    counters_d_d_d <= counters_d_d;
    active_d_d <= active_d;
    active_d_d_d <= active_d_d;
    pwm_buffer_d <= pwm_buffer;
    for i in 0 to 15 loop
      -- Alle verdier bufres i 3 cycler for å få timing til å gå opp
      -- Gjør kalkulering på amplitudeverdien under veis mellom hvert
         buffer
      temp3 := pwm_buffer_offset-positions(i);
      buffer_pos_d(i) <= temp3(7 downto 0);
      values_d_d(i) <= pwm_buffer_d(to_integer(buffer_pos_d(i)));
      temp1(8) := '0';
      temp1(7 downto 0) := signed(amplitudes(i));
      temp2 := values_d_d(i)*temp1;
```

```
      values_d_d_d(i) <= temp2(15 downto 12);

      if (current_state = s_active) and (positions(i) >= 0) and (
          positions(i) < duration) then
        active_d(i) <= true;
      else
        active_d(i) <= false;
      end if;

      -- Set outputs based on buffered values from last cycle.
      if not active_d_d_d(i) then
        pwm_h(i) <= '0';
        pwm_l(i) <= '0';
        pwm_oe(i) <= '0';
      elsif counters_d_d_d(i) <= values_d_d_d(i) then
        pwm_h(i) <= '0';
        pwm_l(i) <= '0';
        pwm_oe(i) <= '1';
      else
        pwm_h(i) <= '1';
        pwm_l(i) <= '1';
        pwm_oe(i) <= '1';
      end if;
    end loop;
  end if;
end process;

-- Prosess som sjekker om buffer skal fylles med nye data fra RAM
process (clk, reset)
begin
  if reset = '1' then
    buffer_fulls <= (others => false);
  elsif rising_edge(clk) then
    if current_state = s_start_reload then
      buffer_fulls <= (others => false);
    else
      for i in 0 to 15 loop
        buffer_fulls(i) <= (to_integer(pwm_buffer_offset)-to_integer(
            positions(i)) > BUFFER_LENGTH-10);
      end loop;
    end if;
  end if;
end process;

process(buffer_fulls)
  variable var_buffer_full: boolean;
begin
  var_buffer_full := false;
  for i in 0 to 15 loop
    var_buffer_full := var_buffer_full or buffer_fulls(i);
  end loop;
  buffer_full <= var_buffer_full;
end process;

-- Kombinatorisk prosess som sjekker om puls er ferdigsendt
process (positions, duration)
  variable var_done: boolean;
begin
  var_done := true;
  for i in 0 to 15 loop
    var_done := var_done and (positions(i) > duration);
  end loop;
  done <= var_done;
end process;

-- Tilstandsmaskin
```

```
process (clk, reset)
begin
  if reset = '1' then
    current_state <= s_reloading;
  elsif rising_edge(clk) then
    current_state <= next_state;
  end if;
end process;

process (current_state, buffer_full, trigger, must_reload, done,
    active_pulse)
begin
  case current_state is
    when s_start_reload =>
      pwm_active <= '0';
      pwm_ready <= '0';
      if must_reload then
        next_state <= s_start_reload;
      else
        next_state <= s_reloading;
      end if;
    when s_reloading =>
      pwm_active <= '0';
      pwm_ready <= '0';
      if must_reload then
        next_state <= s_start_reload;
      elsif buffer_full then
        next_state <= s_ready;
      else
        next_state <= s_reloading;
      end if;
    when s_ready =>
      pwm_active <= '0';
      pwm_ready <= '1';
      if must_reload then
        next_state <= s_start_reload;
      elsif (trigger = '1' or active_pulse = b) then
        next_state <= s_active;
      else
        next_state <= s_ready;
      end if;
    when s_active =>
      pwm_active <= '1';
      pwm_ready <= '0';
      if done then
        next_state <= s_switch_pulse;
      else
        next_state <= s_active;
      end if;
    when s_switch_pulse =>
      pwm_active <= '0';
      pwm_ready <= '0';
      next_state <= s_start_reload;
  end case;
end process;

-- Velg mellom de to settene med parametre ut i fra hvilken puls som er
    aktiv
process (active_pulse, delays_a, delays_b, amplitudes_a, amplitudes_b,
    start_address_a, start_address_b)
begin
  if (active_pulse = a) then
    delays <= delays_a;
    amplitudes <= amplitudes_a;
    start_address <= start_address_a;
  else
```

```
    delays <= delays_b;
    amplitudes <= amplitudes_b;
    start_address <= start_address_b;
  end if;
end process;

process (clk, reset)
begin
  if (reset = '1') then
    active_pulse <= a;
  elsif rising_edge(clk) then
    if (mode = "0000000000000000") then
      -- Kun puls A
      active_pulse <= a;
    else
      -- Puls A først deretter puls B
      if (current_state = s_switch_pulse) then
        if (active_pulse = b) then
          active_pulse <= a;
        else
          active_pulse <= b;
        end if;
      end if;
    end if;
  end if;
end process;

-- Buffer for verdier som leses fra RAM
process (clk, reset)
begin
  if (reset = '1') then
    pwm_buffer_offset <= (others => '0');
    pwm_buffer <= (others => (others => '0'));
    next_read_address <= (others => '0');
    data_read <= (others => '0');
    data_valid <= false;
    buffer_read <= '0';
    pwm_buffer_load <= '0';
  elsif rising_edge(clk) then

    data_read <= avm_m0_readdata;

    if (avm_m0_readdatavalid = '1') then
      data_valid <= true;
    else
      data_valid <= false;
    end if;

    if data_valid then
      for i in 1 to BUFFER_LENGTH-1 loop
        pwm_buffer(i) <= pwm_buffer(i-1);
      end loop;
      pwm_buffer(0) <= signed(data_read);
    end if;

    if current_state = s_start_reload then
      next_read_address <= start_address;
      pwm_buffer_offset <= to_signed(-1, 16);
      -- pwm_buffer <= (others => (others => '0'));
      buffer_read <= '0';
      pwm_buffer_load <= '1';
    else
      if data_valid then
        pwm_buffer_offset <= pwm_buffer_offset + 1;
      end if;
      if not buffer_full then
```

```
          buffer_read <= '1';
          pwm_buffer_load <= '1';
        else
          buffer_read <= '0';
          pwm_buffer_load <= '0';
        end if;

        if (avm_m0_waitrequest = '0' and buffer_read = '1') then
          next_read_address <= next_read_address + 1;
        end if;
      end if;
    end if;
  end process;

  avm_m0_address <= std_logic_vector(next_read_address);
  avm_m0_read <= buffer_read;

end architecture rtl;
```

G.2 VHDL-kode for overordnet struktur

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.numeric_std.all;
library altera;
use altera.altera_syn_attributes.all;

entity beamformer is
  port
  (
-- {ALTERA_IO_BEGIN} DO NOT REMOVE THIS LINE!

--    CLOCK_50 : in std_logic;
--    EXT_CLOCK : in std_logic;
--    PS2_CLK : in std_logic;
--    PS2_DAT : in std_logic;
    UART_RXD : in std_logic;
    UART_TXD : out std_logic;
--    TDI : in std_logic;
--    TCS : in std_logic;
--    TCK : in std_logic;
--    TDO : in std_logic;
--    VGA_HS : in std_logic;
--    VGA_VS : in std_logic;
--    I2C_SCLK : in std_logic;
--    I2C_SDAT : in std_logic;
--    AUD_ADCLRCK : in std_logic;
--    AUD_ADCDAT : in std_logic;
--    AUD_DACLRCK : in std_logic;
--    AUD_DACDAT : in std_logic;
--    AUD_XCK : in std_logic;
--    AUD_BCLK : in std_logic;
--    DRAM_CAS_N : in std_logic;
--    DRAM_CKE : in std_logic;
--    DRAM_CLK : in std_logic;
--    DRAM_CS_N : in std_logic;
--    DRAM_RAS_N : in std_logic;
--    DRAM_WE_N : in std_logic;
--    FL_OE_N : in std_logic;
--    FL_RST_N : in std_logic;
--    FL_WE_N : in std_logic;
    SRAM_CE_N : out std_logic;
    SRAM_BE_N : out std_logic_vector(1 downto 0);
    SRAM_OE_N : out std_logic;
    SRAM_WE_N : out std_logic;
```

```
    SRAM_ADDR : out std_logic_vector(17 downto 0);
    SRAM_DQ : inout std_logic_vector(15 downto 0);
--    DRAM_BA : in std_logic_vector(0 to 1);
    GPIO_0 : inout std_logic_vector(0 to 35);
    GPIO_1 : out std_logic_vector(0 to 35);
    SW : in std_logic_vector(0 to 9);
    HEX0 : out std_logic_vector(6 downto 0);
    HEX1 : out std_logic_vector(6 downto 0);
    HEX2 : out std_logic_vector(6 downto 0);
    HEX3 : out std_logic_vector(6 downto 0);
    KEY : in std_logic_vector(0 to 3);
    LEDR : out std_logic_vector(0 to 9);
    LEDG : out std_logic_vector(0 to 7);
--    CLOCK_27 : in std_logic_vector(0 to 1);
    CLOCK_24 : in std_logic_vector(0 to 1)
--    VGA_R : in std_logic_vector(0 to 3);
--    VGA_G : in std_logic_vector(0 to 3);
--    VGA_B : in std_logic_vector(0 to 3);
--    DRAM_ADDR : in std_logic_vector(0 to 11);
--    DRAM_DQ : in std_logic_vector(0 to 15);
--    DRAM_DQM : in std_logic_vector(0 to 1);
--    FL_ADDR : in std_logic_vector(0 to 21);
--    FL_DQ : in std_logic_vector(0 to 7);
-- {ALTERA_IO_END} DO NOT REMOVE THIS LINE!

  );

-- {ALTERA_ATTRIBUTE_BEGIN} DO NOT REMOVE THIS LINE!
-- {ALTERA_ATTRIBUTE_END} DO NOT REMOVE THIS LINE!
end beamformer;

architecture ppl_type of beamformer is

-- {ALTERA_COMPONENTS_BEGIN} DO NOT REMOVE THIS LINE!

    component beamformer_system is
        port (
            clk_clk                     : in    std_logic                     := 'X
                ';
            io_output_export            : out   std_logic_vector(31 downto
                0);
            io_input_export             : in    std_logic_vector(31 downto
                0) := (others => 'X');
            pwm_signals_trigger         : in    std_logic                     := 'X
                ';
            pwm_signals_pwm_h           : out   std_logic_vector(15 downto
                0);
            pwm_signals_pwm_l           : out   std_logic_vector(15 downto
                0);
            pwm_signals_pwm_oe          : out   std_logic_vector(15 downto
                0);
            pwm_signals_pwm_active      : out   std_logic;
            pwm_signals_pwm_ready       : out   std_logic;
            pwm_signals_pwm_buffer_load: out   std_logic;
            sram_address                : out   std_logic_vector(17 downto
                0);
            sram_outputenable_n         : out   std_logic_vector(0 downto 0)
                ;
            sram_byteenable_n           : out   std_logic_vector(1 downto 0)
                ;
            sram_write_n                : out   std_logic_vector(0 downto 0)
                ;
            sram_data                   : inout std_logic_vector(15 downto
                0) := (others => 'X');
            sram_chipselect_n           : out   std_logic_vector(0 downto 0)
                ;
```

```
            uart_rxd                    : in    std_logic                 :=
                'X';
            uart_txd                    : out   std_logic;
            seven_seg0                  : out   std_logic_vector(6 downto 0)
                  := (others => 'X');
            seven_seg1                  : out   std_logic_vector(6 downto 0)
                  := (others => 'X');
            seven_seg2                  : out   std_logic_vector(6 downto 0)
                  := (others => 'X');
            seven_seg3                  : out   std_logic_vector(6 downto 0)
                  := (others => 'X')
  );
    end component beamformer_system;

   component pll120 is
    PORT
    (
      inclk0    : IN STD_LOGIC  := '0';
      c0        : OUT STD_LOGIC ;
      c2        : out std_logic ;
      locked    : OUT STD_LOGIC
    );
   end component pll120;
-- {ALTERA_COMPONENTS_END} DO NOT REMOVE THIS LINE!

  signal clock_120: std_logic; -- Intern 120MHz klokke fra PLL
  signal ext_trig: std_logic;
  signal charge_counter: unsigned(31 downto 0);
  signal pwm_active, trigger: std_logic; -- Kontrollsignaler for PWM-modul
  signal pll_lock: std_logic;
  signal pwm_ready: std_logic;
  signal pwm_buffer_load: std_logic;
  signal charge, done_n, done_p: std_logic;
  signal cpu_out: std_logic_vector(31 downto 0);
  signal cpu_in: std_logic_vector(31 downto 0);
  signal trig_detect_a, trig_detect_b: std_logic_vector(10 downto 0);
  signal pwm_h, pwm_l, pwm_oe: std_logic_vector(15 downto 0);
begin
-- {ALTERA_INSTANTIATION_BEGIN} DO NOT REMOVE THIS LINE!

   u0 : component beamformer_system
        port map (
            clk_clk                         => clock_120,
            io_output_export                => cpu_out,
            io_input_export                 => cpu_in,
            pwm_signals_trigger             => trigger,
            pwm_signals_pwm_active          => pwm_active,
            pwm_signals_pwm_buffer_load     => pwm_buffer_load,
            pwm_signals_pwm_ready           => pwm_ready,
            pwm_signals_pwm_h               => pwm_h,
            pwm_signals_pwm_l               => pwm_l,
            pwm_signals_pwm_oe              => pwm_oe,
            sram_byteenable_n               => SRAM_BE_N,
            sram_address                    => SRAM_ADDR,
            sram_outputenable_n(0)          => SRAM_OE_N,
            sram_write_n(0)                 => SRAM_WE_N,
            sram_data                       => SRAM_DQ,
            sram_chipselect_n(0)            => SRAM_CE_N,
            uart_rxd                        => UART_RXD,
            uart_txd                        => UART_TXD,
            seven_seg0                      => HEX0,
            seven_seg1                      => HEX1,
            seven_seg2                      => HEX2,
            seven_seg3                      => HEX3
  );
```

```
u1: component pll120
  port map (
    inclk0 => CLOCK_24(0),
    c0 => clock_120,
    locked => pll_lock
  );

LEDG(0 to 7) <= cpu_out(7 downto 0);
LEDR(0) <= pll_lock;
LEDR(1) <= pwm_active;
LEDR(2) <= pwm_ready;
LEDR(3) <= pwm_buffer_load;
LEDR(4 to 9) <= (others => '0');

-- Mapping av I/O-signaler i henhold til pin-out på prototypekortet
GPIO_1(0)  <= pwm_oe(0);
GPIO_1(1)  <= pwm_h(0);
GPIO_1(2)  <= pwm_l(0);
GPIO_1(3)  <= pwm_oe(1);
GPIO_1(4)  <= pwm_h(1);
GPIO_1(5)  <= pwm_l(1);
GPIO_1(6)  <= pwm_oe(2);
GPIO_1(7)  <= pwm_h(2);
GPIO_1(8)  <= pwm_l(2);
GPIO_1(9)  <= pwm_oe(3);
GPIO_1(10) <= pwm_h(3);
GPIO_1(11) <= pwm_l(3);
GPIO_1(12) <= pwm_oe(4);
GPIO_1(13) <= pwm_h(4);
GPIO_1(14) <= pwm_l(4);
GPIO_1(15) <= pwm_oe(5);
GPIO_1(16) <= pwm_h(5);
GPIO_1(17) <= pwm_l(5);
GPIO_1(18) <= pwm_oe(6);
GPIO_1(19) <= pwm_h(6);
GPIO_1(20) <= pwm_l(6);
GPIO_1(21) <= pwm_oe(7);
GPIO_1(22) <= pwm_h(7);
GPIO_1(23) <= pwm_l(7);
GPIO_1(24) <= pwm_oe(8);
GPIO_1(25) <= pwm_h(8);
GPIO_1(26) <= pwm_l(8);
GPIO_1(27) <= pwm_oe(9);
GPIO_1(28) <= pwm_h(9);
GPIO_1(29) <= pwm_l(9);
GPIO_1(30) <= pwm_oe(10);
GPIO_1(31) <= pwm_h(10);
GPIO_1(32) <= pwm_l(10);
GPIO_1(33) <= pwm_oe(11);
GPIO_1(34) <= pwm_h(11);
GPIO_1(35) <= pwm_l(11);
GPIO_0(0)  <= pwm_oe(12);
GPIO_0(1)  <= pwm_h(12);
GPIO_0(2)  <= pwm_l(12);
GPIO_0(3)  <= pwm_oe(13);
GPIO_0(4)  <= pwm_h(13);
GPIO_0(5)  <= pwm_l(13);
GPIO_0(6)  <= pwm_oe(14);
GPIO_0(7)  <= pwm_h(14);
GPIO_0(8)  <= pwm_l(14);
GPIO_0(9)  <= pwm_oe(15);
GPIO_0(10) <= pwm_h(15);
GPIO_0(11) <= pwm_l(15);
GPIO_0(12) <= charge;
done_p <= GPIO_0(13);
done_n <= GPIO_0(14);
```

```
  GPIO_0(15) <= pwm_active;
  ext_trig <= GPIO_0(16);
--  GPIO_0(17 to 35) <= (others => '0');

  cpu_in(9 downto 0) <= SW;
  cpu_in(13 downto 10) <= (not KEY);

  -- Blokk som styrer trigger og lading av kondensatorbank ut i fra knapper
      og ekstern trigger-inngang
  process(clock_120)
  begin
    if rising_edge(clock_120) then
      for i in 1 to 10 loop
        trig_detect_a(i) <= trig_detect_a(i-1);
        trig_detect_b(i) <= trig_detect_b(i-1);
      end loop;
      trig_detect_a(0) <= KEY(0);
      trig_detect_b(0) <= (ext_trig and (not charge or (not done_p and not
          done_n)));
      if (trig_detect_a = "11111111110" or trig_detect_b = "00000000001" or
          (charge_counter = 0 and SW(6) = '1')) then
        trigger <= '1';
      else
        trigger <= '0';
      end if;

      if trigger = '1' then
        charge_counter <= to_unsigned(1, 32);
      elsif charge_counter < 12000000 then
        charge_counter <= charge_counter + 1;
      else
        charge_counter <= to_unsigned(0, 32);
      end if;

      if charge_counter < 6000000 then -- 50ms charge time
        charge <= '0';
      else
        charge <= '1';
      end if;
    end if;
  end process;

-- {ALTERA_INSTANTIATION_END} DO NOT REMOVE THIS LINE!

end;
```

H C-kode

H.1 C-kode for programmet

```
#include <system.h>
#include <stdint.h>
#include <priv/alt_busy_sleep.h>
#include <io.h>
#include <math.h>
#include <unistd.h>
#include "sys/alt_stdio.h"
#include "pwm_generator.h"
#include "pio.h"

#define SEVEN_SEG ( (volatile uint8_t *) (SEVEN_SEG_BASE))

void display(value, digits) {
    int i;
    for(i=0;i<digits;i++) {
        SEVEN_SEG[3-i] = value%10;
        value /= 10;
    }
}

float fastsqrtf( float number ) {
    int32_t i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                       // evil floating point bit
       level hacking
    i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can
       be removed
    return y*number;
}

float fastsinf( float x) {
    while (x < -100.F*M_PI) x += 100.F*M_PI;
    while (x < -M_PI) x += M_TWOPI;
    while (x > 100.F*M_PI) x -= 100.F*M_PI;
    while (x > M_PI) x -= M_TWOPI;
    const float B = 4.F/M_PI;
    const float C = -4.F/(M_PI*M_PI);

    float y = B * x + C * x * (x<0?-x:x);

    //const float P = 0.225;
    //y = P * (y * abs(y) - y) + y;   // Q * y + P * y * abs(y)

    return y;
}

int8_t pulse[2][16000] __attribute__ ((section (".bulk")));
int main()
{
  int32_t i;
  uint32_t input;
  int16_t f0 = 450, f1 = 550, duration = 1000, phasing = 0, d, firststep =
     1, window = 0, focus = 0;
```

```
int16_t delays[16], mindelay;
float ffocus, fchannel;
float ff0, ff1, f, t;
float fduration;

for (i=0;i<4;i++) {
    SEVEN_SEG[i] = i;
}

alt_putstr("Hello from Nios II!\n");
alt_printf("Ready! Address: %x\n", pulse);

/* Event loop never exits. */
while (1) {
  input = PIO_IN;
  //alt_printf("Input: %x\n", input);
  PIO_OUT = 0xffffffff;
  switch ((input>>10)& 0x3) {
  case 1:
      d = -1;
      break;
  case 2:
      d = 1;
      break;
  default:
      d = 0;
      break;
  }
  switch ((input>>4) & 0x3F) {
  case 1:
      f0 += d;
      if (f0 < 0) f0 = 0;
      if (f0 > 999) f0 = 999;
      SEVEN_SEG[0] = 10; //Underscore
      display(f0, 3);
      break;
  case 2:
      f1 += d;
      if (f1 < 0) f1 = 0;
      if (f1 > 999) f1 = 999;
      SEVEN_SEG[0] = 11; //Upperscore
      display(f1, 3);
      break;
  case 3:
      duration += d;
      if (duration < 0) duration = 0;
      if (duration > 2000) duration = 2000;
      display(duration, 4);
      break;
  case 4:
      phasing += d;
      if (phasing < -99) phasing = -99;
      if (phasing > +99) phasing =  99;
      SEVEN_SEG[0] = 14; // P
      if (phasing < 0) {
          SEVEN_SEG[1] = 12; // -
          display(-phasing, 2);
      } else {
          SEVEN_SEG[1] = 13; // blank
          display(phasing, 2);
      }
      break;
  case 5:
      window += d;
      if (window < 0) window = 0;
      if (window > 1) window = 1;
```

```
        SEVEN_SEG[0] = 17; // raised o
        SEVEN_SEG[1] = SEVEN_SEG[2] = 13; // blank
        display(window, 1);
        break;
    case 6:
        focus += d;
        if (focus < -99) focus = -99;
        if (focus > 99) focus = 99;
        SEVEN_SEG[0] = 16; // F
        if (focus < 0) {
            SEVEN_SEG[1] = 12; // -
            display(-focus, 2);
        } else {
            SEVEN_SEG[1] = 13; // blank
            display(focus, 2);
        }
        break;
    default:
        for(i=0;i<4;i++) {
            SEVEN_SEG[i] = 12; // -
        }
        break;
    }
    if ((input>>10)& 0x3) {
        if (firststep) {
            firststep = 0;
            for (i=0;i<10;i++) {
                if ((PIO_IN>>10)& 0x3) {
                    usleep(30000);
                } else {
                    break;
                }
            }
        }
        usleep(10000);
    } else {
        firststep = 1;
    }
    if ((input & 0x1000)) {
        alt_printf("Starting calculation.\n");
        ff0 = (float)f0*1000;
        ff1 = (float)f1*1000;
        fduration = (float)duration*1e-6;
        PWM_DURATION = (uint16_t)(duration*8);
        for (i=0;i<duration*8;i++) {
            if ((i & 0xf) == 0) {
                display(i>>3, 4);
            }
            t = (float)i/FS;
            f = ff0+(ff1-ff0)*t/fduration/2.F;
            pulse[1][duration*8-1-i] = pulse[0][i] =  (int8_t)(127.F*
                fastsinf(M_TWOPI*t*f));
            //alt_printf("i=%x, t=%xus, f=%xkHz, y=%x\n", i, (uint32_t)(t*1
                e6), (uint32_t)(f/1000), pulse[0][i]);
        }

        // Calculate delay values for each channel
        ffocus = 80000.F/(float)focus;
        for(i=0;i<16;i++) {
            fchannel = (float)(i-8)*240.F;
            delays[i] = i*phasing + (int16_t)(fastsqrtf(ffocus*ffocus+
                fchannel*fchannel)-ffocus);
        }

        // Normalize delay values
        do {
```

```
        mindelay = 1;
        for(i=0;i<16;i++) {
            if (delays[i] < mindelay) {
                mindelay = delays[i];
            }
        }
        if (mindelay < 0) {
            for(i=0;i<16;i++) {
                delays[i]++;
            }
        }
        if (mindelay > 0) {
            for(i=0;i<16;i++) {
                delays[i]--;
            }
        }
    } while (mindelay != 0);

    // Set delays
    for(i=0;i<16;i++) {
        alt_printf("delays[%x] = %x\n", i, delays[i]);
        PWM_DELAYS_A[i] = delays[i];
        PWM_DELAYS_B[15-i] = delays[i];
    }

    switch (window) {
    case 0:
        for (i=0;i<16;i++) {
            PWM_AMPLITUDES_A[i] = 255;
            PWM_AMPLITUDES_B[i] = 255;
        }
        break;
    case 1:
        // Precalculated hanning window
        PWM_AMPLITUDES_A[0]  = 9;
        PWM_AMPLITUDES_A[15] = 9;
        PWM_AMPLITUDES_B[0]  = 9;
        PWM_AMPLITUDES_B[15] = 9;
        PWM_AMPLITUDES_A[1]  = 34;
        PWM_AMPLITUDES_A[14] = 34;
        PWM_AMPLITUDES_B[1]  = 34;
        PWM_AMPLITUDES_B[14] = 34;
        PWM_AMPLITUDES_A[2]  = 71;
        PWM_AMPLITUDES_A[13] = 71;
        PWM_AMPLITUDES_B[2]  = 71;
        PWM_AMPLITUDES_B[13] = 71;
        PWM_AMPLITUDES_A[3]  = 117;
        PWM_AMPLITUDES_A[12] = 117;
        PWM_AMPLITUDES_B[3]  = 117;
        PWM_AMPLITUDES_B[12] = 117;
        PWM_AMPLITUDES_A[4]  = 164;
        PWM_AMPLITUDES_A[11] = 164;
        PWM_AMPLITUDES_B[4]  = 164;
        PWM_AMPLITUDES_B[11] = 164;
        PWM_AMPLITUDES_A[5]  = 206;
        PWM_AMPLITUDES_A[10] = 206;
        PWM_AMPLITUDES_B[5]  = 206;
        PWM_AMPLITUDES_B[10] = 206;
        PWM_AMPLITUDES_A[6]  = 238;
        PWM_AMPLITUDES_A[9]  = 238;
        PWM_AMPLITUDES_B[6]  = 238;
        PWM_AMPLITUDES_B[9]  = 238;
        PWM_AMPLITUDES_A[7]  = 255;
        PWM_AMPLITUDES_A[8]  = 255;
        PWM_AMPLITUDES_B[7]  = 255;
        PWM_AMPLITUDES_B[8]  = 255;
```

```
            break;
        }
        if (!(input & 1)) {
            alt_printf("Both pulses (A then B)\n");
            PWM_MODE = PWM_MODE_DUAL;
            PWM_START_ADDRESS_A = (uint32_t)&pulse[0][0];
            PWM_START_ADDRESS_B = (uint32_t)&pulse[1][0];
        } else if (!(input & 2)) {
            alt_printf("Only pulse A\n");
            PWM_MODE = PWM_MODE_SINGLE;
            PWM_START_ADDRESS_A = (uint32_t)&pulse[0][0];
        } else {
            alt_printf("Only pulse B\n");
            PWM_MODE = PWM_MODE_SINGLE;
            PWM_START_ADDRESS_A = (uint32_t)&pulse[1][0];
        }
        alt_printf("Calculation completed.\n");
    }
    PIO_OUT = 0x00000000;
  }
  return 0;
}
```

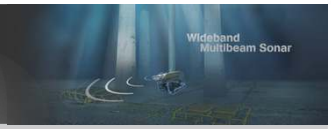
H.2 Header fil for bruk mot PWM-modulen

```
/*
 * pwm_generator.h
 *
 *  Created on: 25. feb. 2013
 *      Author: NorbitOPL
 */

#ifndef PWM_GENERATOR_H_
#define PWM_GENERATOR_H_

#include <system.h>

#define FS 8e6F

#define PWM_DELAYS_A          ( (volatile  int16_t *) (PWM_GENERATOR_BASE +
    0x00))
#define PWM_DELAYS_B          ( (volatile  int16_t *) (PWM_GENERATOR_BASE +
    0x20))
#define PWM_AMPLITUDES_A      ( (volatile uint16_t *) (PWM_GENERATOR_BASE +
    0x40)) // Egentlig bare 8 bit
#define PWM_AMPLITUDES_B      ( (volatile uint16_t *) (PWM_GENERATOR_BASE +
    0x60)) // Egentlig bare 8 bit
#define PWM_DURATION          (*(volatile uint16_t *) (PWM_GENERATOR_BASE +
    0x80))
#define PWM_START_ADDRESS_A   (*(volatile uint32_t *) (PWM_GENERATOR_BASE +
    0xA0))
#define PWM_START_ADDRESS_A_H (*(volatile uint16_t *) (PWM_GENERATOR_BASE +
    0xA0))
#define PWM_START_ADDRESS_A_L (*(volatile uint16_t *) (PWM_GENERATOR_BASE +
    0xA2))
#define PWM_START_ADDRESS_B   (*(volatile uint32_t *) (PWM_GENERATOR_BASE +
    0xA4))
#define PWM_START_ADDRESS_B_H (*(volatile uint16_t *) (PWM_GENERATOR_BASE +
    0xA4))
#define PWM_START_ADDRESS_B_L (*(volatile uint16_t *) (PWM_GENERATOR_BASE +
    0xA6))
#define PWM_MODE              (*(volatile uint16_t *) (PWM_GENERATOR_BASE +
    0xA8))

#define PWM_MODE_SINGLE 0
```

```
#define PWM_MODE_DUAL 1

#endif /* PWM_GENERATOR_H_ */
```

I Wide Band Multibeam Sonar data sheet

NORBIT
- explore more -



DATA SHEET - PS-080001-5 WBMS

Wideband Multibeam Sonar
with Real-time Image Update

Features

- Long range
- High update rate
- High angular resolution
- High range resolution
- Ethernet interface
- Standard video streaming protocols
- Ultra compact single unit solution

Applications

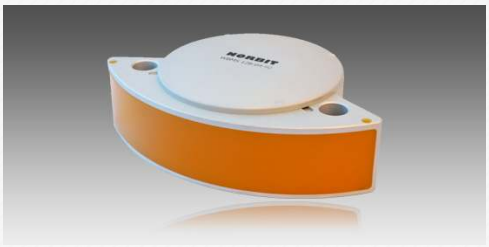
- Inspection (on ROV, AUV and other moving platforms)
- Obstacle avoidance
- Search and recovery
- Leak detection

Extensions

- Ultra high resolution (> 512 channels)
- 3D sonar
- Navigation integration

Options

- 60, 90, 120 or 180° coverage angle
- 4000m depth rating



WBMS 128-04-90

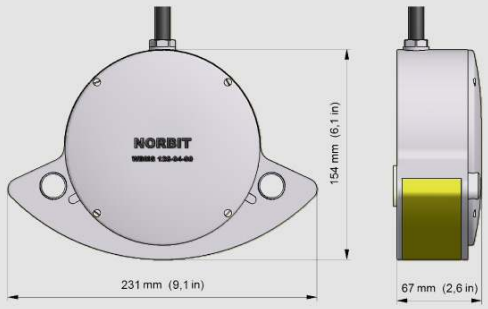
Compact High-Resolution Multibeam Sonar

The WBMS-series are ultra compact sonars designed specifically for use on moving platforms. Norbit's wideband multibeam technology allows long range real-time image updates, whilst simultaneously achieving high range resolution.

The WBMS-series are based on a flexible sonar platform that utilizes the latest in analog and digital signal processing. Combined with our R&D expertise, this flexibility enables us to adapt the technology to allow new applications to benefit from the advantages offered by a compact wideband multibeam sonar.

NORBIT

Sensors, instrumentation, telemetry and communications solutions for harsh environments. Norbit develops and delivers innovative products - allowing you to explore more.



Technical Specifications	
ANGLE HORIZONTAL	90°
ANGLE VERTICAL	20°
ANGULAR RESOLUTION	< 0.9°
OPERATING FREQUENCY	400 KHZ +/- 40 KHZ
RANGE RESOLUTION	< 10 MM
RANGE	> 100 M
UPDATE RATE	UP TO 20 HZ
DEPTH RATING	350 M
MASS	< 2 KG
BUOYANCY	< -7 N
DIMENSIONS	226 X 62 X 154 MM
OPERATING TEMPERATURE	-20°C TO +60°C
VOLTAGES	20 TO 28 VDC
POWER CONSUMPTION	< 30 W
INTERFACES	100 MB/S ETHERNET 16 MB/S RS-485

NORBIT SUBSEA | P.O.BOX. 1858, STIKLESTADVEIEN 1 | N-7441 TRONDHEIM | NORWAY | PHONE: +47 73 98 25 50 | SUBSEA@NORBIT.NO

J Fremdriftsplan

Arbeidsoppgave	Tidsfrist	Ble utført
Oppstartsmøte og gjennomgang av krav	-	18.01.2013
Utarbeide spesifikasjoner basert på krav	31.01.2013	31.01.2013
Blokkskjema	31.01.2013	31.01.2013
Komponentberegninger	07.02.2013	11.02.2013
Skjemategning	15.02.2013	01.03.2013
Bestilling av komponenter	15.02.2013	18.02.2013
Utlegg av mønsterkort og bestilling	01.03.2013	01.03.2013
Ferdig loddet kretskort	15.03.2013	20.03.2013
Ferdig bygget 16-kanals transducer	22.03.2013	13.03.2013
Software for utsending av pulser (FPGA)	22.03.2013	22.03.2013
Elektrisk designtest av kretskort	22.03.2013	05.04.2013
Karakterisering av transducer i vanntank hos Norbit	14.04.2013	10.04.2013
Akustisk designtest av sender i vanntank hos Norbit	14.04.2013	11.04.2013
Software for mottaksanalyse	19.04.2013	18.04.2013
Vertikal karakterisering av transducer i vanntank på Marintek	01.05.2013	23.04.2013
Test av prototype i vanntank hos Marintek	01.05.2013	26.04.2013
Innlevering av rapport	31.05.2013	•

NTNU HMS	Kartlegging av risikofylt aktivitet	Utarbeidet av	Nummer	Dato
		HMS-avd.	HMSRV2601	22-03-11
		Godkjent av	Side	Erstatter
		Rektor		01-12-06

Enhet: IET **Dato:** 03.04.2013

Linjeleder: *(ansv. veileder, faglig ansvarlig)* Ulf R. Kristiansen

Deltakere ved kartleggingen (m/ funksjon):
(Ansv. veileder, student, evt. medveiledere, evt. andre m. kompetanse) Stian Sønderland og Bjørnar Leithe (studenter)

Kort beskrivelse av hovedaktivitet/hovedprosess: Masteroppgave student Sønderland og Leithe. Tittel på oppgaven. 3D object tracking using 2D multibeam sonar

Signaturer: Stian Sønderland , B. Leithe

ID nr.	Aktivitet/prosess	Ansvarlig	Eksisterende dokumentasjon	Eksisterende sikringstiltak	Lov, forskrift o.l.	Kommentar
1	Feilsøking på elektronikk hvor spenning er påsatt.	Begge studentene				
2	Falle ut i vannet ved test på Marinteks laboratorier	Begge studentene				
3						

NTNU HMS /KS	Risikovurdering	Utarbeidet av	Nummer	Dato
		HMS-avd.	HMSRV2603	04-02-11
		Godkjent av	Side	Erstatter
		Rektor		09-02-10

Enhet: *IET* **Dato:** 03.04.2013

Linjeleder: *(ansv. veileder, faglig ansvarlig)* *Ulf R. Kristiansen*

Deltakere ved risikovurderingen (m/ funksjon):
(Ansv. veileder, student, evt. medveiledere, evt. andre m. kompetanse) *Stian Sønderland og Bjørnar Leithe*

Risikovurderingen gjelder hovedaktivitet: Masteroppgave student Sønderland og Leithe. Tittel på oppgaven. 3D object tracking using 2D multibeam sonar

Signaturer: Stian Sønderland, Bjørnar Leithe

ID nr.	Aktivitet/prosess fra kartleggingsskjemaet	Mulig uønsket hendelse	Vurdering av sann-synlighet	Vurdering av konsekvens				Risiko-verdi (menneske)	Kommentarer/status Forslag til tiltak
			(1-5)	Menneske (A-E)	Ytre miljø (A-E)	Øk./ materiell (A-E)	Om-dømme (A-E)		
1	Feilsøking på elektronikk hvor spenning er påsatt.	Elektrisk støt	2	B	B	A	A	B2	Jobber alltid 2 sammen. Tenke seg om før man berører elektronikk med spenning påsatt.
2	Falle ut i vannet ved test på Marinteks laboratorier	Blir våt, i verste fall besvimer som følge av slag mot hode ved fall.	2	B	A	A	A	B2	Jobber alltid 2 sammen. Holde arbeidsplassen ryddig.
3									