Week 6 memory and bus

Random Access Memory (RAM)
Sekvensielt minne betyr at vi må gå gjennom alt data for å komme til et spesifikt del av minne før vi kan lese noe spesifikt.
RAM vil si at det tar like lang til å få tak i hvilken som helst minne som er i RAMen

RAM: statisk (SRAM)

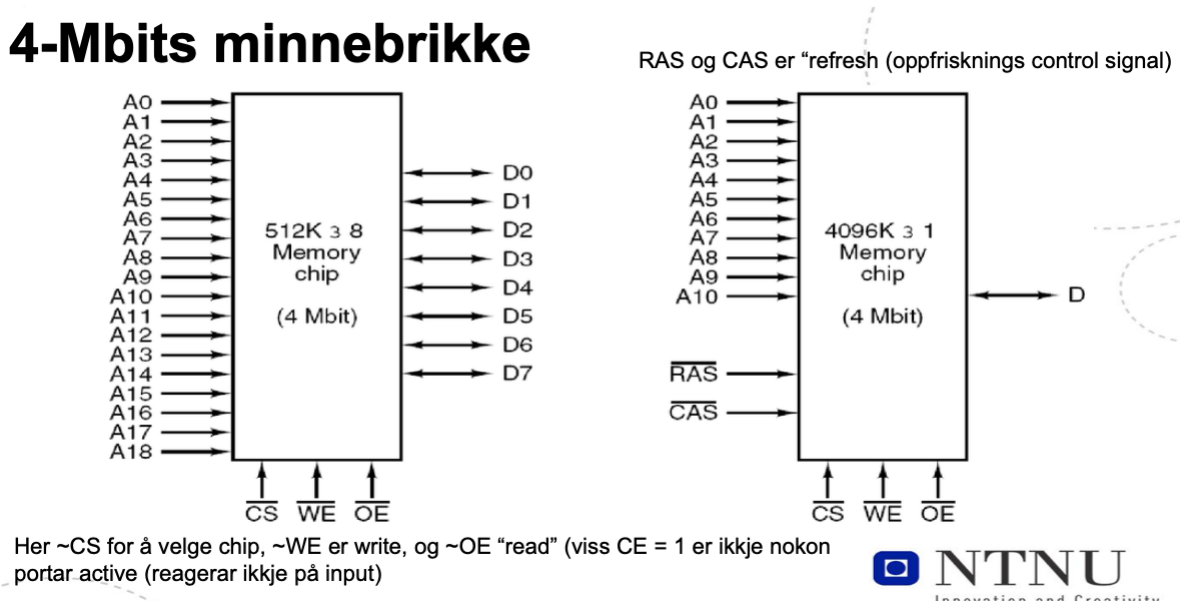
- Statisk
  - rask
  - stor minnecelle (areal)
  - stort effektforbruk
  - må ikke oppfriskes
  - enkelt grensesnitt

RAM: dynamisk (DRAM)

- Dynamisk
  - ikke så rask
  - liten minnecelle (areal)
  - lite effektforbruk
  - må ha oppfriskning
  - mer komplisert grensesnitt (DRAM-kontroller)

4-Mbits minnebrikke
Brikker hvor vi kan adressere minnelokasjoner og vi kan skrive ut eller lese data.

4-Mbits minnebrikke



Den til venstre er statisk

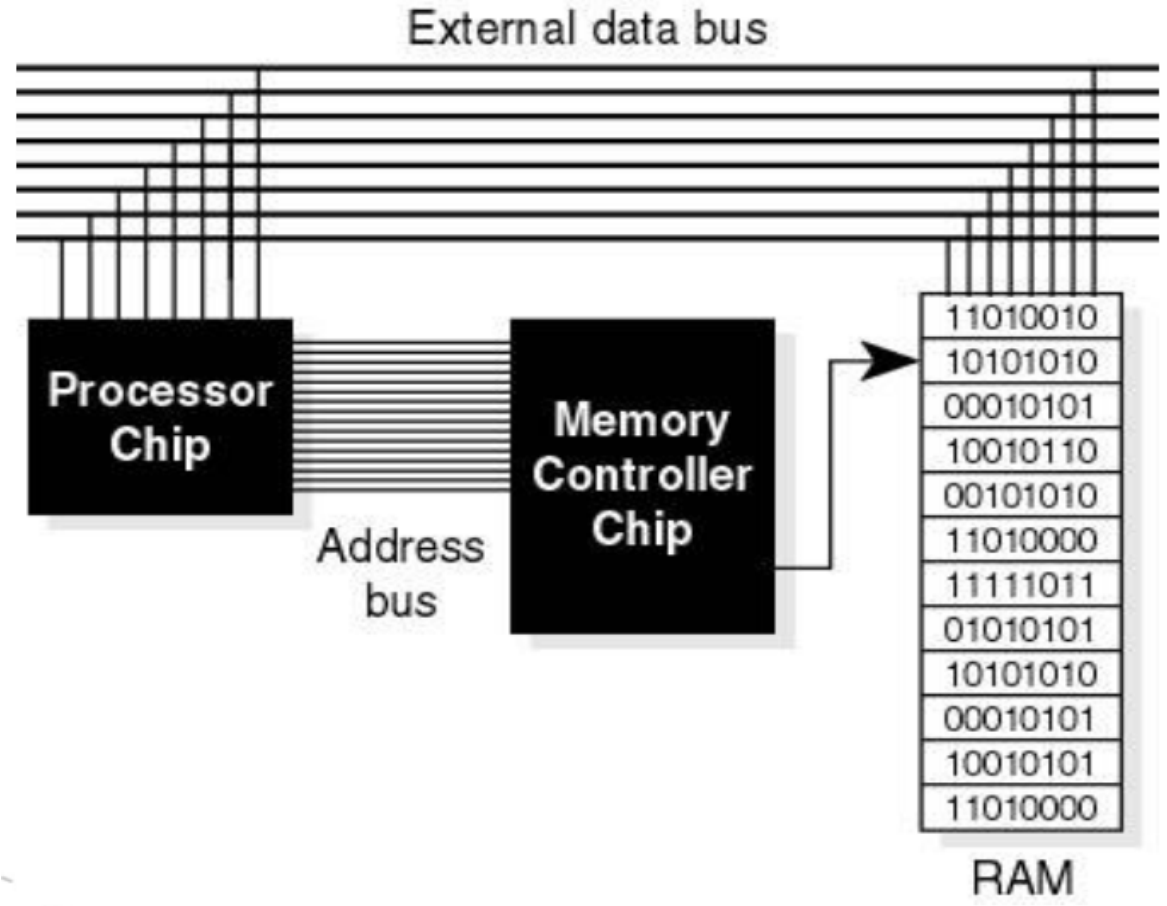
Den til høyre er dynamisk og har noen ekstra porter for blant annet oppfriskning.

Andre minnetyper: nonVolatile

- ROM (read only memory)
  - Lagre program eller data som aldri skal endres
  - Cella i seg selv bruker strøm ekstremt ilte effektforbruk
  - Stort sett kun for masseproduksjon (ved mange billigst)
- PROM (programmable read only memory)
  - Som ROM, men kan programmeres minst en gang
  - Inneholde kan defineres etter produksjon
  - Mange typer:
    - PROM: programmeres en gang
    - EPROM: (erasable) kan slettes
    - EEPROM: (electrically) kan slettes elektrisk
    - ofte lang programmerings tid
    - Aktivkomponent (transistorer) bruker energi (mer enn ROM men ganske lite)
- Flash memory
  - "EEPROM med rask programmering"
  - Er forskjellige transistorer som blir brukt her enn de over
  - Billig
  - Programmeres i blokker
  - Kan endres ca 1 000 000 ganger (blir bedre og bedre)

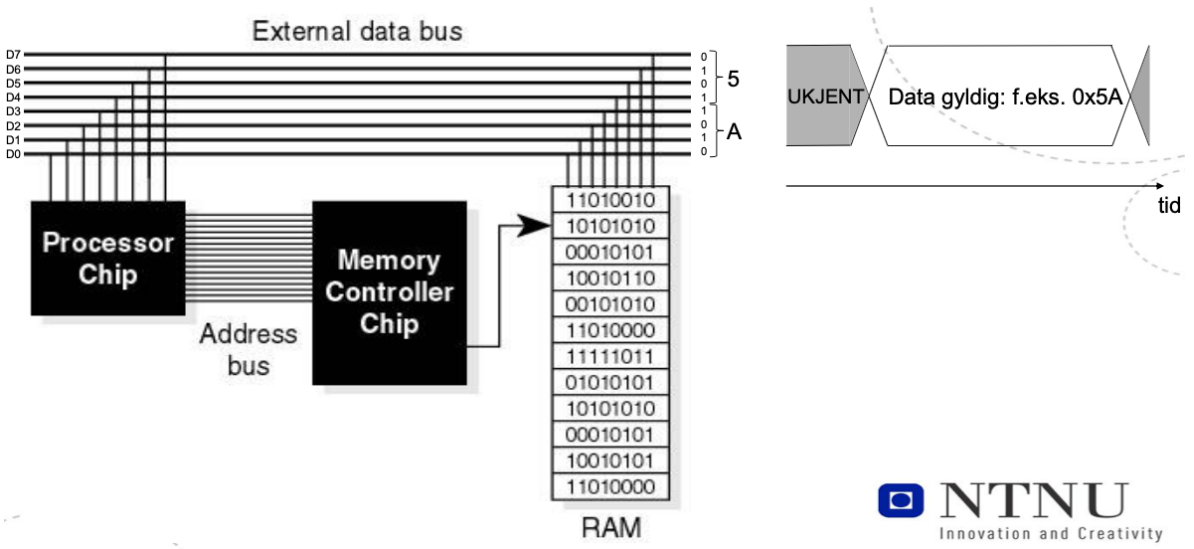
Buss

Vi bruker buss til å overføre en del av data. F.eks fra et register til ALU. Det er en samling av fysiske ledninger

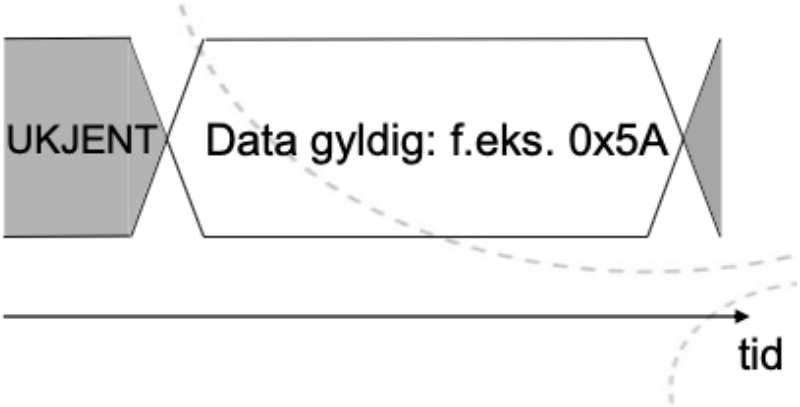


Vi har en ekstern buss og en adresse buss. Poenget med dette her er å vise at det er fysiske ledninger i bussen.

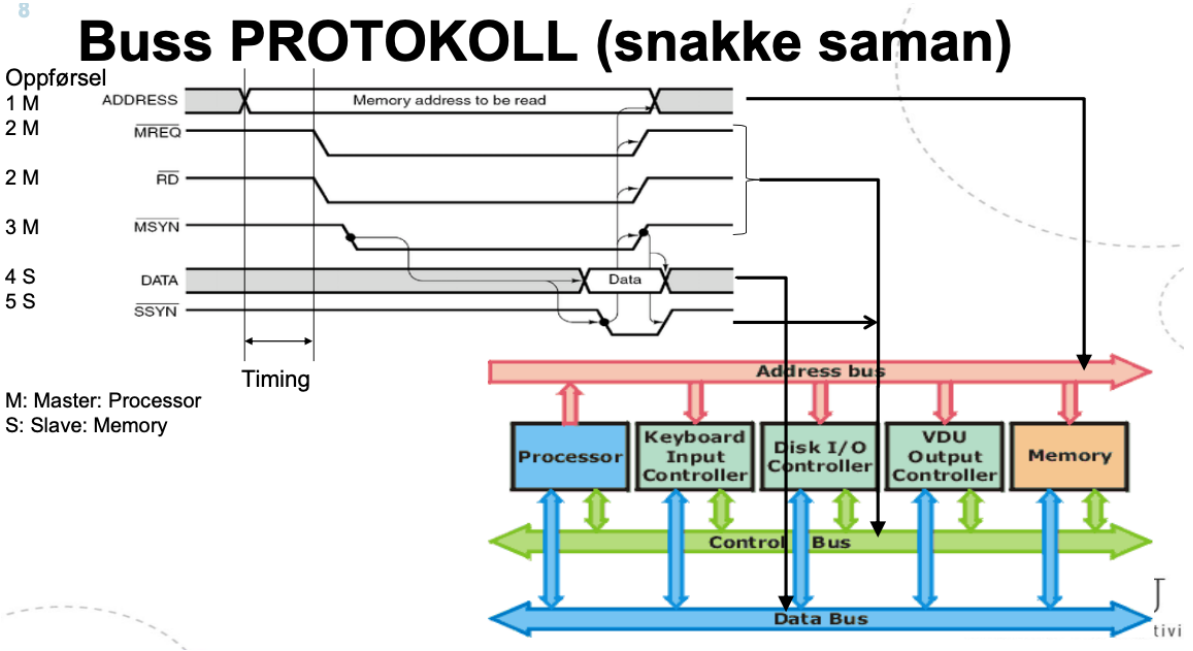
Databussen er 8 linjer. Vi har D0 til D7. Vi kan se på et eksempel:



Her har vi altså noen bit som er i bussen. Vi oversetter dette til hexadesimale-tall 5A.



For å få oversikt over hva som er i bussen over en tidsperiode så kan vi skrive et diagram som over. Inne i den hvite boksen så kan vi skrive data eller adressen som er i bussen.
Vi pleier å ha forskjellige buss-typer:



Vi ser nederst at vi har en:

- Adressebuss
  - Overfører adresser
- Kontrollbuss
  - Overfører kotrollsignaler
- Databuss
  - Overfører selve dataen

Øverst har vi flere busser. Vi har en adresse- og data buss. Vi ser når ting blir lagt til i dem. Alle kontrollsignalene (MREQ, RD, MSYN) går inn i kontrollbussen.
Vi har en bussprotokoll er diagrammet øverst. Det definerer hvordan bussen virker og oppførselen på hvordan signalene og timingen blir lagt frem.

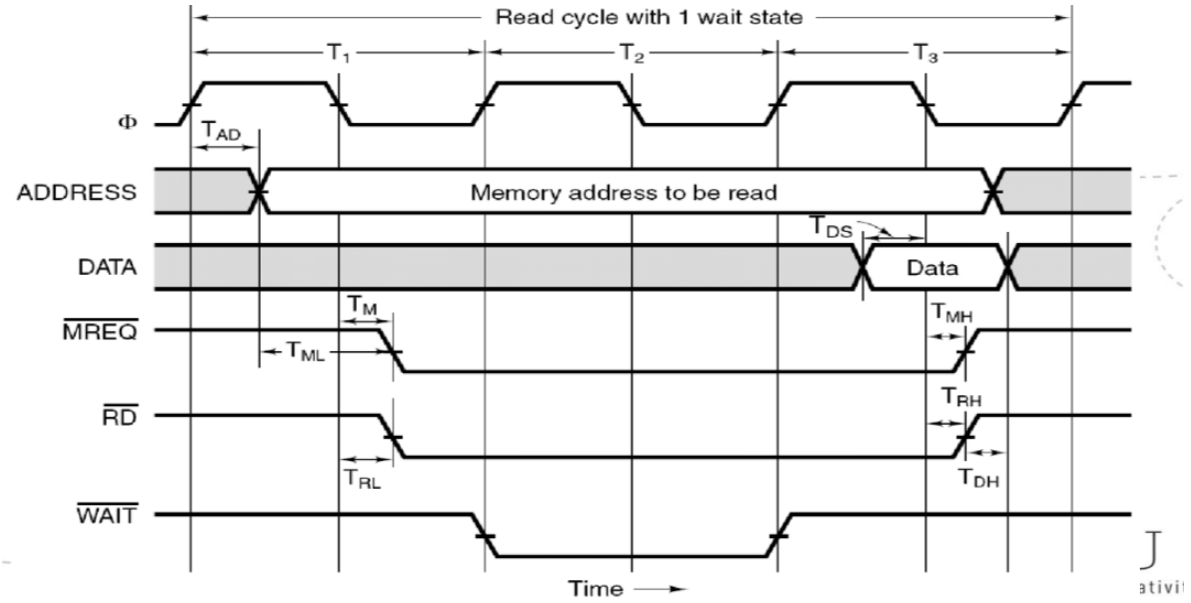
Vi har to forskjellige oppførsler: master og slave. Her er prosessoren master og memory slave, f.eks.

Det første som skjer er at vi legger ut en gyldig minneadresse. Deretter legger vi et et signal som heter master request (MREQ) som vil si at prosessoren vil ha tak i noe. Samtidig bestemmes det om det er en les eller skriv operasjon (RD). Her vil vi gjøre en les operasjon. Etter dette settes det opp et synkroniseringssignal (MSYN). Når busskontrolleren i prosessoren har gjort alt dette her så venter den. Deretter skjønner minnet operasjonen og legger ut dataen i bussen. For å være sikre på at signalet har stabilisert seg så legger minnet ut et slave sync (SSYN) signal. Etter at prosessoren har fått dataen sin så signaliserer MSYN at dette er gjort.

Busstyper

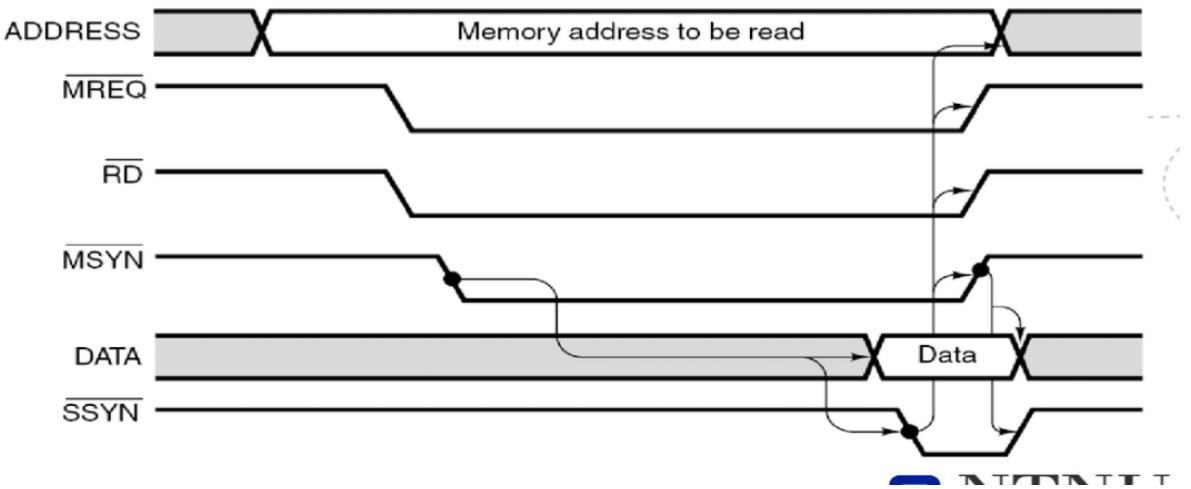
- Synkron
- Asynkron
- Multiplekser (kan være synkron eller asynkron)

Synkron klokke



Vi har en fi klokkepuls øverst. Bussen er aktiv når klokken går fra 0->1 eller fra 1->0. Det går litt tid fra klokken blir aktiv til vi får adressen på bussen. Deretter kan vi legge til et MREQ og RD signal. Det som er spesielt her er at vi har et WAIT signal. Vi har dette her fordi minnet ikke er raskt nok. Dette legger til klokkesykluser for å gi minnet tid til å legge adessen på bussen.
T-verdiene i diagrammet definerer tiden det tar å få gjort enkelte operasjoner.

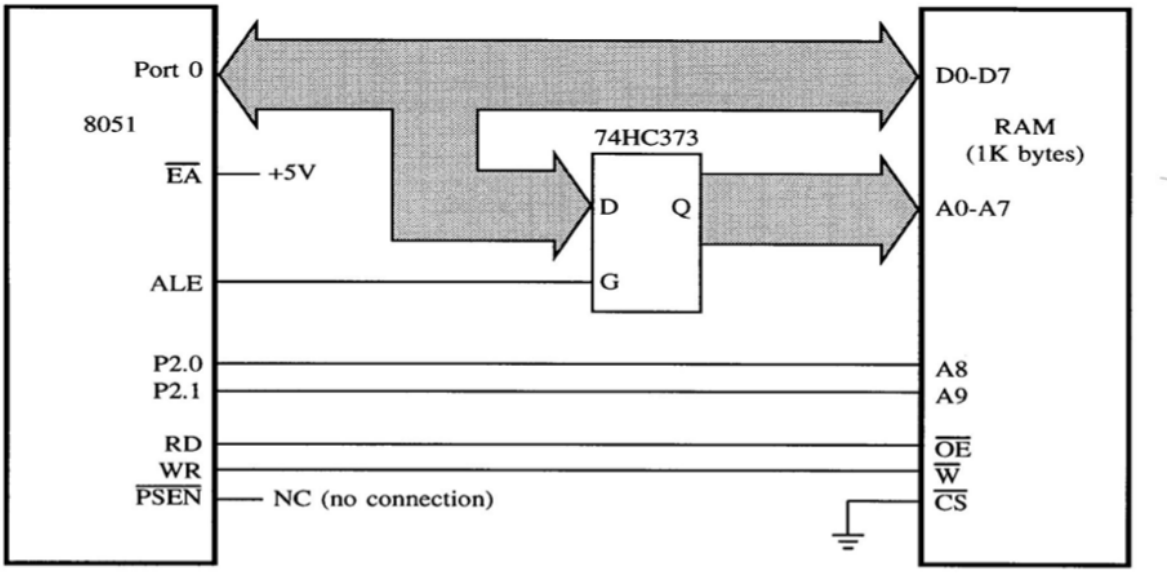
Asynkron "handshaking"



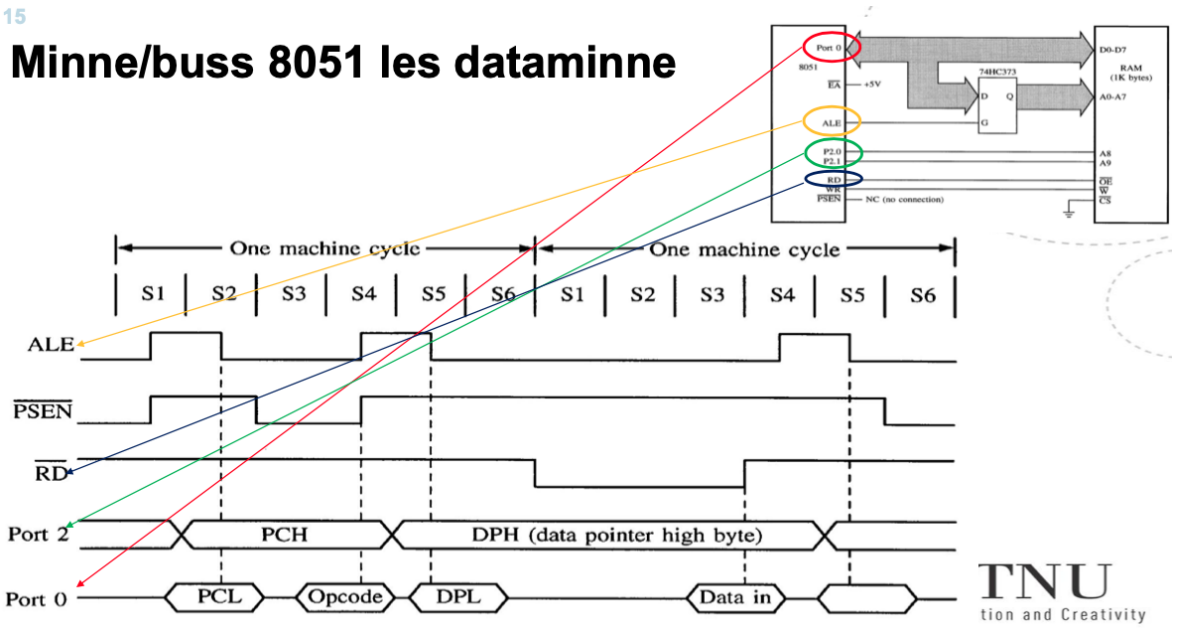
Her har vi ikke klokkesignal. Den store forskjellen er at vi bruker "handshaking". Vi sender et signal fra master MSYN til enheten vi ønsker å få tak i. Masteren står og venter når dette er sendt. SSYN vil sende et signal tilbake og si ifra når dataen er klar. Da kan master lese denne dataen.

Synkron buss bruker altså klokkeperiode for å presist definere når alt skal skje. Dette bruker vi ikke i en asynkron klokke.

Multiplekserbuss



Her prøver vi å bruke mindre input/output i prosessoren. Her har vi laget en latch. Vi kan sende et signal (ALE) som sier at vi skal sende alle 8 bitene D til Q.
Først legger prosessoren ut adressen på port 0 som går til D i latchen. Deretter aktiverer vi ALE og vi får de 8 nederst bitene i adressen i RAM-en.
Vi bruker en 2-bit buss. De neste bitene P2.0 og P2.1 ligger under ALE som blir overført til RAM. Da har vi altså overført 10 bit til RAM-en.
Vi avslutter signalet ALE. Vi oppnår med dette at vi kan sende data direkte fra port 0 til D0-D7.
Med dette så deler vi adresse og data på en buss.



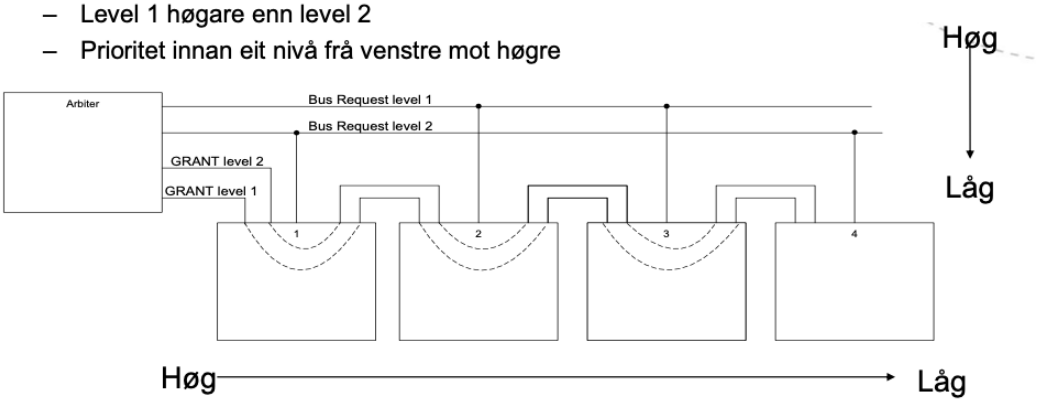
Her vises det et lesesyklus for denne protokollen.

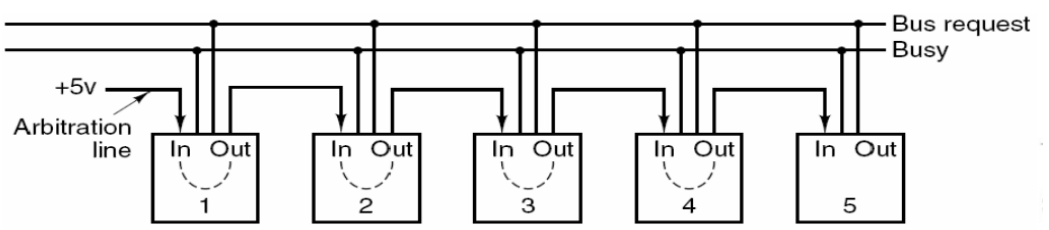
La oss si at vi starer fra S5. Det første som skjer her er at port 0 legger ut adressen sin. Samtidig legger port 2 ut sine to bits. Vi setter i gang ALE for å flytte adressen til RAMen. Etter at vi har lukket ALE så brukes det litt tid å få overført dataen ferdig. Vi ser at dette er ferdig i port 0 på DPL. På det tidspunktet er port 0 ledig. Vi legger ut et RD signal for å signalisere dette. Vi ser at dataen begynner å gå inn i port 0 fra S3.

Arbitrering, hvem får bruke bussen

Vi må ha en standard for hvem som skal få lov til å bruke bussen.

- Sentralisert arbitrering: detaljer
  - Fleire nivå
    - To prioritetsnivå level 1 og level 2
    - Level 1 høgare enn level 2
    - Prioritet innan eit nivå frå venstre mot høgre



    - I sentralisering har vi en utvelger (arbiter). Arbitreringen er gitt av logikk i prosessoren. Prioriteringen er gitt av hvor nær du er utvelgeren og hvilket nivå du er prioritert i. Eksempel over har to prioriteringer.
    - Dersom enhet 2 ønsker å ta bussen i bruk så sender den et request signal. Når den gjør det så vil arbiter sende et GRANT signal på samme nivå gjennom kjeden.
    - Dersom enhet 2 og 3 ønsker å bruke bussen samtidig. Arbiter sender GRANT gjennom kjeden og kommer til enhet 2. Siden enhet 2 ønsker å bruke bussen så sender den ikke signalet videre og begynner bare å bruke bussen. Dermed har enhet 2 prioritering over enhet 3. Når enhet 2 er ferdig med å bruke bussen så holder enhet 3 fremdeles signalet i REQUEST. Når enhet 2 er ferdig så kommer GRANT til enhet 3.
    - Dersom enhet 1 og 2 ønsker å bruke bussen samtidig så får arbiter REQUEST på level 1 og 2. Da vil arbiter sende en GRANT kun på nivå 1 først. Når enhet 2 er ferdig og det er ingen flere requests på nivå 1 så vil arbiter sende GRANT til level 2.
- Desentralisert arbitrering
    - 

    - Her trenger vi ikke logikk for når noen skal bruke bussen. I dette systemet vil den som står nærmest venstre få bruke bussen først. Når bussen er opptatt vil det bli lagt ut et BUSY signal til resten.
    - Desto nærmere prosessoren du er vil du altså få en større prioritet