NTNU
Det skapende universitet

FPGA-plattform for AHEAD

Stian Reiersen Arntsen

Master i elektronikk
Oppgaven levert: Juni 2006
Hovedveileder: Kjetil Svarstad, IET

Norges teknisk-naturvitenskapelige universitet
Institutt for elektronikk og telekommunikasjon

Oppgavetekst

Et prosjekt med likelydende tittel har utprøvd en del muligheter for FPGA-plattformer, og man vil nå gå videre med den som ser mest lovende ut, Suzaku. Oppgaven vil være å spesifisere både konfigurering og software-støtte for dette kortet sammen med et prosessor-kort som CerfCube fra Intrinsyc. Muligheter og alternativer skal finnes og analyseres, før en løsning velges for AHEAD og forsøkes implementert. Det skal også spesifiseres hvordan denne løsningen skal interagere med AHEAD tagserver og klient, spesielt mhp. versjoner av AHEAD utover denne første, enkleste versjonen. Et eksempel innen media-streaming skal forsøkes å implementeres. Hvis tiden tillater det, skal det også analyseres hvorvidt en løsning som Suzaku-arkitekturen kan klare seg uten det eksterne prosessorkortet, og utelukkende bruke den interne prosessoren på FPGA-en til å kjøre det nødvendige AHEADspesifikke server-programmet.

Oppgaven gitt: 23. januar 2006
Hovedveileder: Kjetil Svarstad, IET

FPGA-plattform for AHEAD

Stian Reiersen Arntsen

Vår 2006

Oppgavetekst.

Kandidatens navn: Stian Arntsen
Oppgavens tittel: FPGA-plattform for AHEAD

AHEAD står for "Ambient Hardware—Embedded Architectures on Demand" og er en aktivitet på IET i samarbeid med SINTEF IKT Kommunikasjonssystemer. Dette er en spesifikk problemstilling innen "Ambient Intelligence"—intelligente omgivelser—og det går i korthet ut på å etablere en plattform som gjør det mulig å "bære med seg" arkitekturbeskrivelser på lette, mobile plattformer som PDA eller Smartphone, og ved behov å få denne arkitektur overført til og instantiert på tag'er i omgivelsene (lette datamaskiner med FPGA ombord) for så å få eksekvert programmer på denne arkitekturen. Dermed kan man få gjort oppgaver som er altfor tunge eller vanskelige i utgangspunktet for en PDA eller programmerbar tag ved å spesialisere FPGA'en som en co-prosessor inne i tag'en.

Oppgavens tekst:

Et prosjekt med likelydende tittel har utprøvd en del muligheter for FPGA-plattformer, og man vil nå gå videre med den som ser mest lovende ut, Suzaku. Oppgaven vil være å spesifisere både konfigurering og software-støtte for dette kortet sammen med et prosessor-kort som CerfCube fra Intrinsyc. Muligheter og alternativer skal finnes og analyseres, før en løsning velges for AHEAD og forsøkes implementert. Det skal også spesifiseres hvordan denne løsningen skal interagere med AHEAD tagserver og klient, spesielt mhp. versjoner av AHEAD utover denne første, enkleste versjonen. Et eksempel innen media-streaming skal forsøkes å implementeres. Hvis tiden tillater det, skal det også analyseres hvorvidt en løsning som Suzaku-arkitekturen kan klare seg uten det eksterne prosessorkortet, og utelukkende bruke den interne prosessoren på FPGA-en til å kjøre det nødvendige AHEADspesifikke server-programmet.

Faglærer: Kjetil Svarstad, IET

Sammendrag.

Denne rapporten bygger på arbeidet som er gjort i forbindelse med en FPGA plattform for AHEAD prosjektet. Teori og arbeid i rapporten er bygd rundt den valgte FPGA utviklingsplattformen Suzaku-S. Rapporten begynner med litt beskrivelse av AHEAD og systemet, samt en motivasjon med forklaring av hva dette kan brukes til. Videre går en inn på litt teori om hvilke krav som stilles til slike systemer og hvilke av kravene som er tilfredsstilt ved valget av den nevnte utviklingsplattform. Rapporten inneholder videre en dokumentasjon på arbeidet som er utført og en forklaring på hvordan den ferdige versjon 1 av AHEAD plattformen virker. Resultatet er altså en ferdig FPGA-plattform uten ekstern mikroprosessor, der en heller valgte å bruke FPGAens interne prosessor. Plattformen inngår i en verktøykjede som inneholder utviklings-PC, FPGA-plattform og http-server. Mikroprosessoren i FPGAen kjører en tilpasset uClinux som operativsystem. uClinux er tilpasset spesielt denne prosessoren og dette systemet, og er kompilert på utviklings-PCen. FPGA-plattformen som er implementert er en html/script-basert AHEAD-server. Det vil si at plattformen bruker html kode og en webserver som grensesnitt, samt linker til script for å lage funksjonalitet på plattformen som kan styres eksternt. Den endelige FPGA-plattformen implementerer en dynamisk rekonfigurering styrt eksternt, med to forskjellige maskinvare konfigurasjoner. Resultatet av en aritmetisk operasjon er vist i et webservet grensesnitt, der en kan velge å laste ned en adderende maskinvarekonfigurasjon, og en subtraherende maskinvarekonfigurasjon. Resultatet av operasjonen er da selvfølgelig avhengig av hvilken maskinvarekonstruksjon som er lastet ned til FPGAen. En del av dokumentasjonen i rapporten er direkte skrevet for eventuelt videre arbeid med akkurat denne utviklingsplattform og de designverktøy som er brukt. Det er gitt forslag til hva det kan være lurt å jobbe videre med, og hvilke oppgaver som må prioriteres for å komme nærmere et ferdig AHEAD system. Til slutt er det gitt en konklusjon av arbeidet og hvordan fremdriften har vært.

Forord.

Denne masteroppgaven er utført ved NTNU, institutt for elektronikk og telekommunikasjon, Trondheim. Oppgaven er avsluttende del av femårig masterstudium og har vart hele våren 2006. Jeg vil benytte anledningen å takke veileder Kjetil Svarstad for å ha vært behjelpelig under hele prosjektet, samt vært motiverende drivkraft for hele AHEAD prosjektet. Jeg vil også takke Xilinx Norge for tilgang på utviklingsverktøyene til bruk i FPGA design.

Levanger, 23.06.2006

Stian Arntsen

Innholdsfortegnelse.

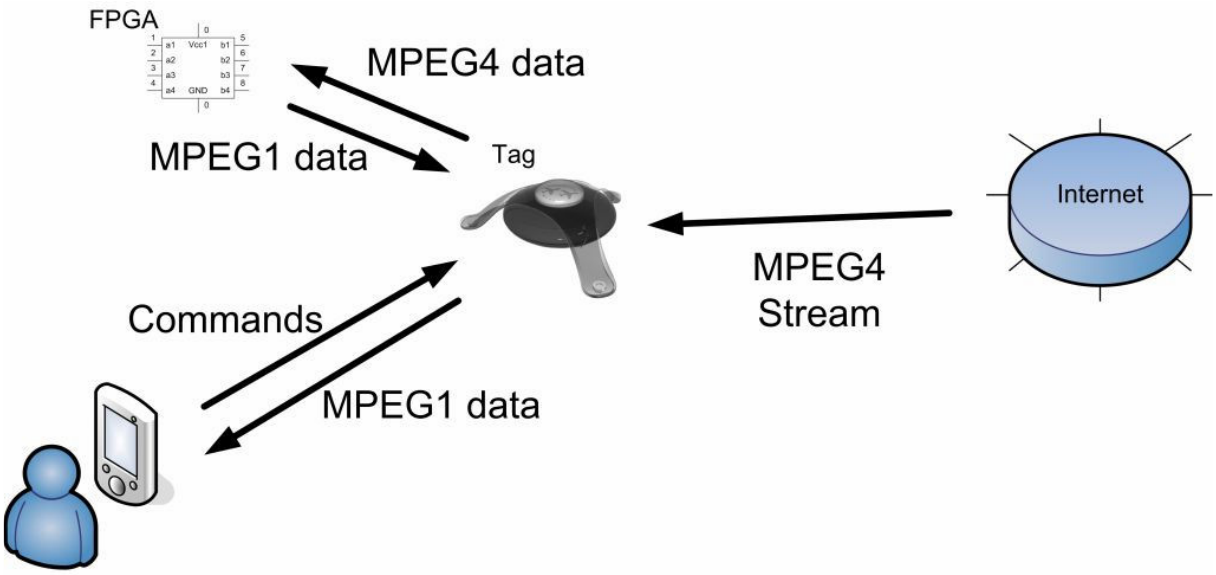
Figurer.

Tabeller.

1 Introduksjon og motivasjon.

AHEAD står for "Ambient Hardware—Embedded Architectures on Demand"[1] og er en aktivitet på IET i samarbeid med SINTEF IKT Kommunikasjonssystemer. Dette er en spesifikk problemstilling innen "Ambient Intelligence"—intelligente omgivelser—og det går i korthet ut på å etablere en plattform som gjør det mulig å "bære med seg" arkitekturbeskrivelser på lette, mobile plattformer som PDA eller Smartphone, og ved behov å få denne arkitektur overført til og instantiert på tager i omgivelsene (lette datamaskiner med FPGA ombord) for så å få eksekvert programmer på denne arkitekturen. Dermed kan man få gjort oppgaver som er altfor tunge eller vanskelige i utgangspunktet for en PDA eller programmerbar tag ved å spesialisere FPGA'en som en co-prosessor inne i tagen.

Et eksempel på bruksområdet for en slik arkitektur kan være media streaming. En bruker vil se en film eller et program på TV. En kan gjøre dette via en Tag der Tagen tar seg av dekoding av media strømmen før den sender den videre til brukeren. Her er MPEG4 en mediastrøm hentet fra Internett med høy oppløsning og høy komprimering, denne dekodes til MPEG1 som har mindre behov for dekoding, samt at tagen tilpasser oppløsningen til den mobile skjermen. Tagen tar seg derfor av all prosessoravhengige oppgaver, samt sørger for at datamengden over bluetooth blir holdt til et minimum. Eksemplet er illustrert i Figur 1.1.

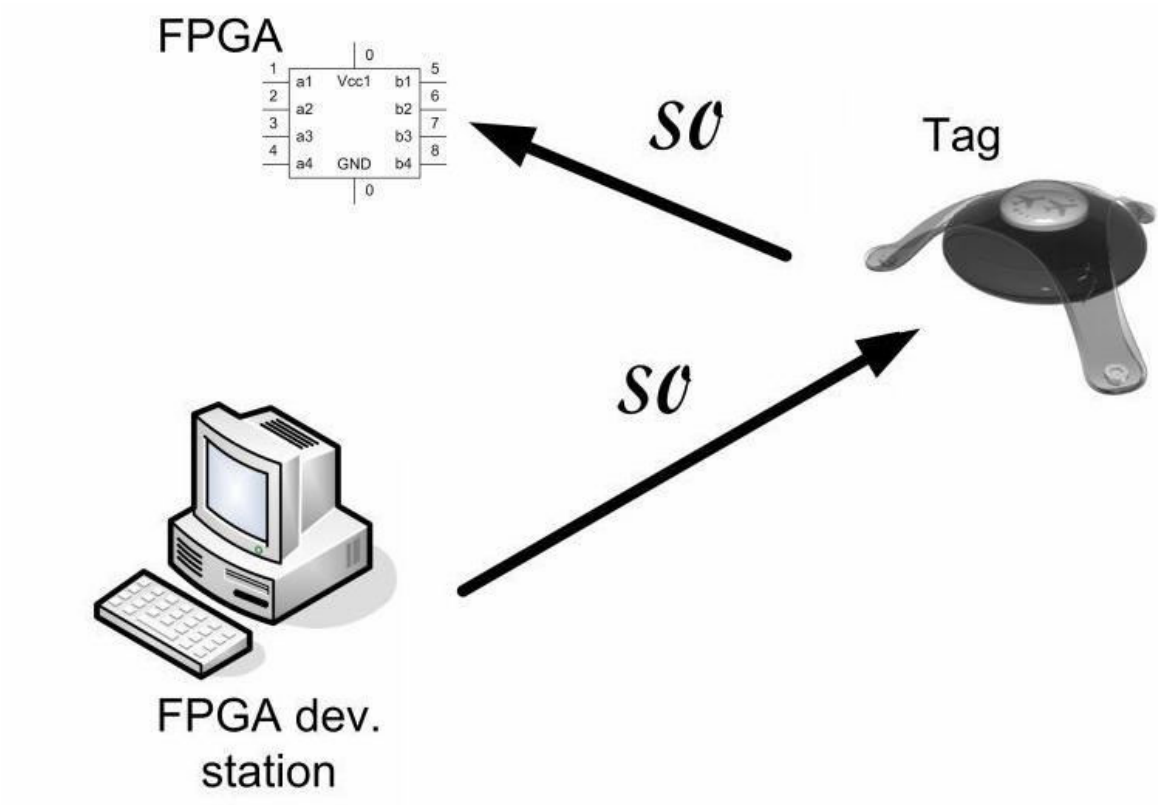


Figur 1.1 Media streaming eksempel.

1.1 Konkret om oppgaven, og hva målet for oppgaven er.

Det er nå gitt en liten innsikt i hvordan AHEAD konseptet er, og litt om hvordan arkitekturen skal oppføre seg i et tenkt eksempel. Oppgaven som denne rapporten er skrevet for, skal hovedsakelig være å utvikle en FPGA-plattform for den tenkte arkitekturen. Kort sagt skal den ferdige FPGA-plattformen kunne ta en gitt konfigureringsfil fra en kjent plassering, og deretter laste inn denne konfigurasjonen automatisk slik at FPGAen forandrer funksjonalitet. I denne første utgaven av FPGA-plattformen ser en for seg at det viktigste er å få FPGA-plattformen til å automatisk skifte funksjonalitet. Dette gjøres ved at en lager en enkel funksjonalitet til å begynne med, deretter skal en kunne enkelt skifte denne funksjonaliteten

ved å forespørre en ny. Når dette hovedmålet er nådd kan en utvikle FPGA-plattformen videre. De neste oppgavene vil være å sette sammen funksjonaliteten til plattformen med de andre oppgavene tilknyttet AHEAD. Det er da tenkt på å koble selve FPGA-plattformen til en ekstern mikroprosessor som tar seg av den videre kommunikasjonen med de andre delene av systemet. Dette er i første omgang trådløs kommunikasjon med PDA. Det er da tenkt at den eksterne mikroprosessoren tar seg av dette, som et mellomledd mellom brukere og FPGA-plattformen. En må også kunne se på mulighetene for at mikroprosessoren om bord på FPGAen kan erstatte denne eksterne mikroprosessoren. Den første utgaven av FPGA-plattformen vil benytte seg mye av en tilkobling til en utviklings PC. I Figur 1.2 er det vist en skisse av hvordan en vil koble sammen systemet til og begynne med. Når systemet fungerer som vist i denne settingen kan en gå videre og introdusere mer funksjonalitet. Dette vil være versjon 1 av FPGA-plattformen.



Figur 1.2 FPGA-plattform og utviklingsstasjon.

2 Tidligere arbeid.

Det er gjort en del arbeid på dette området tidligere ved fakultetet. En sommerstudent jobbet sommeren 2005 med å klargjøre ting rundt partiell dynamisk rekonfigurering av FPGA. Det var også 3 studenter som høsten 2005 jobbet med semesteroppgaver knyttet til AHEAD.

2.1 Hva er gjort?

Sommerstudenten som jobbet sommeren 2005 brukte et utviklingskort som heter ML310 som var bestykket med en kraftig Virtex2Pro FPGA fra Xilinx. Oppgaven bestod i å utforske mulighetene for partiell rekonfigurering og å kompilere et Linux OS som skulle bruke en PowerPC som mikroprosessor. Fra Linux skulle en dynamisk rekonfigurere FPGAen partielt.
Høsten 2005 ble det også jobbet med AHEAD i 3 uavhengige semesteroppgaver. Den ene av disse dreide seg om FPGA-plattformen for AHEAD. Da ble det jobbet med et utviklingskort fra Memec med en Virtex2Pro FPGA. Dette var et utviklingskort som NTNU hadde liggende fra tidligere arbeid med FPGAer. Dey ble også jobbet litt med å finne et nytt utviklingskort som egnet seg for formålet.

2.2 Konklusjon av arbeidet.

Konklusjonen av arbeidet til sommerstudenten var at ML310 utviklingskortet ikke kunne brukes til et slikt system. For å få tilgang til LAN på kortet måtte det implementeres et PCI grensesnitt på FPGAen, dette grensesnittet tok store deler av resursene til FPGAen og dette gjorde at en ikke kunne drive med partiell rekonfigurasjon i tillegg pga plassmangel. Det ble også konkludert med at en partiell rekonfigurasjon fra en kommando i Linux var en avansert oppgave, og at AHEAD var tjent med å legge lista litt lavere til å begynne med. Det ble derfor i starten av semesteroppgaven høsten 2005 tenkt ut et enklere system der en ekstern mikroprosessor skulle brukes for å styre tag arkitekturen, og at FPGAen skulle konfigureres helt for hver gang en ny konfigurasjon skulle bli brukt. Etter å ha brukt et utviklingskortet fra Memec i hele oppgaven, ble det bestemt at en skulle gå til innkjøp av et utviklingskort som var bedre egnet til oppgaven. Dette kortet fantes hos HiTech Global, og heter Suzaku-S. Dette er et kort som har en Spartan3 FPGA. Denne FPGAen har en MicroBlaze soft mikroprosessor om bord, som kjører uCLinux som default konfigurering fra leverandør. Kortet har både LAN og RS232 tilkoblinger. Det som gjorde at en valgte dette kortet var imidlertid at den har en ferdig laget krets som tar seg av konfigureringen til FPGAen under oppstart. Denne kretsen muliggjør derfor dynamisk rekonfigurering på en enkel og ryddig måte.

Dette betyr at før en starter med oppgaven som denne rapporten bygger på, allerede har en del erfaringer, og en del avgjørelser som allerede er tatt. Utviklingskortet Suzaku-S ble bestilt i slutten av semesteroppgaven 2005 og var derfor allerede tilgjengelig da denne oppgaven ble påbegynt. Suzaku-S er vist i Figur 2.1 og en kan se at størrelsen er liten ved å sammenlikne med mynten og med LAN tilkoblingen.

Figur 2.1 Suzaku-S utviklingskort for AHEAD.

Det ble tatt hensyn til størrelsen på kortet da avgjørelsen om Suzaku-S ble tatt. Dette fordi en kan se for seg at dette kortet allerede passer til å implementere i tag arkitekturen. I tillegg har kortet funksjonaliteten som trengs til AHEAD, men ikke noen ekstra og unødvendig.

3 Teori.

Med bakgrunn i manualer og tidligere arbeid på dette området vil dette kapitlet ta for seg teorien for dette systemet, og andre system med lignende funksjonalitet[2][3][4].

3.1 Hva er et Dynamisk Rekonfigurerende System?

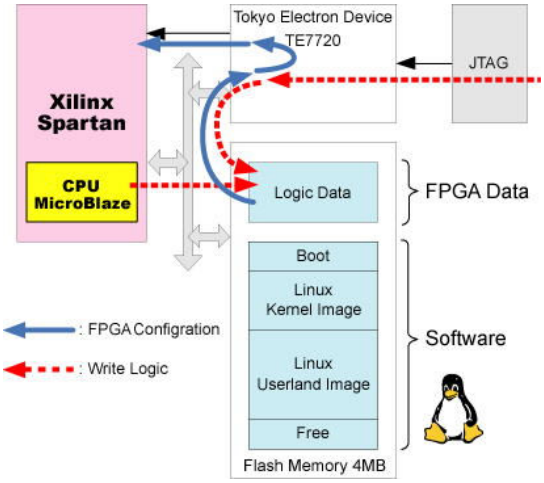
Et dynamisk rekonfigurerende system inneholder som oftest både software og hardware. Hardware er representert med en FPGA, mens software kjører på en mikroprosessor. I disse dager har de fleste FPGAer mulighet for å implementere en soft mikroprosessor. Derfor vil i mange tilfeller et dynamisk rekonfigurerende system være et System on Chip. Det er altså muligheten for dynamisk å skifte funksjonalitet på FPGAen en er ute etter i et slikt system. Hvilken ny funksjon og når den skal tre i kraft på FPGAen er det som regel noen funksjon i software som bestemmer. Det er skrevet mange artikler om hvordan et dynamisk rekonfigurerende system skal virke, og hva slike systemer kan brukes til. Teorier om slike systemer bygger ofte på muligheten enkelte FPGAer har til å rekonfigurerees under kjøring, kalt partiell rekonfigurering. Dette er enda i dag et forholdsvis nytt felt, med liten informasjon om hvordan slike system bygges opp i praksis. De fleste som har skrevet artikler om dette tar for seg teoretiske og mulige fremgangsmåter, men få har virkelig prøvd ut dette noe særlig.

Denne rapporten omhandler et system som skal rekonfigureres dynamisk, men ikke ved hjelp av partiell rekonfigurering. Det vil bli implementert et slikt system både i hardware og software på utviklingskortet Suzaku-S. FPGAen heter Spartan3 og har en mikroprosessor som heter MicroBlaze.

3.1.1 Suzaku-S som Dynamisk Rekonfigurerende system.

Utviklingskortet som brukes kalles Suzaku-S, og har en egen krets som tar seg av konfigureringen av FPGAen. Denne kretsen heter TE7720 og er laget av Tokyo Electronics. TE7720 konfigurerer FPGAen under oppstart\reset. Konfigurasjonen som TE7720 bruker ligger i FLASH minnet som er på Suzaku-S. Det er som vist i Figur 3.1 to muligheter for å legge inn FPGA data inn i FLASH minnet. Den ene er gjennom Jtag koblingen, noe som brukes mest i utviklingsfasen. Den andre muligheten er at MicroBlaze leser inn en konfigurasjon til FLASH minnet, som den mottar for eksempel via LAN. Den blå pilen i figuren representerer dataflyten når hele Suzaku-S starter opp fra en reset eller et power on. Dette betyr at en under kjøring av systemet kan legge inn en ny konfigurasjon i FLASH, og deretter starte systemet på nytt for å endre funksjonalitet. Dette utviklingskortet er derfor godt egnet for utvikling av dynamisk rekonfigurereende systemer.

Utviklingssettet kommer med uClinux som operativsystem på MicroBlaze, dette gjør det enklere å skrive til FLASH fordi det er en egen kommando for dette i denne uClinux. Ulempen for den dynamiske utførelsen er at det tar lenger tid før systemet er oppe etter reset. Dette fordi at uClinux tar en del sekunder å starte.



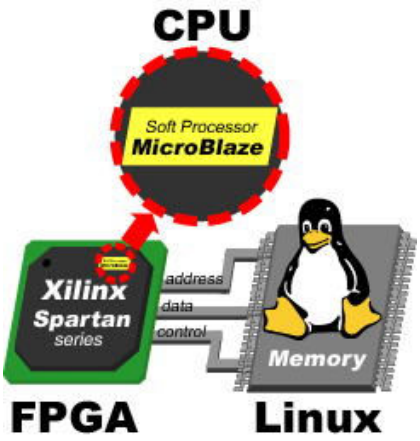
Figur 3.1 Konfigureringsflyt.

3.2 uCLinux i innvevde systemer.

På MicroBlaze skal det altså kjøres et OS som kalles uCLinux. Dette er en avlegger av Linux 2.0 som var ment for mikroprosessorer uten Memory Management Units, MMUs. Datasystemer har som regel RAM tilgjengelig for å kjøre store programmer eller flere programmer samtidig (multi-tasking). Det er dette RAM minnet som i dag er en del av CPU brikken som kalles MMU. MMU implementerer en teknikk kalt virtuelt minne, som gjør at minnet virker større en det egentlig er. Prosessorer med MMU krever mye effekt, og det koster mer med det ekstra minnet, samt større overhead som gjør at slike CPUer er uegnet i dypt innvevde systemer. Heldigvis finnes det MMU frie CPUer som er bedre egnet til innvevde systemer.

Å lage programmer som skal kjøre på MMU-frie prosessorer er en vanskeligere oppgave enn på ordinære CPUer. Designeren må passe på hver oppgaves minneallokering, derfor blir komplekse systemer vanskeligere å lage. Men for enkle applikasjoner, som en gjerne finner på innvevde systemer, er kodeutvikling under uClinux ganske rett frem.

uCLinux er et OS med komplett TCP/IP stakk og er perfekt for innvevde systemer. uClinux-kjernen er mindre enn 512kB, og det er da inkludert flere nettverksprotokoller. Tross uCLinux sin størrelse beholder den Linux sine fordeler som stabilitet, nettverksfunksjonalitet, filsystemstøtte, og APIstøtte. uClinux er derfor skapt for innvevde systemer, og har mange utvidelsesmuligheter slik at alle typer innvevde systemer kan dra nytte av uClinux, både systemer som trenger mye funksjonalitet og systemer som trenger mindre.



Figur 3.2 uClinux og MicroBlaze.

3.2.1 Fordeler og ulemper med å implementere uClinux.

En stor fordel med uClinux er at en enkelt kan lage egne programmer hvis en kan programmeringsspråket C. Muligheten til å lage slike programmer er veldig viktig når en skal drive med systemutvikling, En annen fordel er at uClinux har som sin storebror en åpen kildekode, slik at alle som vil og har kunnskaper om, kan forandre og tilpasse uClinux til sitt eget behov. Akkurat dette gjør uClinux genialt til innvevde systemer, fordi en legger til den funksjonaliteten en vil ha, og tar bort alt som er unødvendig.

En av ulempene med uClinux, er at en ikke bare uten videre kan hente inn alle de funksjoner som finnes i vanlige Linux-distribusjoner. Hvis en har en fin måte å gjøre ting på i en vanlig Linux er det ikke sikkert at en kan gjøre dette i uClinux. Måten en legger til og fjerner funksjoner i uClinux skulle vært bedre, slik at systemet kunne blitt enda mer fleksibelt. En annen ting som er en ulempe er oppstartstiden, men denne kan en vel ikke kommet utenom i et system som kjører et OS. Det er vel også et problem som kun melder seg for systemer som driver med dynamisk rekonfigurering, der hele systemet blir resatt mellom hver konfigurasjonsendring.

3.3 Linux utviklingsmiljø.

En av de største utfordringene med denne oppgaven var å begynne med Linux. Fra tidligere hadde en ingen erfaring med denne typen operativsystem. Det vanskelige med dette var ikke å bruke programmene, men å feilsøke hvis det oppstod problemer. Linux skiller seg for eksempel veldig fra Windows når det gjelder installering av programmer, samt at når en har installert et nytt program er det ikke bestandig like enkelt å finne programmet slik at en kan får kjørt det. I Windows kan en finne igjen installerte programmer på start menyen, men det går ikke på samme måte i Linux sine grafiske grensesnitt. Etter hvert som en lærer seg hvordan ting fungerer og begynner å bli venn med Linux kan en se at systemet ikke er like vanskelig og at det er en grei plattform for bruk til bestemte oppgaver. Når noe av målet er å lage en embedded Linux som uClinux, er det jo en klar fordel å bruke en PC som kjører Linux som OS.

3.4 Ekstern mikroprosessor.

Det ble foreslått tidlig at en skulle bruke en ekstern mikroprosessor som bindeledd til AHEAD systemet. Mikroprosessoren som er tilgjengelig i prosjektet heter CerfCube og er en 32 bits RISC fra IntrinSync. En slik Singel Board Computer (SBC) som CerfCube, skiller seg lite fra andre SBCer. CerfCube har en ferdig bootloader og Linux kjerne som OS. Den har en CF slot hvor det er koblet til utstyr for bluetooth. I tillegg er det Ethernet og RS232 tilkobling, for henholdsvis nettverk og tilkobling for termineringsprogram. Linux kjernen kommuniserer med brukere via RS232, denne kommunikasjonen er toveis og fungerer som et kommandovindu til OS. I den tenkte arkitekturen er det denne mikroprosessoren som fungerer som en slags master. Det er mikroprosessoren som har all kommunikasjon via bluetooth med brukere, og det er den som bestemmer hvilken konfigurasjonsfil som sendes til FPGAen. Det er også denne mikroprosessoren som setter sammen konfigurasjonsfilen ved å bruke det utviklede ADL(Architecture Description Languages) for AHEAD. Når FPGAen er konfigurert, er det tenkt at den skal fungere som en co-prosessor til denne CerfCube. Et annet alternativ er å bruke den innbygde mikroprosessoren som finnes i de aller fleste FPGAer i dag.

3.4.1 Uten ekstern mikroprosessor.

Hvis en benytter seg av den interne mikroprosessoren i FPGAen kan en slippe de ekstra kostnadene og ekstra plass en ekstern mikroprosessor medfører. Det er også andre fordeler, samt noen ulemper.

Ved å benytte Microblaze på FPGAen vil en miste litt prosessorkraft. CerfCube vil typisk ligge på 300-400MHz mens en MicroBlaze på en Spartan3 ligger på 80Mhz. Begge er 32-bits prosessorer, så CerfCube vil være 5 ganger raskere til å behandle data. Hvis data skal overføres fra FPGA-delen av Spartan3 til en mikroprosessor, er det ingen tvil om at dette vil skje raskere hvis denne mikroprosessoren befinner seg om bord på FPGAen. Overføring av data fra FPGAen til en ekstern mikroprosessor vil uansett enklest skje ved å benytte Microblaze, så her har en et stort fortrinn med å benytte mikroprosessoren på FPGAen. En fordel med CerfCube er at den kjører en mer vennlig utgave av Linux enn Microblaze. Det er enklere å legge til programvare og tilleggsfunksjoner. Dette er en stor fordel når en skal utvikle et system. Andre fordeler med CerfCube er ferdig USB kommunikasjon og Ethernet. Dette har også Microblaze, men det er plasskrevende og implementere.

I tillegg til at en totalt har flere fordeler med å bruke en CerfCube som ekstern mikroprosessor må en ta hensyn til at en skal etter hvert begynne å bruke FPGAen til en co-prosessor i er avansert system. Da kan ikke FPGAen være fylt opp med mikroprosessorkjerne, Ethernet og USB kontrollere og annen funksjonalitet som gjør at det ikke er plass til det FPGAen i utgangspunktet var ment til.

Som et forslag til det videre arbeidet vil konklusjonen være at en til å begynne med bruker mikroprosessoren om bord på FPGAen. Og når systemet etter hvert er blitt større og mer avansert må en gå over til en ekstern prosessor som har større kraft. En løpende vurdering vil være det beste for å ikke ta på seg for mye på en gang. En vil derfor ikke innføre denne eksterne prosessoren før en eventuelt begynner å samkjøre de forskjellige aspektene av AHEAD, og kravet til mikroprosessoren øker.

4 Arbeid med FPGA-plattformen.

Dette kapitlet omhandler arbeidet som er gjort med Suzakuen, og avgjørelser som er blitt tatt under veis i arbeidet. Det forutsettes at leseren har litt basiskunnskap om FPGAer og Linux.

4.1 Forberedelser.

For ordens skyld nevnes først et par ting for å unngå forvirringer. Når en i rapporten snakker om webserveren, er dette thttpd serveren i uClinux om bord på Suzakuen. Når en snakker om http-serveren, er det Apache serveren i RedHat Linux.

Før en startet å utforske Suzakuen måtte en ha en utviklings-PC som kjørte Linux som OS. I software manualene[5] til Suzakuen står det anbefalt at en installerte Debian eller RedHat Linux. Det ble først prøvd å installere Debian Linux, og den enkleste måten var å laste ned en nettinstallasjon som installerte de mest nødvendige komponentene, og deretter lastet ned det du ville ha i tillegg. Debian Linux fant aldri drivere til skjermkortet på PC-en, og det grafiske grensesnittet ble derfor aldri bra. Det ble da tatt en avgjørelse på å begynne på nytt, men denne gangen med RedHat9 som OS. Denne versjonen ble lastet ned og installert uten problemer. Merk at for å få et optimalt arbeidsoppsett på Linux PC-en må en under installasjonen av RedHat merke av for at en ønsker å installere kompileringsverktøy og utviklingsverktøy (gcc). Dette for å få med enkelte programmer som trengs for senere å kjøre kommandoer for kryss kompilering av uClinux kjernen. Slik kompilering er avhengige av ncurses-bibliotekene, som ved installering av gcc automatisk blir med. En annen ting som kan være en fordel er å starte Apache http-serveren som er en del av RedHat. Dette gjøres enkelt ved å skrive kommandoen */etc/init.d/httpd start*. Hvis en i tillegg vil starte Apache ved oppstart kan en skrive inn startlinjen for Apache i bunnen av filen */etc/rc.local*. Alle filene som nå ligger under katalogen */var/www/html/* vil nå serves av Apache.

For å kommunisere med Suzakuen måtte en ha en terminal-emulator. Det ble prøvd en del programmer i Windows først. Men både TeraTerm og Windows sitt eget program ble utilstrekkelig, både når det gjelder tyding av spesielle tegn, og bredde på vinduet som tegnene skulle vises i. Det er derfor best å kjøre Minicom som følger med i dokumentasjonen til Suzakuen, og helst kjøre den på en Linux PC. For å kjøre Minicom må en være logget på som root.

For krysskompilering ble også microblaze-elf-tools instalert. Dette er verktøy-kjeden som tar seg av kompileringen av uClinux for vår mikroprosessor, Microblaze. Dette er beskrevet i kapittel 7.1 i software manualen. For å sette omgivelses-variabel og ”enviroment” måtte en skrive:

export PATH=$PATH:/usr/local/microblaze-elf-tools/bin

Dette fordi at når en skal kompilere må en bruke noen biblioteker slik at uClinux blir spesialisert for den mikroprosessoren som er på FPGAen, nemlig microblaze.

Gjennom hele denne oppgaven er det brukt 2 PC-er samt Suzakuen. Den ene PC-en (PC1) kjører Linux og brukes til kompilering av uClinux, og å legge filer som Suzakuen benytter seg

av ut på http-server. Den andre (PC2) kjører Windows og brukes i hovedsak til utviklingsverktøyene fra Xilinx, samt rapportskriving og andre praktiske oppgaver. Alle disse 3 er igjen koblet til samme lokale nettverk via en ruter som igjen er koblet til internett.

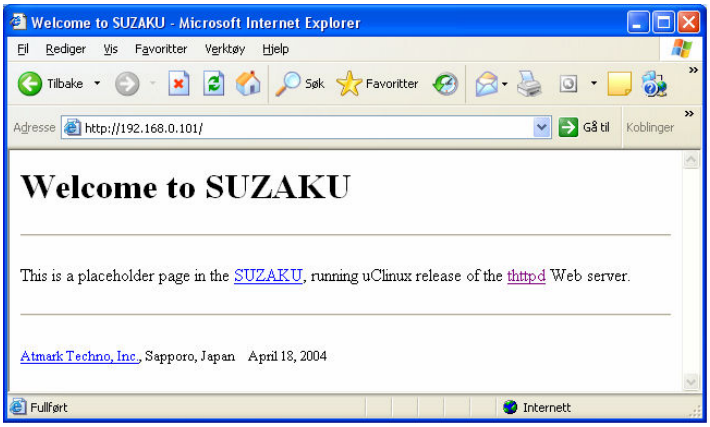
4.2 Eksempler og "steg-for-steg"-manualer.

For å komme i gang med arbeidet ble software manualen til Suzakuen gått nøye gjennom. I denne manualen er det en del eksempler som ble gjennomført.

Det første som ble gjort var at en koblet opp Suzakuen som vist i software manualen. Med strøm, LAN og seriekabler. Når minicom var startet ble strømmen slått på, og en kunne se at uClinux ble startet opp på Suzakuen.

4.2.1 Webserver.

Det første eksemplet fra manualen var websiden i kapittel 4.6. Da en startet Suzakuen skulle en default webside vises fra webserveren i uClinux. Suzakuen ble koblet til en ruter, og fikk en IP adresse via DHCP. Kommandoen *ifconfig* viser den tildelte adressen som, og ved å taste denne adressen direkte inn i Internett Explorer på PC2 på samme nettverket kunne en se index.html som lå på Suzakuen, se Figur 4.1.



Figur 4.1 Default i webserver i Suzaku.

4.2.2 Skrive til FLASH.

Neste eksempel fra software manualen som ble prøvd var å skrive over flash minnet, kapittel 12. For å få til dette ble det lokalt satt opp en ftp-server som Suzakuen skulle hente et image fra. Med kommandoen *netflash –f ftp://192.168.xxx.xxx/image.bin* skulle en skrive over den gamle image.bin filen med en ny. Men Suzakuen fant ikke ftp-serveren. Ftp-serveren er testet mot andre maskiner, og den fungerer fint for maskiner både lokalt og ekstern. Et annet alternativ er å bruke en http-server. I starten ble studentområdet på *folk.ntnu.no* brukt, men dette ble bare upraktisk, pga at filene måtte lastes opp og så lastes ned, noe som ble veldig tungvint. Etter litt utforsking ble det satt opp en lokal http-server på hjemmenettverket, hvordan det ble gjort er forklart i et foregående kapittel. Det ble opprettet en egen mappe i hjemmemappen til http-serveren som het hw. Ved å legge den aktuelle image.bin filen i

mappen hw, kunne en i uClinux skrive netflash-kommandoen slik for å skrive over flash-minnet:

netflash http://192.168.xxx.xxx/hw/image.bin

De filene som legges i hw må være tilgjengelig for alle for at *netflash* skal kunne hente filen, dette gjøres enkelt ved å bruke kommandoen *chmod 777 image.bin*.

4.2.3 Egen uClinux kjerne.

Neste eksempel er å lage en egen uClinux image, kapittel 8. Første gang en prøvde å gjøre dette eksemplet ble det produser mange feilmeldinger når en prøvde å skrive *make menuconfig*. Problemet var at en måtte installere ncurses for å bruke kommandoen *menuconfig*. Plagdes lenge med dette problemet (flere dager) uten å finne svar, tok til slutt og installerte SUSE linux, men samme problemet der. Gikk etter hvert tilbake til RedHat9, men under installasjonen ble det valgt å legge til pakker som ikke var i standard oppsettet. Der var et alternativ for å legge til Development Tools, som for eksempel gcc. Nå kan en bruke menuconfig for å modifisere uClinux. Gikk gjennom *menuconfig* med bare default innstillinger, og en ny image.bin ble laget etter at *make dep; make* ble kjørt. En hadde nå laget en egen uClinux for å laste inn til Suzakuen. For at denne skulle bli gjeldene brukte en netflash kommandoen som tidligere forklart.

4.2.4 Forandre default webside.

For å utforske litt mer, ble det nå prøvd ut om en kunne forandre på filene som webserveren i uClinux kjørte. Det som ble gjort var å gå inn i mappen uClinux-dist-20040408-suzaku6\vendors\AtmarkTechno\SUZAKU\thttpd\ i RedHat og forandret index.html. I denne mappen ligger alle filene som serves av webserveren i uClinux når den kjører på Suzakuen. Deretter ble det laget et nytt image.bin, og så ble denne lastet inn i Suzakuen. Webserveren ville da starte med en modifisert index.html hvis alt ble som antatt. Men det skjedde ikke til å begynne med, fordi tilgangen på index.html var 777 etter redigeringen. For at webserveren i uClinux skal serve en side må tilgangen være 644. Etter at dette var fikset fungerte webserveren som den skulle med de nye endringene. En kan bruke dette grensesnittet til å drive med fjernstyring av Suzakuen, men bare hvis en lager script som brukte uClinux sitt skall. Dette er forklart i et senere kapittel.

4.2.5 Legge til program i uClinux.

I software manualen kan en ved å se på eksemplet i kapittel 10 legge til sine egne kommandoer i uClinux. Eksemplet i manualen legger til en kommando som heter hello til uClinux sine vanlige kommandoer. Når en i kommandolinja skriver hello skriver programmet ut ”Hello SUZAKU World”. Dette er et klassisk eksempel, og ved å bruke Makefile for dette eksemplet kan en lage sine egne programmer å legge til uClinux. Eksemplet med hello ble utprøvd, og det fungerte veldig greit.

Det eneste problemet var at når en skulle kompilere Linux kjernen på nytt fikk en beskjed om *mb-gcc command not found* etter en restart av PC1. Etter litt feilsøking løste dette seg underlig nokk med å eksportere PATH variabelen for tool-chain en gang til, altså skrive:

export PATH=$PATH:/usr/local/microblaze-elf-tools/bin/. Virker som dette må gjøres hver gang etter en restart.

4.3 Rekonfigurering.

Systemet rekonfigurerees ved bruk av uClinux kommandoen *netflash*. For at systemet skal kunne fjernstyres er det laget et cgi script for å muliggjøre dette via webserveren. Det eneste som trengs i cgi scriptet er første linje som inneholder skallet som kommandoene kjøres fra og andre linje er kommandoen som skal rekonfigurere systemet. Et enkelt eksempel er vist under:

#!/bin/sh
netflash http://192.168.xxx.xxx/image.bin

Når en i et html dokument linker til et slikt script vil Suzaku automatisk laste inn konfigurasjonen image.bin og legge den på riktig plass i flash minnet når linken til filen blir fulgt i en nettleser. Scriptfilen må lagres med *.cgi som filnavn og filen må kunne eksekveres av alle brukere(chmod 777). En kan også sikre seg at webserveren eksekverer cgi-script ved å skrive *#httpd –c *.cgi*.

For å gjøre tilsvarende med konfigurering av hardware på Suzaku må en gjennom flere steg. Det første er å lage hardware-endringer til det prosjektet som finnes på SuzakuCDen i mappen fpga_proj. Dette gjøres ved hjelp av utviklingsverktøyene ISE og XPS fra Xilinx. Når endringene er utført og en kompilerer kodene på nytt genereres det en ny bit-fil. Denne bit-filen må igjen editeres for å kvitte seg med informasjons headeren og den må i tillegg paddes slik at den er riktig størrelse for flash minnet. Når dette er gjort kan denne nye bit-filen legges ut på http-serveren og Suzakuen kan benytte følgende kommando for å skrive den nye hardware konfigurasjonen inn i flash minnet:

netflash –n –r /dev/flash/fpga http://192.168.xxx.xxx/hw/ny.bit

4.3.1 Bit-fil tilpasset flash minnet på Suzaku.

For å fikse de genererte bit-filene slik at de passer til flash minnet, er det laget et eget program. Programmet fjerner headeren og fyller på filen med 0xFF slik at størrelsen blir korrekt i forhold til avsatt plass i flash minnet. I vedlegg 1 er C-koden til dette programmet. Alt en trenger å gjøre er å legge denne C-filen i en egen mappe på en Linux PC. Deretter kjører man kommandoen:

gcc –Wall –o bit2flash bit2flash.c

Hvis en nå har generert en bit-fil fra utviklingsverktøyene som en vil legge til i flash minnet via kommandoen netflash, må en benytte seg av dette programmet på følgende måte:

/home/username/egenmappe/bit2flash fil.bit ny.bit

Nå kan en legge filen ny.bit I flash minnet med netflash kommandoen som forklart i det foregående kapitlet.

4.4 Xilinx utviklingsverktøy.

For å lage nye funksjoner i hardware til Suzakuen må en benytte seg av utviklingsverktøyene fra Xilinx. Det er en fordel om en kan bruke disse når en starter på en slik oppgave. Det finnes mange fine dokumenter for at en skal komme fort inn i bruken av verktøyene, så det er ikke nødvendig å ha kunnskap om de fra før.

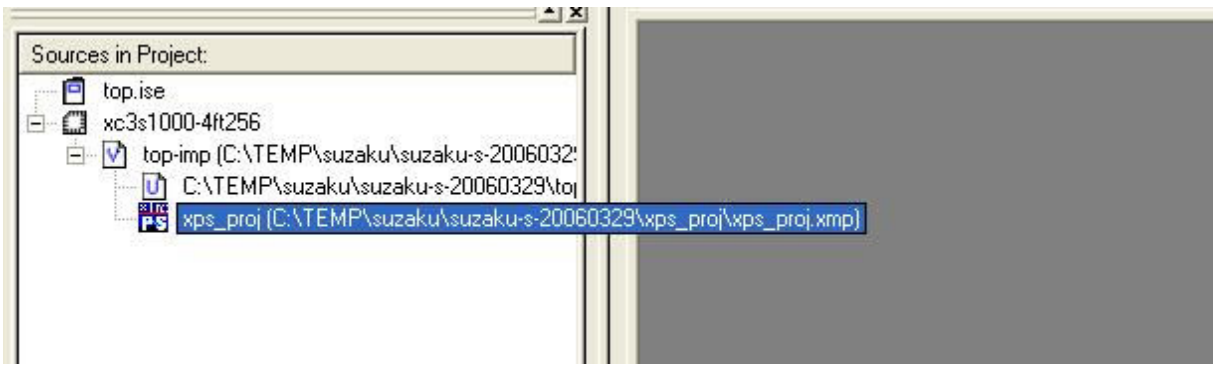
I dette kapitlet skal en nå gå gjennom hva en gjorde for å legge til ny hardware i tillegg til eksiterende på Suzakuens FPGA. Det som er verdt å vite før en begynner er at Xilinx har to forskjellige utviklingsverktøyer. Det ene er ISE som inneholder alle de verktøy som brukes for å syntetisere og kompilere et hardware prosjekt til en FPGA. Det andre er XPS som er det verktøyet som tar seg av software og hardware kommunikasjonen på FPGAen. XPS er egentlig ikke så ulik ISE fordi de bruker mange av de samme underprogrammene for å syntetisere og kompilere.

4.4.1 Editere FPGAens hardware.

Hardware manualens[6] kapittel 9 handler om å editere det allerede eksisterende FPGA prosjektet. Dette prosjektet finnes på Suzaku dokumentasjons CD-en[9] som var med som dokumentasjon når Suzakuen ble kjøpt inn. Hvis en bruker dette eksisterende prosjektet som det er, vil en få akkurat den samme hardwaren slik systemet er når det kommer fra leverandør. I eksemplet tar de for seg hvordan en legger til en UART kobling til det eksisterende prosjektet. I denne oppgaven er det ingen hensikt med en ekstra UART, derfor lages det heller en annen funksjon. For dette prosjektet kan en lage en adder funksjon for demonstrasjonsøyemed.

De utviklingsverktøyene som ble brukt i denne oppgaven var versjon 7.1. Det betyr dermed at noe av det som er forklart i manualen ikke vil stemme overens med seneste versjoner av utviklingsverktøyene. Forskjellene er ikke så veldig store, men det er kommet en ny versjon 8.1, og bruker en denne versjonen vil ting se ganske mye annerledes ut.

En starter altså med å lage seg et nytt prosjekt i ISE, og kaller det for eksempel for *top*. Når en nå har et tomt prosjekt kan en hente inn de source filene (top.ucf og top.vhd) som allerede eksisterer for Suzakuens FPGA. Når dette er gjort må en importere sourse filen for XPS som tilhører dette prosjektet. Når en nå har gjort dette skal skjermbildet være som Figur 4.2 i .

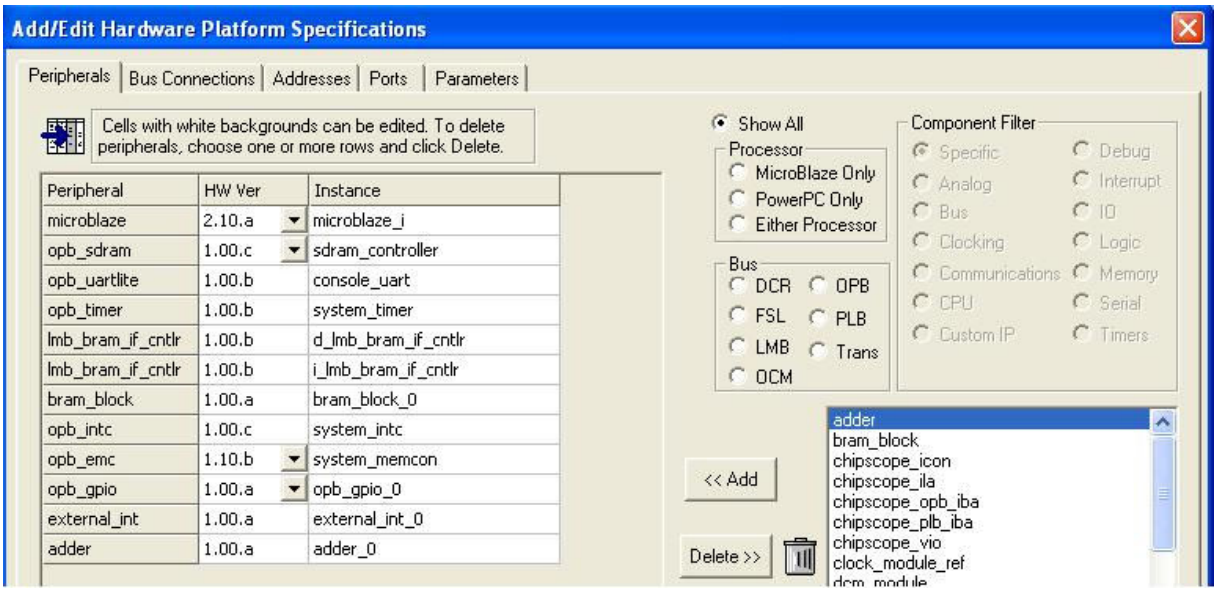


Figur 4.2 ISE prosjekt med xps_proj inkludert.

En må nå dobbelklikke på xps_pro slik at en får frem XPS. Det er i dette programmet en kan editere og legge til hardware. En vil i dette prosjektet ikke legge til egne software koder fordi dette er nødt til å gjøres i uClinux.

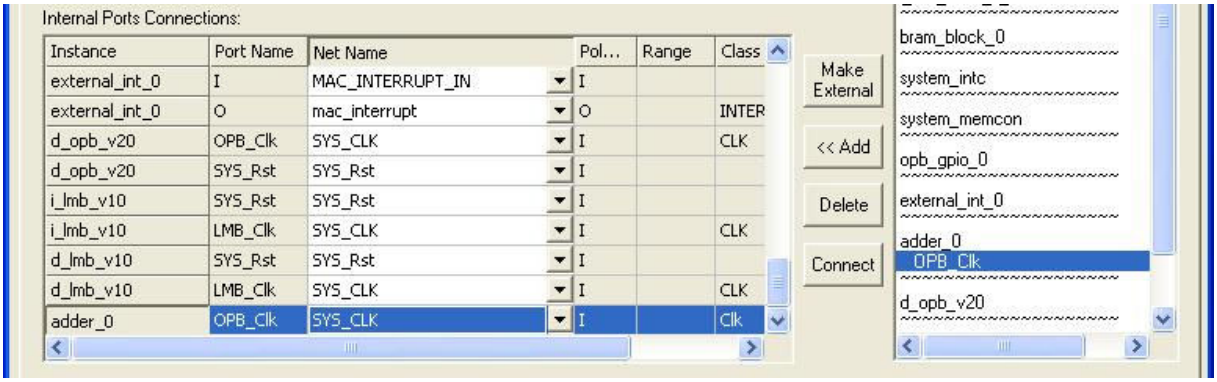
For å nå legge til ønsket funksjonalitet må en gjennom flere steg for å få denne nye funksjonaliteten til å virke sammen med resten av systemet. Først må en lage en ny IP(Intelectual Property), da klikker man på *Project->Create/Import Peripheral*. Det er denne IP-en som skal inneholde den nye funksjonaliteten til systemet. En følger så stegene fram til en ny perifer enhet. Det finnes egne dokumenter som forklarer disse stegene, og hvordan en lager og legger til slike perifere enheter til et system. En vil ikke i denne rapporten gå helt i detalj på akkurat dette, men heller skrive grovt om hvordan en gjør det og vise noen skjermbilder for at en kan se litt hvordan det er gjort.

For å koble den nye IP-en til systemet bruker en *Add/Edit Hardware Platform Specifications*. En velger så adder i ruten til høyre i Figur 4.3 og velger Add slik at adder kommer med i listen til venstre sammen med de andre enhetene som er med i systemet. En kan her kjenne igjen enheter som microblaze_i, sdram_controller, system_timer, osv.



Figur 4.3 Legger til enheten adder til systemet.

I vinduet i Figur 4.3 må en også velge Bus Connections for å koble adder til OPB bussen, samt velge Addresses og gi adder adressen fra 0x81000000 med en bredde på 256. Når dette er gjort må en koble systemets klokke til adder sin klokkeinngang. Velger da adder sin OPB_Clk i ruten til høyre i Figur 4.4 og velger Add. Deretter velges Net Name i ruten til venstre slik at det blir SYS_CLK.



Figur 4.4 Koble sammen klokkene.

Når dette er gjort kan en se at enheten som nå er laget og koblet opp i systemet vises sammen med de andre enhetene for systemet i XPS.

Det som nå gjenstår er å legge til den funksjonaliteten en ønsker i vhdl koden til den nye enheten. I denne oppgaven er det adder_0 sin user_logic.vhd som skal editeres. I denne filen er det allerede lagt inn kommunikasjon med OPB bussen. Det en nå kan gjøre er å lage en adder funksjon som vist under:

```
ADDER_CIRCUIT: process ( Bus2IP_Clk ) is
begin
        if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
                if Bus2IP_Reset = '1' then
                        slv_reg2 <= (others => '0');
                else
                        slv_reg2 <= slv_reg0 + slv_reg1;
                end if;
        end if;
end process ADDER_CIRCUIT;
```

Denne adder funksjonen legges til i filen der det er merket av plass for egen logikk. I tillegg er det verdt å merke seg at en her i denne programsnutten tilordner slv_reg2 en verdi. Det betyr at en i entiteten SLAVE_REG_WRITE_PROC i den samme filen er nødt til å kommentere ut de setningene som har med slv_reg2 å gjøre. Det vil si hele delen av *case* setningen som har med *when "001"* => å gjøre. Når dette er gjort kan en syntetisere og kompilere systemet. Da velger en *Tools->Generate Libraries and BSPs*, deretter *Tools->Build All User Applications*, og til slutt *Tools->Generate Netlist*. Nå er en ferdig med å bruke XPS og kan stenge dette programmet.

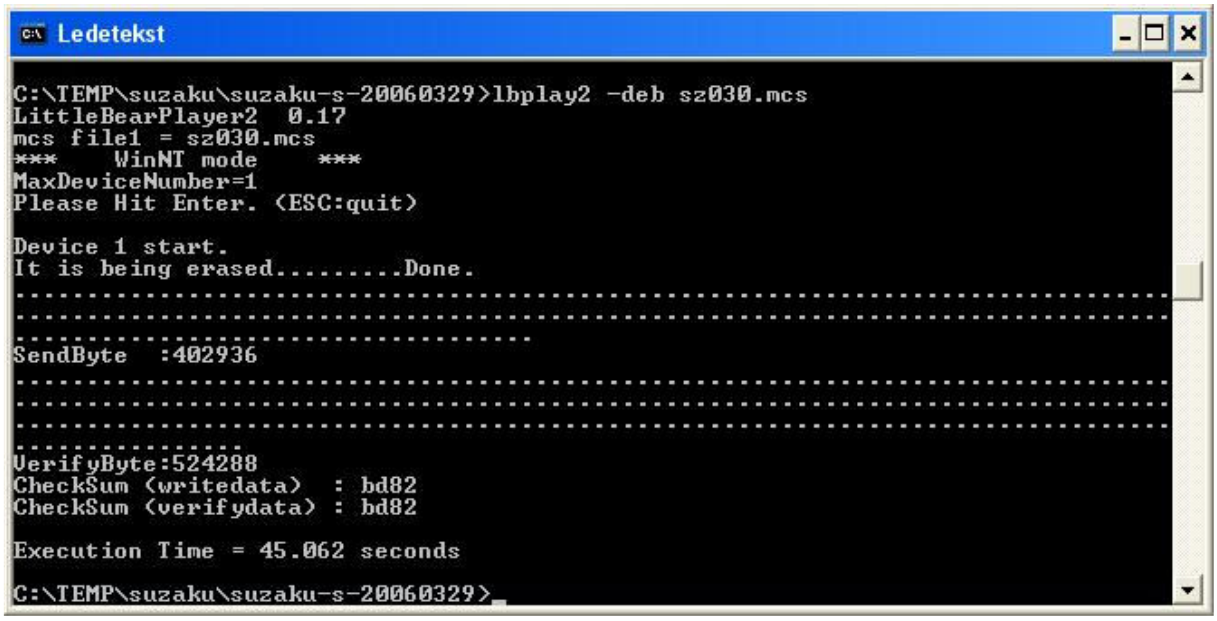
Når en nå går tilbake til ISE vil en kunne generere de nødvendige filene for å senere lage bit-fil av hele det nye systemet. Alt en trenger å gjøre er å dobbelklikke på *Generate Programming File* i Processes for Source vinduet. Når de prosessene som nå er satt i gang er ferdige er systemet ferdig kompilert. Det som gjenstår da er å gjøre om på den nye bit-filen slik at den kan overføres til Suzaku via LAN.

4.4.2 Lbplay2 og mcf-fil generering.

Fra tid til annen kan det hende at en overføring av hardware beskrivelse til FPGAen via LAN vil mislykkes. Det er da en mulighet å lage en mcf-fil som en kan programmere FPGAen med gjennom JTAG kabel. Programmet som gjør dette heter lbplay2. I dette kapitlet skal en ta for seg hvordan en lager en slik mcf-fil og hvordan en gjenoppretter en dårlig utført konfigurasjon.

Det første en må gjøre er å lage en mcf-fil. Alternativt kan en benytte seg av de mcf-filene som ligger på dokumentasjons CD-en. Uansett må en ha en slik fil tilgjengelig. Når en skal lage en slik fil benytter man seg av *Generate PROM, ACE or JTAG File* som er et alternativ i *Processes for Source* vinduet i ISE. Denne mcf-filen som en skal lage er en PROM fil som skal passe til en Xilinx seriell PROM. Den kretsen i systemet som programmerer FPGAen er kompatibel med xc18v04 type PROM. Med de opplysningene som nå er gitt skal det være mulig å lage en mcf-fil, filen det skal lages mcf-fil fra er den bit-filen som er laget for det spesifikke prosjektet.

Når filen er laget må akkurat den filen som skal overføres ligge i samme mappe som programmet lbplay2.exe. For å bruke dette programmet må en ha en Parallell Cabel 4 som JTAG kabel. Denne må kobles til Suzaku og den korrekte jumperen må settes for at en kan programmere FPGAen. Når alt er tilrettelagt kan en starte kommandovinduet i Windows. Kommandoen: *lbplay2 –deb fil.mcf* kjøres fra kommandolinja, og hvis alt går etter planen skal en få et vindu som ligner på det i Figur 4.5.



Figur 4.5 lbplay2 JTAG konfigurering av FPGA.

Merk at det er veldig viktig at de to CheckSum er like, hvis ikke er overføringen mislykket. Da må en gjenta denne prosessen en gang til.

4.5 HW-SW-codesign implementert på Suzaku.

Etter å ha laget hardware endringer i et tidligere kapittel kan en også lage et program som kjører på uClinux som bruker denne nye funksjonen som er lagt til. Adressen på OPB bussen

til det første registret i den IPen som ble laget er 0x81000000, og de påfølgende registrene har adresse 0x81000004 og 0x81000008 pga at et register er 32bit og en adresseplass tar 8 bit. Hvis en nå lager en software kode som sender to tall til maskinvare, ett i slv_reg0 som er den første adressen til IPen, og ett til slv_reg1 som er den neste adressen til IPen. Svaret vil da bli lagt ut i slv_reg2 som har den tredje adressen, altså 0x81000008. Koden for dette programmet er vist her:

```
/* adder.c */

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>

int main(int argc, char * argv[])
{
        int tall;
        printf("ADDERER TO TALL.\n");
        printf("Skriv inn tall A: ");
        scanf("%x", &tall);
        *(volatile unsigned int *)(0x81000000+0) = tall;

        printf("Sriv inn tall B: ");
        scanf("%x", &tall);
        *(volatile unsigned int *)(0x81000000+4) = tall;

        printf("A + B = %x \n", *(volatile unsigned int *)(0x81000000+8));
        return 0;
}
```

Denne C-koden kan kompileres og implementeres i uClinux akkurat slik som ”hello” eksemplet fra software manualen, bortsett fra at Makefile må editeres slik at alle instanser av hello erstattes av adder. Hvis en nå står i mappen som adder.c og Makefile er lagret kan en bare skrive *make*. Deretter *make romfs* for å legge det ferdige programmet i den rette mappen i uClinux(/bin). Det er den samme Makefile som brukes for alle programmene som lages i denne oppgaven, og er vist i vedlegg 8.

Når dette er gjort kan en nå lage et nytt image.bin å lese inn i flash-minnet. En vil da ha en ny kommando som kalles adder, og denne vil ta imot to 32-bits tall, sende tallene til hardware på FPGAen, addere dem og sende svaret tilbake til uClinux.

```
# adder                 # adder                 # adder
ADDERER TO TALL.        ADDERER TO TALL.        ADDERER TO TALL.
Skriv inn tall A:       Skriv inn tall A: 8     Skriv inn tall A: 8
                        Sriv inn tall B:        Sriv inn tall B: 4
                                                A + B = c
                                                #
```

Figur 4.6 adder kommandoen i uClinux.

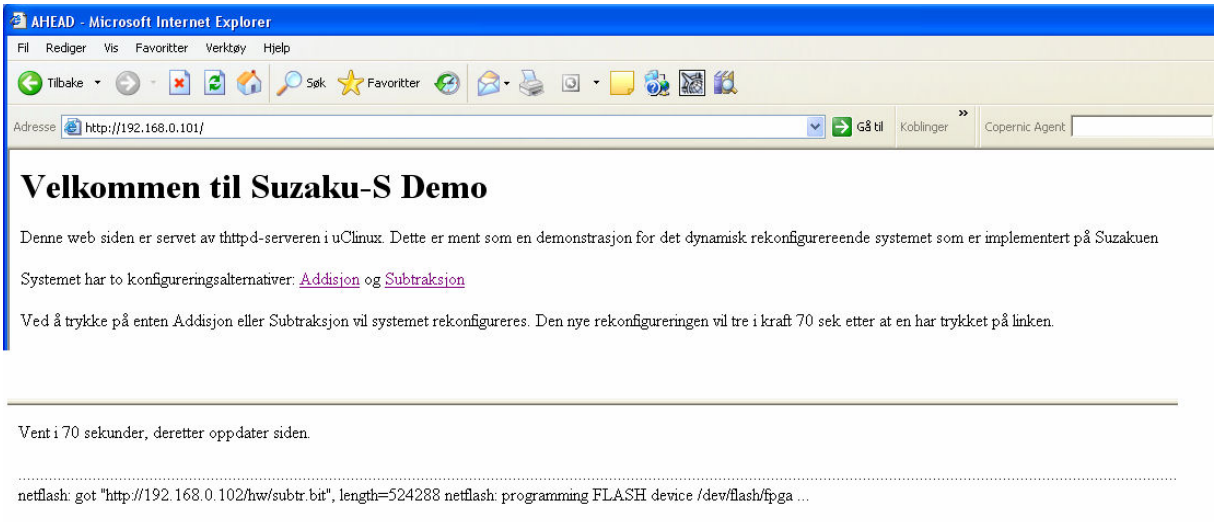
Det er nå laget et program i uClinux som gir informasjon til hardwaren og får hardware akselerasjon på den ”tunge” oppgaven før resultatet sendes tilbake til uClinux. En kan tenke

seg at dette kan brukes til større og tyngre oppgaver som passer bedre til AHEAD konseptet, som for eksempel å bruke en del av hardware som en spesialisert co-prosessor.

4.6 Ekstern rekonfigurering via webserveren.

Hvis en nå ser tilbake på de to forrige kapitlene kan en tenke seg at en kan lage to forskjellige hardware konfigurasjoner til Suzakuen, og deretter benytte html og cgi til å bytte mellom disse konfigurasjonene via en ekstern nettleser.

Det ble da tatt utgangspunkt i adderen i det forrige kapitlet. Det ble så laget en tilsvarende hardware som erstattet addisjonen med en subtraksjon. En hadde nå to forskjellige hardware konfigurasjoner som en skulle bytte på gjennom et eget grensesnitt via webserveren. Disse to forskjellige konfigurasjonen ble lagt ut på http-serveren som henholdsvis add.bit og subtr.bit. Deretter ble det laget et enkelt html grensesnitt der en brukte add.cgi og sub.cgi som scriptene som utførte konfigurasjonen. Disse scriptene er vist i vedlegg 2. Et utsnitt av html-siden som tok seg av denne funksjonaliteten er vist i Figur 4.7.



Figur 4.7 Konfigurering av Suzaku hardware.

Hvis en nå brukte kommandoen adder i uClinux etter at en trykte på Subtraksjon linken i webserveren fikk en ut en subtraksjon som vist i Figur 4.8.

```
# adder
ADDERER TO TALL.
Skriv inn tall A: 8
Sriv inn tall B: 4
A + B = 4
#
```

Figur 4.8 adder kommandoen i uClinux med subtraksjon i hardware.

En har altså nå fjernkonfigurering av Suzakuen med gitte konfigureringer som ligger på en egen http-server. En kan så å si ha uendelig mange forskjellige ferdige konfigurasjonsmuligheter liggende klar til bruk for et hvilket som helst bruk, og disse kan tre i kraft hvis brukeren trykker på den konfigurasjonen man ønsker seg. Ulempen her er at en

enda ikke kan se resultatet av konfigureringen eksternt, men at en må ha Suzakuen tilgjengelig for å se resultatene.

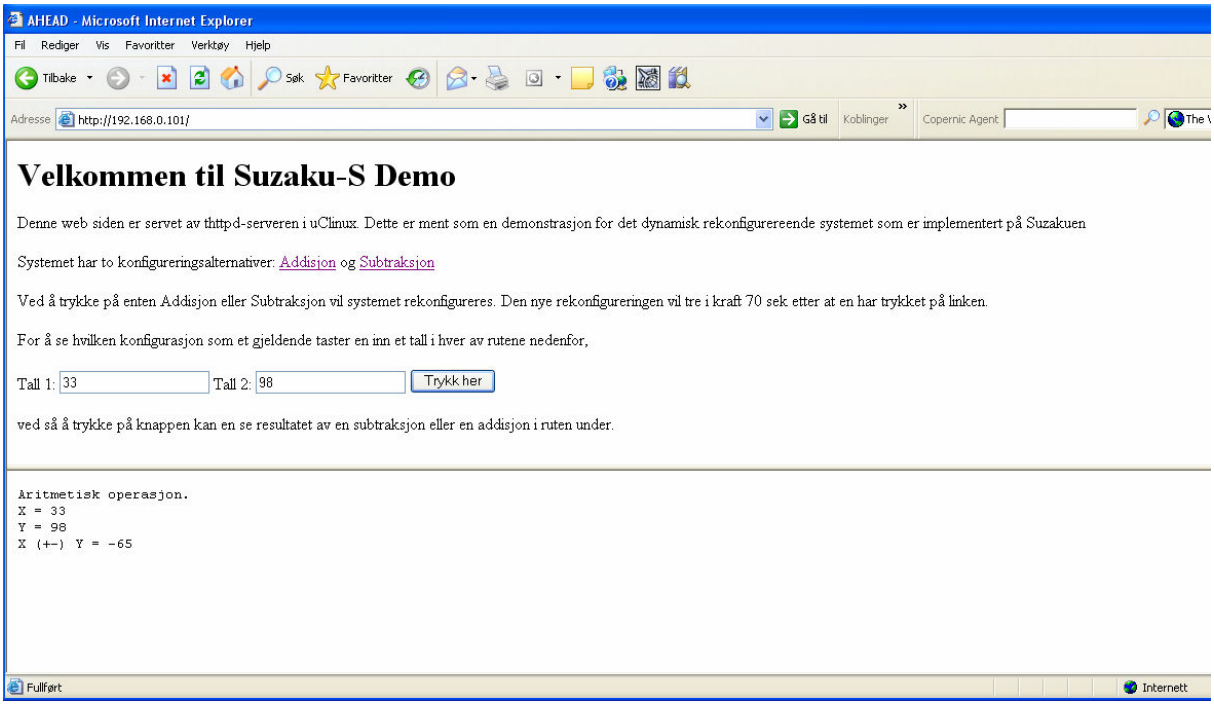
4.7 Ekstern input til interne programmer.

For å forsøke å styre programmet adder i uClinux fra et html grensesnitt ble det laget et alternativt program som er kalt ar_arg (aritmetikk med argumenter). Dette programmet skulle ta to variable som argument. C-koden for dette programmet er vist i vedlegg 3. For at et html dokument skal kunne sende informasjon til et cgi script brukes <form> i html koden. Ved å benytte seg av informasjonen i QUERY_STRING, som er informasjonene som sendes fra <form> sitt <input> attributt, kan en gi ar programmet variable argumenter fra en ekstern nettleser. Det ble så laget et eget cgi script som behandlet QUERY_STRING og skrev verdiene i de forskjellige variabelnavnene. Html koden nedenfor viser navnet på variablene (t1 og t2), og QUERY_STRING for akkurat denne <form> vil typisk se slik ut: t1=33&t2=98

```
<form action="action.cgi">
Tall 1: <input name="t1"> Tall 2: <input name="t2">
<input type=submit value="Trykk her"></form>
```

Når en nå trykker på "Trykk her" knappen linkes det til cgi scriptet action.cgi som er vist i vedlegg 3. Dette cgi scriptet virket i http-serveren (Apache) på utviklings PCen, men kommandoen sed var ikke tilgjengelig i uClinux slik som det var implementert på Suzakuen, derfor kunne ikke scriptet kjøre på Suzakuen. En prøvde en stund å finne denne kommandoen i menuconfig alternativet for uClinux, men klarte altså ikke å implementere denne funksjonaliteten. Så endringen for å gjennomføre dette måtte bli at en i stedet for å ta to variable som argumenter til ar_arg (*ar_arg $t1 $t2*) måtte ta hele QUERY_STRING som argument til et tilsvarende program. Dette tilsvarende programmet kalles bare ar (aritmetikk) og C-koden er vist i vedlegg 4. Det tilhørende cgi scriptet er nå forenklet betraktelig og er vist i vedlegg 5 der en nå linker til programmet slik: *ar $QUERY_STRING*.

QUERY_STRING måtte derfor behandles internt i programmet, men funksjonaliteten vil bli den samme. Bildet for akkurat denne funksjonaliteten er vist i Figur 4.9. Merk at en i dette programmet bruker funksjonen atoi() i C-koden for ar. Dette gjør at en får addisjon og subtraksjon av integer, noe som fører til desimale verdier.



Figur 4.9 Ekstern input til internt program.

En har altså så langt i arbeidet fått et system som kan styres eksternt fra en hvilken som helst PDA eller Smartphone som har internett tilgang. Systemet kan rekonfigurerees med gitte funksjoner, og disse funksjonene kan igjen gis forskjellige input via nettleseren. En har nå oppnådd en slags toveis kommunikasjon mellom for eksempel en PDA og Suzaku. PDA ønsker en type funksjonalitet, Suzaku programmerer seg etter dette ønsket, og deretter jobber Suzaku med de input som PDA gir, og gir ut resultatet.

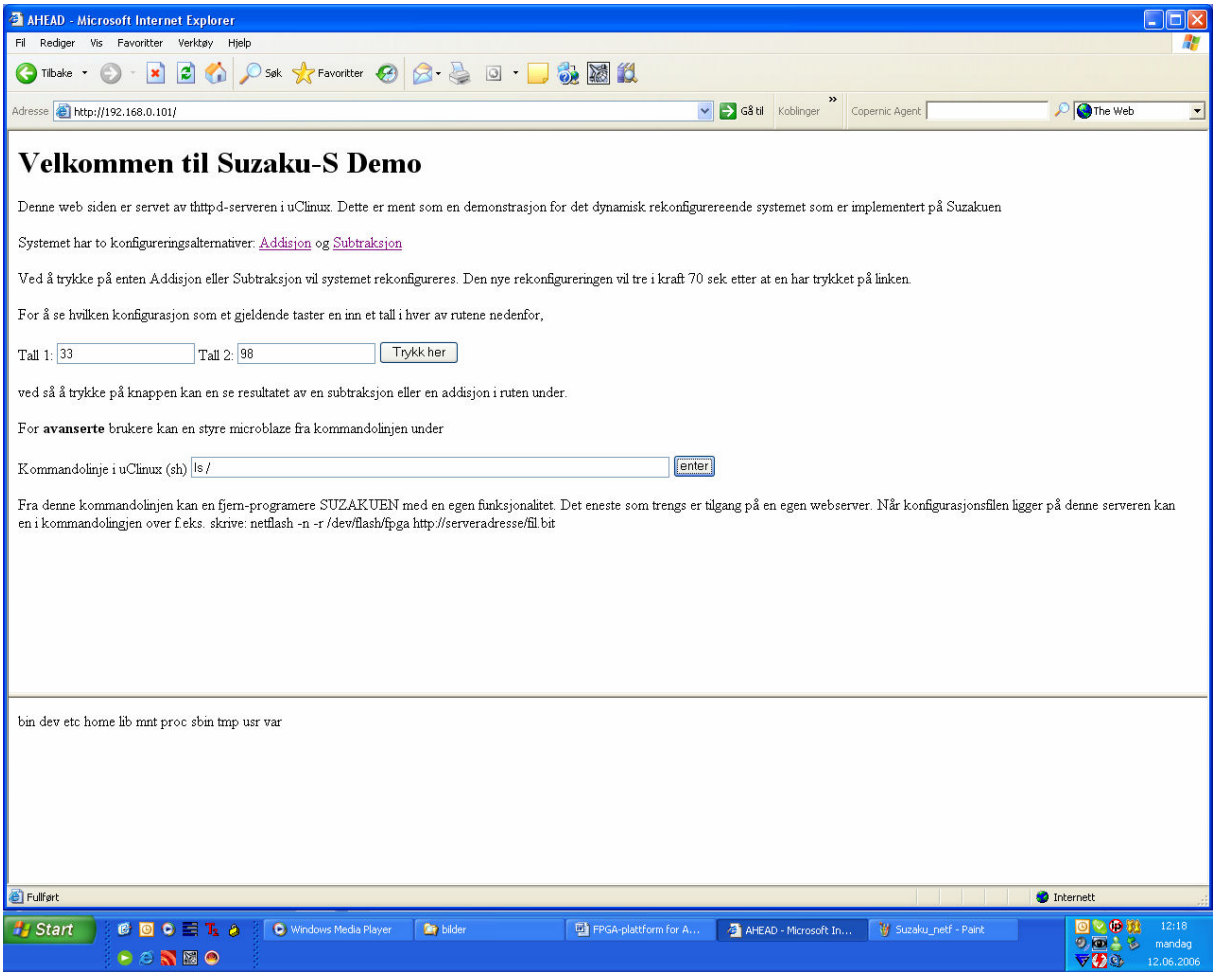
4.8 Ekstern kommandolinje og ekstern rekonfigurering.

For at systemet skulle passe enda bedre til AHEAD spesifikasjonen ville en forsøke å laste inn konfigurasjons filene fra den enkelte PDA eller Smartphone. I den enkle utgaven som allerede er implementert er disse konfigurasjonsfilene representert med bit-filer som er generert av utviklingsverktøy og deretter tilpasset flash minnet. Det betyr at en må lage en upload funksjon i html dokumentet som tar seg av mottak av disse filene. Etter søk på internett om hvordan slike funksjoner er implementert i html, viser det seg av vår webserver ikke støtter så mange måter å gjøre dette på. Den mest vanlige måten er å benytte asp-script, men Java er også mye brukt. Dette er funksjoner som ikke støttes av thttpd webserveren i uClinux. En fant også et script for upload som brukte shell kommandoer for å ta imot filer, men disse shell komandoene ble for avanserte for uClinux sitt shell.

Alternativene til upload var om en kunne få tilgang på netflash kommandoen fra et html grensesnitt. Da kunne en bare legge ut bit filene en ønsket å benytte seg av på en hvilken som helst http-server eller ftp-server, og skrive inn adressen til serveren direkte inn i kommandolinjen til Suzaku via en nettleser. Dette kunne realiseres ved å enkelt lage en tekst basert <form> input, som vist i koden under:

```
<form action="kommando.cgi">Kommandolinje i uClinux (sh) <input type="text"
name="tekst" maxlength="100" size="80"> <input type="submit" value="enter">
</form>
```

Denne programsnutten er linket til cgi scriptet kommando.cgi som er vist i vedlegg 7. Programmet som er laget for å ta imot denne teksten heter prog og C-koden til dette programmet er vist i vedlegg 6. Ved å skrive ut den kommandoen en vil skal bli utført ved bruk av printf setningen, og å sette programmet i haker i cgi scriptet, vil den kommandoen som programmet prog skriver ut bli utført. En kan derfor skrive en hvilken som helst kommando i html dokumentet, prog gjør om QUERY_STRING til kommando, og den blir utført i uClinux og skrevet ut i et eget html vindu. I eksemplet vist i Figur 4.10 er den hele og ferdige websiden for Suzakuen vist. Kodene for html vinduene er vist i vedlegg 9, 10 og 11. Eksemplet viser en enkel kommando som viser hvilke filer som er under root til Suzakuen.



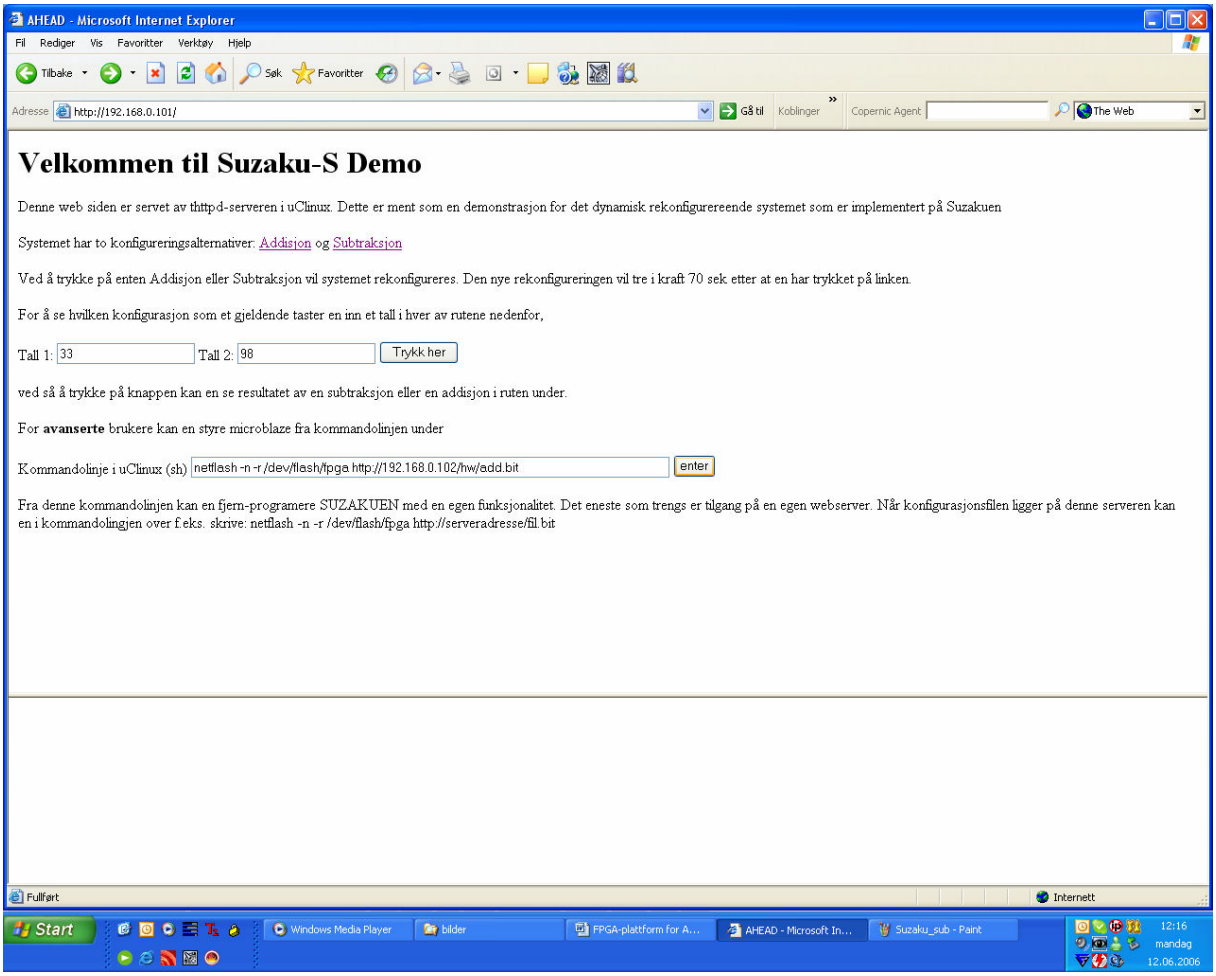
Figur 4.10 Suzaku demo med kommandolinje.

En kan nå se for seg at en skriver inn kommandoen for å rekonfigurere FPGAen direkte inn i kommandolinjen.

netflash –n –r /dev/flash/fpga http://adresse_til_serveren/fil.bit

Hvis en selv nå har lagt konfigureringsfilen fil.bit på sin egen http-server adresse_til_server, kan en konfigurere en Suzaku som befinner seg hvor som helst i verden. Og Suzakuen trenger ikke ha konstant tilgang til en egen server lenger.

Og trykke på enter i Figur 4.11 er nå likestilt med å trykke på linken Addisjon, bortsett fra at kommandolinjen ikke er låst til serveren 192.168.0.102 men kan benytte seg av en hvilken som helst server som har filen add.bit tilgjengelig.



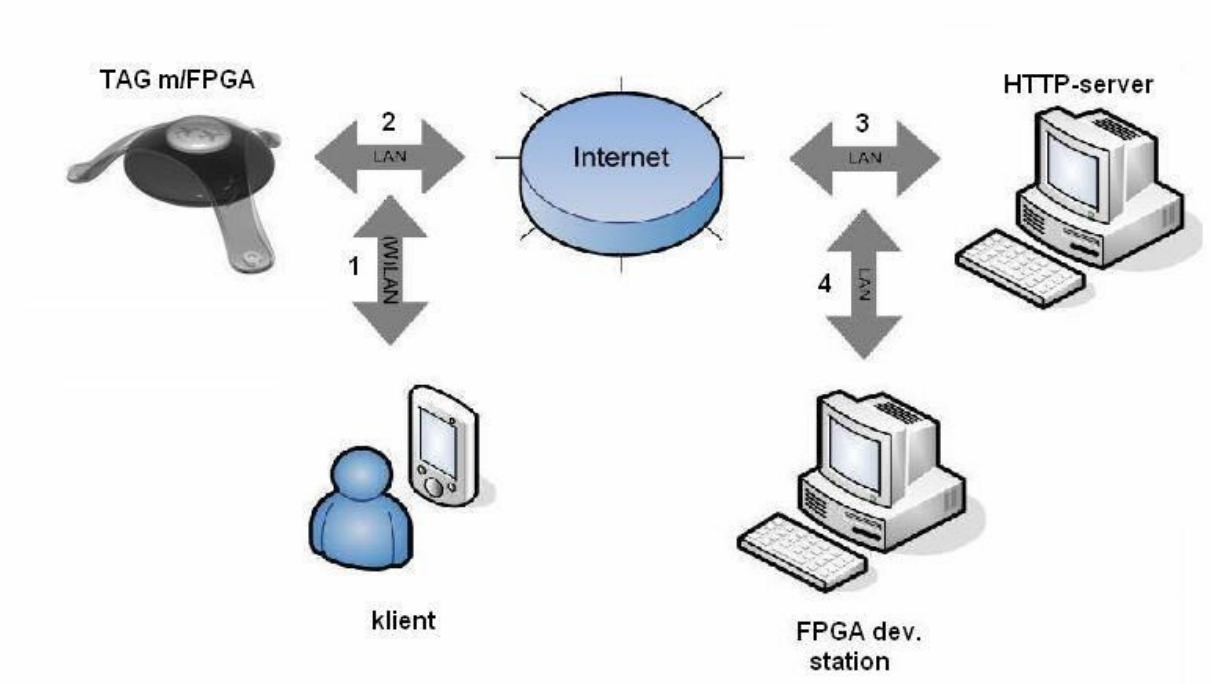
Figur 4.11 Konfigurering av FPGA via kommandolinje.

4.9 AHEAD versjon 1.

I Figur 4.12 er det vist en skisse av hvordan det ferdige systemet nå virker. Det er også prøvd og gitt en forklaring på hvordan kommunikasjonen foregår mellom de forskjellige enhetene i figuren.

For at systemet skal virke det nødt til å lages flere bit-filer med de ønskede funksjonaliteter til FPGAen. Dette skjer på utviklings PC-en, og de overføres derfra til en http-server via lokalt nettverk eller Internett (4). Når http-serveren er utstyrt med maskinvarebeskrivelser kan systemet begynne å fungere uavhengig av utviklings PC-en. Det som så skjer er at klienten logger seg på til tagen. Dette kjer ved at klienten skriver adressen til tagen i sin nettleser. Det er da en forutsetning at klienten er koblet på internett eller samme lokalenettverk som tagen (1). Nå som tagen og klienten er koblet sammen kan klienten styre tagen. Klienten ønsker nå at tagen skal utføre en addisjon på to tall som er for store til at klientens PDA takler det. Han velger derfor linken addisjon i nettleseren. Det som da skjer er at tagen, som er koblet til internett (2), starter scriptet add.cgi som henter en addisjons maskinbeskrivelse fra en http-server som også er koblet til internett (3). Maskinbeskrivelsen blir lastet inn i FLASH minnet på tagen. Tagen resettes og starter opp på nytt med den nye funksjonen. Klienten kan nå taste inn i nettleseren de to tallene en ønsker addert. Når en nå trykker på ”Trykk her”-knappen

starter resultat.cgi som benytter seg av ar.c programmet i uClinux, og vips er addisjonen utført og resultatet vist i nettleseren til klienten.



Figur 4.12 FPGA-plattform versjon 1.

4.10 En liten AHEAD visjon.

Med det arbeidet som er presentert i kapittel 4 kan en nå ta en tur inn i fremtiden for å se hva det som er gjort kan brukes til.

En kan tenke seg at en har kjøpt seg en PDA for en ukes tid siden. På TV går en reklame om AHEAD som selger budskapet om at en PDA kan utføre oppgaver på lik linje med en vanlig PC ved hjelp av en AHEAD-tag. For å få tilgang til disse ekstra funksjonene må en melde seg inn som AHEAD bruker. Som bruker får en tildelt en egen konto med passord, og på denne kontoen har en anledning til å kjøpe funksjoner som en kan utvide PDAen sin med. En har nå mange tilleggsfunksjoner på kontoen sin. Disse kan en programmere en AHEAD-tag med, og disse tagene finnes nå på hvert gatehjørne. Når en nå sitter på Starbucks og nyter en kaffe, kan en ta frem PDAen sin, programmere den AHEAD-tagen som er nærmest der en sitter, og vips har AHEAD brukeren en eller kanskje flere co-prosessorer til sin PDA.

Det tekniske aspektet bak denne muligheten er at det finnes en AHEAD-server. Denne serveren inneholder de konfigurasjonsfilene som trengs for de forskjellige funksjonene til tagen. Når en PDA kobler seg til denne tagen får en se en side lik den i Figur 4.11. Her tastes inn navnet på den ekstrafunksjonen en vil ha, samt passord. Da vil systemet ta imot dataen, og behandle dem i et cgi script som henter de filene en behøver for å utføre de nye funksjonene.

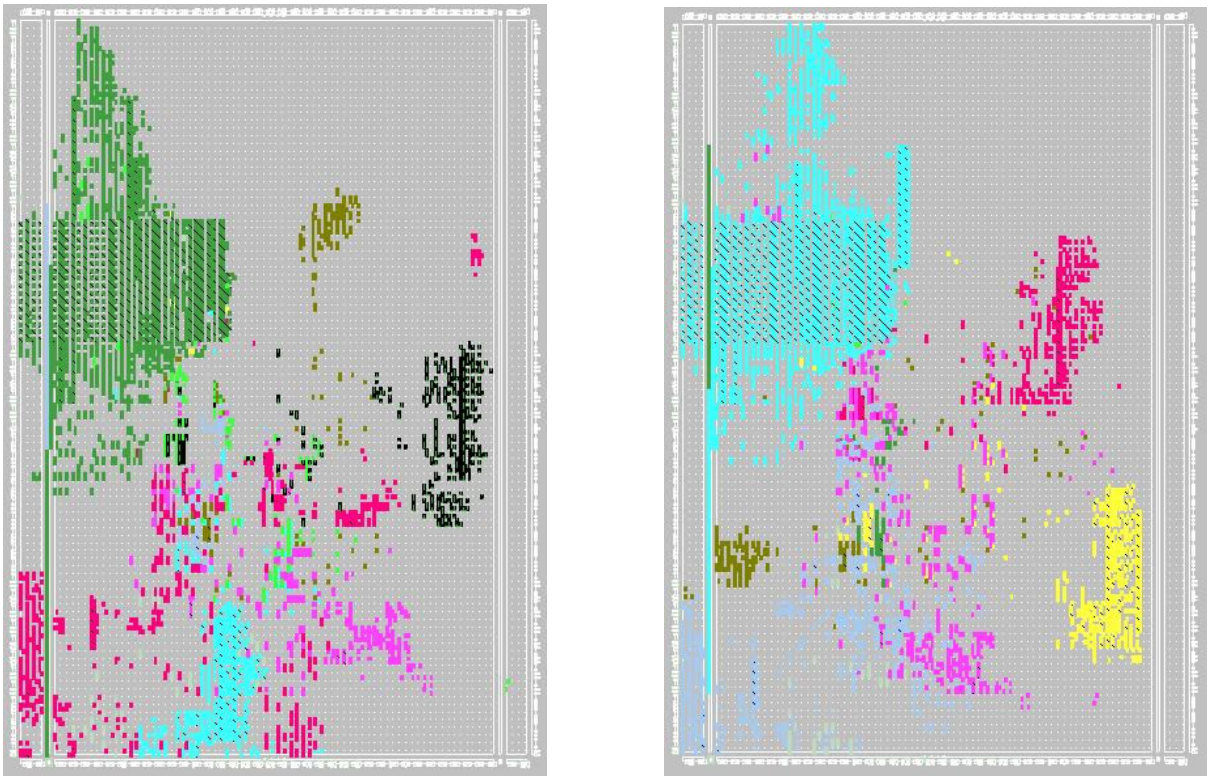
I dagens utgave, eller versjon 1, kan en bare hente en co-prosessor som utfører addisjon eller subtraksjon, men bare fantasien setter begrensinger for hva en kan gjøre med denne typen system. En USB-penn er genialt, mye ekstra lagringskapasitet i lommen. Hvorfor ikke mye ekstra prosessorkraft på hvert gatehjørne?

5 Samhandling med andre AHEAD oppgaver.

Det var i tillegg til denne diplomoppgaven tenkt 3 andre oppgaver innen AHEAD. Her følger en liten forklaring på disse oppgavene, og hvordan de kan samhandle med resultatene fra arbeidet gjort i denne oppgaven.

5.1 (De)komponering av FPGA-beskrivelser for AHEAD

Denne oppgaven går ut på å spesifisere og utvikle et system som skal kunne sette sammen ”dekomponerte” FPGA-beskrivelser, bit filer. Dette systemet skal kunne kjøre på en AHEAD-server-tag, og meningen er at det skal kunne utvikles AHEAD-komponenter ved å løsrive (”dekomponere”) dem fra den originale FPGAbeskrivelsen. AHEAD-serverens oppgave skal kunne være å ta imot en eller flere dekomponerte moduler og muligens referanser til andre kjente moduler, og så kunne sette disse sammen til en komplett FPGA-beskrivelse basert på en slags ”virtuell” plassering og ruting. Denne oppgaven vil i utgangspunktet være litt langt frem i tid i forhold til oppgaven som denne rapporten omhandler. Det som kanskje er verdt og merke seg er at en slik oppbygging av forskjellige moduler vil kreve at designet av bit filen, altså oppbyggingen av hvor på FPGAen de forskjellige funksjoner vil være blir mer krevende. I dagens design er det designverktøyet som tar seg av oppgaven med fysisk plassering av komponenter som microblaze, uart, LAN-kontroller, adder, subtraktor og andre komponenter som inngår i Suzakuen. Denne oppgaven må i fremtiden gjøres manuelt slik at en kan holde forskjellige funksjoner fysisk separert inne i FPGAen. Bare når en har full kontroll på dette kan en begynne å legge til andre funksjoner til et allerede eksisterende design. Et eksempel hvor tilfeldig plasseringene av komponentene i utviklingsverktøyet blir er vist i Figur 5.1.



Figur 5.1 Spartan3 med hhv adder(venstre svart) og subtraktor(høyre mørk rosa).

I tabellen under er det en oversikt over hvor store de fysiske forskjellene på de to designene er, men funksjonsmessig er forskjellen bare – og +.

	Spartan3 med adder	Spartan3 med subtraktor
system_timer	turkis	gul
adder/subtraktor	svart	mørk rosa
system_memcon	lys rosa	lys rosa
microblaze	Grønn	turkis
opb_gpio	lys blå	grønn
system_intc	Brun	brun

Tabell 1 Oversikt over komponentene i FPGA.

5.2 Bluetooth-basert klient-server for AHEAD

Denne delen av AHEAD ble startet med et prosjekt høsten 2005, prosjektet har utviklet en modell og program for nærhets-deteksjon for en AHEAD tagserver mht en AHEAD-klient. Dette skal videreutvikles til et klient-server system i både program- og maskinvare for en PDA som kjører Linux og et system (AHEAD-tag) som inneholder både en FPGA og en prosessor ombord. Systemet skal kunne overføre en lagret FPGA-beskrivelse fra PDA'en og over til tag'ens prosessor via Bluetooth trådløst nettverk (evt. WLAN), og gjøre denne beskrivelsen tilgjengelig for FPGA-systeme. Det skal så overføres nødvendige data som eksekveres på denne FPGAarkitekturen vha. tag-prosessor og kommunikasjons-systemet i mellom dem. Resultatene skal så returneres til PDA for presentasjon. Denne delen av AHEAD systemet er det ikke gått videre med til diplomoppgave, så her er det ikke skjedd noe mer. Dette er den delen som vil måtte samkjøre mest med de oppgavene som denne rapporten omhandler. Per dags dato er det laget et grensesnitt opp mot webserver og nettleser. Det er dette grensesnittet som mest sannsynlig skal erstattes av et bluetooth basert grensesnitt.

5.3 Arkitektur-beskrivelser for AHEAD

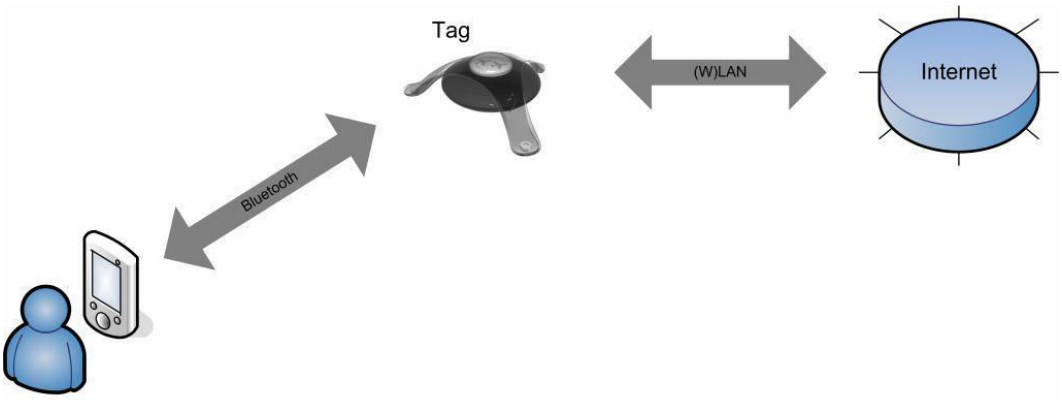
Den tredje oppgaven innen AHEAD består i å generalisere AHEAD-plattformen. Det skal studeres såkalte "Architecture Description Languages", ADL. Et prosjekt høsten 2005 har undersøkt dette mulige AADL (AHEAD ADL), og kommet med en del forslag til hvordan dette kan brukes for en tidlig generasjon AHEAD klient-server funksjonalitet. Det er ønskelig å analysere dette mht å utvide det til å kunne dekke nye generasjoner av AHEAD med mer avansert funksjonalitet, og også å analysere evt. spesifisere et system for hvordan å utvikle og håndtere slike beskrivelser basert på de faktiske verktøy for FPGA-utvikling som finnes i dag.

6 Fremtidig arbeid.

Det er med denne oppgaven laget versjon 1 av FPGA-plattformen for AHEAD. Videre arbeid på dette området vil føre til nye versjoner med mer avanserte funksjoner, og forhåpentligvis vil denne rapporten hjelpe det videre arbeidet på vei mot dette. Med erfaringene som er gjort foreslås det her en del forskjellige oppgaver som vil være relevant og ta tak i for nye versjoner av FPGA-plattformen.

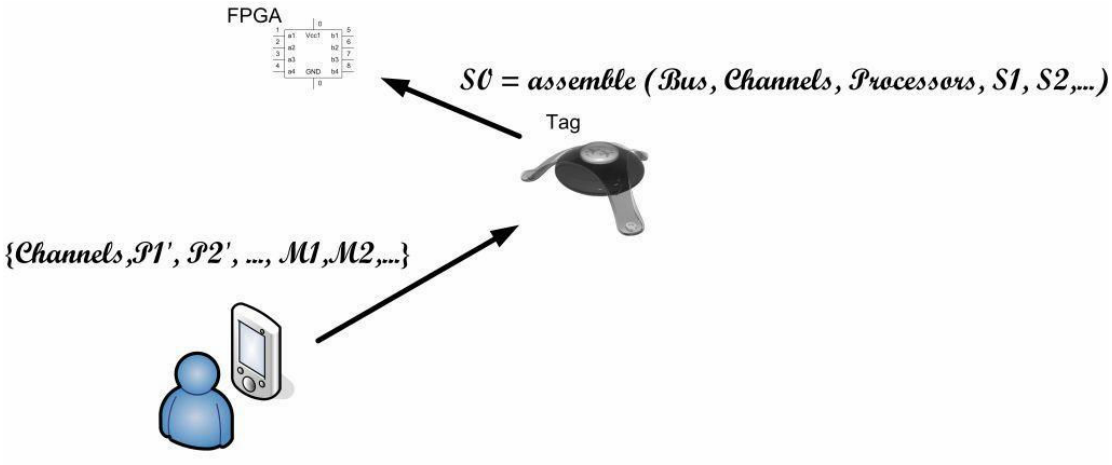
6.1 AHEAD versjonene.

Denne første versjon av AHEAD er som vist i Figur 4.12. Den versjonen som ble vist i Figur 1.1 er antageligvis ikke den neste versjonen, men kanskje versjon 4 eller 5. Den neste versjonen er kanskje en som likner på den en har i dag, men med implementert bluetooth som kommunikasjon mellom klient og tag, Figur 6.1.



Figur 6.1 AHEAD versjon 2.

Den neste versjonen etter dette vil kanskje være å få til det arkitektur beskrivende språket, slik at maskinvarebeskrivelsene ikke lenger hentes fra en http-server, men settes sammen internt på tagen ved å tolke etterspørselen fra klienten. I Figur 6.2 er det vist en klient som forespør en funksjonalitet som tagen gjør om til maskinvarebeskrivelse som den kan programmere FPGAen med.



Figur 6.2 AHEAD versjon 3.

Etter dette kommer kanskje eksemplet med MPEG-rekoding som en eventuell AHEAD versjon 4. Det som er sikkert er at det er mye som skal på plass før en når målet. Kanskje vil MPEG-rekoding være aktuelt før en klarer å implementere et arkitekturbeskrivende språk, slik at disse to versjonene bytter nummer i rekken, det vil tiden vise.

6.2 uClinux

For å være litt dristig kan en prøve å få et samarbeid med en av de andre instituttene på den delen av AHEAD som har med uClinux å gjøre. Det å bygge opp en slikt system er avansert og litt på kanten mye nytt å lære for en som studerer system konstruksjon. Hvis en kunne fått en student som har mer erfaring og kunnskap rundt bygging av operativsystem rundt en gitt mikroprosessor ville en kommet langt på dette området og kanskje avklart en del ting slik at en kunne lette det videre arbeidet rundt akkurat denne biten.

6.3 Partiell rekonfigurering.

Det må på et tidspunkt vurderes og utforskes mulighetene for partiell rekonfigurasjon. Dette området er blitt utforsket en del allerede på instituttet, men med den nye versjon 8.1 av ISE og XPS har Xilinx lagt ned mye arbeid i å forbedre støtten for denne typen design. Et design som bygger på partiell rekonfigurering vil kunne minske tiden det tar for systemet og skifte funksjonalitet betraktelig. Forutsetningene for en slik oppgave må være at en har en utviklingsplattform med en FPGA som støtter denne typen konfigurering, og per dags dato finnes ikke en slik på instituttet.

7 Diskusjon.

Dette prosjektet startet med å finne dokumentasjon på liknende systemer, for å se om en kunne dra nytte av ting som andre har gjort tidligere. De fleste systemer som er prøvd laget for liknende formål har hovedsakelig fokus på partiell rekonfigurering, mens dette prosjektet ikke har utforsket det aspektet. Rapporter fra slike systemer er heller ikke så detaljerte som en kunne ønske. Det fortelles om hvordan ting er gjort, men det er som regel veldig abstrakt og udetaljert. En får imidlertid et inntrykk av hvordan enkelte problemstillinger er løst, og kan dermed dra nytte av slike ideer videre i oppgaven.

Når det gjelder måten dette systemet rekonfigurerees på er det selvfølgelig et problem at det tar 70 sekunder. Denne tiden inkluderer nedlasting av konfigurerings filen, programmering av FLASH minnet og resetting av systemet som innebærer en boot sekvens for uClinux. Det er dessverre ingen av disse tidkrevende aspektene en kommer unna med mindre en tyr til partiell rekonfigurering. Skal systemet i denne oppgaven skifte funksjon, må en bare belage seg på at det tar tid. I senere versjoner av dette systemet vil nok ikke dette bli akseptert, derfor kan en allerede nå begynne å diskutere hvorvidt en skal begynne med partiell rekonfigurering.

Et annet aspekt ved denne oppgaven er at med de begrensede kommandoer og funksjoner som ligger i et OS som uClinux, vil det også være begrenset med muligheter. Som et eksempel kan en nevne mulighetene for å kjøre cgi skript med vanlige shell kommandoer. Hadde en hatt bedre tid kunne en utforsket uClinux mer med tanke på og utvide funksjonaliteten en del. Hadde en gjort det kunne en enklere ha laget mer avanserte systemer og programmer på Suzakuen.

Når det gjelder det eksisterende webgrensesnittet som er brukt i denne oppgaven, er det bare en brøkdel av det som er mulig med html programmering som er dratt nytte av her. Hvis en hadde hatt mer kunnskap om akkurat dette, hadde en kunnet gjort mer avanserte ting. Å bruke en webserver og tilhørende nettlesere er et bra grensesnitt som kanskje er på høyde med det tenkte bluetooth grensesnittet?

I denne oppgaven ble aldri CerfCube brukt. Dette fordi rekonfigureringen av FPGAen ikke var avhengig av denne mikroprosessoren. Det førte til at en ikke trengte bruke denne for å komme frem til den første versjonen av AHEAD. Skulle denne oppgaven tatt i bruk CerfCube måtte en grensesnitt mellom denne og Suzakuen ha blitt definert. Det ble derfor valgt å holde ting litt enklere, å derfor ble CerfCube ikke brukt. Etter hvert vil AHEAD systemet bli mer avansert. Da er det ikke sikkert at Microblaze er tilstrekkelig, og CerfCube må tas ibruk.

For å implementere det tenkte MPEG eksemplet er det mye informasjon om MPEG-rekoding som må hentes. Hvordan en koder MPEG-rekoding i vhdl er en forutsetning, men også at en har bluetooth eller webserver-grensesnitt som støtter dette. Først når disse tingene er på plass vil dette eksemplet vise hva en kan bruke et slikt system til i virkeligheten.

8 Konklusjon.

Etter å ha jobbet med denne oppgaven i hele vår har en fått mange nye erfaringer og resultater. En har laget nye programmer i uClinux spesialisert for de implementerte maskinvarebeskrivelsene. Det er laget flere maskinvarebeskrivelser som en kan dynamisk bytte mellom på FPGA-plattformen. En har laget et html- og cgi-basert grensesnitt mellom AHEAD-server og klient ved hjelp av henholdsvis webserver og nettleser. Alle disse forskjellige aspektene er satt sammen og en kan derfor tross mange hindringer si at en er kommet frem til en FPGA-plattform som en trygt kan kalle versjon 1 i AHEAD sammenheng. Plattformen fungerer etter de gitte spesifikasjonene, fordi en kan med mobilt utstyr som for eksempel PDA, rekonfigurere plattformen dynamisk med forskjellig funksjonalitet. PDAen kan også lese og se resultater fra de forskjellige typene funksjoner som plattformen konfigureres til. En kan si at en har gått det første skrittet mot et fullstendig AHEAD system. Systemet kan nå videreutvikles med trådløs kommunikasjon som for eksempel WLAN eller bluetooth for å komme nærmere et komplett AHEAD system.

9 Referanser.

[1] Petersen S., Rognlien D., Svarstad K., (2001). Context Tag Technology. NTNU / SINTEF.

[2] Andre DeHon, " The DensityAdvantage of Configurable Computing", April 2000

[3] Matthew Scarpino, "The hoplite guide to Run-Time Reconfigurable computing", 08.03.2004

[4] S. Guccione, Xilinx, San Jose, CA, D. Verkest, IMEC, Leuven, Belgium, I.Bolsens, Xilinx, San Jose, CA, "Design Technology for Networked Reconfigurable FPGA Platforms", 2002

[5] Atmark Techno, Inc., "SUZAKU Software Manual", January 31, 2005

[6] Atmark Techno, Inc., "SUZAKU Hardware Manual", June 15, 2004

[7] Xilinx, http://www.xilinx.com/

[8] Suzaku, http://suzaku-en.atmark-techno.com/

[9] Dokumentasjons CD for Suzaku, http://suzaku-en.atmark-techno.com/downloads/all

Vedlegg

Vedlegg 1 – Bit2Flash.c
Dette er et beta-program og brukes KUN på eget ansvar.

```
/* bit2flash.c */
/* Convert Xilinx bit file (from bitgen) to flash image
 * suitable for Suzaku S board (with 4M Flash).
 * step 1: discard the info header of bitfile
 * step 2: pad 0xFF to make a 512K image
 */
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
        FILE *infile;
        FILE *outfile;
        union {
                unsigned int i;
                unsigned short s[2];
                char c[4];
        } head_len;
        unsigned char head_key;

        int pad_len;
        int i;
        char buf[1024];

        if (argc != 3) {
                printf("usage: %s <infile> <outfile>\n", argv[0]);
                exit(0);
        }

        infile = fopen(argv[1], "r");
        if (infile == NULL) {
                printf("Err: cannot find bit file %s\n", argv[1]);
                exit(0);
        }

        outfile = fopen(argv[2], "w");
        if (outfile == NULL) {
                printf("Err: cannot create bit file %s\n", argv[2]);
                exit(0);
        }

        fread(&(head_len.c[1]), 1, 1, infile);
        fread(&(head_len.c[0]), 1, 1, infile);
        fread(buf, head_len.s[0], 1, infile);
        fread(&(head_len.c[1]), 1, 1, infile);
        fread(&(head_len.c[0]), 1, 1, infile);

        head_key = 0;
        while (head_key != 0x65) {
                fread(&head_key, 1, 1, infile);
                fread(&(head_len.c[3]), 1, 1, infile);
                fread(&(head_len.c[2]), 1, 1, infile);
                if (head_key == 0x65) {
                        fread(&(head_len.c[1]), 1, 1, infile);
                        fread(&(head_len.c[0]), 1, 1, infile);
                        printf("size : %d bytes\n", head_len.i);
                        fread(buf, head_len.i%1024, 1, infile);
```

```
                        fwrite(buf, head_len.i%1024, 1, outfile);
                        for (i=0; i<head_len.i/1024; i++) {
                                fread(buf, 1024, 1, infile);
                                fwrite(buf, 1024, 1, outfile);
                        }
                }
                else {
                        fread(buf, head_len.s[1], 1, infile);
                        printf("%s\n", buf);
                }
        }
        fclose(infile);

        for (i=0; i<1024; i++)
                buf[i] = 0xFF;
        pad_len = 512*1024 - head_len.i;
        fwrite(buf, pad_len%1024, 1, outfile);
        for (i=0; i<pad_len/1024; i++)
                fwrite(buf, 1024, 1, outfile);
        fclose(outfile);

        return 0;
}
```

Vedlegg 2 - add.cgi og sub.cgi

add.cgi

```
#!/bin/sh
echo Content-type: text/html
echo "<html><body><pre>"
echo "Vent i 70 sekunder, deretter oppdater siden."
echo "</pre></body></html>"
netflash -nk -r /dev/flash/fpga http://192.168.0.102/hw/add.bit
```

sub.cgi

```
#!/bin/sh
echo Content-type: text/html
echo "<html><body><pre>"
echo "Vent i 70 sekunder, deretter oppdater siden."
echo "</pre></body></html>"
netflash -nk -r /dev/flash/fpga http://192.168.0.102/hw/subtr.bit
```

Vedlegg 3 – ar_arg.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>

int main(int argc, char * argv[])
{
      int tall;
      tall = atoi(argv[1]);
      printf("Aritmetisk operasjon.\n");
      printf("A =  %d \n", tall);
      *(volatile unsigned int *)(0x81000000+0) = tall;
      tall = atoi(argv[2]);
      printf("B = %d \n", tall);
      *(volatile unsigned int *)(0x81000000+4) = tall;
      printf("A + B = %d \n", *(volatile unsigned int *)(0x81000000+8));
      return 0;
}
```

Vedlegg 3 – action.cgi

```
#!/bin/sh
echo Content-type: text/html
echo
echo "<html><body>"
echo "<p>"

LINJE=`echo ${QUERY_STRING} | sed 's/&/ /g'`

for LOOP in $LINJE
do
ID=`echo $LOOP | sed 's/=/ /g' | awk '{print $1}'`
VERDI=`echo $LOOP | sed 's/=/ /g' | awk '{gsub("%","\\\x"); print $2}' |
sed 's/+/ /g'`
V=`printf "$VERDI"`
eval `echo "${ID}=\"${V}\""`
done

ar_arg $t1 $t2

echo "</p>"
echo "</body></html>"
```

Vedlegg 4 – ar.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char * argv[])
{
      int tall;
      char * arg;
      if (argc < 1) return 0;
      arg = argv[1];
      while (*arg++ != '=');
      tall = atoi(arg);
      printf("Aritmetisk operasjon.\n");
      printf("X = %d \n", tall);
      *(volatile unsigned int *)(0x81000000+0) = tall;
      while (*arg++ != '=');
      tall = atoi(arg);
      printf("Y = %d \n", tall);
      *(volatile unsigned int *)(0x81000000+4) = tall;
      printf("X (+-) Y = %d \n", *(unsigned volatile int *)(0x81000000+8));
      return 0;
}
```

Vedlegg 5 – resultat.cgi

```
#!/bin/sh
echo Content-type: text/html
echo
echo "<html><body><pre>"

ar $QUERY_STRING

echo "</pre></body></html>"
```

Vedlegg 6 – prog.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char * argv[])
{
	char * inn;
	char * ut;
	int count = 0;
	int i;
	inn = argv[1];
	while(*inn++ != '=');
	while(*inn != 0) {
		if(*inn == '%') {
			inn++;
			if(*inn == '2') {
				*ut = '/';
			}
			else if(*inn == '3') {
				*ut = ':';
			}
			else {
				printf("error: feil tegn\n");
				return 0;
			}
			inn++;
			inn++;
			ut++;
		}
		else if(*inn == '+') {
			*ut = ' ';
			ut++;
			inn++;
		}
		else {
			*ut = *inn;
			ut++;
			inn++;
		}
		count++;
	}
	*ut = '\0';
	for(i=0; i < count; i++) ut--;
	printf("%s \n", ut);
}
```

Vedlegg 7 – kommando.cgi

```
#!/bin/sh
echo Content-type: text/html
echo
echo "<html><body><p>"
`prog $QUERY_STRING`
echo "</p></body></html>"
```

Vedlegg 8 – Makefile

```
# ROOTDIR = /usr/src/uClinux-dist
ifndef ROOTDIR
ROOTDIR=/home/stian/test_5/uClinux-dist-20040408-suzaku6
endif
ROMFSDIR  = $(ROOTDIR)/romfs
ROMFSINST = romfs-inst.sh
PATH      := $(PATH):$(ROOTDIR)/tools

UCLINUX_BUILD_USER = 1
include $(ROOTDIR)/.config
LIBCDIR = $(CONFIG_LIBCDIR)
include $(ROOTDIR)/config.arch


EXEC = name
OBJS = name.o


all: $(EXEC)

$(EXEC): $(OBJS)
      $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

clean:
      -rm -f $(EXEC) *.elf *.gdb *.o

romfs:
      $(ROMFSINST) /bin/$(EXEC)

%.o: %.c
      $(CC) -c $(CFLAGS) -o $@ $<
```

Vedlegg 9 – index.html

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>AHEAD</title>
</head>

<frameset rows="75%,25%">
      <frame name="topp" src="hoved.htm" target="bunn">
      <frame name="bunn" src="vindu.htm">
      <noframes>
      <body>

      <p>Denne siden bruker rammer, men leseren din støtter ikke disse.</p>

      </body>
      </noframes>
</frameset>

</html>
```

Vedlegg 10 – vindu.htm

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>resultatside</title>
</head>

<body>

</body>

</html>
```

Vedlegg 11 – hoved.htm

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Suzaku-S</title>
<base target="bunn">
</head>

<body>
      <h1>Velkommen til Suzaku-S Demo</h1>
      <p>Denne web siden er servet av thttpd-serveren i uClinux. Dette er
ment som en demonstrasjon for
      det dynamisk rekonfigurereende systemet som er implementert på
Suzakuen</p>
      <p>Systemet har to konfigureringsalternativer:
      <a href="add.cgi">Addisjon</a> og <a
href="sub.cgi">Subtraksjon</a></p>
      <p>Ved å trykke på enten Addisjon eller Subtraksjon vil systemet
rekonfigureres.
      Den nye rekonfigureringen vil tre i kraft 70 sek etter at en har
trykket på
      linken. </p>
      <p>For å se hvilken konfigurasjon som et gjeldende taster en inn et
tall i hver av rutene nedenfor,
      <p><form action="resultat.cgi">
      Tall 1: <input name="t1"> Tall 2: <input name="t2"> <input
type=submit value="Trykk her"></p></form>
      ved så å trykke på knappen  kan en se resultatet av en subtraksjon
eller en addisjon
      i ruten under.</p>
      <p>For <b>avanserte</b> brukere kan en styre microblaze fra
kommandolinjen under</p>
      <p><form action="kommando.cgi">Kommandolinje i uClinux (sh) <input
type="text" name="tekst"
      maxlength="100" size="80"> <input type="submit"
value="enter"></form></p>
      <p>Fra denne kommandolinjen kan en fjern-programere SUZAKUEN med en
egen funksjonalitet.
Det eneste som trengs er tilgang på en egen webserver. Når
konfigurasjonsfilen ligger på denne serveren kan en i
kommandolingjen over f.eks. skrive: netflash -n -r /dev/flash/fpga
http://serveradresse/fil.bit</p>
</body>

</html>
```