Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Automatisering og signalbehandling	Vårsemesteret, 2017 Åpen / ~~Konfidensiell~~
Forfatter: Christoffer J. Jensen	…… (signatur forfatter)
Fagansvarlig: Sven Ole Aase Veileder(e): Kristian Thorsen og Sven Ole Aase	
Tittel på masteroppgaven: Implementasjon av et Volterra prefilter i en klasse-D forsterker Engelsk tittel: Implementation of a Volterra prefilter in a class-D amplifier	
Studiepoeng: 30	
Emneord: Effektforsterkere, pulsbreddemodulasjon, ulineær forvrenging, digital signalbehandling	Sidetall: 50 + vedlegg/annet: 23 + vedlagt fil Stavanger, 15.06.2017 dato/år

Implementering av et Volterra prefilter i en klasse-D forsterker

Masteroppgave ved Universitetet i Stavanger
Det teknisk-naturvitenskapelige fakultet
Institutt for data- og elektroteknikk

Av Christoffer J. Jensen
15. juni 2017

Sammendrag

Målet med denne oppgaven er å foreta en digital filtrering av et musikksignal, og deretter implementere digital konvertering av dette signalet til et pulsbreddemodulert signal. Det endelig målet med prosjektet, som denne oppgaven er en del av, er å få konstruert en heldigital forsterker som kan sanntidsfiltrere musikksignal med oppsampling. I første omgang er arbeidet lagt opp for å fokusere på virkningsgraden til Volterra prefilteret. Hensikten til prefilteret er å eliminere ulineær forvrengning som oppstår når et tidsdiskret signal blir konvertert til et pulsbredde-modulert signal. Ved å bruke matematiske modeller kan det kompenseres for den ulineære forvrengningen i forkant av selve forvrengningen.

I dette oppgavearbeidet ble signalbehandlingsdelen i en heldigital forsterker konstruert. Programmet ble ikke optimalisert nok til å kunne filtrere signaler ved ønsket samplingsfrekvens, men analysering av virkningsgraden til prefilteret kan fremdeles gjøres ved en lavere samplings-frekvens. Den laveste beregningstiden som ble målt er 9,107 µs. Beregningstiden er ønskelig å få under 5,67 µs, da vil det være mulig å sanntidsfiltrere et standard musikksignal (oppsamplet med fire) som har en samplingsfrekvens på 176,4 kHz. Mesteparten av implementasjonen er gjort på en DSC (Digital Signal Controller). Resultatet viser korrekte beregninger, men frekvensanalysen av det rekonstruerte analoge signalet gir utilfredsstillende resultater. Da konstruering av pulsbreddemodulert signal må være perfekt symmetrisk, kan små unøyaktigheter føre til generering av ulineær forvrengning som vil gjøre alle beregningene bortkastet.

Denne oppgaven er et godt grunnlag for videre arbeid. Det er gitt konkrete forslag for økt nøyaktighet i beregningene og raskere beregningshastighet. I tillegg bør DSC-ens pulsbredde-modulerte signalgenerering bli eksaminert for å bekrefte god virkemåte.

Forord

Først og fremst må jeg gi en stor takk til mine to veiledere, Kristian Thorsen og Sven Ole Aase, som har vært til stor hjelp gjennom hele semesteret. Oppgavearbeidet har vært en utfordrende og lærerik prosess.

Hensikten med denne rapporten er å få et innblikk i heldigitale forsterkere og en nyutviklet metode for forvrengningsredusert implementering av heldigitale forsterkere. Oppgaven er skrevet på norsk og det er forsøkt å bruke mest mulig korrekte og lett forståelige oversettelser av engelske fagtermer. Noen termer er blitt omtalt på engelsk i fravær av god oversettelse, disse ordene er skråstilt eller står i parentes. Engelske egennavn har vanlig skrift. I vedlegg A er det en liste over oversettelser med akronymer. Grunnleggende teori er å finne i vedlegg B, som inneholder en kort gjennomgang av diverse tema med definisjoner.

Mye av teorien tar utgangspunkt i et standard musikksignal oppsamplet med en faktor på 4, som resulterer i en samplingsfrekvens på 176,4 kHz. På grunn av manglende optimalisering innen programmets tidsbruk har det blitt nødvendig å velge en noe lavere samplinsfrekvens på 100 kHz. Dette tilsvarer fire gangers oppsampling fra et «kunstig» musikksignal som har en samplingsfrekens på 25 kHz. Simulert hørbar båndbredde blir da cirka $0 - 12,5$ kHz, og det er omtrent denne båndbredden som det blir foretatt frekvensanalyse av.

Innhold

1 Innledning

1.1 Motivasjon

Digitale, såkalte klasse-D, forsterkere har bortsett fra forvrengning mange fordeler i forhold til analoge forsterkere. En ønskelig egenskap er at selv om digitale signaler blir forstyrret av støy vil informasjonen bli beholdt, forbeholdt at støyen ikke er for stor. Informasjonen i analoge signaler vil, i motsetning, bli forringet av støy til en viss grad. I tillegg har digitale forsterkere en praktisk effektivitet på over 90 % (teoretisk 100 %) [5], som er betydelig høyere enn hva som er mulig å oppnå med analoge forsterkere (teoretisk maksimalt 78,5 %, praktisk mye lavere). I de analoge forsterkere blir ofte mer enn halvparten av tilført energi tapt i form av varme, noe som gjør at slike forsterkere trenger store kjøleribber og kraftig strømforsyning. Klasse-D forsterkere kan derimot konstrueres i mindre dimensjoner grunnet mindre behov for elektrisk effekt og fravær av større kjøleribber.

Ved å eliminere ulineær forvrengning kan heldigitale forsterkere oppnå lydkvalitet på nivå med de andre forsterkerklassene, men med den desidert høyeste effektiviteten. Dette åpner for muligheten å lage forsterkere med høy lydkvalitet i relativt små dimensjoner.

1.2 Oppgavebeskrivelse

Et fundamentalt problem er hvordan konvertering fra musikksignal til et PWM-pulstog skal gjøres. For et analogt musikksignal (for eksempel fra grammofon) finnes det metoder som gjør dette, med brukbart resultat. For et digitalt signal (for eksempel fra CD, eller streaming) er det mye vanskeligere. Problemet er at et digitalt signal (tidsdiskret), eksempevis PCM (Pulse Code Modulation), typisk består av en serie med sampler der størrelsen eller høyden på signalet i hvert punkt er representert med en tallverdi (typisk på 16-bit). I utgangstrinnet er det derimot bare påtiden til transistorene det er mulig å endre, og høydene i punktsamplingene må konverteres til bredder i PWM-signalet som bestemmer påtiden til transistorene. I praksis er ikke denne konverteringen perfekt, og det innføres ulineær forvrenging i signalet.

For å eliminere ulineær forvrengning er det foreslått i [1] å gjøre en digital kompensasjon i forkant av selve forvrengningen, ved hjelp av matematiske modeller (Volterra modeller). Oppgaven går ut på implementere signalbehandlingsdelen i en heldigital forsterker, hvor digital kompensasjon og konvertering av kompensert musikksignal til pulsbreddemodulert signal vil finne sted. Signalbehandlingsdelen gjøres ved å ta i bruk en DSC fra Texas Instruments. I første omgang skal enkle sinussignaler bli brukt for å teste virkningsgraden til Volterra prefilteret.

Denne oppgaven er en del av et prosjekt som har det overordnede målet å konstruere en heldigital forsterker som kan sanntidsfiltrere musikksignaler.

Et viktig element oppgaven er å velge en DSC som har høy nok beregningskapasitet og moduler som kan generere nøyaktige pulsbreddemodulerte signaler med høy oppløsning. Hver enkelt samplingsverdi skal konverteres til en pulsbredde og i tillegg skal det kompenseres for ulineær forvrengning i forkant ved bruk av et Volterra prefilter. Et standard digitalt musikksignal fra en CD-plate har en samplingsfrekvens på 44,1 kHz, og det er ønskelig å oppsample signalet før beregning, eksempelvis med fire. Dette betyr at kortet må gjøre beregninger for 176400 unike sampler hvert sekund.

2 | Teoretisk bakgrunn

2.1 Forsterkerklasser

Det finnes mange ulike musikk- eller lydforsterkere på markedet idag. Ut fra oppbygging og virkemåte blir de klassifisert i klasser indikert med bokstaver. De fire hovedklassene er A, B, AB og D. Det finnes ytterligere klasser, men disse er veldig lite tatt i bruk i forhold til hovedklassene og klasse-C blir ikke brukt til lydforsterkning i det hele tatt [4].

Både effektivitet og lydkvalitet varierer mellom klassetypene. Vanligvis blir lydkvaliteten prioritert fremfor effektivitet og det kan derfor være vanskelig å høre forskjell på forsterkertypene. Klassen med best lydkvalitet er klasse-A fordi den ikke har noe overgangsforvrengning, men samtidig har den en lav maksimal teoretisk effektivitet på 25 % (50 % ved bruk av transformator på utgangen) [4]. Klasse-B har en maksimal teoretisk effektivitet på 78,5 %, og klasse-AB, som er en kombinasjon av klassene A og B, har en maksimal teoretisk effektivitet på mellom 50 % og 78,5 % [5]. På grunn av konstruksjonen til klasse-B og -AB kan det forekomme overgangsforvrengning, og dermed dårligere lydkvalitet.

Ved teoretisk utregning blir alle forhold ansett som ideelle og det tas ikke hensyn til indre resistanser til diverse komponenter. Det vil alltid være en merkbart lavere effektivitet i virkeligheten for alle forsterkerklassene.

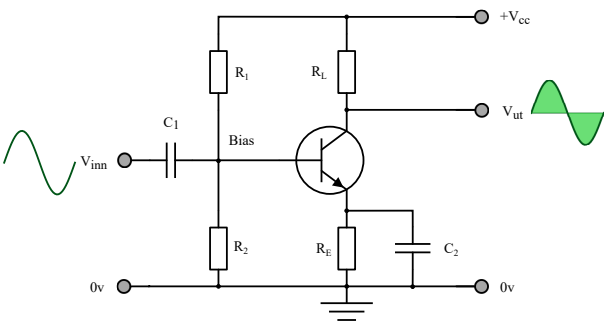
Klasse-D forsterkere, eller digitale forsterkere, har en annen oppbygging enn de andre hovedklassene. Forskjellen er at digitale forsterkere bruker felteffekttransistorer som ideelle brytere, som enter er av eller på. Analoge forsterkere kan bli laget med enten bipolare transistorer eller felteffekttransistorer, men her vil transistorene virke som tilnærmet lineære signalforsterkere. Hovedulikheten mellom digitale og analoge forsterkere er den foregående delen. I digitale forsterkere blir innsignalet (analogt eller tidsdiskret) konvertert til et pulsbreddemodulert signal, som videre blir omtalt som PWM-signal (Pulse Width Modulation), for så å forsterke dette signalet etterfulgt av et analogt lavpassfilter for rekonstruksjon av innsignalet. Siden det er et PWM-signal vil det bare ha to amplitudeverdier, høy eller lav, derfor brukes transistorene som ideelle brytere. Analoge forsterkere derimot forsterker det analoge innsignalet direkte, uten noen form for konvertering.

Det finnes to måter å implementere digitale forsterkere på og det avhenger av innsignalet: Analogt eller digitalt (tidsdiskret). Ved analog implementering blir det analoge inngangssignalet sammenlignet med periodisk triangel- eller sagtannbølgeform, og resultatet fra komparatoren er et PWM-signal. En digital implementering tar i bruk et tidsdiskret signal og konverterer sample-amplituden til en tilsvarende pulsbredde fortløpende. På denne måten er forsterkeren digital hele veien fram til høyttaler.

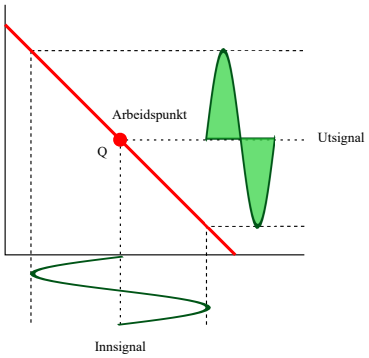
I fortsettelsen vil alle forsterkerklassene bli forklart på et grunnleggende nivå for å vise hovedforskjellene. I praksis vil forsterkere ha mer avansert oppkobling og være optimalisert for minimert forvrengning.

2.1.1 Klasse-A

Klasse-A forsterkere har en enkel utgangstransistor som leder strøm under hele perioden til et sinussignal. Transistoren er påført en forspenning for å alltid være utenfor *cutoff*-område. Fordelen med dette er at forsterkeren vil operere tilnærmet lineært og ikke ha overgangsforvrengning, men ulempen er at transistoren vil lede strøm selv om signalet er null, som fører til mye varmetap og lav effektivitet.



Figur 2.1: Klasse-A kretstegning.

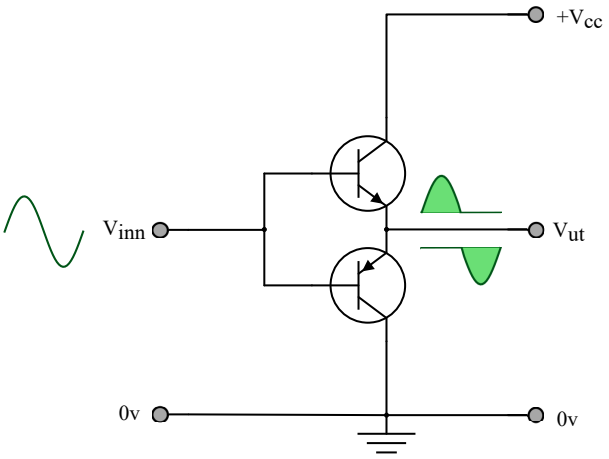


Figur 2.2: Operasjonskurve.

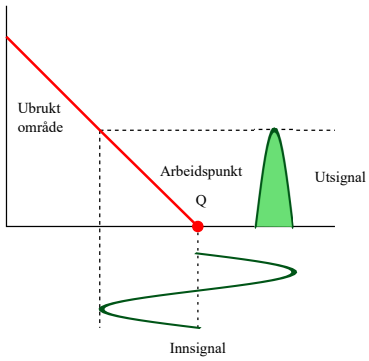
2.1.2 Klasse-B

I motsetning til klasse-A, har klasse-B forsterkere to utgangstransistorer hvor transistorene leder hver sin halvdel av sinussignalet. Fordelen med dette er at når signalet er null vil det ikke være noe strømflyt og ingen energitap. Ulempen er at når signalet er innenfor et intervall fra -0,7 V til 0,7 V, havner transistorene i *cutoff*-området og virker som en åpen krets eller den leder ulineært. Dette fører til såkalt overgangsforvrengning som skapes i overgangen mellom den positive og negative delen av et sinussignal.

I produksjonen av klasse-B forsterkere blir de laget slik at hver transistor leder hver sin halvdel av et sinussignal presist. Ved hjelp av påført forspenning, negativ tilbakekobling og mer avansert oppkobling vil det føre til minimal overgangsforvrengning. Effektivitet av klasse-B forsterkeren er en klar forbedring fra klasse-A, og den er den vanligste. 99 % av alle forsterkere som blir laget er klasse-B forsterkere [13].



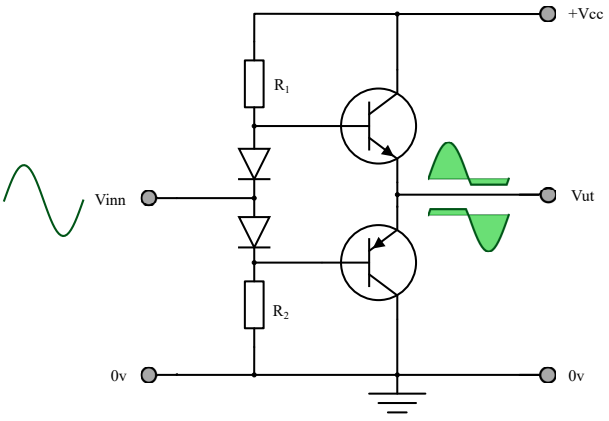
Figur 2.3: Klasse-B kretstegning.



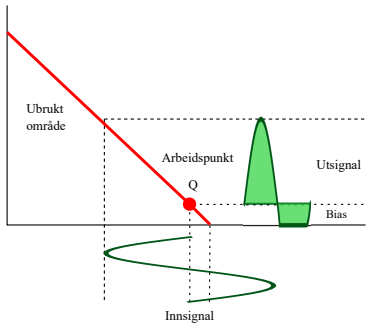
Figur 2.4: Operasjonskurve for en transistor.

2.1.3 Klasse-AB

En klasse-AB forsterker er en variant av klasse-B forsterker, hvor det er påført en forspenning som gjør at hver transistor leder litt mer enn halve sinussignalet hver. Ofte er forsterkere omtalt som «AB» egentlig «B», da forspenningen er akkurat høy nok for at transistorene leder nøyaktig halvparten av signalet hver. Den teoretiske effektiviteten til klasse-AB er en blanding av klasse-A og -B og ligger derfor mellom 50 % og 78,5 %.



Figur 2.5: Klasse-AB kretstegning.

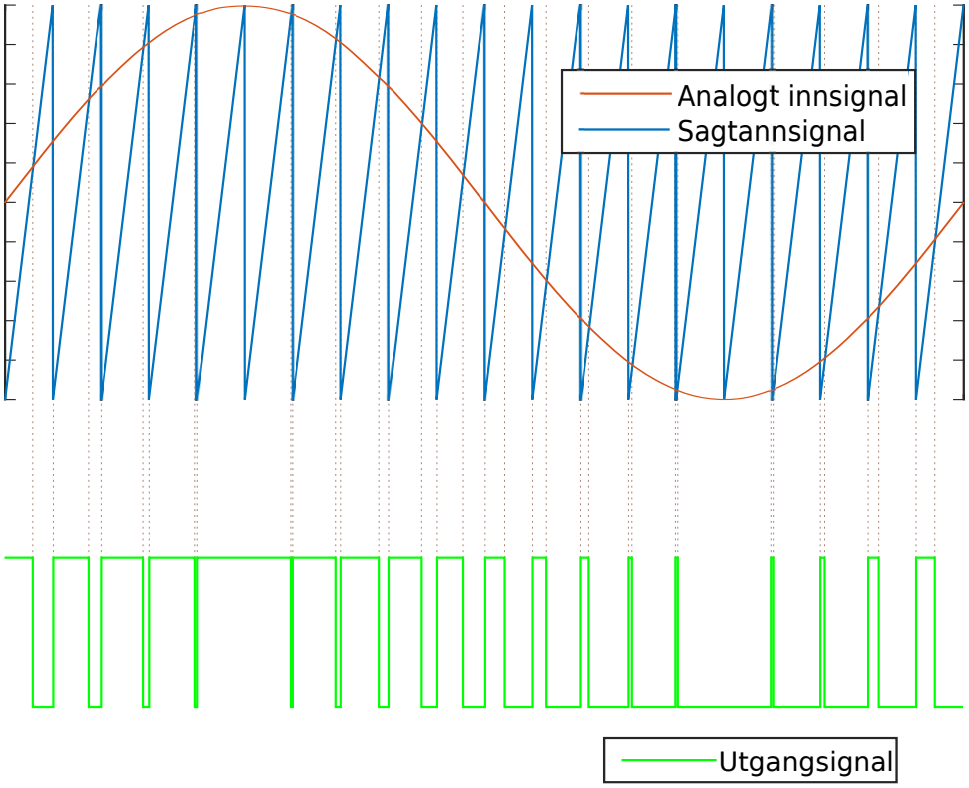


Figur 2.6: Operasjonskurve for en transistor.

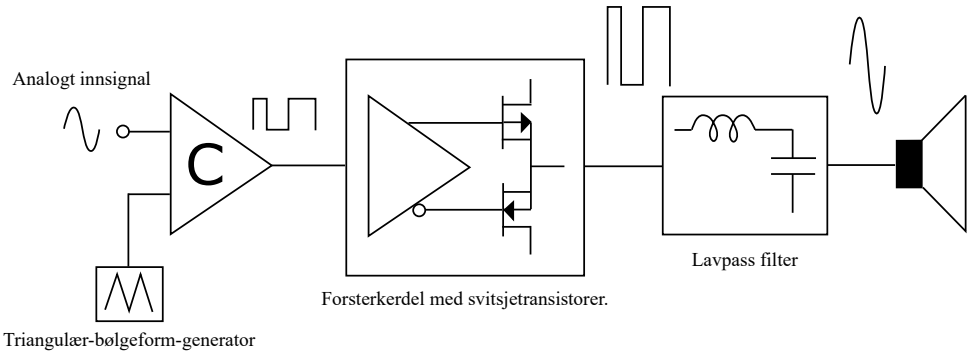
2.1.4 Klasse-D

De fleste såkalte klasse-D forsterkere er ikke rent digitale: Inngangssignalet er et analogt signal som blir sammenlignet med et triangel- eller sagtannbølgeform for å skape et PWM-signal (se figur 2.7). Dette PWM-signalet er da et 2-nivå signal som deretter blir forsterket og filtrert gjennom et lavpassfilter. Dette resulterer i et analogt signal som i teorien skal være identisk til innsignalet, bare forsterket. Denne implementeringen fungerer bra men det vil alltid forekomme dødtidsforvrengning på grunn av transistorenes dødtid, men ved å velge transistorer med lik dødtid ved av- og påslåing, samt så liten dødtid som mulig vil forvrengningen minimeres. Dødtid er forsinkelse i transistorens av- og påslåing. Det er nødvendig når transistorer står i halvbro, slik som i figur 2.8, for å unngå kortslutning mellom forsyningsspenning og jord ved

samtidig svitsjing av begge transistorene. Denne typen forsterkere er vanlig å finne i bilradioer, hvor denne typen fungerer bra. I teorien er effektiviteten til en klasse-D forsterker 100 %, og i praksis er effektivitet noe lavere men fremdeles over 90 %. Dette gjør klasse-D forsterkeren til den mest effektive forsterkeren av dem alle.



Figur 2.7: Illustrasjon som viser virkemåten til en komparator brukt i analog-modulert digital forsterker.



Figur 2.8: Analog-modulert klasse-D forsterker. Bildet er hentet fra [21] og bildeteksten er oversatt til norsk.

For å lage en heldigital forsterker brukes det en annen type pulsbreddemodulasjon, kalt PWM-mapping, istedenfor å sammenligne et analogt signal med et trekantsignal. PWM-mapping går ut på å ta amplituden til hver enkel sample i det digitale signalet og konvertere det til en tilsvarende pulsbredde. Denne metoden er digital hele veien frem til høyttaleren, som gjør den mer robust mot støy og enklere å kontrollere. Ulempen med heldigitale forsterkere er at mappingen fra tidsdiskret signal til PWM-signal gir ulineær forvreng-
ning.

2.2 Harmonisk forvrengning

Et problem med praktisk realisering av audiosystemer er harmonisk forvrengning. Ingen audiosystemer har perfekte lineære komponenter, derfor vil alle systemer legge til ulinearitеter i prosessen. Dette fører til harmonisk forvrengning som tilføyer harmoniske svingninger til det opprinnelige signalet og vil resultere i uønskete endringer etter signaloverføringen. En harmonisk svingning er en komponent som har ren sinussvingning og en frekvens gitt som grunnfrekvensen multiplisert med et positiv heltall. Grunnfrekvensen er definert som den laveste frekvensen i en periodisk bølgeform, på engelsk blir den omtalt som *fundamental frequency* eller bare som *fundamental*.

Ofte brukes et rent sinussignal, som da er grunnfrekvensen, for å enkelt teste mengden harmonisk forvrengning. For eksempel kan det brukes et sinussignal med frekvens 1000 Hz. Når harmonisk forvrengning forekommer, vil det resulterende signalet etter signaloverføringen bestå av det opprinnelig signalet kombinert med komponenter på 2000 Hz, 3000 Hz, 4000 Hz etc.

En måte å måle harmonisk forvrengning på er å regne ut total harmonisk forvrenging, som på engelsk kalles *Total Harmonic Distortion* (THD). I både spenning- og strømbølgeformer kan mengden av forvrengning regnes ut ved hjelp av THD. Det finnes to definisjoner for THD, der definisjonene går ut på å ta summen av alle harmoniske komponenter og sammenligne med enten grunnfrekvensen eller hele bølgeformen; altså grunnfrekvens og alle harmoniske komponenter [8]. Definisjonene bemerkes gjerne som THD_F (sammenligning med *fundamental*) og THD_R (sammenligning med hele bølgeformens RMS-verdi). RMS-verdi (Root Mean Square) er det samme som effektverdi, se B.2 for nærmere forklaring. THD_F er den anbefalte metoden å bruke og er gitt som forholdet mellom summert RMS-verdi av alle harmoniske komponenter sammenlignet med RMS-verdien til grunnfrekvensen. Det er det også mulig å bruke amplitudene fremfor RMS-verdiene for å regne ut THD. I ligningene 2.1 som er hentet fra [8] er det nevnt at I_n kan være enten RMS-verdier eller amplituder.

$$THD_F = \frac{\sqrt{\sum_{n=2}^{\infty} I_n^2}}{I_1}; \quad THD_R = \sqrt{\frac{\sum_{n=2}^{\infty} I_n^2}{\sum_{n=1}^{\infty} I_n^2}} \tag{2.1}$$

Vanligvis ved måling av forvrengning inngår det også støy i regnestykket. Denne metoden kalles *Total Harmonic Distortion plus Noise* (THD+N), men i litterær sammenheng tas det for gitt at leseren vet at støyleddet er med og «+N» benevningen er ofte utelatt [13]. Som navnet tilsier er denne målingen mengden av total harmoniske forvrengning pluss støy. Definisjonen tar utgangspunkt i THD_F eller THD_R men har i tillegg tatt med et støyledd som adderes med summen av de harmoniske komponentene før sammenligning. I ligning 2.2 vises definisjonen på THD+N hvor N er støy. Her vises kun THD+N med utgangspunkt i THD_F.

$$\text{THD+N} = \frac{\sqrt{\sum_{n=2}^{\infty} I_n^2} + \text{N}}{I_1} \tag{2.2}$$

2.3 Overgangsforvrengning

Overgangsforvrengning er forvrengning som oppstår i såkalte *push-pull*-forsterkere, klasse-AB eller klasse-B. I slike forsterkere vil to transistorer bytte på å lede signalet, en transistor vil lede den positive halvdelen mens den andre vil lede den negative halvdelen. Selve forvrengingen oppstår når en transistor er i *cutoff*-området, det vil si hvis Base-Emitter spenningen er mindre enn omtrent 0,7 V (Felles-Emitter oppkobling). I dette området vil transistoren lede signalet ulineært eller være helt avslått og ikke lede noe. Når inngangssignalet faller innenfor intervallet -0,7 V til 0,7 V vil da overgangsforvrengning oppstå, som er i overgangen mellom positiv og negativ halvdel av signalet.

2.4 Støy

I elektrisk sammenheng er støy definert som tilfeldig genererte elektriske signaler som oppstår på grunn av den fysiske virkemåten til elektroniske komponenter [13]. Støy er generelt beskrevet som all uønsket lyd som kan høres fra et audiosystem. Ulyder fra audiosystemer kan være grunnet forvrengning og dermed bli ansett feilaktig som støy. Forvrengning og støy er to adskilte definisjoner som ikke må forveksles. Støy er signaler som består av tilfeldige amplituder og frekvenser, som blir ansett som et signaler uten informasjon.

Termisk støy er en type elektrisk støy som skyldes termiske vibrasjoner av elektroner. I en elektrisk komponent med temperatur over det absolutte nullpunkt, -273,15 °C, vil elektronene i komponenten være utsatt for termiske vibrasjoner som gir opphav til elektriske spenninger. Vibrasjonene er tilfeldig av natur og kan generere støy ved alle frekvenser. Termisk støy har bortimot konstant effekttetthet gjennom hele rekkevidden i frekvensintervallet 5 Hz - 500 kHz, og kan derfor bli ansett som tilnærmet hvit støy [4, s. 346]. Hvit støy er en statistisk modell som har konstant effekttetthet gjennom hele frekvensspekteret og er ikke et spesifikt signal. Navnet hvit støy er lånt fra fargelæren fordi hvit lys inneholder alle frekvenser. I tillegg blir termisk støy kalt også Johnson støy (etter den første personen som forsket på dette område) [4, s. 346].

Et ideelt støyfritt system kan bli gitt ved en transferfunksjon der utgangen $y(t)$ er en funksjon av inngangen x

$$y(t) = F(x(t))$$

Ved en perfekt forsterkning A og tidsforsinkelse T vil transferfunksjonen være gitt ved

$$y(t) = Ax(t - T)$$

og ha ingen forvrengning ved utgangen. Som tidligere nevnt er ingen audiosystemer perfekt lineære, og derfor kan ingen virkelige audiosystemer i praksis bli representert med denne transferfunksjonen.

2.5 Effektivitet

Effektiviteten til en forsterker kan finnes ved å regne ut virkningsgraden. Den er definert som forholdet mellom avgitt effekt (levert til høyttaler) og tilført effekt (typisk fra strømnettet), og har ingen enhet. Virkningsgraden er gitt som et tall mellom 0 og 1, og kan også bli representert som et prosenttall. I praksis vil ingen forsterkere ha 100 % effektivitet, på grunn av termodynamikkens 2. lov som forbyr virkningsgrader lik eller over 1. Definisjonen for virkningsgrad er gitt ved

$$\text{Virkningsgrad} = \eta = \frac{P_{avgitt}}{P_{tilført}} \tag{2.3}$$

Det er i tillegg mulig å finne effekttapet fra forholdet mellom P_{avgitt}, $P_{tilført}$ og P_{tap}. Forholdet er gitt ved

$$P_{tilført} = P_{avgitt} + P_{tap} \tag{2.4}$$

2.6 Pulskodemodulasjon

Pulskodemodulasjon, *Pulse-Code Modulation* (PCM) på engelsk, er en metode som brukes for å lage en digital representasjon av et analogt signal. PCM-signaler er tidsdiskrete signaler. Et analogt signal er kontinuerlig i tid, som betyr at det har signalverdier ved alle tidspunkt. I tillegg har det kontinuerlig amplitude, som betyr at det potensielt kan inneholde et uendelig antall signalverdier. Det egner seg ikke å lagre slike signal digitalt, løsningen på dette er å sample signalet. Sampling betyr å ta punktvise målinger av signalet med konstante intervall, T_s, som kalles tidsteg, sampletid eller -periode. Samplingsfrekvens er hvor mange målinger som blir gjort av signalet i sekundet, og finnes ved

$$f_s = \frac{1}{T_s} \tag{2.5}$$

Sampingsfrekvens er oppgitt i Hz eller Sa/s (sampler per sekund).

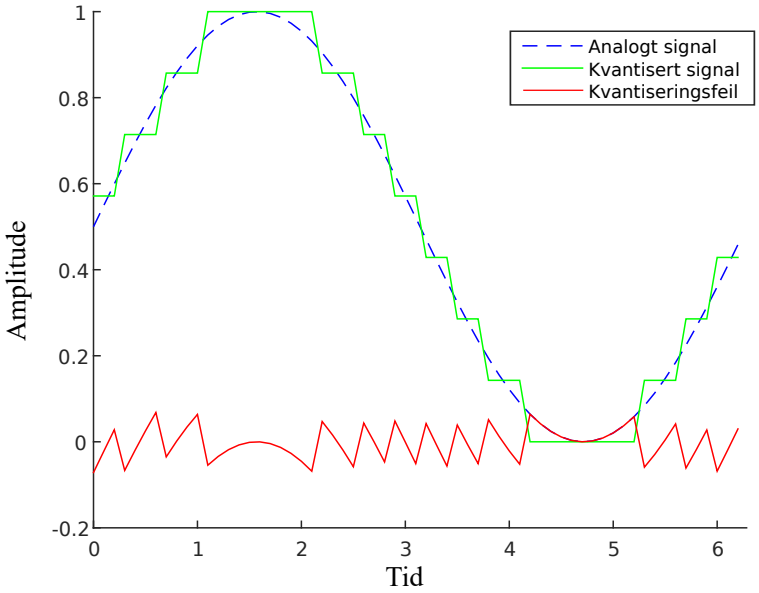
Ved å ta i bruk PCM kan et analogt signal bli representert av mange tidsdiskrete verdier. Oppsampling er en teknikk som brukes for å finne en tilnærming til et tidsdiskret signal hvis det hadde blitt samplet med en høyere frekvens. Nærmere forklaring av dette temaet blir gjort i delkapittel2.8.

Det såkalte Nyquist-teoremet forteller om den minste mulige samplingsfrekvensen for å unngå aliasing. Aliasing oppstår når et signal blir samplet med en for lav frekvens og det rekonstruerte signalet blir forskjellig fra det opprinnelige. Teoremet sier at samplingfrekvensen f_s må være større enn det dobbelte av den høyeste frekvenskomponenten $f_{a(maks)}$ i det analoge signalet. I praksis bør samplingfrekvensen være enda høyere enn dette for å få en god representasjon av det analoge signalet. Nyquist-frekvensen f_s er oppgitt som:

$$f_s > 2f_{a(maks)} \tag{2.6}$$

2.7 Kvantisering

I konvertering fra analogt til tidsdiskret signal forekommer det såkalt kvantiseringsfeil, også kalt avrundingsfeil. Når et analogt signal blir samplet må sampleverdiene bli lagret digitalt, og fordi digitale verdier kun har en viss oppløsning må amplituden i det analoge signalet avrundes til nærmeste kvantiseringsnivå. Økt bitoppløsning eller -dybde vil gi flere kvantiseringsnivåer og økt nøyaktighet av representasjonen av signalet. Ved høyt antall kvantiseringsnivåer kan avrundingsfeilen være så liten at den blir ubetydelig. Standarden for kvantiseringsnivå i musikk-CD-er er satt til å være 16-bits oppløsning [20], som betyr at det er $2^{16} = 65536$ kvantiseringsnivåer tilgjengelig og det er godt nok for en verdensomfattende standardisering.



Figur 2.9: Visualisering av kvantiseringsfeil med en bitdybde på 3 bit.

For å oppnå en god representasjon av et analogt signal må både samplingsfrekvens og bitdybde være passende høy. Standardiseringen av lyd på CD-er, *Compact Disc Digital Audio* (CD-DA), spesifiserer at digital audio skal være kodet med 2 kanaler med toerkomplementform 16-bit LPCM samplet ved 44100 Hz [20]. LPCM står for lineær-PCM og betyr at kvantiseringsnivåene ligger på en lineær skala.

2.8 Oppsampling

Oppsampling er en teknikk som brukes på tidsdiskrete signaler for å finne en tilnærming til det samme tidsdiskrete signalet hvis det hadde vært samplet med en høyere frekvens. Oppsampling med en heltallsfaktor K gjøres i to steg:

1. Lag en sekvens $x_K(n)$ ved å sette $K-1$ nuller mellom hver sample i den originale signalsekvensen $x(n)$.
2. Bruk et interpolasjonsfilter på $x_K(n)$ for å jevne ut sekvensen uten bidrag fra nullverdiene.

Ved en oppsamplingsfaktor $K = 2$ kan interpolasjonsalgoritmen gjøres så enkel som å endre nullverdien til gjennomsnittsverdien av de to nabopunktene. Det er mulig å velge mer avanserte filtre avhengig av tilgjengelig beregningskapasitet.

Oppsampling med et brøktall må gjøres i to deler, først oppsampling (og interpolasjon) med teller etterfulgt av nedsampling med nevner. Nedsampling er den motsatte operasjonen av oppsampling. Ved en nedsampling med M vil kun hver M-te sample bli tatt vare på, resten av samplene blir forkastet.

2.9 Generell pulsbreddemodulasjon

Pulsbreddemodulasjon, eller på engelsk *Pulse Width Modulation* (PWM), er en metode som konverterer et analogt eller digitalt signal til et tidskontinuerlig signal som kun har to amplitudeverdier: en høy og en lav. Modulering av et signal innebærer å endre pulsebreddene mens periodetiden holdes konstant. Visuelt ser signalet ut som et firkantpulstog ved måling med et oscilloskop. En ideell firkantpuls skal endre tilstand umiddelbart, men i virkeligheten brukes det litt tid på tilstandsendringen. Tiden det tar å skifte fra lav til høy amplitude kalles *rise time* og motsatt vei *fall time*. De vertikale kantene som fremkommer av tilstandsendring kalles *rising edge* og *falling edge*, fordi signalet blir «reist» opp til høy amplitude eller det «faller» ned til lav amplitude. Ved å ta utgangspunkt i en *positive-going* puls (se figur 2.10) vil tiden fra *rising edge* til *falling edge* være tidsrommet der signalet har aktiv eller høy amplitude, dette kalles pulsbredde τ. Motsetningen kalles *negative-going* puls (se figur 2.11). Her vil pulsbredden være gitt ved tiden fra *falling edge* til *rising edge*, og aktiv amplitude tilsier derimot lav amplitude.

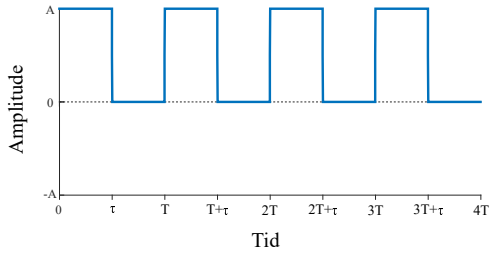
Arbeidssyklus er forholdet mellom pulsbredde τ og periodetid T. Den er oppgit i prosent og finnes ved:

$$Arbeidssyklus = \left(\frac{\tau}{T}\right) 100\% \tag{2.7}$$

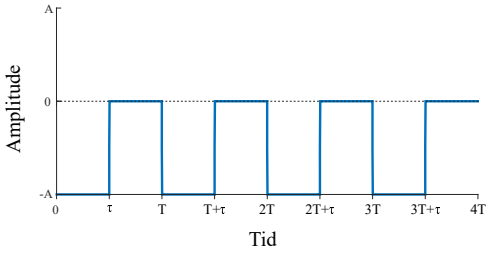
PWM er en metode som ofte brukes til å styre permanent-magnetiserte likestrømsmotorer eller dimming av LED lamper. Arbeidssyklusen vil bestemme hvor stor likespenning lasten vil oppfatte. Den vil være en prosentandel av amplitudeforskjellen mellom høy og lav amplitude: $y_{maks} - y_{min}$. Ofte er $y_{min} = 0$ og da er resulterende likespenning arbeidssyklus multiplisert med y_{maks}. Figur 2.10 viser en *positive-going* puls, det betyr at aktiv amplitude er høyere enn passiv amplitude. Her vil arbeidssyklusen være på 50 % og oppfattet spenning vil være gjennomsnittsspenningen, som er halvparten av amplituden A fordi $y_{min} = 0$. Figur 2.11 viser en *negative-going* puls, det betyr at aktiv amplitude er lavere enn passiv amplitude. Her er også arbeidssyklus 50 %, men gjennomsnittsspennigen vil derimot være negativ.

Et PWM-signal kan være modulert i enten symmetrisk eller asymmetrisk modus. Dette viser til hvilke kanter som blir flyttet for å endre arbeidssyklusen. Asymmetrisk modus flytter enten *rising edge* eller *falling edge*, mens symmetrisk modus flytter begge to samtidig.

Det finnes to typer for asymmetrisk modulering av et PWM-signal: *leading-edge* eller *trailing-edge*. Navnene tilsier hvilken kant som holdes konstant ved enten starten eller slutten av perioden, slik at den andre kanten kan fritt moduleres. Ved *leading-edge* modulering vil kanten holdes konstant ved slutten av perioden, hvis det er en *positive-going* puls vil det si at *falling edge* holdes konstant og *rising edge* blir modulert (se figur 2.12). Ved å bruke den andre asymmetriske

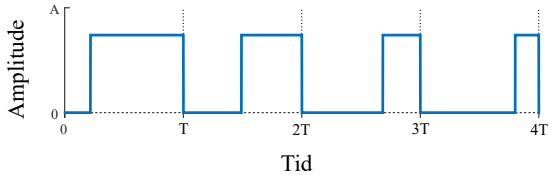


Figur 2.10: *Positive-going* puls.

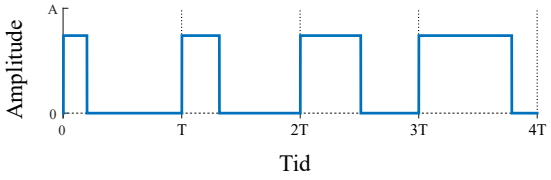


Figur 2.11: *Negative-going* puls.

måten kalt *trailing-edge* modulering, vil kanten være konstant plassert ved starten av en periode og det er *rising-edge* som brukes til modulering, hvis pulsen er *positive-going*.

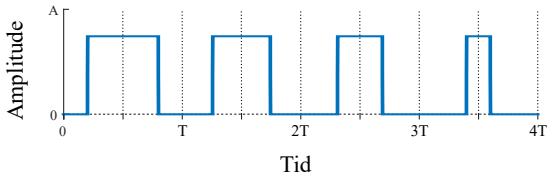


Figur 2.12: *Leading-edge* puls.



Figur 2.13: *Trailing-edge* puls.

Symmetrisk modulering vil derimot bruke begge kantene til modulering og ta utgangspunkt i et referansepunkt, som blir midtpunktet i perioden til det resulterende PWM-signalet. Ved økning av arbeidssyklus vil begge kantene forflyttes symmetrisk vekk fra midtpunktet.



Figur 2.14: Symmetrisk-modululert puls.

2.10 Volterra modell

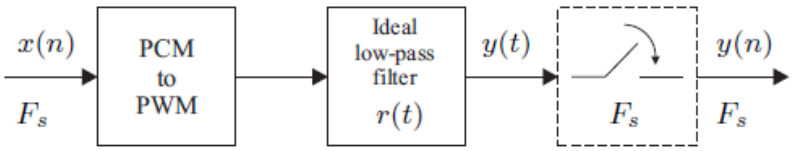
Flere deler i dette delkapittel er i stor grad hentet fra aktuelle deler i [1] og oversatt direkte. Artikkelen omhandler problemet med mapping fra PCM-signal til PWM-signal, hvor hver amplitude til samplene i PCM-signalet blir konvertert til pulsbredder i PWM-signalet. Som nevnt tidligere medfører dette ulineær forvrengning av signalet, men en eventuell løsning er funnet i denne artikkelen: Prekompensering med et digitalt filter før forvrengningen skjer. I eksperimentene som ble gjort i denne artikkelen ble det valgt å implementere prefilteret ved å ta i bruk FIR-filterstrukturer som korresponderer med et ideelt rekonstruksjonsfilter (analogt lavpassfilter).

Journalartikkelen [1] omhandler flere forskjellige Volterra modeller. Disse blir bestemt ut fra symmetrien av PWM-signalet og type analogt lavpassfilter brukt til demodulering av PWM-signalet. Demodulering av et signal betyr å gjenskape det originale signalet fra det modulerte signalet. Analogt filtre som brukes til dette kalles gjerne demodulerings- eller rekonstruksjonsfilter.

Oppgaven tar utgangspunkt i symmetrisk PWM-signal og et ideelt lavpassfilter, de andre alternative metodene i artikkelen vil det ikke bli fokusert på. Et symmetrisk PWM-signal vil medføre et mindre totalt antall filterkoeffisienter fordi filterkjernene med partalls orden vil være null ved alle n. Ved bruk av et femte ordens filter trengs det kun beregning med tre filterkjerner, hvor førsteordens filterkjerne er simpelthen Diracs deltafunksjon som forenkler beregningen ytterligere.

I. Introduksjon

Nye forsøk [2], [3], [14], [15], [16] har vært basert på analysen av forvrengning generert av digital PWM i det tidsdiskrete domene, som vist i figr 2.15: Det modulerende signal som blir brukt til å lage PWM-signal er et tidsdiskret signal $0 \leq x(n) \leq 1$ med samplingsfrekvens $F_s = 1/T_s$. Hver sample $x(n)$ er konvertert til en PWM-puls, og det resulterende pulstoget blir påført et ideelt (*brick-wall*) analogt filter med *cutoff*-frekvens $F_s/2$ for å rekonstruksjon av den analoge bølgeformen $y(t)$. Siden $y(t)$ er båndbegrenset til $F_s/2$, kan dette signalet blir representert av dens sampler $y(n)$ samplet med frekvens F_s. Bemerk at boksen i figur 2.15 er tegnet med stiplet linje, som indikerer at denne ikke er fysisk realisert, men fungerer som et matematisk innretning for å oppnå en tidsdiskret representasjon til den rekonstruerte bølgeform $y(t)$. Hele ideen er at ved oppsettet forklart, er det mulig å evaluere forvrengningen som er generert av digital PWM ved direkte sammenligning mellom det modulerende signalet $x(n)$ og utgangen $y(n)$. Dette var utnyttet i [2], [3] ved å modellere forholden mellom $x(n)$ og $y(n)$ som et Volterra filter [19]. Dette åpnet for muligheten å prekompensere for den ulineære forvrengningen forårsaket av digital PWM ved å prefiltrere det modulerende signal $x(n)$ med et Volterra filter som har de inverse karakteristikkene til filteretmodellen, som fører til eliminering av ulineær forvrengning i den rekonstruerte bølgeformen $y(t)$.



Figur 2.15: Analyse av digital PWM i tidsdiskrete domene [1].

II. Volterra modeller for digital PWM

A. *Definisjonen til det generelle og diagonale Volterra filter*

Definisjonen på et generelt Volterra filter er gitt ved ligning 2.8 og 2.9, hvor $x(n)$ er inngangen til et Volterra filter og $y(n)$ er utgangen:

$$y(n) = \sum_{q=1}^{\infty} \gamma_q(n) \tag{2.8}$$

hvor

$$\gamma_q(n) = \sum_{i_1} \cdots \sum_{i_q} h_q(n)(i_1, \ldots, i_q) x(n-i_1) \cdots x(n-i_q) \tag{2.9}$$

hvor $h_q(i_1,\ldots,i_q)$ kan bli betraktet som en q-te ordens impulsrespons som karakteriserer den ulineære oppførselen til systemet [12].

Hvis verdiene $i_1 = \cdots = iq = i \neq 0$ vil $h_q(i,\ldots,i) \equiv h_q(i) \neq 0$. I dette tilfelle kan den generelle ligningen bli forenklet til

$$\gamma_q(n) = \sum_i h_q(i)x(n-i)^q = h_q(n) * x(n)^q \tag{2.10}$$

hvor $*$ betegner tids-diskret konvolusjon. Dette gir utgangsverdien

$$y(n) = \sum_{q=1}^{\infty} h_q(n) * x(n)^q \tag{2.11}$$

som er en diagonal Volterra model, også betegnet som en parallell Hammerstein struktur [19].

B. *Generelt oppsett og antagelser*

Det modulerende analoge signalet $x(t)$, hvor $0 \leq x(t) \leq 1$, har en maksimal frekvenskomponent F_x, hvor $F_x < F_s/2$. For å oppnå den tidsdiskrete representasjonen $x(n)$ er signalet samplet ved frekvens F_s. Hver sample $x(n)$ er brukt til å lage en puls med bredde $x(n)T$ og høyde 1. Hvis $x(n) = 1\ \forall n$, vil nabopulser berøre hverandre fordi pulsbredden er T_s. Ved å bruke et ideelt lavpassfilter vil den resultere i rekonstruksjonen av det tidskontinuerlige signalet $y(t) = 1$. Bemerk også at dette er system som ekvivalent til et system med $x(n) \in [-1/2, 1/2]$ og pulsamplituder $\pm 1/2$.

Den generell måten å beregne Volterra filterkjernene for ulike typer PWM-symmetri og ulike typer utgangsfilter, er basert på beregning av konvolusjonsintegral mellom en enkel puls $p(t)$ og impulsresponsen $r(t)$ til demoduleringsfilteret. Siden $r(t)$ er impulsresponsen til et lineært tidsinvariant system, er det mulig å beregne det komplette utgangssignalet laget fra et PWM-signal ved å forskyve og addere utgangsverdiene laget fra hver enkelt puls. Dette er fordi det komplette utgangssignalet består av forskjøvede pulser med ulike bredder. Sampling av utgangssignalet, med en fastsatt båndbredde, vil resultere i ulike ulineære tidsdiskrete modeller for ulike PWM-symmetrier.

For å demodulere et PWM-signal brukes det ofte et lavpassfilter. I denne oppgaven vil kun ideelt $sinc(\cdot)$, eller såkalt *brick-wall* filter bli betraktet.

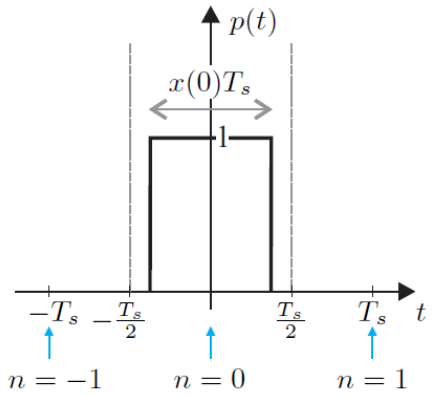
C. *Volterra filterkjerner ved bruk av ideelt analogt lavpassftiler*

Impulsresponsen til et ideelt lavpassfilter med knekkfrekvens $F_s/2$ er gitt ved:

$$r(t) = F_s \frac{sin(\pi F_s t)}{\pi F_s t} = F_s sinc(F_s t) \tag{2.12}$$

hvor $F_s = 1/T_s$ [11].

I digital pulsbreddemodulasjon blir hver sample konvertert til en pulsbredde. Inngangssample x(0) blir omgjort til en puls P(t) sentrert om t=0 med høyde 1 og bredde x(0)Ts. Denne pulsen genererer en respons y(t) ved utgangen til det ideelle filteret r(t) som blir kalkulert ved å bruke konvolusjonsintegral



Figur 2.16: En enkelt puls $p(t)$ (symmetrisk) brukt til å beregne konvolusjonsintegralet til et ideelt lavpassfilter [1].

$$\begin{aligned} y(t) &= \int_{-\infty}^{\infty} r(\tau)p(t-\tau)d\tau \\ &= \int_{t-x(0)\frac{T_s}{2}}^{t+x(0)\frac{T_s}{2}} r(\tau)d\tau \\ &= R\Big(t+x(0)\frac{T_s}{2}\Big) - R\Big(t-x(0)\frac{T_s}{2}\Big) \end{aligned} \tag{2.13}$$

hvor $dR(t)/dt = r(t)$.

Signalet $y(t)$ kan bli samplet ved $t = nT_s$ uten problemer med aliasing fordi filteret $r(t)$ fjerner all frekvenskomponenter over $F_s/2$. Samplet utgangssignal er gitt ved

$$y_n(x) \equiv y(t)|_{t=nT_s} = R\Big(T_s(n+\frac{x}{2})\Big) - R\Big(T_s(n-\frac{x}{2})\Big) \tag{2.14}$$

hvor $x(0) = x$.

Fra videre beregning i [1] finnes ligning for generell filterkjerne ved

$$h_m^s(n) = \frac{T_s^m}{m!2^{m-1}} r^{m-1}(nT_s), \qquad m = 1,3,5,\cdots \tag{2.15}$$

hvor $r^{(m)}(\cdot)$ er den m-te deriverte av impulsresponsen til lavpassfilteret gitt i ligning 2.12.

D. *En komplett diagonal Volterra modell ved bruk av et ideelt analog filter*

Siden lavpassfilteret er et lineært tidsinvariant system, kan det fullstendige utgangsignalet av filteret, som er laget med hensyn på PWM-signal, bli beregnet. Ved å bruke superposisjonsprinsippet, gjøres utregningen ved å summere de tidsforskjøvede utgangsignalene $y_{n-i}(x(i))$ som er generert av hver puls med bredde $x(i)Ts$ som utgjør PWM-signalet. For symmetrisk PWM-signal vil det fullstendige utgangsignalet være

$$y(n) = \sum_{i=-\infty}^{\infty} y_{n-i}(x(i)) = \sum_{q=1}^{\infty}\sum_{i=-\infty}^{\infty} h_{2q-1}^s(n-1)x(i)^{2q-1} = \sum_{q=1}^{\infty} h_{2q-1}^s(n) * x(n)^{2q-1} \tag{2.16}$$

som er summen av resulterende konvolusjoner mellom filtre ${}^{s}_{2q-1}(n)$ og de odde potenser $2q-1$ til inngangsignalet $x(n)$.

III. Inversmodeller

Ved å bruke kunnskapen om inversen av Volterra modellen er det mulig å prekompensere for den ulineære forvrengningen som fremkommer av PWM. Gitt en Volterra model som er karakterisert med filtrene $h_1, h_2, h_3, ...$, finnes en invers Volterra model som er karakterisert med filtrene $f_1, f_2, f_3, ...$, slik at en seriekobling av systemet fører til identitetsfunksjonen. Problemet med å finne et invers Volterra filter for generelle Volterra filter er eksaminert av Fang m.fl. [9]. Ved å bruke en multidimensjonal z-transform metodikk, er det mulig å konstruere p-te ordens lineariseringsoperator for et ulineært system. Fra resultatene i [9], defineres q-te ordens ulineær transferfunksjon av et Volterra filter H ved

$$H_q(z_1,\ldots,z_q) = \sum_{i_1}\cdots\sum_{i_q} h_q(i_1,\ldots,i_q) z_1^{-i_1}\cdots z_q^{-i_1} \tag{2.17}$$

I [9] ble det funnet at inversen til det generelle prefilter var gitt ved følgende transferfunksjoner:

$$\begin{aligned} F_1(z_1) =& H_1(z_1)^{-1} \\ F_p(z_1,\ldots,z_q) =& -H_1(z_1 z_2\cdots z_p)^{-1} \times \sum_{m=2}^{p} \sum_{k_1+\cdots+k_m=p} H_m(z_1\cdots z_k,\ldots,z_{p-k_m+1}\cdots z_p) \\ & \times F_{k_1}(z_1,\ldots,z_{k_1})\cdots F_{k_m}(z_{p-k_m+1},\ldots,z_p), \qquad p=2,3,\ldots \end{aligned} \tag{2.18}$$

hvor den andre summasjonen er over alle mulige kombinasjoner av heltall $k_1,\ldots,k_m$ slik at $k_1,\ldots,k_m = p$. For eksempel for $p=2$ og $m=2$ vil kun kombinasjonen $(k_1 k_2) = (11)$ være inkludert i summen.

Tidligere nevnte Volterra modeller har diagonal filter funksjon, og dette fører til forenkling av utrykkene til det inverse systemet. Dette ble vist i [2] og [3], hvor formlene ved bruk av symmetrisk PWM-signal med ideelt filter [2] og symmetrisk PWM-signal med lav-ordens filter [3] ble utviklet.

A. *Innledning for konstruksjon av inversmodeller*

Et Volterra prefilters utgang $x(n)$ kan skrives

$$x(n) = \sum_{p=1}^{\infty} x_p(n), \tag{2.19}$$

hvor det p-te elementet i summen er gitt som

$$x_p(n) = \sum_{i_1}\cdots\sum_{i_p} f_p(i_1,\ldots,i_p) u(n-i_1)\cdots u(n-i_p), \tag{2.20}$$

hvor $f_p(\cdot)$ er den p-te ordens enhetspulsrespons, eller Volterra filter function. Dens p-dimensjonale z-transform er en p-te ordens ulineær transferfunksjon $F_p(\cdot)$ definert på lik måte som i 2.17.

I praksis vil den P-te ordens approksimasjon $\hat{x}(n) = \sum_{p=1}^{P} x_p(n)$ til det ekte utgangsignalet til filteret $x(n)$ bli brukt til pre-kompensasjon for ulineær forvrengning forårsaket av PWM.

På grunn av den diagonale formen på Volterra filterkjernene $h_m(\cdot)$ kan uttrykket H_m i 2.18 bli forenklet på følgende måte

$$\begin{aligned} H_m(z_1 \cdots z_k, \ldots, z_{p-k_m+1} \cdots z_p) &= \sum_{i_1} \cdots \sum_{i_m} h_m(i_1, \ldots, i_m)(z_1 \cdots z_{k_1})^{-i_1} \cdots (z_{p-k_m+1} \cdots z_p)^{-i_m} \\ &= \sum_{i} h'_m(i)(z_i \cdots z_p)^{-i} \\ &= H'_m(z_1 \cdots z_p) \end{aligned} \tag{2.21}$$

hvor $H'_m(\cdot)$ betegner diagonal form.

Konstruksjonen av inverse modeller avhenger av tre teoremer. Det første teoremet gjelder kun når det brukes symmetrisk PWM-signal, mens de to andre teoremene er et direkte resultat av de diagonale egenskapene til filterkjernene.

Teorem 1: Ved symmetrisk PWM-signaler vil prefilter transferfunksjonene være null for jevn orden, altså $F_p = 0$ ved $p = 2,4,6,\ldots$. Dette teoremet innebærer at i den andre summasjonen i ligning 2.18 trengs det kun å inkludere kombinasjonene hvor alle k_i-ene er odde.

Teorem 2: Gitt m positive heltall $k'_1, \ldots, k'_m$ slik at $k'_1 + \cdots + k'_m = p$ og antall forskjellige permutasjoner av dette settet betegnes ved Q_m. Da vil

$$\sum_{k_1 + \cdots + k_m = p} H'_m F_{k_m} \cdots F_{k_m} = Q_m H'_m F_{k'_1} \cdots F_{k'_m} \tag{2.22}$$

I ligning 2.18, er den andre summasjonen over alle mulige kombinasjoner av heltall $k_1, \ldots, k_m$ slik at $k_1 + \cdots + k_m = p$. Teorem 2 sier at alle kombinasjoner gir identiske uttrykk i summen.

Teorem 3: Gitt p, og m positive heltall $k_1, \ldots, k_m$ slik at $k_1 + \cdots + k_m = p$, vil uttrykket i z-domene

$$H'_m(z_1 \cdots z_p) F_{k_1}(z_1, \ldots, z_{k_1}) \cdots F_{k_m}(z_{p-k_m-1,\ldots,z_p}), \tag{2.23}$$

korrespondere til utrykket i tidsdomene

$$\sum_{i} h_m(i) x_{k_1}(n-i) \cdots x_{k_m}(n-i). \tag{2.24}$$

Det generelle inngang-utgang forholdet i tidsdomene for prefilterets p-te ordens ulineære transferfunksjon F_p er gitt i 2.20. Selv om alle modell-filterkjernene $h_m(\cdot)$ er diagonale, vil ikke prefilterets filterkjerner være diagonale. Derimot nevner teorem 3 en alternativ form for implementasjon av den p-te ordens prefilter komponent hvor inngangsignalene $u(n-i_1) \cdots u(n-i_p)$ erstattes med produktet av tidligere beregnede uttrykk $x_{k_1}(n) x_{k_2}(n) \cdots x_{k_m}(n)$ konvolert med

den (diagonale) modell-filterkjernen $h_m(\cdot)$. I ligning 2.18 er prefilterets p-te ordens ulineære transferfunksjon F_p laget ved å ta summen av komponenter som gitt i 2.23, og påføre et lineært filter på summen $H_1(\cdot)^{-1}$. Se [3] for bevis på teorem 1-3.

I [1] ble det vist at et 5. ordens filter med var nok for å takle PWM-ulinearitetene. Eksperimentene ble gjort ved modulering av symmetrisk PWM-signal med et ideelt demoduleringsfilter. Dette fører til bruken av et prefilter av typen FIR (Finite Impulse Response). De generelle ligningene for inversmodellene er funnet tidligere i artikkelen [1], men ved symmetrisk PWM blir ligningene redusert til:

$$x_1(n) = u(n) \tag{2.25}$$

$$x_3(n) = -h_3(n) * x_1(n)^3 \tag{2.26}$$

$$x_5(n) = -3h_3(n) * [x_1(n)^2 x_3(n)] - h_5(n) * x_1(n)^5 \tag{2.27}$$

$$\hat{x}(n) = x_1(n) + x_3(n) + x_5(n) \tag{2.28}$$

hvor $\hat{x}(n)$ er den fullstendige Volterra prefilter-utgangen og $u(n)$ er det oppsamplede innsignalet (musikk). Filterkjernene er gitt ved

$$h_1(n) = \delta(n)$$

$$h_3(n) = \begin{cases} (-1)^{n+1}/(12n^2) & n \neq 0 \\ -\pi/72 & n = 0 \end{cases}$$

$$h_5(n) = \begin{cases} (-1)^n(\pi^2 n^2 - 6)/(480n^4) & n \neq 0 \\ \pi^4/9600 & n = 0 \end{cases}$$

hvor

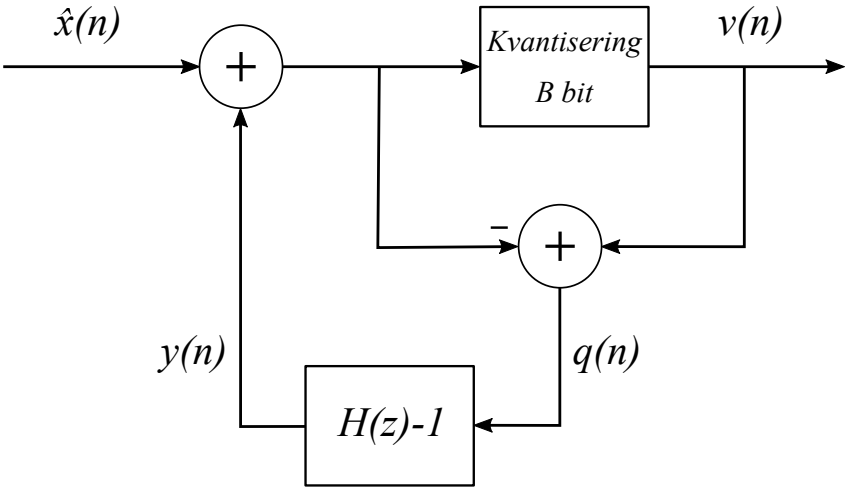
$$h_2(n) = 0 \ \wedge \ h_4(n) = 0 \quad \forall n$$

som fører til

$$x_2(n) = 0 \ \wedge \ x_4(n) = 0 \quad \forall n$$

2.11 Noise Feedback Coding

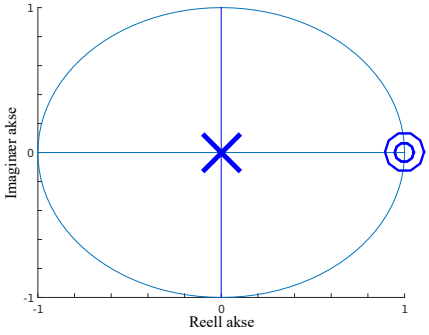
NFC (Noise Feedback Coding) er en type støyforming som har til hensikt å øke signal/støy-forholdet i utgangssignalet fra kvantiseringsprosessen. For ytterligere informasjon om temaet se delkapittel 8.1 i artikkel [2]. Støyforming implementeres ved å tilbakekoble kvantiseringsfeilen. Ved NFC vil det brukes et filter for å fjerne kvantiseringsstøy i hørbart område, som vil si i det omtrentlige område $16 - 20000$ Hz.



Figur 2.17: NFC blokkdiagram basert på figur i [2].

I figur 2.17 er $\hat{x}(n)$ siste sampleverdi fra prefiltrert signal. $v(n)$ er det kvantiserte signalet med B-bits oppløsning og $q(n)$ er kvantiseringsfeilen, som kan anses som hvit støy. Filteret $H(z)$ kan bli konstruert ved å ha P nullpunkt i $1 = e^{j \cdot 0}$:

$$H(z) = (1 - z^{-1})^P = \frac{(1 - z^{-1})^P z^P}{z^P} = \frac{(z-1)^P}{z^P}$$



Figur 2.18: Plot av tilhørende poler og nullpunkt.

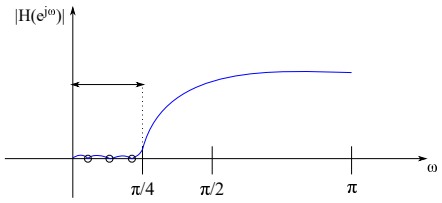
Alternativt kan filteret bli konstruert på en bedre måte ved å ha nullpunktene uniformt fordelt i $[0, \frac{\pi}{N}]$ hvor N er oppsamplingsfaktoren. En samplingsfrekvens på 176,4 kHz vil tilsi en oppsamplingsfaktor $N = 4$, da vil nullpunktene ligge fordelt mellom 0 og 22050 Hz og filtrere støy innenfor dette område. $v(n)$ vil alltid havne akkurat på et av kvantiseringsnivåene.

Eksempel med $P = 3$. Nullpunkt ved $\theta = \frac{\pi}{6N}$, $3\theta = \frac{3\pi}{6N} = \frac{\pi}{2N}$ og $5\theta = \frac{5\pi}{6N}$.

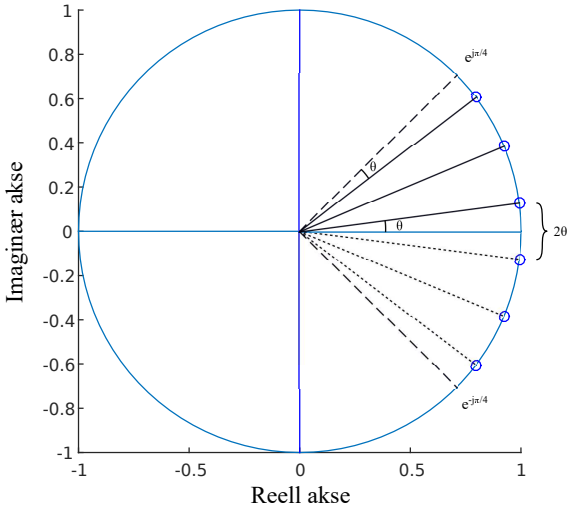
$$H(z) = (z - e^{j\frac{\pi}{6N}})(z - e^{-j\frac{\pi}{6N}})(z - e^{j\frac{\pi}{2N}})(z - e^{-j\frac{\pi}{2N}})(z - e^{j\frac{5\pi}{6N}})(z - e^{-j\frac{5\pi}{6N}}) \cdot z^{-6} \tag{2.29}$$

$$H(z) = (z^2 - 2zcos\frac{\pi}{6N} + 1)(z^2 - 2zcos\frac{\pi}{2N} + 1)(z^2 - 2zcos\frac{5\pi}{6N} + 1) \cdot z^{-6} \tag{2.30}$$

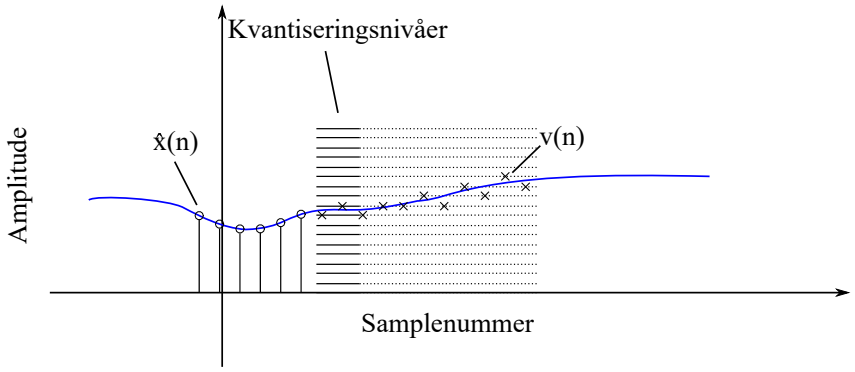
$$H(z) = 1 - 5.4174z^{-1} + 12,7420z^{-2} - 16,6482z^{-3} + 12,7420z^{-4} - 5,4174z^{-5} + z^{-6} \tag{2.31}$$



Figur 2.19: Skisse som viser ønsket frekvensrespons til filteret.



Figur 2.20: Nullpunkter fra eksemplet plottet på enhetsirkelen.



Figur 2.21: Ønsket virkning av NFC. $v(n)$ er kvantisert verdi av $\hat{x}(n)$ og tilsvarer en tilgjengelig pulsbredde. $\hat{x}(n)$ er flyttall i praksis og har mye høyere presisjon enn $v(n)$.

2.12 Analogt lavpassfilter

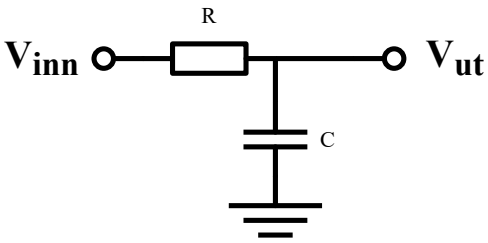
Lavpassfilter er noe som brukes til å filtrere vekk uønskede høye frekvenser av et signal. I praktisk anvendelse kan et førsteordens analogt lavpassfilter bli laget kun ved to elektriske komponenter: en motstand og en kondensator eller en spole. Om ønskelig er det mulig å seriekoble flere filtre for å øke ytelsen og få en brattere skråning i bode-plottet. Et ideelt lavpassfilter slipper gjennom alle frekvenser under en gitt grense og filtrerer andre frekvenser vekk. I praksis vil ikke alle frekvenser over grensen bli filtrert vekk, men jo høyere frekvensen er over grensen desto mer blir den dempet. Skråningen i plottet 2.23 viser dette, her er egenskapene til et førsteordens Butterworth-filter plottet. Ofte brukes Bode-plot for å visualisere virkningsgraden til et forskjellige type filtre. Det er også mulig å konstruere høypass-, båndpass- eller båndstoppfiltre, men disse er ikke brukt i denne oppgaven.

I tillegg kan et lavpassfilter brukes til å rekonstruerte et analogt signal fra et PWM-signal. Dette er en nyttig egenskap som tas nytte av i denne oppgaven.

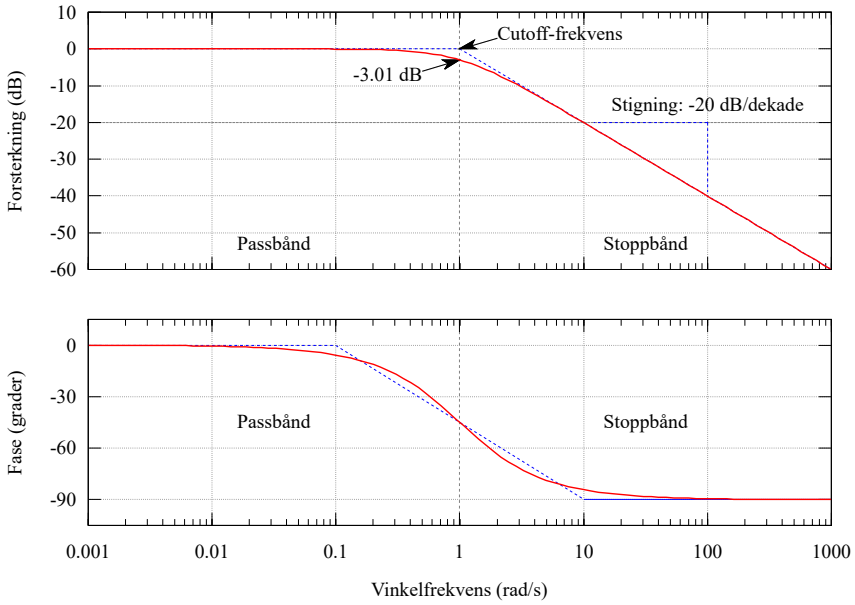
Et analogt lavpassfilter konstruert som vist i figur 2.22, vil ha knekkfrekvens gitt ved

$$f_c = \frac{1}{2\pi RC} \tag{2.32}$$

hvor R er resistans i Ohm og C er kapasitans i Farad.



Figur 2.22: Lavpassfilter konstruert med en resistor og en kondensator.



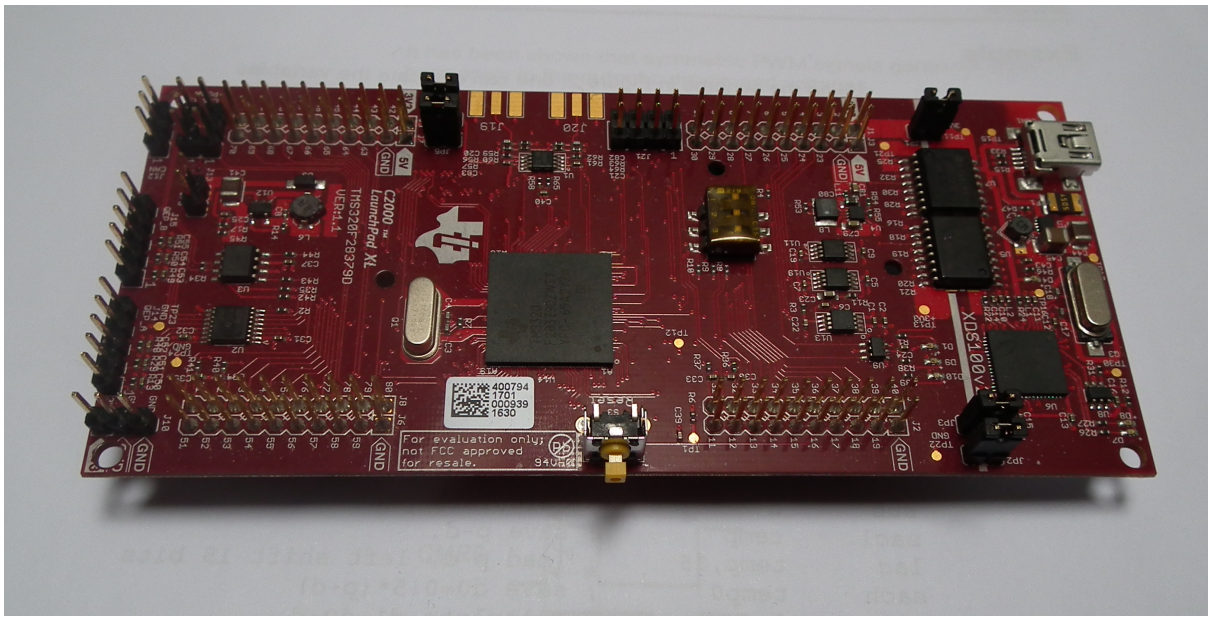
Figur 2.23: Bode-plot av et førsteordens Butterworth-filter [22]. Knekkfrekvensen er normalisert til 1 rad/s og forsterkningen til 0 dB i passbåndet.

3 Implementasjon

3.1 Digital signal kontroller

En DSP (Digital Signal Processor) er en spesialtilpasset mikroprosessor for bruk til signalbehandling i sanntid. En DSC (Digital Signal Controller) er en mellomting mellom en DSP og en mikrokontroller. I en DSC er det eksempelvis tilgjengelig en spesialinstruksjon kalt MAC (Multiply And Accumulate) som gjør flere ting i en syklus, den brukes ofte under filterberegning. Det sto mellom to typer DSP-er og en DSC. Valget falt på DSC-en til fordel for høyere PWM-oppløsning, men med noe lavere beregningshastighet.

Utviklingskortet som ble tatt i bruk under oppgavearbeidet er fra Texas Instruments og heter LAUNCHXL-F28379D, dette inkluderer en TMS320F28379D mikrokontroller (DSC) fra C2000 Delfino-serien. For å programmere kortet ble det brukt Code Composer Studio versjon 7. Måling av signalene ble gjort med gjort med et Keysight-oscilloskop med typenummer MSOX3012T (med 100 MHz båndbredde og maksimal samplerate på 5 GSa/s (5 milliarder sampler per sekund)). Dette oscilloskopet har mulighet til å regne ut DFT-en (Discrete Fourier Transform) til målte signaler, noe som er avgjørende for å kunne se effekten til Volterra prefilteret.



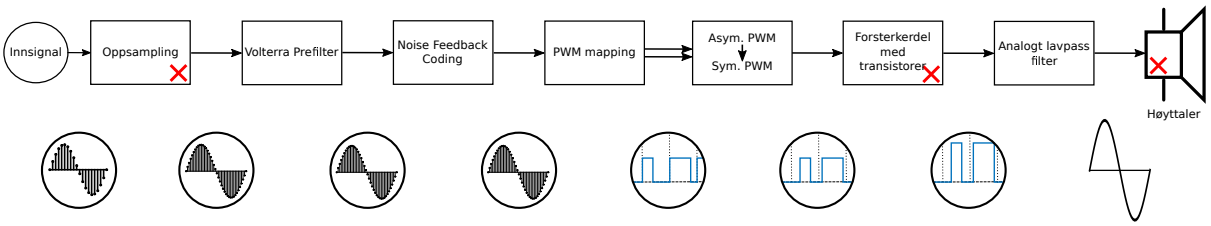
Figur 3.1: Utviklingskortet som ble brukt i prosjektet.

DSC-en har to *fixed-point* prosessorer med ytterligere enheter; FPU (Floating Point Unit), TMU (Trigonometric Math Unit) og VCU-II (Viterbi, Complex Math, and CRC Unit II), som

utvider register- og instruksjonsettet. I tillegg har kortet mange fysiske tilkoblingspunkter hvor de fleste pinnene har flere funksjoner. En nyttig funksjon er PWM-signal-generering. Kortet har flere utganger for å generere PWM-signal hvor noen utganger har høyoppløselig modus i tillegg, men disse virker kun i asymmetrisk modus (nærmere forklaring er gitt i delkapittel 3.6). Det er ønskelig å ta i bruk høyoppløselig modus og i tillegg ha symmetrisk PWM-signal. En løsning på dette er å bruke to signalgeneratorer i asymmetrisk modus og en XOR-port for å få et resulterende symmetrisk PWM-signal.

3.2 Fullstendig oppsett

I dette oppgavearbeidet har det blitt utviklet deler av en heldigital effektforsterker. Konstruksjonen består av flere deler hvor flesteparten er implementert på en DSC (Digital Signal Controller). Volterra prefiltrering, NFC og PWM-mapping blir alle gjort av DSC-en. En eventuell oppsampling ville også blitt implemtert av DSC-en, men er sett bort i fra da testsignalet kan bli laget direkte ved den korrekte samplingsfrekvensen i kildekoden. I første omgang skal signalbehandlingen bli testet med rene sinussignaler, og det er ikke behov for oppsampling før mer kompliserte lydsignaler skal testes eller implementasjon som fungerer i sanntid skal utføres. Videre må det analoge lavpassfilteret implementeres med fysiske komponenter. Det analoge lavpassfilteret virker som et demoduleringsfilter, som betyr at det påførte PWM-signalet blir demodulert til det opprinnelige analoge signalet, selv om prosessen kun har håndtert den tidsdiskrete representasjonen av signalet. Det er heller ikke en nødvendighet å implementere signalforsterkerdelen for å kunne bekrefte ytelsen til prefilteret, da signalforsterkerdelens hensikt er å forsterkere signalet tilstrekkelig for driving av en høyttaler. Bemerk at konstruksjon av signalforsterkerdelen ikke var en del av denne oppgaven. For å måle signalene er kortets PMW-utganger koblet direkte på et analogt lavpassfilter. Ved å velge korrekte komponentverdier under implementasjon av lavpassfilteret, kan strømmen ut fra PWM-portene holdes lave nok for å unngå skade på utviklingskortet (< 3 mA). Figur 3.2 viser et oppsett av en forestilt implementasjon av en komplett heldigital effektforsterker med en oppsamplingsdel. Bemerk delene merket med kryss, da disse ikke er blitt implementert i dette oppgavearbeidet.



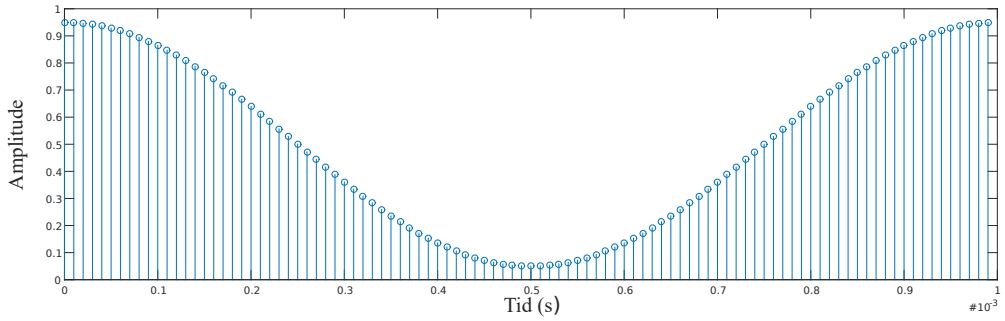
Figur 3.2: Oppsett av komplett konstruksjon. Symmetrisk PWM-signal blir laget ved å bruke to asymmetriske PWM-signaler.

3.3 Innsignal

For testing av virkemåten til prefilteret brukes rene sinuskurver med frekvenser på 500 Hz, 1 kHz og 2 kHz. Signalet blir generert i kildekoden med en samplingsfrekvens på 100 kHz, som da tilsvarer en fire gangers oppsampling fra 25 kHz. Oppgaven er egentlig lagt opp for å bruke et

standard musikksignal (samplingsfrekvens: 44,1 kHz) og sample opp dette. Eksempelvis med en oppsamplingsfaktor på fire vil ny samplingsfrekvens bli 176,4 kHz. På grunn av manglende optimalisering av beregningstid, ble det derfor valgt en noe lavere samplingsfrekvens. Videre tenkes det at menneskelig hørselsområde er omtrent $16-12500$ Hz. NFC-delen vil da filtrere kvantiseringsstøy i dette område.

Testsignalenes amplituder ligger i intervallet $[0,05-0,95]$, dette er for å holde amplitudene innenfor $[0-1]$ etter prefilterets påførte endringer. Figur 3.3 viser et 1 kHz testsignal. Det er ikke behov for lengre testsignal, grunnet signalets periodiske egenskaper og perfekt overgang ved gjentakelse av testsignalet. Signalene som er generert er flyttall, og har en høyere presisjon enn PCM-signal som har vanligvis 16-bits oppløsning. Testsignal på 500 Hz har behov for 200 sampler for å oppnå perfekt periodisitet, mens 2 kHz signalet trenger kun 50 sampler. Testsignalet på 1 kHz trenger 100 sampler for å oppnå perfekt periodisitet, se figur 3.3.



Figur 3.3: Testsignal med 100 sampler: En periode av en 1000 Hz sinuskurve (cosinus-funksjon).

3.4 Volterra prefilter

Teoretisk bakgrunn for Volterra prefilter er forklart i delkapittel 2.10. Artikkelen [1], som forklaringen er basert på, omhandler flere eksperimenter med ulike prefilter-implementasjoner. Det er i denne oppgaven valgt å implementere et femte ordens Volterra prefilter med modulering av symmetrisk PWM-signal og ideelt demoduleringsfilter. Dette fører til bruken av et prefilter av typen FIR (Finite Impulse Response). Filterkjernene er vist i delkapittel 2.10. De forenklede ligningene er også vist i dette delkapittel, men er gjentatt her da det tas utgangspunkt i disse i videre beregning:

$$x_1(n) = u(n) \tag{3.1}$$

$$x_3(n) = -h_3(n) * x_1(n)^3 \tag{3.2}$$

$$x_5(n) = -3h_3(n) * [x_1(n)^2 x_3(n)] - h_5(n) * x_1(n)^5 \tag{3.3}$$

der

$$\hat{x}(n) = x_1(n) + x_3(n) + x_5(n) \tag{3.4}$$

og $u(n)$ er det oppsamplede signalet (musikk).

På grunn av trunkering av filterkjernene er $L = 50$ i videre beregninger. Filterkjernene er ikke-kausale, det er derfor behov for å forsinke signalet for å oppnå kausal filtrering. Ikke-kausal konvolusjon i ligning 3.2:

$$x_3(n) = -h_3(n) * x_1(n) = -\sum_{m=-L}^{L} h_3(m) x_1(n-m)^3 \tag{3.5}$$

Ved tidspunkt n trengs det følgende sampleverdier: $x_1(n-L)$ ... $x_1(n+L)$. Forsinker med L sampler og oppnår kausal konvolusjon:

$$x_3(n-L) = -\sum_{m=-L}^{L} h_3(m) x_1(n-L-m)^3 \tag{3.6}$$

Ved tidspunkt n trengs det derimot følgende sampleverdier $x_1(n-2L)$ ... $x_1(n)$ for å kunne beregne $x_3(n-L)$.

Ligning 3.3 innebærer også ikke-kausal konvolering:

$$\begin{aligned} x_5(n) &= -3h_3(n) * [x_1(n)^2 x_3(n)] - h_5(n) * x_1(n)^5 \\ &= -3 \sum_{m=-L}^{L} h_3(m)[x_1(n-m)^2 x_3(n-m)] - \sum_{m=-L}^{L} h_5(m) x_1(n-m)^5 \end{aligned} \tag{3.7}$$

Beregning av $x_5(n)$ trenger både $x_1(n-L)$ ... $x_1(n+L)$ og $x_3(n-L)$ ... $x_3(n+L)$ ved tidspunkt n. For å oppnå ikke-kausal konvolusjon er det nødvendig å forsinke med 2L sampler, fordi $x_3(n-L)$ er siste tilgjengelige verdi:

$$\begin{aligned} x_5(n-2L) = &-3 \sum_{m=-L}^{L} h_3(m)[x_1(n-2L-m)^2 x_3(n-2L-m)] \\ &- \sum_{m=-L}^{L} h_5(m) x_1(n-2L-m)^5 \end{aligned} \tag{3.8}$$

Trenger her $x_1(n-3L)$ ... $x_1(n-L)$ og $x_3(n-3L)$ ... $x_3(n-L)$ ved tidspunkt n for å beregne $x_5(n-2L)$, som er fullstendig kausal filtrering. Derfor vil det ved tidspunkt n være mulig å regne ut $\hat{x}(n-2L)$. Implementasjon av prefilteret vil forsinke signalet med $2L$ (100) sampler, som svarer til en tidsforsinkelse på 1 ms (100 kSa/s) av musikksignalet. Ved 176,4 kSa/s ville forsinkelse vært på omtrent 0,567 ms.

På grunn av symmetri i filtrene er det mulig å spare multiplikasjoner under implementasjonen. Derfor kan ligning 3.6 og 3.8 henholdsvis bli forenklet til:

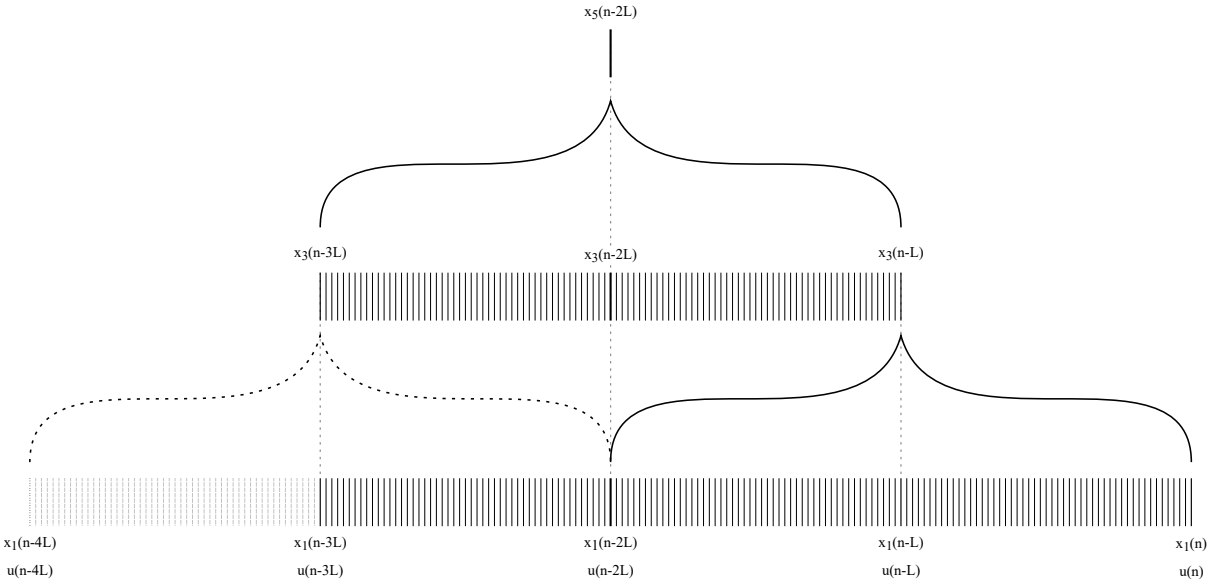
$$x_3(n-L) = -h_3(0) x_1(n-L)^3 - \sum_{m=1}^{L} h_3(m)[x_1(n-L-m)^3 + x_1(n-L+m)^3] \tag{3.9}$$

og

$$
\begin{aligned}
x_5(n-2L) = &-3h_3(0)x_1(n-2L)^2x_3(n-2L) \\
&-3\sum_{m=1}^{L} h_3(m)[x_1(n-2L-m)^2x_3(n-2L-m)+x_1(n-2L+m)^2x_3(n-2L+m)] \\
&-h_5(0)x_1(n-2L)^5-\sum_{m=1}^{L} h_5(m)(x_1(n-2L-m)^5+x_1(n-2L+m)^5)
\end{aligned}
\tag{3.10}
$$

```
// Les inn nytt signal og gjør potensberegninger.
x1 = innSignal[i];
x12 = x1*x1;
x13 = x12*x1;
x15 = x12*x13;

// Legg inn i buffere. Skriv-posisjon økes.
CBUF_Push(X1buf,x1);   // Forsinket med 100
CBUF_Push(X1buf2,x12); // Forsinket med 150
CBUF_Push(X1buf3,x13); // Forsinket med 150
CBUF_Push(X1buf5,x15); // Forsinket med 150

// Regn ut x3(n-50).
x3 = -h3[0]*CBUF_Get(X1buf3,50);
for(j = 1; j <= 50; j++)
{
    x3 -= h3[j]*(CBUF_Get(X1buf3,50-j)+CBUF_Get(X1buf3,50+j));
}
// Legg inn i buffer. Skriv-posisjon økes.
CBUF_Push(X3buf,x3);

// Regn ut x5(n-100).
x5 = -3*h3[0]*CBUF_Get(X1buf2,50)*CBUF_Get(X3buf,50) - h5[0]*CBUF_Get(X1buf5,50);

for(j = 1; j <= 50; j++)
{
    x5 -= 3*h5[j]*(CBUF_Get(X1buf2,50-j)*CBUF_Get(X3buf,50-j)+CBUF_Get(X1buf2,50+j)*CBUF_Get(X3buf,50+j));
    x5 -= h5[j]*(CBUF_Get(X1buf5,50-j)+CBUF_Get(X1buf5,50+j));
}

// Regn ut xhatt, henter x1(n-100) og x3(n-100) fra buffere
xhatt = CBUF_Pop(X1buf) + CBUF_Get(X3buf,50) + x5;

// Øk les-posisjon i buffere med 1. Trenger ikke for X1buf, da Pop-funksjonen i tillegg gjør dette
X1buf2.m_getIdx++;
X1buf3.m_getIdx++;
X1buf5.m_getIdx++;
X3buf.m_getIdx++;
```

Figur 3.4: Utdrag fra kode som viser beregning av $\hat{x}(n-2L)$ ved å bruke ligning 3.9 og 3.10. Her brukes buffere for å lagre verdiene for bruk i senere beregning (forsinket).



Figur 3.5: Figur som illustrerer forsinkelsen og hvilke sampler som trengs for å beregne $x_3(n-L)$ og $x_5(n-2L)$. Ved tidspunkt n blir forsinket prefiltrert signal utregnet som $\hat{x}(n-2L) = x_1(n-2L)+x_3(n-2L)+x_5(n-2L)$.

3.5 NFC-beregning

Etter signalet er blitt prefiltrert, skal det kvantiseres for PWM-mapping. Det er mye lavere presisjon på oppløsningen på pulsbreddene til forskjell fra flyttall, og det er derfor nødvendig å redusere kvantiseringsstøyen ved hjelp av NFC.

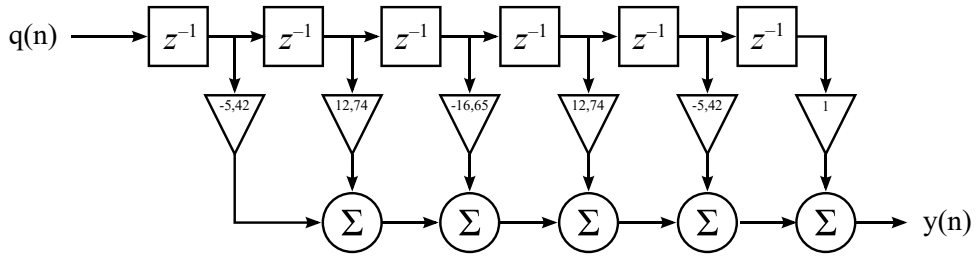
Først må signalet bli kvantisert, altså avrundet til nærmeste mulige pulsbreddeverdi. Det er tilgjengelig 28500 unike symmetriske pulsbreddeverdier. Kvantisert signal finnes ved:

$$v(n) = \text{round}\left(\hat{x}(n) \cdot 28500\right) \cdot \frac{1}{28500} \tag{3.11}$$

Her er da kvantiseringsfeilen

$$q(n) = v(n) - \hat{x}(n) \tag{3.12}$$

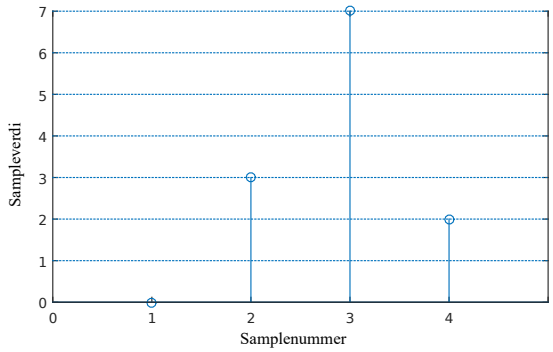
Selve utregningen av NFC-verdien $y(n)$ gjøres med et sjetteordens FIR-filter som vist i figur 3.6.



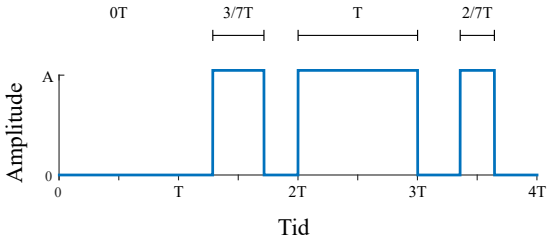
Figur 3.6: NFC FIR-filter, med brukte koeffisienter.

3.6 PWM-mapping

PWM-mapping er en type pulsbreddemodulasjon som går ut på å konvertere sampler fra et tidsdiskret signal til pulsbredder i et PWM-signal. Ideelt sett ville da amplituden til et tidsdiskret signal vært identisk til pulsbredden. Maks amplitude hadde tilsvart en pulsbredde på en periode T, altså arbeidssyklus 100 %. I figur 3.7 vises ideell symmetrisk PWM-mapping, selv om dette er med lav bitoppløsning.



(a) Fire sampler i et PKM signal.

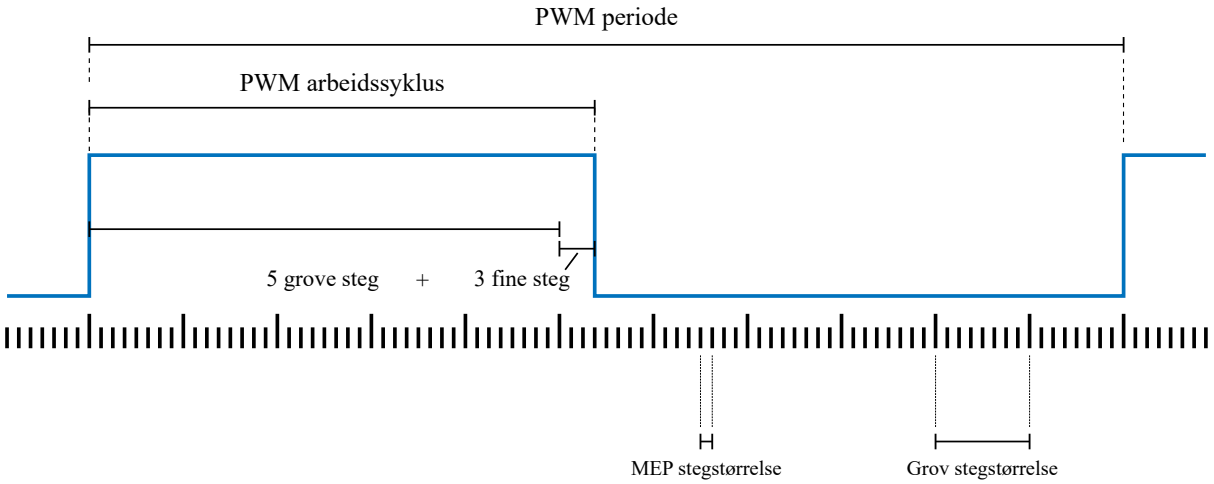


(b) Tilsvarende sampler konvertert til pulsbredder.

Figur 3.7: Ideell symmetrisk PWM-mapping ved 3-bits oppløsning.

Utviklingskortet har to PWM-moduser kalt ePWM (Enhanced Pulse Width Modulation) og HRPWM (High Resolution Pulse Width Modulation). I asymmetrisk modus plasserer ePWM-modulen flanken, enten *leading-* eller *trailing-edge*, med grove steg med størrelse lik en PWM-klokkesyklus. HRPWM er en modus som finjusterer flanken ytterligere for å gi større oppløsning for sykluslengden. Denne modulen har en egen MEP-klokke (Micro Edge Positioner) som har høyere oppløsning enn PWM-klokken. Dette gjør det mulig å plassere flanken innenfor en PWM-klokkesyklus. Typisk oppløsning på MEP-stegstørrelse er 150 ps [17, s. 130]. Denne tiden øker med lavere forsyningsspenning og/eller høyere temperatur. Texas Instruments leverer derfor en funksjon kalt SFO (Scale Factor Optimization) med i biblioteket for DSC-en. Denne funksjonen sjekker forholdene og kalibrerer antall tilgjengelige MEP-steg deretter. Siden MEP-stegstørrelsen kan endre seg vilkårlig er det behov for bruke SFO-funksjonen.

ePWM-modulen består av flere undermoduler, blant andre *Time-Base*, *Counter-Compare* og *Action-Qualifier*. *Time-Base* undermodul holder kontroll på PWM-perioden ved å telle antall PWM-klokkesykluser og når ønsket periode er nådd, nullstilles telleren. I asymmetrisk modus kan det enten telles fra null og til ønsket verdi eller motsatt vei, for å generere *trailing-edge*- eller *leading-edge*-pulser henholdsvis. Undermodulen *Counter-Compare* tar i bruk den tidsbaserte telleren for å determinere arbeidssyklus. Ønsket arbeidssyklus blir oppgitt i PWM-klokkesykluser (må være mindre enn perioden) og når tidsbasert teller når denne verdien, blir signal sendt til *Action-Qualifier* som plasserer ePWM-flanken. I en enkel PWM-klokkeysklus vil HRPWM-modulen starte å telle MEP-steg og omplassere flanken for å finjustere arbeidssyklusen, når en ønsket MEP-stegverdi er nådd. Se figur 3.8.



Figur 3.8: Illustrasjon som viser stegstørrelser og flankeplassering.

3.6.1 Symmetrisk PWM-signal ved bruk av XOR

På grunn av kortets manglende evne til å bruke høyoppløselig modus samtidig med symmetrisk PWM-mapping, er det brukt to PWM-utganger med asymmetrisk mapping koblet på en XOR-port (ekslusiv-eller) for å generere et resulterende symmetrisk PWM-signal. Figur 3.9 viser alle mulige kombinasjoner av to asymmetriske PWM-signaler ved 2-bits oppløsning. Her vil mange kombinasjoner gi ugyldige resultat, men samtidig gir forskjellige kombinasjoner samme gyldige resultat. Totalt sett er det 8 gyldige muligheter for symmetrisk utgang:

Inngang		**Utgang**
A	**B**	**Y**
0	0	0
0	1	1
1	0	1
1	1	0

Tabell 3.1: Sannhetstabell til en XOR-port. $Y = A \oplus B = \bar{A}B + A\bar{B}$

4 gir $0 \cdot T_s$ $(00 \oplus 00,\ 01 \oplus 01,\ 10 \oplus 10,\ 11 \oplus 11)$

2 gir $1 \cdot T_s$ $(00 \oplus 11,\ 11 \oplus 00)$

2 gir $\frac{1}{3} \cdot T_s$ $(01 \oplus 10,\ 10 \oplus 01)$

Denne løsningen reduserer oppløsningen på det symmetriske PWM-signalet noe, pulsbredden $\frac{2}{3} \cdot T_s$ er ikke lenger mulig. For å kunne oppnå denne pulsbredden må flankene plasseres med en tidsoppløsning på $\frac{1}{6} \cdot T_s$, mens 2-bits asymmetrisk PWM-signal har en tidsoppløsning på $\frac{1}{3} \cdot T_s$.

$$\frac{T_s}{3} = \frac{T_s}{2^2 - 1}$$

Et 3-bits asymmetrisk PWM-signal har en tidsoppløsning på:

$$\frac{T_s}{7} = \frac{T_s}{2^3 - 1}$$

Generelt kan da tidsoppløsningen til et n-bits asymmetrisk PWM-signal skrives:

$$\frac{T_s}{2^n - 1}$$

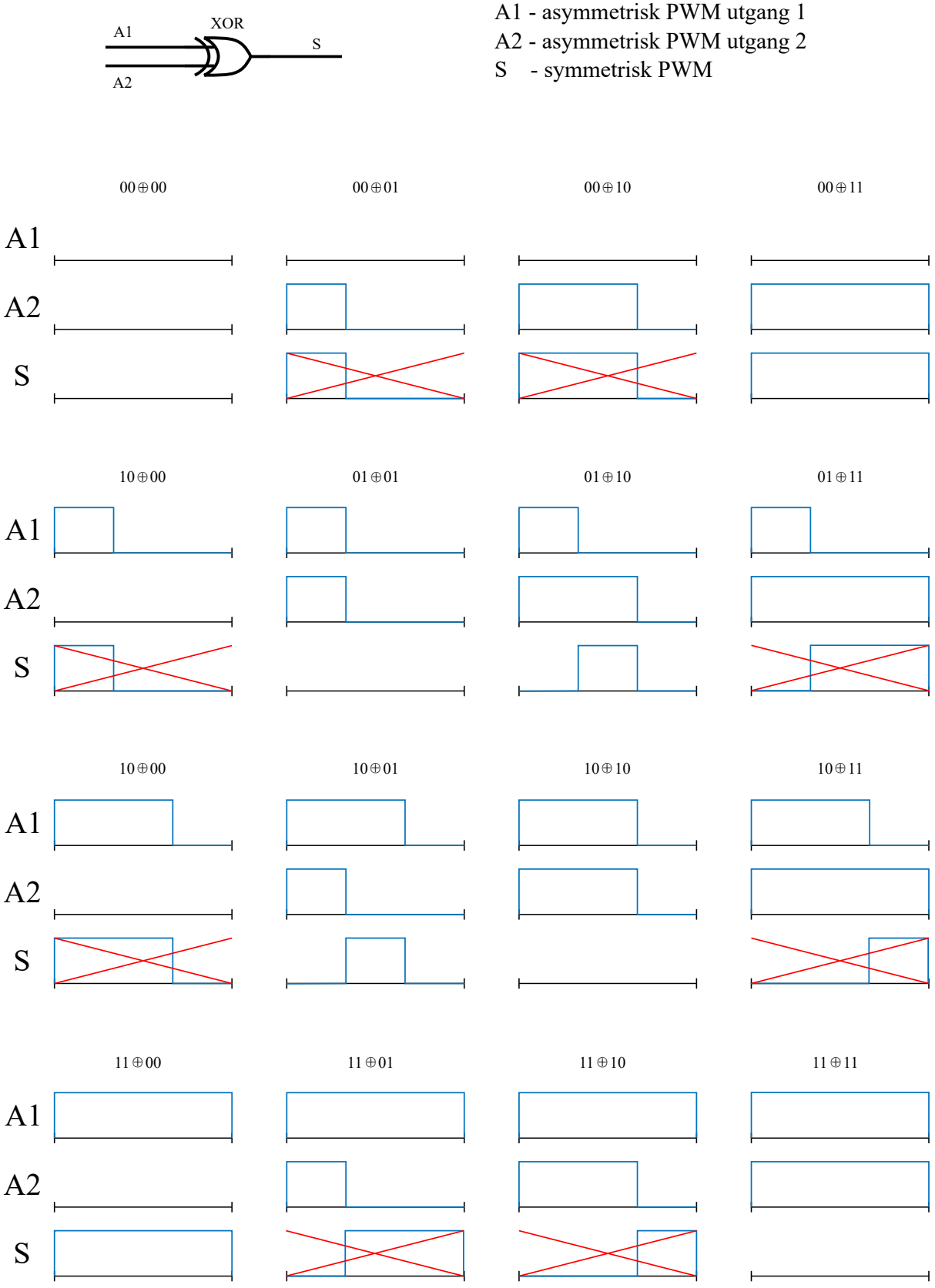
Nødvendig tidsoppløsning for 2-bits symmetrisk PWM-signal:

$$\frac{T_s}{6} = \frac{T_s}{2(2^2 - 1)}$$

Den nødvendige interne tidsoppløsning for to asymmetriske PWM-moduler som lager et symmetrisk PWM-signal er to ganger høyere enn for tilsvarende asymmetrisk PWM. Generelt blir det da

$$\frac{T_s}{2(2^n - 1)}$$

hvor n er antall bits. I praksis vil tidsoppløsningen bli bestemt av MEP-stegstørrelse og valg av periode T_s.



Figur 3.9: Illustrasjon som viser konvertering fra asymmetrisk til symmetrisk PWM-signal ved bruk av XOR-port. Her vises alle mulige kombinasjoner av asymmetrisk PWM-signal med 2-bits oppløsning. Ugyldige kombinasjoner som ikke gir symmetrisk PWM er krysset ut.

3.6.2 Oppløsning og beregning av symmetrisk PWM

Ved en samplingsfrekvens på 100 kHz vil en periode tilsi 10 µs. Kortets systemklokkefrekvens er satt til 200 MHz, og PWM-klokkefrekvensen er satt til 100 MHz, begge maksimale verdier. En PWM-klokkesyklus er 10 ns, og det brukes 1000 sykluser for å sette en PWM-signalperiode på 10 µs. Antall tilgjengelige MEP-steg finnes ved bruk av SFO-funksjonen da antall steg kan endres grunnet temperatur- og spenningsendringer. Funksjonene blir kjørt hver avbruddsrutine for kontinuerlig oppdatering av MEP-stegstørrelse.

Navn	Frekvens	Periode
Systemklokke	200 MHz	5 ns
PWM-klokke	100 MHz	10 ns
PWM-signal	100 kHz	10 µs
Testsignal	1 kHz	1 ms

Tabell 3.2: Alle frekvenser med tilsvarende perioder.

Fra funksjonen ble det utregnet antall MEP-steg per PWM-periode til normalt 57. Tidsoppløsning finnes ved:

$$\Delta = \frac{10 \cdot 10 - 9}{57} \approx 1,754 \cdot 10^{-10}\ s = 175,4\ ps \tag{3.13}$$

som er noe høyere enn typisk størrelse oppgitt i databladet: 150 ps [17, s. 130]. Den minste tilgjengelig tidsbyggeblokk tilgjengelig er Δ. Fra resonnementet i forrige underkapittel er derfor tidsoppløsningen på det symmetriske PWM-signalet 350,8 ps. Per periode i asymmetriske PWM er det tilgjengelig 57000 ulike flankeplasseringer. Siden dette skal brukes til representere symmetrisk PWM, er det kun tilgjengelig 28500 unike flankeplasseringer i symmetrisk PWM.

I NFC-beregningen vil kvantisert signal $v(n)$ bli utregnet. Fordi signalet allerede er kvantisert skal amplitude tilsvare en tilgjengelig pulsbredde, men siden symmetrisk PWM-signal skal bli representert ved to asymmetriske PWM-signaler må noe beregning til.

Først finnes kvantisert symmetrisk pulsbredde, men i asymmetrisk oppløsning som antall tidsbyggeblokker Δ.

$$\tau_s = v(n) \cdot (T \cdot \rho) \tag{3.14}$$

Deretter finnes antall grove steg fra midtpunktet flanken skal bli plassert. Her brukes halvparten av kvantisert pulsbredde til å finne antall grove steg ved å dele på tilgjengelig MEP-steg og runde ned til nærmeste heltall. Dette er en midlertidig verdi som brukes for videre beregning.

$$\alpha = \left\lfloor \frac{\tau_s/2}{\rho} \right\rfloor \tag{3.15}$$

Når antall grove steg fra midtpunktet α er funnet, trengs det kun å adderes med halvparten av totale grove steg. Dette gjelder for høyre grove flankeplassering.

$$\tau_{A2} = \frac{T}{2} + \alpha \tag{3.16}$$

For venstre grove flankeplassering subtraheres $(\alpha + 1)$ fra midtpunktet.

$$\tau_{A1} = \frac{T}{2} - \alpha - 1 \tag{3.17}$$

Neste steg er å beregne finjusteringen av flankeplasseringene. Antall fine steg γ blir beregnet ut fra avrundingsfeilen fra beregning av α. Et grovt steg tilsvarer 57 fine steg og α gjøres om til fine steg. Beregning av antall fine steg på høyre flanke:

$$\gamma_{A2} = \frac{\tau_s}{2} - \alpha \cdot \rho \tag{3.18}$$

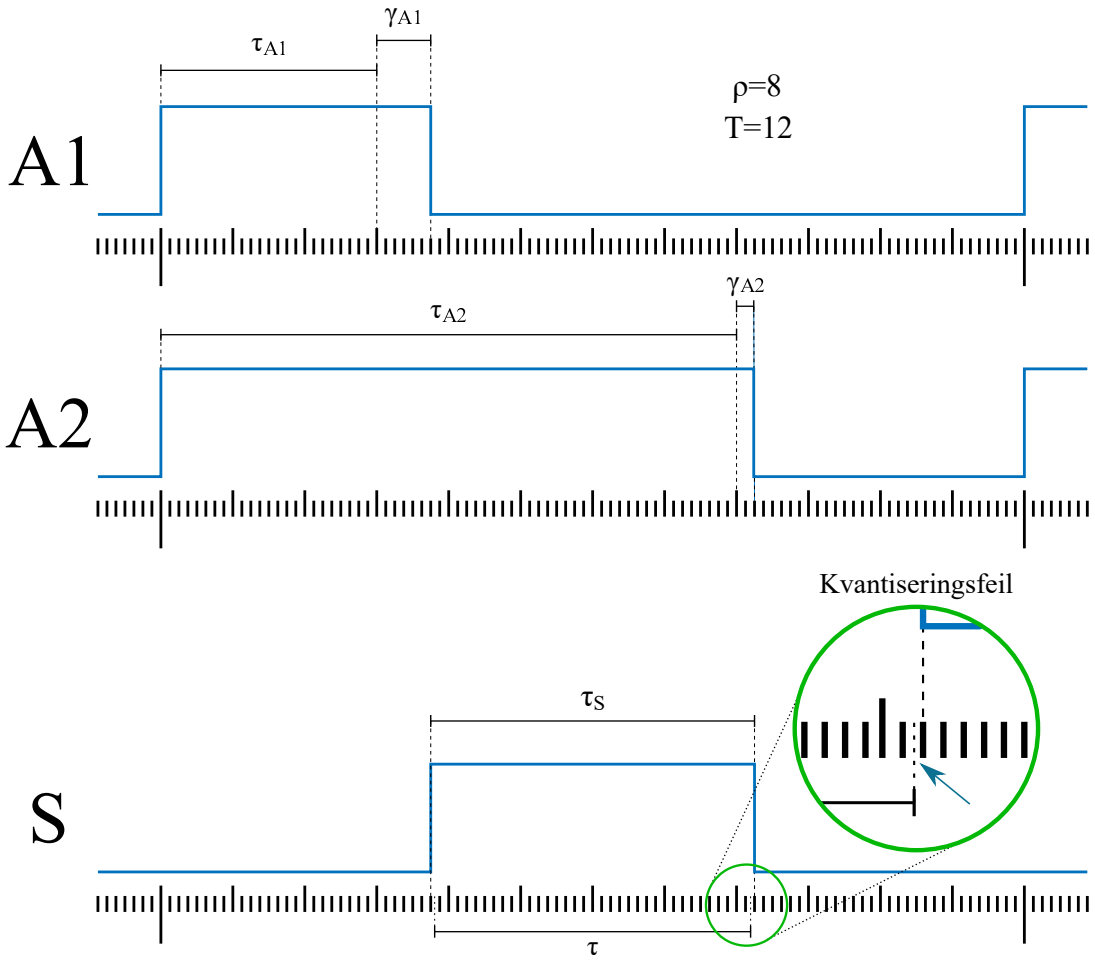
Siden det skal være perfekt symmetrisk ($\gamma_{A1} + \gamma_{A2} = \rho$), finnes antall fine steg på venstre flanke ved:

$$\gamma_{A1} = \rho - \gamma_{A2} \tag{3.19}$$

I koden er denne fremgangsmåten brukt for å plassere flanker. Til et eksempel er $T = 12$, $\rho = 8$ og $\hat{x} + y = 0,3659$ (y er NFC-verdi) som blir kvantisert til $v = 0,375$. Disse tallene vil da gi $\tau_{A1} = 3$, $\tau_{A2} = 8$, $\gamma_{A1} = 6$ og $\gamma_{A2} = 2$. Sampleverdier fra kvantisert prefiltrert signal skal da korrespondere til pulsbredder i det resulterende signalet S. I figur 3.11 er flankene plassert ved bruk av variablene i eksemplet.

```
// ----------------------------------- PWM-mapping ------------------------------------------------
// Se kapittel PWM-mapping / Oppløsning og beregning av symmetrisk PWM for nærmere forklaring.


stegTotal  = v*asymSteg;                    // Størrelse på symmetrisk pulsbredde, oppgitt i antall fine steg
halvTotal  = stegTotal/2;                   // Halverer for å finne avstanden fra midtpunktet til flankene.
temp       = halvTotal/MEP_ScaleFactor;     // Finner antall grove steg. Et grovt steg = 57 fine steg.
                                            // Cast to int. Runder ned uten å bruke floor().

// Regner ut antall grove og fine steg for å bestemme høyre flanke
grovStegA2 = 500 + temp;
finStegA2  = halvTotal - temp*MEP_ScaleFactor;

// Regner ut antall grove og fine steg for å bestemme venstre flanke
grovStegA1 = 500 - temp - 1;
finStegA1  = MEP_ScaleFactor-finStegA2;
```

Figur 3.10: Kodeutdrag fra PWM-mappingsdelen. Det er ikke tatt i bruk greske symboler i koden, da det kan være lettere å forstå selvforklarende variabelnavn.



Figur 3.11: Eksempel på PWM-mapping med illustrert kvantiseringsfeil. Feilen vil være symmetrisk og like stor på begge flanker.

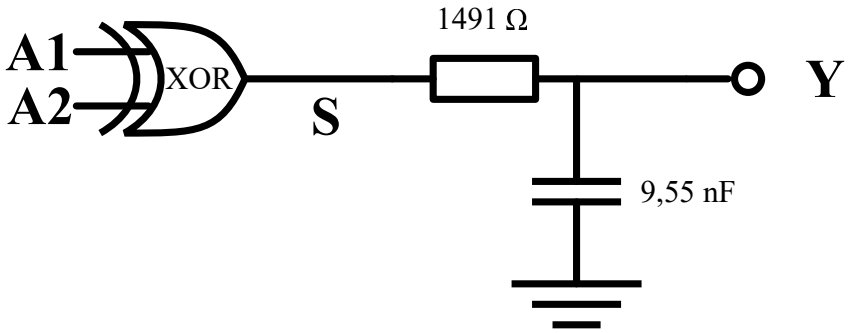
3.7 Demoduleringsfiltrering

Som nevnt i forrige delkapittel er det behov for å bruke en XOR-port for å oppnå symmetrisk PWM. Det har blitt tatt i bruk en XOR IC (Integrated Circuit), med nummer MM74HC86N. All frekvensanalyse er basert på målinger av analogt signal ved Y. For å vise virkemåten til demoduleringsfilteret er det foretatt målinger ved både S (symmetrisk PWM-signal) og Y (rekonstruert analogt signal) samtidig.

For beregning av knekkfrekvensen er det brukt målte verdier av de elektriske komponentene:

$$f_{knekk} = \frac{1}{2\pi \cdot 1491 \cdot 9,55 \cdot 10^{-9}} \approx 11,2\ kHz \tag{3.20}$$

Frekvensresponsen til et førsteordens analogt lavpassfilter av RC-typen (Resistor-Capacitor) vil ha en *roll-off* (dempningsfaktor) på -20 dB/dekade. Det betyr at frekvens på 112 kHz kun er dempet med -20 dB. Dette vil føre til en synlig svitsjefrekvens (PWM-frekvens) i det rekonstruerte signalet. Da frekvensanalysen er begrenset til 10 kHz, vil ikke slik høyfrekvente signaler være med i grafen. I praktisk sammenheng, med tanke på driving av høyttaler, vil de høyfrekvente signalene ikke bidra til hørbart musikksignal. Dette er grunnet høyttalerelementets manglende evne til å vibrere ved høye frekvenser og at mennesker ikke kan oppfatte høyfrekvente lyder. På nettsiden [25] er det nevnt at dyre diskanthøyttalere kan i ekstreme tilfeller ta seg av frekvenser opp til rundt 70 kHz. Vanligvis har høyttalere egen frekvensrespons som har passbåndet et sted i menneskelig hørselsområde. Bass tar seg av de lave frekvens (lavpass), diskant tar seg av de høye frekvensene (høypass) og mellomtone (båndpass) tar seg av frekvensene i mellom.



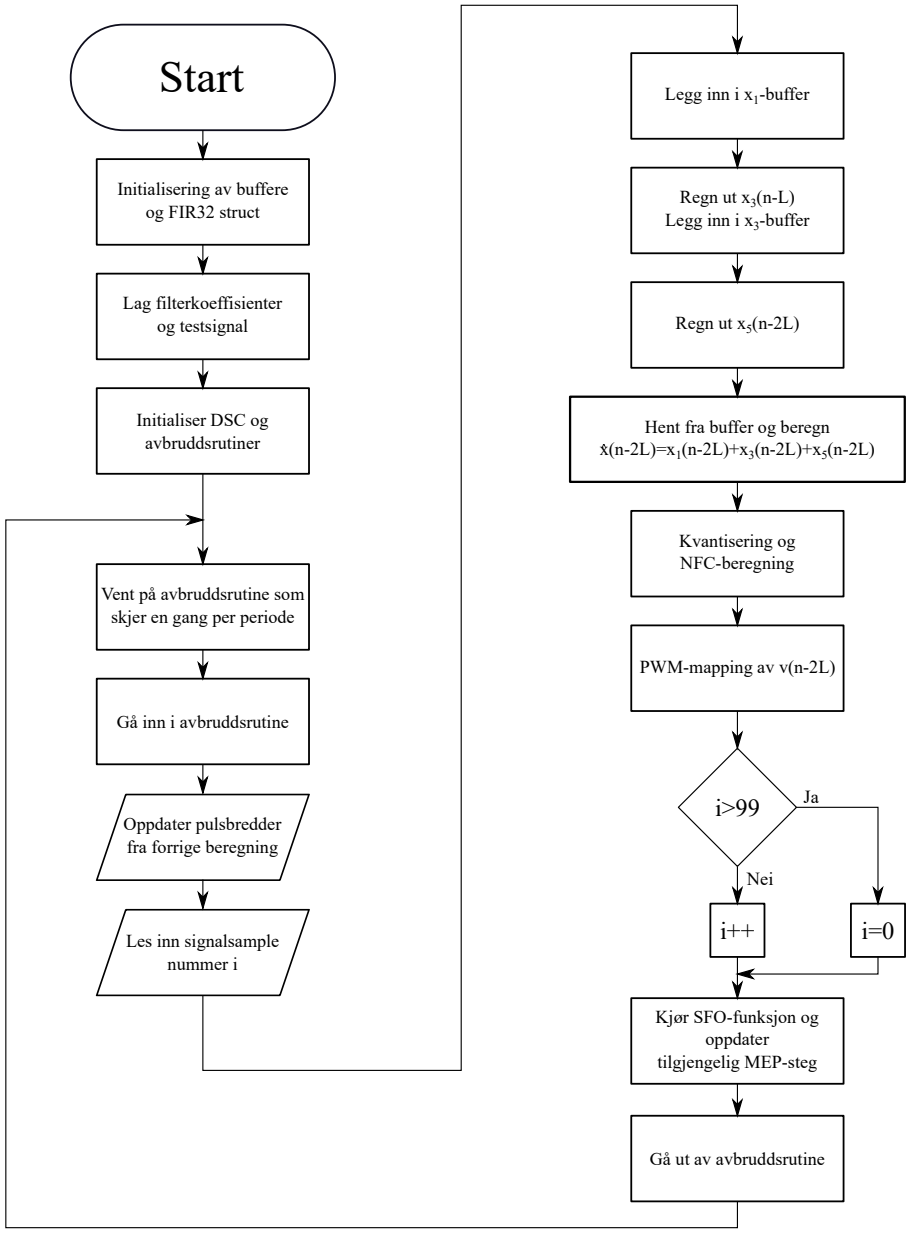
Figur 3.12: Kretstegning av demoduleringsfilteret med XOR-port. A1 og A2 er asymmetriske PWM-signaler fra DSC-en.

3.8 Programoppbygging

Code Composer Studio (CCS) er et gratisprogram fra Texas Instruments (TI), som brukes for å programmere diverse mikrokontrollere fra TI. I CCS brukes prosjektstrukturer som inneholder mange forskjellige typer filer. Såkalte *header files* (.h) må inkluderes i prosjektet for å kunne bruke definisjoner som er definert på forhånd. *Linker Command Files* (.cmd) brukes for minnetildeling for det spesifikke utviklingskortet som skal brukes. Når det skal brukes noe av det tilgjengelige periferiutstyret, må det i tillegg legges ved en ekstra .cmd-fil som brukes til

minnetildelig for de forskjellige enhetene. Videre vil selve kildekoden være i en eller flere .c-filer, som tar i bruk .h-, eller .asm-filer og i tillegg andre .c-filer. I dette prosjektet har det i tillegg blitt brukt filtyper med assemblerkode (.asm). Dette brukes for å utnytte spesialiserte instruksjoner for å optimalisere koden.

TI leverer et bibliotek med ulike ferdiglagde funksjoner og eksempler. Til noen programdeler er det brukt noen slike eksempler med litt endring og tilpasning, som eksempelvis ringbuffer (ikke TI), FIR-filtrering og initialisering av utviklingskortet. I starten av disse filene vil det være oppgitt opphavsrett og bruksrettigheter. Kommentarene i hovedfilene vil være på norsk, men kodesegmenter som er tatt fra andre eksempler vil ha uendrede kommentarer på engelsk. De engelske kommentarene forklarer koden på et mer detaljert og spesifikt nivå, mens de norske kommentarene gir en tilstrekkelig forståelse for programmets virkemåte. All kildekode er vedlagt dokumentet og liste over filer er å finne i vedlegg E.



Figur 3.13: Flytdiagram. Basert på FIR32-løsning.

3.8.1 Klokkefrekvenser

Etter all initialisering er blitt gjort, vil koden virke på en syklisk måte. Det er tatt i bruk avbruddsrutiner som vil skje med et intervall på 10 μs da dette er perioden til en samplingsfrekvens på 100 kHz. All beregning vil skje i avbruddsrutinen for å holde kontroll på når nytt innsignal skal beregnes. Når rutinen er ferdig, vil programmet gå tilbake til main-funksjonen til en evig løkke hvor ingenting skjer frem til neste avbrudd. På denne måten vil utgangsverdien fra prefilteret beregnes og pulsbredde settes innenfor én periode. Se figur 3.13 for flytdiagram.

Koden ble ikke tilstrekkelig optimalisert for å bruke en samplingsfrekvens på 176,4 kHz. Etter all optimalisering ble den gjennomsnittlige beregningstiden målt til ca. 9,107 μs. På grunn av dette kan programmet kun håndtere signaler med samplingsfrekvens på høyst 109,81 kHz omtrentlig. Denne testen ble gjort med systemklokkefrekvens på 200 MHz og PWM-klokkefrekvens på 100 MHz. Tidtaking ble gjort ved å sette en utgang til høy verdi i starten av beregningene, og sette den lav ved slutten av beregningene. Dette vil generere et PWM-signal, her med periode 10 μs, og dens pulsbredde er brukt til å finne beregningstiden. Det er valgt å bruke en samplingsfrekvens 100 kHz, slik at programmet har nok tid til å fullføre alle beregninger per sample. I tillegg er viktig at PWM-klokkefrekvensen er en delelig med samplingsfrekvensen da PWM-signalperioden settes som antall heltallige klokkeperioder og skal korrespondere med samplingsperioden. Sammensetningen av frekvenser nevnt tidligere er en mulig kombinasjon. PWM-klokkeperiode = 10 ns, samplingsperiode = 10 us PWM-signalperiode = 10/0,01 = 1000 PWM-klokkeperioder.

I DSC-en blir alle klokkene basert på en systemklokke. Denne systemklokken bestemmes ut fra en krystalloscillator. En krystalloscillator brukes for å generere elektriske signaler med presis frekvens. Den består av en krystall av piezoelektrisk materiale som har en mekanisk resonans. På grunn av krystalloscillatorens resonansfrekvens er det vanskelig å velge en relativ høy systemklokkefrekvens som i tillegg er delelig med 44100. På kortet er det to tilgjengelige krystaller som kan brukes for å sette systemklokkefrekvensen: 10 MHz og 15 MHz. Eksempelvis kan 110,25 MHz velges som systemklokkefrekvens og da må PWM-klokkefrekvens bli satt til halvparten av dette til 55,125 MHz, som da er delelig med 44100. Her er da systemklokkefrekvens satt til nesten halvparten av maksimalverdi, da vil beregningstid øke og ytterligere optimalisering av koden må til for å kunne sanntidsfiltrere signaler.

Det er tilgjengelig mange ulike krystaller med spesifikke resonansfrekvenser og det er mulig å fysisk bytte ut krystaller som er montert på kortet. På denne måten kan det velges å skifte til en krystall med passende frekvens for å oppnå en høy systemklokkefrekvens som er delelig med 44100.

Innstilling av systemklokkefrekvens $f_{\text{PLLSYSCLK}}$ gjøres på følgende måte [18, s. 114]:

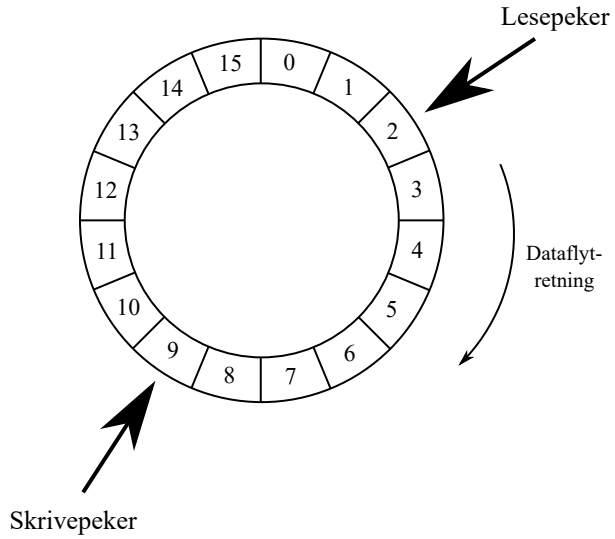
$$f_{\text{PLLSYSCLK}} = f_{\text{OSCCLK}} \cdot \frac{\text{IMULT + FMULT}}{\text{PLLSYSCLKDIV}} \tag{3.21}$$

hvor f_{OSCCLK} er krystalloscillatorens frekvens (10 eller 15 MHz), $IMULT$ er heltallsmultiplikatoren $(0-127)$, $FMULT$ er brøkdelsmultiplikatoren (0, 0,25, 0,5, 0,75) og PLLSYSCLKDIV er divideringsfaktoren (1, 2, 4, 6, ... 128).

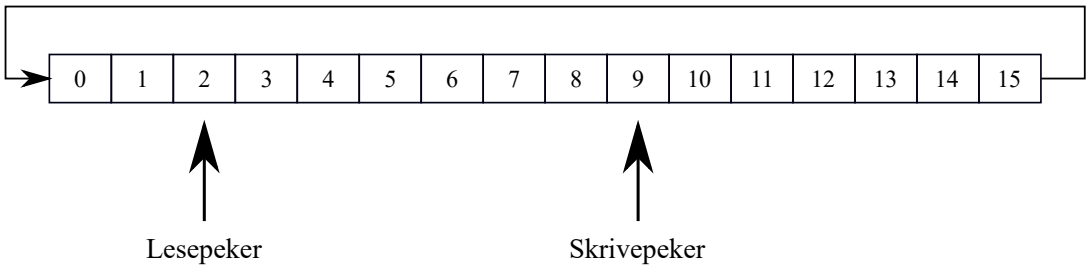
PWM-klokkefrekvensen kan bli satt til å være enten lik eller halvparten av systemklokkefrekvensen, med en maksimalverdi på 100 MHz.

3.8.2 Ringbuffer

Ved diskret signalbehandling vil det komme inn en ny verdi for hver programiterasjon. Siden filtrene ikke er kausale, blir signalet forsinket med 100 sampler for å oppnå kausal filtrering. For å kunne beregne $\hat{x}$, trengs innsignalet som var for 150 sampler siden. Det lengste bufferet trenger da å være mer enn 150 verdier langt. Bufferene må være en viss antall bits langt, altså 128 eller 256. Istedenfor å forskyve hver verdi i et lineært buffer ved nytt innsignal, plasseres den nye verdien inn i et ringbuffer og overskriver den eldste verdien i dette bufferet. På denne måten spares mye tid på grunn av færre instruksjoner gjort av prosessoren. Et ringbuffer er egentlig et navn på en annen virkemåte til et lineært buffer, da minne ikke blir fysisk konstruert som en ring. For å lage et ringbuffer settes det av en del av minne med ringbufferets lengde. For håndtering av bufferet, trengs det pekere som indikerer lese- og skrive-posisjon i minne. Når en peker har nådd enden av minnetildelingen, vil pekeren starte på andre siden og fungere som et endeløst buffer, se figur 3.14 og 3.15. Fra leseposisjon, også kalt hale på engelsk, blir verdien hentet, *«get at tail»*. Skriveposisjonen blir også kalt hode på engelsk, og her blir ny verdi satt, *«put at head»*. I programkoden brukes nøkkelordene *«put»* og *«get»*, da deler av koden er hentet fra engelske eksempler.



Figur 3.14: Ringbuffer struktur.



Figur 3.15: Lineær implementasjon av ringbufferet.

3.8.3 Datatyper

3.8.3.1 Flyttall

I datamaskiner er flyttall en måte å representere reelle tall på. Navnet kommer av at kommaet i tallet kan endre plass. Flyttall består av en desimalbrøk eller mantisse (m) og en eksponent (e), hvor $1 \leq m < 2$. Eksponenten varierer med antall bits og om det brukes toer-komplement benevning (da har eksponent også fortegnsbit). Hvis det brukes slik benevning blir flyttall regnet ut på følgende måte:

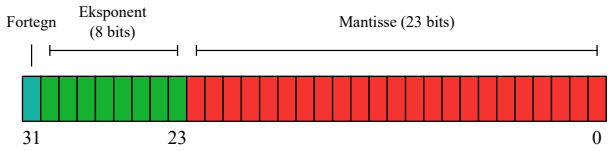
$$Flyttall = m \cdot 2^e$$

Fordelen med dette er en bred rekkevidde for mulige tallrepresentasjoner, men fordelingen i er ikke uniform og presisjonen varierer. En ulempe er at flyttall ikke alltid er i stand til å representere ønsket tall helt nøyaktig. To flyttalls representasjoner av 0,95 er vist i tabellen under.

Representasjonstype	**Faktisk tall**
Reellt tall	0,95
32-bits	0,9499999880790710449218750
64-bits	0.94999999999999995559107901499373838305473327636718750

Programmet bruker for det meste flyttall, men i den ene implementasjonen må det konverteres til Q-format for å bruke den ferdiglagde filtreringsfunksjonen.

Et 32-bits enkelpresisjons flyttall (IEEE 754-standard) bruker 4 byte med minne og inndeling av bits er som følger: 1 bit til fortegn, 8 bits til eksponent, og 23 bits til mantisse.



Figur 3.16: Inndeling av bit i et 32-bits flyttall.

Antall normaliserte ($1 \leq m < 2$) nummer mellom 0 og 1 (ikke talt med 0 og 1) i et 32-bits flyttall er $126 \cdot 2^{23}$. Antall unormaliserte nummer mellom 0 og 1 er $2^{23} - 1$. Hvis unormaliserte nummer telles med, vil det totale antall unike nummer et 32-bits flyttall kan representere mellom 0 og 1 være:

$$127 \cdot 2^{23} - 1 = 1065353215$$

Et 64-bits dobbelpresisjons flyttall (IEEE 754-standard) bruker 8 byte med minne og inndeling av bits er som følger: 1 bit til fortegn, 11 bits til eksponent, og 52 bits til mantisse.

Figur 3.17: Inndeling av bit i et 64-bits flyttall.

3.8.3.2 IQ-format

IQ-format (eller Q-format) har fast kommaplass i motsetning til flyttall og kalles *fixed-point number* på engelsk grunnet dette. Ved bruken av dette formatet velges antall bits som skal brukes før og etter komma, vanligvis er det ofte en fast fortegnsbit som også virker som heltallsdel.

«I» - Heltallsdel (*Integer*)
«Q» - Brøkdel (*Quotient*)

Fordelen med dette formatet er at det er samme presisjon gjennom hele rekkevidden. Ulempen er et mye mindre rekkevidde. Eksempelvis bruker Q31 (I1Q31 eller Q1.31) 1 bit til fortegn og heltallsdelen, og 31 bits til brøkdelen. Dette medfører en rekkevidde $-1 \leq$ Q-tall < 1, hvor all presisjon er innenfor intervallet. Et annet oppsett for formatet er å bruke flere bits til heltallsdelen, for eksempel Q4.28 som da bruker 4 bits til heltallsdelen (1 til fortegn) og 28 bits til brøkdelen. Her vil da rekkevidden bli økt til $-8 \leq$ Q-tall < 8.

Programmet som er implementert bruker kun verdier mellom -1 og 1 for utregning av prefiltrert signal, og ved å bruke et tredjeordens NFC-filter ville den største koeffisienttallverdien ha vært omtrent 5,26, slik at hele programmet kunne brukt I4Q28 til all beregning. Nå som programmet bruker sjetteordens NFC-filter er den største koeffisienttallverdien omtrent 16,65 og da trengs det I6Q26-format som reduserer oppløsningen ytterligere.

3.8.4 Hovedsyklus

3.8.4.1 «FIR32» implementasjon

For å optimalisere beregningene er det forsøkt å bruke en ferdiglaget assembly-fil fra Texas Instruments, ulempene med denne metoden er at funksjonen krever inndata på Q31-format. I tillegg er utdata på Q30-format, som vil si at 1-bits oppløsning blir mistet og unøyaktigheten øker. Til og med konvertering til og fra Q-formatet gir avrundingsfeil. Selv om denne implementasjonen er den raskeste, er resulterende beregninger mindre tilfredsstillende. Her er programmet laget med tanke på sanntidsberegning av signalene, slik programmet er ment til å virke.

Det er tatt i bruk et ekstern buffer og et internt buffer (Q-format) som er i struct-datatypen (holder kontroll på variabler assosiert med assembly-filern). Det interne bufferet blir kun brukt i selve filtreringen, derfor er det laget et eksternt buffer (flyttall) som skal holde kontroll på verdier som trengs ved et senere tidspunkt.

Beregning av $\hat{x}(n)$ gjøres på følgende måte:

1. Les inn ny sampleverdi og legg i x_1-buffer.
2. Regn ut $x_1(n)^3$ og konverter til Q31-format.
3. Legg inn i FIR32-funksjon med h_3-filter.
4. Kalkuler $x_3(n-L)$ og konverter tilbake fra Q30-format.
5. Legg inn i x_3-buffer.
6. Regn ut $x_1(n-L)^2 \cdot x_3(n-L)$ og konverter til Q31-format.

7. Legg inn i separat FIR32-funksjon med h_3-filter.
8. Kalkuler del av $x_5(n-2L)$ og multipliser med 3.
9. Konverter tilbake fra Q30-format.
10. Regn ut $x_1(n-L)^5$ og konverter til Q31-format.
11. Legg inn i FIR32-funksjon med h_5-filter.
12. Kalkuler andre del av $x_5(n-2L)$ og konverter tilbake fra Q30-format.
13. Legg sammen begge deler av $x_5(n-2L)$.
14. Hent nødvendige verdier fra x_1-buffer og x_3-buffer.
15. Regn ut $\hat{x}(n-2L) = x_1(n-2L) + x_3(n-2L) + x_5(n-2L)$.

Ved bruk av FIR32-funksjonen er det i tillegg bruk for konvertering av inn- og utdata fra funksjonen: Til Q31-format som inndata og fra Q30-format som utdata. Dette gjøres direkte i inn- og utlesing av data, sammen med eventuelle potensberegninger av innsignalet.

3.8.4.2 Direkte implementasjon

Denne type implementasjon kalles direkte i den forstand at flyttallene blir brukt til å gjøre beregninger med, imotsetning til å bli konvertert til Q31-formatet for deretter å bli beregnet i ekstern assembler-fil. Det ble tatt i bruk for-løkker med flyttall-multiplikasjoner og ringbuffere for å få riktig tall til riktig tid. Resultatene viser at denne implementasjonen virker bra, men beregningstid er relativt høy.

Beregning av $\hat{x}(n)$ gjøres på følgende måte:

1. Les inn ny sampleverdi.
2. Regn ut $x_1(n)^2$, $x_1(n)^3$ og $x_1(n)^5$.
3. Legg inn i buffere: x_1^2, x_1^3 og x_1^5.
4. Kalkuler $x_3(n-L)$ ved bruk av x_1^3-buffer og h_3-filtertabell.
5. Legg inn i x_3-buffer.
6. Kalkuler $x_5(n-2L)$ ved bruk av x_1^2-buffer sammen med h_3-filtertabell, og x_1^5-buffer sammen med h_5-filtertabell.
7. Hent nødvendige verdier fra x_1-buffer og x_3-buffer.
8. Regn ut $\hat{x}(n-2L) = x_1(n-2L) + x_3(n-2L) + x_5(n-2L)$.

3.8.4.3 Kvantisering og NFC

Videre beregning er lik for begge implementasjonene. Neste steg er å kvantisere $\hat{x}(n-2L) + y(n-2L)$ hvor $y(n-2L)$ er kompenseringsverdien for kvantiseringfeilen. NFC-verdien er regnet ut fra de seks tidligere kvantiseringsfeilene. Kvantisert verdi er $v(n-2L)$, og denne lagres for senere beregning av ny NFC-verdi. I dette programmet er det valgt å bruke et sjetteordens

NFC-filter, og da lagres de seks forrige kvantiseringsfeilene for å utføre FIR-filtreringen. Nylig beregnet NFC-verdi blir så lagt til i neste utregning av kvantisert verdi.

Beregning av NFC-verdien gjøres på følende måte:

1. Finn $v(n-2L)$ ved å kvantisere $\hat{x}(n-2L)+y(n-2L)$.
2. Finn kvantiseringsfeil $q(n-2L)$.
3. Legg i NFC-buffer.
4. Kalkuler ny $y(n-2L)$ ved å bruke NFC-buffer og NFC-filter.

Neste steg er å konvertere $v(n-2L)$ til en pulsbredde ved hjelp av PWM-mapping. Dette vil da gjøres på måten som er forklart i delkapittel 3.6.

3.8.4.4 MATLAB-generert signal

Det ble gitt tilgang til MATLAB-filer som simulerer hele prosessen og viser virkningsgraden til prefilteret. De utregnende verdiene blir brukt som kontrollverdier, da MATLAB bruker 64-bits flyttall som vil føre til bedre nøyaktighet.

4 | Målbare resultater

4.1 Tidsbruk

Programmet må korrespondere med samplingsfrekvensen til testsignalet, og gjøre alle beregninger i løpet av en signalperiode. Ved å måle tidsbruken til programmet er det mulig å finne ut høyst mulig samplingsfrekvens et signal kan ha for sanntidsfiltrering. Manglende optimalisering av koden førte til en maksimal samplingsfrekvens på omlag 100 kHz, hvor dette var FIR32-løsningen. Den noe tregere direkte løsningen kunne bare håndtere testsignaler med ca. 25 kHz samplingsfrekvens. På grunn av de periodiske egenskapene til testsignalet, vil også prefiltrert signal være perfekt periodisk. Ved å gjøre et passende antall beregninger vil det prefiltrerte signal også bli periodisk. 100 sampler av prefiltrert signal blir så lagret for å kunne generere utsignalet ved riktig samplingsfrekvens. På denne måten brukes de mest korrekte verdiene til generering av utsignal med riktig samplingsfrekvens. Hvis programmet bruker mer enn en PWM-periode på utregning vil pulsbredden beholde siste utregnede bredde, som vil føre til en lavere frekvens på utsignalet.

Måling av tidsbruk ble gjort ved å bruke en GPIO-utgang: Sett høy ved start av tidtaking og sett lav ved slutt av tidtaking. På denne måten genereres et PWM-signal med en pulsbredde som vil tilsi tidsbruken, dette kan så måles på et oscilloskop.

Program	**Tidsbruk (µs)**
FIR32 (RAM)	9,107
FIR32 (FLASH)	52,579
Direkte (RAM)	36,573
Direkte (FLASH)	360,535

Tabell 4.1: Tidsbruken til de to programmene, med bruk av FLASH- og RAM-minne. Tidene er funnet ved å ta gjennomsnittet av 10 prøvetakinger.

ROM-minne (Read Only Memory) er permanent og er kun mulig å lese fra, her blir ikke informasjon slettet ved strømavbrudd. RAM-minne (Random Access Memory) derimot er det mulig å skrive til og lese fra, men informasjonen blir slettet ved strømavrbudd. FLASH-minne blir ikke slettet ved strømavbrudd og det mulig å skrive til og lese fra. Ulempen er at FLASH-minne er tregere enn de to andre minnetypene.

Som resultatene viser er bruk av RAM-minne tydelig raskere, og det er mer enn nok RAM-minne på dette kortet til å utføre alle beregningene. Tidsbruken viser at FIR32-implementasjonen er omlag fire ganger raskere enn den direkte, men på bekostning av nøyaktighet i resultatet som blir forklart nærmere i neste delkapittel. Under testing (frekvensanalyse) av utsignalene er det

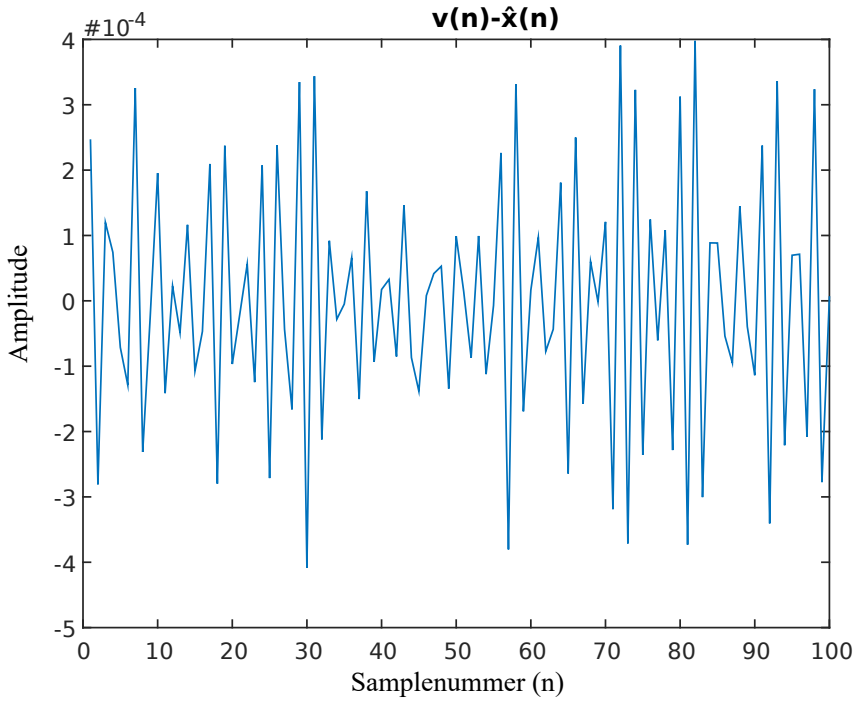
derfor utført beregning med direkte implementasjon i forkant av PWM-mapping.

4.2 Signalbehandling

Forskjellen mellom innsignal $u(n)$ og prefiltrert signal $\hat{x}(n)$ er så små at det bortimot ikke er synlig på et plott. Det er istedenfor laget en tabell i vedlegg C som viser 101 sampler av testsignalet før og etter prefiltrering, her ved begge implementasjonstypene (32-bit) og i tillegg testsignal som er prefiltrert i MATLAB (64-bit) som kontrollverdier. Siden testsignalet er periodisk med 100 sampler, er det tatt med en ekstra sample for å vise periodisiteten til både originalt signal og prefiltrert signal. Det blir også vist ulikheter mellom generert innsignal grunnet forskjellig bitoppøsning fra MATLAB til DSC-en.

4.2.1 NFC-beregning

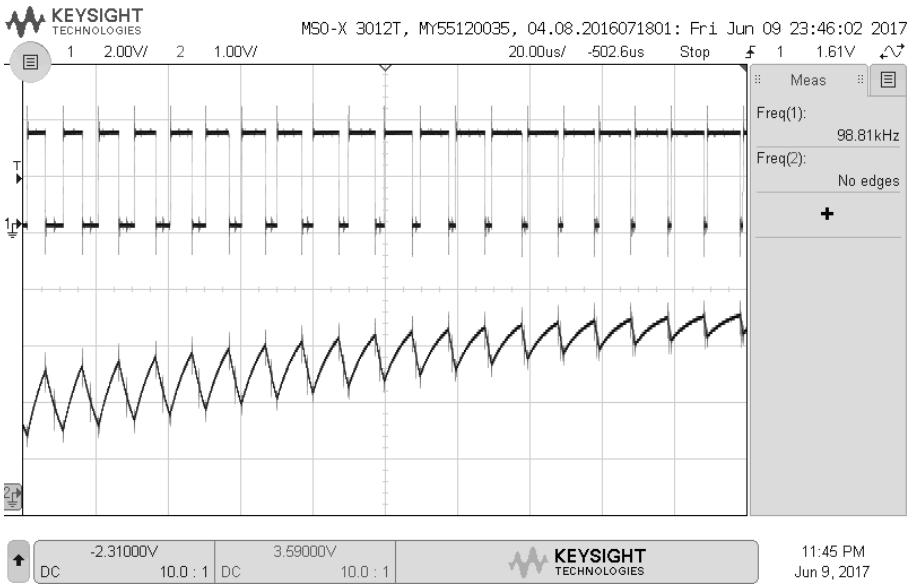
Som nevnt i delkapittel om NFC-teori (2.11), er ønsket virkning å få kvantisert signal $v(n)$ til å bytte på å ligge litt over og litt under prefiltrert signal $\hat{x}(n)$. Dette gjøres for å motvirke kvantiseringsstøy. Da det er små verdier vil det være vanskelig å se effekten i fullstendig plott av signalene. Istedenfor er da differansen i plasseringene plottet ($v(n) - \hat{x}(n)$). Figur 4.1 viser korrekt virkning av NFC-beregningene.



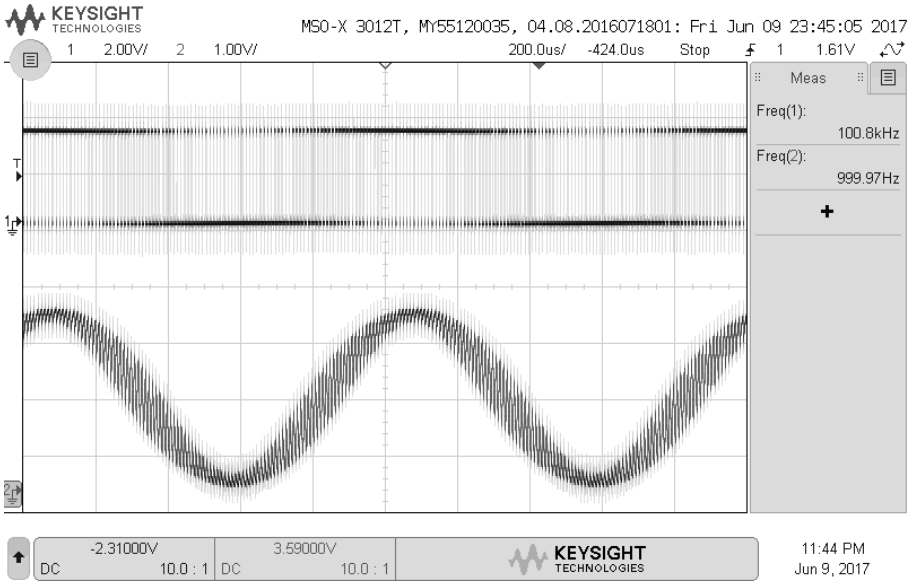
Figur 4.1: Plott av 100 sampler som viser virkningsgraden til NFC-beregningene.

4.2.2 Rekonstruert analogt signal

For å rekonstruere det analoge signalet blir det brukt et demoduleringsfilter som vist i delkapittel 3.7. I figur 4.2 vises synligheten til svitsjefrekvensen i det rekonstruerte analoge signalet. Siden svitsjefrekvensen er på 100 kHz og sinusbølge er på 1 kHz, er det derfor 100 «tagger» for hver sinusperiode. Som nevnt i delkapittel 3.7 er dette forventet oppførsel til demoduleringsfilteret. Figur 4.3 viser sinusbølgeformen til det rekonstruerte analoge signalet, selv om det her inneholder ytterligere høyfrekvente svingninger. Som tidligere nevnt er ikke disse høyfrekvente svingningene tatt i betraktning i DFT-utregningen på grunn av valget av båndbredde.



Figur 4.2: Nærbilde av sinusbølge og PWM-signal. Her er svitsjefrekvensen synlig i sinusbølgen.



Figur 4.3: PWM-signal og rekonstruert sinusbølge. Her med frekvensene 100 kHz og 1 kHz henholdsvis

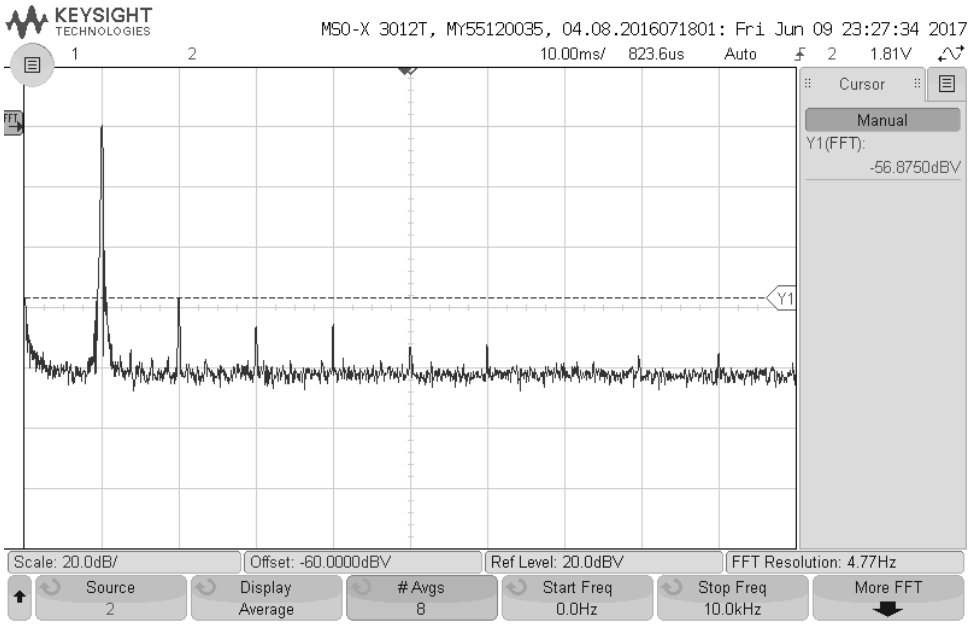
4.3 Frekvensanalyse

For å lage en frekvensanalyse er det rekonstruerte analoge signalet blitt målt av et oscilloskop og beregnet DFT ut fra, hvor ble det brukt båndbreddebegrensning (20 MHz) for å få et renere signal ved å filtrere vekk høyfrekvent støy. Det er blitt testet med sinussignaler med følgende frekvenser: 500 Hz, 1 kHz og 2 kHz. Signalene har blitt målt både med og uten prefiltrering, og i tillegg med og uten NFC-beregning. Da bare dette blir tolv unike grafer er mesteparten av grafene å finne i vedlegg D. Det er også gjort frekvensanalyse av prefiltrert signal fra MATLAB simuleringen.

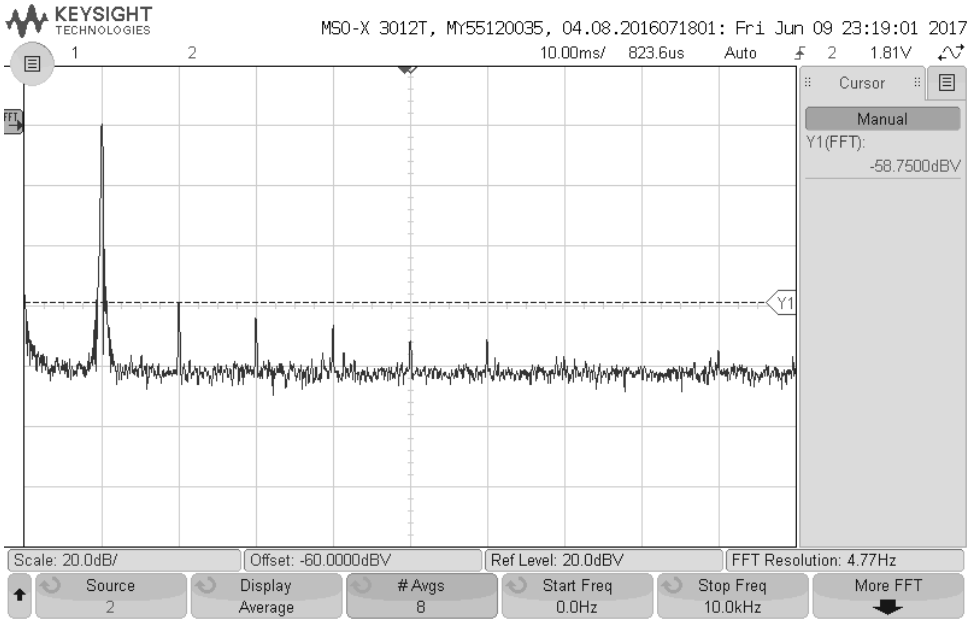
Alle grafer som viser frekvensanalysen har helt likt oppsett, med samme FFT-oppløsning på 4,77 Hz og båndbredde fra 0 Hz til 10 kHz. Se vedlegg B.4 for forklaring av FFT og DFT. En rute tilsvarer 20 dB i vertikal retning og 1 kHz i horisontal retning. Det er i tillegg plassert en markør på den første overharmoniske komponenten for enklere lesing. Grunnfrekvensen ligger på 0 dB. Under plottingen er det brukt gjennomsnittsverdien av 8 utregninger, dette er for å få et renere plott.

Siden støygulvet ligger på omtrent -80 dB burde ingen overharmoniske komponenter i prefiltrert signal være synlig på grafen, da de i teorien skal bli dempet til under -100 dB. Støygulvet er den omtrentlige horisontale linjen som fremkommer av måling av all uønsket støy. Termisk støy ligger i alle frekvensområder, og derfor gir den innslag over hele båndbredden. Et nivå på -80 dB er bra for oppsettet som ble gjort på dette laboratoriet.

Fra grafene kommer det frem at differansen mellom de overharmoniske komponentene i originalt og prefiltrert signal er meget liten. Den første overharmoniske komponenten virker å være litt mindre ved originalt signal, som vil si at prefilteret her virker mot sin hensikt i liten grad. Dette gjelder for testsignalene på 1 kHz og 2 kHz. For testsignalet på 500 Hz er den første overharmoniske komponenten litt mindre ved prefiltrert signal, og filteret virker i liten grad. Da dette er relativt små verdier, er det trolig grunnet unøyaktigheter og tilfeldigheter. Det er vanskelig å fastslå om filteret har noe virkning, positiv eller negativ, med disse resultatene.



Figur 4.4: Prefiltrert testsignal på 1 kHz med NFC.



Figur 4.5: Testsignal på 1 kHz uten prefiltrering med NFC.

5 | Konklusjon

Oppgaven har gått ut på å implementere et Volterra prefilter i signalbehandlingsdelen i en heldigital forsterker. Det har i denne oppgaven vært fokusert på selve signalkonverteringen og det har derfor ikke vært behov for signalforsterkning, da signalene kan måles direkte. Konstruksjon av signalforsterkerdelen har ikke vært en del av denne oppgaven. Fordi testsignalene kan generes direkte med ønsket samplingsfrekvens, har det ikke vært fokusert på oppsampling av signaler.

Fra resultatdelen er det vist at prefiltreringen gir gode tallverdier, med akseptabelt avvik grunnet lavere bitoppløsning. I tillegg er det vist at NFC-beregningene virker som ønsket. Frekvensanalysen av det rekonstruerte signalet gir ikke tilfredsstillende resultater. Da generering av pulsbredder må være perfekt symmetriske for å få riktig effekt av prefilteret, er det mulig at DSC-ens egenskaper medvirker til unøyaktige flankeplasseringer (usymmetriske). Dette gir da ulineær forvrengning og alle beregningene kan ha vært gjort forgjeves.

5.1 Videre arbeid

Siden det endelig målet med prosjektet (som denne oppgaven er en del av) er å konstruere en digital forsterker som kan sanntidsfiltrere musikk, er denne oppgaven et godt grunnlag for videre arbeid.

Ved å ta i bruk Q-format fremfor flyttall vil all presisjon i beregningen være innenfor ønsket område. På denne måten kan nøyaktigheten av beregningene økes. Videre kan det brukes assembler-kode for å ta i bruk spesialiserte instruksjoner for ytterligere optimalisering med tanke på beregningstid. I dette arbeidet ble kun en av to mulige CPU-er på DSC-en brukt. For å redusere beregningstid ytterligere er det en mulighet å benytte begge CPU-ene.

Da samplingsfrekvens må korrespondere med systemklokkefrekvensen, kan det være hensiktsmessig å endre oscillatorkrystall for å få så høy som mulig passende systemklokkefrekvens som er delelig med 44100.

Bibliografi

[1] F. Chierchie og S. O. Aase, *Volterra models for digital PWM and their inverses.* IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 62(10), s. 2606-2616, Oktober 2015.

[2] S. O. Aase, *Digital removal of pulse-width-modulation-induced distortion in class-D audio amplifiers.* IET Signal Processing, vol. 8(6), s. 680-692, August 2014.

[3] S. O. Aase, *A prefilter equalizer for pulse width modulation.* Signal Processing, vol. 92(10), s. 2444-2453, Oktober 2012.

[4] R. L. Boylesand og L. Nashelsky, «Power Amplifiers» i *Electronic Devices and Circuit Theory.* 11. utg. Upper Saddle River, NJ, USA: Pearson Education, 2013. s. 683-721.

[5] M. H. Rashid, *Microelectric Circuits: Analysis and Design.* 2. utg. Stamford, CT, USA: Cengage Learning, 2011.

[6] J. W. Nilsson og S. A. Riedel, *Electric Circuits.* 9. utg. Upper Saddle River, NJ, USA: Prentice Hall, 2011.

[7] T. L. Floyd, *Digital Fundementals.* 10. utg. Upper Saddle River, NJ, USA: Pearson/Prentice Hall, 2009.

[8] D. Shmilovitz, *On the Definition of Total Harmonic Distortion and Its Effect on Measurement Interpretation.* IEEE Transactions on Power Delivery, vol. 20(1), s. 526-528, Januar 2005.

[9] Y.-W. Fang, L.-C. Jiao, X.-D. Zhang og J. Pan, *On the convergence of Volterra filter equalizers using a pth-order inverse approach.* IEEE Transactions on Signal Processing, vol. 49(8), s. 1734-1744, August 2001.

[10] H. Bresch, M. Streitenberger, W. Mathis, *About the demodulation of PWM-signals with applications to audio amplifiers.* IEEE ISCAS 1998, Monterey, CA, USA

[11] A. V. Oppenheim, R. W. Schafer, og J. R. Buck, *Discrete-time signal processing.*, Prentice-Hall Signal Processing Series, 1999.

[12] G. L. Sicuranza, *Theory and approximation of polynomial filters.* IEEE Circuits & Systems Tutorials, ISCAS, vol. 94, s. 50-58, 1996.

[13] G. R. Slone, *The audiophile's project sourcebook.* New York, NY, USA: McGraw-Hill, 2002.

[14] J. Huang, K. Padmanabhan, og O. Collins, *The sampling theorem with constant amplitude variable width pulses.* IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 58, s. 1178-1190, Juni 2011.

[15] F. Chierchie og E. Paolini, *Real-time digital PWM with zero baseband distortion and low switching frequency.* IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 60, s. 2752-2762, Oktober 2013.

[16] F. Chierchie, E. Paolini, L. Stefanazzi, og A. Oliva, *Simple real-time digital PWM implementation for Class-D amplifiers with distortion-free baseband.* IEEE Trans. Ind. Electron., vol. 61, s. 5472-5479, Oktober 2014.

[17] Texas Instruments, *TMS320F2837xD Dual-Core Delfino Microcontrollers (datablad).* lit. nr.: SPRS880G, Mai 2016.

[18] Texas Instruments, *Technical Reference Manual (datablad).* TMS320F2837xD Dual-Core Delfino Microcontrollers, lit. nr.: SPRUHM8F, Desember 2016.

[19] M. Schetzen, *The Volterra and Wiener theories of nonlinear systems.* John Wiley & Sons, 1980.

[20] *Audio recording - Compact disc digital audio system.* IEC 60908:1999.

[21] Wikimedia Commons, (2006) *Block diagram of a PWM switching amplifier.* Tilgjengelig: https://commons.wikimedia.org/wiki/File:Pwm_amp.svg

[22] Wikimedia Commons, (2006) *Butterworth filter bode plot* Tilgjengelig: https://en.wikipedia.org/wiki/File:Butterworth_filter_bode_plot.svg

[23] Sonic Scoop, (2012) *dB power amplitude scale* Tilgjengelig: https://www.sonicscoop.com/wp-content/uploads/2012/11/db-power-amplitude_scale_crop.png

[24] Bright Hub Engineering, (2012) *Understanding Simple Transistor Circuits that use the Emitter as the Common Termination* Tilgjengelig: http://www.brighthubengineering.com/diy-electronics-devices/117387-understanding-simple-transistor-circuits-using-emitter-as-common-termination

[25] Hi-Fi klubben, (2016) *Høyttaleren - Slik fungerer den* Tilgjengelig: https://www.hifiklubben.no/hifimagasinet/eksperthjelp/2016/02/hoyttaleren--slik-fungerer-den/

A | Oversettinger og akronymer

- Total Harmonic Distortion (THD) - Total harmonisk forvrengning
- Total Harmonic Distortion plus Noise (THD+N) - Total harmonisk forvrengning pluss støy
- Pulse Width Modulation (PWM) - Pulsbreddemodulasjon
- Pulse Code Modulation (PCM) - Pulskodemodulasjon
- Crossover distortion - Overgangsforvrengning
- Dead-time distortion - Dødtidsforvrengning
- Root-Mean-Square (RMS) voltage - Effektverdi
- Bipolar Junction Transistor (BJT) - Bipolar transistor
- Field Effect Transistor (FET) - Felteffekttransistor
- Duty Cycle - Arbeidssyklus
- Fundamental Frequency - Grunnfrekvens
- Discrete Fourier Transform (DFT) - Diskret Fouriertransform
- Digital Signal Processor (DSP) - Digital signal prosessor
- Digital Signal Controller (DSC) - Digital signal kontroller
- Fast Fourier Transform (FFT)
- Texas Instruments (TI)
- Code Composer Studio (CCS)

B | Grunnleggende teori

B.1 Transistorer

En transistor er en elektrisk komponent som virker som en varierende resistor styrt av pådratt strøm eller spenning. Generelt vil en transistor ha tre kontaktpunkter, hvor den ene styrer ledeevnen mellom de to andre.

Transistorer finnes i flere forskjellige varianter, men blir delt inn i to hovedgrupper: bipolare eller felteffekt transistorer. Bipolare transistorer består av Base, Emitter og Kollektor, hvor pådratt strøm på Base styrer ledeevnen mellom Kollektor og Emitter. I en felteffekt transistor vil pådratt spenning på Gate, styre ledeevnen mellom Source og Drain.

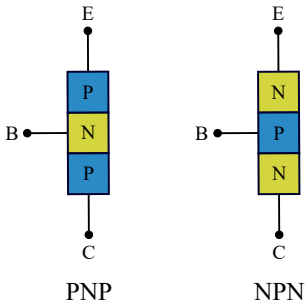
B.1.1 Bipolare transistorer

En bipolar transistor er bygd opp tre terminaler bestående av to typer halvledere, N-type og P-type. Det finnes to typer bipolare transistorer som kan lages, disse kalles enten PNP eller NPN. Navnet kommer fra hvordan halvlederne er plassert i transistoren. Bokstavene N og P indikerer hvordan en halvleder er blitt dopet. N-type halvledere blir dopet med donormateriale slik at den inneholder frie negative ladningsbærere, elektroner, som gir et overskudd av elektroner. P-type halvledere blir dopet med akseptormateriale slik at den inneholder positive ladningsbærere, hull, som gir et underskudd av elektroner. Dette gjøres for å muliggjøre elektronflyt (strømflyt) i halvledere, da halvledere har lav konduktans uten doping.

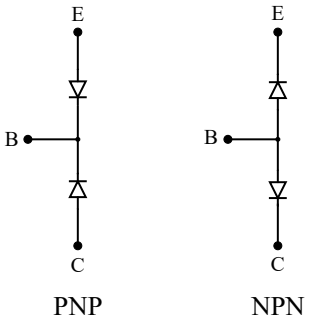
Doping gjøres ved å innføre urenheter i halvlederen. Det kan brukes silisium som halvleder med fosfor eller bor som urenheter for å lage en N- eller P-type halvleder henholdsvis. Begge dopingene øker konduktansen til halvlederen.

En transistor kan anses som to dioder som er som har felles katode eller anode, og blir navngitt deretter. PNP betyr at to dioder deler N-side (katode) og NPN betyr at to dioder deler P-side (anode). Dette er en ofte brukt sammenligning for beskrivelse av virkemåten til en transistor. Se figur B.1 for fysisk oppbygging av en transistor og figur B.2 for en to-dioders analogi.

En diode består av en en N-type halvleder som er i kontakt med en P-type halvleder. N-type avgir frie elektroner og kalles katode, og P-type tar opp frie elektroner og kalles anode. I overgangen mellom halvlederne vil det forekomme naturlig forflytting av frie elektroner fra N-siden til P-siden som gjør P-siden negativt ladet og N-siden positivt ladet. Dette forhindrer videre forflytting av frie elektroner, og kalles utarminsområde (*depletion region*). En påført spenning over dioden med plusspol koblet på N-siden vil føre til de frie elektonene tiltrekkes plusspolen og ingen strømflyt vil skje. Tilsvarende skjer på P-siden hvor hullene blir tiltrukket



Figur B.1: Fysisk oppbygging.

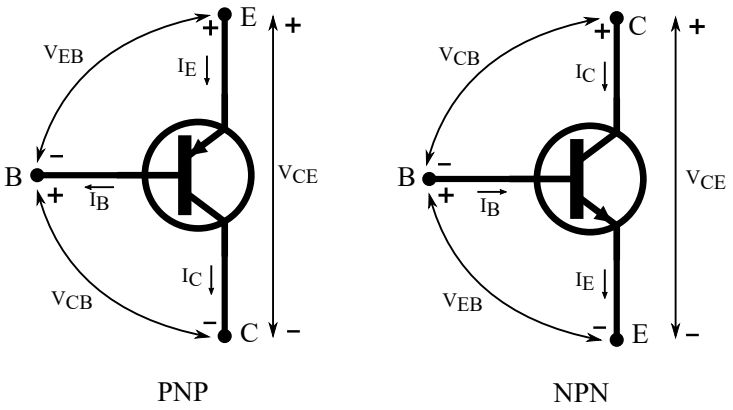


Figur B.2: To-dioders analogi.

minuspolen. Dette kalles sperreretningen på en diode. Ved en å reversere spenningen vil de frie elektronene bli tiltrukket plusspolen, men her finnes det hull å fylle. En strømflyt vil da forekomme hvis påført spenning er over den såkalte diodespenningen. Dette kalles lederetningen til en diode.

I grensen mellom n-type halvleder og en p-type halvleder vil det forekomme naturlig forflytting av elektroner fra n-type til p-type. Dette fører til en positiv ladning av p-typen og negativ ladning av n-typen. Dette er en elektrisk potensialforskjell som kalles diodespenning. I silisiums-dioder vil denne være på omtrent 0,6-0,7 V. Pådratt spenning over diode må overkomme diodespenningen før dioden kan funksjonere i lederetning.

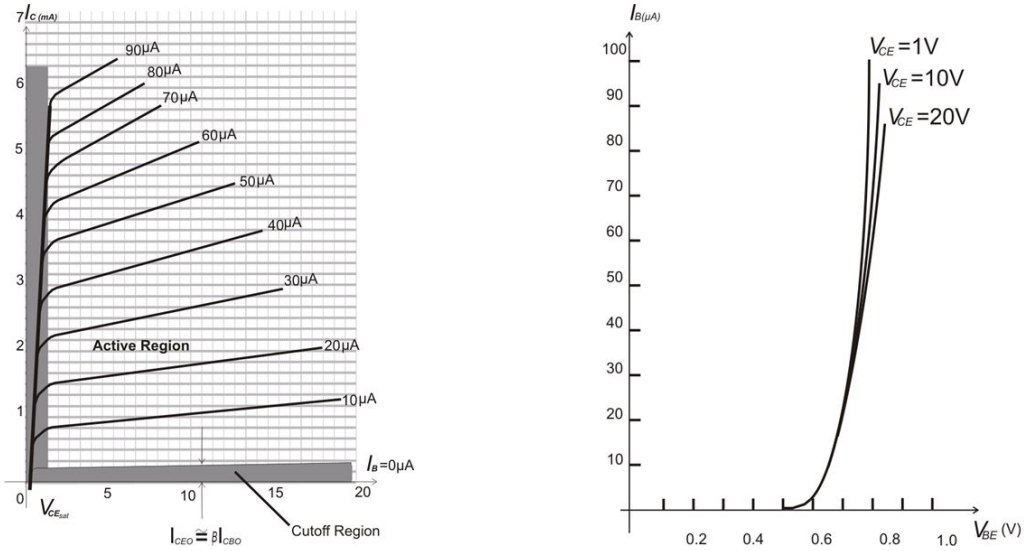
Bipolare transistorer har to forskjellige kretssymbol, en for hver type oppbygging (se figur B.3). I kretstegninger blir Base, Kollektor og Emitter bemerket med B, C og E henholdsvis.



Figur B.3: Kretssymbol for bipolare transistorer med påførte spenninger og strømmer.

En god måte å beskrive virkemåten til en transistor er ved bruk av grafer med oversikt over strøm-spenningskarakteristikk. Det finnes tre typer oppkobling av en transistor; felles-emitter, -kollektor eller -base. Ved en felles-emitter oppkobling vil basen virke som inngang og kollektor virke som utgang. Emitter vil derimot være felles for både inngang og utgang, enten koblet til jord eller spenningsforsyning. I figur B.4 vises karakteristikken til en felles-emitter transistor. En transistor har tre operasjonsområdet: aktivt område, metningsområde og *cutoff*-område. I det såkalt *cutoff*-område vil ikke transistoren lede i det hele tatt eller svært lite og ulinæert. I metningsområde vil økning av base-strømmen føre til lite eller ingen økning av kollektor-strøm. Det aktive område viser til det område hvor økning av base-strømmen fører til bortimot lineær økning av kollektor-strømmen. En transistor må da ha høy nok kollektor-emitter-spenning og base-emitter-spenning før å fungere skikkelig.

I tillegg finnes det et område som kalles *breakdown*-område. Det viser til en kollektor-emitter-spenning som er for høy, noe som resulterer i leding i sperreretning for den ene dioden og kollektor- og emitter-strømmen vil da øke ukontrollerbart. Dette område bør unngås da transistoren kan bli ødelagt.



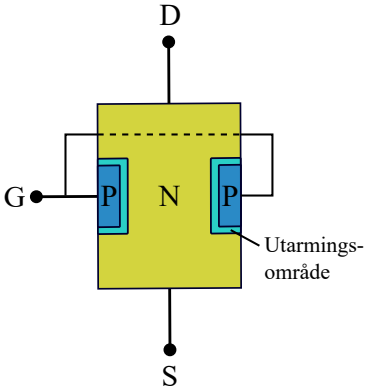
(a) Kollektor-karakteristikk.

(b) Base-karakteristikk.

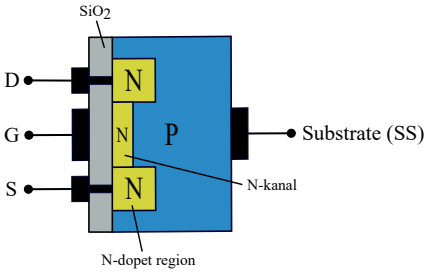
Figur B.4: Karakteristikken til en felles-emitter transistor [24].

B.1.2 Felteffekt transistor

En felteffekt transistor virker på lignende måte som en bipolar transistor, hovedforskjellen er at påtrykt spenning styrer strømflyten mellom to andre terminaler i motsetning til påtrykt strøm. Gate-terminalen styrer strømflyten mellom drain og source. Felteffekt transistorer består av to underkategorier: *Junction Field-Effect Transistor* (JFET) og *Metal-Oxide-Semiconductor Field-Effect Transistor* (MOSFET).



Figur B.5: *Junction field-effect transistor* (JFET).



Figur B.6: N-kanal utarmingstype MOSFET.

Ved å endre påtrykt spenning til en JFET endres utarmingsområde på grunn av egenskapene til P- og N-dopede regionene. Hvis utarmingsområdene blir så store at de krysser N-regionen, vil strømflyten stoppe.

For å endre strømflyten gjennom kanalen i en MOSFET endres utarmingsområdene her også. MOSFET-er blir laget som enten utarmings- eller forøkningstype, som betyr at transistoren leder strøm eller ikke uten påtrykt spenning på gate. Utarming vil lede strøm uten påtrykt spenning og ledeevnen blir redusert ved høyere spenning, mens forøkningstype trenger påtrykt spennning for å begynne å lede. Denne typen transistor har en ekstra terminal kalt substrate som vanligvis blir koblet til jord.

B.2 Effektverdi/RMS-verdi

RMS står for *Root Mean Square* og kalles på norsk kvadratisk gjennomsnitt. RMS kommer fra den engelske definisjonen: «The rms value of a periodic function is defined as the square **r**oot of the **m**ean value of the **s**quared function» [6, s. 331]. Kvadratisk gjennomsnitt blir regnet ut på følgende måte:

$$V_{rms} = \sqrt{\frac{1}{T}\int_{t_0}^{t_0+T} V_m^2 cos^2(\omega t + \phi)dt} \tag{B.1}$$

For rene sinussignaler blir utregning forenklet betraktelig og er alltid:

$$V_{rms} = \frac{V_m}{\sqrt{2}} \tag{B.2}$$

Effektverdi er en alternativ ordbeskrivelse på norsk for kvadratisk gjennomsnitt av en periodisk funksjon. En vekselspenning med sinusbølgeform med effektverdi på 240 V (amplitude 339 V) vil gi like lys fra glødelamper som en likespenning på 240 V. Effekten som begge disse verdiene fører til er derfor den samme.

B.3 Decibel

Bel er en (basis 10)-logaritmisk enhet som brukes for å uttrykke forholdet mellom to mengder. For å finne det logaritmiske forholdstallet mellom effektmengder P_1 og P_2 brukes følgende formel:

$$G = \log_{10}(\frac{P_2}{P_1}) \quad (B) \tag{B.3}$$

Fordi bel er ansett som upraktisk, i den forstand at den er en for stor enhet og gir små tall, brukes heller decibel som er en tiendedels bel (10 decibel = 1 bel).

$$G_{dB} = 10\log_{10}(\frac{P_2}{P_1}) \quad (dB) \tag{B.4}$$

I tillegg kan desibel brukes til lineære mål som spenninger og strømmer. Endringer i tilført effekt i en motstand er proporsjonal med strøm- eller spenningsendringer kvadrert. Derfor gjelder:

$$G_{dB} = 10\log_{10}(\frac{P_2}{P_1}) = 10\log_{10}\left(\frac{V_2^2/R_2}{V_1^2/R_1}\right) = 10\log_{10}\left(\frac{V_2}{V_1}\right)^2 \tag{B.5}$$

$$G_{dB} = 20\log_{10}(\frac{V_2}{V_1}) \quad (dB) \tag{B.6}$$

hvor forskjellene i impendansene R_1 og R_2 er sett bort i fra. Den sistnevnte definisjonen av decibel brukes i Bode-plot, siden den gjelder for strøm- og spenningsendringer. Begrepet dekade betyr et intervall som spenner over en faktor på 10, som eksempel 10 Hz og 100 Hz. Oktav derimot betyr et intervall som spenner over en faktor på 2, som eksempel 20 Hz og 40 Hz.

dB	power ratio	amplitude ratio
100	10 000 000 000	100 000
90	1 000 000 000	31 620
80	100 000 000	10 000
70	10 000 000	3 162
60	1 000 000	1 000
50	100 000	316.2
40	10 000	100
30	1 000	31.62
20	100	10
10	10	3.162
3	1.995	1.413
1	1.259	1.122
0	1	1
-10	0.1	0.316 2
-20	0.01	0.1
-30	0.001	0.031 62
-40	0.000 1	0.01
-50	0.000 01	0.003 162
-60	0.000 001	0.001
-70	0.000 000 1	0.000 316 2
-80	0.000 000 01	0.000 1
-90	0.000 000 001	0.000 031 62
-100	0.000 000 000 1	0.000 01

Figur B.7: Skala som viser omregningen fra forholdet mellom effektmengder (*power ratio*) og amplituder (*amplitude ratio*) til decibel [23].

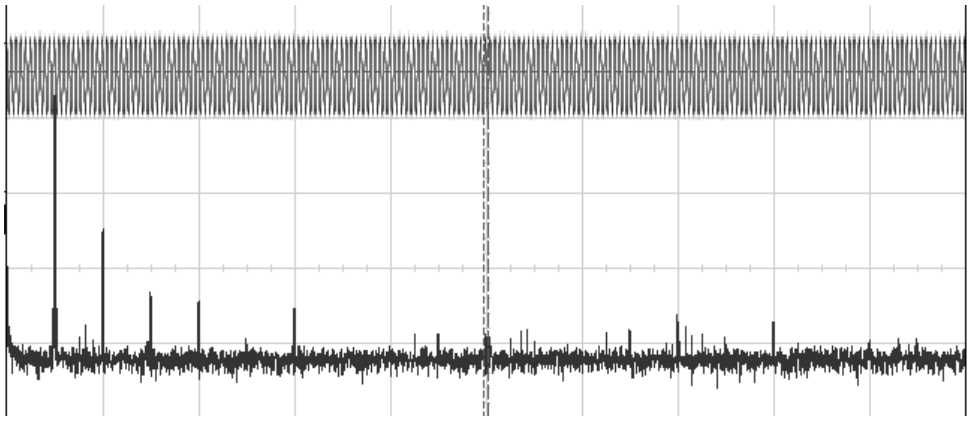
B.4 Diskret fouriertransform

DFT (Discrete Fourier Transform) er en matematisk prosess som analyserer en endelig signalsekvens i tidsdomene og beregner representasjonen av signalet i frekvensdomene. FFT (Fast Fourier Transform) er en numerisk algoritme som beregner DFT-en til et signal og ofte blir disse navnene brukt om hverandre grunnet den hyppige bruken av FFT-algoritmen. Prosessen går ut på å sammenligne signalet med en rekke forskjellige sinussignaler, for å determinere hvilke

frekvenskomponenter signalet består av. Beregningen av en enkelt «bin» gjøres på følgende måte:

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N} \\ &= \sum_{n=0}^{N-1} x_n \cdot [cos(2\pi kn/N) - i \cdot sin(2\pi kn/N)] \end{aligned} \tag{B.7}$$

hvor x_n er en sample av signalet med kompleks verdi og N er antall sampler. Vanligvis er $k \in [0, N-1]$ slik at antall «bins» er lik antall sampler i signalsekvensen, men antallet blir ofte halvert slik at høyeste frekvens observert ikke blir høyere enn Nyquist-frekvensen (for å unnslippe folding: Speiling av frekvenser). Plotting av X_k gjøres i absolutt verdi, da dette også er komplekse tall. Den resulterende grafen vil da vise pigger som indikerer tilstedeværende frekvenskomponenter. Grafens vertikale akse bruker benevningen dBV (decibel utregnet fra signalets amplitude). Den komponent med høyest amplitude vil nå 0 dB og være referanseverdi. Forskjellen mellom signalstyrkene vises i antall desibel.



Figur B.8: Plott av DFT-en til et 1 kHz sinussignal. Her vises også de harmoniske komponentene som kommer fra ulineær forvrengning.

C | Kalkulerte signaler

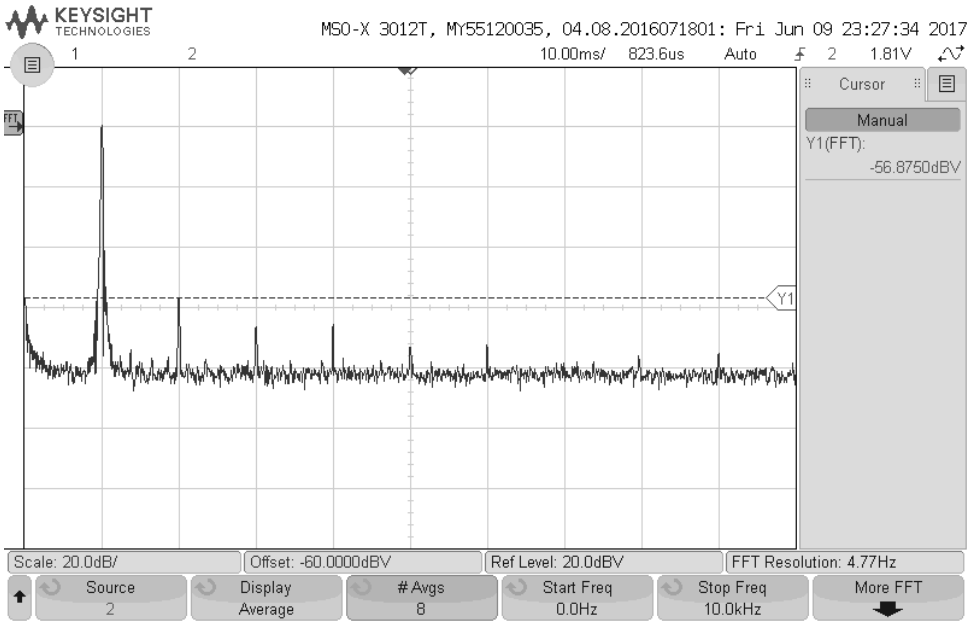
n	u(n) 64-bit	u(n) 32-bit	$\hat{x}$ 32-bit (Direkte)	$\hat{x}$ 32-bit (FIR32)	$\hat{x}$ 64-bit
0	0,95	0.949999988	0.950208724	0.949992418	0,950206253
1	0,949112028	0.949112058	0.949341476	0.94910872	0,949316745
2	0,946451616	0.946451604	0.946698308	0.946451783	0,94665176
3	0,942029263	0.942029238	0.942289412	0.942032516	0,9422219
4	0,935862423	0.935862422	0.936131895	0.9358688	0,936044787
5	0,927975432	0.927975416	0.928249955	0.927985132	0,928144985
6	0,918399419	0.918399453	0.918674469	0.918413222	0,918553904
7	0,907172174	0.907172203	0.907443404	0.907191098	0,907309665
8	0,894338006	0.894338012	0.894601047	0.894363642	0,894456944
9	0,879947566	0.879947543	0.880198479	0.879982114	0,880046785
10	0,864057647	0.86405766	0.86429286	0.864104211	0,864136391
11	0,846730959	0.846730947	0.846947134	0.846793592	0,846788889
12	0,828035882	0.828035831	0.828230381	0.828119695	0,828073072
13	0,808046198	0.808046222	0.808216989	0.808157682	0,808063117
14	0,786840795	0.786840796	0.786986291	0.786987722	0,786838286
15	0,764503364	0.76450336	0.764622569	0.764695048	0,764482605
16	0,741122058	0.741122067	0.741214812	0.741369426	0,741084526
17	0,716789153	0.716789126	0.716855526	0.717104495	0,716736573
18	0,691600681	0.69160068	0.69164145	0.691997766	0,691534972
19	0,665656049	0.66565609	0.665672481	0.666150093	0,665579272
20	0,639057647	0.639057636	0.639051259	0.63966471	0,638971948
21	0,611910449	0.611910522	0.611883342	0.612647474	0,611818
22	0,584321592	0.584321618	0.584275961	0.585205734	0,584224538
23	0,556399955	0.556400001	0.556338191	0.557448566	0,556300367
24	0,528255734	0.528255761	0.528180361	0.529485345	0,528155557
25	0,5	0.5	0.499913663	0.501426339	0,499901018
26	0,471744266	0.471744299	0.471649557	0.473381251	0,471648067
27	0,443600045	0.443599999	0.443499446	0.445459455	0,443507997
28	0,415678408	0.415678412	0.415574372	0.417769372	0,415591644
29	0,388089551	0.388089657	0.387984455	0.390417814	0,388008956
30	0,360942353	0.360942423	0.360838085	0.363509685	0,360868567
31	0,334343951	0.334344029	0.334242404	0.33714813	0,334277376
32	0,308399319	0.30839932	0.308301985	0.311433285	0,308340128
33	0,283210847	0.283210814	0.283119082	0.286462903	0,28315901
34	0,258877942	0.258877933	0.258792907	0.262331337	0,258833247

35	0,235496636	0.235496581	0.23541908	0.239129409	0,235458718
36	0,213159205	0.213159174	0.213089749	0.216944635	0,213127575
37	0,191953802	0.191953868	0.191892892	0.195860386	0,191927885
38	0,171964118	0.171964139	0.171911716	0.175955847	0,171943276
39	0,153269041	0.153269082	0.153225154	0.157306165	0,153252611
40	0,135942353	0.13594237	0.135906667	0.139981657	0,135929671
41	0,120052434	0.120052457	0.120024607	0.124048226	0,12004286
42	0,105661994	0.105662078	0.105641574	0.109566823	0,105654932
43	0,092827826	0.0928278565	0.0928141177	0.0965931863	0,09282274
44	0,081600581	0.0816006064	0.0815929845	0.0851780623	0,081597002
45	0,072024568	0.072024554	0.0720223859	0.0753667206	0,072022101
46	0,064137577	0.064137578	0.0641401708	0.0671989769	0,064135902
47	0,057970737	0.0579707623	0.0579774678	0.0607087463	0,057969599
48	0,053548384	0.0535483956	0.053558588	0.0559241399	0,053547586
49	0,050887972	0.0508879721	0.0509010777	0.0528672487	0,050887359
50	0,05	0.0500000119	0.0500155166	0.0515539683	0,049999446
51	0,050887972	0.0508879721	0.0509054251	0.05199375	0,050887359
52	0,053548384	0.0535483956	0.0535674021	0.0541896559	0,053547586
53	0,057970737	0.0579707623	0.0579909869	0.0581379756	0,057969599
54	0,064137577	0.064137578	0.0641587377	0.0638283715	0,064135902
55	0,072024568	0.072024554	0.0720463842	0.071243614	0,072022101
56	0,081600581	0.0816005766	0.0816228464	0.0803596601	0,081597002
57	0,092827826	0.0928277969	0.0928502604	0.0911454335	0,09282274
58	0,105661994	0.105661869	0.105684288	0.103562899	0,105654932
59	0,120052434	0.120052397	0.120074473	0.117567524	0,12004286
60	0,135942353	0.13594231	0.135963723	0.133107334	0,135929671
61	0,153269041	0.153268963	0.153289333	0.150124043	0,153252611
62	0,171964118	0.17196402	0.171982884	0.168552846	0,171943276
63	0,191953802	0.191953808	0.191970631	0.188322663	0,191927885
64	0,213159205	0.213159174	0.213173345	0.20935595	0,213127575
65	0,235496636	0.23549673	0.235507593	0.231569991	0,235458718
66	0,258877942	0.258878022	0.258884817	0.254876047	0,258833247
67	0,283210847	0.283210874	0.283212811	0.279180706	0,28315901
68	0,308399319	0.30839932	0.308395565	0.30438599	0,308340128
69	0,334343951	0.33434394	0.334333688	0.330389678	0,334277376
70	0,360942353	0.360942483	0.36092487	0.357085884	0,360868567
71	0,388089551	0.388089657	0.388063937	0.38436538	0,388008956
72	0,415678408	0.415678501	0.41564396	0.412116617	0,415591644
73	0,443600045	0.443599939	0.443555981	0.440225691	0,443507997
74	0,471744266	0.47174412	0.471690357	0.468577743	0,471648067
75	0,5	0.5	0.499936134	0.497056663	0,499901018
76	0,528255734	0.528255701	0.528181732	0.525545657	0,528155557
77	0,556399955	0.556399882	0.556316078	0.553928435	0,556300367
78	0,584321592	0.584321499	0.584228396	0.582089543	0,584224538
79	0,611910449	0.611910343	0.611808777	0.609914601	0,611818
80	0,639057647	0.639057696	0.638948858	0.637291312	0,638971948

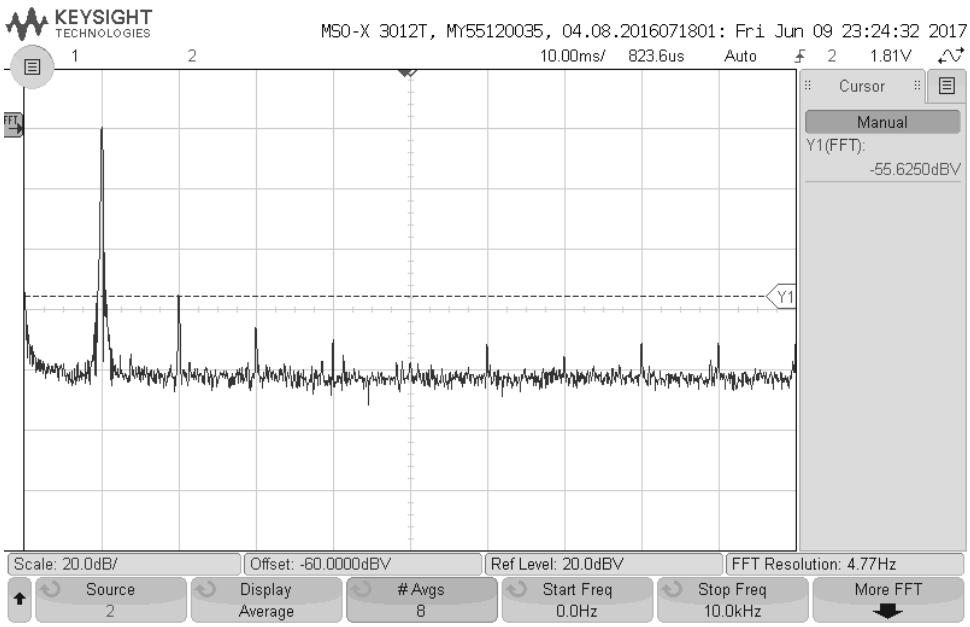
81	0,665656049	0.66565609	0.66554153	0.664108932	0,665579272
82	0,691600681	0.69160068	0.691482246	0.690260112	0,691534972
83	0,716789153	0.716789126	0.716668963	0.715640545	0,716736573
84	0,741122058	0.741121829	0.74100244	0.740148962	0,741084526
85	0,764503364	0.764503121	0.76438725	0.763688982	0,764482605
86	0,786840795	0.786840677	0.786731303	0.78616786	0,786838286
87	0,808046198	0.808046103	0.807946265	0.807497263	0,808063117
88	0,828035882	0.828035772	0.82794857	0.827594161	0,828073072
89	0,846730959	0.846730828	0.846659362	0.846380472	0,846788889
90	0,864057647	0.8640576	0.86400485	0.863783658	0,864136391
91	0,879947566	0.879947543	0.87991631	0.879736662	0,880046785
92	0,894338006	0.894337893	0.894330621	0.894178092	0,894456944
93	0,907172174	0.907172084	0.907190681	0.907052875	0,907309665
94	0,918399419	0.918399334	0.918445408	0.918311954	0,918553904
95	0,927975432	0.927975416	0.928049922	0.927912474	0,928144985
96	0,935862423	0.935862422	0.935965776	0.935817957	0,936044787
97	0,942029263	0.942029238	0.942161083	0.941998482	0,9422219
98	0,946451616	0.946451545	0.946610808	0.946430981	0,94665176
99	0,949112028	0.949112058	0.949297249	0.949098945	0,949316745
100	0,95	0.949999988	0.950208724	0.949992418	0,950206253

Tabell C.1: Testsignal før og etter prefiltrering, 32- og 64-bits beregning. Bemerk at verdiene ved n=0 og n=100 i et enkelt signal er identiske på grunn av periodisitet. Verdiene 0,05 og 0,95 er avrundet på grunn av antall tilgjengelig siffer og er ikke nøyaktig representert som de er.

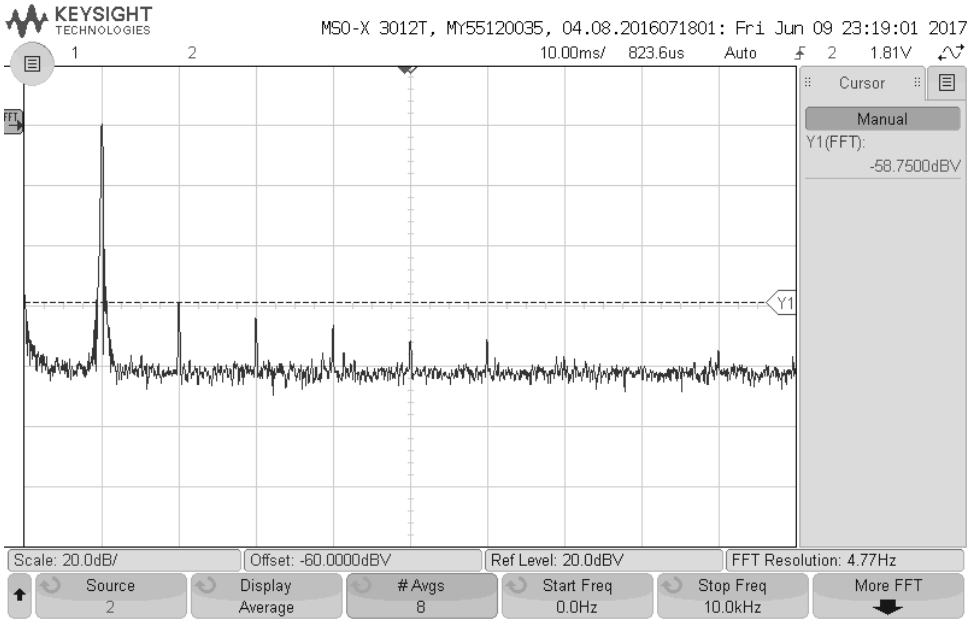
D | Bilder fra oscilloskop



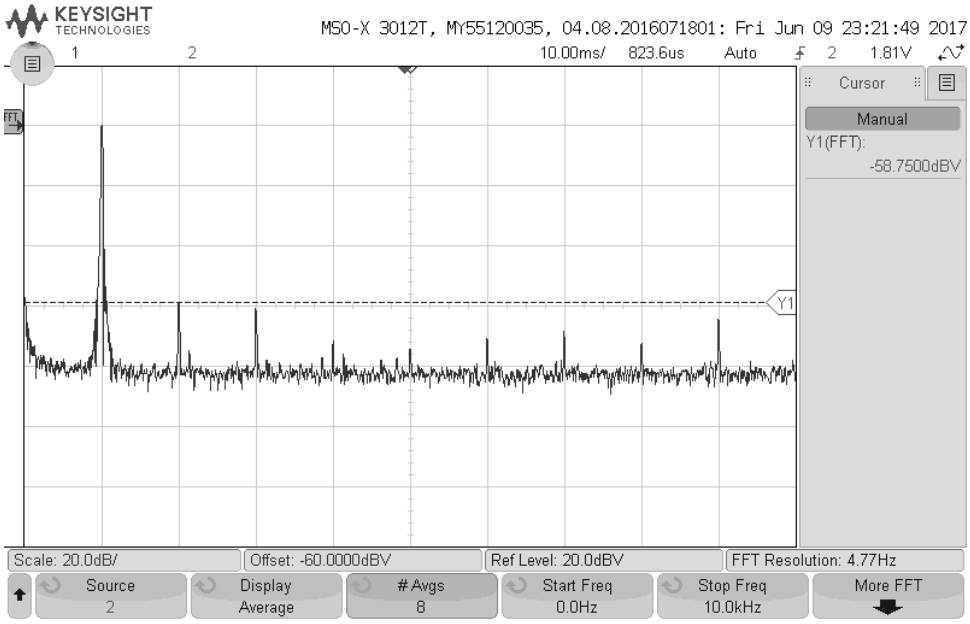
Figur D.1: Frekvensanalyse av prefiltrert testsignal på 1 kHz med NFC.



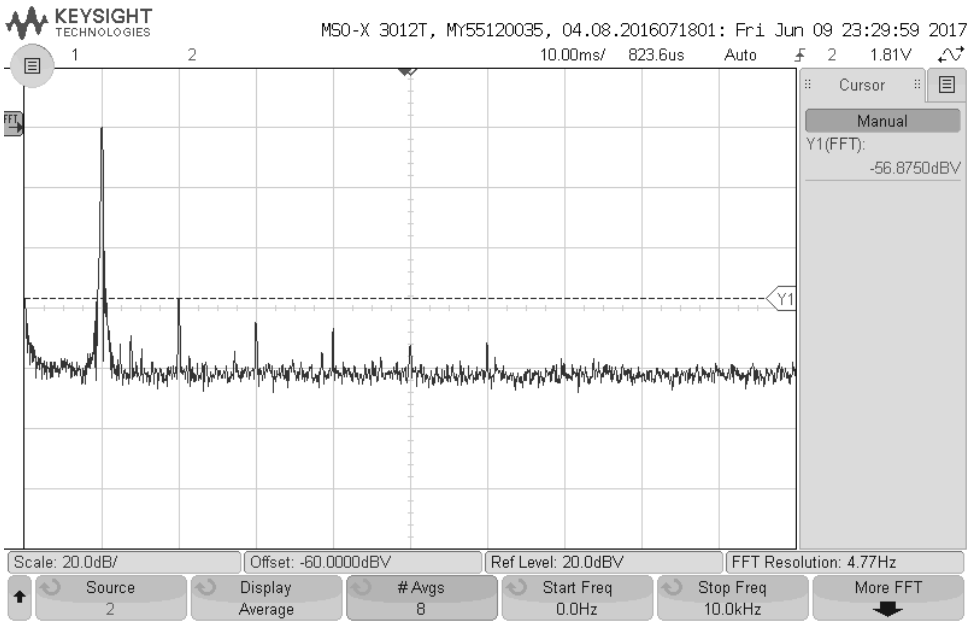
Figur D.2: Frekvensanalyse av prefiltrert testsignal på 1 kHz uten NFC.



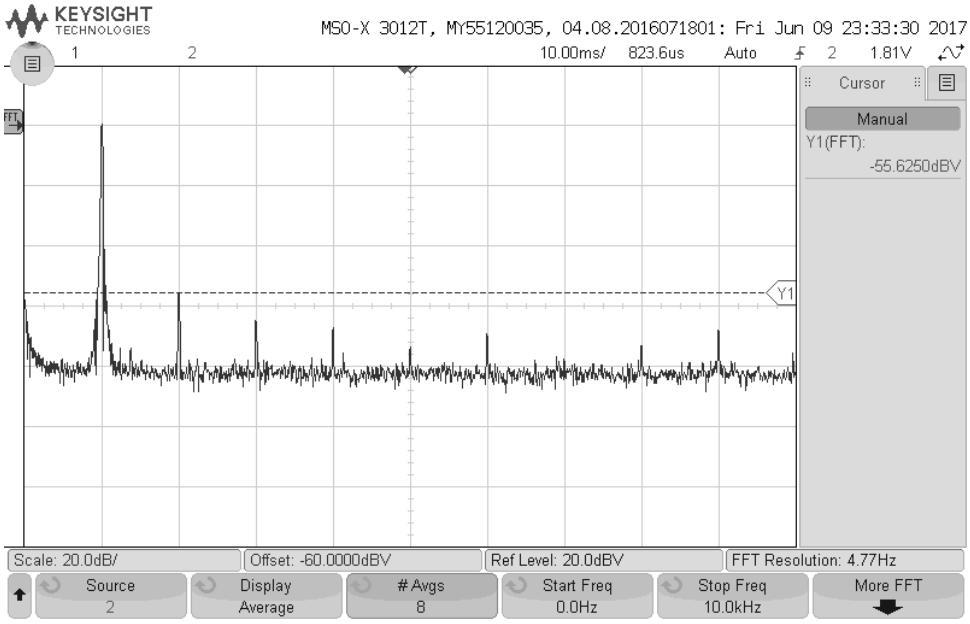
Figur D.3: Frekvensanalyse av testsignal på 1 kHz uten prefiltrering med NFC.



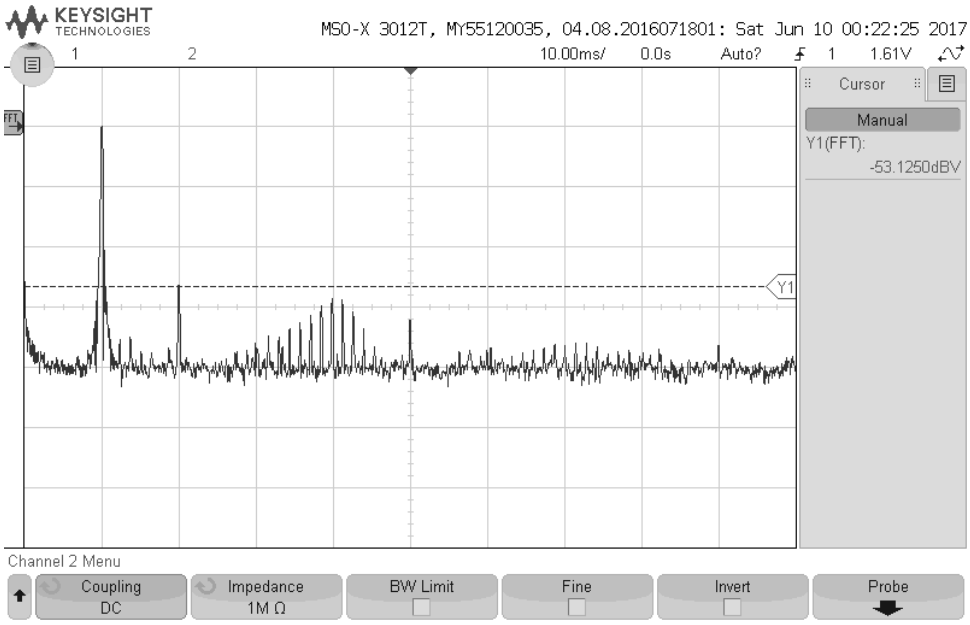
Figur D.4: Frekvensanalyse av testsignal på 1 kHz uten prefiltrering uten NFC.



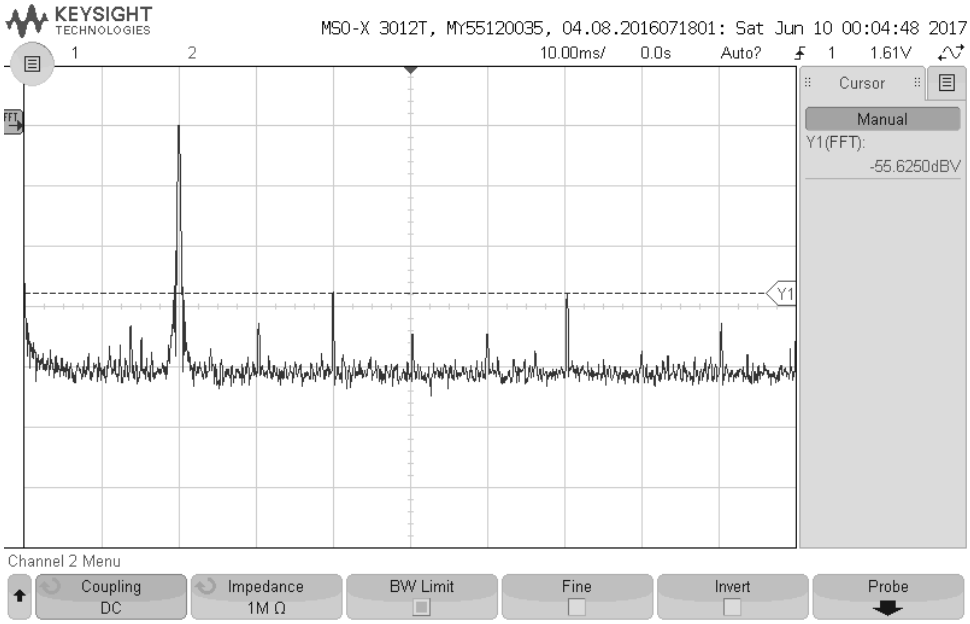
Figur D.5: Frekvensanalyse av prefiltrert testsignal (MATLAB-generert) på 1 kHz med NFC.



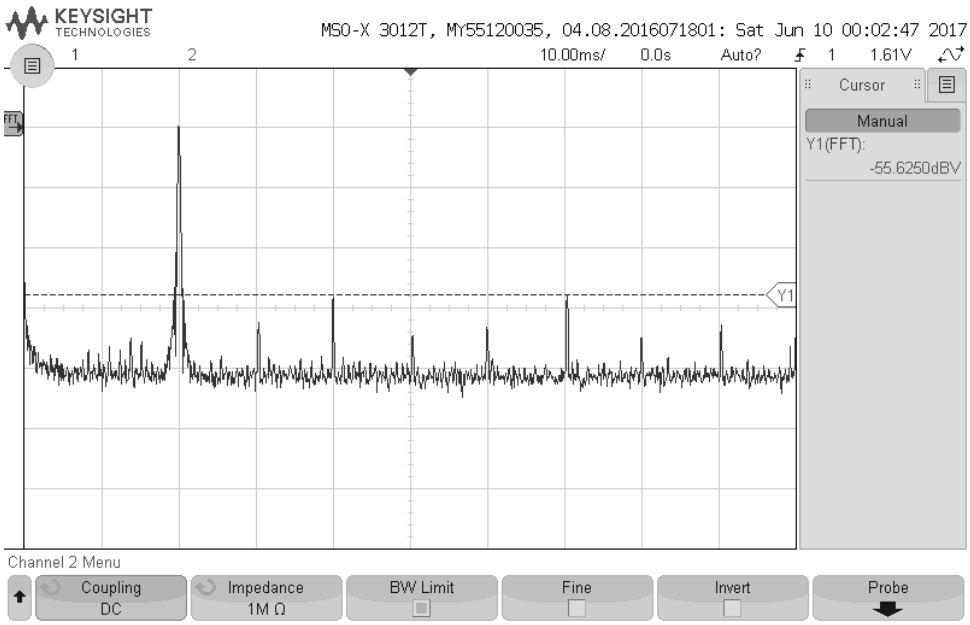
Figur D.6: Frekvensanalyse av prefiltrert testsignal (MATLAB-generert) på 1 kHz uten NFC.



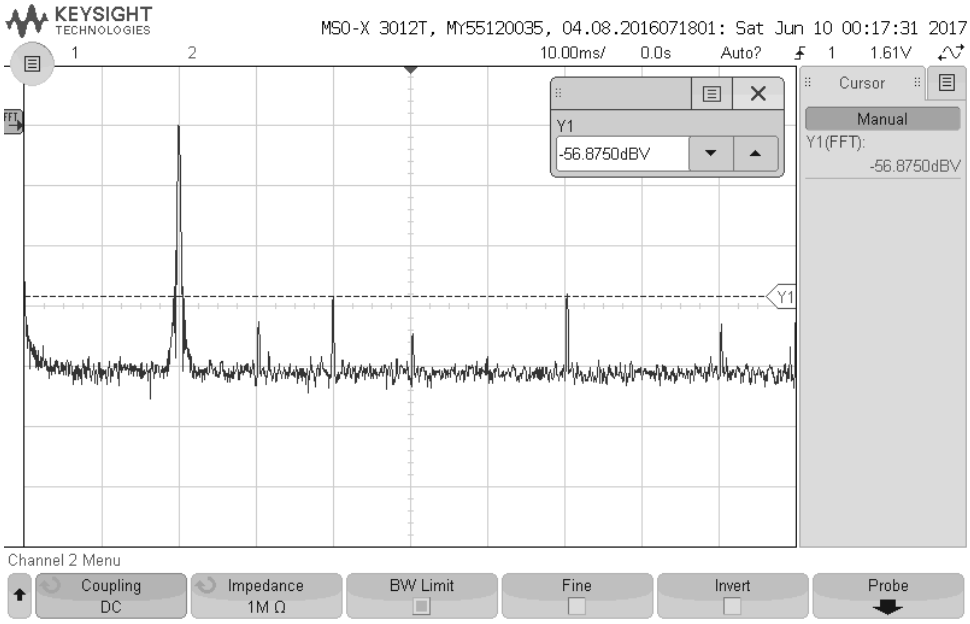
Figur D.7: Frekvensanalyse av prefiltrert testsignal på 1 kHz med NFC. Her er mer støy synlig siden det ikke er brukt begrenset båndbredde (20 MHz) på målingene.



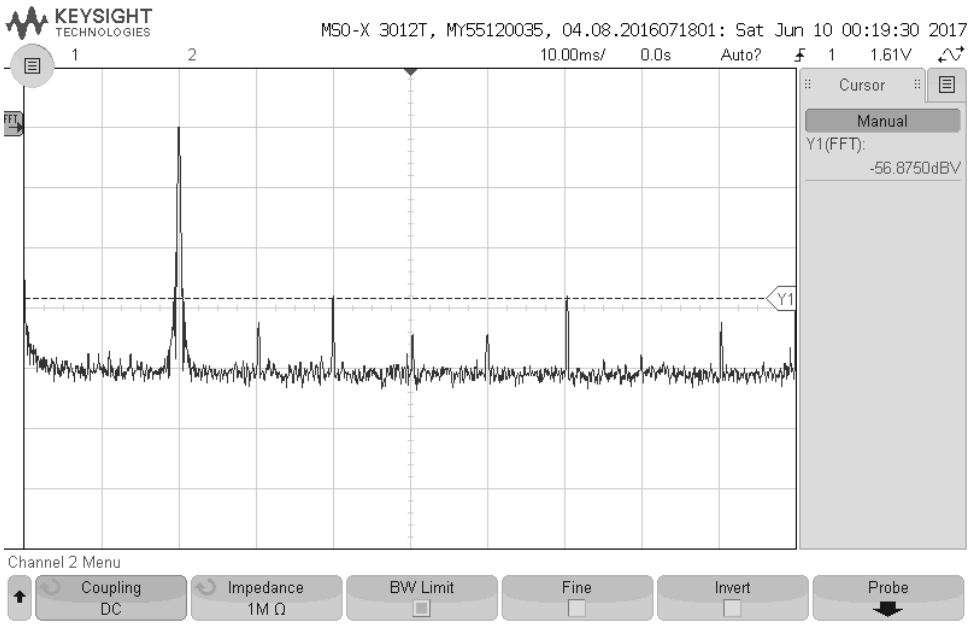
Figur D.8: Frekvensanalyse av prefiltrert testsignal på 2 kHz med NFC.



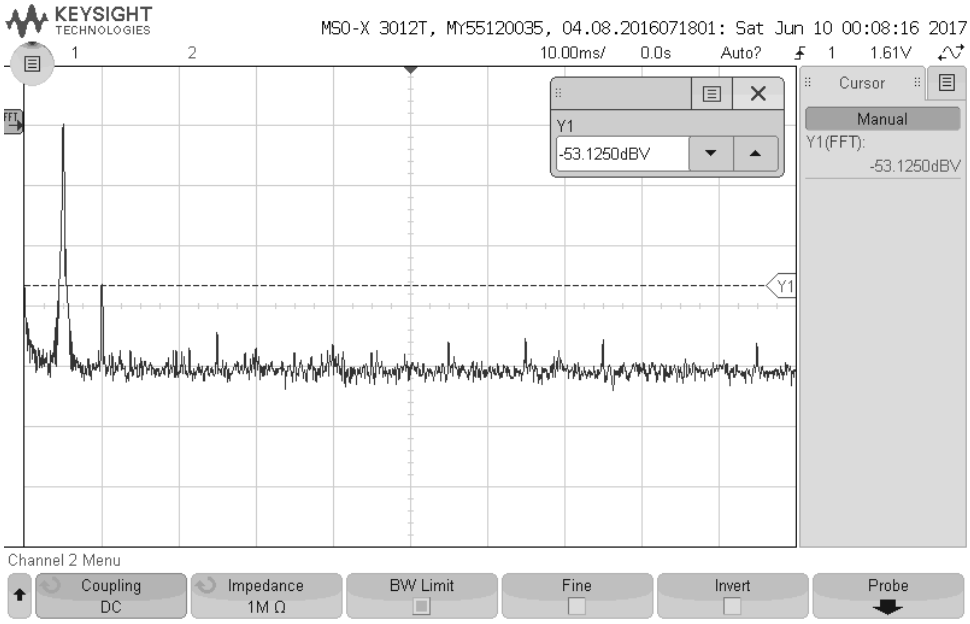
Figur D.9: Frekvensanalyse av prefiltrert testsignal på 2 kHz uten NFC.



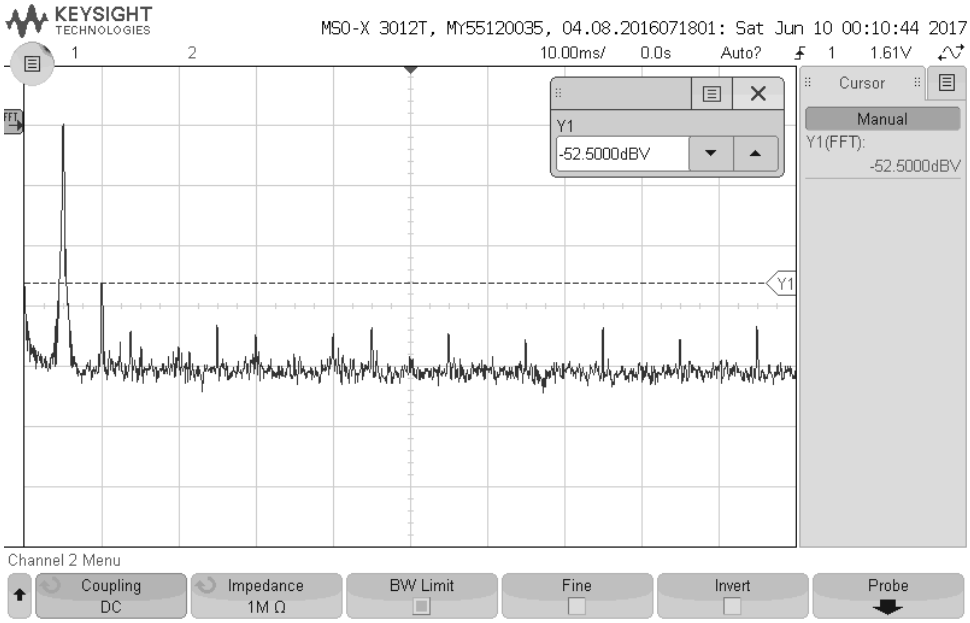
Figur D.10: Frekvensanalyse testsignal på 2 kHz uten prefiltrering med NFC.



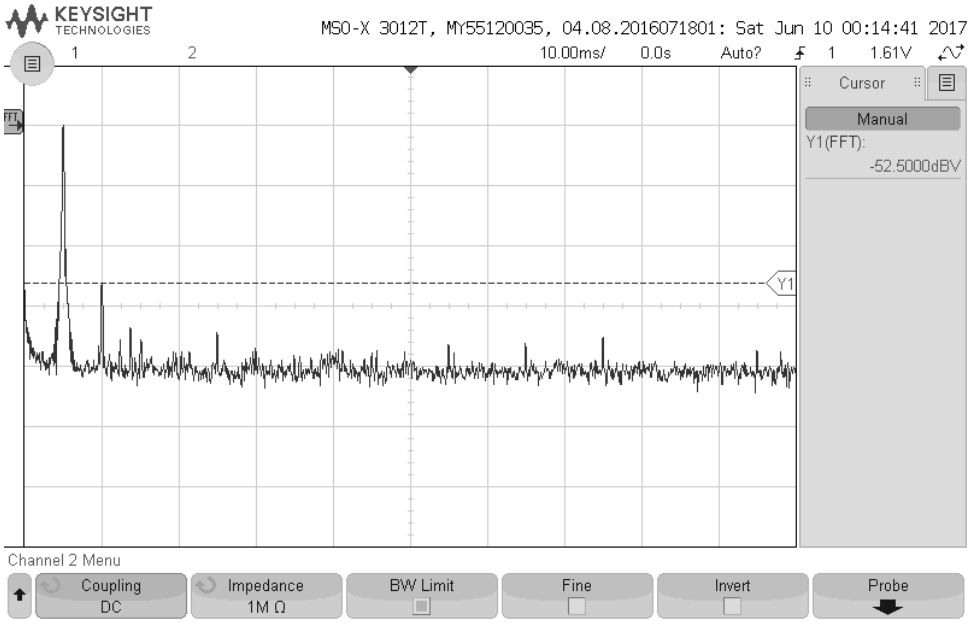
Figur D.11: Frekvensanalyse av testsignal på 2 kHz uten prefiltrering uten NFC.



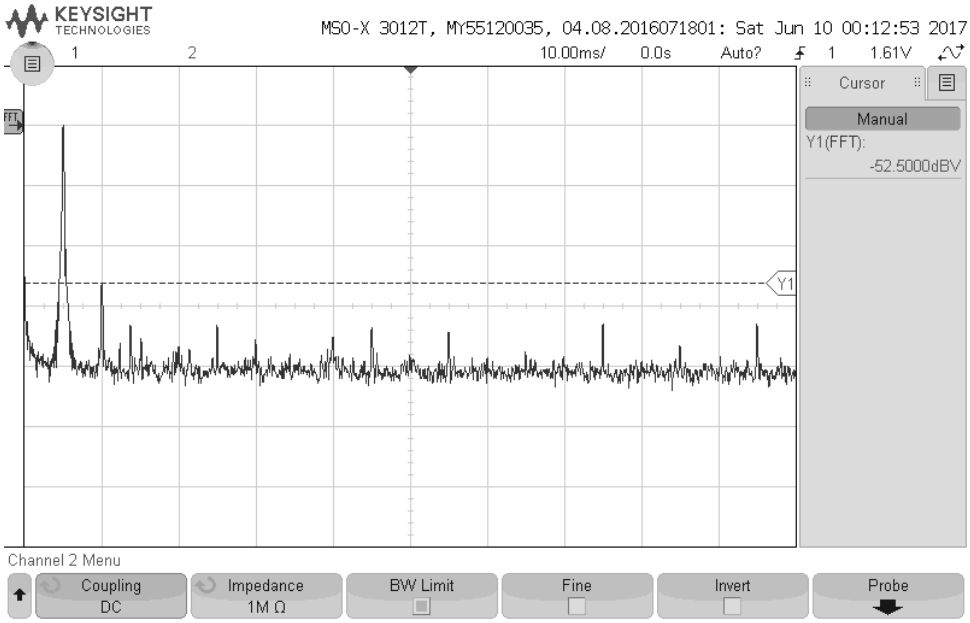
Figur D.12: Frekvensanalyse av prefiltrert testsignal på 500 Hz med NFC.



Figur D.13: Frekvensanalyse av prefiltrert testsignal på 500 Hz uten NFC.



Figur D.14: Frekvensanalyse testsignal på 500 Hz uten prefiltrering med NFC.



Figur D.15: Frekvensanalyse av testsignal på 500 Hz uten prefiltrering uten NFC.

E | Programfiler

- **main.c**: Hovedprogram bestående av følgende funksjoener:
  - main(): Initialiserer buffere og kaller på initKort() og HRPWM_Config(). Eventuelt beregning av $\hat{x}(n)$ i forkant av PWM-mapping. Funksjonen ender i en evig løkke for å vente på avbrudd.
  - lagFilter(): Lager filterkoeffisienter for trunkerte filtre h_3 og h_5.
  - direkte(): Direkte beregning av $\hat{x}(n)$ ved hjelp av for-løkker.
  - filter32(): Beregning av $\hat{x}(n)$ ved hjelp av FIR32.asm.
  - ewpm1_isr(): Avbruddsrutine som først kaller på update_compare() og eventuelt deretter enten direkte() eller filter32().
  - update_compare(): Oppdater pulsbredder.
  - initKort(): Initialisering av DSC (utviklingskort).
  - HRPWM_Config(): Konfigurering av PWM- og HRPWM-modul.
  - error(): Stopper debugging hvis feil oppstår.
- **CBUF.h**: Funksjoner for kontroll av buffer.
- **FIR32.asm**: Funksjoner for initialisering og beregning av FIR-filter.
- **2837xD_RAM_lnk_cpu1.cmd**: Hovedminnetildelig for alle variabler.
- **F2837xD_Headers_nonBIOS_cpu1.cmd**: Minnetildeling for periferiutstyr.

Figurer

Tabeller