NTNU
Det skapende universitet

Dynamisk rekonfigurerbart digitalt filter

Håkon Helsing

Master i elektronikk
Oppgaven levert: Juli 2009
Hovedveileder: Kjetil Svarstad, IET

Norges teknisk-naturvitenskapelige universitet
Institutt for elektronikk og telekommunikasjon

Oppgavetekst

Det er i prosjektoppgaven laget et program som kan brukes i sammenheng med dynamisk rekonfigurering av digitale filtere. Programmet genererer VHDL-kode for et FIR-filter med konstantmultiplikatorer optimalisert med CSD-enkoding(Canonical Signed Digit).

Oppgaven blir videreføring av dette arbeidet med spesielt fokus på:

- Spesifisere hvordan et system for dynamisk filter på FPGA kan fungere, eventuelt analysere alternativer.

- Identifisere nødvendige HW- og SW-komponenter i et slikt system.

- Spesifisere hvordan prosessorressurser(f.eks MicroBlaze), brukes i dynamisk håndtering.

- Designe og implementere et utvalg av SW- og HW-komponenter med hensyn på en rudimentær systemtest.

Oppgaven gitt: 15. januar 2009
Hovedveileder: Kjetil Svarstad, IET

Forord

Denne rapporten er skrevet som en masteroppgave det avsluttende semesteret på Elektronikk ved NTNU. Jeg har spesialisert meg i digital systemdesign, med hovedfokus på hardware/software-codesign og rekonfigurerbare systemer. Dynamisk rekonfigurering fascinerer meg med sin evne til å kombinere det beste fra to verdener der effektiv maskinvare forenes med fleksibiliteten til programvare. Å skape et dynamisk rekonfiguerbart digitalt filter som realiserer dette er derfor en motiverende oppgave. Digitale FIR-filtre vil alltid være nyttige, og jeg håper at dette arbeidet vil kunne bidra til realisering av et slikt filter. Oppgaven er en fortsettelse av mitt prosjekt høsten 2008 som igjen tok utgangspunkt i masteroppgaven *Low power/high performance dynamic reconfigurable filter design* av Vebjørn Bystrøm, våren 2008. En stor takk rettes veileder professor Kjetil Svarstad for inspirasjon og god oppfølging gjennom arbeidet med oppgaven. Jeg vil også takke førsteamanuensis Bjørn B. Larsen for motivasjon og råd gjennom studiet.

Sammendrag

Denne masteroppgaven er et arbeid mot et system med et dynamisk rekonfigurerbart digitalt FIR-filter på FPGA. Filteret benytter konstantmultiplikatorer optimalisert med CSD-kode for lite areal og kort rekonfigureringstid samtidig med høy fleksibilitet. Filterstruktur og partisjoneringsalternativer for oppdeling i dynamisk og statisk design er analysert, det er sett på betydningen ved bruk av bussmakroer, og det er sett på muligheten for bruk med multiplikatormoduler dynamisk størrelse for eventuelt å gjenbruke ledig logikk. Tre filtervarianter er simulert og syntetisert mot Virtex-4 på et testkort av typen Suzaku-V. Et FIR-filter på transponert form med individuelle delvis rekonfigurerbare konstantmultiplikatormoduler gir et godt utgangspunkt for videre arbeid. Arbeidet er en videreføring av prosjektoppgaven der det ble laget et program som genererer FIR-filtre og CSD-enkodede konstantmultiplikatorer i VHDL. Prosjektoppgaven ble skrevet i desember 2008.

Akronymer

- ASIC - Application-Specific Integrated Circuit
- CLB - configurable logic blocks
- DPR-modul - Delvis rekonfigurerbar(Dynamically Partial Reconfigurable) modul
- FPGA - Field-programmable Gate Arrays
- FIR - Finit Impulse Response
- PR-modul - Delvis rekonfigurerbar(Partial Reconfigurable) modul
- RTL - Register Transfer Level
- RTR - Run-time-Reconfiguration

Innhold

Akronymer **v**

1 Introduksjon **1**
1.1 Motivasjon 1
1.2 Oppgavebeskrivelse 2
1.3 Rapportens struktur 3

2 Bakgrunn og teori **5**
2.1 Digitalt filter 5
2.1.1 FIR-filter 5
2.2 Multiplikatorer 7
2.2.1 Konstantmultiplikatorer 7
2.2.2 Optimalisering 8
2.3 FPGA 9
2.3.1 Dynamisk rekonfigurering 10
2.3.2 Designverktøy 12
2.3.3 Bussmakroer 13
2.3.4 Testkort 15
2.4 Tidligere arbeid fra prosjektoppgaven 16

3 Design og arkitektur **19**
3.1 Funksjonalitet 21
3.1.1 Endring av koeffisienter 22
3.2 Oppdeling i statisk og dynamisk design 22
3.2.1 Minst mulig rekonfigurering 22
3.2.2 Oppdelingshensyn 23
3.3 Multiplikatorer 25
3.3.1 CSD-kode 25
3.3.2 Variabel multiplikatorstørrelse 25
3.3.3 Multplikatormoduler 25
3.3.4 Utnyttelse av varierende multiplikatorstørrelse 27

3.4 System med dynamisk rekonfigurerbart filter 28
3.5 Filterstruktur 30
3.5.1 Dynamiske modulstørrelser 30
3.5.2 Direkte form, tre moduler 31
3.5.3 Transponert filter 32
3.5.4 Sammenligning av filterene 33
3.6 Optimalisering 33

4 Implementering **37**
4.1 Implementering av filtre 38
4.1.1 Valg av filtre for implementering 38
4.1.2 Filterspesifikasjoner 38
4.2 Implementering i VHDL 39
4.2.1 Direkte form, uten bussmakroer 39
4.2.2 Direkte form, med bussmakroer 40
4.2.3 Transponert form, uten bussmakroer 41
4.3 Plasering av moduler i PlanAhead 42

5 Syntese og simulering **45**
5.1 Simulering av filter 45
5.2 Syntese av filtre 46
5.2.1 Designverktøy 46
5.2.2 Målplatform 46
5.2.3 Utplassering på FPGA 47

6 Resultater **49**
6.1 Simulering 49
6.2 Syntese 49

7 Diskusjon **53**
7.1 Syntese- og simuleringsresultater 53
7.1.1 Synteseresultater 54
7.1.2 Implementering på FPGA 54
7.2 Designvalg 55
7.3 Generelt 57

8 Konklusjon og videre arbeid **59**

Referanser **62**

RTL-kretsskjema for implementerte filtre **63**

Synteseresultater **67**
.1 Timing 67
.2 Direkte form, med bussmakoer 67
.3 Transponert form, uten bussmakroer 69
.4 Direkte form, uten bussmakroer 71

Bølgediagram for direkte form FIR-filter, uten bussmakroer **73**

Kapittel 1

Introduksjon

Rekonfigurerbar maskinvare har til hensikt å fylle gapet mellom hardware og software. Både hardware og software har begge sterke sider der hardware gir optimaliserte løsninger med hensyn på kjøretid, effekt- og arealforbruk, mens software på sin side gir stor fleksibilitet. Rekonfigurering byr på en sammenslåing av disse egenskapene hvilket gir stort potensiale innenfor ulike applikasjoner. Filteret beskrevet i denne oppgaven optimaliseres med dynamisk rekonfigurering mot lite areal og høy ytelse samtidig med fleksibilitet. Filterets koeffisienter skal kunne endres ved bruk av moduler med CSD-optimaliserte konstantmultiplikatorer som byttes ut under kjøring.

1.1 Motivasjon

FIR-filtere finnes i svært mange systemer for digital signalbehandling. Spesielt i mobile systemer vil behovet for lavt energiforbruk og høy ytelse alltid være tilstede slik at et filter som best mulig forener dette vil være ettertraktet.

Et filter implementert på en hardware på en ASIC gir hurtighet og lavt energiforbruk. Ulemper ved en ASIC er lang utviklingstid uten mulighet for endringer i etterkant av produksjon. Et filter implementert i programvare vil kunne endre karakteristikk både raskt og ofte, men krever mange iterasjoner og er dermed tregere og har høyt effektforbruk.

Et eksempel på effektiv multiplikasjon i hardware er bruk av *konstantmultiplikatorer* optimalisert med *CSD-enkoding*. Disse brukes når den ene multiplikanden er en fast koeffisient, slik som i FIR-filtre. Problemet med denne formen for multiplikatorer er at de nettopp kun kan multiplisere med denne ene verdien. Dette kan løses ved bruk av en såkalt FPGA som på engelsk står for Field-Programmable Gate Array.

FIR-filter strukturen, direkte oversatt til register-transfer nivå(RTL), består av kun de enkle komponentene registre, adderere og multiplikatorer. De to førstnevnte er vanskeligere å optimalisere mens det ligger et stort potensiale i forbedring av multiplikatorene.

Delvis rekonfigurering er et tema innenfor FPGA som er av stadig økende interesse. Delvis rekonfigurering gir fordeler som økt systemytelse, kortere rekonfigureringstid, muligheter for å dele hardware mellom applikasjoner og muligheter for å endre hardware i felt **??**. Et digitalt filter som benytter konstantmultiplikatorer kan gjøres adaptivt ved implementasjon på FPGA. Utbytting, eller delvis rekonfigurering av moduler med konstantmultiplikatorer skal skje i kjøretid mens resten av systemet kjører uendret. Dette vil gi fleksible og arealeffektive FIR-filtre. Denne rapporten omhandler nettopp hvordan slike filtre kan lages.

1.2 Oppgavebeskrivelse

Denne rapporten er en videreføring av arbeidet gjort i mitt prosjekt, *Dynamisk rekonfigurerbart digitalt filter*, høsten 2009. Prosjektet er en fortsettelse av arbeidet gjort i en masteroppgave av Vebjørn Bystrøm[3]. Arbeidet i masteroppgaven omhandler blant annet hvordan CSD-enkoding ytterligere kan redusere antall operasjoner ved konstantmultiplikasjon, og dermed arealet av konstantmultiplikatorene som brukes i FIR-filtre syntetisert for bruk på FPGA.

Oppgaven består i å videreutvikle det dynamisk rekonfigurerbarte filteret. Det er derfor satt opp designmål og egnenskaper for et system med et dynamisk rekonfigurerbart filter:

- Filteret skal kunne bruke *CSD-enkodede konstantmultiplikatorer* da høy ytelse og lite areal er sentralt .
- *Fleksibilitet* og *funksjonalitet*. Mulighet for å endre filterets konfigurasjon på flere nivåer. Nivåer kan være hele filteret, alle koeffisientene eller kun én koeffisient.
- *Filterarkitektur*. Det er viktig at filteret skal ha høy grad av mulighet for spesifisering av attributter som bitbredde, filterorden, og presisjon i henhold til ønsket spesifikasjon. Filtere skal genereres med programmet fra prosjektet.
- *Optimalisering*. Utvikling av det dynamiske filteret er i seg selv et skritt mot et bedre filter. Tilrettelegging for ytterligere optimalisering er derfor vektlagt.

- *Dynamisk og delvis rekonfigurering* skal benyttes. Et mål med oppgaven er ikke bare å lage et bra filter, men å utnytte og utforske potensialet ved rekonfigurering.

  I tillegg til de overordnede designmålene, vil det bli utforsket betydningen partisjonering av filteret i statisk og dynamisk areal, bruk av bussmakroer og transponert form FIR-filter fremfor direkte form, samt muligheten for *gjenbruk av arealet* som blir ledig ved innsetting av en konstantmultiplikator som er mindre enn den forrige.

1.3 Rapportens struktur

Bakgrunn og teori blir beskrevet i kapittel 2. Kapittel 3 fremstiller kort et system med et rekonfigurerbart filter. Deretter beskrives varianter av filtertyper med ulik inndeling i statisk og dynamisk design, og konsekvensene dette har for dynamisk rekonfigurering.

I kapittel 4 presenteres implemetasjon av tre filtre hvorav ett er ment å kunne brukes i videre arbeid. De andre to filtrene skal gi innsikt i forskjeller ved endinger av filterstrukturen og innføring av bussmakroer.

Kapittel 5 beskriver utplassering av filtrene på FPGA. Simulering av ett av filtrene blir også beskrevet dette kapittelet. Utfallet av syntese og simulering presenteres i kapittel 6. Resultatene diskuteres mot forventninger og mot videre arbeid i kapittel 7. Konklusjonen og videre arbeid kommer i kapittel 8.

Kapittel 2

Bakgrunn og teori

Dette kapittelet beskriver det viktigste av teori og bakgrunn for forståelse av arbeidet. Her presenteres digitale filtre, multiplikatorer, FPGA, rekonfigurering, designverktøy og kort om relatert arbeid.

2.1 Digitalt filter

Digitale filtere er svært mye brukt i moderne signalbehandling. Sammenlignet med et analogt filter er det digitale filteret både raskere og mer nøyaktig, men har en endelig oppløsning. Filteretstrukturen som er brukt i denne oppgaven er et FIR-filter, valgt forid denne typen filter er svært mye brukt. Filtert er også valgt av sin enkelhet for å demonstrere egenskaper og fordeler ved dynamisk rekonfigurering av konstantmultiplikatorer i filtersammenheng.

2.1.1 FIR-filter

Filterene som benyttes i denne rapporten er FIR-filtre. Fordeler med FIR-filteret er at det alltid vil være stabilt, og at det er enkelt å oppnå lineær faserespons. FIR-filtre gis ved

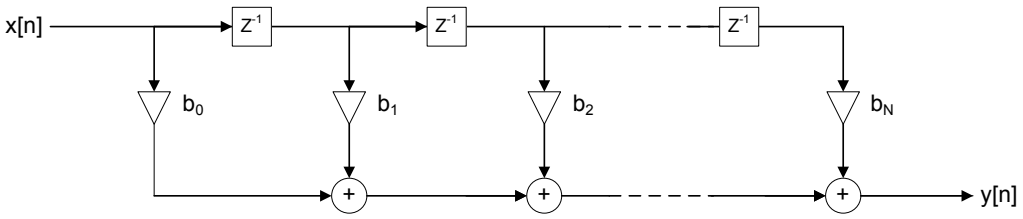
$$y[n] = b_0[n] + b_1x[n-1] + ... + b_Nx[n-N] \tag{2.1}$$

der $x[n]$ er innsignalet, og $y[n]$ er utsignalet og N er graden til filteret. b_i er filterkoeffisientene[15].

Ved Z-transform av impulsresponsen gir dette øverføringsfunksjonen

$$H(z) = \sum_{k=0}^{N} h(k)^{-k} \tag{2.2}$$

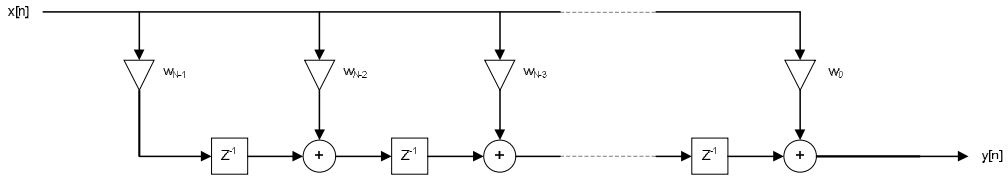
der b_k er settet med filterkoeffisienter, N er graden til filteret, og [7]. Filteret representeres i figur 2.1. Filteret kan realiseres ved å bytte Z^{-1} med registre, koeffisientene med konstantmultiplikkatorer og summasjonenen med adderere.



Figur 2.1: FIR-filter av grad N.

Transponering av filter

Filteret i 2.1 kan uten å endre overføringsfunksejon transponeres og representeres slik som i figur 2.2 [14]. Denne måten å tegne filteret på gjør at multiplikatorene ikke kommer mellom registrene og addererne. Det skal senere, i kapittel 3 vises hvordan dette kan være nyttig for implementasjon av filtre på FPGA.



Figur 2.2: Transponert FIR-filter av grad N.

Filterstruktur

Strukturen i et generelt FIR-filter er vises av figur **??**. Vanligvis vil hardware-implementasjon innebære direkte overføring fra figur til komponent på RTL-level. Sigma vil bli adderere, $Z^(- 1)$ vil bli registere

mens b_0 til b_N blir multiplikatorer. Dersom filter-ordenen økes, utvides filteret med tilsvarende antall multiplikasjoner, registere og addisjoner. Denne økningen utgjør påheng av en ekstra MAC-modul(Multiply and Accumulate). Hvis filteret har grad N, vil filteret kreve N+1 MAC operasjoner[kjeldsberg, switching activity reduction]. Ved en gitt filterspesifikasjon vil hver av multiplikatorene multiplisere med en forhåndsbestemt fast verdi og multiplikatoren kan derfor erstattes med en konstantmultiplikator.

2.2 Multiplikatorer

Bitbredden ved full presisjon for binær multiplikasjon gis av 2.3 [**?**].

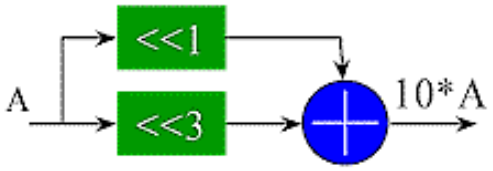
$$P = \sum_{k=0}^{M+N-1} p_k 2^k = (\sum_{i=0}^{M-1} a_i 2^i)(\sum_{j=0}^{N-1} b_j 2^j) = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} a_i b_j 2^{i+j} \qquad (2.3)$$

En konstantmultiplikator må altså ha $M + N = K$ bit på utgangen for å gi full presissjon. Multiplikatorene brukt i filteret i oppgaven skal ha full presisjon. Trunkering kan da gjøres etter ønsket behov, avhengig av applikasjon.

2.2.1 Konstantmultiplikatorer

Multiplikatoren er den komponenten i filteret som har størst potensiale for forbedring. En konstantmultiplikator i maskinvare har som oppgave å multiplisere et tall på inngangen med en fastbestemt verdi som også er kalt en koeffisient. Implementert i hardware varierer konstantmultiplikatorene arealmessig med antall 1'ere i tallet det skal multipliseres med. Dette skyldes at det for hvert siffer i koeffisienten, på binær form, utgjør én addisjon. Figur 2.3 viser en konstantmultiplikatorer som fast multipliserer med 10_{10} (1010_2), ved å addere et venstreskift med tre venstreskift. En oversikt over konstantmultiplikatorer finnes i [1].

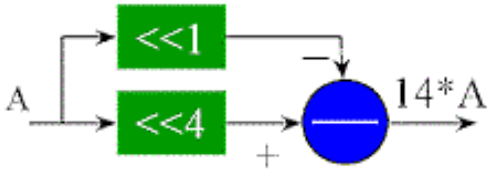
Fordi det kreves en ekstra adderer for hver er i tallet det multipliseres med, er det lønnomt med en algoritme for å representere tallene med færre 1'ere. CSD-enkoding gjør nettopp dette ved å representere tallene med settet $0, 1, -1$ der 1 utgjør en addisjon mens -1 utgjør en subtraksjon. -1 skrives ofte $\overline{1}$. Vi kan da skrive $B = 14_{10} = 1110_2 = 100\overline{1}0_{CSD}$.



Figur 2.3: Konstantmultiplikator

Multiplikasjon av A og B kan gjøres ved $(A <<) - (A << 1)$ og implementeres med 2 adderere/subtraherere i stedet for 3. Konstantmultiplikatorene vil med CSD-kode gjennomsnittlig ha færre operasjoner enn ved bruk av 2's kompliment slik som over. CSD-kode har aldri to 1'ere ved siden av hverandre, hvilket betyr at en vilkårlig CSD-kodet koeffisient med n siffer vil ha $\lceil n/2 \rceil$ maksimum antall 1'ere, hvilket betyr ca 33% færre bit som ikke er 0 enn ved 2's kompliment.

Filtergeneratoren fra prosjektet har implementert denne algoritmen. Multiplikasjonen er vist i figur 2.4.

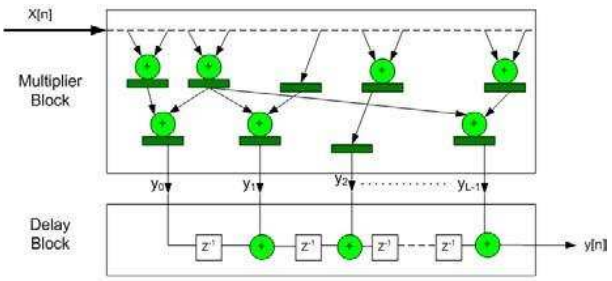


Figur 2.4:

2.2.2 Optimalisering

Andre former for optimalisering av multiplikasjon som vil være aktuell i FIR-filtrene i denne oppgaven er *subexpression sharing* og utnyttelse av symmetri[12]. Såkalt subexpression sharing kan utnyttes til effektiv optimalisering av multiplikasjonene med konstanter for bruk i FIR-filtre. Denne formen for optimalisering gjenbruker delsummene i multiplikatorene for å spare areal. En god algoritme for dette, og en sammenligning av andre algoritmer er beskrevet i [9].

I figur **??** viser hvordan et transponert filter kan optimalieres med subexpression sharing. Det brukes her konstantmultiplikatorer med til å optimalisere, men ikke CSD-kode.



Figur 2.5: Multiplikatormodul optimalsiert med konstantmultiplikatorer[11].

2.3 FPGA

En FPGA er en halvlederkomponenet som forskjell fra en ASIC inneholder programmerbar logikk. Logikken konfigureres før den tas i bruk, men kan endres om igjen mange ganger ved *rekonfigurering*.

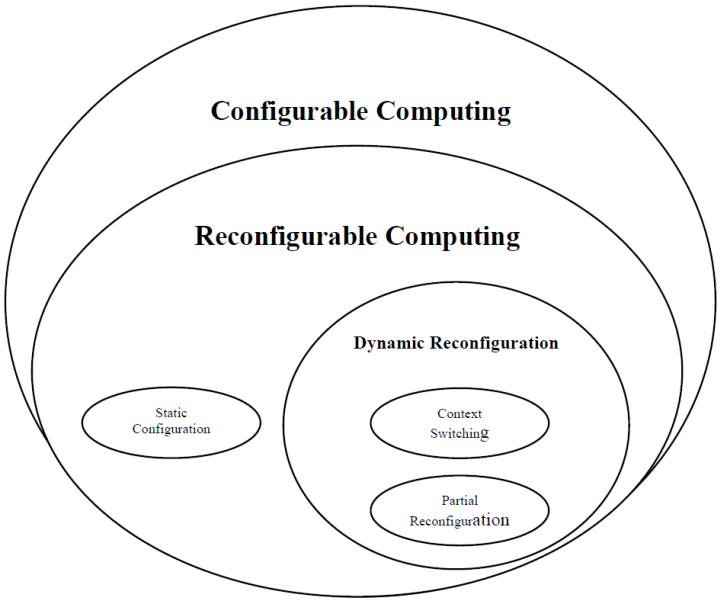
Å flytte applikasjoner fra software over i hardware kan gi stor grad av aksellerering av designet. Dersom dette skal gjøres ved implementasjon på en ASIC (Application Specific Integrated Circuit) er dette en omfattene designprocess som er både langtekkelig og kostbar. Dersom det oppdages feil i designet kan dette medføre store omkostninger ved at man blir satt flere stef tilbake i prosessen.

En FPGA derimot kan oppdateres eller endres etter at den er tatt i bruk, og tilpasse seg nye standarder eller behov for korreksjoner og oppdateringer. Konfigureringen skjer kort fortalt ved at *logiske blokker* kan styres til sammen å utgjøre logiske uttrykk koblet sammen via kompleks *ruting*.

En FPGA kan rekonfigureres ved at et design lastes inn på en FPGA i løpet så kort tid som millisekunder eller minutter. FPGA brukes både i ferdige produkter, eller til prototyping ved at designeren umiddelbart etter konfigurering kan utføre testprosedyrer.

Det er også mulig å konfigurere en FPGA samtidig som det er i bruk, som ved kjøretids-rekonfigurering. Det er et mål at filteret i oppgaven skal implementeres på FPGA, og deretter endres mens resten av systemet kjører på FPGA.

Figure 2.6 beskriver hvordan begreper innen rekonfigurering kan inndeles[4].



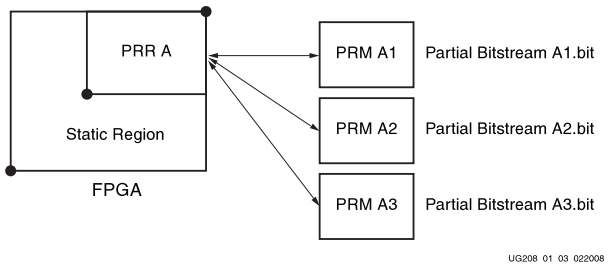
Figur 2.6: Strukturering av konfigurering.[4]

Logikken som er spredt utover silisiumbrikken beskrives i VHDL eller Verilog.

2.3.1 Dynamisk rekonfigurering

Det finnes to hovedmetoder for dynamisk rekonfigurering av FPGA'er fra Xilinx. *Differensialbasert rekonfigurering* brukes for å gjøre mindre endringer i designet, som for eksempel bytte av I/O standarder, LUT-likninger eller innholdet i RAM-blokker [8]. Dynamisk rekonfigurering kan også benyttes i kombinasjon med selvrekonfigurering[13]. *Modulbasert delvis rekonfigurering* brukes når det skal gjøres endringer moduler og mellomstore komponenter og er mest aktuell i denne oppgaven.

[8]



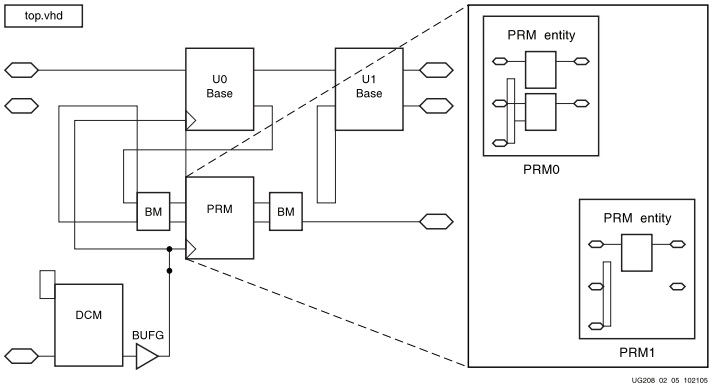
Figur 2.7: Ulike versjoner av en modul kan byttes ut.[6]

Modulpasert rekonfigurering

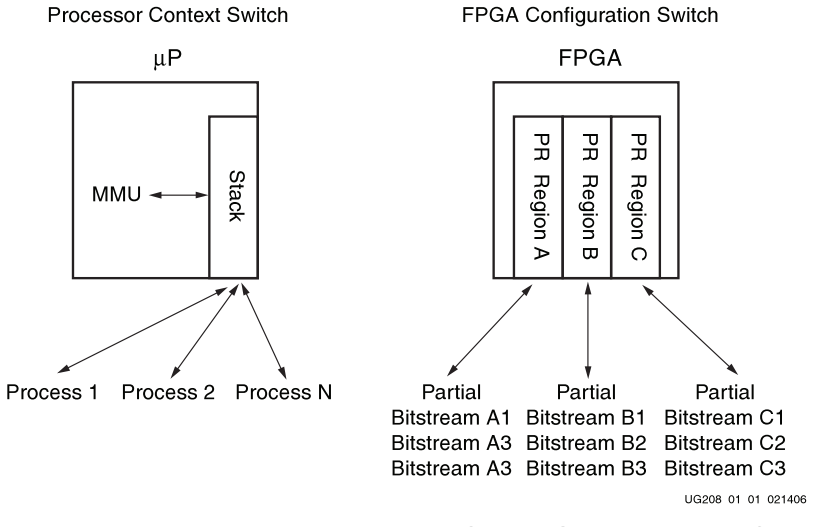
For modulpasert rekonfigurering gjelder et eget sett med designregler. Disse innebærer at alt som skal rekonfigureres må inndeles i moduler for delvis rekonfigurering, eller såkalte PR-moduler(Partial Reconfigurable-module). Det statiske designet, som ikke skal rekonfigureres må også være inndelt i en egen modul. Mellom alle PR-moduler og alt annet design må det være bussmakroer. Videre må bussmakroene, de dynamiske og de statiske modulene alle innkapsles i en toppmodul som skissert i figur **??**.

En viktig parameter for karakterisering av både et adaptivt filter og et rekonfigurerbart system er *rekonfigureringstid*. Desto større areal som må rekonfigureres på FPGA, jo lengre blir rekonfigureringstiden. Designet blir derfor delt inn i de to kategoriene, *rekonfigurerbart design* og *statisk design*.

Bruk av en PR-modul krever at all annen *ruting* av signaler går utenom denne for å unngå at disse blir korrupte etter rekonfigurering. Ruting på FPGA krever mye ressurser, og det at disse ressursene blir blokkert av PR-modulen utgjør nok en grunn til å begrense modulstørrelen. PR-modulenes posisjon bestemmes først, deretter plasseres det statiske designet på det resterende ledige området, innenfor eventuelle ytterligere begrensninger.



Figur 2.8: Toppmodul av et design, inndelt i delvis rekonfigurerbare moduler merket PRM [6]



Figur 2.9: Analogi mellom mikroprosessor og delvis rekonfigurering.[6]

2.3.2 Designverktøy

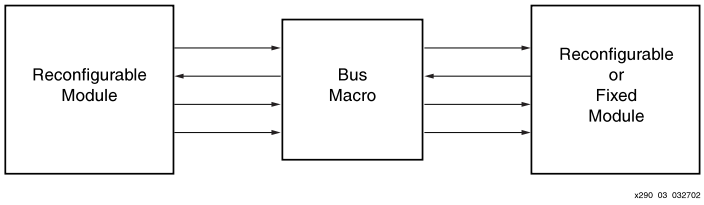
PlanAhead fra Xilinx, Inc. er et verktøy for å forenkle prosessen med å utplassere logikk og ruting på FPGA. Dette muliggjør også en bedre oversikt over designet for å oppnå bedre resultater ved manuelt arbeid enn ved auto-utplasserigsfunksjonen til programmet. *Early Access*

Reconfiguration Tool er et verktøy som gir mulighet for å unngå ruting gjennom de rekonfigurerbare modulene. Dette gjør det da mulig å overskrive området uten fare for å ødelegge det som er rutet gjennom området.

2.3.3 Bussmakroer

For designet på FPGA er det behov for et grensesnitt mellom den rekonfigurerbare hardwaren og det resterende designet. En utfordring med delvis rekonfigurering er å få tilkoblingene mellom de ulike modulene til å være fysisk på akkurat samme sted på FPGA'en både før og etter rekonfigurering. Løsningen på dette er å bruke såkalte bussmakroer.

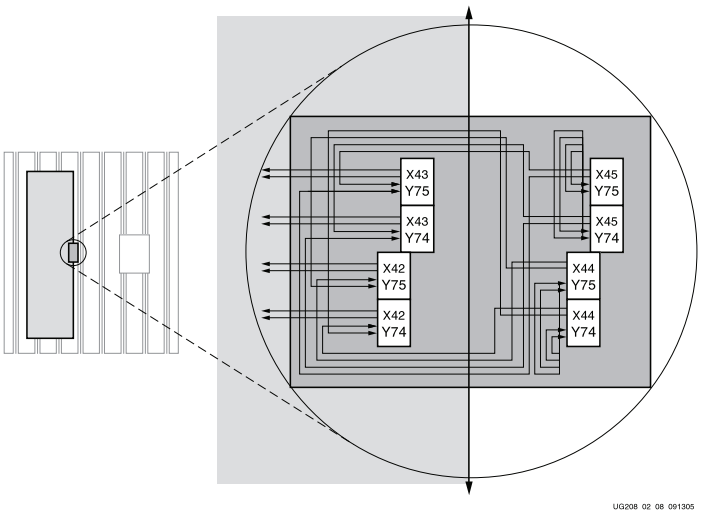
Kommunikasjon mellom modulene skjer via bussmakroer for å sikre at modulene kobles til på en bestemt måte, og på et bestemt sted.



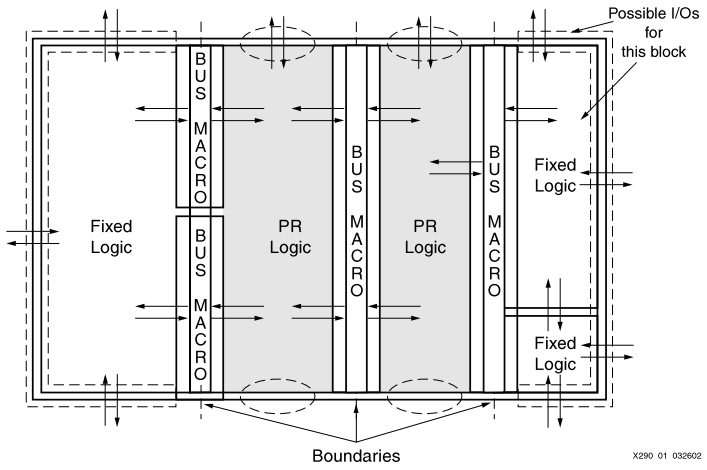
Figur 2.10: Plassering av bussmakroer[16].

Filtrene som beskrives i denne oppgaven vil oppta lite areal på FPGA. Filtrene er strukturer med høy grad av parallellitet, og bruken av bussmakroer har derfor stor betydning for det totale arealet.

Det finnes flere forskjellige bussmakroer å velge mellom. De viktigste forskjellene er bredde, retning og om bussen skal være synkron eller asynkron. I tilleg må macroen som brukes være laget for den aktuelle platformen. Kostnaden for er 2 CLB'er for smale bussmakeroer. Brede bussmakroene strekker seg over 4 CLB'er, slik at modulene de binder sammen kan være lengre fra hverandre. De ledige CLB'ene inne i makroen kan enten benyttes av andre bussmakroer som gir såkalte nøstede bussmakroer, eller de kan benyttes av logikk fra det statiske designet. Nøstede bussmakroer kan gi båndbredde på opptil 24 bit. En fordel med de brede bussmakroene er at de kan nøsted opp slik at det spares CLB'er i mellom. Se figur 2.13.

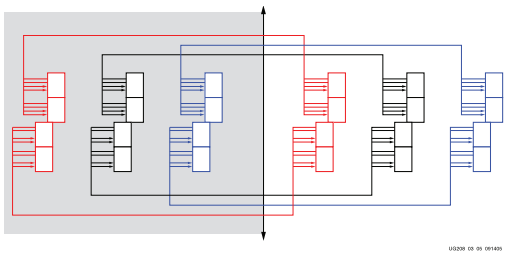


Figur 2.11: Bussmakro [6].



Figur 2.12: Tilkoblingsmuligheter for rekonfigurerbare moduler[16].

PlanAhead et program som brukes til å påvirke og bestemme hvordan logikken skal være fordelt på FPGA. De logiske komponentene i designet blir tildelt plass på det rekonfigurerbare området på bakgrunn av ulike bruker"constraints". Disse begrensningene kan innebefatte både tid og areal. Utover dette



Figur 2.13: Nøstede busmakroer.

2.3.4 Testkort

Som nevnt er masteroppgaven gjort i sammenheng med AHEAD-prosjektet. Test-kortene brukt i tidligere arbeid av Bystrøm[3] og AHEAD har vært av typen Susaku-S og Susaku-V [**?**]. Utgaven av Virtex-4 på testkortet har en PowePC 405 RISC Core ombord som kan operere med opptil 450 MHz. [17]. Kortene kan kjøre Linux på en av de myke prosessorene, MicroBlaze eller PowerPC som kan implementeres på FPGA'ene. Testkortet Suzaku-V vises i figur 2.14.

Figur 2.14: Testkortet Suzaku-V med Virtex-4[**?**].

Virtex-4

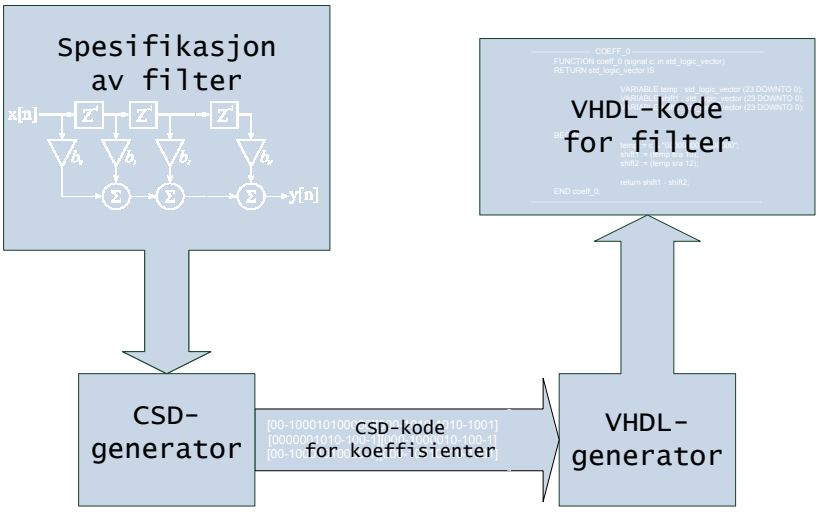
Virtex-4 kan bruke active delvis rekonfigurering. det vil si swappe moduler 'on the fly' mens resten av designet fortsetter å kjøre [6]. Den minste konfigurerbare enheten på Virtex-4 er 16 CLBer høy og 1 CLB bred. De delvise bitfilene beskriver rammer på denne størrelsen slik at å holde den rekonfigurerbare modulen innenfor grensen av disse

rammene vil være gunstig. Dette medfører at færre rammer må rekonfigureres, de delvise bitfilene blir mindre og rekonfigureringstiden blir kortere. Virtex-4 plattformen byr på bedre funskjonalitet og rekonfigureringsverktøy enn hva som finnes for Spartan3. Det er også funksjonalitet som sikrer at dataene ikke endres tilfeldig ved såkalt en 'glitch' i designet rundt det som skal konfigureres. Etter at hoveddesignet er lastet inn i FPGAen, kan mindre deler av designet endres ved at det lastes inn små deler av bitfiler som kan konfigurerere kun et lite, begrenset område av FPGA[6]. Av de ulike konfigureringsportene som finnes er SelectMap, Serial, JTAG og ICAP[6].

2.4 Tidligere arbeid fra prosjektoppgaven

Denne rapporten er skrevet som en videreføring av arbeid fra min prosjektoppgave fra desember 2008. Arbeidet innebar en systemskisse av et dynamisk rekonfigurerbart filter. Systemet har en klient/tjener-arkitektur der klienten kjører filteret, mens tjeneren genererer bitstrømmen for de rekonfigurerbare modulene. Denne delvise bitstrømmen blir generert fra VHDL-kode av et synteseverktøy. VHDL-koden blir generert av et program som leser spesifikasjon av filteret, der spesifikasjonen er laget på bakgrunn av krav fra systemet. Programmet som oversetter spesifikasjonen til VHDL-kode er laget i prosjektet, og innebærer også optimalisering av filterkoeffisientene med CSD-kodede konstantmultiplikatorer. De produserte filtrene har lite areal og ser i ut til å fungere bra.

Programmet som er laget i prosjektet tar inn en spesifikajson i form av den ønskede graden til filteret, samt de ønskede koeffisientverdiene. Programmet regner så ut CSD-kode for verdiene og genererer på bakgrunn av dette VHDL-kode for konstantmultiplikatorene og resten av filteret.



Figur 2.15: Fra spesifikasjon til VHDL-kode.

Kapittel 3

Design og arkitektur

Dette kapittelet er hovedtyngden i rapporten og beskriver ulike designløsninger. Hver løsning er beskrevet med fordeler og ulemper, og til hvilken bruk den er best egnet. Egenskaper diskutert er filterstruktur og partisjonering i moduler. I kapittel 4 beskrives designet som er delvis implementert, simulert og syntetisert i henholdsvis kapittel 4 og 5. Det blir sett noe på hvordan filtrene vil kunne brukes ved dynamiske moduler, men dette er ikke mye vektlagt.

En rekke hensyn må tas ved design av et dynamisk rekonfigurbart filter. Filteret krever et system rundt seg til å holde orden på koeffisientene og rekonfigurering av disse. Systemet er skissert i 3.4, men fokus vil i arbeidet hovedsakelig være rundt filteret og hvordan det best kan fungere i forhold til ulike designkrav. Gjennomgående i arbeidet vektlegges det flexibilitet. Filteretarkitekturen skal kunne tilpasses flere forskjellige behov, ikke bare fungere bra for et enkelt tilfelle. Dette gjelder både før og etter at systemet er konfigurert og satt i gang. Eksempelvis må filteret være skalerbart og fungere med spesifiserbar ønsket grad, bitbredde og presisjon.

Egenskaper som filteret skal kunne optimaliseres mot:

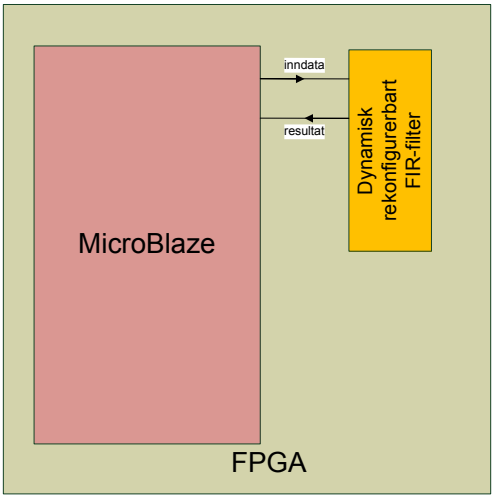
- Adaptivitet
- Lite areal
- Høy ytelse

Filteret vil innenfor et lengre tidsvindu multiplisere med de samme koeffisientene, slik at det er unødvending med bruk av generelle multiplikatorer som er både store og trege. I stedet for disse generelle multiplikatorene tas det i bruk raske konstantmultiplikatorer som krever

lite areal. Problemet blir da å opprettholde adaptiviteten. Filteret skal kunne endre karakteristikk i kjøretid når det oppstår behov. Her kommer dynamisk rekonfigurering kommer inn i bildet.

Bruken av FPGA for implementasjon av filteret muliggjør endringer i hardware og rekonfigurering skal benyttes til å endre konstantmultiplikatorene slik at de utgjør akkurat de koeffisientene som er ønsket. Hvordan slik rekonfigurering kan foregå, tilrettelegges og muliggjøres skal det handle om i dette kapittelet.

Bruken av filteret vil være i innvenvde systemer med mulighet for tilkobling til en ekstern server. Figur 3.1 viser filteret på FPGA sammen med en myk prosessor av typen MicroBlaze eller PowerPC. Filteret skal kunne brukes av applikasjoner som kjøres på prosessoren.



Figur 3.1: Dynamisk rekonfigurerbart filter og MicroBlaze på FPGA.

Det er et designmål i dette arbeidet at filteret skal være anvennelig for mange svært forskjellige formål. Videre vektlegges det at arkitekturløsningene skal ha følgende skalerbare attributter:

- Graden til filteret
- Bitbredde på inngangsignalet
- Bitbredde på koeffisientene
- Presisjon i multiplikasjonen (bitbredde på utgangsignalet)

En sideeffekt ved slik skalerbarhet medfører at økning i de ovennevnte atributtene igjen vil føre til økning av:

- Areal
- Rekonfigureringstid
- Lengste datavei

3.1 Funksjonalitet

Det dynamiske filteret skal være adaptivt, hvilket innebærer at det skal ha evne til å tilpasse seg ulikt påtrykk. Til hvilken grad filteret tilpasser seg avhenger av hvilke funksjonaliteter det på forhånd er tilrettelagt for.

Det blir i **??** presentert løsninger som har en eller flere slike funksjoner. Disse innebærer mulighet for endring av:

- én enkelt koeffisient av gangen
- hele settet med koeffisienter
- graden til filteret.
- hele filteret.
- ekspanderender multiplikatormoduler.

Slik funkskjonalitet krever et støttesystem som vil bli forklart i 3.4.

For ethvert hardwaresystem vil det være mange avgjørelser som gjøres på bakgrunn av flere forskjellig designparametere. Da systemet i dette tilfellet ikke har noe veldefinert bruksområde, men utvikles for bruk i flere forskjellige sammenhenger, er det beskrevet flere ulike løsninger for optimaliseringer i større og mindre grad mot følgende spesifikasjonskrav:

- Adaptivitet
- Arealforbruk
- Ytelse
- Rekonfigureringstid
- Presisjon
- Bitbredde

Noen av disse parameterene må det optimaliseres mot i forkant av kjøretid for systemet, mens andre kan endres *under* kjøretid.

3.1.1 Endring av koeffisienter

For at filteret skal være dynamisk kreves det at filterkoeffisientene skal kunne endres. Siden det er multiplikatorene som utgjør koeffisientene, og disse er konstantmultiplikatorer som ikke kan endres, er det multiplikatorene som må rekonfigureres. Registerene og addererene som utgjør akkumuleringen som er statiske mellom hver konfigurasjon. Det eneste som endres ved konfigurering av en koeffisient er multiplikatorene. Konstantmultiplikatorene som vil bli brukt er beskrevet i 3.3.

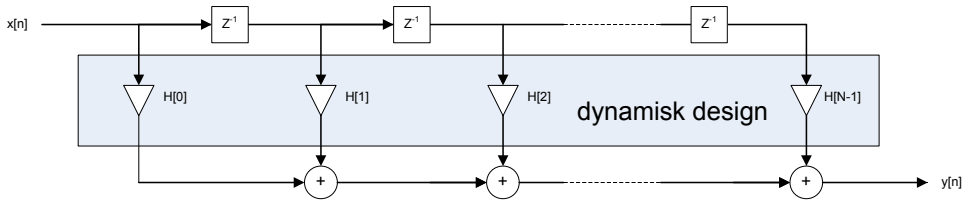
3.2 Oppdeling i statisk og dynamisk design

I dynamisk rekonfigurerbare systemer er det kritisk hvordan designet oppdeles i dynamisk og statisk design. Dette kapittelet handler om hvordan oppdelingen gjøres. For minst mulig rekonfigureringstid er det viktig at arealet som skal rekonfigureres er minst mulig. Alle tilkoblinger til det dynamiske designet, untatt klokke og reset, må gå gjennom bussmakroer. Fordi de dynamiske modulene vil være små er ikke arealkostnadene for bussmakroene neglisjerbare. Bitbredde og størrelse på de rekonfigurerbare modulene er derfor viktige designparametere ved oppdeling i statisk og dynamisk design.

3.2.1 Minst mulig rekonfigurering

Når det kreves endringer i settet med filterkoeffisientene, er det kun multiplikatorene som er forskjellige. For at rekonfigurerinstiden skal bli kortest mulig, bør minst mulig av filterdesignet ha behov for rekonfigurering.

En naturlig oppdeling av filteret ved første øyensyn vil være slik som i figur 3.2. Multiplikatorene tilhører her det dynamiske designet, mens øvrige komponenter blir tilhørende det statiske designet da de vil være like for en ny filterspesifikasjon, og det er en fordel å unngå å konfigurere disse på nytt. Oppdelingen er imidlertid ikke en fullt så enkel affære da det er en rekke ulike hensyn som må tas i betraktning.



Figur 3.2: Ved endring av filterkoeffisientene er det kun konstanmultiplikatorene som krever rekonfigurering.

3.2.2 Oppdelingshensyn

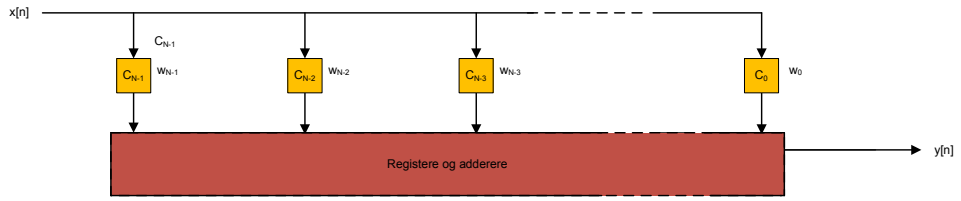
Hensyn som må tas ved beslutning om hva som skal være med i det dynamiske designet er rekonfigureringsmetode, rekonfigureringstid, kostnader ved ulik oppdeling, filterspesifikasjoner, ruting, sammenkoblinger og nivå for hvor mye som ønskes rekonfigurert. Oppdeling av designet kan gjøres på mange måter. Kravet at delene skal kunne kommunisere med hverandre uten for mye ekstra logikk og forsinkelse.

Rekonfigureringsmetode

Delvis rekonfigurering kan med hardware og verktøy fra Xilinx gjøres hovedsakelig ved hjelp av to forskjellige metoder, slik som beskrevet i **??**. Bruk av differensialbasert delvis rekonfigurering er ikke aktuell da den krever at en *før* rekonfigurering vet hvordan alle de delvise bitfilene skal være. Dette er fordi en ikke har kontroll på rutingen av det statiske designet som en risikerer at har ruting gjennomgående det dynamiske designet, med fare for å bli korrupt.

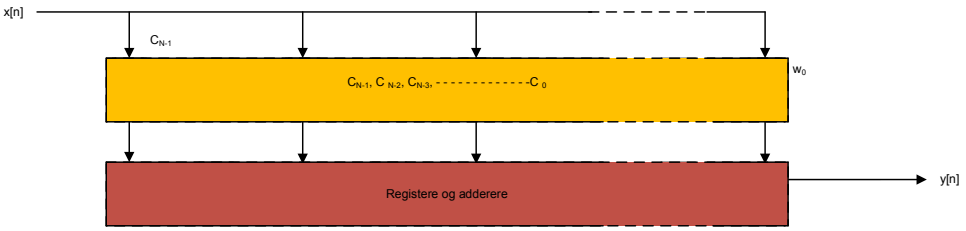
PR-moduler

Med en begrensning som nevnt over fosvinner adaptiviteten til systemet, og metoden vi gjenstår med er modulbasert delvis rekonfigurering. Denne metoden innebærer oppdeling i PR-moduler som kan rekonfigurereres uten fare for å ødelegge ruting innenfor modulen. Størrelsen på modulen fastsettes før kjøretid, og må være så stor at den rommer det største en vil kunne bytte med. Figur 3.3 viser hvordan Hver av multiplikatorene kan deles inn i egne moduler. Hvorfor filteret er transponert forklares i 3.5.



Figur 3.3: Transponert filter med multiplikatormoduler.

Modulene krever imidlertid bruk av bussmakroer for kommunikasjon med det statiske designet. Fordi filteret har relativt lite areal, utgjør bussmakroene dermed nok en kostnad som skal tas i betraktning ved oppdeling. Derfor er det grunn til å vurdere nødvendigheten av mange små dynamiske moduler fremfor få store. Hvor stor betydning bussmakroene har vil bli utforsket i kapittel 5. Få store PR-moduler har igjen en fordel fordi de sperrer for mindre ruting enn de store. Figur 3.4 viser alle filterkoeffisientene i én stor modul. Mer om fordeler ved store moduler blir beskrevet i avsnitt **??**.



Figur 3.4: Transponert filter én stor multiplikatormodul.

Bussmakroer

Typen bussmakro som kan brukes på innsignalet kan være smal hvis størrelsen på PR-modulen går opp med plassering i forhold til oddetalls- og partallsslicer på FPGA'en.

Bussmakroene har til hensikt å tjene som koblinger mellom de modulene som skal rekonfigureres og vil bli benyttet mellom de ulike modulene i filteret. Bussmakroene er beskrevet i 2.3.3. I praksis er bussmakroene implementert som svart bokskomponenter som består av enkel logikk som ruter signalet fra inngangen videre til utgangen.

Kostnaden av bussmakroene er 2-4 CBL'er hvilket betyr en relativt stor økning i arealet for små konstantmultiplikatormoduler. For å designe

filteret slik at det skal ha minst mulig areal, er det derfor nødvendig å benytte færrest mulig mussmakroer. Hensyn til forsinkelsen av signaler gjennom bussmakroene er ikke medregnet i filterdesignet.

3.3 Multiplikatorer

Fordi filteret skal kunne tilpasses ulike forhold, og kreves det et antall forskjellige sett med koeffisienter. Multiplikatorene i filteret hardkodes som konstantmultiplikatorer beskrevet i 2.2.1. Disse har lite areal, vil være rent kombinatoriske kretser uten klokke og er raske i forhold til generelle multiplikatorer.

3.3.1 CSD-kode

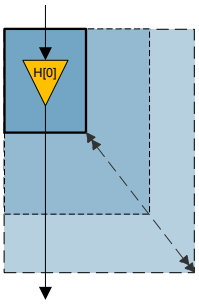
Algoritmen for CSD-kode i **??** vil bli brukt for generering av multiplikatorene. Valget av algoritme er gjort på bakgrunn av at den gir arealmessig små konstantmultiplikatorer samtidig som den er forholdsvis grei å implementere. I prosjektet er det implementert programmvare for generering av slike multiplikatorer. Programvaren vil kreve noe omskriving slik at den nye filterarktitekturen fra arbeidet i denne rapporten.

3.3.2 Variabel multiplikatorstørrelse

Konstantmultiplikatorene har varierende størrelse som kjent fra 2.2.1. Antallet operasjoner i konstantmultiplikatorene varierer med antallet 1'ere i CSD-koden. Med antall operasjoner øker derfor også arealet for konstantmultiplikatorene. Når bitbredden på koeffisientene øker, vil samtidig den gjennomsnittlige forskjellen mellom største og minste multiplikator øke.Økningen får konsekvenser når filterdesignet skal inndeles i dynamisk og statisk design.

3.3.3 Multplikatormoduler

En multiplikatormodul som byttes i kjøretid skal erstatte en modul som allerde er i bruk. Den nye PR-modulen må derfor ha mindre areal enn den som det er der fra før. Når det før kjøretid bestemmes hvor



Figur 3.5: Størrelsen på multiplikatorene varierer avhengig av antall 1'ere i CSD-koden.

stor PR-modulen skal være, må den defineres slik at den rommer den modulen som har det største arealet nødvendig for ønsket funksjon.

Fordi forskjellen mellom konstantmultiplikatorer kan være stor, har det i en tidlig fase i arbeidet vært en idé å utnytte denne plassen som en ressurs som kan brukes til logikk for andre applikasjoner. Mer om dette i 3.3.4.

Én modul for hver multiplikator

Dersom filteret skal ha samme grad under hele kjøretiden vil det fungere med én modul per multiplikator. Hver modul må være så stor som den største multiplikatoren kan bli og alle koeffisienter vil da rommes i denne modulen. Hvor stor modulen må være for å romme alle multiplikatorstørrelser utforskes i kapittel 5. Fordelen med et slikt filter er at det er meget flexibelt med tanke på at det er mulighet for finpussing av filteret ved kun å endre én eller noen få koeffisienter.

Én stor multiplikatormodul

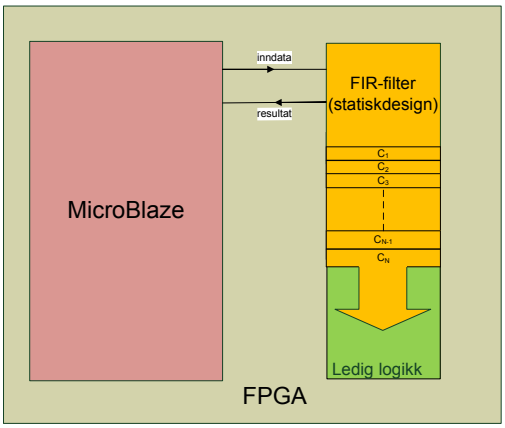
Et aleternativ til mange små dynamiske moduler er én stor modul som rommer alle multiplikatorene. Her må bør det også settes av nok plass til det arealmessig størst tenkelige settet av filterkoeffisienter. En stor fordel ved én stor fermfor mange små moduler er at det er større rom for ytterligere optimalisering av konstantmultiplikatorene. Et eksempel på dette er vist i figur 2.5.

3.3.4 Utnyttelse av varierende multiplikatorstørrelse

Fordi variasjonen i størrelsen på multiplikatorene er stor, og det ofte er mye ubrukt logikk i PR-modulene, var det på et tidlig stadium i arbeidet en idé å gjenbruke denne rekonfigurerbare logikken som blir ledig når det brukes en liten konstantmultiplikator i forhold til en stor.

En utfordring er at når et område av filteret brukes av en annen applikasjon, må dette område kunne kreves tilbake av filteret ved behov. Et overordnet system må ta seg av dette og benytte en slags form for schedulering av de ledige FPGA-ressursene. Dette er et omfattenende og komplisert arbeid som det ikke gås inn på her, men som er utforsket i [2]. Figur 3.6 viser er grov skisse av filteret med mulighet for ekspansjon.

For å gi multiplikatorene mulighet til å ekspandere vekk fra det statiske designet uten hindring, ble det undersøkt alternative metoder for representasjon av filteret. Transponert dirkte form som i 2.2 kan benyttes til dette.



Figur 3.6: System med ekspanderende filter.

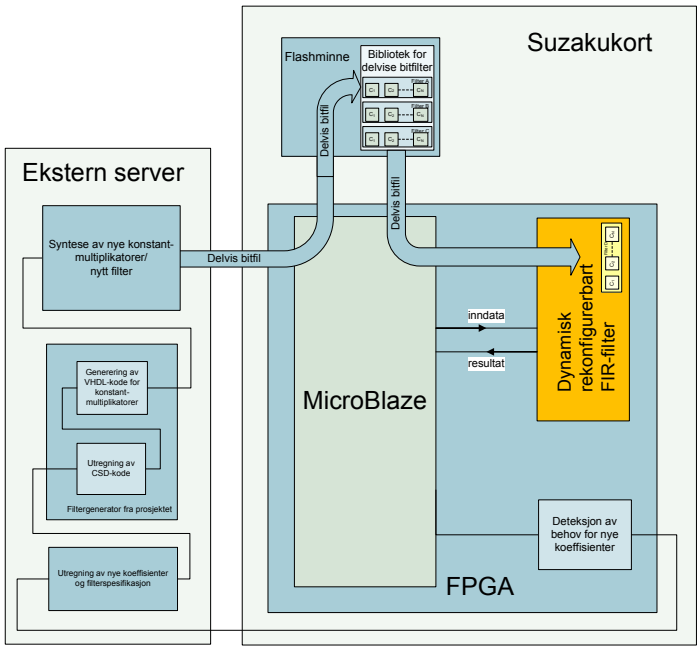
Hvor stort potensialet er for utnyttelse av den variable konstantmultiplikatorstørrelsen er nyttig å vite før det blir gått videre inn på detaljer rundt dette. I kapittel 5 blir denne variable størrelsen utforsket. I det resterende arbeidet blir det for enkelhetsskyld operert med fast arealstørrelse på PR-modulene.

3.4 System med dynamisk rekonfigurerbart filter

Spesielt når det gjelder å oppnå adaptivitet, kreves det et støttesystem. Oppgavene til et slikt systemet vil kunne være å:

- bytte ut koeffisienter.
- kalkulere
  - nye koeffisienter.
  - nye multiplikatormoduler.
  - graden til nytt filteret.
- Ta i bruk ledig plass ved små/ledige multiplikatormoduler som benyttes til maskinvareaksellererte oppgaver.

Noen av disse vil kunne utføres på ved hjelp av en myk prosessor som PowerPC eller MicroBlaze ombord i FPGA. Figur 3.7 viser systemet for det dynamiske filteret.



Figur 3.7: System med rekonfigurerbart FIR-filter.

Filteret får lastet inn nye konfigureringer fra et ferdigsyntetisert bibliotek med delvise bitfilter som ligger lagret i flashminnet. Et bibliotek med konstantmultiplikatormoduler kan eventuelt lagres i en database i den eksterne delen av systemet. Rekonfigurering skjer enten via MicroBlaze eller via rekonfigureringsmodul som ligger på kortet, eller på selve FPGA'en. Slike moduler er beskrevet i **??**.

Til støtte for det dynamiske filteret kreves et system til å holde orden på og utføre oppgaver. Blant disse er følgende:

- Konfigurere FPGA
- detektere behov for nye koeffisienter
- generere
- lagre bitfiler for ny konfigurasjon av konstantmultiplikatorerer

Dersom det kreves flere delvise bitfiler med konstantmultiplikatorer enn det er plass til i biblioteket men dette tar lang tid og nødvendinggjør kjøring av tunge programmer som krever datakraft for å generere nye dbitfiler. En ekstern server vil være nødt til å ta seg av dette.

Det gjelder å finne den løsningen som er optimal for sitt formål. Derfor har jeg sett for meg et system der optimalisering tas med i genereringen av filteredesignet. Avsnitt **??** kommer tilbake til dette.

3.5 Filterstruktur

Utgangspunktet i denne oppgaven er den tradisjonelle fremstillingen av FIR-filteret, direkte form FIR-filter slik som i figur 2.1.

Det blir her vurderert filtere på både direkte form og transponert form som i figur 2.2, som ikke endrer overføringsfunskjon eller oppførsel ved overgang fra det ene til det andre. Likevel endres noen egenskaper som vil bli utforsket og utnyttet. Hver av typene blir også vurdert i forhold til individuelle multiplikatorer og én stor multiplikatormodul. Noe som er viktig i forhold til bruk av dynamiske moduler er muligheten for ekspansjon i én eller flere retninger. Det transponerte filteret gir større mulighet for ekspansjon vekk fra det statiskedesignet, uten fare for fragmentering av ledig logikk når det brukes mange små moduler. For filterene beskrevet under må den høyest ønskelige graden bestemmes før kjøretid.

3.5.1 Dynamiske modulstørrelser

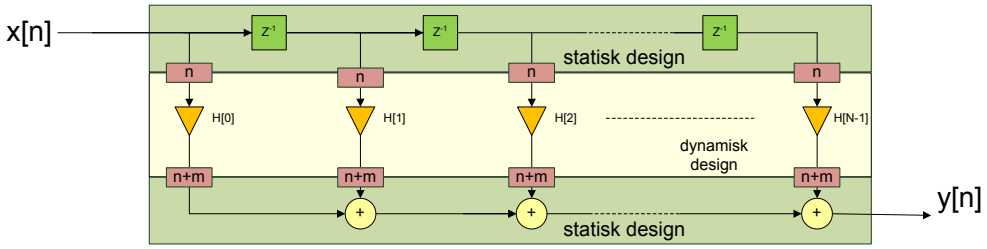
Dersom filteret skal ha dynamisk størrelse og individuelle multiplikatormodulener, vil det være fordelaktig å bruke det transponerte filteret. Dette gir muligheter for at multiplikatormodulene kan ekspandere/kontrahere vekk fra hverandre, og ikke vil kollidere med hverandre

eller skape fragmentering i av den ledige plassen. Det antas at modulene må ha et fast tilkoblingspunkt.

For et filter på direkte form med multiplikatorene i én modul vil det også være mulig å slå sammen det statiske designet slik som i figur 4.5. Denne har mulighet for ekspansjon i tre retninger ved eventuell bruk av DPR-moduler.

3.5.2 Direkte form, tre moduler

Filteret deles opp på intuitivt vis i tre deler, en for hver type komponent. Filteret vises i figur 3.8.

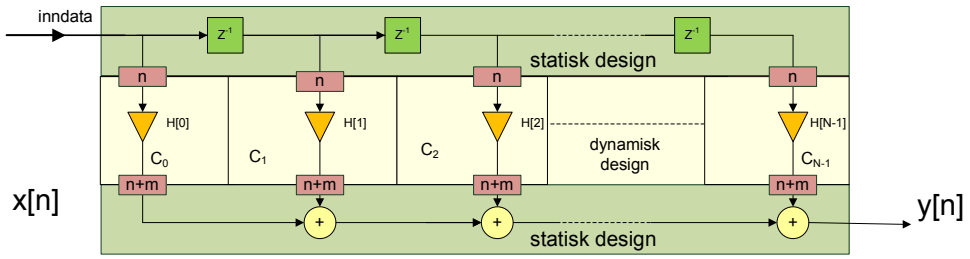


Figur 3.8: Dirkte form FIR-filter oppdelt i tre moduler.

(3.1)

Som det kan sees av figuren blir kostnaden i bussmakroer BM = N*((a/8) + (a+b)/8), der a og b er antall bit på innsignalet og antallet bit i koeffisienten. N er filterets orden og det regnes med bussmakroer som er 8 bit brede.

Direkte form, individuelle multiplikatormoduler



Figur 3.9: Dirkte form FIR-filter oppdelt i individuelle multiplikatormoduler.

3.5.3 Transponert filter

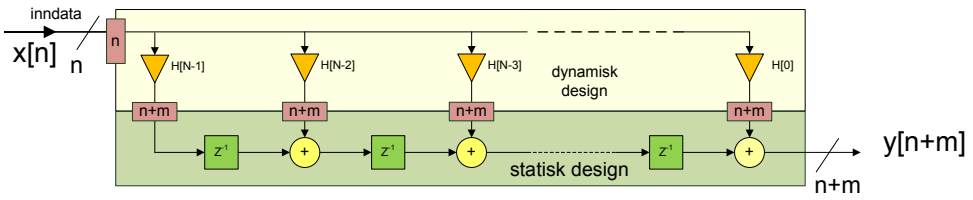
Fordi arealet og kostnaden av å bruke bussmakroer til å overstyre rutingen er så stor som det er, er det ønskelig å minimere antall bussmakroer. Å slå sammen multiplikatorene gir færre bussmakroer for et transponert filter.

Transponert, to moduler

En løsning for å få ned antallet overføringer er ved å dele opp i to moduler i stedet for tre.

Transponering av filteret er en løsning som gjør at filteret kan bli bestående av to deler

Med det transponerte filteret i figur 2.2 er plasseringen slik at multiplikatorene kan komme for seg selv i forhold til addererene og registrene.



Figur 3.10: Transponert filter med én stor multiplikatormodul og bussmakroer.

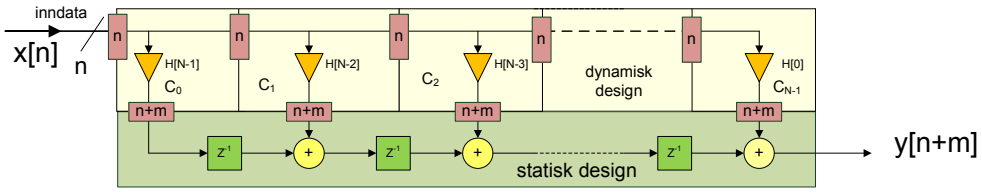
Dette gir større muligheter for utnyttelse av den variable multiplikatorstørrelsen, men krever at det, som nevnt, er et system som tar seg av fordeling av ledige ressurser.

Kostnaden i bussmakroer blir N*(a+b)/8 + a/8.

Videre vil det være mulig å dele opp multiplikatormodulen i flere moduler slik at en enkel modul kan rekonfigureres.

Transponert, individuelle multiplikatormoduler

Kostnaden i bussmakroer er samme som for direkte form.



Figur 3.11: Transponert filter med bussmakroer og individuellle multiplikatormodul er.

3.5.4 Sammenligning av filterene

En stor multiplikatormodul vil ha mindre dødplass enn de små og reduserer sjansen for defragmentering. Optimaliseringsmulighetene er da også større. Viktig å merke seg er at transponert form krever større registere enn direkteform fordi registrene er etter multiplikasjonen.

At multiplikatorene ikke er mellom registrene og addererene er og så viktig av den grunn at det da er mulig for multiplikatorene å vokse i størrelse i retning fra resten av filteret. En stor modul for alle konstantmultiplikatorene vil gi best mulighet for optimalisering. Likevel blir individuelle multiplikatormoduler mest vektlagt for det videre arbeidet. Disse er av større interesse for utforskingen av dynamisk rekonfigurerbarhet. Individuelle moduler vil også være mer hensiktsmessig dersom det skal lages et bibliotek med mange små multiplikatorer hvilket vil gi større grad av flexibilitet ved gjenbruk av moduler. Flere moduler gir også større mulighet for eksperimentering med utplassering på FPGA.

3.6 Optimalisering

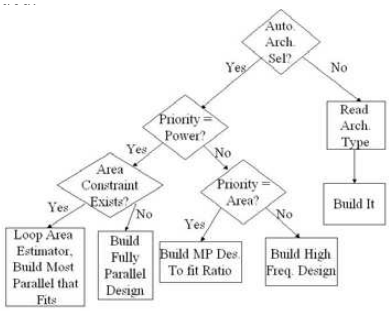
Ved filterdesign er det alltid ulike krav til optimalisering. Noen filtre krever rask respons, noen krever høy presisjon mens andre igjen krever

lite areal eller stor flexibilitet. De to siste er legges det ekstra stor vekt på i dette arbeidet, i og med at målet er å kombinere optimalisering mot areal, men likevel beholde flexibiliteten.

Valg av filter før kjøretid

Dersom de ulike beskrevne filtrene viser seg bedre i forhold til hverandre med hensyn på areal, hastighet eller rekonfigureringstid, men likevel konkurransedyktige, kan de eventuelt implementeres, sammen med annen optimalisering, som rekonfigurerbare moduler som settes inn avhengig av om det er ønske om optimalisering mot areal energiforbruk, ytelse eller rekonfigureringstid. Konfigurering av og bytte mellom slike filtermoduler må eventuelt skje før kjøretid.

Valgene kan gjøres av systemet før kjøring ved at det etter påtrykk av preferanser tas ulike valg slik som beskrives i [10]. Figur 3.12 viser et optimalsieringstre for slike valg. Dette er også lignenende Xilinx FIR-Compiler, beskrevet i kapittel 2.



Figur 3.12: Valgtre for optimaliserte arkitekturvalg.

Optimalisering av stormodulen

En mulig optimalisering mot lite areal er ved kjennskap til sannsynligheten for hvor stor summen av arealet for alle konstantmultiplikatorene er. Dersom de største multiplikatorene utgjevnes av de små, vil det kunne settes rammebegrensninger for størrelsen av PR-modulen som er mindre enn det størst tenkelige tilfellet. Problemet blir dermed hvordan takle nettopp de største tilfellene. Løsning da må bli ved en eller annen form for trunkering. På dette viset er det god nøyaktighet for de fleste tilfellene av sett med koeffisienter, mens det andre ganger

må kunne tollereres feil. Hvordan dette kan tollereres er applikasjonsavhengig.

Registermodul og multiplikatormodul med adderere

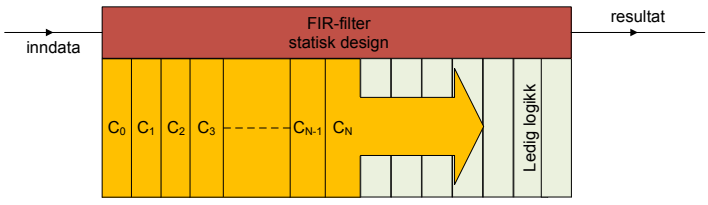
Oppdeling i multiplikatormodul med adderere og registermodul kan brukes for å få ned antallet bit som overføres mellom dynamisk og statisk design, og dermed også antallet bussmakrorer. Filteret deles da inn i to moduler der den ene inneholder alle registrene, mens den andre inneholder multiplikatorene og de akkumulerende addererene. Dette resulterer i N*n bit overført, der N er graden til filteret, mens n er antall bit i innsignalet x. Det økte arealet av PR-modulen som nå innlemmer flere adderee krever imidlertid lengre rekonfigureringstid.

Funksjonalitet og adaptivitetet

Endring av filterorden

En mulighet for å endre filterets orden ved rekonfigurering kunne vært å ha en stor, dynamisk multiplikatormodul, som hadde et antall bussmakroer tilkoblet. Antallet må være så stort som høyeste spesifiserte mulige orden for filtrene. Endring av graden til filteret med én stor modul.

Ved å ha moduler slik som [18], beskrevet i **??** vil det ved filtere med lav orden finnes ubrukte moduler. Dersom det statiske designet i filteret har den nødvendige kontrollogikken, vil disse modulene kunne konfigureres med hardwaremoduler. En skisse av filtret er vist i figur 3.13.



Figur 3.13: Filter av varierende orden.

Kapittel 4

Implementering

Før videre design av systemet kan utføres kreves det kjenskap til metodikk og utfordringer ved implementering på FPGA. God plassering og ruting av logiske design på FPGA er eksperimentelt preget. Dette kapittelet beskriver derfor implementering av 3 filtre som skal gi innsikt i videre design av det dynamisk rekonfigurerbare systemet. Oppgavene til den eksterne serveren til venstre i figur 3.7 i det dynamisk rekonfigurbare systemet kan gjøres manuelt under utvikling av selve systemet. Siden filteret er den viktigste delen av systemet er det derfor denne delen som er valgt ut for implementering og utforskning før det neste steget ville vært å teste filteret sammen med en myk prosessor på testkortet. Et fungerende system for et dynamisk filter er ikke implementert.

De tre utvalgte filterene skal hovedsakelig gi kjennskap til og erfaring med

- transponert form FIR-filter i forhold til direkte form.
- bruk av bussmakroer.
- arealforbruk på FPGA.
- implikajsoner ved utlegg på FPGA.

Forventede resultater er arealforbruk på FPGA, og maksimal frekvens. Filterspesifikkasjonene er ikke laget for funksjonell bruk og filtret vil er derfor kun bli simulert for en initiell verifikasjon før videre syntese mot FPGA. Utviklingsverktøy brukt i forbindelse med implementering er Xilinx ISE 11.1 og PlanAhead 11.1. Målplatform for implementasjonene er Virtex-4 på utviklingskortet Suzaku-V. Det tas utgangspunkt i Early Access Partial Reconfiguration verktøyet når det gjelder designmetodikk

4.1 Implementering av filtre

4.1.1 Valg av filtre for implementering

Filtere med individuelle konstantmultiplikatormoduler har blitt implementert. Valg av filter er gjort dels på grunnlag av hvilken løsning som trolig er den beste, dels utifra hva som lar seg implementere på en grei måte samtidig som det gir avkastning i form av testresultater og erfaring med bruk av flere dynamiske moduler.

For å kunne si noe om hvilket FIR-filter som fungerer bedre av direkte form transponert form, har det blitt implementert filtre av begge typer. For å kunne si hvilken kostnad bussmakroene har for systemet, er det implementert filtre både med og uten bussmakroer. Tre filtre vil kunne teste disse to egenskapene. Filtrene er navngitt `df_im_bm`, `df_im_ub`, `tf_im_ub`. Oversik over filtrenes egenskaper er vist i tabell 4.1.

Tabell 4.1: Egenskaper for implementerte filtre

	Direkte form	Transponert form
Uten bussmakroer	`df_im_ub`	`tf_im_ub`
Med bussmakroer	`df_im_bm`	-

4.1.2 Filterspesifikasjoner

Filterene som implementeres er av 3. grad, hvilket betyr 4 koeffisienter. Få koeffisienter ble valgt av enkelhet fordi utviklingen av filteret byr på en del prøving og feiling i designvektøyene. Koeffisientene til testfilteret valgt blant koeffisientene i filteret i [5]. Koeffeisienene utgjør ikke et fornuftig filter, men er valgt ut for å få et variert antall operasjoner i konstantmultiplikatorene.Filterresponsen er derfor ikke av stor interesse, men er simulert for filter I1 i avsnitt 5.

Bitbredde på innsignalet på filtrene er spesifisert til 8 bit. Koeffisientverdiene er 16 bit brede, slik at det på utgangen av konstantmultiplikatorene vil være $8 + 16 = 24$ bit. Dette følger av ligning 2.3.

Tabell 4.2: Filterspesifikasjon for implementasjon.

k	$h(k)$	$h(k)$ *avrundet*	$h(k)CSD$ / $h(k)hex$	add	sub	tot. add/sub
0	-0.0057534026	-575	0000 00$\bar{1}$0 0$\bar{1}$00 000$\bar{1}$/fdc1	1	2	3
2	0.030410913	30411	1000 $\bar{1}$00$\bar{1}$ 0$\bar{1}$01 0$\bar{1}$0$\bar{1}$/76cb	2	5	7
3	0.00099026691	99	0000 0001 0$\bar{1}$00 010$\bar{1}$/0063	2	2	4
4	-0.12024988	-12025	0$\bar{1}$01 0001 0000 100$\bar{1}$/d107	3	2	5

Filterkoeffisient $k(0)$, $k(1)$, $k(2)$ og $k(3)$ i tabellen under tilsvarer henholdsvis $k(31)$, $k(15)$, $k(30)$ og $k(18)$ fra filteret i vedlegg **??**. Filterkoeffisientene er også generert med CSD-generatoren, og er er kontrollert for overenstemmelse.

Figur 4.1: Konstantmultiplikator med 16-bit koeffisient, 8-bit innsignal og 24-bit resultat.

4.2 Implementering i VHDL

Implementeringen er eksperimentelt preget, og gir mer en pekepinn på hvordan filteret lar seg syntetisere. Individuelle konstantmultiplikatorer vil være de dynamiske modulene. PR-modulene er kombinatoriske og har ingen klokkeinngang. Klokkesignaler og reset behøver defor ikke gå gjennom bussmakroer[18]. Bussmakroene kan lastes ned som ferdige moduler fra Xilinx.com og kan legges til i designet som en egen "svart boks"-komponent i VHDL-koden.

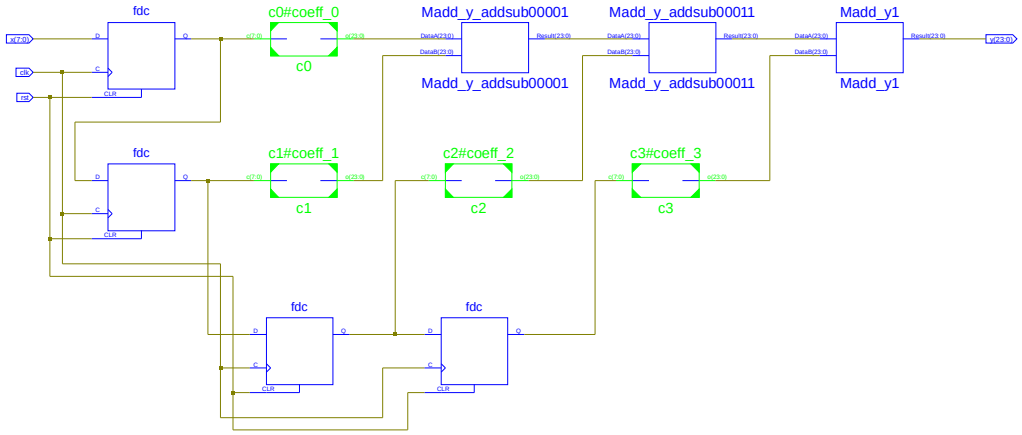
Filtrene er implementert noe motstridende desigreglene for modulbasert delvis rekonfigurering da det statiske designet ikke er innkapslet i en egen modul. Mer om dette kommer i diskusjonen i kapittel 7.

4.2.1 Direkte form, uten bussmakroer

Utgangspunktet for implementasjonen av filtrene i VHDL er Filtergenerator fra prosjektet. Med dette programmet ble det generert et filter i VHDL med spesifikkasjonenen i tabell 4.2. Input i CSD-generatoren

var $h(k)avrundet$. Generert CSD-kode ble sjekket for korrekthet mot tabellen.

Endringer i VHDL-koden var nødvendig for implementering av filteret med PR-moduler. Konstantmultiplikatorene som fra `filtergenerator` var representert som `function` måtte endres til `component` og kobles sammen med resten av filteret. Komponentene ble gjort sensitive på innsignalet x, og er rent kombinatoriske kretser. Filteret er implementert uten bussmakroer, og er i praksis et fullstendig statisk design. Figur 4.2 viser filteret på RTL-nivå, slik det er generert fra Xilinx ISE 11.1. Det er et ekstra register på inngangen av filteret til forskjell fra filteret i figur 3.9.



Figur 4.2: RTL-beskrivelse av filteret på direkte form, med individuelle multiplikatormoduler.

4.2.2 Direkte form, med bussmakroer

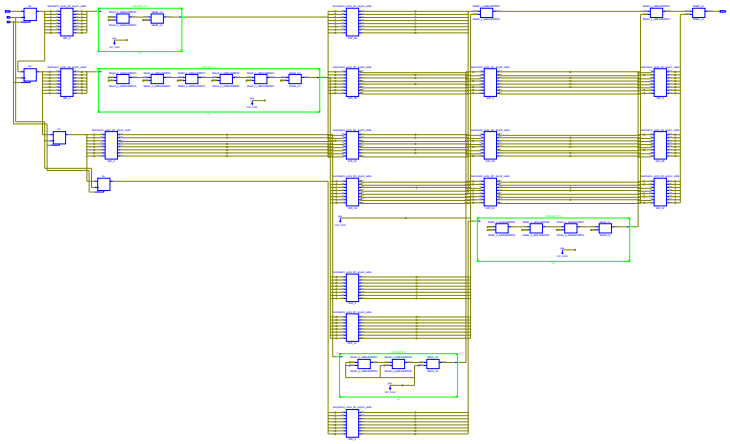
Forskjellen på dette filteret og det forrige er innføring av bussmakroer på tilkoblingen til konstantmultiplikatormodulene som skal være dynamiske. Filterets VHDL-kode tar utgangspunkt i koden fra `df_im_um`.

Bitbredden på inngangssignalet som er satt til 8 bit og passer bredden til en bussmakro. Smal bussmakro kan brukes gitt at PR-modulen kan plasseres slik at det går opp med partall-/oddetall-slice på PR-modulen. Utgangsignalet på 24 bit passer med tre bussmakroer som kan nøstes sammen ved bruk av brede type slik som vist i figur 2.13. Det blir

benyttet asynkrone bussmakroer fordi konstantmultiplikatormodulene er kombinatoriske.

PR-modulen skal ha arealbegrensninger, mens det statiske designet er forventet optimalisert over resterende ledig areal. Dette strider i midlertid imot designreglene for delvis rekonfigurering som krever at det statiske designet innkapsles i en modul som også gis arealbegrensninger. Nyttige resultater forvententes likevel fordi filteret på dette stadiet ikke er ment for faktisk bruk, og synteseresulatat fortsatt vil gi et brukbart arealestimat. Filteret gjenbruker moduleme med konstantmultiplikatorer fra filteret uten bussmakroer.

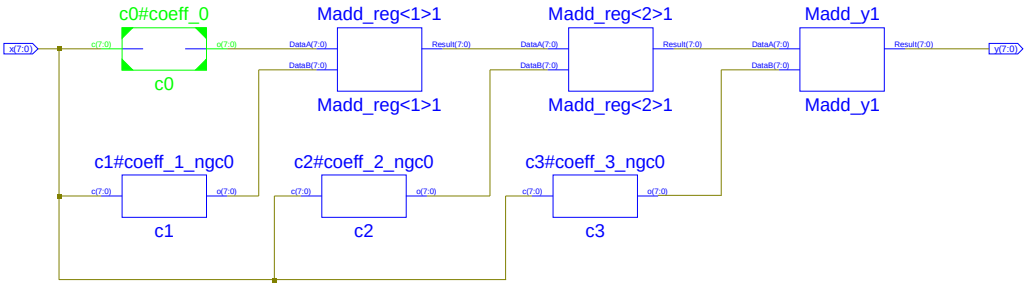
RTL-beskrivelse av filteret på direkte form, med bussmakroer i figur 4.3.



Figur 4.3: Direkte form, individuelle multiplikatormoduler og bussmakroer.

4.2.3 Transponert form, uten bussmakroer

Implementering av filteret i VHDL innebar noe gjenbruk og noe endring av kode fra 4.2.1. Komponentene er de samme, men er koblet sammen og akkumulert og sendt gjennom registre etter konstantmultiplikatormodulene i henhold til spesifikasjon. Filteret er vist på RTL-nivå i figur 4.4.



Figur 4.4: Transponert, moduler.

4.3 Plasering av moduler i PlanAhead

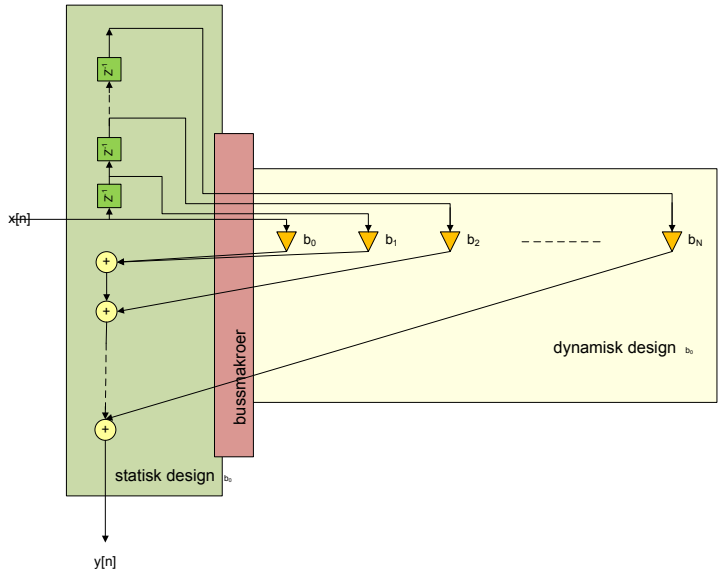
Hver modul i filteret må gis area-constraints slik at de alltid vil være på samme sted og dermed kan plugges ut og inn.

Filteret burde vært implementert med en toppmodul som inneholder moduler for både det statiske designet og det dynamsike designet med bussmakroer som binder dem sammen. En tilsvarende toppmodul kan sees i figur **??** og 4.5. Toppmodulen har I/O som skal kobles mot MicroBlaze, men som for enkelhets skyld er koblet til I/O-pinnene på FPGA for å gi synteseresultater siden det ikke testes med MicroBlaze i denne omgangen. Dette medfører noe unøyaktighet i arealberegningene.

Det er ønskelig med synteseresultater som viser hvordan filterkoeffisientene endrer størrelse med endret verdi. Syntese av et utvalg konstantmultiplikatormoduler inkludert worst-case vil gi svar på hvor store de PR-modulene må være.

Det statiske designet for direkte form med én stor multiplikatormodul kunne enten vært gjort som i figur 3.8, eller de to statiske modulene kunne vært slått sammen til en. En slik sammenslåing ville ikke vært mulig for et filter med individuelle mulitplikatorer.

En skissee av hvordan filteret på direkte form kan oppdeles er vist i figur 4.5. Slik vil det også se ut i toppmodulen til filteret. Filteret i figuren har samme egenskaper som filteret i figur 3.8.



Figur 4.5: Direkte form filter med én modul for multiplikatorer og én modul for statisk design.

Kapittel 5

Syntese og simulering

Utfyllende simuleringer av filtert er ikke gjennomført da det i første omgang er ønskelig å teste funskjonliteten til filteret. Før filteret kan benyttes i forbindelse med det dynamiske systemet eller i andre sammenhenger gjenstår det en grundig uttesting.

Det forventes også svar på hvilket av de utvalgte filtrene som har minst areal til sammen og hvilket som har de minste rekonfigurerbare modulene.

Hvilket av de utvalgte filtrene som har minst areal til sammen, og for de ulike moduelene forventes det å bli funnet noen svar på ved syntese.

5.1 Simulering av filter

Filteret som simuleres er `df_im_ub`. Dette filteret er det enkleste å starte med da det ikke har bussmakroer og er derfor er det minst kompliserte. Simulering er gjort er for å få en initiell verifikasjon på funksjonalitet under implementering i VHDL og utlegg på FPGA.

Programvare for simulering er har vært Modelsim SE 6.3f, revision: 2008.02 av 28. februar 2008. Testbenken som filteret er simulert med er laget med utgangspunkt i testbenken fra [3]. Signaler påtrykkes inngangen `x` på filteret, og kan leses ut på utgangen `y`. For å vite om multiplikatorene gir riktig resultater, er det påtrykt testverdier. Testvektorer påtrykt er tallene 0 til 8, samt tallene 253, 254 og 255.

Påtrykk og resultat fra konstantmultiplikatorene vil være synlig i bølgediagrammet for hver av multiplikatorene slik at det er mulig å verifisere korrekt produkt. Verifisering av bølgediagrammet gjøres ved manuell utregning.

Etter verifikasjon av dette filteret vil det for de andre filtrene kunne simuleres med egne VHDL-filer som simulerer bussmakroer.

5.2 Syntese av filtre

Filtrene er forsøkt gitt arealbegrensninger i PlanAhead. Uten tilgang på delvis rekonfigurerin i programmet skulle dette gi et estimat av arealet forbrukt ved å ha moduler. Dette er gjort ved å sette multiplikatormodulene som partisjoner. Fullstendig utlegg er i midlertid ikke fullført. I stedet er filtrene syntetisert med optimalisering over hele FPGA, uten bruk av arealbegrensninger.

5.2.1 Designverktøy

Det naturlige valget av designverktøy for Xilinx sine FPGA'er er Xilinx ISE og PlanAhead. Forfatteren lykkes kun å få tilgang til dokumentasjonen til den begrensede versjonen av Xilinx ISE som inneholder Partial Reconfiguration Tool har støtte for delvis rekonfigurering. Derimot lykkedes forfatteren i å få tilgang til *PlanAhead 10.1.x* og *Early Acess Reconiguration Software for ISE 9.2i* med en patch for *Early Access Reconfigureation Tool*[6]. Fordi patch'en er til bruk med de eldre versjonene ble designet i stedet gjort med den sist oppdaterte tilgjengelige programvaren. Slik vil overgang bli enklest mulig ved en eventuell tilgang på den begrensede programvaren. Overgang vil også bli enklere til Xilinx ISE 12 vil som skal inneholde funksjonalitet for delvis rekonfigurering.

Siste tilgjengelig versjon, Xilinx ISE 11.1, ble derfor brukt og fungerer godt sammen med PlanAhead 11.1. Til tross for mangelfull funksjonalitet forventes likevel ikke valg av programvare å være av største betydning på dette stadiet i utviklingen av et dynamisk rekonfigurerbart filter.

5.2.2 Målplatform

Testkortet som er valgt for oppgaven, Suzaku-V har en FPGA av typen Virtex-4 FX XC4VF12 ombord. Syntese av av filteret ble gjort mot denne brikken(xc4vfx12-10sf363).

5.2.3 Utplassering på FPGA

Det var planlagt at filtrene skulle legges ut på FPGA i form av moduler med arealbegrensninger så små som den største konstantmultiplikator-modulene. Deretter skulle syntese vise hvor stort areal som ble forbrukt av filteret. Fordi det på et sent tidspunkt ble funnet ut, etter at filteret implementert i VHDL, at designet ikke var delt inn med statiske moduler i henhold til designreglene i **??** ble utplassering av moduler utsatt til videre arbeid. I stedet ble filteret spredt utover FPGA'en med autofunksjonen.

Dette gir kun et grovt og mangelfullt anslag av hvor stort arealet vil være, da det kun er den faktisk fobrukte logikken en får informasjon om, ikke hvor stort areal som går med til den mye diskuterete oppdeligen i moduler og statisk eller dynamisk design.

Kapittel 6

Resultater

Dette kapittelet fremstiller Simuleringsresultater og synteseresultater

6.1 Simulering

Filteret `df_im_ub` på direkte form, med individuelle multiplikatormoduler og uten bussmakroer ble simulert i ModelSim SE. Testvektorer påtrykt er tallene 0 til 8, samt tallene 253-255. Resulterenede bølgediagram foreligger i vedlegg .4.

Korrekthet av diagrammet verifisert ved manuell utregning av utvalgte deler av bølgeformen viser noe inkorrekt oppførsel. Multiplikasjonen utført av konstantmultiplikatorene gir et resultat som på binær form er skiftet én gang for mye mot venstre.

6.2 Syntese

Filtrene `df_im_um`, `df_im_bm` og `tf_im_um` beskrevet i tabell 4.1 er syntetisert for FPGA. Filtrene ble alle syntetisert mot den samme FPGA-brikken på testkortet Suzaku-V. Denne var av typen Virtex-4 FX XC4VF12 (xc4vfx12-10sf363). Videre følger de viktigste synteseresultatene. For flere detaljer, se vedlegg.

Figur 6.2 viser synteseresultater for de tre filtrene.

Figur 7 viser sammenligning mellom avansert syntese av filtrene.

I figur 4 viser de viktigste tidsberegningnene. Filteret med bussmakroer hadde maksimal frekvens på 1347.800MHz, filteret på direkte form, uten bussmakroer hadde frekvens på 972.479MHz mens det transponerte filteret hadde en maksimal frekvens 356.512MHz. Se tabell 6.1. For detaljer, se vedlegg.

Tabell 6.1: Maksimal frekvens fra syntese av filtrene

Filternavn/ maks. frekvens	Direkte form	Transponert form
Uten bussmakroer	`df_im_ub` 972.479MHz	`tf_im_ub` 356.512MHz
Med bussmakroer	`df_im_bm` 1347.800MHz	-

Tabell 6.2: Antall 4-input LUT'er med prosent av totale antall tilgjengelige på FPGA'en.

Filternavn/ ant. 4-input LUT av tot. ant.	Direkte form	Transponert form
Uten bussmakroer	`df_im_ub` 308 av 10.944(2%)	`tf_im_ub` 310 av 10.944 (2%)
Med bussmakroer	`df_im_bm` 558 av 10.944 (5%)	-

Tabell 6.3: Antall slicer med prosent av totale antall tilgjengelige på FPGA'en.

Filternavn/ ant. slices av tot. ant.	Direkte form	Transponert form	Transponert fremfor direkte
Uten bussmakroer	`df_im_ub` 186 av 5.472(2%)	`tf_im_ub` 170 av 5.427 (2%)	-9%
Med bussmakroer	`df_im_bm` 310 av 5.472 (5%)	- -	
Med, fremfor uten bussmakroer	67%	- -	

```
                                        dm_im_bm:   tf_im_ub:   df_im_um:
================================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                    : 18        : 18        : 18
 24-bit adder                           : 8         : 7         : 7
 24-bit subtractor                      : 10        : 11        : 11
# Registers                             : 32        : 96        : 32
 Flip-Flops                             : 32        : 96        : 32

================================================================================
```

Figur 6.1: Resultater fra avansert HDL syntese.

```
                                       df_im_bm:   tf_im_um: df_im_um:
 ==============================================================================
 HDL Synthesis Report

 Macro Statistics
 # Adders/Subtractors                     : 18       : 18      : 18
  24-bit adder                           : 8        : 7       : 7
  24-bit subtractor                      : 10       : 11      : 11
 # Registers                              : 32       : 4       : 32
  1-bit / 24-bit / 8-bit   register      : 32       : 4       : 32

 ==============================================================================
```

Figur 6.2: Resultater fra HDL synthese.

Kapittel 7

Diskusjon

I arbeidet mot et system for et dynamisk rekonfigurerbart filter på FPGA er det sett på systemkrav, gjort analyse av filterstrukturer, partisjonering i dynamisk og statisk design og bruk av bussmakroer. Det er implementert tre filtre som er syntetisert, hvorav ett er simulert. Resultatene gir forståelse for videre design av et dynamisk rekonfigurerbart system.

7.1 Syntese- og simuleringsresultater

Filteret `df_im_ub`, på direkte form, med individuelle multiplikatormoduler, uten bussmakroer ble simulert. Utfyllende tester av filtert er ikke gjennomført da det i første omgang er ønskelig å teste funskjonliteten til filtrene. Om filteret skal brukes videre, krever dette mer omfattende testing av hjørnetilstander og lignennde. Filteret er kun testet for at det fungerer for én påtrykt serie med inndata. De første resulatene viste seg å være feil, og videre simulring ble derfor utsatt til etter retting av feilen.

Bølgeskjema ble studert og kontrollert manuelt for korrekthet. At produktene fra konstantmultiplikatorene er skiftet én plass til venstre betyr at alle verdiene er ytterligere multiplisert med 2_{10} i forhold til ønsket resultat. Som nevnt tidligere er ikke filteret påtenkt noen spesiell bruk og det er derfor ikke avgjørende med korrekt respons. I praksis betyr de uriktige resultatene bare at multiplikatorene er andre koeffisienter enn spesifisert, men feilen betyr mulighet for andre feil. Antallet adderere og subtraherere stemmer derfor ikke overens med hva some er antatt. En

ny tabell slik som 4.1 laget uifra de reelle, syntetiserte koeffisientene vil gi et antall adderere og subtraherere som er én mindre per konstant-multiplikator. Syntese av filtrene uten bussmakroer gir advarsler om at utgangen `ck:o` på koeffisientene er drevet av det konstante signalet `coeff<k><0>`, der `k` fra 0 til 3 er koeffisientnummeret. Det ser derfor ut til at det blir skiftet for langt, slik at LSB på utgangen til koeffisient-modulene alltid vil være `'0'`. Det antas at dette kan ha innvirkning på den høye maksimale frekvensen i tabell 6.1.

7.1.1 Synteseresultater

Forskjellen i tabell 6.3 mellom antall slicer for de syntetiserte filtrene viser at filteret øker med 67% i antall slicer ved innføring av buss-makroer. Dette viser at innføring av bussmakroer utgjør en betydelig kostnad for filteret. Ved overgang fra direkte form til transponert form er forskjellen -9% hvilket betyr at det er noe areal å spare. Den store forskjellen ved innføringen av bussmakroene viser at det er av betydning å bruke færrest mulig av disse.

Løsninger for å få ned antall bussmakroer er enten ved å bruke et transponert filter med en sammenslått multiplikatormodul, trunkering av signalet, eller bruke en form for seriell overføring mellom modulene.

Det totale forventet antall adderer og subtraherere ble justert ned med én per koeffisient ettersom den genererte koden fra `Filtergenerator` var uriktig og konsekvent skiftet én gang for langt.

Filtrenes maksimale frekvenser viser. Filteret `df_im_bm` har en usedvanlig høy maksimal frekvens på 1347.800MHz. Det er derfor grunn til å tro at det foreligger feil i designet. Filteret `tf_im_ub` på transponert form har en mer realistisk maksimal frekvens på 356.512MHz. Med så høye mulige frekvenser vil filtrene klare seg bra sammen med en myk prosessor.

7.1.2 Implementering på FPGA

Filtrenes inn- og utganger er plasert på I/O-pinnene på FPGA-brikken for å få synteseresultater. For test av filterene burde disse har vært koblet til en myk prosessor med et kjørende testprogram.

PR-moduler

Modulene er implementert uten arealbegrensninger og automatisk plassert på et stort område. Ved å plassere multiplikatormodulene omhyggelig med nødvendige bussmakroer inntil, ville det vært mulig å få et mål på hvor mange slice'er som ville vært nødvendig å okkupere for filteret. På denne måten ville det også vært enklere å se hvor stor forskjellene er mellom store og små konstantmultiplikatormoduler, for dermed å vite potensialet for gjenbruk av disse. Det er uvisst hvor mye det lønner seg med flisespikking dersom en hel ramme uansett må rekonfigureres.

Multplikatormoduler

Figur 3 viser store carrychains som gjør at modulene har stort areal, men mye dødplass innenfor arealbegrensningenen. Flere lange, kjeder med addisjoner strekker seg vertikalt på FPGA'en. Denne figuren er RTL-kretsskjema av hvordan filteret konfigureres i logikken på FPGA.

Nødvendige forandringer

Filteret legges i en toppmodul i VHDL-beskrivelsen, og det må lages moduler for det statiske designet. Dette betyr enkle forandringer i koden, men er nødvending for bruk av filtrene i videre arbeid. Det kan da testes filtre som har statiske moduler med arealbegrensninger slik at det ville vært mulig å se konsekvensen av å måtte klemme konstantmultiplikatormodulene mellom to statiske moduler i forhold til å kun ha dem ved siden av hverandre.

7.2 Designvalg

Det er et viktig designmål at filteret ikke skal ha for lang rekonfigureringstid. Det har derfor i denne oppgaven blitt fokusert på hvordan filteret kan inndeles i statisk og dynamisk design slik at minst mulig logikk må konfigureres på nytt ved endringer i filterspesifikkasjonen.

Fordi rekonfigurering er kostnad i form av tid, gjelder det å begrense hvor stort areal som man behøver å rekonfigurere. Det er derfor lagt stor vekt på å dele opp filteret det er minst mulig logikk som krever rekonfigurering.

I filteret er det flere komponenter som vil være der uavhengig av hvilke koeffisienter som er i filtere ellers. Det er derfor ønskelig å bevare dette ved rekonfigurering, for i stedet å rekonfigurere nøyaktig det som er annerledes ved nye koeffisientverdier.

Transponert form fremfor direkte form

Fordi hele det statiske designet også skal innkapsles i moduler med arealbegrensninger og plasseres ut på FPGA vil det antageligvis lønne seg å implementere både registere og aradderere i én modul.

Det transponerte filteret muliggjør dette slik kan det oppnås høyere utnyttelse av arealet i den rektangulære modulen. Antageligvis vil dette gi et godt resultat til tross for at transponering krever større adderere for akkumulering.

Ved å benytte transponert form av FIR-filteret, oppnås en større frihet i forhold til plassering av koeffisientmodulene slik at de kan ekspandere vekk fra resten av filteret.

Multiplikatormoduler

Stor modul ser ut til å ha de beste forutsetninger for å få det minste arealet. Dette gjelder både på grunn av mulighetene for optimalisering, og på grunn av behovet for færre bussmakroer ved kombinasjon med transponert filter.

Det er ønskelig å vite noe om hvordan et filter blir med én modul for multiplikatorer i forhold til én modul for hver multiplikator. For finne ut om dette ville det vært hensiktsmessig å teste med mange forskjellige koeffisienter. De implementerete filtrene er likevel av 3. grad da dette gir betydelig enklere implementering- og testfase enn ved høyere grad. Dette er fordi en dermed ikke trenger lage like mange PR-moduler og arealbegrensninger manuelt.

Forbedring av Filtergenerator

Målet med programmet som genererer VHDL-kode har vært vise hvordan koden kan genereres, men også å forenkle utviklingsfasen ved raskt å kunne generere filtre og konstantmultiplikatorer. Programmet vil måtte skrives om for at det skal kunne automatisk generes filtermoduler som

kan brukes i filteret. Likevel kan programmet brukes til å generere koeffisienter til bruk i den nye strukturen, da dette kun krever litt klipping og liming av koden som genereres av programmet.

Utskrift av koeffisientverdiene tilsier at CSD-verdiene er korrekte, slik at feilen nevnt i forbindelse med simulering stammer fra VHDL-genereringen. Programmet bør i første omgang korrigeres for den simple feilen med for mange skift. Før bruk i et fullstendig system må programmet omskrives slik at det lager komponenter for multiplikatorene, og deretter kontrolleres ytterligere for korrekthet. Det vil være fornuftig å vente med dette til det endelige designet filterene er fastslått.

7.3 Generelt

Antagelser gjort under design av filtrene og oppdeling i moduler er at det er en god sammenheng mellom størrelse på arealet av PR-modulen og rekonfigureringstid.

Filtrene som er implementert i dette arbeidet er forsøkt tilrettelagt slik at det blir enklest mulig å bytte til riktig programvare og dermed ta i bruk delvis rekonfigurering.

Filteret, `df_im_bm` på direkte form, med bussmakroer kan ikke brukes til delvis rekonfigurering før det statiske designet er skrevet om i VHDL-koden til moduler. Sammen med PR-modulene må modulene innkapsles i en toppmodul. Det vil da være mulig å videre utforske dynamiske størrelser på PR-modulene.

Arbeidet med realisering av filtersystemet er svært avhengig av praktiske detaljer rundt hva som er realiserbart med hensyn på FPGA-arkitektur og funksjonalitet i designverktøy. Videre arbeid er eksperimentering med plasseringer av moduler. Det må også avgjøres hvordan selve rekonfigureringen med de delvise bitfilene skal foregå i sammenheng med systemet.

Design av filteret ikke har vært mot et spesifisert bruk, men er gjort mot mange potensielle typer bruk av filter.

Kapittel 8

Konklusjon og videre arbeid

Det er gjort designarbeid mot et system med et dynamisk rekonfigurerbart FIR-filter på FPGA. Filteret benytter genererte konstantmultiplikatorer som er optimalisert med CSD-kode for lite areal. Muliplikatorene er tilrettelagt for å kunne endres dynamisk i kjøretid og dermed skape et hardware-optimalisert, og likevel fleksibelt filter. Multiplikatorene er generert i VHDL fra programmet fra prosjektoppgaven. Det har vært sett på hvordan transponert form av filteret kan gi fordeler fremfor bruk på direkte form. Tre filtre med individuelle moduler for konstantmultiplikatorene er implemenert og syntetisert mot Virtex-4 til for å kunne benyttes med testkortet Suzaku-V. Syntese av filtrene viser at konstantmultiplikatorene er så små at innføring av bussmakroer utgjør en økning på 67% av antallet slicer av et 3. ordens FIR-filter med 16-bit koeffisienter. Filteret bruker da 558 av 10.944 slicer som utgjør 5%. Filteret på transponert form har en maksimal frekvens 356.512MHz. Transponert form gir bedre muligheter for dynamisk ekspansjon av konstantmultiplikatorene i kjøretid enn direkte form. Ved innkapsling av det statiske designet i moduler kan filteret på direkte form med bussmakroer skaleres etter spesifikasjon og brukes i dynamisk rekonfigurering.

Videre arbeid

Videre arbeid består i å implementere det transponerte filteret med bussmakroer. Den statiske logikken må da være i en egen modul. Det er også mulig å jobbe videre med filteret på direkte form som allerede er implentert med bussmakroer. Dette innebærer også å innkapsle det statiske designet i moduler. Ved tilgang på Partial Reconfiguration Tool fra Xilinx, Inc. kan filteret utplasseres på FPGA sammen med MicroBlaze eller PowerPC. Det må også skrives et test-program for å teste filteret på FPGA. Det må også velges en metode for hvordan selve rekonfigureringen skal skje.

Bibliografi

[1] Inc. Andraka Consulting Group. Multiplications in fpgas. web, 2009.

[2] J. Becker, M. Hubner, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka. Dynamic and partial fpga exploitation. *Proceedings of the IEEE*, 95(2):438–452, Feb. 2007.

[3] Vebjørn Bystrøm. Low power/high performance dynamic reconfigurable filter-design. Master's thesis, NTNU, 2008.

[4] Ingar Hauge. Analyse, dekomponering og rekonstruksjon av fpgakonfigurasjoner for ahead, 2006.

[5] R.M. Hewlitt and Jr. Swartzlantler, E.S. Canonical signed digit representation for fir digital filters. *Signal Processing Systems, 2000. SiPS 2000. 2000 IEEE Workshop on*, pages 416–426, 2000.

[6] Xilinx Inc. Early access partial reconfiguration for ise 9.2.04i. 2008.

[7] Dimitris K Manolakis John G. Proakis. *Digital Signal Processing, 4th Edition*. Prentice Hall, 2007.

[8] Cindy Kao. Benefits of partial reconfigurations. *Xcell Journal*, 22(5):65–67, 2005.

[9] M. Martinez-Peiro, E.I. Boemo, and L. Wanhammar. Design of high-speed multiplierless filters using a nonrecursive signed common subexpression algorithm. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 49(3):196–203, Mar 2002.

[10] R.W. Mehler and Dian Zhou. Automated architectural optimization of digital fir filters. In *VLSI Design, 2004. Proceedings. 17th International Conference on*, pages 177–182, 2004.

[11] S. Mirzaei, A. Hosangadi, and R. Kastner. Fpga implementation of high speed fir filters using add and shift method. In *Computer Design, 2006. ICCD 2006. International Conference on*, pages 308–313, Oct. 2006.

[12] T. Rissa, R. Uusikartano, and J. Niittylahti. Adaptive fir filter architectures for run-time reconfigurable fpgas. pages 52–59, Dec. 2002.

[13] Reetinder Sidhu and Viktor K.Prasanna. Efficient metacomputation using self-reconfiguration. *Field-Programmable Logic and Applications:Reconfigurable Computing is Going Mainstream*, 2002.

[14] Lars Wanhammar. *DSP Integrated Circuits*. Academic Press, 1999.

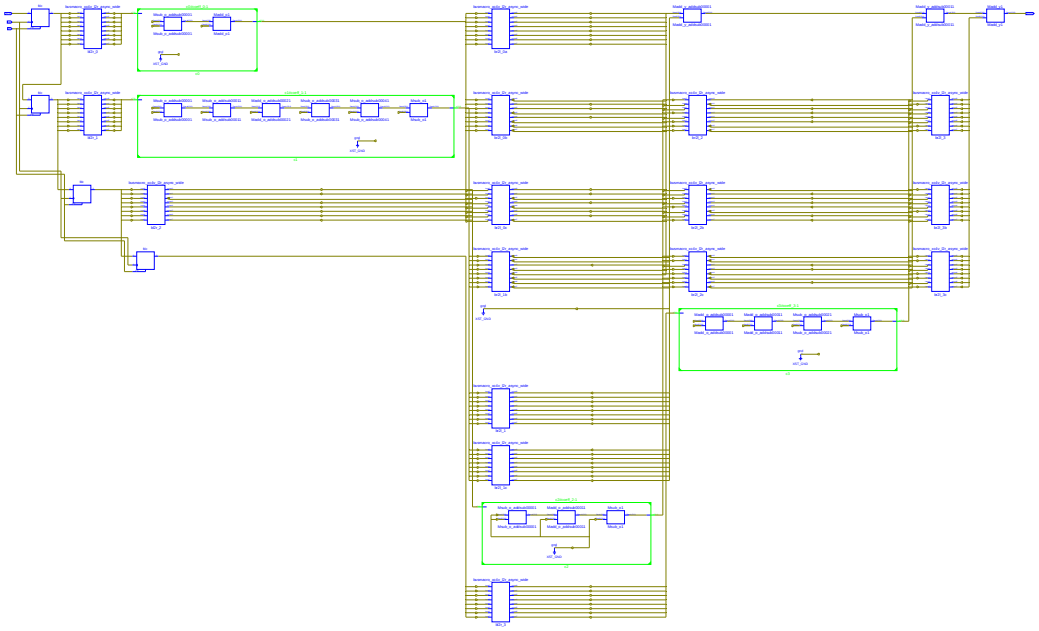
[15] Wikipedia. Finite impulse response. *Wikipedia*, 2009.

[16] Xilinx. Two flows for partial reconfiguration: Module based or small bit manipulations. 2002.

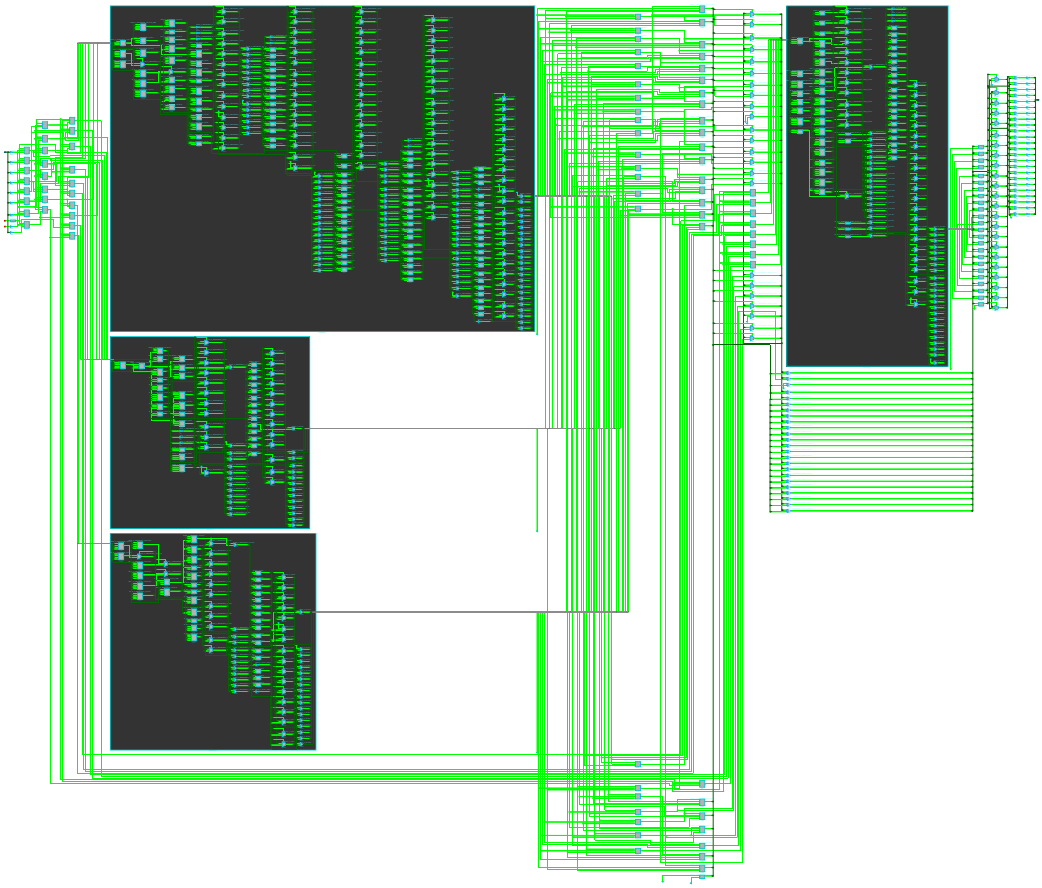
[17] Xilinx. Virtex-4 family overview ds112 (v3.0). *Product Specification*, 2007.

[18] Hanho Lee Yeong-Jae Oh and Chong-Ho Lee. Dynamic partial reconfigurable fir filter design. *Reconfigurable Computing: Architectures and Applications*, 3985:30–35, 2006.

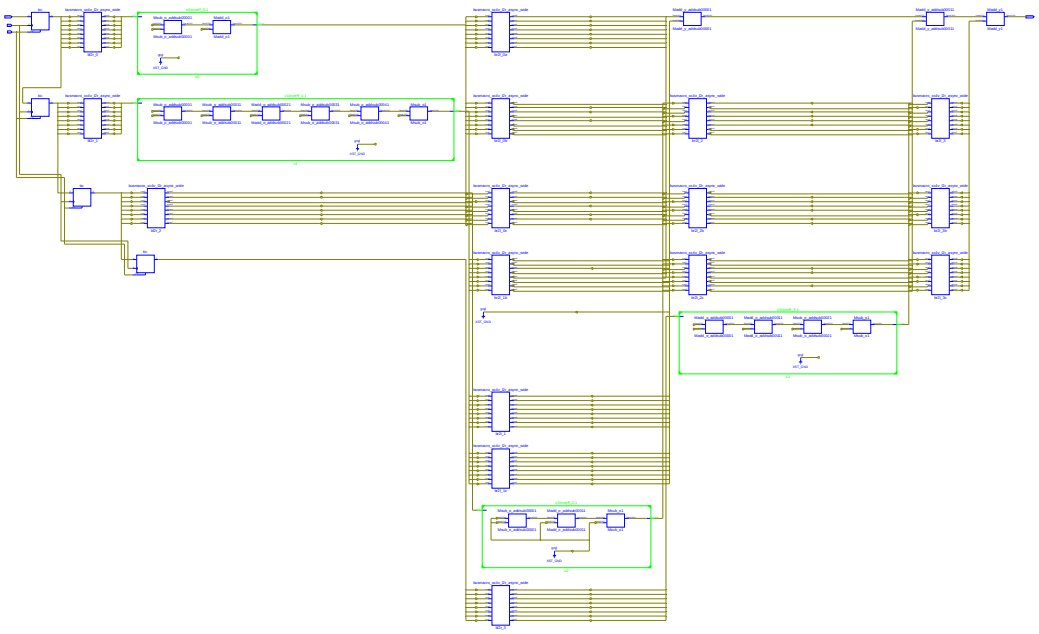
RTL-kretsskjema for implementerte filtre



Figur 1: RTL-skjema for filteret på direkte form med bussmakroer og ekspanderte multiplikatormoduler.



Figur 2: RTL-teknologibeskrivelse av filtertet på direkte form, med ekspanderte multiplikatormoduler.



Figur 3: Direkte form, bussmakroer med ekspanderte moduler, fra PlanAhead.

Synteseresultater

.1 Timing

```
Timing Summary, df_im_bm:
---------------
Speed Grade: -10

   Minimum period: 0.742ns (Maximum Frequency: 1347.800MHz)
   Minimum input arrival time before clock: 1.347ns
   Maximum output required time after clock: 0.720ns
   Maximum combinational path delay: 11.240ns
-------------------------------------------------------------


Timing Summary, df_im_um:
---------------
Speed Grade: -10

   Minimum period: 1.028ns (Maximum Frequency: 972.479MHz)
   Minimum input arrival time before clock: 1.347ns
   Maximum output required time after clock: 17.547ns
   Maximum combinational path delay: No path found
-------------------------------------------------------------


Timing Summary, tf_im_um:
---------------
Speed Grade: -10

   Minimum period: 2.805ns (Maximum Frequency: 356.512MHz)
   Minimum input arrival time before clock: 12.811ns
   Maximum output required time after clock: 4.677ns
   Maximum combinational path delay: No path found
-------------------------------------------------------------
```

Figur 4: Tidsberegninger fra syntese av filtrene.

.2 Direkte form, med bussmakoer

device_utilization_summary.txt

```
df_im_bm:
Device utilization summary:
---------------------------

Selected Device : 4vfx12sf363-10

 Number of Slices:                      166  out of   5472     3%
 Number of Slice Flip Flops:             32  out of  10944     0%
 Number of 4 input LUTs:                300  out of  10944     2%
 Number of IOs:                          34
 Number of bonded IOBs:                  34  out of    240    14%
 Number of GCLKs:                         1  out of     32     3%

---------------------------
Partition Resource Summary:
---------------------------

Partition "/my_fir":
  Number of Slice Flip Flops:          32
  Number of Slice LUTs:                70
  Number of BUFG:                       1
  Number of bonded IOBs:               34

Partition "/my_fir/c0":
  Number of Slice LUTs:                23

Partition "/my_fir/c1":
  Number of Slice LUTs:               111

Partition "/my_fir/c2":
  Number of Slice LUTs:                39

Partition "/my_fir/c3":
  Number of Slice LUTs:                57

---------------------------


================================================================================
```

Side 1

Figur 5: Tidsanalyse av kretsen for filteret på direkte form, med bussmakroer.

.3 Transponert form, uten bussmakroer

```
                        device_utilization_summary.txt

tf_im_um:
Device utilization summary:
---------------------------

Selected Device : 4vfx12sf363-10

 Number of Slices:                      176  out of   5472     3%
 Number of Slice Flip Flops:             96  out of  10944     0%
 Number of 4 input LUTs:                309  out of  10944     2%
 Number of IOs:                          34
 Number of bonded IOBs:                  34  out of    240    14%
 Number of GCLKs:                         1  out of     32     3%

---------------------------
Partition Resource Summary:
---------------------------

Partition "/my_fir":
  Number of Slice Flip Flops:          96
  Number of Slice LUTs:                73
  Number of BUFG:                       1
  Number of bonded IOBs:               34

Partition "/my_fir/c0":
  Number of Slice LUTs:                29
  Number of bonded IOBs:                6

Partition "/my_fir/c1":
  Number of Slice LUTs:               111

Partition "/my_fir/c2":
  Number of Slice LUTs:                39

Partition "/my_fir/c3":
  Number of Slice LUTs:                57

---------------------------


=====================================================================

                                Side 1
```

Figur 6: Tidsanalyse av kretsen for filteret på transponert form, uten bussmakroer.

.4 Direkte form, uten bussmakroer

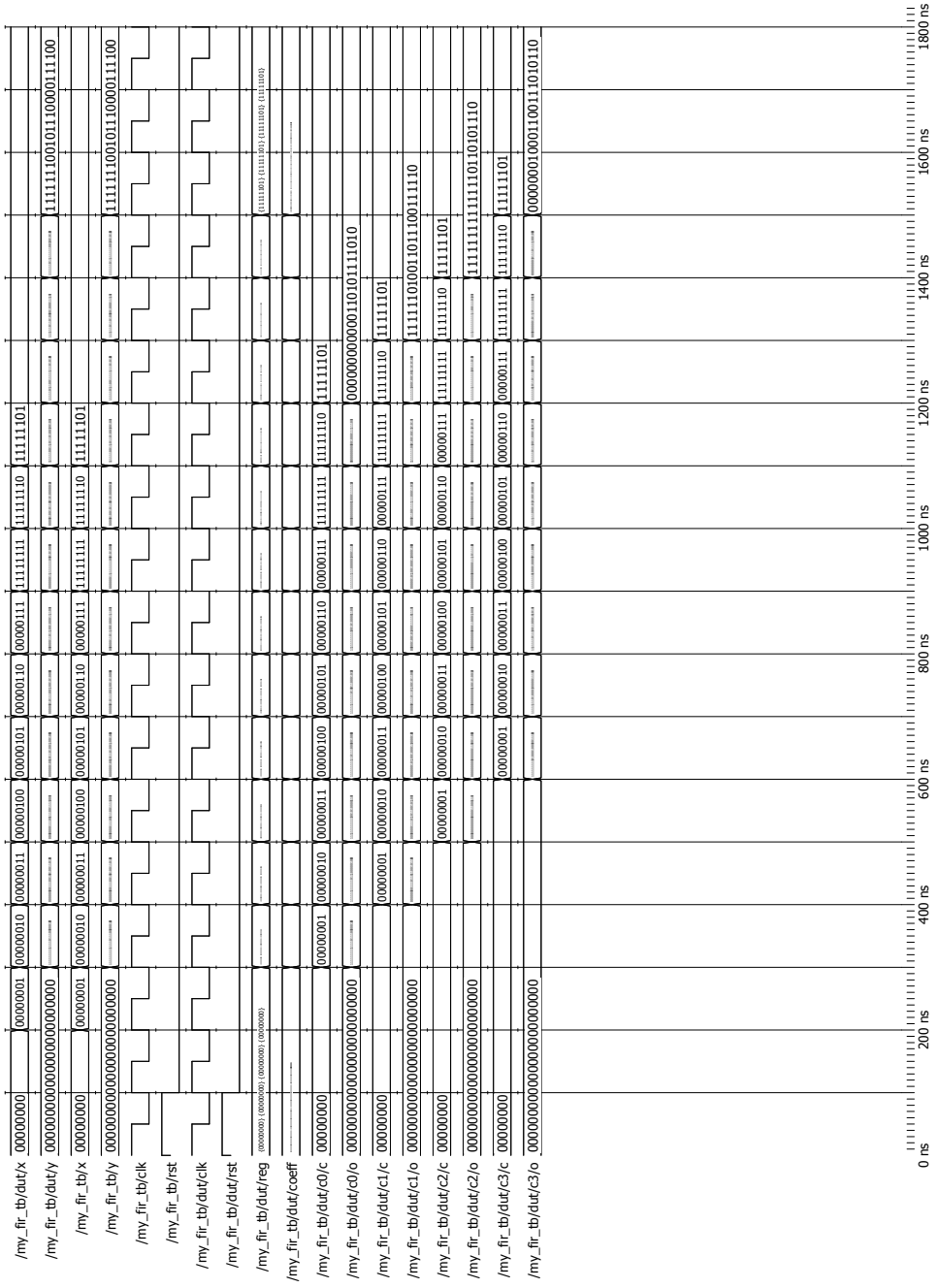
device_utilization_summary.txt

```
df_im_um:
Device utilization summary:
---------------------------

Selected Device : 4vfx12sf363-10

 Number of Slices:                      170  out of   5472      3%
 Number of Slice Flip Flops:             34  out of  10944      0%
 Number of 4 input LUTs:                306  out of  10944      2%
 Number of IOs:                          34
 Number of bonded IOBs:                  34  out of    240     14%
 Number of GCLKs:                         1  out of     32      3%

---------------------------
Partition Resource Summary:
---------------------------

Partition "/my_fir":
  Number of Slice Flip Flops:          34
  Number of Slice LUTs:                70
  Number of BUFG:                       1
  Number of bonded IOBs:               34

Partition "/my_fir/c0":
  Number of Slice LUTs:                29

Partition "/my_fir/c1":
  Number of Slice LUTs:               111

Partition "/my_fir/c2":
  Number of Slice LUTs:                39

Partition "/my_fir/c3":
  Number of Slice LUTs:                57

---------------------------


================================================================================
```

Side 1

Figur 7: Tidsanalyse av kretsen for filteret på direkte form, uten bussmakroer.

Bølgediagram for direkte form FIR-filter, uten bussmakroer



Figur 8: Bølgediagram for direkte form FIR-filter, uten bussmakroer