Funksjonsbuffer i maskinvare

Tord A. Fredriksen

31. august 2005

Sammendrag

Rekonfigurerbar maskinvare har i den siste tiden vist seg å være et særdeles nyttig verktøy i forbindelse med akselerasjon av forskjellige algoritmer. Spesielt gjelder dette algoritmer som enkelt lar seg parallellisere.

Et av hovedproblemene med bruk av rekonfigurerbar maskinvare er at det tar relativt lang tid å rekonfigurere selve maskinvaren for en gitt algoritmeimplementasjon (modul). Hvis en bruker flere forskjellige slike implementasjoner etter hverandre kan dette bli et veldig stort problem.

Partiell rekonfigurering har lettet litt på dette problemet. Dette fordi partiell rekonfigurering gjør det enkelt å ha flere moduler lastet inn i maskinvaren samtidig. I tillegg er det mulig å "skjule" rekonfigureringstiden ved å rekonfigurere deler av maskinvaren mens resten jobber med en oppgave. Flere moduler på samme brikke betyr også at det er større sjanse for at en ikke trenger å rekonfigurerer maskinvaren før en gitt oppgave, siden det er en sjanse for at modulen allerede ligger konfigurert.

I denne oppgaven presenterer vi et system som har mulighet til å ha 4 forskjellige moduler konfigurert til enhver tid. Denne oppgaven ser på muligheten til å benytte temporær lokalitet i bruksmønstret til de forskjellige modulene, til å avgjøre hvor i maskinvaren en ny modul skal lagres. Med dette mener vi å implementere et system som holder rede på hvor ofte de forskjellige modulene blir brukt. Denne informasjonen blir så benyttet til å bestemme hvilke moduler som skal få ligge i maskinvaren og hvilke som skal kastes ut når det blir nødvendig å laste inn en ny modul. Til dette benyttes en enkel LRU- (least recently used) algoritme, og systemets virkemåte har mange likhetstrekk med hvordan hurtig-buffer for minne fungerer på en vanlig PC. Med dette håper en å redusere antall rekonfigureringer og på denne måten redusere den totale rekonfigurasjonsforsinkelsen.

Vi ser også på muligheten til å stokke om på rekkefølgen oppgavene blir sendt inn i, for å prøve å minimere antall rekonfigureringer.

Systemet består av en programvaredel som holder rede på bruksmønsteret og tar avgjørelse om fordelingen av modulene. I tillegg så inneholder systemet også en maskinvaredel som tar seg av kommunikasjon mellom modulene og programvaren. Modulene ligger på egne datafiler som blir lest inn ved behov av brukerprogrammet.

Brukeren sender inn oppgaver til systemet ved å spesifisere hvilken modul han/hun ønsker å benytte og hvilke data som skal sendes inn til den.

Innhold

Figurer

Tabeller

Kapittel 1

Intro

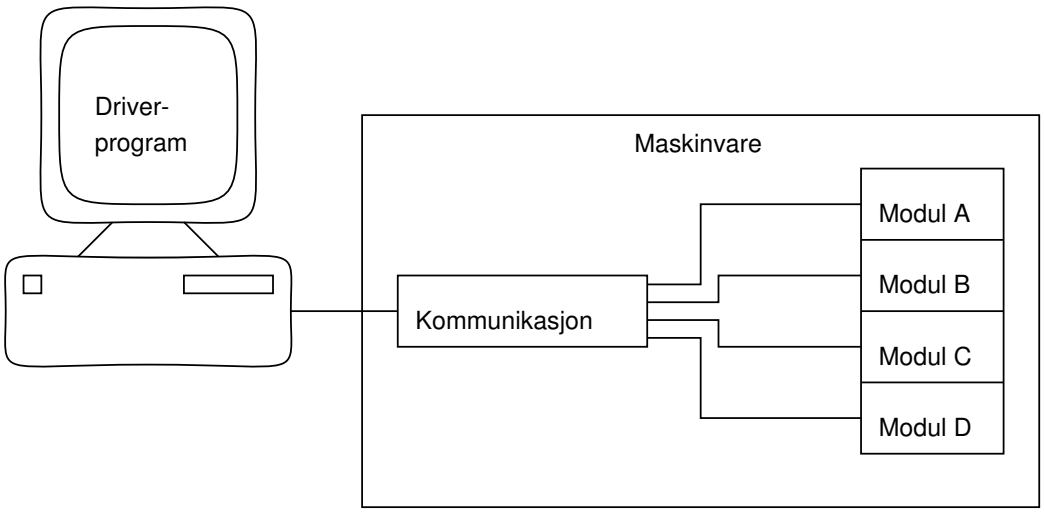
I denne første delen av oppgaven presenterer vi en introduksjon av oppgaven hvor vi beskriver enkelt hvorfor rekonfigurerbar maskinvare kan forbedre ytelsen av egnede systemet. Vi presenterer også en enkel oversikt over hva vi har laget og oversikt over hva resten av denne oppgaven består av. Vi har også med en presentasjon av målene vi ønsket å oppnå i forbindelse med denne oppgaven.

1.1 Introduksjon

I tradisjonelle programvare-løsninger som kjører på en sekvensiell generisk prosesserings-enhet, så kan det være særdeles vanskelig å utnytte den parallelliteten som finnes i endel algoritmer. Det er derfor veldig lett å ende opp med dårligere ytelse for disse algoritmene enn hvis de skulle vært kjørt på maskinvare spesielt designet på å kjøre akkurat disse algoritmene. Men slike sekvensielle prosesserings-enheter har den fordelen at selv om de ikke tilbyr den beste ytelsen så er de fleksible nok til å kunne utføre praktisk talt enhver oppgave.

Problemet med å implementere alt i maskinvare er at en ikke alltid vet ved design-tid av systemet, hvilke algoritmer en vil ha behov for å kjøre. I tillegg så vil selv den minste forandring i designets funksjonalitet gjøre det nødvendig med en ny runde av en kostbar produksjonsprosess.

I det siste har rekonfigurerbar maskinvare gjort det mulig å implementere forskjellige design på en enkel brikke uten å måtte forandre selve brikken. Dette har blant annet medført at det er mulig å drive med rask prototyping av et design før en setter igang med selve produksjonsprosessen. i tillegg til at det har åpnet for å kunne "oppgradere" maskinvaren ved å ganske enkelt bare laste over en ny konfigurasjon.



Figur 1.1: Forenklet oversikt over hva systemet skal bestå av.

Rekonfigurerbar maskinvare har også gjort det mulig å utnytte både paralleliserbarheten i maskinvare og fleksibiliteten fra prosesserings-enheter i et og samme system. Ved å koble sammen en prosesserings-enhet og rekonfigurerbar maskinvare så har en mulighet til å implementere flere av systemets kjernefunksjoner i maskinvare. Dette kan gjøres på en slik måte at en ikke nødvendigvis trenger å vite hvilke funksjoner som skal implementeres i maskinvaren når en designer systemet. Forskjellige maskinvare-implementasjoner kan da lastes inn i den rekonfigurerbare maskinvaren ved behov.

Et av hovedproblemene med bruk av denne type teknologi er at tiden det tar å rekonfigurere maskinvaren for en gitt algoritme veldig lett kan overstige tiden en kan spare ved å kjøre algoritmen i maskinvare.

Det er derfor blitt designet flere teknikker, som forsøker å redusere denne forsinkelsen. En vanlig metode her er å prøve å redusere antall rekonfigureringer som er nødvendig å foreta.

Dette kan oppnås på flere måter. En vanlig metode er å kjøre oppgaver som benytter den samme konfigurasjonen etter hverandre. På denne måten så trenger en ikke å rekonfigurere maskinvaren før hver oppgave.

I den siste tiden så har partiell rekonfigurering vis seg å være et nyttig verktøy i denne forbindelsen. Dette kommer spesielt av to grunner. For det første så betyr det at det ikke lengre er nødvendig å konfigurere hele maskinvaren hvis det bare en en liten del av den som blir brukt av det nye designet. Dette betyr at tiden som går med for å konfigurere inn et design kan gå drastisk ned siden data for ubenyttet areal ikke er nødvendig å overføre. En annen ting som partiell rekonfigurering har gjort mulig er å ha flere moduler

konfigurert i maskinvare samtidig, og å kunne bytte enkelte av dem ut med nye moduler uten å forstyrre andre moduler. Dette betyr at det er mulig å la flere moduler bli liggende over lengre tid selv om andre moduler blir benyttet i mellomtiden. Dette kan reduserer antall rekonfigureringer betraktelig. Det betyr også at en kan skjule rekonfigureringsforsinkelsen med å kjøre rekonfigurering av en modul parallelt med at andre oppgaver kjører.

Denne oppgaven tar sikte på å designe et system med akkurat denne funksjonaliteten. Systemet skal være istand til å ta imot en strøm av oppgaver og prosessere disse i maskinvaren etterhvert som de kommer inn. En oppgave i dette systemet er en kombinasjon av en algoritme-implementasjon (rekonfigurerbar maskinvare-implementasjon) og et sett med data som denne algoritmen skal jobbe på.

Flere lignende systemet er blitt laget tidligere (se kapittel 2.3). Disse har ofte basert seg på å manipulere rekkefølgen oppgavene blir sendt inn i systemet på. Systemet som beskrives i denne oppgaven skiller seg fra disse ved at det i utgangspunktet ikke skal manipulerer rekkefølgen av oppgaver. Systemet skal heller se på rekkefølgen oppgavene blir sendt inn i. På grunnlag av dette forsøker det å finne ut hvilke moduler som skal få ligge igjen i systemet og hvilke som skal kastes ut når dette blir nødvendig grunnet plassmangel.

Måten dette gjøres på er at den holder oversikt over hvor lenge siden det er at en gitt modul ble brukt, og velger dermed å kaste ut de modulene som det er lengst siden ble brukt, altså en LRU-algoritme (Least Recently Used).

Grunnen til at dette kan være interessant og se på er at det er ikke alltid at systemet kjenner til hvilke oppgaver som skal kjøres igjennom det. Dette kan for eksempel komme av at brukeren ikke sender inn alle oppgavene på en gang grunnet avhengigheter mellom de forskjellige oppgavene som forhindrer brukeren å sende inn enkelte oppgaver han/hun har fått resultatene fra tidligere oppgaver. Men gitt at brukeren sender inn flere oppgaver om gangen, så antas det at oppgavene ikke er avhengige av hverandre. Derfor tester vi også ut effekten av å stokke om på rekkefølgen til de innsendte oppgave, for å forsøke å redusere antall rekonfigureringer.

Som det vises i figur 1.1 så skal systemet bestå av 2 deler.

Den første av dem er programvare-driveren. Denne har som oppgave og holde rede på hvilke maskinvare-moduler som til enhver tid er konfigurert, og holde rede på statistikk over bruksmønster. På grunnlag av dette skal den avgjøre hvor eventuelle nye maskinvare-moduler skal plasseres.

Den andre delen av systemet er kommunikasjons-modulen som skal ligge i maskinvare. Denne skal ha som oppgave å ta seg av kommunikasjonen mellom programvare-driveren og maskinvare-modulene.

I tillegg til kommunikasjons-modulen vil maskinvare-designet også bestå av 4 såkalte modulslotter. Disse skal inneholde de rekonfigurerbare modulene.

Denne oppgaven er delt inn i 4 kapitler. Den første av disse tar for seg en introduksjon av oppgaven og hva målet med den er. Kapittel nummer 2 ser på bakgrunns-teorien som ligger bak oppgaven. Dette kapittelet ser også på hva som er blitt gjort tidligere, og hvilke plattformer som var tilgjengelig for oppgaven. Kapittel 3 ser på selve systemene som er blitt laget og diskusjon rundt resultatene av test-kjøringene med dem. I kapittel 4 presenterer vi en oppsummering av oppgaven med konklusjon og forslag til videre arbeid. Tilslutt kommer noen tilleggs-kapitler som inneholder diverse tilleggs-informasjon om hvilke systemer vi benyttet under designet av systemet vårt, og kildekoden til det ferdige systemet.

1.2 Mål for oppgaven

> **Oppgavetekst** Oppgaven går ut på å utrede muligheten av å lage et tilleggs-kort til en vanlig PC som har som oppgave å utføre beregnings-tunge funksjoner i egen maskinvare. Maskinvaren på tilleggs-kortet vil typisk være basert på FPGA*-teknologi, og ressursen skal brukes dynamisk slik at de hyppigst brukte blant de aktuelle funksjoner lagres der, mens mindre hyppig brukte funksjoner hives ut. Prinsippet vil tilsvare det man har i hurtig-buffer for data og instruksjoner, men hovedforskjellen er at det her vil være rene maskinvare-implementasjoner av funksjoner som skyfles inn og ut. Foruten å utrede konseptet, skal minst en mulig løsning simuleres, og dersom det blir tid også prøves i virkelighet på et utviklings-kort med FPGA teknologi.

Målet for oppgaven var som nevnt i oppgaveteksten å lage et system for rekonfigurerbar maskinvare basert på moduler. Systemet skal bestå av et maskinvare-design med utskiftbare moduler, og et programvare-system (driver) som styrer dette. Maskinvare-systemet skal inneholde et sett av modulslotter slik at det er mulig å ha flere moduler inne til samme tid. Når det ønskes å benytte en modul som ikke er konfigurert i maskinvaren og det ikke er noen ledige slotter vil programvare-driveren på bakgrunn av rekkefølgen de forskjellige utskiftbare modulene har vært benyttet, sammen med et sett regler, avgjøre hvilken modulslot som skal rekonfigureres. Det skal også lages et grensesnitt som brukeren kan benytte for å sende oppgaver til systemet. Disse oppgavene vil bestå av en gitt modul som skal benyttes, samt oppgave-data som denne modulen skal behandle. Systemet skal da plassere den i en kø og når maskinvare systemet har ledig kapasitet så skal programvare-driveren avgjøre hvilke modulslot som skal benyttes og hvis nødvendig, rekonfigurere denne før den overfører oppgavedataene. Programvare-driveren vil kontinuerlig overvåke statusen til maskinvaren, og vil kunne oppdage når en oppgave er ferdig behandlet. Den vil da automatisk hente resultatdataene tilbake, og gi beskjed til brukeren ved å markere oppgaven som ferdig.

*Field Programmable Gate Array

Kapittel 2

Bakgrunn

I denne delen av oppgaven så ser vi på teknologien som ligger som grunnlag for denne oppgaven. Vi ser på hva rekonfigurerbar maskinvare er for noe, og hva som menes med partiell rekonfigurering. Vi ser også på enkelte tidligere systemer som har hatt som mål å forbedre ytelsen ved hjelp av rekonfigurerbar maskinvare. Til slutt ser vi på hva vi hadde tilgjengelig av rekonfigurerbar maskinvare som vi kunne bygge systemet vårt på.

2.1 Bakgrunnsteori

En generisk prosessor-enhet i et datasystem har de egenskapene at den er veldig fleksibel, men at alt skjer sekvensielt Det vil si at den ikke er istand til å gjøre mer en en ting om gangen. Et eksempel her kan være at en skal legge sammen to vektorer($\vec{v}$ og $\vec{w}$) hver på n elementer. I en sekvensiell prosessorenhet vil en typisk måtte starte i en av endene av vektorene og gradvis jobbe seg igjennom vektorene gjennom n steg. Denne algoritmen tilhører da kompleksitetsklassen $\mathcal{O}(n)$.

En mer effektiv algoritme ville være en parallell tilnærming, hvor en beregner summen av $\vec{v}_i$ og $\vec{w}_i$ $\forall$i ε [0..n-1] samtidig, som tilhører kompleksitetsklassen $\mathcal{O}(1)$. Dette parallelliserings-steget er vanskelig å oppnå for en sekvensiell prosessor uten spesifikk støtte, men er relativt enkelt å realisere ved hjelp av en ASICs*-implementasjon. Et problem med ASICs er som navnet tilsier, at den er designet for å utføre et spesifikt sett med oppgaver, og er derfor svært lite fleksibel med tanke på gjenbruk i andre typer arbeidsoppgaver.

En tredje mulighet er å bruke rekonfigurerbar maskinvare. Det vil si maskinvare som er designet på en slik måte at de i utgangspunktet ikke har noen

*ASICs - Application Specific Integrated Circuits

spesifikk funksjonalitet, men kan etter behov konfigureres til å utføre oppgaver på samme måte som f.eks. en ASIC-brikke ville gjort. Ved å benytte slik maskinvare kan en for egnede algoritmer oppnå en ytelse betraktelig bedre enn for sekvensielle prosessor enheter (men ikke like raskt som for ASICs brikker). Samtidig beholder en mesteparten av fleksibilitet fra sekvensielle prosessor enheter. En annen fordel med å bruke rekonfigurerbar maskinvare er at siden en kan bruke de samme brikkene til forskjellige applikasjoner er de mye billigere å produsere siden de blir produsert i store antall, og egner seg derfor godt til prototyping av ny maskinvare.

Dette del-kapittelet ser på forskjellige typer av rekonfigurerbar maskinvare. I tillegg så presenteres en kjapp gjennomgang av de teknologiske prinsippene som ligger bak og en rask presentasjon av forskjellige bruksområder. Til slutt i kapittelet så blir noen av problemene med rekonfigurerbar maskinvare presentert.

2.1.1 Typer av rekonfigurasjon

Når en snakker om rekonfigurerbarhet så er det vanlig at en deler det inn i forskjellige klasser avhengig av hvordan og når rekonfigureringen blir gjort. Det vanligste er at en har en hovedinndelingen av statisk og dynamisk rekonfigurerbar.

2.1.1.1 Statisk rekonfigurasjon

I statisk rekonfigurasjon blir en gitt konfigurasjon lastet inn i den rekonfigurerbare maskinvaren ved oppstart (strøm på), og blir liggende der til maskinvaren slås av (strøm av). Dette skjer ofte ved at omliggende maskinvare laster inn en konfigurasjon som ligger lagret på et permanent medium(PROM[†], Flash eller lignende).

Ved å benytte denne teknikken så utnytter en ikke fleksibiliteten til rekonfigurerbar maskinvare fullt ut. Siden rekonfigurerbar maskinvare ikke har den samme ytelsen som en ren ASICs-løsning (lavere klokke-frekvens), så er dette ofte en dårlig løsning for systemer som er ferdig utviklet, og ikke trenger dynamisk rekonfigurering.

Men det finnes likevel grunner til å benytte den. Siden rekonfigurerbar maskinvare som er "hyllevare", er mye billigere en en ASIC-løsning som må spesialproduseres, er dette en ypperlig løsning som plattform for utvikling av ny maskinvare. Dette er fordi en kan utvikle prototyper av det ferdige

[†]PROM - Programable Read Only Memory

produktet uten å måtte produsere nye ASICs (og omliggende maskinvare) for hver versjon en ønsker å teste ut. Dette kunne blitt svært dyrt hvis en måtte produsere en ny ASIC for hver nye testversjon.

Statisk rekonfigurerbar maskinvare kan være aktuelt å benytte i det ferdige systemet også på grunn av lave kostnader (gitt lave produksjons mengder), i tillegg til at det gjør det mulig ved et senere tidspunkt å oppgradere systemet med ny funksjonalitet eller bedre algoritmer hvis det skulle være ønskelig.

2.1.1.2 Dynamisk rekonfigurasjon

Med dynamisk rekonfigurasjon mener man som regel at en har en plattform for rekonfigurerbar maskinvare hvor en laster opp forskjellige konfigurasjoner etterhvert som en trenger dem. Herunder er det vanlig å skille mellom følgende undergrupper:

Kompileringstid-rekonfigurasjon I denne formen for rekonfigurering blir maskinvaren konfigurert ved program-oppstart, og blir liggende statisk gjennom hele programmets kjøretid. Det er derfor mulig å definere denne formen for rekonfigurering som statisk siden den ikke forandrer seg under kjøring. Men siden forskjellige applikasjoner kan benytte den samme rekonfigurerbare maskinvaren med forskjellige konfigurasjoner etter hverandre, er det vanlig å definere denne formen for rekonfigurering for dynamisk. Hovedforskjellen ligger i at det er applikasjonsprogrammet som laster inn konfigurasjonen, istedenfor at det gjøres automatisk ved oppstart av maskinvaren.

Kjøretids-rekonfigurasjon Her blir den rekonfigurerbare maskinvaren rekonfigurert flere ganger under kjøring etter hvert som en trenger forskjellig funksjonalitet. Det vil si at en applikasjon benytter flere forskjellige konfigurasjoner for den samme rekonfigurerbare maskinvaren under en og samme kjøring.

Herunder er det vanlig å skille mellom følgende typer:

Singel-kontekst I denne varianten er maskinvaren istand til å holde på en konfigurasjon om gangen. Endring av funksjonaliteten krever da en rekonfigurasjon av hele den rekonfigurerbare maskinvaren.

Multi-kontekst Denne varianten er litt mer avansert da den er istand til å holde på flere konfigurasjoner samtidig. Brukeren kan da etter å ha konfigurert de forskjellige kontekstene med forskjellige konfigurasjonene, spesifisere hvilken som til enhver tid er aktiv. Denne varianten av rekonfigurering kan sees på som en variant av

partiell rekonfigurering siden det kan være mulig å endre en inaktiv kontekst mens en annen aktiv kontekst er aktiv (blir kjørt).

Partiell rekonfigurasjon Partiell rekonfigurering gjør det mulig å bare konfigurerer deler av den rekonfigurerbare maskinvaren, mens resten av brikken forsetter uforstyrret. Dette krever spesiell støtte i den rekonfigurerbare maskinvaren, og blir sett nærmere på i kapittel 2.2.

Det kan være vanskelig å se forskjellen på multi-kontekst og partiell rekonfigurering, siden de begge tilbyr rekonfigurering av deler av brikken uten å forstyrre resten av brikken. Forskjellen ligger i at ved multi-kontekst rekonfigurering så er bare en kontekst aktiv til enhver tid, selv når brikken ikke undergår rekonfigurering. Ved partiell rekonfigurering så kan alle moduler som er konfigurert på brikken og ikke undergår rekonfigurering være aktive samtidige.

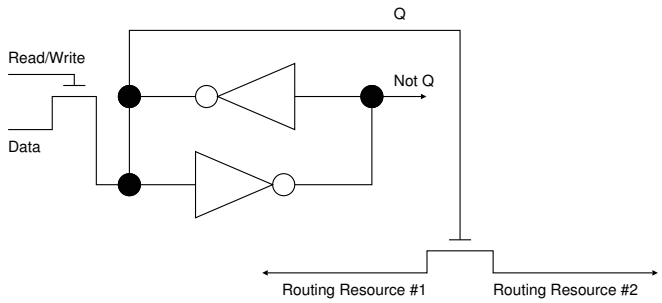
2.1.2 Typer av rekonfigurerbare maskinvare

I dag finnes det flere typer av rekonfigurerbar maskinvare. De aller fleste av disse baserer seg på et SRAM[‡] basert konfigurasjonsminne hvor hver bit styrer et kontroll-punkt. Disse kontroll-punktene bestemmer i sin tur hvordan den rekonfigurerbare maskinvaren oppfører seg.

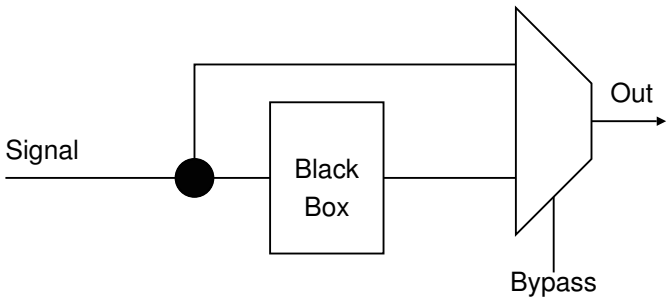
Et eksempel kan være hvordan et slikt minne bestemmer om to rutings ressurser er koblet sammen. Dette ser vi i figur 2.1. Det vi ser er et 1-bits SRAM-basert minne som styrer en transistor som kobler sammen to rutings ressurser. Et annet eksempel kan være at en bruker et slikt konfigureringsbit for å bestemme om et signal skal gå direkte videre eller om det skal innom en eller annen prosess på veien, for eksempel vente på et klokkesignal (se figur 2.2. Denne oppførselen styres av signalet '*Bypass*').

En vanlig metode for FPGA er å benytte disse rekonfigureringsbitene til å bygge opp logiske enheter. Det vil si at en har n-bit med innputt som styrer en multiplekser som velger en av 2^n rekonfigureringsbit som sendes som output. På denne måten kan en definere en hvilken som helst logisk funksjon med n-bits innputt. I Figur 2.3 ser vi dette for en multiplekser som har 3 innputt (I1-I3) og velger blant 8 rekonfigureringsbit (P1-P8). En slik konstruksjon kalles ofte for en LookUp Table (LUT)

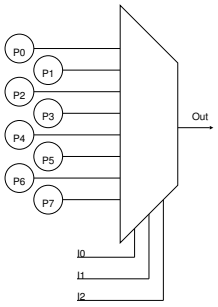
[‡]Bruken av betegnelsen RAM (Random Access Memory) er ikke alltid gyldig da ikke alle arkitekturer støtter direkte tilgang til konfigurasjonsminnet, men er likevel en generelt akseptert betegnelse på dette minnet.[CH02]



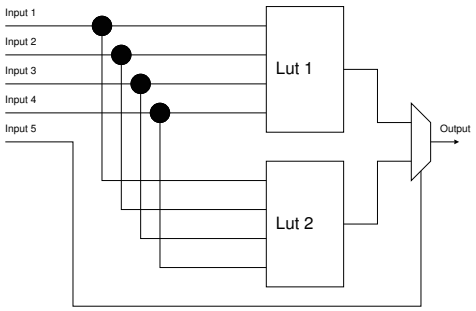
Figur 2.1: SRAM kontrollert ruting ressurs. (tegning basert på [CH02])



Figur 2.2: Valg av signal (tegning basert på [CH02])



Figur 2.3: 3 bits LookUp Table (tegning basert på [CH02])



Figur 2.4: 5bits CLB med 2 4bits LUT (tegning basert på [CH02])

Som sagt så finnes det flere typer av rekonfigurerbar maskinvare. Det er lett og tenke seg hvordan en kan benytte metodene beskrevet ovenfor til å sette sammen en enkel variant.

En av de mer vanlige variantene av rekonfigurerbar maskinvare, og den som vi bruker i denne oppgaven, er FPGA. Disse kan være satt sammen på flere måter, men felles for dem er at de inneholder en matrise av konfigurerbare logiske blokker (CLB), som ofte er implementert ved hjelp av en eller flere LUT'er. (figur 2.4)

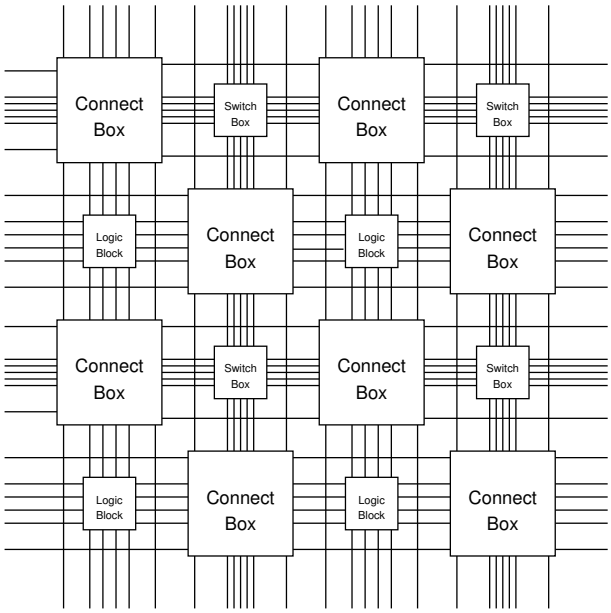
I forskjellige andre typer av rekonfigurerbar maskinvare kan disse blokkene være alt fra slike enkle LUT-blokker til enkle ALU-er (Arithmetic Logic Unit).

En mulighet som eksisterer både for FPGA og andre typer av rekonfigurerbar maskinvare er heterogene matriser. Dvs at hvert element i matrisen ikke nødvendigvis er det samme. Dette konseptet kan dermed utnyttes til å inkludere spesialiserte celler slik at en får tilgang til for eksempel multiplikatorer eller dedikerte minne-enheter uten å måtte bruke plass på brikken for å definere denne funksjonaliteten selv.

Disse blokkene er koblet sammen gjennom et nett av rutings-ressurser. Måten dette gjøres på varierer fra arkitektur til arkitektur. En vanlig teknikk her er å legge inn kobling/switch-bokser imellom de logiske blokkene (figur 2.5).

Koblingsboksene inneholder endel multipleksere som velger hvilke signal som skal sendes ut på de forskjellige linjene, mens switch-boksene inneholder transistorer som kobler sammen en horisontal og en vertikal linje, og på den måten muliggjør at et signal kan skifte retning.

Når en ser på figur 2.5 er det lett å få inntrykk av at mesteparten av FPGA brikkene går med til forskjellige rutings-ressurser. Dette er også tilfelle. Bare en liten andel av arealet på brikken går med til å forme CLB'er. Dette kommer av at effektive rutings-ressurser er nødvendige for å få høyest mulig ytelse.



Figur 2.5: Generisk øy-ruting (tegning basert på [CH02])

I tillegg er det vanlig at en har flere linjer mellom koblingsboksene som kan gå over lengre avstander, for å på den måten muliggjøre raskere signaler mellom deler som ligger lengre fra hverandre siden disse signalene ikke trenger å gå igjennom så mange koblingsbokser på veien.

I [CH02] blir det i hovedsak delt inn i 2 måter å fordele disse koblingene på slik at en får både lokale og globale rutings-muligheter:

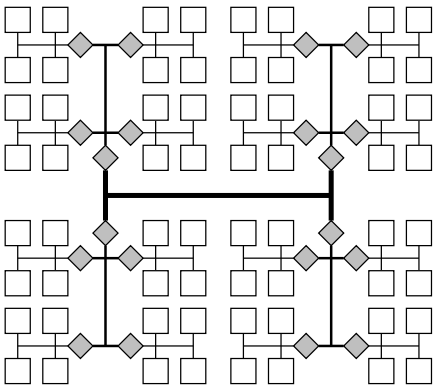
Segmentert ruting-struktur : I denne strukturen benytter en seg av korte signal-linjer som kan kobles sammen ved hjelp av switch-bokser for å emulere lengre linjer. Ofte eksisterer det endel lengre signal-linjer også, slik at en har mulighet til å sende signaler over lengre avstander mer effektivt.

Hierarkisk ruting-struktur : I denne varianten er rutingen på lokalt nivå innenfor en viss gruppe. Disse gruppene er deretter koblet sammen via switch-bokser og lengre signal-linjer til en overordnet gruppe. Dette steget kan gjentas et gitt antall nivåer. (se figur 2.6)

Andre metoder eksisterer også, men er ikke så vanlige.

2.1.2.1 Prosessor-kjerner

Etterhvert som tilgjengelig logikk i rekonfigurerbar maskinvare har økt, så har det blitt mulig å implementere komplette prosessor-kjerner på en enkel



Figur 2.6: Ruting mellom logikk-blokker(hvite blokker) vha. hierarkiske switch-bokser(mørke ruter) (tegning basert på [CH02])

brikke. Dette har gjort det mulig å implementere komplette systemet med både mikro-kontroller og omliggende logikk på en FPGA.

Istedet for å skrive sin egen prosessor-kjerne, så er det mulig å få tak i både gratis nedlastbare kjerner og kommersielt tilgjengelige kjerner.

Eksempel på en gratis nedlastbar kjerne er OpenRISC 1000 som er tilgjengelig fra Opencores.org[ope].

Av kommersielt tilgjengelige prosessor-kjerner kan en nevne ARM[arm] som tilbyr en serie av prosessor-kjerner, som blant annet blir benyttet i endel moderne mobiltelefoner.

For de kommersielt tilgjengelige prosessor-kjernene så får en som regel tilgang til support-tjenester som en del av lisensen, mens for de gratis kjernene så har en vanligvis ingen eller lite support tilgjengelig hvis en skulle støte på problemer under utvikling eller bruk.

Denne type prosessor-kjerner blir ofte referert til som myke prosessor-kjerner (soft processor cores) siden de ikke kommer som en integrert del av den rekonfigurerbare maskinvaren, men istedet er en del av rekonfigurerings-bitstrømmen.

En annen type prosessor-kjerner er de såkalte harde prosessor kjernene (hard processor core). Disse kommer som en integrert del av den rekonfigurerbare maskinvaren og trenger derfor ingen ekstra implementering for å bruke. Som eksempel på denne type prosessor-kjerne så kan en nevne PowerPC[pow]-kjernen som følger med i for eksempel Virtex-II Pro og Virtex-4 FX fra Xilinx[xil].

Programmet som skal kjøre på disse prosessor-kjernene blir da lastet opp på den rekonfigurerbare maskinvaren som en del av konfigurerings-bitstrømmen.

2.1.3 Applikasjons-områder for rekonfigurerbar maskinvare

Rekonfigurerbar maskinvare kan benyttes innen for flere applikasjons-områder. Her presenteres endel områder og scenarioer hvor rekonfigurerbar maskinvare med fordel kan benyttet framfor tradisjonell ASICs eller programvare.

Rekonfigurerbar maskinvare kan med fordel benyttes i systemer der oppgavetypen som skal utføres varierer mye, og ikke alltid er kjent på forhånd. I tillegg så kan det nevnes at rekonfigurerbar maskinvare er et billig alternativ til ASICs siden rekonfigurerbar maskinvare etterhvert er blitt "hyllevare". Dette gjør at prisene er betraktelig lavere enn hvis en skulle produsere en spesialtilpasset ASIC gitt at en bare trenger et begrenset antall brikker.

Typiske applikasjons-områder for rekonfigurerbar maskinvare er bildebehandling, signalbehandling og kryptografi.

For bildebehandling og signalbehandling så kan det være ønskelig å benytte forskjellige type filtre. Et eksempel på dette er [KAD03] hvor Kessal, Abel og Demigny beskriver ARDOISE, et system for sanntids bildebehandling.

Innenfor kryptografi gjør beregnings-intensive aritmetiske operasjoner med veldig store operatorer at rekonfigurerbar maskinvare egner seg godt. Spesielt gjelder dette for nøkkel-basert kryptografi. I [FPG+03] viser Fidanci, Poznanovic, Gaj, El-Ghazawi og Alexandridis at rekonfigurerbar maskinvare er godt egnet til trippel DES[§] (3DES) som er en teknikk for for å benytter tre nøkler til å kryptere en melding 3 ganger ved hjelp av DES-algoritmen.

Andre områder kan også ha nytte av rekonfigurerbar maskinvare. Blant disse har en evolusjonær maskinvare. Rekonfigurerbar maskinvare blir da ofte benyttet som en underliggende plattform for forskjellige evolusjonære algoritmer. Et problem her er at en låser seg til en gitt arkitektur, slik at det blir vanskelig å sammenligne resultater fra forskjellige arkitekturer. I tillegg er ikke dagens plattformer for rekonfigurerbar maskinvare designet på en slik måte at det ikke alltid er like enkelt å avbilde et design for evolusjonær maskinvare på den. I [HT01] presenterer Haddow og Tufte et design for en virtuell FPGA som har den egenskapen at det er enkel å avbilde et evolusjonært design på den, spesielt celle-baserte design. Denne virtuelle FPGAen kan deretter implementeres på en hvilken som helst FPGA-arkitektur.

Som tidligere nevnt i kapittel 2.1.1.1, så kan også mange systemer som ikke trenger dynamisk rekonfigurerbarhet, ha nytte av rekonfigurerbar maskinvare. Rekonfigurerbar maskinvare er nyttig i forbindelse med utvikling av maskinvare-systemet da det tillater å raskt kunne ha en prototype klar til

[§]DES - Data Encryption Standard

testing etter at en har gjort forandringer i systemet uten å måtte gå igjennom en lang og kostbar produksjons-prosess. En annen ting er at når en har et komplett system som kjører i en rekonfigurerbar maskinvare, så kan en spare mye tid og ressurser på å ikke produsere en egen brikke, men istedet benytte den rekonfigurerbare maskinvaren i det ferdige designet. Slike standard rekonfigurerbar maskinvare er ofte produsert i store antall, og derfor ofte tilgjengelig til en mye lavere pris enn hvis en skulle produsert en spesialtilpasset brikke for designet, så lenge en ikke har behov for store mengder. Hvis en har behov for store mengder, kan det lønne seg å benytte en ASICs-løsning. Rekonfigurerbar maskinvare muliggjør også oppgraderinger av designet uten å måtte bytte ut noen fysiske komponenter.

Xilinx[xil] tilbyr en spesial-løsning for dette i deres EasyPath FPGAs[eas]. Dette er et system hvor kundene kan sende inn deres system design sammen men spesifikasjon om hvilke FPGA det kjører på, når de vet at systemet deres er ferdig utviklet og ikke lengre vil trenge den fulle rekonfigureringsmuligheten til en FPGA. EasyPath FPGAene til Xilinx er basert på deres standard FPGA serier. EasyPath FPGAene blir da kontrollert med det spesifiserte designet som kontrollsystem, slik at det er garantert å fungere med dette designet. Det blir da ikke foretatt en like grundig kvalitetskontroll som med en generisk FPGA, da den fulle funksjonaliteten ikke lengre trengs. Dette medfører at EasyPath FPGAene er enda billigere enn de tilsvarende komplette FPGAene.

Likevel bør en fremdeles tenke på at ved store produksjonskvantum, så kan spesialdesignede brikker bli enda billigere enn rekonfigurerbar maskinvare.

2.1.4 Eventuelle problemer ved å benytte Rekonfigurerbar Maskinvare

Selv om rekonfigurerbare maskinvare kan gi ytelse langt over det som er mulig i ordinære prossessorenheter (kapittel 2.3.6), så er det ikke alltid like rett frem.

Et problemene er knyttet til tiden det tar å konfigurerer maskinvaren. For statisk eller kompileringstids-rekonfigurasjon så er ikke dette noe stort problem da en ikke trenger og forandre på konfigurasjonen underveis. Men for mer dynamiske systemer hvor det er nødvendig med rekonfigureringer underveis, så kan tiden som går bort i rekonfigurering veldig lett overstige tiden en sparer på å kjøre applikasjonen i maskinvare. Grunnene til at en kan trenge flere rekonfigureringer underveis er mange.

Plassbehov : Den delen av applikasjonen som skal kjøre i maskinvare er for

stor til å få plass på en enkel brikke, og må derfor deles opp i mindre deler.

Modularisering : En applikasjon kan bruke et sett med ferdig-definerte maskinvare-moduler. Siden utvikler at disse modulene ikke nødvendigvis vet hvordan de vil bli benyttet så er de ofte tilgjengelige som separate moduler.

Fleksibilitet : En vet ikke nødvendigvis nøyaktig hvilke algoritmer en trenger til en applikasjon før under kjøring. Dette kan komme av at valg av algoritme-implementasjon kan være avhengig av størrelse på innputt, eller som i vårt system at brukeren skal kunne spesifisere hva systemet skal benyttet til og dermed også hvilke algoritmer som trengs.

Flere problemer med rekonfigurerbar maskinvare eksisterer og er blitt jobbet med. For eksempel kan en nevne strøm-forbruk[KV04, MM04], men dette er ikke innenfor omfanget av oppgaven vår og vil vil derfor ikke gå nærmere inn på det.

Et annet problem er at hastigheten en får på rekonfigurerbar maskinvare er som tidligere nevnt mindre enn hva en kunne ha fått hvis en hadde valgt å implementere designet i en ASIC. Dette kommer av at designet blir avbildet over på en i utgangspunktet ganske kompleks struktur. Dette gjør av signalene har lengre å gå enn hvis designet hadde blitt direkte implementert på en ASIC, noe som gjør at den høyeste klokkefrekvensen som er mulig å få ut av et rekonfigurerbart maskinvare-design er vesentlig lavere.

2.2 Partiell rekonfigurering

Partiell rekonfigurering er i prinsippet ikke så ulikt vanlig rekonfigurering. Forskjellen ligger i at i bit-filen som en sender til maskinvaren så ligger det informasjon om hvilke deler av maskinvaren som skal rekonfigureres. En kan tenke seg at hvert kontroll-punkt i den rekonfigurerbare maskinvaren (se kapittel 2.1.2) har en gitt adresse. Dette er i motsetning til ordinær rekonfigurering hvor bit-filen begynner på starten av FPGAen og fortsetter til enden av den.

2.2.1 Forskjellige typer FPGA

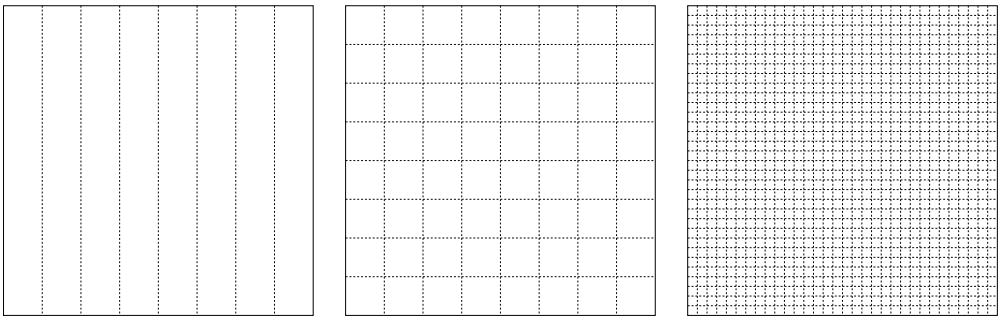
Som det ble nevnt i kapittel 2.1.2, så finnes det flere typer av rekonfigurerbar maskinvare. FPGA er en av disse typene, men ikke alle variantene av FPGA støtter partiell rekonfigurering. Også blant de variantene som støtter partiell rekonfigurering er det variasjoner i på hvilke måte dette gjøres. Jeg har allerede nevnt multi-kontekst FPGAer. I denne varianten så har en muligheten til å konfigurerer en kontekst mens en annen er aktiv uten å forstyrre denne.

Men også i singel-kontekst finnes det endel arkitekturer som støtter partiell rekonfigurering. I disse arkitekturene så kan en rekonfigurere deler av maskinvaren mens resten forblir uforandret og kjører videre uten å bli forstyrret. Men også innenfor disse så er det endel forskjeller i hvordan den partielle rekonfigureringen foregår. For den vanlige brukeren så blir de fleste forskjellene skjult av programvaren som automatisk genererer bit-filen, men noen forskjeller er en nødt til å ta hensyn til selv. En av disse er hvordan den rekonfigurerbare maskinvaren deler inn brikken med tanke på adressering. Dette kan i første omgang høres relativt trivielt ut, men det er knyttet endel restriksjoner til enkelte av disse. I hovedsak så finnes det 3 varianter

1D ramme-inndeling I denne enkleste inndelingen så deles brikken inn i et sett med kolonner* av en viss bredde. Kolonnene opptar hele høyden av brikken. (figur 2.7(a)) og er den minste enheten som kan rekonfigureres. I prinsippet så kan en se på arkitekturer som ikke støtter partiell rekonfigurering som medlemmer av denne klassen, men med bare 1 kolonne.

2D ramme-inndeling En litt finere oppdeling enn den ovenfor da kolonnene her ikke spenner hele høyden av brikken, men i tillegg er delt inn i rader (figur 2.7(b)).

*eller rader, prinsippet er det samme



(a) 1-Dimensjonal ramme-inndeling

(b) 2-Dimensjonal ramme-inndeling

(c) 2-Dimensjonal Fine-Grain ramme-inndeling (RAM modell)

Figur 2.7: Forskjellige nivå av inndeling av partiell rekonfigurerbar maskinvare

2D finkornet ramme-inndeling Denne modellen er egentlig den samme som ovenfor, men her er inndelingen så fin at en kan begynne å snakke om at en har direkte tilgang til kontroll-punktene (figur 2.7(c)).

Når en skal designe et system som benytter seg av partiell rekonfigurering så deler en ofte systemet opp i en del separate moduler som en bytter inn og ut etterhvert som en trenger dem ved hjelp av partiell rekonfigurering. Hver av disse modulene vil da ha sin egen bit-fil. I tillegg så har en ofte en toplevel-modul. Denne består av all logikken som ligger rundt de delene som byttes ut. Denne toplevel-modulen blir da liggende statisk, og kan f.eks. bestå av en logikk som tar seg av kommunikasjon mellom bruker og de forskjellige utbyttbare modulene. Hvis en har satt av plass til å ha flere rekonfigurerbare moduler inne samtidig på forskjellige deler av brikken så kan toplevel-modulen også ha som oppgave og ta seg av kommunikasjon mellom disse.

Her oppstår det riktignok et problem. For at signalene skal kunne gå konsistent inn og/eller ut av de forskjellige modulene, så er det nødvendig å ha nøyaktig samme ruting for disse signalene i ytterkant av samtlige av de aktuelle modulene. Dette kan løses ved å benytte såkalte buss-makroer. Dette er ferdig syntetiserte og rutet logikk som en kan plassere mellom modulene og toplevel-modulen. På denne måten så kan en sikre at signalene kommer fram på korrekt måte. Disse buss-makroene er av fast bitbredde (4bit for Xilinx Virtex-II) og er ofte ganske restriktive med tanke på hvilke vei signalene skal gå. (For eksempel bare inn eller bare out, ikke begge veiene(inout))

For 1D ramme-inndeling eksisterer det et ytterlig problem. En ting er å få et signal til å gå mellom 2 moduler, men å få et signal til å gå forbi en modul til en annen modul er mye verre. For det første så må modulen i midten koble sammen 2 buss-makroer for at signalet skal komme fram, men i tillegg så kommer problemet med at signalet kan bli forstyrret hvis modulen i midten blir rekonfigurert. Dette gjør at modulene på sidene må vente med å sende signaler mellom seg mens rekonfigureringen av midt-modulen finner sted. Dette medfører at eventuelle forsøk på å skjule rekonfigureringstiden ved å gjøre noe annet i parallell ikke alltid fungerer på bra.

2.2.2 Strategier for partiell rekonfigurering

Det finnes 2 hovedstrategier for generering av bitstrømmer for partiell rekonfigurering. Disse blir som regel kalt modul-basert og forskjell-basert.

2.2.2.1 Modul-basert design

Den vanligste metoden for design er at en først deler designet inn i forskjellige moduler, og plasserer disse rundt om kring på brikken. Moduler som skal være rekonfigurerbare blir plassert innenfor det samme området, og et felles grensesnitt mot resten av brikken blir definert ved hjelp av buss-makroer. Et eksempel kan være at en i et design ønsker en prosesserings-enhet, men ikke har plass til å implementere en komplett ALU. Designet blir da delt inn i forskjellige enheter, bl.a en generisk ALU. En kan da implementere forskjellige spesifikke ALU'er. En som tar seg av addisjon/subtraksjon/binæroperatorer, en for multiplikasjon og en for divisjon. For hver av disse så genererer man en bit-fil. En genererer også en bit-fil for resten av designet (kalt toplevel), som ikke nødvendigvis trenger å inneholde noen av ALU-implementasjonene. Ved bruk begynner en da med med å først laste inn bit-filen for toplevel, for så og laste inn bit-filen til en av ALU'ene. Når en ønsker å benytte en annen ALU, så laster en ganske enkelt bare inn den tilsvarende bit-filen, toplevel vil ikke bli berørt, da bit-filene til modulene bare spesifiserer at arealet som den generiske ALU'en opptok skal overskrives. En kan om en ønsker det la flere deler av systemet være rekonfigurerbare, for eksempel forskjellige implementasjoner av minne-behandlere/register-filer.

2.2.2.2 Forskjell-basert design

En annen måte og se på designet er å sammenligne 2 komplette maskinvaredesign og på grunnlag av dem generere bit-filer som bare inneholder informasjon om hvilke konfigureringsbiter som er forskjellige. Denne formen egner

seg for design hvor forskjellene mellom de forskjellige designene er minimale, siden en ikke nødvendigvis må rekonfigurere alle rammene som en modul opptar, men bare de som ved en komplett rekonfigurering, ville blitt endret.

2.3 Tidligere arbeider

Som nevnt så er ett av de største problemområdene innenfor dynamisk rekonfigurerbar maskinvare tiden som går bort i i forbindelse med rekonfigureringer. Hvis en ikke tenker på dette når en designer et system som benytter seg av hyppige rekonfigurering, så risikerer en at den totale kjøretiden øker drastisk selv om maskinvare-modulene en benytter er betydelig raskere enn tilsvarende implementasjoner i programvare. Denne oppgaven har som hovedmål å redusere antall rekonfigureringer og dermed rekonfigurerings-tiden og den totale kjøretiden. Gjennom tidene er det gjort endel andre arbeider som også har som mål å redusere total kjøretid.

Nedenfor presenteres flere arbeider som er blitt gjort med tanke på å redusere forsinkelsen i forbindelse med rekonfigurering i tillegg til et eksempel som viser nytten av disse teknikkene i forbindelse med bildebehandling.

Helt til slutt i dette del-kapittelet kommer en presentasjon av Crays nye superdatamaskin XD1. Dette er en superdatamaskin med større mengder av rekonfigurerbar maskinvare tilgjengelig. Denne har 6 Virtex-II PRO FPGAer fra Xilinx[xil] tett koblet opp mot Opteron prosessorer fra AMD[amda].

2.3.1 Generiske prosesserings-enheter kontra oppgave-spesifikke moduler

En av de meste effektive implementasjonene i maskinvare med tanke på plass er en tradisjonell Von Neumann maskin. Disse tar ofte større plass enn oppgave-spesifikke moduler. Men siden en ofte har behov for å løse flere forskjellige typer oppgaver så kan en spare mye plass på å implementere en enkel maskinvare-modul som er istand til å handtere flere typer oppgaver. Hvis det derimot er hastighet en er ute etter så kan Von Neumann maskiner være et dårlig valg da de utfører oppgaver sekvensielt og derfor sannsynligvis vil bruke lengre tid enn en oppgave-spesifikk modul. Forholdet mellom hvor mye plass en totalt sparer, og hastigheten en taper ved å benytte slike sekvensielle fleksible moduler blir ofte omtalt som "cost of generalization".

På en annen side så kan en totalt spare endel tid ved å kjøre noen av oppgavene på slike sekvensielle prosessor-kjerner. Dette kommer av at siden de er istand til å utføre mange forskjellige typer oppgaver, så trenger en ikke alltid å rekonfigurere maskinvaren når det kommer en oppgave av en annen type inn.

I [LWW03] har Loo, Wells og Winningham forsøkt å se på muligheten for å best mulig utnytte fleksibiliteten som rekonfigurerbar maskinvare tilbyr i forhold til hvor stor del av systemet det er mulig å få plass til på brikken. Dette gjør de ved å designe forskjellige rekonfigurerbare maskinvare-moduler.

Metode benyttet	Tids enheter
Teoretisk minimum	1720
Genetisk Algoritme	1742
Simulated Annealing	2600
Tilfeldig	2866

Tabell 2.1: Resultater fra eksempel i [LWW03].

Maskinvare modulene de benytter er enten Processor Cores (PCs), Task Specific Cores (TSCs) eller Communication Core Elements (CCEs).

I deres modell så representerer PCs von Neumann-baserte prosesseringsenheter med sekvensiell eksekvering, hver med sine spesialfelt. For eksempel så kan en modul ha raskere multiplisering enn en annen modul som kan ha raskere divisjon.

TSCs er i motsetning til PCs ikke generelle av natur, men benyttes for å løse en spesifikk oppgave.

For kommunikasjon mellom PCs og TSCs så benyttes CCEs. Modellen deres støtter både synkrone (buffret) og asynkrone (ubuffret) CCEs.

I [LWW03] ser de på muligheten for å benytte genetiske algoritmer for å på en best mulig måte planlegge i hvilken rekkefølge oppgavene skal sendes inn i systemet, og hvorvidt en oppgave skulle utføres på en TSC eller PC. Som et eksempel så viser de til forsøk med å sende 100 oppgaver til systemet.

Begrensningene i systemet var satt slik at det var ubegrenset med funksjonsenheter, men det ikke var plass til å implementere alle oppgavene i TSCs, og ingen av PCs kunne utføre alle oppgavene. Hvilken art disse oppgaven var av ble ikke beskrevet i [LWW03].

Se tabell 2.1 for resultatet av å benytte en genetisk algoritme til å finne fram til raskeste kjøretid sammenlignet med teoretisk raskeste hastighet, tilfeldig fordeling og simulated annealing. Dette viste at en kunne spare en del av tiden som gikk med til rekonfigurering ved å se på rekkefølgen og kobling mellom oppgave og modul.

2.3.2 Modul-plassering

Et annen aspekt av dynamisk rekonfigurering er oppgave flytting. Det vil si at etter at en eller flere deler av den rekonfigurerbare maskinvaren er blitt konfigurert så kan det være ønskelig å flytte rundt på de konfigurerte modulene. Grunnen til dette kan for eksempel være fragmentering av det tilgjengelige området siden de forskjellige modulene ikke nødvendigvis opptar like mye av arealet tilgjengelig på brikken. Dette kan medføre at systemet må vente med å konfigurerer maskinvaren for en oppgave som det i utgangspunktet er nok

plass til, men på grunn av fragmentering så er det ingen sammenhengende områder som er store nok. Dette vil deretter føre til systemet må vente til noen av de kjørende oppgavene blir ferdige. For det første så medfører dette i seg selv en forsinkelse. Men en ser også at med mindre oppgaven som akkurat ble fullført benyttet samme modul som oppgaven en forsøker å sende inn, så vil systemet bli nødt til å overskrive en potensielt hyppig brukt modul selv om det total ledige arealet på brikken tilsa at det ikke skulle være nødvendig.

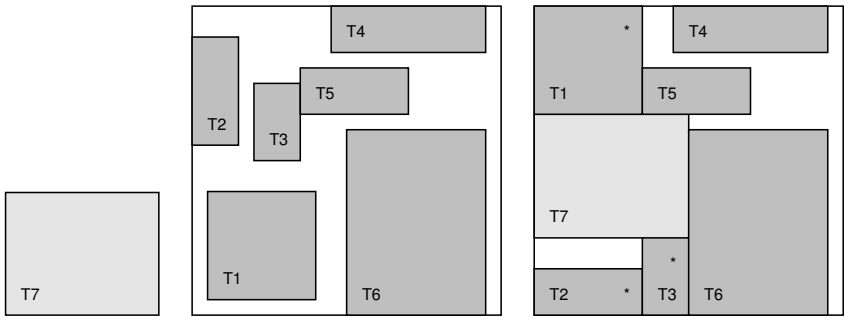
I [EMSS00, MSSE02] ser ElGindy, Middendorf, Scheuermann og Schmeck på muligheten for å utvikle 2 genetiske algoritmer (GA) for å løse dette problemet. Den første av dem skal brukes for å identifisere mulige rearrangeringer, mens den andre skal planlegge en mulig rearrangering på en slik måte at kjøretids-forsinkelsen til de kjørende oppgavene blir minimal.

I deres modell er FPGAen modellert som en rektangulær matrise av rekonfigurerbar logikk og rutings-ressurser fordelt over flere moduler. Hver modul er antatt å være uavhengige av hverandre og ikke-overlappende submatriser på FPGAen. De antar også at rekonfigureringstiden for en modul er proporsjonal med arealet den opptar. I tillegg til å anta at moduler kan flyttes uten å ødelegge intern status i de affekterte modulene så antas det også at modulene kan roteres 90°. Hvordan de tenker å gjøre dette blir ikke forklart, men det kan tenkes at det er mulig for systemet å stoppe en oppgave under kjøring, hente ut statusdata, konfigurere en annen del av FPGAen med den tilsvarende modulen (eller en tilsvarende som bruker et 90° vridd område). Deretter kan det tenkes at det vil være mulig å laste inn statusdata for så å la den ny konfigurerte modulen jobbe videre med den gamle modulen sine data.

I figur 2.8 ser vi et eksempel på hvordan en kan flytte om på modulene 2.8(b) når det kommer en ny oppgave 2.8(a) som det ikke er plass til. Resultatet ser vi i 2.8(c). Modulene som er blitt flyttet på er markert med '*'.

I deres arbeid fant de ut tiden en oppgave måtte vente på å kunne kjøre ble betraktelig redusert. Dette var mulig siden andelen av FPGAen som ble benyttet økte fra 60% til 80%.

En annen måte og løse dette på er å forsøke og plassere moduler som en regner med at vi bli ferdige omtrent samtidig i nærheten av hverandre. Dette ser Ahmadinia, Bobda og Teich nærmere på i [ABT04]. De beskriver en algoritme som holder orden på hvilke deler av den rekonfigurerbare maskinvaren som er ledig og hvilke moduler som kjører hvor. Dette gjøres bla annet ved å holde rede på to horisontale linjer (se figur 2.9), hvorav den ene markerer hvor langt "opp" på FPGAen som er i bruk, mens den andre markerer hvor langt "ned". Arealet mellom disse linjene ansees som i bruk. Når en ny



(a) ny oppgave (b) FPGA før rearrangering (c) FPGA etter rearrangering

Figur 2.8: Re arrangering av moduler (basert på [MSSE02])

modul kommer inn i systemet vil algoritmen deres ikke bare finne ut hvor det er plass til den, men også hvor det passer med tanke på kjøretid. Vi ser av fig 2.9 at hvis modul 6 blir ferdig før modul 2,3,10 og 11 så vil ikke de horisontale linjene forandre seg, og plassen ansees fremdeles som opptatt og vil ikke kunne utnyttes før modul 2,3,10 eller 11 fullfører. Derfor forsøker algoritmen og plassere modulene på en slik måte at nabo-moduler vil bli ferdig på omtrent samme tid.

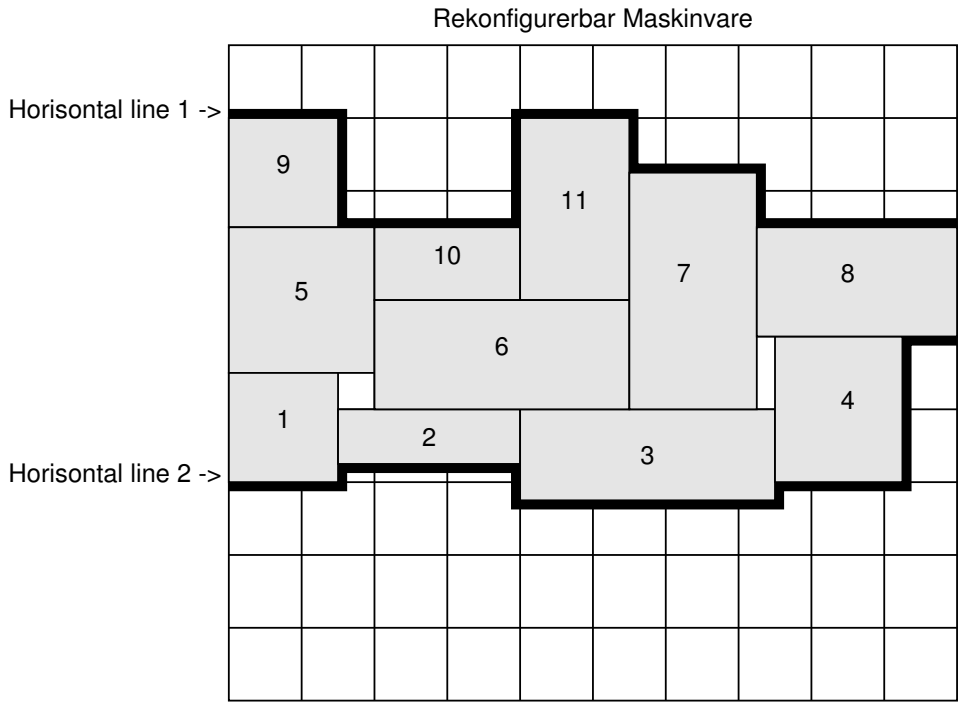
Ved testing fant de ut at de kunne oppnå en ytelses forbedring på opp mot 20% i forhold til lignende arbeider [KB00].

2.3.3 Maskinvare/Programvare-partisjonering

Noguera og Badia beskriver i [NB02b, NB02a] et system for å kjøre applikasjoner på et integrert system bestående av en verts-prosessor og en matrise med dynamisk rekonfigurerbar logikk(DRL)-celler. En applikasjon blir sett på som en strøm av oppgaver eller hendelser som skal kjøres i systemet. I deres arbeid presenterer de 2 algoritmer. Den ene tar seg av partisjonering av oppgaven mellom maskinvare og programvare. Det vil si en algoritme som vurderer lønnsomheten av å kjøre oppgaven i maskinvare kontra det å kjøre den på verts-prosessoren. Dette gjøres ved en kostnadsfunksjon. I [NB02b] foreslår de følgende kostnadsfunksjon

$$F_i = \alpha \cdot (AET_i^{HW} - AET_i^{SW}) + \beta \cdot SVM_i \tag{2.1}$$

hvor



Figur 2.9: Horisontale linjer for å holde rede på ledig areal (basert på [ABT04])

AET_i^{HW} - Gjennomsnittlig kjøretid for maskinvare implementasjon av en oppgave.

AET_i^{SW} - Gjennomsnittlig kjøretid for programvare implementasjon av en oppgave.

SVM_i - Nødvendig minne-størrelse for tilstands variabler for en oppgave klasse.

Denne kostnadsfunksjonen ser altså på forholdet mellom forskjellen i antatt kjøretid for oppgave-klassen ($AET_i^{HW} - AET_i^{SW}$) og størrelsen på tilstands-variablene til oppgave-klassen (SVM_i), og balanserer disse etter de to faktorene α og β. Som en ser av funksjonen så vil oppgaver som lønner seg å kjøre i maskinvare ha en lav kostnadsverdi, så systemet klassifiserer derfor oppgavene fra lav til høy verdi.

Den andre algoritmen de presenterer tar seg av planlegging av hvilke moduler som skal konfigureres inn på de forskjellige DRL-cellene. Siden systemet de beskriver kjører en oppgave om gangen så er det istand til å konfigurere DRL-cellene som trengs for framtidige oppgaver mens den aktive oppgaven kjører.

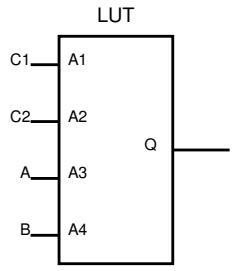
De presenterer 2 varianter av hver algoritme. Både for delt minne arkitekturer og lokalt minne arkitekturer.

2.3.4 Kontekst bytting

En annen interessant metode for å redusere denne tiden på blir presentert av Torresen og Vinger i [TV02]. Her beskriver de en måte å designe en egen multi-kontekst FPGA inne i en allerede eksisterende singel-kontekst FPGA. Ved å benytte multipleksere for å angi hvilken UDCR (User Defined Configuration Registers, eller kontekst-minne) som er aktiv til enhver tid, så har en mulighet til å bytte mellom de forskjellige kontekstene iløpet av bare 1 klokkesykel. Her beskriver de hvordan en kan lagre konfigurasjons-bitstrømmene for de forskjellige UDCR, og hvordan de på en effektiv måte kan implementere logiske funksjoner ved å la en enkel LUT (LookUp Table) implementere en 2-bits funksjon i tillegg til ruting av to innputt A og B. (se figur 2.10).

A og B er innputt til funksjonen mens $C1$ og $C2$ bestemmes av bitstrømmen og definerer funksjonen etter tabell 2.2.

Dette designet vil egne seg godt til implementasjon på rekonfigurerbar maskinvare som ikke støtter partiell rekonfigurering, og mengden av tilgjengelig logikk er tilstrekkelig til å implementere denne funksjonaliteten i tillegg til designet.



Figur 2.10: Utnyttelse av LUT i brukerdefinert logikk (basert på [TV02])

C_1C_2	Function
00	NAND
01	B
10	A
11	NOR

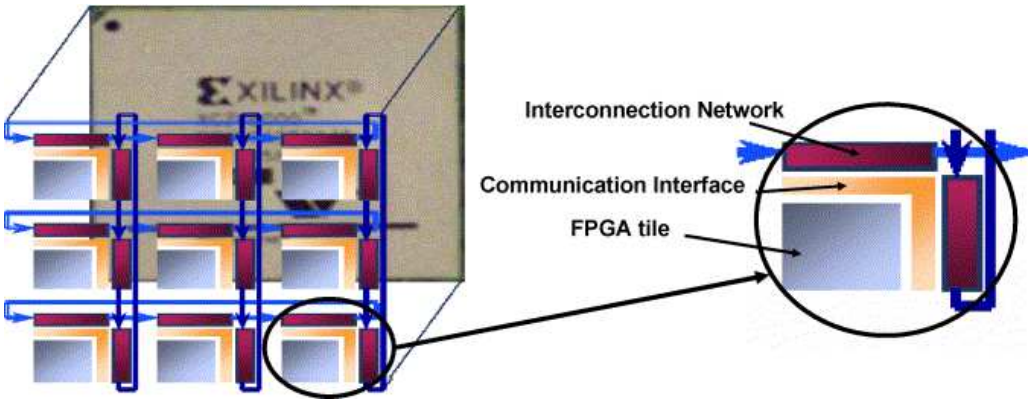
Tabell 2.2: Funksjons-definisjon av LUT i brukerdefinert logikk (basert på [TV02]).

2.3.5 Lignende arbeider

Denne oppgaven fokuserer på å lage et system for å dynamisk fordele maskinvare-moduler til forskjellige deler av en rekonfigurerbar maskinvare. Dette gjør vi på en slik måte at de maskinvare-modulene som oftest benyttes blir liggende. På denne måten håper vi å reduserer tiden som går bort til rekonfigurering.

Lignende forsøk er blitt gjort av blant annet Resano, Mozos, Verkest, Vernalde og Catthoor. I [RMV+03] og [RVM+04] har de sett på muligheten for å redusere denne tiden ved å benytte gjenbruk og forhånds-henting. Gjenbruk vil si at en forsøker å redusere antall rekonfigureringer ved å la modulene bli liggende i maskinvaren over lengre tid uten å bli overskrevet, i tilfelle en senere oppgave vil trenge dem. Forhånds henting vil si at en konfigurerer en modul som en vet vil bli brukt før den trenges. Siden dette kan gjøres parallelt med at andre ferdig-konfigurerte moduler jobber så kan en gjemme konfigurasjons forsinkelsen slik at den ikke merkes når modulen trenges. I deres arbeid benytter de en ICN (Interconnect Network)-modell[MBV+02] hvor de deler en FPGA inn i flere identiske celler (tiles) som er omgitt av et kommunikasjons grensesnitt og Interconnect Network. (se figur 2.11.)

Resultatene av å simulere systemet ser vi i tabell 2.3. Første linje viser tiden testen tok da de antok at rekonfigureringen var gratis (0 ms). Linjen nr 2 i samme tabell viste hvor lang tid det tok når de tok med rekonfigureringsti-



Figur 2.11: ICN-modell for FPGA (hentet fra [RVM+04])

Ideal execution time (for 1000 iterations)	151.750 ms	Overhead(%)
Initial overhead	44.967 ms	+29
Overhead with the prefetch module	11.425 ms	+7.5
Overhead with prefetch and reuse (8 tiles)	5.366 ms	+3.5
Overhead with prefetch and reuse (18 tiles)	1.848 ms	+1.2

Tabell 2.3: Testresultater for ICN-modell[RVM+04]

den i beregningene. Når de i tillegg la til oppgave-planlegger (prefetch) så kunne de se at rekonfigureringsforsinkelsen ble redusert med en faktor på 4 (3. linje), og de kunne konstatere at rekonfigurerings forsinkelsen sank enda mer da de benyttet seg av gjenbruk av modulene (4 og 5 linje).

2.3.6 Eksempler

Det er gjort flere forsøk som viser at det er mulig å øke total-ytelsen i et system ved å benytte disse teknikkene.

Et eksempel er [PBV04] hvor Panainte, Bertels og Vassiliadis ser på hvordan rekonfigurerbar maskinvare påvirker ytelsen av et system med MPEG2 en-koding som eksempel-applikasjon.

De studerte innvirkningen av 3 av kjernefunksjonene i MPEG2-enkodingen (SAD*, DCT† og IDCT‡).

Det første de gjorde var og måle hvor stor andel av tiden MPEG2-enkodingen disse 3 funksjonene utgjorde. Ved å anta at maskinvare-modulene var så raske at de brukte neglisjerbar tid på å utføre oppgavene så kan en si at maksimal ytelses økning ved å legge en spesifikk funksjon ut på rekonfigurerbar maskinvare vil være lik andelen av tiden MPEG2-en-kodingen

*SAD - Sum of Absolute Differences
†DCT - Discrete Cosine Transform
‡IDCT - Inverse Discrete Cosine Transform

Video sequence	SAD	DCT	IDCT
Simple Scheduling Slowdown	1012 x	108 x	131 x
Out-of-loop Scheduling performance Impr.	33.61 %	24.68 %	0.75 %
Theor. Maximal Performance Impr.	37.83 %	25.54 %	1.63 %
Performance Efficiency	89 %	97 %	46 %

Tabell 2.4: Resultater av MPEG2-enkoding akselerasjons timing [PBV04].

tilbrakte innenfor den aktuelle funksjonen. Se linje 4 i tabell 2.4 for en oversikt over hvor stor andel SAD, DCT og IDCT utgjorde, og dermed teoretisk maksimum ytelses økning for hver av dem. Total maksimale ytelses økning ved å legge alle disse tre funksjonene ut på rekonfigurerbar maskinvare kan da beregnes ved å legge sammen summen av ytelses-forbedringen for hver av disse tre funksjonene. Dette gav en total teoretisk ytelses-forbedring på 65%.

Ved å teste ut hastigheten på maskinvare-modulene for disse 3 funksjonene fant de ut at de var opp til 31 ganger raskere enn tilsvarende programvare implementasjon kjørende på en AMD Athlon XP 1900++ (1600 MHz). Men de så også at hvis de benyttet en enkel planlegger som alltid rekonfigurerer maskinvaren i det funksjonen blir kalt økte den totale kjøretiden med en størrelse-orden på 2-3. (linje 2, tabell 2.4)

Men ved å benytte seg av en planlegger som var istand til å konfigurere maskinvaren før den ble brukt (Out-of-loop Scheduling) oppnådde de ved testing en økning i ytelse ved maskinvare akselerasjon (linje 3, tabell 2.4). I linje 5 i tabell 2.4 ser en hvor stor andel av den teoretisk maksimale utnyttelsen (linje 4) som ble utnyttet.

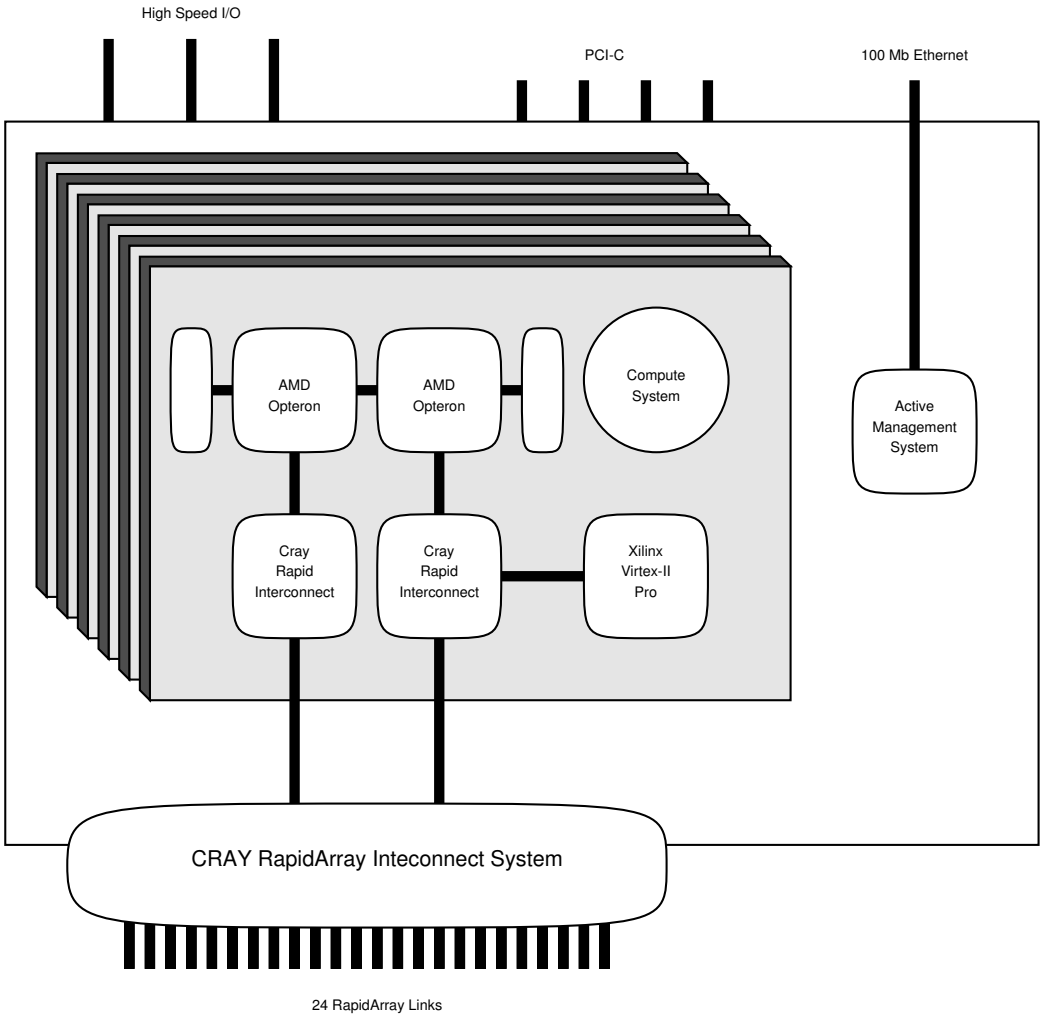
2.3.7 Nyere systemer - Cray XD1

I mange av disse tidligere arbeidene, og også i dette, er det blitt benyttet rekonfigurerbar maskinvare som er koblet opp mot en maskin via et eller annet grensesnitt. Det har tidligere vært svært vanskelig å få tak i maskinvare hvor en har en tett kobling mellom CPU og den rekonfigurerbare maskinvaren. En løsning her har vært og implementere en mikro-kontroller på for eksempel en FPGA (hard eller myk prosessor-kjerne). En annen løsning er å lage sitt eget kort.

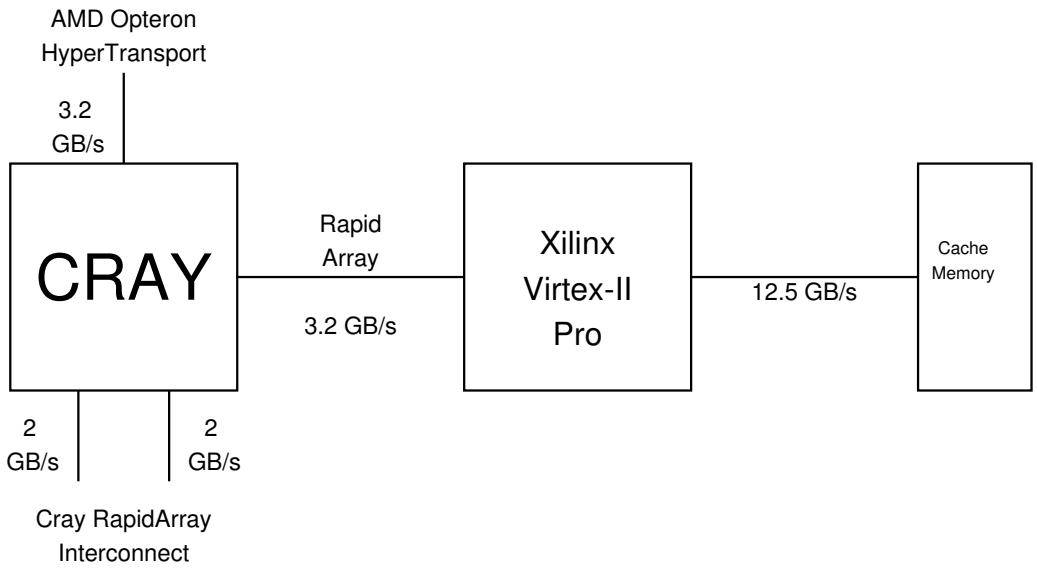
Nå nylig har Cray [crac] kommet med et produkt hvor rekonfigurerbar maskinvare allerede er tilgjengelig.

I Cray XD1 [craa] kombineres AMD Opteron [amdb] prosessorer og Xilinx Virtex-II PRO FPGAer i en superdatamaskin. Selve maskinen består av 6 enheter som er koblet sammen med CRAY Rapid Array Interconnect System. (se figur 2.12)

Hver enhet består av 2 AMD Opteron prosessorer og en Virtex-II Pro



Figur 2.12: Cray Rapid Array Interconnect System (basert på [Mor05])



Figur 2.13: Cray Rapid Array Interconnect (basert på [craa])

(XC2VP50-7) i tillegg til 16 MB med QDR RAM[§] Cache. (se figur 2.13.)

Hver FPGA er tilgjengelig for hver prosessor i maskinen, for på denne måten å tilby maksimal fleksibilitet for algoritme akselerasjon. Totalt består hver maskin-enhet av 12 AMD Opteron prosessorer og 6 Virtex-II Pro FPGAer og har en topp ytelse på 92 GFLOPs[¶]. Cray mener på denne måten å overkomme noen av de tradisjonelle problemene innenfor rekonfigurerbar maskinvare:

- *PCI flaskehals* - Mange tidligerer systemer koblet sammen prosessor og FPGAer via systemets I/O - system, for eksempel over en PCI-buss. Dette førte til begrensninger i overføringshastigheten mellom prosessor og FPGA, og hastigheten som en hadde oppnådd ved å akselerere i maskinvare gikk tapt på grunn av forsinkelser i overføringen av data til og fra. Siden Cray har valgt og koble FPGA'en direkte til deres Rapid Interconnect System, så har de en 3.2 GB/s bi-direksjonal kobling mellom prosessorene og FPGAene (1.6 GB/s hver vei). I tillegg så har Rapid Interconnect System en veldig lav forsinkelse i overføring.

- *Løs kobling mellom prosessor og FPGA* - Mange tidligere rekonfigurerbare systemer var designet som tillegg til eksisterende systemer.

[§]QDR RAM - Quad Data Rate SRAM

[¶]teoretisk topp ytelse med 2.6 GHz AMD Opteron

Dette gjorde at en hadde en begrenset integrasjon mellom operativsystemet og FPGA, noe som igjen gjorde at utvikleren måtte stå for oppgaver som planlegging av oppgave-rekkefølge og avbildning av minne mot FPGA. Dette er støttet direkte i operativsystemet som følger med XD1.

- *Mangel på programmerings-verktøy* - De verktøyene som var tilgjengelig for utvikling av rekonfigurerbar maskinvare var kjente for maskinvare-utviklere, men fremmede for programvare utviklere. Dette medførte at bare de med erfaring innen maskinvare-design kunne benytte seg av mulighetene som rekonfigurerbar maskinvare gav. Cray samarbeider med Xilinx og tilbyr en komplett pakke bestående av både maskinvare og programvare bestående av høy-nivå grafiske utviklingsverktøy for rekonfigurerbar maskinvare.

Det er mulig og koble inntil 12 slike maskiner sammen til ett system i et rack. På denne måten øker en den teoretiske topp-ytelsen til 749 GFLOPs, og antall FPGAer i systemet øker til 72.

I følge [crab] så benyttet en kunde systemet til å generere en randomisert tallrekke, som er en av kjernefunksjonene i forbindelse med en Monte Carlo-simulering. En slik Monte Carlo-simulering modererer overflate-spredningen av atomer over tusenvis av tids-steg. På denne måten oppnådde de en ytelsesforbedring på 35% i forhold til når de kjørte simuleringen uten FPGA-akselerasjon.

2.4 Plattform

Siden vi ønsket å teste ut flere aspekter ved systemet vårt trengte vi en plattform for systemet. Valget sto da mellom å bygge et eget kort med en FPGA, eller å benytte eksisterende kort som vi hadde tilgang til.

Å bygge et eget kort var det mest attraktive løsningen da dette ville medføre at vi kunne bestemme hvordan FPGAen skulle være koblet til resten av systemet. På denne måten så kunne vi optimalisere kommunikasjonen mellom driver-programmet og maskinvaredesignet. I tillegg så ville vi ha full kontroll over hva de forskjellige pinnene på FPGAen ble brukt til med tanke på partiell rekonfigurering.

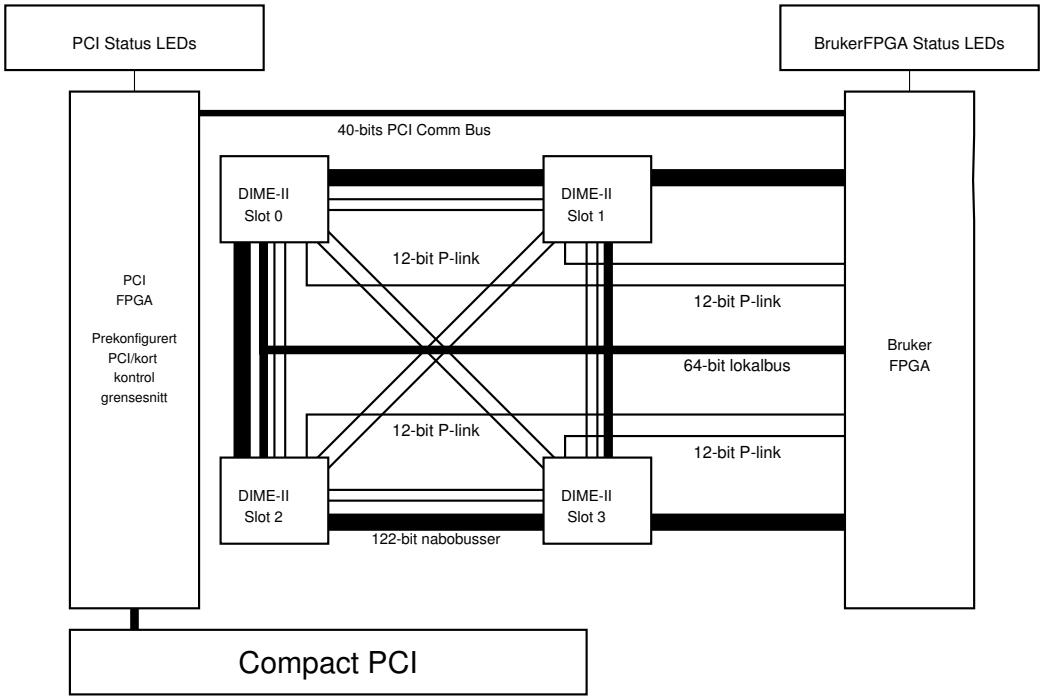
Dessverre så viste det seg at det ikke ble tid til å lage et eget test-kort, så vi bestemt av vi skulle begynne med å bruke eksisterende maskinvare for mesteparten av testingen.

2.4.1 Tilgjengelighet

Tilgjengelig så hadde vi et BenERA-kort fra Nallatech[nal] og et Virtex-II MicroBlaze Development Kit fra Memec[mem].

2.4.1.1 BenERA

BenERA er et Compact PCI (cPCI) kort beregnet for å kunne kjøre en lang rekke av forskjellige typer applikasjoner i maskinvare. Standardutgaven av kortet består av 2 stk. FPGA. Den ene av dem blir automatisk konfigurert ved boot til å ta seg av kommunikasjon med omverdenen over PCI-bussen (blir referert til som PCI FPGA), og den andre er en Xilinx Virtex 1000E (blir referert til som USER FPGA) som det er ment at brukeren av kortet skal kunne bruke til det han/hun vil. I tillegg er det mulig og utvide ressursene på kortet med enda flere FPGAer. Kortet har 4 ekstra "slots" (referert til om DIME-II Slots) som hver kan inneholde inntil 2 ekstra FPGAer (Kortet som vi hadde tilgjengelig hadde ikke noen ekstra FPGAer installert). Kortet har allerede en mengde med databusser mellom de forskjellige delene av kortet. Se figur 2.14 for en oversikt over noen av de ferdig-definerte bussene på kortet. Blant disse finner vi busser mellom PCI FPGA og USER FPGA, nabo-busser mellom de forskjellige DIME-II slottene. I tillegg har det en delt bus mellom USER FPGA og DIME-II slottene(lokal-bus), egne busser mellom USER FPGA og hver enkelt DIME-II slot(P-link), mellom USER FPGA og Status LEDs på kortets front-panel og endel GPIO-busser mellom USER FPGA og cPCI-backplane. Systemet kommer også med endel applikasjoner som gjør at



Figur 2.14: Oversikts-tegning over de viktigste forbindelsene for Inter-FPGA kommunikasjon på BenERA kortet

vi på en rask og enkel måte kan konfigurere systemet. Det følger også med et C-bibliotek som gjør det relativt enkelt å benytte systemet i et uavhengig dataprogram.

2.4.1.2 Virtex-II MicroBlaze Development Kit

Virtex-II MicroBlaze Development Kit (heretter referert til som VIIMDK) er et utviklings-kort som inneholder en Virtex-II 1000 fra Xilinx[xil]. Dette kortet er veldig godt egnet til utvikling/prototyping av forskjellige maskinvare-design. VIIMDK har i tillegg til FPGAen også en Xilinx XC18V04 ISP PROM som kan brukes til å holde konfigurasjoner som kan lastes opp til FPGAen. Kortet har også et sett med brytere, LEDs, 7seg-LEDs og 2Mx16 DDR SDRAM. I tillegg følger det med 2 utvidings moduler som kan kobles til kortet :

P160 Kommunikasjons-modul Denne modulen inneholder både 2Mx32 FLASH minne og 256Kx32 SRAM. I tillegg har den tilkoblings-porter for vanlige grensesnitt som for eksempel 10/100 Ethernet, USB, RS232, PS/2,

LCD display, I^{2}C* og SPI†.

P160 Prototypingsmodul Denne modulen har tilkoblings punkter for 108 av I/O-pinnene som er tilgjengelig for et FPGA-design i tillegg til et lite prototypingsområde som egner seg til å koble sammen eksterne komponenter og LED-lamper.

2.4.1.3 Ren Software-testing

En tredje mulighet var å bare se på en teoretisk modell og kjøre all testing/simuleringer i software. Dette kunne i prinsippet blitt gjort på 2 måter.

Enten ved å på en så nøyaktig måte en måtte ønske, simulere alt som ville skje i systemet ved å kjøre ett og ett tids-steg. Dette ville kunne generere helt nøyaktige data for det vi ønsket å teste med realistiske tilbakemeldinger.

Et problem med dette var at en da ville bli nødt til å lage et eget system som simulerte hvert aspekt av systemet. Dette var ikke ønskelig da vi i utgangspunktet ønsket å lage et fungerende system som vi kunne teste ut.

En annen mulighet som ble vurdert, og til en viss grad brukt til testing av programvare-driveren, før maskinvare systemet var på plass, var å lage en enkel modifikasjon av programvare-driveren. Disse modifikasjonen ble gjort i funksjonene som tok seg av kommunikasjonen med maskinvaren.

Forskjellen besto i at programvare-driveren ikke sender data til maskinvaren i det hele tatt, men heller simulerer den forventede oppførselen. For eksempel ved rekonfigurering ventet driveren ganske enkelt den forventede tiden. For sending av oppgaver og sjekking av oppgave-status så startet systemet opp en egen program-tråd som ventet en viss tid før den satte en kontroll-variabel som indikerte at oppgaven var ferdig behandlet og kunne hentes.

Dette var en minimal modifikasjon av systemet som enkelt lot seg reversere når maskinvaredesignet var på plass. For å oppnå denne funksjonaliteten trenger en bare å bytte ut filen "benera.c" med "bensim.c" (se tillegg C.1.1 og C.1.2).

Et av problemene med dette var at systemet ikke genererte resultater for oppgavene som ble sendt inn, men for testing av ytelse så kunne dette likevel fungert. Dessverre viste det seg at tiden som gikk med på å opprette disse ekstra trådene, i tillegg til at de selv tok opp prosessortid, gjorde at timing-resultatene ville bli unøyaktig.

Denne metoden ble derfor bare brukt for testing av systemet før maskinvaredesignet var på plass.

*I^{2}C - Inter Integrated Circuit bus

†SPI - Serial Peripheral Interface

2.4.2 Forskjellige leverandører av FPGA

Det finnes flere produsenter av FPGAer. Nedenfor presenteres noen av de mer kjente av disse i tillegg til en presentasjon av noen av deres hovedserier av FPGAer.

Xilinx Xilinx[xil] produserer FPGA-seriene Virtex og Spartan.

Virtex-serien kommer som Virtex/E/EM, Virtex-II Platform, Virtex-II Pro/ProX Platform og Virtex-4 FPGAer.

Spartan-serien kommer som Spartan, Spartan-XL, Spartan-II, Spartan-IIE, Spartan-3 og Spartan-3E FPGAer.

Virtex-4 serien er del opp i 3 applikasjons-spesifikke varianter. Virtex-4 LX er designet for høy-ytelses beregnings-intensive applikasjoner og tilbyr fra 13K til 200K logiske celler, 32 til 96 XtremeDSP^{TM} blokker, hvorav hver av disse blokkene består av 2 DSP48-blokker. Hver slik blokk er istand til å utføre vanlige operasjoner innenfor Digital Signal Processing (DSP) som addisjon/subtraksjon (48 bit) og multiplikasjon (18x18 bit) i tillegg til mange andre funksjoner. Denne serien har også fra 864Kbits 6048Kbits med BlokkRAM.

Virtex-4 SX er beregnet på DSP-applikasjoner og har fra 23K til 55K logiske celler, 2404Kbits til 5760Kbits med BlokkRAM og 128 til 512 XtremeDSP-blokker.

Virtex-4 FX er beregnet på embeddede systems og har fra 12K til 142K logiske celler, 648Kbits til 9936Kbits med BlokkRAM og 32 til 192 XtremeDSP-blokker. I tillegg så har den også 1 til 2 innebygde PowerPC-kjerner og 2 til 4 innebygde Ethernet kjerner.

Spartan 3E-serien er designet til å tilby mest mulig ressurser i forhold til pris. Den består av 2160 til 33192 logiske celler, 15Kbits til 231Kbits i Distributed RAM, 72Kbits til 648Kbits med BlockRAM. I tillegg så har den fra 4 til 36 dedikerte 18x18bits multiplikatorer.

Altera Altera[alt] produserer FPGA-seriene Cyclone/Cyclone II og Statix/Statix II hvor Cyclone II og Statix II er nyere versjoner av henholdsvis Cyclone og Statix.

Cyclone-seriene er lavkost FPGAer hvor Cyclone II består av 4.6K til 68.4K logikkelementer(LE) og har fra 26 til 250 M4K blokker med SRAM. Hver M4K blokk inneholder 4kbit i tillegg til 512 bit med paritetsdata. Den har også fra 13 til 150 18x18 bits multiplikatorer innebygget i brikken.

Statix-seriene er en serie med høy-ytelses FPGAer. Statix II bestående av 6240 til 71760 Adaptive Logic Modules (ALMs). Hver av disse tilsvarer 2.5 LE, noe som medfører at antall LEs på brikkene tilsvarer 15.6K til 179.4K.

Disse brikkene har en kombinasjon av 78 til 768 M4K-blokker med RAM og 104 til 930 M512 blokker med RAM (hver på 512 bits + paritetsbits). Istedet for å ha dedikerte multiplikatorer så har den fra 12 til 96 DSP-blokker. Hver av disse kan konfigureres til å fungere enten som 4 18x18 multiplikatorer eller som en enkel 36x36 multiplikator.

Atmel Atmel[atm] produserer FPGA-seriene AT40K og AT40KAL.

Disse består av en matrise av 1024 til 6400 kjerne-celler noe som tilsvarer 5-50K logiske porter. AT40K og AT40KAL seriene er designet for å handtere beregnings-intensive DSP og annen rask logikk. Designet deres er en symmetrisk arkitektur med kjerne-cellene jevnt fordelt i grupper på 16 sammen med blokker av SRAM (32x4).

I tillegg til disse to seriene så har Atmel en annen serie, AT6K, med FPGAer. Disse er av en eldre type, og blir ikke anbefalt å benytte i nye design.

Både AT40K og AT40KAL-seriene fra Atmel støtter partiell rekonfigurering.

2.4.3 Plattform-valg for testing

Selv om Virtex-II MicroBlaze Development Kit gav bedre muligheter for å selv definere hvilke grensesnitt som skulle brukes og ikke minst hadde mye mer plass til logikk, så falt valget på å bruke BenERA-kortet fra Nallatech. Grunnen til dette var at en da ikke trengte å implementere et grensesnitt og lage drivere for kommunikasjonen, men kunne bruke det medfølgende C-biblioteket for å kommuniserer med kommunikasjons-modulen over PCI-bussen. Det var originalt meningen å bruke partiell rekonfigurering for å bytte ut modulene. Dette ville medført at vi ikke kunne bruke BenERA-kortet siden vi da ville trenge mer kontroll over hvordan grensesnittet mot brikken skulle være. Men da vi etterhvert innså at det ikke ville bli tid til å implementere gikk vi for en reserveløsning (se kapittel 3.1.3). Denne løsningen hadde ikke de samme kravene til total kontroll over hvilke pinner på brikken som ble brukt til hva. Derfor endte vi tilslutt opp med å benytte BenERA-kortet, siden det allerede hadde et godt grensesnitt for kommunikasjon mellom programvare-driveren og maskinvaren.

Kapittel 3

Utførelse

I dette kapittelet presenterer vi det arbeidet vi har gjort for å utvikle systemet vårt. Dette kapittelet er delt inn i 3 underkapittel, hvor det første underkapittelet beskriver hvilke deler systemet vårt består av og hvordan de fungerer sammen. Det neste underkapittelet beskriver hvordan vi planla testkjøringene som vi skulle kjøre for å få svar på endel spørsmål vi stilte oss angående systemets ytelse/funksjonalitet. Tilslutt har vi ett delkapittel hvor vi ser på de resultatene vi fikk og prøver og forklare det vi ser.

3.1 Design av Systemet

Til å løse oppgaven ble det designet 2 systemer. Det ene systemet skulle implementeres i maskinvare og hadde som oppgave å ta imot og prosessere arbeidsoppgaver som den fikk fra det andre systemet.

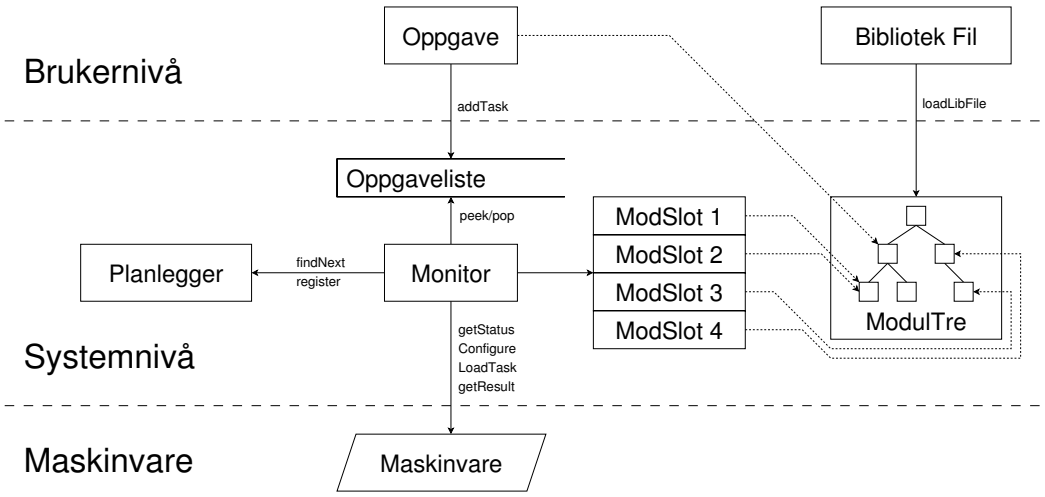
Det andre systemet skulle være i form av et programvare-bibliotek for kommunikasjon mellom brukerprogram og maskinvare systemet, i tillegg til å administrere maskinvare-systemet.

Dette del-kapittelet presenterer en oversikt over disse to delsystemene.

3.1.1 Oversikt

Som en kan se i figur 3.1 så er det komplette systemet delt inn i 3 nivåer.

På toppen er brukernivået. Dette består i all enkelhet av oppgave-objekter som brukeren oppretter, og sender inn i systemet ved hjelp av en funksjon `addTask`. I tillegg består det av biblioteks-filer som brukeren kan sende inn til systemet ved hjelp av funksjonen `loadLibFile`. Disse biblioteks-filene inneholder bitstrømmene som brukes for å rekonfigurere en av modulslotene i systemet med de forskjellige modulene.



Figur 3.1: Oversikt over de forskjellige delene av programvaren

I midten har vi system-nivået. Dette består i hovedsak av 5 deler. Vi har oppgave-listen som inneholder alle oppgavene som er blitt sendt inn i systemet og som venter på å bli kjørt. Vi har også en oppgave-planlegger. Denne har som oppgave å holde rede på hvordan bruksmønsteret til de forskjellige modulene har vært. Modul treet er et binærtre som holder på alle modulene som er blitt sendt inn i systemet av brukeren via `loadLibFile`. På dette nivået har vi også en symbolsk representasjon av maskinvaren. Dvs at vi har en struktur som brukes til å holde rede på hvilke modulslotter som er i bruk, og hvilke maskinvare-moduler som er konfigurert på hver av dem.

Alt dette blir knyttet sammen ved hjelp av en monitor-tråd som kjører i bakgrunnen og overvåker systemet. Denne har som oppgave å sjekke når det er mulig å hente inn oppgaver fra oppgavelisten, om det er mulig å sende inn uten rekonfigurering. Hvis det er nødvendig med rekonfigurering så vil den spørre planleggeren om hvilke modulslot som skal rekonfigureres.

På det nederste nivået av systemet finner vi maskinvaren. Denne består av en kommunikasjons-modul som tar seg av kommunikasjonen mellom programvaren og de 4 rekonfigurerbare modulslottene.

3.1.2 Programvare

Dette del-kapittelet ser på de forskjellige komponentene som programvare-driveren består av, og sammenhengen mellom disse.

3.1.2.1 Oppgave

Når brukeren ønsker å sende en oppgave til systemet, så oppretter han et oppgave-objekt (`task_t`) og initierer dette med å spesifisere hvilken maskinvare-modul den skal benytte, hvilke data som skal sendes til modulen(inn-buffer), hvor resultatet skal lagres(ut-buffer) og et flagg som angir om dataene i utbufferet er klare. Dette oppgave-objektet blir så sendt inn i systemet med kommandoen `void addTask(task_t *task)`

3.1.2.2 Biblioteks-filer

Biblioteks-filene til systemet inneholder binærdata for de rekonfigurerbare modulene som skal sendes inn i systemet. For hver modul så blir det lagret en bit strøm for hver modulslot. Se tillegg C.3.1 og C.3.2 for de som blir benyttet under testing av systemet.

Siden hver modul skal kunne konfigureres inn i hver av de 4 modulslottene, så er det nødvendig å ha lagret forskjellige varianter av bit-filen til hver modul.

Formatet som blir benyttet i disse filene er som følger (alle tallverdier blir lagret med mest signifikante byte først).

```
<antall moduler (integer, 4 byte)>
<Reservert (12 byte)>
<Modul 1 navn (char*, 24 byte)>
  <StartAddresse (integer, 4 byte)>
  <Antall variasjoner med tanke på forskjellige
   modulslotter (integer, 4byte)>
<Modul 2 navn (char*, 24 byte)>
  <StartAddresse (integer, 4 byte)>
  <Antall variasjoner med tanke på forskjellige
   modulslotter (integer, 4byte)>
...
<Størrelse på bitstrøm (integer, 4 byte)>
  <Bitstrøm for 1. variant av modul 1
   (char*, n bytes)>
<Størrelse på bitstrøm (integer, 4 byte)>
  <Bitstrøm for 2. variant av modul 1
   (char*, n bytes)>
<Størrelse på bitstrøm (integer, 4 byte)>
  <Bitstrøm for 3. variant av modul 1
   (char*, n bytes)>
<Størrelse på bitstrøm (integer, 4 byte)>
  <Bitstrøm for 4. variant av modul 1
   (char*, n bytes)>
<Størrelse på bitstrøm (integer, 4 byte)>
 <Bitstrøm for 1. variant av modul 2
```

```
  (char*, n bytes)>
...
```

Dataene i parentes spesifiserer formatet og størrelsen på det aktuelle datafeltet. Start-adressen som lagres sammen med modul-navnet i starten av filen spesifiserer hvor første variasjon av den aktuelle modulen starter, og de forskjellige variasjonene lagres sammen uten mellomrom.

Flere biblioteks-filer kan leses inn i systemet ved hjelp av funksjonen `loadLibFile`, eventuelt så kan en laste inn en og en modul ved hjelp av funksjonen `loadModuleFromLibFile`. Hvis en vet at en ikke lengre trenger å ha en modul i systemet så kan en bruke funksjonen `unloadModule` for å fjerne den ifra modul-treet.

3.1.2.3 Oppgaveliste

Systemet holder rede på hvilke oppgaver som er kommet inn i systemet, men som enda ikke er blitt sendt til maskinvaren ved å legge oppgave-objektene som kommer inn via `addTask` i en egen liste. Denne listen blir benyttet av monitor-tråden for å finne ut hvilke oppgaver som skal kjøres når en maskinvare-ressurs (modulslot) blir ledig.

3.1.2.4 Maskinvare-moduler

Til hver oppgave blir det tildelt en maskinvare-modul. Dette vil i praksis si en partiell rekonfigurerings bitstrøm som vil bli sendt til den rekonfigurerbare maskinvaren i forkant av oppgavestart hvis dette er nødvendig. Brukeren av systemet leser som nevnt inn disse fra en datafil med funksjonen `readLibFile`. De blir da lagret i en datastruktur (binærtre sortert etter modulnavn), og hentes derfra etterhvert som det blir bruk for dem.

3.1.2.5 Planlegger

Planleggeren sin oppgave er å holde rede på bruksmønsteret til de forskjellige modulene. Den skal på grunnlag av dette avgjøre hvilke moduler som skal forkastes når det blir nødvendig å laste inn en ny modul. Den foreløpige implementasjonen av denne baserer seg på LRU*-algoritmen, men planen er å implementere flere varianter og la bruker bestemme hvilke som skal benyttes enten som start-argument eller ved å spesifisere via driver-biblioteket†.

*LRU - Least Recently Used.

†FAS - First Available Slot, ble også implementert, men for å velge denne planleggeren, så var det nødvendig med en re-linking av systemet hvor en benyttet en annen objektfil for planleggeren (fas.o isteden for lru.o), i motsetning til de fleste andre aspekter ved systemet som kunne settes ved oppstart av systemet ved hjelp av parametre

Grensesnittet til planleggeren(scheduler) er ganske enkelt.

`void scheduler_init()` : Initierer den interne datastrukturen til planleggeren.

`int scheduler_findnext()` : Finner ut i hvilken modulslot skal den neste modulen legges. Denne modulslotten må ikke holde på med en oppgave. Hvis alle slottene er aktive (oppgaver som kjører i dem) så returneres -1.

`void scheduler_register(int slotnr)` : Denne funksjonen forteller planleggeren at en gitt slot er blitt benyttet.

Det er ingen krav om at noen andre metoder eller variabler fra planleggeren skal vare tilgjengelig fra andre deler av systemet.

3.1.2.6 Monitor

Før brukeren benytter noen av funksjonene i biblioteket må han/hun kalle en funksjon `hw_init()` som initierer endel parametre og starter opp en egen prosess-tråd(monitor-tråd) som kjører i bakgrunnen. Denne prosess-tråden overvåker statusen til den rekonfigurerbare maskinvaren og sørger for at nye oppgaver blir sendt til de aktuelle modulene når de blir tilgjengelige. Den har også ansvar for å rekonfigurerer nødvendige moduler. Siden vi ikke fikk til partiell rekonfigurering og derfor endret designet (Se kapittel 3.1.3), så ønsket vi å holde biblioteks-filene minimale. Derfor ble det kjørt noen tester for å ta tiden det tok å rekonfigurere maskinvaren, og på grunnlag av disse dataene ble det beregnet hvor lang tid det ville ta å rekonfigurere en modul[‡]. Denne informasjonen blir leste inn i systemet via filen "timing.cfg"(Se tillegg C.3.3). Selve biblioteks-filen inneholder da en minimal bit-strøm som systemet vil bruke neglisjerbar tid på å sende inn. Denne bitstrømmen inneholder akkurat nok informasjon til at kommunikasjons-modulen skjønner hvilken modul den skal sette opp. Rekonfigureringsforsinkelsen vil da istedet bli realisert ved å legge inn en forsinkelse i systemet basert på dataene fra "timing.cfg".

Monitor-tråden har også som oppgave å overvåke maskinvaren slik at den vet når en oppgave er ferdig behandlet. Den skal da hente resultatene tilbake fra maskinvaren, og sette flagget i oppgave objektet som forteller brukeren at resultatet er klart og ligger i utbufferet.

[‡]Basert på antagelse om at hver modul tar 1/4 av arealet, og at rekonfigureringstiden er proporsjonal med arealet.

3.1.2.7 Start argumenter

Som nevnt skal brukeren starte opp bakgrunns-systemet ved å kalle `hw_init()`. Denne funksjonen tar 2 argumenter. Det 2. argumentet er en peker til en liste med strengargumenter, mens den første spesifiserer hvor mange slike argumenter som listen inneholder. Argumentene kan angis i hvilken som helst rekkefølge, men ved gjensidig utelukkende argumenter vil det argumentet som angis sist bli det gjeldende. Under testing henter hovedprogrammet dem fra kommandolinjen. På denne måten slipper vi å rekompilere programmet hver gang vi ønsker å teste ut forskjellige variasjoner av systemet.

Argumentene som for øyeblikket er implementert i systemet er som følger:

-csize <n> : Spesifiserer hvor mange modulslotter som systemet inneholder (mest brukt til testing, forandrer ikke på antall modulslotter i maskinvaren). Standard er 4.

-[no_]allowduplicates : Spesifiserer hvorvidt en skal tillate at mer en en modulslot er konfigurert med en og samme modul. Standard er *-no_allowduplicates*.

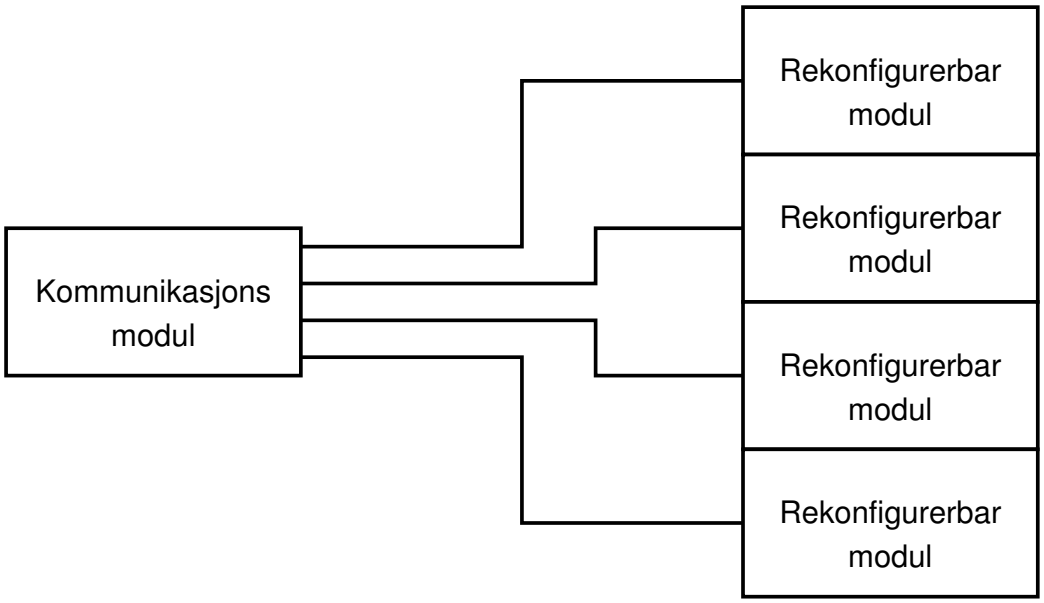
-howschedupdateallsimilar : Hvis en tillater duplikater av moduler, så er spørsmålet hvordan en skal oppdatere planleggeren når den dupliserte modulen blir brukt av en oppgave. Dette argumentet spesifiserer at alle modulslotter med den aktuelle modulen vil bli markert som nylig brukt. Standard

-howschedupdateonlythis : Dette argumentet spesifiserer at bare den modulslotten som blir brukt skal markeres som brukt.

-[no_]taskforwarding : Når en oppgave er ferdig utført på en modulslot skal monitor-tråden automatisk hente inn neste oppgave. Dette argumentet spesifiserer om den da skal lete bakover i oppgavelisten etter en oppgave som benytter samme modul, slik at den kan unngå en kostbar rekonfigurering. Standard er *-no_taskforwarding*.

-maxforwarddepth <n> : Spesifiserer hvor langt bak i oppgavelisten monitortråden skal lete etter en oppgave (gitt at *-taskforwarding* er satt), før den gir opp og henter den første oppgaven i listen. Standard er 10.

-[no_]schedregisteronforwarded : Hvis en oppgave blir startet selv om den ikke lå først i listen (ved taskforwarding), så dukker spørsmålet om modulslotten skal bli registrert som nylig brukt. Dette kan spesifiseres med dette parametere. Standard er *-no_schedregisteronforwarded*.



Figur 3.2: Forenklet oversikt over systemet

-[no_]reconfigcard : Spesifiserer om maskinvaren skal konfigureres med toplevel bitfil ved oppstart. På denne måten an en spare litt tid hvis en vet at maskinvaren allerede er konfigurert. Standard er *-reconfigcard*.

-bitfile <fname> : Spesifiserer hvilken bit-strøm som skal benyttes til innledende konfigurering (toplevel). Standard er "top.bit"

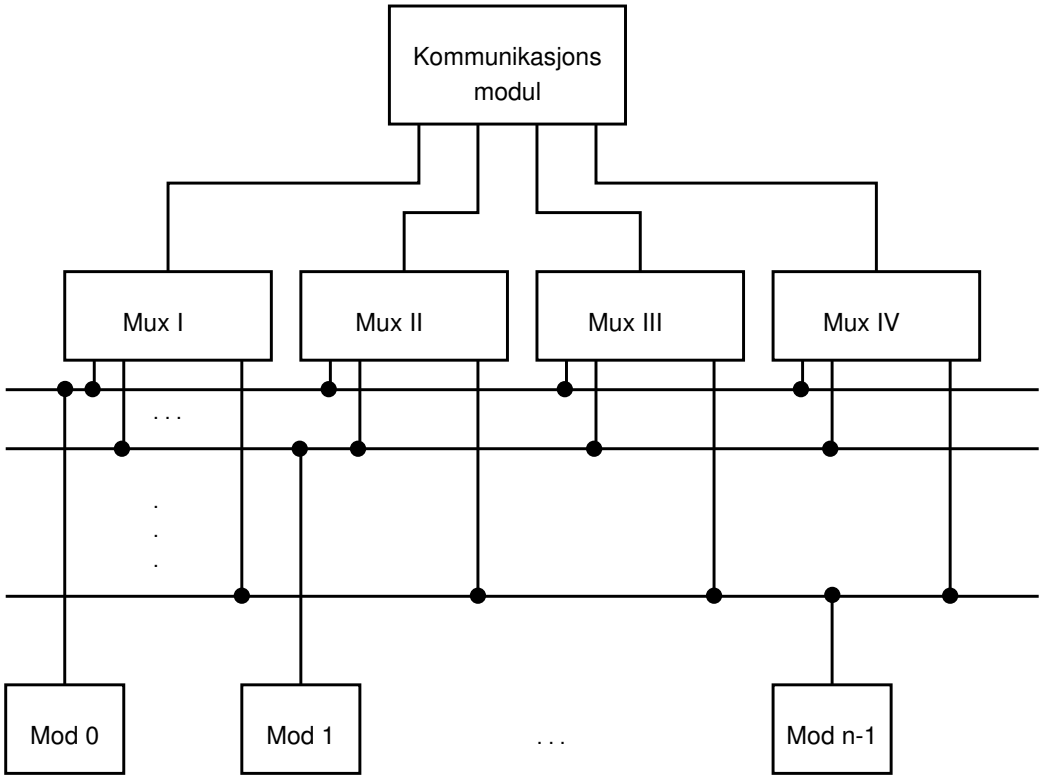
–help : printer ut en beskrivelse av hvilke argumenter som er tilgjengelig.

De fleste av disse argumentene kan settes etter oppstart ved å sette variabler som er definert i "hwglobals.h" (se tillegg C.1.7).

3.1.3 Maskinvaren

Som tidligere nevnt så består systemet også av en maskinvare-del. Denne delen består i tillegg til de rekonfigurerbare modulene også av en kommunikasjons-modul, og et internt buss-system.

Under oppstarten av prosjektet var planen å bare ha en kommunikasjonsmodul som var koblet opp mot modulslottene direkte (figur 3.2), og la programvaren ta seg av rekonfigureringen av de forskjellige modulene. Men da det ikke ble tid til dette grunnet tidspress og tekniske problemer i forbindelse med partiell rekonfigurering, valgte vi en reserveløsning. Denne løsningen gikk

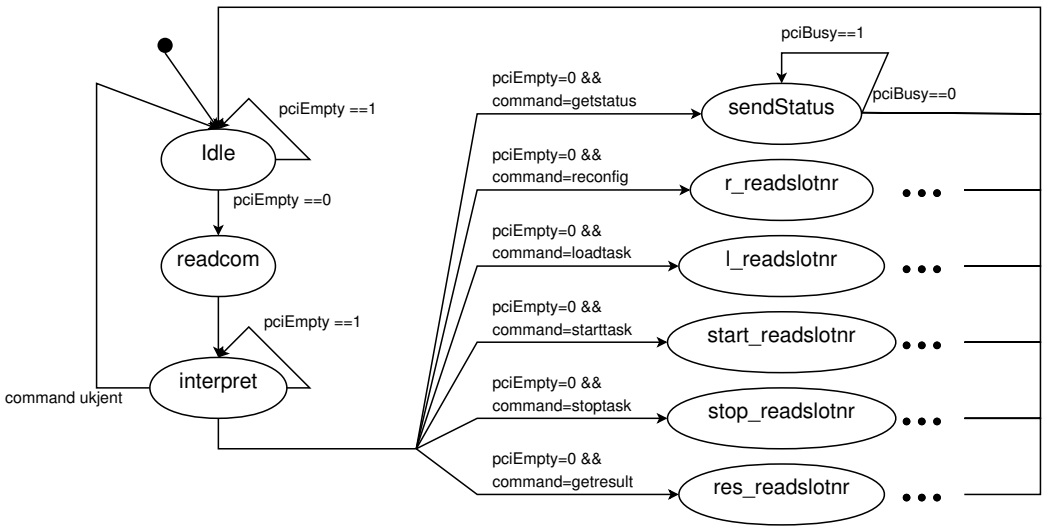


Figur 3.3: Oversikt av modifisert versjon av maskinvare med modulene ferdigkonfigurert på brikken. For beskrivelse av de forskjellige databussene, se kapittel 3.1.3.3.

ut på at istedet for å bytte mellom de forskjellige modulene ved hjelp av partiell rekonfigurering så lot vi maskinvaren være ferdig konfigurert med et sett moduler som kommunikasjons-modulen kan velge mellom ved hjelp av et sett med multipleksere/demultipleksere (se figur 3.3).

For programvaren så er de eneste forandringene at rekonfigureringen må skje gjennom kommunikasjons-modulen, og det første ordet i bitstrømmen for den modulen skal skulle konfigureres bestemmer hvilke av de n modulene som skal settes opp på en aktuell modulslot.

Dette del-kapittelet presenterer de forskjellige enhetene som maskinvaredesignet består av, og hvordan de ble koblet sammen til et komplett system. Først kommer en presentasjon av hvordan kommunikasjons-modulen er satt sammen, og hvordan de forskjellige del-tilstandsmaskinene den besto av fungerer. I tillegg kommer en beskrivelse av hvordan kommunikasjonen mellom kommunikasjons-modulen og de rekonfigurerbare modulene fungerer. For denne kommunikasjonen ble det definert et buss-system som deretter



Figur 3.4: Toppnivå for tilstands-maskin i kommunikasjons-modul.

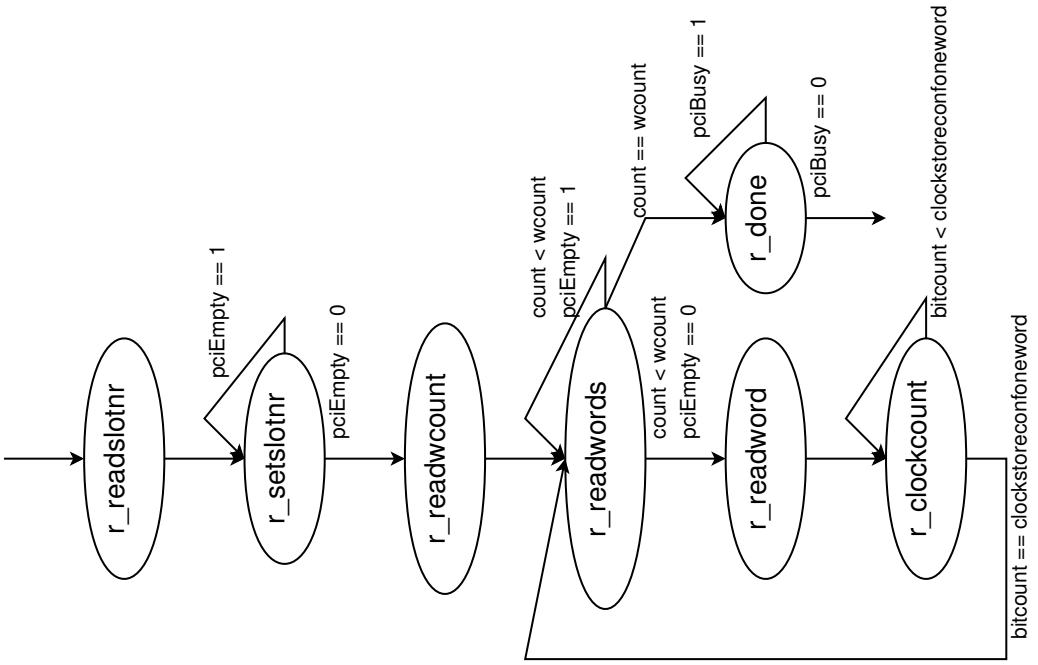
beskrives. Tilslutt i dette delkapittelet kommer en beskrivelse av protokollene for kommunikasjon mellom programvare og kommunikasjons-modul over PCI-bussen, og mellom kommunikasjons-modul og de rekonfigurerbare modulene over modulslot-bussen.

3.1.3.1 Kommunikasjons-modul

Denne modulen har ansvar for å ta seg av kommunikasjonen med programvare-driveren. Den skal også videresende data mellom programvare-driveren og de modulene som er konfigurert på den rekonfigurerbare maskinvaren. Den ble implementert som en tilstands-maskin.

Det kan skje en tilstands-endring hver klokkesykel. Som vi ser i figur 3.4 så starter denne tilstands-maskinen i en tilstand "idle". Et telle-register ved navn *count* blir tilbakestilt til 0 i denne tilstanden. Her vil den bli inntil programvarene sender data til den. Dette vil kommunikasjons-modulen merke ved at *pciEmpty* går lav. Den vil da forandre tilstand og gå til "readcom", hvor den ved neste klokkesykel vil lese av *pciData[31:0]*. Deretter vil den gå over i tilstand "interpret" hvor den tolker dataene som ble lest av. Hvis dataene som ble lest av ikke gir noen mening, så vil den automatisk gå tilbake til "idle". Avhengig av hvilke av data som er blitt sendt så vil den tilstands-maskinen reagerer på forskjellige måter:

Hvis dataene tilsvarte *command_getStatus* så vil tilstands-maskinen gå over i tilstand "sendStatus" hvor den vil holde seg til PCI-bussen blir ledig, dvs *pciBusy* går lav. Da vil den skrive statusordet til *pciData[31:0]* og gå



Figur 3.5: delmengde av tilstands-maskin, for rekonfigurering av moduler

over til tilstanden "idle" igjen.

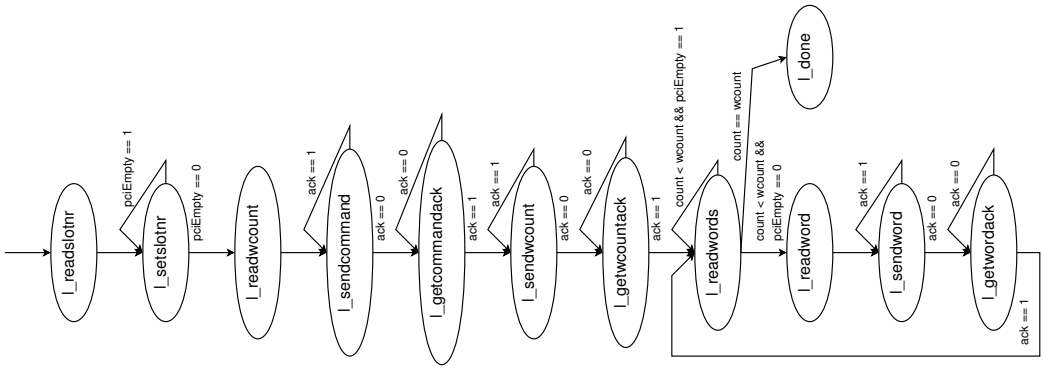
Ved *command_reconfSlot* vil tilstands-maskinen gå over i en tilstand "r_readslotnr", som er den første av et delmengder av tilstander som tar seg av rekonfigurering. Disse er beskrevet lengre ned.

Tilsvarende scenarioer vil skje for kommandoene *command_loadTask*, *command_startTask*, *command_stopTask* og *command_getRes*. Alle disse er beskrevet lengre ned. Felles for alle disse 5 tilfellene er at tilstands-maskinen ikke skifter tilstand før det er kommet mer data på PCI-bussen (*pciEmpty* er lav).

Rekonfigurering Denne delen av tilstands-maskinen tar seg av rekonfigurering av modulene. Dette skulle egentlig blitt gjort direkte ifra programvare-driveren, men grunnet designet som ble valgt, blir det gjort via kommunikasjons-modulen. Denne del-tilstands-maskinen er designet slik at den leser inn ett og ett dataord (32 bit) av bitstrømmen, og kjører i løkke et bestemt antall klokke-sykler for å simulere tiden det tar å sende dette dataordet videre.

Tilstandene i denne del maskinen (figur 3.5) er som følger:

r_readslotnr : Maskinen leser av hvilke modulslot som skal rekonfigureres



Figur 3.6: delmengde av tilstands-maskin, for lasting av oppgavedata

fra *pciData[31:0]*.

r_setslotnr : Slot-nummeret fra forrige tilstand blir lagret i et eget register og venter til det er mer data tilgjengelig på PCI-bussen

r_readwcount : Her blir antall dataord (32 bit) i bitstrømmen avlest og lagret i et eget register (*wcount*).

r_readwords : Hvis telleregisteret *count* tilsvarer *wcount* så vil neste tilstand bli "r_done". Hvis ikke så vil systemet vente i denne tilstanden til det er data tilgjengelig på bussen.

r_readword : Her vil systemet lese av et dataord fra bussen, og initierer et telle-register (*bitcount*) til 0.

r_clockcount : Dette er en tilstand hvor systemet vil holde seg i et gitt klokke-sykler. Dette er for å simulere tiden det ville tatt og sendt et dataord inn til den rekonfigurerbare maskinvaren ved partiell rekonfigurering. Hvis *count* == 1 så vil den aktuelle modulslotten få tildelt en modul utfra hvilke data som ble lest av PCI-bussen sist. Dvs at første dataordet i bitstrømmen styrer hvilken modul som skal benyttes.

r_done : Når systemet kommer i denne tilstanden så er rekonfigureringen av modulslotten ferdig. Her vil systemet stå inntil det er mulig å sende data tilbake over PCI-bussen. Når dette er tilfelle så vil programvaren bli varslet ved at systemet sender et spesifikt ord tilbake. Systemet går deretter tilbake til "idle"

Lasting av oppgavedata Denne delen av tilstands-maskinen (figur 3.6) tar seg av lasting av oppgavedataene. Dette skjer ved at den leser inn hvilke modulslot det er snakk om, hvor mye data og selve dataene. Antall dataord, og selve dataene bli sendt videre til riktig modul via et eget grensesnitt

Tilstandene i denne delen av tilstands-maskinen er som følger:

l_readslotnr : Nummeret til hvilken slot som inneholder modulen som skal kjøres blir lest inn fra bussen.

l_setslotnr : Slot-nummeret blir lagret i eget register, og systemet venter med å gå videre til det er mer data tilgjengelig på bussen.

l_readwcount : Antall dataord i oppgave dataene blir lest inn og lagret i eget register (*wcount*).

l_sendcommand : Systemet sjekker at modulen er klar til å motta data (*ack == 0*) og sender en kommando til riktig modulslot for å indikere at data skal lastes. Dette gjøres ved å skrive til et register som automatisk kopieres over til riktig modulslot-bus (utfra hvilke verdi som ligger i *slotnr*).

l_getcommandack : Systemet venter til modulen har kvittert for mottatt kommando (*ack == 1*).

l_sendwcount : Systemet sender beskjed om antall dataord til modul.

l_getwcountack : Systemet venter til modulen har kvittert for antall dataord.

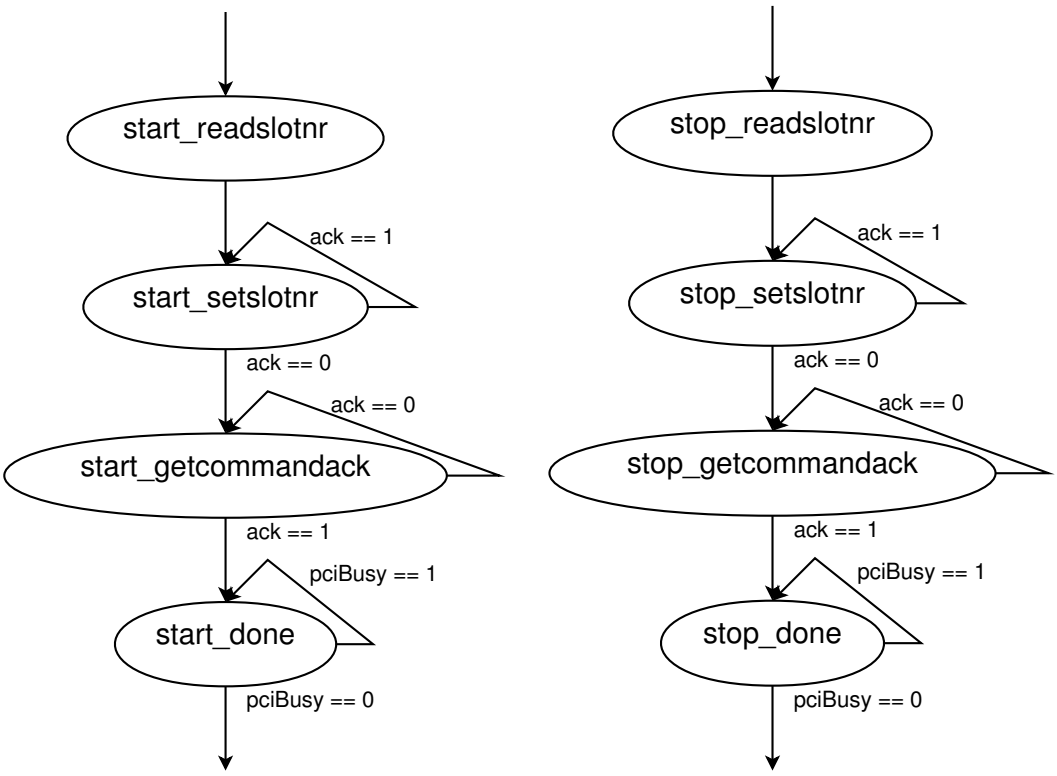
l_readwords : Hvis *count == wcount* så har systemet send all data over til modulen og den går til tilstand "l_done". Hvis ikke så venter den til det er data tilgjengelig på PCI-bussen før den går videre.

l_readword : Systemet leser inn et dataord fra PCI-bussen og sender det til til modulen. *count* blir oppdatert med 1.

l_sendword : Systemet setter et kontrolsignal som forteller modulen at det er data tilgjengelig på modulslot-bussen.

l_getwordack : Systemet venter til modulen har kvittert for dataordet.

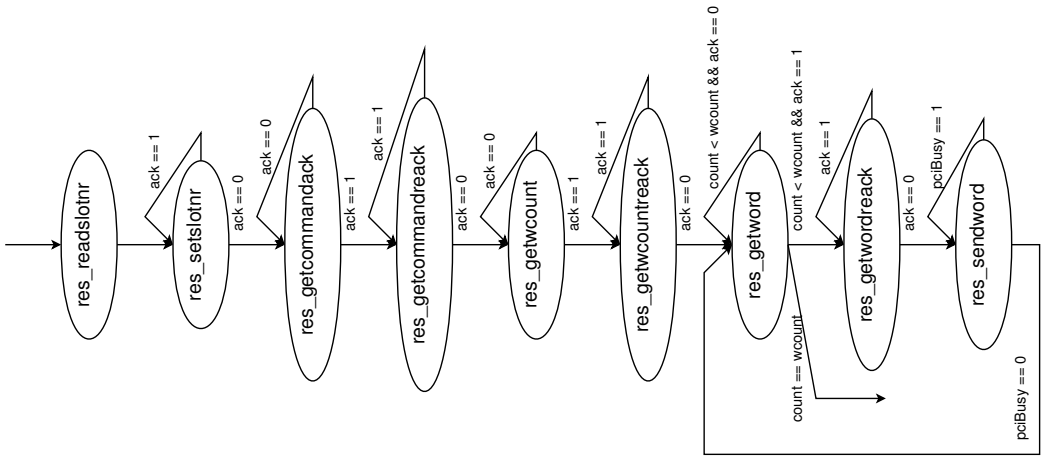
l_done : Lasting av data er ferdig. Systemet blir stående her til det er mulig å sende et kvitterings ord tilbake til programvaren via PCI-bussen.



(a) Start av Moduler

(b) Stop av Moduler

Figur 3.7: delmengder av tilstands-maskin, for starting og stopping av kjørende moduler



Figur 3.8: delmengde av tilstands-maskin, for henting av resultatdata

Start/Stopp av moduler Disse delene av systemet (figur 3.7) muliggjør starting og stopping av kjørende moduler. Disse blir ikke benyttet i systemet da modulene automatisk starter opp når de har fått lastet inn tilstrekkelig med data, og stopper når de er ferdige. Men vi har likevel valgt og inkludere dem da det i framtiden kan være ønskelig med en slik funksjonalitet.

Måten disse to del-tilstands-maskinene fungerer på er i praksis identisk, med unntak av hvilke kommando de sender til modulen. Derfor beskrives bare del-tilstands-maskinen for starting av moduler.

start_readslotnr : Slot-nummeret til den modulen vi ønsker å starte blir lest inn fra PCI-bussen.

start_setSlotnr : Slot-nummeret blir lagret i et eget register for å styre hvilke modulslot-bus vi skal kommunisere over. Kommandoen for starting av modul blir sendt over modulslot-bussen.

start_getcommandack : Systemet venter til modulen ha kvittert for å ha mottatt kommandoen.

start_done : Systemet venter til det er mulig å skrive til PCI-bussen, og gir så beskjed til programvaren om at modulen er startet.

Henting av resultatdata Denne siste delen av tilstands-maskinen inneholder tilstandene som kontrollerer henting av resultatdataene fra moduler som er ferdig med oppgavedataene som de er blitt tilsendt.

res_readslotnr : Slot-nummeret til den modulen vi ønsker å hente data fra bli lest inn fra PCI-bussen.

res_setslotnr : Slot-nummeret blir lagret i et eget register. Kommando for henting av data blir sendt over modulslot-bussen.

res_getcommandack : Systemet venter til modulen har kvittert for å ha mottatt data.

res_getcommandreack : Siden datastrømmen nå skal reverseres (fra modul til kommunikasjons-modul) så venter systemet på at modulen viser at den er klar til å sende. Deretter sier systemet ifra om at det er klar til å motta.

res_getwcount : Systemet venter til modulen har sendt antall dataord i resultatdataene, og kvitterer for dem.

res_getwcountreack : Systemet venter til modulen indikerer at den har mottatt kvitteringen, og gjør seg klar til å motta mer data.

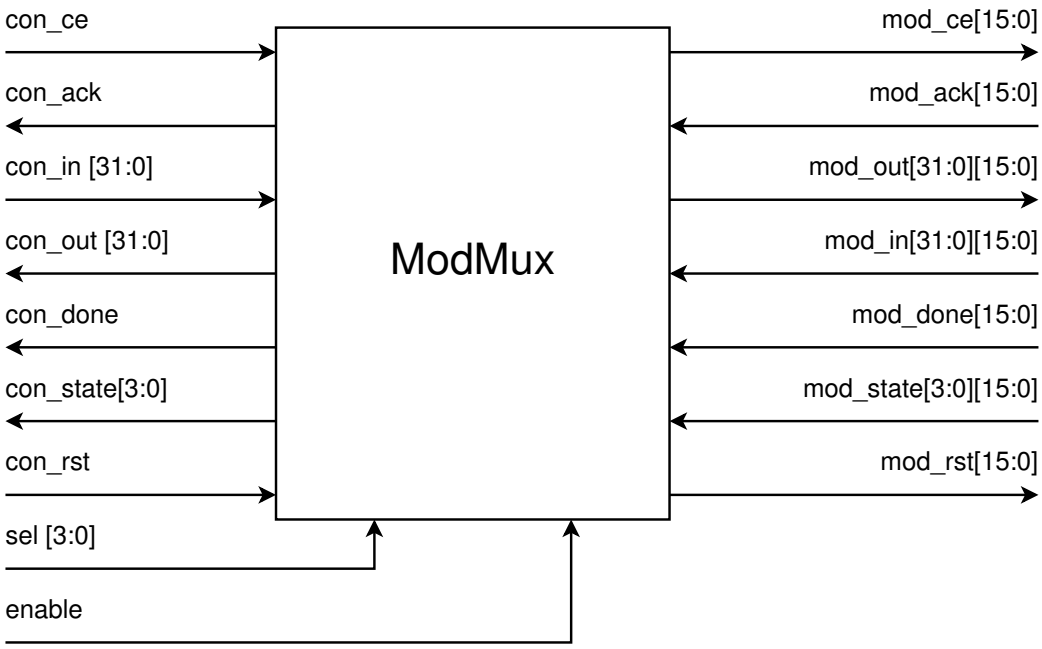
res_getword : Hvis *count* == *wcount* så er alle data mottatt fra modulen og sendt tilbake til programvare-driveren over PCI-bussen. Hvis ikke så venter systemet i denne tilstanden til den har mottatt et dataord og sender en kvittering tilbake til modulen for det.

res_getwordreack : Systemet venter til modulen har sett kvitteringen og gjør seg klar til å motta mer data.

res_sendword : Før systemet mottar mer data fra modulen, så venter den til det er mulig å skrive til PCI-bussen. Når det skjer så sender den dataordet den sist fikk fra modulen tilbake til programvaren over PCI-bussen.

3.1.3.2 Modulslot ruting

Som nevnt ovenfor så blir data sendt til korrekt modul ved å skrive til spesifikke registre *modout* og *modin*, som kopierer dataene over til korrekt modulslot-buss. Disse registrene er av type *modportin* og *modportout* (se tillegg C.2.2). I det første registeret blir data som kommer fra korrekt modulslot-buss lagret. Det andre blir benyttet for å sende data til korrekt modulslot-buss. Registeret som styrer denne funksjonaliteten er *slotnr*, og det første dataordet som blir lest inn fra PCI-bussen for alle operasjoner som krever kommunikasjon med modulene. Selve modulslot-bussene er implementert som fire par av *modportin* og *modportout* register. Maskinvaren kjører en



Figur 3.9: Oversikt over inn- og ut-signaler til en modulslot multiplekser/demultiplekser.

sekvensiell prosess[§] som automatisk kopierer innholdet i *modout* over til korrekt modulslot-buss (*modportout*-del) hver gang det er en endring i enten *slotnr* eller i *modout*. I tillegg er det en parallell prosess som sørger for at *modin* alltid inneholder inndata fra riktig modulslot-buss (*modportin*-del).

3.1.3.3 Modul slot buss system

En modulslot-buss består av 7 signaler som skal sendes til/fra aktuell modul (Med aktuell modul menes den modulen som bussen er satt opp til å kommunisere med ved hjelp av *sel[3:0]*):

ce : Signal som styres av kommunikasjons-modulen. Hvis dataretningen er til modul så forteller dette signalet aktuell modul at det er data tilgjengelig for lesing ($ce == 1$).

ack : Signal som styres av aktuell modul. Hvis dataretningen er til modul så forteller dette signalet enten at modulen er klar til å motta data ($ack == 0$) eller at data er blitt mottatt ($ack == 1$). Hvis dataretningen

[§]med sekvensiell prosess så menes det her funksjonalitet som bare aktiveres ved spesifikke hendelser (f.eks. at et klokkesignal forandrer seg), i motsetning til en parallell prosess som alltid er aktiv.

derimot går fra modul til kommunikasjons-modul så bytter *ce* og *ack* ganske enkelt funksjon.

input : Signal-bus (32 bit) som inneholder data som kommer fra aktuell modul.

output : Signal-bus (32 bit) som inneholder data som skal til aktuell modul.

rst : Signal fra kommunikasjon-modul til modul som brukes for å initiere modulen.

done : Signal fra modul til kommunikasjons-modul som forteller om modulen er ferdig med oppgaven den er tildelt.

state : Signalbuss (4 bit) som inneholder informasjon om hvilken tilstand modulen er i (benyttes til testing av systemet).

Som nevnt så inneholder designet 16 ferdig-konfigurerte moduler som modulslot-bussene kan kobles opp mot. Dette er implementert ved hjelp av multipleksere/demultipleksere som er samlet i en struktur som kalles *Modmux* (se figur 3.9. Disse strukturene blir styrt av 2 kontrollsignaler:

sel : Signal-bus (4 bit) som forteller modmuxen hvilke modul den skal kommunisere med.

enable : Signal som benyttes for å aktivere modmuxen. Når dette signalet er lavt så vil alle ut-signal settes i nøytral[¶].

Måten denne fungerer på er ganske lik måten rutingen i kommunikasjons-modulen foregår på. Signaler som går fra kommunikasjons-modulen til en modul (bestemt av *sel*) blir kopiert over til en av 16 tilsvarende ut-signaler. De andre 15 tilhørende signalene blir satt i nøytral. For eksempel hvis *con_ce* = 1 og *sel* = "0110"(6) så vil *mod_ce* som går til modul 6 bli satt lik 1. Samtidig så vil *mod_ce* til de andre modulene settes i nøytral.

Signaler som går fra en modul til kommunikasjons-modulen blir hentet fra en av 16 signaler (et fra hver modul) og kopiert over til signalet som går til kommunikasjons-modulen. For eksempel hvis *sel* = "0101"(5) så vil *con_state* settes lik *mod_state[5]*.

[¶]Et signal som er satt i nøytral blir ikke presset til verken 0 eller 1. Dette gjør at det er mulig for andre deler av systemet til å skrive disse signalene. I dette tilfelle de andre mod-/demuxene.

3.1.3.4 Kommunikasjon med programvare

Kommunikasjons-modulen kommuniserer med programvare-driveren over PCI-bussen. Dette grensesnittet var ferdig definert på forhånd og fungerte ved at PCI-FPGA på BenERA-kortet automatisk leste av PCI-bussen og gjorde dataene tilgjengelig via et ferdig-definert grensesnitt mellom PCI-FPGA og USER-FPGA. For sending av data så leser PCI-FPGAen av dette grensesnittet og sendte dataene automatisk over PCI-bussen (se figur 2.14).

Dette grensesnittet besto blant annet av disse signalene[‖]:

pciData[31:0] Her blir dataene som enten skal sendes til kortet, eller leses av kortet plassert. Dette er en toveis bus, dvs at både PCI FPGA og USER FPGA kan skrive data til den.

pciEmpty Dette er et signal som blir satt av PCI FPGA for å indikere at det ikke er noen data tilgjengelig for lesing. (aktivt høyt).

pciBusy Settes av PCI FPGA for å indikere at det ikke er plass til å skrive noe mer data til PCI-bussen (dvs bufferet på PCI FPGA er fullt). Når dette signalet går høyt er det mulig å skrive inntil 2 dataord til før data går tapt.

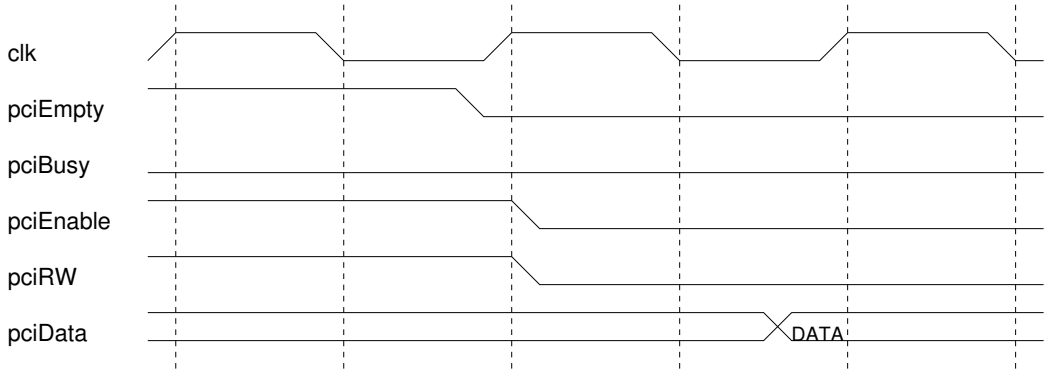
pciRW Settes av kommunikasjons-modulen for å indikere hvilken retning dataene skal gå. Hvis *pciRW* er lav så vil PCI FPGA skrive til *pciData* når *pciEnable* går lav. Hvis *pciRW* er høy, så vil PCI FPGA lese av *pciData* når *pciEnable* går lav.

pciEnable Settes av kommunikasjons-modulen for å fortelle PCI FPGA at det skal være aktivitet på bussen. (aktivt lav.) Hvis *pciEnable* går lav og *pciRW* også er lav (Lesing) så vil PCI FPGA begynne å skrive data til *pciData*, slik at de er tilgjengelig på neste klokke-sykel. Hvis *pciEnable* går lav og *pciRW* er høy (skriving) så vil PCI FPGA lese av dataene som samtidig blir skrevet til *pciData* av kommunikasjons-modulen.

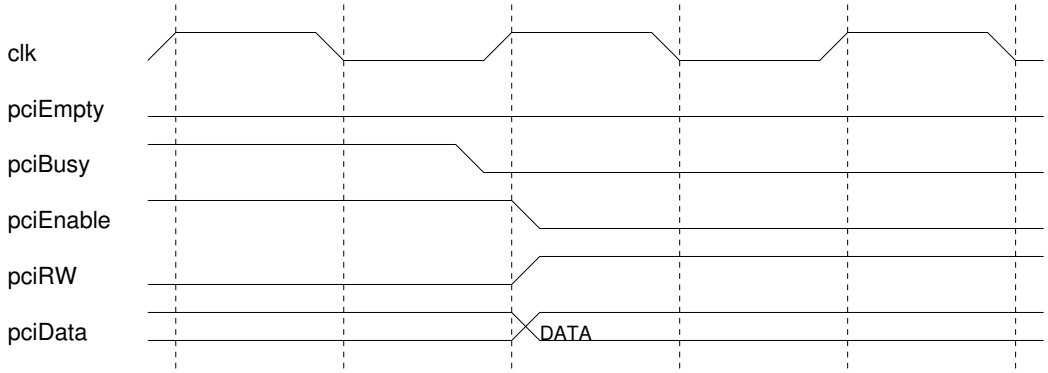
Se figur 3.10 for hvordan signal-mønstrene for disse signalene er ved skriving til og fra PCI-bussen via dette grensesnittet.

Protokollen mellom programvaren og maskinvaren foregikk ved at programvare-driveren fungerte som master, og initierte alle dataoverføringer. Dette ved å først sende et kommandoord, sammen med forskjellige argumenter som fortalte maskinvaren om hvilke data som ville komme og hvor mye av

[‖]Bare signalene som ble benyttet av systemet er nevnt her.



(a) Lesing av PCI-buss



(b) Skriving til PCI-buss

Figur 3.10: Signal-mønster for lesing/skriving over PCI-bussen

dem. Maskinvaren vil da etter å ha mottatt en viss mengde data begynne å sende data tilbake til programvare-driveren. Avhengig av hvilke kommando-ord som ble sendt over så vil denne dataflyten variere med tanke på innhold og mengde.

Tilgjengelige kommandoer er som følger

0 : getstatus - Forespørsel om statusord. Ingen argumenter. Maskinvaren svarer med å sende statusordet.

1 : reconfig - Gir maskinvaren beskjed om at en modulslot skal [re-]konfigureres. Argumenter er hvilken modulslot det er snakk om, antall ord i konfigurerings bit-strømmen og selve bitstrømmen. Maskinvaren svarer med å sende $0x0$ tilbake når rekonfigureringen er ferdig.

2 : load - Gir maskinvaren beskjed om at en oppgave skal lastes opp i en av modulslottene. Modulslotten må være ferdig konfigurert. Argumenter her er hvilken modulslot som skal benyttes og størrelsen på oppgave-dataene (i antall dataord), etterfulgt av selve dataene. Maskinvaren svarer med å sende $0xF0F0F0F0$ når dataene er ferdig lastet.

3 : getresult - Forespørsel om resultat etter endt oppgave-jobb. Argument her er hvilke modulslot resultatene skal hentes fra. Det antas at program-vare-driver vet hvor mye som skal hentes, og maskinvaren svarer med å sende resultatene tilbake direkte uten noe mer metadata.

4 : starttask - Forespørsel om å starte en oppgave som er blitt stoppet. Argument her er hvilke modulslot som skal startes, og maskinvaren svarer med $0x33333333$

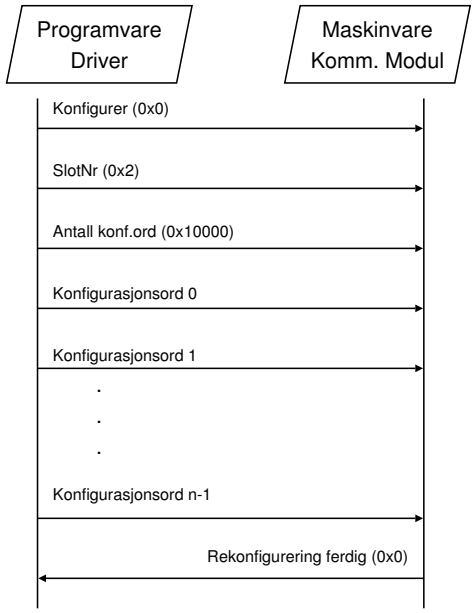
5 : stoptask - Forespørsel om å temporært stoppen en oppgave under utførelse. Argument her er hvilke modulslot som skal stoppes, og maskinvaren svarer med $0xCCCCCCCC$

Verdiene som blir returnert ved disse kommandoene er valgt fordi de er lette å kjenne igjen i bit-form, og fordi de opptar hele dataordet med tanke på testing og simulering.

Se figur 3.11 for eksempel på dataflyt ved rekonfigurasjon av slot nummer 2 med en 256KByte bitstream, eller kapittel 3.1.3.1 for mer detaljer om hvordan dette skjer.

3.1.3.5 Kommunikasjon med modulene

I tillegg til kommunikasjonen mellom programvare driveren og kommunikasjons-modulen så vil det også foregå endel kommunikasjon mellom kommunikasjons-modulen og de forskjellige rekonfigurerbare modulene. Vi valgte

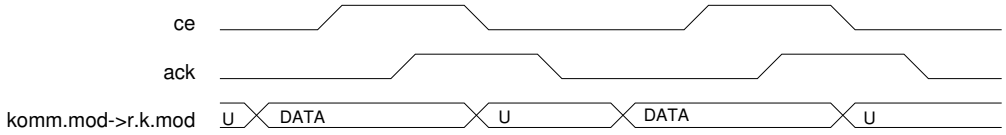


Figur 3.11: Dataflyt mellom programvare driver og kommunikasjons modulen for rekonfigurering.

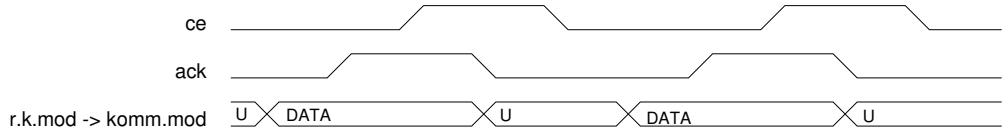
å implementere en dedikert buss til hver av modulslottene. Disse bussene ble definert ved hjelp av to kontrollsignaler (*ce* og *ack*) og et 32-bits datasignal. På grunn av designet som ble valgt ble det bestemt å splitte datasignalene slik at en hadde egne opp og ned linjer.

Kontroll-signalene *ce* og *ack* blir styr av henholdsvis kommunikasjons modulen og de rekonfigurerbare modulene. Når kommunikasjons-modulen ønsker å sende data til en modul sjekker den først at *ack*-signalet fra modulen er 0. Deretter skriver den den ønskede verdien til ut-datalinjene for deretter å sette *ce* til 1. Modulen vil da oppdage dette og lese av verdien fra dens inn-datalinjer, for så å sette *ack* til 1. Når kommunikasjons-modulen ser dette så vet den at dataene er mottatt, og setter *ce* til 0 igjen, og modulen indikerer at den er klar til å motta flere data på nytt ved å selv sette *ack* tilbake til 0. På samme måte kan modulen sende data til kommunikasjons-modulen ved å vente til *ce* er lav, for så å skrive til sine ut-datalinjer og sette *ack* høy. Dette kvitterer kommunikasjons-modulen for ved å sette *ce* til 1, noe som igjen gjør at modulen setter *ack* lav, etterfulgt av at kommunikasjons-modulen setter *ce* lav. Se figur 3.12 for eksempler av sending av to dataord hver vei.

Protokollen mellom kommunikasjons-modulen og de rekonfigurerbare modulene ligner veldig på protokollen mellom programvare-driveren og kommunikasjons-modulen. Kommunikasjons-modulen fungerer som buss-master og starter aller overføringer med ett kommandoord etterfulgt av enten argumenter, data



(a) Dataoverføring fra kommunikasjons-modul til rekonfigurerbar modul



(b) Dataoverføring fra rekonfigurerbar modul til kommunikasjons-modul

Figur 3.12: Bus-grensesnitt mellom kommunikasjons-modul og de rekonfigurerbare modulene

og eller svar fra modulen.

Følgende kommandoer blir benyttet:

0 : start - Starter en temporært stoppet modul. Ingen argumenter eller svar fra modul forventes.

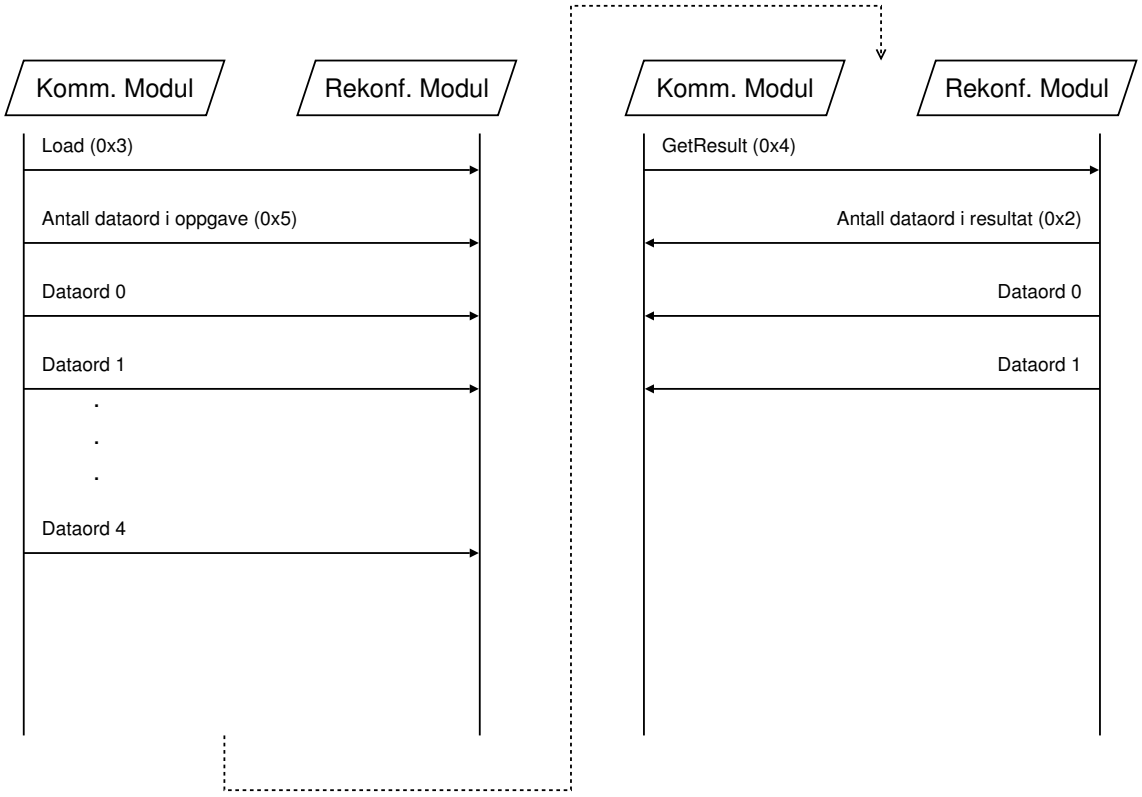
1 : stopp - Stopper en modul temporært. Ingen argumenter eller svar fra modul forventes.

2 : reservert.

3 : load - Laster en oppgave opp til modulen. Som argument sendes antall dataord, etterfulgt av selve dataene. Ingen svar fra modulen forventes.

4 : getresult - Henter resultatene fra modulen. Ingen argumenter blir sendt. Modulen svarer med å sende antall dataord som resultatet består av, etterfulgt av selve resultatdataene.

Se figur 3.13 for eksempel av sending av en oppgave bestående av 5 dataord, og umiddelbar henting av resultatet tilbake (2 dataord).



Figur 3.13: Eksempel-kommunikasjon for lasting av en 5-dataords oppgave og henting av 2-ords svar.

3.2 Testing

Ved å teste systemet ønsket vi å se om vi fikk forbedret ytelsen i et testmodell ved å benytte systemet vi hadde designet. Vi ønsket også å få svar på følgende spørsmål :

- Hvis enkelte moduler i systemet brukes veldig ofte kan det være fristende å ha flere instanser av dem konfigurert på flere modulslotter samtidig. Dette kaller vi modul-duplisering. Hva er effekten av å tillate dette, og i så tilfelle skal alle modulslottene med samme modul-type merkes som brukt når en av dem brukes, eller bare den som bli aksessert?
- Hva var effekten av å la systemet lete igjennom oppgave-køen etter at en oppgave var avsluttet for å finne andre oppgaver som benytter samme modul. Og i så tilfelle, hvor langt bak i køen skal det lete for å på denne måten unngå unødvendige rekonfigureringer (oppgaveframskyving)?
- Hvor effektiv vil en planlegger som benytter LRU-algoritmen være i forhold til en planlegger som alltid velger den første ledige modulslotten (FAS)?
- Hvordan er forskjellen i ytelse for systemet vårt i forhold til hvis vi kjører tilsvarende oppgave i programvare?

I dette del-kapittelet presenterer vi de forskjellige testene vi gjorde med tanke på testdata og måten vi analyserte resultatene på.

3.2.1 Generering av simuleringsdata

Simuleringsdataene våre ble delt i to deler. Den ene delen besto av en fil som spesifiserte hvilke oppgaver som skulle kjøres, og i hvilke rekkefølge, mens den andre delen besto av en bibliotek-fil som inneholdt bitstrømmene som skulle benyttes til å rekonfigurere den rekonfigurerbare maskinvaren for de forskjellige modulene.

3.2.1.1 Generering av oppgave-fil

For å realisere dette ble det laget et lite ekstra-program(gentasks, se tillegg C.1.4) som genererte en liste over tilfeldige oppgaver.

Ved å kalle dette programmet uten argumenter så får en følgende utskrift

```
Syntax : ./gentasks <numberofmodules>
  <mod0name> ... <modn-1name> <mod0maxargs> .. <modn-1maxargs>
  <mod0minargs> .. <modn-1minargs> <mod0res> .. <modn-1res>
  <mod0distr> .. <modn-1distr> <numberoftasks>
```

For at programmet skal gjøre noe som helst er det derfor nødvendig å spesifisere endel parametre. Disse parametrene er som følger

- <numberofmodules> Hvor mange forskjellige moduler ønsker en å benytte.
- <mod0name> .. <modn-1name> Navnet på alle modulene som en ønsker å benytte.
- <mod0maxargs> .. <modn-1maxargs> Maksimalt antall argument-ord en ønsker å sende inn til hver av modulene en ønsker å benytte.
- <mod0minargs> .. <modn-1minargs> Minimum antall argument-ord en ønsker å sende inn til hver av modulene en ønsker å benytte.
- <mod0res> .. <modn-1res> Antall resultat-ord som skal sendes tilbake fra hver av modulene en ønsker å benyttet.
- <mod0distr> .. <modn-1distr> Hvor stor del av oppgavene skal bruke hver av modulene.
- <numberoftasks> Hvor mange oppgaver ønske en å generere.

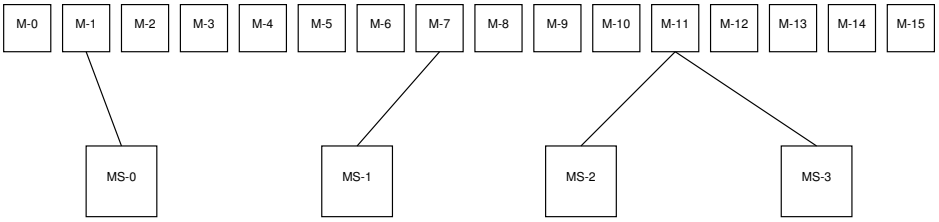
For eksempel kan en si at en ønsker å generere 100 oppgaver med 25% av modul MOD1, 25% av modul MOD2 og 50% av modul MOD3. Hvor MOD1 tar maksimalt 10 og minimum 2 ord, MOD2 maksimalt 4 og minimum 1 ord og MOD3 tar nøyaktig 3 ord som argument. Som resultat tar MOD1 4 ord, MOD2 1 ord og MOD3 5 ord. For å oppnå en slik fordeling kunne en da benyttet følgende kommando :

```
./gentasks  3  MOD1 MOD2 MOD3  10 4 3  2 1 3  4 1 5  25 25 50  100
```

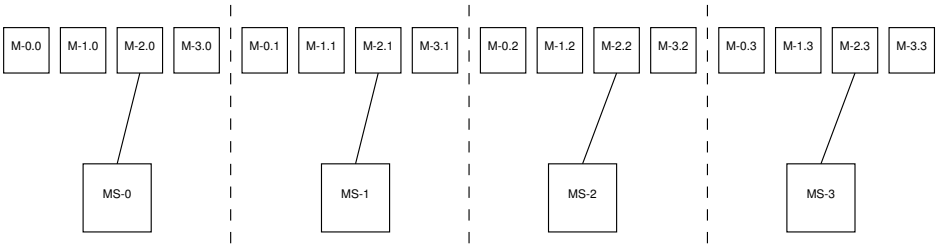
Dette programmet genererer da en fil som sendes til standard output (til skjermen) på følgende format

```
antalloppgaver
modulnavn antallargumentord antallresultatord
modulnavn antallargumentord antallresultatord
...
modulnavn antallargumentord antallresultatord
```

Dette ble kopiert over til en tekstfil som hovedprogrammet vårt leser inn ved oppstart, og blir referert til som oppgavesettet.



(a) To modulslotter er konfigurert for den samme fysiske modul ved bruk at 16 unike moduler. MS-# er de forskjellige modulslottene, mens M-# er de forskjellige modulene.



(b) Fordeling av de fysiske modulene i grupper for å unngå konflikter mellom modulslotter ved 4 unike moduler. M-#.# er de forskjellige modulene. F.eks er M-2.3 modul nummer 2, og kan bare tilkobles MS-3.

Figur 3.14: Problem i forbindelse med modul-duplisering og foreslått løsning

3.2.1.2 4- eller 16-modulers oppgavesett

På grunn av måten maskinvare-modulene var implementert på hadde vi totalt 16 forskjellige moduler. Et problem dukket da opp hvis en valgte å tillate modul-duplisering. Dette kunne medføre at to (eller flere) oppgaver benyttet den samme fysiske modulen og derfor kunne rotet til hverandres oppgavedata (Se figur 3.14(a)). Dette medførte at vi ikke fikk testet modul-duplisering med 16 moduler. Men for å få testet ut modul-duplisering, så vi løste dette ved å dele modulene opp i grupper på 4, og istedet bare operere med 4 moduler. Hver slik modul-gruppe ville da inneholde en modul for hver modulslot (Se figur 3.14(b))*.

På denne måten så ville to oppgaver med samme modul som kjørte samtidig aldri benytte den samme fysiske modulen, bare samme modul-gruppe.

Det ble derfor generert 2 sett med simuleringsdata. Et med 4 moduler beregnet på testing når duplisering av moduler var tillatt, og et sett med 16 moduler for testing hvor duplisering av moduler ikke tillates. Settet med 4 moduler ble også testet ut uten modul-duplisering for sammenligning. Det ble derfor også skrevet 2 modul-filer (Se kapittel 3.1.2.2 og tillegg C.3.1 og C.3.2). Det ble benyttet total 2 forskjellige modul-typer (se kapittel 3.3.1 for beskrivelse av disse) : SUM og EXP. Vi valgte å benytte like mange av hver av disse typene. Felles for disse oppgave-filene var at fordelingen av oppgavetype var eksponensiel. Det vil si at det var 1 andel av SUM0 og EXP0, 2 andeler av SUM1 og EXP1, 4 andeler av SUM2 og EXP2 og så videre..

Se tillegg C.1.24 for skriptet som genererte disse 2 oppgavesettene.

3.2.2 Kjøring av testdata

Disse filene sendes da som argument til hovedprogrammet(Se tillegg C.1.16) sammen med spesifikasjon for hvilke regler som gjelder for kjøringen. Det ble gjennomførte totalt 92 kjøringer hvor vi benyttet maskinvaredesignet.

For kjøringer hvor vi lot planleggeren lete igjennom oppgavelisten etter oppgaver som kunne flyttes fram for umiddelbar kjøring(taskforwarding), testet vi med å la den se 1,2,4,8,16,32,64,128,256,512 og 1024 ($2^n, n \in \{0-10\}$) plasser bakover. For hver av disse testet vi også ut både med og uten å markere modulslotten som brukt ved oppgaveframskyving. I tillegg kjørte vi en kjøring uten denne oppgave-framskyvingen.

Disse 23 tilfellene ble deretter kombinert med 4 forskjellige tilfeller av

*Figur 3.14(b) er litt misvisende da en kan få inntrykk av at M-0.0, M-1.0, M-2.0 og M-3.0 er i samme gruppe. Gruppenummeret er det første at de to nummerene, slik at for eksempel M-2.0, M-2.1, M-2.2 og M-2.3 er i samme gruppe, og representerer en enkel modul.

hvordan systemet skulle handtere duplisering av moduler. Disse 4 tilfellene var som følger

- Duplisering av moduler tillatt, registrer alle modulslotter som benytter samme modul som brukt når en av dem blir benyttet. (Benytter oppgavesettet og modul-filen med 4 moduler)
- Duplisering av moduler tillatt, men bare registrer den modulslotten som blir benyttet som brukt. (Benytter oppgavesettet med 4 moduler)
- Duplisering av moduler ikke tillatt, bruk oppgavesettet med 4 moduler.
- Duplisering av moduler ikke tillatt, bruk oppgavesettet med 16 moduler.

For å forenkle denne prosessen ble det laget et shellskript som sørget for at programmet fikk riktige argumenter (Se tillegg C.1.25).

Hovedprogrammet leser inn en og en linje fra oppgave-filen og oppretter et oppgave-objekt med randomiserte inndata og tildeler plass til å lagre resultatdata etter spesifikasjonene angitt i oppgave-filen.

Når alle oppgavene er lest inn fra filen, så går systemet igjennom listen av nyopprettede oppgave-objekter og sender disse fortløpende inn i systemet ved å kalle `addTask(task_t *task)` for hvert av dem. Da vil programvare-driveren automatisk begynne å sende dem inn til systemet etterhvert som ressursene blir tilgjengelig. I vår modell har vi altså at planleggeren i praksis kjenner til alle oppgavene som skal kjøres fra starten av. Oppgave-framskyvingsøkene kan derfor være litt urealistiske ved høy rekkevidde for framskyvingsøk.

Programvare-driveren genererer automatisk en loggfil med følgende hendelser:

- *AddTask* Forteller at det er kommet en oppgave inn i systemet. Vil bli kjørt automatisk når det blir dens tur.
- *StartTask* En oppgave er blitt sendt inn til korrekt maskinvare-modul, og er startet opp.
- *EndTask* En oppgave er ferdig behandlet og resultat er blitt hentet tilbake og lagret i oppgave-objektet.
- *StartReconf* En modulslot må rekonfigureres før oppgaven kan starte. Rekonfigurering starter umiddelbart etterpå.

- *EndReconf* En modulslot er ferdig rekonfigurert.

I tillegg så spesifiserer det for hver logg-hendelse når denne skjedde. Dette spesifiseres i standard UNIX-tid[†] med en nøyaktighet på 1 μ sekund (10^{-6} sekund). I tillegg så oppgis det hvilken oppgave-id, modul-id og slot-id det til enhver tid er snakk om (for AddTask er slotnr = -1).

Etter at disse 92 kjøringene var fullført med en planlegger som benyttet LRU-algoritmen, så ble de samme kjøringene gjentatt med en planlegger som valgte den første ledige modulslotten (FAS - First Available Slot).

3.2.3 Analyse av loggfil

For å undersøke hvor lang tid som ble benyttet til de forskjellige delene av systemet ble det laget et program som leste inn logg-filen generert av programmet.

Dette programmet (logstats, Se tillegg C.1.13) beregnet tiden som ble benyttet til :

- Rekonfigurering
- Venting på at en oppgave skal bli sendt inn i systemet
- Tiden det tar fra en oppgave er sendt inn i systemet til den er ferdig behandlet.

Dette programmet beregner og presenterer både total-tidene og gjennomsnitts-tidene for aktuell kjøring. I tillegg presenterte det også antall rekonfigureringer som ble foretatt under aktuell kjøring.

Resultatene av kjøringene og diskusjon av disse presenteres i kapittel 3.3.

[†]antall sekunder siden 1. januar 1970

3.3 Diskusjon

I starten av dette del-kapittelet blir de forskjellige modulene som ble brukt diskutert. I siste del av dette delkapittelet ser vi på hvilke resultater vi fikk ved de forskjellige kjøringene som ble utført, og hva de forteller oss.

3.3.1 Effektivitet av algoritmene

Modulene som ble implementert og brukt under testingen inneholdt ikke de mest representative algoritmene som kunne velges. Grunnen til at de ble valgt var at de var enkle å implementere slik at de var tilgjengelig tidlig, og derfor kunne benyttet for å teste at funksjonaliteten i systemet var korrekt. Det ble ikke tid til å sette oss inn i og implementere mer relevante modeller, derfor valgte vi å fortsette testingen med de modulene vi allerede hadde implementert.

I dette delen presenterer vi de forskjellige modulene vi benyttet under testing, og diskuterer relevansen av disse og effektiviteten av måten disse brukes.

3.3.1.1 Sum

Den første modulen som ble implementert var en enkel summerings-modul. Denne tok imot en strøm av dataord og beregnet summen av dem etterhvert som de kom inn. Denne algoritmen egner seg dårlig til akselerering i maskinvare da det er enkelt å lage en tilsvarende funksjon i programvare av samme kompleksitetsgrad ($\mathcal{O}(n)$), for eksempel ved:

```
1   sum = 0;
2   for (i = 0; i < datasize; i++) {
3     sum = sum + data[i];
4   } return sum;
```

Med tanke på at den rekonfigurerbare maskinvare ikke klarer å kjøre på samme klokkefrekvens som en standard prosessor, så er det klart at det vil være mye mer effektivt å implementere denne algoritmen i programvare. En annen ting er at dataene som skal legges sammen må flyttes over til den rekonfigurerbare maskinvaren over ett eller annet grensesnitt, noe som medfører en ganske stor overhead i forhold til mengden av nødvendige beregninger per dataelement.

3.3.1.2 Eksponensiel Multiplikasjon Modulo

Den andre modulen som ble implementert ble valgt fordi vi ønsket en mer avansert og beregnings-intensiv funksjon som ville egne seg bedre til maskin-

vare-akselerering.

Denne modulen hadde som oppgave å beregne følgende funksjon :

$$f(a, b, c, m) = a \cdot b^c \mod m \tag{3.1}$$

Dette gjøres iterativt ved følgende algoritme:

```
1  while (c != 0) {
2    if (c & 1)
3      a = a * b % m;
4    b = b * b % m;
5    c = c / 2;
6  } return a;
```

Uttrykket `(c & 1)` er i standard c-syntaks, og er en test på om det minst signifikante bit'et i c er 1 (dvs om tallet er et oddetall). Måten denne algoritmen fungerer på er at en gradvis øker b mens en samtidig reduserer c inntil c er lik 0. Dette ble gjort etter følgende funksjoner:

$$a \cdot b^c = a \cdot (b^2)^{\frac{c}{2}} \tag{3.2}$$

eller

$$a \cdot b^c = ab \cdot (b^2)^{\frac{c}{2}} \tag{3.3}$$

Hvis c var partall benytter en funksjon 3.2, og hvis c var et oddetall, så benyttet en funksjon 3.3 (uttrykket $\frac{c}{2}$ rundes ned til nærmeste heltall).

Grunnen til at vi beregner modulo av resultatene av multiplikasjonene i linje 3 og 4 er at vi vil unngå at svaret blir så høyt at det kan medføre overflyt. Vi har derfor en egen iterativ algoritme for å ta seg av denne multiplikasjonen som tilsvarer funksjonen:

$$f(a, b, m) = a * b \mod m \tag{3.4}$$

og beregnes slik:

```
1  r = 0;
2  while (b != 0) {
3    if (b & 1) {
4      r = r + a;
5      if (r >= m)
6        r = r - m;
7    }
8    a = a * 2;
9    b = b / 2;
10   if (a >= m)
11     a = a - m;
12 } return r;
```

Måten denne algoritmen fungerer på er i prinsippet ganske lik algoritmen for 3.1. Den forsøker å redusere b samtidig som den øker a og r etter følgende

formler:

$$r + a \cdot b = r + 2a \cdot \frac{b}{2} \tag{3.5}$$

eller

$$r + a \cdot b = (r + a) + 2a \cdot \frac{b}{2} \tag{3.6}$$

Hvis b er ett partall så skal en benytte 3.5, og hvis b er oddetall, så skal en velge 3.6 (uttrykket $\frac{b}{2}$ rundes ned til nærmeste heltall).

Multiplikasjon med 2 og divisjon med 2 blir alle utført ved å skifte bitene i dataregisteret en plass til venstre eller høyre. Vi ser også at måten modulo blir beregnet på (linje 5 og 10), er ved å sjekke om et tall er større eller lik m, og i så tilfelle redusere tallet med m (linje 6 og 11).

En begrensning i denne algoritmen var at startverdiene for a, b og c måtte være mindre enn m for å sikre korrekt resultat. Hvis dette ikke var tilfelle, så kunne en på grunn av måten funksjon 3.4 ble implementert på, ikke være sikker på om resultatet var redusert tilstrekkelig av modulo-funksjonen til å unngå overflyt.

Ved å implementere funksjonen i maskinvare på denne måten fikk vi en modul som var mye mer effektiv enn en rett-frem-implementasjon i programvare av samme funksjon hvor en multipliserer b med seg selv c ganger, deretter med a og tilslutt dividerer med m og henter resten.

Men etter at implementasjonen var ferdig, innså vi det var enkelt å implementere den samme algoritmen i programvare på akkurat samme måte, og på den måten få en programvare-funksjon av samme kompleksitetsklasse ($\mathcal{O}(n^2)$). Som med modulen for summering, så ville da programvare-løsningen være mer effektiv på grunn av høyere klokkefrekvens. Siden denne modulen bare brukte 4 innputt-verdier så ville ikke like mye tid gå bort i forbindelse med overføring av data til den rekonfigurerbare maskinvaren, men forsinkelsen i kommunikasjons grensesnittet ville likevel være såpass stor at forholdet mellom ytelsen til programvare-løsningen og maskinvare-løsningen øker ytterligere.

3.3.1.3 Begrunnelse for bruk av modulene

På tross av at disse funksjonene egner seg dårlig til å måle ytelsen av systemet ble de benyttet likevel. Grunnen til dette var at det etterhvert ikke ble tid til å sette seg inn i og utvikle relevante moduler og koble det opp mot en relevant applikasjon. For å gjøre dette ville vi ha måtte ikke bare utviklet modulene, men vi måtte også ha skrevet ett relevant applikasjonsprogram som benyttet seg av systemet vi har utviklet. I tillegg så ville vi måtte brukt

endel ekstra tid på funksjonalitetstesting av det komplette systemet, med de nye modulene.

3.3.1.4 Rekkefølge av moduler

Hovedmålet med oppgaven var å se på hvordan en kan utnytte rekkefølgen modulene blir benyttet på for å automatisk fordele dem utover et gitt antall modulslotter. Siden vi ikke utviklet et relevant applikasjonsprogram så viste vi heller ikke hvordan rekkefølgen ville være. Vi antok derfor at oppgavene kom inn i tilfeldig rekkefølge, men at enkelte moduler ville bli benyttet oftere en andre. For å sette opp listen over oppgaver som ble kjørt ble det skrevet et lite ekstra-program som genererte en tilfeldig oppgaveliste. I denne listen blir hver oppgave listet opp med antall argumenter som skal sendes inn, og hvilken modul som den skal bruke. Rekkefølgen modulene blir valgt i er som sagt tilfeldig, men en kan spesifisere hvor ofte de forskjellige modulene skal velges, dvs hvor stor sannsynlighet for å velge en gitt modul i forhold til de andre modulene. Se kapittel 3.2.1.1 for mer detaljer.

Rekkefølgen modulene blir sendt inn i systemet er derfor heller ikke særlig representativ for hvordan systemet ville oppføre seg i et reelt scenario.

Vi mener likevel at resultatene vi kom fram til har en viss mening. Vi mener at de manglene i systemet som er beskrevet ovenfor vil hvis de blir utbedret i senere arbeider, bare påvirke systemet i en positiv retning. Selv om resultatene vi fikk ved å teste systemet mot "applikasjonen" ikke var så representative som vi ville ønsket, så mener vi likevel at de kan gi endel indikasjoner på hvordan systemet vil fungere opp mot en reell applikasjon.

3.3.2 Testresultater

I denne delen av oppgaven presenteres de forskjellige testene vi kjørte for å få svar på spørsmålene som ble presentert i starten av kapittel 3.2. De forskjellige resultatene blir diskuterer og vi prøver å forklare det vi ser. En oppsummering av de konklusjonene som ble trukket etter å ha studert resultatene, finner en i kapittel 4.1. Datasettene som bli benyttet ble presentert i kapittel 3.2.

3.3.2.1 Duplisering av moduler

Ett av spørsmålene som ble stilt var angående effekten av å tillate å kjøre to oppgaver som begge benyttet den samme modulen samtidig i maskinvaren.

Det vil si at to (eller flere) av modulslottene er konfigurert med den samme modulen på et gitt tidspunkt.

På den ene siden så vil dette kunne medføre at de oftest brukte modulene har en større sjanse til å bli kjørt uten rekonfigurering først hvis det ligger flere av dem allerede rekonfigurert i maskinvaren. I tillegg så ville det kunne medføre at en oppgave som benytter en veldig populær modul, vil bli nødt til å måtte vente unødvendig på å bli kjørt, fordi en annen oppgave tilfeldigvis benyttet den samme modulen samtidig. Dette er noe vi tror vil kunne forekomme ofte.

Hvis en i tillegg til å ikke tillate modul-duplisering heller ikke tillater oppgaveframskyving, så risikerer en også at oppgave-køen stopper opp selv om det er ledige ressurser i den rekonfigurerbare maskinvaren. Dette kommer av at hele systemet må stå å vente på at den modulen som den første oppgave i oppgave-køen ønsker å benytte, skal bli ledig.

På den andre siden så ville det å tillate modul-duplisering kunne medføre at moduler som ofte var ibruk ble unødvendig overskrevet hvis 2 oppgaver som benyttet en mindre brukt modul ble sendt inn omtrent samtidig.

En annen ting vi ønsket å se på i forbindelse med modul-duplisering, var effekten av hvordan systemet oppfører seg når en modul som er konfigurert på mer enn en modulslot blir benyttet. Skal da bare modulslotten som blir benyttet markert som brukt (*-howschedupdateonlythis*), eller skal alle modulslottene som inneholdt den samme modulen markeres (*-howschedupdateallsimilar*)?

For å teste effekten av å tillate modul-duplisering kjørte vi 4 forskjellige kjøringer:

AA : Denne kjøringen benyttet argumentene *-allowduplicates* og *-howschedupdateallsimilar*. Det vil si at duplisering av moduler var tillatt, og når en oppgave ble kjørt i systemet så ble alle modulslotter som hadde den aktuelle modulen inne markert som nylig brukt, uavhengig av om oppgaven ble kjørt på modulslotten eller ikke. Siden vi tillot duplisering her så benytter vi biblioteks-filen og oppgave-filen beregnet på 4 moduler (se kapittel 3.2.1.2). (Allow AllSimilar)

AO : Denne kjøringen benyttet argumentene *-allowduplicates* og *-howschedupdateonlythis*. I likhet med over-nevnte så tillot denne kjøringen duplisering av moduler, men markerer bare modulslotten hvor oppgaven kjører som brukt. I likhet med kjøringen over så benyttet vi her også 4-modulssettet av oppgavedata og biblioteksdata. (Allow Onlythis)

nAs : Denne kjøringen benyttet argumentene *-no_allowduplicates*. Også her benyttet vi oss av 4 modulssettet av oppgavedata og biblioteksdata. (noAllow small)

nAb : Denne kjøringen var identisk med over nevnte, men her benyttet vi oss av 16-modulssettet for oppgave- og modul-data. (noAllow big)

Ingen av disse kjøringene benyttet oppgaveframskyving.

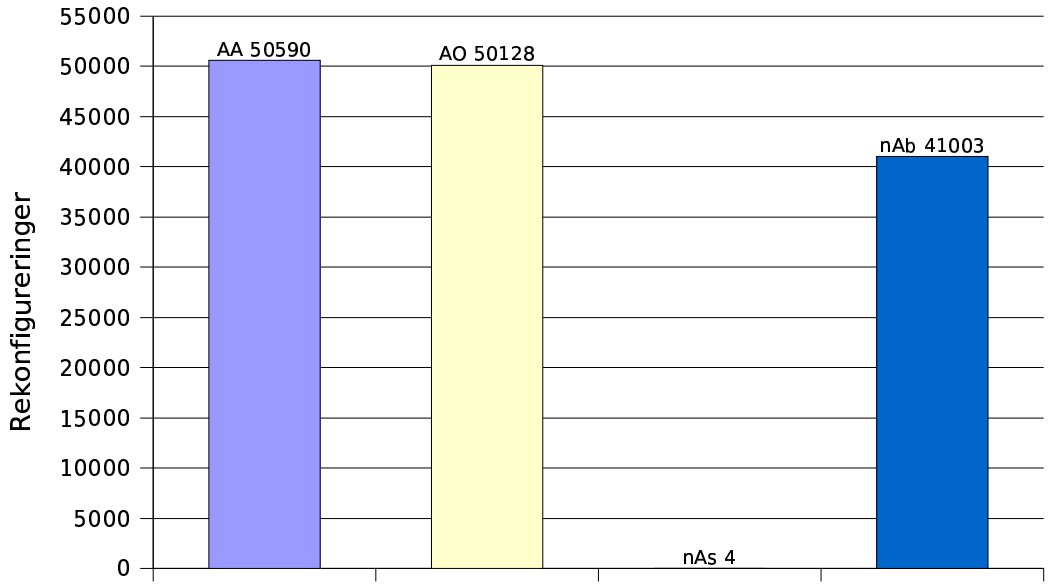
Resultatene av denne kjøringen kan sees i figur 3.15.

Det vi ser der er at hvis duplisering av moduler er tillatt så er det minimal forskjell i antall rekonfigureringer og kjøretiden mellom AA- og AO-kjøringene. AO-kjøringen var litt raskere, men siden denne forskjellen var mindre enn 1%, kan vi ikke si noe om hvilke av disse kjøringene som er den mest optimale.

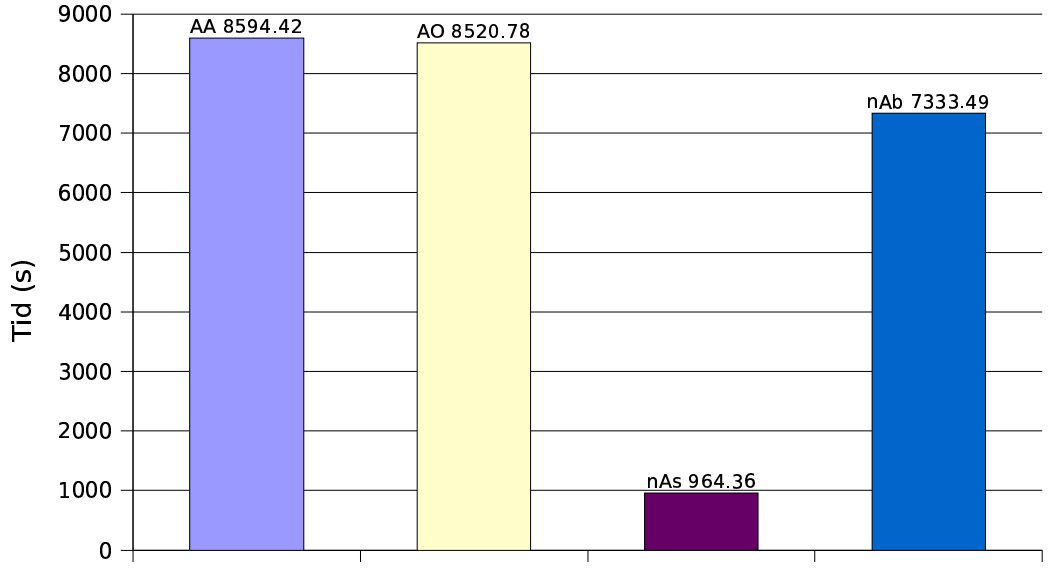
Mer iøynefallende er resultatet for nAs-kjøringen. Her ser vi at det bare ble foretatt 4 rekonfigureringer, og kjøretiden er bare rundt 10% av AA og AO. Dette er imidlertid lett å forklare og egentlig ganske innlysende. Siden denne kjøringen benyttet et modulsett på 4 moduler og ikke tillot duplisering, så medfører dette sammen med det faktum at designet har 4 modulslotter, at hver slot vil bli konfigurert med en modul som ikke vil bli overskrevet av andre moduler. Dette fordi når en oppgave kommer inn i systemet så vil den enten starte opp på en ferdig-konfigurert modulslot, vente til en ferdig-konfigurert modulslot blir ledig, eller hvis den aktuelle modulen ikke har vært brukt enda, så vil den konfigurere opp en ny modul. Forskjellen i kjøretid kommer for det meste av at den ikke trenger å rekonfigurere like mange ganger. Vi ser at det er rundt 50000 færre rekonfigureringer i forhold til AA- og AO-kjøringene, hvorav hver rekonfigurering er angitt til å ta 146310 mikrosekunder* (Se tillegg C.3.3). Dette tilsier at systemet bruker rundt $50000 reconf * 146310 \frac{\mu s}{reconf} = 7,3155 * 10^9 \mu s = 7315,5 s$ mindre tid på rekonfigurering noe som stemmer godt med resultatene. I tillegg kommer tiden systemet står å venter på at en spesifikk oppgave skal bli ferdig, når en annen oppgave ønsker å benytte en opptatt modul.

En annen egenskap med figurene som en legger merke til er at nAb-kjøringene har merkbart bedre resultater en AA- og AO-kjøringen. Denne kjøringen tillater som nevnt ikke duplikater av moduler, og regner alle 16 modulene i maskinvaren som unike moduler (i motsetning til 4-modulssettet for oppgave-filen/biblioteks-filen som regner en gruppe av 4 moduler som en unik modul, se kapittel 3.2.1.2). Et spørsmål som dukker opp her er hvorfor en får bedre ytelse når antall moduler går opp? En skulle kanskje tro at når antall moduler gikk opp så ville antall rekonfigureringer gå opp av den grunn at det er større sjanse for at en gitt oppgave ikke benytter en av de sist brukte

*Regnet ut fra målinger av komplette rekonfigureringer og antagelse om at rekonfigureringstiden er proporsjonal med arealet og at hver modul tar $\frac{1}{4}$ av totalarealet (samme antagelse som i 3.1.2.6)



(a) Rekonfigureringer



(b) Kjøretid

Figur 3.15: Antall rekonfigureringer og kjøretid uten bruk av oppgaveframskyving

	Taskforward Register	Taskforward Noregister
Allowdup, Allowsimilar	AAtr	AAtn
Allowdup, Onlythis	AOtr	AOtn
Noallowdup, Small mod-/task-set	nAstr	nAstn
Noallowdup, Big mod-/task-set	nAbtr	nAbtn

Tabell 3.1: Oversikt over kjøre-koder for testing av oppgaveframskyving.

modulene, men det kan virke som om systemet er istand til å bedre fordele modulene over modulslottene. Men en bør likevel ikke ta dette for gitt da det også kan ha en sammenheng med den eksponentielle distribusjonen av moduler (se kapittel 3.2.1.1), eller andre tilfeldigheter i oppgavesettene våre.

En mulig forklaring er at i 4-modulssettet varierer fordelingen av moduler mellom 1 og 2 andeler for hver av de 4 modulene (totalt 6 andeler), det vil si $\frac{1}{6}$ hver av SUM0 og EXP0, og $\frac{1}{3}$ hver av SUM1 og EXP1 (se 3.3.1 for beskrivelse av disse). Dette kommer i motsetning til 16-modulssettet hvor modul-fordelingen varierer fra 1 til 128 andeler (av totalt 510 andeler), noe som gir en mye jevnere spredning enn 4-modulssettet. Dette kan ha medvirket til at fordelingen av oppgaver i oppgavesettet for 4 moduler har generert mer støy for planleggeren, slik at den oftere tar gale avgjørelser.

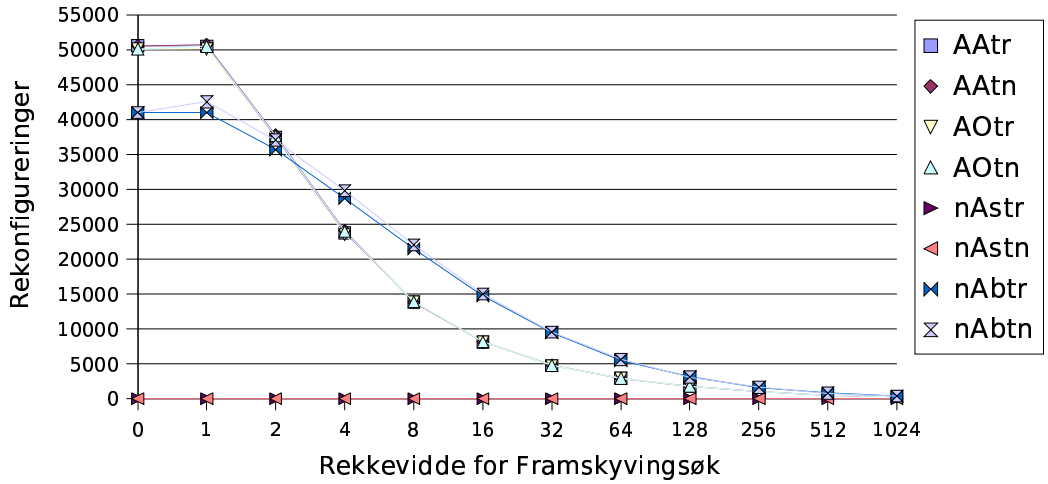
Som en oppsummering kan vi si at vi fant ut at ved modul-duplisering, så var det ingen merkbar forskjell på om en lot systemet markere alle modulslotter med samme modul som brukt når en av dem ble brukt, eller om en lot systemet bare markere den aktuelle modulslotten.

3.3.2.2 Oppgaveframskyving

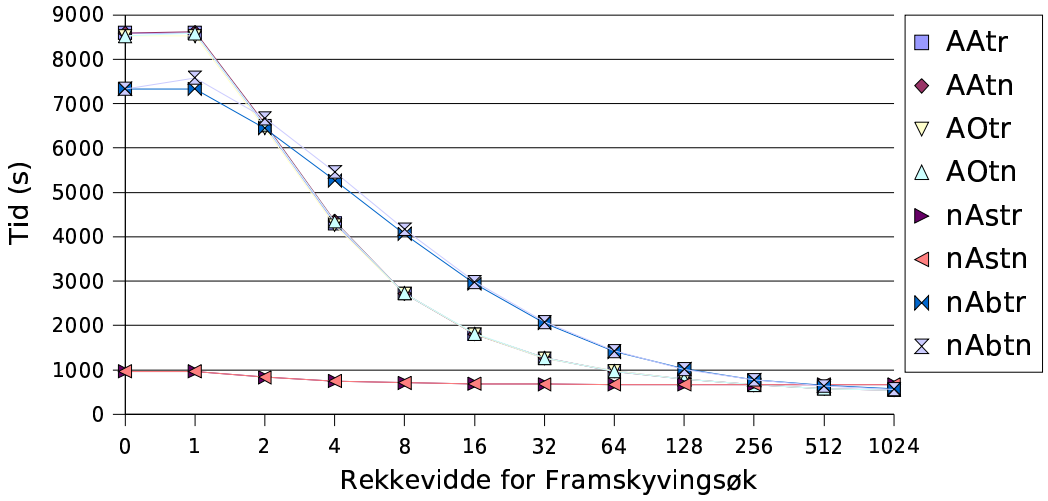
Et annet spørsmål vi ønsket å se på var effekten av å la planleggeren se lengre bak i oppgave-køen og på den måte framskyve oppgaver som kan kjøres uten rekonfigurering. Vi ønsker å se på effekten av å variere hvor langt bak i køen planleggeren søker kombinert med hvorvidt en tillater duplisering av moduler (samme variasjoner som i kapittel 3.3.2.1).

Resultatene av testkjøringene for dette kan sees i figur 3.16. Navnekodene på kjøringene er stort sett de samme som i kjøringene for duplisering av moduler, men har et tillegg som skiller på om en modulslot skal markeres som brukt(tr - taskforward register) eller ikke(tn - taskforward noregister) når en framskyvet oppgave blir kjørt på den Se tabell 3.1 for en oversikt over de forskjellige variantene. Resultatene for kjøringene for duplisering av moduler er også tatt med i figurene og vises i kolonnen for rekkevidde for framskyvingsøk lik 0.

Det vi ser i denne grafen er at det i praksis ikke er forskjell mellom kjøringene for AAtr, AAtn, AOtr og AOtn uansett hvor langt bak i oppgave-



(a) Rekonfigureringer



(b) Kjøretid

Figur 3.16: Antall rekonfigureringer og kjøretid ved bruk av oppgaveframskyving.

køen en lar planleggeren søke. Vi ser også at dette også gjelder for forskjellene mellom kjøringene nAstr og nAstn, og for forskjellen mellom kjøringene nAbtr og nAbnt.

Som i kjøringene i 3.3.2.1, så ser vi at kjøringene for 4-modulssettet uten duplisering (nAstr og nAstn) ligger under de andre kjøringene, og kan sies å forme en slags grense for minimum antall rekonfigureringer og kjøretid for modellen vår.†

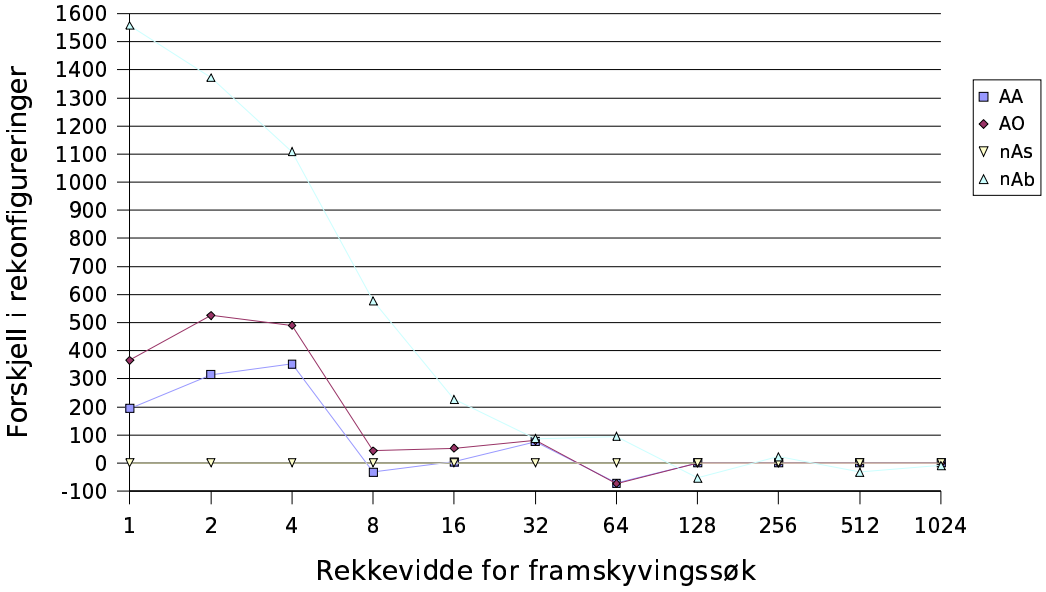
Vi ser også at kjøringene nAbtr og nAbtn (16-modulssettet) har mindre rekonfigureringer og kortere kjøretid enn de kjøringene som tillater modul-duplisering når planleggeren bare ser 1 steg lengre bak i oppgave-køen (det samme som vi fant ut av for duplisering av moduler). Men vi ser også at når planleggeren ser ytterligere flere (4 eller flere) plasser bakover så forandrer dette forholdet seg til at kjøringene som benytter 4-modulssettet (AAtr, AAtn, AOtr og AOtn) er raskere. Dette kommer sannsynligvis av at sjansen for å finne en oppgave som benytter en gitt modul når en ser lengre bak i oppgave-køen blir mindre jo flere moduler en har tilsammen. Men en bør fremdeles tenke på at dette tydeligvis ikke gjelder får en bare ser et 1 eller 2 plasser bakover, eller ikke benytter oppgaveframskyving i det hele tatt.

En annen ting med grafene i figur 3.16 som en bør merke seg er at når planleggeren bare søker 1 plass bakover i oppgave-køen, så går ytelsen i systemet ned for de fleste av kjørings-typene. En forklaring som en lett kan komme til å tenke på er at dette kommer av at planleggeren må gjøre mer arbeid per oppgave. Men for det første så er jobben med å se en plass bakover i køen så liten at det tvilsomt har noe å si. For det andre så ser vi at dette ikke bare gjelder kjøretiden, men også antall rekonfigureringer. Dette kommer best fra i kjøringene for AAtn, AOtn og nAbtn.

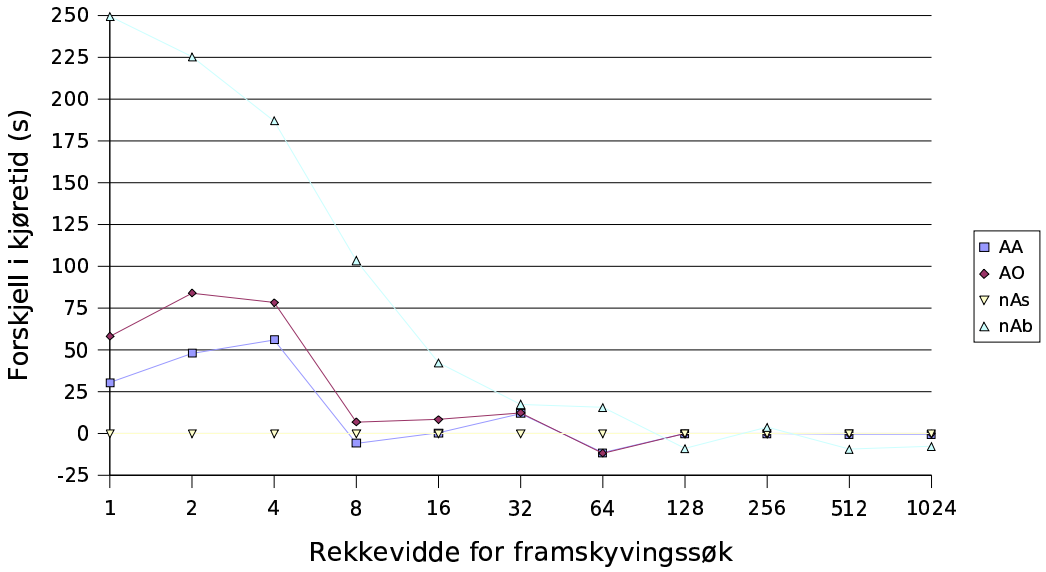
I figurene i 3.17 så ser vi tn-kjøringene har generelt flere rekonfigureringer og bruker lengre tid på å fullføre enn tr-kjøringene når planleggeren bare søker et relativt kort stykke bakover i oppgave-køen. Dette mener vi kommer av at hvis vi har en ofte brukt modul som bruker relativt kort tid på å fullføre en oppgave, så risikerer en at planleggeren ikke oppdager at den aktuelle modulen er ofte brukt, og kan komme til å overskrive den med en langsommere og mindre brukt modul. Vi mener derfor at en bør registrere bruk av modulslotter ved oppgaveframskyving.

Vi ser også av figurene i 3.17 at vi ikke kan si det samme når planleggeren søker lengre bak i oppgave-køen. Dette tror vi kommer av at når planleggeren kan søke så langt bakover, så er den oftere istand til å framskyve en oppgave

†I teorien er det mulig å komme under denne grensen for kjøretid, hvis systemet ikke må vente unødvendig på å starte opp oppgaver, for eksempel hvis første oppgave i oppgavelisten benytter en modul som allerede er ibruk, og modul-duplisering ikke er tillat.



(a) Rekonfigureringer



(b) Kjøretid

Verdien som presenteres her er funnet ved å trekke verdien av tr-kjøringene fra tn-kjøringene.

Figur 3.17: Forskjell mellom tn-kjøringene og tr-kjøringene.

slik at den sjeldnere må rekonfigurere, og på den måten reduserer forskjellen mellom tr- og tn-kjøringene.

En siste observasjon av figur 3.16(b) er at hvis vi øker rekkevidden av framskyvingsøket tilstrekkelig så ender vi opp med kjøretider som matcher nAs-kjøringene, som tidligere nevnt former en slags grense for hvor raskt kjøringene kan gå. Vi ser at ved å søke 512 (eller lengre) plasser bakover i oppgave-køen så er faktisk nAb-kjøringene raskere enn nAs-kjøringene, men forskjellen er så liten at vi ikke kan si noe sikkert om det, eller om det bare er tilfeldigheter.

Av dette mener i å kunne trekke følgende påstander:

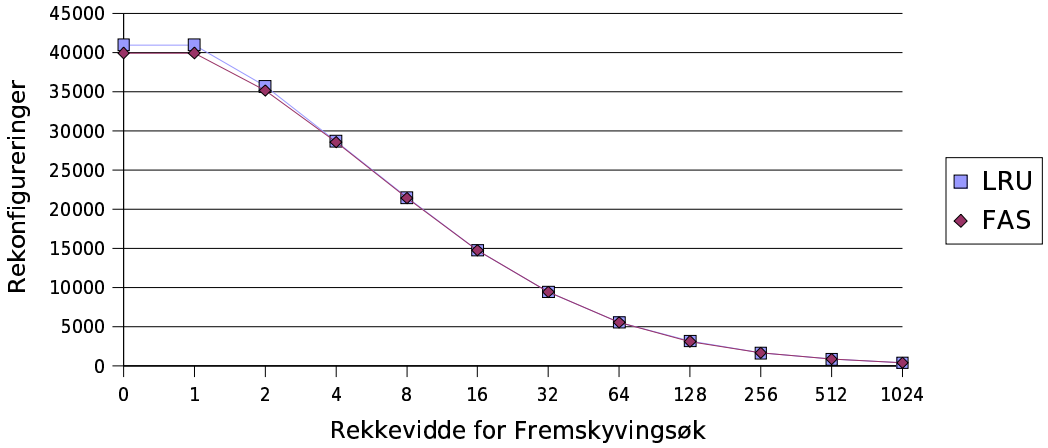
- Ved oppgaveframskyving er fordelen ved å registrere modulslotten som brukt registrerbar ved korte søk i oppgavelisten, men ved lengre søk i oppgavelisten er den neglisjerbar, og kan slå negativt ut.
- Ved oppgaveframskyving er forskjellen mellom AA og AO kjøringene fortsatt neglisjerbar.
- Oppgaveframskyving kan forbedre ytelsen til systemet dramatisk, gitt at systemet har en tilstrekkelig mengde oppgaver liggende i kø.

Vi minner om at lengre søk i oppgavelisten kan være urealistisk på grunn av at mengden av oppgaver som sendes inn samtidig i et mer relevant system sannsynligvis ligger langt lavere enn det vi modellerer i vårt system, hvor alle oppgavene sendes inn samtidig.

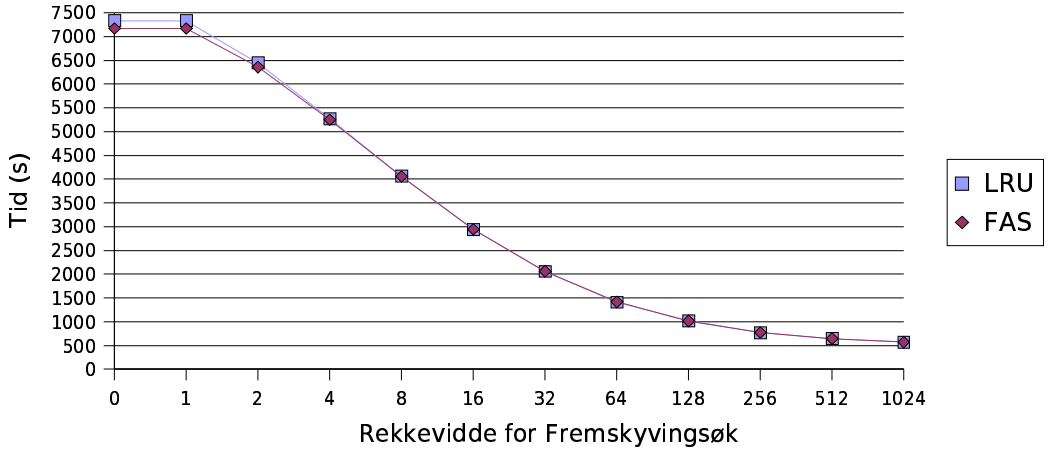
3.3.2.3 LRU kontra FAS

En av de viktigste faktorene vi ønsket å se på var effekten av å benytte en LRU-basert planlegger til å avgjøre hvilke moduler som skulle bli liggende konfigurert og hvilke moduler som skulle kastes ut, når en ny modul skulle konfigureres inn. For å teste dette kjørte vi endel tester hvor vi benyttet to forskjellige planleggere. Den ene benyttet LRU-algoritmen, mens den andre alltid byttet ut den første ledige modulslotten den fant (FAS - First Available Slot).

På grunnlag av det som er presentert hittil i oppgaven så har vi valgt ut nAbtr som testkjøring for å teste ut LRU mot FAS. Dette mente vi var det beste valget på grunn av at nAb-kjøringene er de eneste kjøringene som benytter 16 moduler, noe som vi mente var mer representativt enn å bruke 4 moduler, for å modellere en reell applikasjon. Grunnen til at tr-varianten er valgt (oppgaveframskyving med registrering) istedet for tn-varianten (oppgaveframskyving uten registrering) var det som kom fram til i kapittel 3.3.2.2, nemlig at tr-kjøringene viste en bedre ytelse en tn-kjøringene.



(a) Rekonfigureringer



(b) Kjøretid

Figur 3.18: Forskjell mellom LRU og FAS for nAbtr-kjøring

Her fikk vi oss en overraskelse. Vi hadde regnet med at LRU-planleggeren ville yte bedre en FAS-planleggeren, men det ser vi at ikke er tilfelle. FAS-planleggeren har som vi ser i figur 3.18 litt bedre ytelse enn LRU-planleggeren, men vi ser også at denne forskjellen er så liten at vi ikke kan si noe sikkert om hvorfor.

En ting en bør tenke på at at det bare benyttes 16 forskjellige moduler i denne kjøringen, noe som er relativt lite i forhold til 4 modulslotter. Dette kan medføre at LRU-algoritmen ikke fungerer så bra som vi i starten hadde håpet på. LRU-algoritmen slik den benyttes i cache-strukturer for minne, benytter tusenvis av slotter, med millioner av forskjellige verdier (minneaddresser). I tillegg bør en tenke på at nAbtr-kjøringen ikke tillater modul-duplisering. Som nevnt i 3.3.2.1, så kan dette medføre problemer med at en ofte brukt modul kan skape kø i systemet.

3.3.2.4 Programvare-løsning kontra Maskinvare-løsning

En siste ting vi ønsket å se på var effektiviteten av vårt design sammenlignet med å kjøre tilsvarende oppgaver på en vanlig datamaskin uten maskinvare-akselerasjon. Dette gjorde vi ved å ta test-programmet(Se tillegg C.1.16) vårt og skrive det om slik at istedenfor å sende oppgaver inn i oppgave-køen og la programvare-driveren ta seg av dem, så ble de kjørt en etter en. Til dette så hadde vi laget et programvare-bibliotek(Se tillegg C.1.19 og C.1.18) med programvare-implementasjoner av de modulene vi benyttet i maskinvaredesignet vårt. Det ferdige resultat-programmet finner en i tillegg C.1.17.

Som tidligere nevnt så var ikke modulene som vi hadde implementert i systemet spesielt godt egnet til maskinvare-akselerasjon. Dette fikk vi enda en gang bekreftet da vi så at ved å kjøre det nye test-programmet på samme maskinvare som vi kjørte de andre testene på (men uten å benytte den rekonfigurerbare maskinvaren) ble alle oppgavene i oppgavesettet ferdig kjørt på litt over 2 sekunder. Til denne testen benyttet vi oppgavesettet som besto av 100000 oppgaver som hver kunne være en av 16 forskjellige typer (samme oppgavesett som for nAb-kjøringene). Til sammenligning så vi at den beste kjøringen vi hadde for nAb-kjøringene tok 563 sekunder, og den tregeste kjøringen tok 7583 sekunder.

Dette viser at modellen vi har kjørende på systemet vårt ikke bare er uegnet som eksempel-applikasjon, men også at den gir særdeles dårlig ytelse i forhold til en ren programvare-løsning.

Men vi har tro på at det ved å videreutvikle systemet og implementere et mer relevant modulsett, og koble systemet opp mot en reell applikasjon, så skal det være mulig å få et system med bedre ytelse. Vi har også tro på

at ved å forbedre planleggeren vår, og implementere endel av forbedringene nevnt i kapittel 4.2, så vil vi få et system som gir bedre ytelse enn en ren programvare-løsning.

Kapittel 4

Oppsummering

I denne delen av oppgaven presenterer vi de konklusjonene vi trakk etter å ha studert resultatene vi fikk av de forskjellige testkjøringen våre. Det var flere ting som vi etterhvert fant ut at systemet manglet. Dette kapittelet avsluttes derfor med en presentasjon av forskjellige forbedringer av systemet som vi mener egner seg som videre arbeid.

4.1 Konklusjon

I denne oppgaven har vi designet et system for å dynamisk fordele partielt rekonfigurerbare moduler over et sett med modulslotter.

Vi har sett på en modell hvor systemet i utgangspunktet ikke vet noe om avhengigheter mellom de oppgavene som blir sendt inn til systemet. Vi har valgt å la systemet heller se på rekkefølgen de forskjellige modulene blir brukt i, og på grunnlag av dette prøve å unngå unødvendige rekonfigureringer av ofte brukte moduler.

Vi har designet både programvare-design og nødvendig maskinvare-design fra bunnen av. Prototypen vår av maskinvaredesignet vårt kjører på et Ben-ERA-kort fra Nallatech som er utstyrt med en Virtex 1000E fra Xilinx. Kommunikasjonen mellom programvare-driver og maskinvare-design er blitt gjort via et eget programbibliotek fra Nallatech.

Fordelingen av moduler på modulslotter blir foretatt fortløpende og i henhold til en least recently used (LRU) algoritme.

I vår oppgave har vi forsøkt å få svar på endel spørsmål angående retningslinjer for hvordan en best kan utnytte dette systemet.

Vi har i våre tester funnet at ved duplisering av moduler, gir det ikke noen markant forskjell i ytelse om når en gitt modul benyttes, markerer alle

modulslotter som inneholder denne modulen som brukt, eller bare markerer den modulslotten som faktisk blir brukt.

Videre har vi funnet at det ved oppgaveframskyving kan lønne seg å markere en modulslot som kjører en framskyvet oppgave som brukt, men at denne fordelen blir mindre jo lengre bak i oppgave-køen framskyvingsøket går.

Vi ser også at fordelen av å benytte oppgaveframskyving utveier ulempen med ekstraarbeid for planleggeren.

Vi har også sett at LRU-algoritmen ikke egner seg så godt til denne type bruk, og vi mener at dette kommer delvis av at vi har et ukomplett system og urelevant datamodell, men at hovedgrunnen til dette kommer av at forholdet mellom antall moduler, og antall tilgjengelige modulslotter er alt for lavt. Vi mener derfor at en annen planleggings-strategi bør velges.

Vi viste til slutt at systemet vårt ikke bare feilet i å oppnå forbedring av ytelse, men medførte at den totale kjøretiden økte med en størrelses-orden på 2 til 3, men vi mener at dette primært kommer av valget vår av moduler siden disse i utgangspunktet ikke egnet seg til maskinvare-akselerasjon.

4.2 Videre arbeid

Systemet som presenteres i denne oppgaven er langt ifra perfekt. Det er som tidligere nevnt flere ting som burde vært utbedret. Blant disse finner vi bla. annet fungerende partiell rekonfigurering og bygging av mer egnet test-plattform. I dette del-kapittelet presenteres en del forslag vi har angående videre arbeid på systemet.

4.2.1 Bygging av test-plattform

Til dette systemet lå valget mellom å benytte et BenERA-kort fra Nallatech eller et MicroBlaze-kort fra Memec. Begge disse kortene hadde sine fordeler og ulemper. Det vi til å begynne med hadde ønsket var å lage vårt eget kort tilpasset våre behov.

Det vi ønsket var en rask FPGA som støttet partiell rekonfigurering. Den måtte også ha et raskt grensesnitt mot programvare-driveren. I tillegg ønsket vi kontroll med hvilke pinner på FPGAen som brukes til hva, slik at vi bedre kunne tilrettelegge layouten kortet for partiell rekonfigurering.

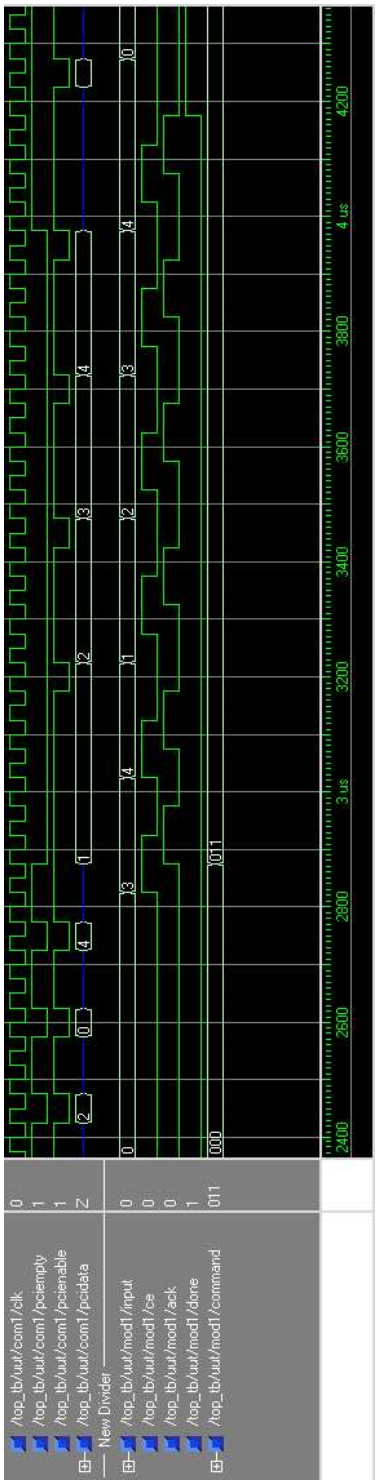
Andre ting vi tenkte på var muligheten til i forbindelse med bygging av et eget kort, var å la modulene kunne kommunisere med hverandre og eventuelle andre periferi-enheter som for eksempel ekstra minne-moduler.

Dette vil medføre at programvare-driveren må utvides med et eget bibliotek for kommunikasjon over for eksempel PCI-bussen.

4.2.2 Grensesnitt mellom programvare-driver og rekonfigurerbar maskinvare

Grensesnittet mellom programvare-driveren og kommunikasjons-modulen i maskinvaredesignet er basert på å sende ett og ett dataord. Dette bør kunne forbedres ved å benytte buffering i maskinvaren slik at det det blir mulig å sende data i såkalt burstmodus. Det vil si at en sender dataene som en kontinuering strøm av dataord, og ikke kreve kvittering for hvert ord.

Måten denne kommunikasjonen fungerer på nå er at kommunikasjons-modulen tar imot et dataord, og sender dette videre til aktuell modul, før den tar imot et nytt dataord. Vi ser i figur 4.1 et lite utsnitt av blant annet sending av 4 dataord til en av modulene, og vi ser at det tar 5 klokke-sykler per ord. Ved å sende data i burstmodus og benytte buffering i kommunikasjons-modulen, så skulle det være mulig å ikke bare sende dataene til og fra kommunikasjons-modulen mye raskere, men det burde også muliggjøre raskere kommunikasjon mellom kommunikasjons-modulen og de rekonfigurerbare modulene.



Figur 4.1: Signal-simulering for sending av data til maskinvare-modul

4.2.3 Raskere kommunikasjon mellom moduler

Grensesnittet mellom kommunikasjons-modulen og modulslottene er per idag synkront med kontrollsignaler som minner om det en finner i asynkrone kommunikasjons-busser. Dette grensesnittet kjører på samme klokke som PCI-bussen. Et forslag til forbedring er å gjøre dette grensesnittet 100% asynkront, eventuelt skrive det om og la det gjøre på en høyere klokkefrekvens. Dette vil kreve at en utvider systemet til å benytter buffering av data i kommunikasjons-modulen og de rekonfigurerbare modulene.

4.2.4 Partiell rekonfigurering

En vesentlig del av systemet som vi ikke fikk til å fungere var partiell rekonfigurering. Dette løste vi som tidligere nevnt ved å ha alle modulene ferdig konfigurert i maskinvaren på forhånd, og benytte et buss-system inne på den rekonfigurerbare maskinvaren til å koble dem opp mot de forskjellige modulslottene.

Det vi ønsker er et komplett fungerende system hvor de forskjellige modulene blir lest inn fra biblioteks-filene og konfigurert inn til den rekonfigurerbare maskinvaren.

Et system for generering og vedlikehold av biblioteks-filene burde også utvikles.

4.2.5 Relevant applikasjon

Datamodellen som vi har benyttet under testing så langt er som tidligere nevnt ikke spesielt representativ for en applikasjon som kan ha nytte av maskinvare-akselerasjon. For å kunne gjøre mer nøyaktige tester og få mer pålitelige data bør systemet testes ut opp mot en mer relevant applikasjon som for eksempel kryptering, bildebehandling eller lignende beregnings-tunge applikasjoner.

4.2.6 Simulator

En annen ide som det ikke ble tid til, var å implementere en fullverdig simulator for systemet. På den måten ville en kunne å en enkel måte testet ut flere ting ved systemet som vi ikke fikk implementert, som effekten av duplisering av moduler med et modulsett på mer en 4 moduler.

4.2.7 Modul-bytte

Et alternativ til å implementere en simulator for systemet hvis en ønsker å teste ut duplisering av moduler med mer en 4 moduler er å legge skrive om programvare-driveren slik at når systemet ønsker å starte opp en modul som allerede er ibruk, så vil systemet heller bruke en annen modul som gjør samme jobben. Dette er mulig siden vi i vårt test-system bare bruker 2 forskjellige typer moduler (SUM og EXP), og maskinvaredesignet vårt har 8 av hver av disse ferdig konfigurert. Derfor skulle det ikke være noe stort problem å for eksempel la en modul som ønsker å benytte SUM2 benyttet SUM7 istedenfor. Systemet må da være istand til å holde rede på at SUM7 også er i bruk, slik at hvis en ny modul ønsker og benytte enten SUM2 eller SUM7, før disse er ferdige, så skal systemet tilordne den en annen SUM-modul.

4.2.8 Interrupt-basert kommunikasjon

Måten systemet henter tilbake status for oppgavene nå, er at programvare-driveren sjekker status maskinvaren kontinuering. Dette er ikke den mest effektive metoden. En mer attraktiv metode hadde vært at maskinvaren selv kunne si ifra om endringer i status for oppgavene ved hjelp av et avbrudd-system.

Tillegg A

Programvare benyttet

Til designet av systemene i denne oppgaven benyttet vi endel programmer. Endel av disse programmene er vanlige tekst program som en benytter daglig til andre oppgaver som GNU Emacs[ema] og TextPad[tex] alt etter-som hvilken plattform en sitter ved. Av andre programmer som vi brukte, som er med spesifikke for vår oppgave vil vi trekke fram ISE Foundation fra Xilinx og ModelSim SE[mod] fra Mentor Graphics[men]. Til kompilering av programvare-designet vår benyttet vi GCC[gcc].

A.1 ISE Foundation

ISE Foundation er en komplett IDE-pakke fra Xilinx som inneholder alt en trenger for utvikling av maskinvare-design. Pakken består av en prosjekt-navigatør hvor en kan opprette forskjellige prosjekter og hvor hoveddelen av utviklingen av maskinvaren foregår. Via denne prosjekt-navigatøren har en mulighet til å starte opp endel andre prosesser som blir tatt hand om av andre programmer fra pakken. Eksempler på slike prosesser er kompilering av HDL-kode som blir gjort av XST, begrensninger av areal/pinner på ferdig brikke/FPGA ved hjelp av PACE og/eller Floorplanner og generering av testbenker. I tillegg så har ISE Foundation støtte for endel tredje-parts programmer til oppgaver som for eksempel simulering ved hjelp av ModelSim fra Mentor Graphics.

A.2 ModelSim

ModelSim er et simuleringsverktøy fra Mentor Graphics som er istand til å kompilere HDL-kode, og kjøre den ved help av en testbenk, enten generert av ISE Foundation, eller manuelt satt opp. I denne oppgaven så valgte vi å sette

opp testbenkene våre selv, da vi ofte ønsket å forandre på test-parametrene som vi enkelt kunne forandre ved hjelp av en vanlig tekst-editor.

Ved å benytte ModelSim sammen med ISE Foundation hadde vi muligheten til kompilere HDL-filene våre på en slik måte at de i ModelSim genererte korrekt simulering med tanke på forsinkelser i datalinjene og komponentene som designet blir avbildet på.

Tillegg B

Programvare-bibliotek

I programvare-designet vårt benyttet vi noen biblioteks pakker. Disse var Pthread som benyttes til å ta seg av behandling av et sett med tråder i et program, og FUSE fra Nallatech[nal] som er et bibliotek vi benyttet til å kommunisere med BenERA-kortet hvor vi kjørte vårt maskinvare-design.

B.1 Pthread

Pthread (POSIX thread) er en tråd-pakke som kjører på UNIX og Linux. Den er basert på IEEE POSIX 1003.1c-1995 (også kjent som ISO/IEC 9945-1:1996) standarden.

I vårt programvare-system blir denne pakken benyttet til å starte opp monitor-tråden som overvåker maskinvare-systemet vårt. Den bli også benyttet til å starte opp prosessene som brukes til å simulere tiden det å kjøre en oppgave på en av de rekonfigurerbare modulene når systemet kjører i simulator-modus.

B.2 FUSE C/C++ API

Dette er et bibliotek fra Nallatech[nal] som tar seg av kommunikasjonen med deres maskinvareakselerasjons-kort (som for eksempel BenERA). Dette biblioteket inneholder funksjonalitet for total kontroll over kortet som rekonfigurering, resetting og setting av forskjellige parameter som for eksempel klokkefrekvens.

Bibliografi

[ABT04] Ali Ahmadinia, Christophe Bobda, and Jürgen Teich. A dynamic scheduling and placement algorithm for reconfigurable hardware. In Christian Müller-Schloer, Theo Ungerer, and Bernhard Bauer, editors, *Proc. of 17th International Conference on Architecture of Computing Systems (ARCS 2004)*, volume 2981 of *Lecture Notes in Computer Science (LNCS)*, pages 125–139, Augsburg, Germany, March 2004. Springer.

[alt] FPGA, CPLD & Structured ASIC Devices Altera, the Leader in Programmable Logic. http://www.altera.com/.

[amda] Advanced Micro Devices,AMD - Homepage. http://www.amd.com/.

[amdb] AMD Opteron Processor. http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_8825,00.html. Product Information.

[arm] ARM. http://www.arm.com.

[atm] Atmel Corporation. http://www.atmel.com/.

[CH02] Katherine Compton and Scott Hauck. Reconfigurable computing: a survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, 2002.

[craa] Cray XD1 Overview. http://www.cray.com/products/xd1/index.html. Technical Specification.

[crab] Mersenne Twister - A Reconfigurable Computing Example. http://www.cray.com/products/xd1/accel_examples.html.

[crac] Cray Inc - The Supercomputer Company. http://www.cray.com/.

[eas] Xilinx : EasyPath Series. http://www.xilinx.com/products/silicon_solutions/fpgas/easypath/index.htm.

[ema] GNU Emacs - GNU Project - Free Software Foundation(FSF). http://www.gnu.org/software/emacs/.

[EMSS00] Hossam ElGindy, Martin Middendorf, Hartmut Schmeck, and Bernd Schmidt. Task rearrangement on partially reconfigurable FPGAs with restricted buffer. In R. W. Hartenstein and H. Grünbacher, editors, *Field Programmable Logic and Applications, 10th Int. Conf. (FPL 2000)*, pages 379–388, Berlin, 2000. Springer Verlag.

[FPG+03] Osman Devrim Fidanci, Daniel S. Poznanovic, Kris Gaj, Tarek A. El-Ghazawi, and Nikitas A. Alexandridis. Performance and overhead in a hybrid reconfigurable computer. In *IPDPS*, page 176, 2003.

[gcc] GCC Home Page - GNU Project - Free Software Foundation(FSF). http://www.gnu.org/software/gcc/.

[HT01] Pauline C. Haddow and Gunnar Tufte. Bridging The Genotype-Phenotype Mapping For Digital FPGAs. In *Evolvable Hardware*, pages 109–115, 2001.

[KAD03] L. Kessal, N. Able, and D. Demigny. Real-time image processing with dynamically reconfigurable architecture. *Real-Time Imaging*, 9(5):297–313, Oktober 2003.

[KB00] Majid Sarrafzadeh Kiarash Bazargan, Ryan Kastner. Fast template placement for reconfigurable computing systems. *IEEE Design and Test of Computers*, 17:68–83, 2000.

[KV04] Jawad Khan and Ranga Vemuri. An efficient battery-aware task scheduling methodology for portable rc platforms. In *FPL*, pages 669–678, 2004.

[LWW03] Sin Ming Loo, B. Earl Wells, and J. D. Winningham. A genetic algorithm approach to static task scheduling in a reconfigurable hardware environment. In Narayan C. Debnath, editor, *Computers and Their Applications, ISCA 18th Int Conf*, pages 36–39, Honolulu, Hawaii, USA, Mars 2003.

[MBV+02] Théodore Marescaux, Andrei Bartic, Dideriek Verkest, Serge Vernalde, and Rudy Lauwereins. Interconnection networks enable fine-grain dynamic multi-tasking on FPGAs. *j-LECT-NOTES-COMP-SCI*, 2438:795–805??, Semptember 2002.

[mem] Memec - Semiconductor Distributor - Programmable Logic, Analog, ASSP. http://www.memec.com/.

[men] Mentor Graphics :: The EDA Technology Leader. http://www.mentor.com/.

[MM04] Rajarshi Mukherjee and Seda Ogrenci Memik. Power-driven design partitioning. In *FPL*, pages 740–750, 2004.

[mod] ModelSim Products. http://www.modelsim.com/.

[Mor05] Kevin Morris. Cray goes FPGA. *FPGA and Programmable Logic Journal*, april 2005.

[MSSE02] Martin Middendorf, Bernd Scheuermann, Hartmut Schmeck, and Hossam ElGindy. An evolutionary approach to dynamic task scheduling on FPGAs with restricted buffer. *Journal of Parallel and Distributed Computing*, 62(9):1407–1420, September 2002.

[nal] Nallatech FPGA Computing Solutions. http://www.nallatech.com/.

[NB02a] Juanjo Noguera and Rosa M. Badia. Dynamic run-time HW/SW scheduling techniques for reconfigurable architectures. In *Proceedings of the tenth international symposium on Hardware/software codesign*, pages 205–210. ACM Press, 2002.

[NB02b] Juanjo Noguera and Rosa M. Badia. HW/SW codesign techniques for dynamically reconfigurable architectures. *IEEE Trans. Very Large Scale Integr. Syst.*, 10(4):399–415, 2002.

[ope] OPENCORES.ORG. http://www.opencores.com.

[PBV04] E. Moscu Panainte, K. Bertels, and S. Vassiliadis. Dynamic hardware reconfigurations: Performance impact on MPEG2. In *Proceedings of the International Workshop on Systems, Architectures, Modeling, and Simulation*, pages 284–292, July 2004.

[pow] PowerPC - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/PowerPC.

[RMV+03] Javier Resano, Daniel Mozos, Diederik Verkest, Serge Vernalde, and Francky Catthoor. Run-time minimization of reconfiguration overhead in dynamically reconfigurable systems. In Peter Y. K. Cheung, George A. Constantinides, and José T. de Sousa, editors, *FPL*, volume 2778 of *Lecture Notes in Computer Science*, pages 585–594. Springer, 2003.

[RVM+04] Javier Resano, Diederik Verkest, Daniel Mozos, Serge Vernalde, and Frencky Catthoor. A hybrid design-time/run-time scheduling flow to minimise the reconfiguration overhead of FPGAs. *Microprocessors and Microsystems*, 28(5-6):291–301, August 2004.

[tex] TextPad - the text editor for Windows. http://www.textpad.com.

[TV02] Jim Torresen and Knut Arne Vinger. High performance computing by context swithcing reconfigurable logic. In Krzysztof Amborski and Hermann Meuth, editors, *In proc. of 16th European Simulation Multiconference (ESM-2002)*, pages 207–210, Darmstadt, Germany, Juni 2002. SCS Europe.

[xil] Xilinx: The Programmable Logic Company. http://www.xilinx.com.

Tillegg C

Kode for systemet

C.1 Programvare

C.1.1 benera.c

```
1  /*
2   * Definisjon av funksjonene fra hwlib.h som tar seg av
3   * kommunikasjon med den rekonfigurerbare maskinvaren over
4   * pci-bussen ved hjelp av FUSE-biblioteket.
5   */
6
7  #include <dimesdl.h>
8  #include "hwlib.h"
9  #include "utils.h"
10
11 #define CARDNR 1
12 #define MODULENR 0
13 #define DEVICENR 2
14 #define FREQ 40
15
16 LOCATE_HANDLE hLocate;
17 DIME_HANDLE hCard;
18 DIME_DMAHANDLE hDMA;
19 double afreq;
20
21
22 void hw_start() {
23
24   FILE *fp;
25   reconfrem.tv_sec = reconfrem.tv_nsec = 0;
26   reconfreq.tv_sec = 0;
27   fp = fopen("timing.cfg","r");
28   if (fp == NULL) { fprintf(logstream, "Warning : Unable to open timing-
         config file, using default 1ms. \n");fflush(logstream); reconfreq.
         tv_nsec = 1E8; }
29   else {
30     fscanf(fp,"%ld", &reconfreq.tv_nsec);
31     fclose(fp);
32   }
33
34   hLocate = DIME_LocateCard(dlPCI, mbtALL, NULL, dldrDEFAULT, dlDEFAULT);
```

```
35    if (hLocate == NULL) fprintf(stderrstream, "ERROR : DIME_LocateCard failed
          \n");
36    else {
37      hCard    = DIME_OpenCard(hLocate, CARDNR, dccOPEN_DEFAULT);
38      if (hCard == NULL) fprintf(stderrstream, "ERROR : DIME_OpenCard failed\n
            ");
39      else {
40        hDMA     = DIME_DMAOpen(hCard,1,0);
41        if (hDMA == NULL) fprintf(stderrstream, "ERROR : DIME_DMAOpen failed\n
              ");
42        else {
43          DIME_SetOscillatorFrequency(hCard, 2, 40, &afreq);
44          if (reconfig_card == reconfig_card_yes) {
45            DIME_SetOscillatorFrequency(hCard, 2, 40, &afreq);
46            DIME_ConfigSetBitsFilenameAndConfig(hCard, MODULENR, DEVICENR,
                  bitfilename, 0, NULL, 0);
47            DIME_CardResetControl (hCard, drONBOARDFPGA, drENABLE, 0);
48            DIME_CardResetControl (hCard, drINTERFACE,   drTOGGLE, 0);
49            DIME_CardResetControl (hCard, drONBOARDFPGA, drDISABLE, 0);
50            printf(" Initial configuration done\n");fflush(stdout);
51          } else { printf("Unable to set oscillator freq.\n"); fflush(stdout);
                }
52        }
53      }
54    }
55  }
56  void hw_stop() {
57    DIME_DMAClose(hCard,hDMA,ddmaCLOSEWAITFORFINISH);
58    DIME_CloseCard(hCard);
59    DIME_CloseLocate(hLocate);
60  };
61
62  void configure(task_t *task) {
63    struct timespec t;
64    //  int i;
65    DWORD command[3] =
66      {Command_reconfig,
67       task->slotnr,
68       (task->hwmod->streamsize[task->slotnr])/4};
69    clock_gettime(CLOCK_REALTIME,&t);
70    fprintf(logstream,
71            "LOG : %ld.%09ld : StartReconf task %d module %d slot %d\n",
72            t.tv_sec, t.tv_nsec, task->id, task->hwmod->id,task->slotnr);
                fflush(logstream);
73    /* Reconfig */
74    //printf("DEBUG : calling write with %ld, %ld, %ld : ",command[0], command
          [1], command[2]);
75    //for (i = 0; i < command[2]; i++) printf("0x%X ",((int*)task->hwmod->
          bitstream[command[1]])[i]); printf("\n");
76    DIME_DMAWrite(hCard, hDMA, command, 0, 3, ddmaBLOCKING);
77    DIME_DMAWrite(hCard, hDMA, task->hwmod->bitstream[command[1]], 0, command
          [2], ddmaBLOCKING);
78    /* In this work all reconfigstreams is of a fixed, small and
79     * ignorable lenght. Using a simple static delay instead */
80    //printf("DEBUG : calling read\n");
81    DIME_DMARead (hCard, hDMA, command, 0, 1, ddmaBLOCKING);
82    nanosleep(&reconfreq,&reconfrem);
83    //printf("DEBUG : read exited with 0x%lX\n",command[0]);
84
85
86    /*    nanosleep(&req,&rem); /\* Simulate *\/ */
87
```

```
88    clock_gettime(CLOCK_REALTIME,&t);
89    fprintf(logstream,
90            "LOG : %ld.%09ld : EndReconf    task %d module %d slot %d\n",
91            t.tv_sec, t.tv_nsec, task->id, task->hwmod->id,task->slotnr);
                  fflush(logstream);
92  }
93  void startTask(task_t *task) {
94
95    struct timespec t;
96    //  int i;
97    DWORD command[3] = {Command_loadTask, task->slotnr, task->taskdatasize};
98    //  slotSet(task->slotnr,&status,0); // Stub-stash for sim. Not needed
          when done.
99    moduleslots[task->slotnr].inuse = 1;
100
101   clock_gettime(CLOCK_REALTIME,&t);
102   fprintf(logstream,
103           "LOG : %ld.%09ld : StartTask    task %d module %d slot %d\n",
104           t.tv_sec, t.tv_nsec, task->id, task->hwmod->id,task->slotnr);
                  fflush(logstream);
105
106   //printf("DEBUG : calling write with %ld, %ld, %ld : ",command[0],command
          [1],command[2]);fflush(stdout);
107   //for (i = 0; i < command[2]; i++) printf("%d ",((int*)task->taskdata)[i]);
          printf("\n");
108   DIME_DMAWrite(hCard, hDMA, command, 0, 3, ddmaBLOCKING);
109   DIME_DMAWrite(hCard, hDMA, task->taskdata, 0, command[2], ddmaBLOCKING);
110   //printf("DEBUG : calling read\n");
111   DIME_DMARead (hCard, hDMA, command, 0, 1, ddmaBLOCKING);
112   //printf("DEBUG : read exited with 0x%lX\n",command[0]);
113
114 }
115
116 void completeTask(task_t *task) {
117   struct timespec t;
118   DWORD command[2] = {Command_getResult, task->slotnr};
119
120   //printf("DEBUG : Getting result from task %d\n",task->id);
121   //printf("DEBUG : Calling write with %ld, %ld\n",command[0], command[1]);
122   DIME_DMAWrite(hCard, hDMA, command, 0, 2, ddmaBLOCKING);
123   //printf("DEBUG : Calling read\n");
124   DIME_DMARead (hCard, hDMA, task->result, 0, task->resultsize, ddmaBLOCKING
          );
125   //printf("DEBUG : read exited with 0x%X\n",*(int*)task->result);
126
127   task->completed = 1;
128   moduleslots[task->slotnr].inuse = 0;
129
130   clock_gettime(CLOCK_REALTIME,&t);
131   fprintf(logstream,
132           "LOG : %ld.%09ld : EndTask      task %d module %d slot %d\n",
133           t.tv_sec, t.tv_nsec, task->id, task->hwmod->id,task->slotnr);
                  fflush(logstream);
134 }
135
136
137
138 status_t getStatus() {
139   DWORD status = Command_getStatus;
140   //  printf("DEBUG : getting status : ");fflush(stdout);
141   DIME_DMAWrite(hCard, hDMA, &status, 0, 1, ddmaBLOCKING);
142   // printf("DEBUG : (status) calling read\n");
```

```
143   DIME_DMARead (hCard, hDMA, &status, 0, 1, ddmaBLOCKING);
144   //  printf("0x%X\n",status);
145   return status;
146 }
```

C.1.2 bensim.c

```
1  /*
2   * Definisjon av funksjonene fra hwlib.h for kommunikasjon med den
3   * rekonfigurerbare maskinvare.
4   * forskjellen mellom funksjonene definert her og de som er
5   * definert i benera.c, er at disse *IKKE* kommuniserer med maskinvaren,
6   * men istede oppfører seg som om de gjorde det. Dette gjør at
7   * disse funksjonenen kan benyttes til å lage et system som simulerer
8   * maskinvaren slik at en kan teste ut funksjonaliteten i programvare-
9   * systemet uten maskinvaren.
10  */
11 #include <time.h>
12 #include <pthread.h>
13 #include <string.h>
14 #include "hwlib.h"
15 #include "hwglobals.h"
16 #include "utils.h"
17
18 status_t sysstatus;
19 pthread_mutex_t mut;
20
21 struct timespec execdelay;
22
23
24
25 /* HACK */
26 struct timespec execdelaytime(task_t *task) {
27   struct timespec res;
28   res.tv_sec = 0;
29   if (strncmp(task->hwmod->hwmodname,"Sum",3) == 0) res.tv_nsec = (long int)
         ((float)task->taskdatasize * 7.695685091);
30   else res.tv_nsec = 24315;
31   return res;
32 }
33
34
35 typedef struct {
36   slotnr_t slotnr;
37   struct timespec req;
38 } sleeperargs_t;
39
40 void slotSet(slotnr_t slotnr, status_t *status, int flag) {
41   pthread_mutex_lock(&mut);
42   if (flag == 0) *status &= ~(1 << (slotnr  + donebits));
43   else *status |= (1 << (slotnr + donebits));
44   pthread_mutex_unlock(&mut);
45 }
46
47 // Debug threadfunction used to wait for a while before marking task as done
48 void *sleeper(void *_args) {
49   if (_args != NULL) {
50     sleeperargs_t args = *(sleeperargs_t*)_args;
51     struct timespec rem;
52     nanosleep(&args.req,&rem);
53     slotSet(args.slotnr,&sysstatus,1);
```

```
54    }
55    free(_args);
56    pthread_exit(NULL);
57  }
58
59
60  void hw_start() {
61    FILE *fp;
62    reconfrem.tv_sec = reconfrem.tv_nsec = 0;
63    reconfreq.tv_sec = 0;
64    fp = fopen("timing.cfg","r");
65    if (fp == NULL) { fprintf(logstream, "Warning : Unable to open timing-
          config file, using default 1ms. \n");fflush(logstream); reconfreq.
          tv_nsec = 1E8; }
66    else {
67      fscanf(fp,"%ld", &reconfreq.tv_nsec);
68      fclose(fp);
69    }
70    pthread_mutex_init(&mut, NULL);
71  }
72
73  void hw_stop() { /* don't do anything */ }
74
75  void configure(task_t *task) {
76    struct timespec t;
77    clock_gettime(CLOCK_REALTIME, &t);
78    fprintf(logstream,
79            "LOG : %ld.%09ld : StartReconf task %d module %d slot %d\n",
80            t.tv_sec, t.tv_nsec, task->id, task->hwmod->id,task->slotnr);
                fflush(logstream);
81  //  nanosleep(&reconfreq,&reconfrem);
82    clock_gettime(CLOCK_REALTIME,&t);
83    fprintf(logstream,
84            "LOG : %ld.%09ld : EndReconf   task %d module %d slot %d\n",
85            t.tv_sec, t.tv_nsec, task->id, task->hwmod->id,task->slotnr);
                fflush(logstream);
86  }
87
88  void startTask(task_t *task) {
89      sleeperargs_t *args = (sleeperargs_t*)calloc(1,sizeof(sleeperargs_t));
90      struct timespec t;
91
92      args->req = execdelaytime(task);
93      args->slotnr = task->slotnr;
94      int res = pthread_create(&moduleslots[task->slotnr].sleeper,NULL,sleeper
            ,args);
95      if (res != 0) {
96        fprintf(stderrstream, "ERROR: unable to create sleeper, ending task
              now\n");
97        fflush(stderrstream);
98        slotSet(task->slotnr, &status, 1);
99      } else {
100       pthread_detach(moduleslots[task->slotnr].sleeper);
101     }
102
103    clock_gettime(CLOCK_REALTIME,&t);
104    fprintf(logstream,
105            "LOG : %ld.%09ld : StartTask   task %d module %d slot %d\n",
106            t.tv_sec, t.tv_nsec, task->id, task->hwmod->id,task->slotnr);
                fflush(logstream);
107 }
108
```

```
109 void completeTask(task_t *task) {
110   struct timespec t;
111   task->completed = 1;
112   moduleslots[task->slotnr].inuse = 0;
113
114   clock_gettime(CLOCK_REALTIME,&t);
115   fprintf(logstream,
116           "LOG : %ld.%09ld : EndTask      task %d module %d slot %d\n",
117           t.tv_sec, t.tv_nsec, task->id, task->hwmod->id,task->slotnr);
                fflush(logstream);
118 }
119 status_t getStatus() {
120   return sysstatus;
121 }
```

C.1.3 fas.c

```
1  /*
2   * Alternativ implementasjon av funksjonene fra hwlib.h som
3   * definerer planleggeren. Benytter FAS - First Available Slot.
4   */
5  #include <stdlib.h>
6  #include "hwtypes.h"
7  #include "hwglobals.h"
8
9  /* First Available Slot
10  * Simply returns the first available slot.
11  * No history of module-usage. */
12
13 void scheduler_init() { /* don't do anything */ }
14 void scheduler_register(slotnr_t slotid) { /* don't do anything */ }
15 slotnr_t scheduler_findnext(hwmodid_t id) {
16   int found = 0;
17   slotnr_t i = 0;
18   while ((i != numberofmoduleslots) && !found) {
19     if (moduleslots[i].inuse == 0) found = 1;
20     else i++;
21   }
22   if (found == 0) return -1;
23   else return i;
24 }
```

C.1.4 gentasks.c

```
1  /*
2   * Støtteprogram som genererer en liste av oppgaver som kan sendes
3   * inn til testsystemet.
4   */
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <time.h>
8
9  int main(int argc, char *argv[]) {
10   struct timespec foo;
11   int numberofmodules,*distr,*maxargs,*minargs,*res,numberoftasks,
         modnameindex,maxargsindex,minargsindex,resindex,distrindex,taskindex,
         totaldistr,count;
12
13
14
```

```
15
16      clock_gettime(CLOCK_REALTIME,&foo);
17
18      srandom((unsigned int)foo.tv_nsec);
19
20      if (argc <= 3) {
21        fprintf(stderr,"needs more arguments\n");
22        fprintf(stderr,"Syntax : ./gentasks <numberofmodules> <mod0name> .. <
              modn-1name> <mod0maxargs> .. <modn-1maxargs> <mod0minargs> .. <modn
              -1minargs> <mod0res> .. <modn-1res> <mod0distr> .. <modn-1distr> <
              numberoftasks>\n");
23        exit(-1);
24      }
25      numberofmodules = atoi(argv[1]);
26      distr = (int*)calloc(numberofmodules, sizeof(int));
27      maxargs  = (int*)calloc(numberofmodules, sizeof(int));
28      minargs  = (int*)calloc(numberofmodules, sizeof(int));
29      res      = (int*)calloc(numberofmodules, sizeof(int));
30      modnameindex = 2;
31      maxargsindex = 2 +  numberofmodules;
32      minargsindex = 2 + (numberofmodules*2);
33      resindex     = 2 + (numberofmodules*3);
34      distrindex   = 2 + (numberofmodules*4);
35      taskindex    = 2 + (numberofmodules*5);
36      totaldistr   = 0;
37
38      // read max number of args.
39      for (count = 0; count < numberofmodules; count++) {
40        maxargs[count] = atoi(argv[maxargsindex+count]);
41      }
42
43      // read min number of args.
44      for (count = 0; count < numberofmodules; count++) {
45        minargs[count] = atoi(argv[minargsindex+count]);
46      }
47
48      // read number of res.
49      for (count = 0; count < numberofmodules; count++) {
50        res[count] = atoi(argv[resindex+count]);
51      }
52
53
54
55      // read distributions
56      for (count = 0; count < numberofmodules; count++) {
57        int i = atoi(argv[distrindex+count]);
58        distr[count] = i;
59        totaldistr+=i;
60      }
61
62      // read numberoftasks
63      numberoftasks = atoi(argv[taskindex]);
64      printf("%d\n",numberoftasks);
65      for (count = 0; count < numberoftasks; count++) {
66        int i = random() % totaldistr;
67        int index = 0;
68        int found = 0;
69        int args;
70        while (!found) {
71          if (i < distr[index]) {
72            //          printf("%s\n",argv[2+index]);
```

```
73          args = (random()%(1+maxargs[index]-minargs[index])) + minargs[index
                ];
74          printf("%s %d %d\n",argv[modnameindex+index],args, res[index]);
75          found=1;
76        } else {
77          i -= distr[index]; index++;
78        }
79      }
80    }
81    return 0;
82  }
```

C.1.5 hwbench.c

```
1   /*
2    * Test-program som måler tiden det tar og rekonfigurere maskinvaren
3    * i tillegg til hvor raskt kommunikasjonen med maskinvare-systemet
4    * vi har laget er
5    */
6
7   #include <stdio.h>
8   #include <dimesdl.h>
9   #include <inttypes.h>
10  #include <time.h>
11
12  #define BitFileName "top.bit"
13
14  #define CARDNR 1
15  #define MODULENR 0
16  #define DEVICENR 2
17
18
19  const uint32_t hw_getstatus = 0;
20  const uint32_t hw_reconf    = 1;
21  const uint32_t hw_load      = 2;
22  const uint32_t hw_getres    = 3;
23  const uint32_t hw_start     = 4;
24  const uint32_t hw_stop      = 5;
25
26  double clockdiff(struct timespec *start, struct timespec *stop) {
27    double res;
28    res = (double)stop->tv_sec - start->tv_sec;
29    res += (((double)stop->tv_nsec  / (double)1E9) -
30            ((double)start->tv_nsec / (double)1E9));
31    return res;
32  }
33
34  int main(int argc, char *argv[]) {
35    LOCATE_HANDLE hLocate;
36    DIME_HANDLE hCard;
37    DIME_DMAHANDLE hDMA;
38    double afreq;
39    char c;
40
41    DWORD *confdata, *taskdata;
42    DWORD single = random();
43
44    struct timespec start,stop;
45    double totalreconftime;
46    double timetosendone, totaltimetosendone;
47    double totalsendtime;
```

```
48    double totalsumtime;
49    double totalemmtime;
50    int antsum, antemm;
51    int count,reconfs,props,words;
52
53
54    if (argc < 4) {
55      reconfs = 10;
56      props = 10;
57      words = 10;
58      printf("more args, using defaults (reconfs=%d, props=%d, words=%d)\n",
            reconfs,props, words); }
59    else {
60      reconfs = atoi(argv[1]);
61      props = atoi(argv[2]);
62      words = atoi(argv[3]);
63    }
64
65    if (hLocate = DIME_LocateCard(dlPCI, mbtALL, NULL, dldrDEFAULT, dlDEFAULT)
        ) {
66      printf("locatecard ok\n");
67      if (hCard = DIME_OpenCard(hLocate, CARDNR,dccOPEN_DEFAULT)) {
68        printf("opencard ok\n");
69        if (hDMA = DIME_DMAOpen(hCard,1,0)) {
70          printf("dmaopen ok\n");
71          if (!DIME_JTAGControl(hCard, djtagCONFIGSPEED, djtagMAXSPEED100)) {
72            printf("jtagcontrol ok\n");
73            DIME_SetOscillatorFrequency(hCard, 2, 40, &afreq);
74            printf("Card configured at %ld MHz.\n",afreq);
75
76
77
78            // Timing av reconfigurering.
79            clock_gettime(CLOCK_REALTIME, &start);
80            for (count = 0; count < reconfs; count++) {
81              DIME_ConfigSetBitsFilenameAndConfig
82                (hCard, MODULENR, DEVICENR, BitFileName, 0, NULL, 0);
83              //       printf("r");
84            } printf("\n");
85            clock_gettime(CLOCK_REALTIME, &stop);
86            totalreconftime = clockdiff(&start, &stop);
87            // totalreconftime = stop.tv_sec - start.tv_sec;
88            // totalreconftime +=
89            //   (((double)stop.tv_nsec  / (double)1E9) -
90            //    ((double)start.tv_nsec / (double)1E9));
91
92
93            DIME_CardResetControl (hCard, drONBOARDFPGA, drENABLE, 0);
94            DIME_CardResetControl (hCard, drINTERFACE,   drTOGGLE, 0);
95            DIME_CardResetControl (hCard, drONBOARDFPGA, drDISABLE, 0);
96
97
98            // Reconfig module.
99
100           // initier datastrøm.
101           confdata = (DWORD*)calloc(words+3, sizeof(DWORD));
102           confdata[0] = hw_reconf; // reconf
103           confdata[1] = 0; // slotnr
104           confdata[2] = props;  // streamsize
105           confdata[3] = 0; // modul nr 0 (sum)
106           for (count = 1; count < props; count++) { confdata[count+3] =
                random(); }
```

```
107
108             // Measuring prob.delay by sending one and one word
109             totaltimetosendone = 0.0;
110             for (count = 0; count < props+3;count++) {
111               clock_gettime(CLOCK_REALTIME, &start);
112               DIME_DMAWrite(hCard,hDMA,&confdata[count], 0, 1, ddmaBLOCKING);
113               clock_gettime(CLOCK_REALTIME, &stop);
114               timetosendone = clockdiff(&start, &stop);
115 /*                timetosendone = stop.tv_sec - start.tv_sec; */
116 /*                timetosendone += */
117 /*                  (((double)stop.tv_nsec  / (double)1E9) - */
118 /*                   ((double)start.tv_nsec / (double)1E9)); */
119               totaltimetosendone += timetosendone;
120             }
121             DIME_DMARead(hCard, hDMA, confdata,0, 1,                    ddmaBLOCKING
                  );
122
123
124             // Send Task
125
126             // Initier datastrøm
127             taskdata = (DWORD*)calloc(words+3, sizeof(DWORD));
128             taskdata[0] = hw_load; // load task
129             taskdata[1] = 0;       // slot nr 0
130             taskdata[2] = words;   // number of words to send
131             for (count = 0; count < words; count ++) { taskdata[count+3] =
                  random(); }
132             DIME_DMAWrite(hCard, hDMA, taskdata, 0, 3, ddmaBLOCKING);
133
134             // Measure bandwidth
135             clock_gettime(CLOCK_REALTIME, &start);
136             DIME_DMAWrite(hCard, hDMA, &confdata[3], 0, words, ddmaBLOCKING);
137             clock_gettime(CLOCK_REALTIME, &stop);
138             totalsendtime = clockdiff(&start,&stop);
139 /*                totalsendtime = stop.tv_sec - start.tv_sec; */
140 /*                totalsendtime += */
141 /*                  (((double)stop.tv_nsec  / (double)1E9) - */
142 /*                   ((double)start.tv_nsec / (double)1E9)); */
143
144             DIME_DMARead(hCard,hDMA, taskdata, 0, 1, ddmaBLOCKING);
145
146
147             //           for (count = 0; count < antsum; count++) {
148
149
150
151
152           } else printf("DIME_JTAGControl failed\n");
153           DIME_DMAClose(hCard, hDMA, ddmaCLOSEWAITFORFINISH);
154         } else printf("DIME_DMAOpen failed\n");
155         DIME_CloseCard(hCard);
156       } else printf("DIME_OpenCard failed\n");
157       DIME_CloseLocate(hLocate);
158     } else printf("DIME_LocateCard failed\n");
159
160     printf("Total time spent reconfiguring : %lf    avg : %lf\n",
          totalreconftime, totalreconftime/reconfs);
161     printf("Propagationdelay                : %lf\n",totaltimetosendone/props);
162     printf("Total time spend sending        : %lf    speed : %lf Bps (%lf Mbps)\
          n",totalsendtime, ((words+2)*4)/totalsendtime, ((words+2)*4)/(
          totalsendtime*(1<<20)));
163
```

```
164
165
166
167     return 0;
168 }
```

C.1.6 hwglobals.c

```
1  /*
2   * Implementasjon av de globale variablene fra hwglobals.h
3   */
4  #include "hwglobals.h"
5  #include "tree.h"
6  #include "list.h"
7
8  list_t tasklist;
9  tree_t moduletree;
10 SlotInfo_t *moduleslots;
11 #ifdef use_modslotmutex
12 pthread_mutex_t modslotmutex = PTHREAD_MUTEX_INITIALIZER;
13 #endif
14 pthread_t deamon;
15 int killdeamon;
16 status_t status = 0;
17
18
19 int       allow_duplicates    = allow_duplicates_no;
20 slotnr_t  numberofmoduleslots = 4;
21 int       how_sched_update = how_sched_update_allsimilar;
22 int       sched_register_on_forwarded = sched_register_on_forwarded_yes;
23 int       use_taskforwarding = use_taskforwarding_no;
24 int       maxforwarddepth = default_maxforwarddepth;
25 int       reconfig_card = reconfig_card_yes;
26 char      bitfilename[20] = defaultbitfilename;
```

C.1.7 hwglobals.h

```
1  /*
2   * Header-fil for alle globale variabler en kan ha bruk for utenfor hwlib
3   */
4
5  #ifndef _fcache_hwglobals_h_
6  #define _fcache_hwglobals_h_
7
8  #include <time.h>
9
10 #include "hwtypes.h"
11 #include "tree.h"
12 #include "list.h"
13 #include "limits.h"
14
15
16
17 extern slotnr_t numberofmoduleslots; /* Number of moduleslots used */
18 extern SlotInfo_t *moduleslots;      /* Array of moduleslots */
19 extern tree_t moduletree;            /* Tree structure containging all
       modules that has been loaded into the system */
20 extern list_t tasklist;              /* List of all tasks that is waiting to
        be executed */
21 #ifdef use_modslotmutex
```

```
22 #include <pthreads.h>
23 extern pthread_mutex_t modslotmutex; /* Mutex to ensure that only one can
       access the moduleslots at the time */
24 #endif
25 extern pthread_t deamon;              /* Threadobject for the deamon */
26 extern int killdeamon;                /* Flag indicating to the deamon that
       it should shut down */
27 extern status_t status;               /* statusflags, used to store the
       status that it otherwise would get from hardware (debug) */
28
29
30 /* Allow to seperate moduleslots to be configured with the same module */
31 extern int allow_duplicates;
32 #define allow_duplicates_no 0
33 #define allow_duplicates_yes 1
34
35 /* If allow_duplicates_yes, and a moduleslot is used, should only that
       moduleslot
36  * be flagged, or should all moduleslots containting the same module be
       flagged? */
37 extern int how_sched_update;
38 #define how_sched_update_onlythis 0
39 #define how_sched_update_allsimilar 1
40
41 extern int sched_register_on_forwarded;
42 #define sched_register_on_forwarded_no 0
43 #define sched_register_on_forwarded_yes 1
44
45 /**/
46 extern int use_taskforwarding;
47 #define use_taskforwarding_no 0
48 #define use_taskforwarding_yes 1
49
50 extern int maxforwarddepth;
51 #define default_maxforwarddepth 10
52
53 extern int reconfig_card;
54 #define reconfig_card_no 0
55 #define reconfig_card_yes 1
56
57 extern char bitfilename[20];
58 #define defaultbitfilename "top.bit"
59
60
61 #endif
```

C.1.8 hwlib.c

```
1  /*
2   * Implementasjoner av funksjoner fra hwlib.h som ikke har med
3   * kommunikasjon eller planleggeren å gjøre.
4   */
5
6  #include <time.h>
7  #include "hwlib.h"
8  #include "utils.h"
9  #include "hwglobals.h"
10
11 #ifdef use_modslotmutex
12 #include "mututils.h"
13 #else
```

```
14 #define LockIt(foo)
15 #define UnLockIt(foo)
16 #endif
17
18
19 struct timespec reconfreq, reconfrem; /* used to simulate correct reconf-
       delay */
20
21 int freeslots();
22 int numofduplicates(hwmodid_t id);
23 slotnr_t findavailableslots(hwmodid_t id);
24 void *deamonmain(void *args);
25
26 void hwlib_init(int argc, char *argv[]) {
27   setErrStream(stderr);  /* Specifies which errorstream that should be used
         by default          */
28   setOutStream(stdout);  /* Specifies which outputstream that should be used
         by default          */
29   setLogStream(stdout);  /* Specifies which logstream that should be used by
         default             */
30   readArgs(argc,argv); /* Read arguments              */
31   initSlots();           /* Initialize the moduleslots */
32   listInit(&tasklist);   /* Initialize the task queue  */
33   treeInit(&moduletree); /* Initialize the moduletree  */
34   hw_start();            /* Starts up the hardware     */
35   scheduler_init();      /* Initialize the scheduler   */
36   startDeamon();         /* Starts the deamon thread   */
37 }
38
39 void hwlib_exit() {
40   stopDeamon(); /* Stop the deamon thread */
41   hw_stop();    /* Releases the hardware-resources used */
42 }
43
44 void initSlots() {
45   int count;
46   moduleslots = (SlotInfo_t*)calloc(numberofmoduleslots,sizeof(SlotInfo_t));
47   for (count = 0;
48        count < numberofmoduleslots;
49        count++) {
50     moduleslots[count].id = nohwmod;
51     moduleslots[count].inuse = 0;
52   }
53 }
54
55 /* Non-destructive functions, don't need mutex on this level */
56 int freeslots() {
57   int res = 0,count;
58   for (count = 0; count < numberofmoduleslots;count++)
59     if (moduleslots[count].inuse == 0) res++;
60   return res;
61 }
62 int numofduplicates(hwmodid_t id) {
63   int res = 0,count;
64   for (count=0;count<numberofmoduleslots;count++)
65     if (moduleslots[count].id == id) res++;
66   return res;
67 }
68
69 /* Looks for availiable slots
70  * (not assinged or assigned to this hwmod,
71  * depending on allow_duplicates (-1 if none found)
```

```
72   */
73  slotnr_t findavailableslot(hwmodid_t id) {
74    slotnr_t res = noslot, count;
75    SlotInfo_t *slot;
76    for (count = 0; count < numberofmoduleslots; count++) {
77      slot = &(moduleslots[count]);
78
79      if (allow_duplicates == allow_duplicates_yes) {
80        if (slot->inuse == 0) {
81          if ((slot->id == nohwmod) || (slot->id == id)) {
82            res = count;
83            if (slot->id == id)
84              count = numberofmoduleslots; // end search now
85          }
86        }
87
88      } else {
89        if (slot->id == id) {
90          if (slot->inuse == 1) // Ooops.. -1
91            res = noslot;
92          else // Perfect, use this
93            res = count;
94          count = numberofmoduleslots; // end search now
95        } else {
96          if ((slot->inuse == 0)&&(slot->id == nohwmod)) res = count;
97        }
98
99      }
100   }
101   return res;
102 }
103 void addTask(task_t *task) {
104   /* Keep trying to lock until sucessfull. */
105   struct timespec t;
106   clock_gettime(CLOCK_REALTIME, &t);
107   fprintf(logstream,
108           "LOG : %ld.%09ld : AddTask      task %d module %d slot %d\n",
109           t.tv_sec,t.tv_nsec,task->id,task->hwmod->id,-1); fflush(logstream)
                ;
110   listAdd(&tasklist, task);
111 }
112 int check() {
113   int res = 0;
114   if (!listEmpty(&tasklist)) {
115
116     if (use_taskforwarding == use_taskforwarding_yes) {
117       /* First check if any preconfigured modules can be used to issue any
              waiting tasks */
118       slotnr_t count;
119       for (count = 0; count < numberofmoduleslots; count++) {
120         //        printf("debug: checking if we can use moduleslot %d\n",
              count);fflush(stdout);
121         if ((moduleslots[count].id != nohwmod) && (moduleslots[count].inuse
              == 0)) {
122           //          printf("debug: checking if there is any tasks needing
              hwmodnr %d\n",moduleslots[count].id);fflush(stdout);
123           //        char c = getchar(); //Wait
124           task_t *task = (task_t*)listFind(&tasklist,(void*)moduleslots[
              count].id,maxforwarddepth,isittaskwithmoduleid);
125           //          printf("debug: did we find a task? task = 0x%X\n",(int
              )task);fflush(stdout);
126           if (task != NULL) {
```

```
127             //              printf("debug: forwarding task nr task %d\n",task
                    ->id);fflush(stdout);
128             listRemove(&tasklist,(void*)moduleslots[count].id,
                    isittaskwithmoduleid);
129             task->slotnr = count;
130             moduleslots[count].inuse = 1;
131             moduleslots[count].task = task;
132             if (sched_register_on_forwarded ==
                    sched_register_on_forwarded_yes)
133               scheduler_register(count);
134             startTask(task);
135             res = 1;
136           }
137         }
138       }
139     }
140
141     if (res == 0) {
142
143       task_t *task = (task_t*)listPeek(&tasklist);
144       hwmodid_t modid = task->hwmod->id;
145       slotnr_t slotnr = findavailableslot(modid);
146       if (slotnr == noslot) {
147
148         /* Ingen helt ledige / perfekte slots tilgjengelige
149          * Ser om scheduler kan finne noen som passer */
150         if (allow_duplicates == allow_duplicates_yes) {
151           slotnr = scheduler_findnext(modid);
152         } else {
153           int duplicates = numofduplicates(modid);
154           if (duplicates == 0) slotnr = scheduler_findnext(modid);
155         }
156       }
157
158       if (slotnr != noslot) { /* Har funnet en slot som kan brukes. */
159
160         listPop(&tasklist);
161
162         task->slotnr = slotnr;
163
164         /* Skjekk om det er nødvendig å reconfigurere slot'n */
165         if (moduleslots[slotnr].id != modid) {
166
167           /* Reconfigurate slot */
168           configure(task);
169
170         }
171
172         moduleslots[slotnr].id = modid;
173         moduleslots[slotnr].inuse = 1;
174         moduleslots[slotnr].task = task;
175
176         scheduler_register(slotnr);
177
178         /* tell caller that a task was started */
179         res = 1;
180
181         /* Start task */
182         startTask(task);
183
184       }
185     }
```

```
186     }
187     return res;
188   }
189
190
191
192
193
194   int slotDone(slotnr_t slotnr, status_t status) {
195     return (status &(1<<(slotnr  + donebits)));
196   }
197
198   void startDeamon() {
199     int res;
200     killdeamon = 0;
201     res = pthread_create(&deamon, NULL, deamonmain, NULL);
202     if (res != 0)
203       fprintf(stderrstream,
204               "Error, cannot start deamonthread (ErrorCode:%d)\n",res);
205   }
206
207   void stopDeamon() {
208     killdeamon = 1;
209     pthread_join(deamon,NULL);
210   }
211
212
213   void *deamonmain(void *args) {
214     //  printf("started deamonmain()\n");
215     struct timespec req,rem;
216
217     req.tv_sec = 0;
218     req.tv_nsec = 1000000; // 1ms.
219
220     while (!killdeamon) {
221       status_t status;
222       slotnr_t i;
223       nanosleep(&req,&rem);     /* sleep for a bit */
224       status = getStatus();
225       LockIt(&modslotmutex);
226       for (i=0;i<numberofmoduleslots;i++) {      /* Look for competed tasks */
227         if ((slotDone(i, status)) && (moduleslots[i].inuse == 1)) {
228           completeTask(moduleslots[i].task);
229         }
230       }
231       while (check());      /* start as many tasks as possible */
232       UnLockIt(&modslotmutex);
233     }
234     //  printf("stopping deamonmain\n");
235     return NULL;
236   }
```

C.1.9 hwlib.h

```
1   /*
2    * Header-fil for hovedsystemet
3    * Definerer prototypene til funksjonenen som skal være tilgjengelig
4    * for applikasjonsprogrammet.
5    */
6
7   #ifndef _fcache_hwlib_h_
```

```
8  #define _fcache_hwlib_h_
9
10 #include <stdlib.h>
11 #include <limits.h>
12 #include <stdio.h>
13 #include <pthread.h>
14
15 #include "list.h"
16 #include "tree.h"
17 #include "hwtypes.h"
18 #include "hwglobals.h"
19
20 #define donebits 0
21
22 #define Command_getStatus 0
23 #define Command_reconfig  1
24 #define Command_loadTask  2
25 #define Command_getResult 3
26 #define Command_startTask 4
27 #define Command_stopTask  5
28
29 extern struct timespec reconfreq, reconfrem; /* used to simulate correct
       reconf-delay */
30
31 void hwlib_init(int argc, char *argv[]); /* Initialization funciton that
       needs to be called by the calling program */
32 void hwlib_exit();                       /* Specifies how the system should
       shutdown */
33
34 void hw_start();
35 void hw_stop();
36
37 void initSlots();           /* Initialize the software-representation of the
        moduleslots */
38 void addTask(task_t *task); /* Add a initialized task object to the task-
       queue */
39 int  check();               /* Checks if it is possible to start a new task
       */
40
41 void configure(task_t *task);                    /* configures the
       designated m-slot */
42 void startTask(task_t *task);                    /* start task in designated
        m-slot */
43 void completeTask(task_t *task);                 /* writes back result and
       free up resources. */
44
45 int  slotDone(slotnr_t slotnr, status_t status);          /* checks if a
       slot is done */
46 // void slotSet(slotnr_t slotnr, status_t *status, int flag); /* Sets a flag
       . (Debug) */
47 status_t  getStatus();                                    /* Returns the
       statusword from the FPGA */
48
49 /* Starting and stopping of the background deamon */
50 void startDeamon();
51 void stopDeamon();
52
53 /* Defined in lru.c */
54 void     scheduler_init();                /* Initialize the scheduler */
55 void     scheduler_register(int slotid);  /* Tells the sheduler that the
       module in slot slotid is being used */
```

```
56 slotnr_t scheduler_findnext(hwmodid_t id); /* Let the scheduler find a
       notinuse slots to use, -1 if none */
57
58 /* Defined in utils.c */
59 void readArgs(int argc, char *argv[]);                  /* Checks argc/argv
       for settings that should be aplied */
60 int loadLibFile(char *fname);                           /* Reads a libfile
       and adds every module defined in it */
61 int loadModuleFromLibFile(char *modname, char *fname); /* Reads a libfile
       and adds a spesific module */
62 int unloadModule(char *modname);                        /* Unloads a module
       */
63
64 #endif
```

g

C.1.10 hwtypes.h

```
1  /*
2   * Definisjoner av alle strukturer i systemet. Grunnen til at de er
3   * deffinert her og ikke i hwlib.h, er at en da ville trengt å
4   * linket inn FUSE-biblioteket, noe som ikke alltid er ønskelig for
5   * en støtteapplikasjon som f.eks. logstats
6   */
7  #ifndef _fcache_hwtypes_h_
8  #define _fcache_hwtypes_h_
9  #define HWMODNAMELENGTH 24
10 #include <pthread.h>
11 typedef signed   int hwmodid_t;       /* Specifies what datatype should be
       used for hardwaremodule id */
12 typedef signed   int hwlibtaskid_t;   /* Specifies what datatype should be
       used for task id */
13 typedef signed   int slotnr_t;        /* Specifies what datatype should be
       used for slotnr */
14 #define nohwmod -1
15 #define notask  -1
16 #define noslot  -1
17 typedef unsigned int status_t;  /* Specifies what datatype should be used
       for statusword (debug?) */
18
19 /* Struct containg all data conserning a spesific hardwaremodule */
20 typedef struct HWModule_t {
21   char hwmodname[HWMODNAMELENGTH]; /* The name, as it is stored in the
         libfile*/
22   hwmodid_t id;                    /* Unique id */
23   void **bitstream;                /* Array of bitstreams, one for each
         moduleslot */
24   int *streamsize;                 /* Array of bitstreamsized, one for each
         moduleslot */
25 } HWModule_t;
26
27 /* Taskobject, created by the users lib-functions */
28 typedef struct task_t {
29   void *taskdata;       /* Inbuffer, contains data that is to be worked uppon
         */
30   int taskdatasize;     /* Size of inbuffer */
31   void *result;         /* Outbuffer, when task is completed result should be
         stored here */
32   int resultsize;       /* Size of Outbuffer */
33   HWModule_t *hwmod;    /* Pointer to the hardwaremodule this task uses */
```

```
34    slotnr_t slotnr;      /* Slotnumber where this task is executed */
35    int completed;        /* Flag set by deamon, indicating if this task is
          completed */
36    hwlibtaskid_t id;          /* Unique id for this task (debug ?) */
37  } task_t;
38
39  /* Software representation of a fysical slot on the hardware */
40  typedef struct SlotInfo_t {
41    hwmodid_t id;       /* the uniqueid of the hardwaremodule configured on
          this slot, -1 if none */
42    task_t *task;       /* Pointer to the taskobj running on this slot */
43    int inuse;          /* Flag indicating if this slot is active (something
          running on it */
44    pthread_t sleeper; /* Thread waiting a while then set current task
          completed (debug) */
45  } SlotInfo_t;
46
47
48  #endif
```

C.1.11 list.c

```
1   /*
2    * Implementasjon av list-strukturen definert i list.h
3    */
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include "list.h"
7
8   #ifdef use_list_mutex
9   #include "mututils.h"
10  #else
11  #define LockIt(foo)
12  #define UnLockIt(foo)
13  #endif
14
15
16  void listInit(list_t *list) {
17    list->next = list;
18    list->prev = list;
19    list->data = NULL;
20  #ifdef use_list_mutex
21    pthread_mutex_init(&(list->mutex), NULL);
22  #endif
23  }
24
25  int listPopFront(list_t *list) {
26    int res;
27    LockIt(&(list->mutex));
28    if (listEmpty(list)) res = 0;
29    else {
30      list_t *node = list->next;
31      list->next = node->next;
32      node->next->prev = list;
33      free(node);
34      res = 1;
35    }
36    UnLockIt(&(list->mutex));
37    return res;
38  }
39  int listPopBack(list_t *list) {
```

```
40    int res;
41    LockIt(&list->mutex);
42    if (listEmpty(list)) res = 0;
43    else {
44      list_t *node = list->prev;
45      list->prev = node->prev;
46      node->prev->next = list;
47      res = 1;
48    }
49    UnLockIt(&list->mutex);
50    return res;
51  }
52  int listPop(list_t *list) { return listPopFront(list); }
53
54
55  void *listPeekFront(list_t *list) {
56    void *res;
57    LockIt(&(list->mutex));
58    res = list->next->data;
59    UnLockIt(&(list->mutex));
60    return res;
61  }
62  void *listPeekBack(list_t *list) {
63    void *res;
64    LockIt(&list->mutex);
65    res = list->prev->data;
66    UnLockIt(&list->mutex);
67    return res;
68  }
69
70  void *listPeek(list_t *list) { return listPeekFront(list); }
71
72
73  void listAddFront(list_t *list, void *data) {
74    list_t *newnode, *nextnode;
75    LockIt(&(list->mutex));
76    newnode = (list_t*)malloc(sizeof(list_t));
77    nextnode = list->next;
78    newnode->data = data;
79    list->next->prev = newnode;
80    list->next = newnode;
81    newnode->prev = list;
82    newnode->next = nextnode;
83    UnLockIt(&(list->mutex));
84  }
85  void listAddBack(list_t *list, void *data) {
86    list_t *newnode,*lastnode;
87    LockIt(&(list->mutex));
88    newnode = (list_t*)malloc(sizeof(list_t));
89    lastnode = list->prev;
90    newnode->data = data;
91    list->prev->next = newnode;
92    list->prev = newnode;
93    newnode->next = list;
94    newnode->prev = lastnode;
95    UnLockIt(&(list->mutex));
96  }
97  void listAdd(list_t *list, void *data) { listAddBack(list, data); }
98
99  int listEmpty(list_t *list) {
100   if (list->next == list) return 1;  else return 0;
101 }
```

```
102 void *listFind(list_t *list, void *data, int depth, int (*isit)(void *item,
        void *data)) {
103   void *res;
104   int found = 0;
105   list_t *p = list;
106   int count = 0;
107   LockIt(&(list->mutex));
108   while ((!found) && (count < depth)) {
109     count++;
110     p = p->next;
111     if (p == list)  found = 2;
112     else if ((*isit)(p->data, data)) found = 1;
113   }
114   if (found) res = p->data;
115   else res = NULL;
116   UnLockIt(&(list->mutex));
117   return res;
118 }
119 int listRemove(list_t *list, void *data, int (*isit)(void *item, void *data)
        ) {
120   int res;
121   int found = 0;
122   list_t *p = list;
123   LockIt(&(list->mutex));
124   while (!found) {
125     p = p->next;
126     if (p == list) found = 2;
127     else if ((*isit)(p->data,data)) found = 1;
128   }
129   if (found) {
130     res = 1;
131     p->next->prev = p->prev;
132     p->prev->next = p->next;
133     free (p);
134   } else res = 0;
135   UnLockIt(&(list->mutex));
136   return res;
137 }
138
139
140
141
142 void listClear(list_t *list) {
143   list_t *p;
144   LockIt(&(list->mutex));
145   while (list != list->next) {
146     p = list->next;
147     list->next = p->next;
148     free(p);
149   }
150   UnLockIt(&(list->mutex));
151   listInit(list);
152 }
153
154 int listSize(list_t *list) {
155   list_t *p;
156   int res = 0;
157   LockIt(&list->mutex);
158   p = list->next;
159   while (p != list) {
160     res++; p = p->next;
161   }
```

```
162   UnLockIt(&list->mutex);
163   return res;
164 }
165
166
167
168
169
170 void listnodeprint(list_t *node) {
171   printf("Node : 0x%x,  Prev : 0x%x,  Next : 0x%x,  Data : 0x%x\n",
172          (unsigned int)node,
173          (unsigned int)node->prev,
174          (unsigned int)node->next,
175          (unsigned int)node->data);
176 }
177
178
179 void listPrint(list_t *list, void (*nodeprint)(list_t *node)) {
180   list_t *p = list->next;
181   if (nodeprint == NULL) nodeprint = listnodeprint;
182   (*nodeprint)(list);
183   while (p != list) {
184     (*nodeprint)(p);
185     p = p->next;
186   }
187 }
```

C.1.12 list.h

```
1  /*
2   * Headerfil for en generisk dobbel-lenket sirkulær liste.
3   */
4
5  #ifndef _circular_dlinked_list_h_
6  #define _circular_dlinked_list_h_
7
8  #ifdef use_list_mutex
9  #include <pthread.h>
10 #endif
11
12 #ifndef NULL
13 #define NULL 0
14 #endif
15
16
17 typedef struct list_t {
18   struct list_t *prev;
19   struct list_t *next;
20   void *data;
21 #ifdef use_list_mutex
22   pthread_mutex_t mutex;
23 #endif
24 } list_t;
25
26 void listInit(list_t *list);
27 int listPopFront(list_t *list);
28 int listPopBack(list_t *list);
29 int listPop(list_t *list); /* default front */
30
31 void *listPeekFront(list_t *list);
32 void *listPeekBack(list_t *list);
```

```
33 void *listPeek(list_t *list); /* default front*/
34
35 void listAddFront(list_t *list, void *data);
36 void listAddBack(list_t *list, void *data);
37 void listAdd(list_t *list, void *data); /* default back */
38
39 int listEmpty(list_t *list);
40
41 /* looks for a list for an item containging data */
42 void *listFind(list_t *list, void *data, int depth, int (*isit)(void *item,
       void *data));
43 int listRemove(list_t *list, void *data, int (*isit)(void *item, void *data)
       );
44 void listClear(list_t *list);
45 int listSize(list_t *list);
46
47
48 void listPrint(list_t* list,void (*nodeprint)(list_t *node));
49
50
51 #endif
```

C.1.13 logstats.c

```
1  /*
2   * Støtteprogram som analyserer logdata etter en test-kjøring
3   */
4  #include <stdio.h>
5  #include <string.h>
6  #include <stdlib.h>
7  //#include "list.h"
8  /* #include "hwlib.h" */
9  #include "utils.h"
10 #include "hwtypes.h"
11 #include "tree.h"
12
13 typedef struct {
14   hwlibtaskid_t id;
15   hwmodid_t hwmodid;
16   struct timespec started;
17   struct timespec ended;
18   double timereconf;
19 } reconfevent_t;
20
21 typedef struct {
22   hwlibtaskid_t id;
23   hwmodid_t hwmodid;
24   struct timespec added;
25   struct timespec started;
26   struct timespec ended;
27   double timewait;
28   double timeexecute;
29   double timetotal;
30 } taskevent_t;
31
32
33 int reconfeventcmp(void *item, void *data) {
34   unsigned int a = messupint(((reconfevent_t*)item)->id);
35   unsigned int b = messupint(((reconfevent_t*)data)->id);
36   if (a > b) return 1;
37   else if (b > a) return -1;
```

```
38    else return 0;
39  }
40  int taskeventcmp(void *item, void *data) {
41    unsigned int a = messupint(((taskevent_t*)item)->id);
42    unsigned int b = messupint(((taskevent_t*)data)->id);
43    if (a > b) return 1;
44    else if (b > a) return -1;
45    else return 0;
46  }
47
48  int isitreconfevent(void *event, void *id) {
49    unsigned int a = messupint(((reconfevent_t*)event)->id);
50    unsigned int b = messupint((int)id);
51    if (b > a) return 1;
52    else if (a > b) return -1;
53    else return 0;
54    //  return messupint((int)id) - a;
55  }
56  int isittaskevent(void *event, void *id) {
57    unsigned int a = messupint(((taskevent_t*)event)->id);
58    unsigned int b = messupint((int)id);
59    if (b > a) return 1;
60    else if (a > b) return -1;
61    else return 0;
62    //  return messupint((int)id) - a;
63  }
64
65
66  void taskeventnodeprint(tree_t *node) {
67    if (node->data != NULL) {
68      taskevent_t *te = (taskevent_t*)node->data;
69      printf("Task:%d(%d), W:%lf E:%lf T:%lf\n",
70             te->id, te->hwmodid,
71             te->timewait, te->timeexecute, te->timetotal);
72    }
73  }
74  void reconfeventnodeprint(tree_t *node) {
75    if (node->data != NULL) {
76      reconfevent_t *re = (reconfevent_t*)node->data;
77      printf("Reconf:%d(%d), R:%lf\n",
78             re->id, re->hwmodid,
79             re->timereconf);
80    }
81  }
82
83  void reconfnodeprint(tree_t *node, int indent) {
84    int i;
85    int id = ((reconfevent_t*)node->data)->id;
86    for (i=0;i<indent;i++) printf(" ");
87    printf("%d , sorted by %d\n",id, messupint(id));fflush(stdout);
88  }
89
90
91  double tdiff(struct timespec *t1, struct timespec *t2) {
92    return
93      ((double)(t2->tv_sec) + (double)(t2->tv_nsec)*1E-9) -
94      ((double)(t1->tv_sec) + (double)(t1->tv_nsec)*1E-9);
95  }
96
97  int main(int argc, char *args[]) {
98    FILE *input;
99
```

```
100     if (argc < 2) {
101       input = stdin;
102       printf("Reading input from stdin\n");
103     } else {
104       input = fopen(args[1],"r");
105       printf("Reading input from %s\n", args[1]);
106     }
107     if (input == NULL) fprintf(stderr, "Unable to open logfile <%s>\n",args
            [1]);
108     else {
109       const int bufsize = 20;
110       char buffer[bufsize];
111       hwlibtaskid_t taskid;
112       hwmodid_t moduleid;
113       slotnr_t slotid;
114       struct timespec logtime;
115       struct timespec firstevent, lastevent;
116       int read_firstevent = 0;
117       tree_t tasks;
118       tree_t reconfs;
119
120       int reconfstartcounter = 0;
121       int reconfendcounter = 0;
122       int taskaddcounter = 0;
123       int taskstartcounter = 0;
124       int taskendcounter = 0;
125
126       int antreconfs = 0,anttasks = 0;
127       double totalreconf = 0.0, totalwait = 0.0, totalexecute = 0.0, totaltime
              = 0.0;
128       double averagereconf = 0.0, averagewait = 0.0, averageexecute = 0.0,
              averagetime = 0.0;
129       tree_t *itr;
130
131       treeInit(&tasks);
132       treeInit(&reconfs);
133
134       while (fscanf(input, "%s",buffer) != EOF) {
135         if (strncmp(buffer, "LOG", bufsize) == 0) {
136           fscanf(input, " : %ld.%ld : %s task %d module %d slot %d", &logtime.
                  tv_sec,&logtime.tv_nsec, buffer, &taskid,&moduleid,&slotid);
137           if (strncmp(buffer, "StartReconf",bufsize) == 0) {
138             reconfevent_t *re = (reconfevent_t*)calloc(1,sizeof(reconfevent_t)
                    );
139             re->id = taskid;
140             re->hwmodid = moduleid;
141             re->started = logtime;
142             treeAdd(&reconfs, re, reconfeventcmp);
143             reconfstartcounter++;
144           } else if (strncmp(buffer,"EndReconf",bufsize) == 0) {
145             reconfevent_t *re = (reconfevent_t*)treeFind(&reconfs,(void*)
                    taskid,isitreconfevent);
146             if (re == NULL) printf("error, could not find reconfevent for task
                    %d\n",taskid);
147             else {
148               re->ended = logtime;
149               re->timereconf = tdiff(&re->started, &re->ended);
150             }
151             reconfendcounter++;
152           } else if (strncmp(buffer,"AddTask",bufsize) == 0) {
153             taskevent_t *te = (taskevent_t*)calloc(1,sizeof(taskevent_t));
154             te->id = taskid;
```

```
155             te->hwmodid = moduleid;
156             te->added = logtime;
157             treeAdd(&tasks,te,taskeventcmp);
158             taskaddcounter++;
159             if (read_firstevent == 0) { firstevent = te->added;
                    read_firstevent = 1; }
160           } else if (strncmp(buffer,"StartTask",bufsize) == 0) {
161             taskevent_t *te = treeFind(&tasks,(void*)taskid,isittaskevent);
162             if (te == NULL) printf("error, could not find taskevent for task %
                    d (starting)\n",taskid);
163             else {
164               te->started = logtime;
165               te->timewait = tdiff(&te->added,&te->started);
166             }
167             taskstartcounter++;
168           } else if (strncmp(buffer,"EndTask",bufsize) == 0) {
169             taskevent_t *te = treeFind(&tasks,(void*)taskid,isittaskevent);
170             if (te == NULL) printf("error, could not find taskevent for task %
                    d (ending)\n",taskid);
171             else {
172               te->ended = logtime;
173               te->timeexecute = tdiff(&te->started,&te->ended);
174               te->timetotal = te->timewait + te->timeexecute;
175             }
176             taskendcounter++;
177             lastevent = te->ended;
178           }
179           else printf("warning, unrecogniced logevent\n");
180         }
181       }
182
183
184       //printf("\nAll reconfevents (%d?)\n",reconfendcounter);
185       for (itr = treeitr_begin(&reconfs); itr != treeitr_end(&reconfs); itr =
              treeitr_next(itr)) {
186         antreconfs++;
187         totalreconf+=((reconfevent_t*)(itr->data))->timereconf;
188         //reconfeventnodeprint(itr);
189       }
190
191       //printf("\nAll taskevents(%d?)\n",taskendcounter);
192       for (itr = treeitr_begin(&tasks); itr != treeitr_end(&tasks); itr =
              treeitr_next(itr)) {
193         anttasks++;
194         totalwait+=((taskevent_t*)(itr->data))->timewait;
195         totalexecute+=((taskevent_t*)(itr->data))->timeexecute;
196         totaltime+=((taskevent_t*)(itr->data))->timetotal;
197         //taskeventnodeprint(itr);
198       }
199
200       averagereconf = totalreconf / antreconfs;
201       averagewait = totalwait / anttasks;
202       averageexecute = totalexecute / anttasks;
203       averagetime = totaltime / anttasks;
204
205       printf("Total number of tasks executed : %d\n",anttasks);
206       printf("Total number of reconfigs done : %d\n",antreconfs);
207       printf("Total time spent from first add to last end : %lf\n", tdiff(&
              firstevent, &lastevent));
208
209
210
```

```
211
212     }
213     printf("\n");
214     return 0;
215   }
```

C.1.14 lru.c

```
1   /*
2    * Implementasjon av planlegger som benytter LRU-
3    * (least recently used) algoritmen
4    */
5   #include <stdlib.h>
6   #include "hwtypes.h"
7   #include "hwglobals.h"
8   #ifdef use_lrumutex
9   #include "mututils.h"
10  #else
11  #define LockIt(foo)
12  #define UnLockIt(foo)
13  #endif
14
15
16  typedef struct lruinfo_t {
17    int count;
18    hwmodid_t id;
19  } lruinfo_t;
20
21  lruinfo_t *lrucounter;
22  #ifdef use_lrumutex
23  pthread_mutex_t lrumutex = PTHREAD_MUTEX_INITIALIZER;
24  #endif
25  /* Finds a notinuse slot allready configured
26   * for this module (-1 on not found)
27   */
28  slotnr_t lru_findid(hwmodid_t id) {
29    slotnr_t res = -1, count;
30    for (count = 0; count < numberofmoduleslots; count++) {
31      if ((lrucounter[count].id == id) &&
32          (moduleslots[count].inuse == 0)) { // don't consider inuse slots
33        res = count; break;
34      }
35    }
36    return res;
37  }
38
39  /* Finds the least resently (notinuse) used slot
40   * If all slots are inuse, return -1
41   */
42  slotnr_t lru_findlru() {
43    slotnr_t res = -1, count;
44    int max = -1;
45    for (count = 0; count < numberofmoduleslots; count++)
46      if ((lrucounter[count].count > max) &&
47          (moduleslots[count].inuse == 0)) { // don't consider inuse slots
48        res = count; max = lrucounter[count].count;
49      }
50    return res;
51  }
52
53  void lru_updateallbut(slotnr_t slotid) {
```

```
54    slotnr_t count;
55    hwmodid_t modid;
56    switch (how_sched_update) {
57    case how_sched_update_onlythis :
58      for (count = 0; count < numberofmoduleslots; count++)
59        if (count != slotid) lrucounter[count].count++;
60      break;
61    case how_sched_update_allsimilar :
62      modid = lrucounter[slotid].id;
63      for (count = 0; count < numberofmoduleslots; count++)
64        if (lrucounter[count].id != modid) lrucounter[count].count++;
65      break;
66    default:
67      break;
68    }
69
70  }
71
72
73  void scheduler_init() {
74    slotnr_t count;
75    LockIt(&lrumutex);
76    lrucounter = (lruinfo_t*)calloc(numberofmoduleslots,sizeof(lruinfo_t));
77    for (count = 0; count < numberofmoduleslots; count++)
78      lrucounter[count].id = nohwmod;
79    UnLockIt(&lrumutex);
80  }
81  void scheduler_register(slotnr_t slotid) {
82    LockIt(&lrumutex);
83    lrucounter[slotid].count = 0;
84    lrucounter[slotid].id = moduleslots[slotid].id;
85
86    lru_updateallbut(slotid);
87    UnLockIt(&lrumutex);
88  }
89
90
91  slotnr_t scheduler_findnext(hwmodid_t id) {
92    slotnr_t slotnr;
93    LockIt(&lrumutex);
94    slotnr = lru_findid(id);
95    if (slotnr == -1) { // no inactive preconfigured slots found
96      slotnr = lru_findlru();
97
98
99    }
100   UnLockIt(&lrumutex);
101   return slotnr;
102 }
```

C.1.15 mututils.h

```
1  /*
2   * Definisjon av endel enkle makroer som brukes
3   * i forbindlese med mutex-er
4   */
5
6  #ifndef _fcache_hwlib_locks_
7  #define _fcache_hwlib_locks_
8  #include <pthread.h>
9  #define LockIt(mut) pthread_mutex_lock(mut)
```

```
10 #define UnLockIt(mut) pthread_mutex_unlock(mut)
11 #endif
```

C.1.16 nalletest.c

```
1  /*
2   * Hovedtest-program som ble benyttet for å teste ut ytelsen
3   * av det komplette systemet vårt (programvaredriver og maskinvare)
4   */
5  #include <stdio.h>
6  #include <time.h>
7  #include <string.h>
8  #include "hwlib.h"
9  #include "utils.h"
10
11 int taskcounter = 0;
12
13 unsigned int max(unsigned int a, unsigned int b, unsigned int c) {
14   unsigned int r = a;
15   if (b > r) r = b;
16   if (c > r) r = c;
17   return r;
18 };
19
20 void initdata(unsigned int *data, int datasize) {
21   int istop = datasize /4;
22   int i;
23   for (i = 0; i < istop; i++) {
24     data[4*i] = random();
25     data[4*i+1] = random();
26     data[4*i+2] = random();
27     data[4*i+3] = max(data[4*i], data[4*i+1], data[4*i+2]) +1;
28   }
29 }
30
31 int initTask(task_t *t, char *mod, void* data, int words, int reswords) {
32   t->id = taskcounter++;
33   t->hwmod = treeFind(&moduletree,mod,isitmodulename);
34   t->completed = 0;
35   t->taskdatasize = words;
36   t->resultsize = reswords;
37
38   //  t->taskdata = data;
39   t->taskdata = (void*)malloc(words*sizeof(int));
40   memcpy(t->taskdata, data, words*sizeof(int));
41
42   t->result = malloc(reswords*sizeof(int));
43   if (t->hwmod == NULL) return 0;
44   else return 1;
45 }
46
47 void initRandomTask(task_t *task, int *data, int datasize, int tasksize, int
       ressize) {
48   int modid = random() % 4;
49   int start = random() % (datasize-3);
50   switch (modid) {
51   case 0 : initTask(task, "Sum0", &data[start], 4, 1); break;
52   case 1 : initTask(task, "Sum1", &data[start], 4, 1); break;
53   case 2 : initTask(task, "ExpMultMod0", &data[start], 4, 1); break;
54   case 3 : initTask(task, "ExpMultMod1", &data[start], 4, 1); break;
55   default : break;
```

```
56    }
57  }
58
59
60  void taskprint(task_t *task) {
61    printf("TASK:\n  id=%d\n  completed=%d\n",task->id,task->completed);
62    printf("  data(%d)={%d,%d,%d,%d}\n",task->taskdatasize,
63           ((int*)task->taskdata)[0], ((int*)task->taskdata)[1],
64           ((int*)task->taskdata)[2], ((int*)task->taskdata)[3]);
65    printf("  res(%d)={%d}\n",task->resultsize,((int*)task->result)[0]);
66  }
67
68  #define datasize (1<<11)
69  int main(int argc, char *argv[]) {
70    int data[datasize];
71    task_t *tasks;
72    int i;
73    int done = 0;
74    struct timespec req,rem;
75    int anttasks;
76    FILE *fp;
77
78    if (argc <= 2) { printf("Must specify taskfile and libfile\n"); exit(-1);
          }
79    else {
80      fp = fopen(argv[1], "r");
81      hwlib_init(argc-3, &argv[3]);
82      loadLibFile(argv[2]);
83
84      fscanf(fp,"%d",&anttasks);
85      tasks = (task_t*)calloc(anttasks,sizeof(task_t));
86
87      initdata(data, datasize);
88
89      for (i = 0; i < anttasks; i++) {
90        int index,max;
91        char modname[24];
92        int args,res;
93        fscanf(fp,"%s %d %d",modname, &args, &res);
94        index = random();
95        max = datasize /2;
96        index = index % max;
97        index = index & ~(0x3);
98        initTask(&tasks[i], modname, &data[index], args, res);
99      }
100
101     for (i = 0; i < anttasks; i++) {
102       addTask(tasks+i);
103     }
104
105     req.tv_sec = 1;
106     req.tv_nsec = 0;
107     while (!done) {
108       nanosleep(&req, &rem);
109       done = 1;
110       for (i = 0; i < anttasks; i++)
111         if (tasks[i].completed == 0) done = 0;
112       // if (tasks[0].completed == 0) done = 0;
113     }
114
115     for (i = 0; i < anttasks; i++) {
116       memcpy(data+4,tasks[i].result,sizeof(int));
```

```
117         printf("Results for task %d (mod %d) :%d\n",i,tasks[i].hwmod->id,((int
                *)tasks[i].result)[0]);
118       }
119
120
121
122
123       hwlib_exit();
124       return 0;
125     }
126   }
```

C.1.17 softwaresol.c

```
1   /*
2    * Testprogram som målte ytelsen av å kjøre de samme oppgavene
3    * som ble kjørt igjennom nalletest.c, ved å benytte en ren
4    * programvare-løsning.
5    */
6   #include <stdio.h>
7   #include <time.h>
8   #include <string.h>
9   #include "swlib.h"
10  #include "hwtypes.h"
11
12  int taskcounter = 0;
13
14  unsigned int max(unsigned int a, unsigned int b, unsigned int c) {
15    unsigned int r = a;
16    if (b > r) r = b;
17    if (c > r) r = c;
18    return r;
19  };
20
21  void initdata(unsigned int *data, int datasize) {
22    int istop = datasize /4;
23    int i;
24    for (i = 0; i < istop; i++) {
25      data[4*i] = random();
26      data[4*i+1] = random();
27      data[4*i+2] = random();
28      data[4*i+3] = max(data[4*i], data[4*i+1], data[4*i+2]) +1;
29    }
30  }
31
32  int initTask(task_t *t, char *mod, void* data, int words, int reswords) {
33    t->taskdatasize = words;
34    t->resultsize = reswords;
35
36    //  t->taskdata = data;
37    t->taskdata = (void*)malloc(words*sizeof(int));
38    memcpy(t->taskdata, data, words*sizeof(int));
39
40    t->result = (void*)malloc(reswords*sizeof(int));
41
42    /* Using slotnr to indikate sum or emm */
43    if (strncmp(mod,"Sum",3) == 0) t->slotnr = 0;
44    else if (strncmp(mod, "Exp", 3) == 0) t->slotnr = 1;
45    else {
46      t->slotnr = 2;
47      printf("Init : unknown tasktype : %s\n", mod);
```

```
48    }
49
50    return 1;
51  }
52
53
54  void runTask(task_t *task) {
55    if         (task->slotnr == 0) { // sum
56      *(unsigned int*)(task->result) = sum((unsigned int*)task->taskdata, task
          ->taskdatasize);
57    } else if (task->slotnr == 1) { // emm
58      *(unsigned int*)(task->result) = expmultmod(((unsigned int*)task->
          taskdata)[0], ((unsigned int*)task->taskdata)[1], ((unsigned int*)
          task->taskdata)[2], ((unsigned int*)task->taskdata)[3]);
59    } else {
60      printf("Error, unknown tasktype : %d\n", task->slotnr);
61    }
62  }
63
64  double tdiff(struct timespec *t1, struct timespec *t2) {
65    return
66      ((double)(t2->tv_sec) + (double)(t2->tv_nsec)*1E-9) -
67      ((double)(t1->tv_sec) + (double)(t1->tv_nsec)*1E-9);
68  }
69
70
71
72  #define datasize (1<<11)
73  int main(int argc, char *argv[]) {
74    int data[datasize];
75    task_t *tasks;
76    int i;
77    int done = 0;
78    struct timespec start,stop;
79    double timespent;
80    int anttasks;
81    FILE *fp;
82
83    if (argc <= 1) { printf("Must specify taskfile\n"); exit(-1); }
84    else {
85      fp = fopen(argv[1], "r");
86
87      fscanf(fp,"%d",&anttasks);
88      tasks = (task_t*)calloc(anttasks,sizeof(task_t));
89
90      // initialize random data.
91      initdata(data, datasize);
92
93      // Read taskinfo and create tasks
94      for (i = 0; i < anttasks; i++) {
95        int index,max;
96        char modname[24];
97        int args,res;
98        fscanf(fp,"%s %d %d",modname, &args, &res);
99        index = random();
100       max = datasize /2;
101       index = index % max;
102       index = index & ~(0x3);
103       initTask(&tasks[i], modname, &data[index], args, res);
104     }
105
106     printf("Run started\n");
```

```
107       clock_gettime(CLOCK_REALTIME, &start);
108       for (i = 0; i < anttasks; i++) {
109         runTask(tasks+i);
110       }
111       clock_gettime(CLOCK_REALTIME, &stop);
112       printf("Run Completed\n");
113       timespent = tdiff(&start, &stop);
114
115       printf("Started : %d  :  %d\n",start.tv_sec, start.tv_nsec);
116       printf("Stopped : %d  :  %d\n",stop.tv_sec, stop.tv_nsec);
117       printf("Total time spent on run = %lf\n",timespent);
118
119
120
121
122
123
124       return 0;
125     }
126   }
```

C.1.18 swlib.c

```
1   /*
2    * Implementasjon av rene programvareløsninger for de modulene
3    * vi benyttet i maskinvaredesignet vårt.
4    */
5
6   #include "swlib.h"
7   unsigned int sum(unsigned int *data, unsigned int size) {
8     unsigned int i=0,res=0;
9     for (i = 0; i < size; i++) res += data[i];
10    return res;
11  }
12
13  unsigned int multmod(unsigned int a, unsigned int b, unsigned int m) {
14    unsigned int r=0;
15    while (b!=0) {
16      if (b & 1) { // oddetall
17        r += a;
18        if (r >= m) r -= m;
19      }
20      a <<= 1; if (a >= m) a-=m;
21      b >>= 1;
22    } return r;
23  }
24
25  unsigned int expmultmod(unsigned int a, unsigned int b, unsigned int c,
        unsigned int m) {
26    while (c != 0) {
27      if (c & 1)  a = multmod(a,b,m);
28      b = multmod(b,b,m);
29      c >>= 1;
30    } return a;
31  }
```

C.1.19 swlib.h

```
1   /*
2    * Header-fil for programvare implementasjonenen av
```

```
 * maskinvare-modulene våre
 */

#ifndef _fcache_swlib_h_
#define _fcache_swlib_h_

unsigned int sum(unsigned int *data, unsigned int size);
unsigned int multmod(unsigned int a, unsigned int b, unsigned int m);
unsigned int expmultmod(unsigned int a, unsigned int b, unsigned int c,
    unsigned int m);

#endif
```

C.1.20 tree.c

```
/*
 * Implementasjon av binær-tre strukturen fra tree.h
 */
#include <stdlib.h>
#include <stdio.h>
#include "tree.h"

#ifdef use_tree_mutex
#include "mututils.h"
#else
#define LockIt(foo)
#define UnLockIt(foo)
#endif

#define _indent(indent) int indentcounter;for (indentcounter=0;indentcounter
    <indent;indentcounter++)printf(" ");fflush(stdout)

tree_t *rightmostnode(tree_t *tree);
tree_t *leftmostnode(tree_t *tree);


tree_t *rightmostnode(tree_t *tree) {
  if (tree == NULL) return NULL;
  else if (tree->rnode == NULL) return tree;
  else return rightmostnode(tree->rnode);
}
tree_t *leftmostnode(tree_t *tree) {
  if (tree == NULL) return NULL;
  else if (tree->parent == NULL) return leftmostnode(tree->rnode);
  else if (tree->lnode == NULL) return tree;
  else return leftmostnode(tree->lnode);
}


void treeInit(tree_t *tree) {
  if (tree != NULL) {
    tree->lnode = tree->rnode = tree->parent = tree->data = NULL;
#ifdef use_tree_mutex
    pthread_mutex_init(&(tree->mutex), NULL);
#endif
  }
}
void treeAdd(tree_t *tree, void *data, int (*comp)(void *data1, void *data2)
    ) {
  tree_t *nnode = (tree_t*)malloc(sizeof(tree_t));
  LockIt(&(tree->mutex));
```

```
45    treeInit(nnode);  nnode->data = data;
46    if (tree->rnode == NULL) { // first entry
47      tree->rnode = nnode;
48      nnode->parent = tree;
49    } else {
50      tree_t *lookat = tree->rnode;
51      int done = 0;
52      while (!done) {
53        int compare = (*comp)(lookat->data, nnode->data);
54        if (compare == 0) {            // match, discard and return
55          free(nnode); done = 1;
56        } else if (compare > 0) {      // go left
57          if (lookat->lnode == NULL) {   //insert on left
58            lookat->lnode = nnode;
59            nnode->parent = lookat;
60            done = 1;
61          } else lookat = lookat->lnode; // look at left node
62        } else {                       // go right
63          if (lookat->rnode == NULL) {   //insert on right
64            lookat->rnode = nnode;
65            nnode->parent = lookat;
66            done = 1;
67          } else lookat = lookat->rnode; // look at right node
68        }
69      }
70    }
71    UnLockIt(&(tree->mutex));
72  }
73
74  int treeEmpty(tree_t *tree) {
75    if (tree->rnode == NULL) return 1; else return 0;
76  }
77
78  void *treeFind(tree_t *tree, void *data, int (*comp)(void *data1, void *
        data2)) {
79    if (tree == NULL) return NULL;
80    else {
81      void *res;
82      if (tree->parent == NULL) { // headnode
83        /* Only perform locking on headnode */
84        LockIt(&(tree->mutex));
85        res = treeFind(tree->rnode, data, (*comp));
86        UnLockIt(&(tree->mutex));
87      } else {
88        int compare = (*comp)(tree->data, data);
89        if (compare == 0) return tree->data;
90        else if (compare < 0) // check left
91          res = treeFind(tree->lnode, data, (*comp));
92        else                  // check right
93          res = treeFind(tree->rnode, data, (*comp));
94      }
95      return res;
96    }
97  }
98
99  void treeClear(tree_t *tree) {
100   if (tree == NULL) return;
101   else {
102     LockIt(&(tree->mutex));
103     treeClear(tree->lnode);
104     treeClear(tree->rnode);
105     free(tree);
```

```
106       UnLockIt(&(tree->mutex));
107     }
108   }
109
110   void *treeRemove(tree_t *tree, void *data, int (*comp)(void *data1, void *
        data2), int lorr) {
111     if (tree == NULL) return NULL;
112     else {
113       void *res = NULL;
114       if (tree->parent == NULL) { /* Headnode, lock it */
115         LockIt(&(tree->mutex));
116         res = treeRemove(tree->rnode,data,(*comp),RightNode);
117         UnLockIt(&(tree->mutex));
118       } else {
119         int compare = (*comp)(tree->data,data);
120         if (compare == 0) { /* remove this */
121           if (lorr == RightNode) {
122             if (tree->lnode == NULL) {
123               if (tree->rnode == NULL) { /* just remove */
124                 tree->parent->rnode = NULL;
125               } else {
126                 tree->parent->rnode = tree->rnode;
127                 tree->rnode->parent = tree->parent;
128               }
129             } else {
130               if (tree->rnode == NULL) {
131                 tree->parent->rnode = tree->lnode;
132                 tree->lnode->parent = tree->parent;
133               } else {
134                 tree_t *rmost = rightmostnode(tree->lnode);
135                 tree->parent->rnode = tree->lnode;
136                 tree->lnode->parent = tree->parent;
137                 rmost->rnode = tree->rnode;
138                 tree->rnode->parent = rmost;
139               }
140             }
141           } else {
142             if (tree->lnode == NULL) {
143               if (tree->rnode == NULL) { /* just remove */
144                 tree->parent->lnode = NULL;
145               } else {
146                 tree->parent->lnode = tree->rnode;
147                 tree->rnode->parent = tree->parent;
148               }
149             } else {
150               if (tree->rnode == NULL) {
151                 tree->parent->lnode = tree->lnode;
152                 tree->lnode->parent = tree->parent;
153               } else {
154                 tree_t *rmost = rightmostnode(tree->lnode);
155                 tree->parent->lnode = tree->lnode;
156                 tree->lnode->parent = tree->parent;
157                 rmost->rnode = tree->rnode;
158                 tree->rnode->parent = rmost;
159               }
160             }
161            }
162            res = tree->data; free(tree);
163           } else if (compare > 0) { /* check left */
164             res = treeRemove(tree->lnode, data,(*comp),LeftNode);
165           } else { /* check Right */
166             res = treeRemove(tree->rnode,data,(*comp),RightNode);
```

```
167         }
168       }
169       return res;
170     }
171   }
172
173   void treenodeprint(tree_t *node, int indent) {
174     _indent(indent);
175     printf("N:0x%X, D:0x%X, P:0x%X, L:0x%X, R:0x%X\n",(int)node,(int)node->
          data,(int)node->parent,(int)node->lnode,(int)node->rnode);
176   }
177
178   void treePrint(tree_t *tree, int indent, void (*nodeprint)(tree_t *node, int
          indent)) {
179     if (nodeprint == NULL) nodeprint = treenodeprint;
180     if (tree == NULL) {
181     } else if (tree->parent == NULL) { // headnode
182       treePrint(tree->rnode, indent, nodeprint);
183     } else {
184       (*nodeprint)(tree, indent);
185       treePrint(tree->lnode, indent +3,nodeprint);
186       treePrint(tree->rnode, indent +3,nodeprint);
187     }
188   }
189
190   int treeSize(tree_t *tree) {
191     if (tree == NULL) return 0;
192     else return treeSize(tree->rnode) + treeSize(tree->lnode);
193   }
194
195
196
197
198   tree_t *treenext(tree_t *treenode, tree_t *calling) {
199     if (treenode->parent == NULL) return treenode;
200     else if (calling == NULL) {
201       if (treenode->rnode != NULL) return leftmostnode(treenode->rnode);
202       else return treenext(treenode->parent, treenode);
203     } else {
204       if (treenode->lnode == calling) return treenode;
205       else return treenext(treenode->parent, treenode);
206     }
207   }
208   tree_t *treeprev(tree_t *treenode, tree_t *calling) {
209     if (treenode->parent == NULL) return treenode;
210     else if (calling == NULL) {
211       if (treenode->lnode != NULL) return rightmostnode(treenode->lnode);
212       else return treeprev(treenode->parent, treenode);
213     } else {
214       if (treenode->rnode == calling) return treenode;
215       else return treeprev(treenode->parent, treenode);
216     }
217   }
218
219
220
221
222   tree_t *treeitr_begin(tree_t *tree) { return leftmostnode(tree->rnode); }
223   tree_t *treeitr_rbegin(tree_t *tree) { return rightmostnode(tree->rnode); }
224   tree_t *treeitr_end(tree_t *tree) { return tree; }
225   tree_t *treeitr_next(tree_t *treenode) { return treenext(treenode, NULL); }
226   tree_t *treeitr_prev(tree_t *treenode) { return treeprev(treenode, NULL); }
```

C.1.21 tree.h

```
1  /*
2   * Headerfil for et generisk ubalansert binær-tre
3   */
4  #ifndef _binary_tree_h_
5  #define _binary_tree_h_
6
7  #ifdef use_tree_mutex
8  #include <pthread.h>
9  #else
10 #define LockIt(foo)
11 #define UnLockIt(foo)
12 #endif
13
14 #ifndef NULL
15 #define NULL 0x0
16 #endif
17
18 #define LeftNode 0
19 #define RightNode 1
20
21 typedef struct tree_t {
22   struct tree_t *lnode;
23   struct tree_t *rnode;
24   struct tree_t *parent;
25   void *data;
26 #ifdef use_tree_mutex
27   pthread_mutex_t mutex;
28 #endif
29 } tree_t;
30
31
32
33
34 void treeInit(tree_t *tree);
35 void treeAdd(tree_t *tree, void *data, int (*comp)(void *data1, void *data2)
       );
36 int treeEmpty(tree_t *tree);
37 void *treeFind(tree_t *tree, void *data, int (*comp)(void *data1, void *
       data2));
38 void treeClear(tree_t *tree);
39 void *treeRemove( tree_t *tree, void *data, int (*comp)(void *data1, void *
       data2), int lorr );
40
41 void treePrint(tree_t *tree, int indent, void (*nodeprint)(tree_t *node, int
        indent));
42 int treeSize(tree_t *tree);
43
44 tree_t *treeitr_begin(tree_t *tree);
45 tree_t *treeitr_rbegin(tree_t *tree);
46 tree_t *treeitr_end(tree_t *tree);
47 tree_t *treeitr_next(tree_t *treenode);
48 tree_t *treeitr_prev(tree_t *treenode);
49
50
51
52 #endif
```

C.1.22 utils.c

```
1  /*
2   * Implementasjon av ekstrafunksjoner definert i utils.h
3   */
4  #include <string.h>
5  #include <stdlib.h>
6  #include "utils.h"
7  #include "hwglobals.h"
8
9  int modidcounter = 0;
10
11 FILE *stderrstream = NULL;
12 FILE *stdoutstream = NULL;
13 FILE *logstream = NULL;
14
15 void freeModule(HWModule_t *module);
16
17
18
19 void setErrStream(FILE *errstrm) {
20   stderrstream = errstrm; }
21 void setOutStream(FILE *outstrm) {
22   stdoutstream = outstrm; }
23 void setLogStream(FILE *logstrm) {
24   logstream = logstrm; }
25
26 void readArgs(int argc, char *argv[]) {
27   int tmp = 0;
28   while(tmp < argc) {
29     if        (strcmp(argv[tmp], "-csize") == 0) {
30       sscanf(argv[tmp+1], "%d", &numberofmoduleslots); tmp+=2;
31
32     } else if (strcmp(argv[tmp], "-allowduplicates") == 0) {
33       allow_duplicates = allow_duplicates_yes; tmp++;
34     } else if (strcmp(argv[tmp], "-no_allowduplicates") == 0) {
35       allow_duplicates = allow_duplicates_no; tmp++;
36
37     } else if (strcmp(argv[tmp], "-howschedupdateallsimilar") == 0) {
38       how_sched_update = how_sched_update_allsimilar; tmp++;
39     } else if (strcmp(argv[tmp], "-howschedupdateonlythis") == 0) {
40       how_sched_update = how_sched_update_onlythis; tmp++;
41
42     } else if (strcmp(argv[tmp], "-taskforwarding") == 0) {
43       use_taskforwarding = use_taskforwarding_yes; tmp++;
44     } else if (strcmp(argv[tmp], "-no_taskforwarding") == 0) {
45       use_taskforwarding = use_taskforwarding_no; tmp++;
46
47     } else if (strcmp(argv[tmp], "-maxforwarddepth") == 0) {
48       sscanf(argv[tmp+1], "%d", &maxforwarddepth); tmp += 2;
49
50     } else if (strcmp(argv[tmp], "-schedregisteronforwarded") == 0) {
51       sched_register_on_forwarded = sched_register_on_forwarded_yes; tmp++;
52     } else if (strcmp(argv[tmp], "-no_schedregisteronforwarded") == 0) {
53       sched_register_on_forwarded = sched_register_on_forwarded_no; tmp++;
54
55     } else if (strcmp(argv[tmp], "-reconfigcard") == 0) {
56       reconfig_card = reconfig_card_yes; tmp++;
57     } else if (strcmp(argv[tmp], "-no_reconfcard") == 0) {
58       reconfig_card = reconfig_card_no; tmp++;
59     } else if (strcmp(argv[tmp], "-bitfile") == 0) {
60       strcpy(bitfilename,argv[tmp+1]); tmp += 2;
61
62     } else if (strcmp(argv[tmp], "--help") == 0) {
```

```
63        printf("-csize <n>                : set number of moduleslots (default
              4)\n");
64        printf("-[no_]allowduplicates     : (default no_allowduplicates)\n");
65        printf("-howschedupdateallsimilar : call scheduler_register on all
              similar modules (default)\n");
66        printf("-howschedupdateonlythis   : only call scheduler_register on
              this moduleslot\n");
67        printf("-[no_]taskforwarding      : Enables the deamon to look ahead
              in the taskqueue to \n");
68        printf("                            find tasks that can be run without
               reconf (default no)\n");
69        printf("-maxforwarddepth <n>      : Specify how far back in the
              taskqueue the deamon should look (default 10)\n");
70        printf("-[no_]schedregisteronforwarded : Should a forwarded task
              trigger a scheduler register (default yes)\n");
71        printf("-[no_]reconfigcard        : (default reconfigure card on
              startup)\n");
72        printf("-bitfile <fname>          : use fname when reconfiguring\n");
73        printf("                            NB!! the one used to reconf the
              entire device, not modulefile\n");
74        printf("--help                    : this text\n");
75        exit(0);
76      } else {
77        printf("Unknown option : %s (ignorded)\n",argv[tmp]); tmp++;
78      }
79    }
80  }
81
82  #define libfileerror(str) \
83    fprintf(stderrstream,"%s\n",str); \
84    count = numberofmodules
85
86  int loadLibFile(char fname[HWMODNAMELENGTH]) {
87    int
88      modulesread = 0,
89      funcres,
90      sizeofentry = sizeof(libfileentry_t),
91      count,count2;
92    slotnr_t numberofmodules;
93    libfileentry_t tmpentry;
94    FILE *fp  = fopen(fname, "r");
95    FILE *fp2 = fopen(fname, "r");
96
97    funcres = fread((void*)&numberofmodules,4,1,fp);
98
99    if (funcres != 1) { fprintf(stderrstream, "Unable to read number of
          entries in libfile\n");}
100   else {
101     funcres = fseek(fp, 16, SEEK_SET);
102     if (funcres != 0) {
103       fprintf(stderrstream, "Unable to seek foreward to first entry.\n");
104     } else {
105       for (count = 0; count < numberofmodules; count++) {
106         funcres = fread((void*)&tmpentry,sizeofentry,1,fp);
107         if (funcres != 1) {
108           fprintf(stderrstream, "Premature end of libfile\n");
109           count = numberofmodules;
110         } else {
111           funcres = fseek(fp2, tmpentry.startoffset, SEEK_SET);
112           if (funcres != 0) {
113             fprintf(stderrstream, "Unable to seek to position in libfile\n")
                  ;
```

```
114             count = numberofmodules;
115           } else {
116             struct HWModule_t *module = (HWModule_t*)calloc(1,sizeof(
                    HWModule_t));
117             if (module == NULL) {
118               fprintf(stderrstream, "Unable to allocate memory for module\n"
                      ); count = numberofmodules;
119             } else {
120               module->bitstream = (void*)malloc(numberofmoduleslots*sizeof(
                      void*));
121               module->streamsize = (int*)malloc(numberofmoduleslots*sizeof(
                      int));
122               if ((module->bitstream == NULL) || (module->streamsize == NULL
                      )) {
123                 fprintf(stderrstream, "Unable to create pointers to
                        bitstreams\n"); count = numberofmodules;
124                 freeModule(module);
125               } else {
126
127                 for (count2 = 0; count2 < numberofmoduleslots;count2++) {
128                   funcres = fread((void*)&module->streamsize[count2],sizeof(
                          int),1,fp2);
129                   if (funcres == 0) { printf("Unable to read streamsize\n");
                          count2 = numberofmoduleslots; count = numberofmodules
                          ; freeModule(module); }
130                   else {
131                     module->bitstream[count2] = malloc(module->streamsize[
                          count2]);
132
133                     if (module->bitstream[count2] == NULL) {
134                       fprintf(stderrstream, "Unable to allocate memory for
                            module bitstream\n");
135                       count = numberofmodules;
136                       freeModule(module);
137                     } else {
138                       funcres = fread(module->bitstream[count2],1,module->
                            streamsize[count2],fp2);
139                       if (funcres < module->streamsize[count2]) {
140                         fprintf(stderrstream, "Unable to read module
                              bitstream\n");
141                         count = numberofmodules;
142                         freeModule(module);
143                       } /* fread bitstream */
144                     } /* malloc bitstream */
145                   } /* fread streamsize */
146
147                 } /* numberofmoduleslots */
148
149                 /* Register module in a set or something...
150                  * Currently a three */
151                 strcpy(module->hwmodname, tmpentry.modulename);
152                 module->id = modidcounter; modidcounter++;
153                 modulesread++;
154                 //printModule(module,0);
155                 treeAdd(&moduletree, module, modulecmp);
156               } /* malloc bitstream&streamsize */
157             } /* malloc module */
158           } /* fseek */
159         } /* fread entry */
160       } /* for count numberofmodules*/
161     } /* seek 16 */
162   } /* numberofmodules */
```

```
163   return modulesread;
164 }
165
166 int loadModuleFromLibFile(char *modname,char *fname) {
167   int
168     modulesread = 0,
169     funcres,
170     sizeofentry = sizeof (libfileentry_t),
171     count,
172     count2;
173   slotnr_t numberofmodules;
174   libfileentry_t tmpentry;
175   FILE *fp = fopen(fname, "r");
176
177   funcres = fread((void*)&numberofmodules,4,1,fp);
178   if (funcres != 1) { libfileerror("unable to read number of entries in
          libfile"); }
179   else {
180     funcres = fseek(fp,16,SEEK_SET);
181     if (funcres != 0) { libfileerror("unable to seek forward to first entry"
          ); }
182     else {
183       for (count = 0; count < numberofmodules; count++) {
184         funcres = fread((void*)&tmpentry,sizeofentry,1,fp);
185         if (funcres != 1) { libfileerror("Unable to read entry nr %d"); }
186         else {
187           if (strncmp(tmpentry.modulename,modname,HWMODNAMELENGTH) == 0) {
188             count = numberofmodules;
189             funcres = fseek(fp,tmpentry.startoffset,SEEK_SET);
190             if (funcres != 0) { libfileerror("Unable to seek forward to
                  entry"); }
191             else {
192               HWModule_t *module = (HWModule_t*)calloc(1,sizeof(HWModule_t))
                    ;
193               if (module == NULL) { libfileerror("Unable to create new
                    moduleobj"); }
194               else {
195                 module->bitstream = (void**)malloc(numberofmoduleslots*
                      sizeof(void*));
196                 module->streamsize = (int*)malloc(numberofmoduleslots*sizeof
                      (int));
197                 if ((module->bitstream == NULL) || (module->streamsize ==
                      NULL)) {
198                   libfileerror("Unable to allocate pointers to bitstream and
                        /or bitstreamsize"); freeModule(module);}
199                 else {
200                   for (count2 = 0; count2 < numberofmoduleslots; count2++) {
201                     funcres = fread((void*)&(module->streamsize[count2]),
                          sizeof(int),1,fp);
202                     if (funcres < 1) { libfileerror("unable to read
                          streamsize"); freeModule(module); }
203                     else {
204                       module->bitstream[count2] = malloc(module->streamsize[
                            count2]);
205                       if (module->bitstream[count2] == NULL) { libfileerror(
                            "Unable to allocate memory for bitstream");
                            freeModule(module); }
206                       else {
207                         funcres = fread(module->bitstream[count2],1,module->
                              streamsize[count2],fp);
208                         if (funcres < module->streamsize[count2]) {
```

```
209                             libfileerror("Unable to read bitstream");
                                    freeModule(module);
210                           } /* fread bitstrem */
211                         } /* malloc bitstream */
212                       } /* fread streamsize */
213                     } /* for count2 */
214                   } /* malloc bitstream*, malloc streamsize*/
215                   strcpy(module->hwmodname,tmpentry.modulename);
216                   module->id =  modidcounter; modidcounter++;
217                   modulesread++;
218                   //printModule(module,0);
219                   treeAdd(&moduletree,module,modulecmp);
220                 } /* malloc HWModule_t */
221               } /* fseek startoffset */
222             } /* if modname = modulename */
223           } /* fread tmpentry */
224         } /* for count */
225       } /* fseek first entry */
226     } /* fread numberofmodules */
227     return modulesread;
228   }
229
230   int unloadModule(char *modname) {
231     HWModule_t *module = (HWModule_t*)treeRemove(&moduletree,modname,
          isitmodulename,RightNode);
232     if (module == NULL) return 0;
233     else {
234       freeModule(module);
235       return 1;
236     }
237   }
238   void freeModule(HWModule_t *module) {
239     if (module != NULL) {
240       free(module->streamsize);
241       if (module->bitstream != NULL) {
242         slotnr_t count;
243         for (count = 0; count < numberofmoduleslots; count++) {
244           free(module->bitstream[count]);
245         }
246       }
247     }
248   }
249
250
251
252
253
254   int isitmodulename(void *module, void *modname) {
255     if (module == NULL) return -2;
256     else return strcmp((char*)modname, ((HWModule_t*)module)->hwmodname);
257   }
258
259   int modulecmp(void *mod1, void *mod2) {
260     if ((mod1 == NULL) || (mod2 == 0)) return -2;
261     else return strcmp(((HWModule_t*)mod1)->hwmodname, ((HWModule_t*)mod2)->
          hwmodname);
262   }
263
264
265   /* Hvorfor buker jeg *(int*)id her egentlig???  burde være (int)id  */
266   int isitmoduleid(void *module, void *id) {
267     if (module == NULL) return -2;
```

```
268   else return (int)(*(int*)id) - ((HWModule_t*)module)->id;
269 }
270 int isittaskwithmoduleid(void *task, void *id) {
271   if (task == NULL) return 0;
272   else if (((task_t*)task)->hwmod->id == (hwmodid_t)id) return 1;
273   else return 0;
274 }
275
276 void printModule(HWModule_t *hwmod, int indent) {
277   slotnr_t i;
278   _indent(indent);printf("Module : %s\n",hwmod->hwmodname);
279   indent++;
280   for (i = 0; i < numberofmoduleslots; i++) {
281     char *buf = (char*)hwmod->bitstream[i];
282     int len = hwmod->streamsize[i];
283     int i;
284     _indent(indent);printf("Version %d (0x%X): ",i,(int)hwmod->bitstream[i])
          ;
285     for (i = 0; i < len; i++) {
286       if (buf[i] == 0) printf(" ");
287       else printf("%c",buf[i]);
288       fflush(stdout);
289     } printf("\n");
290   }
291 }
292
293 void modulenodeprint(tree_t *node, int indent) {
294   HWModule_t *module = (HWModule_t*)node->data;
295   _indent(indent);printf("Module : %s      ",module->hwmodname);
296   if (node->parent == NULL) printf("P:NULL     ");
297   else printf("P:%s     ",((HWModule_t*)node->parent->data)->hwmodname);
298   if (node->lnode == NULL) printf("L:NULL      ");
299   else printf("L:%s     ",((HWModule_t*)node->lnode->data)->hwmodname);
300   if (node->rnode == NULL) printf("R:NULL\n");
301   else printf("R:%s\n",((HWModule_t*)node->rnode->data)->hwmodname);
302 }
303
304
305 int loadTaskFile(char *fname, list_t *list) {
306   int taskcounter = 0;
307   const int bufsize = 200;
308   char buffer[bufsize];
309   FILE *fp = fopen(fname, "r");
310   if (fp == NULL) fprintf(stderrstream, "Error : unable to open taskfile <%s
        >\n",fname);
311   else if (list == NULL) fprintf(stderrstream, "Error : list == NULLn");
312   else {
313     while (fscanf(fp,"%s",buffer) != -1) {
314       task_t *ntask = (task_t*)calloc(1,sizeof(task_t));
315       ntask->id = taskcounter++;
316       ntask->hwmod = treeFind(&moduletree,buffer,isitmodulename);
317       ntask->completed = 0;
318       listAdd(list,ntask);
319     }
320   }
321   return taskcounter;
322 }
323
324 int messupint(int _id) {
325   unsigned int id = (unsigned int)_id;
326   unsigned int odds;
327   unsigned int even;
```

```
328     //  printf("Start       : 0x%08X\n",id);
329     odds = id & 0xAAAAAAAA;  // swap 1-bitwise A = 1010
330     even = id & 0x55555555;  //                5 = 0101
331     id = ((odds >> 1) + (even << 1));
332
333     //  printf("1-bitswap   : 0x%08X\n",id);
334     odds = id & 0xCCCCCCCC;  // swap 2-bitwise C = 1100
335     even = id & 0x33333333;  //                3 = 0011
336     id = ((odds >> 2) + (even << 2));
337
338     //  printf("2-bitswap   : 0x%08X\n",id);
339     odds = id & 0xF0F0F0F0;  // swap 4-bitwise F = 1111
340     even = id & 0x0F0F0F0F;
341     id = ((odds >> 4) + (even << 4));
342
343     //  printf("4-bitswap   : 0x%08X\n",id);
344     odds = id & 0xFF00FF00;  // swap bytewise
345     even = id & 0x00FF00FF;
346     id = ((odds >> 8) + (even << 8));
347
348     //  printf("8-bitswap   : 0x%08X\n",id);
349     odds = id & 0xFFFF0000;  // swap 2-bytewise
350     even = id & 0x0000FFFF;
351     id = ((odds >> 16) + (even << 16));
352
353     //  printf("16-bitswap  : 0x%08X\n",id);
354     return (int)((odds >> 16) + (even << 16));
355 }
```

C.1.23 utils.h

```
1  /*
2   * Headerfil med prototyper for endel ekstrafunksjoner som blir
3   * benyttet i systemet og i endel tillegsprogrammer
4   */
5  #ifndef _fcache_utils_h_
6  #define _fcache_utils_h_
7
8  #include <stdio.h>
9  #include "tree.h"
10 #include "list.h"
11 #include "hwtypes.h"
12
13 #define _indent(indent) int indentcounter;for (indentcounter=0;indentcounter
       <indent;indentcounter++)printf(" ");fflush(stdout)
14
15 extern FILE *stderrstream;
16 extern FILE *stdoutstream;
17 extern FILE *logstream;
18
19
20 typedef struct libfileentry_t {
21   char modulename[HWMODNAMELENGTH]; // Must be \0 - terminated
22   int startoffset;
23   int numbofvariants;
24 } libfileentry_t;
25
26 void setErrStream(FILE *errstrm);
27 void setOutStream(FILE *outstrm);
28 void setLogStream(FILE *logstrm);
29
```

```
30 int isitmodulename(void *module, void *modname);
31 int modulecmp(void *mod1, void *mod2);
32 int isitmoduleid(void *module, void *id);
33 int isittaskwithmoduleid(void *task,void*id);
34
35 void printModule(HWModule_t *hwmod, int indent);
36 void modulenodeprint(tree_t *node, int indent);
37
38 int loadTaskFile(char *fname, list_t *list);
39
40 int messupint(int _id);
41
42
43 #endif
```

C.1.24 gen.sh

```
1 #!/bin/bash
2 ./gentasks 16 Sum0 Sum1 Sum2 Sum3 Sum4 Sum5 Sum6 Sum7 Exp0 Exp1 Exp2 Exp3
      Exp4 Exp5 Exp6 Exp7   1024 1024 1024 1024 1024 1024 1024 1024 4 4 4 4 4
      4 4 4    100 100 100 100 100 100 100 100 4 4 4 4 4 4 4 4    1 1 1 1 1 1
      1 1 1 1 1 1 1 1 1 1   1 2 4 8 16 32 64 128 1 2 4 8 16 32 64 128 $1 >
      btaskfile.txt
3
4 ./gentasks 4 Sum0 Sum1 Exp0 Exp1 1024 1024 4 4 100 100 4 4 1 1 1 1 2 4 2 4
      $1 > staskfile.txt
```

C.1.25 run.sh

```
1 #!/bin/bash
2
3
4 case $2 in
5     "" ) prog="./nalletest";;
6     * ) prog=$2;;
7 esac
8
9 case $3 in
10    "" ) start=1;;
11    * ) start=$3;;
12 esac
13
14
15 echo using prog $prog
16 echo starting on $start
17
18 case $1 in
19     aatr )
20         for ((i=$start;i<=1024;i*=2)); do
21             arg="staskfile.txt libfile.dat -allowduplicates -
                   howschedupdateallsimilar -taskforwarding -
                   schedregisteronforwarded -maxforwarddepth $i"
22             echo running $prog $arg
23             echo generated with $prog $arg \> AaTr$i.log > logs/AaTr$i.log
24             $prog $arg >> logs/AaTr$i.log
25         done;;
26
27     aatn )
28         for ((i=$start;i<=1024;i*=2)); do
```

```
29              arg="staskfile.txt libfile.dat -allowduplicates -
                    howschedupdateallsimilar -taskforwarding -
                    no_schedregisteronforwarded -maxforwarddepth $i"
30              echo running $prog $arg
31              echo generated with $prog $arg \> AaTnr$i.log > logs/AaTnr$i.log
32              $prog $arg >> logs/AaTnr$i.log
33          done;;
34
35      aant )
36          arg="staskfile.txt libfile.dat -allowduplicates -
                howschedupdateallsimilar -no_taskforwarding"
37          echo running $prog $arg
38          echo genrated with $prog $arg \> AanT.log > logs/AanT.log
39          $prog $arg >> logs/AanT.log;;
40
41
42
43      aotr )
44          for ((i=$start;i<=1024;i*=2)); do
45              arg="staskfile.txt libfile.dat -allowduplicates -
                    howschedupdateonlythis -taskforwarding -
                    schedregisteronforwarded -maxforwarddepth $i"
46              echo running $prog $arg
47              echo generated with $prog $arg \> AoTr$i.log > logs/AoTr$i.log
48              $prog $arg >> logs/AoTr$i.log
49          done;;
50
51      aotn )
52          for ((i=$start;i<=1024;i*=2)); do
53              arg="staskfile.txt libfile.dat -allowduplicates -
                    howschedupdateonlythis -taskforwarding -
                    no_schedregisteronforwarded -maxforwarddepth $i"
54              echo running $prog $arg
55              echo generate with $prog $arg $i \> AoTnr$i.log > logs/AoTnr$i.
                    log
56              $prog $arg >> logs/AoTnr$i.log
57          done;;
58
59      aont )
60          arg="staskfile.txt libfile.dat -allowduplicates -
                howschedupdateonlythis -no_taskforward"
61          echo running $prog $arg
62          echo generated with $prog $arg \> AonT.log > logs/AonT.log
63          $prog $arg >> logs/AonT.log;;
64
65
66
67      nastr )
68          for ((i=$start;i<=1024;i*=2)); do
69              arg="staskfile.txt libfile.dat -no_allowduplicates -
                    taskforwarding -schedregisteronforwarded -maxforwarddepth $i
                    "
70              echo running $prog $arg
71              echo generated with $prog $arg \> nAsTr$i.log > logs/nAsTr$i.log
72              $prog $arg >> logs/nAsTr$i.log
73          done;;
74
75      nastn )
76          for ((i=$start;i<=1024;i*=2)); do
77              arg="staskfile.txt libfile.dat -no_allowduplicates -
                    taskforwarding -no_schedregisteronforwarded -maxforwarddepth
                     $i"
```

```
78              echo running $prog $arg
79              echo generated with $prog $arg \> nAsTnr$i.log > logs/nAsTnr$i.
                    log
80              $prog $arg >> logs/nAsTnr$i.log
81          done;;
82
83      nasnt )
84          arg="staskfile.txt libfile.dat -no_allowduplicates -
                no_taskforwarding"
85          echo running $prog $arg
86          echo generated with $prog $arg \> nAsnT.log > logs/nAsnT.log
87          $prog $arg >> logs/nAsnT$i.log;;
88
89
90
91      nabtr )
92          for ((i=$start;i<=1024;i*=2)); do
93              arg="btaskfile.txt libfile2.dat -no_allowduplicates -
                    taskforwarding -schedregisteronforwarded -maxforwarddepth $i
                    "
94              echo running $prog $arg
95              echo generated with $prog $arg \> nAbTr$i.log > logs/nAbTr$i.log
96              $prog $arg >> logs/nAbTr$i.log
97          done;;
98
99      nabtn )
100         for ((i=$start;i<=1024;i*=2)); do
101             arg="btaskfile.txt libfile2.dat -no_allowduplicates -
                    taskforwarding -no_schedregisteronforwarded -maxforwarddepth
                     $i"
102             echo running $prog $arg
103             echo generated with $prog $arg \> nAbTnr$i.log > logs/nAbTnr$i.
                    log
104             $prog $arg >> logs/nAbTnr$i.log
105        done;;
106
107     nabnt )
108         arg="btaskfile.txt libfile2.dat -no_allowduplicates -
                no_taskforwarding"
109         echo running $prog $arg
110         echo run with $prog $arg \> nAbnT.log > logs/nAbnT.log
111         $prog $arg >> logs/nAbnT.log;;
112
113     * )
114         echo need to specify which run
115         echo currently :
116         echo - aatr  \(allowdup, allsim, taskforw, regforw, 1-1024\)
117         echo - aatn  \(allowdup, allsim, taskforw, noregforw, 1-1024\)
118         echo - aant  \(allowdup, allsim, notaskforw\)
119         echo - aotr  \(allowpup, onlythis, taskforw, regforw, 1-1024\)
120         echo - aotn  \(allowdup, onlythis, taskforw, noregforw, 1-1024\)
121         echo - aont  \(allowdup, onlythis, notaskforw\)
122         echo - nastr \(noallowdup,smallsys,taskforw,regforw,1-1024\)
123         echo - nastn \(noallowdup,smallsys,taskforw,noreg,1-1024\)
124         echo - nasnt \(noallowdup,smallsys,notaskforw\)
125         echo - nabtr \(noallowdup,bigsys,taskforw,regforw,1-1024\)
126         echo - nabtn \(noallowdup,bigsys,taskforw,noreg,1-1024\)
127         echo - nabnt \(noallowdup,bigsys,notaskforw\)
128
129         ;;
130 esac
```

C.2 Maskinvare

C.2.1 com.vhd

```
1  -- Implementasjon av kommunikasjonsmodul
2
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.std_logic_unsigned.all;
6  use work.common.all;
7
8  entity com_t is
9
10   port (
11     pciBusy       : in    std_logic;
12     pciEmpty      : in    std_logic;
13     pciRW         : out   std_logic;
14     pciEnable     : out   std_logic;
15     pciData       : inout std_logic_vector(wordsize-1 downto 0);
16
17     rst           : in    std_logic;
18     clk           : in    std_logic;
19
20     status        : out   std_logic_vector(wordsize-1 downto 0);
21
22     mod0in      : in  modportin;
23     mod1in      : in  modportin;
24     mod2in      : in  modportin;
25     mod3in      : in  modportin;
26     mod0select  : out std_logic_vector(3 downto 0);
27     mod1select  : out std_logic_vector(3 downto 0);
28     mod2select  : out std_logic_vector(3 downto 0);
29     mod3select  : out std_logic_vector(3 downto 0);
30     mod0enable  : out std_logic;
31     mod1enable  : out std_logic;
32     mod2enable  : out std_logic;
33     mod3enable  : out std_logic;
34     mod0out     : out modportout;
35     mod1out     : out modportout;
36     mod2out     : out modportout;
37     mod3out     : out modportout);
38
39 --     comstate    : out comstate_t );
40
41 end com_t;
42
43 architecture communication of com_t is
44   type comstate_t is (idle,readcom, interpretcom,
45                       update_sendcommand, update_getcommandack,
                            update_getstatus,
46                       sendStatus, status_done,
47                       r_readslotnr, r_setslotnr, r_readwcount, r_readwords,
                            r_readword, r_clockcount, r_done,
48                       l_readslotnr, l_setslotnr, l_readwcount, l_sendcommand
                            , l_getcommandack, l_sendwcount,
49                       l_getwcountack, l_readwords, l_readword, l_sendword,
                            l_getwordack, l_done,
50
51                       start_readslotnr, start_setslotnr, start_getcommandack
                            , start_done,
```

```
52                          stop_readslotnr, stop_setslotnr, stop_getcommandack,
                                stop_done,
53                          res_readslotnr, res_setslotnr, res_getcommandack,
                                res_getcommandreack, res_getwcount,
54                          res_getwcountreack, res_getword, res_getwordreack,
                                res_sendword
55
56                          );
57    signal state       : comstate_t;
58    signal count       : std_logic_vector(wordsize-1 downto 0);
59    signal bitcount : std_logic_vector(wordsize-1 downto 0);
60    signal wcount    : std_logic_vector(wordsize-1 downto 0);
61    signal data         : std_logic_vector(wordsize-1 downto 0);
62    signal s_status     : std_logic_vector(31 downto 0);
63    signal modin : modportin;
64    signal modout : modportout;
65    signal slotnr : std_logic_vector(1 downto 0);
66
67
68  begin  -- communication
69
70    status <= s_status;
71
72    s_status(slot0statehigh downto slot0statelow) <= mod0in.state;
73    s_status(slot1statehigh downto slot1statelow) <= mod1in.state;
74    s_status(slot2statehigh downto slot2statelow) <= mod2in.state;
75    s_status(slot3statehigh downto slot3statelow) <= mod3in.state;
76
77    with slotnr select
78      modin <=
79      mod0in when "00",
80      mod1in when "01",
81      mod2in when "10",
82      mod3in when others;
83
84    updatemodouts: process(modout, slotnr, rst)
85    begin  -- process updatemodports
86      if rst = '1' then
87        mod0out.ce <= '0'; mod0out.output <= (others => '0'); mod0out.rst <=
              '0';
88        mod1out.ce <= '0'; mod1out.output <= (others => '0'); mod1out.rst <=
              '0';
89        mod2out.ce <= '0'; mod2out.output <= (others => '0'); mod2out.rst <=
              '0';
90        mod3out.ce <= '0'; mod3out.output <= (others => '0'); mod3out.rst <=
              '0';
91      else
92        case slotnr is
93          when "00" =>
94            mod0out <= modout;
95          when "01" =>
96            mod1out <= modout;
97          when "10" =>
98            mod2out <= modout;
99          when "11" =>
100           mod3out <= modout;
101         when others => null;
102       end case;
103     end if;
104   end process updatemodouts;
105
106   updatedone: process (mod0in.done, mod1in.done, mod2in.done, mod3in.done, rst)
```

```
107     begin  -- process updatedone
108       if rst = '1' then
109         s_status(mod0donebit) <= '0'; s_status(mod1donebit) <= '0';
110         s_status(mod2donebit) <= '0'; s_status(mod3donebit) <= '0';
111       else
112          if mod0in.done = '1' then s_status(mod0donebit) <= '1';
113          else s_status(mod0donebit) <= '0'; end if;
114          if mod1in.done = '1' then s_status(mod1donebit) <= '1';
115          else s_status(mod1donebit) <= '0'; end if;
116          if mod2in.done = '1' then s_status(mod2donebit) <= '1';
117          else s_status(mod2donebit) <= '0'; end if;
118          if mod3in.done = '1' then s_status(mod3donebit) <= '1';
119          else s_status(mod3donebit) <= '0'; end if;
120       end if;
121     end process updatedone;
122
123
124     tick: process (clk, rst)
125       variable oldcount : std_logic_vector(wordsize-1 downto 0);
126     begin  -- process tick
127       if rst = '1' then                    -- asynchronous reset (active high)
128         state <= idle;
129         slotnr <= "00";
130         s_status <=  resetstatusword;
131         pciEnable <= '1';                  -- active low
132         pciRW <= '0';
133         pciData <= (others => 'Z');
134         modout.output <= (others => '0');
135         modout.ce <= '0';
136         modout.rst <= '0';
137         mod0select <= "0000"; mod0enable <= '0';
138         mod1select <= "0000"; mod1enable <= '0';
139         mod2select <= "0000"; mod2enable <= '0';
140         mod3select <= "0000"; mod3enable <= '0';
141
142
143
144       elsif clk'event and clk = '1' then  -- rising clock edge
145         pciEnable <= '1';
146         pciRW <= '0';
147         pciData <= (others => 'Z');
148
149         case state is
150           when idle =>
151             s_status(statehigh downto statelow) <= "00000000";
152             modout.output <= (others => '0');
153             modout.ce <= '0';
154
155             count <= (others => '0');
156
157             if pciEmpty = '0' then
158               pciRW <= '0';                -- read
159               pciEnable <= '0';            -- enable interface
160               state <= readcom;
161             end if;
162
163           when readcom =>
164             s_status(statehigh downto statelow) <= "00000001";
165             data <= pciData;
166             state <= interpretcom;
167
168           when interpretcom =>
```

```
169             s_status(statehigh downto statelow) <= "00000011";
170             case data(commandbus-1 downto 0) is
171               when command_getStatus =>   -- getStatus
172                 state <= sendStatus;
173               when command_reconfSlot =>  -- reconfSlot
174                 if pciEmpty = '0' then
175                   pciEnable <= '0';         -- enable interface
176                   pciRW <= '0';             -- read
177                   state <= r_readslotnr;
178                 end if;
179               when command_loadTask =>    -- loadTask
180                 if pciEmpty = '0' then
181                   pciEnable <= '0';         -- enable interface
182                   pciRW <= '0';             -- read
183                   state <= l_readslotnr;
184                 end if;
185               when command_getResult =>   -- getResult
186                 if pciEmpty = '0' then
187                   pciEnable <= '0';         -- enable interface
188                   pciRW <= '0';             -- read
189                   state <= res_readslotnr;
190                 end if;
191               when command_startTask =>   -- startTask
192                 if pciEmpty = '0' then
193                   pciEnable <= '0';         -- enable interface
194                   pciRW <= '0';             -- read
195                   state <= start_readslotnr;
196                 end if;
197               when command_stopTask =>    -- stopTask
198                 if pciEmpty = '0' then
199                   pciEnable <= '0';         -- enable interface
200                   pciRW <= '0';             -- read
201                   state <= stop_readslotnr;
202                 end if;
203
204               when others => state <= idle;
205             end case;
206
207 ------------------------------------------------------------------------------
208
209           when sendStatus =>
210             s_status(statehigh downto statelow) <= "00000100";
211             if pciBusy = '0' then
212               pciEnable <= '0';         -- enable interface
213               pciRW <= '1';             -- write
214               pciData <= s_status;
215               state <= idle;
216             end if;
217 ------------------------------------------------------------------------------
218
219           when r_readslotnr =>
220             s_status(statehigh downto statelow) <= "00000101";
221             data <= pciData;
222             state <= r_setslotnr;
223
224
225           when r_setslotnr =>
226             s_status(statehigh downto statelow) <= "00000110";
227             slotnr <= data(1 downto 0);
228             if pciEmpty = '0' then
229               pciEnable <= '0';
230               pciRW <= '0';
```

```
231             state <= r_readwcount;
232
233           end if;
234
235         when r_readwcount =>
236           s_status(statehigh downto statelow) <= "00000111";
237           wcount <= pciData;
238           state <= r_readwords;
239
240         when r_readwords =>
241           s_status(statehigh downto statelow) <= "00001000";
242           if count = wcount then
243             state <= r_done;
244             modout.rst <= '1';
245           elsif pciEmpty = '0' then
246             pciEnable <= '0';
247             pciRW <= '0';
248             state <= r_readword;
249           end if;
250
251         when r_readword =>
252           data <= pciData;
253           count <= count +1;
254           bitcount <= (others => '0');
255           state <= r_clockcount;
256
257         when r_clockcount =>
258           s_status(statehigh downto statelow) <= "00001001";
259           if count = 1 then
260             case slotnr is
261               when "00"   => mod0select <= data(3 downto 0); mod0enable <=
                      '1';
262               when "01"   => mod1select <= data(3 downto 0); mod1enable <=
                      '1';
263               when "10"   => mod2select <= data(3 downto 0); mod2enable <=
                      '1';
264               when others => mod3select <= data(3 downto 0); mod3enable <=
                      '1';
265             end case;
266           end if;
267           if bitcount = clockstoreconfoneword then
268             state <= r_readwords;
269           else
270             bitcount <= bitcount +1;
271           end if;
272
273         when r_done =>
274           s_status(statehigh downto statelow) <= "00001010";
275           if pciBusy = '0' then
276             modout.rst <= '0';
277             pciEnable <= '0';
278             pciRW <= '1';
279             pciData <= reconfigdoneword;
280             s_status(mod0configured+conv_integer(slotnr)) <= '1';
281             state <= idle;
282           end if;
283
284 ------------------------------------------------------------------------------
285
286         when l_readslotnr =>
287           s_status(statehigh downto statelow) <= "00001011";
288           data <= pciData;
```

```
289             state <= l_setslotnr;
290
291           when l_setslotnr =>
292             s_status(statehigh downto statelow) <= "00001110";
293             slotnr <= data(1 downto 0);
294             if pciEmpty = '0' then
295               pciEnable <= '0';
296               pciRW <= '0';
297               state <= l_readwcount;
298             end if;
299
300           when l_readwcount =>
301             s_status(statehigh downto statelow) <= "00001111";
302             wcount <= pciData;
303             count <= (others => '0');
304             state <= l_sendcommand;
305
306           when l_sendcommand =>
307             s_status(statehigh downto statelow) <= "00010000";
308             if modin.ack = '0' then
309               modout.output(wordsize-1 downto commandbus) <= (others => '0');
310               modout.output(commandbus-1 downto 0) <= module_loadTask;
311               modout.ce <= '1';
312               state <= l_getcommandack;
313             end if;
314
315           when l_getcommandack =>
316             s_status(statehigh downto statelow) <= "00010001";
317             if modin.ack = '1' then
318               modout.ce <= '0';
319               state <= l_sendwcount;
320             end if;
321
322           when l_sendwcount =>
323             s_status(statehigh downto statelow) <= "00010010";
324             if modin.ack = '0' then
325               modout.output <= wcount;
326               modout.ce <= '1';
327               state <= l_getwcountack;
328             end if;
329
330           when l_getwcountack =>
331             s_status(statehigh downto statelow) <= "00010011";
332             if modin.ack = '1' then
333               modout.ce <= '0';
334               state <= l_readwords;
335             end if;
336
337           when l_readwords =>
338             s_status(statehigh downto statelow) <= "00010100";
339             if count = wcount then
340               state <= l_done;
341             elsif pciEmpty = '0' then
342               pciEnable <= '0';
343               pciRW <= '0';
344               state <= l_readword;
345             end if;
346
347           when l_readword =>
348             s_status(statehigh downto statelow) <= "00010101";
349             modout.output <= pciData;
350             count <= count +1;
```

```
351             state <= l_sendword;
352
353           when l_sendword =>
354             s_status(statehigh downto statelow) <= "00010110";
355             if modin.ack = '0' then
356               modout.ce <= '1';
357               state <= l_getwordack;
358             end if;
359
360           when l_getwordack =>
361             s_status(statehigh downto statelow) <= "00010111";
362             if modin.ack = '1' then
363               modout.ce <= '0';
364               state <= l_readwords;
365             end if;
366
367           when l_done =>
368             s_status(statehigh downto statelow) <= "00011000";
369             if pciBusy = '0' then
370               pciEnable <= '0';
371               pciRW <= '1';
372               pciData <= loadtaskdoneword;
373               state <= idle;
374             end if;
375
376 ------------------------------------------------------------------------------
377 -- Ikke nødvendig å bruke.
378 -- Modulen stater automatiskt opp når data er lastet inn.
379           when start_readslotnr =>
380             s_status(statehigh downto statelow) <= "00011001";
381             data <= pciData;
382             state <= start_setslotnr;
383
384           when start_setSlotnr =>
385             s_status(statehigh downto statelow) <= "00011010";
386             slotnr <= data(1 downto 0);
387             if modin.ack = '0' then
388               modout.output(wordsize-1 downto commandbus) <= (others => '0');
389               modout.output(commandbus-1 downto 0) <= module_start;
390               modout.ce <= '1';
391               state <= start_getcommandack;
392             end if;
393
394           when start_getcommandack =>
395             s_status(statehigh downto statelow) <= "00011011";
396             modout.ce <= '0';
397             if modin.ack = '1' then
398               state <= start_done;
399             end if;
400
401           when start_done =>
402             s_status(statehigh downto statelow) <= "00011100";
403             if pciBusy = '0' then
404               pciEnable <= '0';
405               pciRW <= '1';
406               pciData <= taskstartedword;
407               state <= idle;
408             end if;
409
410 ------------------------------------------------------------------------------
411 -- Ser ikke helt grunnen til å ha denne med, men hvorfor ikke ...:)
412           when stop_readslotnr =>
```

```
413             s_status(statehigh downto statelow) <= "00011101";
414             data <= pciData;
415             state <= stop_setslotnr;
416
417           when stop_setslotnr =>
418             s_status(statehigh downto statelow) <= "00011110";
419             slotnr <= data(1 downto 0);
420             if modin.ack = '0' then
421               modout.output(wordsize-1 downto commandbus) <= (others => '0');
422               modout.output(commandbus-1 downto 0) <= module_stop;
423               modout.ce <= '1';
424               state <= stop_getcommandack;
425             end if;
426
427           when stop_getcommandack =>
428             s_status(statehigh downto statelow) <= "00011111";
429             if modin.ack = '1' then
430               modout.ce <= '0';
431               state <= stop_done;
432             end if;
433
434           when stop_done =>
435             s_status(statehigh downto statelow) <= "00100000";
436             if pciBusy = '0' then
437               pciEnable <= '0';
438               pciRW <= '1';
439               pciData <= taskstoppedword;
440               state <= idle;
441             end if;
442 ------------------------------------------------------------------------------
443
444           when res_readslotnr =>
445             s_status(statehigh downto statelow) <= "00100001";
446             data <= pciData;
447             state <= res_setslotnr;
448
449           when res_setslotnr =>
450             s_status(statehigh downto statelow) <= "00100010";
451             slotnr <= data(1 downto 0);
452             if modin.ack = '0' then
453               modout.output(wordsize-1 downto commandbus) <= (others => '0');
454               modout.output(commandbus-1 downto 0) <= module_getResult;
455               modout.ce <= '1';
456               state <= res_getcommandack;
457             end if;
458
459           when res_getcommandack =>
460             s_status(statehigh downto statelow) <= "00100011";
461             if modin.ack = '1' then
462               modout.ce <= '0';
463               state <= res_getcommandreack;
464             end if;
465
466           when res_getcommandreack =>
467             s_status(statehigh downto statelow) <= "00100100";
468             if modin.ack = '0' then
469               state <= res_getwcount;
470             end if;
471
472           when res_getwcount =>
473             s_status(statehigh downto statelow) <= "00100101";
474             if modin.ack = '1' then
```

```
475               count <= (others => '0');
476               wcount <= modin.input;
477               modout.ce <= '1';
478               state <= res_getwcountreack;
479             end if;
480
481           when res_getwcountreack =>
482             s_status(statehigh downto statelow) <= "00100110";
483             if modin.ack = '0' then
484               modout.ce <= '0';
485               state <= res_getword;
486             end if;
487
488           when res_getword =>
489             s_status(statehigh downto statelow) <= "00100111";
490             if count = wcount then
491               state <= idle;
492             elsif modin.ack = '1' then
493               data <= modin.input;
494               modout.ce <= '1';
495               count <= count +1;
496               state <= res_getwordreack;
497             end if;
498
499           when res_getwordreack =>
500             s_status(statehigh downto statelow) <= "00101000";
501             if modin.ack = '0' then
502               modout.ce <= '0';
503               state <= res_sendword;
504             end if;
505
506           when res_sendword =>
507             s_status(statehigh downto statelow) <= "00101001";
508             if pciBusy = '0' then
509               pciEnable <= '0';
510               pciRW <= '1';
511               pciData <= data;
512               state <= res_getword;
513             end if;
514
515
516
517
518
519
520 ------------------------------------------------------------------------------
521
522         when others => state <= idle;
523       end case;
524
525
526     end if;
527   end process tick;
528
529 end communication;
```

C.2.2 common.vhd

```
1 -- Datastruktur og typedefinisjoner som er felles for systemet
2
3 library ieee;
```

```
4  use ieee.std_logic_1164.all;
5
6  package common is
7
8    constant wordsize : integer := 32;
9    constant buffersize : integer := 64;
10   constant clockstoreconfoneword : integer := 8;--wordsize*2;
11   constant wordzero : std_logic_vector(wordsize-1 downto 0) := (others =>
         '0');
12   type bufferarray is array (0 to buffersize-1) of std_logic_vector(wordsize
         -1 downto 0);
13 --  type bufferindexvector is array (9 downto 0) of std_logic;
14   subtype bufferindex is integer range 0 to buffersize-1;
15
16   type modvectorarray_t is array (0 to 15) of std_logic_vector(wordsize-1
       downto 0);
17   type modstatearray_t is array (0 to 15) of std_logic_vector(3 downto 0);
18   type modportin is record
19                       input  : std_logic_vector(wordsize-1 downto 0);
20                       ack    : std_logic;
21                       state  : std_logic_vector(3 downto 0);
22                       done   : std_logic;
23                     end record;
24   type modportout is record
25                        output : std_logic_vector(wordsize-1 downto 0);
26                        ce     : std_logic;
27                        rst    : std_logic;
28                      end record;
29
30 -- statusword :
31   -- comstatebits 8 bits
32   -- reserved 16 bits
33   -- configured 4 bits
34   -- done 4 bits
35
36   constant mod0donebit     : integer :=  0;
37   constant mod1donebit     : integer :=  1;
38   constant mod2donebit     : integer :=  2;
39   constant mod3donebit     : integer :=  3;
40   constant mod0configured  : integer :=  4;
41   constant mod1configured  : integer :=  5;
42   constant mod2configured  : integer :=  6;
43   constant mod3configured  : integer :=  7;
44   constant slot0statelow   : integer :=  8;
45   constant slot0statehigh  : integer := 11;
46   constant slot1statelow   : integer := 12;
47   constant slot1statehigh  : integer := 15;
48   constant slot2statelow   : integer := 16;
49   constant slot2statehigh  : integer := 19;
50   constant slot3statelow   : integer := 20;
51   constant slot3statehigh  : integer := 23;
52   constant statelow        : integer := 24;
53   constant statehigh       : integer := 31;
54
55
56   -- set the donebits to Z since they will be set somewhere else.
57   -- Same with the slotXstate-bits
58   constant resetstatusword : std_logic_vector(31 downto 0)
59     := "00000000ZZZZZZZZZZZZZZZZ0000ZZZZ";
60
61   -- returned from commodule after action.
62   constant reconfigdoneword : std_logic_vector(wordsize-1 downto 0) :=
```

```
63      "00000000000000000000000000000000";
64    constant loadtaskdoneword : std_logic_vector(wordsize-1 downto 0) :=
65      "00001111000011110000111100001111";
66    constant taskstartedword  : std_logic_vector(wordsize-1 downto 0) :=
67      "00110011001100110011001100110011";
68    constant taskstoppedword  : std_logic_vector(wordsize-1 downto 0) :=
69      "11001100110011001100110011001100";
70    constant moduledoneword   : std_logic_vector(wordsize-1 downto 0) :=
71      "11111111111111111111111111111111";
72
73    constant commandbus : integer := 3;
74    constant command_getStatus : std_logic_vector(commandbus-1 downto 0)
75      := "000";
76    constant command_reconfSlot : std_logic_vector(commandbus-1 downto 0)
77      := "001";
78    constant command_loadTask : std_logic_vector(commandbus-1 downto 0)
79      := "010";
80    constant command_getResult : std_logic_vector(commandbus-1 downto 0)
81      := "011";
82    constant command_startTask : std_logic_vector(commandbus-1 downto 0)
83      := "100";
84    constant command_stopTask : std_logic_vector(commandbus-1 downto 0)
85      := "101";
86
87    constant module_start    : std_logic_vector(commandbus-1 downto 0)
88      := "000";
89    constant module_stop     : std_logic_vector(commandbus-1 downto 0)
90      := "001";
91    constant module_done     : std_logic_vector(commandbus-1 downto 0)
92      := "010";
93    constant module_loadTask : std_logic_vector(commandbus-1 downto 0)
94      := "011";
95    constant module_getResult : std_logic_vector(commandbus-1 downto 0)
96      := "100";
97
98
99
100   component com_t
101     port (
102       pciBusy       : in    std_logic;
103       pciEmpty      : in    std_logic;
104       pciRW         : out   std_logic;
105       pciEnable     : out   std_logic;
106       pciData       : inout std_logic_vector(wordsize-1 downto 0);
107
108       rst           : in    std_logic;
109       clk           : in    std_logic;
110
111       status        : out   std_logic_vector(wordsize-1 downto 0);
112
113       mod0in      : in  modportin;
114       mod1in      : in  modportin;
115       mod2in      : in  modportin;
116       mod3in      : in  modportin;
117       mod0select : out std_logic_vector(3 downto 0);
118       mod1select : out std_logic_vector(3 downto 0);
119       mod2select : out std_logic_vector(3 downto 0);
120       mod3select : out std_logic_vector(3 downto 0);
121       mod0enable : out std_logic;
122       mod1enable : out std_logic;
123       mod2enable : out std_logic;
124       mod3enable : out std_logic;
```

```
125         mod0out     : out modportout;
126         mod1out     : out modportout;
127         mod2out     : out modportout;
128         mod3out     : out modportout);
129 --        comstate    : out comstate_t );
130     end component;
131
132     component comwrap
133       port (
134         rst          : in    std_logic;
135         clk          : in    std_logic;
136         pciBusy      : in    std_logic;
137         pciEmpty     : in    std_logic;
138         pciRW        : out   std_logic;
139         pciEnable    : out   std_logic;
140         pciData      : inout std_logic_vector(31 downto 0);
141         startreconf  : out   std_logic;
142         reconfdone   : out   std_logic);
143     end component;
144
145     component modmux_t
146       port (
147         sel       : in  std_logic_vector(3 downto 0);
148         enable    : in  std_logic;
149         con_ce    : in  std_logic;
150         con_rst   : in  std_logic;
151         con_ack   : out std_logic;
152         con_done  : out std_logic;
153         con_state : out std_logic_vector(3 downto 0);
154         con_in    : in  std_logic_vector(wordsize-1 downto 0);
155         con_out   : out std_logic_vector(wordsize-1 downto 0);
156
157         mod_ce    : out std_logic_vector(0 to 15);
158         mod_rst   : out std_logic_vector(0 to 15);
159         mod_ack   : in  std_logic_vector(0 to 15);
160         mod_done  : in  std_logic_vector(0 to 15);
161         mod_state : in  modstatearray_t;
162         mod_in    : in  modvectorarray_t;
163         mod_out   : out modvectorarray_t);
164     end component;
165
166     component modmuxwrap
167       port (
168         modul_ce      : out std_logic_vector(0 to 15);
169         modul_ack     : in  std_logic_vector(0 to 15);
170         modul0_input  : out std_logic_vector(wordsize-1 downto 0);
171         modul1_input  : out std_logic_vector(wordsize-1 downto 0);
172         modul2_input  : out std_logic_vector(wordsize-1 downto 0);
173         modul3_input  : out std_logic_vector(wordsize-1 downto 0);
174         modul4_input  : out std_logic_vector(wordsize-1 downto 0);
175         modul5_input  : out std_logic_vector(wordsize-1 downto 0);
176         modul6_input  : out std_logic_vector(wordsize-1 downto 0);
177         modul7_input  : out std_logic_vector(wordsize-1 downto 0);
178         modul8_input  : out std_logic_vector(wordsize-1 downto 0);
179         modul9_input  : out std_logic_vector(wordsize-1 downto 0);
180         modulA_input  : out std_logic_vector(wordsize-1 downto 0);
181         modulB_input  : out std_logic_vector(wordsize-1 downto 0);
182         modulC_input  : out std_logic_vector(wordsize-1 downto 0);
183         modulD_input  : out std_logic_vector(wordsize-1 downto 0);
184         modulE_input  : out std_logic_vector(wordsize-1 downto 0);
185         modulF_input  : out std_logic_vector(wordsize-1 downto 0);
186         modul0_output : in  std_logic_vector(wordsize-1 downto 0);
```

```
187         modul1_output : in  std_logic_vector(wordsize-1 downto 0);
188         modul2_output : in  std_logic_vector(wordsize-1 downto 0);
189         modul3_output : in  std_logic_vector(wordsize-1 downto 0);
190         modul4_output : in  std_logic_vector(wordsize-1 downto 0);
191         modul5_output : in  std_logic_vector(wordsize-1 downto 0);
192         modul6_output : in  std_logic_vector(wordsize-1 downto 0);
193         modul7_output : in  std_logic_vector(wordsize-1 downto 0);
194         modul8_output : in  std_logic_vector(wordsize-1 downto 0);
195         modul9_output : in  std_logic_vector(wordsize-1 downto 0);
196         modulA_output : in  std_logic_vector(wordsize-1 downto 0);
197         modulB_output : in  std_logic_vector(wordsize-1 downto 0);
198         modulC_output : in  std_logic_vector(wordsize-1 downto 0);
199         modulD_output : in  std_logic_vector(wordsize-1 downto 0);
200         modulE_output : in  std_logic_vector(wordsize-1 downto 0);
201         modulF_output : in  std_logic_vector(wordsize-1 downto 0);
202
203         con_ce        : in  std_logic_vector(0 to 3);
204         con_ack       : out std_logic_vector(0 to 3);
205         con0_input    : out std_logic_vector(wordsize-1 downto 0);
206         con1_input    : out std_logic_vector(wordsize-1 downto 0);
207         con2_input    : out std_logic_vector(wordsize-1 downto 0);
208         con3_input    : out std_logic_vector(wordsize-1 downto 0);
209         con0_output   : in  std_logic_vector(wordsize-1 downto 0);
210         con1_output   : in  std_logic_vector(wordsize-1 downto 0);
211         con2_output   : in  std_logic_vector(wordsize-1 downto 0);
212         con3_output   : in  std_logic_vector(wordsize-1 downto 0);
213
214         con0_sel      : in  std_logic_vector(3 downto 0);
215         con1_sel      : in  std_logic_vector(3 downto 0);
216         con2_sel      : in  std_logic_vector(3 downto 0);
217         con3_sel      : in  std_logic_vector(3 downto 0)
218         );
219     end component;
220
221     component sum_t
222       port (
223         input  : in  std_logic_vector(wordsize-1 downto 0);
224         output : out std_logic_vector(wordsize-1 downto 0);
225         ce     : in  std_logic;
226         ack    : out std_logic;
227         done   : out std_logic;
228         state  : out std_logic_vector(3 downto 0);
229         clk    : in  std_logic;
230         rst    : in  std_logic;   -- local reset
231         grst   : in  std_logic ); -- global reset
232     end component;
233
234     component multmod
235       port (
236         A      : in  std_logic_vector(31 downto 0);
237         B      : in  std_logic_vector(31 downto 0);
238         M      : in  std_logic_vector(31 downto 0);
239         clk    : in  std_logic;
240         run    : in  std_logic;
241         done   : out std_logic;
242         R      : out std_logic_vector(31 downto 0) );
243     end component;
244
245     component expmultmod
246       port (
247         A,B,C,M    : in  std_logic_vector(31 downto 0);
248         clk, run   : in  std_logic;
```

```
249         done        : out std_logic;
250         R           : out std_logic_vector(31 downto 0) );
251
252     end component;
253
254     component emmod_t
255       port (
256         input  : in  std_logic_vector(wordsize-1 downto 0);
257         output : out std_logic_vector(wordsize-1 downto 0);
258         ce     : in  std_logic;
259         ack    : out std_logic;
260         done   : out std_logic;
261         state  : out std_logic_vector(3 downto 0);
262         clk    : in  std_logic;
263         rst    : in  std_logic;
264         grst   : in  std_logic);
265     end component;
266
267
268
269
270     component top
271       port (
272   -- common signals
273         rst          : in    std_logic;
274         clk       : in    std_logic;
275
276   -- status signal
277       status       : out std_logic_vector(wordsize-1 downto 0);
278       slot0state   : out std_logic_vector(3 downto 0);
279       slot1state   : out std_logic_vector(3 downto 0);
280       slot2state   : out std_logic_vector(3 downto 0);
281       slot3state   : out std_logic_vector(3 downto 0);
282
283   -- ledsignals
284       leda : out std_logic;     ledb : out std_logic;
285       ledc : out std_logic;     ledd : out std_logic;
286       lede : out std_logic;     ledf : out std_logic;
287       ledg : out std_logic;     ledh : out std_logic;
288       ledi : out std_logic;     ledj : out std_logic;
289       ledk : out std_logic;     ledl : out std_logic;
290       ledm : out std_logic;     ledn : out std_logic;
291       ledo : out std_logic;     ledp : out std_logic;
292
293   -- com signals
294         pciBusy      : in    std_logic;
295         pciEmpty     : in    std_logic;
296         pciRW        : out   std_logic;
297         pciEnable    : out   std_logic;
298         pciData      : inout std_logic_vector(31 downto 0));
299   --       comstate     : out   comstate_t );
300     end component;
301
302   end common;
```

C.2.3 emmod.vhd

```
1   -- Implementasjon av wrapper-modul for eksponensiell multiplikasjon
2   -- modulo (a*b^c % m), dvs en modul som tar seg av kommunikasjon med
3   -- kommunikasjonsmodulen
4
```

```
5  library ieee;
6  use ieee.std_logic_1164.all;
7  use work.common.all;
8
9  entity emmod_t is
10
11   port (
12     input  : in  std_logic_vector(wordsize-1 downto 0);
13     output : out std_logic_vector(wordsize-1 downto 0);
14     ce     : in  std_logic;
15     ack    : out std_logic;
16     done   : out std_logic;
17     state  : out std_logic_vector(3 downto 0);
18     clk    : in  std_logic;
19     rst    : in  std_logic;
20     grst   : in  std_logic);
21
22 end emmod_t;
23
24 architecture emmod_impl_1 of emmod_t is
25
26   type state_t is (nostate, getcom, interpret, getwcount,getwcountreack,
         geta, getareack, getb, getbreack, getc, getcreack, getm, getmreack,
         sendwcount, getwcountack, sendres, getresack, wait_for_ce_low);
27   signal constate : state_t;
28   signal command : std_logic_vector(commandbus-1 downto 0);
29
30   signal A, B, C, M, R : std_logic_vector(31 downto 0);
31   signal moduledone, startmodule : std_logic;
32
33 begin  -- emmod_impl_1
34
35   expmultmod1 : expmultmod port map (
36     A     => A,
37     B     => B,
38     C     => C,
39     M     => M,
40     clk   => clk,
41     run   => startmodule,
42     done  => moduledone,
43     R     => R);
44
45
46   doncontrol: process (moduledone,startmodule)
47   begin  -- process doncontrol
48     if moduledone = '1' and startmodule = '1' then
49       done <= '1';
50     else
51       done <= '0';
52     end if;
53   end process doncontrol;
54
55   control: process (clk, grst)
56   begin  -- process control
57     if grst = '1' then                    -- asynchronous reset
58       constate <= nostate;
59       ack <= '0';
60       output <= (others => '0');
61       command <= (others => '0');
62       state <= "0000";
63
64       startmodule <= '0';
```

```
65        A <= (others => '0'); B <= (others => '0');
66        C <= (others => '0'); M <= (others => '0');
67      elsif clk'event and clk = '1' then  -- rising clock edge
68
69        case constate is
70          when nostate =>
71            state <= "0001";
72            if rst = '1' then
73              constate <= getcom;
74            end if;
75
76          when getcom =>
77            state <= "0010";
78            if ce = '1' then
79              command <= input(commandbus-1 downto 0);
80              ack <= '1';
81              constate <= interpret;
82            end if;
83
84          when interpret =>
85            state <= "0011";
86            if ce = '0' then
87              ack <= '0';
88              case command is
89                when module_start =>
90                  constate <= getcom;
91                when module_stop =>
92                  startmodule <= '0';
93                  constate <= getcom;
94                when module_loadTask =>
95                  startmodule <= '0';
96                  constate <= getwcount;
97                when module_getResult =>
98                  constate <= sendwcount;
99                when others => constate <= getcom;
100             end case;
101           end if;
102
103         when getwcount =>
104           state <= "0100";
105           if ce = '1' then
106             ack <= '1';
107             constate <= getwcountreack;
108           end if;
109
110         when getwcountreack =>
111           state <= "0101";
112           if ce = '0' then
113             ack <= '0';
114             constate <= geta;
115           end if;
116
117         when geta =>
118           state <= "0110";
119           if ce = '1' then A <= input; ack <= '1'; constate <= getareack;
                end if;
120
121         when getareack =>
122           state <= "0111";
123           if ce = '0' then ack <= '0'; constate <= getb; end if;
124
125         when getb =>
```

```
126           state <= "1000";
127           if ce = '1' then B <= input; ack <= '1'; constate <= getbreack;
                  end if;
128
129         when getbreack =>
130           state <= "1001";
131           if ce = '0' then ack <= '0'; constate <= getc;  end if;
132
133         when getc =>
134           state <= "1010";
135           if ce = '1' then C <= input; ack <= '1'; constate <= getcreack;
                  end if;
136
137         when getcreack =>
138           state <= "1011";
139           if ce = '0' then ack <= '0'; constate <= getm; end if;
140
141         when getm =>
142           state <= "1100";
143           if ce = '1' then M <= input; ack <= '1'; constate <= getmreack;
                  end if;
144
145         when getmreack =>
146           state <= "1101";
147           if ce = '0' then ack <= '0'; constate <= getcom; startmodule <=
                  '1'; end if;
148
149         when sendwcount =>
150           if ce = '0' then
151             output <= (0 => '1', others => '0');
152             ack <= '1';
153             constate <= getwcountack;
154           end if;
155
156         when getwcountack =>
157           if ce = '1' then
158             ack <= '0';
159             constate <= sendres;
160           end if;
161
162         when sendres =>
163           if ce = '0' then
164             output <= R;
165             ack <= '1';
166             constate <= getresack;
167           end if;
168
169         when getresack =>
170           if ce = '1' then
171             ack <= '0';
172             constate <= wait_for_ce_low;
173           end if;
174
175         when wait_for_ce_low =>
176           if ce = '0' then
177             constate <= getcom;
178           end if;
179
180
181       end case;
182
183     end if;
```

```
184   end process control;
185
186 end emmod_impl_1;
```

C.2.4 expmultmod.vhd

```
1  -- Implementasjon av eksponensiell multiplikasjon modulo
2  -- (a*b^c % m).
3
4  library ieee;
5  use ieee.std_logic_1164.all;
6  use ieee.std_logic_unsigned.all;
7  use ieee.std_logic_arith.all;
8  use work.common.all;
9
10
11
12 entity expmultmod is
13
14   port (
15     A,B,C,M     : in  std_logic_vector(31 downto 0);
16     clk, run    : in  std_logic;
17     done        : out std_logic;
18     R           : out std_logic_vector(31 downto 0) );
19
20
21 end expmultmod;
22
23 architecture expmultmod_impl_1 of expmultmod is
24   signal sa,sb,sc,sm : std_logic_vector(31 downto 0);
25   signal sna, snb,snc : std_logic_vector(31 downto 0);
26   signal stmpb : std_logic_vector(31 downto 0);
27   constant zero : std_logic_vector(31 downto 0) := (others => '0');
28   type state_t is (running,setvalues,waiting,finished);
29   signal state : state_t;
30 --  signal clk_i : std_logic;
31
32   signal startmmod1,startmmod2,donemmod1,donemmod2 : std_logic;
33 begin  -- expmultmod_impl_1
34
35 --  clk_i <= not clk;
36
37
38   mmod1 : multmod port map (
39     A     => sa,
40     B     => stmpb,
41     M     => sm,
42     clk   => clk,
43     run => startmmod1,
44     done  => donemmod1,
45     R     => sna);
46
47   mmod2 : multmod port map (
48     A     => sb,
49     B     => sB,
50     M     => sm,
51     clk   => clk,
52     run   => startmmod2,
53     done  => donemmod2,
54     R     => snb);
55
```

```
56
57    tick: process (clk, run)
58    begin  -- process tick
59      if run = '0' then                   -- asynchronous reset
60        sa <= A; sb <= B; sc <= C; sm <= M; done <= '0';
61        state <= running;
62        startmmod1 <= '0'; startmmod2 <= '0';
63      elsif clk'event and clk = '1' then  -- rising clock edge
64
65        case state is
66          when running =>
67            if sc = zero then
68              R <= sa; state <= finished;
69            else
70              if sc(0) = '1' then
71                stmpb <= sb;
72              else
73                stmpb <= conv_std_logic_vector(1,32);
74              end if;
75              snc <= shr(sc, "1");
76              state <= setvalues;
77            end if;
78
79          when setvalues =>
80              startmmod1 <= '1';
81              startmmod2 <= '1';
82              state <= waiting;
83
84          when waiting =>
85            sc <= snc;
86            if donemmod1 = '1' and donemmod2 = '1' then
87              sa <= sna; sb <= snb;
88              startmmod1 <= '0'; startmmod2 <= '0';
89              state <= running;
90            end if;
91
92          when finished =>
93            done <= '1';
94
95        end case;
96      end if;
97    end process tick;
98
99
100   end expmultmod_impl_1;
```

C.2.5 modmux.vhd

```
1   -- Implementasjon av buss-struktur som tar seg av ruting mellom
        modulslottene
2   -- og de forskjellige fredigkonfigurerte modulene i systemet.
3
4   library ieee;
5   use ieee.std_logic_1164.all;
6   use ieee.std_logic_unsigned.all;
7   use ieee.std_logic_arith.all;
8   use work.common.all;
9
10  entity modmux_t is
11
12    port (
```

```
13      sel      : in  std_logic_vector(3 downto 0);
14      enable   : in  std_logic;
15      con_ce   : in  std_logic;
16      con_rst  : in  std_logic;
17      con_ack  : out std_logic;
18      con_done : out std_logic;
19      con_state : out std_logic_vector(3 downto 0);
20      con_in   : in  std_logic_vector(wordsize-1 downto 0);
21      con_out  : out std_logic_vector(wordsize-1 downto 0);
22
23      mod_ce   : out std_logic_vector(0 to 15);
24      mod_rst  : out std_logic_vector(0 to 15);
25      mod_ack  : in  std_logic_vector(0 to 15);
26      mod_done : in std_logic_vector(0 to 15);
27      mod_state : in modstatearray_t;
28      mod_in   : in  modvectorarray_t;
29      mod_out  : out modvectorarray_t );
30
31  end modmux_t;
32
33  architecture modmux_impl_1 of modmux_t is
34
35  begin  -- modmux_impl_1
36
37    con_out <= mod_in(0)  when (sel = "0000" and enable = '1') else
38               mod_in(1)  when (sel = "0001" and enable = '1') else
39               mod_in(2)  when (sel = "0010" and enable = '1') else
40               mod_in(3)  when (sel = "0011" and enable = '1') else
41               mod_in(4)  when (sel = "0100" and enable = '1') else
42               mod_in(5)  when (sel = "0101" and enable = '1') else
43               mod_in(6)  when (sel = "0110" and enable = '1') else
44               mod_in(7)  when (sel = "0111" and enable = '1') else
45               mod_in(8)  when (sel = "1000" and enable = '1') else
46               mod_in(9)  when (sel = "1001" and enable = '1') else
47               mod_in(10) when (sel = "1010" and enable = '1') else
48               mod_in(11) when (sel = "1011" and enable = '1') else
49               mod_in(12) when (sel = "1100" and enable = '1') else
50               mod_in(13) when (sel = "1101" and enable = '1') else
51               mod_in(14) when (sel = "1110" and enable = '1') else
52               mod_in(15) when (sel = "1111" and enable = '1') else
53               (others => '0');
54    con_ack <= mod_ack(0)  when (sel = "0000" and enable = '1') else
55               mod_ack(1)  when (sel = "0001" and enable = '1') else
56               mod_ack(2)  when (sel = "0010" and enable = '1') else
57               mod_ack(3)  when (sel = "0011" and enable = '1') else
58               mod_ack(4)  when (sel = "0100" and enable = '1') else
59               mod_ack(5)  when (sel = "0101" and enable = '1') else
60               mod_ack(6)  when (sel = "0110" and enable = '1') else
61               mod_ack(7)  when (sel = "0111" and enable = '1') else
62               mod_ack(8)  when (sel = "1000" and enable = '1') else
63               mod_ack(9)  when (sel = "1001" and enable = '1') else
64               mod_ack(10) when (sel = "1010" and enable = '1') else
65               mod_ack(11) when (sel = "1011" and enable = '1') else
66               mod_ack(12) when (sel = "1100" and enable = '1') else
67               mod_ack(13) when (sel = "1101" and enable = '1') else
68               mod_ack(14) when (sel = "1110" and enable = '1') else
69               mod_ack(15) when (sel = "1111" and enable = '1') else
70               '0';
71    con_done <= mod_done(0)  when (sel = "0000" and enable = '1') else
72                mod_done(1)  when (sel = "0001" and enable = '1') else
73                mod_done(2)  when (sel = "0010" and enable = '1') else
74                mod_done(3)  when (sel = "0011" and enable = '1') else
```

```
75                mod_done(4)  when (sel = "0100" and enable = '1') else
76                mod_done(5)  when (sel = "0101" and enable = '1') else
77                mod_done(6)  when (sel = "0110" and enable = '1') else
78                mod_done(7)  when (sel = "0111" and enable = '1') else
79                mod_done(8)  when (sel = "1000" and enable = '1') else
80                mod_done(9)  when (sel = "1001" and enable = '1') else
81                mod_done(10) when (sel = "1010" and enable = '1') else
82                mod_done(11) when (sel = "1011" and enable = '1') else
83                mod_done(12) when (sel = "1100" and enable = '1') else
84                mod_done(13) when (sel = "1101" and enable = '1') else
85                mod_done(14) when (sel = "1110" and enable = '1') else
86                mod_done(15) when (sel = "1111" and enable = '1') else
87                '0';
88    con_state <= mod_state(0)  when (sel = "0000" and enable = '1') else
89                 mod_state(1)  when (sel = "0001" and enable = '1') else
90                 mod_state(2)  when (sel = "0010" and enable = '1') else
91                 mod_state(3)  when (sel = "0011" and enable = '1') else
92                 mod_state(4)  when (sel = "0100" and enable = '1') else
93                 mod_state(5)  when (sel = "0101" and enable = '1') else
94                 mod_state(6)  when (sel = "0110" and enable = '1') else
95                 mod_state(7)  when (sel = "0111" and enable = '1') else
96                 mod_state(8)  when (sel = "1000" and enable = '1') else
97                 mod_state(9)  when (sel = "1001" and enable = '1') else
98                 mod_state(10) when (sel = "1010" and enable = '1') else
99                 mod_state(11) when (sel = "1011" and enable = '1') else
100                mod_state(12) when (sel = "1100" and enable = '1') else
101                mod_state(13) when (sel = "1101" and enable = '1') else
102                mod_state(14) when (sel = "1110" and enable = '1') else
103                mod_state(15) when (sel = "1111" and enable = '1') else
104                "1111";
105
106   modout_gen: for i in 0 to 15 generate
107     mod_out(i) <= con_in when ((conv_std_logic_vector(i,4) = sel) and enable
          = '1') else (others => 'Z');
108   end generate modout_gen;
109
110   modce_gen: for i in 0 to 15 generate
111     mod_ce(i) <= con_ce when ((conv_std_logic_vector(i,4) = sel) and enable
          = '1') else 'Z';
112   end generate modce_gen;
113
114   modrst_gen: for i in 0 to 15 generate
115     mod_rst(i) <= con_rst when ((conv_std_logic_vector(i,4) = sel) and
          enable = '1') else 'Z';
116   end generate modrst_gen;
117
118 --    with sel select
119 --      mod_out <=
120 --      (0  => con_in, others => (others => 'Z')) when "0000",
121 --      (1  => con_in, others => (others => 'Z')) when "0001",
122 --      (2  => con_in, others => (others => 'Z')) when "0010",
123 --      (3  => con_in, others => (others => 'Z')) when "0011",
124 --      (4  => con_in, others => (others => 'Z')) when "0100",
125 --      (5  => con_in, others => (others => 'Z')) when "0101",
126 --      (6  => con_in, others => (others => 'Z')) when "0110",
127 --      (7  => con_in, others => (others => 'Z')) when "0111",
128 --      (8  => con_in, others => (others => 'Z')) when "1000",
129 --      (9  => con_in, others => (others => 'Z')) when "1001",
130 --      (10 => con_in, others => (others => 'Z')) when "1010",
131 --      (11 => con_in, others => (others => 'Z')) when "1011",
132 --      (12 => con_in, others => (others => 'Z')) when "1100",
133 --      (13 => con_in, others => (others => 'Z')) when "1101",
```

```
134 --      (14 => con_in, others => (others => 'Z')) when "1110",
135 --      (15 => con_in, others => (others => 'Z')) when "1111",
136 --      (others => (others => 'Z'))               when others;
137 --    with sel select
138 --      mod_ce <=
139 --      (0  => con_ce, others => 'Z') when "0000",
140 --      (1  => con_ce, others => 'Z') when "0001",
141 --      (2  => con_ce, others => 'Z') when "0010",
142 --      (3  => con_ce, others => 'Z') when "0011",
143 --      (4  => con_ce, others => 'Z') when "0100",
144 --      (5  => con_ce, others => 'Z') when "0101",
145 --      (6  => con_ce, others => 'Z') when "0110",
146 --      (7  => con_ce, others => 'Z') when "0111",
147 --      (8  => con_ce, others => 'Z') when "1000",
148 --      (9  => con_ce, others => 'Z') when "1001",
149 --      (10 => con_ce, others => 'Z') when "1010",
150 --      (11 => con_ce, others => 'Z') when "1011",
151 --      (12 => con_ce, others => 'Z') when "1100",
152 --      (13 => con_ce, others => 'Z') when "1101",
153 --      (14 => con_ce, others => 'Z') when "1110",
154 --      (15 => con_ce, others => 'Z') when "1111",
155 --      (others => 'Z')               when others;
156 --    with sel select
157 --      mod_rst <=
158 --      (0  => con_rst, others => 'Z') when "0000",
159 --      (1  => con_rst, others => 'Z') when "0001",
160 --      (2  => con_rst, others => 'Z') when "0010",
161 --      (3  => con_rst, others => 'Z') when "0011",
162 --      (4  => con_rst, others => 'Z') when "0100",
163 --      (5  => con_rst, others => 'Z') when "0101",
164 --      (6  => con_rst, others => 'Z') when "0110",
165 --      (7  => con_rst, others => 'Z') when "0111",
166 --      (8  => con_rst, others => 'Z') when "1000",
167 --      (9  => con_rst, others => 'Z') when "1001",
168 --      (10 => con_rst, others => 'Z') when "1010",
169 --      (11 => con_rst, others => 'Z') when "1011",
170 --      (12 => con_rst, others => 'Z') when "1100",
171 --      (13 => con_rst, others => 'Z') when "1101",
172 --      (14 => con_rst, others => 'Z') when "1110",
173 --      (15 => con_rst, others => 'Z') when "1111",
174 --      (others => 'Z')               when others;
175
176
177 end modmux_impl_1;
```

C.2.6 multmod.vhd

```
1  -- Implementasjon av multiplikasjon modulo (a*b % m)
2  -- Dette er en delmodul av eksponensiell multiplikasjon modulo
3  -- (a*b^c % m)
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7  use ieee.std_logic_unsigned.all;
8  use ieee.std_logic_arith.all;
9
10 entity multmod is
11
12   port (
13     A       : in  std_logic_vector(31 downto 0);
14     B       : in  std_logic_vector(31 downto 0);
```

```
15      M       : in  std_logic_vector(31 downto 0);
16      clk     : in  std_logic;
17      run     : in  std_logic;
18      done    : out std_logic;
19      R       : out std_logic_vector(31 downto 0) );
20
21  end multmod;
22
23  architecture multmod_impl_1 of multmod is
24    signal sa,sb,sm : std_logic_vector(31 downto 0);
25    signal sr : std_logic_vector(32 downto 0);
26    constant zero : std_logic_vector(31 downto 0) := (others => '0');
27  begin  -- multmod_impl_1
28
29    tick: process (clk, run)
30      variable va : std_logic_vector(31 downto 0);
31      variable vr : std_logic_vector(32 downto 0);
32    begin  -- process tick
33      if run = '0' then                 -- asynchronous reset (active low)
34        sa <= A;
35        sb <= B;
36        sm <= M;
37        sr <= (others => '0');
38        done <= '1';
39        R <= (others => 'Z');
40      elsif clk'event and clk = '0' then  -- falling clock edge
41        if sb = zero then
42          R <= sr(31 downto 0);
43          done <= '1';
44        else
45          done <= '0';
46          if (sb(0) = '1') then
47            vr := sr+sa;
48            if (vr >= sm) then
49              sr <= vr-sm;
50            else
51              sr <= vr;
52            end if;
53          end if;
54
55          va := shl(sa,"1");
56          if (va >= sm) then
57            sa <= va - sm;
58          else
59            sa <= va;
60          end if;
61
62          sb <= shr(sb,"1");
63
64        end if;
65      end if;
66    end process tick;
67
68
69  end multmod_impl_1;
```

C.2.7 sum.vhd

```
1  -- Modul for summering av tall.
2  -- Tar seg av kommunikasjon med kommunikasjonsmodulen selv.
3
```

```
4  library ieee;
5  use ieee.std_logic_1164.all;
6  use ieee.std_logic_unsigned.all;
7  use work.common.all;
8
9  entity sum_t is
10
11   port (
12     input   : in  std_logic_vector(wordsize-1 downto 0);
13     output  : out std_logic_vector(wordsize-1 downto 0);
14     ce      : in  std_logic;
15     ack     : out std_logic;
16     done    : out std_logic;
17     state   : out std_logic_vector(3 downto 0);
18     clk     : in  std_logic;
19     rst     : in  std_logic;
20     grst    : in  std_logic);
21
22 end sum_t;
23
24 architecture sum_impl_1 of sum_t is
25   constant localbufsize : integer := 256;
26   signal sum_sendt : std_logic;
27   signal sum : std_logic_vector(wordsize-1 downto 0);
28
29   type controlstate_t is (nostate, getcom, interpret, senddone, getwcount,
         readwords, readword, sendwcount, getwcountack, sendword, getwordack);
30   signal controlstate : controlstate_t;
31
32   signal command : std_logic_vector(commandbus-1 downto 0);
33   signal wcount : std_logic_vector(wordsize-1 downto 0);
34   signal count : std_logic_vector(wordsize-1 downto 0);
35
36
37 begin  -- sum16_impl_1
38
39   control: process (clk, grst)
40   begin  -- process dummy
41     if grst = '1' then                      -- asynchronous globalreset
42       controlstate <= nostate;
43       ack <= '0';
44       output <= (others => '0');
45       sum <= (others => '0');
46       done <= '0';
47       sum_sendt <= '0';
48       command <= (others => '0');
49       wcount <= (others => '0');
50       count <= (others => '0');
51       state <= "0000";
52     elsif clk'event and clk = '1' then     -- rising clock edge
53       case controlstate is
54         when nostate =>
55           state <= "0010";
56           if rst = '1' then
57             controlstate <= getcom;
58           end if;
59
60         when getcom =>
61           state <= "0011";
62           if ce = '1' then
63             command <= input(commandbus-1 downto 0);
64             ack <= '1';
```

```
65              controlstate <= interpret;
66            end if;
67
68          when interpret =>
69            state <= "0100";
70            if ce = '0' then
71              ack <= '0';
72              case command is
73                when module_start =>
74                  controlstate <= getcom;
75                when module_stop =>
76                  controlstate <= getcom;
77                when module_loadTask =>
78                  done <= '0';
79                  controlstate <= getwcount;
80                when module_getResult =>
81                  controlstate <= sendwcount;
82                when others =>
83                  controlstate <= getcom;
84              end case;
85            end if;
86
87          when getwcount =>
88            state <= "0101";
89            if ce = '1' then
90              wcount <= input;
91              count <= (others => '0');
92              ack <= '1';
93              controlstate <= readwords;
94              sum <= (others => '0');
95            end if;
96
97          when readwords =>
98            state <= "0110";
99            if ce = '0' then
100             ack <= '0';
101             if count = wcount then
102               done <= '1';
103               controlstate <= getcom;
104             else
105               controlstate <= readword;
106             end if;
107           end if;
108
109         when readword =>
110           state <= "0111";
111           if ce = '1' then
112             count <= count +1;
113             controlstate <= readwords;
114             sum <= sum + input;
115             ack <= '1';
116           end if;
117
118         when sendwcount =>
119           state <= "1000";
120           if ce = '0' then
121             output <= (0=>'1', others => '0');
122             ack <= '1';
123             sum_sendt <= '0';
124             controlstate <= getwcountack;
125           end if;
126
```

```
127           when getwcountack =>
128             state <= "1001";
129             if ce = '0' then
130               ack <= '0';
131               controlstate <= sendword;
132             end if;
133
134           when sendword =>
135             state <= "1010";
136             if ce = '0' then
137               if sum_sendt = '1' then
138                 controlstate <= getcom;
139               else
140                 output <= sum;
141                 ack <= '1';
142                 controlstate <= getwordack;
143               end if;
144             end if;
145
146           when getwordack =>
147             state <= "1011";
148             if ce = '1' then
149               ack <= '0';
150               sum_sendt <= '1';
151               controlstate <= sendword;
152             end if;
153
154           when others => state <= "1100";
155         end case;
156       end if;
157     end process control;
158
159 end sum_impl_1;
```

C.2.8 top.vhd

```
1  -- Toplevel modul for det rekonfigurerbare systemet.
2
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.std_logic_unsigned.all;
6  use work.common.all;
7
8  entity top is
9
10   port (
11 -- common signals
12     rst          : in     std_logic;
13     clk       : in     std_logic;
14
15 -- status signal
16     status       : out std_logic_vector(wordsize-1 downto 0);
17     slot0state : out std_logic_vector(3 downto 0);
18     slot1state : out std_logic_vector(3 downto 0);
19     slot2state : out std_logic_vector(3 downto 0);
20     slot3state : out std_logic_vector(3 downto 0);
21
22 -- ledsignals
23     leda : out std_logic;     ledb : out std_logic;
24     ledc : out std_logic;     ledd : out std_logic;
25     lede : out std_logic;     ledf : out std_logic;
```

```
26      ledg : out std_logic;     ledh : out std_logic;
27      ledi : out std_logic;     ledj : out std_logic;
28      ledk : out std_logic;     ledl : out std_logic;
29      ledm : out std_logic;     ledn : out std_logic;
30      ledo : out std_logic;     ledp : out std_logic;
31
32  -- com signals
33      pciBusy      : in    std_logic;
34      pciEmpty     : in    std_logic;
35      pciRW        : out   std_logic;
36      pciEnable    : out   std_logic;
37      pciData      : inout std_logic_vector(31 downto 0));
38
39  --     comstate     : out   comstate_t );
40
41  end top;
42
43  architecture top of top is
44    signal rst_i : std_logic;
45    signal s_status : std_logic_vector(wordsize-1 downto 0);
46
47    signal sel0 : std_logic_vector(3 downto 0);
48    signal sel1 : std_logic_vector(3 downto 0);
49    signal sel2 : std_logic_vector(3 downto 0);
50    signal sel3 : std_logic_vector(3 downto 0);
51
52    signal mod0in : modportin;  signal mod0out : modportout;
53    signal mod1in : modportin;  signal mod1out : modportout;
54    signal mod2in : modportin;  signal mod2out : modportout;
55    signal mod3in : modportin;  signal mod3out : modportout;
56
57    signal modul_ce    : std_logic_vector(0 to 15);
58    signal modul_rst   : std_logic_vector(0 to 15);
59    signal modul_ack   : std_logic_vector(0 to 15);
60    signal modul_done  : std_logic_vector(0 to 15);
61    signal modul_state : modstatearray_t;
62    signal mod_input   : modvectorarray_t;
63    signal mod_output  : modvectorarray_t;
64
65    signal slot0enable : std_logic;  signal slot1enable : std_logic;
66    signal slot2enable : std_logic;  signal slot3enable : std_logic;
67
68
69
70  begin  -- top
71
72    rst_i <= not rst;
73    status <= s_status;
74    slot0state <= mod0in.state;
75    slot1state <= mod1in.state;
76    slot2state <= mod2in.state;
77    slot3state <= mod3in.state;
78
79    leda <= s_status(statehigh);
80    ledc <= s_status(statehigh-1);
81    lede <= s_status(statehigh-2);
82    ledg <= s_status(statehigh-3);
83    ledi <= s_status(statelow+3);
84    ledk <= s_status(statelow+2);
85    ledm <= s_status(statelow+1);
86    ledo <= s_status(statelow);
87
```

```
88    ledb <= s_status(slot3statehigh);
89    ledd <= s_status(slot3statehigh -1);
90    ledf <= s_status(slot3statelow+1);
91    ledh <= s_status(slot3statelow);
92    ledj <= s_status(mod2configured);
93    ledl <= s_status(mod3configured);
94    ledn <= s_status(mod2donebit);
95    ledp <= s_status(mod3donebit);
96
97
98
99    com1 : com_t port map (
100     pciData      => pciData,
101     pciBusy      => pciBusy,
102     pciEmpty     => pciEmpty,
103     pciRW        => pciRW,
104     pciEnable    => pciEnable,
105     rst          => rst_i,
106     clk          => clk,
107     status       => s_status,
108     mod0select => sel0,
109     mod1select => sel1,
110     mod2select => sel2,
111     mod3select => sel3,
112     mod0enable => slot0enable,
113     mod1enable => slot1enable,
114     mod2enable => slot2enable,
115     mod3enable => slot3enable,
116     mod0in => mod0in,
117     mod1in => mod1in,
118     mod2in => mod2in,
119     mod3in => mod3in,
120     mod0out => mod0out,
121     mod1out => mod1out,
122     mod2out => mod2out,
123     mod3out => mod3out);--,
124 --    comstate => comstate);
125
126   slot0 : modmux_t port map (
127     sel      => sel0,
128     enable   => slot0enable,
129     con_ce   => mod0out.ce,
130     con_rst  => mod0out.rst,
131     con_ack  => mod0in.ack,
132     con_done => mod0in.done,
133     con_state => mod0in.state,
134     con_in   => mod0out.output,
135     con_out  => mod0in.input,
136     mod_ce   => modul_ce,
137     mod_rst  => modul_rst,
138     mod_ack  => modul_ack,
139     mod_done => modul_done,
140     mod_state => modul_state,
141     mod_in   => mod_output,
142     mod_out  => mod_input);
143   slot1 : modmux_t port map (
144     sel      => sel1,
145     enable   => slot1enable,
146     con_ce   => mod1out.ce,
147     con_rst  => mod1out.rst,
148     con_ack  => mod1in.ack,
149     con_done => mod1in.done,
```

```
150       con_state => mod1in.state,
151       con_in  => mod1out.output,
152       con_out => mod1in.input,
153       mod_ce  => modul_ce,
154       mod_rst => modul_rst,
155       mod_ack => modul_ack,
156       mod_done => modul_done,
157       mod_state => modul_state,
158       mod_in  => mod_output,
159       mod_out => mod_input);
160     slot2 : modmux_t port map (
161       sel     => sel2,
162       enable  => slot2enable,
163       con_ce  => mod2out.ce,
164       con_rst => mod2out.rst,
165       con_ack => mod2in.ack,
166       con_done => mod2in.done,
167       con_state => mod2in.state,
168       con_in  => mod2out.output,
169       con_out => mod2in.input,
170       mod_ce  => modul_ce,
171       mod_rst => modul_rst,
172       mod_ack => modul_ack,
173       mod_done => modul_done,
174       mod_state => modul_state,
175       mod_in  => mod_output,
176       mod_out => mod_input);
177     slot3 : modmux_t port map (
178       sel     => sel3,
179       enable  => slot3enable,
180       con_ce  => mod3out.ce,
181       con_rst => mod3out.rst,
182       con_ack => mod3in.ack,
183       con_done => mod3in.done,
184       con_state => mod3in.state,
185       con_in  => mod3out.output,
186       con_out => mod3in.input,
187       mod_ce  => modul_ce,
188       mod_rst => modul_rst,
189       mod_ack => modul_ack,
190       mod_done => modul_done,
191       mod_state => modul_state,
192       mod_in  => mod_output,
193       mod_out => mod_input);
194
195     mod0 : sum_t    port map
196       (mod_input(0),  mod_output(0),  modul_ce(0),  modul_ack(0),  modul_done
              (0),  modul_state(0),  clk, modul_rst(0),  rst_i);
197     mod1 : sum_t    port map
198       (mod_input(1),  mod_output(1),  modul_ce(1),  modul_ack(1),  modul_done
              (1),  modul_state(1),  clk, modul_rst(1),  rst_i);
199     mod2 : emmod_t port map
200       (mod_input(2),  mod_output(2),  modul_ce(2),  modul_ack(2),  modul_done
              (2),  modul_state(2),  clk, modul_rst(2),  rst_i);
201     mod3 : emmod_t port map
202       (mod_input(3),  mod_output(3),  modul_ce(3),  modul_ack(3),  modul_done
              (3),  modul_state(3),  clk, modul_rst(3),  rst_i);
203     mod4 : sum_t    port map
204       (mod_input(4),  mod_output(4),  modul_ce(4),  modul_ack(4),  modul_done
              (4),  modul_state(4),  clk, modul_rst(4),  rst_i);
205     mod5 : sum_t    port map
```

```
206       (mod_input(5), mod_output(5), modul_ce(5), modul_ack(5), modul_done
              (5), modul_state(5), clk, modul_rst(5), rst_i);
207     mod6 : emmod_t port map
208       (mod_input(6), mod_output(6), modul_ce(6), modul_ack(6), modul_done
              (6), modul_state(6), clk, modul_rst(6), rst_i);
209     mod7 : emmod_t port map
210       (mod_input(7), mod_output(7), modul_ce(7), modul_ack(7), modul_done
              (7), modul_state(7), clk, modul_rst(7), rst_i);
211     mod8 : sum_t   port map
212       (mod_input(8), mod_output(8), modul_ce(8), modul_ack(8), modul_done
              (8), modul_state(8), clk, modul_rst(8), rst_i);
213     mod9 : sum_t   port map
214       (mod_input(9), mod_output(9), modul_ce(9), modul_ack(9), modul_done
              (9), modul_state(9), clk, modul_rst(9), rst_i);
215     modA : emmod_t port map
216       (mod_input(10), mod_output(10), modul_ce(10), modul_ack(10), modul_done
              (10), modul_state(10), clk, modul_rst(10), rst_i);
217     modB : emmod_t port map
218       (mod_input(11), mod_output(11), modul_ce(11), modul_ack(11), modul_done
              (11), modul_state(11), clk, modul_rst(11), rst_i);
219     modC : sum_t   port map
220       (mod_input(12), mod_output(12), modul_ce(12), modul_ack(12), modul_done
              (12), modul_state(12), clk, modul_rst(12), rst_i);
221     modD : sum_t   port map
222       (mod_input(13), mod_output(13), modul_ce(13), modul_ack(13), modul_done
              (13), modul_state(13), clk, modul_rst(13), rst_i);
223     modE : emmod_t port map
224       (mod_input(14), mod_output(14), modul_ce(14), modul_ack(14), modul_done
              (14), modul_state(14), clk, modul_rst(14), rst_i);
225     modF : emmod_t port map
226       (mod_input(15), mod_output(15), modul_ce(15), modul_ack(15), modul_done
              (15), modul_state(15), clk, modul_rst(15), rst_i);
227
228
229
230
231   end top;
```

C.3 Andre filer

C.3.1 libfile.dat

```
00000000  04 00 00 00 4e 75 6d 2e  20 6f 66 20 4d 6f 64 2e  |....Num. of Mod.|
00000010  53 75 6d 30 00 00 00 00  00 00 00 00 00 00 00 00  |Sum0............|
00000020  00 00 00 00 00 00 00 00  10 03 00 00 04 00 00 00  |................|
00000030  53 75 6d 31 00 00 00 00  00 00 00 00 00 00 00 00  |Sum1............|
00000040  00 00 00 00 00 00 00 00  50 03 00 00 04 00 00 00  |........P.......|
00000050  45 78 70 30 00 00 00 00  00 00 00 00 00 00 00 00  |Exp0............|
00000060  00 00 00 00 00 00 00 00  90 03 00 00 04 00 00 00  |................|
00000070  45 78 70 31 00 00 00 00  00 00 00 00 00 00 00 00  |Exp1............|
00000080  00 00 00 00 00 00 00 00  d0 03 00 00 04 00 00 00  |................|
00000090  41 62 6f 76 65 3a 20 45  61 63 68 20 65 6e 74 72  |Above: Each entr|
000000a0  79 20 69 73 20 4e 61 6d  65 28 32 34 29 2c 73 74  |y is Name(24),st|
000000b0  61 72 74 28 34 29 2c 65  6e 74 69 72 65 73 28 34  |art(4),entires(4|
000000c0  29 20 41 6c 6c 20 76 61  6c 75 65 73 20 69 6e 20  |) All values in |
000000d0  4c 65 61 73 74 20 53 69  67 2e 20 42 79 74 65 20  |Least Sig. Byte |
000000e0  46 69 72 73 74 20 28 4c  53 42 46 29 00 00 00 00  |First (LSBF)....|
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000200  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000210  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000220  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000230  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000240  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000250  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000260  54 68 69 73 20 70 72 6f  74 6f 74 79 70 65 20 6f  |This prototype o|
00000270  66 20 61 20 6c 69 62 66  69 6c 65 20 68 61 76 65  |f a libfile have|
00000280  20 61 6c 6c 20 62 69 74  73 74 72 65 61 6d 73 20  | all bitstreams |
00000290  61 73 20 30 2d 74 65 72  6d 69 6e 61 74 65 64 20  |as 0-terminated |
000002a0  73 74 72 69 6e 67 73 2e  2e 00 00 00 00 00 00 00  |strings.........|
000002b0  54 68 65 20 62 69 74 73  74 72 65 61 6d 73 20 73  |The bitstreams s|
000002c0  74 61 72 74 73 20 61 66  74 65 72 20 74 68 69 73  |tarts after this|
000002d0  2e 20 46 69 72 73 74 20  77 6f 72 64 20 69 73 20  |. First word is |
000002e0  73 69 7a 65 2c 20 6e 65  78 74 20 69 73 20 61 63  |size, next is ac|
000002f0  74 75 61 6c 6c 20 6d 6f  64 75 6c 20 69 64 2e 20  |tuall modul id. |
00000300  28 4c 73 74 53 69 67 42  79 74 65 46 73 74 29 2e  |(LstSigByteFst).|
00000310  0c 00 00 00 00 00 00 00  53 75 6d 30 2e 30 00 00  |........Sum0.0..|
00000320  0c 00 00 00 04 00 00 00  53 75 6d 30 2e 31 00 00  |........Sum0.1..|
00000330  0c 00 00 00 08 00 00 00  53 75 6d 30 2e 32 00 00  |........Sum0.2..|
00000340  0c 00 00 00 0c 00 00 00  53 75 6d 30 2e 33 00 00  |........Sum0.3..|
00000350  0c 00 00 00 01 00 00 00  53 75 6d 31 2e 30 00 00  |........Sum1.0..|
00000360  0c 00 00 00 05 00 00 00  53 75 6d 31 2e 31 00 00  |........Sum1.1..|
00000370  0c 00 00 00 09 00 00 00  53 75 6d 31 2e 32 00 00  |........Sum1.2..|
00000380  0c 00 00 00 0d 00 00 00  53 75 6d 31 2e 33 00 00  |........Sum1.3..|
00000390  0c 00 00 00 02 00 00 00  45 78 70 30 2e 30 00 00  |........Exp0.0..|
000003a0  0c 00 00 00 06 00 00 00  45 78 70 30 2e 31 00 00  |........Exp0.1..|
000003b0  0c 00 00 00 0a 00 00 00  45 78 70 30 2e 32 00 00  |........Exp0.2..|
000003c0  0c 00 00 00 0e 00 00 00  45 78 70 30 2e 33 00 00  |........Exp0.3..|
000003d0  0c 00 00 00 03 00 00 00  45 78 70 31 2e 30 00 00  |........Exp1.0..|
000003e0  0c 00 00 00 07 00 00 00  45 78 70 31 2e 31 00 00  |........Exp1.1..|
000003f0  0c 00 00 00 0b 00 00 00  45 78 70 31 2e 32 00 00  |........Exp1.2..|
00000400  0c 00 00 00 0f 00 00 00  45 78 70 31 2e 33 00 00  |........Exp1.3..|
00000410  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000420  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000430  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000440  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000450  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000460  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000470  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000480  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000490  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000004a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000004b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

```
000004c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000004d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000004e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000004f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000500  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000510  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000520  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000530  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000540  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000550  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000560  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000570  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000580  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000590
```

C.3.2 lifbile2.dat

```
00000000  10 00 00 00 4e 75 6d 2e  20 6f 66 20 4d 6f 64 2e  |....Num. of Mod.|
00000010  53 75 6d 30 00 00 00 00  00 00 00 00 00 00 00 00  |Sum0............|
00000020  00 00 00 00 00 00 00 00  10 03 00 00 04 00 00 00  |................|
00000030  53 75 6d 31 00 00 00 00  00 00 00 00 00 00 00 00  |Sum1............|
00000040  00 00 00 00 00 00 00 00  50 03 00 00 04 00 00 00  |........P.......|
00000050  45 78 70 30 00 00 00 00  00 00 00 00 00 00 00 00  |Exp0............|
00000060  00 00 00 00 00 00 00 00  90 03 00 00 04 00 00 00  |................|
00000070  45 78 70 31 00 00 00 00  00 00 00 00 00 00 00 00  |Exp1............|
00000080  00 00 00 00 00 00 00 00  d0 03 00 00 04 00 00 00  |................|
00000090  53 75 6d 32 00 00 00 00  00 00 00 00 00 00 00 00  |Sum2............|
000000a0  00 00 00 00 00 00 00 00  10 04 00 00 04 00 00 00  |................|
000000b0  53 75 6d 33 00 00 00 00  00 00 00 00 00 00 00 00  |Sum3............|
000000c0  00 00 00 00 00 00 00 00  50 04 00 00 04 00 00 00  |........P.......|
000000d0  45 78 70 32 00 00 00 00  00 00 00 00 00 00 00 00  |Exp2............|
000000e0  00 00 00 00 00 00 00 00  90 04 00 00 04 00 00 00  |................|
000000f0  45 78 70 33 00 00 00 00  00 00 00 00 00 00 00 00  |Exp3............|
00000100  00 00 00 00 00 00 00 00  d0 04 00 00 04 00 00 00  |................|
00000110  53 75 6d 34 00 00 00 00  00 00 00 00 00 00 00 00  |Sum4............|
00000120  00 00 00 00 00 00 00 00  10 05 00 00 04 00 00 00  |................|
00000130  53 75 6d 35 00 00 00 00  00 00 00 00 00 00 00 00  |Sum5............|
00000140  00 00 00 00 00 00 00 00  50 05 00 00 04 00 00 00  |........P.......|
00000150  45 78 70 34 00 00 00 00  00 00 00 00 00 00 00 00  |Exp4............|
00000160  00 00 00 00 00 00 00 00  90 05 00 00 04 00 00 00  |................|
00000170  45 78 70 35 00 00 00 00  00 00 00 00 00 00 00 00  |Exp5............|
00000180  00 00 00 00 00 00 00 00  d0 05 00 00 04 00 00 00  |................|
00000190  53 75 6d 36 00 00 00 00  00 00 00 00 00 00 00 00  |Sum6............|
000001a0  00 00 00 00 00 00 00 00  10 06 00 00 04 00 00 00  |................|
000001b0  53 75 6d 37 00 00 00 00  00 00 00 00 00 00 00 00  |Sum7............|
000001c0  00 00 00 00 00 00 00 00  50 06 00 00 04 00 00 00  |........P.......|
000001d0  45 78 70 36 00 00 00 00  00 00 00 00 00 00 00 00  |Exp6............|
000001e0  00 00 00 00 00 00 00 00  90 06 00 00 04 00 00 00  |................|
000001f0  45 78 70 37 00 00 00 00  00 00 00 00 00 00 00 00  |Exp7............|
00000200  00 00 00 00 00 00 00 00  d0 06 00 00 04 00 00 00  |................|
00000210  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000220  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000230  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000240  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000250  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000260  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000270  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000280  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000290  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000002a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000002b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000002c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000002d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000002e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000002f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000300  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000310  0c 00 00 00 00 00 00 00  53 75 6d 30 2e 30 00 00  |........Sum0.0..|
00000320  0c 00 00 00 00 00 00 00  53 75 6d 30 2e 31 00 00  |........Sum0.1..|
00000330  0c 00 00 00 00 00 00 00  53 75 6d 30 2e 32 00 00  |........Sum0.2..|
00000340  0c 00 00 00 00 00 00 00  53 75 6d 30 2e 33 00 00  |........Sum0.3..|
00000350  0c 00 00 00 01 00 00 00  53 75 6d 31 2e 30 00 00  |........Sum1.0..|
00000360  0c 00 00 00 01 00 00 00  53 75 6d 31 2e 31 00 00  |........Sum1.1..|
00000370  0c 00 00 00 01 00 00 00  53 75 6d 31 2e 32 00 00  |........Sum1.2..|
00000380  0c 00 00 00 01 00 00 00  53 75 6d 31 2e 33 00 00  |........Sum1.3..|
00000390  0c 00 00 00 02 00 00 00  45 78 70 30 2e 30 00 00  |........Exp0.0..|
000003a0  0c 00 00 00 02 00 00 00  45 78 70 30 2e 31 00 00  |........Exp0.1..|
000003b0  0c 00 00 00 02 00 00 00  45 78 70 30 2e 32 00 00  |........Exp0.2..|
000003c0  0c 00 00 00 02 00 00 00  45 78 70 30 2e 33 00 00  |........Exp0.3..|
000003d0  0c 00 00 00 03 00 00 00  45 78 70 31 2e 30 00 00  |........Exp1.0..|
000003e0  0c 00 00 00 03 00 00 00  45 78 70 31 2e 31 00 00  |........Exp1.1..|
000003f0  0c 00 00 00 03 00 00 00  45 78 70 31 2e 32 00 00  |........Exp1.2..|
```

```
00000400  0c 00 00 00 03 00 00 00  45 78 70 31 2e 33 00 00  |........Exp1.3..|
00000410  0c 00 00 00 04 00 00 00  53 75 6d 32 2e 30 00 00  |........Sum2.0..|
00000420  0c 00 00 00 04 00 00 00  53 75 6d 32 2e 31 00 00  |........Sum2.1..|
00000430  0c 00 00 00 04 00 00 00  53 75 6d 32 2e 32 00 00  |........Sum2.2..|
00000440  0c 00 00 00 04 00 00 00  53 75 6d 32 2e 33 00 00  |........Sum2.3..|
00000450  0c 00 00 00 05 00 00 00  53 75 6d 33 2e 30 00 00  |........Sum3.0..|
00000460  0c 00 00 00 05 00 00 00  53 75 6d 33 2e 31 00 00  |........Sum3.1..|
00000470  0c 00 00 00 05 00 00 00  53 75 6d 33 2e 32 00 00  |........Sum3.2..|
00000480  0c 00 00 00 05 00 00 00  53 75 6d 33 2e 33 00 00  |........Sum3.3..|
00000490  0c 00 00 00 06 00 00 00  45 78 70 32 2e 30 00 00  |........Exp2.0..|
000004a0  0c 00 00 00 06 00 00 00  45 78 70 32 2e 31 00 00  |........Exp2.1..|
000004b0  0c 00 00 00 06 00 00 00  45 78 70 32 2e 32 00 00  |........Exp2.2..|
000004c0  0c 00 00 00 06 00 00 00  45 78 70 32 2e 33 00 00  |........Exp2.3..|
000004d0  0c 00 00 00 07 00 00 00  45 78 70 33 2e 30 00 00  |........Exp3.0..|
000004e0  0c 00 00 00 07 00 00 00  45 78 70 33 2e 31 00 00  |........Exp3.1..|
000004f0  0c 00 00 00 07 00 00 00  45 78 70 33 2e 32 00 00  |........Exp3.2..|
00000500  0c 00 00 00 07 00 00 00  45 78 70 33 2e 33 00 00  |........Exp3.3..|
00000510  0c 00 00 00 08 00 00 00  53 75 6d 34 2e 30 00 00  |........Sum4.0..|
00000520  0c 00 00 00 08 00 00 00  53 75 6d 34 2e 31 00 00  |........Sum4.1..|
00000530  0c 00 00 00 08 00 00 00  53 75 6d 34 2e 32 00 00  |........Sum4.2..|
00000540  0c 00 00 00 08 00 00 00  53 75 6d 34 2e 33 00 00  |........Sum4.3..|
00000550  0c 00 00 00 09 00 00 00  53 75 6d 35 2e 30 00 00  |........Sum5.0..|
00000560  0c 00 00 00 09 00 00 00  53 75 6d 35 2e 31 00 00  |........Sum5.1..|
00000570  0c 00 00 00 09 00 00 00  53 75 6d 35 2e 32 00 00  |........Sum5.2..|
00000580  0c 00 00 00 09 00 00 00  53 75 6d 35 2e 33 00 00  |........Sum5.3..|
00000590  0c 00 00 00 0a 00 00 00  45 78 70 34 2e 30 00 00  |........Exp4.0..|
000005a0  0c 00 00 00 0a 00 00 00  45 78 70 34 2e 31 00 00  |........Exp4.1..|
000005b0  0c 00 00 00 0a 00 00 00  45 78 70 34 2e 32 00 00  |........Exp4.2..|
000005c0  0c 00 00 00 0a 00 00 00  45 78 70 34 2e 33 00 00  |........Exp4.3..|
000005d0  0c 00 00 00 0b 00 00 00  45 78 70 35 2e 30 00 00  |........Exp5.0..|
000005e0  0c 00 00 00 0b 00 00 00  45 78 70 35 2e 31 00 00  |........Exp5.1..|
000005f0  0c 00 00 00 0b 00 00 00  45 78 70 35 2e 32 00 00  |........Exp5.2..|
00000600  0c 00 00 00 0b 00 00 00  45 78 70 35 2e 33 00 00  |........Exp5.3..|
00000610  0c 00 00 00 0c 00 00 00  53 75 6d 36 2e 30 00 00  |........Sum6.0..|
00000620  0c 00 00 00 0c 00 00 00  53 75 6d 36 2e 31 00 00  |........Sum6.1..|
00000630  0c 00 00 00 0c 00 00 00  53 75 6d 36 2e 32 00 00  |........Sum6.2..|
00000640  0c 00 00 00 0c 00 00 00  53 75 6d 36 2e 33 00 00  |........Sum6.3..|
00000650  0c 00 00 00 0d 00 00 00  53 75 6d 37 2e 30 00 00  |........Sum7.0..|
00000660  0c 00 00 00 0d 00 00 00  53 75 6d 37 2e 31 00 00  |........Sum7.1..|
00000670  0c 00 00 00 0d 00 00 00  53 75 6d 37 2e 32 00 00  |........Sum7.2..|
00000680  0c 00 00 00 0d 00 00 00  53 75 6d 37 2e 33 00 00  |........Sum7.3..|
00000690  0c 00 00 00 0e 00 00 00  45 78 70 36 2e 30 00 00  |........Exp6.0..|
000006a0  0c 00 00 00 0e 00 00 00  45 78 70 36 2e 31 00 00  |........Exp6.1..|
000006b0  0c 00 00 00 0e 00 00 00  45 78 70 36 2e 32 00 00  |........Exp6.2..|
000006c0  0c 00 00 00 0e 00 00 00  45 78 70 36 2e 33 00 00  |........Exp6.3..|
000006d0  0c 00 00 00 0f 00 00 00  45 78 70 37 2e 30 00 00  |........Exp7.0..|
000006e0  0c 00 00 00 0f 00 00 00  45 78 70 37 2e 31 00 00  |........Exp7.1..|
000006f0  0c 00 00 00 0f 00 00 00  45 78 70 37 2e 32 00 00  |........Exp7.2..|
00000700  0c 00 00 00 0f 00 00 00  45 78 70 37 2e 33 00 00  |........Exp7.3..|
00000710
```

C.3.3 timing.cfg

Denne filen spesifiserer hvor lang tid en partiell rekonfigurering av en av modulene vil ta, oppgitt i nano-sekunder.

```
146310000
```

C.3.4 Makefile for programvare

```
1  CC = gcc
2  LD = gcc
3  CFlags = -Wall
4  LFlags = -lpthread -lrt -ldl -ldimesdl
5  LocalLFlags = -lrt
6  Defines = -Duse_list_mutex -Duse_tree_mutex
7  Debug = -g
8  Optimize = 0
9  Compile = $(CC) -c $(Debug) $(CFlags) $(Defines) -O$(Optimize)
10 Link = $(LD) $(LFlags)
11 LocalLink = $(LD) $(LocalLFlags)
```

```
12  Filelist = hwlib.c hwlib.h list.c list.h Makefile lru.c fas.c mututils.h
        tree.c tree.h test.c utils.c utils.h nalletest.c libfile.dat timing.cfg
        gen.sh gentasks.c logstats.c libfile2.dat hwtypes.h hwglobals.h
        hwglobals.c hwbench.c benera.c run.sh
13
14
15        Libs      = hwlib.o lru.o utils.o list.o tree.o hwglobals.o benera.o
16  StupidLibs      = hwlib.o fas.o utils.o list.o tree.o hwglobals.o benera.o
17        LocalLibs = hwlib.o lru.o utils.o list.o tree.o hwglobals.o bensim.o
18  StupidLocalLibs = hwlib.o fas.o utils.o list.o tree.o hwglobals.o bensim.o
19
20  default: all
21
22  all : test logstats listtest treetest gentasks
23
24  remake : clean all
25
26  %.o : %.c *.h Makefile
27          $(Compile) $<
28
29  test : test.o ${Libs}
30          $(Link) test.o $(Libs) -o $@
31
32  LogstatsLibs = logstats.o tree.o utils.o hwglobals.o list.o
33
34  logstats : $(LogstatsLibs)
35          gcc $(LogstatsLibs) -lrt $(Debug) -O$(Optimize) -o $@
36
37  gentasks : gentasks.o
38          gcc gentasks.o -lrt -o $@
39
40  listtest : listtest.o $(Libs)
41          $(Link) listtest.o $(Libs) -o $@
42
43  treetest : treetest.o
44          $(Link) treetest.o tree.o -o $@
45
46  messuptest : messuptest.o $(LocalLibs)
47          $(LocalLink) messuptest.o $(LocalLibs) -o $@
48
49
50  hwtest :
51          scp hwtest.c Makefile tordanf@nalle.idi.ntnu.no:HF/
52          ssh tordanf@nalle make -C HF hwtestcompile
53
54  hwtestcompile :
55           gcc hwtest.c -o hwtest -ldimesdl -ldl -g -ggdb -O0
56
57  nalle: copytonalle
58          ssh tordanf@nalle make -C HF nalletest gentasks logstats
59
60  copytonalle:
61          scp  $(Filelist) tordanf@nalle:HF/
62
63  nalletest:  nalletest.o $(Libs) $(StupidLibs)
64          $(LD) nalletest.o $(Libs) $(LFlags)  -o $@
65          $(LD) nalletest.o $(StupidLibs) $(LFlags) -o $@stupid
66
67  local : nalletest.o $(LocalLibs) $(StupidLocalLibs)
68          $(LD) nalletest.o $(LocalLibs) $(LocalLFlags)  -o local
69          $(LD) nalletest.o $(StupidLocalLibs) $(LocalLFlags) -o localstupid
70
```

```
71 swbench : swbench.o swlib.o
72         $(LD) swbench.o swlib.o -lrt -o swbench
73
74
75 softwaresol :
76         gcc -o $@ ${@}.c swlib.c -lrt -g -ggdb -O0
77
78
79 clean :
80         rm -rf *.o *.bin *~ *.E
```

C.3.5 Makefile for maskinvare

```
1  # Makefile for synthesis
2  #
3  # Requires: GNU make and other GNU utilities (e.g Cygwin)
4  #           Xilinx ISE
5  #
6  # Asbjørn Djupdal 2003
7
8  # Modified by Tord A. Fredrikse 2005
9
10 # Project name, used for naming generated files
11 PROJECT_NAME = top
12
13 # white space separated list of VHDL-files
14 SOURCE_FILES = common.vhd com.vhd sum.vhd multmod.vhd expmultmod.vhd emmod.
       vhd modmux.vhd top.vhd
15
16 # FPGA to synthesize for
17 SYNT_DEVICE = xcv1000e-6-fg860
18
19 ##############################################################################
       ###
20
21 CONSTRAINTS_FILE = $(PROJECT_NAME).ucf
22 TOP_LEVEL = top
23
24 WINE =
25
26 .PHONY : bit
27 bit : $(PROJECT_NAME).bit
28
29 .PHONY : all
30 all : $(PROJECT_NAME).bit $(PROJECT_NAME)_timing.vhd timing_report.twr
31
32 .PHONY : timing
33 timing : $(PROJECT_NAME)_timing.vhd timing_report.twr
34
35 # synthesis
36
37 $(PROJECT_NAME).ngc : $(SOURCE_FILES)
38         echo $(SOURCE_FILES) | xargs -n 1 echo > tmp.prj
39         echo -n "run " > tmp.scr
40         echo -n "-ifn tmp.prj " >> tmp.scr
41         echo -n "-ifmt VHDL " >> tmp.scr
42         echo -n "-ofn $@ " >> tmp.scr
43         echo -n "-ofmt NGC " >> tmp.scr
44         echo -n "-p $(SYNT_DEVICE) " >> tmp.scr
45         echo -n "-opt_mode Speed " >> tmp.scr
46         echo -n "-opt_level 1 " >> tmp.scr
```

```
47          echo -n "-top $(TOP_LEVEL) " >> tmp.scr
48          echo -n "-hierarchy_separator / " >> tmp.scr
49          echo -n "-register_duplication no " >> tmp.scr
50          echo -n "-keep_hierarchy yes " >> tmp.scr
51          $(WINE) xst -ifn tmp.scr
52
53  # implementation
54
55  $(PROJECT_NAME).ngd : $(PROJECT_NAME).ngc $(CONSTRAINTS_FILE)
56          $(WINE) ngdbuild -p $(SYNT_DEVICE) -u -uc $(CONSTRAINTS_FILE) $< $@
57
58  $(PROJECT_NAME).pcf : $(PROJECT_NAME).ngd
59          $(WINE) map -p $(SYNT_DEVICE) -o map.ncd -u -l $< $@
60
61  $(PROJECT_NAME).ncd : $(PROJECT_NAME).pcf
62          $(WINE) par -w -ol 2 map.ncd $@ $<
63
64  # bitfile generation
65
66  $(PROJECT_NAME).bit : $(PROJECT_NAME).ncd $(PROJECT_NAME).pcf
67          $(WINE) bitgen -b -l -w -g ConfigRate:4 -g DriveDone:No \
68                  -g StartUpClk:JTAGCLK -g DONE_cycle:4 -g GTS_cycle:5 -g GSR_
                        cycle:6 \
69                  -g DonePipe:No -g GWE_cycle:6 -g LCK_cycle:NoWait -g
                        Security:NONE $<
70
71  # timing
72
73  timing_report.twr : $(PROJECT_NAME).ncd $(PROJECT_NAME).pcf
74          $(WINE) trce -v $^ -o $@
75
76  $(PROJECT_NAME).nga : $(PROJECT_NAME).pcf $(PROJECT_NAME).ncd
77          $(WINE) ngdanno -s -6 -o $@ -p $(PROJECT_NAME).pcf $(PROJECT_NAME).
                ncd
78
79  $(PROJECT_NAME)_timing.vhd : $(PROJECT_NAME).nga
80          $(WINE) ngd2vhdl -w $< $@
81
82  # clean
83
84  .PHONY : clean
85  clean :
86          rm -rf $(PROJECT_NAME) *.fst *.log *.bgn *.bit *.bld *.dly *.drc *.
                edf \
87                  *.ll *.msd *.msk *.ncd *.ncf *.ngd *.ngo *.pad *.par *.pcf *
                        .rba *.rbb \
88                  *.rbd *.rbt *.xpi *.nav *.mrp *.ngm *.lst *.out *.cel \
89                  *.twr *.nga time_sim.edn *.alf *.itr tmp.* $(PROJECT_NAME)_
                        timing.* \
90                  *.prj *.ngc *.sdf *.xml *.ngr xst
```