SAMMENDRAG AV HOVEDPROSJEKT

Tittel:	Digitalt filter	Nr. : 6 Dato : 23/5-01
Deltaker(e):	Pål Anders Floor Even Berge	
Veileder(e):	Arne Wold	
Oppdragsgiver:	Alcatel Space Norway	
Kontaktperson:	Øyvind Andreassen	
Stikkord (4 stk)	”Multirate teori”, ”Digitalt antialiasing filter”, Quartus, VHDL	
Antall sider: 116	Antall bilag: 0	Tilgjengelighet (åpen/konfidensiell): Åpen

Kort beskrivelse av hovedprosjektet:

Oppgaven går i hovedsak ut på å simulere og vurdere ulike filterstrukturer for en DVB prosessor, og komme med en konklusjon på den best egnede strukturen.Utviklingsverktøyet QuartusII skal i tillegg testes for å se om det er egnet til bruk ved senere implementering.

Oppgaven inneholder en ”multirate teori” del som er nødvendig bakgrunnsteori i forbindelse med hovedemne i oppgaven, som er et ”digitalt antialiasing filter”(DAF). DAF filterets virkemåte og struktur drøftes inngående. QuartusII benyttes for å gi en VHDL beskrivelse av DAF filteret, dokumentasjonen av dette er formet som en bruker manual til QuartusII.

En konklusjon angående filterstrukturen ble gjort, og en VHDL beskrivelse ble programmert. VHDL koden ble simulert, for å se om den fungerer som forutsatt.

Forord

Denne rapporten er et resultat av en hovedoppgave utført av Even Berge og Pål Anders Floor, ved Høgskolen i Gjøvik våren 2001. Vi går begge studieretningen teleteknikk på elektro linjen, hvor hovedoppgaven er et 4 vektalls emne som avslutter den 3 år lange ingeniørutdanningen.

Vi valgte denne oppgaven fordi vi synes digital signalbehandling er et interessant område, og fordi oppgaven virket utfordrende.

Vår oppdragsgiver er Alcatel Space Norway(ASN) i Horten, som er Norges ledende selskap innen utvikling og produksjon av elektronikk for romfartsformål. Hos ASN er Øyvind Andreassen vår kontakt person.

Prosjekt veilderen ved skolen er Arne Wold. Foruten at han er veilederen vår, er han faglærer i fagene Høyfrekvensteknikk, Digital signalbehandling, Digital kommunikasjon og Data kommunikasjon.

Vi vil gi en ekstra takk til følgende personer:

Øyvind Andreassen, for at han tok seg tid til å formulere denne spennende oppgaven, og for alltid å ha vært behjelpelig ved hendvendelser.

Arne Wold, for veldig bra oppfølging av prosjektet, og god bistand innen digital signalbehandling.

Ole Jonny Berg (lærer ved Høgskolen i Gjøvik), for å ha bidratt med mye hjelp og støtte i forbindelse med VHDL delen i prosjektet.

Bjørn Roger Andersen (Telenor Satellite Services), for å ha bidratt med veldig gode teoretiske forklaringer på spesielt vanskelige emner innenfor filter teorien i prosjektet.

Vidar Ringset (SINTEF), for å ha bidratt med veldig gode forklaringer innenfor filter teorien.

Nikolay Rognlien (Arrow), for å ha gitt oss gode råd i forbindelse med problemer rundt QuartusII.

Gjøvik, 23 mai 2001

Pål Anders Floor   Even Berge

_______________   _______________

Innhold

1. INNLEDNING 1
1.1. OPPGAVE DEFINERING 1
1.2. MÅL 1
1.3. MOTTAKERGRUPPE 1
1.4. STUDENTENES FAGLIGE BAKGRUNN 2
1.5. ARBEIDSFORM 2
1.6. ORGANISERING AV RAPPORTEN 2
2. PRINSIPPER – TEORI 4
2.1. GRUNNLEGGENDE DIGITAL SIGNALBEHANDLING 4
2.1.1. Foldning 4
2.1.2. Aliasing 4
2.1.3. Fourier transformen 4
2.1.4. Diskrete fourier transformen (DFT) 5
2.1.5. Vindus funksjoner 5
2.1.6. Laplace transformen 6
2.1.7. Z-transformen 7
2.1.8. Digitale filtre 7
2.1.9. Korrelasjon 9
2.2. PROGRAMMERBAR LOGIKK OG GRUNNLEGGENDE VHDL 10
2.2.1. FPLD og MPLD 10
2.2.2. VHDL 11
2.2.2.1. Bibliotek 11
2.2.2.2. Entiteter 11
2.2.2.3. Arkitektur 11
2.3. STØY 14
2.3.1. Hvit støy. 14
2.3.2. Kvantiseringsstøy 15
2.3.3. Avrundingsstøy. 16
2.4. GRUNNLEGGENDE MULTIRATE TEORI 18
2.4.1. Multiratesystem 18
2.4.2. Representasjon av diskrete signaler 18
2.4.2.1. Diskret sampling 18
2.4.2.2. Polyfaserepresentasjon 21
2.4.2.3. Modulasjonsrepresentasjon 23
2.4.3. Endring av sampelfrekvens 25
2.4.3.1. Reduksjon av sampelfrekvens(Decimation) 31
2.4.3.2. Øking av sampelfrekvens(Interpolation). 32
2.4.4. Half Band Filtre(HBF). 32
2.4.4.1. Antikausalt Half-Band Lav-Pass Filter(HBF-LP) 32
2.4.4.2. Kausalt Half Band Lavpass Filter. 34
3. APPARATUR – UTSTYR 36
4. UTFØRELSEN 37
4.1. TEKNISK OPPGAVE BESKRIVELSE 37
4.2. INNLEDNING TIL DAF. 37
4.3. DAF FILTER 39
4.3.1. Matematisk utledning av transferfunksjonen 40
4.3.2. Bestemmelse av filter lengde 41
4.3.3. Manuell utregning av filter koeffisienter 45
4.3.4. Overføring til båndpass respons 46
4.3.5. Bestemmelse av antall bit for representasjon av koeffisientene 48

4.3.6. Test av filterets koeffisient vektor ... *52*
4.3.7. Overføring til struktur ... *54*
4.3.8. Overgang til praktisk design ... *55*
Total støy på DAF filterets utgang ... *58*
4.4. QUARTUS ... 58
4.4.1. Oppstart av Quartus ... *58*
4.4.2. Opprettelse av nytt prosjekt ... *58*
4.4.3. Opprettelse av BDF fil ... *60*
4.4.4. Programmering av entiteten "downsampler" ... *61*
4.4.5. Opprettelse av multiplikasjons enhetene ... *65*
4.4.6. Innsetting av multiplikasjons entitetene i designet ... *68*
4.4.7. Programmering av entiteten "round_a" ... *69*
4.4.8. Opprettelse av tidsforsinkelses enhetene ... *71*
4.4.9. Opprettelse av addisjons kretser ... *72*
4.4.10. Konstruksjon av subtraksjons kretsene ... *75*
4.4.11. Opprettelse av entiteten "round_b" ... *75*
4.4.12. Opprettelse av entiteten "round_c" ... *75*
4.4.13. Innsetting av "delay_13", "delay_12", "delay_8", "add" og "sub" I BDF filen ... *75*
4.4.14. Innsetting avI/O pinner ... *76*
4.4.15. Trekking av forbindelser ... *77*
4.4.16. Kompilering ... *80*
4.4.17. Simulering ... *84*
4.4.18. Megacore funksjonen "FIR compiler" ... *91*

5. RESULTATER ... 93

5.1. SOFTWARE SIMULERING - FUNKSJONELL ... 93
5.2. SOFTWARE SIMULERING – TIMING ... 94

6. DISKUSJON AV RESULTATENE ... 96

7. KONKLUSJON ... 98

8. LITTERATURLISTE ... 99

9. VEDLEGG ... 100

VEDLEGG A – MATLAB KODEN TIL "H_DAF.M" ... 101
VEDLEGG B – MATLAB KODEN TIL "INN_SIGNAL.M" ... 103
VEDLEGG C – MATLAB KODEN TIL "TEST_DAF.M" ... 104
VEDLEGG D – BLOKKSKJEMATISK OVERSIKT OVER VHDL TOPP FILEN ... 106
VEDLEGG E – UTDRAG AV VHDL KODEN ... 107

1. Innledning

1.1. Oppgave definering

Alcatel Space Norway i Horten har definert følgende oppgave:

Det skal gjennomføres en oppgave med å simulere og vurdere diverse filterstrukturer for en DVB prosessor. Mulige filterstrukturer og en beskrivelse av systemet blir gitt fra oppdragsgiver i prosjektet.

Et sentralt emne vil være valg av design og modellerings verktøy rettet mot fremtidig implementering. Det skal vurderes en designpakke fra Altera og se om denne er egnet for problemstillingen. Det skal i parallell gjøres simulering og modellering i MATLAB.

Etter vurderingen av verktøy skal de konkrete filtrene vurderes og andre filterstrukturer / løsninger undersøkes og sammenlignes.

Det ble valgt å samle all teknisk beskrivelse av problemstillingen i avsnitt 4.1.

1.2. Mål

Målene med oppgaven er:

Hovedmål:

Få god innsikt og forståelse av ulike filterstrukturer, og videreføre dette til implementering og testing av digitale filtre.

Oppgaven kan ut fra dette deles inn i følgende delmål:

- Lære et utviklingsverktøy fra Altera, og vurdere om dette er et egnet verktøy for ”implementering” og uttesting av ønskede filtre.
- Ved simuleringer og beregninger komme fram til den best egnede filterstrukturen.

1.3. Mottakergruppe

Denne rapporten er beregnet for personer med ingeniør utdanning inne teleteknikk/elektronikk. De fag som spesielt forutsettes for å få utbytte av denne rapporten er matematikk på ingeniør nivå, signalteori/digital signalbehandling, grunnleggende statistikk og grunnleggende VHDL programmering.

1.4. Studentenes faglige bakgrunn

Begge studentene går studieretningen teleteknikk ved Høgskolen i Gjøvik, i klassen 98HINEA.

1.5. Arbeidsform

Arbeidsformen har i hovedsak gått ut på å tilegne seg nødvendige kunnskaper gjennom litteratur studier, og telefon samtaler med aktuelle personer.
Ved prosjektets oppstart ble det gjennomført et bedriftsbesøk hos Alcatel Space Norway i Horten, i tillegg ble det deltatt på et møte for avslutningen av fase2 i prosjektet ”Nordic onboard DVB processor project” på Norsk Romsenter i Oslo.

1.6. Organisering av rapporten

Rapporten er delt opp på følgende måte:

”Kapitel 1, Innledning”.

”Kapitel 2, prinsipper – teori” inneholder aktuelt bakgrunns stoff for å løse oppgaven, og er delt inn på følgende måte:

> Avsnitt 2.1 inneholder en kort innledning om emner fra signalteori og digital signalbehandling. Dette er stoff som forutsettes kjent, men er tatt med i korte trekk som en naturlig del av dette prosjektet.
>
> Avsnitt 2.2 inneholder en kort innledning om programmerbar logikk og grunnleggende sammenhenger i VHDL.
>
> Avsnitt 2.3 inneholder en innføring i nødvendig støy teori for denne oppgaven. Dette er stoff som til dels har blitt studert under prosjektet, og til dels forutsettes kjent.
>
> Avsnitt 2.4 inneholder grunnleggende multirate teori som er nødvendig tilleggsteori til digital signal behandling for å få teoretisk dekning for hoveddelen i rapporten. Dette er stoff som har blitt studert under prosjektet.

”Kapitel 3, Apparatur – utstyr” inneholder spesifikasjoner for nødvendig utstyr for løsning av oppgaven.

”Kapitel 4, Utførelsen” inneholder all informasjon angående løsningen av problemstillingen, og er delt inn på følgende måte:

> Avsnitt 4.1 inneholder en teknisk oppgave beskrivelse.

Avsnitt 4.2 inneholder en generell beskrivelse av et *Digitalt Antialiasing Filters* virkemåte og betydning.

Avsnitt 4.3 inneholder en inngående teoretisk beskrivelse av DAF filteret basert på gitt struktur. Avsnittet omhandler også hvordan dette filteret kan realiseres.

Avsnitt 4.4 omhandler softwaren QuartusII fra Altera. Det ble ytret ønske fra skolen om at denne delen av rapporten kunne skrives som en bruker manual for QuartusII, da den ønskes brukt i undervisnings sammenheng senere. Denne delen er derfor utformet som en bruker manual, hvor DAF filteret brukes som et gjennomgående eksempel.

”Kapitel 5, Resultater” inneholder resultater fra teoretisk uttesting av den programmerte filter strukturen, i form av simulerings resultater.

”Kapitel 6, Diskusjon av resultatene” inneholder en drøfting av resultatene i kapitel 5.

”Kapitel 7, Konklusjon”.

”Kapitel 8, Litteraturliste”.

”Kapitel 9, Vedlegg”.

2. Prinsipper – teori

2.1. Grunnleggende digital signalbehandling

Innholdet i dette avsnittet er stoff som i stor grad skal være kjent fra fagene ”Signalteori” og ”Digital signal behandling”, men er så sentralt for denne prosjekt oppgaven at det velges å gi en kort oppsummering. Stoffet er basert på bøkene ”System analysis and signal processing”, av: ”Denbigh”, og ”Advanced digital signal processing”, av: ”Zelniker” og ”Taylor”.

2.1.1. Foldning

Anta et lineært system med impulsrespons h(t), påtrykt innsignalet x(t). For å bestemme utsignalet y(t) i tidsdomene foldes x(t) med h(t).
Dette kalles kontinuerlig foldning og har følgende definisjon:

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau) \cdot h(t-\tau) d\tau \qquad (2.1\text{-}1)$$

For et diskret(digitalt) system med impulsrespons h(n), påtrykt innsekvensen x(n) blir utgangssekvensen y(n) gitt ved diskrete foldning som har følgende definisjon:

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k) \cdot h(n-k) \qquad (2.1\text{-}2)$$

Foldning i tidsdomene tilsvarer det samme som multiplikasjon i frekvensdomene, og grunnet dualiteten til transformene som benyttes mellom domenene er også foldning i frekvensdomene det samme som multiplikasjon i tidsdomene.

2.1.2. Aliasing

Før et kontinuerlig signal samples må det båndbredde begrenses slik at dets høyeste frekvenskomponent er lavere enn halve samplingsfrekvensen. Dette er for å hindre at nedre sidebånd av det samplede signalet (PCM) overlapper spekteret til det opprinnelige signalet. Dette kan ses på som en foldning om halve samplingsfrekvensen. En slik overlapping kalles aliasing og dersom den forekommer vil uønskede frekvenskomponenter tilføyes i det gjenskapte kontinuerlige signalet.

2.1.3. Fourier transformen

For å kunne transformere et kontinuerlig signal over i frekvensdomene benyttes den kontinuerlig Fourier transformen som har følgende definisjon:

$$F(\mathbf{w}) = \int_{-\infty}^{\infty} f(t) \cdot e^{j\mathbf{w}t} dt \qquad (2.1\text{-}3)$$

Absolutt verdien til dette uttrykket gir amplitudespekteret til signalet, og argumentet gir fasespekteret.

For å finne frekvens responsen til et lineært system med impulsresponsen h(t), kan Fourier transformen benyttes. Det tas nå kun hensyn til systemet i dets stasjonære tilstand, transient perioden vil ikke bli tatt hensyn til.

For å transformere fra frekvensdomene til tidsdomene, benyttes den invers Fourier transformen som har følgende definisjon:

$$f(t) = \frac{1}{2\mathbf{p}} \int_{-\infty}^{\infty} F(\mathbf{w}) \cdot e^{j\mathbf{w}} d\mathbf{w} \qquad (2.1\text{-}4)$$

2.1.4. Diskrete fourier transformen (DFT)

Et diskre signal med N sampler kan transformeres over til frekvensdomene ved å benytte DFT. Definisjonen på DFT er følgende:

$$X(m) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j\frac{2\mathbf{p}mn}{N}}, 0 \le m \le (N-1) \qquad (2.1\text{-}5)$$

For å transformere fra frekvensdomene til tidsdomene, benyttes den invers diskre Fourier transformen som har følgende definisjon:

$$x(n) = \frac{1}{N} \cdot \sum_{m=0}^{N-1} X(m) \cdot e^{j\frac{2\mathbf{p}mn}{N}}, 0 \le n \le (N-1) \qquad (2.1\text{-}6)$$

Fast Fourier transformen (FFT) er en effektiv algoritme for å beregne DFT dersom antall sampler er 2^N.

2.1.5. Vindus funksjoner

Når blant annet DFT skal benyttes i praksis må det tas utgangspunkt i et endelig antall sampler. Dette er det samme som å multiplisere datasettet med et rektangulært vindu. Siden frekvensspekteret til et rektangulært vindu har formen som en sinc kurve, vil en slik multiplikasjon tilsvarer en foldning mellom signalet og en sinc kurve i frekvensdomene. Dette vil gi feil i spekteret til signalet grunnet rippel kalt ”leakage”. For å minimalisere denne effekten benyttes såkalte vindus funksjoner som har lavere sidelober og derfor mindre ”leakage” fra hovedloben, eksempler på noen av disse er Han, Hamming og Kaiser. Han og Hamming vinduer har den karakteristikken at større dempingen av sidelobene fører til en bredere hovedlobe. Kaiser vinduet derimot er mer fleksibelt ved at det er mer kontroll over

hovedlobe bredde og sidelobe amplitudene ved en gitt lengde N. Kaiser vindu med lengde N = 2n + 1 er gitt av følgende uttrykk:

$$\boldsymbol{w}_{k(k)} = \frac{I_0\left(\boldsymbol{b}\sqrt{1-\left(\frac{k}{m}\right)^2}\right)}{I_0(\boldsymbol{b})}, k \leq m$$

$$\text{,hvor } I_0(x) = 1 + \sum_{k=1}^{\infty}\left(\left(\frac{1}{k!}\cdot\frac{x}{2}\right)^k\right)^2 \qquad (\ 2.1\text{-}7\ )$$

Vindus lengden N bestemmer hovedlobe bredden, mens sidelobe bredden er konstant. Større N ⇒ smalere hovedlobe. β er et parameter som hovedsakelig endrer sidelobe nivået men den påvirker også hovedlobe bredden. Større β ⇒ mindre sidelobe nivå og noe bredere hovedlobe.

2.1.6. Laplace transformen

Laplace transformen av en tids funksjon f(t) er gitt av:

$$F(s) = \int_0^{\infty} f(t)\cdot e^{-st}dt \qquad (\ 2.1\text{-}8\ )$$

Denne transformerer f(t) over i s-planet. For å finne transferfunksjonen til et kontinuerlig lineært system kan Laplace transformen benyttes. Ettersom s = σ +jω ⇒ Laplace transformen tar hensyn til systemets transiente periode, dvs egnesvigninger. For å finne systemets amplitude- og faserespons, settes σ = 0 ⇒ s = jω, dette blir det samme som å ta Fourier transformen av et kausalt system.

Dersom en tar $e^{st} = e^{t(\sigma + j\omega)}$ i betraktning, vil dette beskrive en eksponensielt avtagende svingning dersom σ er negativ ⇒ dersom et lineært systems poler ligger i venstre halvplan av det komplekse plan(hvor σ er negativ) vil systemet være stabilt.

Et systems gruppe forsinkelse har følgende definisjon:

$$T_g = -\frac{d\boldsymbol{f}}{d\boldsymbol{w}} \qquad (\ 2.1\text{-}9\ )$$

, hvor φ(ω) er systemets fase respons.

2.1.7. Z-transformen

Z-transformen av en diskret tids funksjon x(n) er gitt av:

$$X(z) = \sum_{n=-\infty}^{\infty} x(n) \cdot z^{-n} \qquad (2.1\text{-}10)$$

Denne transformere x(n) over i z-planet. For å finne transferfunksjonen til et diskret lineært system kan z-transformen benyttes. Sammenhengen mellom den uavhengige variabelen s i Laplace transformen og z i z-transformen, er som følgende:

$$z = e^{st_s} \qquad (2.1\text{-}11)$$

For å finne et lineært systems amplitude- og faserespons, settes z lik $e^{\omega ts}$ som er lik $e^{j\Omega}$(dvs. å sette σ i s lik 0).

Dersom et systems poler ligger innenfor enhetssirkelen i det komplekse planet, er systemet stabilt.

Et systems gruppe forsinkelse har følgende definisjon:

$$T_g = -\frac{d\boldsymbol{f}}{d\Omega} \qquad (2.1\text{-}12)$$

, hvor $\phi(\Omega)$ er systemets fase respons.

2.1.8. Digitale filtre

Det finnes to hoved typer digitale filtre. Disse er ”Finite Impulse Response”(FIR) filtre som har en endelig impulsrespons, og ”Infinite Impulse Response”(IIR) filtre som har en uendelig impulsrespons. Dersom et FIR filter har symetrisk impulsrespons, vil det ha lineær fase i passbåndet $\Rightarrow$ konstant gruppeforsinkelse. Et IIR filter vil aldri ha lineær fase.

For konstruksjon av et digitalt IIR filter brukes det vanligvis tre metoder:

- Plassering av poler og nullpunkter direkte i det komplekse plan.

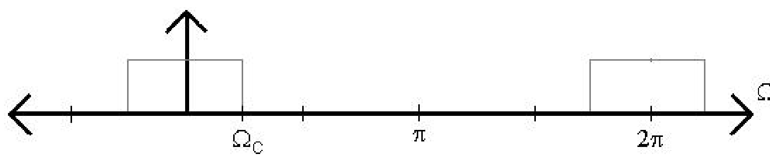
- Impulsinvariant metoden.
  Denne ta utgangspunkt i en kjent analog impulsrespons, og prøver å tilnærme denne med diskrete verdier.

- Bilineær transform.
  Denne tar utgangspunkt i en kjent analog impulsrespons, og foretar følgende substitusjon:

$$s \longrightarrow 2 \cdot f_s \frac{z-1}{z+1} \qquad (2.1\text{-}13)$$

For konstruksjon av et digitalt FIR filter brukes det vanligvis to metoder:

- Fourier metoden.
  Denne metoden tar utgangspunkt i en antikausal lavpass "brickwall" respons. Det som ønskes er responsen senterert rundt $\Omega = 0$, men ettersom systemet er digitalt vil responsen gjenta seg periodisk. Dette blir som vist på Figur 2.1.1.



Figur 2.1.1 Brickwall respons

Som kjent vil invers Fourier transformen av dette gi en sinc som ikke er kontinuerlig, men vil bestå av impuls funksjoner med avstand t_s vektet med verdier som fremkommer av følgende uttrykk:

$$b_i = t_s \int_{-\frac{f_s}{2}}^{\frac{f_s}{2}} A(f) \cdot e^{j2\pi i t_s f} df, \quad i \in \mathrm{N} \qquad (2.1\text{-}14)$$

Dette kan også uttrykkes ved Ω_C ettersom $\Omega_C = 2\pi f t_s$. Ettersom funksjonen ideelt sett er 0 mellom Ω_C og fs/2, kan integrasjonsgrensene gå fra $-\Omega_C$ til Ω_C og følgende integral oppnås:

$$b_i = \frac{1}{2\pi} \int_{-\Omega_c}^{\Omega_c} e^{j\Omega i} d\Omega, \quad i \in \mathrm{N} \qquad (2.1\text{-}15)$$

Dersom integrasjonen utføres, og forskjellige verdier av i settes inn, vil de forskjellige koeffisientene fremkomme. For å minke rippelen som oppstår ved å gjøre antall koeffisienter endelig, kan koeffisient vektoren multipliseres med en vindus funksjon.

- Parks & McClellan.
  Dette er en optimal måte å konstruere FIR filtre på. Den er basert på Remez algoritmen, denne algoritmen er veldig matematisk komplisert og ligger utenfor prosjektoppgavens omfang.
  Metoden gir koeffisient verdier som er mest mulig optimale med tanke på minst mulig feil mellom den ønskede ideelle responsen og den egentlige responsen, representert med et gitt endelig antall koeffisienter.

2.1.9. Korrelasjon

Autokorrelasjons funksjonen oppstår når et signal korreleres med seg selv. Signalets autokorrelasjons funksjon fremkommer da av følgende uttrykk:

$$r_{xx}(\tau) = \int_{-\infty}^{\infty} x(t) \cdot x(t+\tau) dt \qquad (2.1\text{-}16)$$

2.2. Programmerbar logikk og grunnleggende VHDL

Innholdet i dette avsnittet er stoff basert på boken ”Digital systems design and protoyping using field programmable logic”, av: ”Salcic” og ”Smailagic”.

2.2.1. FPLD og MPLD

FPLD står for Field Programmable Logic Devices, mens MPLD står for Mask PLD. Begge inneholder en ”array” av logiske celler(LC) som kan koples sammen ved programmering for å implementere forskjellige design med forskjellige funksjoner. Forskjellen på FPLD og MPLD er: MPLD masse programmeres ut fra de gitte spesifikasjoner, ved å legge metallforbindelser(masker) direkte i selve kretsen(kan derfor aldri omprogrammeres). Disse er konstruert for bruk i stor skala. FPLD programmeres ved bruk av elektronisk programmerbare svitsjer, og kan derfor programmeres på en enkel måte med forholdsvis enkelt utstyr. Det er her snakk om å programmere en brikke for et spesielt formål(i mindre skala). Noen FPLDer kan omprogrammeres.

En logisk celle i en FPLD kan være alt fra en transistor til en komplisert mikroprosessor. Den er i stand til å implementere både kombinatorisk og sekvensiell logikk.

FPLDer på markedet i dag har LCer som bygger på en av følgende måter:
- Transistorpar.
- Grunnleggende små porter, som to inputs NAND eller XOR.
- Multipleksere.
- Loock up tables(LUT).
- Wide fan in AND-OR strukturer.

Vanlige programmeringsteknologer er:
- Statisk RAM (SRAM), hvor selve svitsjen er en transistor styrt av tilstanden til et SRAM bit.
- EPROM, hvor svitsjen er en ”floating gate” transistor som sperres ved å injisere ladninger i ”floating gate”.
- Antifuse, som danner forbindelse ved å ”svi av” isolasjonsmaterialet i en ”sikring” som sitter i et array for å lage forbindelse. Når den er programmert har den kun lavohmige veier, dvs at den er optimal i forhold til høye hastigheter.

Dersom programmeringsteknologier og interne arkitekturer kombineres, fås følgende tre hovedkategorier:
-SRAM Field Programmable Gate(Logic) Arrays(SRAM FPGA).
SRAM celler holder programmet. Logikken er implementert som loock up tables(LUT).
-Complex Programmable Logic Devices(CPLD).
Programmeres vha EPROM eller EEPROM transistorer for å danne wide fan in porter.
-Antifuse FPGA.
Programmeres ved Antifuse. Logikken er implementert som LUT.

2.2.2. VHDL

VHDL er VHSIC(Very High Speed Integrated Circuits) Hardware Description Language, dette brukes for å utvikle, dokumentere, simulere og syntetisere digitale systemer og brikker. VHDL er akseptert av som en IEEE standard, og er derfor kompatibelt med de fleste leverandører av programmerbar logikk.

VHDL består av flere deler:

- VHDL språket
- Et WORK bibliotek, reservert for bruker design
- Leverandør pakker, med leverandør biblioteker
- Bruker pakker, med bruker bibliotek

Design metoden som brukes er alltid top-down, dvs. at det først tas utgangspunkt i et abstrakt system(entiteter) og etterhvert går ned på en mer detaljert beskrivelse av oppførselen(arkitekturen).
Kommandoer og datatyper skrives som oftest med store bokstaver, mens brukerdefinerte objekter skrives med små bokstaver. VHDL er ikke *case sensitive*, men det er likevel vanlig å gjøre ovenforstående, for å øke lesbarheten i koden.

2.2.2.1. Bibliotek

Det er to innebygde biblioteker i VHDL, disse er WORK og STANDARD. VHDL blir kompilert inn i WORK, hvis ikke noe annet er spesifisert.
For å få adgang til et bibliotek og dets enheter, må biblioteket først deklareres ved hjelp av følgende syntaks:

LIBRARY <bibliotek navn>;

Enheter i biblioteket kan nå fås adgang til på en av følgende tre måter:

- USE <bibliotek navn>.<pakke navn>.<enhets navn>
- USE <bibliotek navn>.<enhets navn>
- USE <enhets navn>, denne er brukt dersom en jobber i WORK.

Ved å skrive <all> som <enhets navn> i det øverste alternativet, fås det tilgang til alle enheter under den aktuelle pakken i biblioteket.

2.2.2.2. Entiteter

En *entitet* definerer nye komponent navn, dens inngangs- og utgangstilkoblinger, og relaterte deklarasjoner. Entiteter representerer I/O grensesnittet til et komponent design, dvs. at VHDL skiller I/O grensesnittet til en komponent fra dens arkitektur. Entiteten beskriver typen og retningen til signaltilkoblingen, mens arkitekturen beskriver oppførelsen. Etter en entitet er kompilert inn i et bibliotek, kan den brukes som en komponent i et annet design.

Syntaks brukt for å deklarere en entitet er som følger:

```
ENTITY <entitets navn> IS

        PORT(<deklarasjon av innganger og utganger>);

END <entitets navn>;
```

2.2.2.3. Arkitektur

Arkitekturen spesifiserer oppførselen, innbyrdes forbindelser og komponentene til en på forhånd definert entitet. Den spesifiserer sammenhengen mellom innganger og utganger, som må uttrykkes i form av oppførsel, dataflyt eller struktur. En entitet kompileres alltid før dens arkitektur, dvs at dersom en entitet rekompileres må også alle dens arkitekturer rekompileres. En entitet kan implementeres med mer enn en arkitektur. Alle arkitekturene har da indentiske grensesnitt, men hver av dem må ha et unikt arkitekturnavn. Programmereren velger en bestemt arkitektur for entiteten.
Syntaks brukt for å definere arkitektur er som følger:

```
ARCHITECTURE <arkitektur navn> OF <entitets navn> IS

        <arkitektur deklarasjoner(konstanter, variable, signaler etc.)>

BEGIN

        <Arkitektur beskrivelse>

END <arkitektur navn>;
```

Arkitekturbeskrivelsen kan gjøres på tre forskjellige måter:

1). Oppførselsbeskrivelse.
Denne beskriver oppførselen til enheten algoritmisk. Dette skrives på følgende måte:

```
ARCHITECTURE <arkitektur navn> OF <entitets navn> IS

BEGIN
        PROCESS(a, b);

        BEGIN
                <oppførselsbeskrivelse>
        END PROCESS;

END <arkitektur navn>;
```

Signalene prosessen er følsom for(a og b) settes i parenteser etter PROCESS. Prosessen venter på forandringer i innkommende signal, og aktiveres så fort en av signalene prosessen er følsom for endres.

2). Dataflyt beskrivelse.
En dataflyt arkitektur beskriver dataflyt oppførselen til kombinatoriske logiske funksjoner som addere, komparatorer, multipleksere osv.
Når et slikt design er kompilert inn i et bibliotek, kan det brukes som en komponent i et annet design ved å referere til entitetsnavnet.

3). Struktur beskrivelse.
Struktur beskrivelse brukes i hovedsak for å definere forbindelser mellom komponenter.
PORT MAP er en kommando som brukes for å ”koble sammen” signalforbindelsene som skal brukes.

2.3. Støy

Innholdet i dette avsnittet er stoff basert på boken ”The Digital signal processing handbook”, av: ”Madisetti” og ”B.Williams”, og et kompendium av ”Hermann Lia”.

2.3.1. Hvit støy.

I det påfølgende klargjøres begrepet hvit støy.
En prosess autokorrelasjonsfunksjon er som kjent(formel 2.1-16):

$$R(\boldsymbol{t}) = \int_{-\infty}^{\infty} x(t) \cdot x(t+\boldsymbol{t})dt \qquad (\ 2.3\text{-}1\ )$$

Effekttetthetsspekteret til en *stasjonær stokastisk prosess* er Fouriertransformen til prosessens autokorrelasjonsfunksjon.

$$S(\boldsymbol{w}) = \int_{-\infty}^{\infty} R(\boldsymbol{t}) \cdot e^{-j\boldsymbol{wt}} d\boldsymbol{t} \qquad (\ 2.3\text{-}2\ )$$

Den totale effekten over en båndbredde B er gitt av(midlere effekt):

$$P(\boldsymbol{w}) = \frac{1}{2\boldsymbol{p}} \int_{-2\boldsymbol{p}B}^{2\boldsymbol{p}B} S(\boldsymbol{w})d\boldsymbol{w} \qquad (\ 2.3\text{-}3\ )$$

For en prosess hvor korrelasjonslengden er null, vil:

$$R(\boldsymbol{t}) = \boldsymbol{d}(\boldsymbol{t}) \Rightarrow R(\boldsymbol{t}) = \frac{N_0}{2} \cdot \boldsymbol{d}(\boldsymbol{t}) \qquad (\ 2.3\text{-}4\ )$$

Spektraltettheten til denne prosessen blir da(tosidig spektraltetthet):

$$s(\boldsymbol{w}) = \int_{-\infty}^{\infty} \frac{N_0}{2} \cdot \boldsymbol{d}(\boldsymbol{t}) \cdot e^{-j\boldsymbol{wt}} d\boldsymbol{t} = \frac{N_0}{2}. \qquad (\ 2.3\text{-}5\ )$$

Dette er et helt flatt spekter, dvs et *hvitt* spekter.⇒at hvit støy er støy hvor korrelasjonslengden er 0, dvs at det ikke er noen samvariasjon mellom støysignalet og en hvilken som helst forsinket utgave av det.⇒at dersom hvit støy samples, er det ideelt sett ingen ”sammenheng” mellom to sampler, de er helt uavhengig av hverandre.
Total støyeffekt over en viss båndbredde B, for hvit støy vil være:

$$P(\boldsymbol{w}) = \int_{-B}^{B} \frac{N_0}{2} df = N_0 \cdot B \qquad (2.3\text{-}6)$$

Dersom hvit støy i tillegg er Normalfordelt, er det snakk om *Hvit Gaussisk støy* eller AWGN(Additive White Gaussian Noise) som kan beskrives statistisk med T~N(μ,σ).
For AWGN er $\sigma^2 = N_0/2$.

2.3.2. Kvantiseringsstøy.

Når et signal digitaliseres må det representeres med et endelig antall bit. Dette er bestemt av A/D-konverterens oppløsning. Desto bedre oppløsning(i antall bit), desto mindre blir ”feilen”(støyen). En A/D-konverter som bruker B bit vil ha *kvantiseringstrinn* gitt av følgende uttrykk:

$$\Delta = 2^{-B} \qquad (2.3\text{-}7)$$

Dvs at kvantiseringstrinnet blir mindre desto flere bit som benyttes.
Feilen ved å benytte en endelig bitlengde vil være gitt av:

$$\boldsymbol{e} = Q(X) - X \qquad (2.3\text{-}8)$$

Hvor X er det opprinnelige signalet og Q(X) er det kvantiserte representert med en endelig bit-lengde.
ε kalles *kvantiseringsstøy.'*
Feilen grunnet kvantisering kan ses på som en tilfeldig variabel uniform fordelt over det aktuelle ”feil området”.$\Rightarrow$at kalkulasjoner med endelig lengde kan ses på som kalkulasjoner som har blitt feilaktige grunnet additiv hvit uniformfordelt støy med forventningsverdi E og varians σ^2.
For å avgjøre hvilken bit verdi et nivå på inngangen til en A/D-konverter skal tilordnes, kan for eksempel avrunding eller trunkering benyttes.

Støy ved avrunding.

Ved avrunding vil kvantisert verdi nærmest ukvantisert verdi(på det analoge signalet) velges.
Dette medfører at feilen ε_r, vil ligge i følgende intervall:

$$-\frac{\Delta}{2} \leq \boldsymbol{e}_r \leq \frac{\Delta}{2} \qquad (2.3\text{-}9)$$

Støyens forventning vil da være gitt av:

$$m_{\varepsilon_r} = \frac{1}{\Delta} \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} \varepsilon_r d\varepsilon_r = 0 \qquad (2.3\text{-}10)$$

Og variansen er gitt av:

$$\sigma^2 = \frac{1}{\Delta} \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} (\varepsilon_r - m_{\varepsilon_r})^2 d\varepsilon_r = \frac{\Delta^2}{12} \qquad (2.3\text{-}11)$$

<u>Støy ved trunkering.</u>

Ved trunkering avskjæres de minst signifikante bittene(LSB) uten å ta hensyn til nærmeste ukvantiserte verdi(på det analoge signalet). Dette medfører at feilen ε_t, vil ligge i følgende intervall:

$$-\Delta < \varepsilon_t \leq 0 \qquad (2.3\text{-}12)$$

Støyens forventning vil da bli:

$$m_{\varepsilon_t} = \frac{1}{\Delta} \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} \varepsilon_t d\varepsilon_t = -\frac{\Delta}{2} \qquad (2.3\text{-}13)$$

Og variansen blir:

$$\sigma^2 = \frac{1}{\Delta} \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} (\varepsilon_t - m_{\varepsilon_t})^2 d\varepsilon_t = \frac{\Delta^2}{12} \qquad (2.3\text{-}14)$$

2.3.3. Avrundingsstøy.

Avrundingsstøy forekommer når resultatet etter aritmetiske operasjoner som for eksempel multiplikasjon avrundes. For å bestemme denne støyen antas det at støy grunnet kvantisering er stasjonær, hvit og ukorrelert med et systems utgang og interne variabler.
Dersom et lineært system med impulsrespons g(n) blir befengt med støy med middelverdi m_x og varians σ_x^2, får støyen på utgangen en middelverdi:

$$m_y = m_x \sum_{n=-\infty}^{\infty} g(n) \qquad (2.3\text{-}15)$$

og varians:

$$\sigma_y^2 = \sigma_x^2 \sum_{n=-\infty}^{\infty} g^2(n) \qquad (2.3\text{-}16)$$

Dersom g(n) er impulsresponsen fra der hvor avrunding finner sted, vil avrundingsstøyens varians på utgangen være gitt av σ_y^2 dersom σ_x^2 byttes ut med variansen til støyen grunnet selve avrundingen. Dersom det er flere bidrag til avrundingsstøy i systemet og disse er ukorrelerte, blir utgangsvariansen lik summen av variansene til hver enkelt kilde.

Avrundigsstøy i FIR-filtre med heltallskoeffisienter(fixed point).

Når *fixed point aritmetikk* benyttes og en avrunding utføres etter hver multiplikasjon, oppstår det samme type støy som ved kvantisering. Dersom det er N multiplikasjoner totalt, blir variansen til støy grunnet avrunding:

$$\sigma_0^2 = N \frac{2^{-2B}}{12} \qquad (2.3\text{-}17)$$

Dersom det er mange kilder til hvit uniform fordelt støy i et system, kan disse totalt etter sentralgrenseteoremet ses på som hvit Gaussisk støy.

2.4. Grunnleggende multirate teori

Innholdet i dette avsnittet er basert på boken ”Multirate digital signal processing”, av: ”Fliege”.

2.4.1. Multiratesystem

I et multiratesystem kan sampelfrekvensen til individuelle signaler økes(*interpolation*) eller minkes(*decimation*), før eller samtidig som disse signalene prosesseres.

2.4.2. Representasjon av diskrete signaler

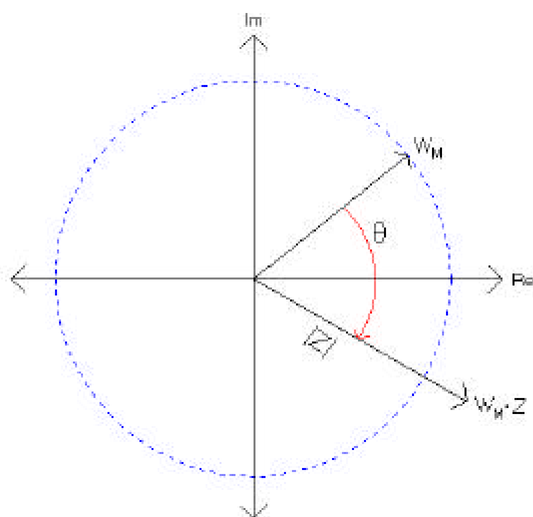
Det er tre vanlige måter å beskrive et diskrete signal på, disse er behandlet i etterfølgende avsnitt.

2.4.2.1. Diskret sampling

Diskrete signaler er ofte beskrevet ved hjelp av det komplekse tallet

$$W_M = e^{-j\frac{2\pi}{M}} = \sqrt[M]{1} \qquad (2.4\text{-}1)$$

Dette er en av de M forskjellige M-te røttene av 1. Tallet W_M ligger på enhetssirkelen i det komplekse plan, som vist på Figur 2.4.1.
Dersom W_M multipliseres med et tilfeldig komplekst tall Z, vil |Z| (lengden) forbli den samme(ettersom $|W_M|=1$). Argumentet(vinkelen) vil derimot endres(rotasjon med klokka), derfor kalles multiplikasjon med W_M for en *rotasjonsoperasjon.* Figur 2.4.1 viser et eksempel på dette.



Figur 2.4.1 Komplekse plan

Diskret sampling:
At et diskret signal x(n) samples, vil si at hver M-te verdi av det ”plukkes” ut mens de resterende sampler settes til 0. For å beskrive samplingen av et diskret signal benyttes den ”Diskrete samplings funksjonen”:

$$\omega_M(n) = \frac{1}{M}\sum_{k=0}^{M-1} W_M^{k\cdot n} = \begin{cases} 1, & n = m\cdot M, \quad m \in \mathrm{N} \\ 0, & ellers \end{cases} \qquad (2.4\text{-}2)$$

Dersom n=mM, hvor $m \in \mathrm{N}$, vil alle $W_M^{k\cdot n}$ fra k=0 til M-1 ha samme fase(n*2π, n=1,2,3…) en sum av M slike vil tilsvare verdien M(dividert på M gir verdien $\omega_M(n)=1$). For alle andre verdier av n, vil alle $W_M^{k\cdot n}$ være fordelt på en slik måte på enhetssirkelen, at deres vektorsum er 0(viserene opphever hverandre).
Dette kan illustreres med et eksempel:

<u>Eksempel 1</u>
Det foretas diskret sampling med M=3. Diskret samplingsfunksjon blir da:

$$\omega(n) = \frac{1}{3}\sum_{k=0}^{2} e^{-j\frac{2\pi k n}{3}}$$

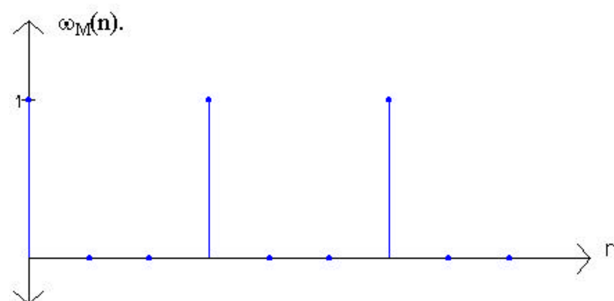
$$\omega(0) = \frac{1}{3}\sum_{k=0}^{2} e^{-j0} = \frac{1}{3}(1+1+1) = 1$$

$$\omega(1) = \frac{1}{3}\sum_{k=0}^{2} e^{-j\frac{2\pi k}{3}} = \frac{1}{3}(e^{-j0} + e^{-j\frac{2\pi}{3}} + e^{-j\frac{4\pi}{3}}) = \frac{1}{3}\cdot 0 = 0 \qquad likefordelt \qquad (2.4\text{-}3)$$

$$\omega(2) = \frac{1}{3}\sum_{k=0}^{2} e^{-j\frac{2\pi k\cdot 2}{3}} = \frac{1}{3}(e^{-j0} + e^{-j\frac{4\pi}{3}} + e^{-j\frac{8\pi}{3}}) = \frac{1}{3}\cdot 0 = 0 \qquad likefordelt$$

$$\omega(3) = \frac{1}{3}\sum_{k=0}^{2} e^{-j\frac{2\pi k 3}{3}} = \frac{1}{3}(e^{-j0} + e^{-j2\pi} + e^{-j4\pi}) = 1 \qquad osv.$$

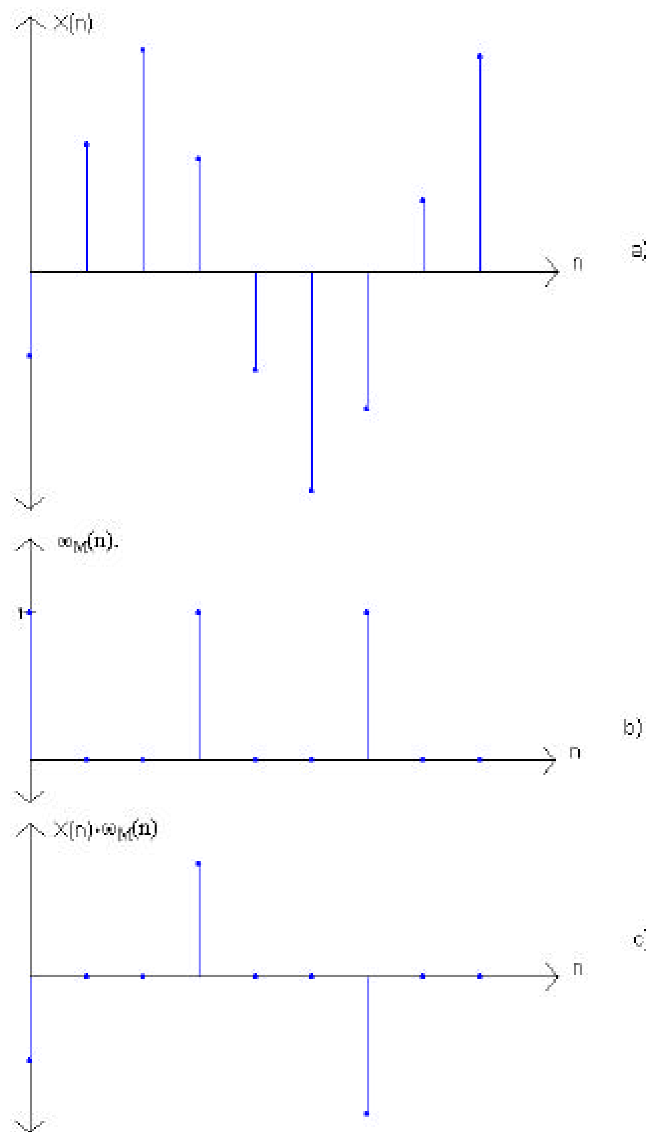
Et plot av denne funksjonen er vist i Figur 2.4.2.



Figur 2.4.2 Diskret samplings funksjon

Dersom x(n) multipliseres med $\omega_M(n)$, oppnås et diskret samplet signal x(n) $\omega_M(n)$.
Dette kan illustreres med et eksempel:

Eksempel 2.
Har signalet x(n) med n=9 sampler vist på Figur 2.4.3.a), dette signalet samples med avstand M=3, dette gir den diskrete samplingsfunksjonen $\omega_M(n)$, vist på Figur 2.4.3.b). Dersom x(n) multipliseres med $\omega_M(n)$, vil resultatet bli det diskret samplede signalet vist i Figur 2.4.3.c).



Figur 2.4.3 Diskret samplings prosess

Merk at original samplingfrekvens er uforandret. Forskjellen er at noen av de opprinnelige samplene er satt lik 0.

Det er også mulig å bruke en fase offset λ ved diskret sampling(tidsforskyvning av $\omega_M(n)$), dette gir følgende diskrete samplingsfunksjon:

$$w_M(n-\lambda) = \frac{1}{M}\sum_{k=0}^{M-1} W_M^{k\cdot(n-\lambda)} = \begin{cases} 1, & n = \lambda + m\cdot M, \quad m \in \mathrm{N} \\ 0, & ellers \end{cases} \qquad (2.4\text{-}4)$$

Formel (2.4-2) = formel(2.4-4) dersom λ=0.

2.4.2.2. Polyfaserepresentasjon

Har signalet x(n). Ut fra dette signalet kan det oppnås M forskjellige diskrete samplede signaler, hver med en forskjellig faseoffset λ. ⇒ at x(n) kan uttrykkes ved hjelp av disse M diskrete samplede signalene på følgende måte:

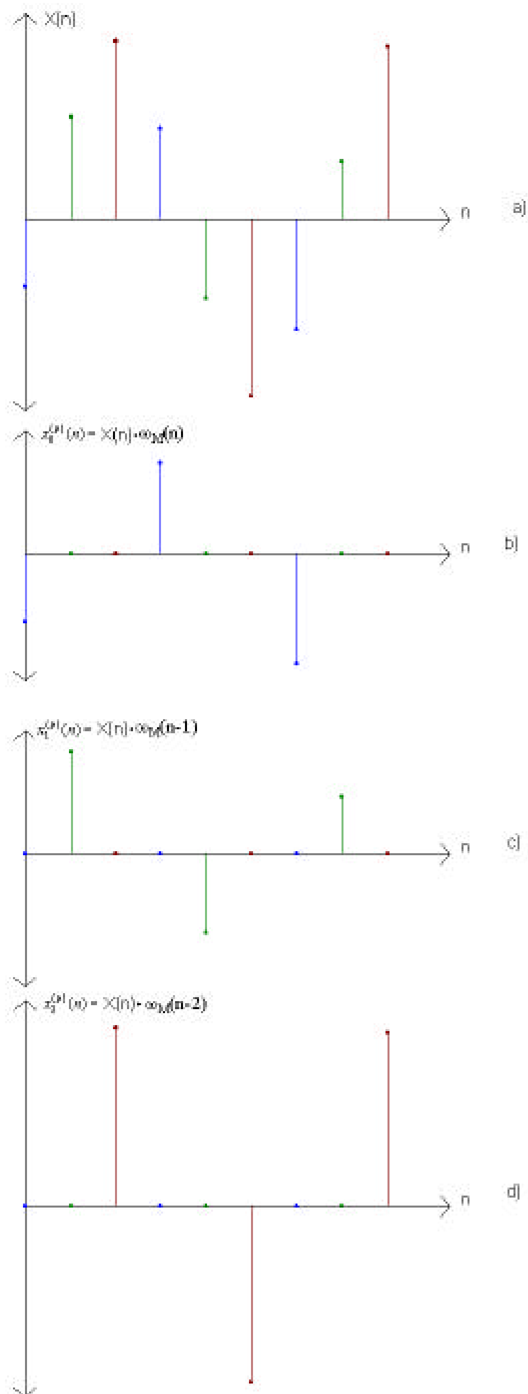
$$x(n) = \sum_{\lambda=0}^{M-1} x(n) w_M(n-\lambda) = \sum_{\lambda=0}^{M-1} x_\lambda^{(p)}(n) \qquad (2.4\text{-}5)^*$$

Denne måten å representere et signal på kalles *polyfaserepresentasjon.*

Dette kan illustreres med et eksempel:

<u>Eksempel 3.</u>
Har signalet x(n) bestående av n=9 sampler, vist i fig Figur 2.4.4.a). Dersom dette signalet samples med avstand M = 3, kan tre forskjellige diskre samplede signaler oppnås (for λ = 0, 1, 2). x(n) kan uttrykkes ved hjelp av de tre signalene vist i Figur 2.4.4.b)-d), ut fra formel (2.4-5).

[*] (p) står for standard polyfaserepresentasjon.



Figur 2.4.4 Polyfase representasjon av x(n)

Dersom en tar z-transformen(formel 2.1-10) av x(n) gitt i eksempel 3, fås følgende:

$$X(z) = \sum_{n=-\infty}^{\infty} x(n) \cdot z^{-n} \qquad (2.4\text{-}6)$$

$$X(z) = x(0) + x(1)\cdot z^{-1} + x(2)\cdot z^{-2} + \dots\dots\dots\dots\dots + x(8)\cdot z^{-8}$$
$$= x(0)\cdot z^{-0} + x(3)\cdot z^{-3} + x(6)\cdot z^{-6}$$
$$+ x(1)\cdot z^{-1} + x(4)\cdot z^{-4} + x(7)\cdot z^{-7}$$
$$+ x(2)\cdot z^{-2} + x(5)\cdot z^{-5} + x(8)\cdot z^{-8} \qquad (2.4\text{-}7)$$

$$= z^{-0}(x(0)\cdot z^{-0} + x(3)\cdot z^{-3} + x(6)\cdot z^{-6})$$
$$+ z^{-1}(x(1)\cdot z^{-0} + x(4)\cdot z^{-3} + x(7)\cdot z^{-6})$$
$$+ z^{-2}(x(2)\cdot z^{-0} + x(5)\cdot z^{-3} + x(8)\cdot z^{-6})$$

Ut fra dette kan det observeres at z-transformen for et generelt tall M kan skrives:

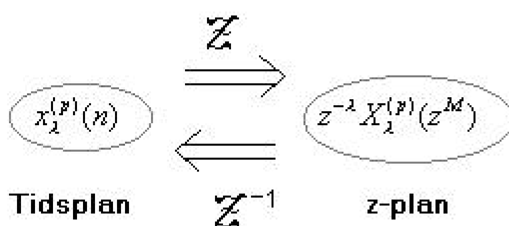
$$X(z) = \sum_{l=0}^{M-1} z^{-l} \sum_{m=-\infty}^{\infty} x(m\cdot M + l)\cdot z^{-m\cdot M} = \sum_{l=0}^{M-1} z^{-l} X_l^{(p)}(z^M) \qquad (2.4\text{-}8)$$

Hvor

$$X_l^{(p)}(z^M) = \sum_{m=-\infty}^{\infty} x(m\cdot M + l)\cdot z^{-m\cdot M} \qquad (2.4\text{-}9)$$

(2.4-8) kalles *polyfaserepresentasjonen* av z-transformen, (2.4-9) gir innholdet i parentesene i (2.4-7)(dersom M=3).

Sammenheng mellom tids- og frekvensplan må da være som vist i Figur 2.4.5.



Figur 2.4.5 Sammenhengen mellom polyfase representasjon i tids- og frekvens plan

2.4.2.3. Modulasjonsrepresentasjon

Modulasjon av z-transformen gjøres ved å multiplisere den uavhengige variabelen z med W_M^k.
Dette gir modulert z-transform:

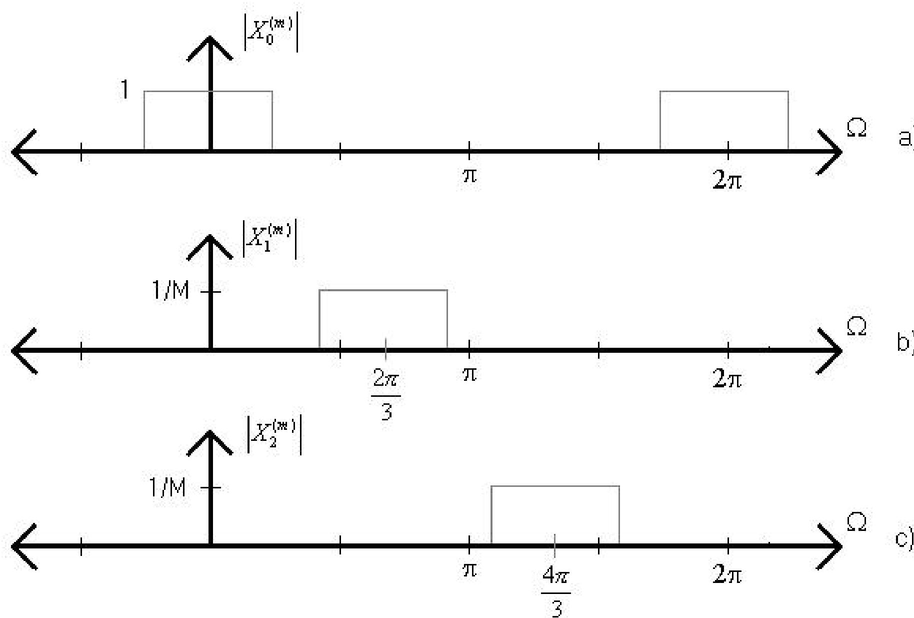
$$X_k^{(m)}(z) = X(z \cdot W_M^k), k = 1,2,3.........., M-1 \qquad (2.4\text{-}10)^*$$

Dersom en foretar substitusjonen $z \to e^{j\Omega}$ og tar DFT(formel 2.1-5) i betraktning, finner en ut at det å modulere z-transformen med en faktor $W_M^{\ k}$, tilsvarer å forskyve den normaliserte vinkelfrekvensen Ω i DFT med 2πk/M i positiv retning

$$X_k^{(m)}(e^{j\Omega}) = X(e^{j\Omega} \cdot e^{-j\frac{2\pi k}{M}}) = X(e^{(\Omega - j\frac{2\pi k}{M})}) \qquad (2.4\text{-}11)$$

Dette kan illustreres med et eksempel:

Eksempel 4.
La ”diskret samplingsfaktor” være M=3 og k=0,1,2 dette gir de tre mulige modulasjonskomponentene vist i Figur 2.4.6.a)-c).



Figur 2.4.6 Modulasjons komponenter

En viktig sammenheng er sammenhengen mellom polyfaserepresentasjon og modulasjonsrepresentasjon(utledning er utelatt i denne rapporten, men for interesserte henvises det til: ”Multirate Digital Signal Processing” av N.J. Fliege.).
Denne er gitt av følgende likhet:

$$z^{-1} X_1^{(p)}(z^M) = \frac{1}{M} \sum_{k=0}^{M-1} X_k^{(m)}(z) \cdot W_M^{1k} \qquad (2.4\text{-}12)$$

* (m) står for standard modulasjonsrepresentasjon.

Dette uttrykket viser at en polyfasekomponent for en gitt λ fremkommer ved å summere alle de ulike modulasjonskomponentene og deretter dividere med diskret samplingsfaktor M.

2.4.3. Endring av sampelfrekvens

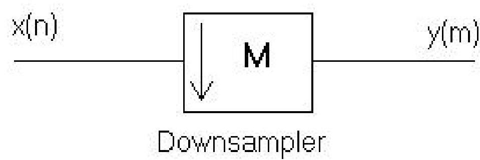
2.4.3.1. Reduksjon av sampelfrekvens(Decimation)

Reduksjon av sampelfrekvens er ofte ønskelig dersom original samplingsfrekvens er mye høyere enn det dobbelte av høyeste signalfrekvens i det samplede signalet.

Sampelfrekvensen til et diskret signal x(n) reduseres med en faktor M, ved å sample hver M-te verdi av det, dette gir det nedsamplede signalet y(m).

$$y(m) = x(m \cdot M), \quad m \in \mathrm{N} \qquad (2.4\text{-}13)$$

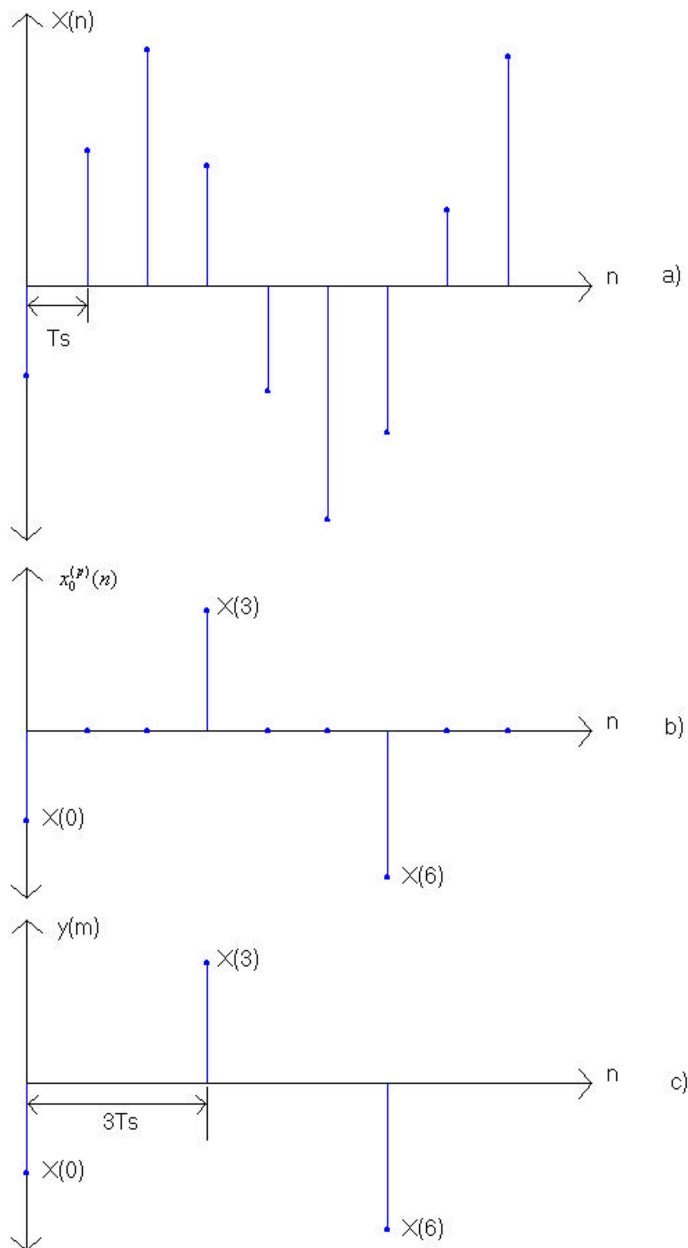
Dette uttrykkes skjematisk som vist i Figur 2.4.7:



Figur 2.4.7 Downsampler

Nedsamplingen av et signal kan beskrives ved bruk av polyfaserepresentasjonen. Dette gjøres som følger:

Figur 2.4.8 viser et eksempel på nedsampling med M=3.



Figur 2.4.8 Nedsampling med en faktor M = 3

Figur 2.4.8.a) viser signalet x(n) med en sampelavstand på Ts. Polyfasekomponenten $x_0^{(p)}(n)$ er vist i Figur 2.4.8.b), denne fremkom ved diskret sampling av x(n) med M=3. Det nedsamplede signalet y(m) vist på Figur 2.4.8.c) oppnås ved å utelate M-1 nuller mellom hvert sampel.
Denne beskrivelsen kan gis matematisk på følgende måte(for generell M):
z-transformen av det originale signalet er gitt av formel(2.4-6)
z-transformen av polyfasekomponenten $x_0^{(p)}(n)$ er gitt av formel (2.4-9) med λ=0(fase 0):

$$X_0^{(p)}(z^M) = \sum_{m=-\infty}^{\infty} x(m \cdot M) \cdot z^{-m \cdot M} \qquad (2.4\text{-}14)$$

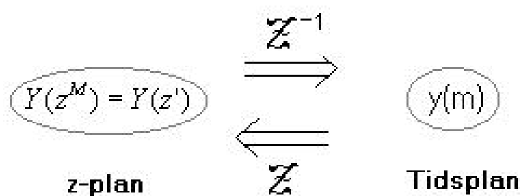
Dette kan omformes til(fra formel 2.4-13):

$$= \sum_{m=-\infty}^{\infty} y(m) \cdot (z^M)^{-m} = Y(z^M) \qquad (2.4\text{-}15)$$

Dersom z^M settes lik en ny variabel z' blir:

$$= \sum_{m=-\infty}^{\infty} y(m) \cdot (z')^{-m} = Y(z') \qquad (2.4\text{-}16)$$

Det å utelate M-1 0'er, kan beskrives ved hjelp av relasjonen i Figur 2.4.9.



Figur 2.4.9 Relasjon

Dette "klargjøres" i det påfølgende:
Dersom avstanden mellom hvert sampel i det originale signalet x(n) er T_s, kan variabelen z i X(z) som kjent skrives(fra formel 2.1-11):

$$z = e^{sT_s} \qquad (2.4\text{-}17)$$

hvor s er den uavhengige variabelen i Laplacetransformen.

La T' være avstanden mellom hvert sampel i det nedsamplede signalet y(m). Sammenhengen mellom original sampelavstand og redusert sampelavstand er:

$$T' = M \cdot T_s \qquad (2.4\text{-}18)$$

Det kan nå defineres en ny variabel :

$$z' = e^{sT'} \qquad (2.4\text{-}19)$$

Vha likning (2.4-17) kan en finne sammenhengen mellom z' og z:

$$z' = e^{sT'} = e^{s \cdot MT_s} = e^{(sT_s)M} = z^M \qquad (2.4\text{-}20)$$

Følgende kan nå sies:
I z-transformen som gir $Y(z^M)$ i formel 2.4-15, er summasjonen utført ved original samplingsrate. Derimot er det bare hver M'te verdi som er av betydning, ettersom alle andre verdier er 0. I z-transformen av Y(z') i formel 2.4-16, utføres i hovedsak samme summasjon, bortsett fra at bare sampler ≠ 0 tas hensyn til, disse har sampelavstand T'. ⇒Overgangen fra $X_0^{(p)}(z^M) = Y(z^M)$ til Y(z') i frekvensdomene vil tilsvarer overgangen fra $x_0^{(p)}(n)$ i Figur 2.4.8.b) til y(m) i Figur 2.4.8.c) i tidsdomene(jmf Figur 2.4.9).

Frekvensspekteret til et nedsamplet signal.
For å bestemme frekvensspekteret til et nedsamplet signal $Y(z^M)$, uttrykkes det ved hjelp av dets modulasjonskomponenter.

Ved å sette λ=0 i formel 2.4-12, fås følgende:

$$Y(z^M) = X_1^{(p)}(z^M) = \frac{1}{M}\sum_{k=0}^{M-1} X_k^{(m)}(z) \cdot W_M^k$$
$$= \frac{1}{M}\sum_{k=0}^{M-1} X(z \cdot W_M^k) \qquad (2.4\text{-}21)$$

Ved å utføre substitusjonen $z \rightarrow e^{j\Omega}$ blir:

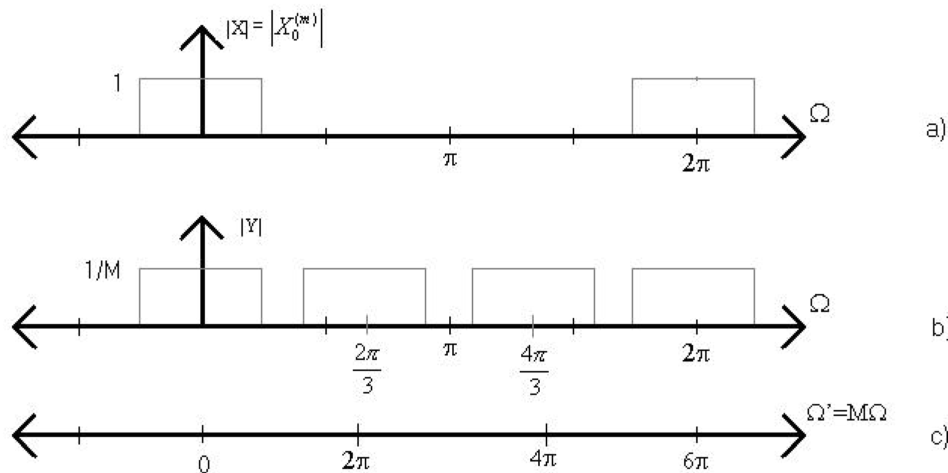
$$Y(e^{jM\Omega}) = \frac{1}{M}\sum_{k=0}^{M-1} X(e^{j\Omega} \cdot e^{-j\frac{2pk}{M}})$$
$$= \frac{1}{M}\sum_{k=0}^{M-1} X(e^{j\Omega - j\frac{2pk}{M}}) \qquad (2.4\text{-}22)$$

Absolutt verdien av dette uttrykket vil gi amplitudespekteret til y(m). Ut fra formel 2.4-22 kan en se at $Y(e^{jM\Omega})$ består av summen av alle modulasjonskomponentene til det originale signalet x(n)(dvs modulasjonskomponenter fra 0→M-1, skalert med 1/M i amplitude).
Dette kan illustreres med et eksempel:

Eksempel 5:
Det foretas diskret sampling med M=3 av signalet x(n). Spekteret til x(n) er vist i Figur 2.4.10.a). Det nedsamplede signalets frekvensspekter vil da ut fra formel 2.4-22 bli:

$$Y(e^{jM\Omega}) = \frac{1}{3}\sum_{k=0}^{2} X(e^{j\Omega - j\frac{2pk}{3}}) \qquad (2.4\text{-}23)$$

Amplitudepekteret til det nedsamplede signalet y(m) vil da bli som vist i Figur 2.4.10.b).



Figur 2.4.10 Amplitude spekter ved nedsampling

Figur 2.4.10.c) viser frekvensaksen til den nye variabelen, dvs digital vinkelfrekvens for 1/3 av original samplingsfrekvens.

Sammenhengen mellom de to variablene Ω og Ω' fremkommer ut fra formel 2.4-24:

$$e^{jM\Omega} = e^{jMwT} = e^{jwT'} = e^{j\Omega'} \qquad (\ 2.4\text{-}24\ )$$

Det kan observeres ut fra figur Figur 2.4.10 i eksemplet ovenfor at spekteret til y(m) er summen av alle modulasjonskomponentene til x(n).
En annen og meget viktig observasjon er at spektrene vil overlappe og derfor gi aliasing dersom de ikke er båndbreddebegrenset til Ω=π/3 før nedsamplingen.
Det generelle vil da være:

Et signal som skal nedsamples med en faktor M, må på forhånd være båndbreddebegrenset til min W=p/M, dersom aliasing skal unngås.

Nedsampling kan også gjøres med diskret sampling med faseoffset. Dette kan generelt uttrykkes vha modulasjonskomponentene på følgende måte:

$$z^{-l} X_l^{(p)}(z^M) = \frac{1}{M}\sum_{k=0}^{M-1} X(z \cdot W_M^k) W_M^{kl} \qquad (\ 2.4\text{-}25\ )$$

Ved å sette z= $e^{j\Omega}$ fremkommer spekteret, dette blir:

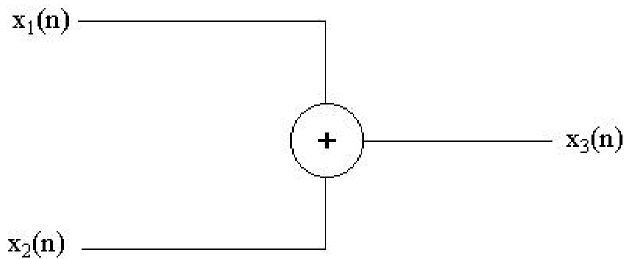
$$Y_l(e^{jM\Omega}) = z^{-l} X_l^{(p)} = \frac{1}{M}\sum_{k=0}^{M-1} X(e^{j\Omega - j\frac{2p \cdot k}{M}}) W_M^{kl} \qquad (\ 2.4\text{-}26\ )$$

Dette gir spekteret til et nedsamplet signal, hvor samplene er faseforskjøvet med en faktor λ.

Viktige identiteter - Noble identities.

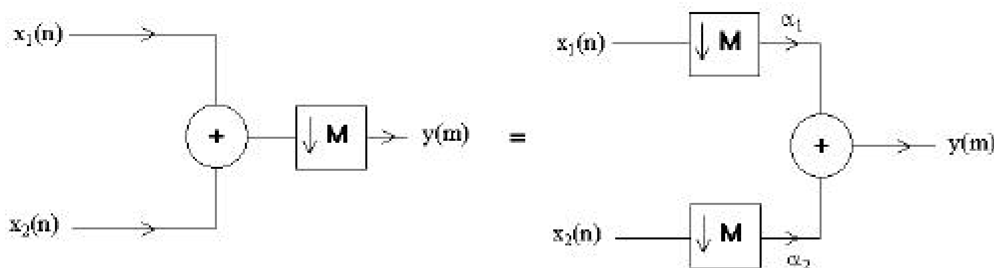
Det er hovedsakelig tre viktige identiteter i forbindelse med nedsampling, disse er:

Identitet 1:
Anta to signaler $x_1(n)$ og $x_2(n)$, som kommer i hver sin gren og adderes sammen til ett signal $x_3(n)$. Dette er vist på Figur 2.4.11.



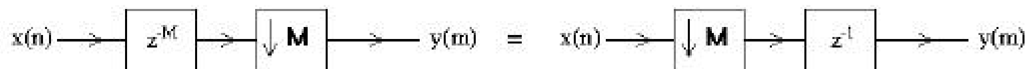
Figur 2.4.11 Addering av signaler

Dersom $x_3(n)$ skal nedsamples er det likegyldig om det gjøres før eller etter signalene adderes, ettersom skaleringen av signalene og addisjonen av dem er uavhengig av sampelfrekvensen. Dette gir *identitet 1*, vist på Figur 2.4.12.



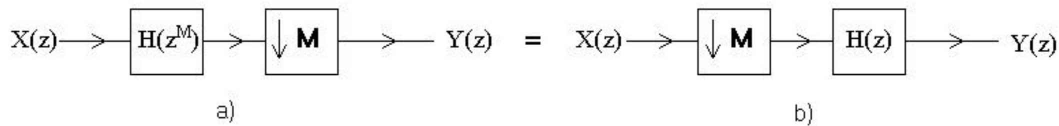
Figur 2.4.12 Identitet 1

Identitet 2:
Ettersom en forsinkelse av M sampel perioder før nedsampling tilsvarer en sampelperiode etter, fremkommer *identitet2*. Denne er vist på Figur 2.4.13.



Figur 2.4.13 Identitet 2

Identitet 3:
Identitet 3 er en mer generell versjon av *identitet 2*. Det tas her i betraktning et helt lineært system med transferfunksjon H(z). Denne identiteten er vist på Figur 2.4.14.



Figur 2.4.14 Identitet 3

Bevis for *identitet 3*:
Utgangen på nedsampleren på figur Figur 2.4.14.b) er gitt av formel 2.4-22

$$U(\Omega)=\frac{1}{M}\sum_{k=0}^{M-1}X(e^{j\Omega-j\frac{2\pi k}{M}})=\frac{1}{M}\sum_{k=0}^{M-1}X(e^{j\frac{\Omega'}{M}-j\frac{2\pi k}{M}})=\frac{1}{M}\sum_{k=0}^{M-1}X(e^{j\frac{\Omega'-2\pi k}{M}})=\frac{1}{M}\sum_{k=0}^{M-1}X(\frac{\Omega'-2\pi k}{M})$$

(2.4-27)

Utgangen Y(z) blir da:

$$Y(\Omega')=U(\Omega'\cdot H(\Omega')=\frac{1}{M}H(\Omega')\sum_{k=0}^{M-1}X(\frac{\Omega'-2\pi k}{M}) \qquad (\ 2.4\text{-}28\ )$$

På Figur 2.4.14.a), fås følgende på nedsamplerens inngang:

$$U(\Omega)=H(M\cdot\Omega)X(\Omega) \qquad (\ 2.4\text{-}29\ )$$

Utgangen fra nedsampleren blir da:

$$Y(\Omega')=\frac{1}{M}\sum_{m=0}^{M-1}H(\frac{(\Omega'-2\pi k)M}{M})X(\frac{\Omega'-2\pi k}{M})=\frac{1}{M}H(\Omega')\sum_{k=0}^{M-1}X(\frac{\Omega'-2\pi k}{M}) \qquad (\ 2.4\text{-}30\ )$$

Fordi H(Ω’-2πk)=H(Ω’).
Som en kan se er resultatet fra formel 2.4-28 og 2.4-29 like⇒at Figur 2.4.14.a) og b) er like.

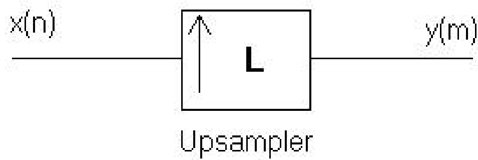
2.4.3.2. Øking av sampelfrekvens(Interpolation).

Når flere smalbåndede signaler skal kombineres til ett bredbåndet signal, må deres samplingsfrekvens først økes. Dette gjelder også når et smalbåndet signal skal få bedre oppløsning.

Samplingsraten til et diskret signal y(m) økes med en faktor L ved å ”sette inn” L-1 nuller med lik avstand, mellom hvert sampel. Det resulterende signalet er gitt ved:

$$u(n)=\begin{cases} y(\frac{n}{L}), & n=mL,\quad m\in \mathrm{N} \\ 0, & ellers \end{cases} \qquad (\ 2.4\text{-}31\ )$$

Figur 2.4.15 viser symbolet for en oppsampler.



Figur 2.4.15 Oppsampler

Denne rapporten omhandler ikke mer om oppsampling, ettersom dette ikke har noe med løsningen av prosjektoppgaven å gjøre. Det er likevel valgt å nevne oppsamling for de det kommer inn som en naturlig del under multirateteorien.
For mer teori om oppsampling henvises det til: "Multirate Digital Signal Processing" av N.J. Fliege.

2.4.4. Half Band Filtre(HBF).

Half Band Filtre opererer med bruk av halvparten så mange kalkulasjoner som konvensjonelle FIR-filtre.
HBF brukes derfor ofte i forbindelse med nedsampling i de tilfeller når M=2(halvering av sampelfrekvens). En filterbank hvor det er snakk om nedsampling med M>2 er en avart av HBF.

2.4.4.1. Antikausalt Half-Band Lav-Pass Filter(HBF-LP)

Har et antikausalt FIR filter med en "like" impulsrespons a(n). Denne vil ha transfer funksjon A(z).
A(z) kan som kjent beskrives vha polyfasekomponenter. Dersom A(z) splittes i to polyfasekomponenter, fås følgende:

$$A(z) = A_0^{(p)}(z^2) + z^{-1}A_1^{(p)}(z^2) \qquad (2.4\text{-}32)$$

Det som er spesielt for et HB-filter er at den første polyfasekomponenten i formel 2.4-32, er en konstant, vanligvis valgt til 0.5.

$$A_0^{(p)}(z^2) = 0.5 \qquad (2.4\text{-}33)$$

Formel 2.4.-32 og 2.4-33, beskriver et FIR filter med et odde antall koeffisienter N. For et HBF vil alle "partallskoeffisienter" være 0, bortsett fra ved n=0($A_0^{(p)}(z^2) = 0.5$).
Dette kan illustreres ved et eksempel.

Eksempel 6:
Har et HBF med N=11 koeffisienter. Transferfunksjonen til dette filteret er da gitt av formel 2.4-32, hvor:

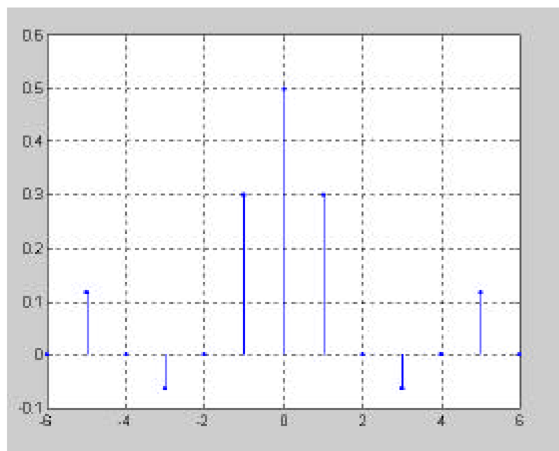
$$a_0^{(p)}(n) = \{0{,}0{,}0.5{,}0{,}0\} \xrightarrow{z-transform} A_0^{(p)}(z^2) = 0.5$$

Og, $a_1^{(p)}(n) = \{0.0117, -0.0625, 0.301, 0.301, -0.0625, 0.0117\}$(disse koeffisienter har framkommet vha Park's & Mc Clellan)

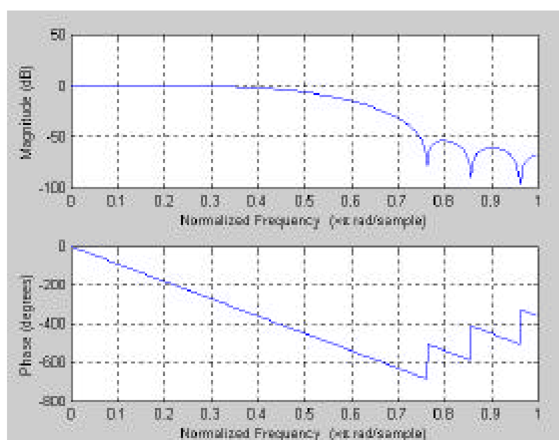
De to polyfase koeffisientene vil nå til sammen gi transferfunksjonen:

$$A(z) = 0.0117z^5 - 0.0625z^3 + 0.301z + 0.5 + 0.301z^{-1} - 0.0625z^{-3} + 0.0117z^{-5}$$

Dette gir impulsresponsen vist Figur 2.4.16, og frekvensresponsen på Figur 2.4.17.



Figur 2.4.16 Impulsrespons, antikausalt filter



Figur 2.4.17 Frekvens respons

Ut fra formel 2.4-32 og 2.4-33, fremkommer følgende:

$$A(z) + A(-z) = 1 \Rightarrow A(e^{j\Omega}) + A(e^{j(\Omega - p)}) = 1 \qquad (2.4\text{-}34)$$

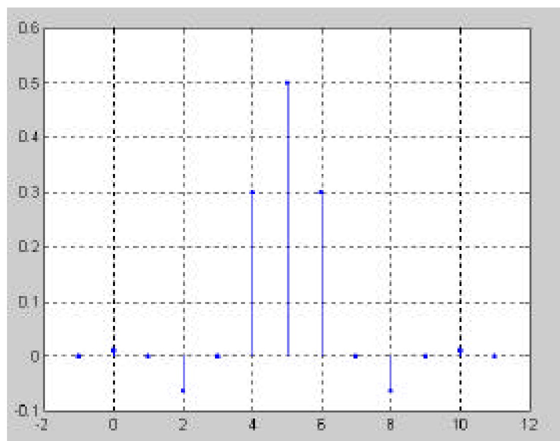
Dersom man tar utgangspunkt i transferfunksjonen i eksemplet ovenfor, kan dette enkelt vises:

$$0.0117z^{5} - 0.0625z^{3} + .. + 0.5 + .. + 0.0117z^{-5} + 0.0117(-z)^{5} - 0.0625(-z)^{3} + .. + 0.5 + .. + 0.0117(-z)^{-5}$$
$$= 0.5 + 0.5$$

Dvs at for en hver frekvens Ω, vil summen av den originale frekvensresponsen og denne responsen forskjøvet med π være 1⇒ at et HBF-LP er symmetrisk om π/2.
Filter lengden N for et slik filter er alltid et oddetall. Skal filterets orden økes må det økes med fire, dvs. at filterets orden blir 2, 6, 10, 14, osv.

2.4.4.2. Kausalt Half Band Lavpass Filter.

Et HBF kan gjøres kausalt ved å tidsforskyve impulsresponsen for den antikausale prototypen med (N-1)/2.
Dersom en forskyver impulsresponsen i Figur 2.4.16, (11-1)/2 = 5 enheter til høyre fremkommer impulsresponsen til et kausalt filter. Denne er vist i Figur 2.4.18.
Amplituderesponsen vil forbli den samme.



Figur 2.4.18 Kausal impulsrespons

Tidsforskyvningen vil transformere polyfasekomponentene i formel 2.4-32 til:

$$H(z) = z^{-\frac{N-1}{2}} A(z) = z^{-\frac{N-1}{2}} A_0^{(p)}(z^2) + z^{-\frac{N-1}{2}} \cdot z^{-1} A_1^{(p)}(z^2) \qquad (2.4\text{-}35)$$

Ettersom antall koeffisienter for et HBF er N=4i-1 hvor i∈N, som er et oddetall, vil (N-1)/2 alltid være et oddetall. Polyfasekomponentene vil derfor "stokkes om" i overgangen fra et antikausalt til et kausalt filter. Dette fremkommer klart fra Figur 2.4.18.
Dersom:

$$z^{-\frac{N-1}{2}} A_0^{(p)}(z^2) = z^{-1} H_1(z^2) \qquad (2.4\text{-}36)$$

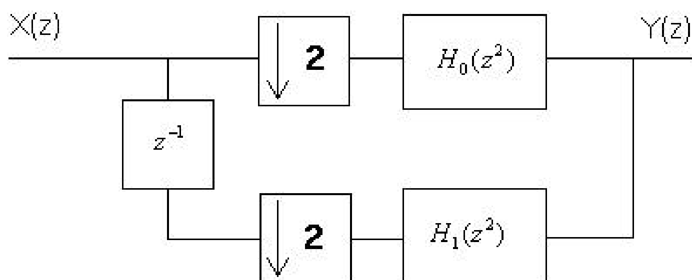
,og $$z^{-\frac{N-1}{2}} \cdot z^{-1} A_1^{(p)}(z^2) = H_0(z^2) \qquad (2.4\text{-}37)$$

kan transferfunksjonen for et kausalt filter skrives:

$$H(z) = H_0(z^2) + z^{-1} H_1(z^2) \qquad (2.4\text{-}38)$$

Hvor $H_1(z^2)$ blir en konstant, vanligvis valgt til 0.5.

Figur 2.4.19 viser hvordan et kausalt HBF kan realiseres. Subfilter $H_0(z^2)$ klokkes med en sampelfrekvens lik $f_s/2$.
$H_1(z^2)$ er kun en forsinkelse som også klokkes med $f_s/2$, denne ligger 1 sampel etter $H_0(z^2)$.



Figur 2.4.19 Blokk diagram for et half band filter

Dersom andre responser enn LP-responser ønskes, kan dette gjøres ved tradisjonelle metoder kjent fra signalteorien, eller vha en modulasjon av transferfunksjonen. En modulasjon gjøres ved å multiplisere z i H(z) med $e^{-j\Omega}$, hvor Ω er ønsket digital vinkelfrekvens som responsen ønskes flyttet til.$\Rightarrow H(z\, e^{-j\Omega})$.

3. Apparatur – utstyr

En del av dette prosjektet gikk ut på å teste ut software Quartus fra Altera, denne hadde store krav til PC utstyret.

Følgende PC utstyr ble benyttet:

- Pentium II PC, 733MHz.
- 40 Gbytes harddisk plass.
- 512 Mbytes til 1 Gbytes RAM.
- Operativ system: Windows 98.

Versjonen av Quartus som ble testet, QuartusII versjon 1.0. Denne software pakken tilhører Høgskolen i Gjøvik, og ble lånt ut til dette prosjektet.

Softwaren Max+plussII fra Altera ble også benyttet til timing simulering. Den versjonen som ble brukt er en student versjon, som fulgte med boken ” Digital systems design and protoyping using field programmable logic - Zoran Salcic & Asim Smailagic.

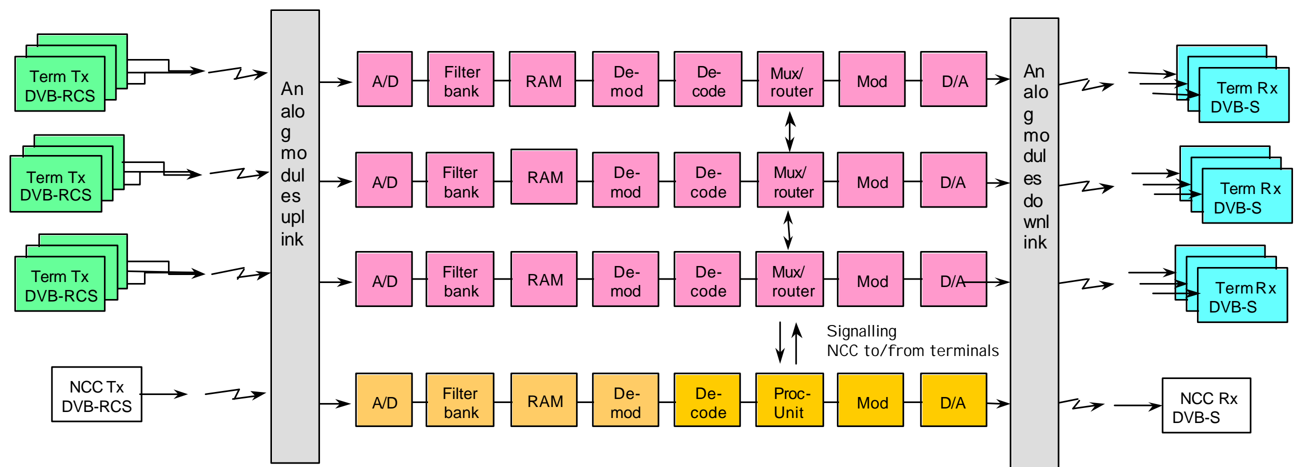
Det ble i tilegg brukt veldig mye Matlab under dette prosjektet, da dette er et veldig godt matematisk verktøy. Versjonen som ble brukt er Matlab versjon 5.3, og er lisensiert Høgskolen i Gjøvik.

4. Utførelsen

Avsnitt 4.1 tom. 4.3 er i store trekk basert på teorien i de to kompendiene ”Nordic onboard DVB processor project” tilsendt av Alcatel Space Norway, og ”Design of a configurable 4-64 channel digital frequency demultiplexer for an onboard DVB-RCS/DVB-S processor” av ”Bjørn Roger Andersen”.

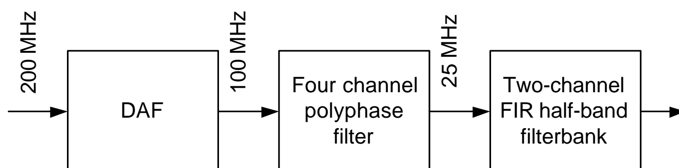
4.1. Teknisk oppgave beskrivelse

Alcatel Space Norway (ASN) i Horten ønsker å utvikle en DVB prosessor i samarbeid med flere andre bedrifter. For å realisere en slik DVB prosessor er det behov for stor kunnskap innen digital signalbehandling, som blant annet omhandler digital filtrering.



Figur 4.1.1 DVB prosessor (hentet fra rapporten: Nordic onboard DVB processor project, phaseII)

Figur 4.1.1 viser DVB prosessorens system arkitektur. Filterbanken på figuren foretar en reell til kompleks omforming av signalet, og separerer de individuelle bærebølgene i DVB-RCS signalet. Dette signalet inneholder video signaler i form av MPEG2.
Denne filterbanken kan deles i tre blokker som vist på Figur 4.1.2.



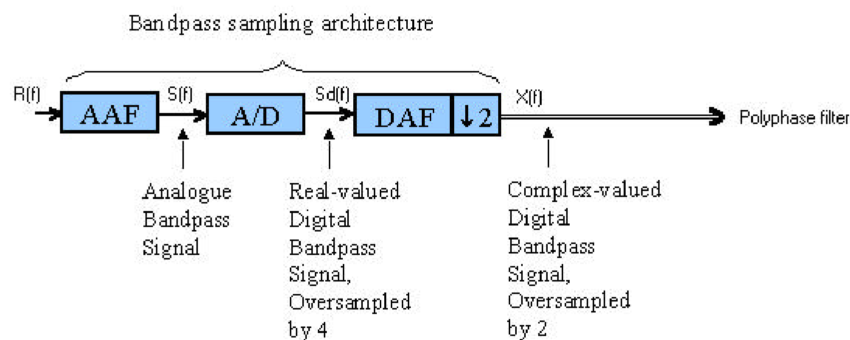
Figur 4.1.2 Filterbank (hentet fra rapporten: Nordic onboard DVB processor project, phaseII)

Den første blokken er et ”Digitalt Antilaliasing Filter”(videre omtalt som DAF), og det er dette filteret denne prosjektoppgaven i hovedsak omhandler. Inn på DAF kommer det et reelt digitalt båndpass signal som er oversamplet med 4, og ut et komplekst digitalt båndpass signal oversamplet med 2.

Den andre blokken er et ”Fire kanal polyfase filter”. Dette filteret splitter det innkommende signalets bånd i fire like store deler.
Den tredje og siste blokken er en ”To kanal FIR half-band filterbank”. Denne filterbanken har i oppgave å splitte hver av de fire båndene ut fra ”Fire kanal polyfase filter” i opptil 16 nye bånd. Dette separerer frekvensbåndet totalt i 64 kanaler.
Frekvensene er som gitt på Figur 4.1.2.

4.2. Innledning til DAF.

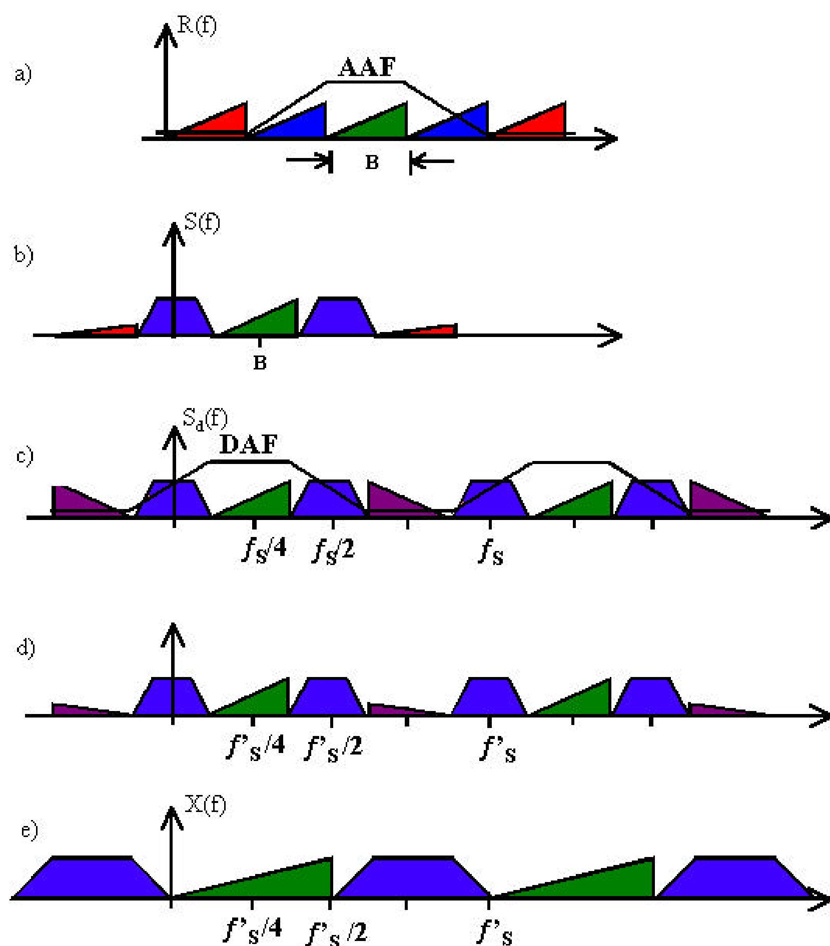
Foran A/D konverteren må det alltid sitte et ”Analogt Antialiasing Filter”(videre omtalt som AAF) for å båndbredde begrense det analoge innsignalet til maksimalt halve samplings frekvensen. For tilfredsstillende filtrering bør vanligvis steilheten til transisjonsbåndet i dette filteret være stor. Desto steilere flanker filteret har, desto høyere blir kostnadene med tanke på bruk av hardware. Det er ønskelig å finne en løsning hvor resultatet blir tilfredsstillende selv om kravet til filtersteilhet er lavere. Dette vil redusere kostnadene nevnt ovenfor.
En måte å få til dette på er å sette inn et digitalt antialiasing filter i tillegg til det analoge, som vist på Figur 4.2.1.



Figur 4.2.1 Bandpass sampling arkitektur (hentet fra rapporten: Nordic onboard DVB processor project, phaseII)

Figur 4.2.2 viser signalbehandlingen gjenomm de tre blokkene i Figur 4.2.1. Inngangsignalet R(f) skal være et reell analogt signal som inneholder mange bærebølger. AAF plukker ut det ønskede frekvensbåndet(grønn) med båndbredde B vist på Figur 4.2.2.a). Ut fra AAF kommer det filtrerte signalet S(f) som er nedkonvertert til en senterfrekvens B. Dette signalet består i tillegg til det ønskede signalet av “leakage”-komponeter fra AAF filterets transisjons(blå) og stoppbånd(rød) vist på Figur 4.2.2.b). Deretter samples signalet med $f_s = 4*B$ og A/D konverteres for og sendes inn på DAF filteret som $S_d(f)$. Ved sampling vil frekvensspekteret gjenta seg for multiplum av samplingsfrekvensen, dette medfører at det ønskede signalet(grønn) med senter i $f_s/4$, også vil opptre med en senterfrekvens på $3*f_s/4$(lilla). Denne komponenten er meget uheldig når signalet skal nedsamples, og må dempes tilstrekklig. Dersom DAF filteret konstrueres som et båndpass filter med en senterfrekvens på $f_s/4$, og med pass- , transisjon- og stopp bånd lik B, vil den uheldige komponenten rundt $3*f_s/4$ dempes tilstrekelig, slik at den ikke overlapper med nyttesignalet når nedsampling foretas. Komponentene i transisjons båndet(blå) vil overlappe hverandre, og vil derfor ikke gjøre noen skade på nyttesignalet. Frekvensspekteret etter DAF filteret blir som vist i Figur

4.2.2.d). Deretter nedsamples signalet med en faktor M=2, og frekvensspekteret blir som vist på Figur 4.2.2.e). Denne nedsamplinge skal gjøres i DAF filteret.
Det kan observeres at både AAF og DAF kan realiseres med forholdsvis lave krav til steilhet i transisjonsbåndet.



Figur 4.2.2 Signal behandlingen (hentet fra rapporten: Nordic onboard DVB processor project, phaseII)

Det er gitt følgende spesifikasjoner for DAF filteret:

- Stoppbåndsdemping på 60dB.
- Innsignal fra A/D konverteren er representert med 8 bit.
- Utgangen skal representeres med 12 bit. Dette har fremkommet ut fra beregninger som en nedre grense med tanke på "overflow".

4.3. DAF filter

Ettersom det er snakk om video signaler(MPEG2) vil det være strenge krav til lik forsinkelse av alle aktuelle frekvens komponenter ⇒ konstant gruppeforsinkele i passbånd ⇒ lineær fase i passbånd. Ettersom det er krav til lineær fase, utelukkes en realisering ved hjelp av IIR filter. DAF filteret må derfor være et FIR filter.

Samplings frekvensen f_s er bestemt til å være 200MHz, dette er en forholdsvis høy frekvens og bør reduseres så tidlig som mulig i systemet. DAF filteret bør derfor realiseres slik at det i tillegg til å ivareta oppgavene drøftet i avsnitt 4.1 og 4.2, reduserer samplingsfrekvensen til 100MHz. DAF filteret må da derfor realiseres som et multirate filter.

Som nevnt i avsnitt 2.4 om multirate teori, er det gunstig å bruke "halfband filter" når samplingsfrekvensen skal halveres siden annenhver koeffisient er lik 0 bortsett fra senterkoeffisienten.

En kan se ut fra Figur 4.2.2.c) at DAF responsen ikke er symetrisk rundt $f s/2 \Rightarrow$ filteret kan ikke ha kompleks konjugerte nullpunkter $\Rightarrow$ filteret må være komplekst. Dette gås det nærmere inn på senere i avsnitt 4.3.6.

4.3.1. Matematisk utledning av transferfunksjonen

Første trinn i realiseringen vil være å lage en lavpass modell for filteret med tilstrekkelig stoppbåndsdemping. Denne transformeres deretter over til den ønskede båndpass responsen. Som kjent fra avsnitt 2.4 om multirate teori, kan et kausalt filterets transferfunksjon uttrykkes ved hjelp av to polyfase komponenter på følgende måte:

$$H(z) = H_0(z^2) + z^{-1} H_1(Z^2) \qquad (4.3\text{-}1)$$

Hver polyfase komponentene kan ses på som et sub-filter med sine egne koeffisienter. Introduserer to koeffisient vektorer $h_0(n)$ og $h_1(n)$.
Vektoren $h_0(n)$ har lengde $(Q+1)/2$, som er et like antall, denne inneholder koeffisientene til $H_0(z^2)$. Vektoren $h_1(n)$ har lengde $(Q-1)/2$, som er et odde antall, denne inneholder koeffisientene til $H_1(z^2)$. Uttrykket for det totale filterets transferfunksjon kan da uttrykkes ved disse vektorene og polyfase koeffisientene på følgende måte:

$$\begin{aligned} H(z) &= \sum_{i=0}^{\frac{Q-1}{2}} h_0(i) z^{-2i} + z^{-1} \sum_{i=0}^{\frac{Q-1}{2}-1} h_1(i) z^{-2i} \\ &= h_0(0) + h_1(0) z^{-1} + h_0(1) z^{-2} + \ldots\ldots + h_1\left(\frac{Q-1}{2} - 1\right) z^{-(Q-3)} + h_0\left(\frac{Q-1}{2}\right) z^{-(Q-1)} \end{aligned} \qquad (4.3\text{-}2)$$

Ettersom det skal tilstrebes en halfband respons vil $H_1(i) = \begin{cases} 0.5, & for\,\frac{Q+1}{4} - 1 \\ 0, & ellers \end{cases}$ (4.3-3)

,i tillegg er: $h_0(0) = h_0((Q-1)/2)$, $h_0(1) = h_0((Q-1)/2-1)$, osv. $\Rightarrow$ symmetrisk koeffisient vektor.

Dette medfører at H(z) kan skrives som følgende uttrykk:

$$H(z)=\sum_{i=0}^{\frac{Q+1}{4}-1} h_0(i)\left(z^{-2i}+z^{-2\left(\frac{Q+1}{2}-1-i\right)}\right)+z^{-1}0.5z^{-2\left(\frac{Q+1}{4}-1\right)} \qquad (\text{ 4.3-4 })$$

Dette er transferfunksjonen til et reelt halfband lavpass filter. For å oppnå ønsket båndpass respons forflyttes denne opp til $f_s/4 \Rightarrow$ modulering av transferfunksjonen til en digital vinkelfrekvens Ω lik π/2 ⇒ multiplikasjon av z i lavpass modellen med $e^{-j\pi/2}$:

$$H_{DAF}(z)=H\left(e^{-j\frac{\pi}{2}}\cdot z\right)$$

$$=\sum_{i=0}^{\frac{Q+1}{4}-1} h_0(i)\left(\left(e^{-j\frac{\pi}{2}}\cdot z\right)^{-2i}+\left(e^{-j\frac{\pi}{2}}\cdot z\right)^{-2\left(\frac{Q+1}{2}-1-i\right)}\right)+\left(e^{-j\frac{\pi}{2}}\cdot z\right)^{-1}0.5\left(e^{-j\frac{\pi}{2}}\cdot z\right)^{-2\left(\frac{Q+1}{2}-1\right)} \qquad (\text{ 4.3-5 })$$

$$=\sum_{i=0}^{\frac{Q+1}{4}-1} h_0(i)\left(e^{+j\pi i}\cdot z^{-2i}+e^{+j\pi\left(\frac{Q+1}{2}-1-i\right)}\cdot z^{-2\left(\frac{Q+1}{2}-1-i\right)}\right)+e^{+j\frac{\pi}{2}}\cdot z^{-1}\cdot 0.5\cdot e^{+j\left(\frac{Q+1}{4}-1\right)}\cdot z^{-2\left(\frac{Q+1}{2}-1\right)}$$

Ettersom $e^{j\pi i},\ i\in \mathrm{N} \longrightarrow e^{j\pi i}=\cos(\pi i)=(-1)^i$ (4.3-6)

, og $e^{j\pi\left(\frac{Q+1}{2}-1-i\right)}=\cos\left(\pi\left(\frac{Q+1}{2}-1-i\right)\right)=(-1)^{\left(\frac{Q+1}{2}-1-i\right)}$ (4.3-7)

, og $e^{j\frac{\pi}{2}}=j\sin\left(\frac{\pi}{2}\right)=j$ (4.3-8)

, blir formel 4.3-5:

$$H_{DAF}(z)=\sum_{i=0}^{\frac{Q+1}{4}-1} h_0(i)\left((-1)^i\cdot z^{-2i}+(-1)^{\left(\frac{Q+1}{2}-1-i\right)}\cdot z^{-2\left(\frac{Q+1}{2}-1-i\right)}\right)+j\cdot z^{-1}\cdot 0.5\cdot(-1)^{\left(\frac{Q+1}{4}-1\right)}\cdot z^{-2\left(\frac{Q+1}{4}-1\right)}$$

(4.3-9)

Formel 4.3-9 er transferfunksjonen til et halfband båndpass filter med senterfrekvens lik $f/4$.

Filteret har nå gått fra å være et reelt lavpass filter, til å bli et komplekst bånd pass filter.

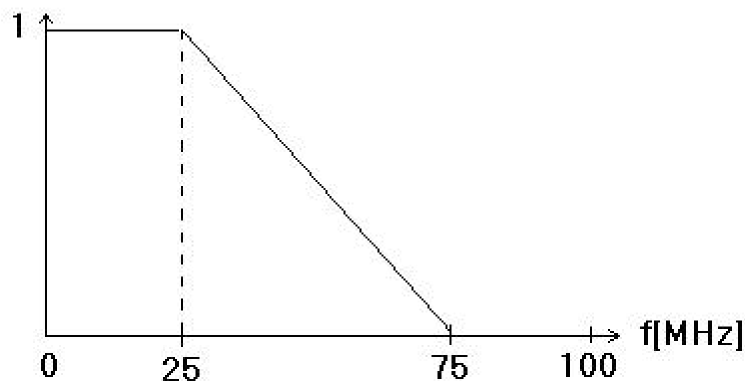
4.3.2. Bestemmelse av filter lengde

Filter lengden Q må nå bestemmes slik at kravet om stoppbåndsdemping på 60dB oppfylles. Dette gjøres enklest ved å se på lavpass modellens stoppbånds demping. For å få frem responsen til lavpassmodelen må koeffisientene være kjent. Dvs. at det må prøves forskjellige

filterlengder, finne koeffisientene ved disse lengdene, og plotte responsene. Koeffisientene finnes på en mest mulig ideell måte ved å bruke Parks & McClellan i Matlab. Dimensjonene til lavpass responsen må først bestemmes. Som kjent fra avsnitt 4.2 innledningen til DAF, skal pass- transisjons- og stopp båndene være like store når båndpass responsen er sentrert rundt $\pi/2$. Når samplingsfrekvensen er 200MHz, må pass- transisjons- og stopp båndene være 50MHz $\Rightarrow$ lavpass responsens dimensjoner må derfor være:

Passbånd: 0-25MHz
Transisjonsbånd: 25MHz-75MHz
Stoppbånd: 75MHz-100MHz

Dette kan illustreres ved hjelp av Figur 4.3.1.



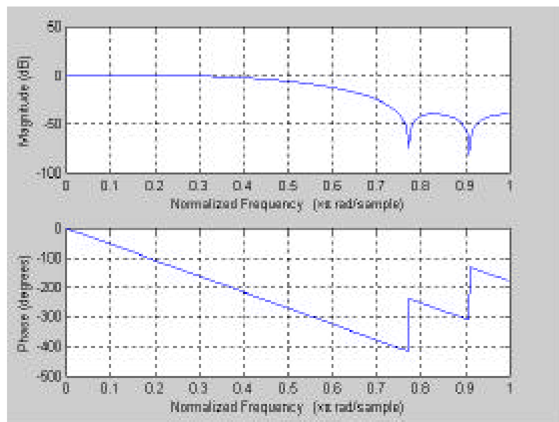
Figur 4.3.1 Lavpass modell

Matlab koden for å finne koeffisientene og responsen blir som vist nedenfor hvor Q sendes med som parameter. Som kjent fra ”multirate teorien” må Q være et oddetall, og skal filterets orden økes, må den økes med 4 hver gang $\Rightarrow$ 2,6,10,14,18,22 osv.

```
f=[0 25/100 75/100 1]; %Vektor som inneholder overganger mellom
                       de
                       %forskjellige båndene normalisert til fs/2

m=[1 1 0 0];           %Vektor som bestemmer nivået ved de
                       forskjellige
                       %overgangene
h=remez(Q-1,f,m);      %Finner koeffisientene ved hjelp av Remez
                       %algoritmen
freqz(h);              %Plotter filterets frekvens respons
```

Det ble prøvd med Q lik 7, 11 og 15 som vist i etterfølgende figurer. Det er her viktig å tenke over at Matlab “default” bruker 64 bit ved bergninger. I praksis skal koeffisientene rundes av til så få bit som mulig, dette er snakk om bitantall langt færre enn 64. Det kan derfor ikke ut fra etterfølgende figurer avgjøres hvilken filterlengde som er den beste, siden disse responsene er tilnærmet ideelle.

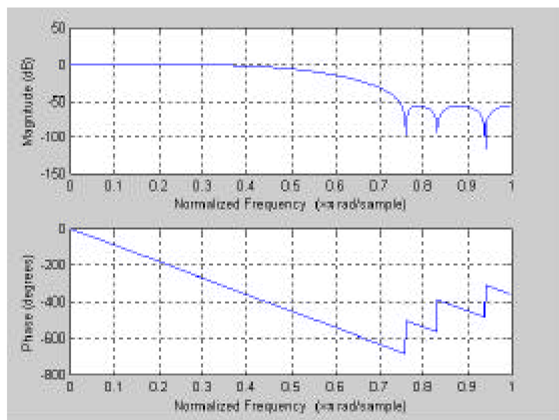
Figur 4.3.2 viser responsen med 7 koeffisienter.



Figur 4.3.2 Lavpass med 7 koeffisienter

Ut fra Figur 4.3.2 kommer det tydlig fram at stoppbånds dempingen er allt for lav i forhold til kravet. Denne løsningen er derfor ikke aktuell.

Figur 4.3.3 viser responsen med 11 koeffisienter.



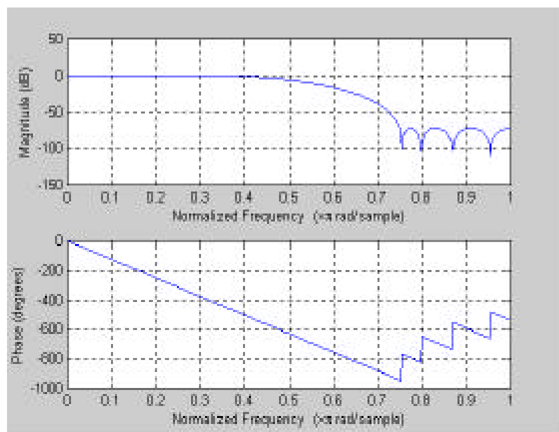
Figur 4.3.3 Lavpass med 11 koeffisienter

Ut fra Figur 4.3.3 kan man se at stoppbåndsdempingen ligger på ca. 55dB, dette er noe lavere enn kravet. Man bør muligens vurdere å øke til 15 koeffisienter. De 11 koeffisientene er som følger(disse vil det bli vist til senere i rapporten):

```
ans =

  Columns 1 through 7

    0.0130    0.0000   -0.0639   -0.0000    0.3016    0.5000    0.3016

  Columns 8 through 11

   -0.0000   -0.0639    0.0000    0.0130
```

»

Figur 4.3.4 viser responsen med 15 koeffisienter.



Figur 4.3.4 Lavpass med 15 koeffisienter

Ut fra Figur 4.3.4 kan man se at stoppbåndsdempingen ligger på ca. 72dB, dette ligger godt innenfor kravet. De 15 koeffisientene er som følger:

ans =

Columns 1 through 7

-0.0037 0.0000 0.0206 -0.0000 -0.0723 0.0000 0.3054

Columns 8 through 14

0.5000 0.3054 0.0000 -0.0723 -0.0000 0.0206 0.0000

Column 15

-0.0037

»

Det vil under simulering i Matlab naturlig nok føre til en enda bedre respons å øke til mer enn 15 koeffisienter, men dette vil være ved bruk av 64 bit. Det er ingen grunn til å velge et større antall koeffisienter enn det som er nødvendig, og det vil derfor videre i denne oppgaven være naturlig å velge 11 eller 15 koeffisienter. Med tanke på senere implementering og bruk av logiske celler, kan gevinsten ved å bruke færre koeffisienter være større enn gevinsten ved enn vedig høy stopp båndsdemping. I tillegg må det tas i betraktning at koeffisientene sannsynligvis må representeres med flere antall bit ved høyere filter orden. Det er derfor for tidlig å allerede nå avgjøre hvilken løsning som er best egnet.

4.3.3. Manuell utregning av filter koeffisienter

Koeffisientene i foregående avsnitt 4.3.2 ble funnet ved hjelp av Parks & McClellan, denne metoden er veldig matematisk avansert og ligger utenfor denne prosjektoppgaven å forklare. For å vise hvordan man faktisk kan komme fram til disse koeffisientene ved hjelp av manuell regning, følger det her en utregningen ved hjelp av Fouriermetoden.

Ut fra Fourirermetoden tas det utgangspunkt i en ”brickwall” respons. Som kjent fra grunnleggende teori(formel 2.1-15) vil impulsresponsen være gitt av følgende integral:

$$h(n)=\frac{1}{2\pi}\int_{-\Omega_c}^{\Omega_c} e^{j\Omega n}d\Omega=\frac{1}{2\pi}\cdot\frac{1}{j\Omega}\left[e^{j\Omega n}\right]_{-\Omega_C}^{\Omega_C}=\frac{\Omega_C}{\pi}\cdot\frac{\sin(n\Omega_C)}{n\Omega_C}=\frac{\Omega_C}{\pi}\cdot\sin c(n\Omega_C) \qquad (4.3\text{-}10)$$

Dette er impulsresponsen for et antikausalt filter.
For å få et kausalt filter må man som kjent tidsforskyve impulsresponsen med (Q-1)/2, i tillegg til dette er $\Omega_C = \pi/2$ for et halfband lavpassfilter(kjent fra avsnitt 2.4).
Følgende uttrykk oppnås:

$$h(n)=\frac{1}{2}\cdot\sin c\left(\left(n-\frac{Q-1}{2}\right)\frac{\pi}{2}\right) \qquad (4.3\text{-}11)$$

Velger å regne ut koeffisientene for et filter med lengde Q = 11, dette gir:

$$h(n)=\frac{1}{2}\cdot\frac{\sin\left((n-5)\frac{\pi}{2}\right)}{(n-5)\frac{\pi}{2}} \qquad (4.3\text{-}12)$$

Dette gir da:

$h(0)=h(10)=0.06366$
$h(2)=h(8)=-0.1061$
$h(4)=h(6)=0.3183$
$h(1)=h(9)=0$
$h(3)=h(7)=0$
$h(5)=0$, men settes som kjent fra teorien om half band filter lik 0,5.

Dette gir følgende koeffisient vektor: [0.06366 0 –0.1061 0 0.3183 0.5 0.3183 0 –0.1061 0 0.06366].
En kan se at denne koeffisient vektoren ikke stemmer helt overens med koeffisientene funnet med Parks & McClellan i avsnitt 4.3.2. Grunnen til dette er at Parks & Mcclellan gir koeffisient verdier som er mest mulig optimale med tanke på minst mulig feil mellom den ønskede ideelle responsen og den egentlige responsen, representert med et gitt endelig antall koeffisienter. Dersom man bruker et Kaiser vindu, kan det vises at koeffisient vektoren får

tilnærmet samme verdi som ved Parks & McClellan dersom den multipliseres med et Kaiser vindu med β=3,2. Teorien om Kaiser vindu β-verdier er beskrevet i avsnitt 2.1.5. Bruker funksjonen for Kaiser vindu i Matlab til å regne ut dette ved hjelp av følgende kode:

```
h=[0.06366 0 -0.1061 0 0.3183 0.5 0.3183 0 -0.1061 0 0.06366];
hw=h.*kaiser(11,3.2)';
```

Dette gir følgende koeffisient vektor: : [0.0111 0 –0.0636 0 0.3018 0.5 0.3018 0 –0.0636 0 0.0111], denne koeffisient vektoren stemmer bra overens med koeffisientene funnet med Parks & McClellan metoden i avsnitt 4.3.2.

4.3.4. Overføring til båndpass respons

For å finne den ønskede båndpass responsen for en filterlengde lik for eksempel 11 tas det utgangspunkt i formel 4.3-9. Ettersom $h_0(i)$ er koeffisient vektoren til det ene sub filteret, substitueres $h_0(i)$ med h(2*i) for å frambringe koeffisientene til båndpass filteret.

$$
\begin{aligned}
H_{DAF}(z) &= \sum_{i=0}^{2} h(2i)\left((-1)^i \cdot z^{-2i} + (-1)^{(5-i)} \cdot z^{-2(5-i)}\right) + j \cdot z^{-1} \cdot 0.5 \cdot (-1)^2 \cdot z^{-4} \\
&= h(0)\left(1 - z^{-10}\right) + h(2)\left(-z^{-2} + z^{-8}\right) + h(4)\left(z^{-4} + z^{-6}\right) + j \cdot 0.5 \cdot z^{-5} \\
&= h(0) - h(2)z^{-2} + h(4)z^{-4} + j \cdot 0.5 \cdot z^{-5} - h(4)z^{-6} + h(2)z^{-8} - h(0)z^{-10}
\end{aligned}
\qquad (4.3\text{-}13)
$$

Transformerer dette over til tidsplanet, dette gir følgende impulsrespons for filteret med 11 koeffisienter:

$$
h_{DAF}[n] = \left\{ \underset{\uparrow}{h(0)}, 0, -h(2), 0, h(4), j0.5, -h(4), 0, h(2), 0, -h(0) \right\} \qquad (4.3\text{-}14)
$$

Dersom de 11 koeffisientene som ble funnet med Parks & McClellan i avsnitt 4.3.2 legges inn i formel 4.3-14, fås følgende:

$$
h_{DAF}[n] = \{0.0130,0,0.0639,0,0.3016,0.5,-0.3016,0,-0,0639,0,-0.0130\} \qquad (4.3\text{-}15)
$$

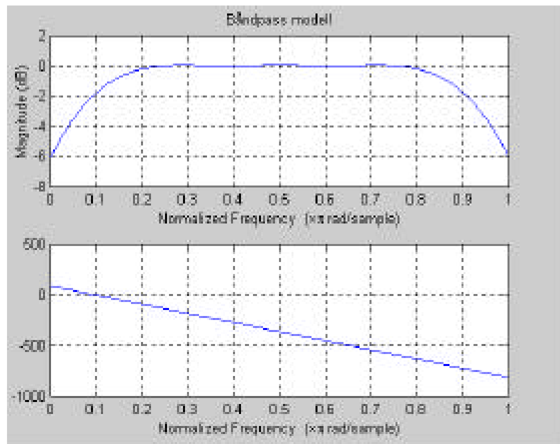
Dette gir koeffisient vektoren til et half band båndpass filter med lengde 11.

For å kunne få ut koeffisienter og responser til filtere med et generelt antall koeffisienter, ble det tatt utganspunkt i formel 4.3-9 og programmert en funksjon i Matlab kalt "H_DAF.m", fullstendig program kode for denne funksjonen er vist i vedlegg A. "H_DAF.m" inneholder også flere små kode biter som ble tilført underveis i oppgaven. Denne funksjonen tar inn antall koefisienter Q som parameter, og returnerer de aktuelle båndpass koeffisientene. Den plotter i tillegg filterets lavpass model- og båndpass respons. Koden vist nedenfor er hentet ut fra "H_DAF".m", og viser hvordan uttrykket i formel 4.3-9 ble programmert.

```
%Finner den første polyfase komponenten..................
for i=0:((Q+1)/4)-1    %..............................
```

```
   H_BP(2*i+1)=((-1)^i)*h(2*i+1);%.......................
   H_BP(Q-2*i)=((-1)^(((Q+1)/2)-1-i))*h(2*i+1);%.........
end;                    %................................


H_BP((Q+1)/2)=j*h((Q+1)/2)*(-1)^(((Q+1)/4)-1);%Setter den
andre polyfase komponenten til 0.5
```

Figur 4.3.5 viser båndpass responsen ved Q = 11 koeffisienter.



Figur 4.3.5 Båndpass med 11 koeffisienter

Funksjonen gir ut følgende 11 koeffisienter:

ans =

Columns 1 through 4

0.0130 0 0.0639 0

Columns 5 through 8

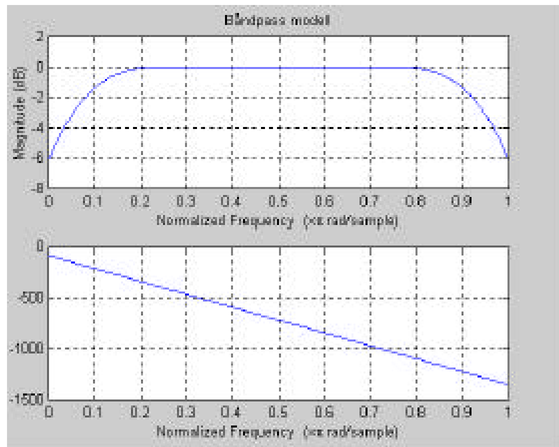
0.3016 0 + 0.5000i -0.3016 0

Columns 9 through 11

-0.0639 0 -0.0130

»
En kan se at dette stemmer helt overens med verdiene i formel 4.3-15.

Funksjonen brukes for å finne koeffisientene og responsen for et tilsvarende filter med lengde 15. Bånpass responsen ble som vist i Figur 4.3.6.



Figur 4.3.6 Båndpass med 15 koeffisienter

Funksjonen gir ut følgende 15 koeffisienter:

ans =

Columns 1 through 4

-0.0037 0 -0.0206 0

Columns 5 through 8

-0.0723 0 -0.3054 0 - 0.5000i

Columns 9 through 12

0.3054 0 0.0723 0

Columns 13 through 15

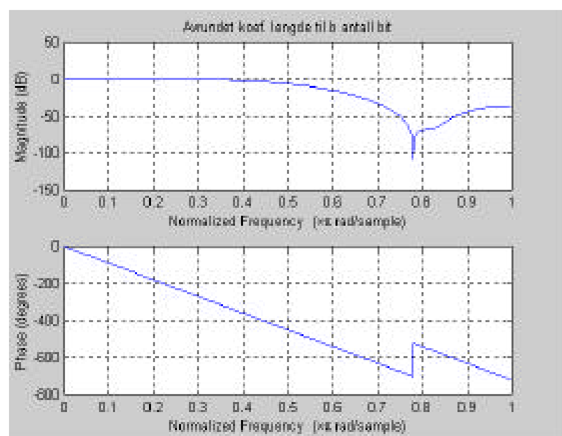
0.0206 0 0.0037

»

4.3.5. Bestemmelse av antall bit for representasjon av koeffisientene

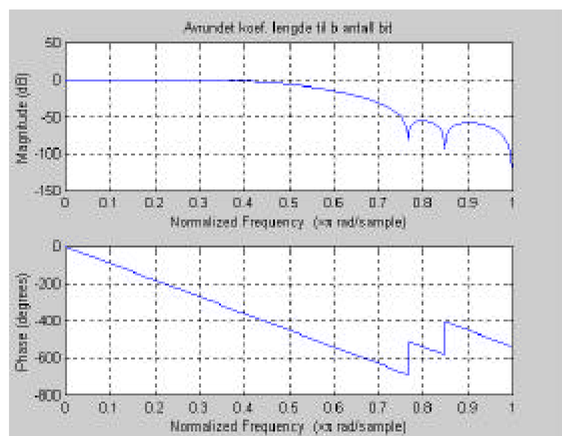
Det neste som må bestemmes er hvor mange bit koeffisientene skal representeres med. Som nevnt tidligere bruker Matlab 64 bit ved beregninger, men ved realisering må antall bit sterkt reduseres i forhold til dette. Det ble bygd videre på funksjonen "H_DAF.m" i Matlab(hele program koden er vist i vedlegg A) hvor antall koeffisienter Q, og bit lengden på koeffisientene b ble sendt med som parameter. Lengden på koeffisientene ble ved hjelp av "round" funksjonen avrundet fra 64 til b bit. Funksjonen plotter den ideelle lavpass responsen og responsen med avrunede koeffisienter.

Det ble først prøvd med forskjellig antall bit ved 11 koeffisienter.
Figur 4.3.7 viser 11 koeffisienter representert med 7 bit.



Figur 4.3.7 Koeffisienter = 11, representert med 7 bit

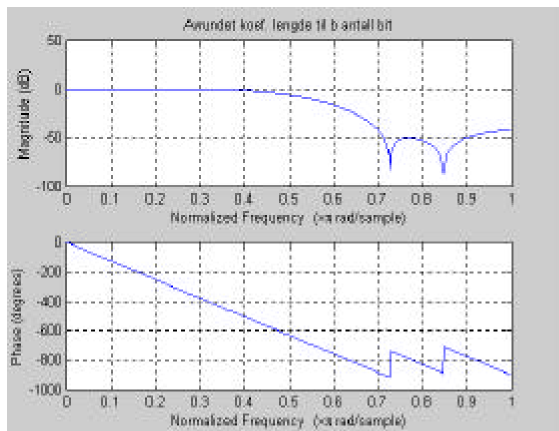
En kan se ut fra Figur 4.3.7 at en slik begrensning av antall bit fører til en for dårlig stopp bånds demping.
Figur 4.3.8 viser 11 koeffisienter representert med 8 bit.



Figur 4.3.8 Koeffisienter = 11, representert med 8 bit

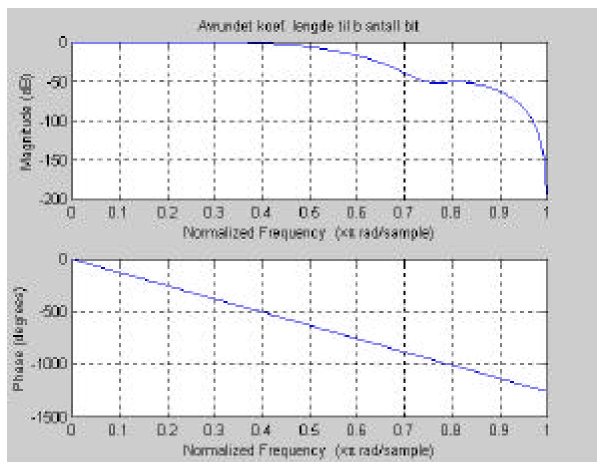
En kan se ut fra Figur 4.3.8 at responsen blir tilnærmet lik den ideelle responsen vist i Figur 4.3.3. En avrunding av koeffisientene til 8 bit vil derfor få lite å si for responsen, og foretrekkes dersom løsningen med 11 koeffisienter velges.

Det ble deretter prøvd med forskjellig antall bit ved 15 koeffisienter.
Figur 4.3.9 viser 15 koeffisienter representert med 8 bit.



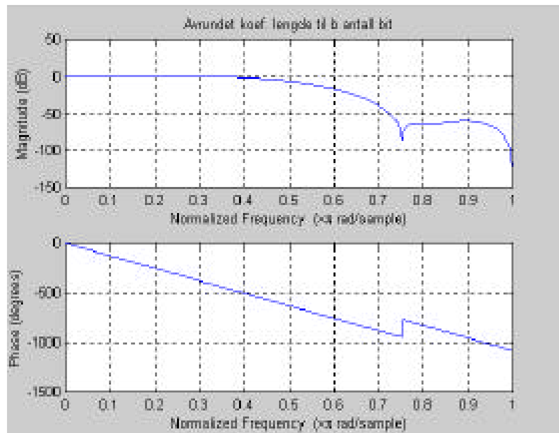
Figur 4.3.9 Koeffisienter = 15, representert med 8 bit

En kan se ut fra Figur 4.3.9 at en slik begrensning av antall bit fører til en for dårlig stopp bånds demping.
Figur 4.3.10 viser 15 koeffisienter representert med 9 bit.



Figur 4.3.10 Koeffisienter = 11, representert med 9 bit

En kan se ut fra Figur 4.3.10 at også denne begrensning fører til for dårlig stopp bånds demping.
Figur 4.3.11 viser 15 koeffisienter representert med 10 bit.



Figur 4.3.11 Koeffisienter = 15, representert med 10 bit

En kan se ut fra Figur 4.3.11 at stopp bånds dempingen til et filter med 15 koeffisienter er tilfredstillende. Et filter men 15 koeffisienter må ha 10 bits lengde på koeffisientene for å oppnå tilfredstillende respons. Likevell kan ikke denne responsen sies å være vesentlig bedre en responsen til et 11 koeffisientes filteret med 8 bits lengde på koeffisientene som vist i Figur 4.3.8. En økning fra 11 til 15 koeffisienter, og i tillegg til dette en økning fra 8 til 10 bits representasjon av koeffisientene, vil føre til mye ekstra logikk bruk i forhold til gevinst i filter respons.

Det velges derfor videre å se nærmere på en løsning med filter lengde 11, og 8 bits representasjon av koeffisientene.

Det er ressurs sparende at koeffisientene representeres som heltall. h vektoren funnet ved hjelp av Parks & McClellan må multipliseres med 2^8 og avrundes, for at dette skal innfris. Dette gjøres i Matlab som vist nedenfor.

```
» ans*256

ans =

  Columns 1 through 7

    3.3406    0.0000  -16.3511   -0.0000   77.2129  128.0000   77.2129

  Columns 8 through 11

   -0.0000  -16.3511    0.0000    3.3406

» round(ans)

ans =

     3     0   -16     0    77   128    77     0   -16     0     3
```

»

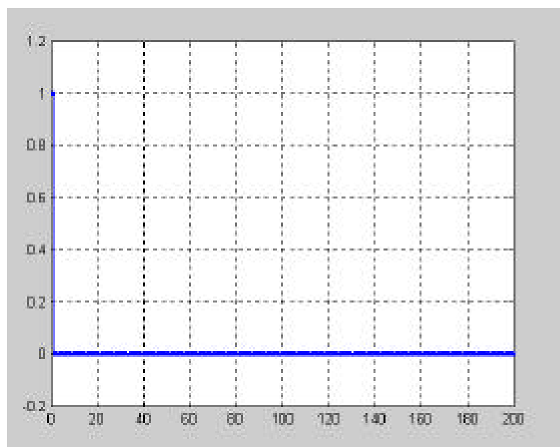
Dersom disse koeffisientene legges inn i formel 4.3-14 framkommer følgende impulsrespons:

$$h_{DAF} = \{ \underset{\uparrow}{3}, 0, 16, 0, 77, j128, -77, 0, -16, 0, -3 \} \qquad (4.3\text{-}16)$$

Ettersom alle koeffisientene er multiplisert med 2^8, vil dette gi en “gain” på 256, eller ca. 48dB.

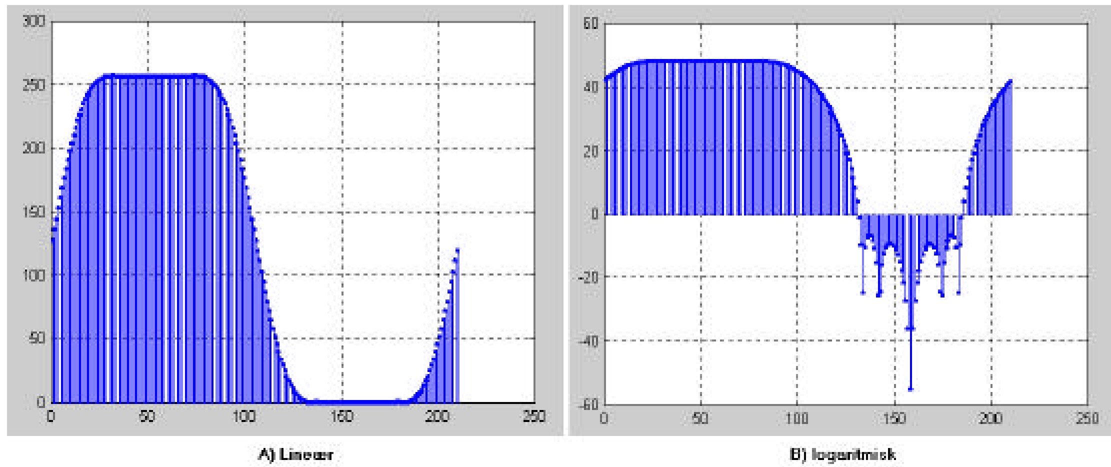
4.3.6. Test av filterets koeffisient vektor

Det neste som nå må gjøres er å teste om dette filteret i teorien fungerer som forutsatt. Det velges å tilnærme en dirac delta puls ved å legge sammen praktiske signaler i form av diskrete sinussignaler. Dette signalet bygges ved hjelp av Matlab programmet “inn_signal.m”, hele program koden er vist i vedlegg B.
Dette programmet legger sammen 200 diskrete sinussignaler med avstand 1MHz, fra 1MHz opp til 200MHz. Det sammensatte signalet deles deretter på 200 for å få en amplitude på 1, og blir som vist i Figur 4.3.12.



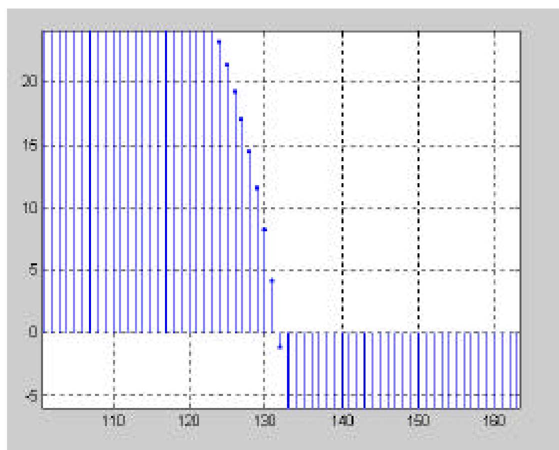
Figur 4.3.12 Sammensatt inn signal

Det sammensatte signalet vist i Figur 4.3.12 ble foldet med filterts impulsrespons ved hjelp av å utvidde Matlab programmet “H_DAF.m”(hele program koden er vist i vedlegg A).
Filter responsen ble som vist i Figur 4.3.13.



Figur 4.3.13 Filter respons

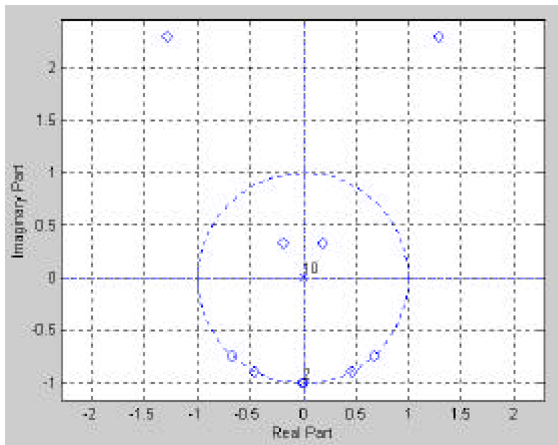
Ser fra Figur 4.3.13 at gainet i passbåndet er som forventet, men at stoppbånds dempingen ikke er tilstrekklig ved 125MHz. Ser nærmere på dette ved å zoome inn på Figur 4.3.13.b), vist i Figur 4.3.14.



Figur 4.3.14 Forstørret filter respons

En kan se ut fra denne figuren at dempingen ikke er tilstrekkelig før ved ca. 133MHz. Dette vil medføre at komponentene som ligger i området fra 125MHz til 133MHz vil gir mer støy ettersom de ikke er dempet tilstrekkelig. Grunnen til at dempingen her ikke er tilstrekkelig er vanskelig å forklare, siden dempingen i lavpass modellen var tilstrekkelig. Det ble derfor valgt å gå videre med denne løsningen.

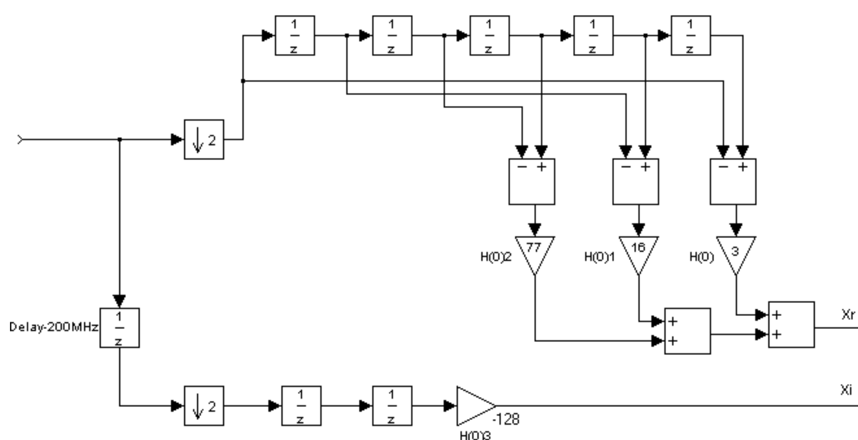
Ut fra Figur 4.3.13 kan en observere at filteret ikke er symetrisk om fs/2, dvs. som nevnt tidligere at nullpunktene ikke er kompleks konjugerte. Dette vises i Figur 4.3.15. Går man nærmere inn på det logaritmiske plottet på Figur 4.3.13.b), og sammenligner mot Figur 4.3.15, ser man at dette sammenfaller.



Figur 4.3.15 Poler og nullpunkt

4.3.7. Overføring til struktur

Når filter arkitekturen skal bestemmes tas det utgangspunkt i filterets transfer funksjon vist i formel 4.3-13. I denne formelen er det tatt utganspunkt i nedsampling på utgangen av filteret, dvs. at forsinkelser er sett i forhold til 200MHz. I følge identitet 3 i avsnitt 2.4.3.1 kan denne nedsamplingen flyttes til inngangen av filteret, dette gjøres ved å dividere eksponenten til z med M=2. Som kjent vil filteret spilttes i to greiner, hvor annenhver koeffisient vil legges i hver sin grein. Det er en fordel å gjøre det slik med tanke på å redusere frekvensen i systemet så tidlig som mulig.
Filter strukturen vil bli som vist på Figur 4.3.16.



Figur 4.3.16 Filter struktur

Den imaginære enhet j som står foran den andre polyfase komponenten i formel 4.3-13, indikerer at den ene greinen er forskøvet π/2 (j) i forhold til den andre. Ettersom det er π/2 mellom samplene i de to greinene, vil det bli π mellom sampler innen for samme grein(alt sett i forhold til 200MHz). Siden det er π mellom samplene innen for samme grein, medfører dette at fortegnene på koeffisientene må inverteres for at resultatet ut ikke skal gi fase endring i forhold til responsen vist i Figur 4.3.5. I tilegg til dette vil det spares et bit i H(3)

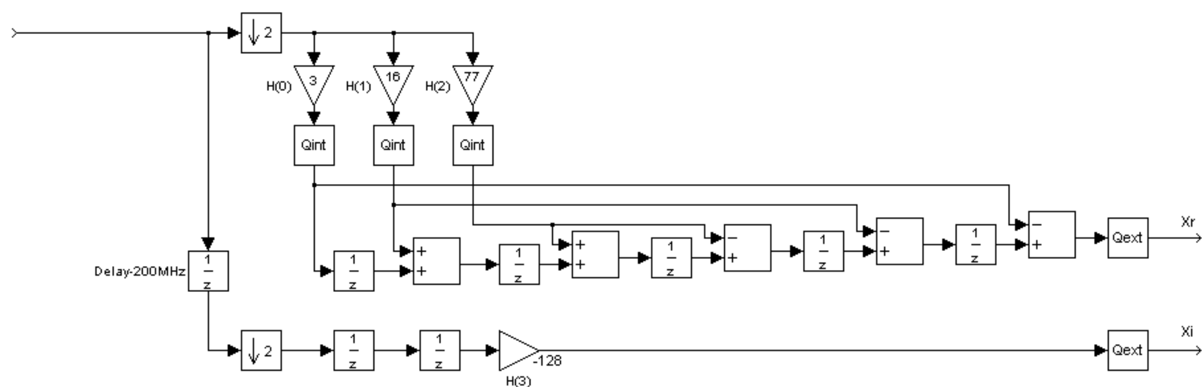
koeffisienten, ettersom –128 kan representeres med 8 bit, mens +128 krever 9 bit(2's komplement form).

For å klargjøre litt:
Dersom filteret hadde blitt bygd opp ut fra transferfunksjonen til lavpass filteret i formel 4.3-4, ville det hatt samme struktur som i Figur 4.3.16, bortsett fra at koeffisientene ville hatt fortegn som stemmer overens med lavpass vektoren. Det ville da ha vært et rent reelt lavpass filter. Ettersom koeffisientene i Figur 4.3.16 har fremkommet ved å modulere transferfunksjonen i formel 4.3-4, vil dette bli et komplekst båndpass filter.

Ettersom innsignalet er modulert slik at det er senterert rundt $\pi/2$ istedenfor 0, vil avstanden mellom hvert sample være $\pi/2$ i forhold til frekvensen på inngangen til filteret(200MHz). Dette er en fordel med tanke på den filter strukturen som er laget, da samplene enkelt kan plukkes rett ut.

4.3.8. Overgang til praktisk design

For å minske eventuelle problemer med glitcher i det realiserte filteret, kan det være hensiktsmessig å forandre på strukturen i Figur 4.3.16, slik at lange kjeder med aritmetiske operasjoner ikke forekommer. Dette kan løses effektivt uten bruk av ekstra logikk, ved å bruke forsinkelsene som pipelining. Stukturen vil da bli som vist i Figur 4.3.17.



Figur 4.3.17 Filter struktur-b

På Figur 4.3.17 er det også satt inn noen ekstra blokker, QINT og QEXT. QEXT skal avskjære antall bit ut fra filteret, slik at det i møte kommer kravet til antall bit på utgang. QINT skal avskjære antall bit etter multiplikatorene, slik at det kan brukes færre antall bit gjennom tidsforsinkelsene og adderne, og av den grunn redusere bruken av logikk. To metoder for avskjæring av antall bit er kjent fra avsnitt 2.3, dette er "truncation" og "rounding". "Truncation" ville vært det enkleste å realisere, men ettersom dette gir en forventningsverdi på støyen lik $E=-\Delta/2$, som vil opptre som et DC nivå, er ikke dette særlig gunstig. Grunnen til dette er at filteret som kjent har en gain ved DC på ca. 42dB. Denne komponentene vil da være meget uheldig. Det velges å bruke "rounding" på dette filteret, ettersom forventningsverdien på støyen da er lik 0.

Avrundingsfunksjonene vil medføre avrundings støy.

Ser på real greinen:
Dersom antall bit på utgangen er w+1(1 bit til fortegn, w bit til selve tallverdi), vil støyen grunnet denne avrundingen som kjent fra avsnitt 2.3, være gitt av:

$$\frac{2^{-2w}}{12} \qquad (4.3\text{-}17)$$

Dersom antall bit internt er b+1(1 bit til fortegn, b bit til selve tallverdien), vil støyen grunnet denne avrundingen være(for en generell filterlengde Q):

$$\left(\frac{Q+1}{4}\right)\cdot\frac{2^{-2b}}{12} \qquad (4.3\text{-}18)$$

Det er hensiktsmessig å velge antall bit ut fra QINT, slik at total intern avrundings støy blir mindre en avrundings støyen på utgangen grunnet QEXT. Antall bit b+1 finnes derfor ved å løse følgende ulikhet med hensyn på b:

$$\frac{2^{-2w}}{12} > \left(\frac{Q+1}{4}\right)\cdot\frac{2^{-2b}}{12}$$

$$\log_2\left(\frac{2^{-2w}}{12}\right) > \log_2\left(\frac{Q+1}{4}\cdot\frac{2^{-2b}}{12}\right)$$

$$\log_2\left(2^{-2w}\right) - \log_2(12) > \log_2(Q+1) + \log_2\left(2^{-2b}\right) - \log_2(4) - \log_2(12)$$

$$(4.3\text{-}19)$$

$$-2w - \log_2(12) + \log_2(12) - \log_2(Q+1) + 2 > -2b$$

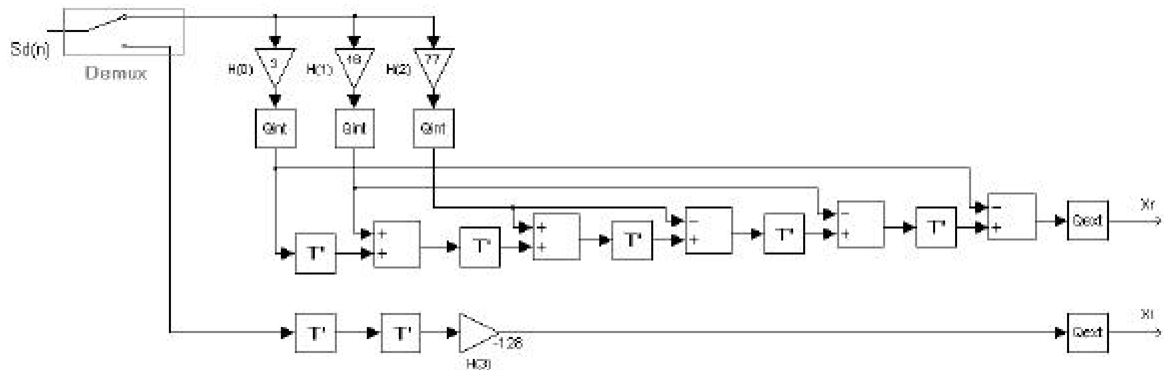
$$b > w + \frac{1}{2}\log_2(Q+1) - 1$$

Har gitt fra krav spesifikasjonene at w+1=12bit, og Q=11 er tidligere valgt, dette gir:

$$b > 10 + \frac{1}{2}\log_2(12) \qquad (4.3\text{-}20)$$

, dette gir b > 11,79, dvs. at internt bit antall(b+1) må være minst 13bit.

For å få frem en struktur som kan realiseres må blokkskjema vist i Figur 4.3.17 transformeres over i tidsdomene. Denne strukturen vil bli som vist i Figur 4.3.18, T' i denne figuren er sett i forhold til 100MHz.



Figur 4.3.18 Filter struktur i tidsdomene

Filteret skal realiseres ved hjelp av VHDL beskrivelse, og for å få en oversikt over hvordan alle entitetene skal kobles sammen i topp filen, og for å gjøre programmerings jobben ryddigere, er det laget et skjema vist i vedlegg D. Den endelige VHDL koden er lagt med som vedlegg E.

Total støy på DAF filterets utgang

I tillegg til den interne støyen grunnet avrunding, vil det ved påtrykk av et inngangs signal dannes "leakage" komponenter på utgangen grunnet endelig demping(α) i DAF filtererts stoppbånd. Noe av den totale støyen fra ADC(σ^2_{ADC}), vil slippe gjenom passbåndet til DAF.

Den totale støyen på utgangen til DAF filteret, vil da være gitt av følgende formel:

$$\sigma^2_{DAF} = \frac{2^{-2w}}{12} + \sigma^2_{ADC} \cdot \frac{B}{fs} + 10^{-\frac{\alpha}{10}} \cdot (Signal\ og\ støy\ fra\ tidligere\ i\ systemet) \quad (4.3\text{-}21)$$

Grunnen til at støyen grunnet avrunding internt er utelatt fra denne formelen, er at avrundingsstøyen på utgangen dominerer.

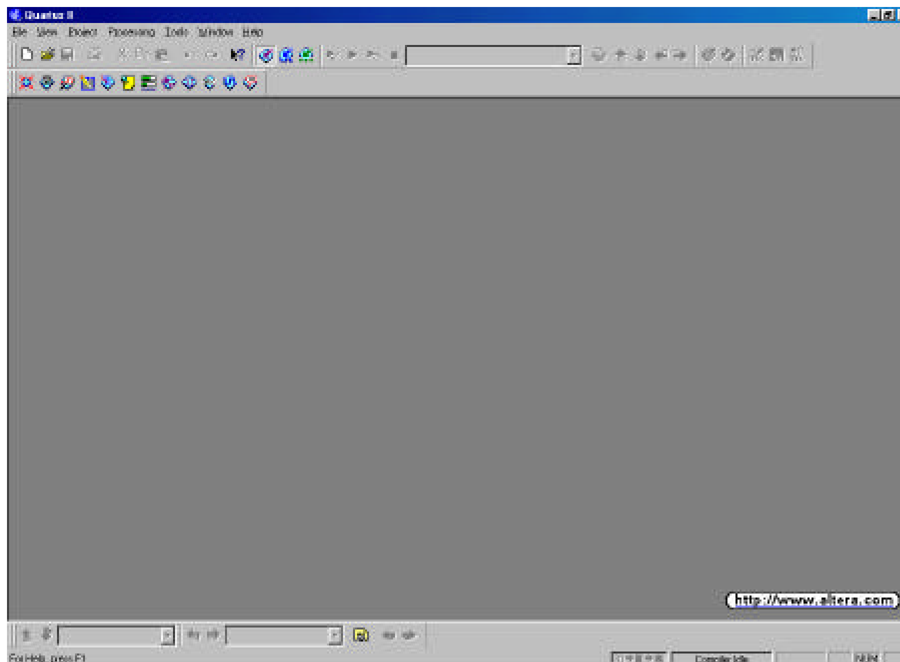
For en mer grundig støy analyse, henvises det til "Design of a configurable 4-64 channel digital frequency demultiplexer for an onboard DVB-RCS/DVB-S processor" av "Bjørn Roger Andersen".

4.4. Quartus

Dette avsnittet er skrevet som en bruker manual for QuartusII. Det tar for seg design av DAF filteret omtalt tidligere i rapporten, som et gjennomgående eksempel. Dette avsnittet forutsetter ikke innsikt i foregående kapitler, men en viss kjennskap til strukturen til DAF filteret er en fordel(se Figur 4.3.18).

4.4.1. Oppstart av Quartus

Når man startet QII så vil skjermbilde være som vist i Figur 4.4.1.

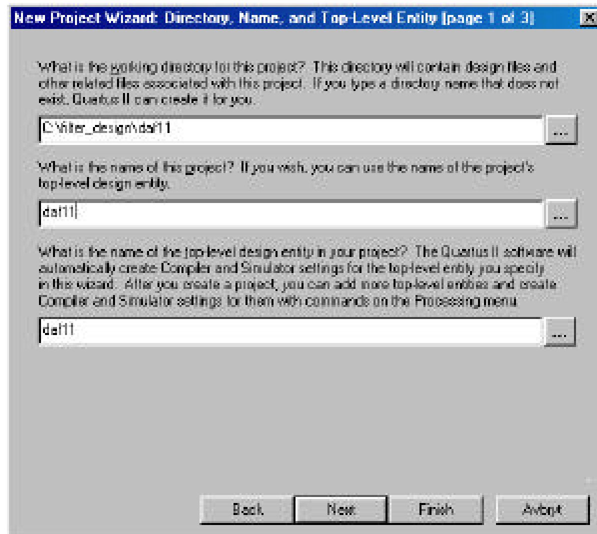


Figur 4.4.1 Oppstart

Det første man må gjøre når man skal lage et nytt design er å opprette et nytt prosjekt.

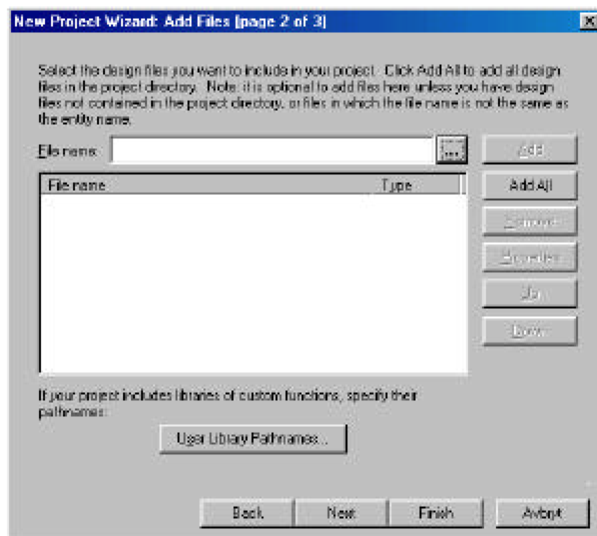
4.4.2. Opprettelse av nytt prosjekt

For å opprette et nytt prosjekt kan man gå på menyvalget "File->New project wizard". Det vil da starte en Wizard som viser hvordan et nytt prosjekt kan opprettes. Det første skjermbildet som dukker opp er vist i Figur 4.4.2. Den øverste ruten angir hvilken katalog prosjektet skal ligge i, den midtre ruten angir prosjektets navn og den nederste ruten angir toppfilens navn. I denne oppgaven ble det som vist i Figur 4.4.2 valgt å legge prosjektet i katalogen c:\filter_design\daf11, mens både prosjektet og prosjektets toppfil fikk navnet daf11. Det kan med fordel velges samme navn på både prosjektets katalog, på selve prosjektet og på toppfilen slik som det er gjort i denne oppgaven.



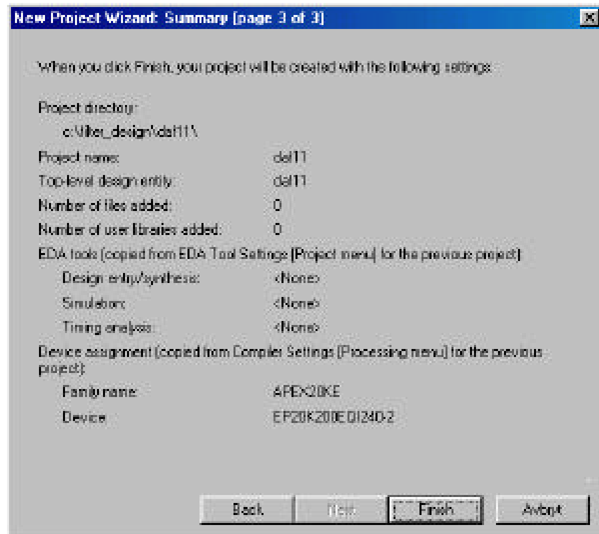
Figur 4.4.2 Opprettelse av nytt prosjekt_figur1

Etter at de ønskede navnene er skrevet inn klikker man på "next". Det vil da sansynligvis dukke opp en melding på skjermen om at denne mappen ikke eksisterer og et spørsmål om den skal opprettes. Her svarer man ja og neste skjermbilde vist i Figur 4.4.3 dukker opp.



Figur 4.4.3 Opprettelse av nytt prosjekt_figur2

Figur 4.4.3 viser neste skjermbilde hvor man kan hente inn design filer til det nye prosjektet. Dette betyr at hvis man tidligere har laget design filer som man ønsker å bruke i dette nye prosjektet så kan man legge til disse her.
Etter å eventuellt ha lagt til disse filene klikker man "Next" og neste skjermbilde vist i Figur 4.4.4 dukker opp. Dette skjermbilde viser slutt status over opprettelsen av det nye prosjektet, og hvis alt ser riktig ut klikker man "Finish" og opprettelsen av prosjektet er fullført.

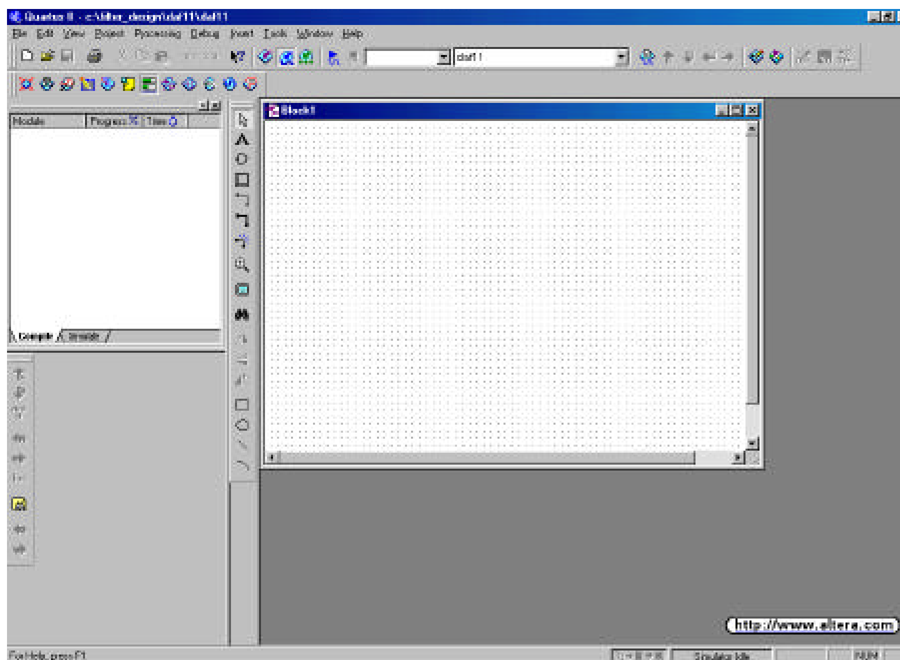


Figur 4.4.4 Opprettelse av nytt prosjekt_figur3

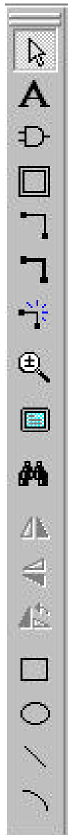
4.4.3. Opprettelse av BDF fil

Når man skal lage et nytt design kan man velge flere forskjellige format på topp filen, blant annet VHDL eller BDF.

I denne oppgaven ble det valgt å bruke en BDF fil som topp fil, da dette gir et oversiktlig bilde av designet. En slik BDF fil opprettes ved å klikke på menyvalget "File->New->Device design files->Block diagram/schematic file". En blank BDF fil med navnet "Block1"vil da dukke opp som vist i Figur 4.4.5. Denne BDF filen må lagres i prosjekt mappen, med samme navn som prosjektet. I dette tilfeldet blir det da "daf11" i mappen c:\filter_design\daf11.



Figur 4.4.5 Opprettelse av BDF fil



Figur 4.4.6
Toolbar

Alle tigjengelige funksjoner i QII kan hentes fram ved hjelp av menylinjen øverst på skjermen. I tillegg til dette er mange funksjoner med en viss tilhørlighet til hverandre samlet i såkalte verktøylinjer, slik at de lettere kan hentes fram. Det er en stor fordel å legge fram på skjermen de verktøylinjene man skal brukes mest. Dette gjøres ved å gå på menyvalget "Tools->Toolbars" og merke av for den som ønskes.

Mange av de funksjonene som brukes regelmessig når man skal bygge opp et design med en BDF fil som topp fil, er samlet i en verktøy linje som kalles "block & symbol editors". Dette er den vertikale verktøylinjen på skjermen i Figur 4.4.5 , og er vist i sin helhet på Figur 4.4.6.

Hvis denne ikke allerede ligger framme på skjermen anbefalles det at den hentes fram som beskrevet ovenfor.

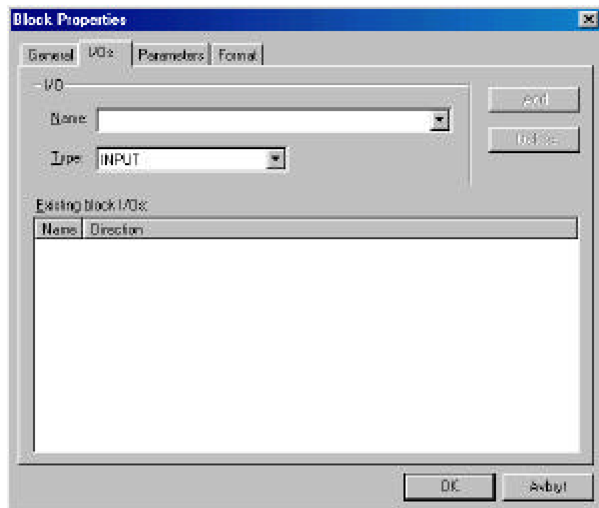
Ved å holde musepekeren over de forskjellige valgene på verktøylinjene i ca 1sekund, kommer navnet på det aktuelle valget fram. Det er disse navnene det henvises til i resten av rapporten.

4.4.4. Programmering av entiteten "downsampler"

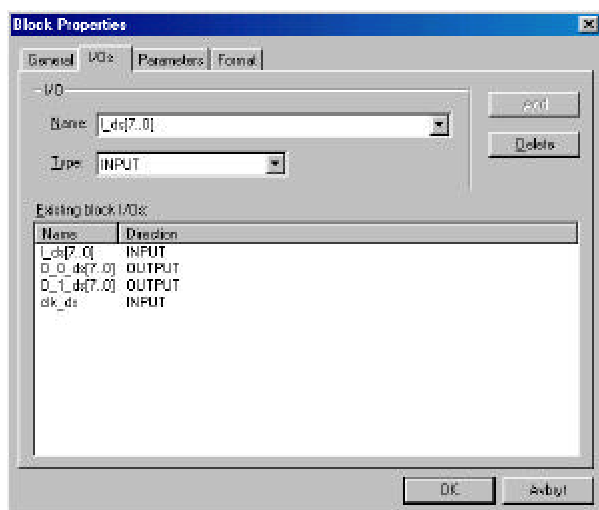
Man kan opprette såkalte entiteter ved å legge inn blokker med inn- og utganger i BDF filen, og deretter programmere arkitekturen til entiteten ved hjelp av VHDL, AHDL eller VerilogHDL.
Den første entiteten i denne oppgaven er "Downsampleren". Funksjonen til "downsampler" kan beskrives på følgende måte:
Den får inn sampler representert med 8 bit på inngangen "I_ds[7..0]", og et klokkesignal på inngangen "clk_ds"som er det dobbelte av klokkesignalet som brukes videre i kretsen. Annen hvert sampel sendes ut til utgangen "O_0_ds[7..0]" og de resterende til utgangen "O_1_ds[7..0]".

Velg "block tool"(nr 4 fra toppen) fra "block & symbol editors" verktøylinjen og lag en blokk ute i BDF filen. Velg deretter "selection tool"(nr 1 fra toppen) fra den samme verktøylinjen og dobbeltklikk på blokken som ble laget i BDF filen. Det dukker da opp et nytt skjermbilde som vist i Figur 4.4.7 som inneholder blokkens egenskaper.
Her må man gå under menyen "General" og skrive inn blokkens navn, dette navnet ble i denne oppgaven valgt til "downsampler". Deretter må man gå på menyvalget "I/Os" hvor blokkens inn- og utganger defineres. I denne oppgaven ble det valgt å gi den 8bits inngangen,

klokke inngangen og de to 8bits utgangene hendholdsvis navnene I_ds[7..0], clk_ds, O_0_ds[7..0] og O_1_ds[7..0].

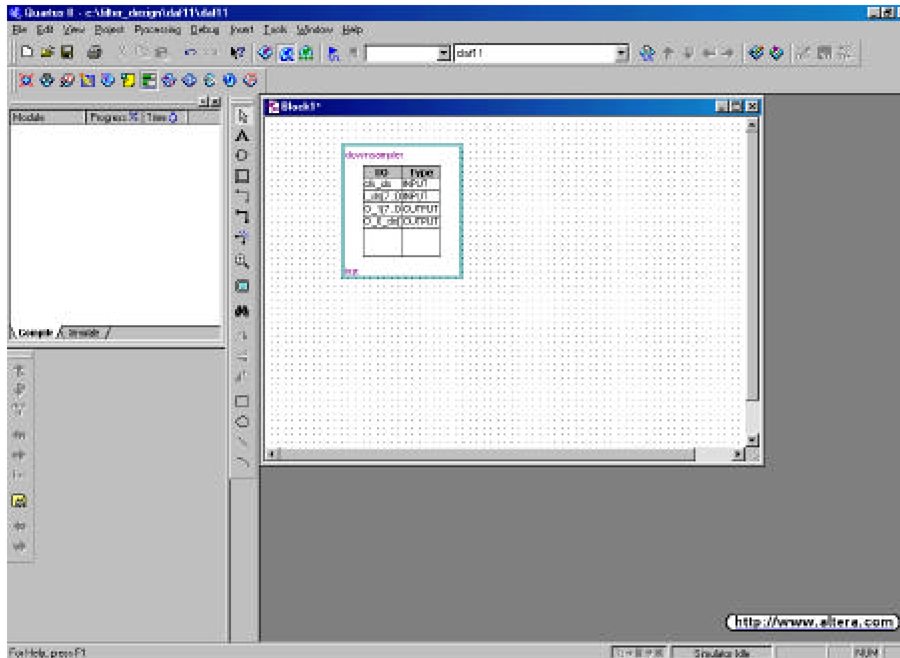


Figur 4.4.7 Programmering av ”downsampler”_figur1



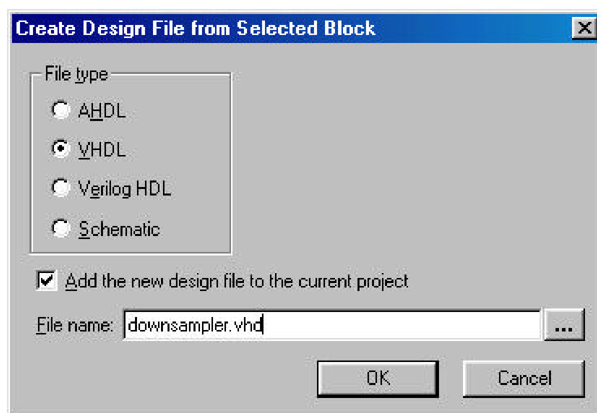
Figur 4.4.8 Programmering av ”downsampler”_figur2

Når blokkens egenskaper er bestemt klikker man ”OK” og skjermbildet vil bli som vist på Figur 4.4.9.



Figur 4.4.9 Programmering av ”downsampler”_figur3

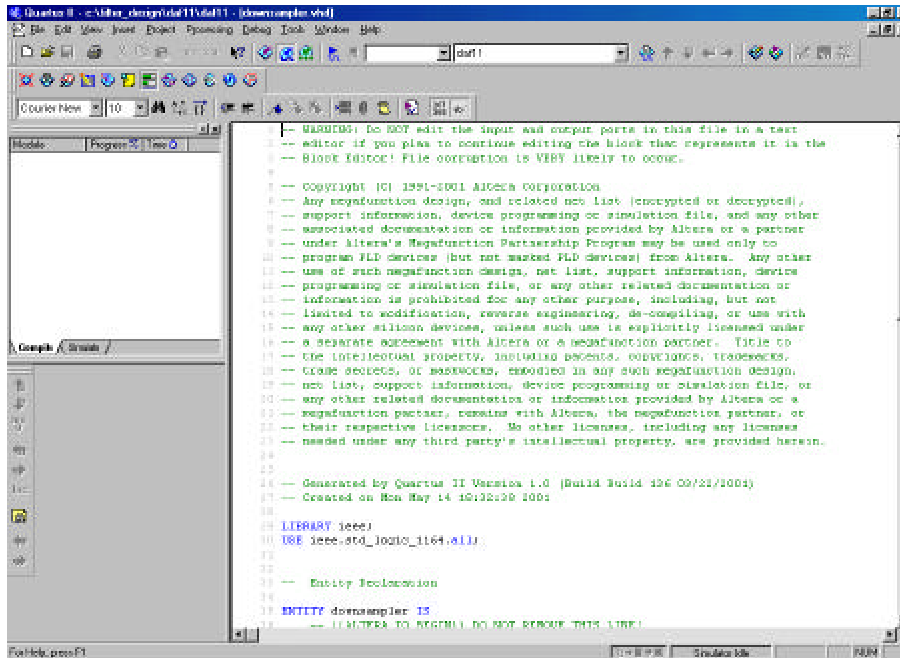
Det neste som må gjøres er å programmere entitetens arkitektur. Det må først opprettes en design fil for entiteten. Dette gjøres ved å høyreklikke på boksen og velge “create design file from selected block”. Et nytt skjermbilde som vist i Figur 4.4.10 vil dukke opp. Her kan man velge om man vil programmere i AHDL, VHDL, VerilogHDL eller skjematisk. Navnet på filen må være det samme som blokkens navn, altså “downsampler”. Det ble i denne oppgaven valgt å bruke VHDL som programmerings språk.



Figur 4.4.10 Programmering av ”downsampler”_figur4

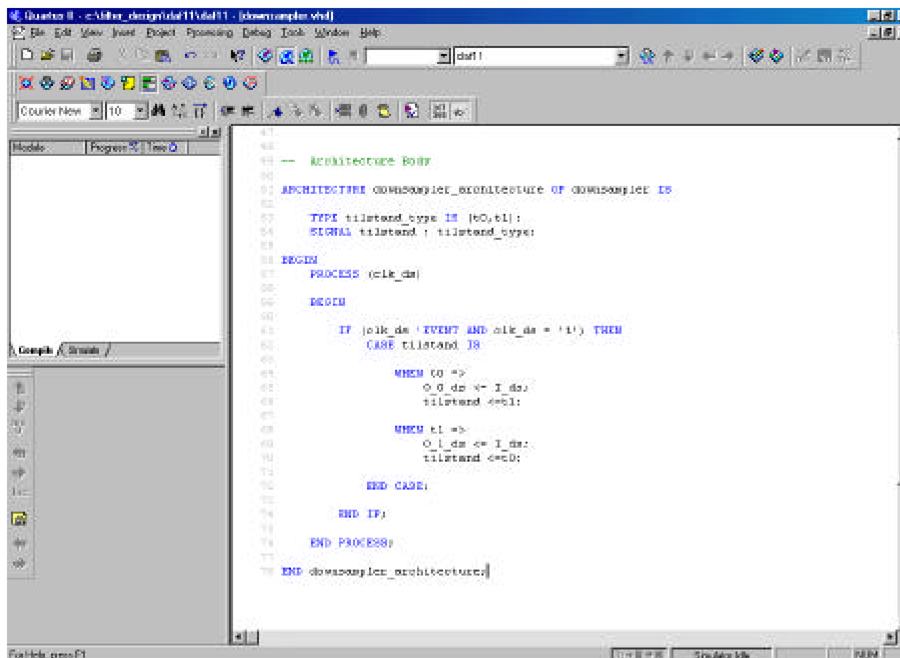
Etter at programmeringsspråket er valgt, og man har forsikret seg om at navnet på filen er satt til å være det samme som blokkens navn, klikker man “OK” og design filen genereres.

Når dette er gjort dukker design filen til entiteten “downsampler” opp på skjermen som vist på Figur 4.4.11.



Figur 4.4.11 Programmering av "downsampler"_figur5

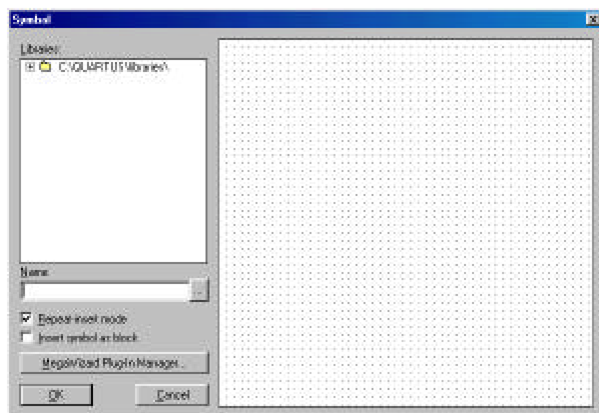
Denne filen inneholder all deklarering av alle inn- og utganger som ble definert under blokkens egenskaper, og all annen nødvendig "heading" for at filen skal være kjørbar. Entitetens arkitektur er derimot helt blank, og må programmeres. Arkitekturen til "downsampler" ble programmer og er vist i Figur 4.4.12. Hele program koden til "downsampler er vist i vedlegg E.



Figur 4.4.12 Programmering av "downsampler"_figur6

4.4.5. Opprettelse av multiplikasjons enhetene

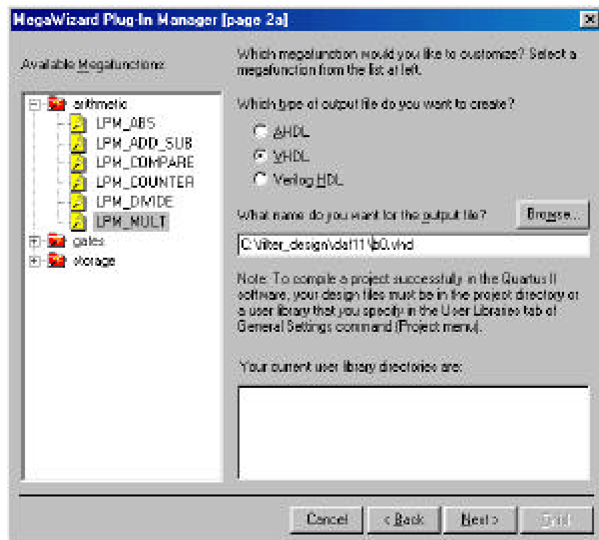
I QII har man tilgang på mange såkalte megafunksjoner som er standard funksjoner som kan benyttes i designet. Spesielt interessante er LPM(Library of Parameterized modules) megafunksjonene som ligger under IEEE biblioteket, siden dette er en standard som ”alle” leverandører av aktuelt utstyr skal følge. I denne oppgaven ble det i første omgang benyttet megafunksjonen ”LPM_MULT”, som er en multiplikasjons krets. Denne ble brukt til å lage fire forskjellige multiplikasjons filer, b0, b1, b2 og b3.
Det første man må gjøre for å lage disse filene er å klikke på ”symbol tool”(nr 3 fra toppen) på “block & symbol editors” verktøylinjen. Man får da opp et skjermbilde som vist på Figur 4.4.13.



Figur 4.4.13 Opprettelse av ”b0”_figur1

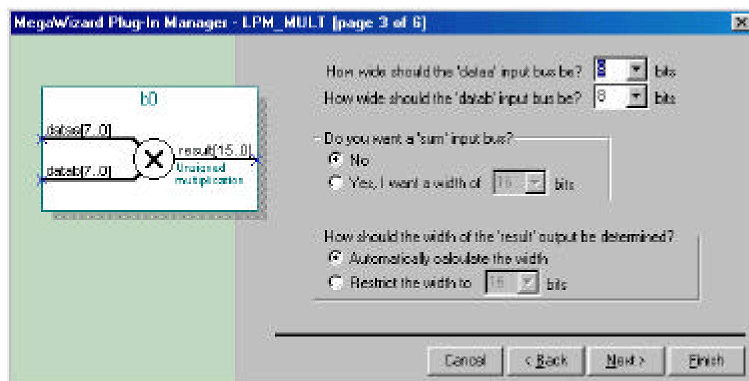
Her kan man enten gå rett på c:\QUARTUS\libraries\ og finne den megafunksjonen man ønsker, eller klikke på “Megawizard plug-in manager” og bli guidet gjenom opprettelsen ved hjelp av en Wizard. Bakdelen med det første alternativet er at symbolet som blir tegnet inn i BDF filen inneholder alle mulige inn- og utganger som finnes på den aktuell megafunksjonen. Ved å velge det andre alternativet kan man hele veien bestemme hvilken av inn- og utgangene som finnes på den aktuelle megafunksjonen som skal benyttes. De som velges bort blir da ikke synlig på det ferdige symbolet. Det anbefalles derfor å velge dette andre altentivet.

Klikk på “Megawizard plug-in manager”, det vil dukke opp et valg på om det skal lages et nytt symbol eller redigeres et gammelt. Klikk “create new” og et nytt skjermbilde som vist på Figur 4.4.14 dukker opp.



Figur 4.4.14 Opprettelse av "b0"_figur2

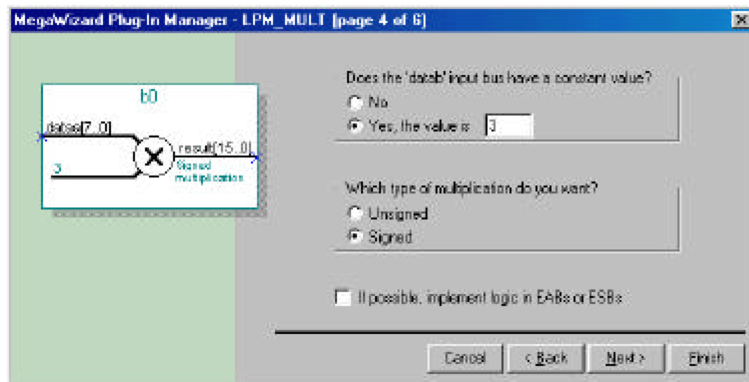
Her må man velge hvilken mega funksjon man vil bruke, om koden til entitets filen skal være av typen AHDL, VHDL eller VerilogHDL, og navnet på entiteten. Som nevnt ovenfor skal det først lages en multiplikator ved hjelp av mega funksjonen LPM_MULT, denne velges under menyen "arithmetic" øverst til venstre i figuren. Det ble valgt å bruke VHDL, og navnet på denne første multiplikatoren ble b0.
Etter at valgene er gjort kan man klikke "next" og neste skjermbilde vist på Figur 4.4.15 dukker opp.



Figur 4.4.15 Opprettelse av "b0"_figur3

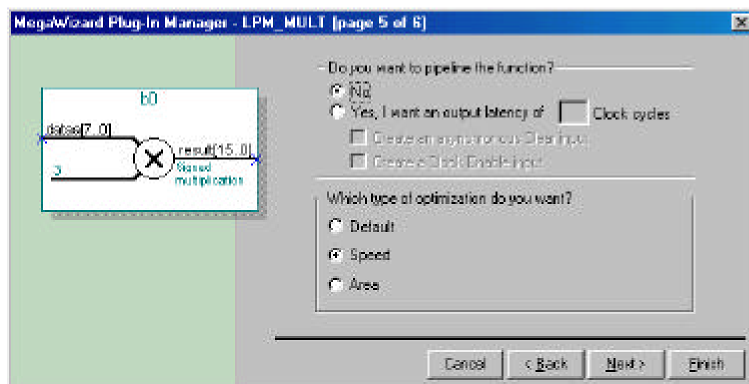
Her må man velge hvor mange bit signalene på de to inngangene skal representeres med, om man vil ha en egen "sum" inngang og hvor mange bit man vil at ut signalet skal representeres med. Hvis man her velger "Automatically calculate the with" vil ut signalet bli representert med garantert mange nok bit. Her er det viktig å være klar over at hvis man velger "Restrict the with to .. bits" foretas en trunkering, dvs. at et nødvendig antall av LSB bitene fjernes uten noen form for avrunding. *NB Denne trunkeringen avskjærere bit direkte også når de er på 2's kompliment form, dette blir feilaktig.*
I denne oppgaven er innsignalet til multiplikatoren representert med 8 bit, det samme er koeffisienten. Etter multiplikatorene skal signalet avrundes til 13 bit. Det finnes ingen

funksjoner på multiplikasjons kretsen som gjør dette direkte, her er det kun muligheter for "truncation". Denne avrundingen må derfor utføres i en egen entitet senere. Ut signalet fra multiplikatorene må derfor representeres med 16 bit som vist på figur?.
Etter at valgene er gjort klikker man på "next" og skjermbildet på Figur 4.4.16 dukker opp.



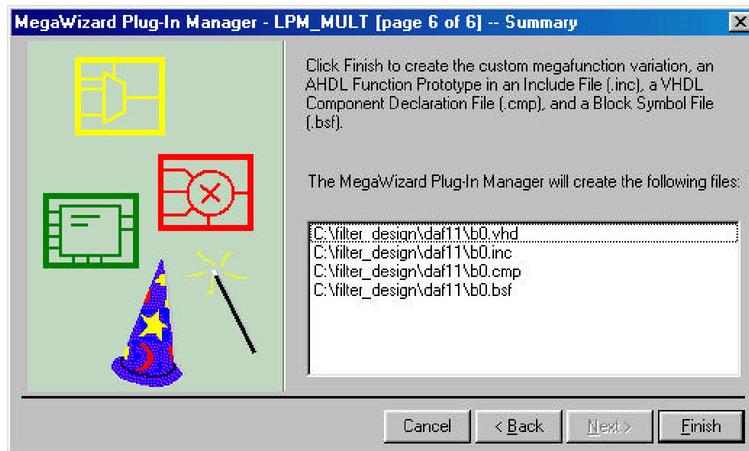
Figur 4.4.16 Opprettelse av "b0"_figur4

Her kan man velge om den ene inngangen på multiplikatoren skal være en konstant, og om multiplikatoren skal regne med "signed" eller "unsigned", dvs. hendholdsvis på 2's komplement form eller ikke. I denne oppgaven skal den ene inngangen representere en fast koeffisient, og det skal brukes 2's komplement form.
Klikk "next" for å få fram neste skjermbilde vist i Figur 4.4.17.



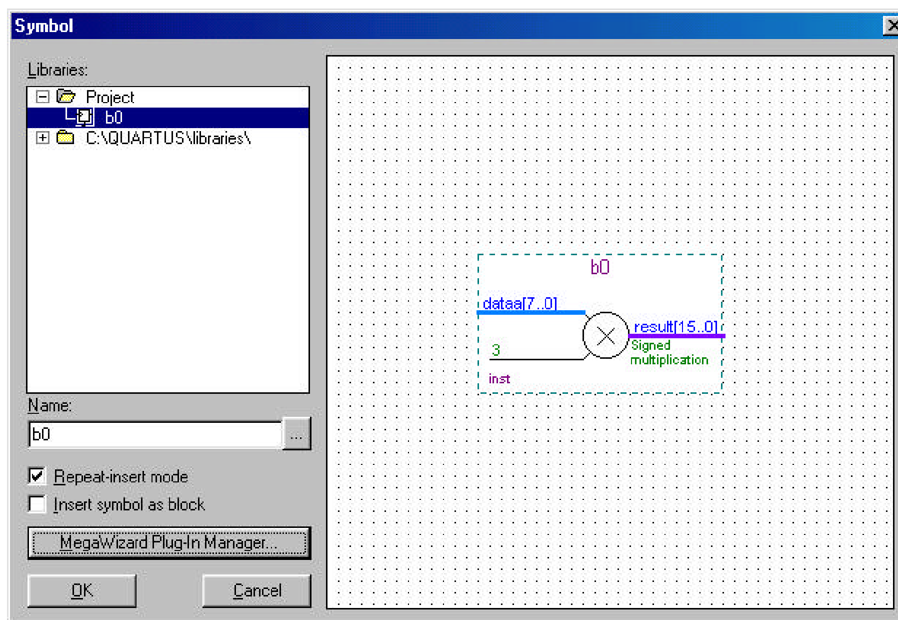
Figur 4.4.17 Opprettelse av "b0"_figur5

Her kan man velge om man vil "pipeline" multiplikatoren, og om man vil optimalisere med hensyn på hastighet eller besparelse av plass på brikken. Det kan ofte være hensiktsmessig å "pipeline" slike funksjoner, men det gjøres likevel ikke i denne oppgaven. Grunnen er at filteret er bygd opp på en slik måte at det settes inn skift registre mellom hver aritmetisk opperasjon. Dette vil føre til tilstrekelig "pipelining". Det velges her å optimalisere med tanke på hastighet siden klokkefrekvensen skal være såpass høy. Ved å klikke "next" vil en status over hvilken filer som kommer til å bli generet dukke opp som vist på Figur 4.4.18.



Figur 4.4.18 Opprettelse av "b0"_figur6

Klikk "finish", og den første multiplikator filen er ferdig opprettet. Symbolet er vist på Figur 4.4.19.



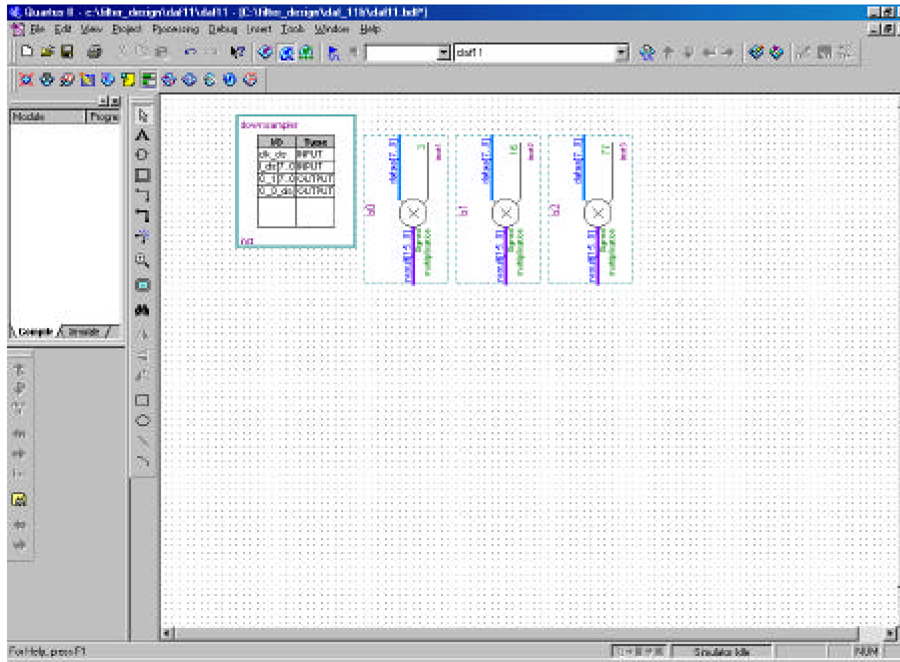
Figur 4.4.19 Opprettelse av "b0"_figur7

Denne første multiplikator filen ble kalt "b0". Hele prossesen omtalt i dette avsnittet må gjøres på nytt for å lage "b1", "b2" og "b3", med koeffisient verdier på henholdsvis 16, 77 og -128.

4.4.6. Innsetting av multiplikasjons entitetene i designet

Etter at alle multiplikasjons enhetene er ferdig opprettet kan man begyne å legge dem inn i BDF filen. Dette gjøres ved å gå på meny valget "Insert" og deretter "Symbol" eller "Symbol as block". Velger man "Symbol" vil den aktuelle kretsen bli lagt rett inn i BDF filen, velger

man “Symbol as block” vil den aktuelle kretsen bli lagt inn i BDF filen i en boks på samme måte som entiteten “downsampler”. I denne oppgaven ble det valgt å bruke “Symbol”, og b0, b1 og b2 ble plassert som vist på Figur 4.4.20. På “edit” menyen ligger det muligheter for å rotere symbolene etter at de er satt inn i BDF filen.

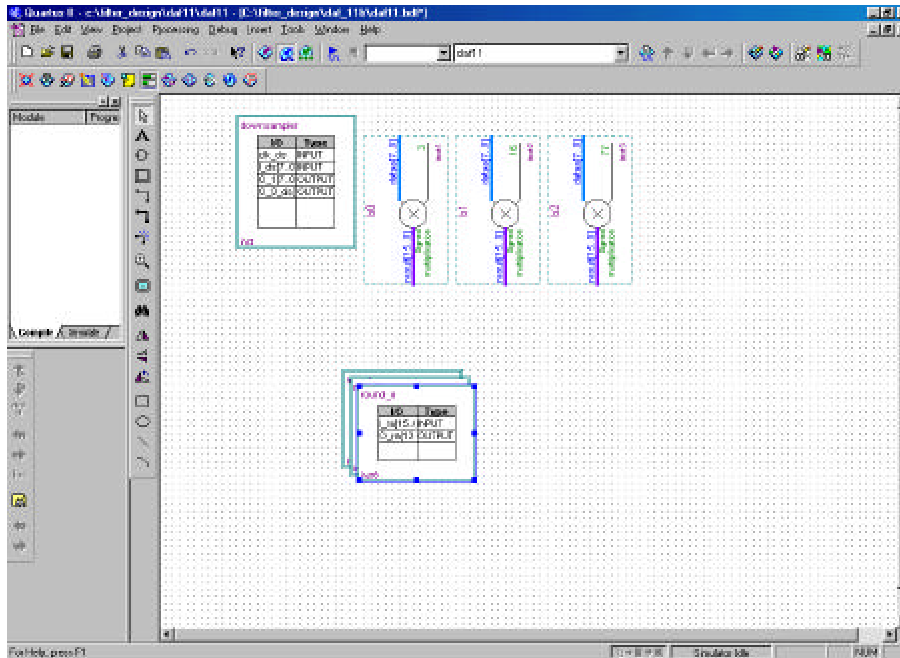


Figur 4.4.20 Innsetting i BDF filen_figur1

4.4.7. Programmering av entiteten “round_a”

Det neste som skal gjøres er å lage en entitet som avrunder 16 bit til 13 bit, etter de tre innsatte multiplikatorene. Entiteten ble valgt til å hete “round_a”, og har en 16 bits inngang “I_ra[15..0]” og en 13 bits utgang “O_ra[12..0]. Selve opprettelsen av entiteten foregår på akkurat samme måte som under “opprettelsen av entiteten “downsampler””, man kan bla tilbake til dette avsnitt hvis man ikke husker hvordan det gjøres.
Funksjonen til “round_a” kan beskrives på følgende måte:
Den får inn et signal representert med 16 bit på inngangen “I_ra[13..0]. Hvis det tredje siste bitet(I(2)) er 0, legges I_ra[13..1] direkte på utgangen O_ra[12..0], det vil si en avrunding nedover. Hvis derimot I(2) er 1, legges I_ra[13..1] +1 på utgangen O_ra[12..0], det vil si en avrunding oppover.
Funksjonen tar hensyn til om signalet inn er negativt. Signalet tilbake komplemeteres da før behandling, og komplementeres etter.

Etter at entiteten er lagt ut i BDF filen kan man lage to kopier av den siden den skal brukes tre steder. Dette gjøres ved å merke boksen og trykke “clt+c” en gang og “clt+v” to ganger. BDF filen vil da se ut som på Figur 4.4.21. Størrelsen på blokkene kan justeres ved å merke blokken og dra i de blå firkantene i rammen.



Figur 4.4.21 Programmering av "round_a"_figur1

Deretter dette må arkitekturen til "round_a" programmeres. Dette gjøres med akkurat samme framgangsmåte som ved programmeringen av "downsampler", og deler av programmet er vist I Figur 4.4.22. Hele program koden til "round_a" er vist i vedlegg E.

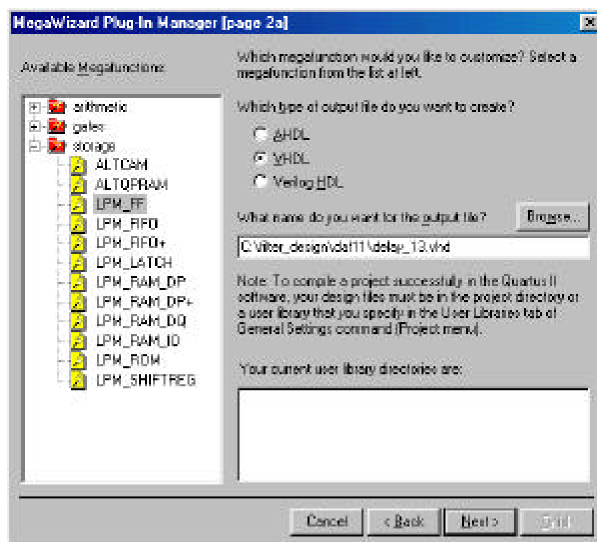
```
    );

END round_a;

--  Architecture Body

ARCHITECTURE round_a_architecture OF round_a IS
    SIGNAL temp_inn : STD_LOGIC_VECTOR(15 downto 0);     --Hjelpe signal
    SIGNAL temp_inn_2 : STD_LOGIC_VECTOR(15 downto 0);   --Hjelpe signal
    SIGNAL temp_ut : STD_LOGIC_VECTOR(12 downto 0);      --Hjelpe signal
    SIGNAL temp_ut_2 : STD_LOGIC_VECTOR(12 downto 0);    --Hjelpe signal


BEGIN
    PROCESS(I_ra)                --Prosessen er "følsom" for inngangen "I_ra"
    BEGIN

        temp_inn <= I_ra;        --Setter hjelpe signalet lik inngangssignalet


        IF I_ra(15) = '1' THEN temp_inn_2 <= 0 - temp_inn;  --Hvis MSB=1, dvs. negativt tall, tas 2'
        ELSE temp_inn_2 <= temp_inn;                         --hvis ikke brukes det videre
        END IF;

        temp_ut(12 downto 0) <= temp_inn_2(15 downto 3);     --De høyeste 12 bitene kopieres

        IF temp_inn_2(2) = '1' THEN temp_ut_2 <= temp_ut +1;--Hvis det skal rundes opp, adderes tall
        ELSE temp_ut_2 <= temp_ut;                           --hvis ikke brukes det videre
        END IF;

        IF I_ra(15) = '1' THEN O_ra <= 0 - temp_ut_2;        --Hvis MSB=1, dvs. negativt tall, tas 2'
        ELSE O_ra <= temp_ut_2;                              --hvis ikke brukes det videre
        END IF;                                              --I begge tilfelder sendes det til utgan

    END PROCESS;
END round_a_architecture;
```

Figur 4.4.22 Programmering av "round_a"_figur2

Denne arkitekturen gjelder da for alle de tre blokkene som heter "round_a".

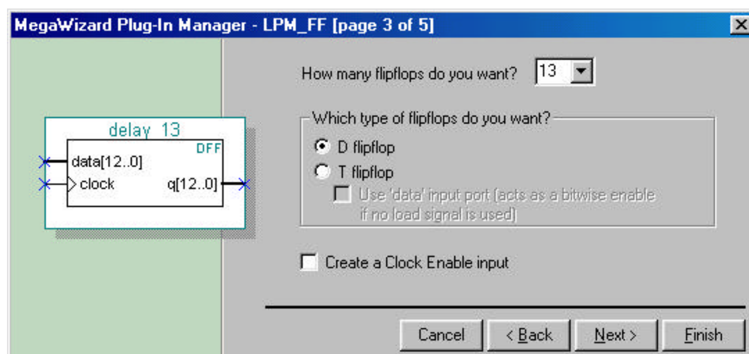
4.4.8. Opprettelse av tidsforsinkelses enhetene

Tidsforsinkelsene kalles "delay_13", "delay_12" og "delay_8", hvor dette henholdsvis er en 13 bits D-vippe, en 12 bits D-vippe og en 8 bits D-vippe. "delay_13" og "delay_8" har som hovedoppgave å skape en klokkepuls forsinkelse på signalet. I tillegg til dette vil vippene virke som en "pipelining", noe som gjør at frekvensen på signalet gjenom kretsen kan økes. "delay_12" skal sitte på hver av de to utgangene til filteret som en ren "pipelining", og vil ikke ha noen funksjonell innvirkning på filter responsen. Konstruksjonen av disse tidsforsinkelsene gjøres på samme måte som med multiplikasjons kretsene i avsnittet "konstruksjon av multiplikasjons kretsene". Det er likevel vist fort uten noen utfyllende forklaring hvordan det gjøres. Se først Figur 4.4.23.

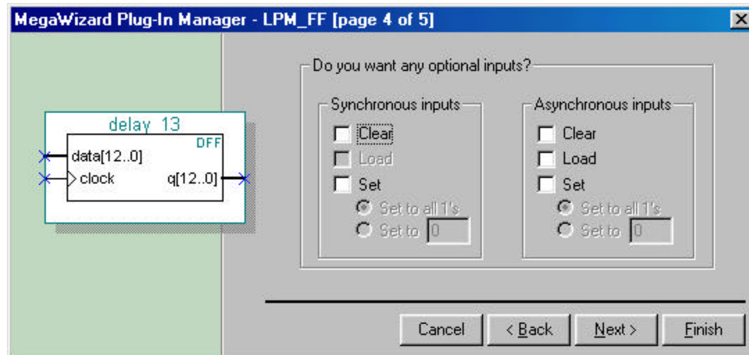


Figur 4.4.23 Opprettelse av "delay_13"_figur1

Bruker LPM_FF som er en standard Flip Flop. En kan se ut fra Figur 4.4.24 at det ble valgt å bruke en D-vippe, med i første omgang 13 bit.

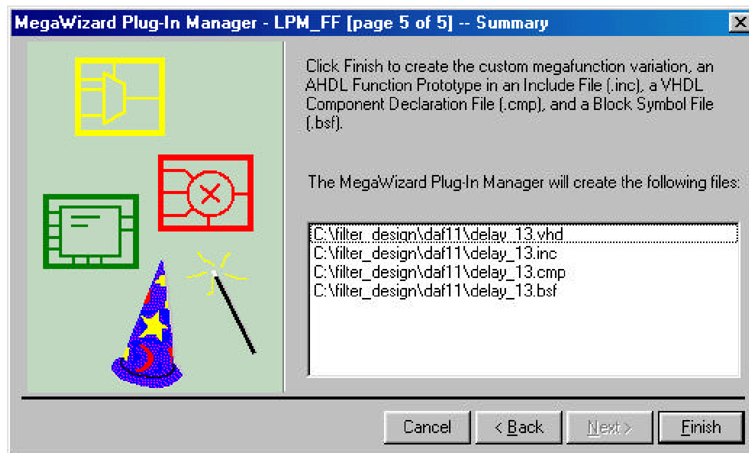


Figur 4.4.24 Opprettelse av "delay_13"_figur2



Figur 4.4.25 Opprettelse av "delay_13"_figur3

I denne oppgaven brukes ingen av mulighetene vist i Figur 4.4.25.

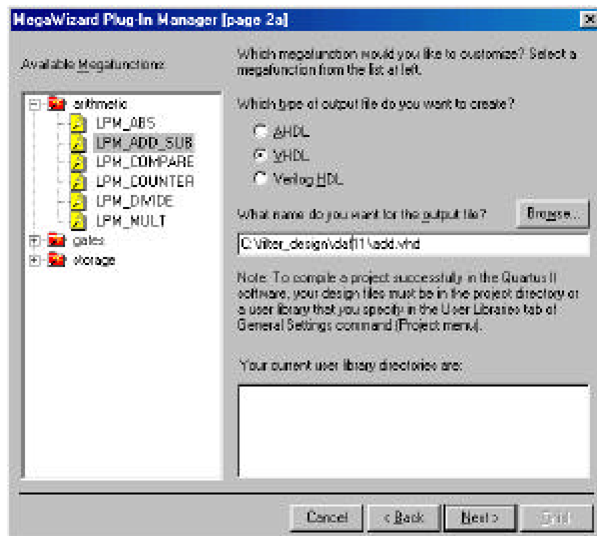


Figur 4.4.26 Opprettelse av "delay_13"_figur4

Klikk "finish" og "delay_13" er ferdig laget. For å lage "delay_12" og "delay_8" brukes samme fremgangsmåte.

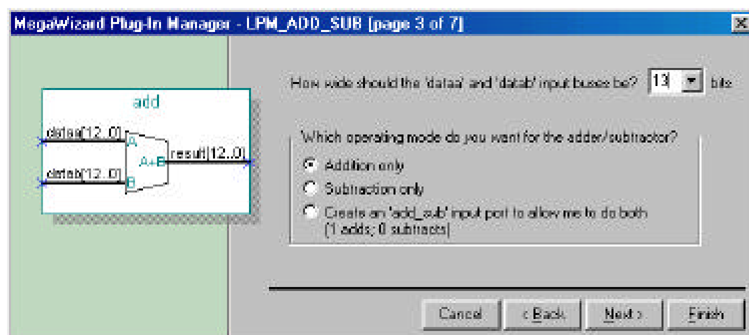
4.4.9. Opprettelse av addisjons kretser

Det neste som skal lages er rene 13 bits addisjonskretser, disse kalles "add". Konstruksjonen av disse addisjonskretsene gjøres på samme måte som med multiplikasjons kretsene og tidsforsinkelsene. Det er likevel vist fort uten noen utfyllende forklaring hvordan det gjøres. Se først Figur 4.4.27.



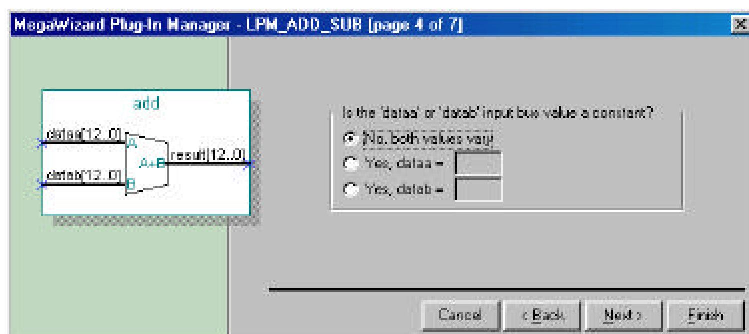
Figur 4.4.27 Opprettelse av "add"_figur1

Bruker LPM_ADD_SUB som er en standard addisjonskrets og/eller subtraksjons krets.



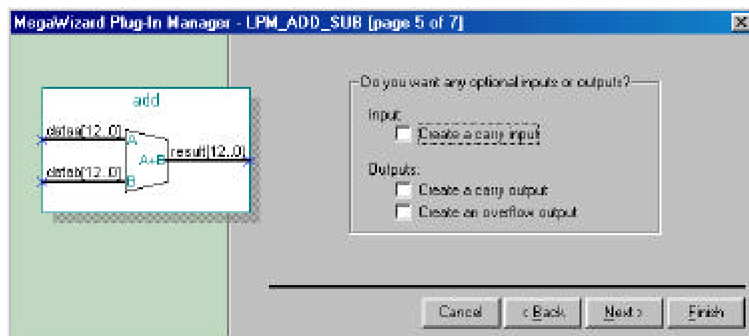
Figur 4.4.28 Opprettelse av "add"_figur2

En kan se ut fra Figur 4.4.28 at det ble valgt en ren addisjonskrets med 13 bits innganger.



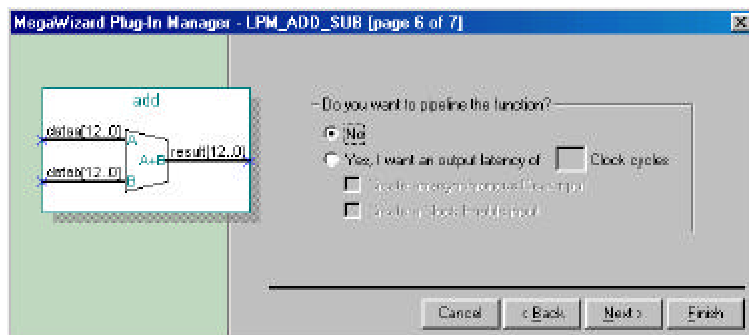
Figur 4.4.29 Opprettelse av "add"_figur3

I skjermbildet vist I Figur 4.4.29 kan det velges om noen av inngangene skal være konstanter.



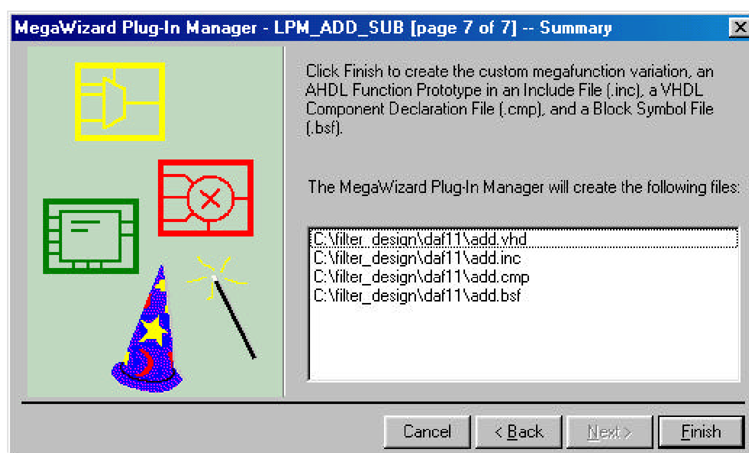
Figur 4.4.30 Opprettelse av "add"_figur4

Ingen av mulighetene vist i Figur 4.4.30 ble benyttet.



Figur 4.4.31 Opprettelse av "add"_figur5

Som vist I Figur 4.4.31 kan det velges om addisjons kretsen skal ha en klokke inngang, dette vil gi "pipelining". Dette benyttes ikke ettersom tidsforsinkelsene som kjent allerede gir tilstrekkelig "pipeling".



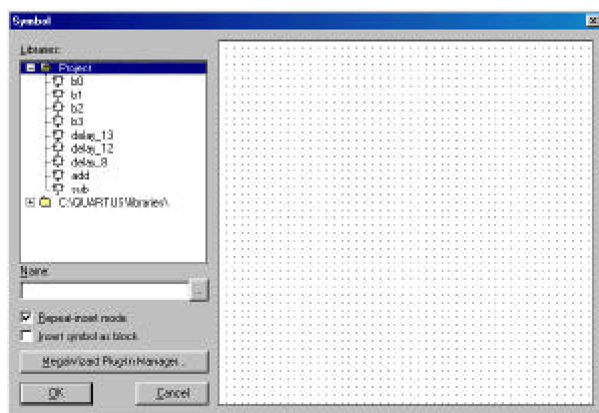
Figur 4.4.32 Opprettelse av "add"_figur6

Klikk "finish" og addisjons kretsene blir ferdig laget.

4.4.10. Konstruksjon av subtraksjons kretsene

Det trengs også noen addisjonskretser hvor den ene inngangen er negativ, dette blir da en subtraksjonskrets som kalles “sub”. Denne kretsen lages nesten identisk som “add”, bortsett fra at “ subtraction only” velges istedenfor “addition only” i skjermbildet vist i Figur 4.4.28.

Nå er det blitt opprettet tidsforsinkelser, multiplikasjons-, addisjons-, og subtraksjons enheter. Alle disse enhetene vil ligge listet opp under ”Project” mappen som vist i Figur 4.4.33.



Figur 4.4.33 Oversikt

4.4.11. Opprettelse av entiteten “round_b”

Det trengs en avrundings funksjon på utgangen av filteret for å runde av fra 13 til 12 bit på slutten av greinen. Denne entiteten har fått navnet “round_b”. Denne lages på akkurat samme måte som “round_a”, og funksjonen er den samme bare med et annet antall bit. Opprettelsen er derfor ikke vist her, da avsnittet “opprettelse av “round_a” kan brukes som en hjelp. Program koden er vist i vedlegg E.

4.4.12. Opprettelse av entiteten “round_c”

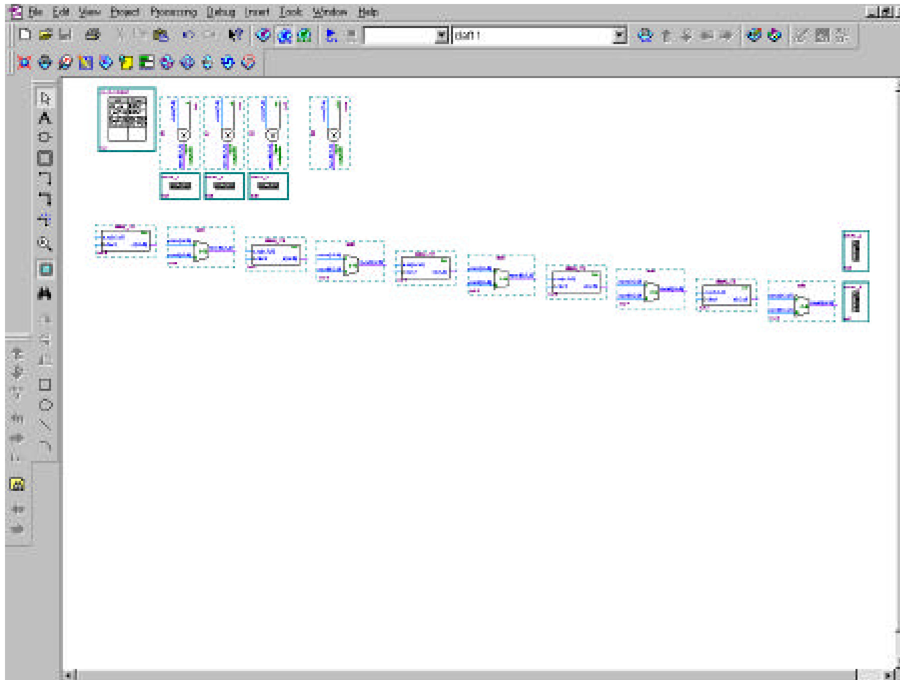
Det trengs også en egen avrundings funksjon etter multiplikasjonen i den imaginære greinen, siden det her avrundes fra 16 til 12 bit direkte. Denne entiteten har fått navnet “round_c”, og lages på akkurat samme måte som “round_a” og “round_b”. Opprettelsen er derfor ikke vist her, da avsnittet “opprettelse av “round_a” kan brukes som en hjelp. Program koden er vist i vedlegg E.

4.4.13. Innsetting av “delay_13”, “delay_12”, “delay_8”, “add” og “sub” I BDF filen

Disse kretsene kan da settes inn på akkurat samme måte som multiplikasjons kretsene vist tidligere, ved å bruke meny valget “insert->symbol”. Disse kretsene innsatt i BDF file er vist i Figur 4.4.34.

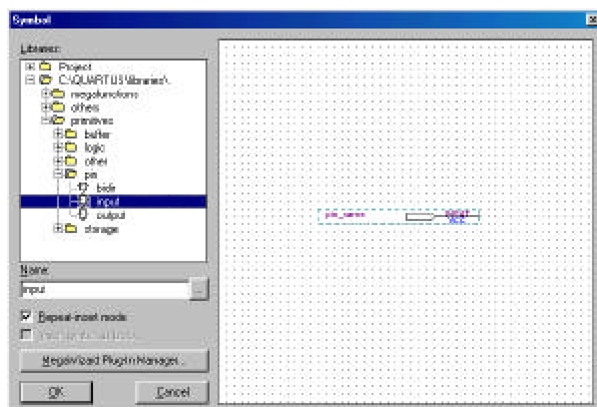
4.4.14. Innsetting avI/O pinner

Etter at alle enheter i filteret er satt inn, ser BDF filen ut som på Figur 4.4.34.
For å få med hele designet på et skjermbilde kan man zoome ut ved å trykke "Clt+Shift+Space" og zoome inn ved å trykke "Clt+Space".



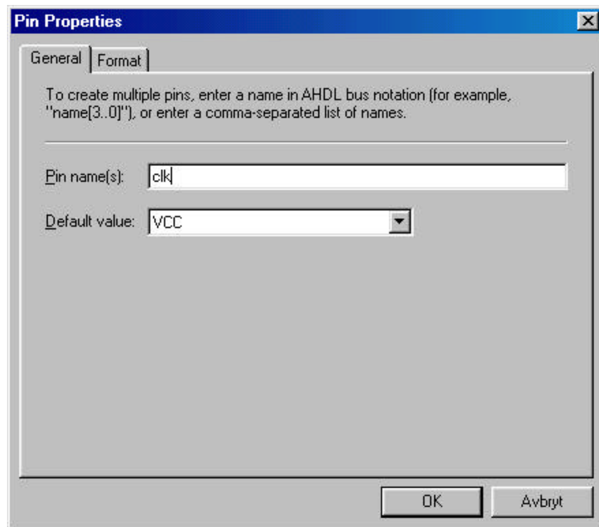
Figur 4.4.34 Innsetting i BDF filen_figur2

Det neste som gjøres er å legge til inn- og utgangs pinner. Dette gjøres ved å klikke på "Symbol tool"(nr 3 fra toppen) fra "block & symbol editors" og velge "c:\QUARTUS\libraries\primitives\pin\ og input eller output som vist på Figur 4.4.35.



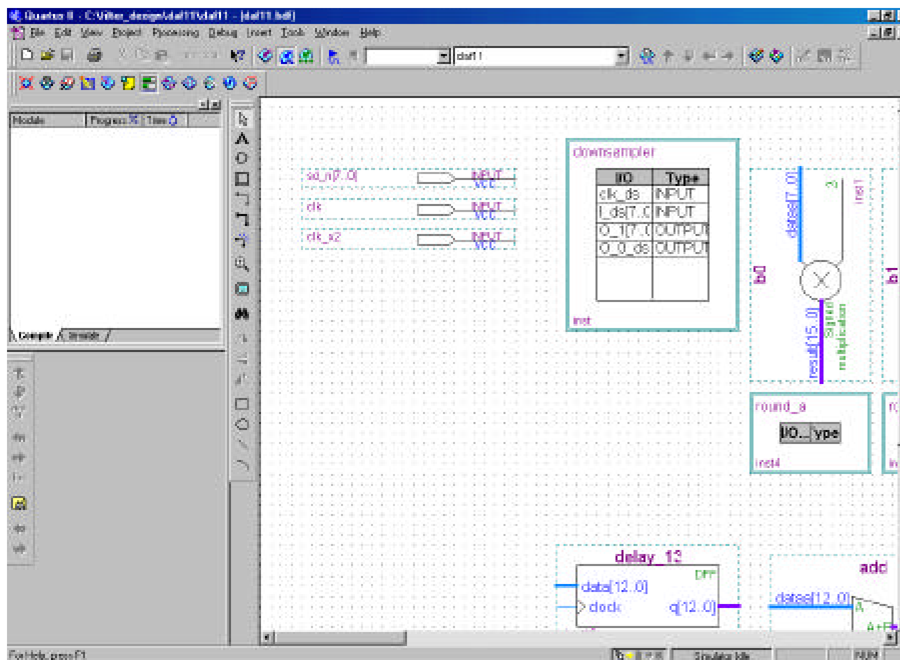
Figur 4.4.35 Innsetting av I/O_figur1

Inn- og utgangs pinnene kan plasseres rett ut i BDF filen, og gis navn ved å dobbeltklikke på dem. En ny meny vist på Figur 4.4.36 dukker opp, her skrives navnet inn.



Figur 4.4.36 Innsetting av I/O_figur2

I denne oppgaven skal det lages tre innganger med navnene "clk", "clk_x2" og "sd_n[7..0], og to utganger med navnene "xr[11..0]" og "xi[11..0]. Inngangene er vist på Figur 4.4.37.

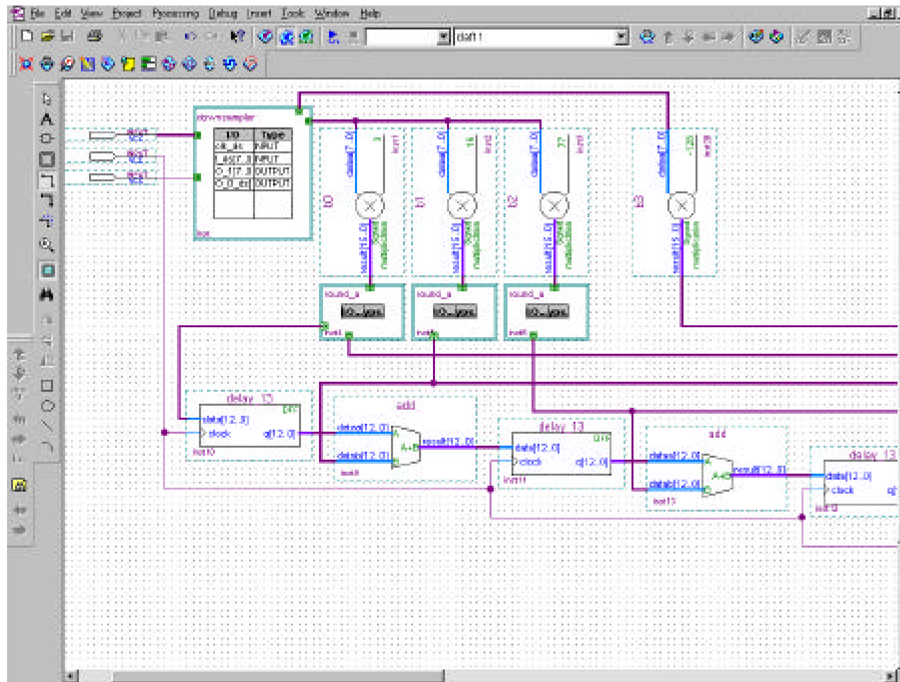


Figur 4.4.37 Innsetting av I/O_figur3

4.4.15. Trekking av forbindelser

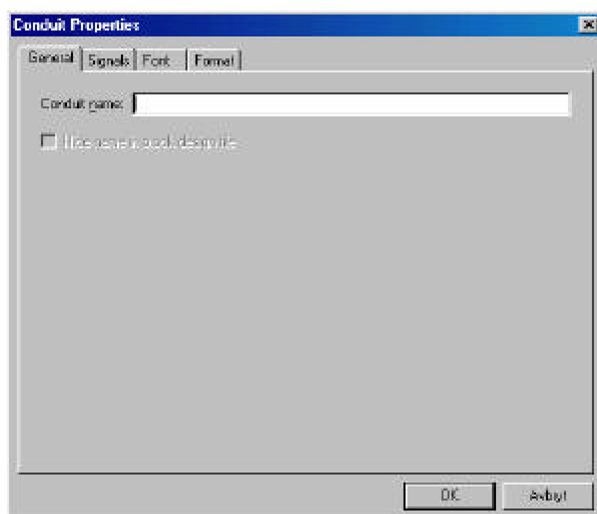
Etter at alle entiteter, kretser og I/O pinner er lagt inn i BDF filen, er neste trinn å trekke forbindelser mellom dem. Det brukes valgene "Orthogonal bus tool"(nr 6 fra toppen) fra "block & symbol editors" for å trekke opp busser, og "Orthogonal node tool"(nr 5 fra toppen) fra "block & symbol editors" for å trekke opp enkle forbindelser. Det man gjør er ganske

enkelt å klikke på den ene termineringen med mus knappen, flytte musen til den andre termineringen og klikke på nytt.
Figur 4.4.38 viser deler av BDF filen etter at forbindelsene er trekt opp.



Figur 4.4.38 Trekking av forbindelser_figur1

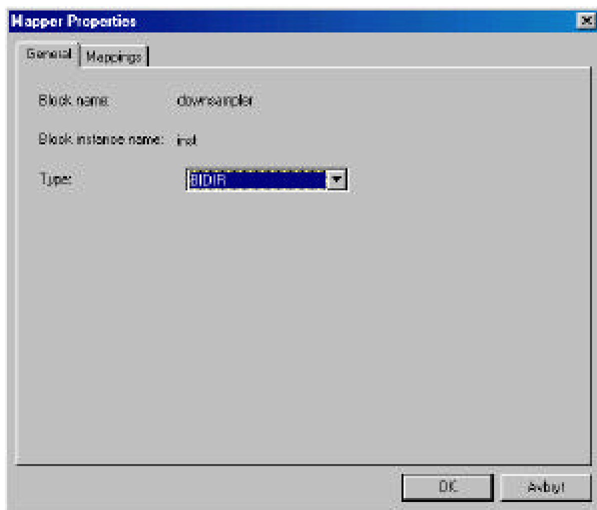
Etter at forbindelsene er trekt opp og terminert, er det best å gi de forskjellige bussene navn for å unngå problemer med at signaler "smitter" over fra en buss til en annen. Bussene gis navn ved å skifte til "Selection tool"(nr 1 fra toppen) fra "block & symbol editors" og dobbeltklikke på den aktuell bussen. Et skjermbilde som vist på Figur 4.4.39 dukker da opp.



Figur 4.4.39 Trekking av forbindelser_figur2

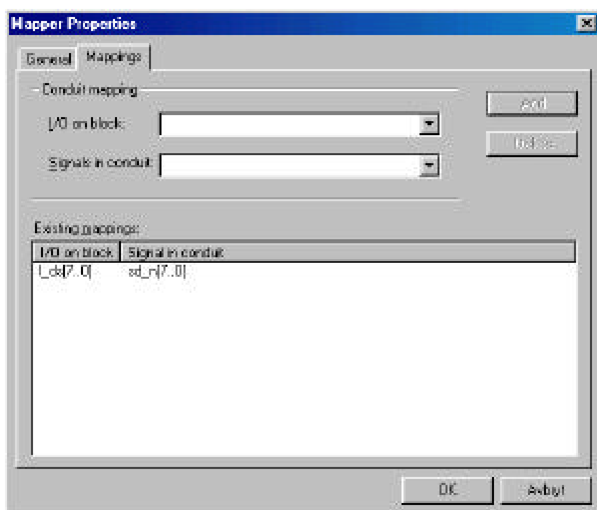
Under "General" i ruten "Conduit name" gis bussen et vilkårlig navn.

Nå må tilkoblingspunktene der forbindelsene er terminert konfigureres. Bruk fortsatt valget "Selection tool"(nr 1 fra toppen) fra "block & symbol editors" og dobbeltklikk på det aktuelle tilkoblingspunktet. Dette gjelder ikke for tilkoblingspunkter på kretser som er lagt inn i BDF filen kun som symboler("h0, h1, h2, delay_13, delay_12, delay_8, add og sub), siden det her er helt entydig hvilken I/O som kobles til den aktuelle forbindelsen. Når man dobbelt klikker på det aktuelle tilkoblings punkteet, dukker et skjermbilde som vist på Figur 4.4.40 Opp, det er her vist et eksempel med den øverste inngangen på "downsampler".



Figur 4.4.40 Trekking av forbindelser_figur3

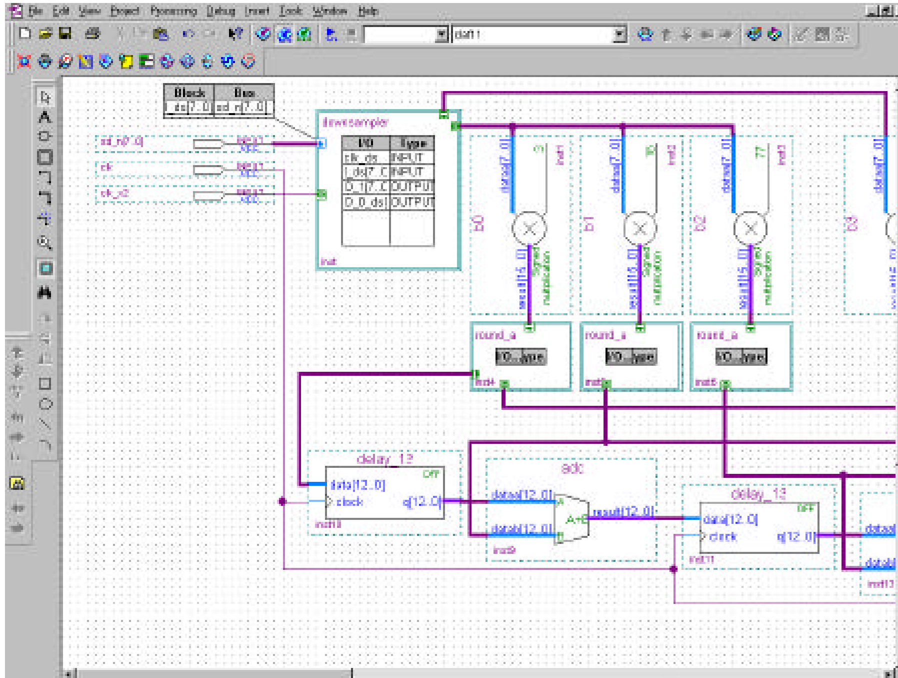
Under menyvalget "Type" på menyen "General" kan det velges om forbindelsen skal kobles til en inngang, utgang eller bidireksjonal port, som vist på Figur 4.4.40. Den øverste forbindelsen på "downsampler" skal være en inngang.



Figur 4.4.41 Trekking av forbindelser_figur4

Under menyen "Mappings" bestemmes ved menyvalget "I/O on block" hvilken inngang, utgang eller bidireksjonal port på blokken som skal kobles til den aktuelle forbindelsen, mens

det ved menyvalget “Signals in conduit” bestemmes hvilket signal på bussen som skal kobles til denne aktuelle inngangen, utgangen eller bidireksjonale porten. I dette eksemplet velges det som vist i Figur 4.4.41 at inngangen “I_ds[7..0]” kobles til signalet “sd_n[7..0]”.
Etter at valget er gjort, klikker man “ok” og koblingen indikeres med en merkelapp som vist på Figur 4.4.42.

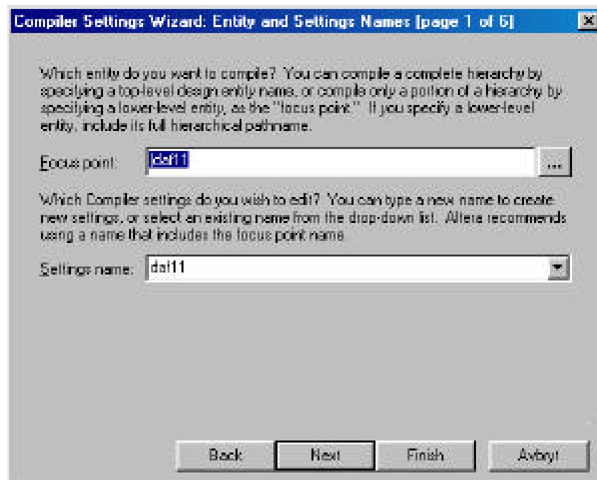


Figur 4.4.42 Trekking av forbindelser_figur5

Etter at alle forbindelser er trekt, er designet klar til å kompileres.

4.4.16. Kompilering

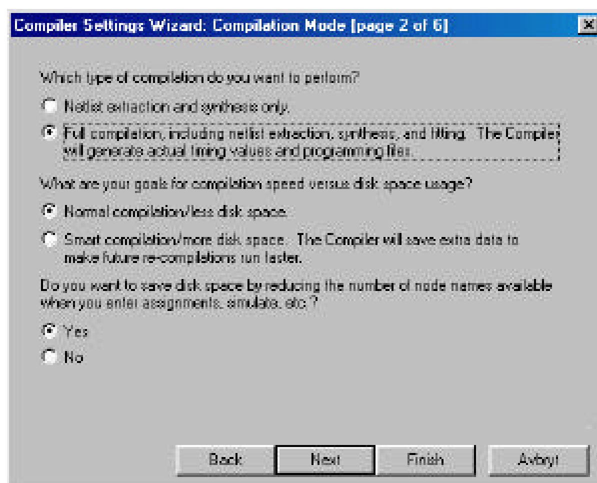
Det første man må gjøre er å skifte til ”compile mode”, dette gjøres ved å klikke på menyvalget ”processing->compile mode”. Det er en del parameter som skal settes opp før en kompilering, slik at det anbefales å følge en ”compiler settings wizard” som hjelper en gjennom prosessen. For å starte denne guiden klikker man på menyvalget ”processing->compiler settings wizard”, og det første skjermbilde dukker opp. Dette er kun en introduksjon, så her klikker man bare ”next”. Skjermbilde vist på Figur 4.4.43 dukker opp.



Figur 4.4.43 Kompilering_figur1

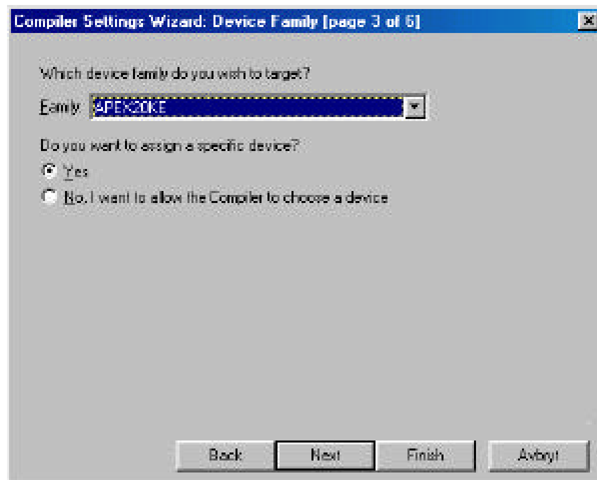
Her bestemmer man hvilken fil som skal kompileres, og hvilket komilerings oppsett som skal redigeres eller eventuelt opprettes. Velger i dette tilfellet "daf11" i begge rutene.

Skjermbilde vist i Figur 4.4.44 er da det neste. Ingen av disse valgene forandres i denne oppgaven.



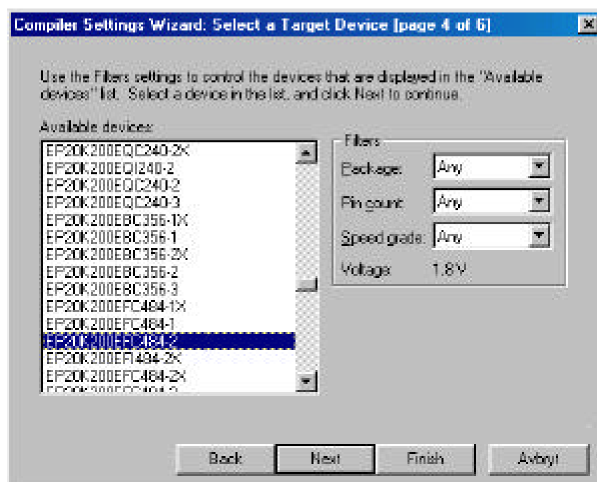
Figur 4.4.44 Kompilering_figur2

I det neste bildet, vist i Figur 4.4.45, må man velge hvilken type brikke man vil kompilere designet mot. Velg her den typen brikke som skal brukes(skal NIOS simuleringskortet brukes, velges "APEX20KE").



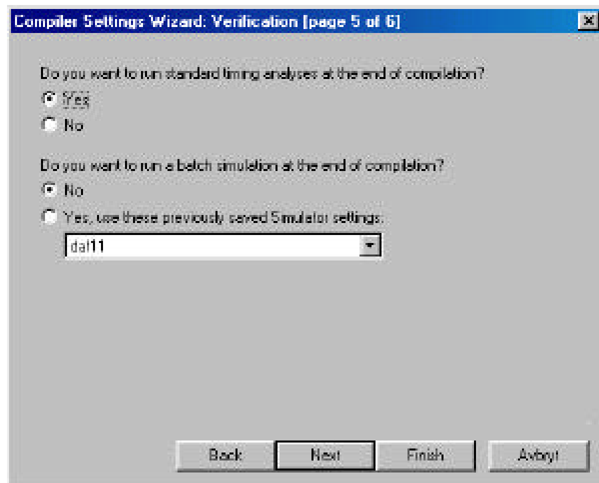
Figur 4.4.45 Kompilering_figur3

I det neste bildet, vist på Figur 4.4.46, må man velge eksakt hvilken brikke man vil kompilere mot. Velg her den brikken som skal brukes(skal NIOS simuleringskortet brukes, velges "EP20K200EFC484-2").



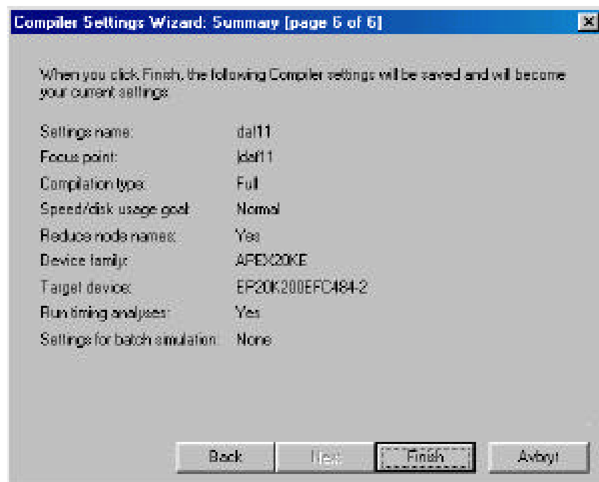
Figur 4.4.46 Kompilering_figur4

Klikk "next" og bildet i Figur 4.4.47 dukker opp.



Figur 4.4.47 Kompilering_figur5

Forandrer ingen av disse valgene i denne oppgaven. Klikk "next" og en oversikt over oppsettet vises. Klikk "finish", og oppsettet er fullført.



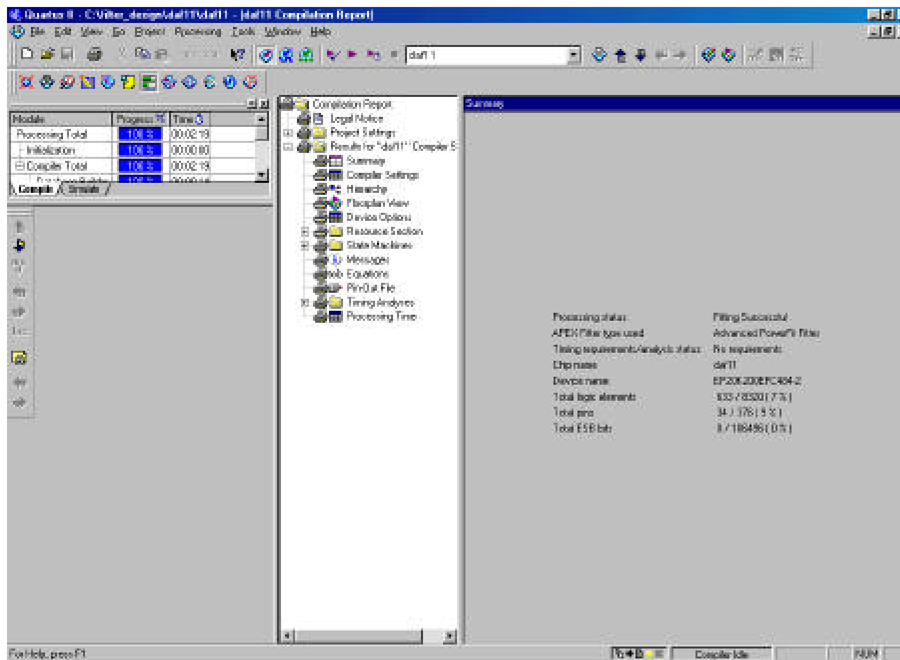
Figur 4.4.48 Kompilering_figur6

Man kan nå starte kompileringen ved å klikke på meny valget "processing->start compilation", og designet vil forhåpentligvis kompileres uten feil. Etter kompileringen dukker det opp en "compilation repport" på skjermen, samt en status over kompileringen. Dette er vist på Figur 4.4.49 nedenfor. Skulle det dukke opp feil under kompileringen, får man et feilmeldings vindu opp på skjermen, under "compilation repport->results for "daf11" compiler settings->messages". Disse meldingene skal være rimelig greie å tyde. En eventuell feilretting må da utføres, og en ny kompilering må foretas.

Under dette prosjektet ble designet compilert flere ganger under oppbyggingen av filteret, for å sikre at det man gjorde hele veien kunne kompileres. Dette ble gjort fordi QuartusII var veldig ustabilt på den maskinen det ble jobbet på, og fordi programmet var nytt og ukjent. Etter at hele designet var ferdig, oppstod det store problemer. Det oppstod blant annet problemer ved å bruke en entitet flere ganger i designet. Dette skal egentlig ikke være noe

problem, da dette nettopp er poenget med entiteter. Det oppstod også store problemer med at signaler "smittet" over mellom busser, selv om alle bussene var gitt entydige navn. Problemene ble til slutt så store at det ble valgt å sette sammen top filen "manuelt" som en ren VHDL kode.

Program koden for topp filen er vist i vedlegg E.

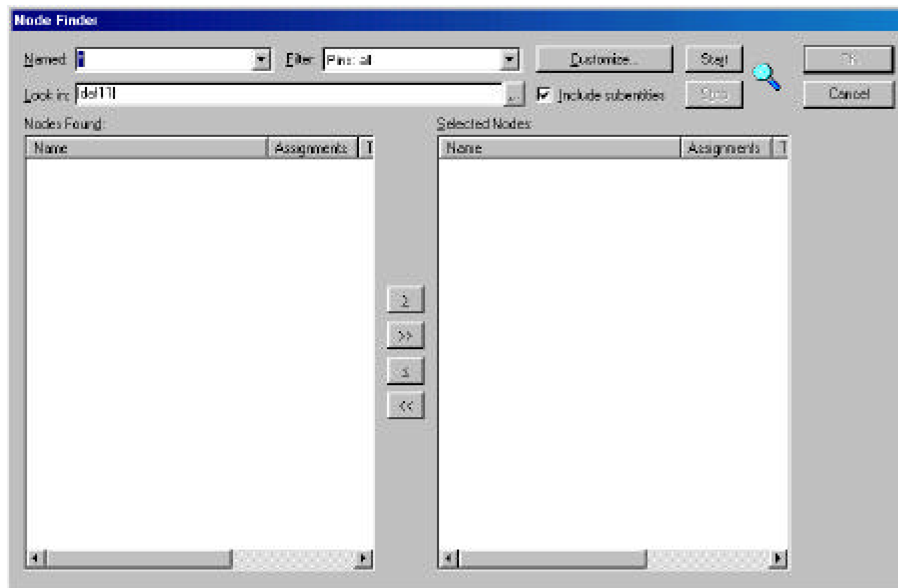


Figur 4.4.49 Kompilering_figur7

Når topp filen er laget som en BDF-fil, kan dens VHDL kode genereres ved å gå på menyvalget "Tools->Create HDL Design file for current file->VHDL".

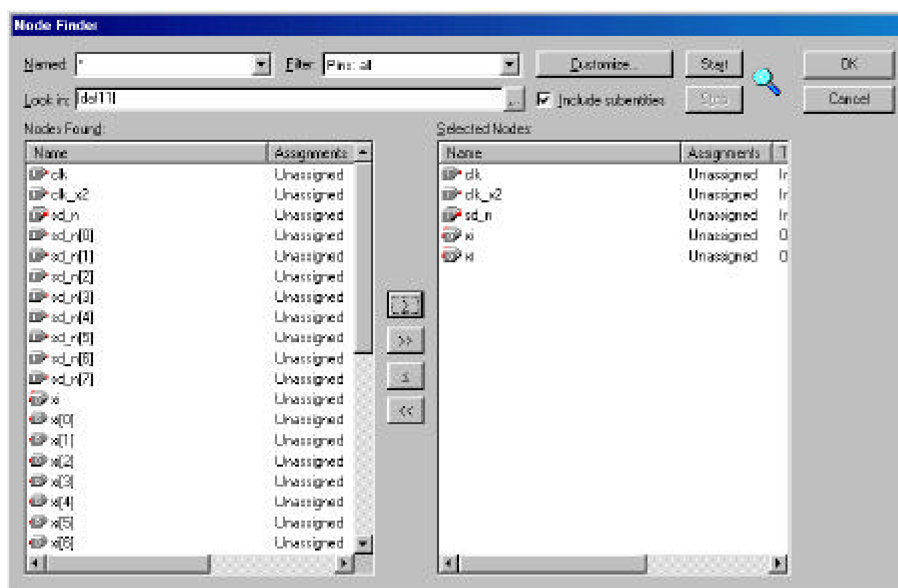
4.4.17. Simulering

Det første man må gjøre er å lage en "vector waveform file". Dette gjøres ved å klikke på menyvalget "file->new->other files->vector waveform file", og lagre denne i prosjektmappen, med samme navn som prosjektet. I dette tilfellet c:\filter_design\daf11\daf11.vwf. For å legge til inn- og utganger, klikker man på menyvalget "insert->node or bus", og "node finder" på skjermbilde som dukker opp. Man vil da få opp et skjermbilde som vist på Figur 4.4.50.



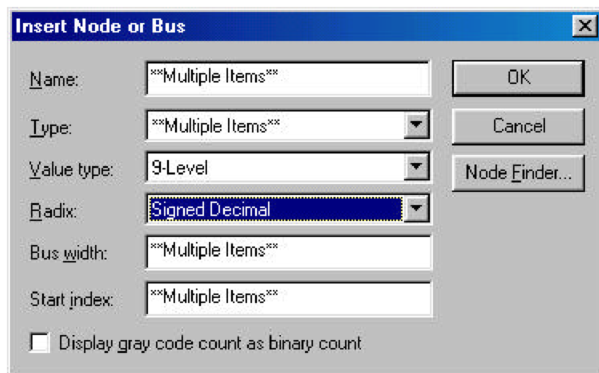
Figur 4.4.50 Simulering_figur1

Her velger man "Pins:all" for alle pinner i ruten "filter", og "|daf11|" i ruten "look in". Alle pinner listes da opp, disse overføres ved å merke dem og klikke på "≥" knappen. Når alle ønskede pinner er lagt over i det høyre feltet vist på Figur 4.4.51, klikker man "ok".



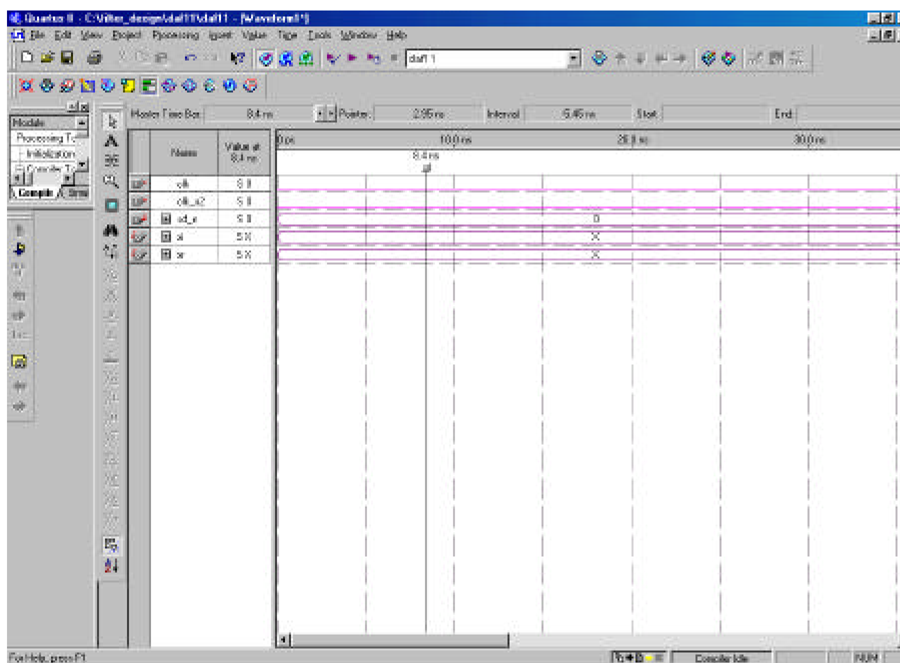
Figur 4.4.51 Simulering_figur2

Figur 4.4.52 vil dukke opp, her velger man hvilket format ingangene skal representeres på. Under valget "radix" kan det være greit å velge "signed decimal" for å få verdiene presentert på negativ og positiv desimal form.



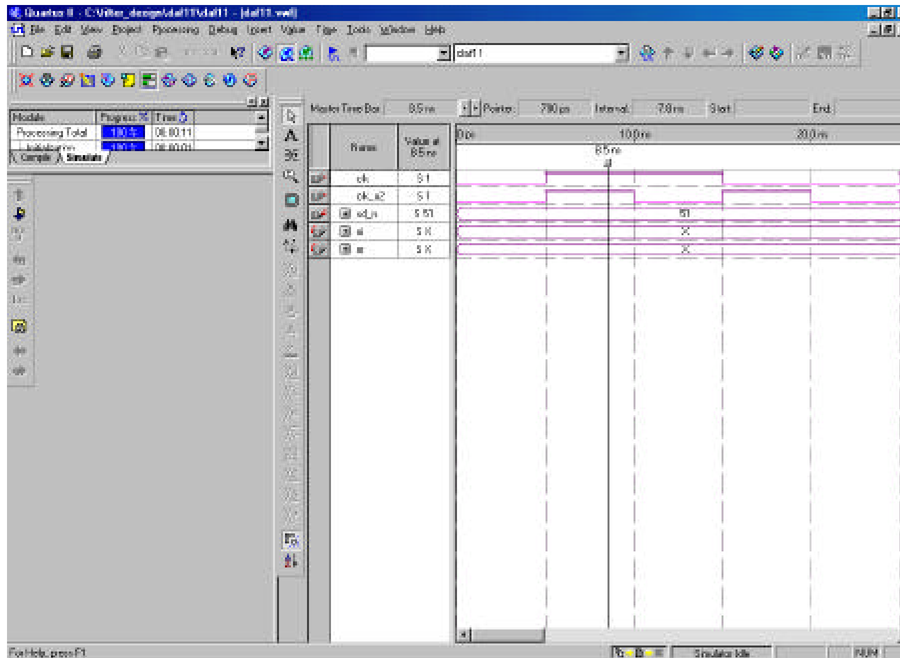
Figur 4.4.52 Simulering_figur3

Klikk "ok" og simuleringsfilen vil se ut som på Figur 4.4.53.



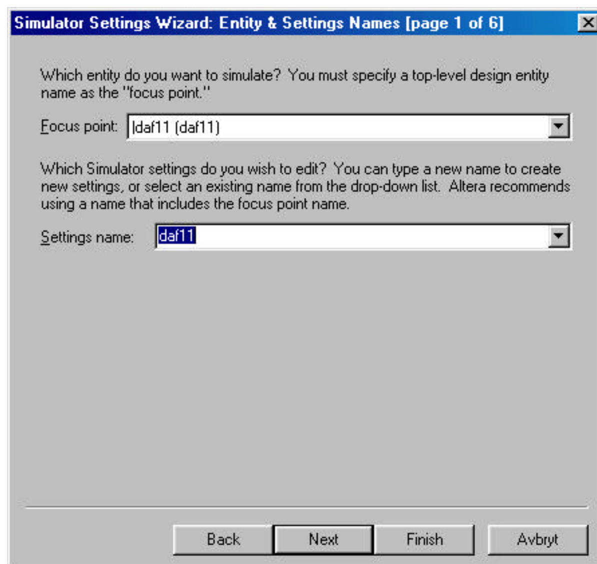
Figur 4.4.53 Simulering_figur4

Man kan sette de to klokkene ved å klikke på dem slik at de blir merket blå, gå på menyvalget "value->clock" og sette periode tiden. I dette tilfellet skal "clk_x2" ha den dobble klokkefrekvensen av "clk". Inngangen "sd_n[7..0] kan gis verdier ved å merke et område slik at det blir blått, dobbeltklikke og skrive inn verdien. Etter at dette er gjort ser simuleringsfilen se ut som i Figur 4.4.54, dersom "clk" = 50MHz, "clk_x2" = 100MHz og "sd_n[7..0]" er et step med en tilfeldig amplitude lik 51.



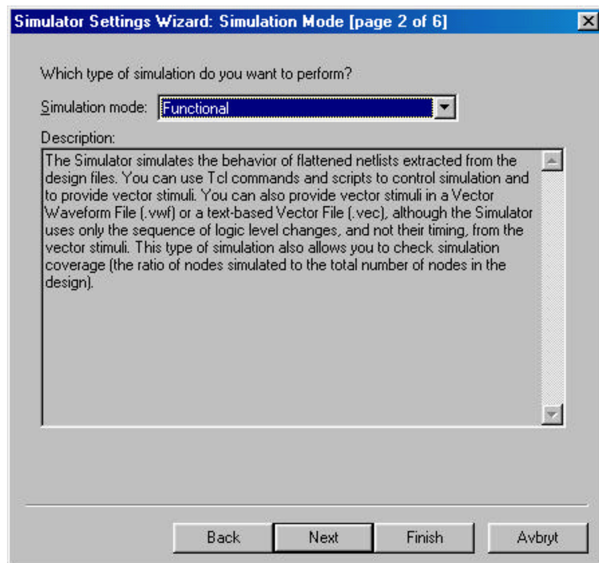
Figur 4.4.54 Simulering_figur5

Det neste man må gjøre er å skifte til ”simulate mode”, dette gjøres ved å klikke på menyvalget ”processing->simulate mode”. Det er en del parameter som skal settes opp før en simulering, det anbefales å følge en ”simulator settings wizard” som hjelper en gjennom prosessen. For å starte denne guiden klikker man på menyvalget ”processing->simulator settings wizard”, og det første skjermbilde dukker opp. Dette er bare en introduksjon, så her klikker man bare ”next”. Skjermbilde som vist på Figur 4.4.55 Dukker da opp.



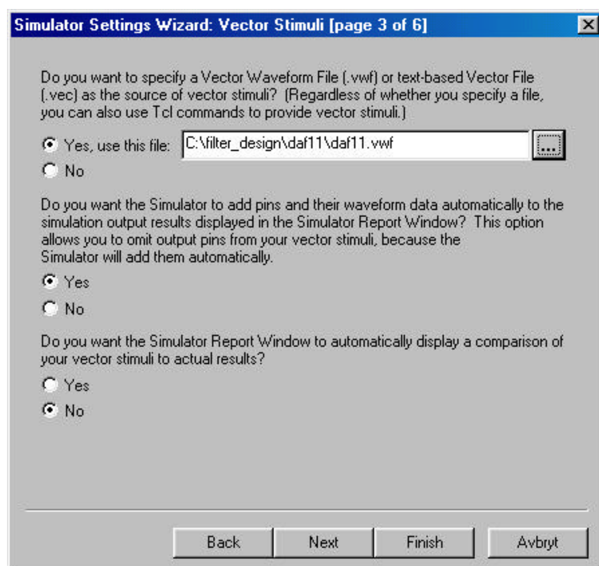
Figur 4.4.55 Simulering_figur6

Her bestemmer man hvilken fil som skal simuleres, og hvilket simulerings oppsett som skal redigeres eller eventuelt opprettes. Velg i dette tilfellet "daf11" i begge rutene, og klikk "next".



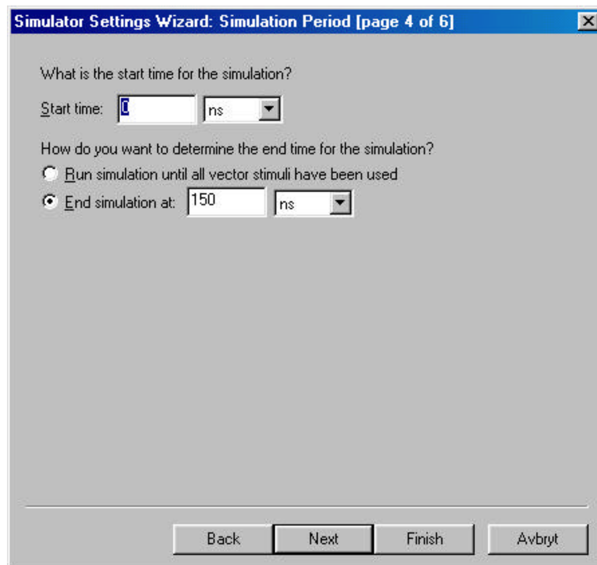
Figur 4.4.56 Simulering_figur7

Det neste valget som dukker opp er vist i Figur 4.4.56. Her kan man velge om man vil foreta en "functional" eller en "timing" analyse. "functional" analysen gir en ren funksjonell simulering, uten å ta hensyn til glitcher, setup-tider, hold-tider og andre kritiske faktorer som spiller inn på designet. Det kan være gunstig å kjøre denne typen simulering først, for å se om man har konstruert designet logisk riktig. Gir ikke den funksjonelle simuleringen forventet resultat, er det ingen vits å gå videre før man har rettet opp feilen i designet.
"timing" analysen tar i motsettning til "functional" hensyn til ovenfornevnte kritiske faktorer, og gir en mye mer reell simulering. Velg i første omgang "functional" og klikk "next".



Figur 4.4.57 Simulering_figur8

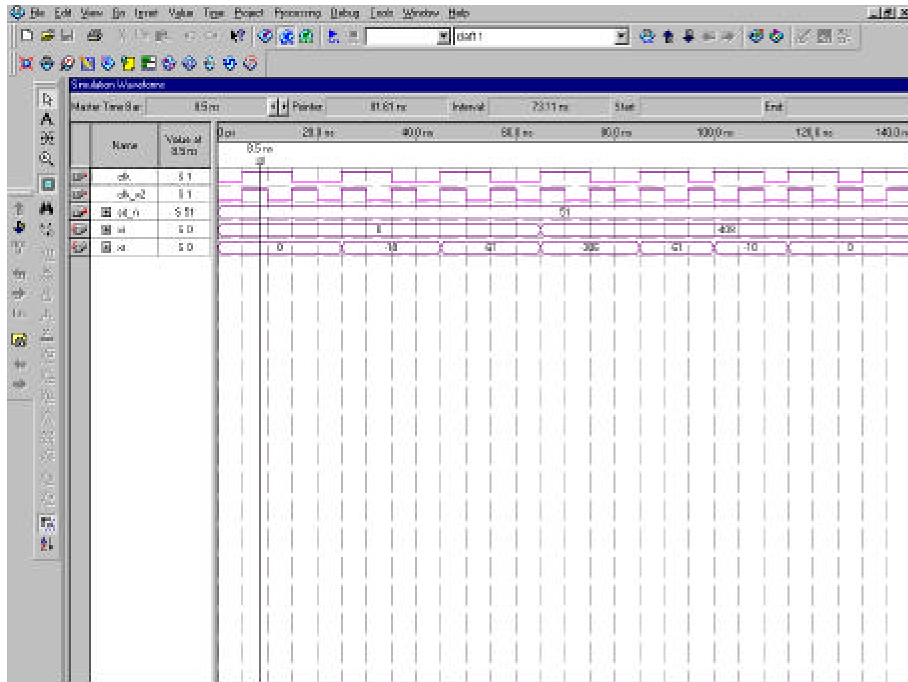
Øverst i Figur 4.4.57 velges ”daf11.vwf”. Klikk ”next” og skjermbilde i Figur 4.4.58 dukker opp.



Figur 4.4.58 Simulering_figur9

Her setter man start- og stopp tider for simuleringen. Det er helt greit å starte på 0ns, og stopp tiden settes etter hvor langt det har blitt lagt inn verdier i simulerings filen. I dette tilfellet er denne 150ns. På det neste skjermbilde klikkes bare “next”, og en oppsummering av simulerings oppsettet dukker opp. Her klikkes bare “finish”, og det er klart for simulering.

Simuleringen startes ved å klikke på menuvalget “processing->run simulation”, og resultatet blir som vist i Figur 4.4.59.



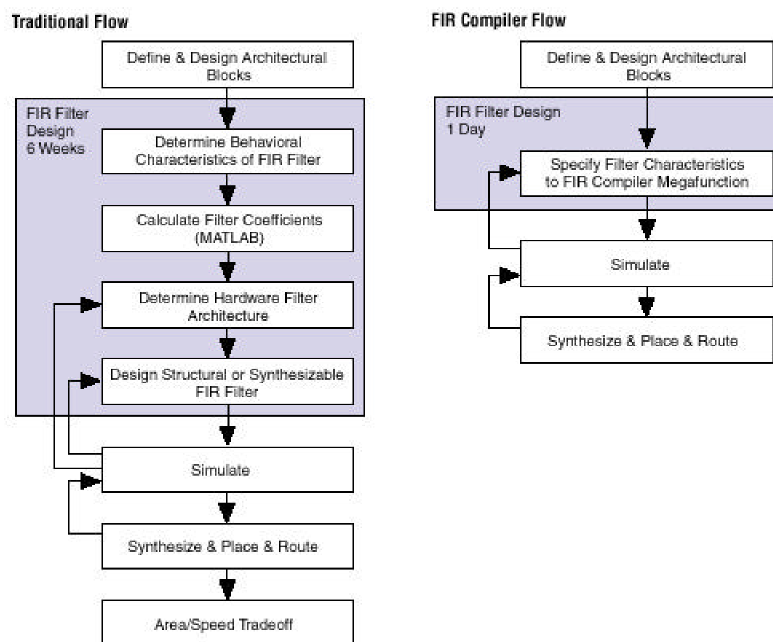
Figur 4.4.59 Simulering_figur10

4.4.18. Megacore funksjonen ”FIR compiler”

Alteras DSP løsninger inneholder ”MegaCore” funksjoner som er utviklet og støttes av altera, og ”Altera Megafunction Partners Program” funksjoner. I tillegg til dette finnes mye brukte funksjoner som multiplikatorer og adderere i industri standard biblioteket ”Library of Parameterizied Modules (LPM).

Alteras ”FIR compiler” er en ”MegaCore” funksjon, som er et hjelpemiddel for å realisere FIR filtre på en PLD. Det ble i dette prosjektet sett nærmere på ”FIR compiler” for å bestemme om denne funksjonen er egnet til å bruke ved oppbyggingen av filteret.

Man kan bruke ”FIR compiler” til å finne filterkoeffisientene, eller man kan finne koeffisientene på en annen måte(f.eks. i Matlab) lagre disse som en ASCII fil og importere dem inn i ”FIR compiler.
Figur 4.4.60 er hentet fra bruker manualen til ”FIR compiler” og viser et eksempel på hvordan en prosess for å konstruere et filter kan gjennomføres.



Figur 4.4.60 FIR design

”FIR compiler” kan generere filter koeffisienter for følgende filter:

- Lavpass og høypass
- Bandpass og båndstopp
- Raised cosinus og root raised cosinus

Når ”FIR compiler” benyttes for å finne filter koeffisientene, kan brukeren bestemme antall koeffisienter, knekkfrekvens, samplings rate, filter type og hvilken vindus metode som skal benyttes for å finne filter koeffisientene.

Filter koeffisientene kan være flyttall eller heltall, både som ”signed” og ”unsigned”. Utsignalet fra filteret kan representeres med et fullt antall bit, eller det kan trunkeres eller avrundes.

”FIR compiler” undersøker filter koeffisientene og detekterer automatisk filterets symmetri. En slik symmetri kan være even symmetri, odd symmetri eller ingen symmetri i det hele tatt. Etter dette finner ”FIR compiler” den mest optimale algoritmen for å minimalisere antall operasjoner.

FIR compiler kan generere tre forskjellige FIR strukturer:

- Parallell, som er optimalt for hastighet.
- Seriell, som er optimalt for plassbesparelse.
- Variabel, som er mer fleksibelt.

Man kan velge om man vil ”pipeline” filteret konstruksjonen eller ikke.

Det finnes også muligheter for ”interpolation” og ”decimation”.

Når designet kompileres genereres Matlab Simulink kompatible simulerings modeller for system verifikasjon.

Det ble i denne oppgaven bestemt å ikke bruke ”FIR compiler”, men å bygge opp designet med standard LPM megafunksjoner.

5. Resultater

For å kontrollere at designet fungerer som det skal, ble det laget en funksjon i Matlab med navnet ”test_daf.m”. Denne funksjonen er bygd opp på følgende måte:
Den deler koeffisientevektoren i formel 4.3-16(med alle koeffisienter invertert) i to vektorer, hvor disse representerer to sub filtre. Like koeffisientene vil da bli den reelle greinen, og odde koeffisientene vil bli den imaginære. Funksjonen tar inn en tallvektor, og fordeler denne i to vektorer(downsampling). Den reelle greinen foldes med den første tall vektoren, her rundes det av til 13 bit etter hver multiplikasjon i foldningen (egen foldnings funksjon). Den reelle greinen rundes også av til 12 bit til slutt. Den imaginære koeffisient vektoren foldes med den andre tallvektoren, og disse verdiene rundes av direkte til 12 bit. Funksjonen plotter ut signalet i hver av de to greinene, og gir ut tallverdiene på utgangen.
”test_daf.m” gir ut de verdier som bør forventes ved en realisering av filteret.
Hele Matlab koden for ”test_daf.m” er vist i vedlegg C.

5.1. Software simulering - funksjonell

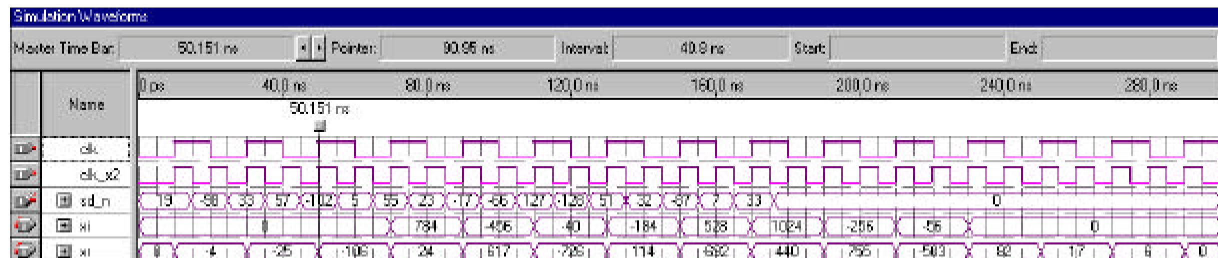
Det ble først foretatt en ren funksjonell simulering, det vil si en ren ideell simulering som ikke tar hensyn til glitcher, setup-tider, hold-tider og andre kritiske faktorer som spiller inn ved implementering på en brikke. Dette gjøres for å teste at designet fungerer logisk riktig.
Dette ble gjort ved først å påtrykke en tallvektor på ”daf11” i QuartusII(jmf. Avsnitt 4.4.17), med en tilfeldig valgt klokkefrekvens lik 100MHZ. De samme tallene ble påtrykt ”test_daf.m” for å finne ut om designet gir ut korrekte verdier.
Verdiene ble samlet i Tabell 5.1-1.

Den påtrykte tallvektoren er følgende: [19, –98, 33, 57, –102, 5, 55, 23, –17, –66, 127, –128, 51, 32, –87, 7, 33].

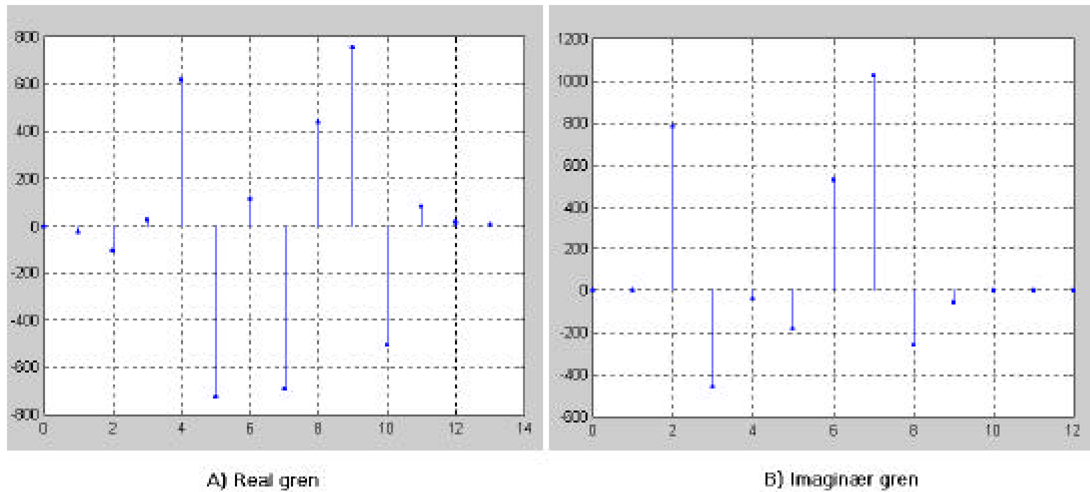
Matlab xr	-4	-25	-106	24	617	-726	114	-692	440	755	-503	-82	17	6	0
Matlab xi	0	0	0	784	-456	-40	-184	528	1024	-256	-56	0	0	0	0
Quartus xr	-4	-25	-106	24	617	-726	114	-692	440	755	-503	-82	17	6	0
Quartus xi	0	0	0	784	-456	-40	-184	528	1024	-256	-56	0	0	0	0

Tabell 5.1-1 Måle resultater

Simuleringen i Quartus er vist i Figur 5.1.1, og plottene fra ”test_daf.m” er vist i Figur 5.1.2.



Figur 5.1.1 Quartus funksjonell simulering

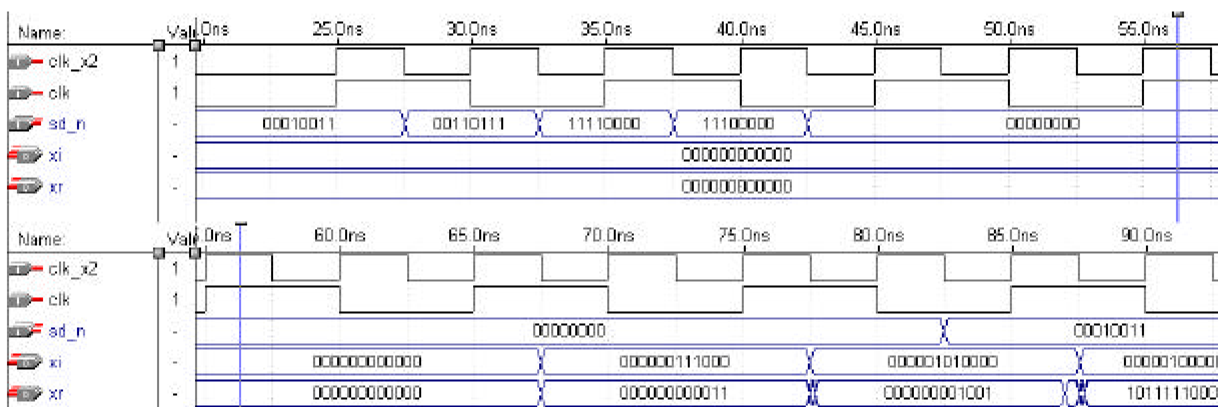


Figur 5.1.2 Matlab simulering

5.2. Software simulering – timing

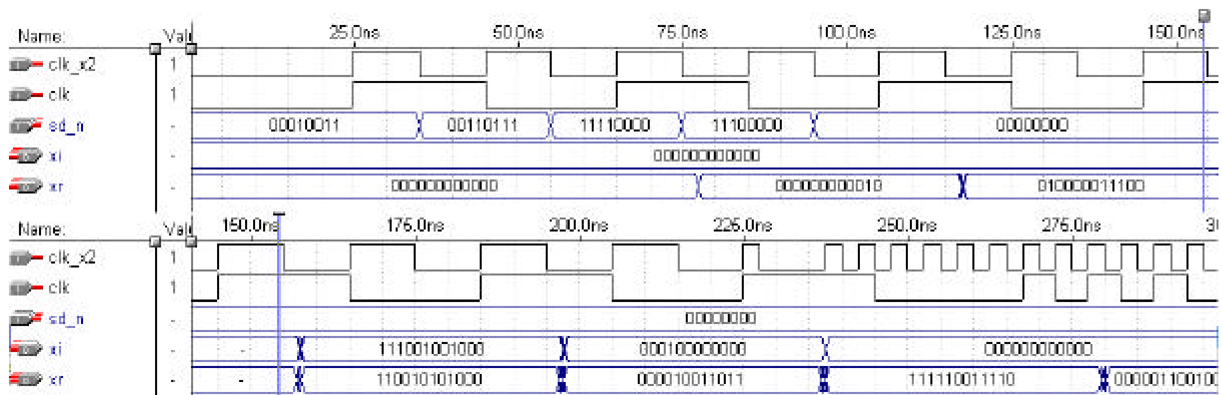
Det ble deretter forøkt med en timing simulering i QuartusII, men ustabilitet i softwaren førte til store problemer. Det var ikke mulig å få hentet ut noen simuleringer, da programmet låste seg ved hvert eneste forsøk.
En student versjon av ”Max+PlusII” fra Altera ble derfor brukt til denne timing simuleringen. Denne softwaren har bare støtte for to brikker, her ble den største valgt(FLEX10k).
Simuleringen i Max+PlusII viser ikke negative verdier på desimal form, det kan heller ikke skrives inn slike tall i simuleringsfilen. Det er derfor veldig tungvint å kontrollere resultatene, det ble derfor påtrykt kun noen få verdier(fungerer det for disse er det sannsynlig at det fungerer for andre verdier).

Noen tilfeldige tall(19d, 55d, -16d,-32d) ble påtrykt, og klokkefrekvensen(clk_x2) ble i første omgang satt til 200MHZ. Resultatet ble som vist i Figur 5.2.1.



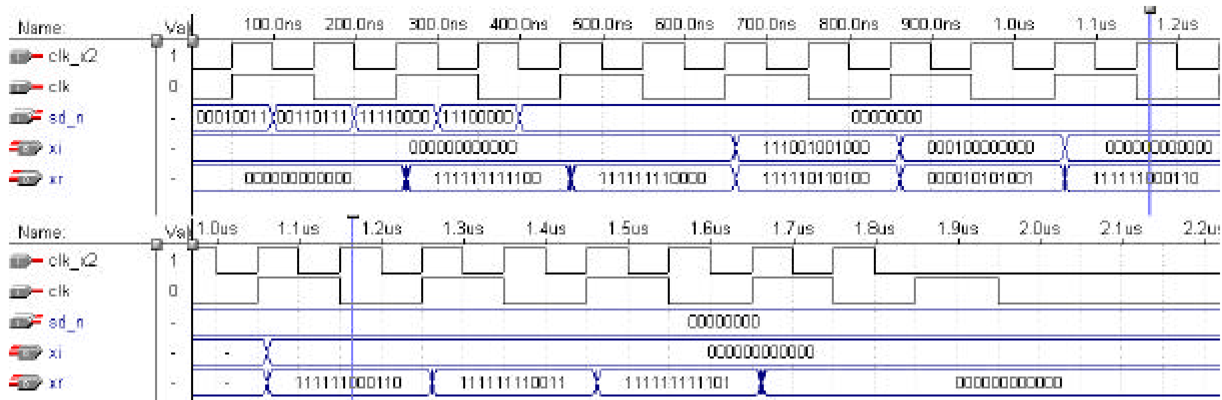
Figur 5.2.1 Timing simulering ved 200MHz

Klokkefrekvensen ble deretter redusert til 50MHZ. Resultatet er vist i Figur 5.2.2.



Figur 5.2.2 Timing simulering ved 50MHz

Designet ble også testet ved en betydelig lavere klokkefrekvens lik 10MHz. Resultatet av denne simuleringen er vist i Figur 5.2.3.



Figur 5.2.3 Timing simulering ved 10MHz

Verdiene [19, 55 , -16, -32] ble påtrykt Matlab funksjonen ”test_daf.m”, det gav følgende resultat:

Xr = [-4 -16 -76 169 -58 -13 -3], og Xi = [0 0 -440 256 0 0].

6. Diskusjon av resultatene

Den funksjonelle software simuleringen i avsnitt 5.1 gav tilfredsstillende resultater. Det ble her påtrykket en innsekvens bestående av 17 tilfeldige verdier, både på Matlab koden "test_daf.m", og VHDL designet i QuartusII. Resultatene fra de to simuleringen ble helt identiske(jmf tabell 5.1-1), dette betyr at VHDL programmet fungerer korrekt rent funksjonelt.

Timing simuleringen i avsnitt 5.2 gav derimot ikke like oppløftende resultater. For det første ble denne simulerings prosessen veldig tungvin og utføre, siden QuartusII er så ustabil som den er. Simuleringen ble derfor utført i Max+plussII mot en brikke av typen FLEX10k, denne er vesentlig dårligere(er tregere og har mindre plass) enn de brikker som er tilgjengelige i dag og aktuelle for implementering. Det kan derfor ikke forventes at en timing simulering mot en slik brikke gir tilfredsstillende resultater. En slik simulering kan likevell gi en indikasjon på hvor effektivt designet er, med tanke på kritiske faktorer nevnt under avsnitt 5.1(funksjonell simulering).

Det ble i første omgang i timing simuleringen testet med aktuell klokkefrekvens på 200MHz. Resultatet er vist i Figur 5.2.1. Denne figuren viser at designet ikke fungerer ved denne klokkefrekvensen på FLEX10k. En kan se at tidsforsinkelsen gjennom kretsen er større enn en klokkeperiode, dette vil aldri kunne gi tilfredsstillende resultat. Dette skyldes antageligvis at brikken FLEX10k er for dårlig ved såpass høye frekvenser.

Deretter ble frekvensen redusert til 50MHz. Resultatet er vist i Figur 5.2.2. Her er forsinkelsen i kretsen mindre enn en klokkeperiode, men verdiene ut er likevel ikke korrekte. Dette skyldes antageligvis forsinkelser i avrundingsenhetene. Dette er naturlig siden avrundingsfuksjonene utfører mange operasjoner i løpet av en klokkepuls(jmf "round_a", "round_b" og "round_c" i vedlegg E).
Resultatet var dessverre ikke uventet, da det under prosjektet ikke ble tid til optimalisering av disse enhetene. Dersom en raskere brikke hadde blitt brukt, ville antageligvis grensen for hvilken frekvenser avrundingsfunksjonene kunne klare, heves.

Det ble gjort et forsøk ved å kjøre klokkefrekvensen betraktelig ned, for å se om designet virkelig fungerer ved timing simulering. Klokkefrekvensen ble satt til 10MHz, resultatet er vist i Figur 5.2.3.
Ut fra denne figuren kan det observeres at verdiene stemmer overens med verdiene fra Matlab simuleringen.

En kan ut fra disse foregående opplysningene konkludere med at designet kompilert på en FLEX10k brikke, ikke vil klare de høye frekvens kravene det i denne oppgaven er snakk om. Det har heller aldri vært tenkt at designet skal implementeres på en slik brikke, men det oppstod så store problemer med QuartusII at dette var den eneste muligheten for å foreta en timing simulering. Designet vil sannsynligvis kunne kjøres med en høyere klokkefrekvens på en nyere og bedre brikke, og fungere tilfredsstillende ved disse frekvensene. Dette har ikke blitt testet ut og dokumentert i denne rapporten, grunnet nevnte problemer med QuartusII. Det må også påpekes at avrundings enhetene burde vært programmert mer effektivt, ettersom disse utfører mange tidkrevende operasjoner i løpet av en klokkepuls. Slik det ser ut så er det

avrundingsfunksjonene som er ”flaskehalsen” i designet, og en mer effektiv koding av disse vil antageligvis kunne bidra med at klokkefrekvensen kan økes. Det ble i dette prosjektet ikke nok tid til å optimalisere disse funksjonene.

Det ble også tenkt at designet skulle legges ut på en brikke(APEX-EP20K200EFC484-2) montert på et simuleringskort fra Altera, hvor designet kunne blitt fysisk testet. Grunnet problemene i QuartusII ble dette ”lagt på is”.

7. Konklusjon

Denne oppgaven gikk i hovedsak ut på å sette seg inn i forskjellige filterstrukturer, og komme med en konklusjon på hvilken struktur som er den best egnede for en gitt problemstilling.
Det skulle i tillegg testes ut et utviklingsverktøy, QuartusII fra Altera, og vurdere om dette var et egnet verktøy for ”implementering” av valgt filter struktur.

Det forelå fra oppdragsgiver et forslag til filter struktur, og vi startet med å analysere og sette oss inni dette. Vi oppdaget fort at denne filterstrukturen var forholdsvis komplisert i forhold til vårt kunnskaps nivå, og nivået på de relaterte fagene ved vi har hatt ved HIG. Det var derfor helt nødvendig at vi brukte mye tid på å sette oss inn i nødvendig tilleggs teori, for å få tilstrekkelig forståelse av den gitte filterstrukturen. Etter hvert som vi fikk bedre innsikt i filterstrukturen, fant vi ut at det var vanskelig for oss med våres forutsettninger å komme med forslag til en ny struktur som skulle fungere på samme linje med, eller bedre enn den gitte. Vi valgte derfor å gi en detaljert beskrivelse av den gitte strukturen, for samtidig å drøfte dens sterke og svake sider. Ut fra våre forutsettninger konkluderer vi med at den best egnede filterstrukturen er den gitte, som er et 11 koeffisients komplekst halfband båndpass filter, med 8 bits representasjon av koeffisientene.

Dette filteret realiserte vi ved hjelp av VHDL beskrivelse. Til dette benyttet vi QuartusII, og underveis lærte oss bruken av det. Det ble ytret ønske fra skolen om å utarbeide rapporten på denne delen som en bruker veiledning for QuartusII, dette ble gjort.
Under realiseringen fant vi både sterke og svake sider ved programmet. Mange av problemene som oppstod, virket som om de skyldtes usabilitet. Sent i prosjektperioden fikk vi en anbefalling fra leverandøren om å ikke kjøre QuartusII under Windows 98, på grunn av dette operativsystemets manglende evne til å takle store applikasjoner. Det ble anbefalt å bruke Windows NT eller Windows 2000 i forbindelse med QuartusII, men det var da for sent for oss å skifte operativ system. Det må også nevnes at mange av problemene vi opplevde rundt QuartusII, skyldtes tydlige svakheter i programmet. Vi brukte på grunn av nevnte problemer ekstra mye tid på denne delen av prosjektet. Vi har fått inntrykk av at QuartusII ville vært et bra program for design av digitale systemer dersom det hadde fungert som forutsatt, men med alle de problemene vi har erfart rundt bruken av QuartusII, kan vi ikke anbefale å bruke dette programmet.

Vi har påvist ved hjelp av en funksjonell simulering, at VHDL designet fungerer korrekt logisk sett. På grunn av QuartusII problemer fikk vi ikke gjennomført en tilstrekkelig timing analyse, men fikk indikasjoner ved å gjennomføre analysen i Max+plussII. Indikasjonene pekte i den retning at designet trenger en optimalisering for å fungere under de forhold den er ment for.

Vi føler at oppgaven var veldig utfordrende. Vi har under prosjekt perioden utviklet oss innenfor områdene ”multirate signalbehandling” og ”design av digitale filtre”. Denne oppgaven inneholder elementer fra mange forskjellige fag vi har hatt ved HIG, en oppgave med en slik ”bredde” var en type oppgaveform vi ønsket oss når hovedprosjektene ble tildelt.

Vi er fornøyd med oppgaven, og mener vi har besvart den på en tilfredstillende måte ut fra våres forutsetninger.

8. Litteraturliste

- Multirate digital signal processing – N J. Fliege – ISBN 0-471-93976-5
- Advanced digital signal processing – Glenn Zelniker & Fred J. Taylor – ISBN 0-8247-9145-2.
- System analysis and signal processing – Philip Denbigh – ISBN 0-201-17860-0.
- Digital systems design and protoyping using field programmable logic - Zoran Salcic & Asim Smailagic – ISBN 0-7923-9935-8.
- The Digital signal processing handbook – Vijay K. Madisetti & Douglas B. Williams – ISBN 0-8493-8572-5.
- Et utdrag av et kompendium av ”Hermann Lia”, med ukjent tittel.
- ”Nordic onboard DVB processor project, phaseII” tilsendt av Alcatel Space Norway.
- ”Design of a configurable 4-64 channel digital frequency demultiplexer for an onboard DVB-RCS/DVB-S processor” av ”Bjørn Roger Andersen”.
- FIR Compiler, User guide version 2.1 - Altera

9. Vedlegg

- Vedlegg A – Matlab koden til ”H_DAF.m”.
- Vedlegg B – Matlab koden til ”inn_signal.m”.
- Vedlegg C – Matlab koden til ”test_daf.m”.
- Vedlegg D – Blokkskjematisk oversikt over VHDL topp filen.
- Vedlegg E – Utdrag av VHDL koden.

Vedlegg A – Matlab koden til "H_DAF.m"

```
function[H_BP]=H_DAF(Q,b);
%Brukes: H_BP = H_DAF(Q,b);
%
%Q er antall filter koeffisienter, b er antall bit som koeffisientene skal
representeres med.
%H_BP som returneres er båndpasskoeffisientene til filteret.

%Kommentarene ovenfor skrives ut når "help H_DAF" skrives i Matlab
promptet.

H_LP=[zeros(size(0:Q-1))]; %Vektor som skal inneholde lavpasskoeffisientene
H_BP=[zeros(size(0:Q-1))]; %-------------||--------- båndpasskoeffisientene
f=[0 25/100 75/100 1];   %Vektor som inneholder overganger mellom de
                         % forskjellige båndene normalisert til fs/2

m=[1 1 0 0];    %Vektor som bestemmer nivået ved de forskjellige overgangene
h=remez(Q-1,f,m); %Finner koeffisientene ved hjelp av Remez algoritmen

X=inn_signal;     %Henter innsignalet fra funksjonen "inn_signal.m"

%--------------------- Lavpass modellen------------------------------------
----------------
%Kommenter bort dette felt, dersom den ikke ønskes
freqz(h);               %Plotter filterets lavpass respons
Title('Lavpass modell');
grid;
grid;
figure;
h_r=round(h*2^b);      %Avrunder koeffisientene til lavpassmodellen
freqz(h_r/2^b);        %Plotter lavpass respons med avrundede koeffisienter
Title('Lavpass modell med avrundede koeffisienter')
grid;
grid;

%-------------------------------------------------------------------------
----------------


%--------------------- Finner polyfase komponentene -----------------------
---------------
%NB!!! må ikke endres
%Finner den første polyfase komponenten(kodet med utgangspunkt i formel
%4.3-9)
for i=0:((Q+1)/4)-1
   H_BP(2*i+1)=((-1)^i)*h(2*i+1);
   H_BP(Q-2*i)=((-1)^(((Q+1)/2)-1-i))*h(2*i+1);
end;


H_BP((Q+1)/2)=j*h((Q+1)/2)*(-1)^(((Q+1)/4)-1);  %Setter den andre polyfase
                                                %kommponenten til 0.5

%-------------------------------------------------------------------------
--------------------
```

```
%--------------------- Ideell respons -----------------------------------
---------------
%Kommenter bort dette felt, dersom den ikke ønskes
figure;
freqz(H_BP);            %Plotter den ideelle båndpass responsen
Title('Ideell båndpassmodell');
figure;
Y_ideell=conv(X,H_BP);  %Folder det "ideele"(64 bit) innsignalet med
                        %"ideell" lengde på koeffisientene???
stem(abs(fft(Y_ideell)),'.');
Title('Respons ved signalpåtrykk uten avrunding');
grid;
%-----------------------------------------------------------------------
--------------


%--------------------- Denne delen foretar avrunding-----------------------
--------------
%Kommenter bort dette felt, dersom den ikke ønskes

X_round=round(X*2^8);         %Runder av inn signalet til 8 bit
X_round=X_round/2^8;          %Normaliserer

H_BP_round=round(H_BP*2^b);   %Runder av til b antall bits lengde på
                              %koeffisientene
Y=conv(X_round,H_BP_round);   %Folder 8 bit innsignalet med b bit
                              %koeffisienter
Y=round(Y*2^12);              %Runder av signalet på utgangen til 12 bit
Y=Y/2^12;                     %Normaliserer

figure;
stem(abs(fft(Y)),'.');       %Normaliserer og plotter
Title('Respons ved signalpåtrykk med avrunding');
grid;

figure;
stem(20*log10(abs(fft(Y))),'.'); %Normaliserer og plotter i logaritmisk
                                 %skala.
Title('Respons ved signalpåtrykk med avrunding(logaritmisk)');
grid;
```

Vedlegg B – Matlab koden til "inn_signal.m"

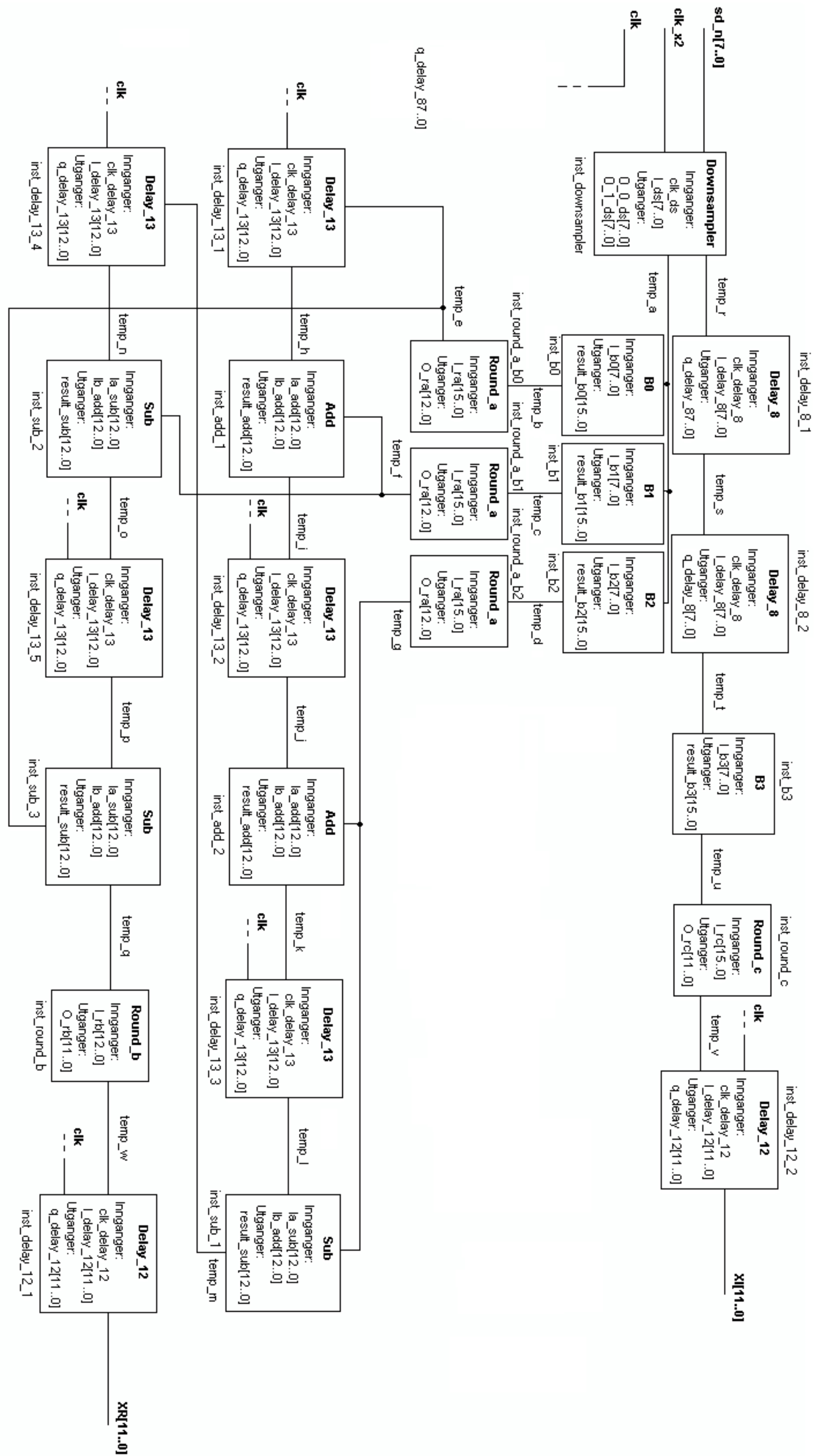
```
function[X]=inn_signal();
%Brukes: X = inn_signal;
%
%Krever ingen medsendte parametre, returnerer en vektor med summen av 200
diskre sinussignaler.

%Kommenteringen ovenfor skrives ut når det skrives "help H_DAF".

fs=200e6;                      %Sampelfrekvens
w=[0:(fs/1e6)-1]*1e6;          %Frekvens akse(MHz)
t=[0:1/fs:(length(w)-1)/fs];   %tidsakse

%Følgende loop genererer en matrise med aktuelle signaler(200
%cossinussignaler fra 1MHz til 200MHz)
for i=1:length(w)
   x(i,:)=cos(2*pi*w(i)*t);
end;

X=zeros(1,length(t));  %vektor som skal inneholde sammensatt signal

%Følgende loop summerer sammen alle signalene til et sammensatt signal.
for j=1:length(w)
   X=X+x(j,:);
end;

X=X/200;               %Skalerer signalet til 1.
stem(X,'.');
grid;
```

Vedlegg C – Matlab koden til "test_daf.m"

```
function []=test_daf(); %Kan være gunstig å sende med x som parameter slik:
%function []=test_daf(x);
%Brukes: test_daf;
%
%Krever ingen medsendte parametre, og returnerer ingen parametre

%Kommenteringen ovenfor skrives ut når det skrives "help test_daf".

hr=[-3 -16 -77 77 16 3];        %Koeffisientene til realgrenen
hi=[0 0 -j*128 0 0];            %Koeffisienetne til imaginærgrenen

x=[19 -98 33 57 -102 5 55 23 -17 -66 127 -128 51 32 -87 7 33 0];
%Testvektor. Nb!! må kommenteres bort dersom x
%sendes med som parameter til funksjonen
N=length(x);
xr=[zeros(size(1:N/2))]; %Skal inneholde de sampler fra testvektor som går
                         %i realgrenen.
xi=[zeros(size(1:N/2))]; % Skal inneholde de sampler fra testvektor som går
                         % i imaginærgren

%Følgende loop fordeler samplene i testvektoren på xr og xi
for i=1:N/2
   xr(i)=x((2*i)-1);
   xi(i)=x(2*i);
end;

yr=[zeros(size(1:(length(hr)+length(xr)-1)))]; %Vektor som skal inneholde
                                               %utsignalet
hxr(length(xr),length(yr))=0; %Matrise som skal brukes ved
                              %foldningsberegningen

%Følgende loop utfører mutiplikasjonene i foldningen
for i=1:length(xr)
   for j=1:length(hr)
      hxr(i,(j-1)+i)=hr(j)*xr(i);
   end;
end;

hxr_r=round(hxr/2^3);    %Runder av etter multiplikasjon(tilsvarer avrunding
                         %til 13 bit)

%Følgende loop utfører summasjonen i foldningen
for k=1:length(xr)
   yr=yr+hxr_r(k,:);
end;

yr=round(yr/2^1)    %Runder av for å finne verdiene på reell utgang(tlsvarer
                    %avrunding fra 13 til 12 bit)
yi=conv(xi,hi);  %Folder

Nr=length(yr);
Ni=length(yi);

yi=round(yi/2^4)    %Runder av for å finne verdiene på imaginær
                    %utgang(tilsvarer avrunding fra 16 til 12 bit)
```

```
stem((0:Nr-1),real(yr),'.');
grid;
figure;

stem((0:Ni-1),imag(yi),'.');
grid;
```

Vedlegg D – Blokkskjematisk oversikt over VHDL topp filen



Vedlegg E – Utdrag av VHDL koden

Følgende kode er ikke fullstendig, da det er mange neste like kode biter. Forskjellene er ofte bare en konstant eller et parameter. Det er her valgt å ta med det mest vesentlige, siden det er kommentert hvor i koden forandringen må gjøres. Disketten som leveres ned rapporten inneholder en fullstendig kode.

```
-- Toppfilen i designet: daf11


LIBRARY ieee;                   --Kaller opp IEEE bibiloteket
USE ieee.std_logic_1164.all;    --Standard biblioteket

ENTITY DAF11 IS
        port
        (
                clk_x2 :  IN  STD_LOGIC;                        -- 200MHz klokke inngangen
                clk :  IN  STD_LOGIC;                           -- 100MHz klokke
                sd_n :  IN  STD_LOGIC_VECTOR(7 downto 0);       -- Signal inngangen
                xi :  OUT  STD_LOGIC_VECTOR(11 downto 0);       -- Utgangen fra real grenen
                xr :  OUT  STD_LOGIC_VECTOR(11 downto 0)        -- Utgangen fra imaginær grenen
        );
END DAF11;

ARCHITECTURE bdf_type OF DAF11 IS

------------ Deklarering av komponenter --------------

COMPONENT downsampler
        PORT(clk_ds : IN STD_LOGIC;
                I_ds : IN STD_LOGIC_VECTOR(7 downto 0);
                O_0_ds : OUT STD_LOGIC_VECTOR(7 downto 0);
                O_1_ds : OUT STD_LOGIC_VECTOR(7 downto 0));
END COMPONENT;

COMPONENT  delay_8
        PORT(clock_delay_8 : IN STD_LOGIC;
                I_delay_8 : IN STD_LOGIC_VECTOR(7 downto 0);
                q_delay_8 : OUT STD_LOGIC_VECTOR(7 downto 0));
END COMPONENT;

COMPONENT delay_13
        PORT(clock_delay_13 : IN STD_LOGIC;
                I_delay_13 : IN STD_LOGIC_VECTOR(12 downto 0);
                q_delay_13 : OUT STD_LOGIC_VECTOR(12 downto 0));
END COMPONENT;

COMPONENT delay_12
        PORT(clock_delay_12 : IN STD_LOGIC;
                I_delay_12 : IN STD_LOGIC_VECTOR(11 downto 0);
                q_delay_12 : OUT STD_LOGIC_VECTOR(11 downto 0));
END COMPONENT;

COMPONENT add
        PORT(Ia_add : IN STD_LOGIC_VECTOR(12 downto 0);
```

```
                    Ib_add : IN STD_LOGIC_VECTOR(12 downto 0);
                    result_add : OUT STD_LOGIC_VECTOR(12 downto 0));
END COMPONENT;

COMPONENT sub
          PORT(Ia_sub : IN STD_LOGIC_VECTOR(12 downto 0);
                    Ib_sub : IN STD_LOGIC_VECTOR(12 downto 0);
                    result_sub : OUT STD_LOGIC_VECTOR(12 downto 0));
END COMPONENT;

COMPONENT round_b
          PORT(I_rb : IN STD_LOGIC_VECTOR(12 downto 0);
                    O_rb : OUT STD_LOGIC_VECTOR(11 downto 0));
END COMPONENT;

COMPONENT round_a
          PORT(I_ra : IN STD_LOGIC_VECTOR(15 downto 0);
                    O_ra : OUT STD_LOGIC_VECTOR(12 downto 0));
END COMPONENT;

COMPONENT round_c
          PORT(I_rc : IN STD_LOGIC_VECTOR(15 downto 0);
                    O_rc : OUT STD_LOGIC_VECTOR(11 downto 0));
END COMPONENT;

COMPONENT b0
          PORT(I_b0 : IN STD_LOGIC_VECTOR(7 downto 0);
                    result_b0 : OUT STD_LOGIC_VECTOR(15 downto 0));
END COMPONENT;

COMPONENT  b1
          PORT(I_b1 : IN STD_LOGIC_VECTOR(7 downto 0);
                    result_b1 : OUT STD_LOGIC_VECTOR(15 downto 0));
END COMPONENT;

COMPONENT b2
          PORT(I_b2 : IN STD_LOGIC_VECTOR(7 downto 0);
                    result_b2 : OUT STD_LOGIC_VECTOR(15 downto 0));
END COMPONENT;

COMPONENT b3
          PORT(I_b3 : IN STD_LOGIC_VECTOR(7 downto 0);
                    result_b3 : OUT STD_LOGIC_VECTOR(15 downto 0));
END COMPONENT;
------------------------------------------------------------------------------------------
---------------------- Deklarering av interne hjelpe signaler --------------------------

SIGNAL    temp_a :  STD_LOGIC_VECTOR(7 downto 0);   --fra ds til h0, h1 og h2 ("real greinen")

SIGNAL    temp_b :  STD_LOGIC_VECTOR(15 downto 0);            --ut av h0
SIGNAL    temp_c :  STD_LOGIC_VECTOR(15 downto 0);            --ut av h1
SIGNAL    temp_d :  STD_LOGIC_VECTOR(15 downto 0);            --ut av h2

SIGNAL    temp_e :  STD_LOGIC_VECTOR(12 downto 0);            --ut av round_a_b0
SIGNAL    temp_f :  STD_LOGIC_VECTOR(12 downto 0);            --ut av round_a_b1
SIGNAL    temp_g :  STD_LOGIC_VECTOR(12 downto 0);            --ut av round_a_b2
```

```
SIGNAL   temp_h :  STD_LOGIC_VECTOR(12 downto 0);          --ut av delay_13_1
SIGNAL   temp_i :  STD_LOGIC_VECTOR(12 downto 0);          --ut av add_1
SIGNAL   temp_j :  STD_LOGIC_VECTOR(12 downto 0);          --ut av delay_13_2

SIGNAL   temp_k :  STD_LOGIC_VECTOR(12 downto 0);          --ut av add_2
SIGNAL   temp_l :  STD_LOGIC_VECTOR(12 downto 0);          --ut av delay_13_3
SIGNAL   temp_m :  STD_LOGIC_VECTOR(12 downto 0);          --ut av sub_1

SIGNAL   temp_n :  STD_LOGIC_VECTOR(12 downto 0);          --ut av delay_13_4
SIGNAL   temp_o :  STD_LOGIC_VECTOR(12 downto 0);          --ut av sub_2
SIGNAL   temp_p :  STD_LOGIC_VECTOR(12 downto 0);          --ut av delay_13_5

SIGNAL   temp_q :  STD_LOGIC_VECTOR(12 downto 0);          --ut av sub_3

SIGNAL   temp_r :  STD_LOGIC_VECTOR(7 downto 0);   --fra ds til delay_8_1 ("imaginær greinen")
SIGNAL   temp_s :  STD_LOGIC_VECTOR(7 downto 0);           --ut av delay_8_1
SIGNAL   temp_t :  STD_LOGIC_VECTOR(7 downto 0);           --ut av delay_8_2
SIGNAL   temp_u :  STD_LOGIC_VECTOR(15 downto 0);          --ut av h3
SIGNAL   temp_v :  STD_LOGIC_VECTOR(11 downto 0);          --ut av round_c
SIGNAL   temp_w :  STD_LOGIC_VECTOR(11 downto 0);          --ut av round_b
-------------------------------------------------------------------------------------------
BEGIN
--------- Tilkobler "downsampler" ----------
ints_downsampler : downsampler
PORT MAP(clk_ds => clk_x2,
                  I_ds => sd_n,
                  O_0_ds => temp_a,
                  O_1_ds => temp_r);
---------------------------------------------
---------- Tilkobler real grenen -----------
inst_b0 : b0
PORT MAP(I_b0 => temp_a,
                  result_b0 => temp_b);

inst_b1 : b1
PORT MAP(I_b1 => temp_a,
                  result_b1 => temp_c);

inst_b2 : b2
PORT MAP(I_b2 => temp_a,
                  result_b2 => temp_d);

inst_round_a_bo : round_a
PORT MAP(I_ra => temp_b,
                  O_ra => temp_e);

inst_round_a_b1 : round_a
PORT MAP(I_ra => temp_c,
                  O_ra => temp_f);

inst_round_a_b2 : round_a
PORT MAP(I_ra => temp_d,
                  O_ra => temp_g);

inst_delay_13_1 : delay_13
PORT MAP(clock_delay_13 => clk,
                  I_delay_13 => temp_e,
```

```
                q_delay_13 => temp_h);

inst_add_1 : add
PORT MAP(Ia_add => temp_h,
                Ib_add => temp_f,
                result_add => temp_i);

inst_delay_13_2 : delay_13
PORT MAP(clock_delay_13 => clk,
                I_delay_13 => temp_i,
                q_delay_13 => temp_j);

inst_add_2 : add
PORT MAP(Ia_add => temp_j,
                Ib_add => temp_g,
                result_add => temp_k);

inst_delay_13_3 : delay_13
PORT MAP(clock_delay_13 => clk,
                I_delay_13 => temp_k,
                q_delay_13 => temp_l);

inst_subb_1 : sub
PORT MAP(Ia_sub => temp_l,
                Ib_sub => temp_g,
                result_sub => temp_m);

inst_delay_13_4 : delay_13
PORT MAP(clock_delay_13 => clk,
                I_delay_13 => temp_m,
                q_delay_13 => temp_n);

inst_subb_2 : sub
PORT MAP(Ia_sub => temp_n,
                Ib_sub => temp_f,
                result_sub => temp_o);

inst_delay_13_5 : delay_13
PORT MAP(clock_delay_13 => clk,
                I_delay_13 => temp_o,
                q_delay_13 => temp_p);

inst_subb_3 : sub
PORT MAP(Ia_sub => temp_p,
                Ib_sub => temp_e,
                result_sub => temp_q);

inst_round_b : round_b
PORT MAP(I_rb => temp_q,
                O_rb => temp_w);

inst_delay_12_1 : delay_12
PORT MAP(clock_delay_12 => clk,
                I_delay_12 => temp_w,
                q_delay_12 => xr);
--------------------------------------------
--------- Tilkobler imaginær grenen --------
```

```
inst_delay_8_1 : delay_8
PORT MAP(clock_delay_8 => clk,
                    I_delay_8 => temp_r,
                    q_delay_8 => temp_s);

inst_delay_8_2 : delay_8
PORT MAP(clock_delay_8 => clk,
                    I_delay_8 => temp_s,
                    q_delay_8 => temp_t);

inst_b3 : b3
PORT MAP(I_b3 => temp_t,
                    result_b3 => temp_u);

inst_round_c : round_c
PORT MAP(I_rc => temp_u,
                    O_rc => temp_v);

inst_delay_12_2 : delay_12
PORT MAP(clock_delay_12 => clk,
                    I_delay_12 => temp_v,
                    q_delay_12 => xi);
--------------------------------------------
END;
```

```
-- Entiteten: b0

-- Tar inn et 8 bits signal, og multipliserer med b0 koeffisientet.
-- B0 er en konstant som settes i programmet.

-- Entitetene "b0", "b1", "b2" og "b3" er alle bygd opp ved hjelp av LPM_MULT megafunksjonen.
-- Det er derfor stor likhet på programkoden til disse entitetene. Den eneste forskjellen er der koeffisient verdien
– settes, denne linjen er kommentert i koden.

LIBRARY ieee;                    --Kaller opp IEEE bibiloteket
USE ieee.std_logic_1164.all;     --Standard biblioteket

ENTITY b0 IS
        PORT
        (
                I_b0            : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                result_b0       : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
        );
END b0;

ARCHITECTURE arkitektur_b0 OF b0 IS

        SIGNAL sub_wire0        : STD_LOGIC_VECTOR (15 DOWNTO 0);
        SIGNAL sub_wire1_bv     : BIT_VECTOR (7 DOWNTO 0);
        SIGNAL sub_wire1        : STD_LOGIC_VECTOR (7 DOWNTO 0);

        COMPONENT lpm_mult
        GENERIC (
                lpm_widtha              : NATURAL;
                lpm_widthb              : NATURAL;
                lpm_widthp              : NATURAL;
```

```
            lpm_widths              : NATURAL;
            lpm_representation              : STRING;
            lpm_hint                : STRING
        );
        PORT (
                    dataa   : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    datab   : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    result  : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
        );
        END COMPONENT;

BEGIN                                               -- B0 = 00000011B => 3D
        sub_wire1_bv(7 DOWNTO 0) <= "00000011";
                                    -- Her settes koeffisient verdien b0 på 2's Komplement form
        sub_wire1   <= To_stdlogicvector(sub_wire1_bv);
        result_b0   <= sub_wire0(15 DOWNTO 0);

        lpm_mult_component : lpm_mult
        GENERIC MAP (
            lpm_widtha => 8,        -- Angir hvor mange bit inngang I_bo skal ha
            lpm_widthb => 8,        -- Angir hvor mange bit konstanten skal ha
            lpm_widthp => 16,       -- Angir hvor mange bit utgangen skal ha
            lpm_widths => 16,       -- Angir hvor mange bit en eventuell sum inngang skal ha
            lpm_representation => "SIGNED",
                        -- "SIGNED"->2's komplement form, "UNSIGNED"->vanlig binær form

            lpm_hint => "INPUT_B_IS_CONSTANT=YES,MAXIMIZE_SPEED=1"
                                    -- Setter om inngang B skal være konstant eller ikke.
        )
        PORT MAP (
            dataa => I_b0,
            datab => sub_wire1,
            result => sub_wire0
        );

END arkitektur_b0;
```

```
-- Entitet: delay_8

-- Tar inn et 8 bits signal på inngangen, og sender dette ut
-- på en 8 bits utgang etter en klokkepuls.

-- Entitetene "delay_8", "delay_12" og "delay_13" er alle bygd opp ved
-- hjelp av LPM_FF megafunksjonen.
-- Det er derfor stor likhet på programkoden til disse entitetene, eneste forskjellen er antall bit som forsinkes.
-- Det er kommentert i koden der dette gjøres.

LIBRARY ieee;                   --Kaller opp IEEE bibiloteket
USE ieee.std_logic_1164.all;    --Standard biblioteket

ENTITY delay_8 IS
        PORT
        (
            clock_delay_8           : IN STD_LOGIC ;
            I_delay_8               : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
            q_delay_8               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
```

```
        );
END delay_8;


ARCHITECTURE arkitektur_delay_8 OF delay_8 IS

        SIGNAL sub_wire0        : STD_LOGIC_VECTOR (7 DOWNTO 0);

        COMPONENT lpm_ff
        GENERIC (
                lpm_width               : NATURAL;
                lpm_fftype              : STRING
        );
        PORT (
                        clock   : IN STD_LOGIC ;
                        q       : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
                        data    : IN STD_LOGIC_VECTOR (7 DOWNTO 0)
        );
        END COMPONENT;

BEGIN
        q_delay_8   <= sub_wire0(7 DOWNTO 0);

        lpm_ff_component : lpm_ff
        GENERIC MAP (
                lpm_width => 8,         -- Angir hvor mange bit signalet skal være
                lpm_fftype => "DFF"     -- Angir hvilken vippe type som skal brukes, "DFF" eller "TFF"
        )
        PORT MAP (
                clock => clock_delay_8,
                data => I_delay_8,
                q => sub_wire0
        );

END arkitektur_delay_8;
```

```
-- Entitet: downsampler

-- Tar inn et 8 bits signal og demultiplekser dette ut på to utganger.

LIBRARY ieee;                   --Kaller opp IEEE bibiloteket
USE ieee.std_logic_1164.all;    --Standard biblioteket

ENTITY downsampler IS

        PORT
        (
                I_ds : IN STD_LOGIC_VECTOR(7 downto 0);
                clk_ds : IN STD_LOGIC;
                O_0_ds : OUT STD_LOGIC_VECTOR(7 downto 0);
                O_1_ds : OUT STD_LOGIC_VECTOR(7 downto 0)
        );

END downsampler;

ARCHITECTURE arkitektur_downsampler OF downsampler IS
```

```
        TYPE tilstand_type IS (t0,t1);      -- Danner en tilstandsmaskin med to tilstander
        SIGNAL tilstand : tilstand_type;    -- Hjelpe variabel

BEGIN
        PROCESS (clk_ds)            -- Prosessen er følsom på klokkesignalet

        BEGIN

                IF (clk_ds 'EVENT AND clk_ds = '1') THEN          -- Reagerer på hver positive flanke
                        CASE tilstand IS

                                WHEN t0 =>      -- Inne i tilstand 0
                                        O_0_ds <= I_ds;             -- Sendes inn signalet ut på O_0_ds[7..0]
                                        tilstand <=t1;              -- Hopper til tilstand 1

                                WHEN t1 =>      -- Inne i tilstand 1
                                        O_1_ds <= I_ds;             -- Sendes inn signalet ut på O_1_ds[7..0]
                                        tilstand <=t0;              -- Hopper til tilstand 0

                        END CASE;

                END IF;

        END PROCESS;

END arkitektur_downsampler;
```

```
-- Entitet: round_a

-- Tar inn et 16 bits signal på inngangen,
-- og runder dette av til et 13 bits signal på utgangen.

-- Entitetene " round_a ", " round_b " og ”round_c” er har stor likhet på programkoden.
-- Eneste forskjellden er antall bit.
-- Det er kommentert i koden hvor disse verdiene settes.

LIBRARY ieee;                   --Kaller opp IEEE bibiloteket
USE ieee.std_logic_1164.all;    --Standard biblioteket
USE ieee.std_logic_arith.all;   --For å kunne utføre aritmetiske operasjoner
USE ieee.std_logic_signed.all;  --For å kunne jobbe med STD_LOGIC_VECTOR som "signed

ENTITY round_a IS
        PORT
        (
                I_ra : IN STD_LOGIC_VECTOR(15 downto 0);        --16 bit på inngangen
                O_ra : OUT STD_LOGIC_VECTOR(12 downto 0)        --13 bit på utgangen
        );

END round_a;

ARCHITECTURE arkitektur_round_a OF round_a IS
        SIGNAL temp_inn : STD_LOGIC_VECTOR(15 downto 0);        --Hjelpe signal
        SIGNAL temp_inn_2 : STD_LOGIC_VECTOR(15 downto 0);      --Hjelpe signal
        SIGNAL temp_ut : STD_LOGIC_VECTOR(12 downto 0);         --Hjelpe signal
        SIGNAL temp_ut_2 : STD_LOGIC_VECTOR(12 downto 0);       --Hjelpe signal
```

```
BEGIN
        PROCESS(I_ra)                           --Prosessen er "følsom" for inngangen "I_ra"
        BEGIN

                temp_inn <= I_ra;               --Setter hjelpe signalet lik inngangssignalet

                IF I_ra(15) = '1' THEN temp_inn_2 <= 0 - temp_inn;
                                        --Hvis MSB=1, dvs. negativt tall, tas 2's kompliment av tallet,

                ELSE temp_inn_2 <= temp_inn;    --hvis ikke brukes det videre direkte
                END IF;

                temp_ut(12 downto 0) <= temp_inn_2(15 downto 3);        --De høyeste 12 bitene kopieres

                IF temp_inn_2(2) = '1' THEN temp_ut_2 <= temp_ut +1;
                                                --Hvis det skal rundes opp, adderes tallet med 1,

                ELSE temp_ut_2 <= temp_ut;      --hvis ikke brukes det videre direkte
                END IF;

                IF I_ra(15) = '1' THEN O_ra <= 0 - temp_ut_2;
                                --Hvis MSB=1, dvs. negativt tall, tas 2's kompliment av tallet,

                ELSE O_ra <= temp_ut_2;         --hvis ikke brukes det videre direkte
                END IF;                         --I begge tilfeller sendes det til utgangen

        END PROCESS;
END arkitektur_round_a;
```

```
-- Entitet: sub

-- Tar inn et 8 bits signal inn på den ene inngangen,
-- og trekker fra på et annet 8 bits signal på den andre inngangen.

-- Entitetene "add" og "sub" er begge bygd opp ved hjelp av LPM_ADD_SUB megafunksjonen.
-- Det er derfor stor likhet på programkoden til disse entitetene.
-- Eneste forskjellden er at det velges addisjon eller subtraksjon, det er kommentert i koden hvor dette gjøres.

LIBRARY ieee;                   --Kaller opp IEEE bibiloteket
USE ieee.std_logic_1164.all;    --Standard biblioteket

ENTITY sub IS
        PORT
        (
                Ia_sub          : IN STD_LOGIC_VECTOR (12 DOWNTO 0);-- Den "positive" inngangen
                Ib_sub          : IN STD_LOGIC_VECTOR (12 DOWNTO 0);-- Den "negative" inngangen
                result_sub              : OUT STD_LOGIC_VECTOR (12 DOWNTO 0)
        );
END sub;

ARCHITECTURE arkitektur_sub OF sub IS

        SIGNAL sub_wire0        : STD_LOGIC_VECTOR (12 DOWNTO 0);
```

```
	COMPONENT lpm_add_sub
	GENERIC (
		lpm_width		: NATURAL;
		lpm_direction		: STRING;
		lpm_hint		: STRING
	);
	PORT (
			dataa	: IN STD_LOGIC_VECTOR (12 DOWNTO 0);
			datab	: IN STD_LOGIC_VECTOR (12 DOWNTO 0);
			result	: OUT STD_LOGIC_VECTOR (12 DOWNTO 0)
	);
	END COMPONENT;

BEGIN
	result_sub	<= sub_wire0(12 DOWNTO 0);

	lpm_add_sub_component : lpm_add_sub
	GENERIC MAP (
		lpm_width => 13,
		lpm_direction => "SUB",		-- Velger om det er addisjon eller subtraksjon
		lpm_hint => "ONE_INPUT_IS_CONSTANT=NO" -- Kan velge den ene inngangen konstant
	)
	PORT MAP (
		dataa => Ia_sub,
		datab => Ib_sub,
		result => sub_wire0
	);

END arkitektur_sub;
```