NTNU
Kunnskap for en bedre verden

Institutt for informatikk og e-læring, NTNU

Moderne primærminne

Geir Ove Rosvold

14. november 2017

Moderne primærminneteknologier

Resymé: Leksjonen beskriver hvordan CPU, cache, buss og minne samarbeider for å oppnå maksimal ytelse. De viktigste RAM-typene beskrives. Det legges vekt på prinsippene om SDRAM og DDR-SDRAM.

Innhold

1.1. Systemarkitektur: å koble det hele sammen

Ut fra det som er sagt i de forrige leksjonene er det lett å se at prosessoren er svært avhengig av å få rask tilgang til instruksjoner og data.

CPU, cache, buss og primærminne er altså nødt til å jobbe effektivt sammen for ytelsen skal bli høy. Tidligere har vi diskutert disse komponentene hver for seg. Nå skal vi se litt på samspillet mellom dem. Dette området er en del av området som ofte kalles Systemarkitektur, altså skal vi se hvordan de ulike komponenter samarbeider og samvirker. En viktig del av maskinvaren i denne forbindelse er det såkalte brikkesettet, som introduseres i siste del av denne leksjonen.

1.1.1. Bussens klokkefrekvens

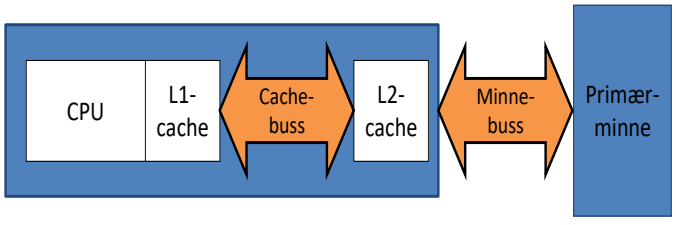
Busser brukes til å overføre informasjon mellom de ulike deler av datamaskinen. Bussen som overfører informasjon mellom minne og cache er en parallell synkron buss. I leksjonen om busser lærte vi at det på slike busser finnes en klokke som gir pulser med en konstant frekvens. Dette pulstoget overføres over en av kontroll-linjene, og alle handlinger starter på en slik puls. Frekvensen – altså antall pulser pr sekund –kalles bussens *klokkefrekvens*, eller bare *bussfrekvensen*.

Vi har også lært at det finnes en egen klokke inne i CPU som gir klokkepulser til CPU. Alle hendelser internt i CPU starter på en slik puls. På de første PCene brukte man samme klokke-frekvens både på prosessoren og på bussen. Vi har sett at klokkefrekvensen på prosessorer har økt jevnt og trutt i mange år. De øvrige komponentene i datamaskinen har ikke greid å henge med på denne økningen i frekvens. Dette gjelder også bussfrekvensen. På moderne maskiner er derfor bussenes klokkefrekvens mye lavere enn prosessorens klokkefrekvens.

På de første PCene brukte man bare en buss, og koplet både primærminne og kontrollere på denne ene bussen. Som vi har sett i leksjonen om busser, opererer vi nå med et busshierarki; vi har flere busser med ulike klokkefrekvenser og ulik bredde. Blant annet har vi:

- PCI Express (svitsjet seriell buss). Det er på denne bussen man kobler til nettverkskort, lydkort og andre enheter med moderate båndbreddekrav.
- Minnebuss: På denne bussen overføres instruksjoner og data mellom minne og CPU Minnebussen er en synkron parallell-buss med svært høy klokkefrekvens. På moderne PCer brukes en 64-bits minnebuss som kan ha en klokkefrekvens i GHz-området. Et annet vanlig navn på minnebussen er systembuss.
- Internt i CPU finnes det busser som tar seg av trafikken mellom cachene. Moderne prosessorer har innebygget L2-cache og gjerne L3-cache også.

Dette er fremstilt i Figur 1. I første halvdel av denne leksjonen skal vi se hvordan minne-bussen, primærminnet og L2-cachen samhandler.



Figur 1. Vi skal se hvordan cachene, primærminne og bussene samarbeider om å fore CPU med instruksjoner og data på moderne datamaskiner.

1.1.2. Prosessor uten cache (repetisjon)

Bare for repetisjonens skyld skal vi begynne med å se på en enkel prosessor uten cache. I den enkleste beskrivelsen av datamaskinens virkemåte (i leksjonen *Grunnleggende virkemåte*) sa vi at CPUen leser en instruksjon fra minnet og utfører den. Så leser den neste instruksjon og utfører den. Slik fortsetter det til programmet er ferdig.

Vi lærte fremgangsmåten for å hente en instruksjon i *leksjonen om busser*:

1. Med hjelp av adressebussen overføres instruksjonens adresse fra prosessoren til minnebrikken. Dette tar en bussklokkesyklus.
2. Deretter må vi vente til minnebrikken har fremskaffet informasjonen. Dette tar minst en klokkesyklus, men i praksis flere. Se om Waitstates i leksjonen om busser.
3. Med hjelp av databussen overføres innholdet av lokasjonen fra minnebrikken og til prosessoren. Dette tar en bussklokkesyklus.

Så i praksis: Hvor lang tid tar en slik lesing fra minnet? Det avhenger av to ting; minnets aksesstid og bussens klokkefrekvens.

Aksesstid: I leksjonen om primærminne lærte vi at aksesstiden til primærminnet måles i nanosekunder (10^{-9} sekund). DRAM har aksesstid på flere 10-talls nanosekund; typisk 35-70 nanosekund selv for moderne DRAM.

Klokkefrekvens: På en minnebuss starter alle handlinger på en ny klokkepuls. Det vil si at alt som skjer vil ta et helt antall klokkepulser. På nye PCer brukes gjerne minnebusser med klokkefrekvens på rundt 1000 MHz (1 GHz). På billig utstyr (og gammelt utstyr) er klokkefrekvensen lavere. På utstyr med lav ytelse – men med krav til lavt strømforbruk og begrenset mulighet til kjøling – kan minnebussens klokkefrekvens være ned i noen ti-talls MHz.

La oss se på et regneeksempel med tall som er enkle å regne med:

Anta et DRAM-minne med aksesstid på 70 ns. Aksestid er den tiden det tar fra minnebrikken har fått adressen, og til innholdet av denne lokasjonen er gjort tilgjengelig på databussen.

Anta videre en minnebuss på 33 MHz. Dette er en lav frekvens, men prinsippene er nøyaktig de samme når frekvensen er høyere. Hvor lang tid er det mellom hver klokkepuls med en frekvens på 33 MHz? Det må bli èn dividert med 33 millioner, som er 30 ns (egentlig 30,3 men vi regner med 30).

Ta vi utgangspunkt i de tre punktene ovenfor vil vi altså første trenge en puls til å overføre adressse. Deretter må vi vente til minnet har funnet frem til rett lokasjon og lest innholdet derfra: Minnets aksesstid er 70 ns. Hver klokkepuls er 30 ns, og vi må vente tilstrekkelig mange klokkepulser til å være sikker på at minnet har funnet frem ønsket innhold. Etter èn puls har det gått 30 ns og det er for lite. Etter to pulser har det gått 60 ns som også er mindre enn 70. Derfor må vi vente tre pulser (90 ns). Til sist trengs en puls for å overføre lokasjonens innhold over databussen.

Konklusjon: Å hente en minnelokasjon krever totalt 5 klokkepulser med denne aksesstiden og denne bussfrekvensen: En klokkepuls for å overføre adresse, tre for å vente til lokasjonens innhold er tilgjengelig, og en til å overføre innholdet. Fem klokkepulser à 30 ns vil si at hele overføringen tar 150 ns.

Med andre ord: På en datamaskin uten cache (og med aksesstid og bussfrekvens som ovenfor) tar det fem klokkesykluser å hente hver instruksjon. Siden hver eneste instruksjon skal hentes

fra minnet før den utføres, vil det si at prosessoren – uansett hvor hurtig den selv jobber – bruker minst 150 ns pr instruksjon.

Hvis hver instruksjon tar 150 ns er antall instruksjoner pr sekund lik 1/150 ns = 6,67 millioner instruksjoner pr sekund. Dette er en relativt langsom prosessor. Så hva er det som gjør at moderne prosessorer ikke trenger å vente så mye på instruksjoner og data? Jo, fordi de bruker cache!

1.1.3. Prosessor med cache

Med cache blir dette litt modifisert. Med cache henter vi aldri en enkelt instruksjon fra minnet. Isteden leser vi en stor (sammenhengende) *blokk* fra minnet. Så legger vi blokka i cache. Deretter kan prosessoren lese instruksjoner fra cachen i relativt lang tid.

Hvor stor er denne blokka? Det varierer fra datamaskintype til datamaskintype. På en moderne PC brukes vanligvis en blokkstørrelse på 64 bytes.

Overføring av blokker

Moderne minnebusser er 64 bits busser. Det vil si at databussen har 64 linjer. Derfor kan vi overføre 64 bits samtidig. Hvor mange bytes er 64 bits? Jo, det er 8 bytes.

På en klokkesyklus kan vi altså overføre 8 bytes. Men vi skal ikke overføre bare 8 bytes; vi skal overføre 64 bytes. Derfor må 64-bytes blokka deles opp i smådeler à 8 bytes som overføres etter hverandre. For å overføre 64 bytes over en 64 bits (8-bytes) buss må vi bruke åtte ”bussturer”.

Jeg eier en minibuss med plass til 8 passasjerer. Dersom jeg skal frakte en klasse på 64 studenter fra A til B må jeg kjøre 8 bussturer for få med alle studentene.

Hvis vi bruker samme fremgangsmåte som ovenfor (kapittel 1.1.2) vil hver av de 8 "buss-turene" trenge fem klokkesykluser. Spørsmålet er om vi kan optimalisere systemet vårt for å gjøre overføringen kjappere.

Når jeg skal frakte studentene må jeg først gå til parkeringsplassen og hente minibussen der. Det tar en god del tid. Det betyr at hele klassen må vente litt mens jeg går.

Men når jeg er ferdig med å frakte de første åtte studentene, så parkerer jeg ikke bussen mellom hver av de syv neste turene. Jeg fortsetter til jeg er helt ferdig. Dermed sparer jeg mye tid på hver av de siste syv turene.

Med andre ord: Det er en ventetiden før jeg kommer i gang med den første turen, men når jeg først er i gang så går det fort med resten av turene.

Moderne minneteknologier er laget slik at når vi har overført en adresse, så ser minnet på dette som en startadresse. Den henter rett adresse, og deretter fortsetter den å hente de påfølgende adressene helt til hele blokka har blitt lest.

Overføring av en blokk skjer derfor på følgende måte:

1. Overfør en adresse til minnet (dette tar èn klokkesyklus)
2. Vent til minne-elektronikken har funnet frem til rett lokasjon (ventetiden er bestemt av aksesstiden, men man må vente et helt antall buss-sykluser)

3. For hver ny klokkesyklus inntil hele blokka er lest: Legg innholdet av den påfølgende lokasjonen ut på databussen

Det betyr at når man først har kommet i gang så kan man aksessere en hel minneblokk fortløpende: en ny lokasjon hver gang det kommer en ny klokkepuls. Dette kalles *burst mode* på engelsk. På norsk brukes gjerne begrepet *kontinuerlig aksess*.

Kontinuerlig aksess utføres altså ved at man begynner med å overføre en startadresse. Deretter må vi vente noen klokkesykluser (aksesstiden). Men så kommer hele blokka fortløpende uten pause! I hver eneste klokkesyklus overføres nytt innhold. Sagt på en annen måte: Det tar litt tid å komme i gang, men så kommer data som erter av en sekk.

På prosessorer med cache er det alltid burst-modus vi benytter. Grunnen er at vi skal overføre en hel blokk, og bussen er for smal til at hele blokka kan overføres samtidig.

1.1.4. Tidsforbruk for overføring i burst-modus

Tidsforbruket for å overføre en blokk fra minnet til cache i burst-modus består dermed av to hovedbidrag:

1. Finn frem til rett startlokasjon
2. Les fortløpende lokasjoner så kjapt som mulig med burst

La oss se på de to bidragene.

Å finne startlokasjonen

Tiden som går med til å finne startlokasjonen er bestemt av aksesstiden til minnet. For vilkårlig aksess er ventetiden 35-70 ns på moderne DRAM. Legg merke til at ventetiden er uavhengig av klokkefrekvensen på bussen – utover det at vi må vente et helt antall sykluser.

Antall sykluser vi må vente er avhengig av bussfrekvensen. Har vi en høy bussfrekvens så går det kort tid mellom hver puls, og vi må bruke mange sykluser før aksesstiden har gått. Med lavere bussfrekvens går det lenger tid mellom hver puls, og antall pulser i løpet av ventetiden blir færre.

Figur 2 viser dette for to ulike bussfrekvenser. Øverst ser vi tidsforbruket på en langsom buss, og nederst fremstilles tidsforbruket på en hurtig buss. Legg merke til at aksesstiden er like lang i begge tilfeller. Antall klokkepulser i løpet av aksestiden er høyere dess høyere bussfrekvensen er. I kap 1.3 skal vi for øvrig se hvordan vi kan gjøre den reelle ventetiden så kort som mulig.

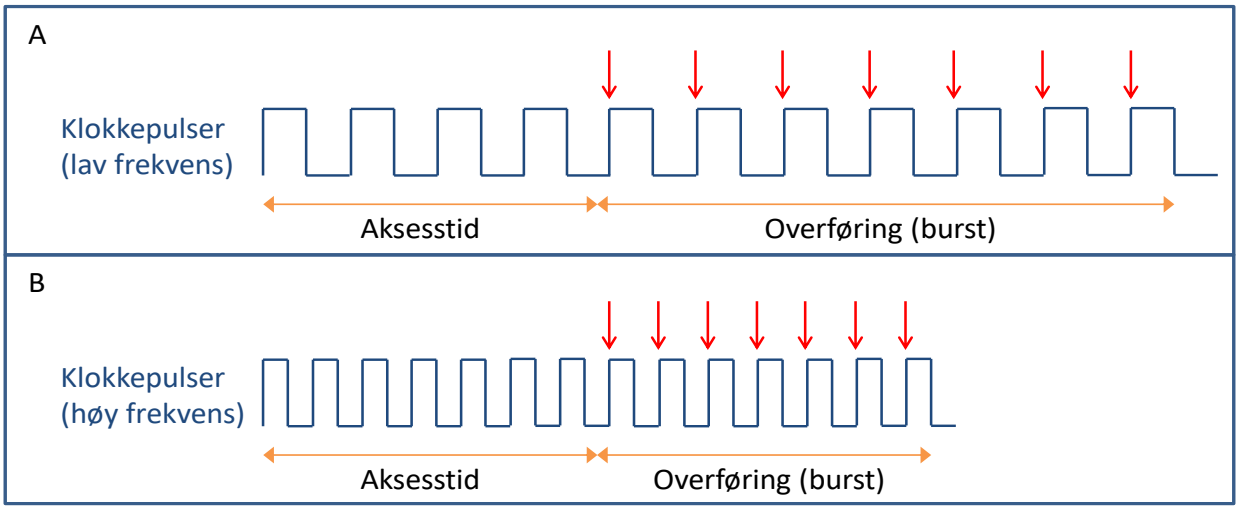
Fortløpende lesing

I burst-modus tar det altså litt tid å komme i gang, men når vi først har kommet i gang så skjer det en overføring i hver klokkesyklus. Når vi først er i gang så er det derfor klokkefrekvensen som bestemmer hvor lang tid vi trenger for å bli ferdig.

Legg merke til at nå har vi god nytte av en høy klokkefrekvens. Ved høy klokkefrekvens får vi overført blokka så fort som mulig.

Figur 2 viser dette også. Med høy klokkefrekvens er tidsforbruket for overføring (burst) kortere, fordi det går kortere tid mellom hver overføring.

Det er elektronikken i minnebrikken som bestemmer hvor høy frekvens vi kan bruke. Hver ny generasjon av minnebrikker tåler høyere frekvens enn tidligere generasjoner.



Figur 2 Tidsforbruk i burst-modus med lav (a) og høy (b) bussfrekvens. Det totale tidsforbruket er kortere ved høy frekvens. Tidsforbruket har to hovedbidrag: Aksesstiden og overføringstiden. Aksesstiden er uavhengig av bussfrekvens, og derfor like lang i begge tilfeller. Derimot er overføringstiden avhengig av bussfrekvensen og blir kortere ved høyere bussfrekvens.

1.1.5. Utviklingstrekk for moderne RAM-minne:

Det er nevnt nesten til det kjedsommelige i tidligere leksjoner: Prosessorens ytelse dobles omtrent annenhvert år i følge Moores lov. Dobling av ytelse betyr at at prosessoren utfører dobbelt så mange instruksjoner pr sekund. For at dette skal være mulig må cache, buss og primærminne være istand til å levere dobbelt så mange instruksjoner annenhvert år.

Et viktig poeng er at DRAM-teknologiene ikke utvikler seg like fort som prosessoren. Det tar produsentene ca ti år å utvikle en DRAM-teknologi der aksesstiden er halvert.

Så hva er det som gjør at moderne primærminne likevel greier å holde tritt med prosessorutviklingen? Jo, grunnen er at produsentene greier å utvikle styreelektronikken til primærminnet slik at vi kan bruke stadig høyere bussfrekvens. Hvis vi ser på Figur 2 så betyr dette at aksesstiden bare langsomt blir kortere, mens tiden som går med til overføring (burst) blir forbedret langt fortere.

I leksjonen om cache så vi på utviklingstrekk for cache: Cachen blir større og større, og antall cache-nivå øker. Når vi kombinerere dette med primærminne som tåler en stadig høgere bussfrekvens oppnår vi altså at cache, buss og primærminne sammen greier å levere instruksjoner og data tilstrekkelig fort.

1.2. Synkrone RAM-typer

For å få burst-modus til å bli mest mulig effektivt må man ha en høyest mulig klokkefrekvens. Da må samarbeidet mellom minneelektronikken og minnebussen være optimal. Moderne minneteknologier er av en type som kalles *synkrone* minneteknologier. I denne sammenhengen betyr *synkron* at minne-elektronikken er nøye synkronisert med klokken til bussen. Dette har den store fordelen at vi kan øke bussens klokkefrekvens fordi busselektronikken og minnebrikkens styreelektronikk er nøye avpasset hverandre.

Nå for tiden brukes det utelukkende synkrone RAM-typer på PC[1]. Hver enkelt minnecelle er fortsatt DRAM. En DRAM-brikke med synkron styreelektronikk kalles Synkron DRAM, eller bare SDRAM (*Syncronous Dynamic RAM*).

1.2.1. SDRAM

Den første synkrone RAM-typen som kom ble bare kalt SDRAM. Det er virkemåten til denne minneteknologien som er beskrevet i kap. 1.1.4 og vist i Figur 2.

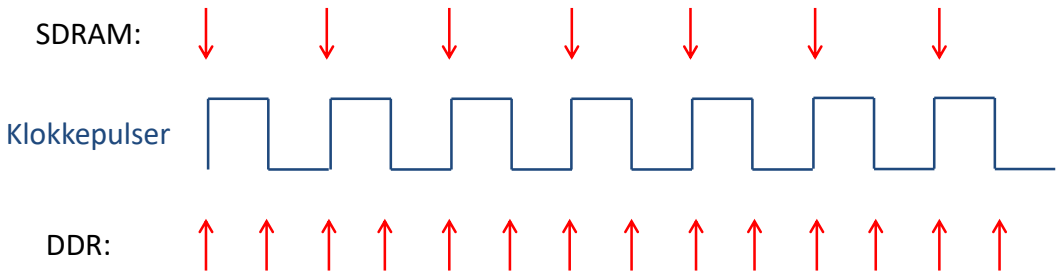
Lagringscellene er ordinær DRAM og har derfor den aksesstid som alle DRAM-teknologier. Når lesingen først har kommet i gang overføres det en ny lokasjon ved hver klokkepuls. SDRAM-brikkene som ble brukt på PC hadde en frekvens på opp til 133 MHz.

Slike minnebrikker er ikke lenger i vanlig bruk fordi det ble avløst av en nyere variant som kalles DDR-SDRAM.

1.2.2. DDR-SDRAM

En spesiell type SDRAM er DDR-SDRAM. DDR står for Double Data Rate, og betyr rett og slett at når data først begynner å komme, så bruker man både overgangen til høy og overgangen til lav puls til å overføre data. På den måten får man to overføringer i løpet av hver klokkeperiode.

Tidspunktene for dataoverføringer på vanlig SDRAM kontra DDR SDRAM er fremstilt i Figur 3.



Figur 3 ccSDRAM med og uten DDR. I burst-modus vil en tradisjonell SDRAM overføre data hver klokkepuls (øverste røde piler). Med DDR SDRAM overføres data både på stigende og fallende puls (nederste røde piler). Dermed skjer det to dataoverføringer i hver klokkepuls og vi oppnår at overføringstakten blir det dobbelte av klokkefrekvensen.

Det er viktig å legge merke til at aksesstiden fortsatt er den samme. Det tar like lang tid å komme i gang. Men når vi først har kommet i gang, så går det dobbelt så fort med DDR.

Det er også verdt å merke seg at tiden det tar å komme i gang oppgis i antall hele pulser.

Dette at antall overføringer pr sekund er det dobbelte av bussfrekvensen gir lett opphav til misforståelser av begrepet *bussfrekvens*. Når DDR-SDRAM blir oppgitt som f. eks. DDR-1600, så står 1600 ikke for 1600 MHz bussfrekvens, men for 1600 millioner overføringer pr

[1] På andre systemer brukes det asynkrone teknologier som er billigere og har mye lavere bussfrekvens. Bl.a. bruker mange innebygde systemer med små krav til ytelse asynkrone minneteknologier.

sekund (forkortet 1600 MT/s, *million transfers pr second*). På norsk omtales dette gjerne som *overføringstakt*. Overføringstakten er den dobbelte av bussfrekvensen.

1.2.3. DDR2, DDR3 og DDR4

Dette er nyere standarder som tillater høyere klokkefrekvens enn den opprinnelige DDR-SDRAM standarden. I dette kurset går vi ikke inn på forskjellene mellom dem, da alle har samme prinsippielle virkemåte.

Den hurtigste DDR2-typen kalles DDR2-800. Den tillater inntil 400 MHz klokkefrekvens (overføringstakten i burst-modus blir 800 MT/s).

DDR3 tillater inntil 800 MHz klokkefrekvens (DDR3-1600 har overføringstakt på 1600 MT/s i burst-modus).

DDR4 ble introdusert i 2014 og har spesifisert en bussfrekvens på inntil 1,6 GHz (og dermed en overføringstakt på 3200 MT/s (=3,2 GT/s, *gigatransfer pr second*).

1.2.4. Hvilken RAM-type kan jeg bruke?

Vi har sett på flere forskjellige RAM-typer. Alle disse typene kan imidlertid ikke brukes på alle PCer. PCen må være forberedt for den RAM-typen som skal brukes. Det er det såkalte brikke-settet som bestemmer hvilken RAM-type som kan brukes på en maskin. Brikke-sett gjennomgås senere i denne leksjonen.

1.2.5. Oppsummering av synkrone teknologier så langt

Vi har nettopp sett at de synkrone teknologiene tillater bruk av høgere klokkefrekvens, og at dette er en stor fordel når vi først har kommet i gang med fortløpende lesing. Spesielt gjelder dette for DDR-SDRAM der vi virkelig får sus over sakene når vi er i burst-modus.

Selv om det har blitt gjentatt menge gang så presiseres det enda en gang: Høy klokkefrekvens og DDR gir kjapp overføring når vi har kommet i gang med lesingen og overfører i burst-modus. Det hjelper oss i imidlertid ikke på aksesstiden – den er uavhengig av bussfrekvensen. I neste kapittel skal vi se om vi kan gjøre noe med dette.

1.3. Timing-parametrene

Til nå har vi slått oss til ro med at aksesstiden til DRAM er lang (35-70 ns). Vi har også vært fornøyd med at den **alltid** er like lang.

På moderne datasystemer må vi imidlertid bruke alle muligheter til optimalisering. Og hvis vi ser mer i detalj på virkemåten til en DRAM-brikke så vil vi oppdage at tiden det tar å finne startlokasjonen ikke nødvendigvis trenger å være like lang alltid. Enkelte ganger kan vi få den kortere. Det viser seg nemlig at tiden i praksis er avhengig hvor vi hentet data ved forrige aksess.

For å forstå dette må vi repetere oppbyggingen av RAM-brikkene.

1.3.1. DRAM med kvadratisk organisering

DRAM er bygget opp av kvadratiske RAM-brikker. Fra leksjonen om primærminne husker vi at cellene organisert som en matrise med like mange rekker og kolonner.

Basert på adressen som skal aksesseres vil èn dekoder plukke ut rett rad, og en annen dekoder velger rett kolonne.

La oss se litt mer i detalj hvordan en celle aksesseres:

Først presenteres adressen for rekke-dekoderen. Samtidig settes et kontrollsignal som kalles RAS (Row Access Strobe). Dette kontrollsignalet er et varsel om at rekke-adressen er gyldig.

Nå aktiviseres rett rekke. Det vil si at rekken klargjøres for aksess, noe som tar litt tid. Tiden det tar å aktivisere en linje kalles *RAS-to-CAS-delay* og angis i antall klokkesykluser. Denne forsinkelsen skrives ofte T_{RCD}.

Når linjen er aktivisert kan man begynne å aksessere hver enkelt celle i linja.

Nå presenteres adressen for kolonne-dekoderen, og et kontrollsignal som kalles CAS (Column Access Strobe) settes. CAS er et varsel om at kolonne-adressen er gyldig. Etter at CAS er satt, vil det være en ventetid før den rette cellen i aktiv rekke er tilgjengelig. Denne ventetiden kalles *CAS-latency* (CL), og angis i antall klokkesykluser. Latency betyr ventetid. Typisk størrelse for CL er en plass mellom 2 og 11 klokkesykluser.

1. Legg merke til at tiden det tar å lese en celle innenfor en allerede aktivisert rekke er CL.
2. Tiden det tar å lese en celle når ingen rekker er aktivisert er T_{RCD} + CL.

I tillegg til disse to ventetidene er det to andre forsinkelser som ofte oppgis:

T_{RP}: Dette er antall klokkesykluser som trengs dersom feil rekke er aktivisert. Da må man vente litt ekstra i påvente av oppfriskningen av DRAM-cellene (se *leksjon om Primærminne*) før man aktiviserer den rette rekka.

3. Ventetiden hvis feil rekke er aktivisert blir: T_{RP} + T_{RCD} + CL.

Og den siste forsinkelsen vi skal se på:

T_{RAS}: Dette er antall klokkesykluser fra en bank aktiviseres og til man kan sette RAS.

4. Total ventetid hvis feil bank er aktivisert blir: T_{RAS} + T_{RP} + T_{RCD} + CL.

1.3.2. Hva betyr dette?

Det vi har sett nå er at tiden det tar å fremskaffe en lokasjon er avhengig av hva som har skjedd før. Vi har fire muligheter:

1. Hvis aksess skjer i samme rekke som forrige aksess blir ventetiden:
   CL (CAS-latency).
2. Hvis ingen rekker er aktive blir ventetiden:
   T_{RCD} + CL (aktiviser rett rekke, deretter rett kolonne).
3. Hvis feil rekke er aktivert blir ventetiden:
   T_{RP} + T_{RCD} + CL
4. Hvis hele banken må aktiviseres først blir ventetiden:
   T_{RAS} + T_{RP} + T_{RCD} + CL

Moderne RAM-teknologi utnytter dette, og ventetiden for første aksess av en blokk er derfor avhengig av hvor forrige aksess var.

I **verste** fall er tiden lik T_{RAS} + T_{RP} + T_{RCD} + CL. I øvingen skal vi se at denne summen blir mellom 35 og 70 ns, og tilsvarer det vi kaller aksesstid fordi det er tiden som trengs for å aksessere en vilkårlig aksess.

Men i **beste** fall er tiden bare CL. Og CL er bare noen få klokkesykluser. Hva betyr dette? Jo det betyr av ved stor grad av (romlig) lokalitet vil tiden det tar å fremskaffe data være mye kortere enn 40-70 ns.

Har vi løyet før? Nei. Aksesstiden er definert som tiden det tar å finne en **vilkårlig** lokasjon – altså en hvilken som helst lokasjon – og da må man angi det verste tilfellet.

1.3.3. Virkning ved bruk av cache

Vi må huske på at hele vår diskusjon startet ved at vi bruker cache,. Når vi leser fra minnet skal vi så fort som mulig overføre en blokk fra minnet til cache.

SDRAM og DDR SDRAM er spesial-laget til dette. Når rett rekke på rett bank er aktivisert, kan man sette minnekretsen i burst-mode. Da vil kretsen starte på adressert kolonne og begynne å spy ut innholdet av fortløpende celler så kjapt den greier. På SDRAM kommer det en ny verdi hver klokkepuls, og på DDR-teknologiene kommer de hver halve klokkepuls. Slik forsetter det til man har lest ønsket antall celler.

Konklusjon:

Det tar tid å klargjøre minnebrikken – hvor lang tid er avhengig av hvor forrige aksess var – men når vi har gjort dette får vi overført en hel blokk svært fort.

1.3.4. Hva oppgis i BIOS-oppsett og datablader

Ofte oppgis CL, T_{RCD}, T_{RP}, og T_{RAS}. Gjerne på følgende form: 5-5-5-15. Det er imidlertid ikke uvanlig at T_{RAS} utelates. I reklamen oppgis ofte bare CL.

Enkelte ganger ser man at CL kalles *aksesstid*. Dette er imidlertid aksesstiden kun hvis rett rekke allerede er aktivisert. De fleste lærebøker definerer aksesstid som tiden det tar å aksessere en helt vilkårlig lokasjon. Med denne definisjonen blir CL noe annet enn aksesstid.

Enkelte ganger ser man en femte parameter som kalles Command rate. Dette er klokke-frekvensen til bussen oppgitt i MHz.

1.3.5. De viktigste synkron-RAM-typene

RAM-teknologien utvikler seg hurtig, og stadig nye kommer til. Disse er de viktigste så langt:

PC100 og PC133.
Dette er SDRAM. Tallet angir klokkefrekvensen i MHz.

DDR333 og DDR400.
Dette er DDR og tallet angir overføringstakt i burstmode (altså det dobbelte av klokke-frekvensen).

DDR2-667 og DDR2-800.
Dette er DDR2 og tallet bak bindestreket angir overføringstakten i burstmode (altså det dobbelte av klokkefrekvensen).

DDR3-1066, DDR3-1333, DDR3-1375 og DDR3-1600
Dette er DDR3 og tallet bak bindestreket angir overføringstakten i burstmode (altså det dobbelte av klokkefrekvensen).

1.3.6. SPD

Nyere DIMM-kort har en innebygget ROM som inneholder informasjon som gjør at BIOS på PC kan lese anbefalte verdier for timing-parametrene. Disse verdiene brukes ved automatisk konfigurasjon av minnet.

Mange BIOS'er tillater brukerne å overstyre disse parametrene. Det kalles overklokking av minnet. Hvis man eksperimenterer med parametrene kan man få en viss ytelsesforbedring. Man bør vite hva man gjør, ellers kan maskinen bli ustabil eller helt slutte å fungere.