Institutt for datateknologi og informatikk, NTNU

Busser på datamaskiner

Geir Ove Rosvold

09. januar 2018

Busser på datamaskiner

Resymé: I denne leksjonen ser vi på busser. Vi ser på oppbygging og virkemåten til busser, og diskuterer de viktigste bussene som er brukt på PC.

Innhold

1.1. Leksjonens oppbygging

Denne leksjonen omhandler busser på datamaskiner. Dette er et omfattende tema, og vil sikkert virke overveldende og uoversiktlig ved første gangs lesing. I grove trekke er leksjonen bygget opp med følgende hovedpunkter:

- Introduksjon (som forklarer en del grunnleggende begreper)
- Parallelle busser
- Serielle busser
- Svitsjede busser

Det kan være greit å huske denne grovoppbyggingen når man leser leksjonen.

1.2. Introduksjon

Vi har sett at datamaskinen består av mange komponenter, der de viktigste hovedtypene er prosessor, primærminne og ulike typer IO-utstyr. De ulike komponenter må nødvendigvis kunne kommunisere med hverandre. Dette skjer ved at komponentene utveksler informasjon over en såkalt buss. En buss består i utgangspunket av ledninger som overfører elektriske signaler, og dermed knytter komponentene sammen.

Det finnes mange typer busser. Før vi ser nærmere på de ulike typene, skal vi se generelt på datauveksling. Vi skal se på:

- Seriell og parallell overføring
- Antall enheter som kommuniserer og behov for adressering
- Hvilken informasjon overføres

1.2.1. Parallell og seriell overføring

Når informasjon overføres over en buss, så overføres det informasjon i form av bitmønstre. Bussens oppgave er altså å overføre bit-verdier mellom komponentene. Hver enkelt bitverdi overføres som et binært spennings-signal over en ledning.

I praksis skal vi aldri overføre en enkelt bit; vi skal overføre mange bits. Dermed står vi ovenfor et helt grunnleggende valg: Skal bitene overføres etter hverandre i tid, eller skal to eller flere bits overføres samtidig (over hver sin ledning)?

Dersom bitene overføres etter hverandre i tid, har vi det som kalles en *seriell overføring*. Dersom man skal overføre 32 bits på denne måten, sendes de 32 bitene etter hverandre over samme ledning.

Alternativet er å bruke flere ledninger, og dermed overføre flere bits nøyaktig samtidig. Dersom man bruker 32 ledninger, kan man overføre 32 bits samtidig. Når vi bruker flere parallelle ledninger kaller vi det en *parallell overføring*.

Serielle og parallelle busser har ulike egenskaper og ulike virkemåter. Vi skal se nærmere på det ganske snart.

1.2.2. Port kontra buss

De enkleste typene av busser er laget for at nøyaktig **to** komponenter skal kunne utveksle informasjon med hverandre. De to komponentene knyttes sammen med bussen, og det den

ene enheten sender ut vil bli blir mottatt av den andre. En slik buss – som altså knytter sammen nøyaktig to enheter – kalles gjerne for *port* istedenfor buss..

På en port er det aldri tvil om hvem som skal motta informasjonen som sendes på bussen. Den ene enheten sender, og den andre skal motta. På mer avanserte busser kan vi koble til mange enheter. Da oppstår det en viktig endring: I tillegg til selve informasjonen som skal overføres, må vi også fortelle *hvem* som skal ha dataene. Vi får altså behov for adressering av de ulike enheter slik at vi kan skille mellom dem på en enkel måte.

1.2.3. Båndbredde

Det viktigste egenskapen til en buss er selvsagt at at den overfører binær informasjon i form av bits. Overføringskapasiteten (eller *båndbredden* som er det korrekte ordet) oppgis i antall bits som kan overføres pr sekund (NB legg merke til at det er antall *bits* og ikke antall *bytes* pr sekund).

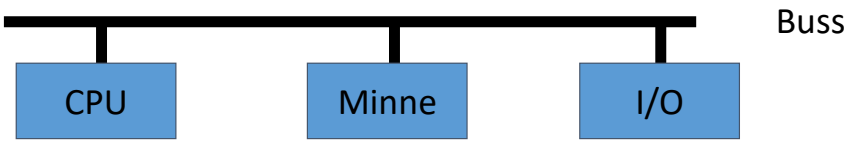
Imidlertid er det slik at det må sendes mer enn bare data over bussen. Vi har allerede sett at på busser med mange enheter må det sendes mottageradresse slik at enhetene vet hvem som skal ha informasjonen. I tillegg må det sendes signaler som synkroniserer og kontrollerer det som foregår på bussen. Nøyaktig hva slags synkroniserings- og kontrollsignaler som brukes vil avhenge av hva slags buss vi bruker, og dette skal vi se på senere.

Poenget her er imidlertid at en del av bussens overføringskapasitet i praksis vil måtte brukes til å overføre adresser og kontrollsignaler. Vi ønsker selvsagt å holde omfanget av dette på en minimum for å få en mest mulig effektiv buss med minst mulig ”administrativ overhead”.

Når man oppgir båndbredde til en buss så er det den potensielt maksimale overførings-kapasiteten vi oppgir. I praksis oppnår man aldri så høy båndbredde. I hvert fall ikke over lengre tid.

1.2.4. Arbitrering. Sentralisert/distribuert buss

En buss består av et sett av ledninger som binder de ulike enheter sammen. Enhetene kobles til bussen på forskjellige steder langs ledningene. Bussen i seg selv er passiv, og har ikke annen funksjon enn å overføre elektriske signaler.



Figur 1 Buss. Bussen binder de ulike deler sammen og gjør at de kan kommunisere med hverandre. Enhetene kobles til bussen på forskjellige steder langs bussen. Derfor blir bussen en delt ressurs, der bare en enhet kan skrive på bussen til et hvert tidspunkt.

Legg merke til at siden komponentene kobles til langs bussen, vil en slik buss alltid være en *delt ressurs*. Det vil si at signalene som en enhet sender ut på bussen kan leses av alle andre. Altså en form for *èn-skriver-og-alle-lytter*. Når bitverdier overføres vil det dermed være en komponent som legger spenning ut på ledningene på bussen. Vi sier at komponenten *skriver* på bussen. Spenningsverdiene på ledningene registereres av alle de andre komponentene, som dermed kan *lese* fra bussen. Da trengs det imidlertid en mekanisme som sørger for at det bare

er en enhet som sender til enhver tid. Dersom flere prøver å sende noe samtidig, vil det bli kaos. (Derimot kan alle enheter når som helst *lese* fra bussen).

Det er lett å tenke seg at det kan oppstå situasjoner der flere enheter ønsker å bruke bussen samtidig. Da oppstår det en konflikt. Vi skiller mellom to hovedmetoder for å hindre slike konflikter. Vi snakker om sentraliserte og distribuerte busser. Forskjellen mellom disse er teknikken som brukes for å håndtere slike konflikter. Ofte kalles dette *arbitrering* (*arbitrate* betyr dømme/avgjøre)

På en sentralisert buss finnes det en *busskontroller* som tildeler bussen til enhetene ved behov. Denne busskontrolleren er en elektronisk krets som er tilknyttet bussen. Busskontrolleren er en del av det såkalte brikkesettet som vi skal lære mer om senere i kurset.

På en distribuert buss kan alle enheter reservere bussen så fremt den er ledig. I dette tilfellet må hver enhet inneholde mer elektronikk for tilgangskontroll.

1.2.5. Overføringsretning

Følgende begreper brukes i forbindelse med datakommunikasjon mellom to enheter:

Simpleks: Bare den ene enheten kan sende informasjon. Den andre enheten kan bare motta.

Halv dupleks: Begge enheter kan både lese og skrive, men ikke samtidig. Når den ene skriver, må den andre vente til den første er ferdig.

Full dupleks: Begge enheter kan både lese og skrive samtidig. Det kan altså gå trafikk begge veier samtidig.

1.2.6. Bruk av flere busser - bussherarki

På de aller enkleste datamaskinene har vi bare en buss, og alle komponenter er koplet på denne bussen slik Figur 1 viser. I praksis vil dette fungere dårlig fordi komponentene i en datamaskin spenner over en stor skala av overføringshastigheter.

Mellom minne og CPU må det overføres data med svært høy hastighet, mens det kan tolereres langt lavere hastighet mot for eksempel en skriver. Den beste måten å løse dette på er å bruke flere busser der de ulike bussene er tilpasset ulik overføringshastighet. Vi snakker derfor om et buss-hierarki der de hurtigste bussene er på toppen, og så blir båndbredden lavere etter hvert som vi går nedover i hierarkiet..

Hovedårsakene til at det er fornuftig å spre datatrafikken over flere busser er altså:

- Når vi sprer trafikken over flere busser blir det totale båndbredden større.
- Vi kan ha ulik båndbredde på de ulike bussene
- Vi kan tilpasse bussen til spesielle behov

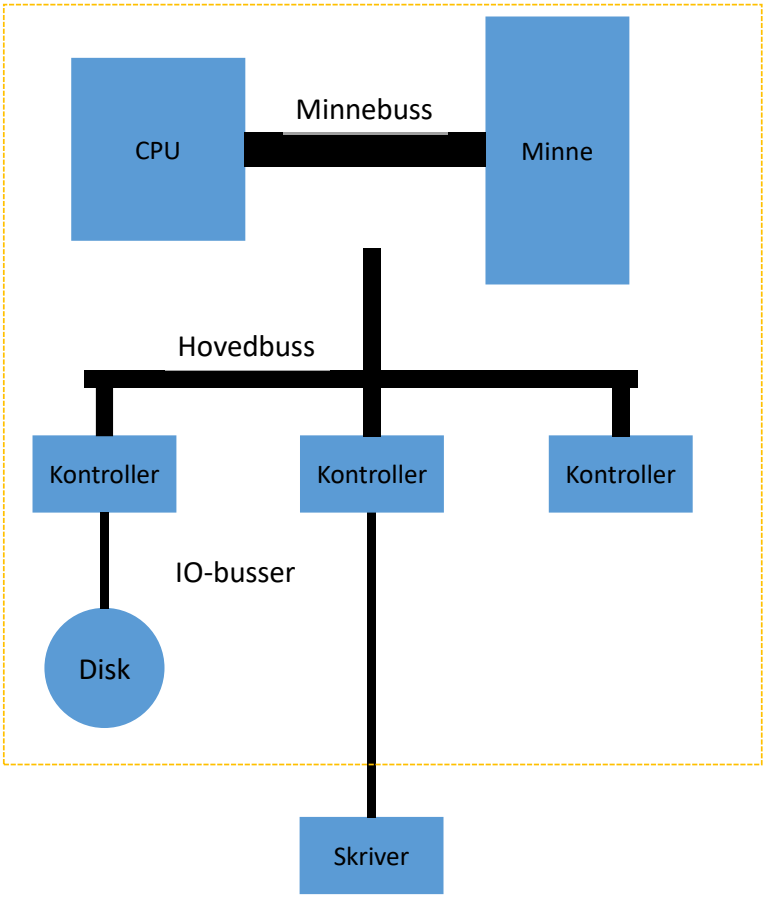
Figur 2 viser hovedkategoriene av busser på dagens datamaskiner:

1. Den hurtigste bussen (bortsett fra interne busser inne i prosessoren) er bussen mellom prosessor og minne. Denne bussen kalles minnebussen. Her settes det svært store krav til båndbredde siden prosessor og minne er de hurtigste komponentene i datamaskinen. Derfor er det hensiktsmessig å isolere denne trafikken på en egen buss.
2. Den nest hurtigste bussen kobler sammen alle kontrollere (IO-moduler, se leksjon om grunnleggende virkemåte). Eksempler på slike kontrollere (IO-moduler) er disk-kontroller, nettverkskort, grafikk-kort og mange andre.

3. Den siste kategorien busser er de såkalte IO-bussene. Slike busser danner forbindelsen mellom IO-utstyr og kontroller (IO-modul). Et eksempel på IO-buss er USB.

Resten av leksjonen vil beskrive hvordan disse kategoriene av busser realiseres på moderne datamaskiner. Da lærer vi bl.a. om følgende begreper:

1. Parallelle busser
2. Serielle busser
3. Svitsjede busser



Figur 2 Bruk av flere busser. Hovedkategoriene er minnebuss, hovedbuss og IO-busser.

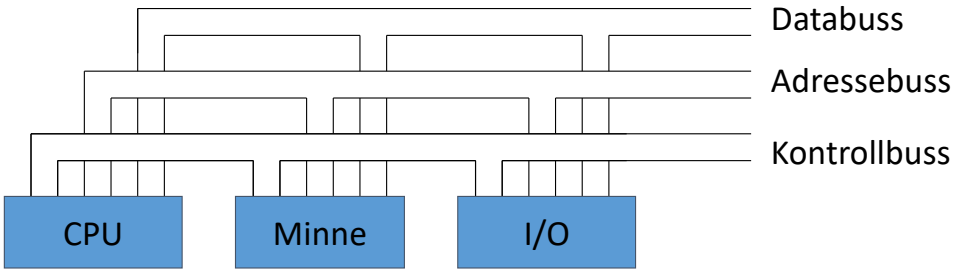
1.3. Parallelle busser

En parallell buss består av et sett av parallelle ledninger. Med en slik oppbygging kan mange bits overføres samtidig (i parallell) i hver sin ledning.

En typisk parallellbuss i en datamaskin består av 50-100 ledninger. Vi deler alle disse ledningene opp i tre grupper som vi kaller delbusser. De tre delbussene overfører hver sin type signal (se Figur 3):

- Databuss - Her overføres de binære data
- Adressebuss - Her overføres (binære) adresser
- Kontrollbuss - Her overføres signaler som styrer og synkroniserer overføringen

Når en CPU skal lese en minnelokasjon er fremgangsmåten at CPU først reserverer bussen slik at andre ikke skal skrive samtidig. Deretter legger CPU ut adressen til minnelokasjonen på adressebussen samtidig som den setter et kontrollsignal som betyr *les minne* ut på kontrollbussen. Siden alle enheter lytter på bussen vil minnet oppfatter at dette er en henvendelse til seg. Minneelektronikken leter opp den rett minnelokasjonen, og legger innholdet ut på databussen slik at CPU kan lese det.



Figur 3. Parallell buss. En parallell buss består av tre delbusser som overfører hver sin type signal.

1.3.1. Databussen

Databussen er de ledningene som de binære data overføres over. Typisk består en databuss av 16, 32 eller 64 linjer. Antallet kalles databussens *bredde* og er en nøkkelfaktor for maskinens totale hastighet siden dataflytting ofte er en begrensende faktor i datamaskiner.

1.3.2. Adressebussen

Disse linjene overfører adressen til data som skal overføres. Når CPU for eksempel skal lese en lokasjon i minnet, så legges adressen til minnelokasjonen på adressebussen.

Bredden til adressebussen avhenger av minnekapasiteten til datamaskinen. Adressebussen må være så bred at en hvilken som helst adresse kan legges på adressebussen.

Adressebussen brukes også til å adressere de forskjellige I/O-moduler.

1.3.3. Kontrollbussen

Kontrollinjene brukes til å synkronisere og kontrollere utvekslingen av informasjon. Forskjellige buss-standarder kan ha ulike kontrollinjer, og vi skal snart se på de viktigste kontrollinjene.

1.3.4. Datakommunikasjon på en parallell buss

En buss brukes til å overføre binære data mellom de tilkoblede enhetene. Enhetene må være enige om hvordan bussen skal brukes, slik at det ikke oppstår forvirring om hvem som er avsender, hvem som er mottager, når det skal sendes data, hvor lenge data er gyldige og så videre.

De fleste parallelle busser er synkrone busser. Det vil si at de bruker et klokkesignal til å styre aktivitetene på bussene. Et klokkesignal er et signal som med helt faste mellomrom hopper mellom høy og lav spenning. Det er altså et pulstog med helt fast frekvens. Poenget er at en ny handling bare kan starte når det kommer en ny puls; det vil si i det øyeblikk spenningen hopper fra lav til høy verdi. Frekvensen på moderne parallelle busser kan være svært høy – gjerne flere hundre millioner pulser pr sekund. Antall pulser pr sekund kalles klokkefrekvensen og oppgis i antall Hz (Hertz).

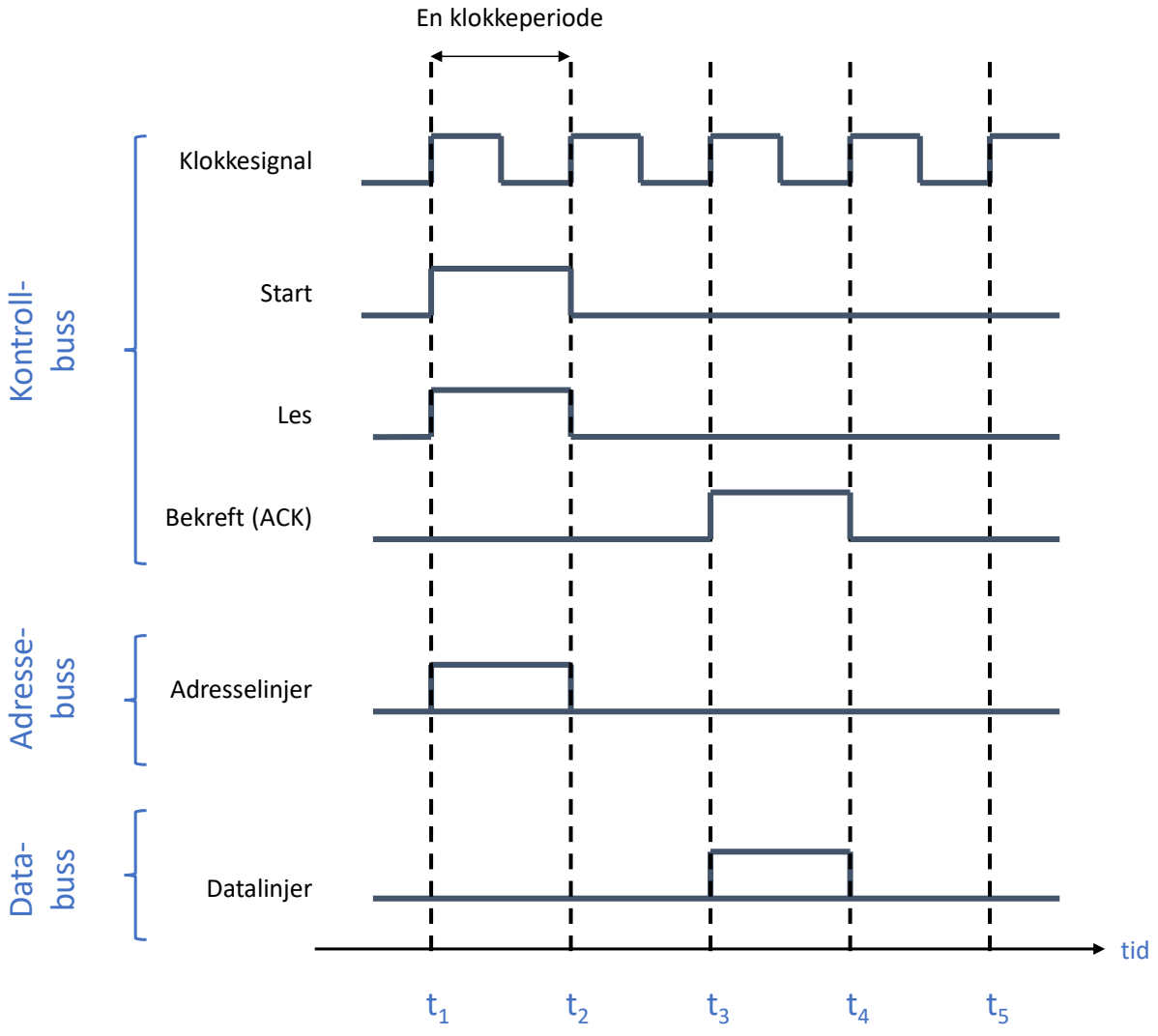
Klokkesignalet overføres på en av linjene til kontrollbussen, slik at alle enheter har tilgang til dem. Vi ser et eksempel på et slikt signal øverst i Figur 4.

Et *timing diagram* er en figur som viser hvordan linjene på bussen brukes under en overføring. Diagrammene viser den binære verdien til de forskjellige busslinjene som funksjon av tid. Tiden følger x-aksen mot høyre, og diagrammet viser hvordan hver busslinje veksler mellom høye og lave verdier til ulike tidspunkter. Timing diagram er nærmere beskrevet i Vedlegg A – Timing diagram (på side 18).

Figur 4 viser et timing diagram for lesing på en synkron parallell-buss. CPU skal lese en bestemt adresse fra minnet, og figuren viser hvordan linjene på bussen brukes under lesingen.

Øverst i figuren ser vi de enkelte linjer til kontrollbussen. Nedenfor ser vi både adressebussen og databussen.

Den første linjen til kontrollbussen viser klokkesignalet, som altså er et pulstog. Omslagene fra lav til høy spenning indikerer starten på en klokkepuls. Poenget nå er at enhver handling starter når det kommer en ny klokkepuls, og så har handlingen en klokkepuls på seg til å gjøre seg ferdig. Neste handling kan tidligst starte ved neste klokkepuls.



Figur 4. Parallell buss. Figuren viser hvordan busslinjene brukes ved lesing over en parallell buss.

I den første klokkepulsen, t_1, ser vi at CPU legger rett adresse på adressebussen. Samtidig setter den også de to kontrollinjene Start og Les. Kontrollsignalet Start varsler at en ny operasjon skal utføres. Dette kontrollsignalet gjør at alle tilkoblede enheter sjekker om adressen på adressebussen tilhører seg. Kontrollsignalet Les indikerer at det skal utføres en leseoperasjon.

Et minne vil alltid trenge litt tid til å frembringe innholdet av en lokasjon. På en parallell buss får minnet tildelt en klokkepuls til å finne frem rett lokasjon, og hente bitmønsteret som ligger der[1]. Dette er klokkepuls to, t_2.

På den tredje klokkepulsen, t_3, legger enheten ut data på databussen, og bekrefter at alt er klart ved å sette kontrollsignalet Bekreft (ACK = Acknowledge). Dermed kan CPU lese data på databussen. CPU har en klokkepuls til å lese data. Deretter nullstilles alle busslinjene, og bussen er frigitt slik at andre kan bruke den.

1.3.5. Båndbredde på en parallell buss

Som tidligere nevnt er båndbredde det antall bits som kan overføres pr sekund. På en parallell buss er båndbredden en overføring pr klokkepuls. Som vi så ovenfor oppnår vi aldri denne maksimale ytelsen fordi enkelte klokkepulser må brukes til å overføre adresser og til å vente på at dataoverføringen skal klargjøres.

Eksempel på båndbreddeberegning:
På en 32-bits buss med klokkefrekvens på 1GHz er det teoretisk mulig å overføre 32 bits over bussen 1 milliard ganger i sekundet. Det gir en båndbredde på 32 Gb/s (32 gigabits pr sekund).

For å finne båndbredde må man altså multiplisere klokkefrekvens og bredden på databussen.

1.3.6. Hva begrenser båndbredden

Etter hvert som prosessorer og andre enheter har fått stadig høyere ytelse så kreves det stadig høyere båndbredde på bussene. Vi har nettopp sett at båndbredden til en parallell buss er gitt av to ting: Bredde på databussen og klokkefrekvens. For å øke båndebredden kan vi altså øke databussbredden og/eller øke klokkefrekvensen.

Dersom vi øker bredden på databussen så øker vi også de fysiske dimensjonene på bussen og alt tilkoblingsutstyr. Parallelle busser har vært 8-bits, 16-bits, 32-bits og 64-bits. Av praktiske hensyn er det ikke gunstig å øke den ytterligere.

Men vi kan øke klokkefrekvensen. La oss se hvorfor det er grenser for hvor mye vi kan øke den. Problemet er at når man virkelig skur opp klokkefrekvensen på en parallell buss, så introduseres et viktig effekt. Signalene i de parallelle buss-ledningene vil nemlig ikke bruke nøyaktig like langt tid gjennom bussen.

Selv om elektroniske signaler forplantes hurtig gjennom en elektrisk leder, så så går det ikke uendelig fort. Signalet bruker en viss tid gjennom ledningen, og siden alle elektroniske signaler påvirkes av omgivelsene (og av hverandre), vil det være små individuelle forskjeller på tiden som hver av signalene i en parallelbuss bruker.

[1] Dersom minnet trenger mer enn en klokkepuls til å fremskaffe data må vi legge inn en pause på tilstrekkelig antall klokkepulser. Disse ekstrapulsene kalles Waitstates. Vi skal lære mer om det senere.

- Dess bredere bussen er (dess flere parallelle ledninger) dess større vil de individuelle forskjellene bli.
- Dess lenger ledningene til bussen er dess større bli ulikhetene i gangtid.

Dette har ingen betydning på lave klokkefrekvenser, fordi forskjellene vil bli små sammenliknet med tiden som går mellom hver gang det sendes en ny bit. Men det blir et problem hvis bitene kommer så tett at tiden mellom dem nærmer seg forskjellene i gangtid. Det vil til syvende og sist være denne faktoren som begrenser maksimal frekvens for en parallell buss.

Man kan altså ikke skru opp frekvensen i det uendelige; både effekten med gjensidig påvirkning mellom signalene og lengden på bussen vil begrense klokkefrekvensen. Det samme vil kvaliteten på styreelektronikken og på ledninger og koblingsmatriale.

Korte busser parallelle busser kan ha svært høy klokkefrekvense og dermed høy båndbredde, men lange busser har begrenset båndbredde.

1.3.7. Hvor bruker vi parallelle busser?

Tidligere var parallelle busser svært mye brukt. Etter hvert har serielle busser tatt over mange bruksområder, men ett viktig område står fortsatt igjen: Parallell buss brukes som *minnebuss*; dvs bussen mellom primærminnet og prosessoren.

Minnebussen er den aller hurtigste bussen vi har på en datamaskin (bortsett fra interne busser som er innebygget i prosessoren). På moderne PC er minnebussen en 64 bits parallell buss med klokkefrekvens i GHz-området.

Dersom vi betrakter et moderne hovedkort vil vi se at prosessoren og primærminnet står med bare noen centimeters mellomrom. Det er for at minnebussen skal være kortest mulig, og dermed kunne ha høyest mulig klokkefrekvens.

1.4. Serielle busser

Vi har sett at en parallell buss overfører flere bits samtidig (parallelt) i hver sin ledning på databussen. Vi har også nevnt at på en seriell buss overføres bitene èn for èn (det vil si: etter hverandre i tid). Under ellers like vilkår vil selvsagt seriell overføring ta lengre tid, men til gjengjeld har vi langt færre ledninger og en langt mindre plasskrevende mekanisk løsning.

1.4.1. Båndbredde på serielle busser

Man skulle jo tro at en parallell buss alltid vil være mye kjappere enn en seriell buss. En 16-bits parallell buss bør jo være 16 ganger kjappere enn en tilsvarende seriell buss. Isolert sett er dette sant. Men nettopp fordi det på en seriell buss bare overføres en bit i slengen så trenger man ikke ta hensyn til vekselvirkningen mellom parallelle signaler. Man har derfor ikke den samme begrensningen på klokkefrekvens og ledningslengde som man har på parallelle busser.

Derfor kan man bruke høyere frekvenser og lengre ledninger uten å få problemer. Dette, sammen med at elektronikken har blitt både billig og god, gjør at moderne seriebusser har høy båndbredde, lange ledninger og små dimensjoner på kontakter og annet koplingsmateriell.

De senere år har vi sett at serielle busser blir mer og mer populær, og tar over der det tidligere har vært brukt parallelle busser.

1.4.2. Hvor bruker vi serielle busser?

På moderne datamaskiner kobler vi til mye ulikt IO-utstyr. Eksempler kan være tastatur, mus, skanner, skriver, eksterne harddisker, kamera, videokamera osv.

Busser for IO-utstyr kalles IO-busser og for noen år siden ble IO-utstyr koblet til maskinen med mange ulike IO-busser, hvorav mange var parallelle.

Imidlertid har kravene til slike busser har endret seg dramatisk de senere år, spesielt på to områder:

1. Krav til båndbredde har økt voldsomt.
2. Krav til fleksibilitet og brukervennlighet har økt.

Dette har gjort at de gamle bussene ble alt for dårlige, og det ble utviklet nye og langt bedre IO-busser. Det mest kjente eksemplet på en slik buss er antagelig USB (Universal Serial Bus).

Det er tre ting som skiller disse moderne IO-bussene fra de mer tradisjonelle bussene.

1. Serielle busser istedenfor parallle
   Serielle busser kommer til sin rett når vi skal koble IO-utstyr til datamaskinen, fordi det er en stor fordel med små dimensjoner på koplingsmatriell, lange ledninger og god båndbredde.
2. Hot swap.
   På de fleste tradisjonelle busser må man skru av datamaskinen før man kobler til nytt utstyr eller kobler fra eksisterende enheter. På moderne IO-busser som USB kan vi koble til nye enheter selv om strømmen står på og maskinen kjører (*hot swap* på engelsk). Grunnen til at hot swap er nyttig er selvfølgelig at bussene brukes mot mange ulike typer IO-utstyr som det er kjekt å kunne koble til og fra uten å stoppe maskinen. Det er viktig å merke seg at hot swap **ikke** har noe å gjøre med at bussen er seriell (vi kan fint ha hot swap på en parallell buss også). For å få hot swap til å fungere må elektronikken være laget for det, og vi må ha programvare som håndterer det.
3. Tettere samarbeid mellom bussen og operativsystemet enn før.
   På moderne busser er operativsystemet mye mer involvert i bussoverføringene enn på mer tradisjonelle busser. Dette har sammenheng med mulighetene til å koble til/fra utstyr mens maskina er på (hot swap). Siden operativsystemet inneholder kode som styrer IO-utstyret er det viktig at operativsystemet får beskjed når nytt utstyr kobles til. Når operativsystemet får beskjed om hvilket utstyr som ble koblet til vil det laste inn rett driver, og sørge for at utstyret kan tas i bruk.

Det er prinsippielt ingenting i veien for at en parallell buss også kan bruke hot swap, og hatt det samme nære samspillet med operativsystemet som moderne serielle busser. Da man skulle utvikle nye busser for tilkobling av IO-utstyr valgte man serielle busser fordi den har god båndebredde samtidig som ledninger, plugger og alt annet koblingsmatriale blir av betydelig mindre fysisk størrelse når man har få ledninger. Noe som er en praktisk fordel i forbrukersammenheng.

Vi skal se på noen eksempler på moderne IO-busser og utviklingen de har vært gjennom.

1.4.3. USB og Firewire

Nye PCer er utstyrt med en såkalt USB (Universal Serial Bus) der man kan koble til svært mye ulikt utstyr. USB er en seriell buss, og standarden ble utviklet i et samarbeid mellom Compaq, Intel, Microsoft og NEC. Den første spesifikasjonen kom i 1994, og ble senere fulgt

opp av versjon 1.1, og deretter versjon 2.0 og versjon 3.0. Opprinnelig var hensikten med USB å lage en enkel tilkoblingsstandard for sammenkobling av datamaskinen og telefon, men etter hvert ble målet atskillig mer ambisiøst; Nemlig å lage en tilkoblingsstandard som var enkel, der man kunne koble mange enheter med tilstrekkelig hastighet til at "det meste" av utstyr kunne kobles til. USB 2.0 har tilstrekkelig hastighet til at for eksempel videokamera kan kobles til datamaskinen. USB 3.0 er hurtig nok til høyoppløselig grafikk.

På en USB kan man koble inntil 127 enheter. USB 1.1 spesifiserte to hastigheter. Senere har USB 2.0 og deretter USB 3.0 spesifisert høyhastighetsmodi i tillegg. Derfor snakker man om følgende hastigheter på en USB:

- Low Speed - 1.5Mbits/s
- Full Speed - 12Mbits/s (USB 1.1)
- High Speed - 480Mbits/s (USB 2.0)
- SuperSpeed – 5Gbits/s (USB 3.0)

USB har slått an i PC-markedet, og det er jo ikke rart siden både Intel og Microsoft har vært med å utvikle standarden.

Den viktigste utfordreren til USB har vært Firewire (der blant annet Apple har deltatt i utviklingen). En av de grunnleggende tingene som skiller USB fra Firewire er arbitreringen; USB er en ekstremt sentralisert buss der PCen kontrollerer alt som skjer, og blant annet løser alle konflikter. Firewire er en distribuert buss. Dermed kan (i hvert fall i prinsippet) for eksempel to videokamera kobles sammen med Firewire uten at det er noen datamaskin med i bildet i det hele tatt.

1.4.4. IDE/EIDE/ATA ble til SATA

Mens tastatur, mus og skriver er eksempler på ekstern IO-utstyr så er harddisk og DVD eksempler på internt IO-utstyr. I tidligere leksjoner har vi lært at slikt I/O-utstyr alltid koples til en kontroller som står mellom hovedbussen og I/O-utstyret. Som regel er det slik at når vi først har en kontroller, så kan denne kontrolleren styre mer enn ett eksemplar av I/O-utstyret. For eksempel kan en harddiskkontroller gjerne styre to eller flere harddisker. For å få en enkel og fleksibel løsning kobles ofte I/O-utstyret til kontrolleren via en bussløsning.

På PC er standarden IDE den vanligste I/O-bussen for tilkobling av harddisker. Denne standarden kalles også ATA (vel, egentlig er ikke EIDE og ATA helt det samme, men det skyldes i alt vesentlig en strid mellom produsenter om beskyttelse av merkenavn, i leksjonen bruker vi EIDE og ATA som om det var det samme).

IDE/EIDE/ATA ble opprinnelig laget for å koble til harddisker. Med etter hvert har det kommet mye annet utstyr for EIDE/ATA. Det finnes cdrom-lesere, cdrom-brennere, dvd-spillere og annet utstyr.

EIDE/ATA benyttet i mange år parallelle busser. På de nyeste versjonene har man imidlertid innført seriell kommunikasjon; såkalt Serial ATA eller S-ATA. Dette har falt sammen med at vi fikk SSD-disker. SSD-disker har mye høyere krav til båndbredde enn tradisjonelle magnetiske harddisker, og derfor har vi sett en voldsom utvikling av SATA-standarden de siste årene.

1.4.5. SCSI-bussen ble til SAS

Mens EIDE/ATA er vanlig på ordinære PCer så benytter dyrere installasjoner (f.eks tjener-maskiner i nettverk) oftest en annen standard som kalles SCSI. I/O-utstyret til SCSI er dyrere enn vanlig PC-utstyr, men til gjengjeld er spesifikasjonene bedre. Det finnes mye SCSI-utstyr.

Det mest vanlige er harddisker med høy ytelse (ofte i form av flere harddisker som samarbeider i en såkalt RAID-løsning) og båndstasjoner for sikkerhetskopiering.

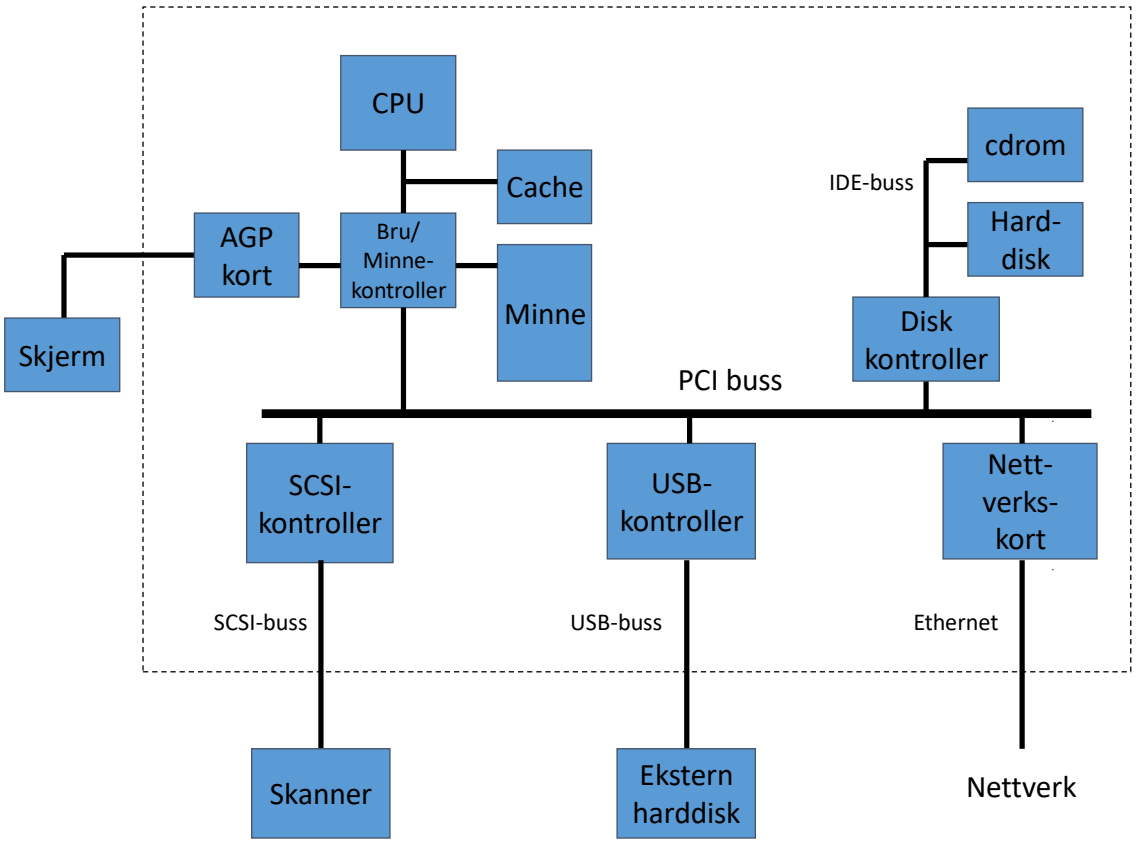
Den opprinnelige SCSI-standarden ble spesifisert på 80-tallet, og har vært gjennom en lang rekke ulike versjoner. De senere årene har vi sett tilsvarende utviklking som på IDE/ATA, nemlig en overgang til seriell dataoverføring. Disse nye versjonene kalles SAS (Serial Attached SCSI), og har den samme voldsomme utviklingen av båndbredde som SATA.

1.5. Hovedbussen

På eldre PCer var det en buss som het PCI som var «hovedbussen» på PC. Den hadde tilstrekkelig hastighet for det aller meste av I/O-utstyr, og det fantes svært mye I/O-utstyr som var tilpasset PCI-bussen.

Figur 5 viser en typisk buss-struktur på en slik PC. Vi ser at PCI-bussen er sentral, og at den kommuniserer med kontrollerene (I/O-modulene). PCI-bussen var en parallell buss med 32 bits bredde.

I figuren er det også tegnet inn AGP. AGP var tilkoblet omtrent på samme måte som minnet: med en mer eller mindre direkte forbindelse til bro/minnekontrolleren. På en PC uten AGP ville videokontroller være tilkoblet PCI-bussen.



Figur 5. Typisk oppbygging av en eldre PC bygget rundt en parallell PCI-buss.

1.5.1. Problemer med PCI

I likhet med alle andre busser har også hovedbussen vært gjennom en voldsom utvikling de senere år. Årsaken er den samme: Kravene til båndbredde og fleksibilitet har økt.

Delte busser har den fordelen at de er enkle, billige og enkle å implementere. Men, som det fremgår av kapittel 1.2.4, så er det et fundamentalt problem med delte busser at det bare kan foregå en overføring i slengen. Så lenge det ikke går alt for mye trafikk fungerer delte busser utmerket. Men når datatrafikken blir stor vil vi stadig oftere oppleve at flere enheter får behov for kommunikasjon samtidig. Når man har presset klokkefrekvensen til det ytterste, og fortsatt ønsker en høyere båndbredde må man ta i bruk andre metoder enn en slik delt buss. Man må benytte teknikker der det kan foregå flere overføringer samtidig.

De som kan nettverkslære vil kunne dra paralleller til utviklingen innen lokale nettverk, der Ethernet i mange år har vært den vanligste standarden. På 80- og 90-tallet var Ethernet basert på delte løsninger; løsninger der alle konkurrerte om tilgang til den samme ledningen. Dette fungerte bra så lenge det var lite datatrafikk på nettet, men dårligere og dårligere etter hvert som trafikken økte. På midten av 90-tallet kom svitsje-teknologien for nettverk. Med svitsje-teknologi settes det opp en direkte forbindelse mellom to enheter som skal utveksle informasjon, samtidig som det også kan settes opp parvise sammenkoblinger mellom andre enheter. På denne måten kan flere overføringer foregå samtidig, og den totale båndbredden forbedres vesentlig. Moderne lokale nettverk benytter kun svitsjeteknologi, og busstopologien er helt forlatt innenfor nettverk.

På mikromaskiner som PC har vi sett en lignende historisk utvikling. Båndbreddekravene har blitt så store – og elektronikken så billig – at man har begynt å bruke svitsjeteknologien.

1.6. Svitsjede busser

Nye PCer er utstyrt med standarden *PCI-Express*, som er en buss som baserer seg på svitsje-teknologi.

1.6.1. Hva er svitsjet kommunikasjon?

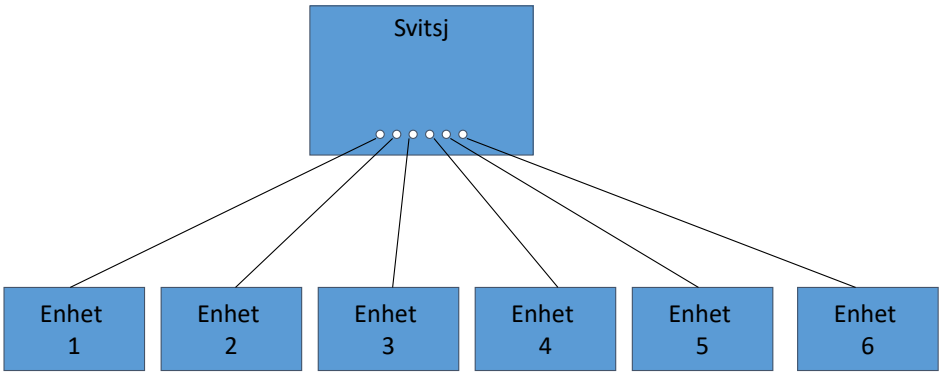
For å forklare hva svitsjet kommunikasjon er vil jeg bruke et eksempel, nemlig telefon-sentralen.

Telefonsentralen gjør det mulig å koble sammen to telefonapparater, slik at man kan overføre informasjon mellom dem. Virkemåten til telefonsentralen er kort fortalt at den er et knute-punkt for mange telefoner. Det går en ledning fra hvert apparat og til telefonsentralen. Ved behov – det vil si når noen ringer – kobler sentralen sammen de to respektive telefonlinjene som skal kommunisere. Telefonsentralen setter altså opp en punkt-til-punkt-forbindelse mellom to telefonapparater.

Men en annen egenskap er enda viktigere i forbindelse med båndbredde. Telefonsentralen kan nemlig sette opp flere telefonsamtaler samtidig. Disse telefonsamtalene pågår helt uavhengig av hverandre. Dette er et viktig poeng: Uansett hvor mange sammenkoblinger som er gjort, kan telefonsentralen sette opp en til, forutsatt at mottaker ikke er opptatt i en annen sammen-kobling. Alle de samtidige samtalene foregår helt uavhengig av hverandre.

I forbindelse med busser og kommunikasjon på datamaskiner, er det ikke telefoner som skal kobles sammen; det er datamaskinens ulike komponenter. Den enheten som tilsvarer telefon-sentralen er en såkalt *svitsj*. Svitsjen har en rekke tilkoblingspunkter, og på hvert punkt kan vi koble til en komponent. Svitsjens oppgave er å koble sammen de ulike komponentene ved

behov. Når to komponenter er koblet sammen kan de kommunisere. Dessuten – og det er den viktige forskjellen fra delte busser – kan to andre komponenter kommunisere akkurat samtidig over en annen sammekobling.



Figur 6 Svitsj. En svitsj kobler sammen enheter på parvis basis. Flere samtidige sammenkoblinger kan settes opp, og hver av dem vil være en punkt-til-punkt-forbindelse mellom to enheter som skal kommunisere.

Når man skal lage en standard for svitsjede busser må man gjøre en rekke valg. For eksempel: Hvordan skal kommunikasjon mellom en enhet og svitsjen foregå? Skal den være seriell eller parallell? Hvilken frekvens skal brukes? Hvordan skal man skille mellom de ulike enhetene? Man trenger jo en form for adressering. Hvordan skal man prioritere enhetene dersom flere ønsker å kommunisere med samme enhet samtidig. Forhåpentligvis er det lett å se at man trenger en standard.

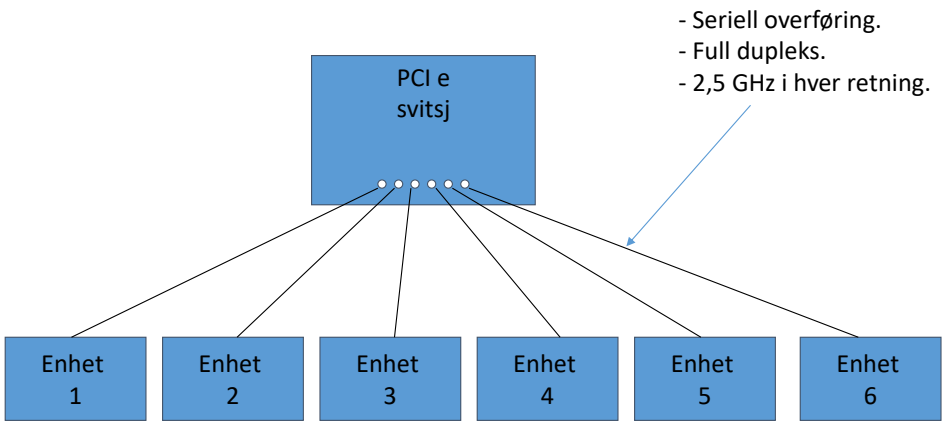
1.6.2. PCI-Express

PCI-Express er en svitsjet buss som ble utviklet for å erstatte PCI som hovedbuss på mikro-datamaskiner. PCI-Express har en rekke interessante egenskaper som vi skal se på. Ofte brukes den kortere skriveformen *PCI e* for PCI-Express.

Den sentrale delen på en svitsjet buss er selvsagt selve svitsjen (den tilsvarer telefonsentralen i eksemplet lenger opp). Rent fysisk er svitsjen på PCI-Express en del av det såkalte brikke-settet. Brikkesettet er noen elektroniske kretser som finnes på alle PCer. Vi skal se nærmere på brikkesettet i en annen leksjon som heter *Systemarkitektur*. Foreløpig slår vi bare fast at det finnes en sentral elektronisk krets som vi kaller *svitsjen*. Til svitsjen kobles alle de ulike komponentene som skal kunne kommunisere.

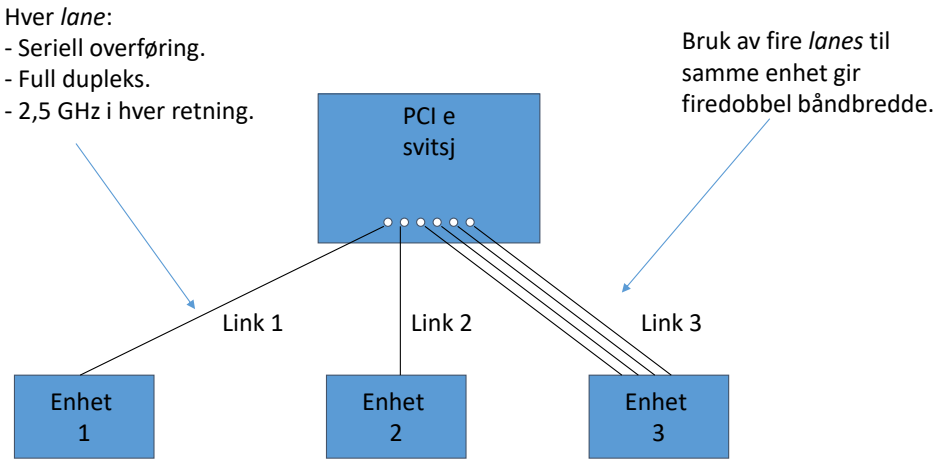
På PCI-Express benyttes det seriell kommunikasjon mellom svitsj og hver av de tilknyttede enheter. Bitene sendes altå etter hverandre igjennom et ledningspar. Det brukes for øvrig to ledningspar; ett i hver retning, slik at kommikasjonen er *full dupleks* (se kap 1.2.2). Bånd-bredden på overføringen er 2.5 Gb/s i hver retning[2]. I PCI-Express-terminologi kalles en slik sammenkoblingen av svitsj og en enhet for en *lane* (*lane* betyr kjørebane på engelsk). Figur 7 viser seks enheter som er knyttet til en svitsj via hver sin *lane*.

[2] Det er versjon 1 som bruker 2,5 GHz. Standarden åpnet for senere utvidelse av båndbredden (høyere klokkefrekvens) etter hvert som bedre elektronikk kommer på markedet. Versjon 2 bruker 5 GHz. Versjon 3 bruker 8 GHz, men har en mer effektiv dataoverføring slik at den reelle båndbredden er nær en dobling sammenlignet med versjon 2. Versjon 4 bruker 16 GHz.



Figur 7 PCI-Express. PCI-Express er en svitsjet buss der svitsjen setter opp punkt-til-punkt-forbindelse mellom to enheter når de skal kommunisere. I figuren er det 6 enheter som er knyttet til en svitsj via hver sin *lane*. Hver *lane* har en båndbredde på 2.5 Gb/s i hver retning.

En av de smarte tingene med PCI-Express er at båndbredden mellom svitsjen og en enhet kan utvides ved å bruke flere *lanes* mellom svitsjen og enheten. Hver *lane* vil da gi en båndbredde på 2.5 GHZ. Dersom man bruker to *lanes* blir båndbredden dobbel, og dersom man bruker 4 *lanes* blir den firedobbel. Man snakker om PCI-Express X1, X2, X4, X8 osv. , der tallet angir antall *lanes* som brukes. På denne måten kan man tilpasse båndbredde etter behov. Et grafikk-kort kan for eksempel bruke 8 *lanes* siden det er båndbreddekrevende, mens et nettverkskort kan bruke en *lane*.



Figur 8 Linker i PCI-Express. På PCI-Express er hver enhet tilknyttet svitsjen via en link. Hver link kan bestå av en eller flere *lanes*.

Men er vi ikke da tilbake til det samme problemet som med parallelle busser? Ved bruk av flere lanes vil det jo gå flere ledninger mellom svitsj og enhet, og parallelle busser fikk jo problemer når bussbredden ble stor. Svaret er nei. Og grunnen til at vi ikke får problemer når vi bruker mange *lanes* til en og samme enhet i PCI-Express, er at trafikken på hver lane er uavhengig av trafikken på de andre. Dermed vil små ulikheter i gangtiden på de ulike lanes være uten betydning. Vi innfører et nytt begrep med det samme, nemlig *link*. Tilkoblingen

mellom svitsjen og en enhet på PCI Express kalles alltid en *link*, og en link kan bestå av èn, to, fire, åtte eller 16 *lanes.* Figur 8 viser en svitsj med tre enheter tilknyttet via tre *linker*. Link 1 og Link 2 består av 1 *lane* hver, mens Link 3 består av fire *lanes*.

1.6.3. Datakommunikasjon på PCI-Express

Selv ved først øykast minner jo figurene vi har sett av PCI-Express på et lite nettverk. Og faktum er at PCI-Express oppfatter svitsjen og tilknyttede enheter som et høyhastighets data-nettverk, og har tatt i bruk svært mange av de teknikkene som vi kjenner fra nettverkslæren.

Siden dette ikke er et kurs i datakommunikasjon og kommunikasjonsstandarder skal vi ikke gå dypt inn i materien. Men vi nevner at PCI-standarden definerer en protokoll og en lagdelt arkitektur som minner om datanettverk vi kjenner fra andre sammenhenger. PCI har implementert de fire laveste lagene i OSI-modellen.

På PCI-Express brytes datamengden som skal overføres opp i pakker som sendes over bussen. Hver pakke utstyres med adresseinformasjon (og en del annen administrativ informasjon), og svitsjen sender pakken til rett mottaker. Mottager samler opp pakkene, og setter dem sammen igjen. Dette er svært likt virkemåten til vanlige datanettverk.

Standarden gjør det mulig å prioritere pakker på ulike måter. Blant annet ved å prioritere sanntids-trafikk høyt, og ved å tilby en garantert båndbredde over tid. Det siste er blant annet nyttig i forbindelse med multimediaanvendelser. Disse involverer nemlig streaming av video og lyd. Det vil si langvarige strømmer av bitmønstre som må overføres i en bestemt hastighet.

Det finnes for øvrig to typer pakker: datapakker (som inneholder data, og som er mest interessante for oss) og kommandopakker. Kommandopakker brukes til å administrere bussen og for å sende ulike former for kontrollbeskjeder. Mange av de signalene som går over egne linjer på kontrollbussen til en parallell buss, går som kontrollpakker PCI. For eksempel over-føres avbruddssignaler (som vi skal se på i en senere leksjon) som kontrollpakker på PCI, mens dette signalet går over en egen dedikert kontroll-linje på de fleste parallelle busser (for eksempel på ISA-bussen).

1.7. Oppsummering

I denne leksjonen har vi sett hvordan de ulike komponentene i en datamaskin utveksler informasjon via busser.

Bussene kan realiseres på mange ulike vis, og vi har sett at hovedtypene er

- parallelle busser
- serielle busser
- svitsjede busser

På en datamaskin brukes det mange busser som sammen utgjør et busshierarki. Bussene har ulik båndbredde og ulike egenskaper som er tilpasset bruken. På dagens maskiner brukes følgende hovedkategorier med tilhørende busstype:

- Minnebuss: Bred parallellbuss med høy klokkefrekvens.
- Hovedbuss: Svitsjet buss av type PCIe som tilpasser båndbredden til mange ulike bruksområder ved å tildele et passende antall *lanes* til hver enhet.
- IO-busser: Serielle busser med god båndbredde og små dimensjoner på koplings-materialet og som kan benytte lange ledninger. Eksempler er USB og SATA.

Vi har sett på mange ulike begreper og mange ulike busstyper. Dersom vi går litt bort fra detaljene, og heller ser på utviklingen over litt tid, så ser vi et par interessante utviklingstrekk.

I mange år ble et stadig høyere krav til båndbredde besvart med stadig bredere parallelle busser med stadig høyere frekvens. Slike busser har imidlertid sine problemer. Både ved at elektromagnetisk vekselvirkning gjør at signalene påvirker hverandre, og ved at kabler og koblingsmatriell blir fysisk romstore. De siste årene har pendelen derfor slått andre veien, og fokus har kommet på hurtige serielle busser.

Vi har også sett at etter hvert som svitsjeteknologien har blitt billigere har svitsjede busser med seriell kommunikasjon ut mot hver enkelt enhet, blitt en realitet selv på mikromaskiner.

I en litt annen gate kan vi se en dreining fra at bussen tidligere bare var en mer eller mindre passiv maskinvare, mens vi nå har fått et større fokus på programvare og integrasjon med operativsystemet. På tidligere busser var bussen i større eller mindre grad utelukkende et sett av ledninger som førte elektriske signaler mellom maskinvarekomponenter. Moderne busser tar i bruk kunnskap fra nettverkslære, og opptrer i stor grad som hurtige små nettverk der det overføres pakker. Avsender deler en datamengde opp i pakker og sender disse, mens mottaker tar i mot pakkene og setter sammen datamengden igjen. Dette skjer med hjelp av program-vare. Moderne bussteknologi baserer seg altså på nær integrasjon mellom maskinvare og programvare.

Vedlegg A – Timing diagram

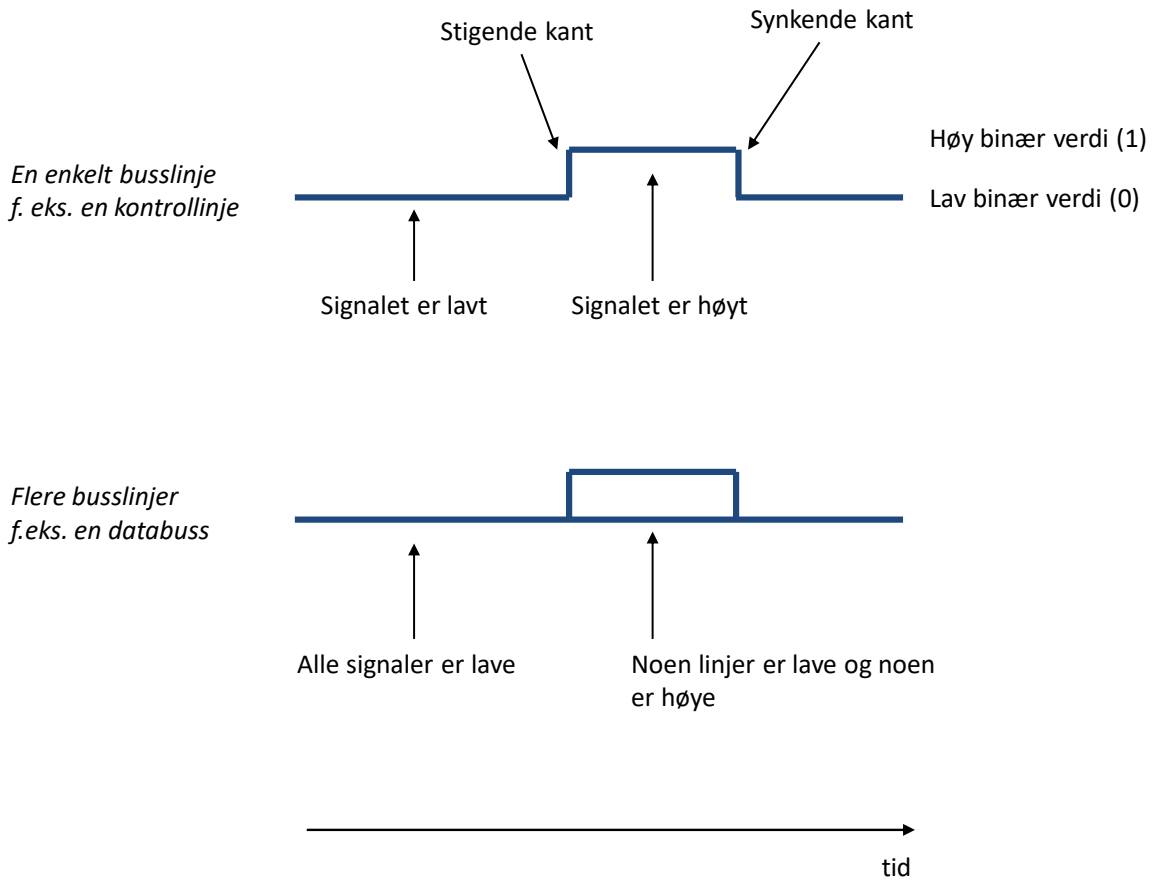
I forbindelse med busser ble det brukt timing-diagram for å vise hvordan signalene på de ulike busslinjene varierte som funksjon av tiden.

En busslinje kan overføre binære signaler som enten er høy eller lav, altså binær verdi 1 og 0. I timing-diagrammene brukes en høy verdi som den binære verdien 1 og en lav verdi som binærverdien 0 (selv om enkelte datamaskiner er implementert motsatt). På bussens linjer er normal "hvilestilling" den lave verdien. Det betyr at det er den lave verdien som ligger på linjene når bussen ikke er i bruk til signaloverføring.

I timing-diagrammene tegnes omslagene mellom spenningsnivåene slik at det ser ut som omslaget skjer momentant – det vil si plutselige hopp mellom spenningsnivåene. I praksis vil det ta litt tid å endre spenningene på en reell buss, men siden denne tiden er kort i forhold til varigheten av alle buss-signaler tegner vi omslagene som momentane omslag.

Omslag fra lav til høy verdi kalles stigende kant og omslag fra høy til lav verdi kalles synkende kant.

Som regel tegnes diagrammene slik at vi ser hvert kontrollsignal for seg. Et diagram for en enkelt linje er vist øverst i Figur 9. For andre linjer, for eksempel alle linjene i databussen eller adressebussen, er ikke signalverdien for hver enkelt linje interessant. Da tegner vi alle linjene sammen på en slik måte at enkelte linjer er høy og andre er lave. Et eksempel er vist nederst i Figur 9.



Figur 9. Timing diagram. Øverst ser vi et diagram for en enkelt busslinje. Nedenfor ser vi et diagram for en gruppe av linjer.