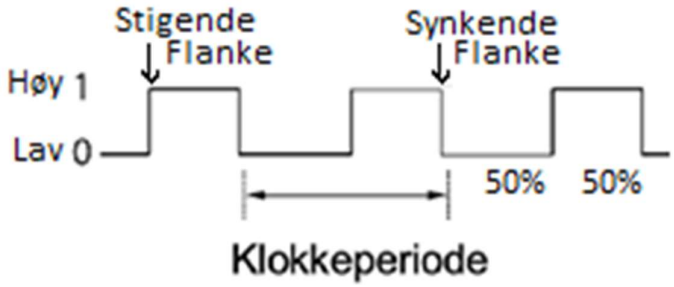
Klokke og Synkronisering

Klokkesignal

- Klokkesignal er et digitalt signal som går fra 0 til 1 i en fast takt.
    - Vi betegner 1 som at klokken er høy og 0 som at klokken er lav.
    - Klokkesignalet har jevn takt. Ofte 50% 50%.
- CPUen trenger klokkesignaler for å vite når og hvordan den skal kjøre de programmerte instruksjonene.
- I en klokkesykel er de vanligste instruksjonene hente fra minnet, lagre data i minnet, hoppe til angitt adresse, hente en instruksjon. (fetch, load, execute)
- Når klokken er høy utfører vi instruksjoner, når klokken er lav er det pause(med unntak om at vi kan lagre data).
- Klokkesignaler gjør synkronisering mulig, at vi kan kjøre instruksjoner i parallell (pipeline).



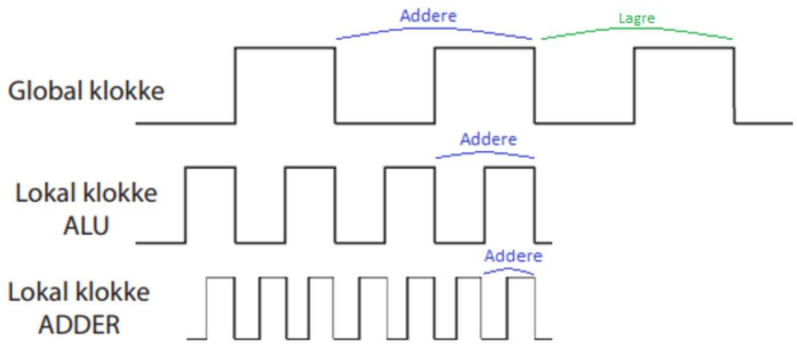
Klokke/ synkronisering

- Tenk på hvordan vi bruker klokker i dag!:
    - Si at jeg og en venninne skal møte hverandre kl.14. Om vi har synkrone klokker, så møtes vi klokken 14, men om en av oss ikke har det (klokkene våre er asynkrone) så møtes vi ikke til riktig tid.
    - Om klokkene våre er asynkrone har vi ikke noe forhold til tid, eller når ting skal gjøres/ skje.
    - Det samme gjelder i prosessoren.
- En prosessor uten en klokke kalles asynkron. Slik som LMC. Den kan ikke gjøre instruksjoner i pipeline(parallell), siden da kan de krasje.

    - (krasje: for eksempel om vi prøver å hente (fetche) data fra minnet et sted før vi har lagra denne dataen i minnet).
- En prosessor med en klokke kalles synkron. Den tillater oss å kjøre instruksjoner i pipeline (i parallell), uten at instruksjoner krasjer.
    - Vi har klokke for å kunne synkronisere instruksjoner.

Synkronisering

- Alle elementene i en prosessor styres av en global klokke.
- (Elementene har hver sin lokale klokke, som synkroniseres til den globale klokken.)
    - (Slik: her er viser vi bare at klokken til ALU og ADDER er synkronisert med den globale klokken, som skal utføre instruksjonen addere
        - Så lang tid det tar å addere er dobbelt så lang tiden (klokkeperioden) det tar ALUen å addere i en ADDER og produsere et svar.
        - Tiden ALUen bruker er igjen dobbelt så lang som tiden Adder bruker på å addere.
        - Bygger på regelen om at klokkeperioden må være like lang som den lengste forsinkelsen(tiden) for enhver instruksjon gjennom data-pathen. Om ADDER bruker 10 ps, må klokkeperioden til ALU dobbelt av det osv…



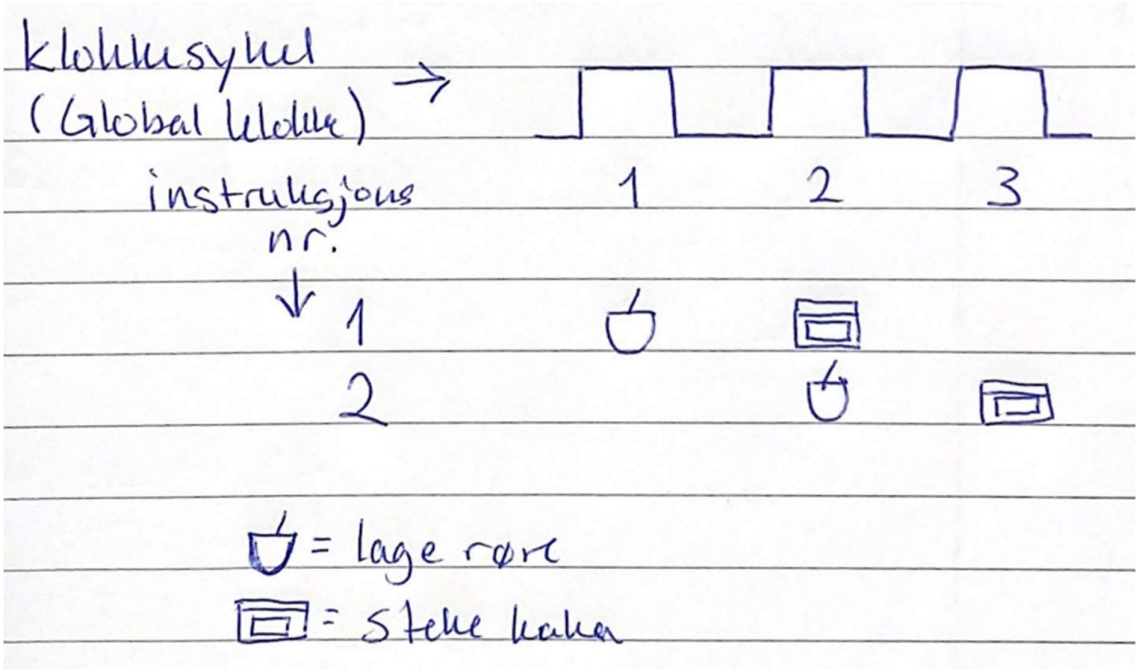
- Det er mulig å synkronisere klokkene på grunn av klokkesignaler.
- Når klokken er høy, utfører vi instruksjoner, når klokken er lav er det pause (eller vi gjør oss klar til neste instruksjon)

- Når den globale klokken er lav i klokkesignalet. Er vi sikre på at alle de lokale klokkene i parallelle instruksjoner(pipeline) er synkronisert før de neste instruksjonene (når klokken blir høy igjen).
- Dette betyr at **synkronisering er** at hver instruksjon kjøres akkurat når den skal og er gjort helt ferdig før neste tidsavhengige instruksjon kjøres, slik at instruksjonene ikke krasjer med hverandre.

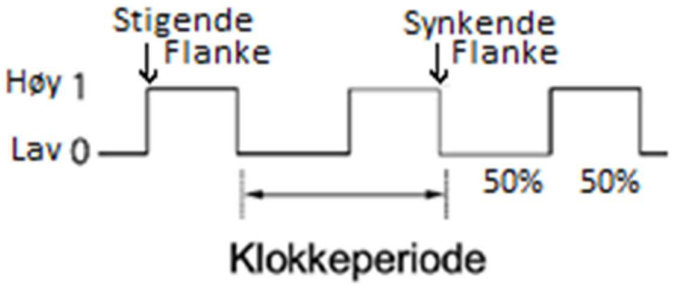
Eksempel på synkronisering

Forklaring:

- Prosessoren har fått to instruksjoner om å lage to kaker.
- For å lage en kake har vi to delinstruksjoner som er å lage røra og steke kaka.
- Vi har bare en bolle å lage røre i og bare en ovn å steke kake i. Så dette betyr at vi kan ikke lage to rører eller steke to kaker samtidig.
- I tillegg er delinstruksjonen 'steke kaka', er tidsavhengig av 'lage røra', siden vi må være ferdig med å lage røra før vi kan steke den.
- Det er her klokka kommer inn. Den kan fortelle oss om en instruksjon er ferdig.
- (De forskjellige delinstruksjonene har være sin klokke som er synkronisert opp til den globale klokken.)
- Så vi starter med å lage røre 1 i den første klokkesykelen.
- Så steker vi røre 1, og mens røre 1 steker er bollen ledig, så da kan vi lage røre 2. (siden de ikke er tidsavhengige av hverandre kan vi gjøre de parallellt)
- Nå er kake 1 ferdig å da steker vi røre 2
- På grunn av klokken her så vet prosessoren akkurat når hver instruksjon skal kjøres, slik at vi ikke starter å for eksempel steke kake 2 før vi er ferdig å lage røre 2.

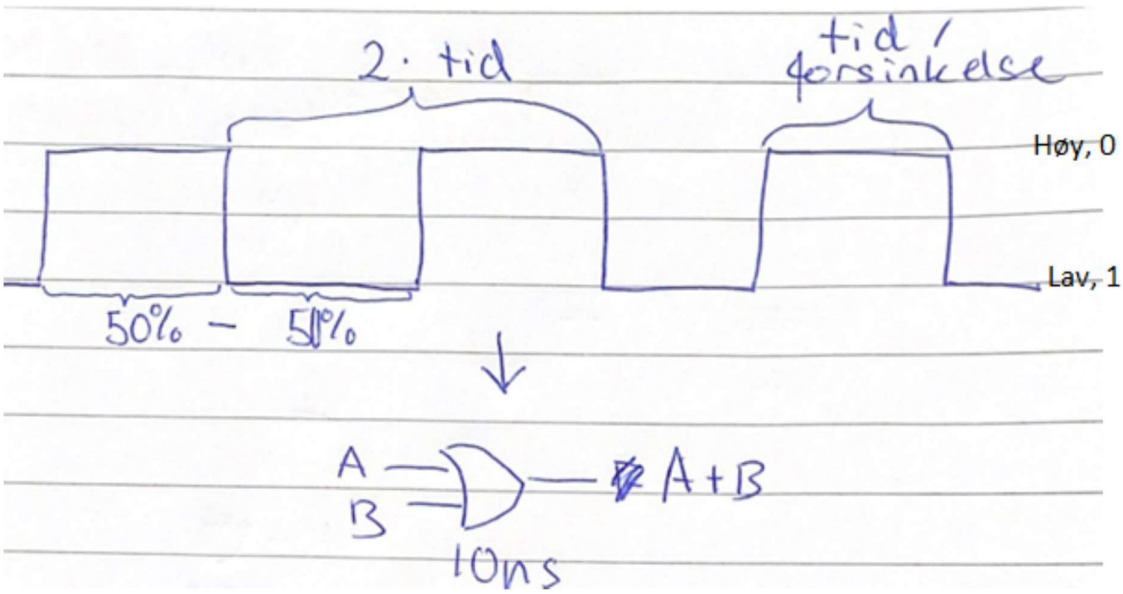


Klokkeperiode/klokkesykel og klokkefrekvens



- En klokkeperiode måles fra stigende flanke til stigende flanke. Eller fra fallende flanke til fallende flanke.
- Klokkeperioden = klokkesykel.
- Om vi kjenner klokkeperioden kan vi regne ut frekvensen til klokkesignalet.
- Klokkefrekvensen er antall sykler(fetch, load, execute) CPUen klarer å utføre per sekund
- Formelen for frekvens er: frekvens = 1/klokkeperioden.

- Klokkefrekvensen har alltid et jevnt klokkesignal. Ofte er denne 50%-50% klokkesignal. Dette betyr at klokken er høy like lenge som den er lav.
- I mange oppgaver snakker vi om forsinkelse/portforsinkelse for å utføre instruksjoner. Instruksjoner utføres når klokken er høy. Så om vi har en forsinkelse på 10ps, betyr det at det tar 10 ps å utføre den instruksjonen.
- Man kan finne ut av klokkeperioden fra hvor lang tid det tar å utføre en instruksjon og fra klokkesignalet. Om klokkesignalet er 50%-50% og det tar 10 ps og utføre en instruksjon (da er klokken er høy), vil også klokken være lav i 10ps. Da blir klokkeperioden 10ps * 2 = 20ps. Det tar 20 ps for å utføre en instruksjon.



Gruppetimen forklaring:

- Så hvordan får vi vite om klokkeperioden i en oppgave?
- Ofte i oppgaver så betegner vi tiden det tar å utføre en instruksjon som en forsinkelse /portforsinkelse.
- Se på tegningen, si at vi skal addere to tall.
- Tiden det tar for OR-porten vår å produsere et svar er 10 ps.
- Instruksjoner kjører bare når klokken er høy, så pga 50%-50% klokkesignal, så vet vi at halvparten av klokkeperioden er 10 ps.
- Dermed er klokkeperioden her 10ps * 2 = 20ps.

Oppgaver klokkefrekvens

Prefix tabell:

- Trenger denne til utregningene:

Symbol	Prefix	Multiplication Factor	
T	tera	10^{12}	1,000,000,000,000
G	giga	10^{9}	1,000,000,000
M	mega	10^{6}	1,000,000
k	kilo	10^{3}	1,000
h	hecto	10^{2}	100
da	deka	10^{1}	10
d	deci	10^{-1}	0.1
c	centi	10^{-2}	0.01
m	milli	10^{-3}	0.001
μ	micro	10^{-6}	0.000,001
n	nano	10^{-9}	0.000,000,001
p	pico	10^{-12}	0.000,000,000,001

Oppgave 1 Ukesoppgave 27

- Om frekvens ved forsinkelse(portforsinkelse)

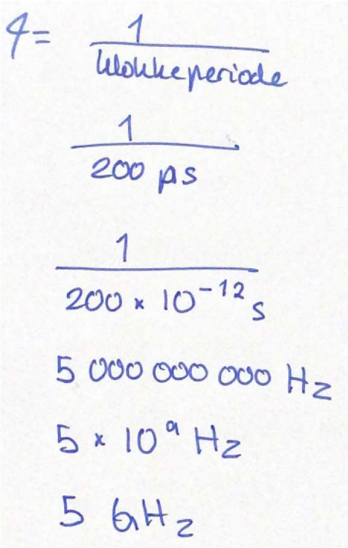
Oppgave: 27) Gitt en forsinkelse på 100ps for å lese ut data fra et register, hva er den høyeste frekvensen vi kan bruke for utlesing av et register?

Svar:

- For å lese fra et register må vi registeret først skrives til og så kan vi lese det.
- Det er to måter å tenke dette på her:
    - En med synkronisering, der det først skrives til registeret og så leser vi. Siden vi er avhengig av at noe er i registeret før vi leser fra det.
    - Eller en uten synkronisering mellom disse to instruksjonene, der vi både skriver og leser i samme klokkesykel.

1. Om vi skal skrive til registeret og så lese fra registeret (ved synkronisering):
    - Vi trenger minimum 100ps i når klokken er høy for å gjøre denne instruksjonen. (siden forsinkelsen er 100ps)

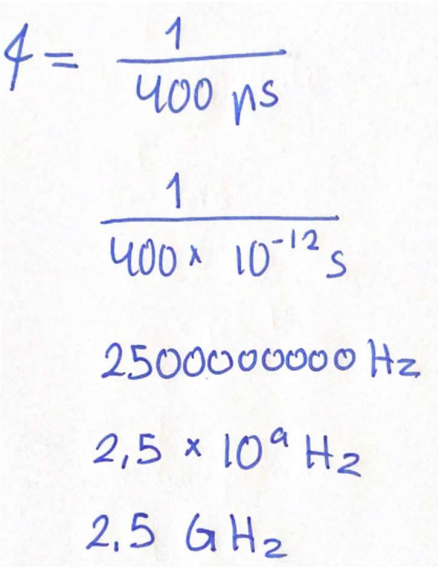
- Da er klokkeperioden $100ps * 2 = 200ps$ om vi antar 50%-50% klokkesignal.
- Formel for $frekvens = \frac{1}{Klokkeperiode}$
- Utregning:



- Den høyeste frekvensen vi kan bruke for utlesing av et register = 5 GHz

2. Om vi skal skrive til og lese fra registeret i samme klokkesykel. (uten synkronisering mellom disse to instuksjonene)

- Da må vi gange klokkeperioden med 2 (pga. vi trenger skal både lese og skrive mens klokken er høy): $200ps * 2 = 400ps$
- Utregning:



- Den høyeste frekvensen vi kan bruke for utlesing av et register = 2,5 GHz