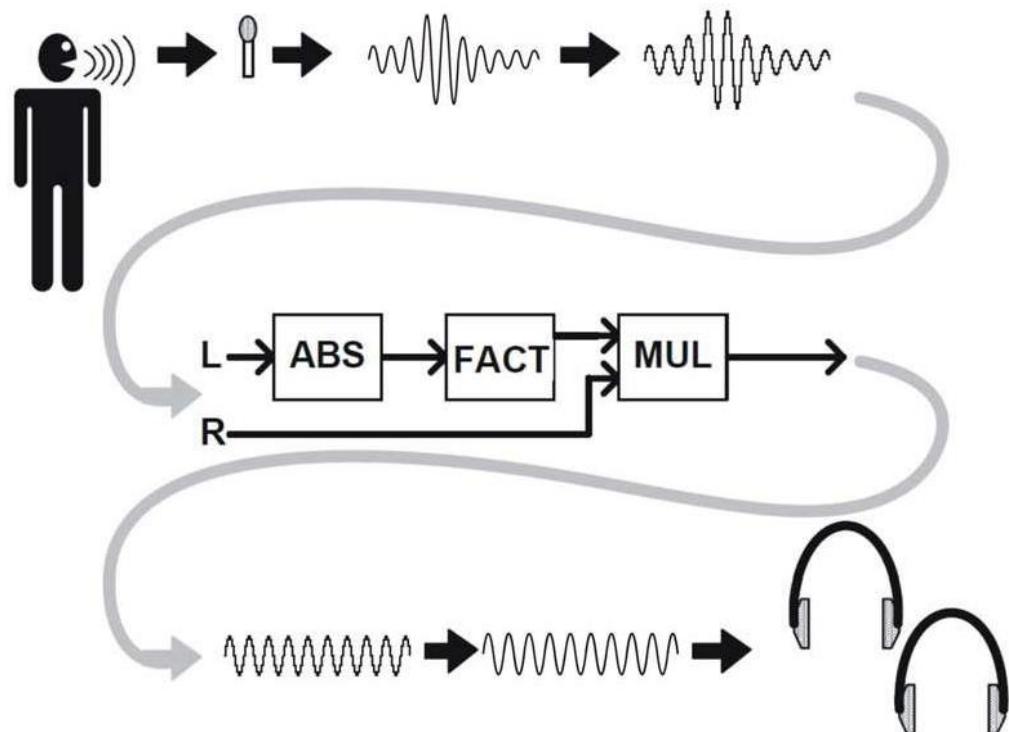


# TFE4208

## Innlevde systemer designprosjekt



**LAB**

**Vår 2023**

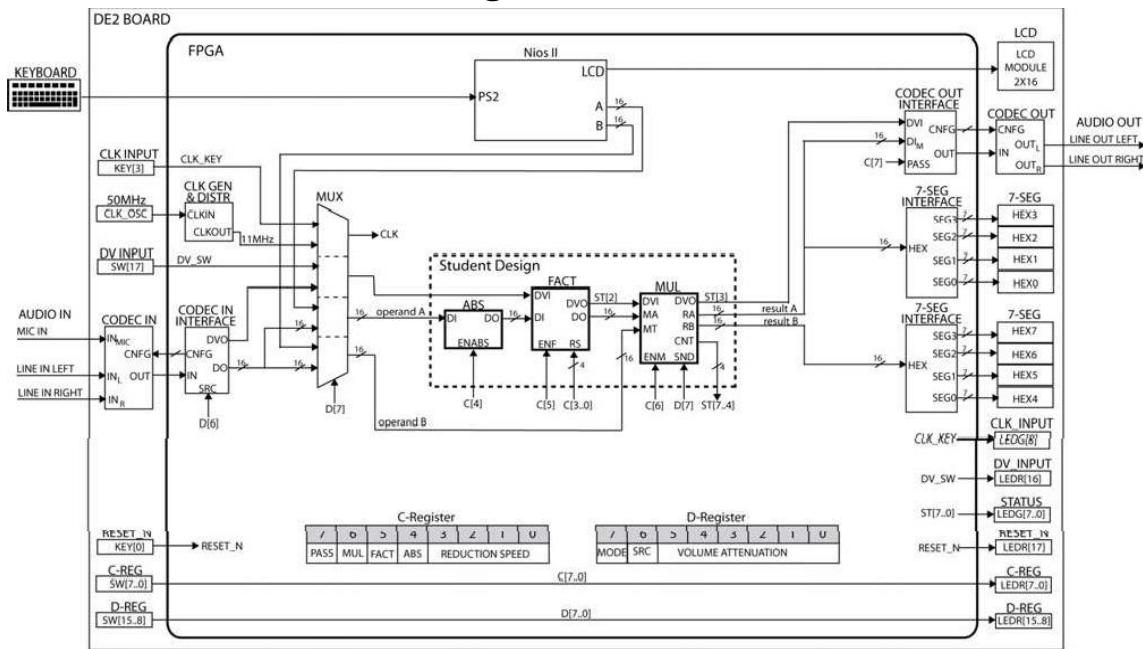


Fakultet for informasjonsteknologi  
og elektronikk

---

**Institutt for elektroniske systemer**

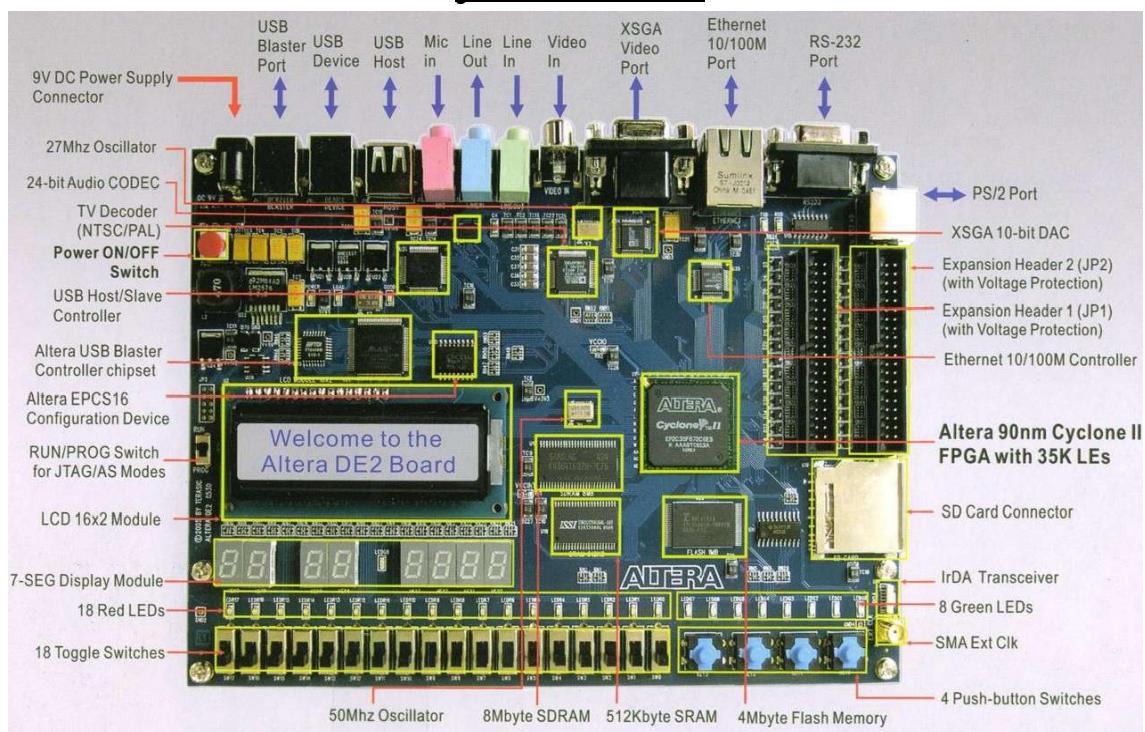
## Logisk skisse



Merk på den logiske skissen:

- De to Codec-symbolene (CODECIN og CODECOUT) er *en* og samme fysiske krets. Tilsvarende er Codec grensesnittene (In og Out Interface) *en* og samme modul i FPGA'en.
- Klokkesignaler er ikke tegnet i detalj på skissen. Generering og distribuering av disse er illustrert med modulen CLK GEN & DISTR.

## Fysisk skisse



# Forord

*TFE 4208 Embedded Systems Design Project* inneholder en omfattende lab basert på et lab-kort fra Altera, en av verdens største produsenter av såkalte programmerbare logiske kretser. Lab-kortet er bygget rundt en brikke med programmerbar logikk (såkalt Field Programmable Gate Array, forkortet til FPGA). Inne i denne FPGA-en er det dessuten implementert en enkel mikroprosessor ( $\mu$ P). Vi kan lett realisere ulike logiske funksjoner i maskinvare ved å legge disse inn i FPGA-kretsen. Dette skjer ved at vi laster ned en kodefil til kretsen som sørger for å konfigurere den i forhold til ønsket funksjon (dette kalles gjerne å *programmere kretsen*, men har ingen ting med programvare å gjøre). Slik kan systemet få ønsket logisk oppførsel uten at vi trenger å bruke loddebolten. I mikrokontrolleren kan vi kjøre våre C- eller assembler-programmer som også er med på å definere systemets oppførsel. Vi kan dermed demonstrere en rekke viktige prinsipper både fra digitalteknikk- og datamaskin-delen av faget. I tillegg til FPGA med  $\mu$ P, er lab-kortet (blant annet) utstyrt med brytere, lysdioder, LCD-display og en lyd-krets (audio codec), samt tilkobling til tastatur.

Vi ønsker at du gjennom denne labben skal få en grunnleggende innsikt i hvordan moderne digitale systemer og små datamaskiner kan bygges opp. Gjennom LAB2 til LAB5 vil du jobbe med problemstillinger knyttet til aktuelt forelest stoff. Samtidig vil du og din labpartner, ved å løse konkrete problemer, også realisere ulike deler av et «produkt»: et system for «samtids» lyd-bearbeiding. Vi håper at lab-oppgavene vil hjelpe deg å forstå hvordan ulike begreper og teknikker i faget er knyttet til hverandre i et virkelig system, og dermed bidra til å gi en helhetsforståelse. Videre er det et mål at lab-arbeidet skal gi deg et innblikk i arbeidsmetoder og verktøy som vil bli sentrale i videre studier og arbeid.

Hovedmålsettingen har vært å inkludere viktige elementer fra faget på en pedagogisk og lærings-stimulerende måte. Lyd-bearbeiding ble valgt som tema da både generering av input og tilbakemelding fra systemet skulle være enkelt å få til. Videre skulle det gi et system med realistisk kompleksitet.

**Husk at læringen i faget står du selv ansvarlig for.** Det vi tilbyr er kun undervisning og veiledning, og et forhåpentligvis godt tilrettelagt opplegg. Som student må du være aktiv, kritisk og selvstendig, samtidig som du gjennom lab- og øvings-arbeid i grupper kan bearbeide og drøfte stoffet med medstudenter og dermed fremme læring og faglig modning. Vi vil kreve at du er godt forberedt når du møter fram på labben. Dette er nødvendig fordi vi bare har begrenset plass og tid på labben.

Fram til og med 2007 ble det i dette faget benyttet et egenutviklet kretskort. Dette er nå byttet ut, men oppgavene som kjøres i dag bygger i stor grad på det som ble gjort tidligere. Vi er derfor en stor takk skyldig et entusiastisk team (se neste side) av faglærere, vit.asser, stipendiater og studenter, som har vært involvert i planleggingen, ferdigstillingen og videreutviklingen av lab-kort og lab-oppgaver opp gjennom årene.

Det opprinnelige utviklingsteamet var, i tillegg til faglærerne Tormod Njølstad og Lasse Natvig, ledet av nøkkelpersonene Gunnar Tufte og Bianca Tjore. Videre bidro Kjell Magne Sæterbø, Eivind Vea, Åge Stien, Mufrid Krilic, Per Blix, Karstein Kristiansen, Knut Førland, Steinar Line, Ingebrigt Hole, Robin Hoel og Morten Hartmann vesentlig på hver sin måte til ferdigstillelsen og til gjennomføringen av teoriøvinger og lab-opplegg i 1998, 1999 og 2000. Somrene 2001 – 2003 oppdaterte henholdsvis Are B. Willumsen, Øystein Andres Krogsæter og Øystein Ellingsson ulike deler av oppgavene og labheftet. Våren 2004 utviklet Øystein Ellingsson en ny lab 1 for å gi en lettere innføring i digital design. Denne ble forbedret av Hallvard Kringstad våren 2007. Sommeren 2008 har Per Gunnar Kjeldsberg og Ingulf Helland jobbet med å flytte lab-oppgavene over på det nye kortet fra Altera og å tilpasse oppgavene og labheftet til dette. I denne omleggingen ble lydoppgaven endret fra en automatisk volumkontroll til en lydkompressor. Somrene 2009 og 2010 er det gjort forbedringer basert på erfaringene fra de første kjøringene av den nye labben. Senere år er det gjort mindre språklige oppdateringer. I løpet av høstsemesteret 2013 ryddet vit.ass Yahya H. Yassin opp i konfigurasjonfilene på alle labøvingene slik at alle feil relatert til kryssing av klokkedomener ble løst på riktig måte. I tillegg ble ASM-delen i labøving 4 tatt ut av pensum. En ny lab 4 ble laget av Yahya H. Yassin der ASM-delen ble byttet ut med en tilsvarende tilstandsmaskin. En feil ble oppdaget på en tegning i vedlegget høsten 2013, og denne ble rettet opp høsten 2014.

Vi vil gjerne oppfordre dere som nå skal følge dette kurset til å gi konstruktive tilbakemeldinger om mulige forbedringer til fagets øvingsansvarlige.

På vegne av teamet ønsker vi dere herved: LYKKE TIL!

**Professor Per Gunnar Kjeldsberg**

Koordinator for TFE4208.

## Innhold

<b><u>FORORD</u></b>	<b>1</b>
<b><u>ORDLISTE</u></b>	<b>VIII</b>
<b>CODEC</b>	<b>IX</b>
<b><u>INTRODUKSJON</u></b>	<b>11</b>
OVERSIKT	11
SIGNALBEHANDLINGSSYSTEMETS VIRKEMÅTE	12
DESIGNVERKTØY	14
DESIGNMETODIKK	15
LAB-ARBEIDSMÅTE	15
<b><u>1 LABORATORIEØVING:</u></b>	<b>17</b>
<b><u>2 LABORATORIEØVING: ABSOLUTTVERDIKRETS 16BIT - FPGA</u></b>	<b>18</b>
INNLEDNING	18
TEORI	19
FORARBEID	20
LABORATORIEARBEID	28
<b><u>3 LABORATORIEØVING: DETEKSJON AV FORSTERKNINGSFAKTOR</u></b>	<b>47</b>
INNLEDNING	47
SYSTEMBESKRIVELSE	48
FORARBEID	52
LABARBEID	53
<b><u>4 LABORATORIEØVING</u></b>	<b>60</b>
INNLEDNING	60
MODULBESKRIVELSE	61
FORARBEID	65
LABARBEID	70
<b><u>5 LABORATORIEØVING: VOLUMSTYRING AV SYSTEMET MED ENKEL PROSESSOR</u></b>	<b>78</b>
INNLEDNING	78
BESKRIVELSE	78
FORARBEID	84

<b>LABARBEID</b>	<b>87</b>
<b><u>6 VEILEDNING FOR QUARTUS II</u></b>	<b>91</b>
GRUNNLEGGENDE FUNKSJONALITET I BLOKKSJEMAEDITOREN	91
OVERSIKT OVER VERKTØY FOR BLOKKSJEMAREDIGERING	98
OFTE BRUKTE PROSEDYRER I QUARTUS	99
<b><u>7 OPPKOBLING OG BRUK - ALTERA DE2</u></b>	<b>107</b>
BRUK AV ANTISTATISK ARMLENKE	107
ALTERA DE2 - OPPKOBLING OG BRUK	108
<b><u>8 TEKTRONIX TDS-2000-SERIEN OSCILLOSKOP</u></b>	<b>111</b>
TILKOBLINGER	112
PROSEDYRE FOR ENKELT MÅLEOPPSETT	113
KONTROLLER VALIDITETEN AV MÅLERESULTATET - VOLT/DIV OG SEC/DIV	113
LAGRING AV OSCILLOSKOPBILDER PÅ PC	113
MÅLETEKNIKK	114
STRØMMÅLING	115
SIKKERHET	116
BETJENINGSKNAPPER	116
HJELP OG TILBAKESTILLING AV OSCILLOSKOPET	117
BETJENINGSKNAPPER UNDER «HORIZONTAL»	119
KNAPPER FOR DATAINNSAMLINGSMODUS	121
MÅLEPROBER	125
PROSEDYRE FOR KALIBRERING AV PROBER	126
<b><u>9 SIGNALGENERATOREN</u></b>	<b>127</b>
VALG AV FREKVENS	128
KURVEFORM	128
AMPLITUDE, NULLPUNKT OG SIGNALFORHOLD	129
VANLIGE BRUKERFEIL	130
<b><u>10 VEDLEGG: VHDL – EN INTRODUKSJON</u></b>	<b>131</b>
GENERELT	131
ANNET	135
<b><u>11 DATABLADE FOR PORTKRETSER</u></b>	<b>137</b>
FORKLARING AV FORKORTELSER	137
TIMINGDATA FOR CD4030B	138
TIMING DATA FOR CD4081B	139
PINNEKONFIGURASJON FOR CD4011(U)B, CD4030B OG CD4081B	140

<b><u>12 RYDDING AV LABORATORIEBENK</u></b>	<b><u>141</u></b>
<b>UTSTYR PÅ LABORATORIEBENKEN</b>	<b>142</b>
<b>UTSTYR PÅ LABORATORIEBENKENS HYLLE</b>	<b>143</b>

# Ordliste

Norsk	Engelsk	Betydning
Programmerbar logikk-krets	Programmable Logic Circuit	En digital krets som kan <i>konfigureres</i> (programmereres)
Altera	Altera	Et firma i San Jose som lager programmerbare logikk-kretser Brukes også om selve <i>kretsene</i>
FPGA	FPGA	Forkortelse for Field Programmable Gate Array Dette er en type programmerbar logikk-krets
LE	LE	Forkortelse for Logic Element. Dette er «byggeklossene» inni Altera-kretsen (den har 33 216 LE-er)
Ruting	Routing	Å lage en elektrisk forbindelse fra et sted til et annet
Nios II	Nios II	Mikrokontrolleren som er implementert inne i FPGA-en.
Punktprøving	Sampling	Omforming av et kontinuerlig analogt signal (f.eks. et lydsignal) til et digitalt signal
Sample	Sample	Øyebliksverdi av lydsignalet. Her digitale tallverdier.
Adressemapping	Address Mapping	Tilordning av adresser til de enheter som deler en felles buss
IC	IC	Forkortelse for Integrated Circuit (en <i>chip</i> ) Det vil si en mikrokrets som er laget i et stykke silisium
RAM	RAM	Forkortelse for Random Access Memory Dette er et lager som: <ul style="list-style-type: none"> <li>• Kan skrives til</li> <li>• Kan leses fra</li> <li>• Mister innholdet ved strømbrudd</li> </ul>
Kodek (Codec)	Codec	Forkortelse for Koder Dekoder. Den oversetter signalet fra mikrofonen til et digitalt signal, og systemets digitale signal til et analogt signal som kan settes ut på høyttalerne. Dette gjøres ved hjelp av en AD- og en DA-omformer.
Buss	Bus	En samling elektriske ledere som utgjør en gruppe signaler
Nedlasting	Downloading	Overføring av et design fra PC-en til Altera-kretsen Overføring av et mikrokontrollerprogram fra PC-en til µC.
DSP	DSP	Forkortelse for Digital Signal Processing/Digital Signal Processor På norsk: digital signalbehandling/digital signalprosessor
Mux	Mux	Forkortelse for Multiplexer Det vil si en velger, som velger ut ett av flere inngangssignalene
Nettliste	Netlist	Liste over fysiske forbindelser mellom komponentene i et design. Er vanligvis en ren tekstfil
Syntese	Synthesis	Den prosessen som går ut på å omforme en (formell) spesifikasjon til en nettliste
LUT	LUT	Look-Up-Table: Oppslagstabell

## **CODEC**

Det er ikke nødvendig å vite alt hva CODEC-en gjør. Men dette kan være greit å lese igjennom for å få en oversikt.

CODEC-ens inn-del utfører disse operasjonene:

- Forsterking av (det svake) inngangssignalet fra lydinngangen.
- Digitalisering (sampling) av det analoge signalet 44100 ganger i sekundet ved hjelp av en analog-til-digital omformer (AD).
- Kvantisering av det samplede signalet til et av 65536 amplitudenivåer.
- Koding av de 65536 amplitudenivåene. Dette gjøres ved å oversette dem til tilsvarende 16 bits toerkomplements binære tall (som kan anta alle heltallsverdier fra -32768 til 32767).

CODEC-ens ut-del utfører disse operasjonene:

- Omforming av signalet fra digitalt til hakkete analogt (DA-konvertering), som har 65536 mulige amplitudenivåer.
- Lavpassfiltrering («glatting») av det hakkete analoge signalet.
- Forsterking av det glattede analoge signalet slik at det kan drive en høyttaler eller hodetelefon (det vil si at CODEC-en har en forholdsvis lavohmig utgang).



# Introduksjon

## Oversikt

På labben bruker vi kretskortet DE2 fra Altera. Dette er et utviklingskort med en rekke ulike funksjoner. Det benyttes i industrien til å designe og teste ut nye systemer inntil man har fått laget mer spesialiserte kretskort. Slike kort brukes også mye i utdannings- og undervisningssammenheng. Den viktigste komponenten på kretskortet er en Cyclone II FPGA med navn EP2C35. Det er inne i denne kretsen at vi kan legge våre digitale design. Rundt denne kretsen finnes det i tillegg en rekke andre moduler og komponenter som gjør at vi med dette kortet kan bygge opp et komplett system. Av dette skal vi i denne labben blant annet bruke

- lysdioder
- trykknapper
- brytere
- sjusegmentdisplay
- LCD-skjerm
- mikrofon- og lydinngang
- lydutgang

Inne i FPGA-en vil det dessuten i denne labben være en Nios II mikroprosessor. Dette er en såkalt myk mikroprosessor. Det vil si at den ikke eksisterer som noen egen krets, men at den legges inn som en del av FPGA-en hver gang man laster ned et nytt design.

Når man skal lage et digitalt design som skal kjøres på utviklingskortet, for eksempel vårt design for lydbearbeiding, så trenger vi en PC. PC-en brukes til å:

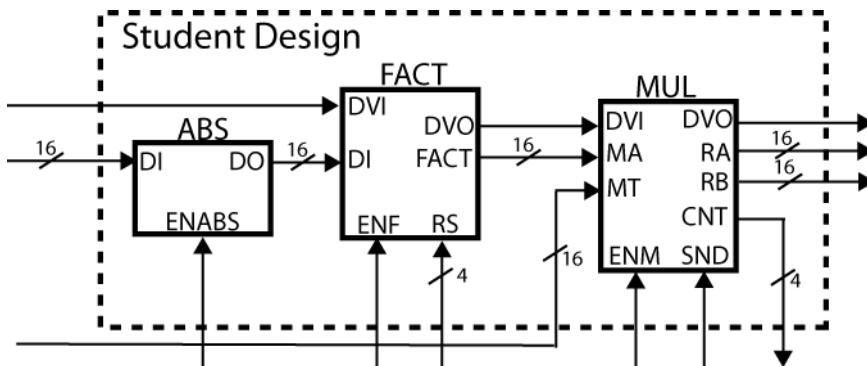
- *Konstruere* digitale kretser.
- *Simulere* kretsene.
- *Laste ned* kretsenes funksjonalitet i FPGA-en.
- *Skrive* mikroprosessorprogrammer.
- *Simulere* mikroprosessorprogrammene.
- *Laste ned* mikroprosessorprogrammene til mikroprosessorens RAM.

Øverst på innsiden av omslaget vises den *logiske* oppbygningen av vårt lydbehandlingsdesign. PC-en er ikke med på skissen, da den ikke har noen funksjon i systemets *logiske* struktur. Utenfor FPGA-rammen er utviklingskortet. Kun de enheter av utviklingskortet som skal brukes i dette faget er tegnet inn. Inne i FPGA-rammen er det en egen ramme rundt det studentdesignet dere skal jobbe med. I tillegg er det tegnet inn moduler som angir grensesnittet mellom utviklingskortet og studentdesignet. Dette grensesnittet gjør det meget enkelt å teste ut modulene som skal designes. Design som lages kan testes vha. grensesnittet, både i testmodus og i lydmodus.

Signalbehandlingssystemet er «rammeverket» for lab-øvingene. Igjennom labben skal det komplette signalbehandlingssystemet bygges opp:

Modul	Betydning	Fysisk plassering	LAB
ABS	Tar absoluttverdien av signalet (likeretter)	Inne i FPGA-en	1 og 2
FACT	Finner ønsket forsterkning av inngangssignal	Inne i FPGA-en	3
MUL	Multipliserer to signaler med hverandre	Inne i FPGA-en	4

Tabell 0-1: Oversikt over signalbehandlingssystem



Figur 0-1: Signalbehandlingssystem

Signalbehandlingssystemet består av modulene ABS, FACT, og MUL (se figur 0-1). De forskjellige modulene har i tillegg til inn- og utganger for data, forskjellige styreinnganger. Disse blir forklart nærmere i kapitlene for de respektive labbene. Under følger en prinsipiell forklaring av virkemåten til signalbehandlingssystemet.

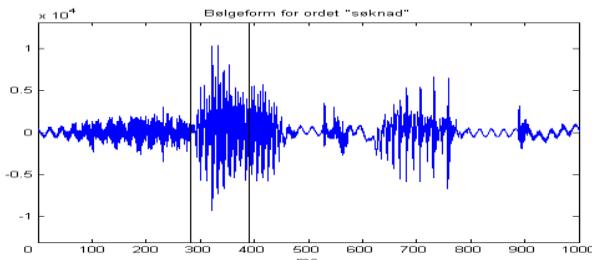
## Signalbehandlingssystemets virkemåte

Systemet som skal designes er en forenklet lydkompressor. En lydkompressor jevner ut forskjeller i lydstyrke på inngangssignalet, slik at utgangssignalet har mest mulig jevnt volum. Svake inngangssignaler forsterkes, mens sterke inngangssignaler dempes. Dette skjer ved først å detektere styrken på inngangssignalet (skjer i ABS og FACT). Dette benyttes så til å finne hvor mye signalet skal forsterkes eller dempes (skjer i FACT). Til slutt styrer modulen MULT volumet på utgangssignalet ved å multiplisere inngangssignalet med den ønskede forsterkningen/dempingen.

En lydkompressor benyttes i mange sammenhenger (på godt og vondt). Når reklamene på TV høres ut som om de har dobbelt så høyt volum som programmene så skyldes det at lyden på disse er kjørt gjennom en lydkompressor. Alt blir dermed en jevn gjennomtrengende høy lyd. Underveis i programmene er det derimot viktig å kunne la volumet variere, så her benyttes ikke lydkompressoren. En mer nytlig anvendelse kan være tilpasning av lyden i mikrofonen til forelesere på NTNU. Denne kan typisk være høy når foreleser har munnen vendt mot mikrofonen, mens den er lav når hodet er vendt den andre veien. Med en lydkompressor vil lyden ut til studentene bli passe høy hele tiden. Ytterligere detaljer om lydkompressorer finner dere på følgende Wikipedia-artikkel: [http://en.wikipedia.org/wiki/Audio\\_compressor](http://en.wikipedia.org/wiki/Audio_compressor)

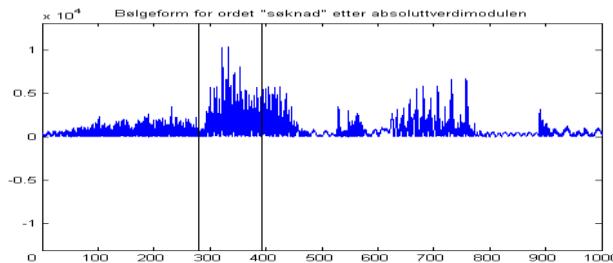
Vår lydkompressor er som nevnt en del forenklet i forhold til kommersielle produkter. En del av løsningene som er valgt er kanskje heller ikke de enkleste og beste for et virkelig system. Dette er gjort for å lage en pedagogisk fornuftig oppgave med passe kompleksitet. Vi tror likevel at systemet på en forhåpentligvis god måte vil demonstrere hvordan et slikt system kan bygges opp.

Hva er det så som skjer inne i de tre modulene ABS, FACT og MUL? La oss se på et eksempel på et lydsignal slik det er vist i *Figur 0-2*. Her ser vi at volumet varierer kraftig (bestemmes av amplituden på signalet). *Figur 0-2* gjengir egentlig en kort lydsekvens, mens en lydkompressor typisk fungerer over lengre sekvenser. Vi kan imidlertid illustrere forklaringen nedenfor godt med eksemplet i *Figur 0-2*.



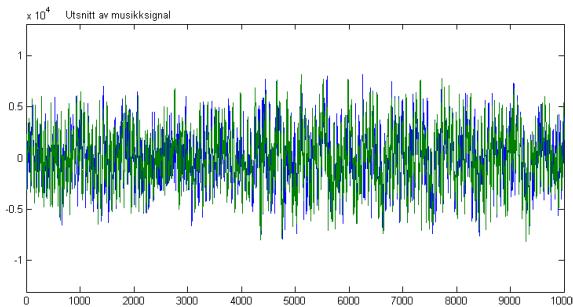
*Figur 0-2: Lydsignal på inngangen*

Lydsignalet i *Figur 0-2* varierer rundt null. Dette er uheldig for vår løsning for deteksjon av signalstyrke, og det første vi vil gjøre er derfor å sørge for at signalet bare har positive verdier. Dette gjør vi ved å ta absoluttverdien av inngangssignalet i modulen ABS. Resultatet av denne operasjonen sees på *Figur 0-3*.



*Figur 0-3: Absoluttverdi av lydsignal*

Inne i modulen FACT detekteres signalstyrken på de ulike delene av signalet og det beregnes hvilken forsterkning/dempning som skal til for å få et jevnt signal ut. Detaljene rundt dette vil bli presentert i oppgaveteksten for lab 3. I modulen MULT blir så det opprinnelige inngangssignalet multiplisert med den verdien som ble funnet i FACT. Ideelt sett skulle utgangssignalet fra MULT se ut som i *Figur 0-4*. I vårt forenklede system blir det nok ikke riktig så pent, men det kommer vi tilbake til i oppgaveteksten for lab 4.



*Figur 0-4: (Ideelt) lydsignal på utgangen*

Modulene CODEC IN, CODEC OUT og 7-SEG med sine INTERFACE, samt CLK GEN & DISTR (se omslaget) skal dere ikke jobbe direkte med i labbene. Dere bare henter signaler fra og/eller sender signaler til dem. CODEC IN omformer et analogt lydsignal (enten fra LINE IN eller MIC) til digitalt signal, mens CODEC OUT omformer et digitalt signal til analogt høyttalersignal (LINE OUT). Internt i designet benyttes mono digitalt signal, mens LINE IN og LINE OUT er stereosignal. Omformingen mellom mono og stereo skjer i CODEC INTERFACE. CODEC INTERFACE genererer også nødvendige konfigureringsignal for CODEC (for eksempel å velge mellom LINE IN og MIC som analog lydkilde). CLK GEN & DISTR genererer og distribuerer ulike klokkesignaler i systemet. Dette gjelder blant annet en 11,29MHz masterklokke og et 44,1KHz klokkesignal som brukes av CODEC.

## Designverktøy

For å håndtere dagens store og komplekse designoppgaver er man nødt til å ta i bruk designverktøy. Verktøyet som benyttes i denne labben er Quartus II, et PC-basert designverktøy fra Altera, spesielt laget med tanke på Alteras egen familie med FPGA-brikker (slik som Cyclone II EP2C35 som finnes på utviklingskortet).

Designverktøy i dag er mer enn avanserte tegneprogrammer. De kan blant annet simulere kretsens virkemåte, uten at vi trenger å koble den opp fysisk. Verktøyene kan også foreta syntese. Det vil si at vi skriver en formalisert spesifikasjon, og lar verktøyet «oversette» dette til en nettliste (vi må fortelle verktøyet hvilke komponenter det får lov til å bruke). En nettliste er en tekstfil som inneholder komponentene kretsen består av samt sammenkoblingen mellom disse.

Designverktøyets «input» er ikke bare skjemategninger. Man kan også bruke maskinvarebeskrivende språk (Hardware Description Language – HDL). Eksempler på HDL er VHDL og Verilog. Quartus II håndterer begge disse språkene. En HDL-utskrift «ser ut som» et vanlig dataprogram, men det beskriver maskinvare. Skal man mestre HDL må man «tenke maskinvare», og være klar over at det er forskjellig fra programmering. VHDL inngår i lab 4.

## Designmetodikk

Grunnregelene i god designmetodikk er:

- Skill spesifikasjon og implementering
- Del opp i moduler

Det er altså viktig å skille spesifikasjonsarbeidet fra implementeringen. Selv om det kan være fristende å starte med å implementere og dermed komme i gang, er det vanskelig å få oversikt og implementere en korrekt løsning uten først å ha gjennomført spesifikasjonen.

Videre er det helt nødvendig å tenke hierarkisk og forstå oppdelingen i moduler for å klare å ha oversikt over større digitale systemer.

## Lab-arbeidsmåte

### Selvstendighet og egen hjelp

Labben i dette faget er krevende, og det er begrenset mengde med stud.asser tilgjengelig. Labheftet inneholder heller ikke *alle* detaljer, og noen skjermbilder kan avvike noe fra det hver gruppe får på sin egen skjerm.

Det er derfor absolutt nødvendig å være godt forberedt før oppmøte på labben. Likeledes gjelder det å være aktiv selv med å løse problemer som måtte dukke opp, både gjennom å benytte hjelpefunksjonene i Quartus II, og ved å tilstrebe en forståelse av det som gjøres i labben, i motsetning til å følge opplegget slavisk uten å tenke så mye på hva man gjør. Å ta seg god tid til forståelse i begynnelsen betaler seg raskt tilbake senere i semesteret.

Skulle det er være vanskelig å få fatt i en stud. ass., kan det være andre studenter som er villige til å hjelpe dersom man står fast.

### Forarbeid

Før man møter på labben *må* man ha gjennomført et forarbeid. Dette varierer fra lab til lab, men går i store trekk ut på å studere teori, sette seg inn i virkemåten til en eller flere kretser, tegne en eller flere kretser, og foreta utregninger. Dette gjøres både for å øve god designmetodikk, og for å kunne gjennomføre labben på en ryddig og god måte. Om man møter uforberedt på labben, vil man *kanskje* få modulen til å virke til slutt, etter mye prøving og feiling, og etter mange henvendelser til stud.ass (men ikke innenfor den tilmalte tid). **Alle utregninger og skjemategninger skal føres inn i en labprotokoll.**

Før man møter på labben bør man ha kjørt en «skrivebordtest» av de kretser man har tegnet. Det vil si å kjøre en «manuell simulering». Man går kritisk gjennom designet og prøver å overbevise seg selv om at alt er riktig, slik at sannsynligheten for overraskelser på et senere tidspunkt reduseres til et minimum.

Man skal *implementere* og *teste* designet. Det vil si å tegne kretsen inn i en skjemaeditor, simulere, laste ned i FPGA-en, og teste. Når man tester, må man huske at det er spesifikasjonen som er fasiten. Derfor:

- **Bring modulens oppførsel i samsvar med de spesifikasjonene dere har bestemt.**
- ***Ikke* bring spesifikasjonen i samsvar med hvordan dere ser at modulen oppfører seg.**

## **Blackboard**

Tilleggsinformasjon om labbene vil være tilgjengelig på kursets side på Blackboard.

# 1 Laboratorieøving:

Hurray, there is no lab 1 😊

## 2 Laboratorieøving: Absoluttverdikrets 16bit - FPGA

### ***Innledning***

#### **Formål**

I denne labben skal det lages en absoluttverdi krets i designverktøyet Quartus II. Absoluttverdikretsen blir altså studentdesigndelen av FPGA-en. Resten av innholdet i FPGA-en er allerede laget, og vil fungere som et grensesnitt mot funksjonaliteten på utviklingskortet DE2. Absoluttverdikretsens utgang skal således vises på sjusegment displayet på DE2-kortet via dette grensesnittet.

#### **Hensikt**

- Få kjennskap til utviklingskortet DE2 og FPGA-en.
- Få kjennskap til (og øve bruk av) FPGA-ens grensesnitt mot DE2-kortet.
- Få kjennskap til (og øve bruk av) Alteras designverktøy Quartus II.
- Øve design av kombinatoriske kretser ved hjelp av modulen ABS.
- Øve god designmetodikk
- Danne basis for lab 3 og 4
- Bli selvhjulpen på labben.

I de videre lablene vil det bli henvist til lab 2 når det gjelder bruk av Quartus II.

#### **Bakgrunn**

- Teori om designprosessen og om kombinatoriske kretser finnes i: Daniel Gajski: «Principles of Digital Design»), kapittel 1 til 5
- Bruk help-funksjonen i Quartus II

#### **Hva som skal gjøres**

- Som forarbeid skal dere arbeide med teori, være på lab-forelesningen om lab 2, og lese lab-infoen på fagets side på Blackboard.
- På labben skal en gå gjennom en veiledning punkt for punkt for å bygge opp kjennskap til Quartus II og utviklingskortet.
- Det skal designes en kombinatorisk krets som finner absoluttverdien til et tall. Designprosessen skal utføres i flere trinn.

#### **Merk**

Lab 2 er omfattende for å opparbeide selvstendighet på labben og gjøre det enklere å gjennomføre de øvrige lablene.

## Teori

### Negative tall og absoluttverdi

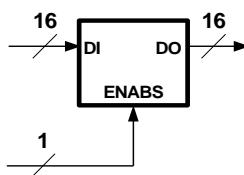
Matematisk er det enkelt å ta absoluttverdien av et tall. Med vanlig tallnotasjon setter man bare fortegnet til tallet til pluss, uavhengig av hva det var fra før. En slik metode kan også benyttes dersom tallet er på fortegn-tallverdi form. For tall på toerkomplement form blir operasjonen mer komplisert. Vi vet at å bytte fortegnet til et tall skrevet med toerkomplement kan gjøres med først invertere alle bitene i tallet og deretter legge til en. Hvis vi tar en krets som gjør dette har vi nesten en absoluttverdikrets for tall representert på toerkomplement form. Den vil gi rett svar hvis verdien på inngangen er negativ, men er den positiv vil resultatet på utgangen være en negativ verdi. For å løse dette må positive tall detekteres og slippes uforandret igjennom. Å avgjøre om et tall er positivt er en enkel operasjon. Som i fortegn-tallverdi form angir den mest signifikante biten om tallet er positivt eller negativt. Denne biten kalles fortegnsbit:

- 0 -> tallet er positivt
- 1 -> tallet er negativt

Altså, er fortegnsbiten null må kretsen deaktivertes og slippe tallet uforandret igjennom. Kretsen må derfor kunne deaktivertes og aktiveres ved behov. Ved å legge til en aktiv høy styreinngang for denne ser vi at fortegnsbiten kan benyttes direkte til å styre toerkomplement kretsen. Ved å hardkoble styreinngangen til fortegnsbiten på inngangen vil resultatet bli en absoluttverdikrets.

### Grensesnitt

Kretsen som er målet i denne labben har følgende grensesnitt. ENABS bestemmer om absoluttverdikretsen skal virke eller ikke. Det vil si om den skal ta absoluttverdien av inngangstallet, eller bare slippe det igjennom.



Figur 2-1: ABS-modul

Tabell 2-1: ABS-modul grensesnitt

Navn	Antall bit	Retning	Verdi	Betydning
DI	16	Inn		Data
DO	16	Ut		Data
ENABS	1	Inn		Kontrollbit: $DO = DI$
			0	Dersom signbit = 1: $DO = (\sim DI) + I$
			1	Dersom signbit = 0: $DO = DI$

$\sim DI$  betyr at alle bitene i DI inverteres ( $\sim$  er en «tilde»)

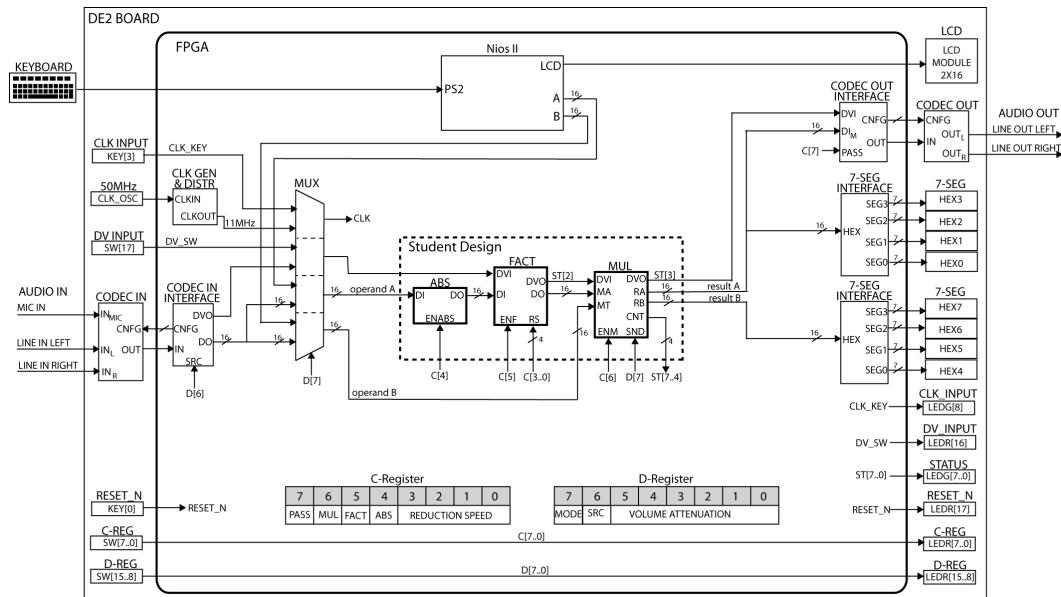
## **Forarbeid**

# Grunnlag

- Les vedlegget om grunnleggende funksjonalitet i Quarts.
  - Gjør deg kjent med hvilke prosedyrer som er beskrevet i avsnittet om «Ofte brukte prosedyrer i Quartus»
  - Les lab-infoen på Blackboard, og møt opp på lab-forelesningen om lab 2.
  - Repeter introduksjonskapittelet.
  - Les hele lab 2
  - Få oversikt over logisk og fysisk skisse på innsiden av omslaget.

## **Introduksjon til FPGA-designet.**

Her skal grensesnittet rundt studentdesignet studeres. Dette grensesnittet er en ferdigtegnet del av FPGA-en som inngår i alle labbene. Den utgjør koblingen mellom deres egen krets og I/O-enhetene (tastaturet, brytere, LCD-display, sjusegmentdisplay og lysdioder) på DE2-kortet. Når man skal koble en ledning fra en krets til en lysdiode, gjøres det ved å koble ledningen til et av bitene på STATUS-bussen (se figur 2-2). De 8 bitene på STATUS-bussen er koblet til hver sin grønne lysdiode på DE2-kortet. A og B registrene gis verdi ved hjelp av tastaturet som er tilkoblet DE2-kortet. Verdien til C og D registrene settes opp ved å stille inn et sett skyvebrytere. Gjeldende verdi for den enkelte skyvebryter vises på røde lysdioder. DVI kommer også fra en skyvebryter, mens CLK kommer fra en trykkbryter. Verdien på signalene Result\_A og Result\_B vises på DE2-kortets sjusegmentdisplay (henholdsvis HEX 0 til 3 og HEX 4 til 7).



*Figur 2-2: Logisk skisse av totalsystemet*

Enhetens fulle navn	Antal i bit	Settes av	Vises på DE2-kort	Betydning
A	16	Tastaturet	LCD	Påtrykk av data til studentdesign
B	16	Tastaturet	LCD	Påtrykk av data til studentdesign
C	8	SW[7..0]	LEDR[7..0]	Kontrollpåtrykk til studentdesign
D	8	SW[15..8]	LEDR[15..8]	Kontrollpåtrykk til studentdesign og grensesnitt
DV INPUT	1	SW[17]	LEDR[17]	Data Valid til studentdesign
RESET_N	1	KEY[0]	LEDR[17]	Reset av studentdesign og Nios II (aktivt lavt)
CLK	1	KEY[3]	LEDG[8]	Klokke til studentdesign
RESULT_A	16	Studentdesign	Sjusegment / Lydkanal	Data fra studentdesign
RESULT_B	16	Studentdesign	Lydkanal	Data fra studentdesign
STATUS	8	Studentdesign	LEDG[7..0]	Statussignaler fra studentdesign

Tabell 2-2: Oversikt over signaler i FPGA

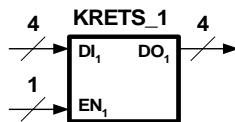
SW[xx] er skyvebrytere, KEY[xx] er trykksensorer, LEDR[xx] er røde lysdioder og LEDG[xx] er grønne lysdioder. Se kapittel 7 Oppkobling og bruk - Altera DE2 for mer info.

Figur 2-2 viser det komplette systemet i detalj. Studer beskrivelsen, og pass på at dere forstår grensesnittet mellom din egen krets inne i FPGA-en og I/O-enhetene på DE2-kortet.

Figuren viser også hvilke deler av systemet som befinner seg inne i FPGA-en, hvilke som sitter på DE2-kortet utenfor FPGA-en, og hva som er utenfor DE2-kortet igjen. Tabell 2-2 gir en oversikt over betydningen til signalene inne i FPGA-en.

## Kretsdesign

For å lage en absoluttverdikrets må vi altså designe en krets som beregner toerkomplement. Å ta toerkomplement er som beskrevet tidligere en todelt operasjon og designet kan derfor lett deles inn i to kretser. KRETS\_1 som spesifisert i figur 2-3 og tabell 2-3 inverterer bitene. Merk at dette krets inverterer alle bitene dersom EN<sub>1</sub> er satt. Mens KRETS\_2 som spesifisert i figur 2-3 og tabell 2-3 adderer 1. Merk at dette krets adderer 1 dersom EN<sub>2</sub> er satt.



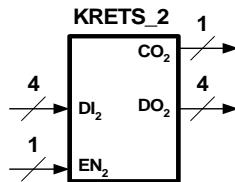
**Figur 2-3: KRETS\_1**

Navn	Antall bit	Retning	Funksjon
DI <sub>1</sub>	4	Inn	Data-inn
DO <sub>1</sub>	4	Ut	Data-ut
EN <sub>1</sub>	1	Inn	Kontrollbit: 0 $DO_1 = DI_1$ 1 $DO_1 = \sim DI_1$

~DI betyr at alle bitene i DI inverteres (~ er en «tilde»)

**Tabell 2-3: KRETS\_1 grensesnitt**

*Tips:* Studer sannhetstabellen til en XOR-port.



Figur 2-4: KRETS\_2

Tabell 2-4: KRETS\_2 grensesnitt

Navn	Antall bit	Retning	Verdi	Betydning
DI <sub>2</sub>	4	Inn		Data
DO <sub>2</sub>	4	Ut		Data
EN <sub>2</sub>	1	Inn	0	Kontrollbit: $DO_2 = DI_2$
			1	$DO_2 = DI_2 + 1$
CO <sub>2</sub>	1	Ut		Carrybit: DI <sub>2</sub> = 1111 og EN <sub>2</sub> = 1: CO <sub>2</sub> = 1 Ellers: CO <sub>2</sub> = 0

Dette er en krets som legger til 1 dersom EN<sub>2</sub> er satt. Merk at siden dette er en adderer, vil CO<sub>2</sub> bli 1 om DI<sub>2</sub> = 1111 og EN<sub>2</sub> =:

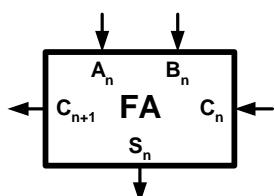
$$\begin{array}{r}
 & 1111 \\
 & + 1 \\
 \hline
 = & 10000
 \end{array}$$

Det vil si at CO<sub>2</sub> = 1 og DO<sub>2</sub> = 0000

*Tips:* Studer sannhetstabellen til en adderer. Se spesielt på Carry In. Det skal benyttes kun porter, ikke ferdige komponenter (som f.eks. en adderer).

### Ripple Carry adderer

La oss se litt på teorien bak addisjonen som skal utføres av KRETS\_2. Som beskrevet i Gajski, kapittel 5.1, blir en adderer realisert ved å koble sammen flere like blokker som hver har en enklere funksjonalitet. Disse blokkene kalles fulladderere. Det er en fulladderer for hvert av bitene i inngangstallene som skal adderes. Hver fulladderer behandler et bit fra hvert av inngangstallene A og B, med samme signifikansnivå, og mentebiten fra signifikansnivået under. Av disse inngangene beregnes sum og mente (carry).



a) Fulladderer symbol

$A_n$	$B_n$	$C_n$	$S_n$	$C_{n+1}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

b) Funksjonstabell

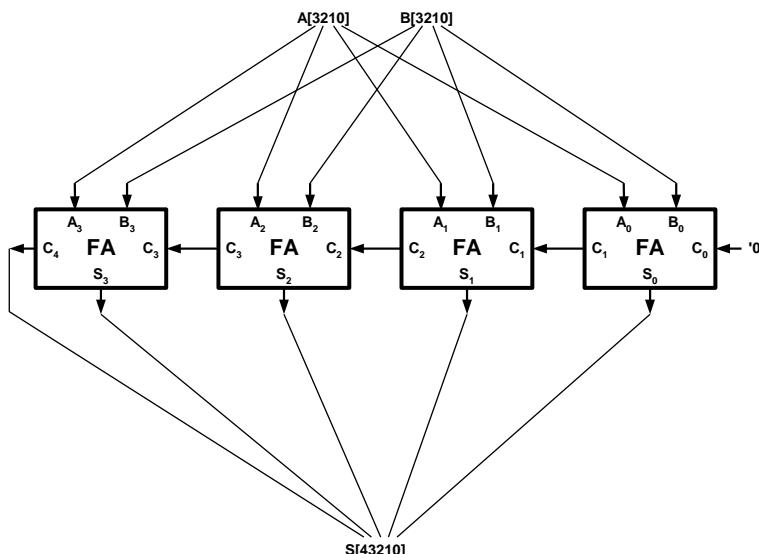
$$S_n = A_n \oplus B_n \oplus C_n$$

c) Sum-bit, funksjon

$$C_{n+1} = A_n \cdot B_n + A_n \cdot C_n + B_n \cdot C_n$$

d) Mentebit, funksjon

Figur 2-5: En-bits fulladderer



Figur 2-6: Fire-bits ripple carry adderer

### Modifisert Ripple Carry adderer

Adderer som benyttes i labben er litt spesiell, og trenger ikke samme funksjonalitet som en generell adderer. Det som er spesielt med labben er at det ene tallet (B) alltid er 1 (0001<sub>BIN</sub>). Det medfører at noen endringer og forenklinger kan gjøres (se Figur 2-7).

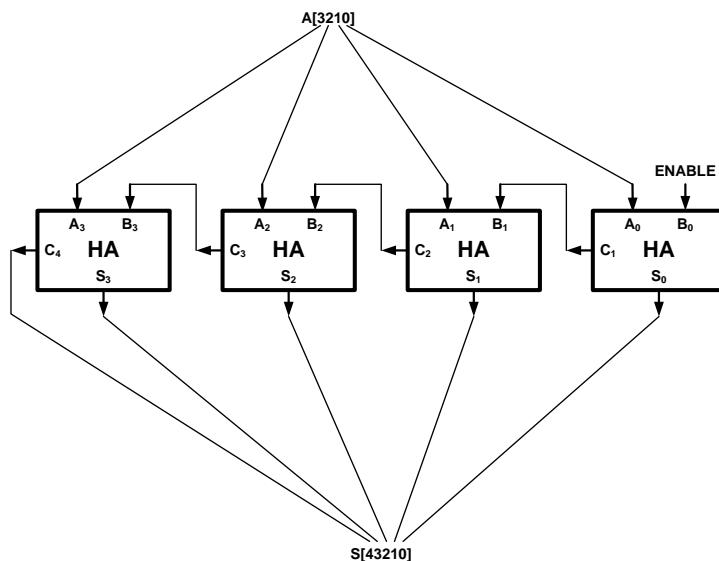
- De tre mest signifikante bit i B, som alltid er "0", kan fjernes fra funksjonen, uten at resultatet blir endret.
- C-inngangen for den minst signifikante fulladdereren er alltid "0". Dette bidrar heller ikke, så den kan også fjernes.

Det vi sitter igjen med etter disse forenklingene er fire fulladdererkretser med en av inngangene fjernet. Siden alle inngangene på en adderer er likeverdige (Se Figur 2-5 c) og d)**Error! Reference source not found.**, kan vi fjerne C-inngangen i stedet for B-inngangen. En fulladderer uten c-inngang kalles halvadderer.

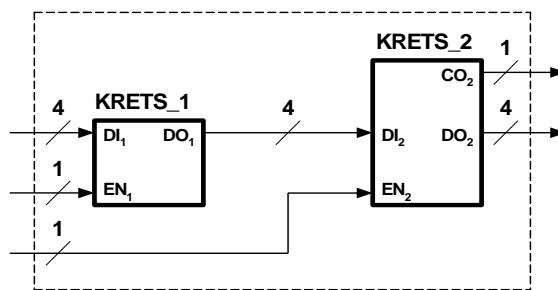
En annen spesiell ting med adderereren i denne labben er at den må ha en inngang som styrer om kretsen skal slippe tallet, A, uforandret igjennom eller ikke. Vi kan gjøre følgende betrakting; å slippe tallet uforandret gjennom er det samme som å legge til null. Ved å kalle styreinngangen «ENABLE» og sette den til aktiv høy så vil følgende gi ønsket funksjonalitet:

$$S = A + \text{ENABLE}$$

Er «ENABLE» null så legger den til null, med andre ord slipper A uforandret igjennom. Er «ENABLE» en så legger den til en. Styresignalet kan dermed legges inn på den minst signifikante biten til B, og resultatet er kretsen vist i Figur 2-7.



**Figur 2-7:** Modifisert Ripple Carry adderer



**Figur 2-8:** KRETS\_12 (KRETS\_1 og KRETS\_2 i serie)

Dersom KRETA\_1 (Figur 2-3) og KRETS\_2 (Figur 2-4) kobles sammen som vist i figuren, fåes følgende funksjonalitet:

Tabell 2-5: KRETS\_12 grensesnitt

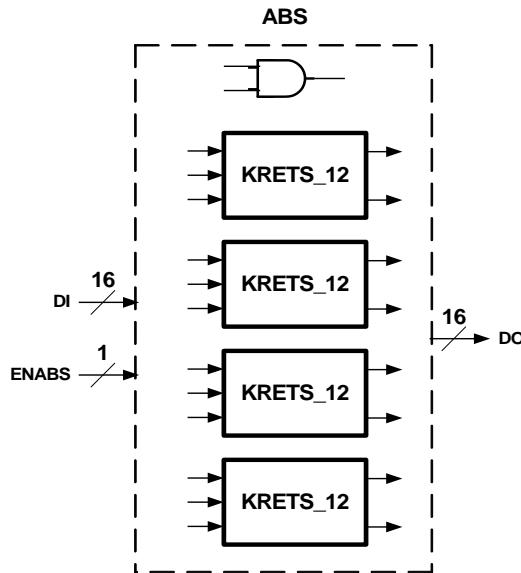
Navn	Antall bit	Retning	Betydning
DI <sub>1</sub>	4	Inn	Data
DO <sub>2</sub>	4	Ut	Data
EN <sub>1</sub>	1	Inn	Kontrollbit:
EN <sub>2</sub>	1	Inn	EN <sub>1</sub> = 0 og EN <sub>2</sub> = 0: DO <sub>2</sub> = DI <sub>1</sub> EN <sub>1</sub> = 0 og EN <sub>2</sub> = 1: DO <sub>2</sub> = DI <sub>1</sub> + 1 EN <sub>1</sub> = 1 og EN <sub>2</sub> = 0: DO <sub>2</sub> = ~DI <sub>1</sub> EN <sub>1</sub> = 1 og EN <sub>2</sub> = 1: DO <sub>2</sub> = (~DI <sub>1</sub> ) + 1
CO <sub>2</sub>	1	Ut	Carrybit: DI <sub>2</sub> = 1111 og EN <sub>2</sub> = 1: CO <sub>2</sub> = 1 Ellers: CO <sub>2</sub> = 0

Merk at om ingen av ENene er satt, vil KRETS\_12 slippe DI<sub>1</sub> rett gjennom. Om begge ENene er satt, vil KRETS\_12 gi toerkomplementet.

- ☒ Design KRETS\_12 ved å kombinere innholdet fra KRETS\_1 og KRETS\_2.
- ☒ Design hele ABS-kretsen.  
*Tips:* Bruk 4 stk. KRETS\_12 og koble dem sammen.

ABS-kretsen skal bare gi ut to-komplement av inngangstallet dersom den er enablet *og* inngangstallet er negativt.

- ☒ Bruk en ekstra port (Figur 2-9) til å oppnå denne funksjonaliteten.



Figur 2-9: Moduler til bruk ved design av ABS

- ☒ Tabell 2-6 inneholder to testvektorer for forventede utgangssignaler for KRETS\_12, som følge av inngangssignalene. Fyll ut resten av tabellen med testvektorer, som dere mener vil påse at kretsen virker.

Tabell 2-6: Testplan for KRETS\_12

Tabell 2-7 inneholder to testvektorer og forventede DO-verdier til ABS-modulen.

- Fyll ut resten av tabellen.

0x... betyr at tallet er angitt i heksadesimalt tallsystem. Senere vil 0b... bli benyttet til å angi binære tall.

 Hva skjer når en prøver å finne to-komplement av 0x8000?

Tabell 2-7: Testplan for ABS-modul

## Laboratoriearbeid

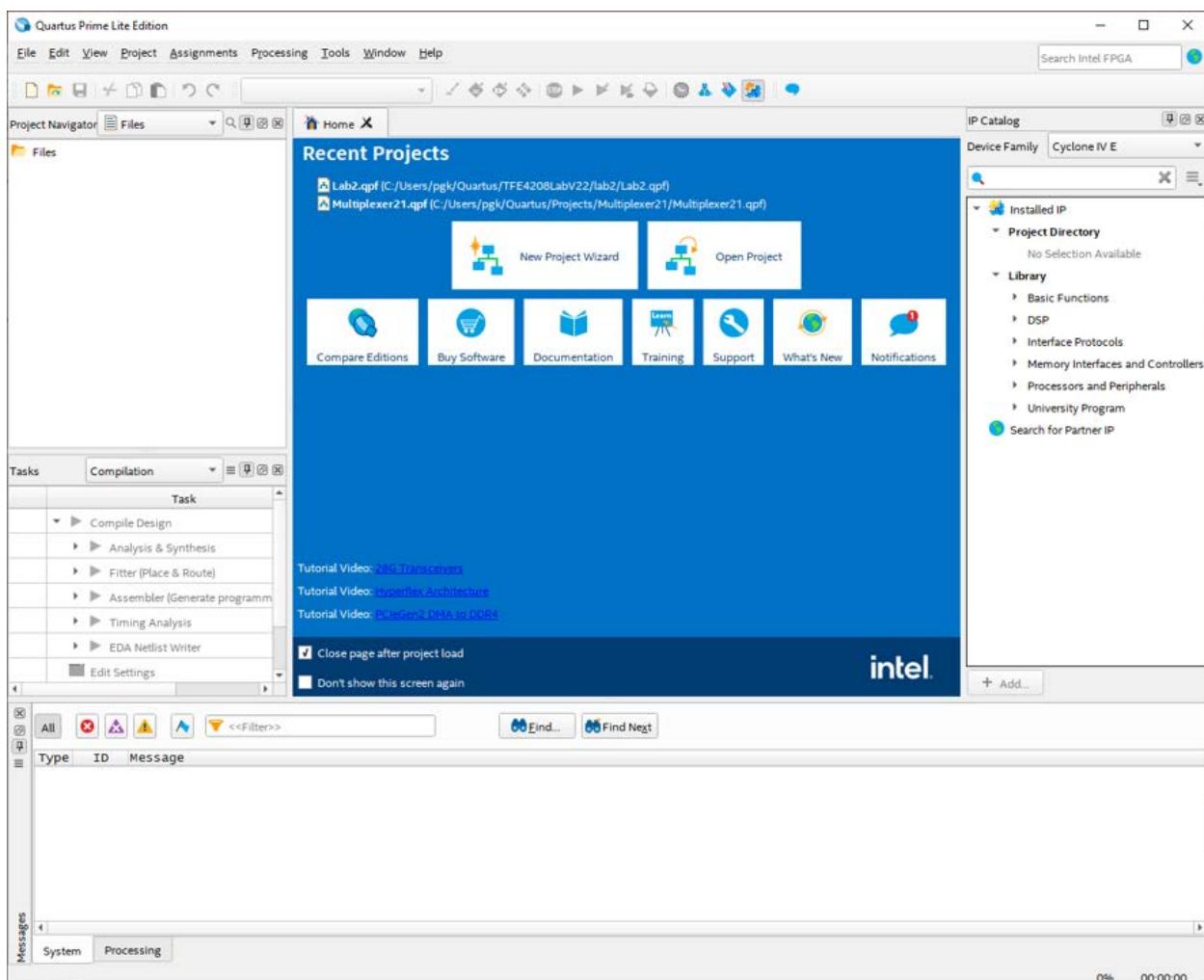
### Oversikt

Labarbeidet i lab 2 består av 6 oppgaver:

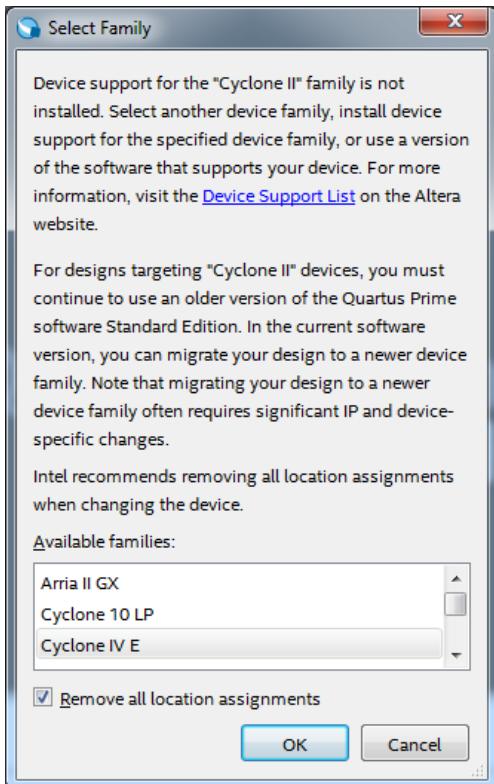
Oppgave	Innhold
1	Kopiering og åpning av prosjekt
2	Hjelp i Quartus II
3	Bruk av Schematic File i Block Editor (tegne inn KRETS_12)
4	Bruk av simulatoren (simulere KRETS_12 og ABS-kretsen)
5	Implementasjon og nedlasting av design til FPGA
6	Avslutning

Sjekk Blackboard for eventuelle nye beskjeder om labben.

### Laboppgave 1: Kopiering og åpning av prosjekt



Figur 2-10: Quartus åpningsvindu



Figur 2-11 Bytte til Cyclone IV E

### Kopier prosjekt til lokal maskin

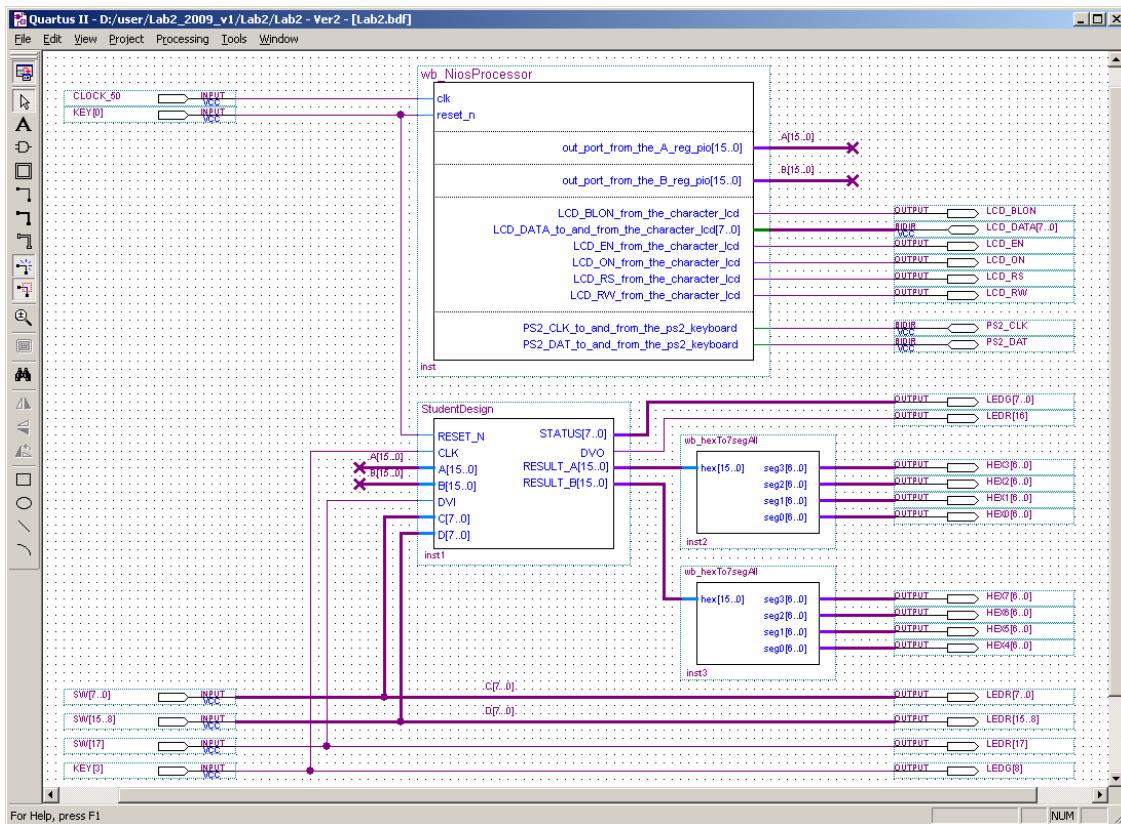
Nå skal et påbegynt prosjekt kopieres og åpnes. Dette prosjektet danner grunnlaget som det skal jobbes videre med (bl.a. ved at grensesnittet mot DE2-kortet er lagt klar for bruk).

Før en åpner prosjektet må en ha en plass å åpne det på. Opprett derfor en katalog (folder, directory).

Hent så prosjektet (filen LAB2.ZIP) fra fagets side på Blackboard. Lagre filen i den nye katalogen dere opprettet. Dobbeltklikk på filen. Programmet WinZip starter og dere kan ta *Extract* på filene slik at de blir liggende i den samme katalogen der dere står. Merk at dere **MÅ** utføre labarbeidet med filene liggende i denne katalogen. Dersom de ligger et annet sted på nettverket eller på en minnepinne, vil langsom aksessering av filene gjøre at programmet går svært tregt.

Prosjektet åpnes slik i Quartus II: Velg *File* → *Open Project* (**NB:** ikke *Open File*). Finn filen *Lab2.qpf* i katalogen *Lab2* under katalogen dere nettopp opprettet. Når dere har funnet den riktige filen, trykk *Open*. Første gang Quartus åpnes må krets endres til Cyclone IV E og «Remove all location assignments» klikkes (Figur 2-11).

For å se hovedskjemaet i designet velger dere fanen *Files* under *Project Navigator*, til venstre i arbeidsområdet i Quartus II. Dobbeltklikk på filen *Lab2.bdf*. Et skjema som viser det øverste hierarkiske nivået i designet vårt åpnes slik det er vist i Figur 2-12. Vi skal lære om hierarki senere. I skjemaet ser vi en rekke blokker (også kalt moduler og komponenter) med ulike navn. I tillegg ser vi inn- og utgangspinnene, som svarer til pinnene på FPGA-en og som kobler denne til DE2-kortet. Vi ser også et antall signaler som kobler pinnene og blokkene sammen. Nå i lab 2 er ikke alle delene av det komplette lydbehandlingssystemet med (figur 2-2). Resten vil komme i senere labber.

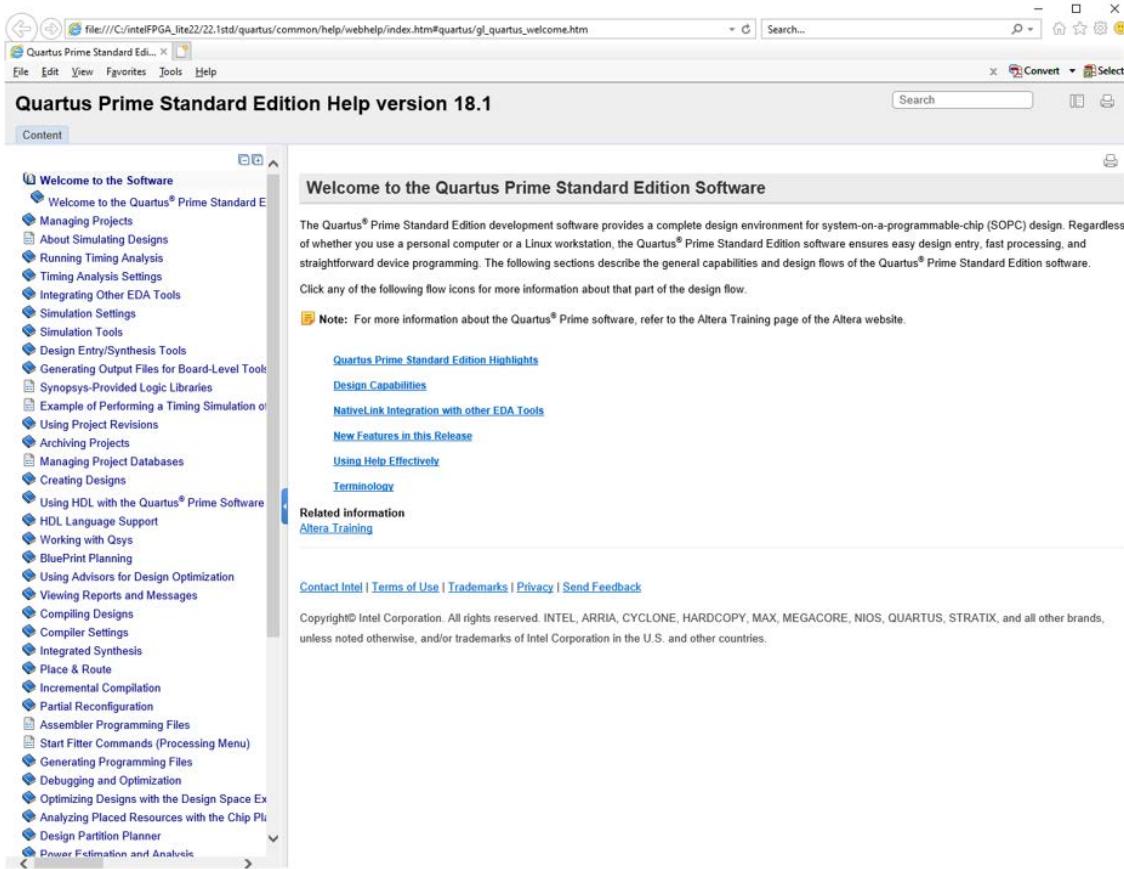


Figur 2-12: Skjemategning av toppnivået i FPGA-designet lab 2

## Laboppgave 2: Hjelp i Quartus II

I Quartus II finnes det en rekke hjelpefunksjoner under menyen *Help*. Hvis dere velger *Topics* her, så dukker vinduet i figur 2-13 opp. Her kan dere lese artikler listet i innholdsfortegnelsen, finne søkeord i indexen, eller foreta frisøk etter ord. Hvis dere vil kan dere også gå gjennom interaktive øvelser. *Help → PDF Tutorial* eller *Help → On the Web → Training*. For øvrig vil de funksjonene dere normalt vil trenge bli forklart her i labheftet.

- ☒ Bruk hjelpefunksjonene til å finne ut hva forkortelsen *bdf* står for.



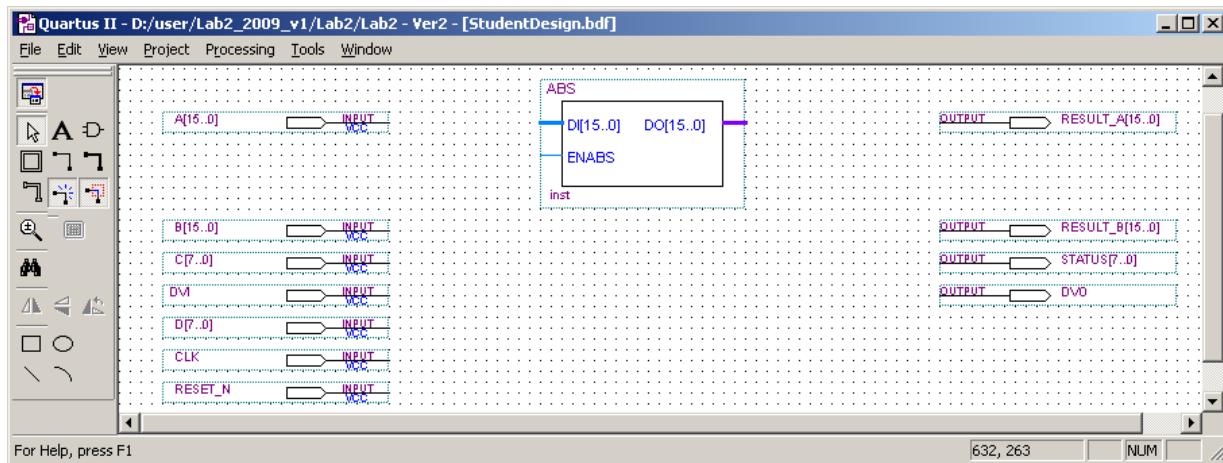
Figur 2-13: Quartus II Help Contents

### Laboppgave 3: Bruk av Schematic File i Block Editor (tegne inn KRETS\_12)

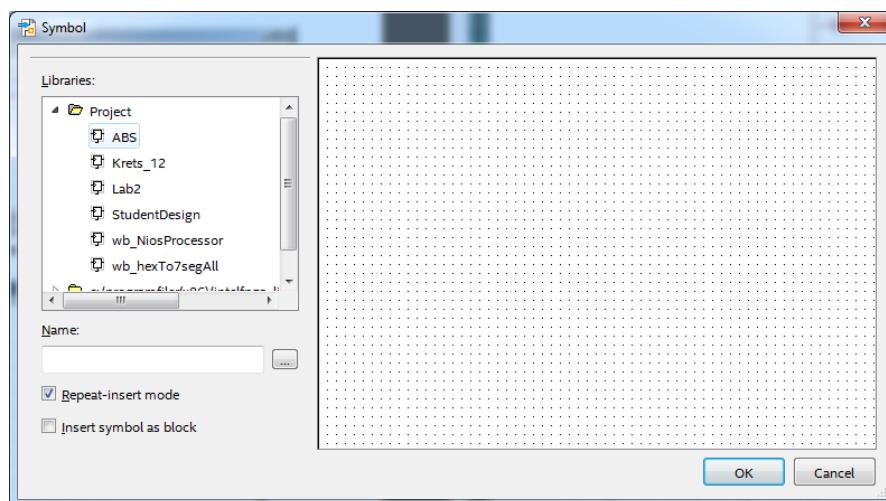
Quartus består av flere deler. Et av disse er *Block Editor* som blant annet kan benyttes til å tegne kretsskjema. Denne oppgaven skal gi en enkel innføring i hvordan dette kan gjøres.

Skjemaet *Lab2.bdf* som vises i figur Figur 2-12 er allerede tegnet i *Block Editor*. Her kan vi benytte et hierarki av blokker for å skjule detaljer på ulike nivå. Det som vises i *Lab2.bdf* er for eksempel i hovedsak bare grensesnittet mot DE2-kortet. Dette skal ikke dere gjøre endringer på. Deres del av totaldesignet skal befinne seg inne i blokken *StudentDesign*, midt på skjemaet. Dobbeltklikk på denne blokken for å gå ned et nivå i hierarkiet, det vil si inn i denne blokken. Et nytt skjema, *StudentDesign.bdf*, åpner seg. Dette ser nesten ut som figur 2-14. Det som mangler er modulen ABS, denne modulen må en legge til fra biblioteket. En åpner komponentbiblioteket ved å trykke på -knappen, eller velge *Insert → Symbol...* fra menyen dere får opp ved å trykke på høyre musknapp (hurtigmenyen). Dere får da opp et vindu som vist i figur 2-15. Under *Libraries* ligger det komponenter knyttet til dette prosjektet (under *Project*) og en rekke standardkomponenter som er innebygget i Quartus (under *c:/.../quartus/libraries*). Standardkomponentene skal vi komme tilbake til senere. Nå skal dere velge komponenten ABS fra biblioteket *Project*. Et bilde av komponenten kommer opp, og dere kan trykke *OK*. Flytt komponenten dit dere vil ha den i skjemaet, og trykk en gang på venstre mus-knapp for å plassere den.

**Overalt hvor det i teksten står ABS skal det stå ABSLT.**

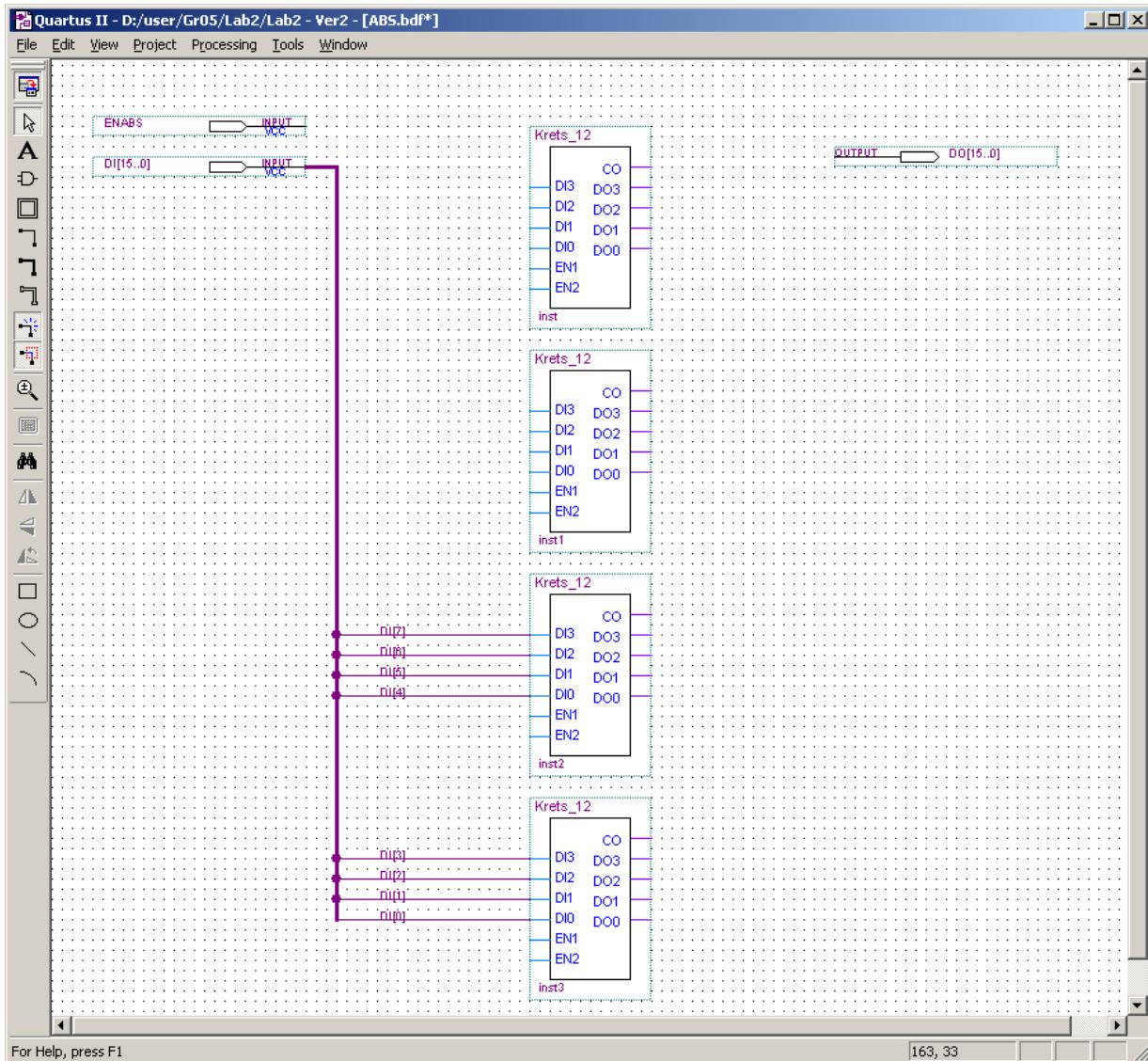


Figur 2-14: Skjemavindu med studentdesign og ABS-modul



Figur 2-15: Vindu for henting av biblioteksymboler

Nå som komponenten er på plass, kan vi se litt nærmere på den. Igjen skal vi bevege oss et trinn ned i hierarkiet. Dobbelttrykk på ABS-modulen. Nå skal dere få skjermbilde som vist i figur 2-16. Gå tilbake til StudentDesign ved å velge denne fanen øverst i arbeidsområdet til Quartus II.



Figur 2-16: Skjema for ABS-modul

ABS-modulen skal nå kobles til inn- og utgangspinner på komponenten StudentDesign. Dette gjøres på følgende måte:

Bruk buss til å koble sammen StudentDesigns inngang A med ABS-modulens inngang DI (direkte kobling). Se figur 2-14.

Velg busstegner (*Orthogonal Bus Tool*) med -knappen.

- Trykk på inngangspinnen A[15..0] og dra så musa bort til pinnen DI[15..0] på ABS-modulen. En Buss skal da være tegnet mellom de to pinnene.
- For å gjøre det enklere ved simulering setter en navn på bussen. Hent fram hurtigmenyen ved å høyreklikke mens dere peker på bussen, og velg *Properties*. Skriv inn et navn i *Name* feltet (f.eks. DI[15..0]) og trykk på *OK*.

Nå er inngangspinnen A koblet sammen med inngangen DI, og bussen som kobler dem samme har et navn som kan brukes under simuleringen senere.

Man kan også benytte navn på signaler til å koble disse sammen (indirekte kobling). Vi skal nå bruke dette til å koble sammen utgangspinnen RESULT\_A[15..0] med ABS-modulens utgang DO[15..0]:

- Velg busstegner.
- Trykk på DO[15..0] pinnen og dra musa et lite stykke ut fra ABS modulen.
- Slipp museknappen.
- Gi bussen ut fra DO[15..0] et navn (for eksempel DO[15..0]) på samme måte som over.
- Trykk på RESULT\_A[15..0] pinnen, og dra musa et lite stykke ut fra pinnen.
- Slipp museknappen.
- Gi bussen ut fra RESULT\_A[15..0] et navn på samme måte som over. Navnet må være det samme som navnet på bussen ut fra DO[15..0]. Trykk på *OK*.

Nå er utgangen DO[15..0] koblet til inngangen RESULT\_A[15..0], selv om det ikke ser slikt ut på skjemategningen. *Det er navnet som binder dem sammen.* Alle busser og ledninger med samme navn (i en skjemategning) er koblet sammen, uansett om de ikke er synlig koblet sammen. Kobling ved navn kan benyttes til å koble ledninger til enkeltledninger i en buss.

Bruk buss og ledning til å koble inngangen ENABS til ledning 4 i C[7..0] bussen:

- Tegn en buss ut fra C[7..0] pinnen og kall den C[7..0].
- Velg ledningstegner (*Orthogonal Node Tool*) med -knappen.
- Trykk på ENABS pinnen og dra musa et lite stykke ut fra ABS modulen.
- Slipp museknappen.
- Gi ledningen navnet: C[4] på samme måte som for bussene over.

En ledning som har samme navn som en buss, og et nummer [#] etter navnet er koblet til ledning # i bussen. Bruk også buss til å koble inngang B[15..0] til Result\_B[15..0]. Når dere er ferdige skal skjemaet StudentDesign se ut omtrent som på figur 2-17. Her er det en del inngangs- og utgangsspinner som ikke er tilkoblet noe. Det gjør ingen ting. Vi vil benytte disse pinnene i senere labber.

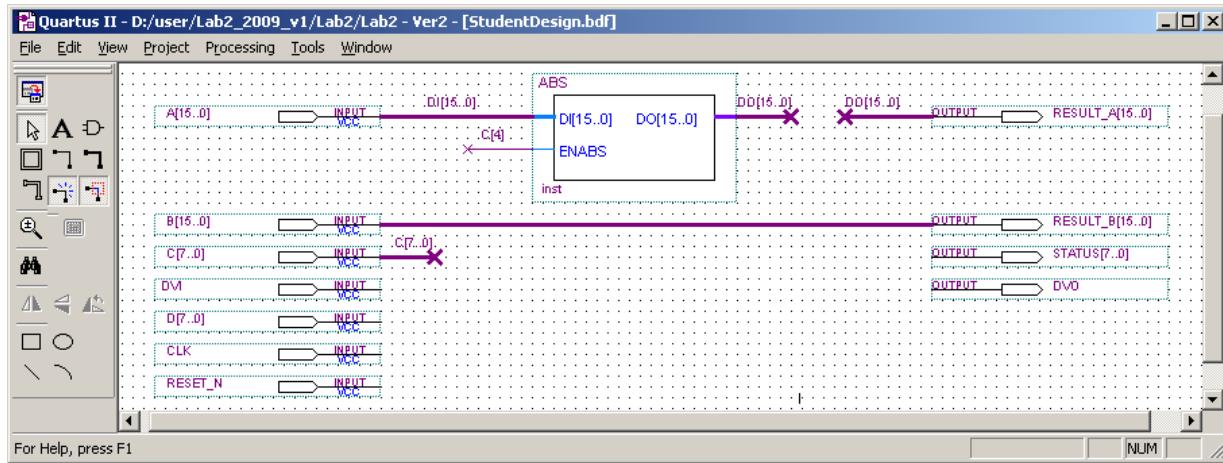
**Merk:** ledningene i bussen telles fra 0. Så en buss på 8 bit har ledning nummer 0 til 7.

Dobbeklikk dere nedover i hierarkiet til dere kommer inn i en av KRETS\_12 modulene. Dere får nå opp et skjema som vist i figur 2-18. Bruk det som dere har lært om skjemategneren til å tegne inn kretsene som dere kom fram til i forarbeidets 0 og 0, og koble de sammen som i 0. (uten å dele de opp i to modularer). For å hente inn logisk porter henter dere opp komponentbiblioteket i figur 2-15 og åpner standardbiblioteket *primitives → logic*. Merk av for *Repeat-insert mode* dersom dere skal hente inn flere porter av samme type. Når dere har hentet inn de dere skal, trykker dere *Esc-knappen* på tastaturet.

Dersom en til tider er usikker på om en buss eller ledning er koblet til en komponent, kan en forsøke å dra komponenten litt rundt. Dersom ledningene er koblet til vil de følge med. En kan bruke CTRL-Z til å angre flyttingen etterpå.

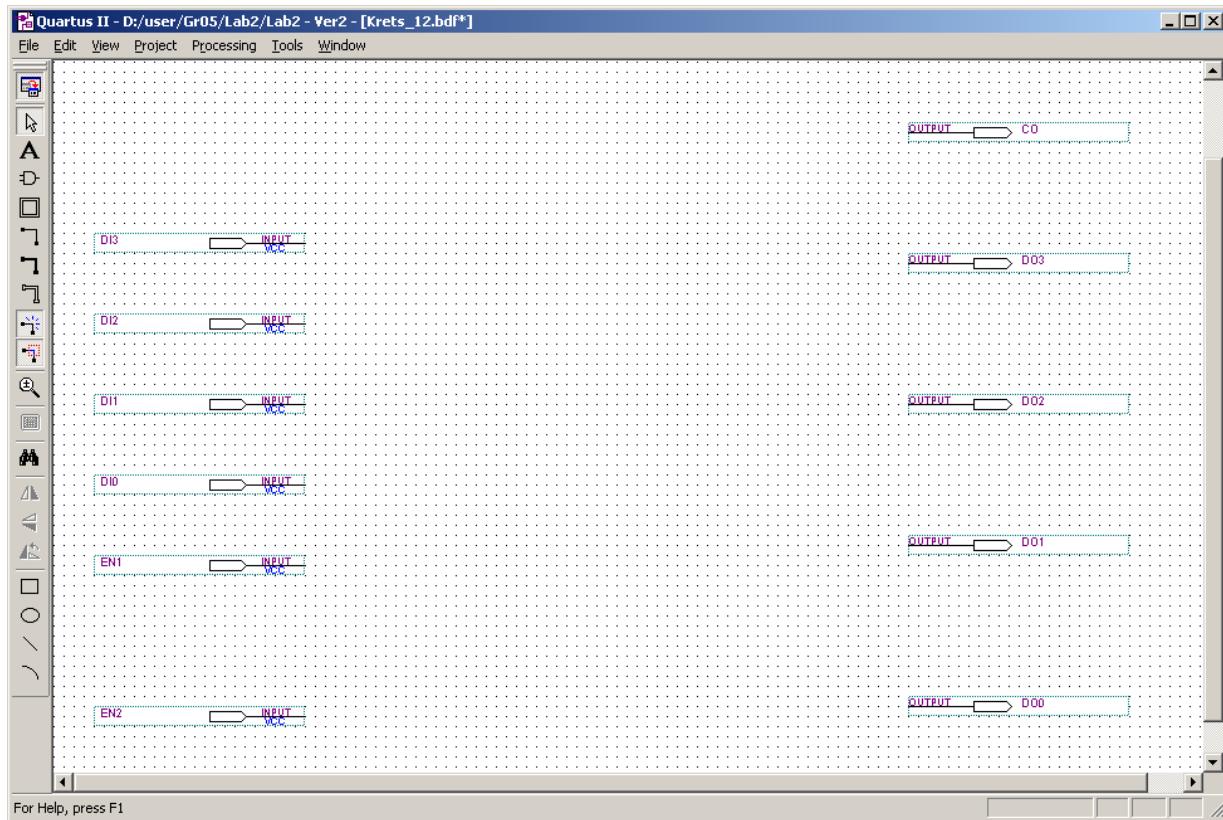
Et kryss på en ledning indikerer en ikke tilkoblet ende. Vær oppmerksom på dette dersom dere ikke kobler til ved navn, da det vil indikere at dere ikke har fått koblet på ledningen. Klikk på krysset (dere går da automatisk til ledningstegnemodus) og dra musa til koblingspunktet for å få den koblet på.

En pinnes koblingspunkt er helt på tuppen av pinnen.



Figur 2-17: Skjemavindu med studentdesign og ABS-modul fullt tilkoblet

(Merk at DO-bussen er koblet ved hjelp av navn)

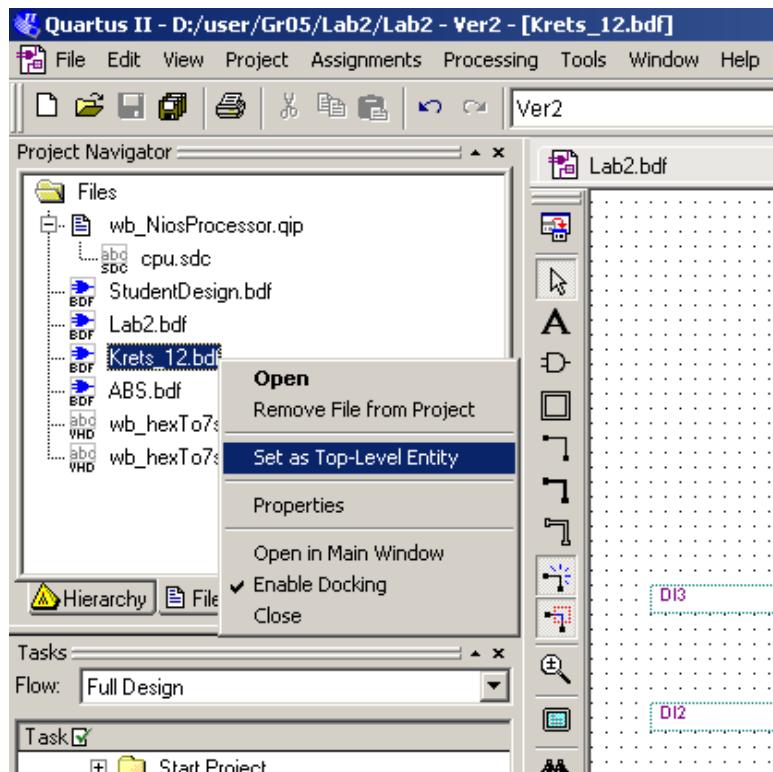


Figur 2-18: Skjema for KRETS\_12

Når dere er ferdige å tegne inn kretsen må dere lagre den (*File → Save*). Gå tilbake til ABS-modulet vist i figur 2-16. Hvis dere her klikker på en av de andre KRETS\_12 modulene, så vil dere se at den endringen dere gjorde i en, også er gjort i de andre. I ABS modulen mangler det noen tilkoblinger fra inn- og utgangsbussene til KRETS\_12 modulene. For å koble på disse benytter dere ledningstegneren, , til å trekke ledninger mellom pinnene på KRETS\_12 modulene og de to bussene. I tillegg må dere gi hver enkelt ledning et navn som svarer til den bussledningen dere vil koble til. I figur 2-16 er for eksempel allerede busslinje DI[0] koblet til pinnen DI0 på den nederste instansen av KRETS\_12-modulen.

Til slutt legger dere til AND-porten og kobler opp de ledningene som mangler i henhold til forarbeidet 0.

Sjekk at designet stemmer overens med figurene, og *lagre designet*.



Figur 2-19: Å sette Krets\_12 til Top Level Entity

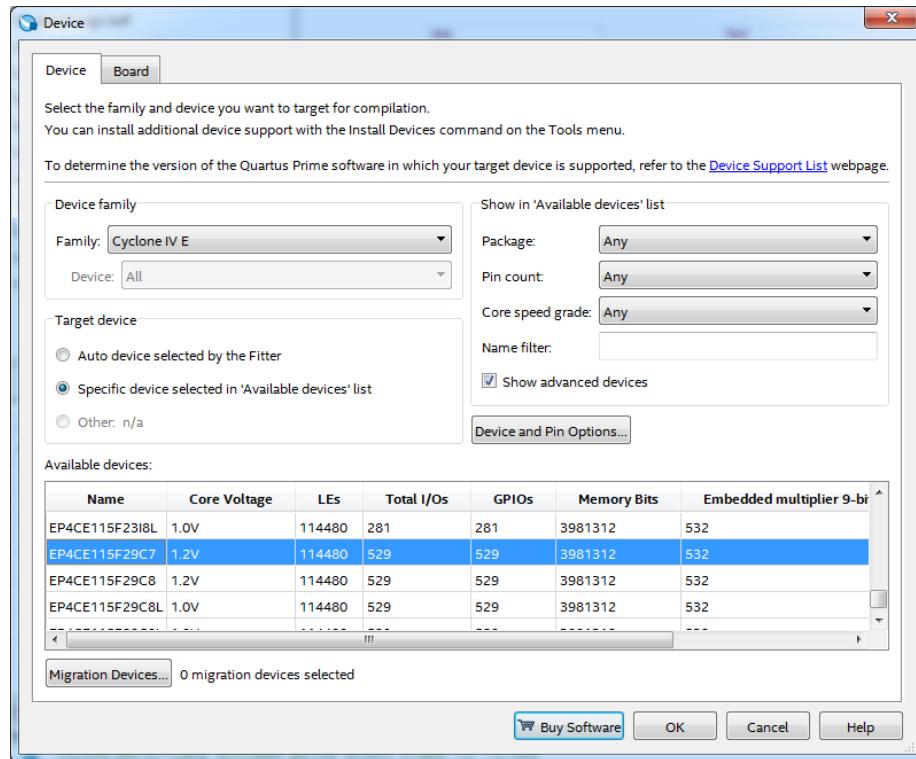
#### Laboppgave 4: Bruk av simulatoren (simulere KRETS\_12 og ABS-kretsen)

Erfaringsmessig er det bruk av simulatoren som er den mest krevende delen av labben. Legg derfor vekt på å forstå det som skjer. Eksperimenter gjerne på egenhånd.

Nå som vi har laget et design ønsker vi å simulere det, slik at vi kan luke ut eventuelle feil før vi senere laster designet ned til FPGA-brikken på DE2-kortet.

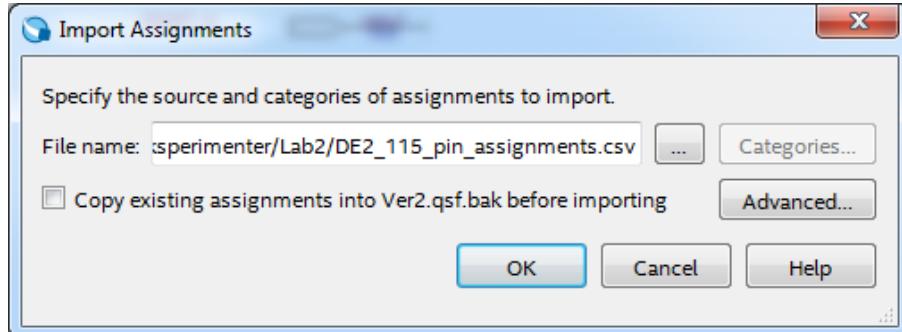
Sett først følgende:

- Velg komponenten i menyen Assignment -> Device (se Figur 2-20)



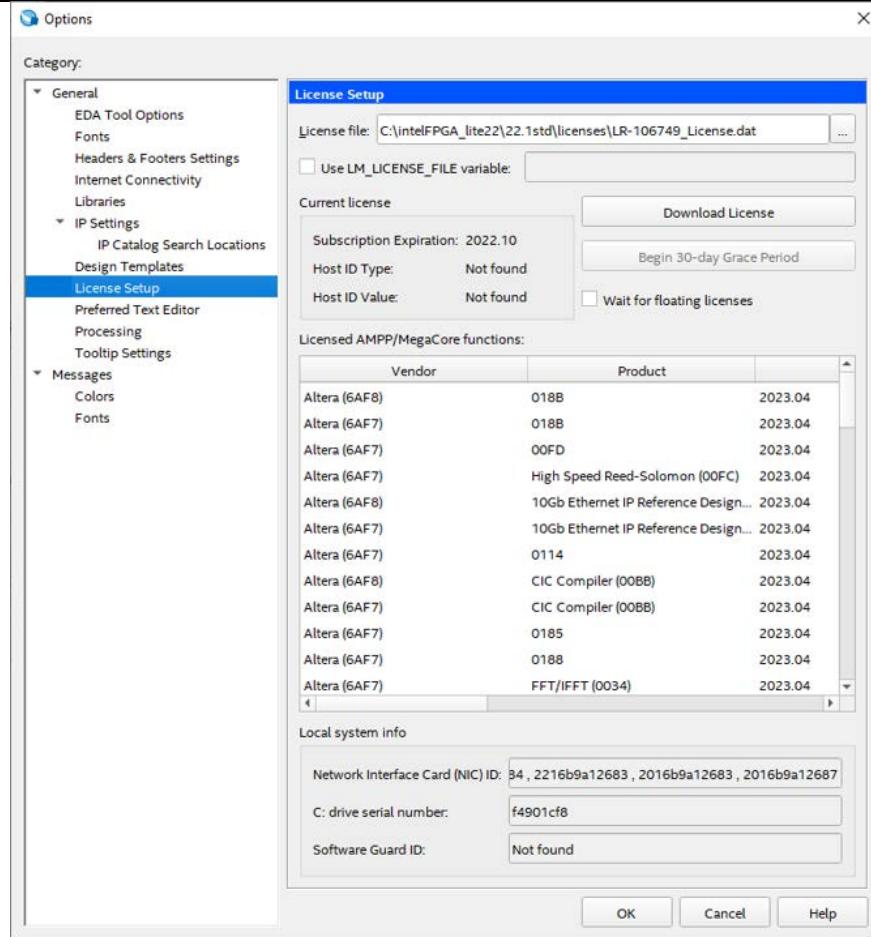
Figur 2-20 Vælg komponent

- Les inn fil med pinnetilordning i menyen Assignment -> Import Assignment (se Figur 2-21). .csv-filen ligger på Blackboard.



Figur 2-21 Pinnetilordning

- Les inn lisensfilen i menyen Tools -> Options (se Figur 2-22)

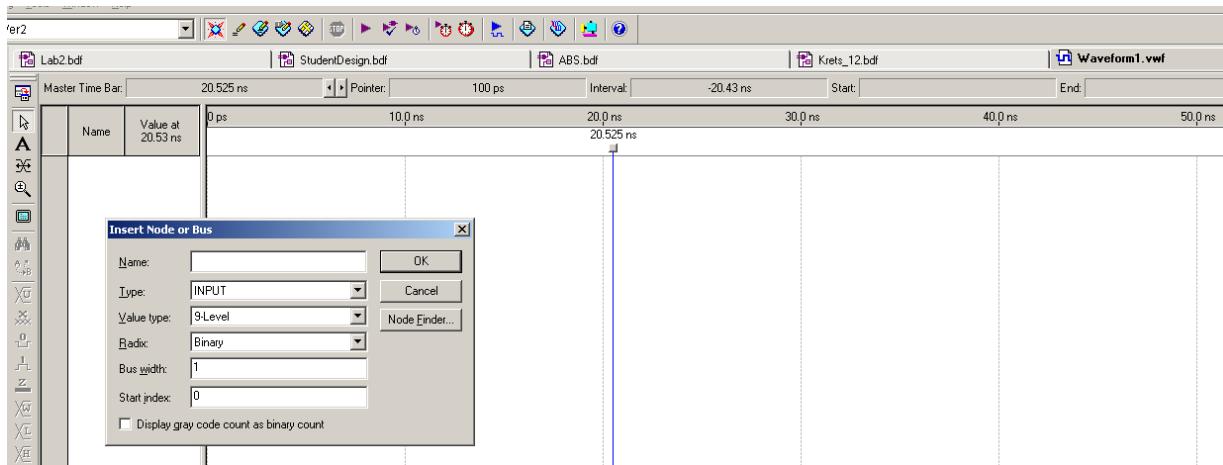


Figur 2-22 lisensserver

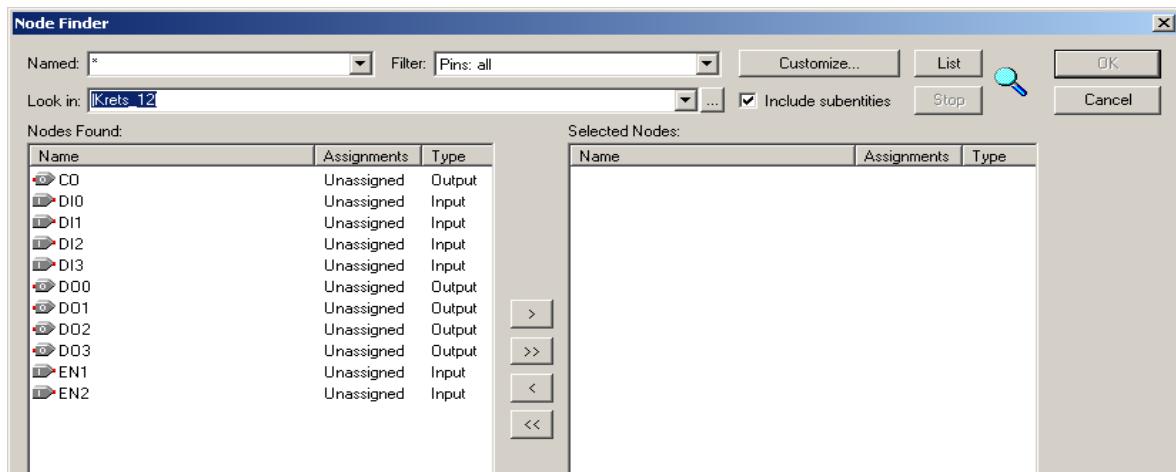
Det er fornuftig å starte med å simulere de minste byggeblokkene først, slik at vi er sikre på at disse fungerer korrekt. Deretter kan vi simulere større moduler, og til slutt hele systemet. Vi vil starte med å simulere Krets\_12. Sett denne som *Top Level Entity* ved å høyreklikke på den under *Files* i *Project Navigator* (se figur 2-19). Start deretter kompilering av denne modulen ved å velge *Processing* → *Start Compilation* (eller ved å trykke på knappen ). Dere vil antageligvis få en del advarsler (*Warning*) i denne sammenheng, men de kan dere (normalt) ignorere. Kontroller imidlertid at det ikke er noen viktige signaler som har beskjeden ”*Pin <signal name> is missing source*”, eller ”*Pin <signal name> not connected*”. I så fall må dere kontrollere at disse er koblet til slik de skal i skjemaene. Dersom dere får feilmeldinger, må dere dessuten rette opp disse. Dobbeltklikk på advarselen eller feilmeldingen, så blir dere brakt inn i det skjemaet der problemet befinner seg. Typisk vil også årsaken til feilen bli utevet.

Nå skal dere laget et bølgeformvindu med stimuli som skal påtrykkes kretsen. Velg *File* → *New* → *Verification/Debugging Files* → *University Program VWF*. I vinduet som åpnes dobbeltklikker dere i det tomme feltet under *Name*. Dere får da opp en dialogboks vist i figur 2-23. Her trykker dere *Node Finder ...* og en ny dialogboks dukker opp. Under *Filter* velger dere her *Pins: all* og trykker deretter *List*. Dette skal gi et resultat som vist i figur 2-24. Trykk nå på -knappen for å overføre alle inn- og utgangspinnene til *Selected Nodes*. Trykk OK to ganger. De valgte inn- og utgangspinnene vil nå komme til syne i *Waveform Editor* slik det er vist i figur 2-25.

Merk at vi her valgte modulens inn- og utgangspinn under *Filter*. Det er mulig å se på andre signaler ved å gjøre andre valg her. Eksperimenter gjerne med dette selv. I *Node Finder* er det også mulig å overføre bare valgte enkeltsignal ved hjelp av -knappen.



Figur 2-23: Waveform Editor

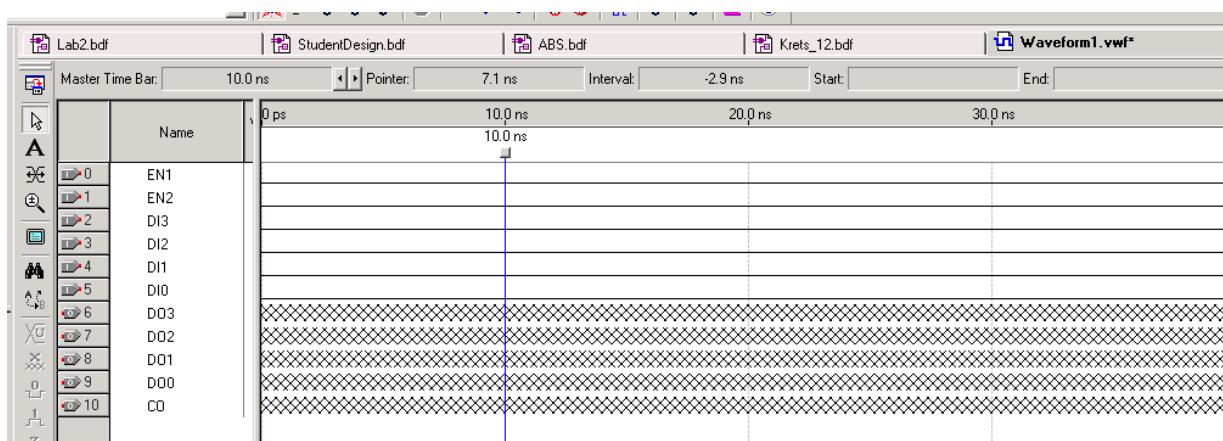


Figur 2-24: Node Finder

### Endre på visning av signaler i simulatoren

For å gjøre det enklere å lese resultatene av simuleringen, skal vi her benytte oss av noen funksjoner i *Waveform Editor*.

Det første en gjør er å sortere signalene i den rekkefølgen de står i figur 2-25. Velg signaler dere ønsker å flytte og dra dem til rett plass.



Figur 2-25: Waveform Editor med målesignaler

Neste trinn er å kombinere DI3-DI0 og DO3-DO0 til hver sin buss. Dette gjør en ved å markere DI3, DI2, DI1 og DI0, så velger en *Grouping* → *Group* på hurtigmenyen (høyreklikk). Gi bussen et navn, for eksempel DI, og angi hvordan verdier skal vises, i dette tilfellet *Radix: Binary*. Merk at det signalet som opprinnelig sto øverst (DI3 her) vises som mest signifikant (lengst til venstre) på bussen. Dette kan dere eventuelt endre under *Group and Bus bit order* på hurtigmenyen. Gjør det samme for DO3-DO0. Merk at dere kan studere enkeltbit på bussen ved å trykke på +-knappen foran bussnavnet.

Vi skal nå definere oppløsningen og simuleringslengden i *Waveform Editor*. Dette gjøres med henholdsvis *Edit* → *Grid Size...* og *Edit* → *End Time...*. Velg 50 ns og 10 us. Dere kan zoome inn og ut ved å trykke på -knappen, og deretter høyreklikke eller venstreklikke på signalene. Zoom så langt ut at dere ser et antall gridlinjer. Hvis dere zoomer for langt inn blir det vanskelig å sette påtrykk på inngangssignalene, slik vi nå skal gjøre.

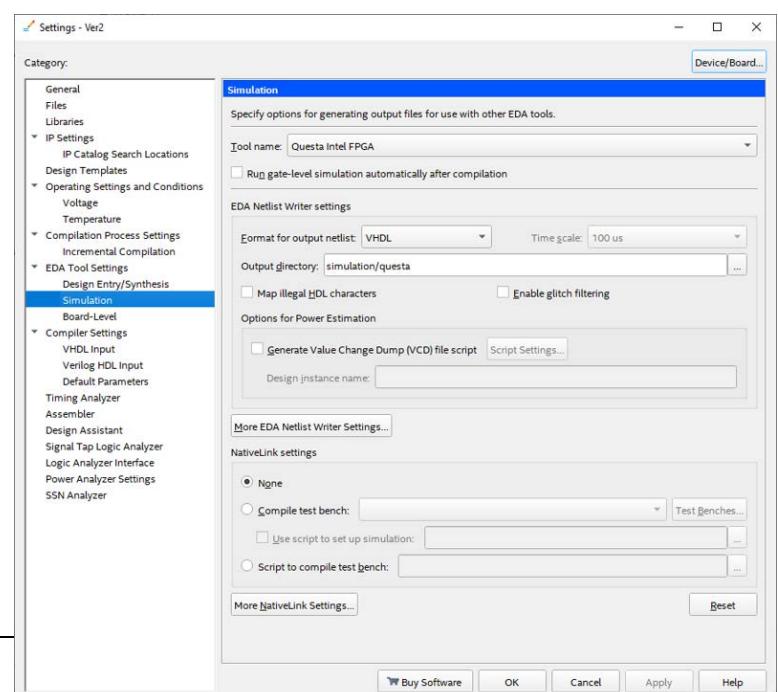
### Sette påtrykk for inngangssignalene

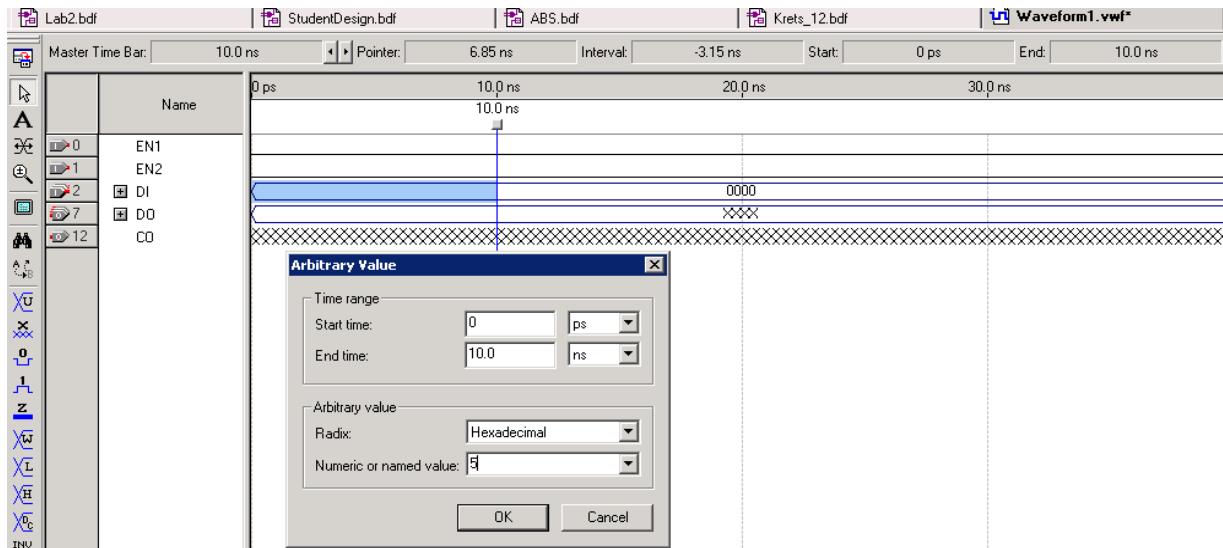
Nå som signalene er klare, kan vi sette påtrykk på inngangssignalene og simulere kretsen. Enkeltsignaler, slik som EN1 og EN2, setter vi enklest ved å trykke *Waveform Editing Tool* knappen () og deretter dra musa over det området man ønsker å invertere. I utgangspunktet er et signal satt til 0, så vi kan sette det til 1 ved å trykke venstre musknapp og dra denne over det aktuelle området. Dersom man har valgt et område på et signal så kan man også sette dette til andre verdier, for eksempel uinitialisert () eller ukjent (). Merk at hvis dere har zoomet så langt inn at dere ikke ser noen grid-linjer, så er det ikke mulig å endre verdi på signalene. Da er det bare å zoome ut litt først.

For bussignaler, slik som DI, er det enkleste først å trykke *Selection Tool* knappen (, velge det signalintervallet man vil endre, og så velge *Value* → *Arbitrary Value...* på hurtigmenyen. Under *Radix*, kan man her velge ulike visningsformat, for eksempel heksadesimal. Deretter kan man skrive inn den verdien man vil sette bussen til under *Numeric or named value*. Et eksempel på dette er vist i figur 2-26. Hvis man velger en radix som er forskjellig fra den man har angitt for bussgruppen, så vil det dukke opp en advarsel. Det er imidlertid ikke noe i veien for at de er forskjellige. Eksperimenter gjerne med dette.

De to siste signalene, bussen DO og signalet CO, er utgangssignal og skal få verdier fra kretsen basert på hvilke inngangssignaler en har.

Før simulering kan starte må man velge verktøy under Assignments -> Settings. Velg "Questa Intel FPGA" som "Tool name" og "VHDL" for "Format for output Netlists" (her kan annet språk velges om man ønsker det).



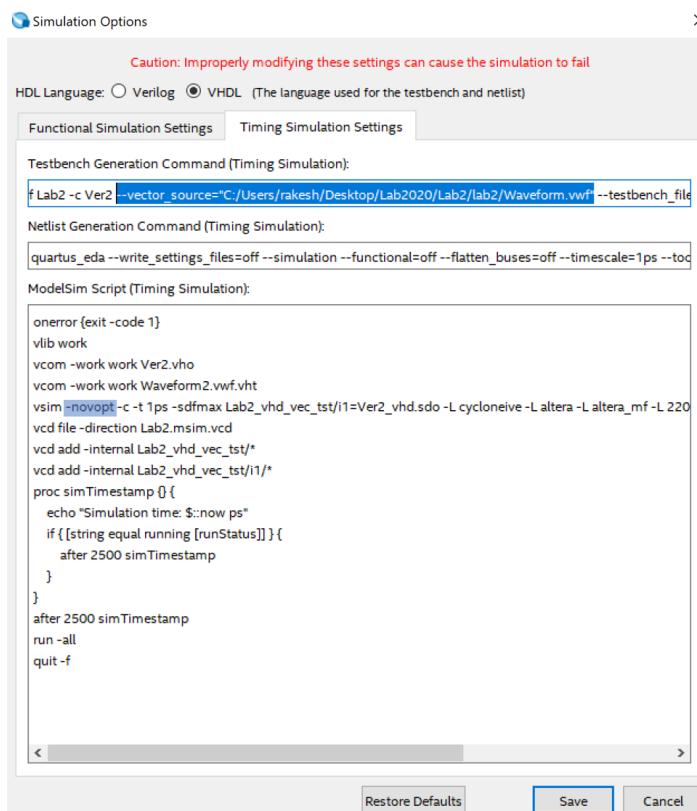


Figur 2-26: Vindu for å påtrykke simuleringseverdier

### Innstilling av simuleringsegenskaper

Lagre bølgeformvinduet med *File* → *Save As*. Dere kan for eksempel kalle filen Krets\_12\_sim.vwf. Pass på at *Add file to current project* er merket av i dialogboksen.

Simuleringsinnstillingen kan endres fra Simulation Waveform Editor ved å klikke på *Simulation -> Simulation Settings*. Forsikre deg om at du har valgt riktig bølgeformfil under *--vector\_source* som vist i Figur 2-27. Husk også å fjerne *-novopt* flagget ettersom det er utfaset.

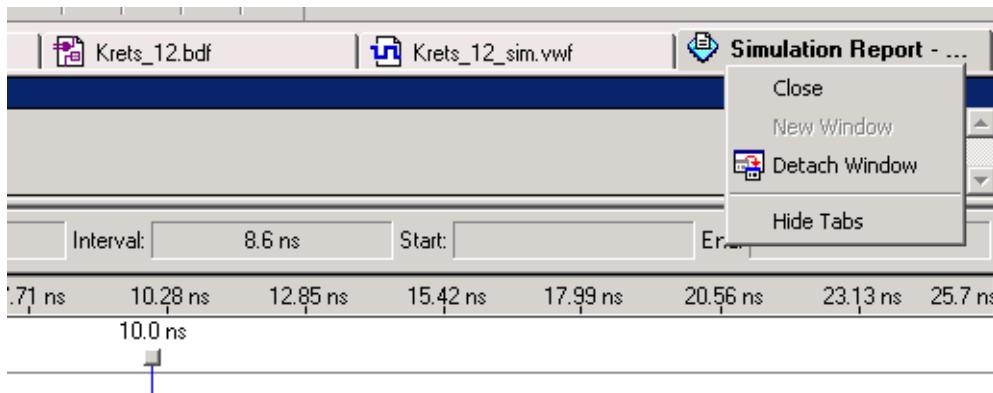


Figur 2-27 Simulator Setting

Noen ganger vil dere kanskje lage en ny versjon av bølgeformfilen med *File → Save As*, og så jobbe videre med denne. Da må dere imidlertid huske å skifte *Simulation input* som angitt over.

### Utføre simuleringen

Da er alt klart til å utføre simuleringen. Sett påtrykkene til det første dere har i simuleringsplanen for KRETS\_12 fra 0. Trykk så på *Start Simulation* knappen (▶). Simuleringen starter, og når den er ferdig dukker det opp en simuleringsrapport og (forhåpentligvis) en beskjed om at *Simulator was successful*. Trykk OK på denne. Lukk simuleringsrapporten ved å høyreklikke på *Simulation Report* og velge *Close* (se figur 2-28). Svar *Yes* på at dere vil oppdatere den opprinnelige bølgeformfilen. I denne skal nå DO og CO bli det som dere regnet ut i forarbeidet. Dersom det ikke stemmer overens, må kretstegningen, simuleringsoppsettet og simuleringsplanen sjekkes for feil. Får dere Z eller X er det mest sannsynlig feilkoblinger i kretsen.



Figur 2-28: Lukk simuleringsrapport.

Sett på et eller flere nye påtrykk og simuler på nytt. Gjenta til dere har utført alle testene i testplanen deres.

- ☞ I overgangen mellom to stabile utgangssignal kan det hende at bølgeformvinduet viser flere signalverdier. Hva kan årsaken til dette være?

### Simulering av hele ABS-kretsen

Dere skal nå simulere hele ABS-modulen. For å få en simulering så lik utførelsen på DE2-kortet som mulig, skal dere imidlertid benytte StudentDesign.bdf som *Top-Level Entity*. Siden det nå simuleres utenfra ABS-modulen, vil dere kunne detektere feil i sammenkoblingene av KRETS\_12 modulene og også feil i koblingen av ABS-modulen til signalene på utsiden av StudentDesign.

Kompiler designet, lag tilpasset bølgeformfil og utfør simuleringen. Husk å endre navn på *Simulation Input* som vist i figur 2.27. Sammenlign med resultatene fra forarbeidet.

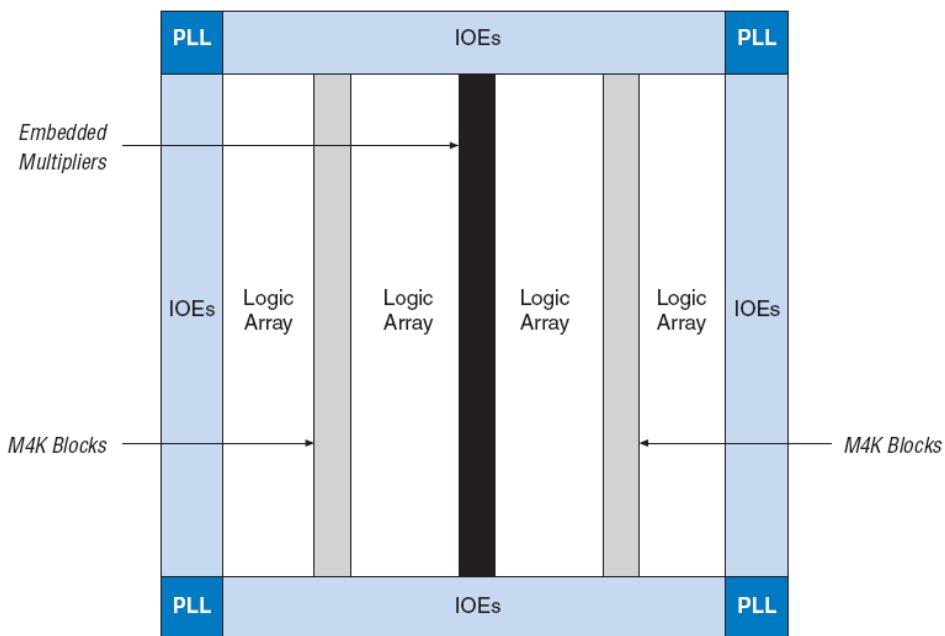
Eksperimenter også med bruk av testbenk og Modelsim som alternativ til å bruke bølegformfil.

## Laboppgave 5: Implementasjon og nedlasting av design til FPGA

Designet dere har laget skal nå lastes ned på DE2-kortet. På dette kortet sitter det altså en FPGA som kan konfigureres til å inneholde den maskinvaren som trengs for å realisere deres design pluss det ferdige grensesnittet mot DE2-kortet.

- Sett Lab2.bdf til å være *Top-Level Entity*.
  - Start generering av nødvendige filer for konfigurering av FPGA-en ved å trykke på Start Compilation knappen ().

Det som nå faktisk skjer er at det komplette designet blir omformet til en flat nettliste. Denne inneholder informasjon om alle de ulike komponentene i designet, og hvordan disse er koblet sammen. Nettlisten blir så optimalisert og tilpasset de ressursene som er tilgjengelig på FPGA-en. Basisressursene på en Cyclone II FPGA er logisk elementer (LE) og I/O elementer (IOE). En LE består litt forenklet av en oppslagstabell (look-up table) for realisering av boolske funksjoner samt en D-vippe (les kapittel 5.12 Read-Only Memory for en beskrivelse av hvordan en oppslagstabell kan realiseres). Det finnes 35.000 slike LE-er på FPGA-en på DE2-kortet. IOE er rutingskanaler som benyttes til å koble sammen de ulike LEene og til å koble LE-ene til FPGA-ens inn- og utgangsspinner. I tillegg inneholder vår Cyclone II FPGA 105 Random Access Memory (RAM) blokker a' 4kbit, 35 multiplikatorer 18x18 bit og fire faselåste sløyfer (PLL) som kan benyttes til å generere klokkesignaler. Figur 2-29 viser et blokkdiagram av en typisk Cyclone II FPGA. Her er det bare tegnet opp IOE langs kanten av kretsen. Det går imidlertid også horisontale og vertikale rutingskanaler mellom LEene inne i hvert Logic Array.



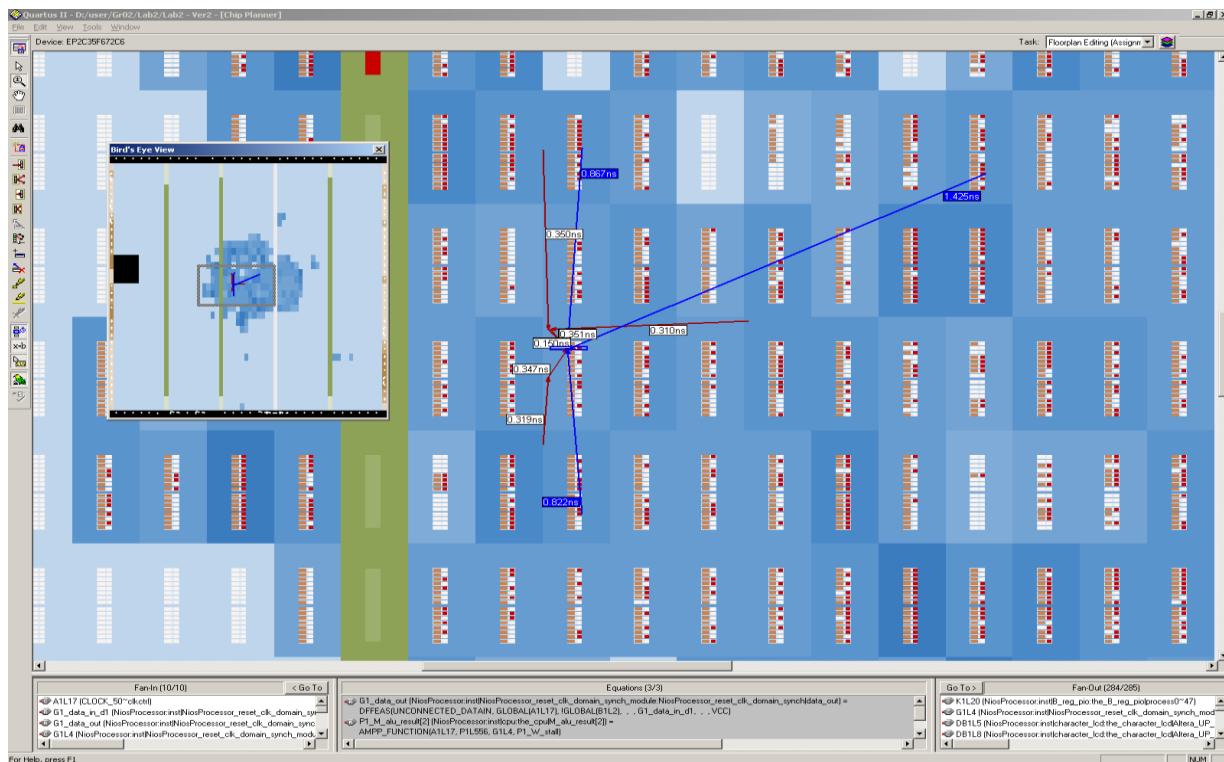
*Figur 2-29: Cyclone II FPGA blokkdiagram*

Det siste som skjer i kompileringsprosessen er at komponentene i designet blir tilordnet fysiske LE-er i FPGA-en og at det blir bestemt hvordan IOE-linjene skal kobles sammen for å sette opp kanaler mellom LE-ene. Ut fra denne tilordningen genereres det så en konfigureringsfil som deretter kan lastes ned i FPGA-en. Denne bestemmer da hva som skal stå i de ulike oppslagstabellene og hvordan IOE-linjene er koblet sammen.

Kompileringen kan ta en del tid og vil gi en god del advarsler. Så sant dere er sikre på at StudentDesign er korrekt, så kan dere ignorere disse. Det vil til slutt komme opp en kompileringsrapport. Velg her *Fitter → Summary*.

- #### ❖ Hvor mange LEer brukes totalt i designet?

Den resulterende ressursbruken i FPGA-en kan også vises i et program som heter *Chip Planner*. Velg *Tools* → *Chip Planner*. Her kan man blant annet zoome inn på en enkelt LE og se hvordan denne er koblet mot andre LEer (velg en LE og ta deretter *View* → *Generate Fan In Connections*, eller andre *Generate ...* muligheter). Se figur 2-30 for et eksempel.



Figur 2-30: *Chip Planner*

Det som gjenstår nå er å laste konfigureringsfilen ned på FPGA-en.

- **Les notat om antistatisk armlenke som dere finner på Blackboard FØR dere henter DE2-kortet.**
- Hent et DE2-kort, en USB-kabel, batterieliminator og et tastatur (ikke mikrofon eller hodetelefoner).
- **Ta på dere ledende håndlenker og sett disse han-bananpluggene i hun-pluggene på bunnplaten under kortet.**
- Koble DE2-kortets *Blaster* inngang (til venstre i bakkant) til PC-en med USB-kabelen.
- Koble DE2-kortets *PS2* inngang (øverst på høyre kant) til tastaturet.
- Koble DE2-kortet til en stikkontakt med batterieliminatoren.
- La SW17 på DE2-kortet stå i posisjon RUN beständig.
- Skru på DE2-kortet med den røde ON/OFF knappen. Pass på at RUN-PROG bryteren (midt på venstre kant) står på RUN.

**NB: DE2-KORTET ER ØMFINTLIG**

Komponentene kan blant annet skades av statisk elektrisitet. Vær forsiktig når dere bærer det. Forsøk å ikke røre på andre deler enn pleksiglasset og bunnplaten.

**Ha alltid på dere håndlenke koblet til kortet.**

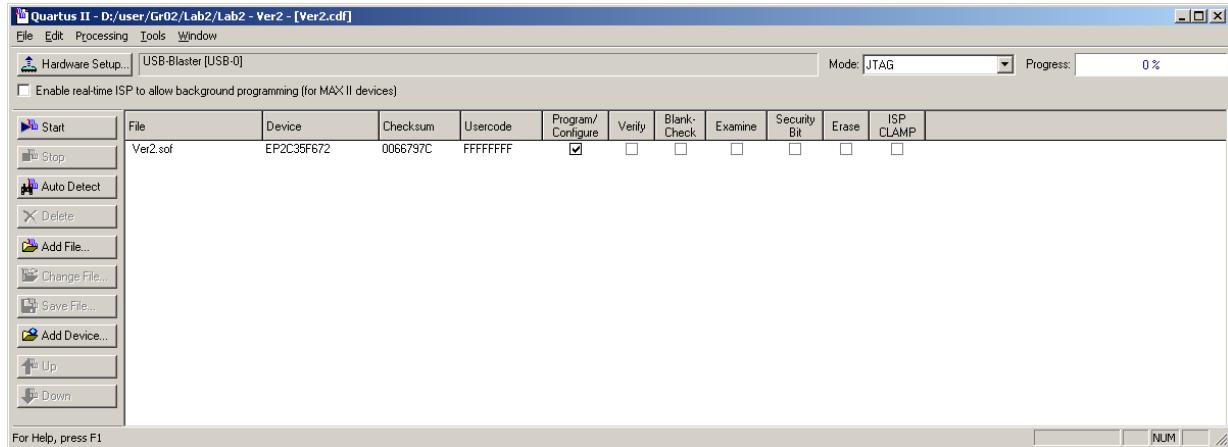
**Første gang dere bruker kortet fra en PC må driveren for nedlasting (USB-Blaster) installeres.**

**Se:**

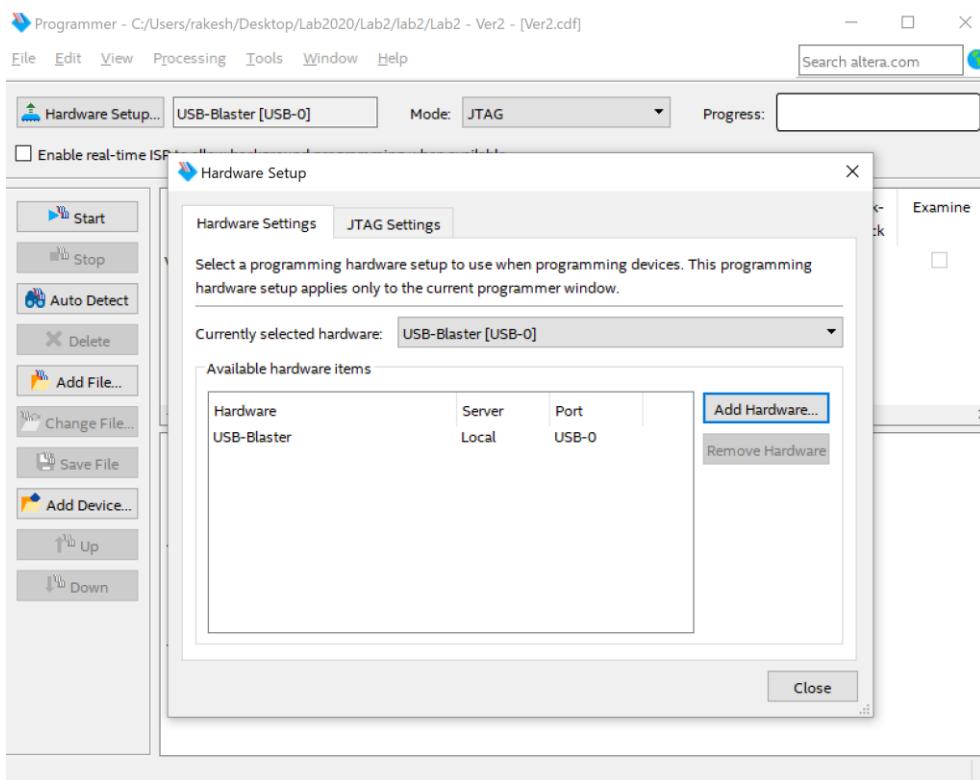
44

[http://www.terasic.com.tw/wiki/Altera\\_USB\\_Blaster\\_Driver\\_Installation\\_Instructions](http://www.terasic.com.tw/wiki/Altera_USB_Blaster_Driver_Installation_Instructions)

Start nedlastingsprosessen ved å velge *Tools → Programmer* (eller ved å trykke på knappen ). Det vil da komme opp et vindu som minner om det dere ser i figur 2-31 (filnavnene kan være noe annerledes). Forsikre deg om at alternativet *Program/Configure* er merket. Klikk *Hardware Setup* og velg *USB-Blaster [USB-0]* fra rullegardinmenyen *Currently Selected Hardware* som vist i Figur 2-32. Trykk *Start* her. Hvis alt går bra vil Progress etter en stund vise 100 %. På DE2-kortet vil alle sjusegmentdisplayene vise 0, og LCD-displayet vil kun vise en posisjonsmarkør.



Figur 2-31: Program for nedlasting av design



Figur 2-32 Choose USB-Blaster

Designet som nå er lastet ned i FPGA-en (Figur 2-12) tilsvarer i grove trekk det som er avbildet på den logiske skissen på innsiden av omslaget til labheftet. Forskjellene består i at CODEC INTERFACE ikke er inkludert og at Student Design bare består av ABS-modulen. Siden CODEC INTERFACE ikke er med, er det heller ikke behov for multipleksere (MUX) på

inn- og utgangssignalene. Den siste forskjellen er at utgangsbussen RESULT\_B er lagt ut på sjusegmentdisplay HEX7-HEX4.

Verdiene på C- og D-registrene settes med brytere, henholdsvis SW7-SW0 og SW15-SW8. Verdien som er satt vises samtidig på lysdioden over bryteren. Lys på lysdioden svarer til aktiv høyt signal (1) på den tilsvarende biten i registeret. Den eneste registerbiten som skal benyttes i denne labben er C(4), som da altså kan settes med bryter SW4.

Verdiene på A- og B-registrene settes ved hjelp av tastaturet med følgende prosedyre.

- Tast inn fire heks-sifre på tastaturet.
- Sifrene vises nederst til venstre på LCD-displayet, med en t foran seg.
- Tast bokstaven A eller B avhengig av hvilket register dere ønsker å legge verdien i.
- Sifrene overføres til øverste linje etter henholdsvis bokstaven A eller B.
- Hvis dere taster feil kan verdien som står etter t slettes ved å taste et annet heks-siffer enn A eller B noen ganger. Deretter kan man taste inn de riktige sifrene.

Eksempel: Man skal skrive 0A89 til register A (*t* viser \_ \_ \_ \_):

- Tast første 0 A 9 0 på tastaturet. *t* viser 0 A 9 0 (0 er altså det *nest* signifikante sifferet)
- Tast A på tastaturet (for register A). Som ny A vises nå 0A89. *t* «tømmes», og viser igjen \_ \_ \_ \_.

Sett opp registrene A og C slik at dere får testet ABS-kretsen på kortet i henhold til testplanen i Tabell 2-7. Resultatet fra ABS-kretsen vil vises på sjusegmentdisplay HEX3-HEX0.

Tips: Se innsida av omslaget for forklaring på register C.

- ❖ Gå inn i modulen StudentDesign i designet og fjern bussen mellom B[15..0] og Result\_B[15..0]. Kompiler og last ned designet på nytt. Hvilke konsekvenser har dette for systemet?

## Avslutning

- Skru av DE2-kortet med den røde ON/OFF knappen.
- Koble fra batterieliminator og USB-kabelen (i den rekkefølgen). Sett tilbake DE2-kort, parallellkabel, batterieliminator og tastatur.
- Avslutt Quartus II.

# 3 Laboratorieøving: Deteksjon av forsterkningsfaktor

## Innledning

### Formål

Å lage en modul som bestemmer hvor mye vi må forsterke inngangssignalet for å få ønsket nivå på utgangssignalet. Mye av modulen er allerede laget, og oppgaven vil bestå i å forstå hovedtrekkene av det som allerede er laget samt å designe resterende deler.

### Hensikt

- Øve design av kombinatoriske og sekvensielle kretser ved hjelp av modulen FACT
- Analysere og forstå en «større» digital krets
- Øve god designmetodikk og bruk av Altera designverktøy

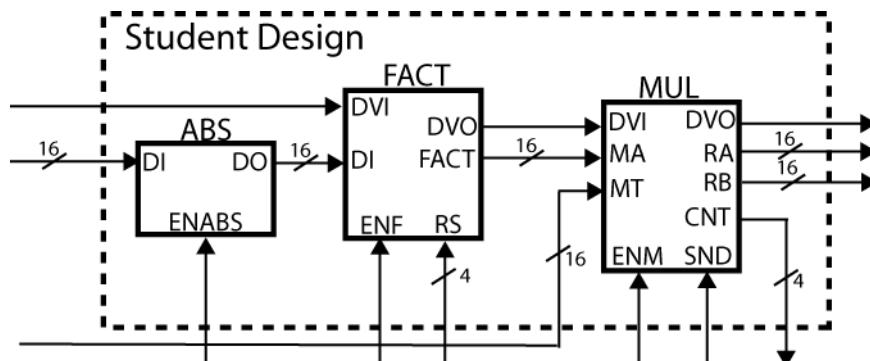
### Bakgrunn

Teorien om dette finnes i:

- Daniel Gajski: «Principles of Digital Design», kapittel 4, 5, 6 og 7.
- Detaljer om lydkompressorer finner dere på følgende Wikipedia-artikkel:  
[http://en.wikipedia.org/wiki/Audio\\_compressor](http://en.wikipedia.org/wiki/Audio_compressor) (1. august 2013)

### Hva som skal gjøres

Det skal designes en kombinatorisk og to sekvensielle kretser i henhold til gitte spesifikasjoner. Videre skal det eksperimenteres med disse for å få en bedre forståelse av systemet vi jobber med.



Figur 3-1: Signalbehandlingssystem

## Systembeskrivelse

### Lydkompressorens funksjon

Vår lydkompressor består som før nevnt av tre hoveddeler som vist i figur 3-1. I forrige lab designet dere absoluttverdikretsen ABS. FACT-modulen som dere skal fullføre nå, tar utgangspunkt i den beregnede absoluttverdien av inngangssignalet og finner hvilken forsterkning som trengs for å gi ønsket lydnivå ut. I lab 4 vil dere så multiplisere denne forsterkningen med det opprinnelige lydsignalet, for å få et jevnt sterkt lydsignal ut. I vår lydkompressor vil i praksis signalet først bli dempet, for deretter å forsterkes til ønsket nivå. FACT-modulen foretar dempingen og finner deretter hvilken forsterkningsfaktor som trengs.

### Kommunikasjonsprotokoll

Modulene i designet vårt kommuniserer med hverandre og med omverdenen ved hjelp av en enkel protokoll basert på signalene Data Valid In (*DVI*) og Data Valid Out (*DVO*). Når en modul mottar aktiv *DVI*, tar den de data som står på sin datainngang (*DI*) og utfører sin oppgave. Når modulen er ferdig, setter den sine prosesserte data på utgangen (*DO*) og aktiviserer sin egen *DVO*. Dette registreres av neste modul, som starter på sin oppgave, osv.

I figur 3-1 er det en del signaler som for enkelhets skyld ikke er tegnet inn. I tillegg til VDD og GND, har vi global CLK og RESET\_N.

CLK er systemklokken, som går hele tiden. RESET\_N er et kontrollsignal. Aktiv lav RESET\_N medfører synkron reset av samtlige registre i hele systemet. Alle registre skal drives av CLK. Det skal ikke forekomme asynkrone registre.

Da klokkesignalet (CLK) går til alle vipper, er det for enkelhets skyld ikke alltid tatt med i skjemategninger og i spesifikasjoner (i lab-heftet). Det er likevel tatt med om spesielle forhold skulle tilsi det (testformål). Det samme gjelder for reset-signalet (RESET\_N).

Merk at det ikke «koster» noe å ha reset på registrene, da dette er innebygget i LE-enes registre. Man bruker således ingenting av LE-enes kombinatorikk til dette. Se ordlisten for forklaring på LE.

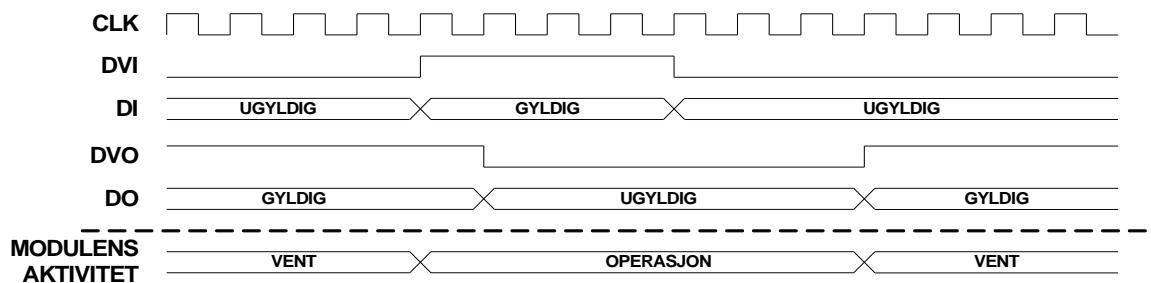
For hver modul gjelder:

- Vent på aktiv DVI.
- Når aktiv DVI detekteres:
  - Sett DVO inaktiv.
  - Les DI (Data In) og eventuelle kontrollinnganger.
  - Utfør aktuell operasjon
- Når operasjonen er utført, sett DVO aktiv.

I tillegg gjelder:

- Så lenge en modul registrerer at dens DVI er aktiv, er dens DI gyldig.
- Så lenge en modul holder sin DVO aktiv, er dens DO gyldig.

Nedenfor vises et typisk tidsdiagram:



Figur 3-2: Typisk tidsdiagram

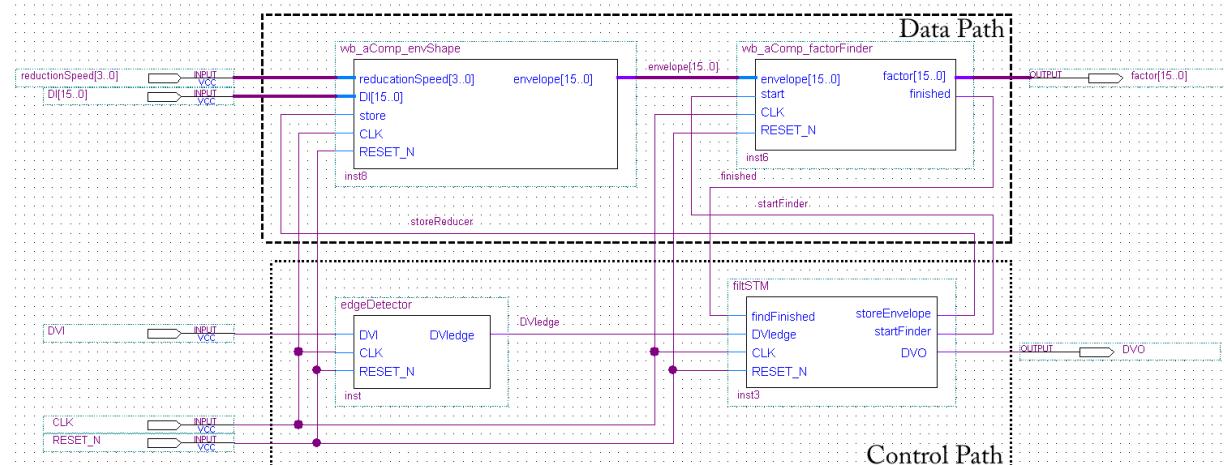
Merk at DVO settes inaktiv én klokkeperiode etter at DVI er blitt aktiv. Dette er typisk for et synkront system. At operasjonen i diagrammet varer i 7 klokkeperioder, er tilfeldig valgt.

Systemet har en fast «throughput» på 44100 sample-er pr sekund og en systemklokke på 11,2896 MHz. Dette gir oss  $11,2896 \cdot 10^6 / 44100 = 256$  klokkeperioder til rådighet til å behandle et sample. Se ordlista for forklaring av sample.

Systemet vil være i ventetilstand i mesteparten av tiden, da systemets prosessering av et sample tar mye mindre enn 256 klokkeperioder. Dette betyr at det garantert ikke vil bli noen «oppnopning» av sample-er i signalveien.

## FACT-modulen

FACT-modulen kan deles i en kontrollenhet og en datastien som vist i figur 3-3. Komponentene i datastien utfører selve databehandlingen i systemet, mens kontrollenheten bestemmer hva som skal gjøres og når. FACT-modulen har i tillegg inngangssignalet ENFACT (ENF i figur 3-1). Dette skal imidlertid ikke brukes før i lab 4, og er derfor ikke tatt med i figur 3-3.



Figur 3-3: FACT-modul

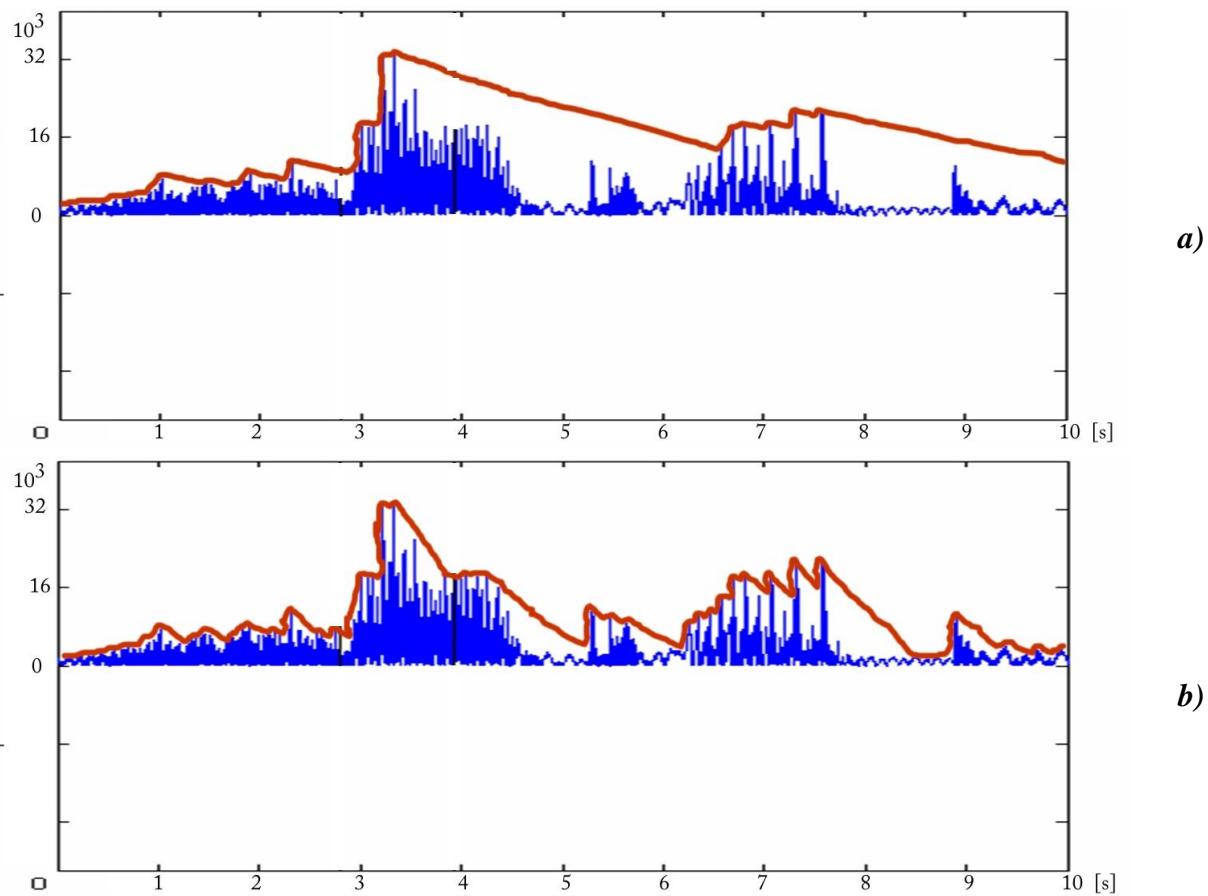
Den positive flankedektoren (*edgeDetector*) leter etter positive flanker på signalet *DVI* og gir da ut en puls som er en klokkeperiode lang. Dette er nødvendig fordi det i et system som arbeider etter protokollen beskrevet over er rom for «misforståelse» modulene imellom. La oss si at modul A sin *DO*- og *DVO*-utgang er koblet til modul B sin *DI*- og *DVI*-inngang. Hvis *DVO<sub>A</sub>* er aktiv i lang tid, vil modul B kunne prosessere samme data flere ganger, hvis den har «*DVI aktiv*» som kriterium for å starte prosesseringen.

Det er altså ikke nok å sette «*DVI aktiv*» som betingelse for å gå ut av ventetilstanden, i en tilstandsmaskin, da man kan risikere å ankomme ventetilstanden igjen (når man er ferdig med modulens prosessering) før *DVI* blir inaktiv.

Man kan kanskje tenke at dette ikke gjør noe, siden det samme resultatet kommer ut uansett. Problemet er at modul B vil sette sin *DVO*-utgang høy og lav en gang *ekstra*, og eventuell etterkommende modul vil motta flere data enn det opprinnelig ble sendt inn i systemet.

På DE2-kortet ville dette ha medført at interfacet til CODEC OUT vil motta sample-er med for høy frekvens. I tillegg vil mange påfølgende sample-er være like (og feil).

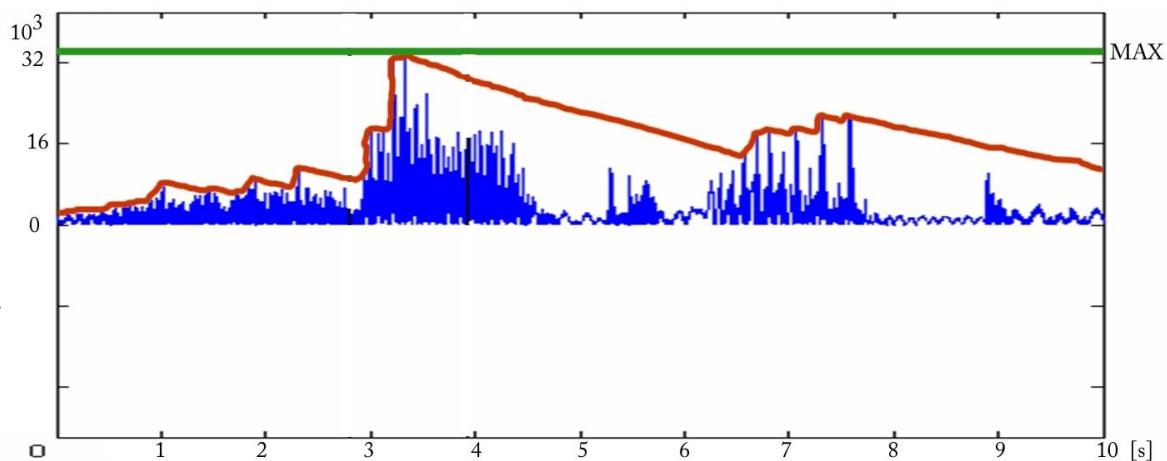
Tilstandsmaskinen *filtSTM* i figur 3-3 vil gå ut av sin ventetilstand når *DVIedge* pulses. Den vil da starte behandlingen av det nye lydsampllet som ligger på *DI*. Modulen *envShaper* har som oppgave å lage en omhylningskurve rundt absoluttverdien av de innkommende lydsamplene. For signaler med stigende lydstyrke vil omhylningskurven umiddelbart følge signalet oppover. For signaler med fallende lydstyrke vil omhylningskurven gradvis falle. Den vil altså ikke følge signalet direkte nedover. Hastigheten omhylningskurven faller med avhenger av verdien på inngangen *reductionSpeed*. Hvis denne er satt til 0000 vil vi få det langsomste fallet, mens 1111 gir det raskeste fallet. Dette er røft skissert for langsom hastighet i figur 3-4 a) og for rask hastighet i figur 3-4 b). *envShaper* er rent kombinatorisk med unntak av et register på utgangen som lagrer gjeldende verdi av omhylningskurven. Det er *filtSTM* som med signalet *storeEnvelop* styrer lagring av ny verdi i dette registeret.



Figur 3-4: Omhylningskurver. a) langsom, b) rask

Modulen *factorFinder* har som oppgave å bestemme hvilken forsterkningsfaktor man må multiplisere lydsignalet med for å få det ønskede lydnivået ut. I figur 3-5 er dette nivået angitt med *MAX*. *factorFinder* finner med andre ord det vi må multiplisere den enkelte envelopeverdi med for å forsterke denne opp til *MAX*. Den samme faktoren skal benyttes til lydsignalet. Merk her at dersom *reductionSpeed* er rask, så vil lydsignalet raskt bli forsterket opp når inngangsvolumet reduseres. Dersom *reductionSpeed* er langsom vil det ta lengre tid før fallende inngangsvolum blir forsterket opp.

Å finne forsterkningsfaktoren tar et antall klokkesykler. *filtSTM* starter *factorFinder* med signalet *startFinder* når det ligger en ny envelopeverdi i utgangsregisteret til *envShaper*. Når *factorFinder* er ferdig legger den ut forsterkningsfaktoren på sin utgang *factor* og aktiviserer signalet *findFinished*. *filtSTM* kan da sette DVO høy for å indikere at data ut er gyldig.



*Figur 3-5: Forsterkning til maksverdi*

## Grensesnitt

Kretsen som er målet i denne labben gjengitt i figur 3-3 har følgende grensesnitt:

Tabell 3-1: FACT-modul grensesnitt

Navn	# Bit	Retning	Verdi	Betydning
DI	16	Inn		Data inn
redFactor	16	Ut		Data ut
DVI	1	Inn		Kontrollbit:
			0	DI er ugyldig
			1	DI er gyldig
DVO	1	Ut		Kontrollbit:
			0	DO er ugyldig
			1	DO er gyldig
redSpeed	4	Inn		Kontrollbit:
			0000	Lavest fart
			0001	Nest lavest fart
			...	
			1110	Nest høyest fart
			1111	Høyest fart
CLK	1	Inn		Klokke inn
RESET_N	1	Inn		Resetter kretsen

## Forarbeid

### Blackboard

- Les lab-infoen på Blackboard og hele kapittelet om lab 3 (også delen om labarbeidet). Møt opp på lab-forelesningen om lab 3.

### FACT-modulen

- ❖ Inne i FACT-modulen trengs det flere multipleksere (MUX). Dere skal designe en av dem. Nå i forarbeidet skal dere skissere en 1-bit 2-1 MUX. Dette er altså en krets med to 1-bit innganger. Avhengig av om et styresignal er null eller en, vil verdien på den ene eller den andre inngangen overføres til utgangen. Se kapittel 5.7 i Gajski for detaljer rundt multipleksere.
- ❖ Gjennomfør en skrivebordstest av designet.
- ❖ På labben skal multiplekseren velge mellom to busser med åtte eller flere bit hver. Dere trenger ikke å skissere en slik multiplekser, men dere skal lage en testplan for en 8-bit 2-1 MUX.

### Flankedetektor

Her skal protokollen studeres og den positiv flankedetektor designes. Flankedetektoren kan designes som en tilstandsmaskin, men det er også mulig å designe den direkte ved hjelp av to D-vipper i serie, en inverterer og en to-inngangs OG-krets. Med to D-vipper i serie menes at utgangen på den første er koblet direkte til inngangen på den siste. En positiv flanke som klokkes gjennom de to D-vippene kjennetegnes ved at det på et tidspunkt står en ener på den første D-vippen og en nul på den siste. I denne situasjonen skal flankedetektoren gi en ener ut.

- ❖ Hvordan vil dere koble invertereren og OG-kretsen til D-vippene?
- ❖ Skisser hvilke verdier som vil stå på utgangen av hver vipp og på utgangen av flankedetektoren når stimuli som vist i figur 3-6 påtrykkes flankedetektoren. Det forutsettes at begge vippene i utgangspunktet er resatt til 0, og at resetsignalet nå ikke er aktivt.



Figur 3-6: Stimuli for positiv flankedetektor

### Tilstandsmaskinen

Dere skal designe tilstandsmaskinen *filterSTM*. Tilstandsmakinen er i en ventetilstand inntil den får en puls på *DVledge*. Da setter den først ut en én klokkeperiode lang puls på *storeEnvelope*, og på de påfølgende klokkeperiodene, en to klokkeperioder lang puls på *startFinder*. Deretter venter den på signalet *finished* fra *factorFinder*. Merk at det vil ta et antall klokkeperioder før dette kommer. Når det er mottatt setter den *DVO* høy og returnerer til ventetilstanden. I ventetilstanden holdes *DVO* høy inntil det kommer en ny puls på *DVledge*.

Det gjør ikke noe om *DVO* settes høy når kretsen resettes. Siden inngangssignalene *DVIedge* og *finished* kommer fra synkrone komponenter som klokkes på samme klokke som tilstandsmaskinen, kan vi trykt lage en Mealy-maskin. Det er da mulig å klare seg med fire tilstander i tilstandsmaskinen.

- ❖ Tegn et tilstandsdiagram som oppfyller spesifikasjonen over.
- ❖ Velg en fornuftig tilstandskoding slik at dere får så lite kombinatorikk som mulig.
- ❖ Skriv opp tilhørende nestetilstands- og utgangstabell og benytt Karnaughdiagram til å forenkle likningene for nestetilstander og utgangsverdier.
- ❖ Tegn tilstandsmaskinen med vipper og kombinatorikk.
- ❖ Lag en testplan for tilstandsmaskinen. Ta hensyn til hvordan inngangssignalene oppfører seg med tanke på lengder og når de settes høy og lav.

## Labarbeid

### Oversikt

Labarbeidet i denne labben består av 7 oppgaver:

Oppgave	Innhold
1	Klargjøring av lab
2	Design av generisk multiplexer
3	Implementering av positiv flankedetektor
4	Implementering av tilstandsmaskin
5	Simulering av studentdesign
6	Implementasjon og nedlasting av design til FPGA
7	Avslutning

### Laboppgave 1: Klargjøring av lab

- Les lab-infoen på Blackboard

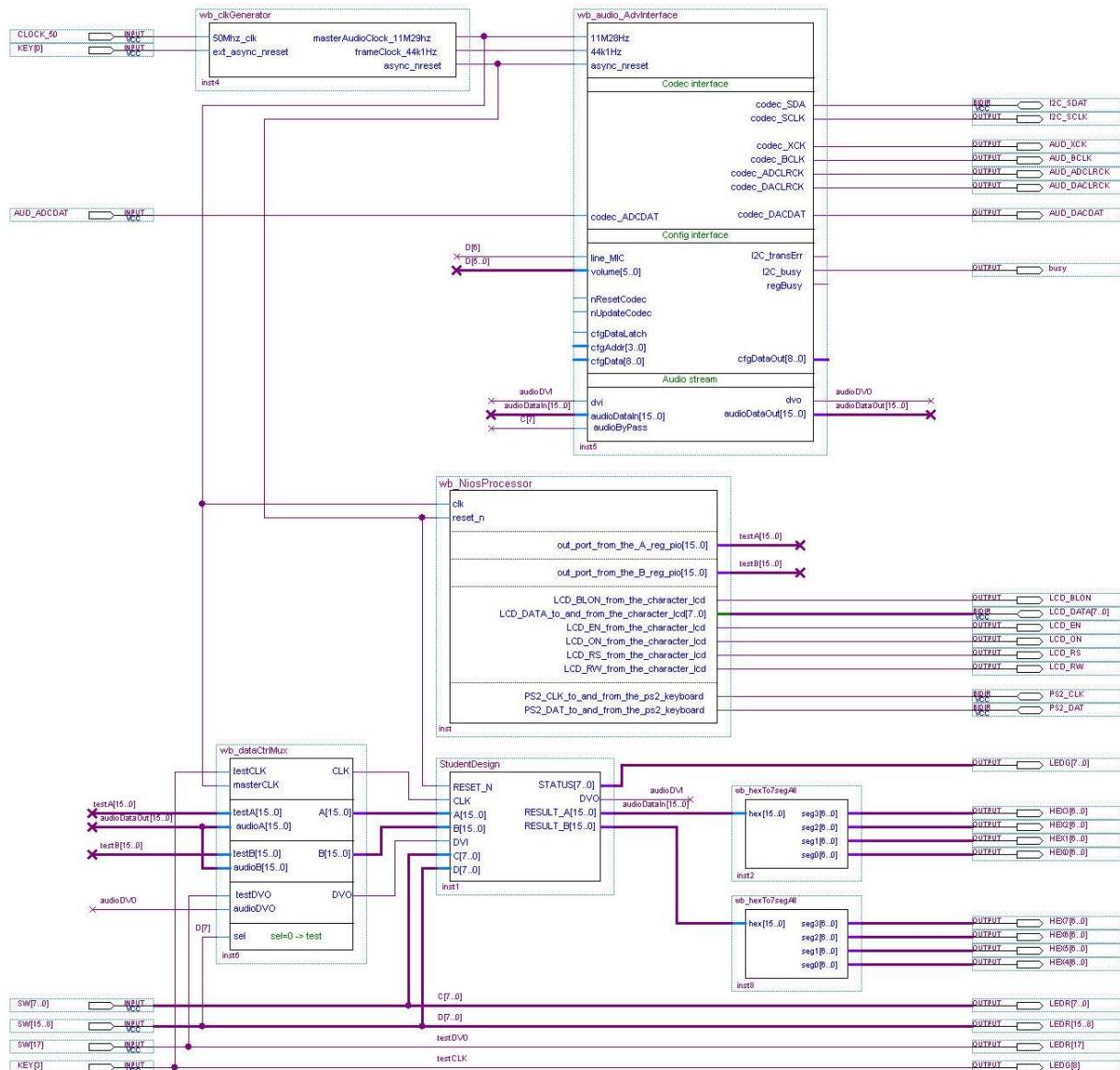
Hent *LAB3.ZIP* fra Blackboard og legg filene i en egen katalog slik det er forklart i kapittel 2 Absoluttverdikrets. Start Quartus II, åpne prosjektet *Lab3.qpf* og deretter filen *Lab3.bdf* (se kapittel 2

Absoluttverdikrets for detaljert fremgangsmåte).

Toppnivåskjemaet i *Lab3.bdf* inneholder nå vesentlig flere moduler enn i lab 2 (se figur 3-7). Det er ikke så lett å se detaljene på denne figuren (det blir bedre i Quartus på labben), men raskt forklart er de nye modulene øverst i skjemaet grensesnittet mot lyd-CODEC-en, samt en klokkegenerator som genererer de to klokkefrekvensene som trengs i CODEC-en (44,1 KHz og 11,29 MHz). Den raskeste klokken benyttes også som klokke i resten av systemet.

Til venstre for blokken *StudentDesign* er det lagt inn en multiplexer. Denne gjør det mulig enten å kjøre systemet i lydmodus eller i testmodus. I testmodus påtrykker man data ved hjelp av tastaturet. *CLK* og *DVO* settes ved hjelp av trykknapper på DE2-kortet.

Gå inn i blokken *StudentDesign*. Her ligger det nå en ferdig ABS-modul, tilsvarende den dere laget på lab 2, samt en ny blokk *FACT*. Gå inn i *FACT*. Denne inneholder fire moduler. En av dem, *wb\_aComp\_factorFinder*, er helt ferdig. Dere må gjerne se på innholdet av den, men pass på å ikke lagre når dere lukker den igjen. Det er denne modulen som til syvende og sist bestemmer hvilken forsterkningsfaktor som skal benyttes. De andre modulene vil bli behandlet i oppgavene nedenfor.

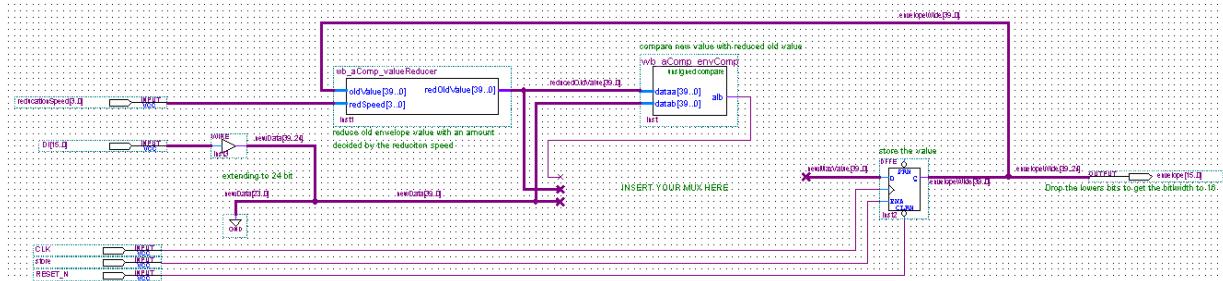


Figur 3-7: Skjemategning av toppnivået i FPGA-designet lab 3

## Laboppgave 2: Design av generisk multiplekser

I modulen *FACT* er det *wb\_aComp\_envShaper* som finner omhylningskurven rundt absoluttverdien av det innkommende lydsignalet. Gå inn i denne modulen. Skjemaet i figur 3-8 dukker opp. Her mangler det helt tydelig en komponent. Det som mangler er en 40-bits 2-1 multiplekser. Denne velger enten en reduserte versjon av den gamle omhylningskurveverdien eller den nye verdien som har kommet inn, avhengig av hvilken som er størst. Det er *wb\_aComp\_valueReducer* som foretar reduseringen. Hvor mye det reduseres med mellom hvert sample, bestemmes som nevnt av signalet *reductionSpeed*. Det kan kanskje virke litt

merkelig at vi trenger så mye som 40 bit i et design som jobber med 16-bit lydsignal. Dette skyldes imidlertid at vi må øke bitbredden en del steder for å få tilstrekkelig kvalitet på sluttresultatet etter diverse divisjoner og multiplikasjoner.



Figur 3-8: Skjemategning av wb\_aComp\_envShaper

Dere skal designe MUXen som mangler, og den skal designes så generell at den kan benyttes også til andre bitbredder enn 40. Dette er nyttig i forbindelse med gjenbruk av komponenter i ulike deler av designet. Senere trenger dere kanskje en 16-bit 2-1 multiplekser. Da kan dere bruke den samme modulen uten å måtte gjøre designjobben på nytt.

Velg *File -> New* og deretter *Design Files -> Block Diagram/Schematic File*. I det tomme skjemaet som dukker opp henter dere inn komponentene dere trenger til den 1-bit 2-1 multiplekseren dere designet i forarbeidets 0. Hent også tre inngangspinner og en utgangspinne. Navngi inngangspinnene

```
data0[busWidth-1..0]
data1[busWidth-1..0]
sel
```

Navngi utgangspinnen

```
dataOut[busWidth-1..0]
```

Datapinnene har nå en bussbredde som avgjøres av bredden på det inngangssignalet som kobles til (spesifiseres på det hierarkiske nivået over). Bruk busstegner (Orthogonal Bus Tool) til å koble datapinnene til portene i multiplekseren. Bruk nettegneren (Orthogonal Node Tool) til å koble valgsignalet *sel* til portene i multiplekseren. De bussene dere nå har tegnet må også gies navn med bussbredder for å sikre at komponenten blir generisk. Busser knyttet til inn- og utgangspinner gis samme navn som pinnen. Interne signal gis navn av typen

```
internal0[busWidth-1..0]
```

For å gjøre multiplekseren generisk slik at den kan benyttes til alle bussbredder, må parameteren *busWidth* defineres og gies en standardverdi. Hent frem symbolvelgeren (høyreklikk inne i skjemaet og velg *Insert -> Symbol*) og velg *param under libraries -> primitives -> other*. Dobbeltklikk på boksen og endre *PARAMETER\_NAME* til *busWidth* og *<none>* til 8. Lagre komponenten under et egnet navn og sett den til å være *Top-Level Entity*. Kompiler komponenten og simuler i henhold til testplanen fra forarbeidet for å forsikre dere om at den virker som den skal. Rett opp eventuelle feil i designet. Se kapittel 2 Absoluttverdikrets for detaljer rundt kompilering og simulering.

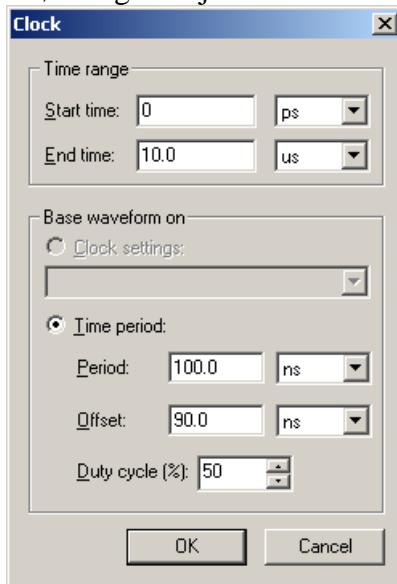
Når dere er sikre på at multiplekseren virker som den skal, kan dere lage et symbol for den nye komponenten. Sørg for å ha skjemaet for multiplekseren fremme. Velg *File -> Create / Update -> Create Symbol Files for Current File*. Det blir nå generert et symbol for multiplekseren som kan hentes inn i andre skjema.

Gå tilbake til skjemaet *wb\_aComp\_envShaper* igjen. Hent inn den nye multipleksermodulen dere nettopp har designet. Dere finner den under *Project* i symbolvelgeren. Koble til bussene i designet til inngangene og utgangene på kretsen. *reducedOldValue[39..0]* skal kobles til *data0[busWidth-1]*. *newDataValue[39..0]* skal kobles til *data1[busWidth-1]*. **Pass på at dette blir riktig.** Valgsignalet *sel* skal kobles til utgangen *alb* på *wb\_aComp\_envComp*. I tillegg må det nå spesifiseres konkret hvilken verdi for *busWidth* som skal benyttes for denne spesifikke instansen av multiplekseren. Dette gjøres i en egen parameterboks som automatisk dukker opp når dere setter inn komponenten. Dersom den ikke vises så kan dere få den fram ved å velge *View -> Show Parameter Assignments*. Endre *Value* til *40* i denne boksen. Merk at hvis dere senere ønsker å benytte multiplekseren i en annen del av designet (eller et annet design) der det bare trengs, for eksempel, 16 bit, så er alt dere trenger å gjøre å hente inn den samme komponenten og sette *Value* til *16*.

### Laboppgave 3: Implementering av positiv flankedetektor

Dere skal nå tegne inn den positive flankedetektoren dere designet i forarbeidets 0. Gå inn i modulen *edgeDetector*. Dere kommer da inn i et skjema som bare inneholder inn- og utgangspinner. Tegn flankedetektoren og koble til inn- og utgangspinner. Kompiler modulen.

Dere skal nå simulere kretsen med stimuli som i figur 3-6. Når dere skal sette opp et klokkesignal i *Waveform Editor*, velger dere signalet *CLK* ved å trykke på dette under *Name*. Trykk så på knappen *Overwrite Clock* (X) og dialogboksen i figur 3-9 dukker opp. Gitt at *Grid Size* er satt til 50 ns (*Edit -> Grid Size...*), så angir dialogboksen fornuftige verdier. Dette vil gi en klokke som endres 10 ns før en gridlinje.



Figur 3-9: Dialogboks for setting av klokkesignal.

- ❖ Forklar hvorfor et oppsett med en slik offset gjør det mulig å påtrykke andre signaler slik at det kan stemme overens med oppførselen i en virkelig sekvensiell krets.  
HINT 1: dere kan bare endre inngangsverdier nøyaktig på gridlinjer.  
HINT 2: i virkelige kretser tar det en viss tid fra en vippe klokkes og til tilhørende endring vises på inngangen av neste vippe.

Simuler kretsen for å forsikre dere om at den virker som den skal.

## Laboppgave 4: Implementering av tilstandsmaskin

Gå inn i modulen *factSTM* og tegn inn tilstandsmaskinen dere designet i forarbeidets 0. Kompiler modulen og simuler i henhold til testplanen fra forarbeidet.

## Laboppgave 5: Simulering av studentdesign

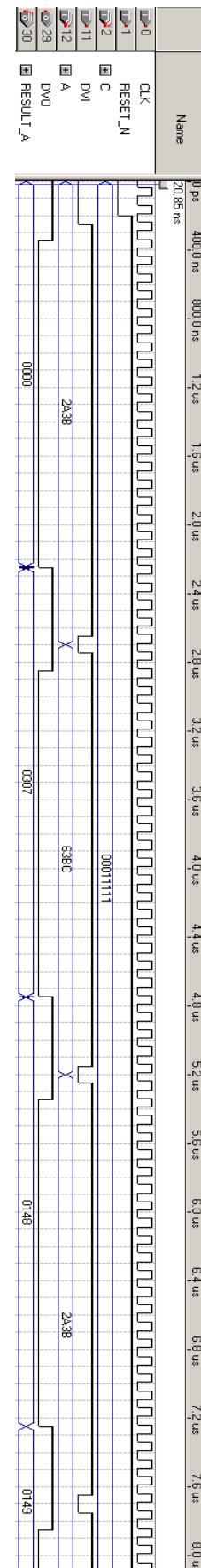
Gå tilbake til modulen *StudentDesign*. I denne er forsterkningsfaktoren som beregnes lagt ut på utgang *RESULT\_A*. I lab 4 vil den bli brukt som inngangssignal til multiplikatoren dere skal designe der. Nå skal dere imidlertid simulere hele *StudentDesign* for å forsikre dere om at de ulike enkeltmodulene dere har designet inne i FACT faktisk fungerer korrekt sammen, og sammen med ABS-modulen.

Kompiler modulen. Dobbeltklikk deretter på filen *StudentDesign\_sim.vwf* i *Project Navigator* (opp til venstre i Quartus II). Dere vil da få opp en fil som ligner på figur 3-10. Forskjellen er at utgangene DVO og *RESULT\_A* ikke har fått verdier enda (dette er indikert med X), og at bølgeformfilen også inneholder utgangene *STATUS* og *RESULT\_B*. *STATUS* og *RESULT\_B* er ikke tilkoblet noe, så de kan vi ignorere i simuleringen. Simuleringstiden er dessuten en god del lengre enn i figur 3-10.

Verdien til *factor* som vises på *RESULT\_A* er et heksadesimalt tall mellom 01,00 og 7F,FF (1,0 og 127,996 desimalt). Det er altså egentlig et komma midt i resultatet. Inngangssignalet A, er lydsamler på to-komplementformat som med 16 bit kan ha verdier mellom 8000 (-32 768 desimalt) og 7FFF (+32 767 desimalt).

Simuler *StudentDesign* med filen *StudentDesign\_sim.vwf*. Kontroller at resultatet blir som i figur 3-10. Hvis så ikke skjer må dere sjekke enkeltmodulene deres inne i FACT igjen.

- ❖ Forklar de ulike verdiene dere får ut på *RESULT\_A*. Hva betyr det at resulterende factor er 03,07 (heksadesimalt) når man påtrykker 2A3B? Når man påtrykker den vesentlig høyere verdien 63BC, reduseres factor til 01,48. Hvorfor det? Når man så igjen påtrykker 2A3B, så øker factor bare til 01,49. Hvorfor det?
- ❖ C-registeret er i figur 3-10 satt til 00011111. Eksperimenter med lavere reduksjonshastigheter. Hva skjer da, og hvorfor? Pass på at ABS-modulen alltid er aktivisert (dvs. at C[4] = 1).



Figur 3-10: Fasit for simulering av *StudentDesign*

## Laboppgave 6: Implementasjon og nedlasting av design til FPGA

Når simuleringsresultatene for *StudentDesign* viser korrekt oppførsel, skal dere kompilere det komplette designet (*Lab3.bdf*). Konfigureringsfilen som da genereres skal dere så laste ned i FPGA-en. Se kapittel 2

Absoluttverdikrets for detaljer rundt dette.

Hent et DE2-kort, tastatur, USB-kabel, batterieliminator og mikrofon (hodetelefoner trengs ikke denne gangen). Koble sammen utstyret på samme måte som i lab 2. Koble mikrofonen til DE2-kortets *MIC* inngang (rosa plugg i bakkant av kortet).

### **Husk å bruke armlenke når dere håndterer DE2-kortet!**

Koble opp og last designet ned på DE2-kortet. Det er viktig å sette opp C-registeret og D-registeret riktig. Figur 3-11 gir en bitvis oversikt over hva de to registrene styrer og hvilken skyvebryter på kortet som styrer hvilket bit. De enkelte bitene er forklart i tabell 3-2 og 3-3.

C-Registeret									
7	6	5	4	3	2	1	0		
SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0		
PASS	MUL	FACT	ABS	Reduction Speed					

D-Registeret							
7	6	5	4	3	2	1	0
SW15	SW14	SW13	SW12	SW11	SW10	SW9	SW8
MODE	SRC	Volume					

Figur 3-11: Bitvis oversikt over C- og D-registeret

C[6] og C[5] er ikke implementert i denne versjonen av signalbehandlingssystemet. De vil bli inkludert og brukt i lab 4. FACT-modulen er alltid aktivert (og multiplikatoren finnes ikke enda).

Tabell 3-2: C-Registeret

Navn	Betydning
PASS	Bestemmer om lyden inn fra CODEC-en skal sendes direkte tilbake ut på CODEC-en igjen (PASS=1) eller om det er lyden som genereres i signalbehandlingen som skal sendes til CODEC-en.
MUL	Aktiverer multiplikatoren (MUL=1) i signalbehandlingen.
FACT	Aktiverer FACT-modulen (FACT=1) i signalbehandlingen.
ABS	Aktiverer ABS-modulen (ABS=1) i signalbehandlingen.
Reduction Speed	Bestemmer reduksjonshastigheten, dvs. hvor raskt envelopen i FACT-modulen følger fallende lydsignaler.

Tabell 3-3: D-Registeret

Navn	Betydning
MODE	Setter testbenken i lydmodus (MODE=1) eller testmodus.
SRC	Velger om det er mikrofoninngangen (SRC=1) eller linjeinngangen til kortet som skal være kilde for lyd inn til systemet.
Volume	Justering av lydvolumet ut til hodetelefonene. 111111: sterkeste volum, 000000: svakeste volum

Resultatet ut av FACT-modulen, dvs. forsterkningsfaktoren, vises i dette designet på sjusegmentdisplay HEX3 – HEX0. Det står også nå et komma mellom de to midterste hex-verdiene (mellan HEX2 og HEX1).

- ❖ Snakk inn i mikrofonen med forskjellige lydvolum og eksperimenter med ulike reduksjonshastigheter på register C[3..0]. Observer hvordan forsterkningsfaktoren endres og forklar hvorfor den oppfører seg slik den gjør.

## Avslutning

- Koble fra batterieliminator, tastatur, USB-kabel og mikrofon. Sett tilbake utstyret.
- Avslutt Quartus II.

# 4 Laboratorieøving

## *Innledning*

### **Formål**

Å lage en multiplikator til lydbehandlingssystemet. Multiplikatoren lages med skjemategninger og maskinvarebeskrivende språk. Med multiplikatoren vil vi endelig få det komplette signalbehandlingssystemet slik det fremkommer på innsiden av omslaget.

### **Hensikt**

- Se mer på inndeling av et system i en utførende (utførende enhet, datapath) og en styrende (styreenhet, control unit) del.
- Få enkel innføring i maskinvarebeskrivende språk.
- Analysere og forstå en ”større” digital krets.

### **Bakgrunn**

- Daniel Gajski: ”Principles of Digital Design”, kapittel 2.7, 6.6, 6.7
- Enkel innføring i VHDL i ”10 Vedlegg: VHDL – En introduksjon”

### **Hva som skal gjøres**

Dere skal sette dere inn i en gitt maskinvarebeskrivelse av tilstandsmaskinen i multiplikatorens styreenhet, og fullføre denne. Eventuelle nye signaler skal legges til i tilstandsmaskinen og kobles til eksisterende enheter. En enhet i den utførende enheten skal lages ved hjelp av ferdig funksjonalitet i designverktøyet Quartus II.

## Modulbeskrivelse

### Teori

Ved multiplikasjon av to tall med ”papirmetoden” spiller det ingen rolle hvilket tallsystem det opereres med. Man har to operander, multiplikand MAND og multiplikator MTOR, som skal multiplisieres med hverandre. Man tar siffer for siffer i MTOR og multipliserer med MAND. Man starter med minst signifikante siffer og skriver delsvaret rett under MAND. Man går så til neste siffer i MTOR som multiplisieres med MAND. Delsvaret her skrives under forrige delsvare forskjøvet en plass til venstre. Slik fortsetter man til og med mest signifikante siffer. Man summerer alle delsvarene og får svaret DO (Data Out) på multiplikasjonen. Legg merke til at antall siffer i svaret kan maksimum være summen av antall siffer i MAND og MTOR.

Vi vil her bruke binære tall på to-komplementsform. I det binære tallsystemet er det mulig å forenkle denne algoritmen noe:

- Ved utregning av alle delsvarene skal MAND multiplisieres med et siffer i MTOR. Siden det her er totallsystemet vil det si at MAND skal multiplisieres med enten 0 eller 1. Det vil si at alle delsvar er enten 0 eller det samme som multiplikanden MAND.
- Eneste unntaket til a) er når MTOR er negativ. Da vil den mest signifikante biten i MTOR ha en negativ vekt. Derfor vil en for MTOR's MSB når MTOR er negativ multiplisere med -1 og ikke 1. Dette kan vi gjøre ved å ta to-komplementet av MAND før denne legges til.
- Summering av alle delsvar kan utføres etter hvert som delsvarene er klare.

Tre eksempler er vist i Figur 4-1. Ett med positive tall, ett med negativ multiplikator, og ett med negativ multiplikand.

Sign ext	0 1 1 0 1	(13)	Multiplikand MAND
	x 0 1 0 1 1	(11)	Multiplikator MTOR
	0 0 0 0 0 0 1 1 0 1 1	$2^0 * 1 * 13$	
	0 0 0 0 0 1 1 0 1 0	$2^1 * 1 * 13$	
	0 0 0 0 1 1 0 0 0 0	$2^2 * 0 * 13$	
	0 0 0 1 1 0 1 0 0 0	$2^3 * 1 * 13$	
	0 0 1 0 0 0 0 0 0 0	$-2^4 * 0 * 13$	
0 0 1 0 0 0 0 0 0 1 1 1 1	(143)	Resultat DO	

Sign ext	0 1 1 0 1	(13)	Multiplikand MAND
	x 1 0 1 0 1	(-11)	Multiplikator MTOR
	0 0 0 0 0 0 1 1 0 1 1	$2^0 * 1 * 13$	
	0 0 0 0 0 0 0 0 0 0	$2^1 * 0 * 13$	
	0 0 0 0 0 1 1 0 1 0 0	$2^2 * 1 * 13$	
	0 0 0 0 0 0 0 0 0 0	$2^3 * 0 * 13$	
1 1 0 0 1 1 0 0 0 0	(-143)	Resultat DO	

Sign ext	1 0 1 0 1	(-11)	Multiplikand MAND
	x 0 1 1 0 1	(13)	Multiplikator MTOR
	1 1 1 1 1 1 1 0 1 0 1	$2^0 * 1 * -11$	
	0 0 0 0 0 0 0 0 0 0	$2^1 * 0 * -11$	
	1 1 1 1 0 1 0 1 0 0 0	$2^2 * 1 * -11$	
	1 1 1 0 1 0 1 0 0 0	$2^3 * 1 * -11$	
0 0 0 0 0 0 0 0 0 0	(-143)	Resultat DO	

a) To positive tall

b) Negativ multiplikator

c) Negativ multiplikand

Figur 4-1: Multiplikasjon på papirmetoden

Algoritmen kan uttrykkes i såkalt pseudo-kode som følger:

```

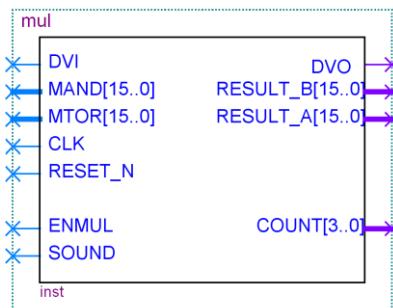
DO := 0
bit_teller := MSB bitposisjon
signext MAND

for ( "Alle bitposisjoner (fra 0 til og med MSB bitposisjon) i MTOR" )
    if ( "LSB av MTOR er lik 1" ) then
        if ( bit_teller lik null ) then
            DO := DO + ~MAND + 1
        else then
            DO := DO + MAND
        endif
    endif
    bit_teller := bit_teller - 1;
    Shift left MAND en bit-posisjon
    Shift right MTOR en bit-posisjon
end for

```

## Grensesnitt

Kretsen som er målet i denne labben har følgende grensesnitt:

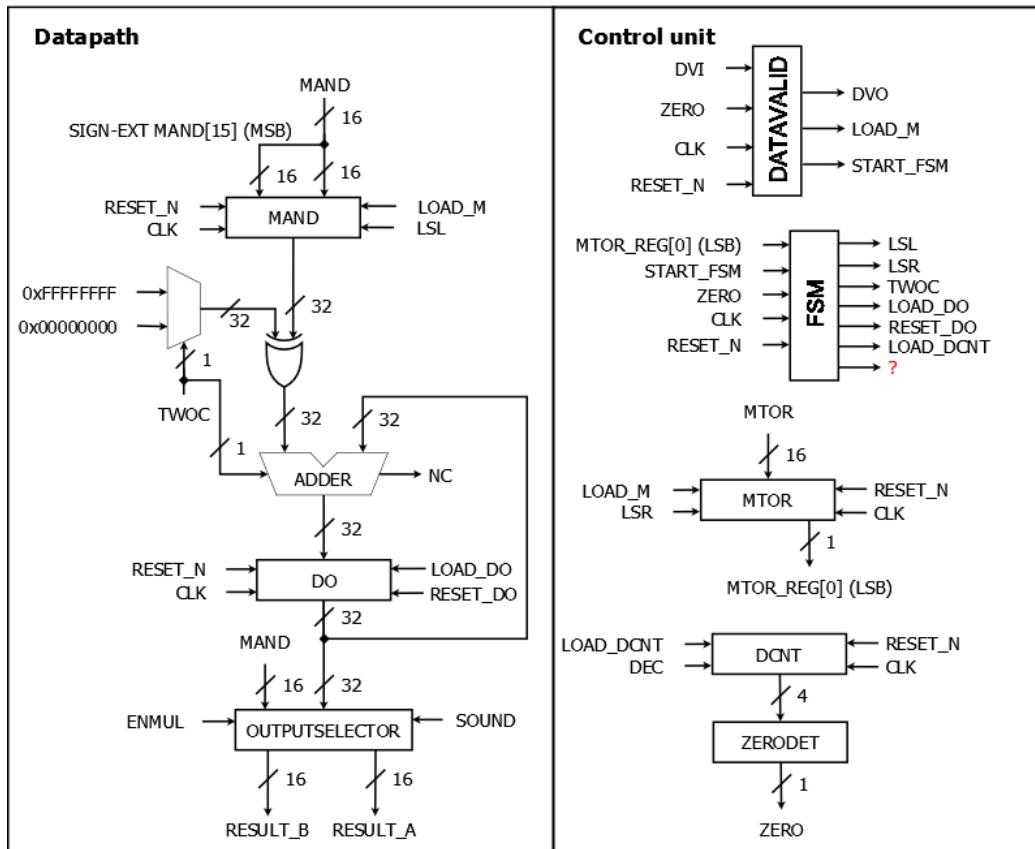


Figur 4-2: Multiplikatormodul

Tabell 4-1 Multiplikator grensesnitt

Navn	# Bit	Retning	Verdi	Betydning
MAND	16	Inn		Multiplikand
MTOR	16	Inn		Multiplikator
RESULT_A	16	Ut		16 LSB av resultatet
RESULT_B	16	Ut		16 MSB av resultatet
COUNT	16	Ut		Verdi på teller DCNT
DVI	1	Inn		Kontrollbit
			0	MAND og MTOR er ugyldig
			1	MAND og MTOR er gyldig
DVO	1	Ut		Kontrollbit:
			0	RESULT er ugyldig
			1	RESULT er gyldig
ENMUL	1	Inn		Kontrollbit:
			0	RESULT_A = MAND
			1	RESULT_BA = MAND * MTOR
SOUND	1	Inn		Kontrollbit (overstyrer ENMUL):
			0	Normal utgang, se ENMUL.
			1	RESULT_A = ( MAND * MTOR )[24..9] RESULT_B = 0x0000

## Realisering



Figur 4-3: Blokkskjema for 16-bits multiplikator

Multiplikatoren som skal benyttes i labben (Figur 4-3) er relativt enkel, og benytter de samme konseptene som papirmetoden. Papirmetoden ble illustrert som en 5-bit multiplikator for å forenkle beskrivelsen. Multiplikatoren som benyttes i labben er 16-bit, og håndterer også negative tall (inngangsverdien kan da være i området  $-2^{15}$  til  $+2^{15}-1$ ).

Figur 4-3 er delt i to deler, en utførende enhet og en styreenhet. Den utførende enheten starter med et register MAND, som holder på multiplikanden. Videre går dataene gjennom en samling XOR porter til en kombinatorisk modul, ADDER. Denne modulen summerer disse dataene med dataene i DO. DO er et register som lagrer dataene på registerinngangen dersom LOAD\_DO er høy. Dataene ut fra DO er svarene fra mellomregningene og det endelige produktet.

Siden produktet fra multiplikatoren er 32 bit og resultatet vi ønsker ut skal være 16 bit, må en velge hvilke 16 bit av produktet en vil bruke. Dette tar OUTPUTSELECTION modulen seg av. Den har to kontrollbit som bestemmer hvilke bit av produktet som skal brukes i RESULT. ENMUL bestemmer om en skal bruke multiplikatoren i det hele tatt. Er signalet ikke satt høyt, vil RESULT\_A settes lik multiplikanden. Neste steg er å bestemme om en er i testmodus (SOUND = 0) eller lydmodus (SOUND = 1). I lydmodus settes RESULT\_A lik bit 24 til 9 av produktet. RESULT\_B settes lik den forstekningsfaktor som blir funnet i modulen FACT (se lab 3). I testmodus settes RESULT\_B lik de 16 mest signifikante bitene og RESULT\_A lik de 16 minst signifikante bitene i produktet.

Kontrolldelen av multiplikatoren (Controller i Figur 4-3) består av flere synkrone moduler. Øverst i figuren har dere modulen DATAVALID. Denne sender ut et signal START\_FSM og LOAD\_M når DVI går fra  $0 \rightarrow 1$ , dette starter tilstandsmaskinen, FSM. FSM står for Finite-State-Machine. Under multiplikasjonen tar FSM seg av å generere kontrollsinalene som skal til for korrekt oppførsel, se forarbeidets oppgave 4 for nærmere forklaring av FSM-modulens oppførsel. MTOR er et register som holder på multiplikatoren. Den minst signifikante biten til MTOR brukes som et kontrollsignal til FSM. DCNT er en firebits nedteller. ZERODET sjekker om telleren er null. Når multiplikasjonen er ferdig, har DCNT telt ned fra 15 til 0 og ZERO settes høy. DATAVALID modulen setter da DVO høy.

## Peak-meter

Som nevnt er det lagt til en ekstra enhet her i lab 4. Dette er et "peak-meter" som gir en stolperepresentasjon av top-verdien på det digitale lydsignalet som sendes ut til codecen. Peak-meteret aktiviseres ved å aktivisere skyvebryter SW16 (PEAKEN = 1). Lysdiodene over skyvebryterne slutter da å vise verdien til den enkelte skyvebryter, og fungerer som peak-meter isteden.

Peak-meteret benytter seg av absoluttverdien til signalet. Det spesielle tilfellet med maks negativ verdi 0x8000 løses ved at denne verdien blir tolket som 0x7FFF av peak-meteret. Dermed begrenses absoluttverdien til å gå fra 0x0000 til 0x7FFF. Peak-meteret tar ut de 5 mest signifikante bitene. For hver tallverdi høyere for disse, tennes en ny diode. Den mest signifikante biten (fortegnsbiten) er alltid 0.

LEDR0 tennes for verdier høyere eller lik 0x0000 (alltid tent)

LEDR1 tennes for verdier høyere eller lik 0x0800

LEDR2 tennes for verdier høyere eller lik 0x1000

LEDR3 tennes for verdier høyere eller lik 0x1800

LEDR4 tennes for verdier høyere eller lik 0x2000

OSV...

LEDR15 tennes for verdier høyere eller lik 0x7800

LEDR17 indikerer "klipping". Klipping vil si at vi har nådd maks verdi og vi "klipper" toppen av kurven vi prøver å representer digitalt. Klipping fører til forvrengning av lyden og er særdeles uønsket i de fleste audio-sammenhenger.

Meteret har også en "sakte fall" egenskap som gjør at meteret går umiddelbart opp, men faller sakte av. Dette er på samme måte som for envelopedektoren. Klipping-indikatoren har en egen heng-tid, slik at en klipping, som kan ha en varighet på bare et lydsample (1/44100 sekund) blir indikert i 1 sekund etter at klippingen har skjedd. 1/44100 sekund ville vært alt for raskt for øyet til å oppfatte.

## Forarbeid

### Forarbeidsoppgave 1: Blackboard

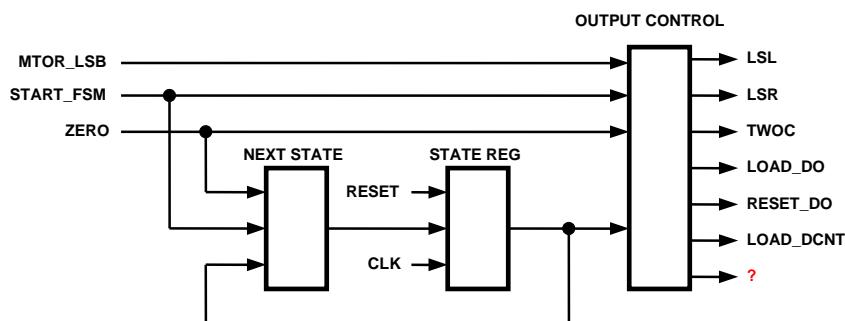
Les lab-informasjonen på Blackboard, og møt opp på lab-forelesningen.

### Forarbeidsoppgave 2: Multiplikatorens virkemåte

Les hele multiplikator-kapitlet, sett dere godt inn i multiplikatorens virkemåte.

### Forarbeidsoppgave 3: Manglende signal

- Finn hvilket signal som mangler i tilstandsmaskinen (merket med ? i Figur 4-4). Legg til dette signalet i Tabell 4-2 og på Figur 4-4.



Figur 4-4: Multiplikatorens tilstandsmaskin (FSM)

Figur 4-4 viser tilstandsmaskinen til multiplikatoren. NEXT STATE og OUTPUT CONTROL er kombinatoriske kretser, mens STATE REG er et 1-bits register. Signalene er forklart i Tabell 4-2.

Tabell 4-2: Tilstandsmaskinens grensesnitt

Navn	# Bit	Retning	Betydning
MTOR LSB	1	Inn	Minst signifikante bit i multiplikatorregisteret (MTOR)
START_FSM	1	Inn	Startsignal for tilstandsmaskinen
ZERO	1	Inn	Forteller om telleren har kommet ned til 0 (DCNT = 0)
LSL	1	Ut	Venstreskift for multiplikandregisteret (MAND)
LSR	1	Ut	Høyreskift for multiplikatorregisteret (MTOR)
TWOC	1	Ut	Ta toerkomplement av multiplikanden
LOAD_DO	1	Ut	Last addererens utgang inn til Data ut-registeret (DO)
RESET_DO	1	Ut	Reset Data ut-registeret (DO = 0x00000000)
LOAD_DCNT	1	Ut	Last 15 (DCNT = 0b1111) i telleren (down-count)
	1	Ut	

Tips: Finn ut hvilket kontrollsignal som ikke har noen kilde i Figur 4-3.

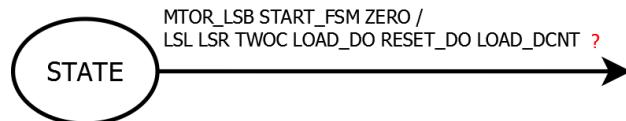
## Forarbeidsoppgave 4: Fullførelse av multiplikatorens tilstandsmaskin

I denne oppgaven skal dere vise at dere forstår kretsens virkemåte ved å identifisere et manglende signal, og fullføre et tilstandsdiagram ved å tilordne riktige utgangsverdier på deler av tilstandsdiagrammet.

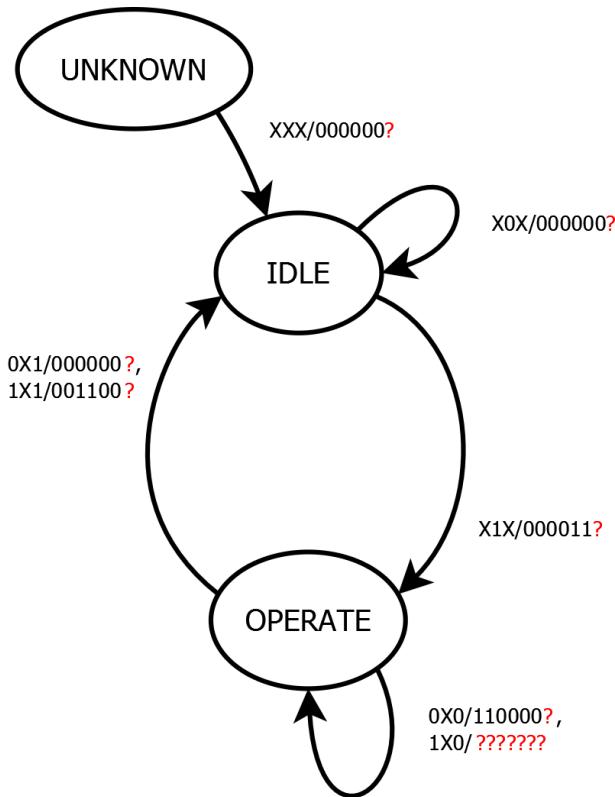
- ❖ Legg til navnet på signalet som ble funnet i forarbeidsoppgave 3 sist i notasjonen for tilstandsdiagrammet (Figur 4-5) og fyll ut verdiene til dette signalet i tilstandsdiagrammet vist i Figur 4-6 (bytt ut alle **?** med 0 eller 1).

Om vi studerer blokkskjemaene i Figur 4-3 og Figur 4-4, og tilstandsdiagrammet i Figur 4-6 får vi følgende virkemåte:

- Multiplikatorens tilstandsmaskin venter i tilstand IDLE inntil den får signalet START\_FSM. Den vil da gå inn i OPERATE tilstanden, hvor den vil utføre 16 iterasjoner av en løkke. Disse iterasjonene tilsvarer de enkelte addisjonene i papirmetoden. For hver iterasjon skiftes altså multiplikanden MAND et hakk til venstre (LSL), akkurat slik som vi flytter oss et hakk til venstre for hver addisjon i papirmetoden.
  - Høyreskiftingen (LSR) av multiplikatoren MTOR er kun en enkel måte å plukke ut den bitposisjonen i MTOR som benyttes i tilhørende iterasjon (en iterasjon for hvert enkelt bit i MTOR). Ettersom MTOR høyreskiftes vil den biten i MTOR som skal benyttes i kommende iterasjon alltid være LSB i MTOR. Igjen helt analogt til papirmetoden. Det midlertidige resultatet fra addisjonene ligger i Data Out-registeret (DO).
  - Det sjekkes om den til enhver tid minst signifikante biten i MTOR er 1 (MTOR\_LSB = 1), og dersom det er tilfelle utføres en addisjon. Dersom denne laveste biten er 0 (MTOR\_LSB = 0) utføres ingen addisjon, tilsvarende papirmetoden hvor vi adderte med 0b0000. For hver iterasjon telles DCNT ned, slik at vi avslutter addisjonene etter 16 iterasjoner (ZERO = 1).
  - Den siste addisjonen utføres på en spesiell måte for å ta hensyn til negative inngangsverdier til multiplikatoren. Dette kommer frem i tilstandsdiagrammets ZERO = 1 forgrening samt XOR-porten og mente-inngangen (carry in) til adderereren i skjemategningen. **Sammenlikn dette med hvordan vi tok to-komplement av et tall i lab 1 og 2.**
- ❖ I OPERATE tilstanden mangler det verdier for utgangssignalene når MTOR = 1 og ZERO = 0. Fyll ut disse (erstatt alle **?** med 0 eller 1).



Figur 4-5: Notasjon for tilstandsdiagrammet i Figur 4-6.



Figur 4-6: Tilstandsdiagram for tilstandsmaskinen (FSM'en) i Figur 4-4.

### Forarbeidsoppgave 5: VHDL-kode

I denne labben er tilstandsmaskinen ikke beskrevet med blokkskjema som tidligere, men ved hjelp av VHDL kode. VHDL er et maskinvarebeskrivende språk, se ”10 Vedlegg: VHDL – En introduksjon” for en enkel innføring i VHDL.

Med utgangspunkt i tilstandsdiagrammet fra forarbeidsoppgave 4, skal nødvendige endringer gjøres i VHDL koden for tilstandsmaskinen gitt i .

- ❖ Det manglende signalet funnet i forarbeidsoppgave 3 må legges til blant utgangssignalene og på flere steder i koden (hint: se tilstandsdiagrammet fra Figur 4-6).
- ❖ En blokk med kontrollsignaler har kun signalverdier 0, og er kommentert -- UKJENT. Denne blokken tilsvarer det ene tilfellet der utgangsverdiene er ukjente i OPERATE tilstanden i tilstandsdiagrammet (Figur 4-6). Disse signalene må settes til korrekte verdier.

```

-- "entity" definerer tilstandsmaskinens grensesnitt i form av innganger og utganger
entity fsm is
    port (clk      : in STD_LOGIC; -- 1 bit, system clock
          reset   : in STD_LOGIC; -- 1 bit, system reset, active high
          mtor_lsb : in STD_LOGIC; -- 1 bit, lsb from multiplicator register
          start_fsm: in STD_LOGIC; -- 1 bit, signal to start the fsm, active high
          zero    : in STD_LOGIC; -- 1 bit, zero detection from down counter
          );
    begin
        -- **** HER MANGLER ET UTGANGSSIGNAL SOM MAA LEGGES TIL ****
        lsl      : out STD_LOGIC; -- 1 bit, shift left command to multiplicand register
        lsr      : out STD_LOGIC; -- 1 bit, shift right command to multiplicator register
        twoc    : out STD_LOGIC; -- 1 bit, two's complement command to adder
        load_do : out STD_LOGIC; -- 1 bit, load command to output register
        reset_do : out STD_LOGIC; -- 1 bit, reset command to output register
        load_dcnt: out STD_LOGIC -- 1 bit, load command to down counter
    end fsm;
    -- "architecture" beskriver innholdet i en entity
    architecture fsm_arch of fsm is
        -- "constant" gir verdier til navn, i dette tilfellet navngis de to tilstandene IDLE og OPERATE
        constant IDLE : STD_LOGIC := '0';
        constant OPERATE: STD_LOGIC := '1';
        -- "signal" definerer signaler og registre i designet. I dette tilfellet trenger vi et 1-bit register for tilstanden
        signal state: STD_LOGIC; -- 1 bit state register
        -- Etter "begin" beskrives logikken i tilstandsmakininen
        begin
            -- Foelgende prosessdeklarasjon forteller at for hver endring i signalet "clk" trigges prosessen
            -- Denne prosessen styrer overgangen mellom tilstandene
            process (clk) is begin
                if (clk'event and clk = '1') then
                    if (reset = '1') then
                        state <= IDLE;                                         -- dersom vi har en positiv klokkeflanke...
                    elsif ((state = IDLE) and (start_fsm = '1')) then
                        state <= OPERATE;                                     -- dersom reset er 1, gaa til IDLE
                    elsif ((state = OPERATE) and (zero = '0')) then
                        state <= OPERATE;                                     -- osv...
                    elsif ((state = OPERATE) and (zero = '1')) then
                        state <= IDLE;
                    else
                        state <= IDLE;
                    end if;
                end if;
            end process;
            -- Denne prosessen setter tilstandsmaskinens kontrollsinaler avhengig av tilstanden og inngangssignalene
            process (state, start_fsm, zero, mtor_lsb) is begin
                -- Hver if-linje som innleder en blokk med kontrollsinaler tilsvarer forgreningsflyten i tilstandsdiagrammet
                -- Hver blokk med kontrollsinaler tilsvarer en blokk i tilstandsdiagrammet
                -- I tillegg kommer den siste blokken i denne prosessen, som for ryddighets skyld setter alle kontrollsinaler
                -- til null dersom ingen av de andre tilfellene skulle inntrefte
                -- En av blokkene med kontrollsinaler maa endres fullstendig slik det er beskrevet lengre ned
                -- I tillegg maa et ukjent kontrollsignal ges verdier i ALLE blokkene
                if ((state = IDLE) and (start_fsm = '1')) then
                    lsl      <= '0';
                    lsr      <= '0';
                    twoc    <= '0';
                    load_do <= '0';
                    reset_do <= '1';
                    load_dcnt <= '1';
                elsif ((state = OPERATE) and (zero = '0') and (mtor_lsb = '0')) then
                    lsl      <= '1';
                    lsr      <= '1';
                    twoc    <= '0';
                    load_do <= '0';
                    reset_do <= '0';
                    load_dcnt <= '0';
                elsif ((state = OPERATE) and (zero = '0') and (mtor_lsb = '1')) then
                    -- I FØLGENDE BLOKK SKAL KONTROLLSIGNALENE ENDRES (alle er foreløpig bare satt til 0):
                    -- *** BLOKK FOR ENDRING - START ***
                    lsl      <= '0'; -- UKJENT
                    lsr      <= '0'; -- UKJENT
                    twoc   <= '0'; -- UKJENT
                    load_do <= '0'; -- UKJENT
                    reset_do <= '0'; -- UKJENT
                    load_dcnt <= '0'; -- UKJENT
                -- *** BLOKK FOR ENDRING - SLUTT ***
                elsif ((state = OPERATE) and (zero = '1') and (mtor_lsb = '0')) then
                    lsl      <= '0';
                    lsr      <= '0';
                    twoc    <= '0';
                    load_do <= '0';
                    reset_do <= '0';
                    load_dcnt <= '0';
                elsif ((state = OPERATE) and (zero = '1') and (mtor_lsb = '1')) then
                    lsl      <= '0';
                    lsr      <= '0';
                    twoc    <= '1';
                    load_do <= '1';
                    reset_do <= '0';
                    load_dcnt <= '0';
                else
                    lsl      <= '0';
                    lsr      <= '0';
                    twoc    <= '0';
                    load_do <= '0';
                    reset_do <= '0';
                    load_dcnt <= '0';
                end if;
            end process;
        end fsm_arch;
    end;

```

Figur 4-7: VHDL kode for tilstandsmaskinen

## Forarbeidsoppgave 6: Testplan for tilstandsmaskinen

Det er viktig å forstå systemet når man skal simulere. Når det påtrykkes stimuli må man på forhånd ha klart for seg hvilken respons som forventes ellers blir hele simuleringen verdiløs. Spesielt gjelder dette kontrolldelen og tilstandsmaskinen (FSM) i denne. Prøv derfor å forstå hvilke verdier dere kan vente dere ut fra tilstandsmaskinen (FSM), ut fra hvilken tilstand multiplikatoren er i og hva den får av inngangssignaler.

- ❖ Sett opp en testplan for tilstandsmaskinen (FSM). Tips: Sørg for at testplanen dere lager går gjennom alle tilstandene i tilstandsdiagrammet på Figur 4-6. Kan dere komme på ukjente tilstander tilstandsmaskinen kan være i som krever at vi må håndtere dette også?

## Forarbeidsoppgave 7: Testplan for multiplikatoren

- ❖ Lag en plan for toppnivå testing av multiplikatoren.

Planen skal inneholde forskjellige inngangsverdier og svaret disse skal gi. Det er gitt et eksempel på en testplan i Tabell 4-3 under. Finn ut hvilke tall dere vil ha med for å kontrollere at multiplikatoren fungerer (positive og negative tall bør bl.a. være med). Det lønner seg å skrive tallene i hexadecimal form, slik at de blir lettere å sammenligne med tallene som framkommer på DE2-kortet. Multiplikatoren har to 16-bit innganger, og svaret blir på 32-bit.

*Tabell 4-3: Testplan for multiplikatoren*

MAND	MTOR	PRODUCT	RESULT_B	RESULT_A
0xFFFF (-1)	0x9765 (-26779)	0x0000689B (26779)	0x0000	0x689B
...	...	...	...	...

## Forarbeidsoppgave 8: TWOC-signalet

- ❖ Forklar hvordan TWOC-signalet (Tabell 4-2 og Figur 4-3) benyttes for å håndtere negative tall:
  - **Hva er det som skjer** i siste iterasjon av multiplikasjonen som gjør at multiplikatoren gir ut korrekt produkt når multiplikator har en negativ verdi (toerkomplement form)? **Hvorfor** gir dette korrekt svar?
  - I tilstandsmaskinen sjekkes det bare om **multiplikatorens (MTOR)** fortegn er negativt før TWOC-signalet eventuelt gis. Hvorfor fungerer multiplikasjonen likevel dersom bare **multiplikanden (MAND)** er negativ?

## Labarbeid

Når DE2-Kortet skrus på kommer det en  
**KRAFTIG TONE PÅ LINE OUT UTGANGEN**

med mindre sw17 er satt. Vent derfor med å koble til hodetelefonene til dere har lastet ned deres eget design. Lyden vil kunne komme tilbake dersom dere skrur kortet av og på. Ta derfor alltid av hodetelefonene før spenningen skrus på.

## Oversikt

Labarbeidet i multiplikatorlabben består av 8 oppgaver:

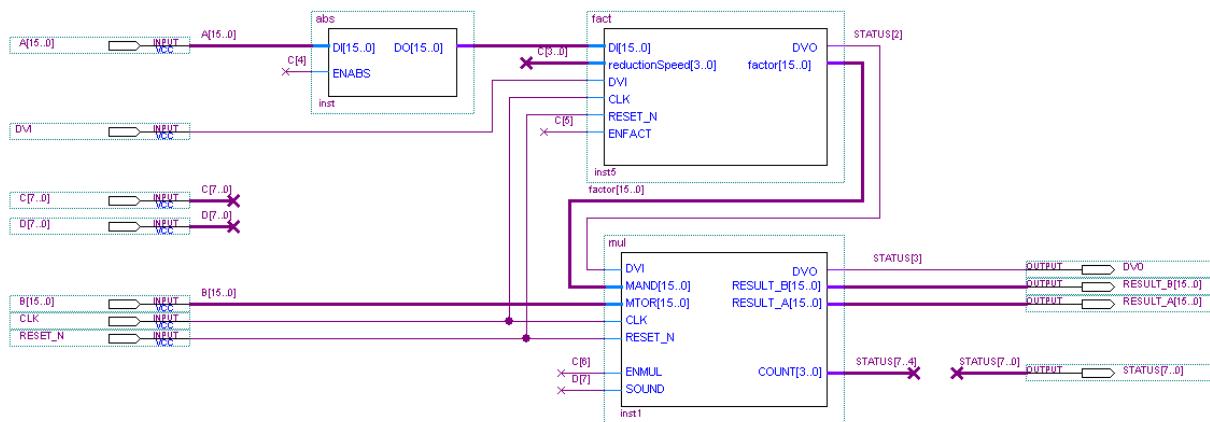
Oppgave	Innhold
1	Klargjøring
2	Adderer
3	Skriv inn VHDL-kode
4	Generere nytt symbol for FSM
5	Simulering av multiplikator
6	Implementasjon og nedlasting av design til FPGA
7	Lyd
8	Avslutning

## Laboppgave 1: Klargjøring

- Les lab-infoen på Blackboard.

Hent *LAB4.ZIP* fra Blackboard og legg filene i en egen katalog slik det er forklart i kapittel 2 Absoluttverdikrets. Start Quartus II, åpne prosjektet *Lab4.qpf* og deretter filen *Lab4.bdf* (se kapittel 2 Absoluttverdikrets for detaljert fremgangsmåte).

Gå videre inn i modulen *StudentDesign*. Her finner dere modulen *ABS*, modulen *FACT* og modulen *MUL*. Disse er ferdig sammenkoblet (figur 4-8). *ABS* og *FACT* er helt ferdige. I *FACT* er nå inngangen *ENFACT* implementert. Når *ENFACT* (kontrollregisterbit *C[5]*) er høyt, vil *FACT*-modulen generere sine normale *factor*-verdier. Når *ENFACT* er lavt, vil modulen sette det som står på dens datainngang (*DI*) ut på utgangen *factor*. Tilsvarende vil den rute sin *DVI* direkte ut på sin *DVO*.



Figur 4-8: StudentDesign med ABS-, FACT- og MUL-modul tilkoblet

Klikk dere nedover i *MUL*-modulen og se nærmere på de forskjellige submodulene. Sammenlign med figurene fra modulbeskrivelsen og forarbeidet tidligere i dette kapitlet og prøv å få en forståelse for hvordan multiplikatoren fungerer.

- ❖ Hva må skje på multiplikatorens innganger dersom tilstandsmaskinen skal gå ut av *IDLE*-tilstanden?

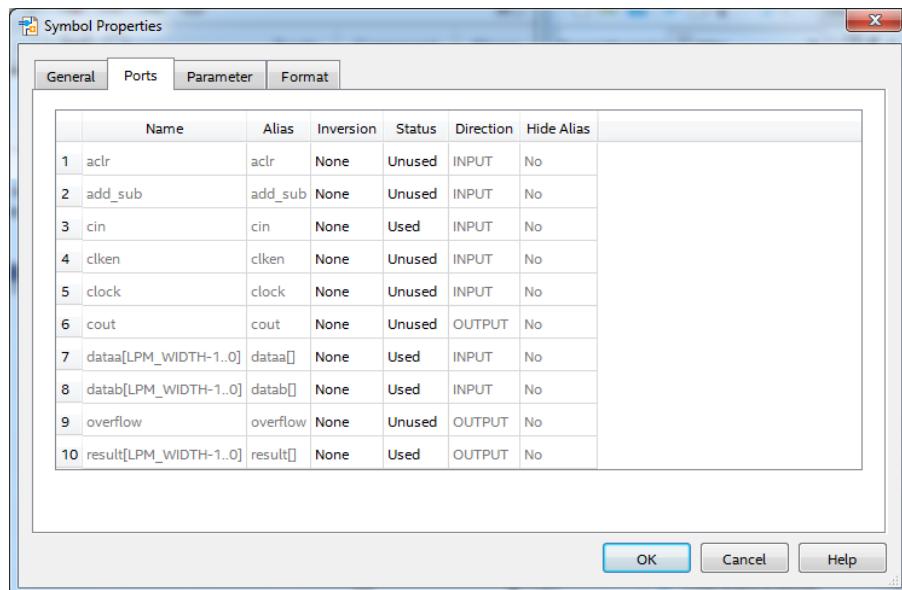
## Laboppgave 2: Adderer generert med Megafuctions

Gå inn i den utførende enheten til multiplikatoren. Ved nøyere ettersyn vil dere se at det mangler en komponent, addereren.

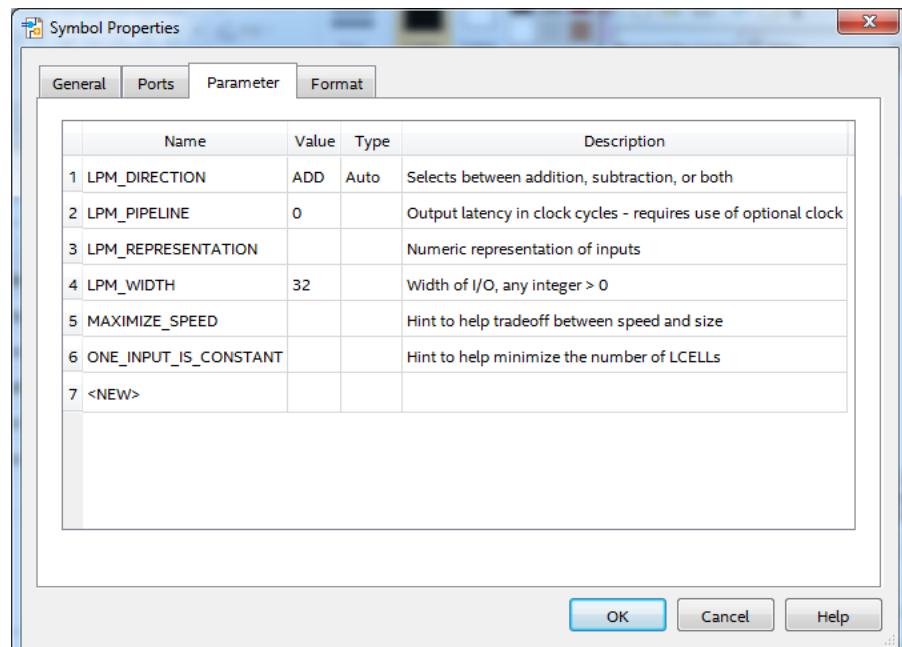
- ❖ Det skal nå lages en binær adderer ved å sette inn en såkalt *Megafunction*.
  - Høyreklikk i skjemaet og velg *Insert -> Symbol* fra hurtigmenyen.
  - Åpne biblioteket fra altera og deretter *Megafunctions -> Arithmetic*

Det kommer da opp en liste med ulike aritmetiske komponenter som man kan tilpasse sitt behov og inkludere i designet sitt, slik det er vist i figur 4-11.

- Velg *lpm\_add\_sub* og trykk *OK*
- Høyreklikk på symbolet etter at det er kommet inn i skjemaet og deretter tilpasser porter (Figur 4-9) og parametre (Figur 4-10). Dobbeltklikk på en celle for å endre verdi.

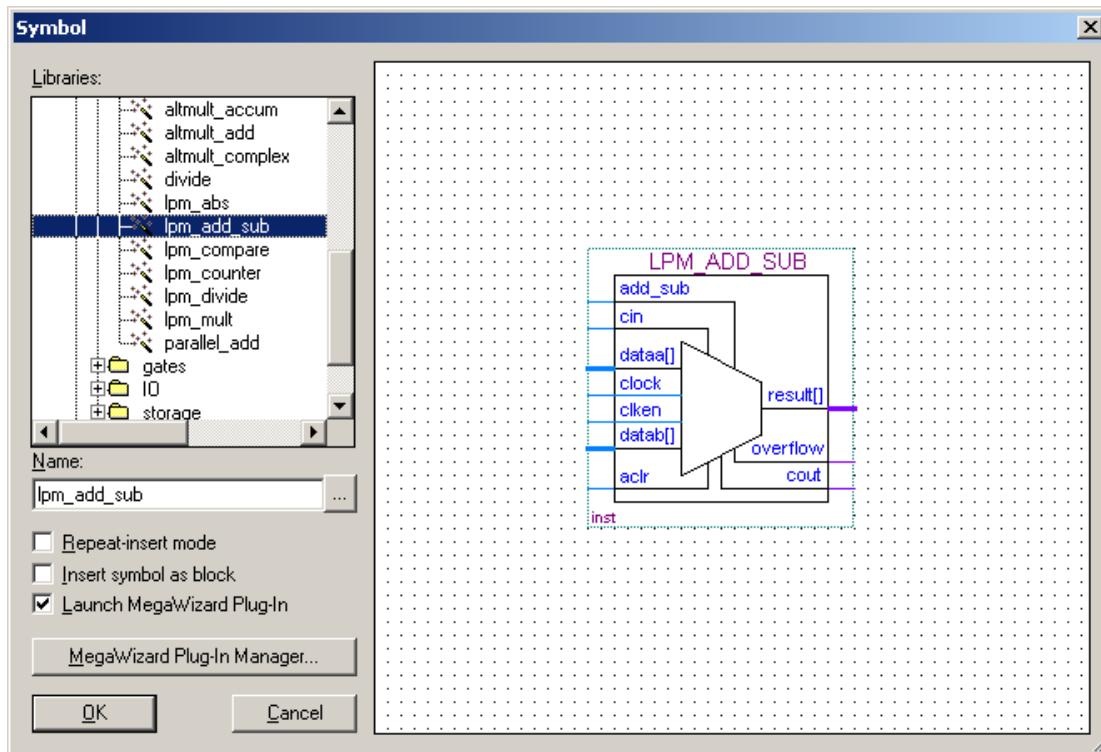


Figur 4-9 tilpass porter.

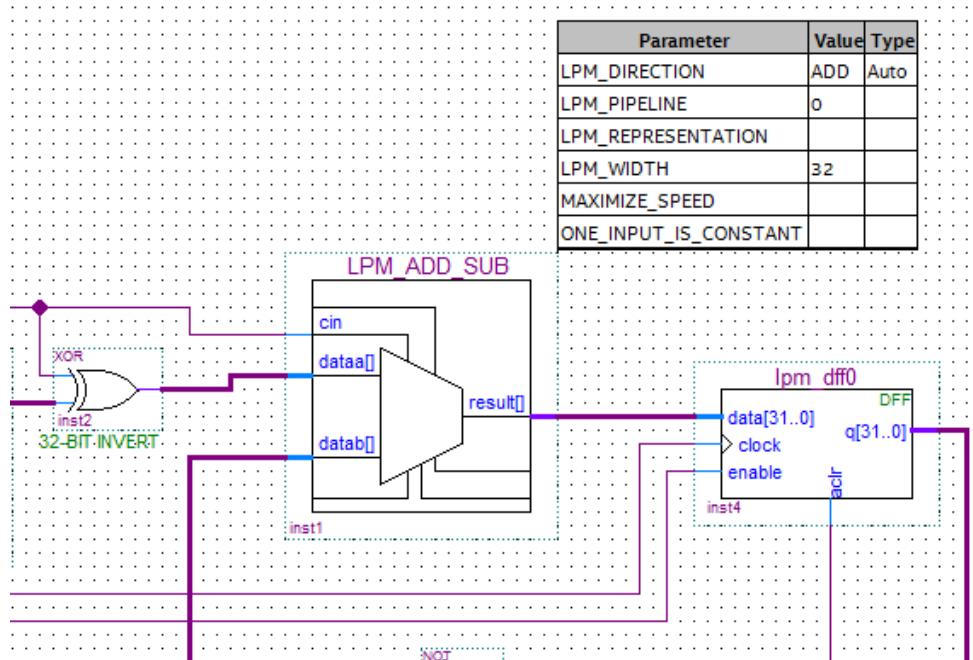


Figur 4-10 tilpass parametre.

Det dukker nå opp et addisjonssymbol i skjemaet deres. Plasser dette slik at pinnene blir korrekt tilkoblet. Skjemaet for den utførende enheten i multiplikatoren skal nå se ut som i figur 4-12.



Figur 4-11: Valg av aritmetisk megafunction



Figur 4-12: Multiplikatorens utførende enhet med adderer.

### Laboppgave 3: Skriv inn VHDL-kode

Neste trinn er å fullføre tilstandsmaskinen. Klikk dere inn kontrollenheten *mul\_control* sin *FSM* modul. Et *HDL Editor*-vindu vil åpnes med VHDL-koden til tilstandsmaskinen.

- ❖ Gjør endringene som dere kom fram til i Forarbeidsoppgave 5: VHDL-kode i forarbeidet.

## Laboppgave 4: Generere nytt symbol for FSM

Nå som det er lagt til en ny utgang på tilstandsmaskinen vår må det også genereres et nytt symbol.

- ❖ Generer et nytt symbol (egentlig oppdater et gammelt) på samme måte som dere gjorde i lab 3 Laboppgave 2: Design av generisk multiplexer. Pass på at dere har VHDL-filen oppe når dere oppdaterer symbolet.

I skjemaet *mul\_control* velger dere nå symbolet for tilstandsmaskinen, høyreklikker, og velger *Update Symbol or Block*. Trykk *OK* i dialogboksen som dukker opp.

Kontroller at pinnene som var der fra før fortsatt er tilkoblet riktig. Koble til den nye pinnen på rett sted i skjemaet. Dersom pinnene ikke er plassert slik dere vil ha dem er det mulig å endre på dette i symboleditoren. Høyreklikk på symbolet og velg *Edit Selected Symbol*. I det nye vinduet som dukker opp kan dere flytte på pinner og også endre størrelsen på selve symbolboksen.

## Laboppgave 5: Simulering av multiplikator

Fremgangsmåte for simulering er beskrevet i lab 2 og lab 3. Pass spesielt på å stille *Grid Size = 50ns* og *End Time = 10us* slik det ble gjort i lab 2. For å kunne tolke simuleringsresultatene korrekt, må dere *forstå* hele kretsen godt. Merk spesielt hvordan *DVI* og *DVO* brukes i systemet.

- ❖ Start simulatoren og utfør simulering av:
  - FSM modulen, i henhold til det dere kom fram til i Forarbeidsoppgave 6: Testplan for tilstandsmaskinen.
  - MUL modulen, i henhold til det dere kom fram til i Forarbeidsoppgave 7: Testplan for multiplikatoren.

Det er viktig at dere ikke går videre til neste modul før dere har fått korrekt oppførsel.

## Laboppgave 6: Implementasjon og nedlasting av design til FPGA

Når simuleringsresultatene for multiplikator viser korrekt oppførsel, skal dere kompile det komplette designet (*Lab4.bdf*). Konfigureringsfilen som da genereres skal dere så laste ned i FPGA-en. Se kapittel 2

Absoluttverdikrets for detaljer rundt dette.

Hent et DE2-kort, tastatur, USB-kabel, batterieliminator, mikrofon, hodetelefoner og audiokabel. Koble sammen utstyret på samme måte som i lab 3. Audiokablene skal kobles fra PC-ens *LINE OUT* til DE2-kortets *LINE IN* inngang (blå plugg i bakkant av kortet). Hodetelefonene skal kobles til DE2-kortets *LINE OUT* utgang (grønn plugg i bakkant av kortet).

**VENT MED Å KOBLE TIL HODETELEFONENE TIL DERE HAR  
LASTET NED DERES EGET DESIGN!**

**HUSK Å BRUKE ARMLENKE NÅR DERE HÅNDTERER DE2-KORTET!**

Når designet deres er lastet ned er det viktig å sette opp C-registeret og D-registeret riktig. Oversikt over hva de enkelte bitene i registrene styrer er gjentatt i figur 4-13. Dere vil også finne igjen de forskjellige bitene i skjemaene i Quartus II.

Når systemet er i lydmodus får StudentDesign sin klokke fra systemets klokkegenerator og DVI fra CODEC-ens DVO. Når systemet er i testmodus aktiviserer dere klokkesignalet inn på *StudentDesign* ved å trykke på knappen merket KEY3. Signalet fra denne knappen er lavt mens knappen er trykket inn og høyt når den ikke er trykket inn. I det knappen slippes vil vi følgelig få en positiv flanke på klokkesignalet. I testmodus er det SW17 som bestemmer nivået på DVI inn til *StudentDesign*. Når SW17 er skjøvet opp (tilhørende lysdiode LR17 lyser) er DVI høy. I testmodus er det dessuten verdien dere setter i A- og B-registrene fra tastaturet som påtrykkes inngangene A og B på *StudentDesign*. Dette er oppsummert i tabell 4-4. Merk at det i lydmodus er det samme 16 bit lydsignalet som går inn på både A- og B-inngangen.

C-Registeret							
7	6	5	4	3	2	1	0
SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
PASS	MUL	FACT	ABS	Reduction Speed			
D-Registeret							
7	6	5	4	3	2	1	0
SW15	SW14	SW13	SW12	SW11	SW10	SW9	SW8
MODE	SRC	Volume					

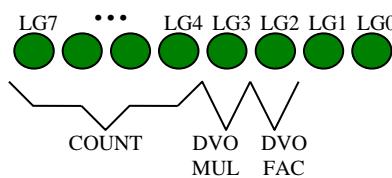
Figur 4-13: Bitvis oversikt over C- og D-registeret

Tabell 4-4: Kilder for inngangssignal til *StudentDesign*

Signal inn på <i>StudentDesign</i>	Kilde i lydmodus	Kilde i testmodus
CLK	Klokkegenerator	KEY3
DVI	DVO fra CODEC	SW17
A[15..0]	audioData[15..0] fra CODEC	Register A satt fra tastatur
B[15..0]	audioData[15..0] fra CODEC	Register B satt fra tastatur

I testmodus vises resultatet av beregningene inne i *StudentDesign* på sjusegmentdisplayene HEX[7..0]. Hvert sjusegmentdisplay viser et heksadesimalt tall svarende til 4 bit. Totalt får vi følgelig vist alle de 32 resulterende bit ut fra multiplikatoren.

For å kunne se gangen i hele systemet på DE2-kortet, er en del informasjon om aktiviteten i multiplikatoren lagt ut på de grønne lysdiodene på kortet (se figur 4-14). Lysdiodene LG[7..4] viser hvilken verdi nedtelleren i multiplikatoren har, LG3 lyser når DVO fra multiplikatoren er høy og LG2 lyser når DVO fra FACT er høy.



Figur 4-14: Grønne LED på DE2-kortet

Sett systemet i testmodus og deaktiver modulene *ABS* og *FACT* (se figur 3-11 for hvordan dette gjøres). Påtrykk ønskede multiplikand- og multiplikatorverdier ved hjelp av tastaturet og aktiviser deretter *CLK* og *DVI* manuelt. Får dere korrekt svar?

- ❖ Dere skal nå gjennomføre noen av multiplikasjonene fra testplanen dere laget i Forarbeidsoppgave 7: Testplan for multiplikatoren.
- ❖ Forsøk å aktivisere modulene *ABS* og/eller *FACT*. Hvordan oppfører systemet seg nå?

## Laboppgave 7: Lyd

Sett systemet i lydmodus og velg mikrofonen som kilde for lyd. Sett volumet ut på hodetelefonene til å være 0b110000 (0b betyr at det etterfølgende er en binær verdi). Alt dette styres av D-registeret. Aktiver alle modulene i systemet.

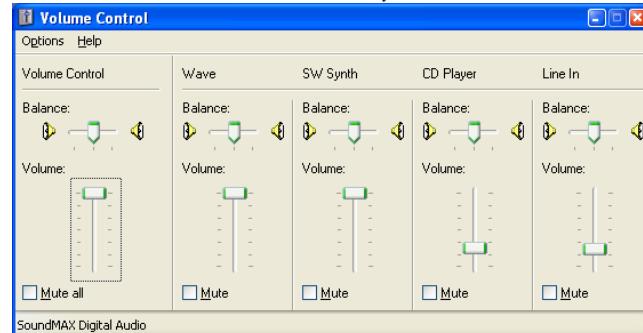
- ❖ Aktiviser peak-meteret med skyvebryter SW16. Snakk i mikrofonen og hør resultatet i hodetelefonene. Hør hva som skjer når dere varierer reduksjonshastigheten. Studer også oppførselen på signalet ved å se på peak-meteret. Snakk først høyt og deretter lavere og lavere. Sammenlikn med ubehandlet lyd (aktiviser BYPASS ved hjelp av SW7 i C-registeret). Beskriv hva dere hører og ser og forsøk å forklare det.

Nå skal vi eksperimentere med en lydfil på PC-en som lydkilde. Velg *line in* som kilde for lyd på DE2-kortet (velges som SRC ved hjelp av SW14 i D-registeret). For denne deloppgaven må vi kalibrere nivåene både fra PC til DE2 og fra DE2 til oscilloskop. Vær nøyne med å følge denne prosedyren eksakt.

- Dobbeltklikk på høytaleseren på PC-ens oppgavelinje for å starte lydmikseren



- Åpne Windows lydmikser og sett Volume Control silder og Wave slider til maks.



- Åpne spilleren *Winamp* og sett Volume til maks. Se figuren til høyre for korrekte innstillingar.



- Sett DE2-kortet i bypass (C-registeret) og skru på peak-meteret (SW16).
- Bruk *Winamp* til å spille av filen «kalibreringstone440Hzminus6dB.mp3» som ligger i Blackboard. Det anbefales å ikke ha på hodetelefoner når denne spilles.

- Juster deretter Volume Control slider i Windows lydmikser fra maks og nedover, et steg om gangen, til LED8R lyser, og LED9R er slukket. For å justere enkeltsteg, klikk på slideren, og bruk piltastene opp og ned på PC-tastaturet.
- Klikk så på volumsliden i *Winamp* og juster den nedover, et steg om gangen, til forsterkningsfaktoren vist på det venstre 7-segmentdisplayet i DE2 kortet er så nært 2,00 som mulig, men ikke under 2,00. Bruk piltastene opp og ned for å justere enkeltsteg på samme måte som over. For noen PC-er er det umulig å komme nærmere enn ca. 2,2).

Med denne kalibreringen gjort, vil det bare en sjeldent gang skje at lydsignalet på inngangen er så sterkt at systemet klipper lyden. Se beskrivelsen av peak-meteret i teoridelen for informasjon om klipping.

- ❖ Bruk *Winamp* til å spille av filen «Forelesning12.wav» som ligger i Blackboard. Hør først på lyden ubehandlet (bypass), og deretter kjørt gjennom lydkompressoren. Kunne dette utstyret vært nyttig i en forelesningssituasjon?

## Avslutning

- Koble fra batterieliminator, tastatur, USB-kabel, mikrofon, hodetelefon, kabler og prober. Sett tilbake utstyret.
- Avslutt Quartus II.

# 5 Laboratorieøving: Volumstyring av systemet med enkel prosessor

## Innledning

### Mål

Å programmere en prosessorkjerne til å styre volumet i lydbehandlingssystemet. Prosessoren vil syntetiseres og eksistere i FPGA-en sammen med lydbehandlingssystemet.

### Hensikt

- Forstå oppbygningen av en enkel prosessorkjerne, sammenhengen mellom instruksjoner og styreord, og hvordan prosessorkjernen programmeres.
- Avluse (debugge) et program som kjører på prosessoren
- Forstå hvordan en prosessorkjerne kan kommunisere med et I/O-grensesnitt

### Bakgrunn

Teori for denne oppgaven finner dere i: Mano og Kime: «Logic and Computer Design Fundamentals», kapittel 10:

- 10.1-10.4 gir generell bakgrunn
- 10.5 Utførende enhet
- 10.6 Styreord
- 10.7 A simple computer architecture
- 10.8 Single-cycle hardwired control
- 10.9 Multiple-cycle hardwired control

Prosessoren som brukes her har mange likheter med prosessorene som er gitt i 10.8 og 10.9.

### Hva som skal gjøres

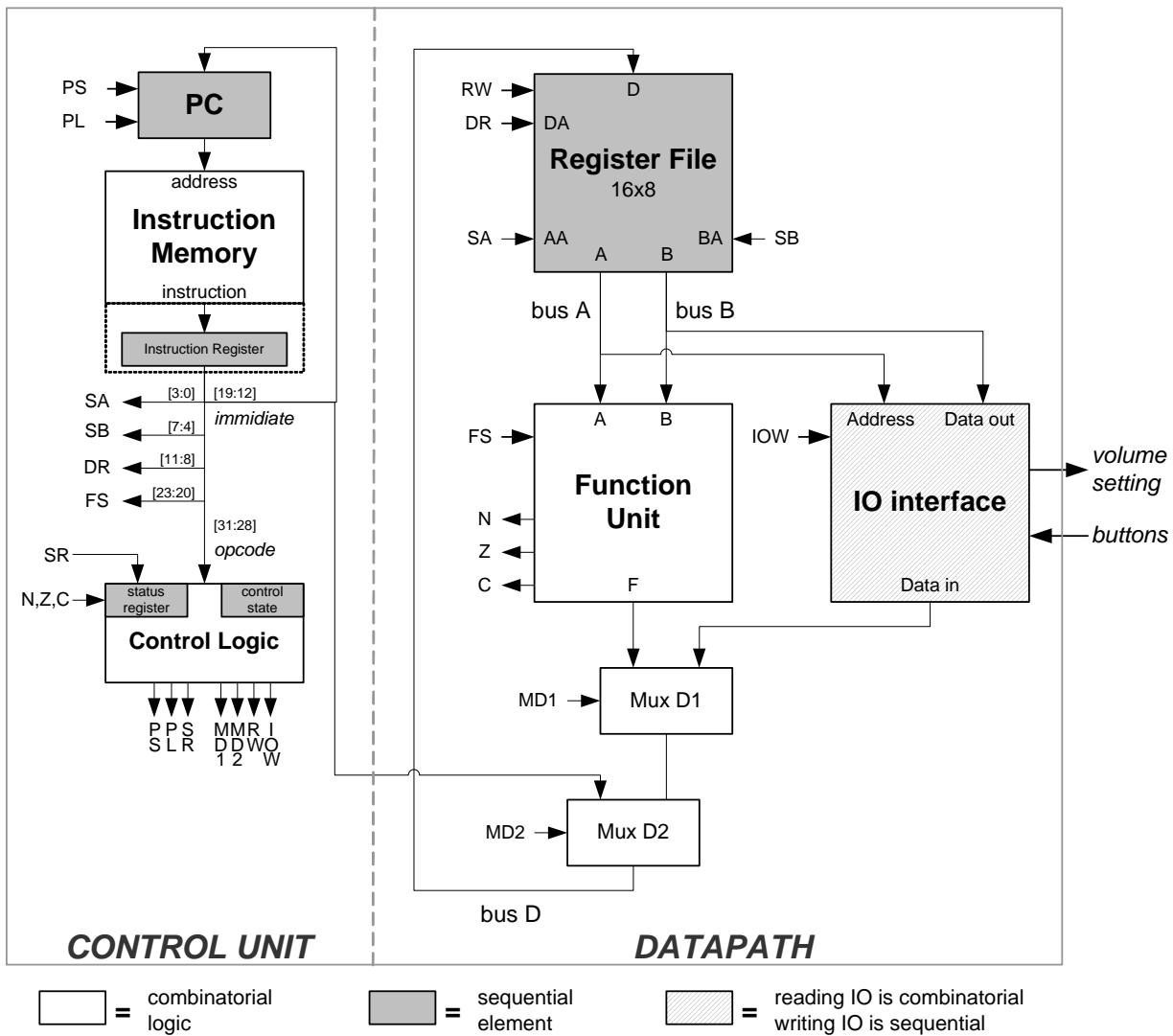
Dere skal sette dere inn i virkemåten til en enkel prosessor. Dere skal fullføre styreenheten ved å legge til styresignaler for en av prosessorens instruksjoner. Dere skal lære hvordan man skriver og avluser programmer på denne enkle prosessoren.

## Beskrivelse

### Prosessorens virkemåte

Figur 5-1 viser prosessorens logiske oppbygning. Jamfør med prosessoren i figur 10-15 i læreboken. Prosessoren slik den er implementert i Quartus er vist i figur 5-2 (uten styresignalene fra styrelogikken).

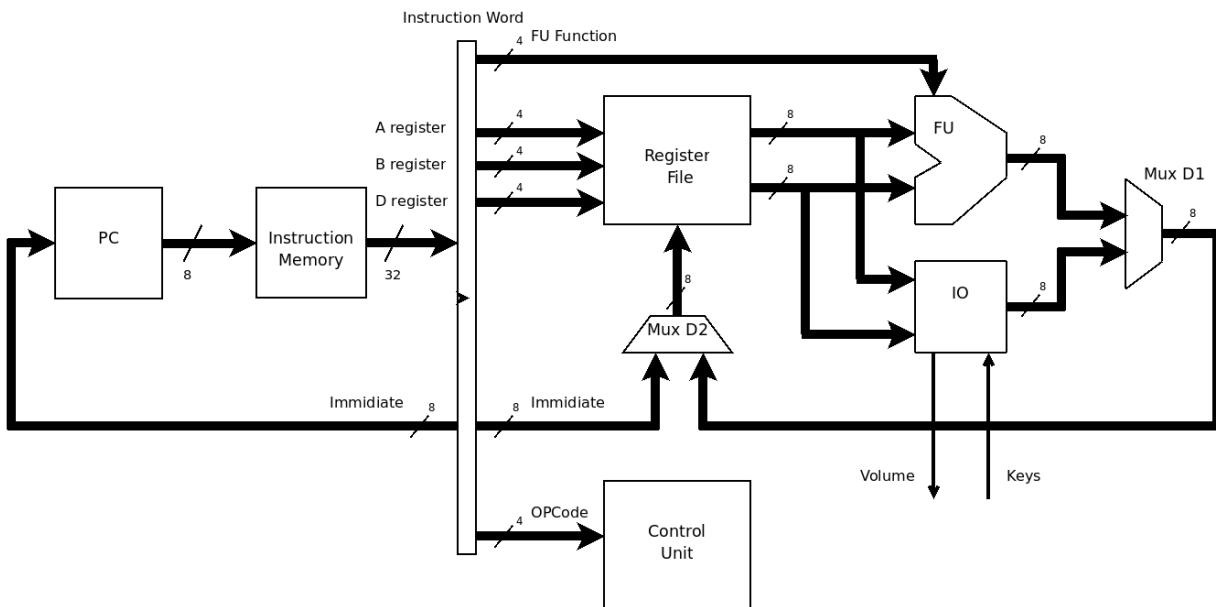
Prosessoren er en to-sykel maskin. Det vil si at den utfører en instruksjon annenhver klokkesykel. Den første klokkesykelen kalles «fetch» og den andre heter «execute». Grovt fortalt hentes instruksjonen som skal utføres i «fetch» steget, mens alt arbeidet foregår i «execute» steget. Maskinen har ikke datalager, men derimot et I/O-grensesnitt den skriver til og leser fra. Dette I/O grensesnittet er på sin side tilkoblet trykknapper og volumstyringen til audioCODEC-en.



Figur 5-1: Logisk diagram over prosessoren

Tabell 5-1: Forkortelser brukt i labheftets lab 5

<b>SA</b>	Source register A	<b>PC</b>	Program Counter
<b>SB</b>	Source register B	<b>PS</b>	PC Select (select PC input)
<b>DR</b>	Destination Register	<b>PL</b>	PC Load
<b>FU</b>	Functional Unit (funksjonell enhet)	<b>SR</b>	Status Register write
<b>FS</b>	FU Function Select	<b>RW</b>	Register Write
<b>MD1</b>	Mux D1 select	<b>IOW</b>	I/O Write
<b>MD2</b>	Mux D2 select	<b>N,Z,C</b>	Negative, Zero, and Carry status bits



Figur 5-2: Prosessoren implementert i Quartus

### «Fetch» steget:

Instruksjonslageret er en sekvensiell enhet. Det vil si at den opererer synkront med klokken på positiv flanke. Det som skjer i denne fasen er følgende:

1. I utgangspunktet inneholder programtelleren den nåværende posisjonen i programmet.
2. Programtelleren adresserer instruksjonslageret.
3. På positiv klokkeflanke (altså samtidig med at man skifter fra forrige «execute»-steg til nåværende «fetch» steg) legges innholdet av instruksjonslageret på et internt instruksjonsordregister, slik at dette holdes på bussen.

### «Execute» steget:

I «execute» - steget skjer alle operasjoner i utførende enhet (datapath). Alt i utførende enhet er kombinatorisk logikk med unntak av:

- Skriving til destinasjonsregisteret (DR)
- Oppdatering av programtelleren (PC)
- Skriving (av volumet) til I/O enheten.

Alle disse sekvensielle operasjonene skjer på neste positive klokkeflanke. Altså i overgangen til neste «fetch» steg. Listen under gir et overblikk over hva som skjer i en normal operasjon (ikke I/O – dette er dekket i eget avsnitt). Det er viktig å huske at selv om dette er en sekvensiell liste, så skjer mange av disse operasjonene samtidig. Rekkefølgen er bare gitt for å lette forståelsen.

Enkelt forklart fungerer den på følgende måte.

1. Instruksjonsordet fra instruksjonslageret splittes opp i komponentene angitt i tabell 5-2.
2. Opkoden i instruksjonen sendes til styrelogikken. På bakgrunn av hvilken opkode det er settes kontrollsignaler. Se tabell 5-4 for en oversikt over opkoder, og tabell 5-3 for en

oversikt over styresignalene. Disse styresignalene avgjør hva som skjer i den utførende enheten.

3. Verdiene av SA og SB – feltene i instruksjonen avgjør hvilke registre som skal legges ut på A og B-bussen. DR – feltet avgjør hvilket register det eventuelt skal skrives til.
4. Verdiene av SA og SB registeret blir sendt via A- og B-bussen til **åde** FU-en og IO-enheten. Dette er uavhengig av hvilken operasjon vi faktisk ønsker å få utført.
5. FU-en utfører aritmetiske operasjoner, uavhengig av om resultatet kommer til å bli brukt eller ikke. Hvilken operasjon som blir utført avhenger av «FS» feltet i instruksjonen (se tabell 5-5). Samtidig utfører I/O-enheten I/O, uavhengig av om operasjonen er en I/O operasjon eller ikke.
6. Resultatet fra FU-en og IO-enheten blir sendt inn i Mux D1. Med denne multiplekseren kan styreenheten velge om den ønsker resultatet fra FU-en eller IO-enheten.
7. Resultatet fra Mux D1 kommer til Mux D2. Her kan styreenheten velge om den ønsker resultatet fra FU-en og IO-enheten eller verdien fra immediate feltet i instruksjonen.
8. Sluttresultatet kan så bli skrevet til register DR, dersom styresignalet RW er satt av styreenheten.
9. Deretter blir programtelleren oppdatert. Her kan styreenheten gjøre to ting. Enten kan den inkrementere PC med 1 (for å fortsette med neste instruksjon i programmet) eller så kan den legge immediate-verdien inn som ny PC. I siste fall får vi et hopp i den normale programflyten.

*Tabell 5-2: Instruksjonsformatet, bit 31..0*

31..28	27..24	23..20	19..12	11..8	7..4	3..0
Opcode	Ubrukt	FS – Function Select	Immediate	DR – Destination Register	SB – Source register B	SA – Source register A

*Tabell 5-3: Styresignaler*

Styresignal	Enhet	Funksjon
control_PS	Program Counter	Velger hvordan programtelleren skal oppdateres. 0: PC = PC + 1, 1: PC = Immediate
control_PL	Program Counter	Velger om programtelleren skal oppdateres. 0: Ikke oppdater, 1: Oppdater
control_RW	Register File	Velger om det skal skrives til D-registeret. 0: Nei, 1:Ja
control_IOW	IO	Velger om man skal utføre skriving av dataene man får inn på IO-porten. 0: Ingen skriving, 1: Skriv data.
control_MD1	Mux D1	Velger mellom data fra FU eller IO. 0: FU, 1: IO
control_MD2	Mux D2	Velger hva som skal skrives til D-registeret. 0: Verdi fra FU eller IO (IO-Mux), 1: Immediate feltet.
control_SR	Control Unit	Velger om statusregisteret i styreenheten skal oppdateres eller ikke. 0: Ikke oppdater, 1: Oppdater

Tabell 5-4: Opkoder

Instruksjon	Opkode	FS	Funksjon
MOV DR, SA	0x1	0x0	$R[DR] = R[SA]$
AND DR, SB, SA	0x1	0x1	$R[DR] = R[SB] \wedge R[SA]$
OR DR, SB, SA	0x1	0x2	$R[DR] = R[SB] \vee R[SA]$
XOR DR, SB, SA	0x1	0x3	$R[DR] = R[SB] \oplus R[SA]$
NOT DR, SA	0x1	0x4	$R[DR] = \sim R[SA]$
SHL DR, SA	0x1	0x5	$R[DR] = sl R[SA]$
SHR DR, SA	0x1	0x6	$R[DR] = sr R[SB]$
ADD DR, SB, SA	0x1	0x7	$R[DR] = R[SA] + R[SB]$
SUB DR, SB, SA	0x1	0x8	$R[DR] = R[SA] - R[SB]$
BNZ Imm	0x2	X	If ( $Z \neq 0$ ) PC = Imm
LDI DR, Imm	0x3	X	$R[DR] = Imm$
JMP Imm	0x4	X	PC = Imm
IN DR, SA	0x5	X	$R[DR] = IO[SA]$ Leser inn fra eksterne IO-enheter og legger den innleste verdien i DR. Porten man ønsker å lese legges i SA-registeret.
OUT SA, SB	0x6	X	$IO[SA] = R[SB]$ Output. Skriver ut register SB på porten gitt ved register SA.

~ betyr bitvis invertering

Tabell 5-5: FU-funksjonstabell

FS-funct	Semantikk	Funksjon
0x0	MOV	$F = A$
0x1	AND	$F = A \wedge B$
0x2	OR	$F = A \vee B$
0x3	XOR	$F = A \oplus B$
0x4	NOT	$F = \sim A$
0x5	SHL	$F = sl A$
0x6	SHR	$F = sr A$
0x7	ADD	$F = A + B$
0x8	SUB	$F = A - B$

~ betyr bitvis invertering

## I/O operasjoner

Gjennom I/O – grensesnittet (Input/Output) kommuniserer prosessoren med resten av systemet. I dette enkle systemet er det kun mulig å kommunisere med 2 knapper og med volumkontrollen til audioCODEC-en. Dette gjøres gjennom porter (se tabell 5-6 for portnumre). Bus A settes til portnummeret til I/O enheten man ønsker å kommunisere med. Dersom dette er en lese-port vil resultatet ligge på utgangen av I/O enheten umiddelbart (kombinatorisk).

*Tabell 5-6: I/O porter*

Portnummer	Funksjon
1	Leser inn KEY1 (Kun lesing). Gir 0 dersom den er nedtrykt, 1 om ikke.
2	Leser inn KEY2 (Kun lesing). Gir 0 dersom den er nedtrykt. 1 om ikke.
3	Setter volumet (Kun skriving). Maks = 0x7E, mindre verdier enn 0x60 er knapt hørbare.

## Bryterne på kortet

I denne labben er audiokompressoren alltid tilkoblet. C- og D- registrene i audiokompressoren fungerer på samme måte som tidligere labber, med unntak av at volumdelen av D-registeret ikke lengre gjør noe. SW17 styrer nå kun om prosessoren skal kjøres på 11MHz (nedre stilling) eller om den skal klokkes av KEY3 (øvre stilling). KEY0 resetter kretsen. OBS! Trykker man på denne vil programmet på NIOS-en også slettes, dermed forsvinner debug-informasjonen fra LCD-displayet. KEY1 og 2 er koblet til prosessorens I/O porter.

## Forarbeid

### Blackboard

Les lab-info-en på fagets side på *Blackboard*, og møt opp på lab-forelesningen.

### Styresignal til LDI

Instruksjonen Load Immediate (LDI) er ikke ferdig implementert i designet dere får utdelt på labben. Denne instruksjonen laster inn data fra immediate-feltet i instruksjonen inn i destinasjonsregisteret (DR).

Studer oppbygningen av prosessoren i bakgrunnskapitlet over.

- ❖ Hvilke styresignal må settes av styreenheten for å utføre denne instruksjonen?

### Avlusing

På labben skal dere se på hvordan man kan avluse (debugge) et program på prosessoren. Programmet dere skal avluse er følgende:

```
0  LDI    R1,  5
1  LDI    R2,  1
2  SUB   R1,  R2,  R1
3  BNZ    2
4  LDI    R3,  11
5  JMP    0
```

- ❖ Hvordan forventer dere at dette programmet oppfører seg? Forklar.

### Elementær I/O

Nå skal dere oversette et gitt program til maskinkode. Dere skal også se hvordan elementær I/O fungerer ved å se på hvordan dere kan lese trykknappene KEY1 og KEY2. Programmet er:

```
0  LDI    R4,  1
1  LDI    R5,  2
2  IN     R1,  R4
3  IN     R2,  R5
4  JMP    2
```

- ❖ Hva gjør dette programmet?
- ❖ Hvilken oppførsel forventer dere når dere laster dette ned på kortet?
- ❖ Oversett programsnutten til maskinkode. Dere må bruke tabellen over opkoder, FU-funksjoner og instruksjonsformatet beskrevet over. Skriv ned oversettelsen i tabell 5-7.

Tabell 5-7: Maskinkode

Adresse	Kode	Opkode	FU-funksjon	Immediate	DR	SB	SA
0	LDI R4, 1	0x3	X	0x1	0x4	X	X

X er don't care.

HUSK! Etter opkoddefeltet er det fire bit som ikke brukes. Disse kan dere godt sette til 0.

## Volumstyring

I den siste oppgaven skal dere implementere volumstyring på prosessoren. Oppgaven er som følger:

- Brukeren (altså dere) skal kunne stille inn volumet på systemet ved å bruke KEY1 og KEY2 for henholdsvis opp og ned.
- For å justere ett hakk opp eller ned så skal knappen først trykkes inn og så slippes. Det betyr at man ikke kan holde knappen inne for å gå fort igjennom volumet.
- Det gjeldende volumet skal lagres i R1 slik at det synes på 7-segmentet på DE2-kortet.
- Programmet skal påse at volumet aldri skal være over 0x7F, eller under 0x60.

I figur 5-3 er det gjengitt et rammeverk i VHDL som dere kan bruke (denne koden vil også allerede være inkludert i designet dere skal jobbe med på labben). Syntaksen er som følger:  
`mem_storage(0) <= X"30001400"`

Dette betyr at instruksjonen 0x30001400 skal legges på plass 0 (desimalt) i instruksjonsminne.

- ❖ Skriv koden for det utevede partiet (Student-kode) på symbolsk form (dvs. med LDI, JMP, BNZ osv.).
- ❖ Oversett koden til maskinkode.

### Tips:

- Branch if zero kan implementeres ved å bruke branch if not zero til instruksjonen to hakk under, og legge inn en jump i mellom.
- Husk at adressen dere angir i hopp-instruksjoner er gitt i heksadesimalt, mens verdien dere setter på mem\_storage(X) er desimalt.
- Det kan være lurt å bruke en iterativ løsningsmetodikk. Først løs problemet med en høynivå algoritme, deretter oversett enkeltbestanddelene til symboler (LDI, JMP etc.), for så å oversette til maskinkode.
- Husk at BNZ tester på hva som skjedde med forrige FU-operasjon, ikke forrige IN eller liknende.
- Prøv å bruke R0-R4 så mye som mulig. Det letter debuggingen betydelig.

```

-- Initial values
mem_storage(0) <= X"30060100";      -- LDI R1, 60 (current volume)

-- Read KEY1
mem_storage(1) <= X"30001400";      -- LDI R4, 1 (Port 1 is KEY1)
mem_storage(2) <= X"50000204";      -- IN R2, R4 (Read port R4 and store the
                                         -- result in R2.
                                         -- 0 means that the key has been pressed.
mem_storage(3) <= X"30001400";
mem_storage(4) <= X"10800242";
mem_storage(5) <= X"20030000";
                                         -- LDI R4, 1
                                         -- Sub R2, R4, R2 ( R2 = R2 - R4)
                                         -- BNZ 30 (if the subtraction was not zero,
                                         -- then the key was pressed). Jump to 0x30 (48)
                                         -- to handle it.

-- Read KEY2
mem_storage(6) <= X"30002400";      -- LDI R4, 2 (Port 2 is KEY2)
mem_storage(7) <= X"50000304";      -- IN R3, R4 (Read port R4 and store the
                                         -- result in R3
                                         -- 0 means that the key has been pressed.
mem_storage(8) <= X"30001400";
mem_storage(9) <= X"10800343";
mem_storage(10)<= X"20050000";
                                         -- LDI R4, 1
                                         -- Sub R3, R4, R3 ( R3 = R3 - R4)
                                         -- BNZ 50 (if the subtraction was not zero,
                                         -- then the key was pressed). Jump to 0x50 (80)
                                         -- to handle it

-- Jump back to top
mem_storage(11)<= X"40001000";      -- JMP 1 (Repeat until something is pressed)

-- Student-kode begynner her:

-- Handle keypress 1
-- First, wait till user lets go of the key
-- mem_storage(48) <=
-- Don't increase it beyond 0x7F (max volume)

-- Handle keypress 2
-- First, wait till user lets go of the key
-- mem_storage(80) <=
-- Don't increase it below 0x60 (min volume)

-- Adjust volume
mem_storage(110)<= X"30003400";      -- LDI R4, 3 (Port 3 is volume)
mem_storage(111)<= X"60000014";      -- OUT R4, R1 (Write vFUE R1 into R4)
mem_storage(112)<= X"40001000";      -- JMP 1 (Back to main)

```

*Figur 5-3: VHDL kode for volumstyring*

## Labarbeid

### Oversikt

Labarbeidet består av 6 oppgaver:

*Tabell 5-8: Oppgaver i labarbeidet*

Oppgave	Innhold
1	Klargjøring
2	Implementering av instruksjon
3	Avlusning
4	Oversetting av program til maskinkode
5	Volumstyring
6	Avslutning

### Laboppgave 1: Klargjøring

- Les lab-infoen på Blackboard.

Hent *LAB5.ZIP* fra Blackboard og legg filene i en egen katalog slik det er forklart i kapittel 2 Absoluttverdikrets. Start Quartus II, åpne prosjektet *Lab5.qpf* og deretter filen *Lab5.bdf* (se kapittel 2

Absoluttverdikrets for detaljert fremgangsmåte).

Merk: Denne gangen er krets og pinnetilordning allerede gjort.

### Laboppgave 2: Implementering av instruksjon

Ta utgangspunkt i løsningen av oppgave 2 fra forarbeidet. Dere skal gå inn i prosjektet for å implementere LDI ved å endre på VHDL-koden til styreenheten.

- Klikk dere inn på *wb\_processor*. Dette er den enkle prosessoren som er beskrevet i teoridelen.
- Ser dere sammenhengen mellom illustrasjonen i figur 5-2 og skjemategningen?
- Klikk dere inn på *wb\_cpu\_control*. Dette er styreenheten.
- Dere vil nå få opp VHDL kildekoden til styreenheten.
- Stedet dere skal legge inn kode er markert med «studentkode». Alle styresignaler skal settes her. Merk at dere ikke trenger å eksplisitt sette de signaler dere vil ha til 0, bare de som skal være 1. Dere trenger heller ikke å sette signalet *control\_PL*. Dette er allerede satt.
- Kompiler prosjektet.

Hent et DE2-kort, tastatur, USB-kabel, batterieliminator, mikrofon, hodetelefoner og audiokabel. Koble sammen utstyret på samme måte som i lab 4.

**VENT MED Å KOBLE TIL HODETELEFONENE TIL DERE HAR  
LASTET NED DERES EGET DESIGN!**

**HUSK Å BRUKE ARMLENKE NÅR DERE HÅNDTERER DE2-  
KORTET!**

- Skru av alle brytere (SW0 – SW17) på DE2-kortet (posisjon ned).
- Last ned designet deres på DE2-kortet.
- Dersom implementeringen av LDI er korrekt vil det lyse FF på den midterste 7-segmentetdisplayet.

### **Laboppgave 3: Avlusing**

Nå skal dere kjøre det programmet dere studerte i oppgave 3 av forarbeidet på prosessoren skritt for skritt, og observere hva som skjer. I andre programmeringsfag har dere jobbet med avlusing (debugging) ved hjelp av egne programmer. Her skal vi isteden avluse programmet ved å observere oppførselen direkte på DE2-kortet. Begge disse metodene er nyttige, hver til sin tid.

- Åpne *wb\_processor* igjen
  - Klikk dere inn på *wb\_cpu\_mem*. Denne inneholder maskinkoden til programmet.
  - Kommenter **ut** alt mellom «Oppgave 1» og «Oppgave 2». I VHDL begynner en kommentarlinje med «--».
  - Kommenter **inn** alt mellom «Oppgave 2» og «Oppgave 3».
  - Sett bryter SW17 i 1-posisjon (opp). Dette gir dere avlusnings-muligheten.
  - Kompiler designet og last det ned på DE2-kortet.
  - På displayet vises nå innholdet i Programtelleren (P:), samt innholdet av register 0-4. De to første syv-segmentet viser programtelleren. De neste seks viser R1, R2 og R3.
  - Trykk på KEY3 for å steppe gjennom programmet.
- ❖ Oppførte programmet seg som forventet?

**MERK:** Dersom dere bruker reset-knappen (KEY0) vil lcd-displayet bli dødt. Last ned designet på nytt for å fikse dette.

### **Laboppgave 4: Oversetting av program til maskinkode**

Nå skal dere teste ut maskinkodeprogrammet dere skrev i oppgave 4 av forarbeidet.

- Åpne *wb\_cpu\_mem* igjen.
  - Kommenter ut Oppgave 2.
  - Skriv inn deres program.
  - Kompiler og last designet ned på DE2-kortet
  - Sett switch SW17 på (opp).
- 
- ❖ Stepp gjennom programmet med KEY3. Oppfører det seg som forventet?
  - ❖ Trykk ned KEY1 og/eller KEY2 mens dere stepper gjennom programmet. Hva skjer med verdien av R1 og R2?
  - ❖ Sett SW17 av. Dette gjør at prosessoren kjører på 11MHz. Hva skjer med PC? Hva skjer med R1 og R2? Hva skjer når dere trykker på knappene?

## Laboppgave 5: Volumstyring

Nå skal dere teste ut programmet dere skrev i oppgave 5 av forarbeidet.

- Skriv inn koden deres i *wb\_cpu\_mem* som i de foregående oppgavene.
  - Kompiler og last designet ned på DE2-kortet.
- 
- ❖ Kjør først programmet i debuggingsmodus. Oppfører programmet seg slik dere forventer? Hvis ikke, gjør de nødvendige endringer i koden.
  - Koble opp datamaskinen som en lydkilde slik det er gjort i oppgave 7 av lab 4.
  - Sett opp audiokompressoren. Se tidligere labber for oppsett av C- og D-registeret.
- 
- ❖ Spill av en musikkfil. Kan dere styre volumet med knappene?

Merk: Husk å gjennomføre kalibrering av volum slik det ble gjort i lab 4.

## Presenter arbeid for stud.ass.

Før dere tilkaller student assistent for presentasjon, gå gjennom følgende punktliste og se etter at dere har alt klart.

## Avslutning

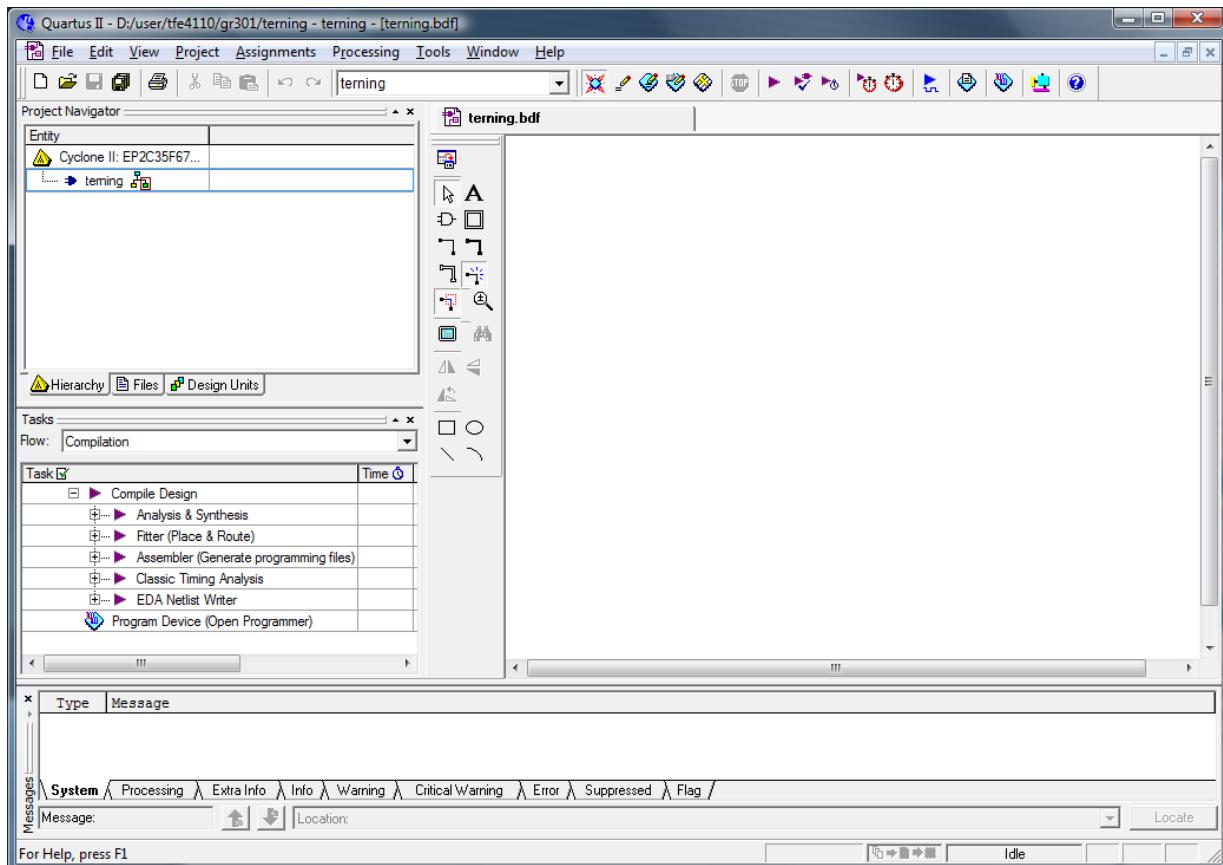
- Koble fra batterieliminator, tastatur, USB-kabel, mikrofon, hodetelefon og audiokabel.  
Sett tilbake utstyret.
- Avslutt Quartus II.



# 6 Veiledning for Quartus II

I dette kapitlet vil noen av Quartus' funksjoner gått gjennom. Første delkapittel er en gjennomgang av de grunnleggende funksjonene i Quartus. Deretter følger en steg-for-steg prosedyre på noen av de mest brukte prosedyrene. Det anbefales at første del leses gjennom. Siste del er ment som et oppslagsverk for når funksjonene skal brukes.

## Grunnleggende funksjonalitet i blokkskjemaeditoren



**Figur 6-1:** Quartus' blokkskjemaeditor

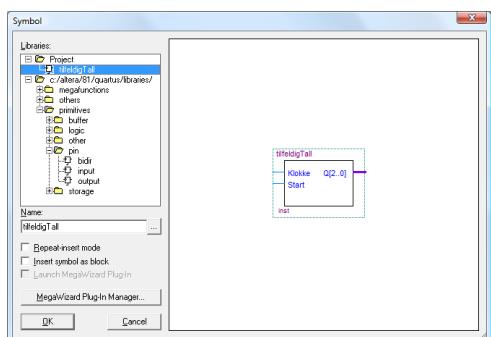
Hovedvinduet i Quartus består av hovedvinduet til høyre, og 3 mindre informasjonsvindu: «Project navigation», «Tasks», «Messages».

Informasjonsvinduene har følgende funksjon:

- «Project navigation» inneholder 3 underfaner. Hierarkifanen viser strukturen av blokker og underblokker. Øverst i hierarkiet er filen som er satt som toppnivåentiteten, «Top-level entity». «Files» fanen viser filene som er lagt til i prosjektet. Under denne fanen settes og hvilken fil som skal være toppnivåentiteten. Under «Design Units» er entitetene (blokkene) som brukes, listet opp.
- «Tasks» inneholder en oversikt over de oppgave en kompilering består av. Hver oppgave kan startes enkeltvis ved å dobbeltklikke på den, eller så kan alle kjøres sekvensielt ved å dobbeltklikke på «Compile Design». I dette vinduet viser og status og fremdrift for de forskjellige oppgavene under kompilering.
- «Messages»-vinduet viser tekstukskrift og detaljert status fra programmet.

Øverst i Quartus har vi hovedverktøylinjen. Langs kanten av hovedvinduet er redigeringsverktøyene. En opplisting av redigeringsverktøyenes funksjon finnes i Tabell 6-1: Menyknapper i editoren.

## Legge til komponenter



**Figur 6-2:**Vindu for nytt symbol

Ved å klikke på symbolverktøyet , eller dobbeltklikke på et tomt område på skjemaet får vi frem vinduet for valg av komponenter. Her kan vi velge de komponentene, både integrerte og egendefinerte logiske funksjoner, eller blokkene vi ønsker å legge til i designet.

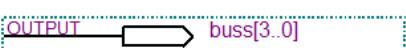
Egenlagde blokker ligger under «Project». Ved å skrive navnet direkte i «Name» feltet vil komponenten komme frem dersom navnet stemmer overend med et navn i databasen. Eksempel på dette er AND2, for 2-inngangs AND-port, AND3 for 3-inngangs AND-port, NOT for inverter, XOR for XOR-port og DFF for D-vippe. Inn- og utganger er også definert som symbol, og har navn INPUT og OUTPUT. Både enkeltsignal og busser bruker samme inn- og utganger. For mer informasjon om busser, se underkapitlet om bruk av busser.

Når riktig symbol er funnet, klikk OK og plasser symbolet på ønsket sted i tegningen. Det er viktig å tenke over, og lage en plan, for hvordan kretsen skal se ut, slik at minst mulig linjer krysser hverandre.

## Inn- og utganger



**Figur 6-3:** Buss-inngang



**Figur 6-4:** Buss-utgang



**Figur 6-5:** Signal-inngang

Inn og utganger legges til på samme måte som komponenter.

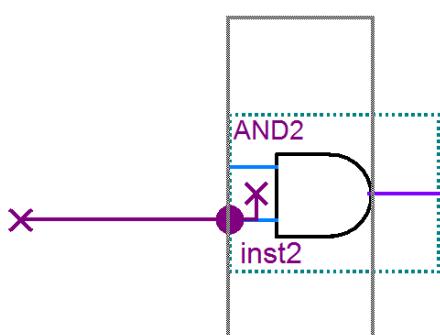
De kan hentes direkte opp ved å skrive «INPUT» eller «OUTPUT» i «name»-feltet i vinduet for nye komponenter. For toppnivå-entiteten må inn og utgangers navn være i samsvar med det som er definert i definisjonsfilen. Dette er som regel spesifisert i oppgaveteksten. Mer informasjon om definisjonsfilen for inn og utganger er i underkapitlet om hvordan laste denne filen inn.

For underblokker kan navn velges fritt. Navnet spesifiserer om inngangen er en buss eller et enkeltsignal. Navn som identifiserer en buss, vil medføre at inn-/utgangen er en buss. Eksempel: «busUtgang[3..0]». Husk at inn- og utganger, på lik linje med andre signalnavn, ikke kan inneholde mellomrom.

Merk at Quartus krever at alle innganger på komponenter / symboler blir brukt. Utganger kan gjerne stå ubrukt, men alle innganger må kobles til. Dersom en inngang skal være fast '1', eller '0', kan den knyttes direkte til VCC eller GND. Disse finnes som symboler under symbolverktøyet.

## Forbinde komponenter

For å tegne signaler bruk verktøyene «Orthogonal Node Tool» fra redigeringsverktøyene. Det er viktig å ikke forveksle dette verktøyet med lineverktøyet, som ikke lager elektriske forbindelser. Busser tegnes med «Orthogonal Bus Tool» .



**Figur 6-6:** Velge ledning under komponent

Dersom man bruker «Selection tool», vanlig pil, og plasserer den i enden, eller hjørnet av en allerede eksisterende buss, eller signal, vil pekeren endre seg til et kryss, men det aktuelle verktøyet under. Da er det bare å venstrekk og begynne å tegne. Det samme gjelder ved å plassere musa direkte over inn og utganger på komponenter. Men merk at det er ikke alltid denne automatikken velger riktig av signal og buss, samt at den er meget følsom for å treffe nøyaktig.

Dersom ledningen blir trukket for langt, er det mulig å bruke «selection tool» og velge det som er for mye, ved å

holde inne venstre museknapp og tegne en boks som dekker det som er tegnet for mye. Dette kommer godt til nytte dersom en ledning ligger for langt inn under en komponent. Komponenten blir ikke valgt med mindre området som velges fullstendig dekker komponenten, men ledningen under velges ved å dekke bare deler av komponenten. Figur 6-6 viser et slikt tilfelle. «Velgeboksen» tegnes da rundt den bakre delen av komponenten, og

ledningen som stikker opp ved nedre inngang vil da bli valg, og kan slettes ved å trykke delete. Merk at «Use Partial Line Selection»-knappen må være valg i redigeringsverktøyene .

Ledninger uten forbindelse videre indikeres med et kryss i enden av ledningen. Dette kan vi se på Figur 6-6, hvor både enden til venstre, og enden under komponenten er uten forbindelse. Når vi har en forbindelse på midten av en leder, vil det indikeres med en sirkel. Ledninger som krysser hverandre uten denne sirkelen er ikke forbundet.

## Signal og busser

Multi-inB[2..0]

Multi-inB[2]

MultiLinRT11

**Figur 6-7:** Buss og enkeltsignal

### Busser

Busser er en samling signal som tilsammen utgjør en binær vektor. En buss kan bestå av et vilkårlig antall signal med samme navn som bussen, men med forskjellig subnummerering. Eksempel på dette er en buss bestående 4 bit, og omtalt som et 4-bit signal. Si at vi kaller bussen vår TEST. Da vil vi i Quartus måtte navngi bussen vår TEST[3..0]. Dette indikerer at vi har signalene TEST[3], TEST[2], TEST[1] og TEST[0] som del av bussen. Merk at bussnummerering har to «punktum» mellom øvre og nedre signal. Vi kan også ta ut en del av denne bussen til en mindre buss, og navngi den TEST[3..1]. Av dette eksemplet ser vi også at vi ikke nødvendigvis må begynne en buss på index 0.

### Signal

Signal som ikke er del av en buss, kan ha et vilkårlig navn, men uten mellomrom og spesialtegn. Enkeltsignal kan hentes ut av en buss ved å angi bussens navn og deretter sette bit-/ledningsnummer som skal ut av bussen, i firkantparanteses [ ].

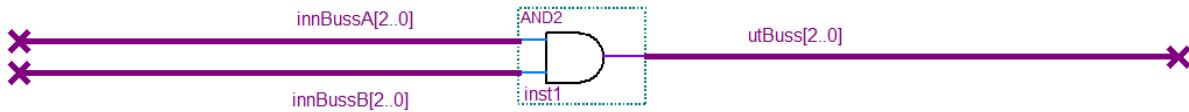
**MERK:** Signal med samme navn er koblet sammen, til tross for om at de ikke har noen fysisk ledning mellom. Navn på busser og signal må ikke inneholde mellomrom.

For å navngi en buss eller et signal, velg signalledningen og trykk på rettetasten på tastaturet (ikke «delete»). Det kan også gjøres med å høyreklikke og gå på «properties». Kun busser som splittes opp i enkeltsignal, samt inn- og utganger må navngis. For de øvrige signalene er det valgfritt.

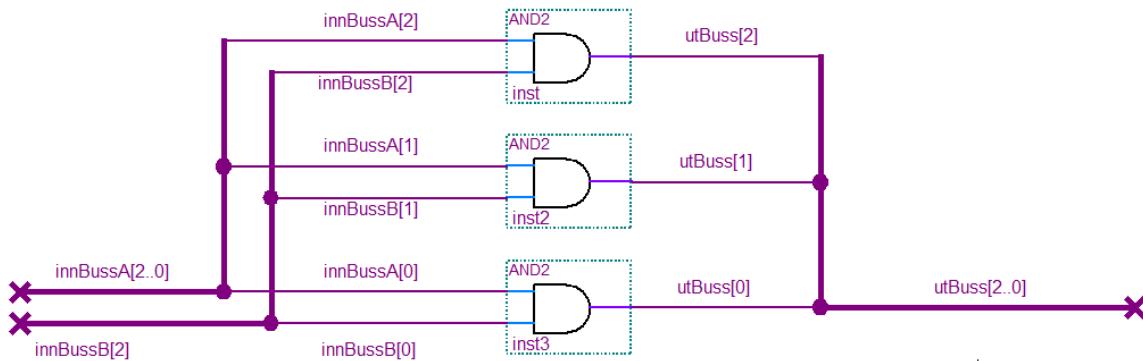
Figur 6-7 viser en buss, hvor det hentes ut / settes inn enkeltsignal. Merk at tykkelsen på bussstreken er tykkere enn enkeltsignal.

## Triks ved bruk av signal og busser

Dersom vi har en buss, hvor hvert signal skal følge samme enkle logiske funksjon, kan vi koble en buss direkte inn på en logisk komponent. Dette er et triks som kan spare mye tid. Men vær oppmerksom på at det er kun i helt entydige tilfeller dette kan gjøres. Under følger noen eksempler:

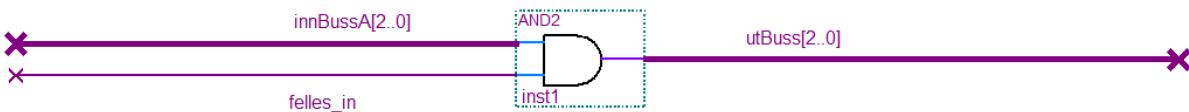


Figur 6-8 viser en OG-port med buss på innganger og utgangen. Dette tilsvarer at alle signalene med samme subindex går gjennom en og funksjon, som vist på Figur 6-9.

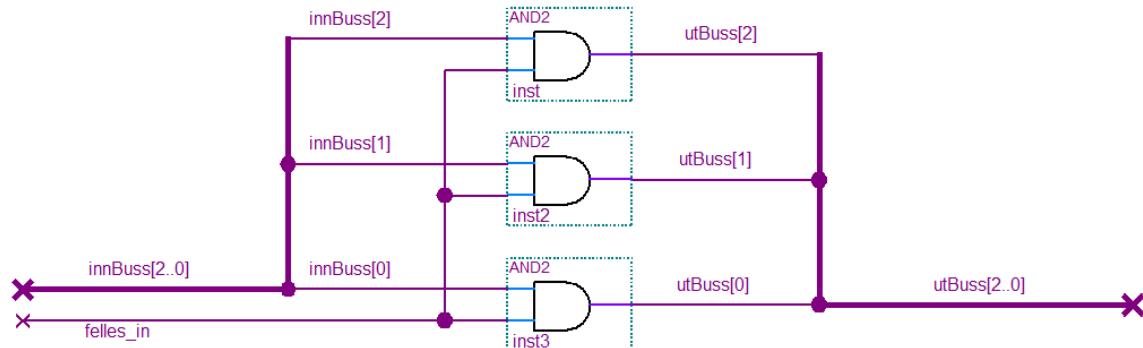


**Figur 6-9:** Full tegnet skjema for OG-funksjon mellom like indexer på en buss

For å bruke denne teknikken MÅ alle bussene ha samme bussbredde. Det vil si samme antall bit. Eneste unntak er når enkeltsignal brukes som en av inngangene. Da vil dette signalet bli brukt i en OG-port for hvert enkelt signal på bussen. Figur 6-10: Kompakt skjemateknikk for OG-funksjon mellom et enkelsignal og en buss viser et eksempel på dette. Tilsvarende som for over viser neste Figur 6-11.

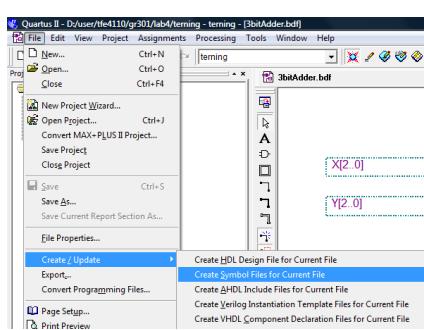


**Figur 6-10:** Kompakt skjemateknikk for OG-funksjon mellom et enkelsignal og en buss



**Figur 6-11:** Full tegnet skjema for OG-funksjon mellom et enkelsignal og en buss

## Blokksymboler - opprettelse og bruk.

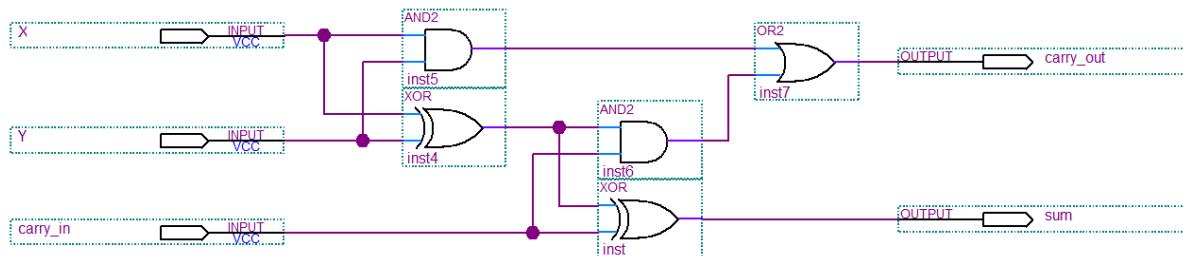


**Figur 6-12:** Menyvalg for å lage symbol av blokkskjema.

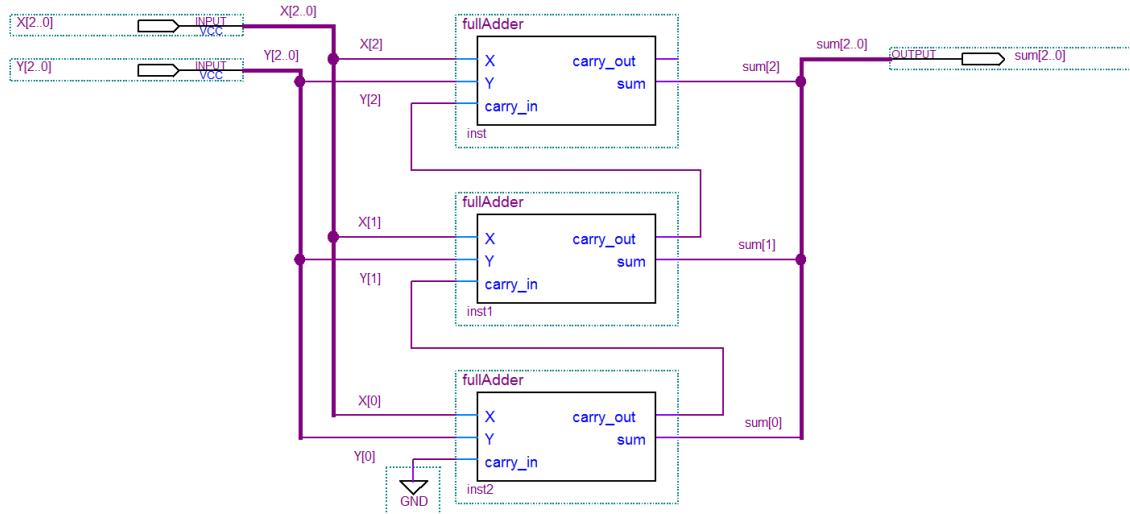
Når et blokkskjema er ferdig, med inn og utganger, kan det lages et blokksymbol av det. Dette er nyttig for store kretsskjema for å redusere kompleksiteten av hvert enkelt skjema. Samtidig kan funksjoner / skjemategninger som repeteres lett brukes om igjen. Eksempel på dette vil være å bygge en ripple-carry adderer ved å lage underblokker for en full-adderer. Disse blokkene kan da igjen brukes på neste skjemanivå for hvert enkelt bit, for å lage en fler-bits full-adderer.

Figur 6-13 viser et eksempel med en full-adderer. I Figur 6-14 er det satt sammen 3 full-adderer til en 3-bits full-adderer.

For å lage blokksymbol av denne, må vi gå inn under menyen og velge: File => Create/Update => Create Symbol Files for Current File. Se Figur 6-12. Lagre filen under filnavnet som blir foreslått.

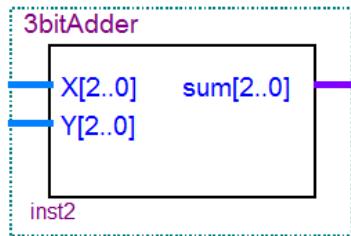


**Figur 6-13:** Full-addere med inn og utganger.



**Figur 6-14:** 3-bit full-adder med inn og utganger.

Igjen er det mulig å lage et blokksymbol for denne, og bruke videre i enda et høyere nivå. Som blokksymbol vil 3-bit adderer se ut som i Figur 6-15.



**Figur 6-15:** 3-bit full-adder som blokksymbol.

Merk at innganger og utganger blir lagt til i symbolet i den rekkefølge de er i skjemaet. Innanger havner på venstre side og utganger på høyre side av blokksymbolet.

## Enkle navigasjonstriks

Quartus har noen ganger problem med å definere arbeidsområdet riktig på skjemategninger. Resultatet av dette er at det ikke er mulig å flytte vinduet over hele skjemaområdet. For å flytte vinduet lengre til en side enn hva vindunavigatøren, eng. «scroll-bar», tillater, bruk pilene i enden av vindunavigatøren.

For å zoome inn og ut, hold nede Ctrl på tastaturet og bruk musens rullehjul.

Navigasjon sidelengs, hold nede shift og bruk musens navigasjonshjul. For navigasjon opp og ned, bruk rullehjulet uten å holde nede noen tast.

*Dersom fanene for underskjema, som normalt er lokalisert mellom skjemategningen og hovedverktøylinjen, forsvinner, kan den slås på igjen ved å gå i menyen under: Tools => Options. På første side, «General» finnes valget «Display tabs for child window». Merk av denne.*

## Oversikt over verktøy for blokkskjemaredigering

Ikon	Navn	Funksjon
	Detach Window	Frigjør det aktuelle vinduet fra Quartus
	Selection Tool	Velg ledninger og komponenter
	Text Tool	Legger til kommentartekst i designet
	Symbol Tool	Legger til nye symbol/komponenter
	Orthogonal Node Tool	Tegner signal
	Orthogonal Bus Tool	Tegner bussignal
	Orthogonal Conduit Tool	Tegner kanaler
	Use Rubberbanding	Gjør at ledninger henger fast i komponenter når de flyttes
	Use Partial Line Selection	Tillater delvis valg av ledninger
	Zoom Tool	Zoomer inn og ut. Hold nede shift for å zoome ut.
	Full Screen	Endrer vinduet til fullskjerm-modus.
	Flip Horizontal	Vender valgt komponent horisontalt.
	Flip Vertical	Vender valgt komponent vertikalt.
	Rotate Left 90	Roterer valgt komponent 90 mot klokken
	Rectangle Tool	Tegner rektangel
	Oval Tool	Tegner oval.
	Line Tool	Tegner linje. ( <b>ikke ledning</b> )
	Arc Tool	Tegner Bue. ( <b>ikke ledning</b> )

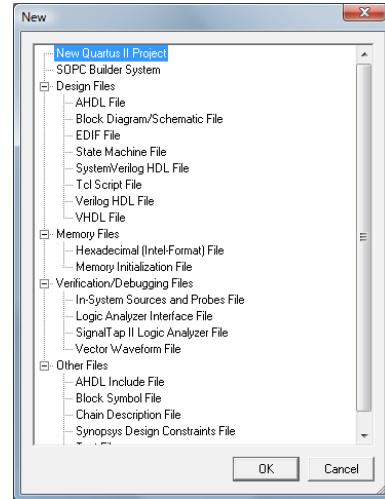
**Tabell 6-1:** Menyknapper i editoren

## Ofte brukte prosedyrer i Quartus

### Opprette nytt prosjekt

For å opprette et nytt prosjekt i Quartus, følg prosedyren under.

- Gå til menyen og velg: File => New
- Velg så «New Quartus II project» fra «New» vinduet som kom opp, Figur 6-16. Klikk deretter OK.
- Det vil da komme opp en rask introduksjon til veiviseren. Kan leses om man ønsker. Figur 6-17.
- Klikk 



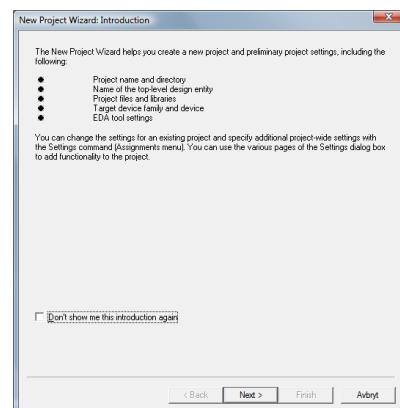
**Figur 6-16**

### Nytt prosjekt side 1/5

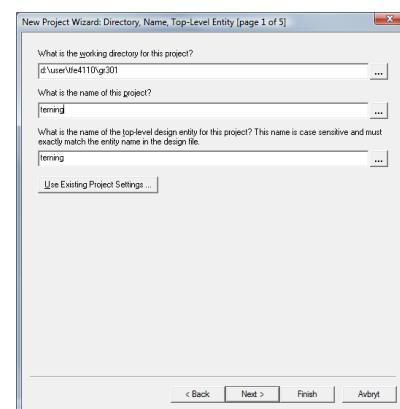
Skriv inn arbeidskatalogen angitt i oppgaven i øverste felt, se Figur 6-18. Alternativt kan man bla frem til katalogen ved å trykke på: 

Skriv deretter inn navnet på prosjektet ifeltet for prosjektnavn. Feltet for «top-level entity» vil da bli automatisk fylt inn med samme navn. Med mindre det eksplisitt er oppgitt i oppgaven at dette feltet skal endres, så la det være likt som prosjektnavnet.

Klikk 



**Figur 6-17: Nytt prosjekt veiviser informasjon**

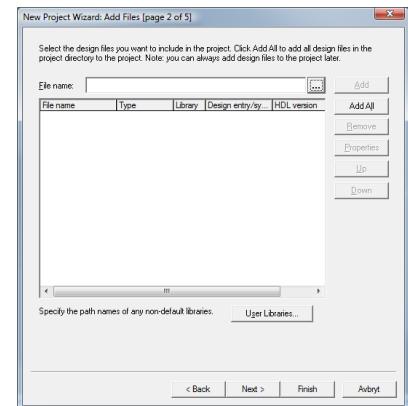


**Figur 6-18: Nytt prosjekt side 1/5**

## Nytt prosjekt side 2/5

Her kan vi legge til filer som vi ønsker skal være del av prosjektet vårt. Dette vil typisk være ferdiglagde filer som blir utlevert. Med mindre det står i eksplisitt i oppgaveteksten at filer skal legges til under opprettelsen av prosjektet, hopper vi over dette og klikker **Next >**

Dersom filer skal legges til bruk knappen **...** for å bla frem til filene. Husk å klikke «Add» etter å ha bladd frem til filen. Filen vil da vise i listen under.



**Figur 6-19:** Nytt prosjekt side 2/5

## Nytt prosjekt side 3/5

På denne siden må vi angi hvilken av Alteras FPGA-kretser vi ønsker å bruke i prosjektet.

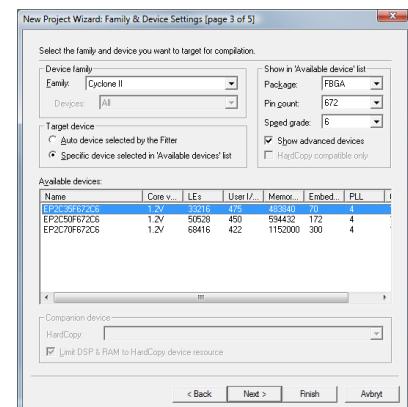
For DE2-kortene velg følgende:

- Family: Cyclone II
- Package: FBGA
- Pin Count: 672
- Speed Grade: 6

Vi står da igjen med 3 valg på listen. Velg «EP2C35F672C6»

Tallet 35 midt i koden, som for de andre to i listen er 50 og 70, angir hvor mange tusen logiske element som er på FPGA-en. For DE2 er det 35000.

Klikk **Next >**

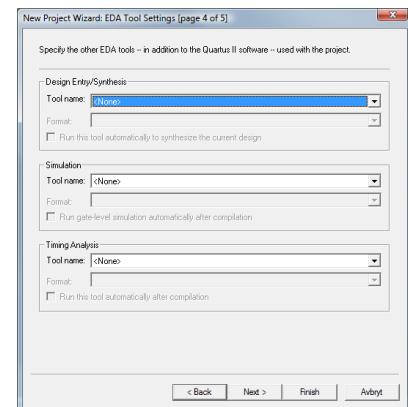


**Figur 6-20:** Nytt prosjekt side 3/5

## Nytt prosjekt side 4/5

Her kan vi spesifisere ekstraverktøy for Quartus. Vi bruker kun Quartus' egne verktøy slik at her skal alle felt ha valgt <none>.

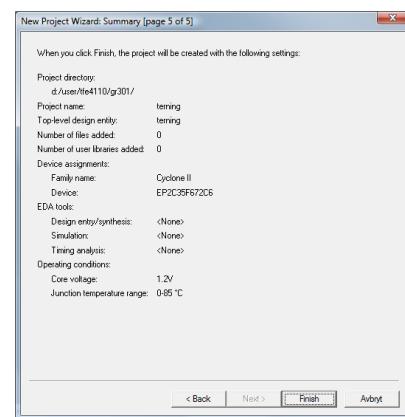
Klikk **Next >**



**Figur 6-21:** Nytt prosjekt side 4/5

## Nytt prosjekt side 5/5

Se over oppsummeringssiden og klikk **Finish** for å opprette prosjektet.



**Figur 6-22:** Nytt prosjekt side 5/5

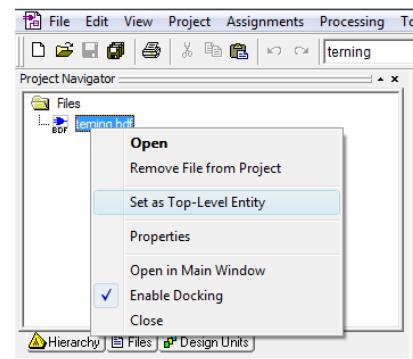
## «Top-level entity»

«Top-level entity» angir hvilket av blokkdiagrammene eller HDL-filene som et toppnivå for hierarkiet. Det er denne filen Quartus vil begynne på når den kompilerer og syntetiserer designet.

### Sette «top-level entity»

«Top-level entity» settes under fillisten i «Project navigator» vinduet. Pass på å velge «files» fanen under «Project navigator» vinduet først (ikke «file» menyen).

Høyreklikk på filen som ønskes som «top-level entity» og velg «Set as Top-Level Entity».



**Figur 6-23:** Top-level entity

## Definisjonsfil for pinne-tilordning

Ved å laste inn ferdiglagde definisjonsfiler i Quartus kan vi spare mye arbeid som ellers ville blitt brukt på å sette opp hvilke signaler som kommer ut på hvilke pinner på den fysiske kretsen.

Som eksempel på dette har vi navngiving av inn og ut pinnene på FPGA-en. Opprinnelig angis inn og ut pinner på en BPGA-kapsling ved hjelp av koordinater, hvor den første aksen er angitt med bokstaver og andre rad med tall. A1 vil være første pinne, mens pinne AB21 vil være pinne 29 ned og 21 bort. Dette blir det veldig vanskelig å holde styr på, spesielt med tanke på at FPGA-kretsen på DE2 har 672 pinner.

Terrasic, som har laget DE2 kortet, har derfor laget en definisjonsfil for pinnene slik at vi kan bruke navn som LEDG[1] for å angi LEDG nr. 1, i stede for pinnekoder som C21.

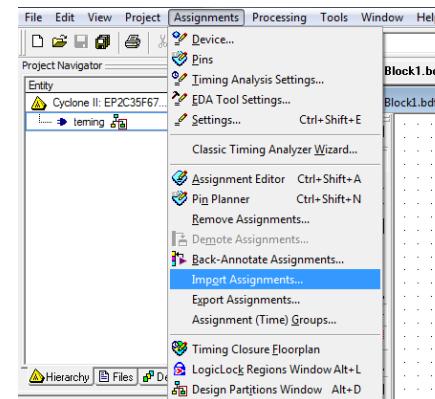
### Gjør følgende for å laste inn definisjonsfila:

Fra menyen, velg Assignments => Import assignments.

I vinduet som kommer frem, velg og finn filen med definisjonene. For pinndefinisjonene er det fila: «DE2\_pin\_assignments.csv».

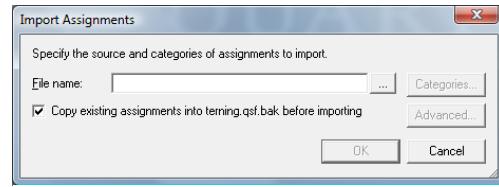
«Copy existing assignments»... osv. kan gjerne være avmerket.

Klikk



**Figur 6-24:** Importere definisjonsfiler menyvalg

I meldingsvinduet skal det da komme en melding som begynner med «Info: Import Completed»



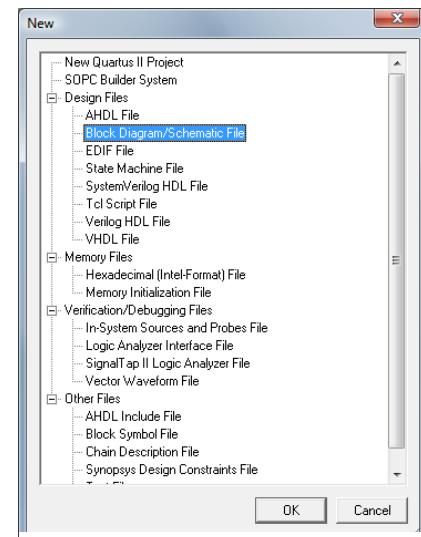
**Figur 6-25:** Importere definisjonsfiler

### Ny blokkdiagramfil

Velg fra menyen: File => New.

I «New» vinduet, velg «Block Diagram/Schematic File»

Klikk



**Figur 6-26:** Ny blokkdiagramfil

## Laste designet over på DE2 - «programmere»

Sjekk at DE2 er slått på og at USB kabelen er koblet til PC-en og «Blaster» inngangen på DE2.  
See delkapittel om oppkopling av DE2.

Kontroller at «RUN/PROG» bryteren på DE2 kortets venstre side ved LCD-skjermen er satt til «RUN»

Start programmereren. Programmereren finnes under menyen Tools => Programmer

Alternativt kan den startes fra verktøylinjen med ikonet 

Når programmereren har startet, kontroller at den er satt opp med «USB-Blaster» programmereren<sup>1</sup>.

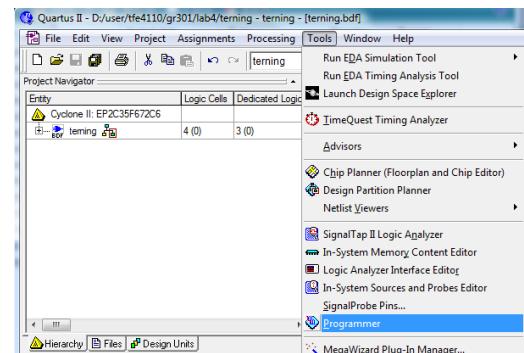
Dersom det allerede står «USB-Blaster», Figur 6-30, klikk «Start» for å starte programmeringen.

Dersom det står «No-Hardware» ved siden av «Hardware Setup» knappen, Figur 6-28, har ikke programmereren kontakt med USB-Blaster-en.

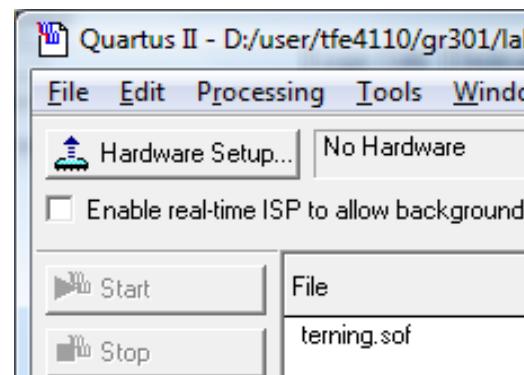
For å koble til USB-Blaster-en klikk på «Hardware Setup» knappen. I Hardware Setup vinduet, velg USB-Blaster fra nedfallslisten «Current Selected Hardware».

Klikk deretter «Close»

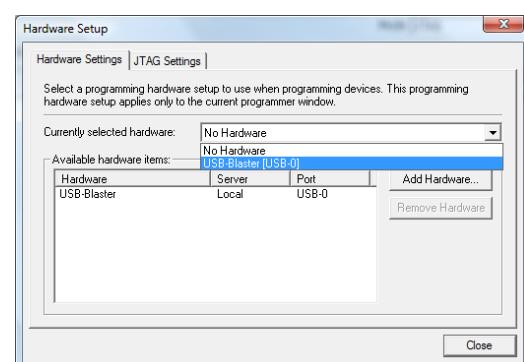
Dersom USB-Blaster ikke viser i listen, lukk hele programmereren. Sjekk at alle ledninger er koblet riktig og at DE2 er påslått. Prøv så igjen. Virker det fremdeles ikke, hent en studentassistent.



**Figur 6-27:** Programmerer menyvalg



**Figur 6-28:** Programmerer uten hardware

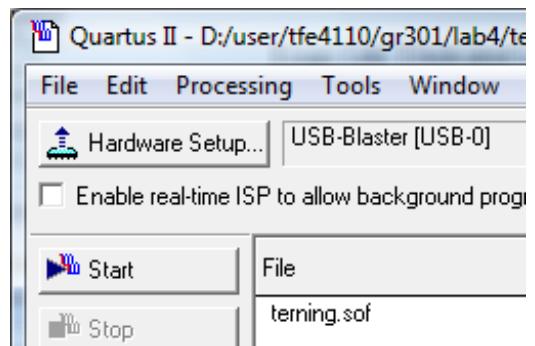


**Figur 6-29:** Valg av hardware til programmereren

<sup>1</sup> USB-Blaster er Alteras programmerer for FPGA-er. Den er integrert på DE2 kortet.

Med USB-Blaster i feltet ved siden av «Hardware Setup» knappen, er programmereren klar til bruk

Klikk «Start» for å starte programmeringen.

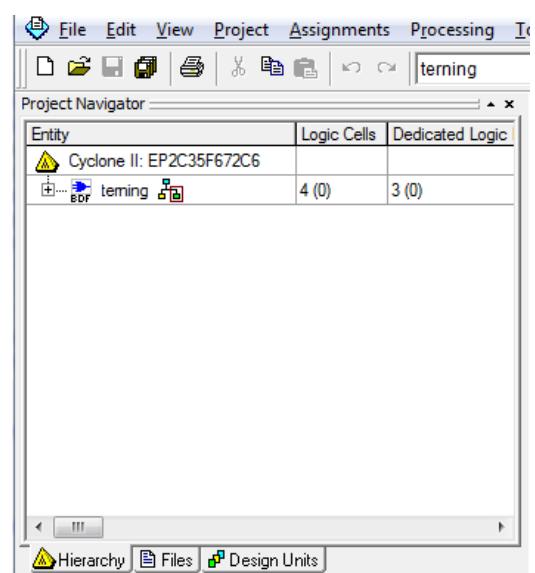


**Figur 6-30:** Programmerer klar til programmering

## Kompilere designet

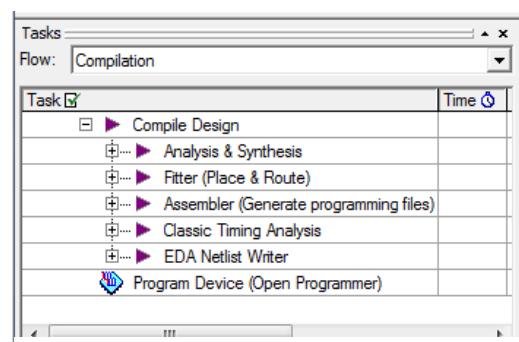
Sjekk at riktig design er satt som «top-level entity». Dette kan sjekkes ved å klikke på «Hierarchy» fanen i «Project navigator» vinduet. Filen som står under «Cyclone II EP2C35F672C6» er toppnivåentiteten.

Klikk på ikonet i verktøylinjen, for å starte kompilatoren. Alternativt dobbeltklikk på «Compile Design» i «Task menu».



**Figur 6-31:** Toppnivåfil i designhierarkiet

Under kompileringen er det vanlig at det kommer flere «warnings». I de aller fleste tilfeller kan disse ignoreres. Når kompileringen er ferdig, sjekk at det ikke har kommet noen feil. Eksempel på sluttlinje i «Messages»-vinduet:



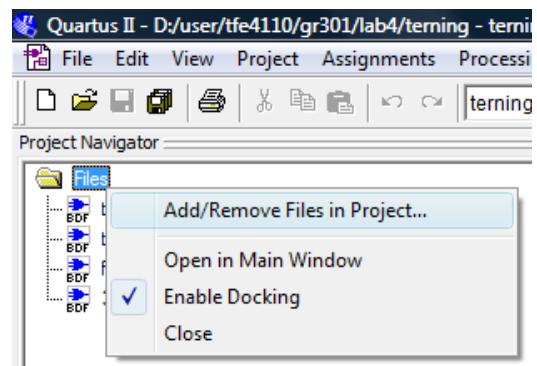
**Figur 6-32:** Task-vinduet med kompileringsvalg.

Info: Quartus II Full Compilation was successful. 0 errors, 439 warnings.

## Legge til filer i prosjektet

I «Project navigation» vinduet, klikk på fanen files. Høyreklikk på den øverste mappen med navn files, og velg «Add/Remove Files in Project» fra menyen. Se Figur 6-33.

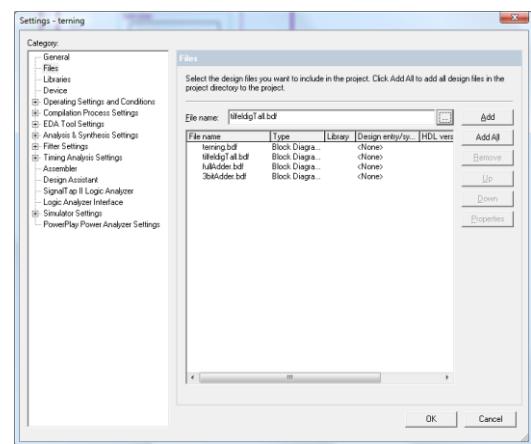
Klikk deretter  for å bla frem til filen som skal legges til.



**Figur 6-33:** Menyvalg for å legge til eller fjerne filer

Etter at filen er valgt, må add velges for å legge filen til i listen under. Alternativt kan knappen «Add All» brukes om alle filene i katalogen for den fila som er valgt, skal legges til.

Gjenta dette for alle filer som skal legges til. Deretter klikk OK.



**Figur 6-34:** Vindu for å legge til og fjerne filer i prosjektet.



# 7 Oppkobling og bruk - Altera DE2

## **Bruk av antistatisk armlenke**

DE2-kortene er følsomme mot statisk elektrisitet. Derfor må det brukes antistatisk armlenke under alt laboratoriearbeid som innbefatter bruk av DE2-kortet. Videre er det viktig å ikke berøre kortets kontakter, kontaktpunkter eller kretser unødvendig.

**Antistatisk lenke skal ALLTID brukes så lenge DE2-kortet befinner seg på labplassen.**  
 Armlenken festes rundt armleddet og strammes ved behov til passe størrelse.  
 Armlenken skal ikke henge løst rundt armen.



(a) Antistatisk armlenke

(b) Fest rundt håndleddet

(c) Tilkobling til DE2

**Figur 7-1:** Bruk av armlenke

(a) Til PC-en

(b) Til DE2

**Figur 7-2:** Oppkobling av DE2 kortet

Armlenken plugges inn i bunnplaten på DE2-kortet. Det er 2 jordingskontakter på hver side av DE2-kortet. Jordingen til DE2-kortet kommer gjennom USB-kabelen. Det er viktig at DE2-kortet kobles til PC-en før strømforsyningen settes i.

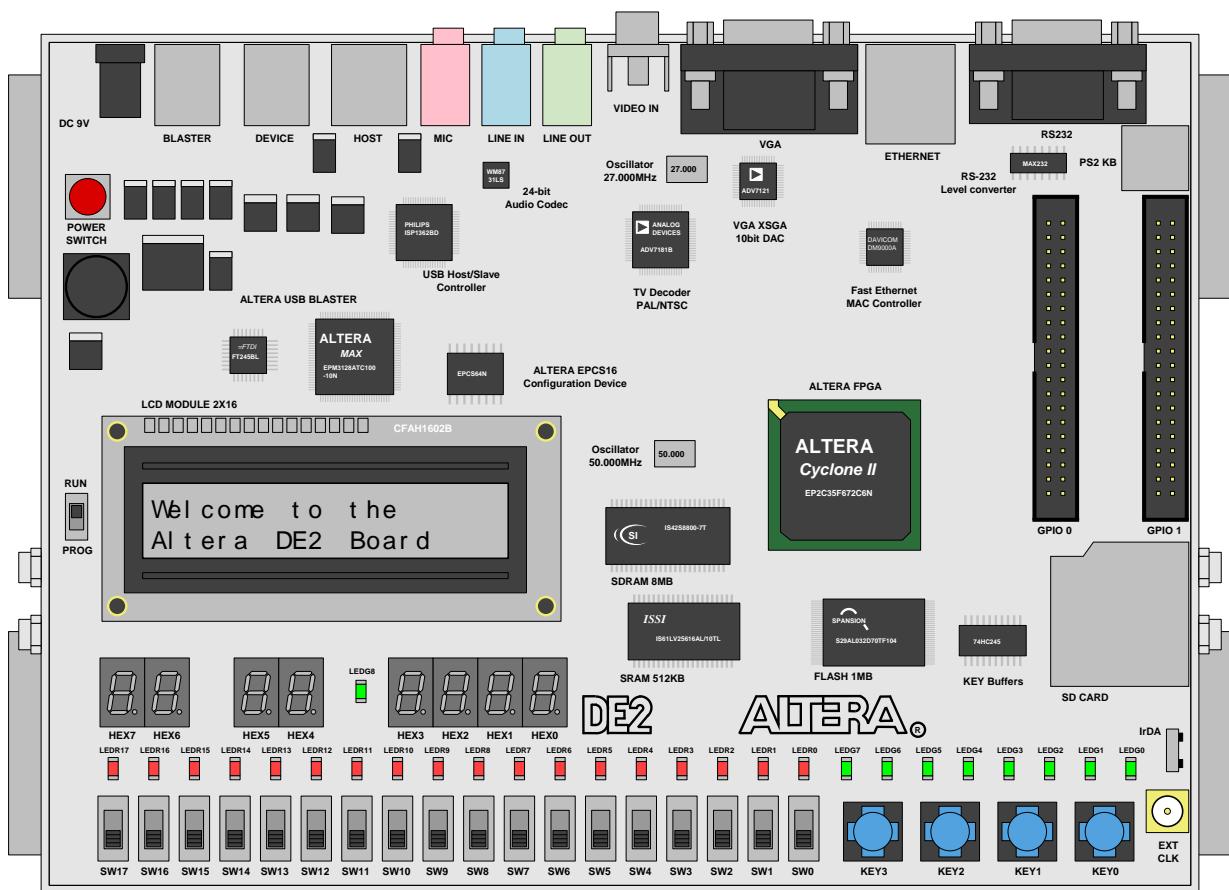
Bruk USB-kontakten på forsiden av PC-en.

## ALTERA DE2 - oppkobling og bruk

### Oppkobling av DE2

**Ta først på de antistatiske lenkene.** Deretter sett i USB-kontakten til PC-en. DE-2 kortet har 3 USB-kontakter. To USB-B-kontakter som er kvadratiskformet, og en USB-A kontakten som er rektangulær. For tilkobling til PC, er det USB-B-kontakten merket med «BLASTER» som skal brukes. Til slutt kobles strømmen til «9V DC»-pluggen.

**Når tastatur skal brukes, skal denne normalt kobles til «PS2 KB»-kontakten på kortets side, og ikke USB-A-kontakten med mindre det står i oppgaven.**



Figur 7-3: Altera DE2

For vanlig programmering må «RUN/PROG»-bryteren til venstre for LCD-skjermen på DE2-kortet, stå på i posisjon «RUN».

Kortet er utstyrt med 18 generelle brytere, navngitt SW0 til SW17, og 4 trykkbrytere merket KEY0 til KEY3. I tillegg kommer 9 grønne LED med navn LEDG0 til LEDG8, og 18 røde LED med navn LEDR0 til LEDR17. Den 9. grønne LED-en er lokalisert mellom 7-segment-tallene.

Dersom definisjonsfila for inn- og utganger fra Terasic er brukt, brukes følgende inn- og utgangsnavn i toppnivåentiteten for å få tilgang til disse:

Trykkbryterne, bryterne og LED-ene er satt opp som en buss. Det medfører at SW0 må angis som SW[0], SW1 som SW[1] og videre. Busser kan kobles direkte til disse ved å angi SW[3..0] for å koble SW3 til SW0 til en 4-bit buss. Det samme gleder for KEY, LEDG og LEDR.

**MERK:** KEY-inngangene er aktiv-lav. Det vil si de blir logisk 0 når de trykkes ned, og er logisk 1 når de ikke er trykket. Lysdiodene er aktiv-høy.

For 7-segmentene er hvert segment en egen buss. signal nummer 0 tilsvarer segment a, signal 1, segment b, osv. Bussene er navngitt HEX0[6..0] for segment merket HEX0, HEX1[6..0] for segment HEX1, og tilsvarende for de resterende opp til HEX7.

**MERK:** 7-segmentene er aktiv-lav. Det vil si at de lyser når utgangen er 0, og er slukket når inngangen er 1. For å «slukke» alle diodene, må alle 7-segment-utgangene settes til logisk 1.

Alle datablad for Alteras DE2-kort er tilgjengelig på laboratoriets hjemmeside. Der finnes også definisjonsfilen for inn- og utganger fra Terasic. Hjemmeside for laboratoriet er <http://intern.iet.ntnu.no/p15kretslab/index> På de etterfølgende sidene vil oppgavene bli listet opp. Veiledering til hvordan de forskjellige punktene utføres finnes på sidene etter oppgavebeskrivelsen.

**MERK:** Quartus har forskjellige typer «ledninger» som kan tegnes: Signal, buss og conduit. Vi skal kun bruke signal og buss. Buss brukes når navnet angis med .. mellom tallene i indeksen, for eksempel for bussen S[3..0]. Et enkelt signal på den samme bussen vil være signalet S[3] eller signalet S[2]. Vær nøyne med dette og bruk det rette «ledningstegningsverktøyet» i Quartus.

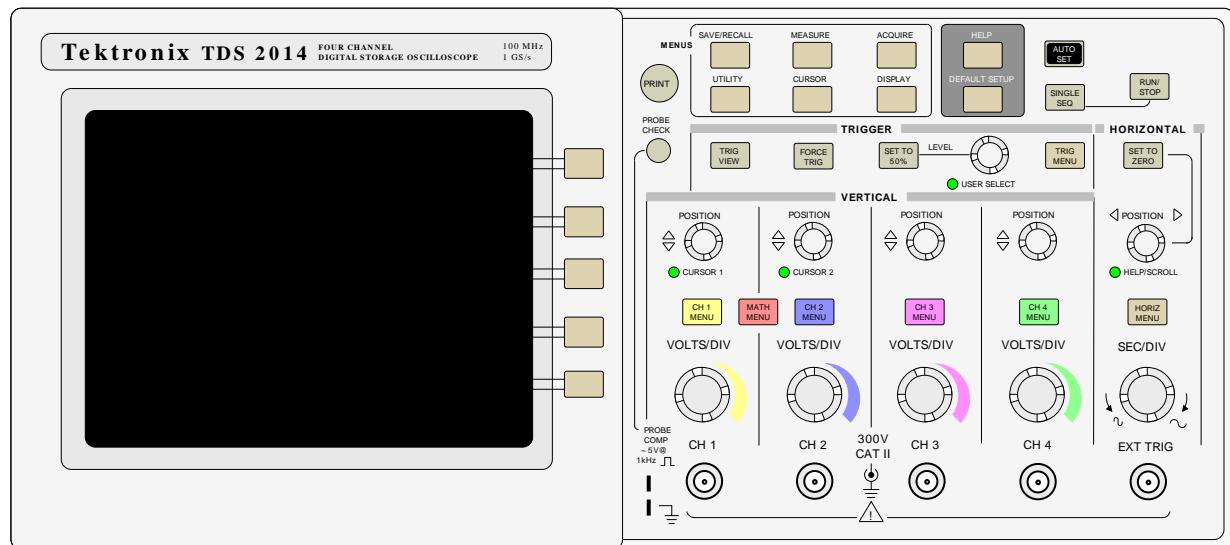


## 8 Tektronix TDS-2000-serien oscilloskop

Oscilloskopet er et måleinstrument som primært lar oss se spenning som funksjon av tid. Oscilloskopet måler kun spenning, og vil kun vise oss målinger som innbefatter spenning og tid. Dersom vi for eksempel ønsker å måle strøm, må vi måle spenningsfallet over en gitt motstand, og så regne oss til strømmen. På grunn av at vi har spenning over tid, eigner oscilloskopet seg meget bra til måling av periodetid, fall- og stigetid og andre tidsavhengige spenningsmålinger.

Oscilloskop har tradisjonelt sett vært basert på en elektronstråleskerm. Oppbygningen gikk på at elektronstrålen hadde en fast hastighet over skjermen i x-retning, og spenningen på inngangen styrte elektronstrålens y-retning. Ved å variere hastigheten strålen beveget seg i x-retning, samt hvor mye forsterkning inngangssignalet hadde i x-retning, kunne oscilloskopet måle saktevarierende, hurtigvarierende, sterke og svake signal.

Dagens oscilloskop gjør samme jobb, og er bygget på samme prinsipp, men gjør jobben digitalt. Overgangen til digitale oscilloskop har gjort de fleste målinger enklere og mer presise. Men overgangen er ikke uten problem. Vi vil helt i slutten av denne veilederingen nevne noen av problemene vi har med digitale oscilloskop.



**Figur 8-1:** Tektronix TDS2014

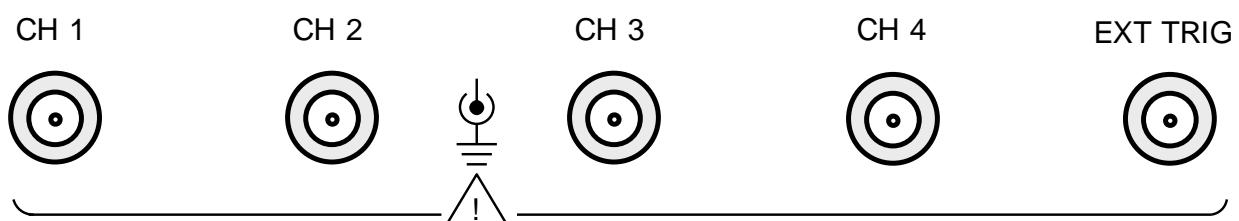
Oscilloskopet på laboratoriet er et Tektronix TDS-2000. Eksakt modell varierer litt mellom laboratorieplassene. Alle oscilloskopene har 4 kanaler. Noen oscilloskop har mulighet for USB-minnepenn, mens andre står oppkoblet til PC. Denne beskrivelsen baserer seg på TDS-2014 med PC-tilknytting. Oscilloskopene med USB-minnepennmulighet kan ha enkel menyavvik fra denne beskrivelsen, men er i de fleste tilfeller identiske.

TDS-2000-serien fra Tektronix er en avansert serie oscilloskop godt egnet for opplæringsformål. Menysystemet er relativt enkelt å navigere og funksjonene er begrenset i kompleksitet. På grunn av oscilloskopets relativt enkle oppbygging oppfordres dere til å prøve dere litt frem selv. Oscilloskopet kan ikke ødelegges av menyinnstillinger. Skulle dere rote dere bort i menyer, kan oscilloskopene lett tilbakestilles til standardinnstillinger. Se avsnittet: «Hjelp og tilbakestilling av oscilloskopet»

Til tross for at menyinnstillinger ikke kan ødelegge oscilloskopet, kan uvettig tilkobling gjøre det. Av den grunn må oscilloskopets måleinnganger ALDRI kobles til nettspenningen. Videre bør man være forsiktig med måleprobene. Les gjennom avsnittet om måleprober før disse tas i bruk.

## Tilkoblinger

Oscilloskopet har 5 BNC-tilkoblinger på fremsiden. Tilkoblingene CH1 til CH4 går henholdsvis til kanal 1 opp til kanal 4.



**Figur 8-2:** Oscilloskopets innganger.

Tilkoblingen merket med EXT-TRIG er for eksternt triggingsignal. Ekstern triggings kan aktiveres fra tigger-menyen. Normalt vil ikke denne inngangen bli benyttet.



Vær oppmerksom på at jord på innsignalet er koblet til nettspenningens jord. Jordsignalet på alle inngangene er derfor også koblet sammen. Se avsnittet om måleteknikk for videre forklaring.

For bruk av oscilloskopets probene, les avsnittet om probene for å bruke probene riktig.

## Prosedyre for enkelt måleoppsett

- Start målingen med å tilbakestille oscilloskopet. Trykk på knappen «Default setup». Funksjonen tar ca. 5 sekunder for å fullføre.
- Trykk på «CH1 MENU» og kontroller innstillingen «Probe». Dersom måleprobe benyttes, og dempeleddet på proben er satt til 10X, skal probe stå på 10X. Brukes ikke måleprobe, eller at probens dempeledd er satt til 1x, skal probe stå på 1X.
- Kontroller innstillingen «Coupling». Normalt vil «DC» være innstillingen som skal brukes. Dersom et svakt vekselspenningssignal som i tillegg har en stor likespenning skal måles, må «AC» velges. «AC» medfører at likespenningskomponenten fjernes ved å koble en kondensator i serie med signalet, internt i oscilloskopet. GND er for å se hvor nullpunktet er.
- Juster VOLTS/DIV og POSITION til spenningskurven dekker minst halve skjermen, men ikke går utenfor.
- Trykk på «SET TO 50%» under trigger-knappene eller juster LEVEL til ønsket triggennivå.
- Juster SEC/DIV til skjermen viser mellom en og tre perioder av signalet. Alternativt mellom 1 til 3 ganger det måleområdet som er av interesse.

## Kontroller validiteten av måleresultatet - Volt/div og sec/div

Det er lett å gjøre en feil måling, og ta målingen for gitt. Derfor er det viktig at man i ettertid kontrollerer målingens validitet. I nedre venstre hjørne er volt/div angitt etter CH x. Volt/div sier hvor mange volt en rute er i y-retning. Bruk dette til å estimere det målte signalets spenning og vurder om dette kan stemme. Gjør samme prosedyre for signalets periode. Under midten av rutenettet står verdien for sec/div. Sec/div angir sekunder på rute i x-retning. Kontroller det målte signalets periode og eventuelt frekvens, ut fra det forventede signalet.

## Lagring av oscilloskopbilder på PC

Oscilloskopene som er koblet til PC, lar brukeren lagre både skjermbilder og måledata fra oscilloskopet. Programmet OpenChoice Desktop er Tektronix' eget program for kommunikasjon med oscilloskopet. Programmet finnes både på datamaskinens skrivebord og på startmenyen.

Etter oppstart av programmet, må oscilloskopet velges ved å gå inn på «Select Instrument». Oscilloskopet kommer somregel opp som «ASRL1:INSTR», men dette kan variere. Velg oscilloskopet og klikk OK.

De forskjellige fanene i programmet representerer forskjellige måter å hente data på fra oscilloskopet. For rapportskriving egner skjermdump seg best. Skjermdump fåes under fanen «Screen Capture».

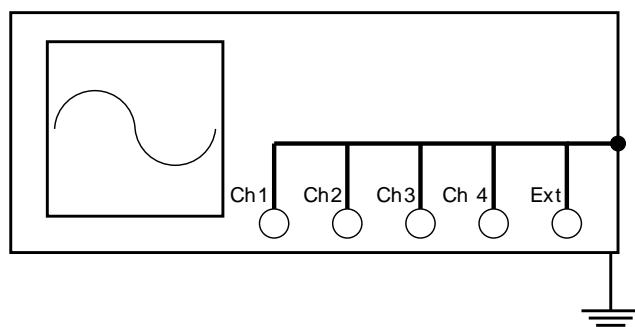
For å ta skjermdump, klikk på «Get Screen». Denne prosessen tar tid, opp til 1 minutt. Bruk «Save As» for å lagre. For ikke å få for store rapportfiler anbefales det å bruke filformatet .PNG.

## Måleteknikk

Ved målinger med oscilloskop er det nødvendig å være klar over endel begrensninger og forholdsregler som må overholdes.

### Referansepunkt

Inngangene på oscilloskopet er knyttet sammen i et felles referansepunkt som vist i **Figur 8-3:** Felles referanse på oscilloskopets innganger..

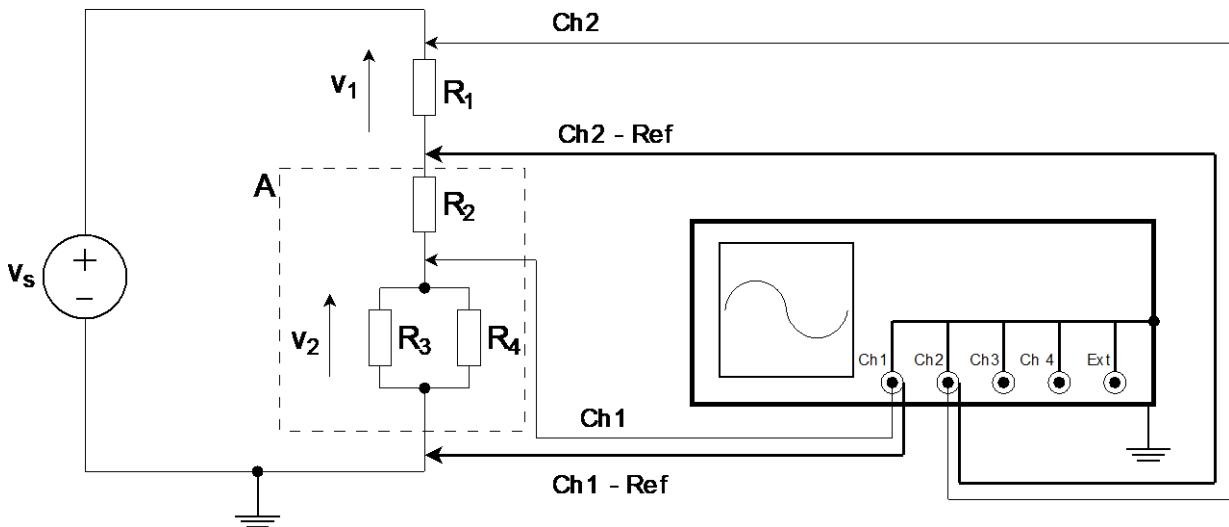
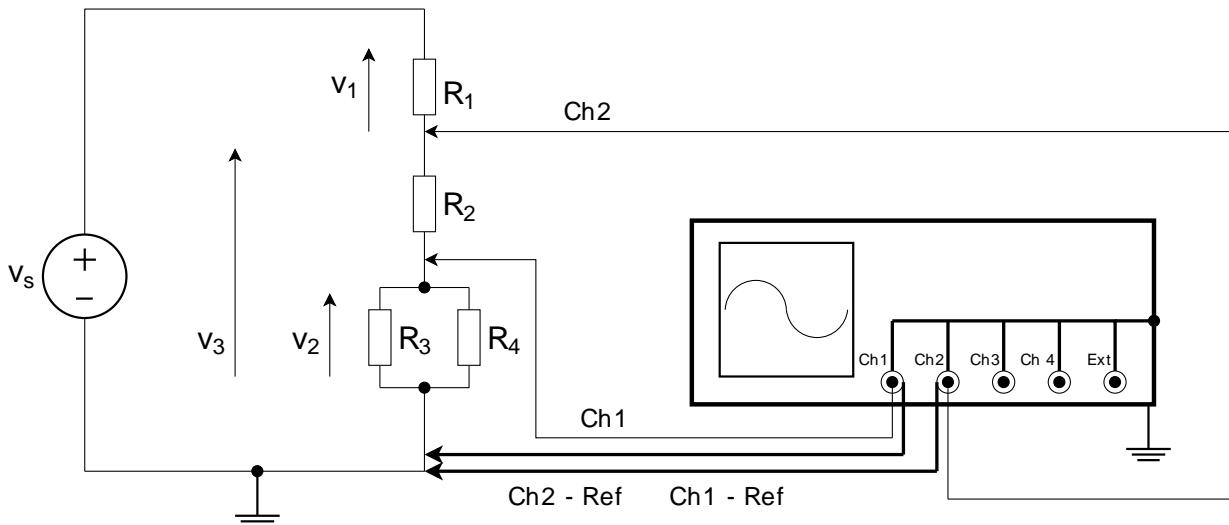


**Figur 8-3:** Felles referanse på oscilloskopets innganger.

Denne referansen er viderekoblet til oscilloskopets kasse, som igjen, av sikkerhetsmessige årsaker, er koblet til strømnettets jord. Ved måling med oscilloskopet i en krets vil derfor referansepunktet for alle inngangskanalene være knyttet sammen med strømnettets jord. Denne felles referanse begrenser de mulighetene for å måle fritt med oscilloskopet. Se **Figur 8-4:** Feil bruk av oscilloskopet.

Referansepunktet er her koblet til to ulike punkter i kretsen. Dermed kortsluttes de komponentene som ligger mellom referansetilkoblingene. Kortsluttet område er merket med A på figuren. Uthevet linje viser kortslutningens vei gjennom referansen til oscilloskopet. Resultatet av denne koblingen er at kretsen det skal måles på har blitt drastisk forandret. Dette kan også medføre store feilstrømmer gjennom oscilloskopet.

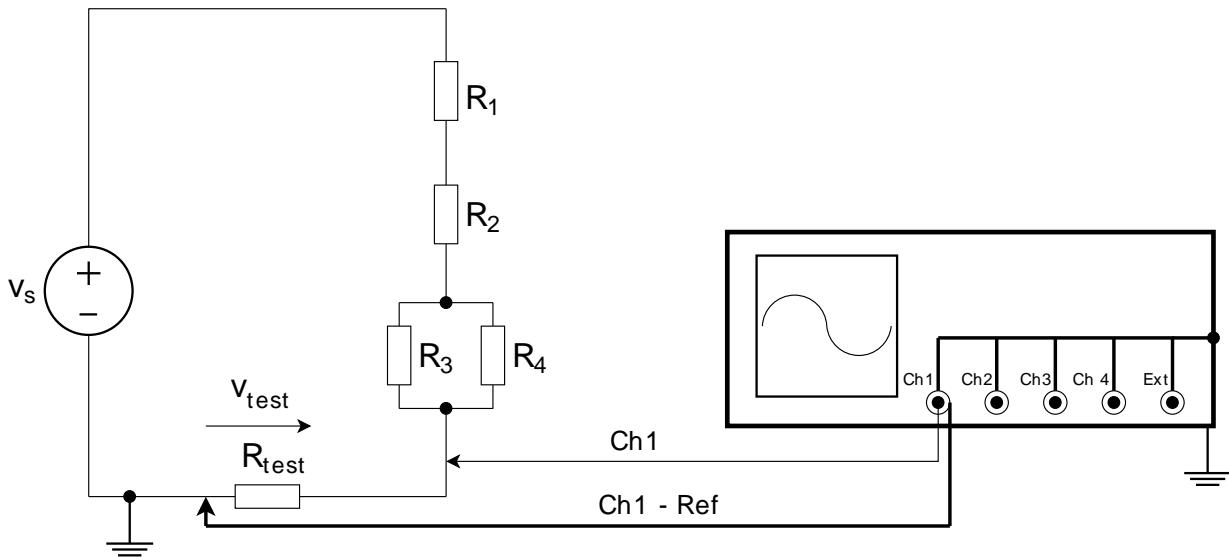
Korrekt bruk av oscilloskopet vil være som gjengitt i **Figur 8-5:** Korrekt bruk av oscilloskopet.. For å finne de ønskede spenningene  $v_1$  og  $v_2$ , måles  $v_2$  direkte, mens  $v_1$  beregnes som differansen  $v_s - v_3$ . For å være sikker på at det ikke gjøres feil med koblingen av referanse, kan det ved lave frekvenser under 100 kHz, og sterke signal over 10mV, med fordel benyttes kun én tilkobling til oscilloskopets referanse. I slike tilfeller er det unødvendig å benytte begge inngangenes referansesignal.

**Figur 8-4:** Feilbruk av oscilloskopet.**Figur 8-5:** Korrekt bruk av oscilloskopet.

## Strømmåling

Oscilloskopet i seg selv kan kun måle spenninger. For å måle strøm må vi innføre et system som omgjør strømmen vi skal måle, til spenning. Dette gjør vi ved å sette inn en målemotstand i målekretsen. Denne motstanden må være liten, slik at den får minimal virkning på den opprinnelige kretsen. Strømmen finnes ved å måle spenningen over den innsatte motstanden, og deretter beregne strømmen ved hjelp av Ohms lov.

Samme teknikk som forklart bruk her, bruker et multimeter for å måle strøm.



**Figur 8-6:** Strømmåling ved hjelp av oscilloskop.

## Sikkerhet

Oscilloskopets innganger er følsomme for høye spenninger. Spenningen i strømnettet overgår den maksimale innspenningen. I tillegg representerer sammenkoblingen av referansepunktet med nettstrømmens jord en fare for kortslutning som er kraftig nok til å ødelegge oscilloskopets inngang. Av denne grunn må oscilloskopet aldri benyttes til måling av nettstrøm eller spenning uten spesialutstyr.

## Betjeningsknapper

Oscilloskopets betjeningsknapper er delt inn i 5 områder:

**Vertical** - Kanalknapper. Tegning av innspenningen langs Y-aksen.

**Horizontal** - Tidsaksen - Hastigheten tegning skjer langs X-aksen.

**Trigger** - Når opptegning starter.

**Menus** - Undermenyer for innstillinger og avansert funksjonalitet.

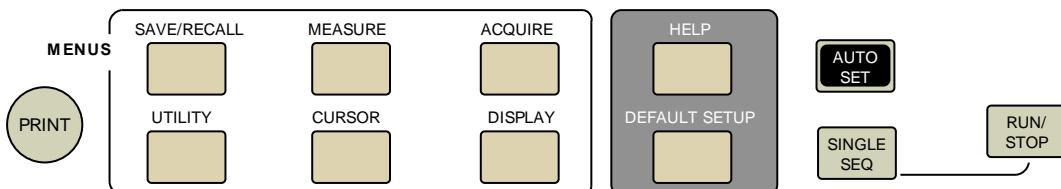
**Skjerm taster** - Varierer med hvilken meny som er åpen. Funksjon viser på skjermen.

Hver seksjon med betjeningsknapper vil, hver for seg, bli gjennomgått på de følgende sider.

## Hjelp og tilbakestilling av oscilloskopet

Siden oscilloskopene på laboratoriet står i et læringsmiljø hvor det oppfordres til litt prøving og feiling på egenhånd, skjer det ofte at oscilloskopets innstilling kan være helt på tur.

En av de første knappene dere bør kjenne til er derfor knappen «DEFAULT SETUP». «Default setup»-knappen stiller oscilloskopet tilbake til standardinnstillingen. «Default setup» tar ca. 5 sekunder. Ved å trykke på  er man tilbake til standardmenyen for kanal 1.



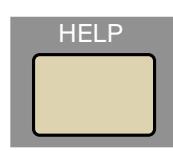
**Figur 8-7:** Betjeningsknapper - menyradene

Standardinnstilling vil si at kun kanal 1 er på skjermen. Videre blir kanalen satt til 1 volt pr divisjon. Tidsaksen blir satt til 500 $\mu$ s pr. divisjon. Trigging blir satt til kanal 1, stigende flanke ved passering gjennom origo.



**OBS:** Default settings antar at du bruker en måleprobe med 10 ganger demping. Inngangen blir derfor satt til å gange inngangsspenningen med 10. Dersom det brukes ren BNC-kabel, eller måleprobe som ikke har dempeleddet påslått MÅ denne innstillingen endres for å få riktig måleresultat.

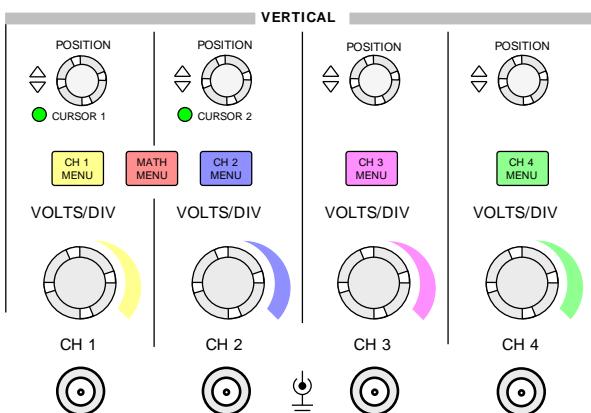
Dersom «Default Setup» feilaktig ble trykket, kan man trykke på «Undo Default Setup» I skjerm-menyen. Dette vil bringe oscilloskopet tilbake til de innstillingene det hadde før «Default Setup» ble trykket.



Help-knappen viser oscilloskopets innebygde hjelpe-meny. Ved å trykke på help, kommer hjelpe-siden for menyen / siden du var på automatisk opp. Videre navigasjon i hjelpe-menyen foregår med skjermtastene og posisjons-skru-knappen under «horizontal». Posisjons-skru-knappens endrede funksjon indikeres ved at lysdioden under skru-knappen lyser.

De øvrige knappene i menyradene vil bli gjennomgått senere.

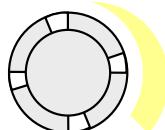
## Betjeningsknapper under «VERTICAL»



**Figur 8-8:** Betjeningsknapper for kanaler.

Tilsvarende knapper på alle kanalene har i utgangspunktet samme funksjon. Posisjonsknappene for kanal 1 og 2 kan utføre alternativ posisjonering på ekstramarkører når disse aktiveres. Når posisjonsknappene har alternativ funksjon, lyser diodene merket «CURSOR 1» og «CURSOR 2». Kanaler aktiveres ved å trykke på «CH1 MENU». Dersom kanalmenyen allerede er fremme på skjermen, og «CH1 MENU» trykkes igjen, deaktiveres kanalen.

VOLTS/DIV

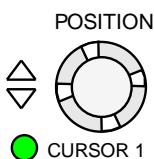


CH 1

Ved å skru på denne skruknappen endres skaleringen langs Y-aksen på oscilloskopet. Signalet kan ved hjelp av denne gjøres større eller mindre.



Åpner menyen for kanalinnstillingene. De forskjellige innstillingene kommer opp på skjermen og kan betjenes med skermknappene.

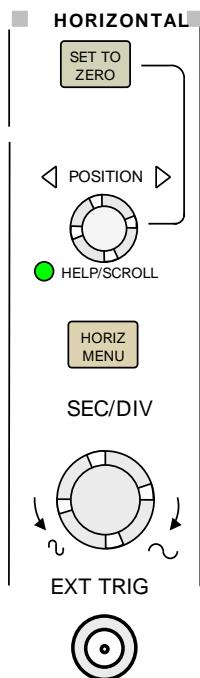


Flytter kanalens nullpunkt opp eller ned langs Y-aksen. Det nåværende nullpunktet vises ved hjelp av en pil i kanalens farge i venstre side av skermens rutenett.



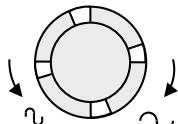
Åpner menyen for matematiske funksjoner. Under denne menyen ligger funksjon som addisjon og subtraksjon av to kanaler, samt FFT-analyse av ønsket kanal. De forskjellige innstillingene kommer opp på skjermen og kan betjenes med skermknappene.

## Betjeningsknapper under «HORIZONTAL»



**Figur 8-9:** Betjeningsknapper for tidsaksen.

SEC/DIV

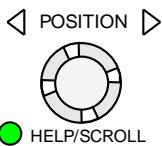


SEC/DIV-skru-bryteren velger hvor mange sekund pr. rute langs X-aksen oscilloskopet har. Ved å skru på denne vil kurven strekkes eller forminskes i X-retning.

EXT TRIG



Åpner menyen for tidsinnstillingene. De forskjellige innstillingene kommer opp på skjermen og kan betjenes med skermknappene. Disse innstillingene ansees for å være utenfor de elementære forklaringene i denne veilederingen og det henvises derfor til oscilloskopets hjelpe-menü eller manual for forklaring.



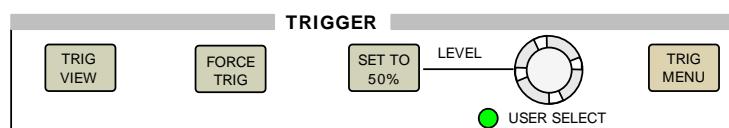
Skrubbryteren justerer forskyvningen av nullpunktet ut fra der hvor triggningen til oscilloskopet skjer. Justering av denne vil flytte kurven langs X-aksen.



Denne knappen nullstiller X-aksens nullpunkt til å ligge i origo.

## Betjeningsknapper under «TRIGGER»

Triggerknappene styrer når på en signalkurve skjermen begynner å tegne signalet.



**Figur 8-10:** Betjeningsknapper for trigger.



Når knappen holdes inne tegnes en stiplet linje på skjermen som representerer spenningsnivået trigging skjer på.



Utfører manuelt trigging av skermtegning. Denne funksjonen er for når osciloskopet opererer utenfor automatisk trigging.



Setter triggepunktet til 50% av amplituden til signalet som triggingen er satt til å følge.



Skru-knappen «LEVEL» er for å justere manuelt hvor på kurven triggenivået skal ligge. På skjermen viser triggenivået ved hjelp av en gul pil til høyre i rutediagrammet.



Triggefunktionaliteten har også en egen undermeny med innstillinger som kan aksesseres med «TRIG MENU»-knappen.

## Knappene print og probe check



Printknappen brukes normal sett ikke. Den er for de tilfeller der osciloskopet er koblet til skriver, eller når den brukes compact-flash minnekort for lagring av kurver.

For lagring av kurver på datamaskin, brukes eget program.



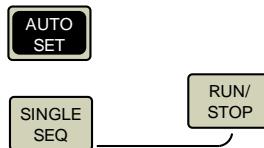
«Probe check»-knappen brukes for automatisert probetest. Ved å koble proben til «Probe comp»-terminalene lokalisert rett venstre for CH1-inngangen, kan osciloskopet selv teste om proben er feilkalibrert. Problemet dempeledd må være aktivert når testen skal utføres. Vær oppmerksom på at denne testen bør utføres manuelt om ytterste presisjon kreves av målingene.



«Probe comp»-terminalene gir ut en firkant-puls på 5V og 1kHz. Disse terminalene brukes om til kalibrering av proben. Merk at «probe check» ikke tester probene i forhold til kalibrering. Dette må gjøres manuelt. Se eget avsnitt om måleprober.

## ***Knapper for datainnsamlingsmodus***

Lokalisert til høyre på øvre knapplinje, disse knappene styrer oscilloskopets datainnsamling.



***Figur 8-11: Knapper for datainnsamlingsmodi.***



Ved bruk av autoset vil oscilloskopet gjøre en kvalifisert gjetting på hvilken måling du prøver å utføre basert på signaler på inngangen. Deretter vil det sette innstillingene deretter og gi valg om utførelse av noen automatiserte målinger via skjermtastene. Merk at autoset kan være utkoblet for å ivareta opplæringsgrunnlaget.

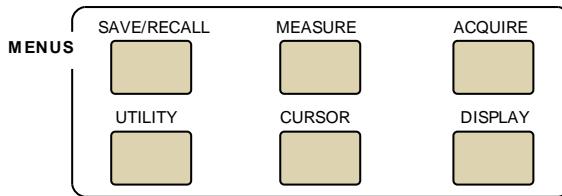


Knappen starter og stopper oscilloskopets innsamling av måledata. Ved å trykke på denne knappen fryses oscilloskopets bilde. Aktiv modus viser i midten over rutenettet på skjermen.



«Single seq» aktiverer enkel-innsamlingsmodus. Oscilloskopet vil i den modusen kun samle inn data én gang etter at triggetilstanden har skjedd. Denne modusen er tiltenkt innsamling av hendelser som skjer sjeldent. For å komme tilbake til normal modus trykkes «RUN/STOP».

## Menyknapper



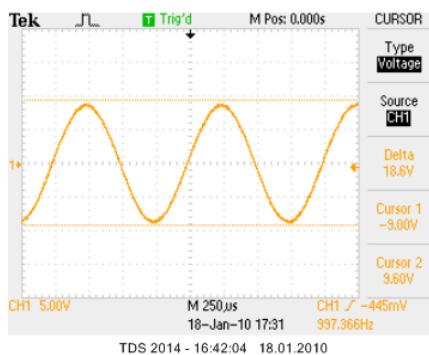
**Figur 8-12:** Menyknapper

Menyknappene gir adgang til oscilloskopets avanserte funksjoner. De fleste menyene styres med skjermknappene. Enkelte menyer bruker og noen av posisjons-skruknappene for tilleggsstyring. Når skruknappene endrer funksjon, vil en lysdiode under skruknappen indikere dette.

### «Measure» - automatiserte målinger



«Measure»-menyen har 5 målinger tilgjengelig. Disse er automatiserte målinger som kan settes opp for hver kanal.



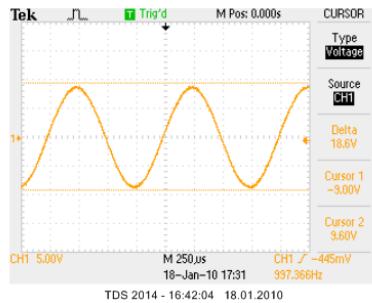
**Figur 8-13:** «Measure»-menyen

Oppsett av målingene skjer ved hjelp av skjermtastene. Tilgjengelige målinger er: frekvens, periode, snittverdi, topp-til-topp, eng. peak-to-peak, RMS for første periode, minimumsverdi, maksimumsverdi, stigetid, falltid, positiv pulsbredde og negativ pulsbredde. Videre hjelp om disse målingene finnes ved å trykke på HELP på oscilloskopet. Vær obs på kilder til målefeil når disse automatiserte målingene benyttes.

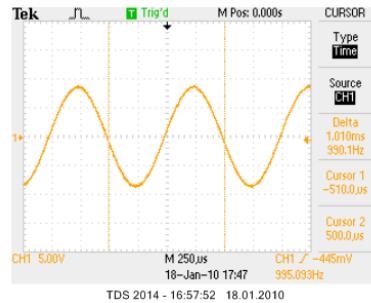
## «Cursor» - målelinjer



«Cursor»-menyen styrer to målelinjer som enten kan settes opp til å måle spenning, langs y-aksen, eller tid, langs x-aksen. Hvilken måling som nyttes velges under «type» på skermknappene. «Source» angir hvilken kanal målingene hentes fra.



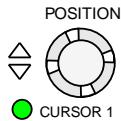
(a) Spenningsmåling



(b) Tidsmåling

**Figur 8-14:** Måling ved hjelp av målelinjer.

Når en måling er aktivert, vises verdien ved de to målelinjenes posisjon, samt differansen mellom dem.



Posisjonen til målelinjene kan endres ved hjelp av «position»-skruknappene for kanal 1 og kanal 2. Når disse skruknappene styrer målelinjene, lyser lysdioden under skruknappene.

## «Acquire» - Målemetode



Menyen «Acquire» styrer oscilloskopets datainnsamlingsmetode. Normalt sett vil oscilloskopet samle data ved hjelp av enkel punktprøving. Dette vil i de fleste målinger være godt nok. Men enkel punktprøving kan til tider medføre en del støy på målingene. I slike tilfeller vil det være lurt å bruke oscilloskopet om å ta snitt målinger.

Snittmålinger kan tas som snittet av 4, 16, 64 eller 128 enkeltpunktprøvinger. Vær oppmerksom på at kurvens respons til endinger i påtrykk vil være meget treig når det brukes høy snittmåling. Dette nettopp fordi kurven vist på skjermen er snittet av nåværende måling og et gitt antall tidligere målinger.

I «Peak Detect»-modus vil oscilloskopet tegne to streker, en for øvre grense og en for nedre. Disse strekene representerer de topp- og bunnivåene signalet er mellom.

## «Display» - Visningsmetode og kontrast



«Display»-menyen presenterer to måter eller format for å representere de målte verdiene på. Oscilloskopet opererer normalt med tid langs x-aksen og spenning langs y-aksen. Dette er kjent som YT-modus.

Oscilloskopet har også en annen modus, kjent som XY-modus. I XY-modus vil signal på CH1-inngangen angi verdi langs x-aksen, og signal på CH2-inngangen angi verdi langs y-aksen. Dette medfører at signalene nå blir plottet i et x-y-koordinatsystem.

Videre kan vi under displaymenyen spesifisere om vi ønsker å tegne målingene våre som vektorer eller punkter. Som vektorer vil oscilloskopet tegne streker mellom målepunktene slik at vi får en kontinuerlig kurve. I «Dots»-modus tegner oscilloskopet kun de enkelte målepunktene og vi har ikke lengre en kontinuerlig kurve.

«Persist»-innstillingen ser hvor lenge oscilloskopet skal beholde de forrige kurvene på skjermen, til tross for at nye kurver tegnes over. Dette er et «billigalternativ» til digitalfosforeffekt som finnes på svært dyre oscilloskop.

## «Utility» - Generelle innstillinger



«Utility»-menyen inneholder oscilloskopets innstillinger. Her finnes også filsystemoperasjoner for lagring til minnekort. Videre finnes oppsett av skriver og link til PC.

Skulle oscilloskopet ende opp med en språkinnstilling som er uforståelig, kan dette endres ved å trykke seg inn på «utility», og så benytte nederste skjernknapp til å bla gjennom de forskjellige språkene.

## «Save/Recall» - Lagring på minnekort



I «Save/Recall»-menyen kan man lagre og hente frem innstillinger eller skjermbilder. Disse kan enten lagre på et svært begrenset antall plasser internt i oscilloskopet, eller på minnekort satt i på baksiden av oscilloskopet. Bruk help-menyen for videre veiledning.

## Måleprober

Måleprober for oscilloskop er spesiallagde ledninger laget for å gi best mulig målinger. Probene er dyre presisjonsinstrument, og sårbarer for feil bruk, derfor bør de behandles forsiktig.

Probene skal KUN brukes for signaler inn på oscilloskopet. De skal ikke kobles som utledninger for signalgeneratoren.

Vær og påpasselig med loddebolten, da en liten skade fra loddebolten vil ødelegge probens presisjon.

For å utføre de mest presise målingene må probene kalibreres. Prosedyre for dette er forklart i neste avsnitt.

Probene er utstyrt med et 10 gangers dempeledd. Dette dempeleddet aktiveres og deaktiveres ved hjelp av en bryter i probens måle-ende. Pass på at tilstanden til probens dempeledd gjenspeiles i oscilloskopets «Probe»-innstilling under kanalmenyen.

Probenettuppen har innebygget fjærbelastet krok som kan hektes fast på et målepunkt. For å få frem kroken trekkes tuppens plast bakover. Videre er det mulig å ta av probetuppen. Da har proben en spiss som egner seg for de vanskelig å nå målepunktene. IKKE MIST probetuppen om den tas av. Sett den på igjen når målingen av ferdig. Tuppene kan ikke kjøpes løst, så mistes tuppen må hele proben erstattes.

Rundt målespissen til proben ligger det en metallring. Denne ringen er koblet til jord. Pass på slik at den ikke kortslutter andre kretser når målinger gjøres uten probetuppen.

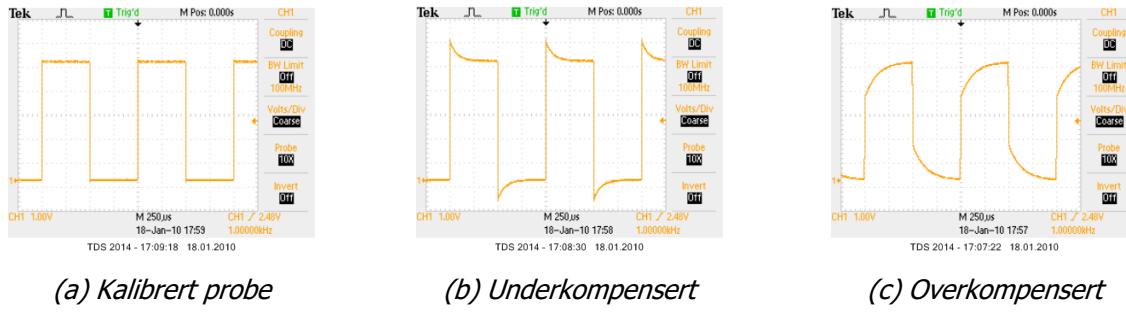
## Prosedyre for kalibrering av prober

Probene kan sjekkes automatisk av oscilloskopet, men denne sjekken er ikke like presis som å gjøre sjekken manuelt. For å la oscilloskopet utføre sjekken, kan «PROBE CHECK» trykkes.

For enkle målinger hvor presisjon ikke er kritisk, kan denne funksjonen brukes. Men for målinger hvor ytterste presisjon kreves, bør følgende prosedyre følges:

1. Koble fra eventuelle måleledninger fra alle inngangene på oscilloskopet.
2. Koble proben til «Probe check» terminalene, og inn på kanal-1. Pass på hvilken av terminalene er merket med jord.
3. Sett proben til 10x.
4. Tilbakestill oscilloskopet til standardinnstillinger ved å trykke på «Default Setup».
5. Juster kanalens nullpunkt ned ved hjelp av skruknappen «POSITION» for kanal-1, til hele kurven viser.
6. Trykk «SET TO 50%» under trigger for å stabilisere kurven.
7. Vri SEC/DIV skru-knappen et hakk med klokken.
8. Bruk skrujern i plast til å justere probens kapasitive last til firkantpulsen ikke lengre har avrundede eller overskytende flanker. IKKE bruk skrujern med metall. Dersom plastsrukjern ikke finnes på labplassen, sjekk komponenttorget, eller glasskap i enden av laboratoriet.
9. Gjenta prosessen for alle probene.

Probens kapasitive last justeres på BNC pluggen som er koblet inn på kanalen. Det finnes et lite hull på metalkapslingen hvor justeringen utføres.



**Figur 8-15:** Kalibrering av måleprobenes kapasitive last.

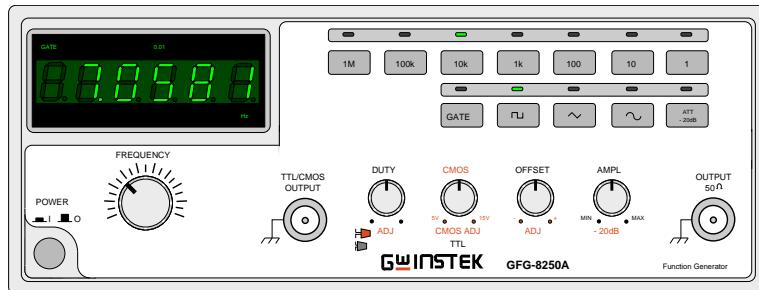
## 9 Signalgeneratoren

Signalgeneratoren brukes for å påtrykke tidsavhengige testsignaler. Avhengig av signalgeneratorens kompleksitet kan vi generere alt fra helt enkle repeterende signal, til svært komplekse signal. Signalgeneratoren på laboratoriet er av enkel type. Enkle signalgeneratorer har mindre innstillingar og passer derfor meget bra til opplæringsformål.

Generatoren elektrolaboratoriet er utstyrt med er av type GW Instek GFG-8250A. Generatoren kan generere signal fra 0,5Hz til 5MHz i form av sinus-, trekant-, firkantkurver. Generatoren er bygget opp rundt en analog frekvensgenerator, og en frekvensteller. Telleren viser frekvensen generert av generatoren på 7-segment skjermen. Vær oppmerksom på at under 7-segment-tallene er det et lite lys som indikerer Hz, kHz eller MHz.

Videre har generatoren to utganger. En merket med «OUTPUT 50Ω» og «TTL/CMOS OUTPUT». Dersom ikke annet er oppgitt i laboratorieoppgaven skal utgangen «OUTPUT 50Ω» benyttes.

«TTL/CMOS OUTPUT»-utgangen har høyere indre motstand og vil ikke være i stand til å drive kretser med lav inngangsmotstand.



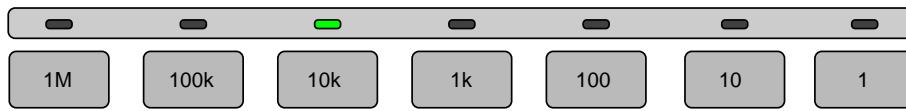
**Figur 9-1:** Signalgenerator: GW Instek GFG-8250A

Vær oppmerksom på at jord på utsignalet er koblet sammen med jord fra nettspenningen. **Dette åpner for at jord kan være koblet til jord på andre instrumenter gjennom strømnettet.**

## Valg av frekvens

Signalgeneratorens frekvensrekkevidde er delt inn i 7 frekvensområder. Øverste rad med knapper velger hvilket område som kan justeres med skruknappen «FREQUENCY».

Frekvensområdet velger senterfrekvensen til frequency-skruknappen. Det vil si frekvensen den har når velgeren står rett opp. Frequency-skruknappen kan deretter senke eller øke frekvensen med fra ca. 0,5 til 5 ganger frekvensen som er valgt. En lysdiode vil lyse over det frekvensområdet som er valgt.



**Figur 9-2:** Frekvensområdeknapper

## Kurveform



**Figur 9-3:** Frekvensområdeknapper

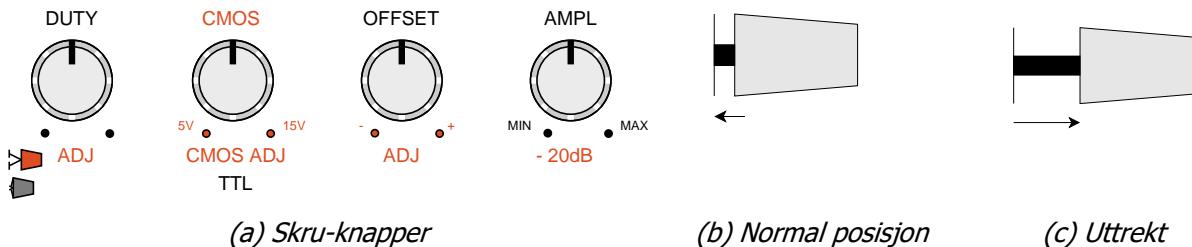
**Kurveform** De tre knappene i senter av nappraden under velger hvilken kurveform generatoren skal generere. Valgt signalform vil indikeres med lysdiode over valgt signalform.

**GATE** Gate-knappen velger hvor lang tid, frekvenstelleren skal telle, før resultatet viser på 7-segment skjermen. Jo høyere telletid, jo mer presist resultat. Over frekvensen på skjermen viser et lite tall mellom 0,01 og 10. Dette angir hvor lang tid i antall sekund tellingen foregår. Dette tallet endres ved hjelp av gate-knappen.

**ATT -20dB** Ved å aktivere denne knappen, dempes utsignalet på OUTPUT 50 med 10. Det vil si at når denne knappen er aktiv, er utsignalet 1/10 av det opprinnelige. Denne knappen kan kombineres med å trekke ut AMPL-justeringsskappen for å få en kombinert demping på 100.

## Amplitude, nullpunkt og signalforhold

Signalgeneratorens 4 små skru-knapper har to funksjoner. Ved å trekke ut knappene aktiveres den sekundære funksjonen



**Figur 9-4:** Signalgeneratorens skruknapper

**DUTY** Duty-skruknappen justerer signalets signalforhold, eng. duty cycle. Knappen må trekkes ut for å aktivere denne funksjonen. Signalforholdet er forholdet mellom hvor stor andel av signalet er høy periode. «Duty cycle» regnes ut på følgende måte, hvor  $T_H$  er tiden signalet er høyt og  $T_P$  er signalets periode.

$$DC = \frac{T_H}{T_P}$$

**CMOS** Skruknappen justerer amplituden på TTL/CMOS-utgangen. Når skruknappen er i normal posisjon er utsignalet på TTL/CMOS i henhold til TTL nivå. Det vil si høy = 5V og lav = 0 V. I CMOS kan driftsspenningen variere, og det er derfor viktig å kunne justere amplituden. Ved å trekke ut skruknappen, aktiveres CMOS-signal på TTL/CMOS-utgangen og amplituden han varieres.

**OFFSET** Normalt vil signalet på utgangen OUTPUT  $50\Omega$  variere med topp-til-topp verdi sentrert rundt 0V. Det vil si at signalets har halvparten av top-til-top spenningen på negativ, og den andre halvparten positiv. Dette kan endres ved å aktivere «OFFSET». Ved å trekke ut skruknappen vil signalets «nullpunkt» kunne flyttes. Vær oppmerksom på at mengden nullavvik som kan justeres påvirkes av generatorens dempeledd. Dersom dempeledd er aktivt, begrenses nullavviket i like stor grad.

**AMPL** Signalets amplitude på OUTPUT  $50\Omega$  justeres ved hjelp av denne skruknappen. Dersom skruknappen trekkes ut, aktiveres et 20dB dempeledd. Da dempes signalet med en faktor på 10. Denne funksjonen bør brukes dersom det er svake signal som skal genereres.

## ***Vanlige brukerfeil***

- Signalet er for svakt: Utilsiktet bruk av dempeledd, enten på -20dB knappen eller ved at AMPL-justeringsknappen er trekt ut.
- Ikke noe eller svært lite signal: Generatorens utgang er kortsluttet mot jord, ofte gjennom at andre instrumenter og har koblet signal-jord til nettspenningens jord.
- Feil frekvens: Dobbeltjekk det indikerte området under 7-segment tallene at frekvensen målt er i riktig område: Hz, kHz, MHz.
- Det er ikke mulig å justere nullpunktet, offset, tilstrekkelig langt. Dette skyldes at et av dempeleddene er innkoblet. Dempeleddet påvirker og rekkevidden av nullpunktjusteringen.

# 10 Vedlegg: VHDL – En introduksjon

## Generelt

Dette er en enkel og ikke-komplett beskrivelse av VHDL. VHDL er et stort og omfattende språk. Her går vi kun gjennom et lite subsett, men det er nok til å klare seg her. VHDL er en forkortelse. HDL står for Hardware Description Language (Maskinvarebeskrivende språk), mens V-en står for VHSIC, som igjen er en forkortelse for Very High Speed Integrated Circuits. Maskinvarebeskrivende språk vil si et språk som kan beskrive digital maskinvare eksakt, med andre ord en tekstbasert kretsbeskrivelse. Vi skal her ikke se på elektriske aspekter, som driverstyrke, tidsforsinkelser, osv., men kun se på den rent digitale funksjonen. VHSIC er navnet på et prosjekt i regi av det amerikanske forsvarsdepartementet på 80-tallet. Man ville her lage et språk som kunne tjene som en formell spesifikasjon av en digital krets. Resultatet ble VHDL. Det ble senere standardisert av IEEE (Institute of Electrical and Electronics Engineers). Da VHDL er et formelt språk, kan det også simuleres (slik at man kan se om spesifikasjonen har ønsket oppførsel) og syntetiseres ("omforme" den oppførselsbaserte VHDL-koden til en nettliste som igjen kan brukes til å lage maskinvare).

En krets kan altså beskrives ved hjelp av en tekstfil istedenfor en skjemategning. Med dette oppnår vi flere fordeler: Vi får en beskrivelse av en digital krets hvor beskrivelsen er eksakt (det er ikke alltid at det er entydig hva et symbol i en skjemategning betyr), uavhengig av teknologi (produksjonsprosess), skjemategnings-programvare, leverandør, osv. I tillegg er det svært enkelt å flytte en tekstfil fra et system til et annet.

Det er meget viktig å være klar over at man *ikke* kan oppnå en bedre krets ved å bruke VHDL enn ved å bruke skjemategning. Før man starter med å skrive VHDL-kode, må man vite *hva* man skal frem til. Spesifikasjonene må være gitt. Eksempel: Om man skal skrive en arbeidsordre til en som kun snakker engelsk må man skrive orden på engelsk, men først må man vite *hva* man skal skrive (hva orden skal inneholde).

**VHDL er ikke programvare, det er beskrivelse av maskinvare.**

## Eksempel

Det må her understrekkes at vi kun ser på et lite subsett av VHDL, men nok til å kunne bruke VHDL til å beskrive kretser. En fil med VHDL består av tre deler. Den første delen angir hvilke bibliotek som skal brukes. Den andre delen beskriver inn- og utgangene (pinnene), og kalles *entity*, mens den tredje delen beskriver kretsens oppførsel (innholdet), og kalles *architecture*. En slik *architecture* består av en eller flere prosesser. En komplett VHDL-kode som beskriver en enkelt ledning rett gjennom kretsen, blir (legg merke til bruk av semikolon):

```

library IEEE;
use IEEE.std_logic_1164.all;

entity example is
    port (in_pin : in STD_LOGIC;
          out_pin: out STD_LOGIC
        );
end example;

architecture example_arch of example is
    process (in_pin) is begin
        out_pin <= in_pin;
    end process;
end example_arch;

```

La oss nå se på detaljene i dette eksempelet. Husk at oppbygningen er den samme selv om kretsen er mye større.

De to første linjene utgjør del 1 (hvilket bibliotek som skal brukes). Den første linjen, *library IEEE*, forteller hvilket bibliotek det er, mens den andre linjen, *use IEEE.std\_logic\_1164.all*, forteller hvilken del av biblioteket som skal brukes. Her trenger vi ikke andre bibliotek, så når du selv skriver VHDL-kode lar du del 1 være akkurat som i eksempelet ovenfor.

Del 2 beskriver inn- og utganger, altså kretsens «pinner». Denne delen begynner med det reserverte ordet *entity*, deretter modulens navn, og det reserverte ordet *is*. Syntaksen i port-listen (inne i parentesen) er: Navn på pinnen, kolon, det reserverte ordet *in* eller *out*, mellomrom, og tilslutt datatypen *STD\_LOGIC* (og *STD\_LOGIC\_VECTOR* for busser). For å skille de forskjellige pinnene i port-listen brukes semikolon, og det siste semikolonet kommer utenfor slutt parentesen. Legg merke til at vi *ikke* beskriver jord og power supply, kun de logiske (digitale) pinnene. En 8 bits buss med navn *multiplicand* beskrives slik:

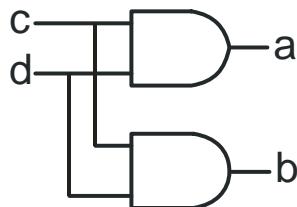
multiplicand: in STD\_LOGIC\_VECTOR (7 downto 0);

Del 3 begynner (her) med *architecture example\_arch of example is*. De reserverte ordene er *architecture*, *of*, og *is*. Navnet på arkitekturen er *example\_arch*, mens *example* er navnet som knytter denne arkitekturen til *entity-en* i del 2 (som heter *example*). En *architecture* er bygget opp av mange parallele prosesser. I vårt eksempel har vi bare en enkelt prosess. Det som står i parentes etter det reserverte ordet *process* er *sensitivitetslisten*. Det betyr at prosessen skal kjøres om en eller flere av signalene i listen endrer seg. Her er regelen enkel: for en ren kombinatorisk krets *skal alle* inngangssignaler i prosessen (som en utgang blir tilordnet) være med i sensitivitetslisten. I vårt eksempel er *in\_pin* i sensitivitetslisten. Hver gang *in\_pin* endres, blir prosessen kjørt, og *out\_pin* blir tilordnet *in\_pin*. Husk at dette *ikke* er noe program, men en tekstbasert måte å beskrive maskinvare på.

En annen kombinatorisk prosess kan se slik ut:

```
process (c, d) is begin
    a <= c and d;
    if ((c = '1') and (d = '1')) then
        b <= '1';
    else
        b <= '0';
    end if;
end process;
```

Legg merke til at tilordning skjer med tegnet  $\leq$ . Vi ser at da  $c$  og  $d$  er inngangene, står de i sensitivitetslisten. Her har utgangene  $a$  og  $b$  samme funksjon. Signalet  $a$  er beskrevet som et boolsk uttrykk, mens signalet  $b$  er beskrevet med en «programvarealignende» konstruksjon, nemlig en if-setning. Resultatet etter syntese blir imidlertid det samme for begge, slik det er vist i figuren nedenfor.



Legg merke til at i en if-setning må alle muligheter være beskrevet. Det nytter ikke kun å ta med *then*  $b = 1$ , og utelate *else*  $b = 0$ . I så fall må den gamle verdien til  $b$  bli «husket» om det som står etter *then* ikke slår til. Med andre ord må vi ha hukommelse i kretsen, for eksempel en lås (latch), og kretsen er ikke lenger kombinatorisk. Dette ønsker vi normalt ikke i slike tilfeller og vi kan i verste fall også risikere at simulert oppførsel og oppførselen i den virkelige kretsen blir forskjellige.

Tilslutt nevnes hvordan vi beskriver D-vipper og tellere (synkrone kretser) i VHDL:

```
process (clk) is begin
    if (clk'event and clk = '1') then
        q <= d;
    end if;
end process;
```

Om vi vil ha med synkron reset, skriver vi:

```
process (clk) is begin
    if (clk'event and clk = '1') then
        if (reset = '1') then
            q <= '0';
        else
            q <= d;
        end if;
    end if;
end process;
```

Vi skal kun ha med  $clk$  i sensitivitetslisten, da de andre inngangene ikke er interessante om  $clk$  ikke endres (synkron krets). Legg merke til at tallene skrives '0' og '1'. Denne prosessen «leses» slik: Hver gang  $clk$  endres, skal prosessen kjøres. Om  $clk$  endres ( $clk'event$ ) og  $clk$  (etter endringen) er lik 1 (det vil si at  $clk$  har en positiv flanke), skal den indre if-setningen kjøres. Videre: Om  $reset$  er 1, skal  $q$  (D-vippens utgang) settes til 0, ellers ( $reset$  er 0) skal  $q$  settes til  $d$  (D-vippens inngang). Legg merke til at vi her ikke angir hva som skal skje om den

ytre if-setningen ikke slår til (ingen *else*), fordi det gamle resultatet huskes (da det er en vippe vi har her).

Det er viktig å legge merke til at dette er et maskinvarebeskrivende- og *ikke* et programmeringsspråk. Derfor må vi ha med både process (clk) og if (clk'event...). Rent logisk så er dette samme ting sagt to ganger. Siden man har process (clk) så må clk'event være sann. Men dette realiseres av porter, og beskrivelsen må være slik at programmet som skal simulere og syntetisere kretsen forstår at din intensjon er å beskrive en synkron klokket krets. Det vil den gjøre når du skriver på denne måten. Ellers kan resultatet bli feil. F.eks. modulen reagerer asynkront.

En 3 bit nedteller med *decrement*, *load*, og *reset* beskrives slik:

```
process (clk) is begin
    if (clk'event and clk = '1') then
        if (reset = '1') then
            counter <= 0;
        elsif (load = '1') then
            counter <= 7;
        elsif (decrement = '1') then
            counter <= counter - 1;
        else
            counter <= counter;
        end if;
    end if;
end process;
```

Her har *reset* høyest prioritet, det betyr at *reset* er det første som sjekkes (etter *clk'event and clk = '1'*). Midterst prioritet har *load*, og lavest prioritet har *decrement* (telleren dekrementeres med 1). Om verken *reset*, *load*, eller *dec* er 1, skal telleren ikke endres, derfor avsluttes if-then-elsif-kjeden med *else counter <= counter* (ingen endring). Merk at siden dette er en maskinvarebeskrivelse så vil syntesen resulttere i en fysisk krets med fire D-vipper og diverse logiske porter for å endre verdier på vippene. Selv om det kanskje kan se slik ut så er altså dette IKKE et program beregnet på å bli kjørt på en mikroprosessor.

I senere årskurs vil dere få mulighet til å velge fag som gir inngående kjennskap til maskinvarebeskrivende språk.

## Annet

I en kombinatorisk krets som beskrives med en if-then-elsif-kjede, skal alle utganger være definert i *hver* avgrenning. Se VHDL-koden for multiplikatorens tilstandsmaskin i LAB 4.

I VHDL-editoren som brukes i labben, er der en «Language Assistant» som anbefales. Det er ingen lærebok, men den er fin for å sjekke syntaks. Bruk *Tools → Language Assistant*.

Merk at det er ikke nødvendig med noen spesiell editor for å skrive VHDL. Spesielle VHDL-editorer (som den i Foundation) har spesielle farger for reserverte ord, syntakssjekk, osv. Dette er ganske greit, men ikke *nødvendig*. Det er fullt mulig å bruke en helt «vanlig» tekst editor til formålet.

I VHDL-koden i LAB 4 (multiplikatorens tilstandsmaskin) er der lagt inn mange kommentarer. Bruk den som eksempel.

I tillegg til VHDL finnes et tilsvarende språk: Verilog, som også er meget utbredt.

VHDL Reference Guide ligger lokalt på maskinene på labben: I programkatalogen til Foundation under \synth\help\pdf\vhdlref.pdf

Det anbefales å bruke internett. Sjekk f eks <http://www.model.com/>. Du kan også bruke en søkermotor, og søke på «VHDL». Da vi du garantert finne mye interessant. Det er *absolutt* ikke nødvendig å kjøpe noen VHDL-bok i dette faget, men om man er spesielt interessert nevner vi at det på Tapir finnes bøker om VHDL. Sjekk f.eks. pensumlisten til faget TFE4140 Modellering og analyse av digitale systemer.



# 11 Datablad for portkretser

## Forklaring av forkortelser

For å forstå databladet riktig, er det viktig å ha klart for seg hvilke data som er oppgitt. De fleste datablad følger en standard navngiving av variable, men det er meget viktig at dere ved fremtidig bruk av datablad sjekker med produsenten hva eksakt mener med de forskjellige dataene. Enkelte avvik kan forekomme.

Under følger en kort forklaring på de data dere har behov for i laboratorieøvingene.

<b>Typ</b>	Typisk verdi: Dette er den «forventede» verdien. Verdien er som oftest både under og over denne verdien.
<b>Max</b>	Maks verdi. Den oppgitte parameter skal ikke under noen omstendighet overgå denne verdien. Her er det viktig å merke seg at denne verdien er gitt for et gitt forhold.
<b>Min</b>	Minimumsverdi: Den oppgitte parameter skal ikke under noen omstendighet under denne verdien.
$t_{PLH}$	Forplantningstiden for når signalet går fra «lav» til «høy».
$t_{PHL}$	Forplantningstiden for når signalet går fra «høy» til «lav».
$t_{TLH}$	Stigetiden. Signalet går fra «lav» til «høy».
$t_{THL}$	Falltiden. Signalet går fra «høy» til «lav».
$V_{DD}$	Driftspenningen verdien er gyldig for.
$V_{IL}$	Den maksimale spenningen et innsignal kan ha, og fremdeles bli tolket som logisk 0
$V_{IH}$	Den minimale spenningen et innsignal kan ha, og fremdeles bli tolket som logisk 1

**MERK:** For de påfølgende databladene, med en gitt temperatur TA, kapasitiv last CL og resistiv last RL. I tillegg har den krav til inngangssignalet.

## Timingdata for CD4030B

Texas Instruments		CMOS Quad Exclusive-OR Gate			CD4030B			
Data sheet acquired from Harris Semiconductor SCHS035C – Revised September 2003								
DYNAMIC ELECTRICAL CHARACTERISTICS at $T_A = 25^\circ C$ ; Input $t_r, t_f = 20 \text{ ns}$ , $C_L = 50 \text{ pF}$ , $R_L = 200 \text{ k}\Omega$								
CHARACTERISTICS		CONDITIONS $V_{DD} (\text{V})$	LIMITS		UNITS			
			Min.	Typ.				
Propagation Delay Time, $t_{PLH}, t_{PHL}$	5	-	140	280	ns			
	10	-	65	130				
	15	-	50	100				
Transition Time, $t_{THL}, t_{TLH}$	5	-	100	200	ns			
	10	-	50	100				
	15	-	40	80				
Input Capacitance, $C_{IN}$	Any Input	-	5	7.5	pF			
Input "Logic Low" Max Voltage $V_{IL}$	5	-	-	1,5	V			
	10	-	-	3				
	15	-	-	4				
Input "Logic High" Min Voltage $V_{IH}$	5	3,5	-	-	V			
	10	7	-	-				
	15	11	-	-				

AMBIENT TEMPERATURE ( $T_A = 25^\circ C$ )

TRANSITION TIME ( $t_{THL}, t_{TLH}$ ) — ns

LOAD CAPACITANCE ( $C_L$ ) — pF

SUPPLY VOLTAGE ( $V_{DD}$ ) = 5V, 10V, 15V

92CS-24322

AMBIENT TEMPERATURE ( $T_A = 25^\circ C$ )

PROPAGATION DELAY TIME ( $t_{PLH}, t_{PHL}$ ) — ns

LOAD CAPACITANCE ( $C_L$ ) — pF

SUPPLY VOLTAGE ( $V_{DD}$ ) = 5V, 10V, 15V

92CS-30053

Typical transition time as function of load capacitance.

Typical propagation delay time as function of load capacitance.

## Timing data for CD4081B

Texas Instruments		CMOS Quad 2-input AND Gate			CD4081B
DYNAMIC ELECTRICAL CHARACTERISTICS at $T_A = 25^\circ C$ ; Input $t_r, t_f = 20 \text{ ns}$ , $C_L = 50 \text{ pF}$ , $R_L = 200 \text{ k}\Omega$					
CHARACTERISTICS	$V_{DD} (\text{V})$	LIMITS			UNITS
		Min.	Typ.	Max.	
Propagation Delay Time, $t_{PLH}, t_{PHL}$	5	-	125	250	ns
	10	-	60	120	
	15	-	45	90	
Transition Time, $t_{THL}, t_{TLH}$	5	-	100	200	ns
	10	-	50	100	
	15	-	40	80	
Input Capacitance, $C_{IN}$	Any Input	-	5	7.5	pF
Input "Logic Low" Max Voltage $V_{IL}$	5	-	-	1,5	V
	10	-	-	3	
	15	-	-	4	
Input "Logic High" Min Voltage $V_{IH}$	5	3,5	-	-	V
	10	7	-	-	
	15	11	-	-	

AMBIENT TEMPERATURE ( $T_A = 25^\circ C$ )

TRANSITION TIME ( $t_{THL}, t_{TLH}$ ) — ns

LOAD CAPACITANCE ( $C_L$ ) — pF

SUPPLY VOLTAGE ( $V_{DD} = 5\text{V}$ )

10V 15V

92CS-24322

AMBIENT TEMPERATURE ( $T_A = 25^\circ C$ )

PROPAGATION DELAY TIME ( $t_{PLH}, t_{PHL}$ ) — ns

LOAD CAPACITANCE ( $C_L$ ) — pF

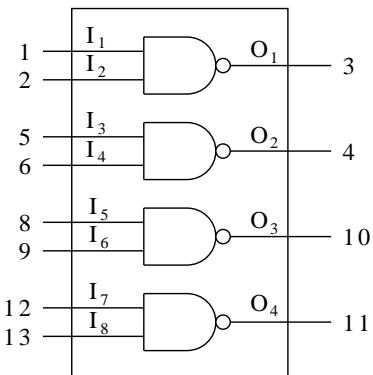
SUPPLY VOLTAGE ( $V_{DD}$ ) — 5V 10V 15V

92CS-2927I

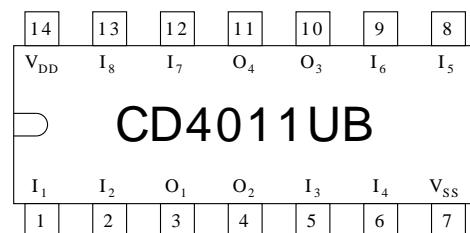
Typical transition time as function of load capacitance.

Typical propagation delay time as function of load capacitance.

## Pinnekonfigurasjon for CD4011(U)B, CD4030B og CD4081B

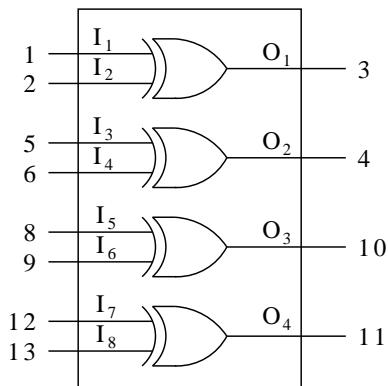


(a) CD4011B / CD4011UB Skjema

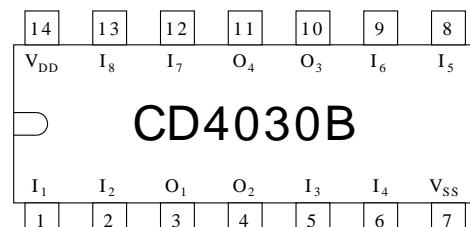


(b) CD4011B / CD4011UB Pinnekonfigurasjon

Figur 11-1: CD4011B / CD4011UB

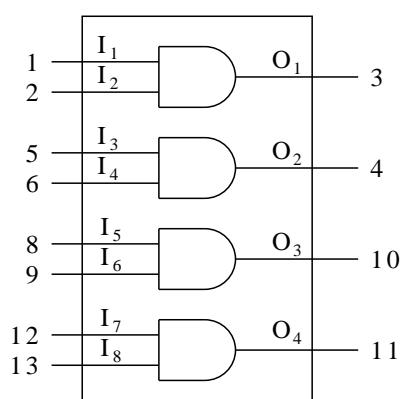


(a) CD4030B Skjema

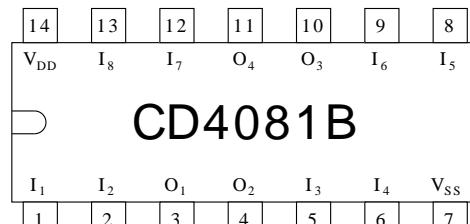


(b) CD4030B Pinnekonfigurasjon

Figur 11-2: CD4030B



(a) CD4081B Skjema



(b) CD4081B Pinnekonfigurasjon

Figur 11-3: CD4081B

## 12 Rydding av laboratoriebenk

For å ta best mulig vare på utstyret på laboratoriet, og gjøre vedlikehold til en overkommelig oppgave, er det viktig at dere som brukere av laboratoriet er med på å holde laboratoriet ryddig. Ikke bare er det hyggeligere å komme til en ryddig laboratoriebenk, men når vi i fagstaben slipper å bruke tid på å rydde, får vi mer tid til vedlikehold og forbedringer for dere som studenter. På en ryddig laboratoriebenk er det også lettere å unngå uhell som skader utstyret.

Det er derfor viktig at dere, når dere er ferdig med laboratorieøvingen, rydder laboratoriebenken etter dere. Dette innebærer å sette alle instrumenter på plass, legge tilbake utstyr lånt fra andre steder i laboratoriet, sjekke at alt utstyr er tilstede, og gi beskjed dersom utstyr mangler eller er for mye av.

Alle instrumenters nettkontakter skal trekkes ut. La leselampens ledning stå i. Strømledningene til instrumentene skal legges på baksiden av instrumentene. Dette er viktig for å hindre at ledningene ved uhell, blir brent ved bruk av loddebolt. For å få plass til alle ledningene, må instrumentene stå på skrå i hvert hjørne som vist på bildet. Sett instrumentene tett inntil hverandre slik at ledningen ikke kommer frem.

Halvparten av laboratoriebenkene har speilvendt oppsett. Sjekk hvilken side skrivebordslampen er festet. Er den festet på motsatt side i forhold til bildet, skal oppsettet speilvendes.

Manglende rydding av labplassen kan føre til tilbaketrekking av laboratorieøvingens godkjenning.



Figur 12-1: Mal for ryddet laboratoriebank

## Utstyr på laboratoriebenken

Instrumentene skal stå som vist på bildet. På selve benken skal det stå:

- En signalgenerator: GW Instek GFG-8250A
- Et benkmultimeter: LG eller EZ Digital DM-448B
- To Mascot 719 strømforsyninger
- Et oscilloskop: Tektronix TDS-20x4(B)
- En filtervifte

Merk at oscilloskopet skal være tilkoblet PC-en, dersom det er PC på labplassen.

Husk å feie bort avklippte komponentbein og annet rask fra benken.

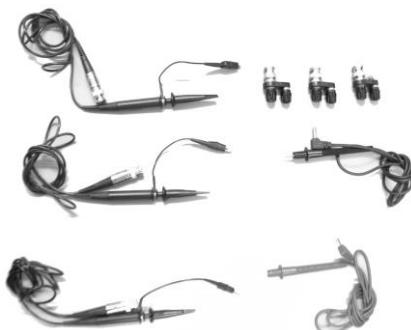
## Utstyr på laboratoriebenkens hylle

### Boksen merket med prober skal inneholde:

- 3 Tektronix oscilloskop-prober.
- 2, fortrinnsvis en rød og en svart, multimeter-prober.
- 3 BNC-til-banan-overgang.

### Boksen merket med verktøy skal inneholde:

- Et flatt skrujern.
- En nebbtang.
- En sideavbiter.
- Et kalibreringsskrujern (kan mangle)
- En håndbor for veroboard.
- En tinnsguler.
- En avisoleringstang (ikke avbildet)



(a) Innhold av prober



(b) Innhold av verktøy

Figur 12-2: Innhold i verktøy- og probekasser

### Plassering av utstyr på hyllen

Det viktig at loddebolten blir plassert ved siden av lampen. Rett ved siden av loddebolten skal kretskortsokkelen stå. Legg ledningen til loddebolten på andre side av kretskortsokkelen slik at den ikke kommer i kontakt med en varm loddebolt. Selv om din loddebolt er kald, kan loddebolten på labbenken ovenfor være varm.

Vektorvoltmeter og transformatorer og andre instrumenter med strømledning, dersom labplassen har disse, plassers på motsatt ende av laboratoriebenkens hylle i forhold til loddebolten. Ved dette utstyret plasseres boksene med måleprober og verktøy. Dette for å få dette utstyret tilstrekkelig langt vekk fra potensiell varm loddebolt.

Andet utstyr og instrumenter uten fastmontert ledning, plasseres på midtre del av hyllen.

Rester av loddetinn kan legges i boksen merket verktøy. Rester av fortinnet kobbertråd kasseres.

**Husk å sette stolene på plass når dere går. Slå av PC-ene med mindre dere har fått beskjed om annet.**