

UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

IN2060 – Forelesning 3

Sekvensiell logikk

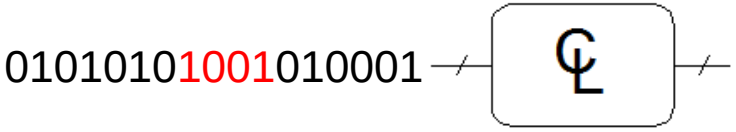
Tema forelesning 3

Sekvensiell logikk

- Definisjoner
- Minnelementer
  - Bistabile kretser
  - SR-latch
  - D-latch
  - D-flipflop (synkron logikk)
- Synkron logikk basert på D-flipflop
- Tilstandsmaskiner
  - Tilstandsdiagrammer og tilstandstabeller.
- Litt fysiske/elektriske detaljer.

Hvorfor sekvensiell logikk?

- Kombinatorisk logikk: output bestemmes kun av input **akkurat nå**
- Men mange systemer må huske hva som har skjedd tidligere:
  - Hvordan lage en teller?
    - 0000 – 0001 – 0010 … ?
  - Hvordan lage en sekvensdetektor?
    - Et system som kan detektere en spesifikk bit rekkefølge?



- Kombinatorisk logikk kan ikke løse dette
  - Kan ikke holde rede på foregående verdier.

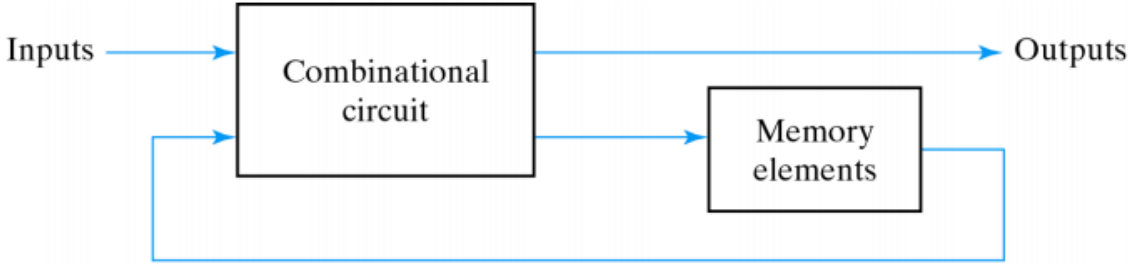
Motivasjon forts.

- Vi trenger et system som kan holde rede på tidligere inngangsverdier.
  - *Sekvensiell logikk*
- Vi skal se at sekvensiell logikk er basis for *tilstandsmaskiner*, som igjen er selve fundamentet for digital teknologi.
- *Tilstand*
  - All informasjon som kretsen/systemet trenger for å bestemme seinere oppførsel.

Sekvensiell logikk

Generell definisjon:

– Utgangen er et resultat av nåværende og tidligere innganger



Merk tilbakekoblingen

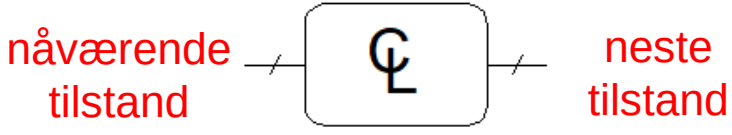
Sekvensiell logikk

- Sekvensiell logikk må ha minne, for å
  - Eksplisitt huske tidligere inngangsverdier.
  - Sammenfatte tidligere innganger for å utføre noen spesifikke arbeidsoppgaver.

- Asynkron sekvensiell logikk
  - Vanskelig å analysere og designe
- Begrenser oss til synkron sekvensiell logikk

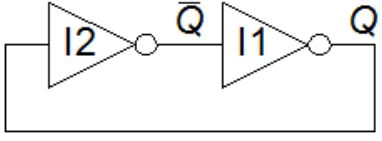
Synkron sekvensiell logikk

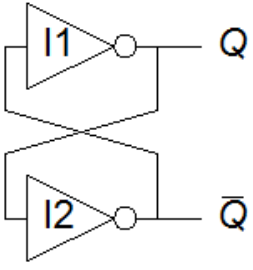
- Forenkler (abstraherer) tiden ned til to tilstander.
  1. Nåværende tilstand
  2. Neste tilstand
- Den kombinatoriske logikken sørger for å regne ut neste tilstand ut ifra nåværende tilstand.
- Men for å holde rede på disse tilstandene trenger vi
  – Minneelementer!



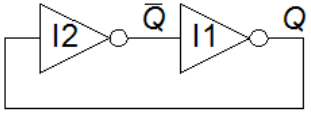
Bistabil krets

- Minneelement – et bistabilt element
  - Krets med to stabile tilstander.



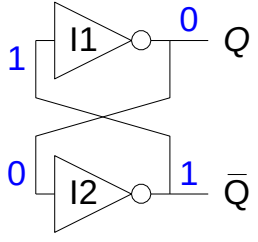


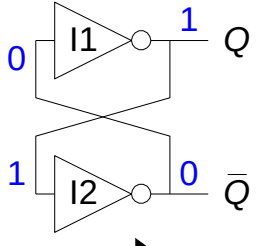
Figur til høyre synliggjør symmetrien.



Bistabil krets - analyse

- Vi har to mulige tilstander:
  - ***Q* = 0:**
    gir *Q'* = 1
  - ***Q* = 1:**
    gir *Q'* = 0



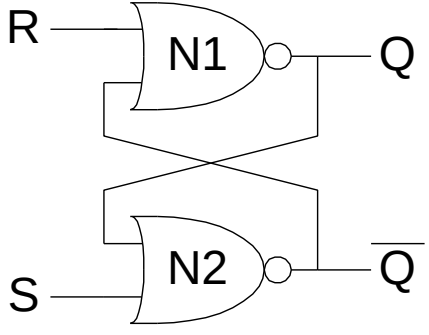


Not portene «jobber sammen»

- Kretsen *låser* verdien - *låsekrets*
  - Lagrer 1 bit i tilstandsvariabelen Q.
- Men vi har ingen muligheter for å kontrollere verdien.

SR-latch (Set / Reset)

- Hvordan virker denne?
- Fire mulige inngangskombinasjoner :



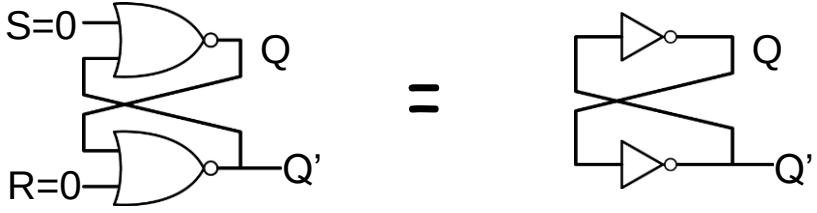
S	R	Q
0	0	
0	1	?
1	0	
1	1	

- Analyserer kretsen ved å studere inngangskombinasjonene hver for seg.

AB	NOR
00	1
01	0
10	0
11	0

SR-latch – Analyse 1

- R = S = 0

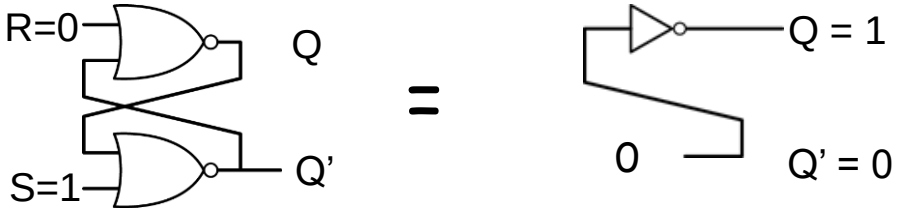


- Gir bistabil krets, dvs. tilstanden Q er låst
  - Utgangsverdien Q holdes låst

AB	NOR
00	1
01	0
10	0
11	0

SR-latch – Analyse 2

- R = 0, S = 1

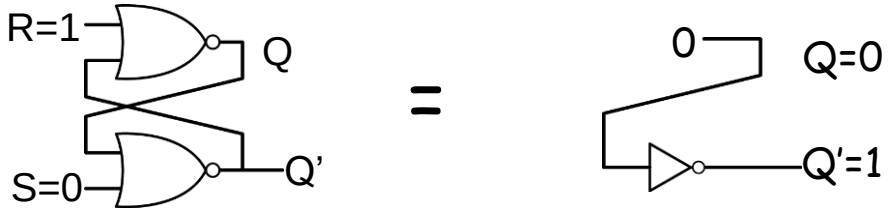


- Vi tvinger kretsen til å ha verdi Q = 1
  - Dvs. vi setter Q til 1.

AB	NOR
00	1
01	0
10	0
11	0

SR-latch – Analyse 3

- R = 1, S = 0

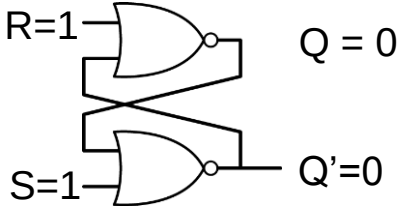


- Vi tvinger kretsen til å ha verdi Q = 0
  - Dvs. vi resetter Q til 0.

AB	NOR
00	1
01	0
10	0
11	0

SR-latch – Analyse 4

- R = 1, S = 1

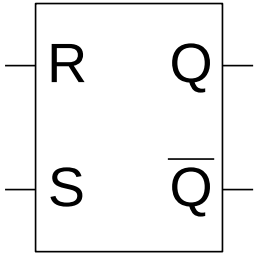


- I denne tilstanden er utgang Q' ikke den inverterte av Q.
- Ikke ønsket oppførsel

SR-latch oppsummering

- SR står for Set/Reset Latch
  - Lagrer tilstanden til et bit (*Q*)
- Kan kontrollere verdien til Q med *R og S*
  - *S* = *R* = *0: låser utgangen Q*
  - *S* = 0, *R* = 1: **Reset**ter utgang Q til 0
  - *S* = 1, *R* = 0**: Set**ter utgang Q til 1
  - *S* = *R* = 1 gir en ugyldig tilstand (må unngås)
- SR latch er ikke en praktisk krets.

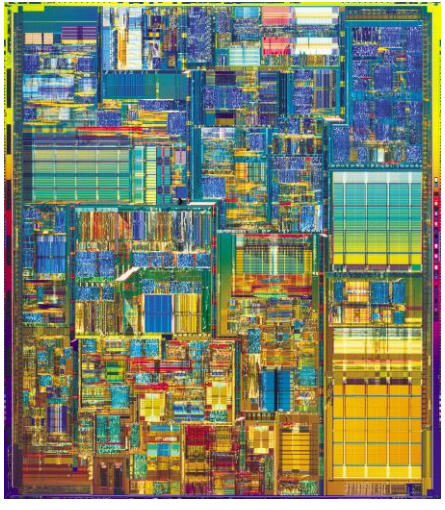
SR Latch Symbol

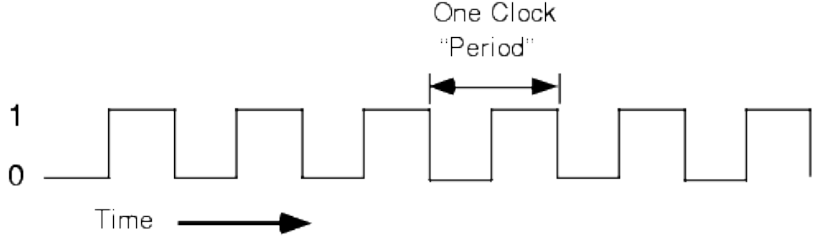


S	R	Q
0	0	låst
0	1	0
1	0	1
1	1	0

Store sekvensielle kretser

- I større digitale system har man behov for å synkronisere dataflyten.
  - Modulene må kunne snakke sammen.
  - Til dette bruker vi et globalt klokkesignal.
- Vi trenger minneelementer som støtter synkronisering.

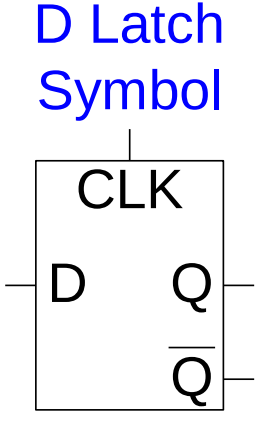




D-latch

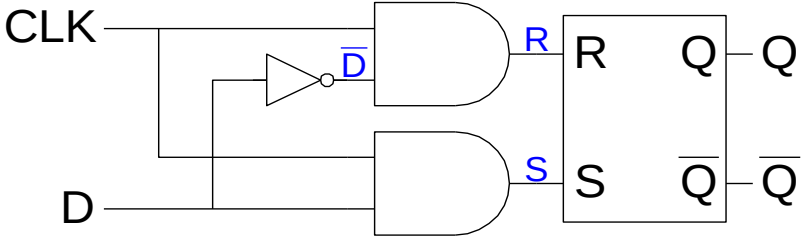
Lar et klokkesignal kontrollere når vi kan forandre utgangen Q

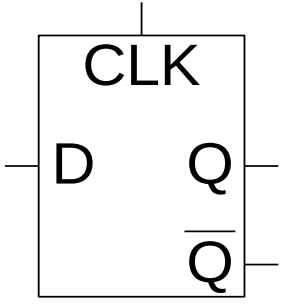
- To innganger : CLK, D
  - **CLK:** kontrollerer når utgangen (Q) kan forandre verdi
  - **D** (data inngang): hva utgangen (Q) skal bli satt til
- Funksjon
  - Når **CLK = 1**,
    Q blir satt til D (transparent)
  - Når **CLK = 0**,
    Verdien til Q låses (låst)



D-latch implementasjon og analyse

- Liten utvidelse av SR-latch





CLK	D	S	R	Q
0	X	0	0	låst
1	0	0	1	0
1	1	1	0	1

Analyse:
CLK = 0: Q låst
CLK = 1: Q = D

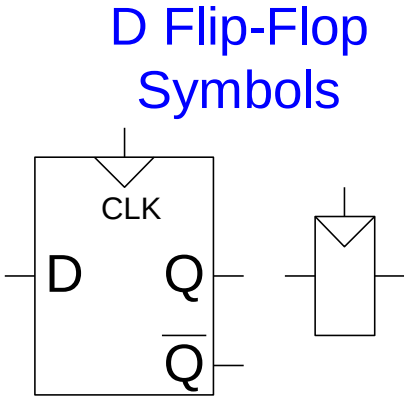
Merk at vi unngår SR-latch problemet med R=S=1.

D-latch

- D-latch er transparent så lenge CLK er høy. (~50% av tiden)
- I synkron logikk må vi ha mer kontroll på når minneelementer kan forandre verdi.
- Trenger noe som er mer presist
  - Transparent for et spesifikt lite tidsområde.

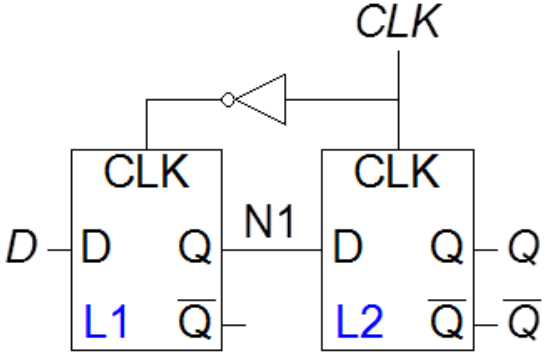
D-FlipFlop

- **Innganger:** *CLK*, *D*
- **Funksjon**
  - *Kun transparent ved positiv klokkeflanke*
    - Når *CLK* øker fra 0 to 1, blir Q satt til verdien av D.
- Merk symbolet ved CLK-inngangen.
  - = *flankettrigget*.
- Finnes mange variasjoner og utvidelser.
  - Negativ klokkeflanke.
  - Resettable-, Enable-, JK-, T-flipflop osv.



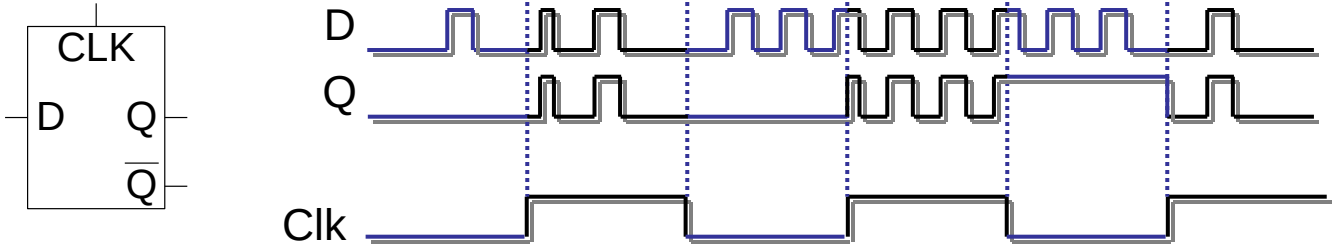
Implementasjon av D-flipflop

- En D-flipflop kan lages av to D-latcher
  - Når ***CLK* = 0**
    - L1 er transparent
    - L2 is låst
    - *N1 blir satt til D*
  - Når ***CLK* = 1**
    - L2 er transparent
    - L1 er låst
    - Q blir satt til N1
  - Dvs. på klokkeflanken (idet ***CLK*** går fra **0 til 1**)
    - *Verdien til Q blir satt til D*
    - *Kretsen fanger «klokkeflanke verdien til D»*

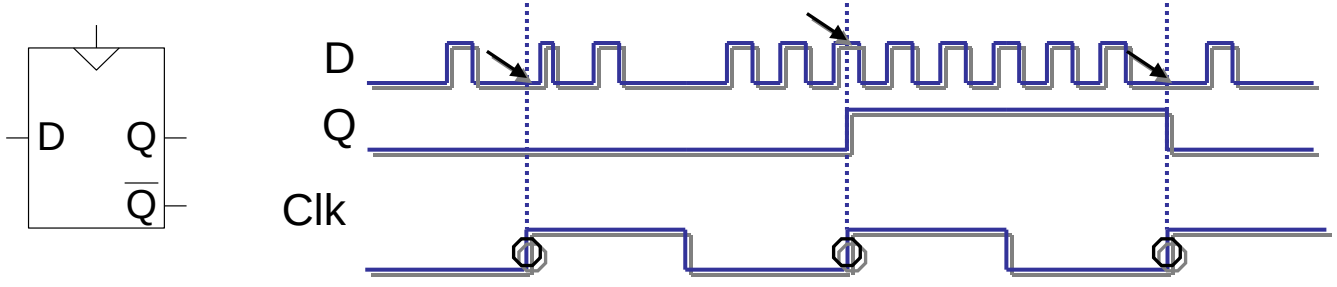


Sammenligning D-latch og D-flipflop

- En D latch er *transparent* når Clk=1, låses i det Clk går til 0



- D flip-flop er *transparent* ved positiv klokkkeflanken, låst ellers.



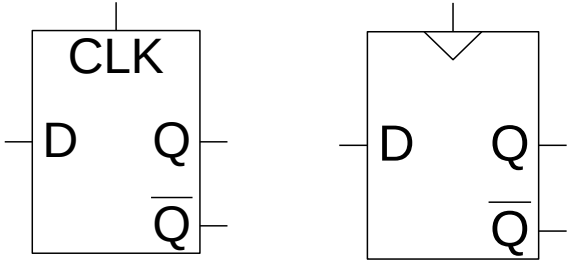
Merk!

- Når Latch/Flipflop låst:
  - Holder fast på gammel verdi.
- Når Latch/Flipflop transparent:
  - Slipper gjennom ny verdi.

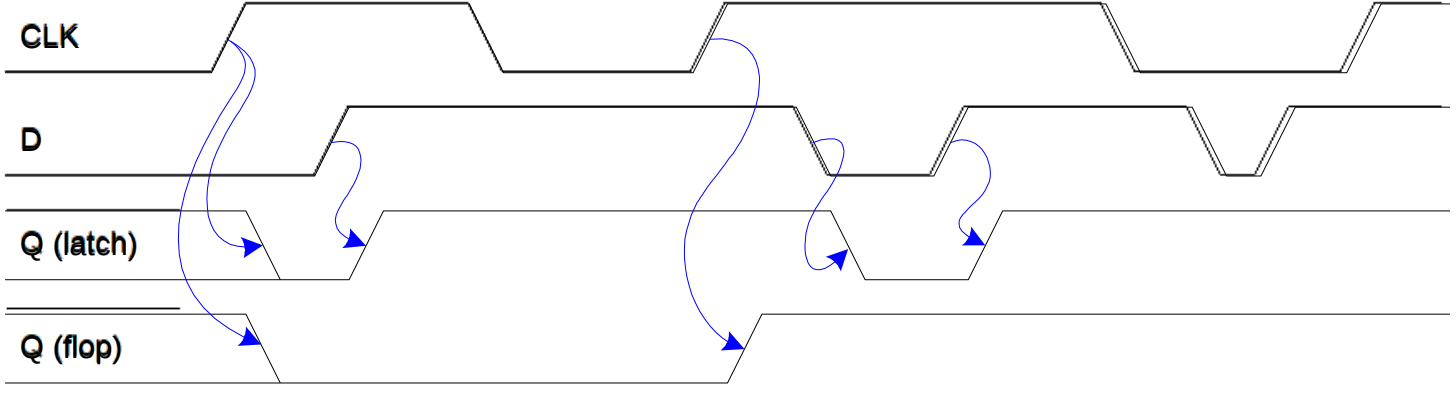
- D-latch er klokke-nivå sensitiv.
  - Transparent når CLK = 1.
- D-flipflop er flanketrigget.
  - Kun transparent på positiv klokkeflanke.

Tidsdiagram eksempel

D-latch transparent når CLK = 1

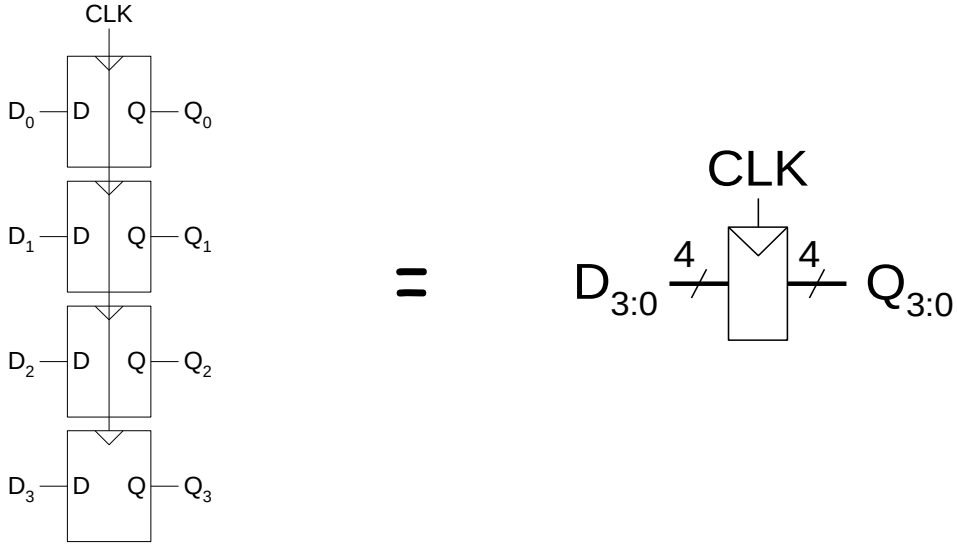


D-flipflop transparent på positiv klokkeflanke



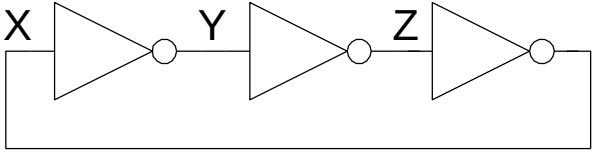
Registre

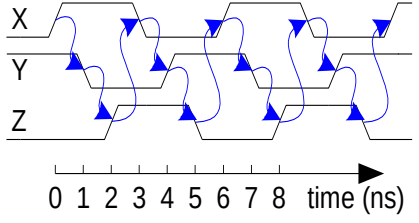
- En samling av flip-flop’s med felles klokke.



Sekvensiell logikk

- Sekvensielle kretser
  - Alle kretser som ikke er kombinatoriske
- Eksempel





  - Ustabil krets, oscillerer (ring oscillator)
  - Frekvensen avhenger av forsinkelsen til portene.
  - Direkte tilbakekobling
    - Dvs. utgangen er koblet direkte til inngangen

Asynkron sekvensiell logikk

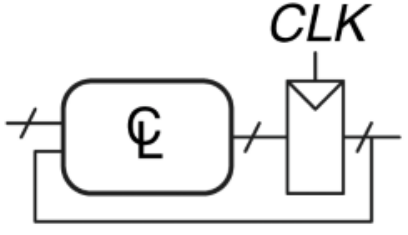
- Tilbakekobling går ikke gjennom registre.
  - Utganger er koblet direkte på innganger.
- Vanskelig å designe.
  - Mindre kontroll på funksjonalitet.
  - Mindre tilgjengelig verktøy og standarder.
  - Men fortsatt et forskningsfelt (Arm Amulet – 90-tallet)
    - Stort potensiale for raske og energieffektive systemer?
    - Interessant for Ai Chips?
      - AI's enorme energibehov kan gi asynkron logikk en ny sjanse.

Prosesseringslogikk er nesten alltid synkron – men asynkron logikk er vanlig i kommunikasjon mellom deler med ulike klokker, f.eks. modem, radio og I/O.

Synkron sekvensiell logikk

Ønsker sekvensielle systemer der vi har full kontroll på tiden.

- Løsningen:
  - Kontrollerer tilbakekoblingene ved å benytte *registre* (flipflop'er)
  - Registrene inneholder *tilstanden* av systemet
  - Tilstanden kan *kun forandres* på klokkeflankene

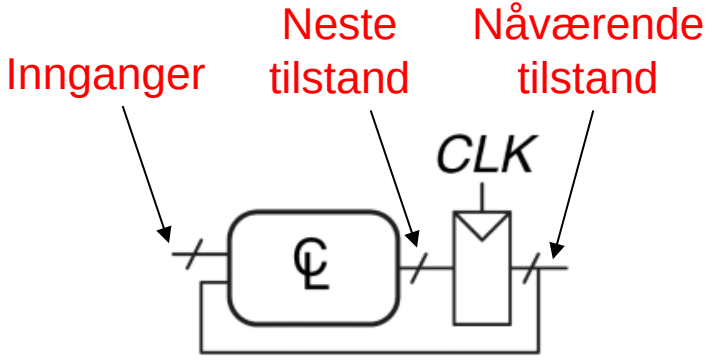


Synkrone sekvensielle systemer

«Forenkler» tiden ned til to tilstander.

- *Nåværende tilstand*
  - Den tilstanden vi har nå
- *Neste tilstand*
  - Den vi går til ved neste klokkeflanke

- Resultat: Vi kan sette opp en tilstandstabell.

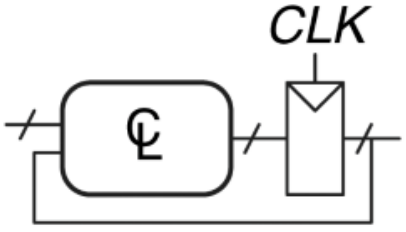


Nåværende tilstand $Q_{nå}$	Innganger Inn	Neste tilstand Q_{neste}
0	0	?
0	1	?
1	0	?
1	1	?

Synkron sekvensiell logikk – «Definisjon»

Må tilfredsstille følgende.

- Skal kun bestå av register og kombinatoriske elementer.
- Trenger minst et register (en flipflop)
- Alle registrene må få det samme klokkesignalet (global klokke)
- Alle tilbakekoblinger må gå via et register.



- To vanlige synkrone sekvensielle kretser
  - Tilstandsmaskiner (*Finite State Machines*)
  - Pipelines

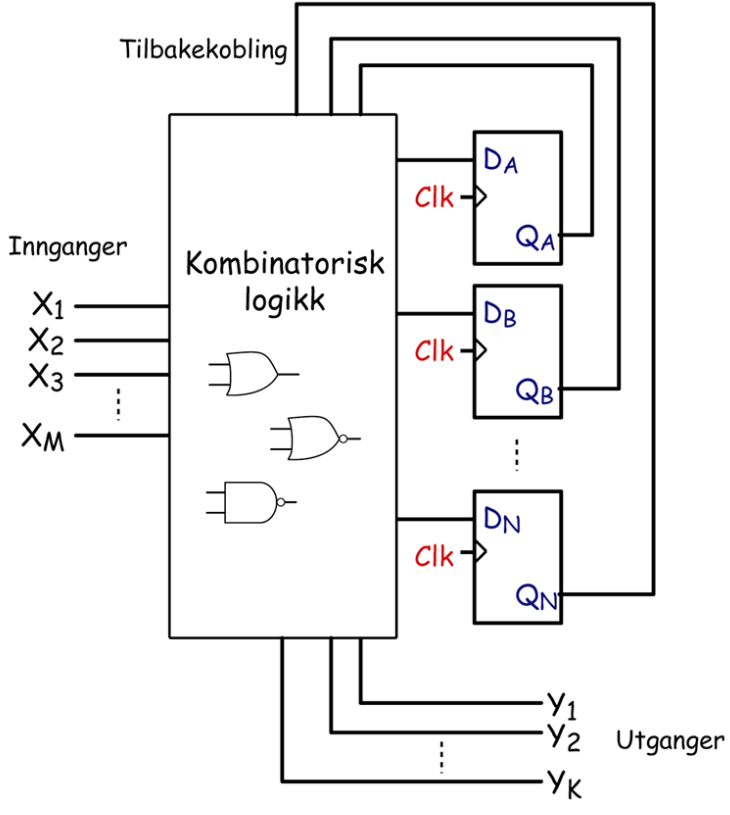
Tilstandsmaskiner

- En tilstandsmaskin er et sekvensielt system som gjennomløper et sett med tilstander styrt av verdiene på inngangssignalene.
- Tilstanden systemet befinner seg i, pluss evt. inngangsverdier, bestemmer utgangsverdiene og neste tilstand.

Tilstandsmaskins-konseptet gir en systematisk metode for å designe synkrone sekvensielle kretser.

Generell tilstands-maskin basert på D flip-flops

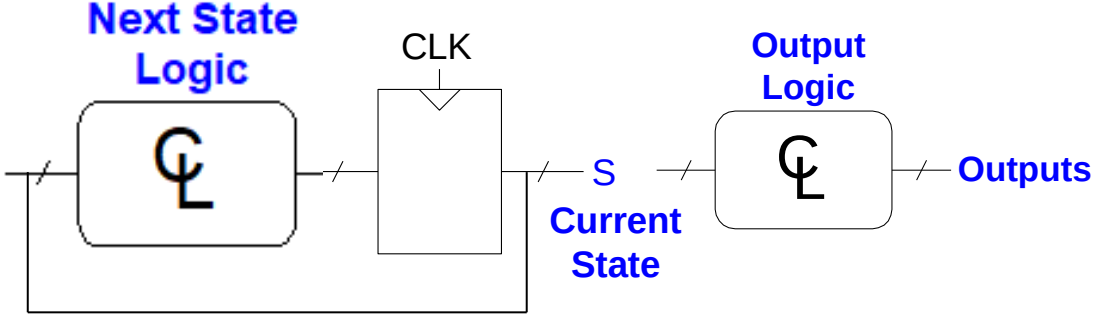
- N-stk flip-flops gir 2^N forskjellige mulige tilstander
- På hver klokkeflanke, går vi til neste tilstand, som blir bestemt av nåværende tilstand samt innganger.
- Dvs.
  - $Q_{neste} = F(X, Q_{nå})$



Tilstandsmaskiner – Nødvendige moduler/deler

Vi trenger følgende

- Tilstandsregister
  - Lagrer nåværende tilstand
  - Laster inn *neste tilstand* ved hver klokkeflanke
- Kombinatorisk logikk
  - Regner ut neste tilstand
  - Eventuelt utgangslogikken.

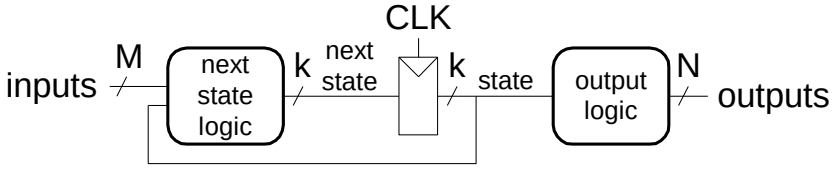


- ***Moore*** *enklere men tregere*
- ***Mealy-****raskere og færre tilstander, men mer kompleks og glitch-sårbar.*

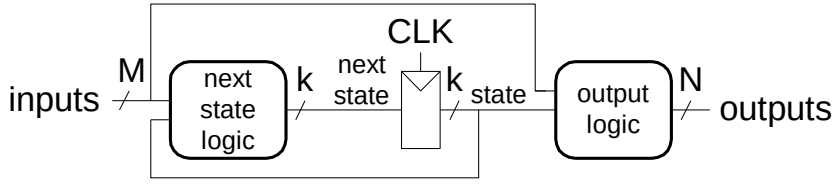
Moore vs Mealy

- Vi har to hovedtyper av tilstandsmaskiner
  - **Moore**: utgangen er kun bestemt av nåværende tilstand.
  - **Mealy**: utgangen er bestemt av både nåværende tilstand samt innganger.

Moore FSM



Mealy FSM



Hvordan designe tilstandsmaskiner?

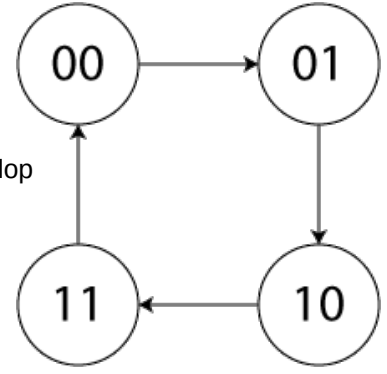
Verktøy

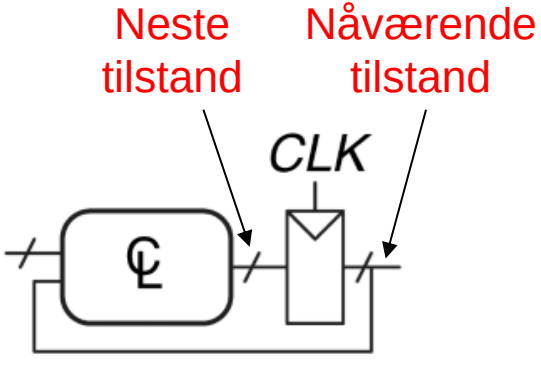
- *Tilstandsdiagrammer*
  - Grafisk beskrivelse
  - Fint abstraksjonsnivå for å skissere/prøve mulige løsninger
- *Tilstandstabell*
  - Den komplette logiske beskrivelsen

Eksempel – Design av teller.

- Skal syklisk telle fra 0 til 3
  - Dvs. 0-1-2-3-0-1... osv.
  - Tellefrekvensen vil være gitt av CLK
- Kan la tilstanden representere tallet direkte
  - Trenger 2 bit for å representere tallene 0 til 3 (Dvs. 2 D-flipflop'er)

Hvor mange D-flipflop trenger vi?



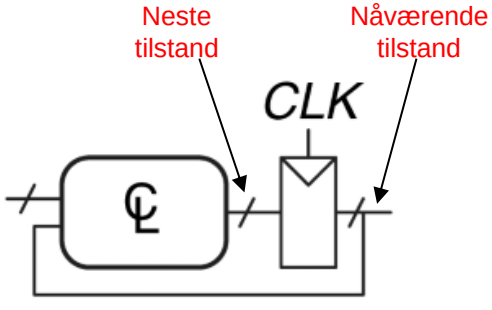


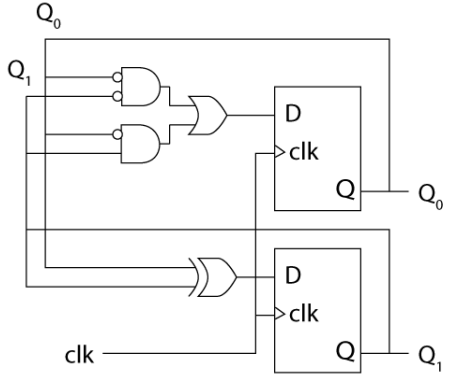
Nåværende tilstand Q	Neste tilstand Q_{neste}
00	01
01	10
10	11
11	00

Designe en teller - 2

- Kan finne den kombinatoriske logikken ved å lese mintermene ut av tilstandstabellen.
  - $Q_{neste0} = Q'_1 Q'_0 + Q_1 Q'_0$
  - $Q_{neste1} = Q_1 Q'_0 + Q'_1 Q_0$

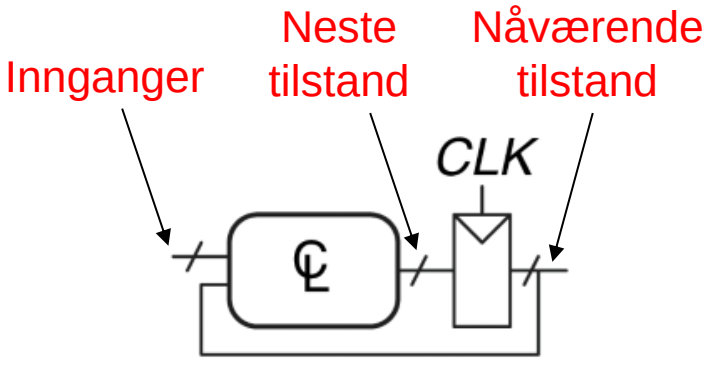
Nåværende tilstand Q	Neste tilstand Q_{neste}
00	01
01	10
10	11
11	00





Designe en teller - 3

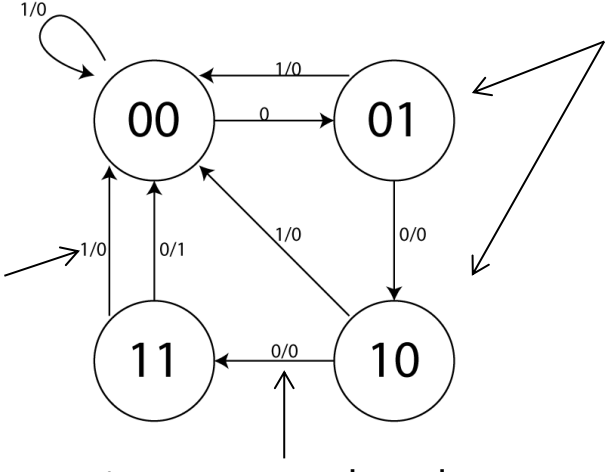
- La oss utvide systemet til å inkludere en *Reset* inngang, som resetter telleren til 0.



Nåværende tilstand Q	Innganger *Reset*	Neste tilstand Q_{neste}
0 0	0	0 1
0 1	0	1 0
1 0	0	1 1
1 1	0	0 0
x x	1	0 0

Tilstandsdiagram

Gir en grafisk illustrasjon av egenskapene til en tilstandsmaskin.



Tilstander

Inngangsverdi x som medfører ny tilstand.

Her kan vi bruke Y til å indikere hver gang vi har telt rundt og går fra 3 til 0.

Dersom utgangen er avhengig av inngangsverdien x, fører vi også opp utgangsverdi y (x/y). (Ikke mulig med Moore)

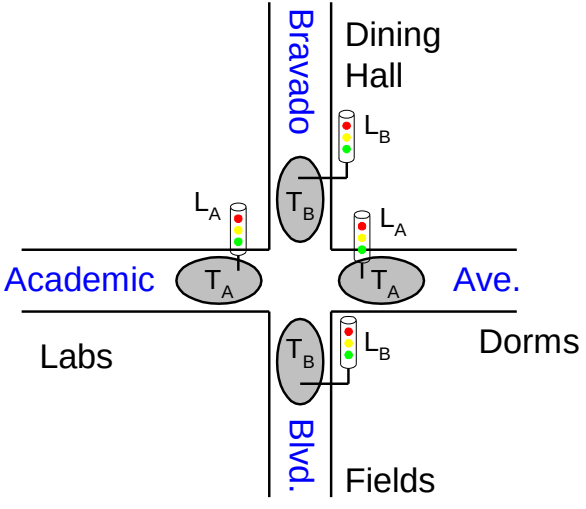
Tilstandstabell (eksempel)

- «Sannhetstabellen» for tilstandsmaskin.
- Den kombinatoriske logikken er **entydig** gitt av tilstandstabellen.
- Kan lese ut funksjonsutrykkene via mintermer.
  - $D_A=F(Q_A,Q_B,R)$
  - $D_B=F(Q_A,Q_B,R)$
  - $y=F(Q_A,Q_B,R)$

Nåværende tilstand		Inngang	Neste tilstand		Utgang for nåværende tilstand
Q_A	Q_B	R	D_A	D_B	y
0	0	1	0	0	0
0	0	0	0	1	0
0	1	1	0	0	0
0	1	0	1	0	0
1	0	1	0	0	0
1	0	0	1	1	0
1	1	1	0	0	0
1	1	0	0	0	1

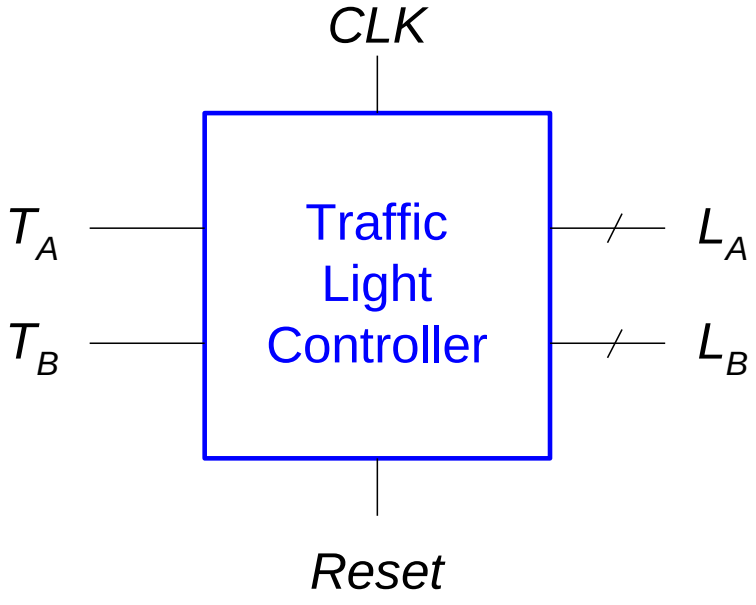
Trafikklys eksempel fra boka

- Trafikk-sensorer: T_A, T_B
  - TRUE når det er trafikk over sensoren
  - Skal ikke bli rødt så lenge det er trafikk.
- Trafikklys : L_A, L_B
  - Disse skal vi kontrollere.



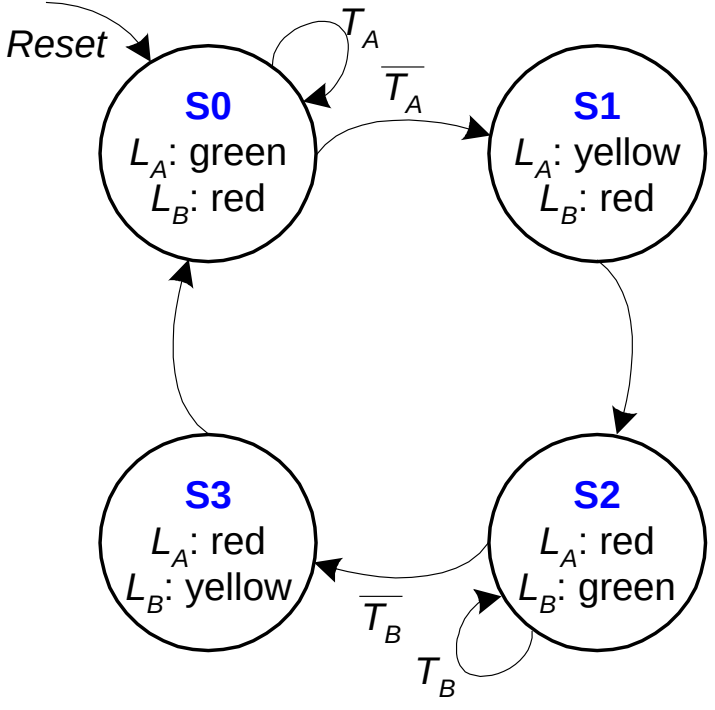
Nødvendige inn- og utganger

- Innganger: *CLK*, *Reset*, T_A, T_B
- Utganger: L_A, L_B

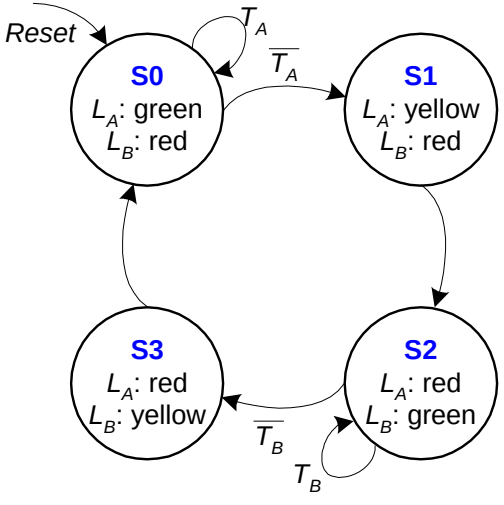


Tilstandsdiagarm

- Moore tilstandsmaskin:
  - Utgangene kun gitt av tilstanden
- Merk at vi kun trenger 4 tilstander for å beskrive ønsket funksjonalitet.
- Merk pilene som viser hvilke innganger som er nødvendig for å forandre tilstand.



Tilstandstabell



Merk notasjon **S'**

Current State	Inputs		Next State
S	T_A	T_B	S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

Tilstandstabell og tilstandskoder

- Finner passende tilstandskoder for tilstandene.

State	Encoding
S0	00
S1	01
S2	10
S3	11

Ny tilstandstabell med tilstandskoder

Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = \overline{S}_1 \overline{S}_0 \overline{T}_A + S_1 \overline{S}_0 \overline{T}_B$$

Utgangslogikk for å få riktige farger

Tilstand	S_1	S_0	L_A	L_B
S0	0	0	green	red
S1	0	1	yellow	red
S2	1	0	Red	green
S3	1	1	red	yellow

Output	Encoding
green	00
yellow	01
red	10

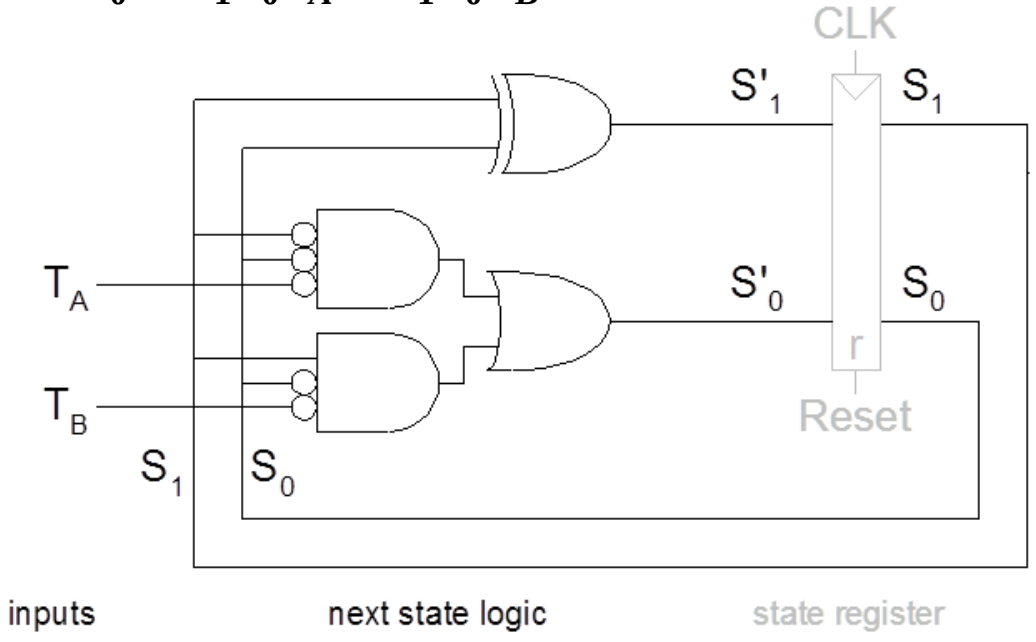
Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

$$L_{A1} = S_1$$
$$L_{A0} = \overline{S_1} S_0$$
$$L_{B1} = \overline{S_1}$$
$$L_{B0} = S_1 S_0$$

Tegne tilstandslogikken

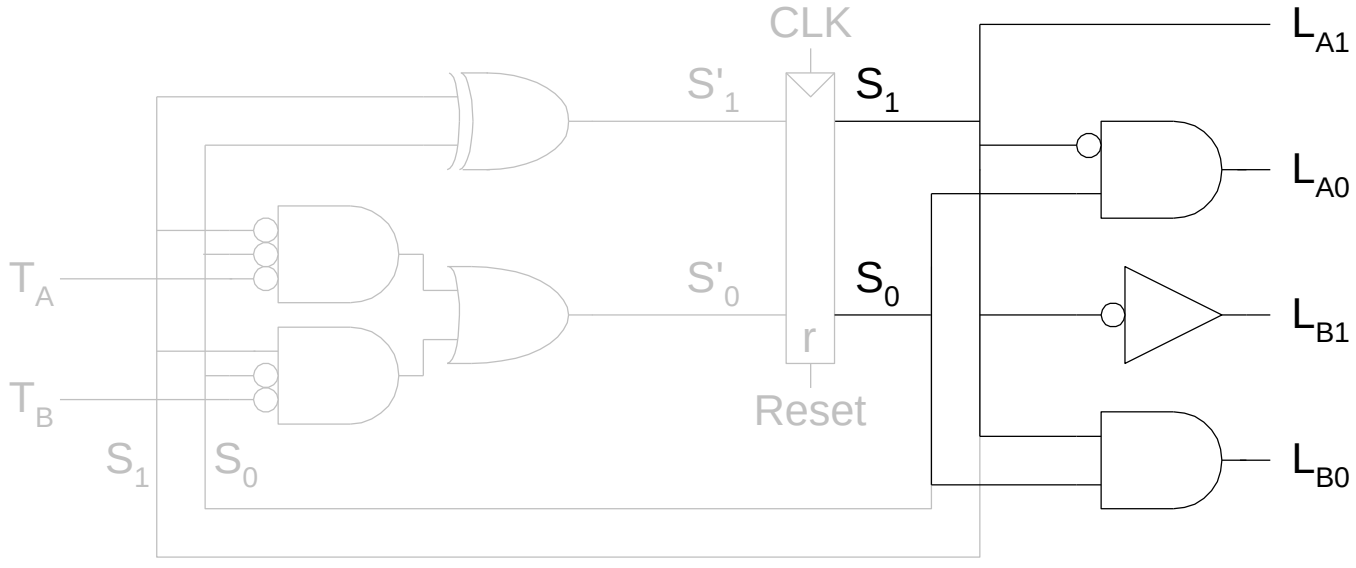
$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = \overline{S_1 S_0 T_A} + S_1 \overline{S_0 T_B}$$



Tegne utgangslogikken

$$L_{A1} = S_1$$
$$L_{A0} = \overline{S_1} S_0$$
$$L_{B1} = \overline{S_1}$$
$$L_{B0} = S_1 S_0$$



inputs | next state logic | state register | output logic | outputs

Generell designprosedyre tilstandsmaskiner

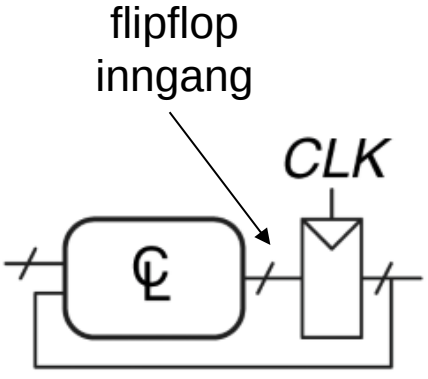
1. Identifiser tilstander, innganger og utganger
2. Velg tilstandskoder og tegn tilstandsdiagrammer
3. Sett opp tilstandstabell
4. Valider kretsen, se eventuell om den kan optimaliseres.
   - Sjekk eventuelt ubrukte tilstander
5. Finn boolske uttrykk for tilstand- og utgangslogikk
6. Tegn opp kretsen.

Diverse momenter

- Ubrukte tilstander
  - Oppstår dersom man ikke bruker alle 2^N tilgjengelige tilstander.
  - Må håndteres slik at vi ikke havner i uendelig løkker og andre uønskede situasjoner.
- Tilstandskoder kan påvirke løsningen
  - Anbefalte fremgangsmåte - prøv og feil i tilstandsdiagrammet
  - CAD (Computer-aided design) programvare kan finne fornuftige løsninger.
- Merk at vi får forskjellige løsninger ved å velge Mealy og Moore.
  - Typisk trenger vi færre tilstander med Mealy.

Timing av tilstandsmaskiner

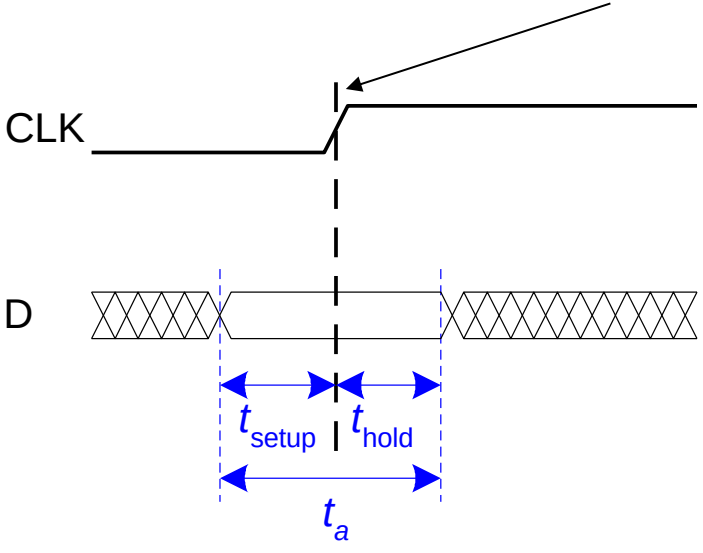
- Vi kan si at flip-flop'ene «sampler» inngangsverdiene på klokkeflankene.
- Inngangen til flip-flop'ene må være stabile når de «samples».
  - Tilsvarende for når man tar et fotografi.
  - Motivet må være tilstrekkelig stille når man tar et bilde.
- Dersom vi sampler en ikke stabil inngang kan vi få *metastabilitet*.



Tidsbegrensninger på inngangene av flipflop'ene.

- **Setup time:** t_{setup}
- **Hold time:** t_{hold}
- ***Aperture* time:** $t_a = t_{setup} + t_{hold}$

Tidsperiode som inngangsdataene må være stabile ved klokkeflanken

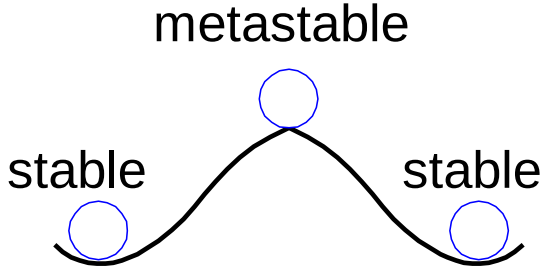


Tidsbegrensninger synkron logikk

- Innganger til synkrone sekvensielle kretser må være stabile i bestemte perioder, slik at flipflop'ene får med seg riktig verdi («sampler» riktig verdi).
- Vi har derfor noen tidsbegrensninger som vi må holde oss innenfor når vi designer synkrone kretser.
  - CAD verktøy hjelper oss her.
  - Flere detaljer i boken.

Metastabilitet

- Generelt kan vi ikke garantere stabilt signal når flipflop'ene skal lese inngangssignalet.
  - Alle *bistabile* kretser har også en metastabil tilstand
    - Når utgang Q får en verdi mellom 0 og 1
  - *Resolution time* - tiden det tar til å få en stabil tilstand igjen.
    - Regnes ut vha. statistiske metoder.
  - *Mean time between failures* (MTBF)
    - Den gjennomsnittlige tiden mellom feil i systemet.



Til neste gang

- VHDL med Philipp
  - VHDL (Very High Speed IC Hardware Description Language)
  - Et programmeringsspråk for å beskrive digitale systemer / hardware.
- Bygger på
  - Kombinatorisk logikk
  - Sekvensiell logikk
    - D-latch og D-flipflop
  - Tilstandsmaskiner.
    - Tilstandstabeller, tilstandsdiagram og designprosedyren

Ukeoppgaver uke 3

- Excercises fra boka:
  - 3.1, 3.3, 3.5, 3.7, 3.9, 3.12, 3.14, 3.18, 3.19, 3.20, 3.22, 3.27, 3.29, 3.30 og 3.31.
  - *Ekstra*: 3.10, 3.24, 3.25, 3.32