IE1205 Digital Design:

F13: Asynkrona Sekvensnät

Denna föreläsning

- Ämne: Asynkrona sekvenskretsar
- Intro
- Analys
- Syntes
- Kretssjukdomar

Asynkrona Sekvenskretsar

- Asynchronous finite-state machine
- Asynchronous sequential circuit
- Asynkron sekvenskrets
- Asynkron sekvensmaskin (svengelska)
- Asynkron tillståndsmaskin (svengelska)
- Asynkront sekvensnät
- Fundamental-mode circuit (bredare begrepp)

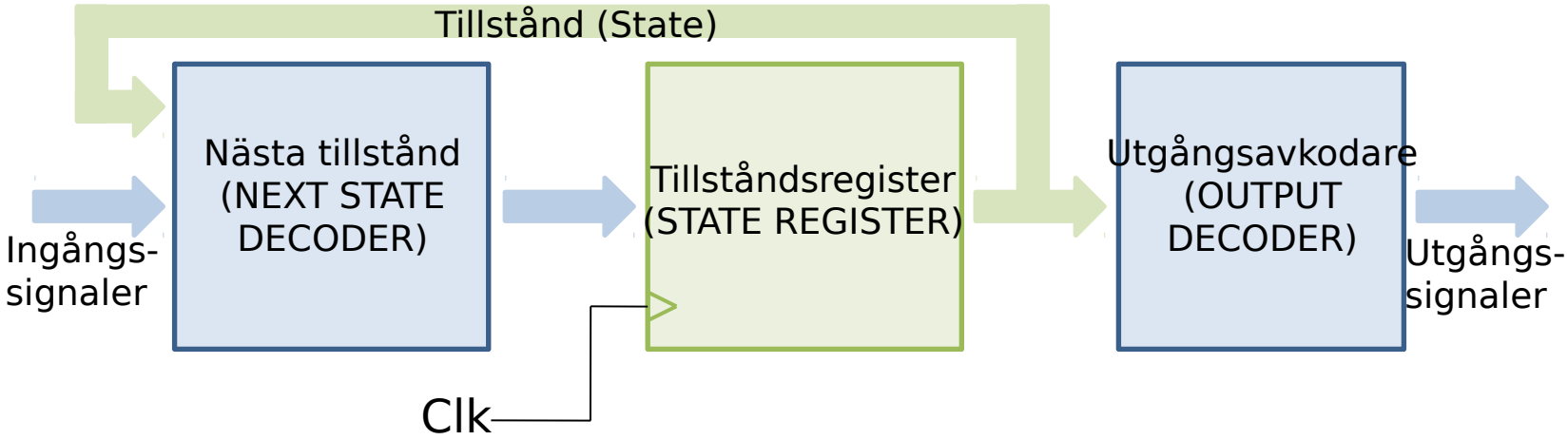
Asynkrona Sekvensmaskiner

- En asynkron sekvensmaskin är en sekvensmaskin utan vippor
- Asynkrona sekvensmaskiner bygger på en analys av återkopplade kombinatoriska grindnätverk
- **<u>Antagandet</u>: Endast EN signal i taget i grindnätet kan förändra sitt värde vid någon tidpunkt**

Asynkrona tillståndsmaskiner

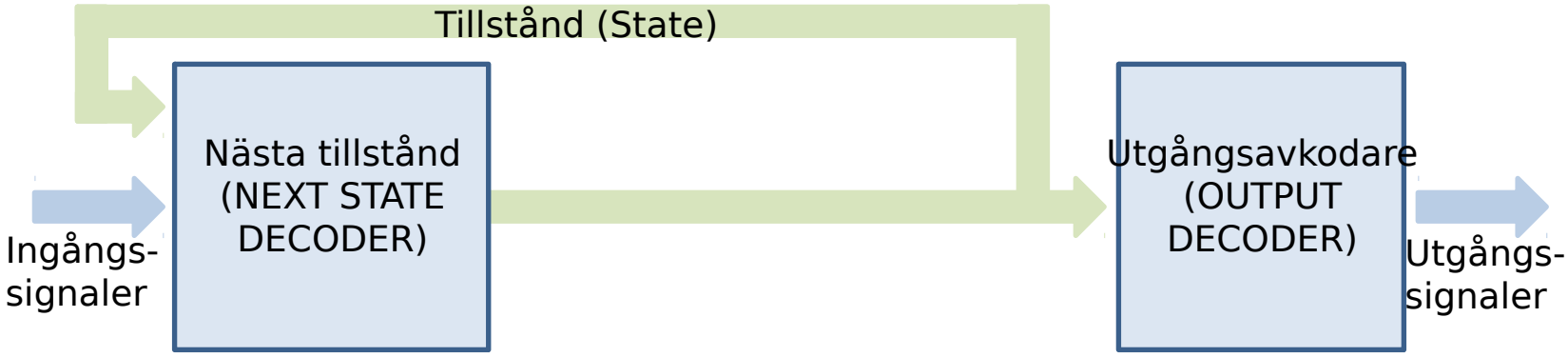
- Asynkrona tillståndsmaskiner används då det är nödvändigt att bibehålla ett tillstånd, men då det inte finns någon klocka tillgänglig.
  - Alla vippor och latchar är asynkrona tillståndsmaskiner
  - Användbara för att synkronisera händelser i situationer där metastabilitet är/kan vara ett problem

Synkron Moore-Automat



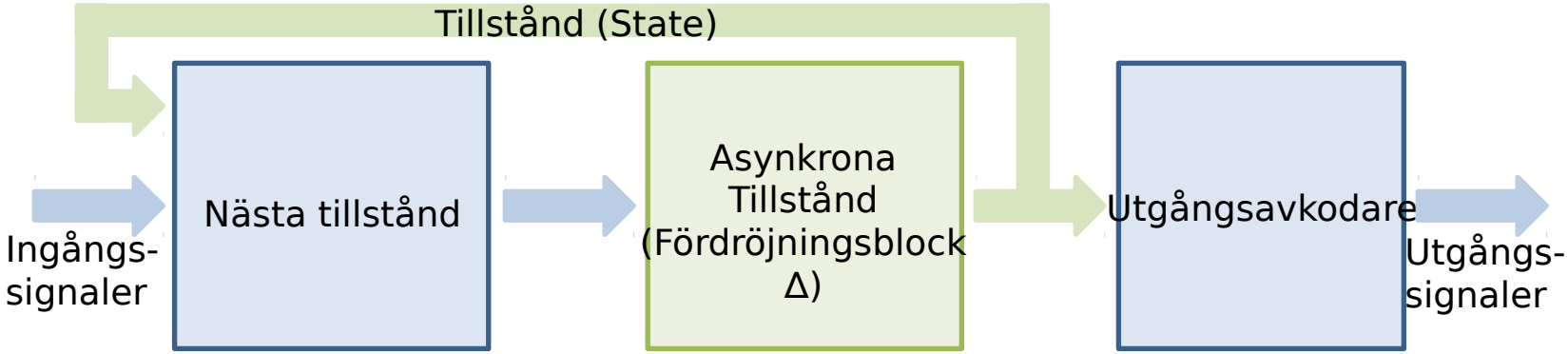
- I en synkron Moore-automat klockas tillståndsregistret av en klocka

Asynkron Moore-Automat



- Asynkrona sekvensnät har liknande uppbyggnad som synkrona sekvensnät, men inte några klockade vippor
- Kräver att vi har en fördröjning i nästa tillstånds kodaren (vilket i praktiken aldrig är något problem)

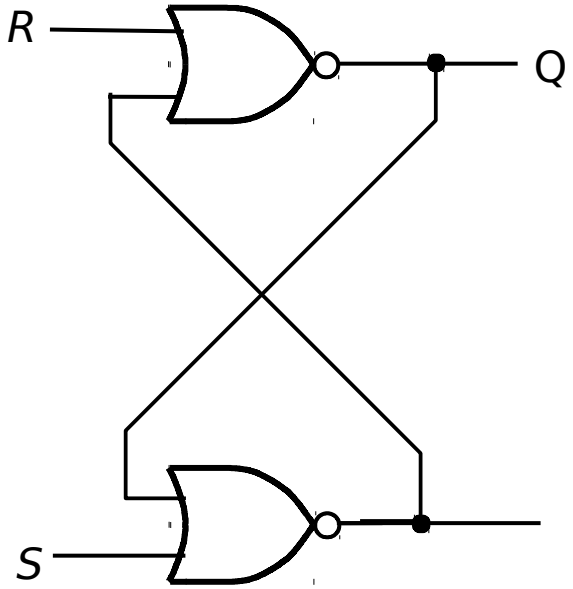
Asynkrona tillståndsmaskiner
Moore-kompatibel automat



- För att förenkla analysen och kunna teckna ekvationer, gruppera alla grindfördröjningar i ett block
- I en verklig implementering utnyttjar vi grindarna i fördröjning

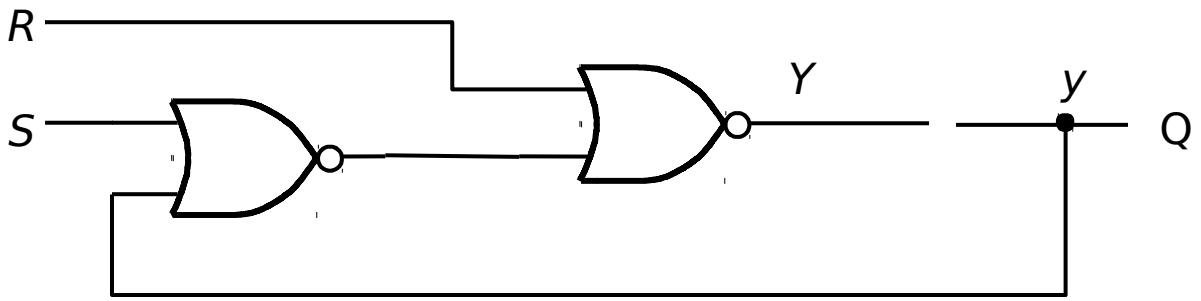
Gyllene regeln

Analys av asynkron sekvenskrets
SR-latch med NOR-grindar



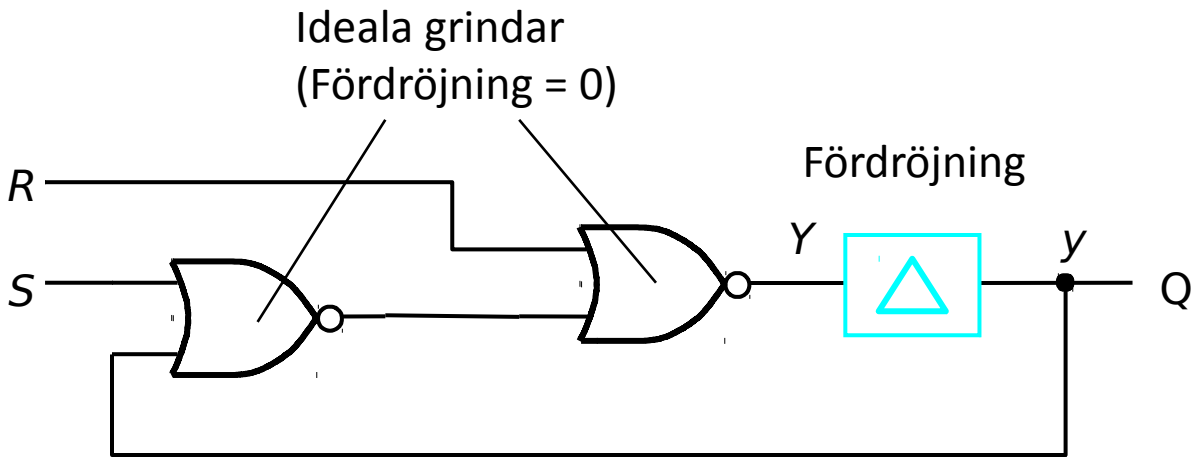
Analys av asynkron sekvenskrets SR-latch med NOR-grindar

- "Dra i kretsen" så att återkopplingarna blir så få som möjligt (minimalt snitt)
- "Klipp upp kretsen"
- y – Nuvarande tillstånd
- Y – Nästa tillstånd



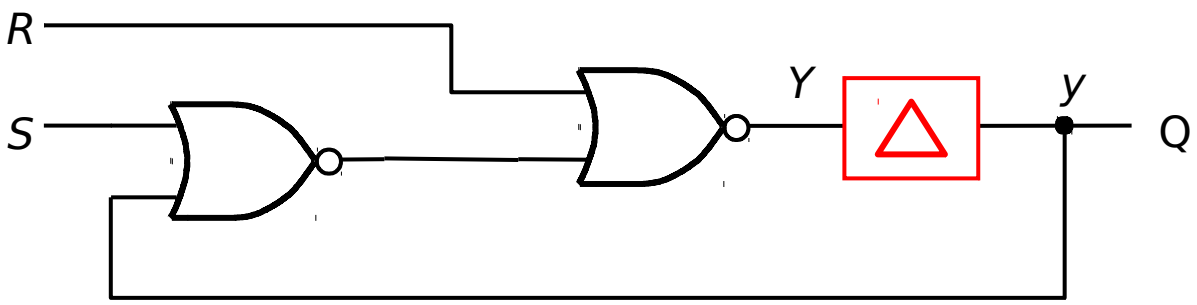
Asynkron sekvenskrets
SR-latch med NOR-grindar

- För att analysera beteendet av en asynkron krets så antar man ideala grindar och sammanfattar fördröjningen till ett enda block märkt Δ.



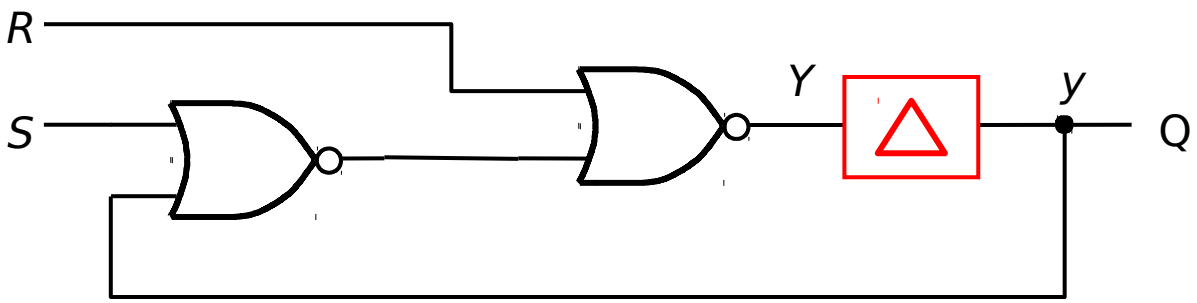
Analys av det asynkrona sekvensnätet

- Genom att vi har ett fördröjningsblock kan vi betrakta
  - y som nuvarande tillstånd
  - Y som nästa tillstånd



Tillståndstabell

- Därmed kan vi ta fram tillståndstabellen där nästa tillstånd Y beror på insignalerna och nuvarande tillstånd y



$$Y = \overline{R + \overline{(S + y)}}$$

Tillståndstabell

Present state y	Next state			
	SR = 00	01	10	11
	Y	Y	Y	Y
0	0	0	1	0
1	1	0	1	0

$$Y = R + \overline{(S + y)}$$

Tillståndstabell

BV använder binärkodsordning

Från tillståndsfunktion till sanningstabell

$$Y = \overline{R + \overline{(S + y)}}$$

y	S	R	$Y = \overline{R + \overline{(S + y)}}$
0	0	0	$0 = \overline{0 + \overline{(0 + 0)}}$
0	0	1	$0 = \overline{1 + \overline{(0 + 0)}}$
0	1	0	$1 = \overline{1 + \overline{(1 + 0)}}$
0	1	1	$0 = \overline{1 + \overline{(1 + 0)}}$
1	0	0	$1 = \overline{0 + \overline{(0 + 1)}}$
1	0	1	$0 = \overline{1 + \overline{(0 + 1)}}$
1	1	0	$1 = \overline{0 + \overline{(1 + 1)}}$
1	1	1	$0 = \overline{1 + \overline{(1 + 1)}}$

Present state y	Next state SR = 00	01	10	11
	Y	Y	Y	Y
0	0	0	1	0
1	1	0	1	0

Eller som på övningen – med hjälp av Karnaughdiagram …

Stabila tillstånd

Present state y	Next state			
SR =	00	01	10	11
	Y	Y	Y	Y
0	0	0	1	0
1	1	0	1	0

- Eftersom vi inte har vippor utan bara kombinatoriska kretsar kan en tillståndsändring medföra ytterligare tillståndsändringar
- En tillstånd är
  - stabil om Y(t) = y(t + Δ)
  - instabil om Y(t) ≠ y(t + Δ)

$$Y = y$$ stabilt

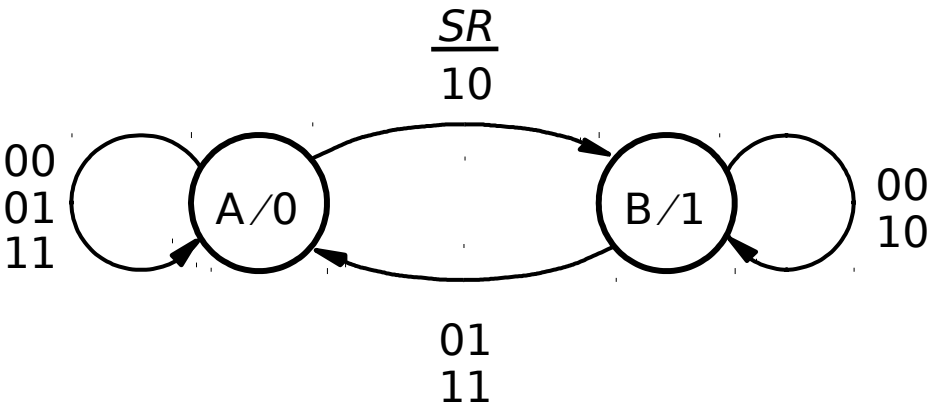
Den asynkrona tillståndstabellen Excitationstabellen

- Den kodade asynkrona tillståndstabellen kallas för *Excitationstabell*
- Stabila tillstånd (next state = present state) ringas in

Present state y	Next state			
SR =	00	01	10	11
	Y	Y	Y	Y
0	(0)	(0)	1	(0)
1	(1)	0	(1)	0

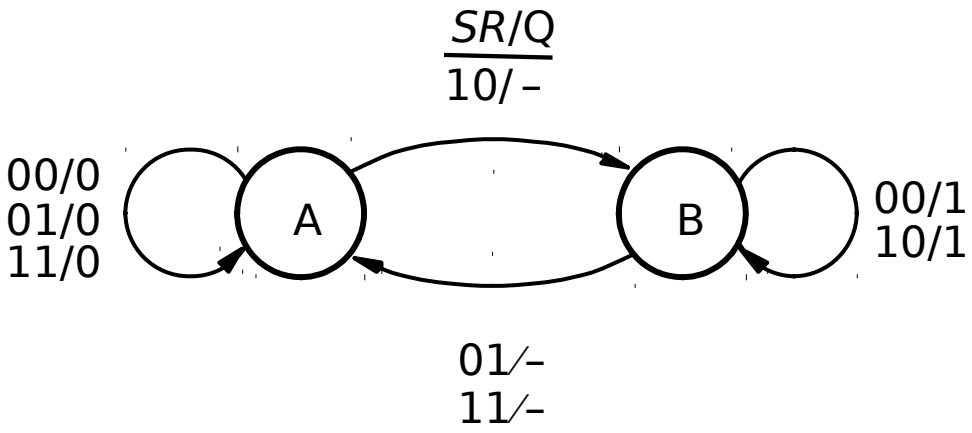
Flödestabell och Tillståndsdiagram (Moore)

Present state	Next state				Output Q
	SR = 00	01	10	11	
A	(A)	(A)	B	(A)	0
B	(B)	A	(B)	A	1



Flödestabell och Tillståndsdiagram (Mealy)

Present state	Next state				Output, Q			
	SR = 00	01	10	11	00	01	10	11
A	(A)	A	B	(A)	0	0	–	0
B	(B)	A	(B)	A	1	–	1	–

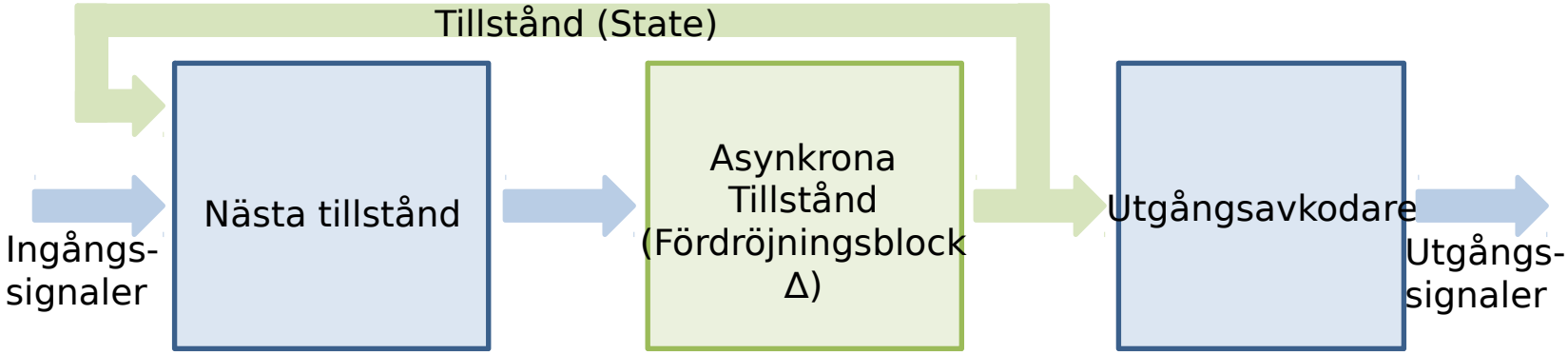


Don’t care (‘-’) har valts för utgångsavkodaren, eftersom utgången ändras direkt efter tillståndsövergången (enklare implementering)

Terminologi

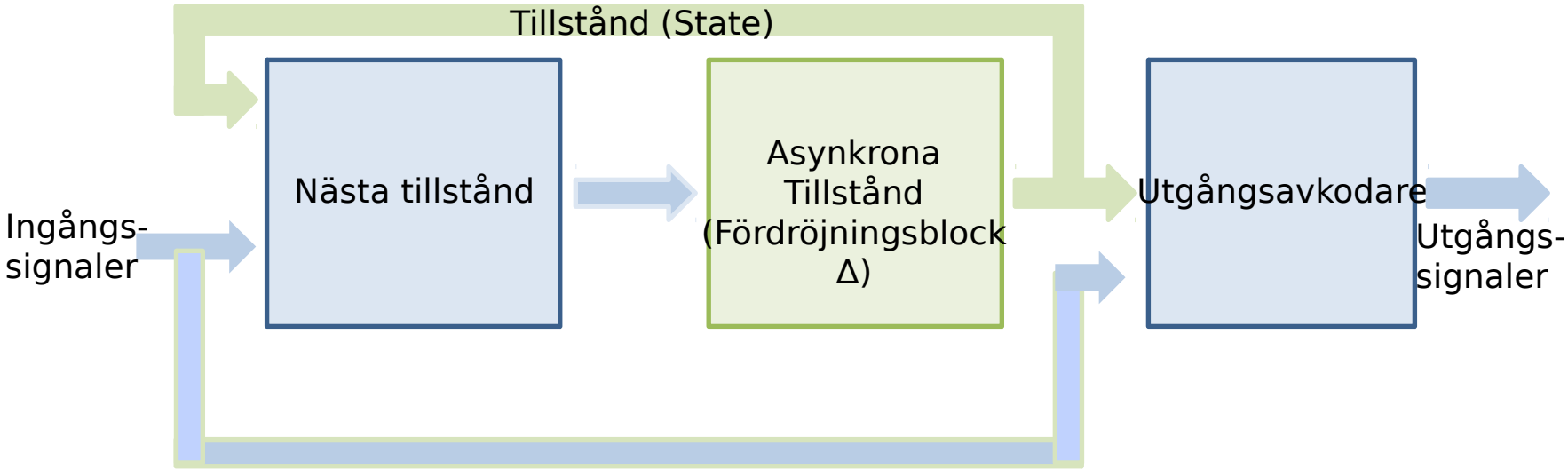
- När man arbetar med asynkrona sekvensnät så används det en annan terminologi
  - Den symboliska asynkrona tillståndstabellen kallas *flödestabell*

Asynkrona tillståndsmaskiner
Moore-kompatibel automat



- Asynkrona sekvensnät har liknande uppbyggnad som synkrona sekvensnät
- I stället för vipppor har man fördröjningsblock

Asynkrona tillståndsmaskiner
Mealy-kompatibel automat



- Asynkrona sekvensnät har liknande uppbyggnad som synkrona sekvensnät
- I stället för vippor har man fördröjningsblock

Analys av Asynkrona kretsar

- Analysen görs i följande steg:
  1) Ersätt återkopplingar i kretsen med ett delay-element Δ_i. Insignalen till delay-elementet bildar nästa tillstånd (next state) signalen Y_i, medan utsignalen y_i representerar nuvarande tillstånd (present state).
  2) Ta fram uttryck för nästa-tillstånds-signaler och utsignaler
  3) Ställ upp excitationstabell
  4) Skapa en flödestabell och byt ut kodade tillstånd mot symboliska
  5) Rita ett tillståndsdiagram om så behövs

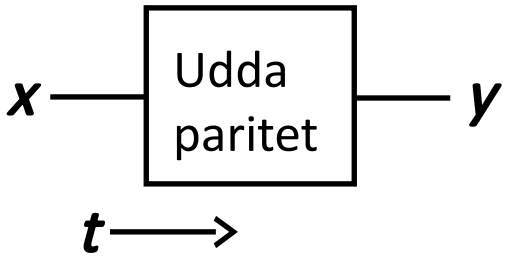
Syntes av asynkrona kretsar

- Syntesen genomförs i följande steg:
  1) Skapa ett tillståndsdiagram enligt funktionsbeskrivningen
  2) Skapa en flödestabell och reducera antalet tillstånd om möjligt
  3) Tilldela koder till tillstånden och skapa excitationstabellen
  4) Ta fram uttryck (överföringsfunktioner) för nästa tillstånd samt utgångar
  5) Konstruera en krets som implementerar ovanstående uttryck

Exempel: seriell paritetskrets

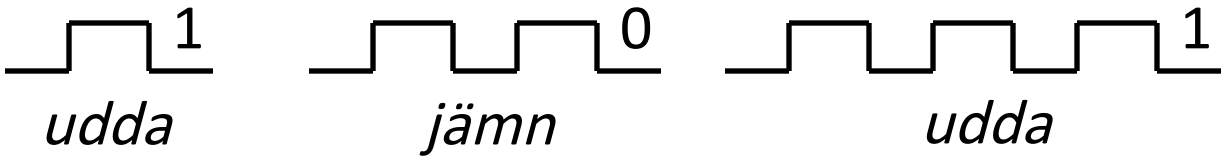
Ingång ***x*** Utgång ***y***

y = 1 om antalet pulser på ingången ***x*** har varit udda.

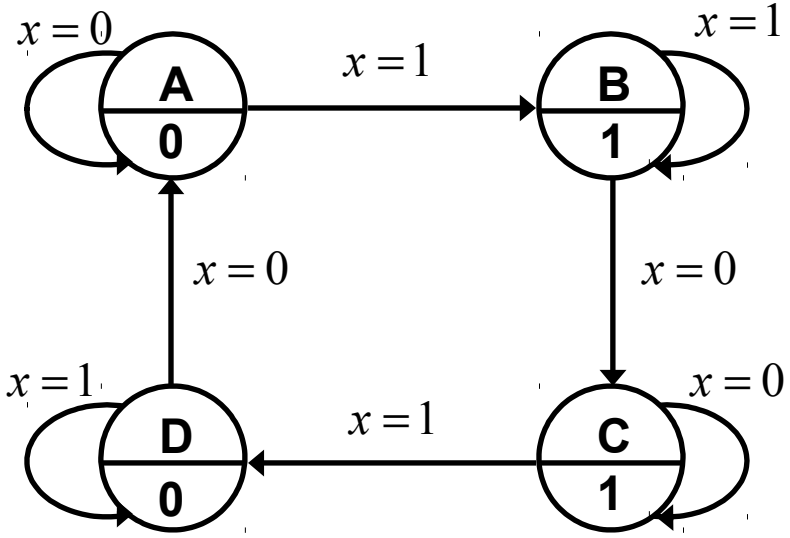


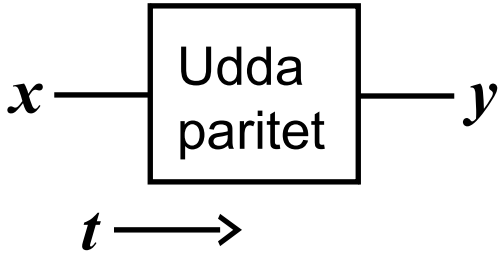
Med andra ord en ”varannangång” krets

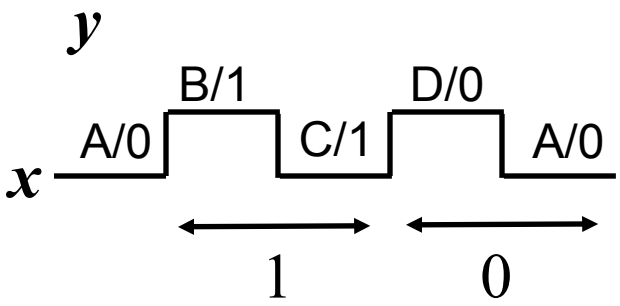
…



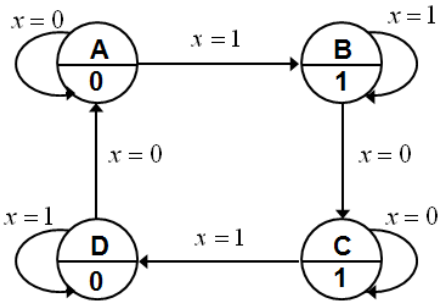
Skapa tillståndsdiagram

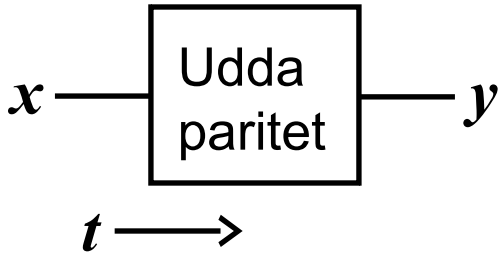






Skapa flödestabellen





Pres state	Next State		Q
	X=0	1	
A	(A)	B	0
B	C	(B)	1
C	(C)	D	1
D	A	(D)	0

Vad är bra tillståndskod?

00, 01, 10, 11 - binärkod

Pres state	Next State		Q
	X=0 ← 1		
y_2y_1	Y_2Y_1		
00	00	01	0
01	10	01	1
10	10	11	1
11	00	11	0

Dålig kodning (HD=2!)

- *Antag*

$X = 1 \quad Y_2 Y_1 = 11$

- *därefter*

$X \rightarrow 0 \rightarrow Y_2 Y_1 = 00?$

$11 \rightarrow 10!$

$11 \rightarrow 01 \rightarrow 10! \quad ? \rightarrow 00$

Vi når aldrig 00?

Vad är bra tillståndskod?

00, 01, 11, 10 - graykod

- *Antag*

$X = 1 \quad Y_2 Y_1 = 10$

- *därefter*

$X \rightarrow 0 \rightarrow Y_2 Y_1 = 00$

$10 \rightarrow 00$

Pres state	Next State		Q
	X=0 ← 1		
y_2y_1	Y_2Y_1		
00	00	01	0
01	11	01	1
11	11	10	1
10	00	10	0

Bra kodning (HD=1)

Tillståndskodning

- I asynkrona sekvensnät är det omöjligt att garantera att två tillståndsvariabler ändrar värdet samtidigt
  - Därmed kan en övergång 00 → 11 resultera i
    - en övergång 00 → 01 → ???
    - en övergång 00 → 10 → ???
- För att säkerställa funktionen **MÅSTE** alla tillståndsövergångar ha ***Hamming distansen* 1**
  - Hamming distansen är antalet bitar som skiljer sig i två binära tal
    - Hamming distansen mellan 00 och 11 är 2
    - Hamming distansen mellan 00 och 01 är 1

Richard Hamming

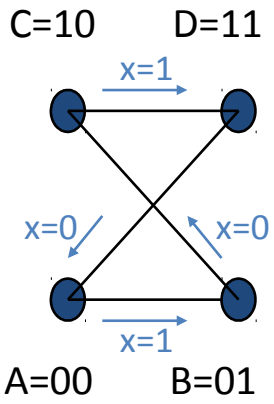
Gyllene regeln

Tillståndskodning

- Procedur för att erhålla bra koder:
  1) Rita transitionsdiagram längs kanterna i hyperkuben som bildas av koderna
  2) Ta bort ev korsande linjer genom att
     a) byta plats på två närliggande noder
     b) utnyttja tillgängliga icke använda koder (utnyttja instabila tillstånd)
     c) introducera fler dimensioner i hyperkuben

Exempel: Serial Parity Generator

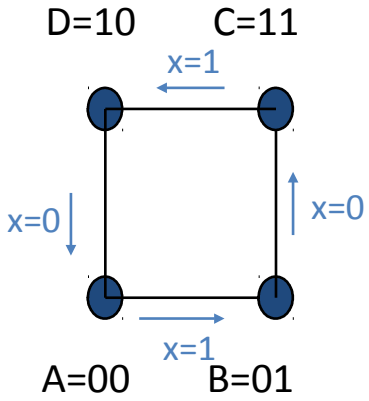
Pres state	Next State		Q
	X=0	1	
y_2y_1	Y_2Y_1		
00	00	01	0
01	10	01	1
10	10	11	1
11	00	11	0



Dålig kodning –
Hamming Distance = 2
(korsande linjer)

Exempel: Serial Parity Generator

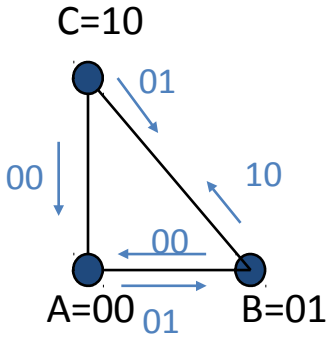
Pres state	Next State		Q
	X=0	1	
y_2y_1	Y_2Y_1		
00	00	01	0
01	11	01	1
11	11	10	1
10	00	10	0



Bra kodning
Hamming Distance = 1
(inga korsande linjer)

Icke stabila tillstånd

Present state	Next state $r_2\, r_1$ = 00	01	10	11	Output $g_2\, g_1$
A	(A)	B	C	–	00
B	A	(B)	C	(B)	01
C	A	B	(C)	(C)	10

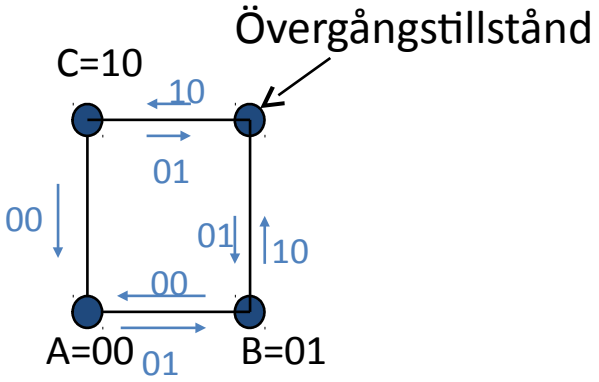


Dålig kodning

Vid övergången från B to C (eller C to B) är Hamming distansen 2!
Risk att man fastnar i ett ospecificerat tillstånd (med kod 11)!

Icke stabila tillstånd

- Lösning: Införandet av ett övergångstillstånd som säkerställa att man inte hamnar i ett odefinierat läge!

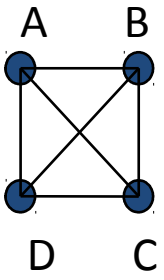


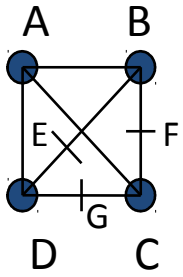
Bra kodning

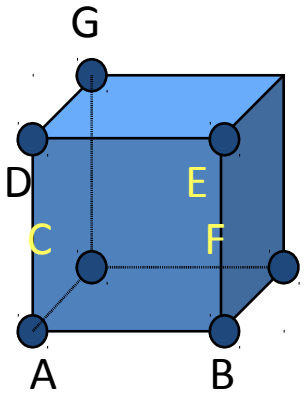
	Present state $y_2\,y_1$	Next state $r_2\,r_1$ = 00	01	11	10	Output $g_2\,g_1$
		$Y_2\,Y_1$				
A	00	(00)	01	–	10	00
B	01	00	(01)	(01)	11	01
D	11	–	01	–	10	dd
C	10	00	11	(10)	(10)	10

Extra tillstånd (fler dimensioner)

- Man kan öka antalet dimensioner för att kunna införa säkra tillståndsövergångar



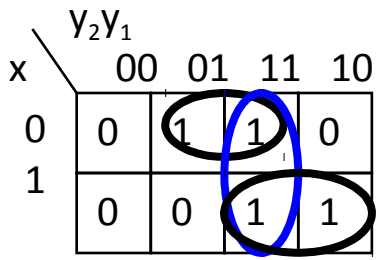




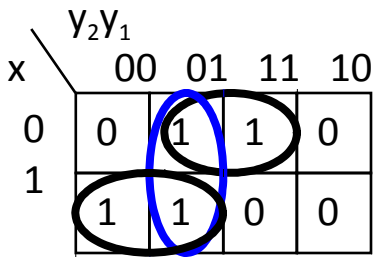
Om det inte på något sätt går att rita om diagrammet till HD=1 får man lägga till fler tillstånd genom att lägga till extra dimensioner. Man tar då närmsta större hyperkub och drar övergångarna genom tillgängliga icke stabila tillstånd.

Steg 4:
Skapa Karnaugh-diagram

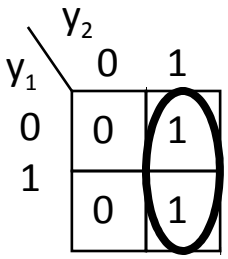
Pres state	Next State		Q
	X=0	1	
y_2y_1	Y_2Y_1		
00	00	01	0
01	11	01	1
11	11	10	1
10	00	10	0



$Y_2 = \bar{x}\, y_1 + y_2 y_1 + x\, y_2$



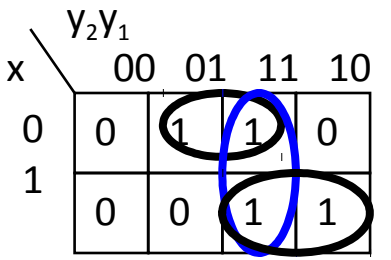
$Y_1 = \bar{x}\, y_2 + \bar{y}_2 y_1 + \bar{x}\, y_1$



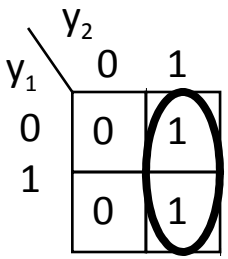
$Q = y_2$

De blå inringningarna är pga att undvika Hazard (se senare avsnitt)!

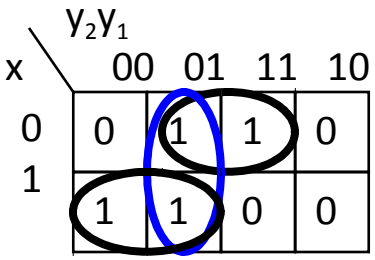
Steg 5. Färdig krets



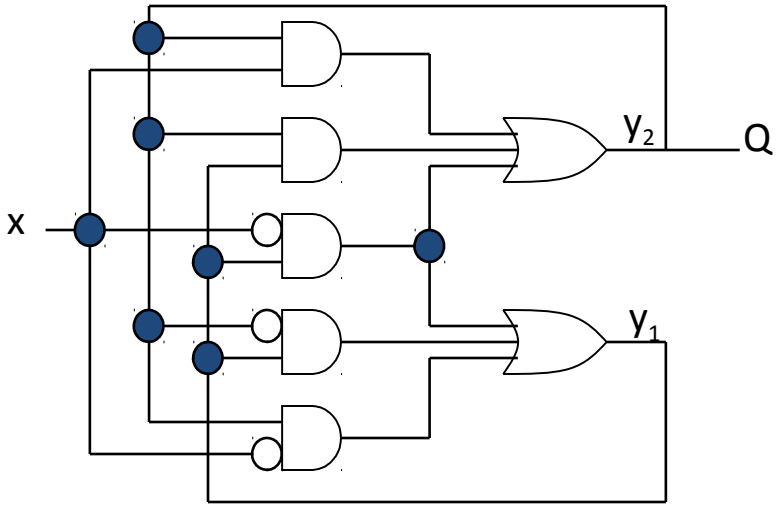
$Y_2 = \bar{x}\, y_1 + y_2 y_1 + x\, y_2$



$Q = y_2$



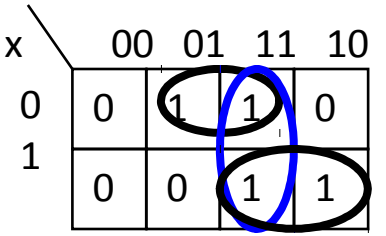
$Y_1 = \bar{x}\, y_2 + \bar{y}_2 y_1 + \bar{x}\, y_1$



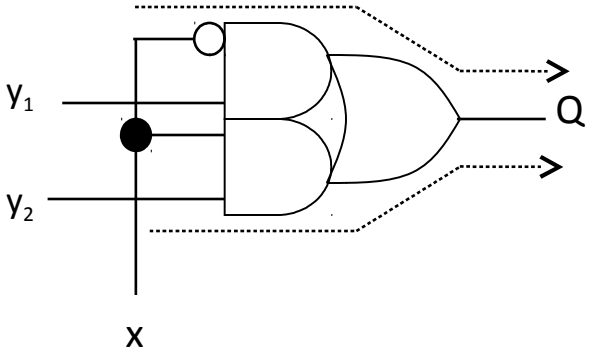
Vad är Hazard?

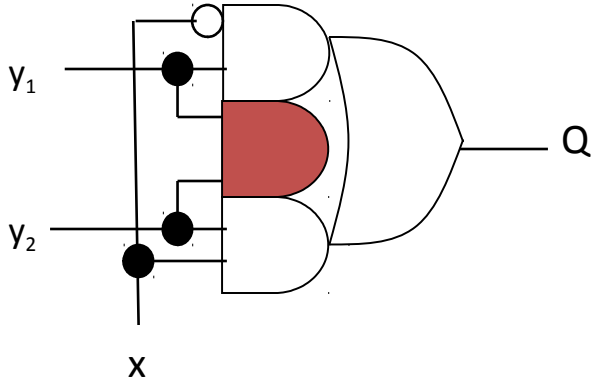
- Hazard är ett begrepp som innebär att det finns en fara för att utgångsvärdet inte är stabilt, utan att det kan blinka till vid vissa ingångskombinationer.
- Hazard uppkommer om det är olika långt från olika ingångar till en utgång, s.k., signal-kapplöpning.
- För att motverka detta måste man lägga till prim-implikanter för att täcka upp den farliga övergången.

Exempel på Hazard: Muxen



$Y_2 = \overline{x}\, y_1 + y_2 y_1 + x\, y_2$





Vid övergång från xy_2y_1=(111)->(011) kan utgången Q blinka till, eftersom vägen från x till Q är längre via den övre AND-grinden än den lägre (kapplöpning).

Tillståndsminimering

Asynkrona statemaskiner har många ”ospecifierade” positioner i flödestabellen som man kan utnyttja för att minimera antalet tillstånd.

Sannolikheten för att färre tillstånd leder till en enklare realisering är **hög** ***när det gäller asynkrona nät!***

Tillståndsminimering

Två steg:

Ekvivalens – ekvivalenta tillstånd. Samma steg som vid tillståndsminimering av synkrona sekvensnät, full flexibilitet finns kvar.

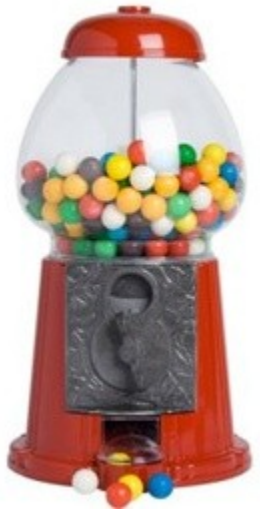
Kompatibilitet – kompatibla tillstånd blir olika för Moore-kompatibel eller Mealy-kompatibel realisering, de val man nu gör påverkar den fortsatta flexibiliteten.

Tillståndsminimering

- **Procedur för minimering av antalet tillstånd**

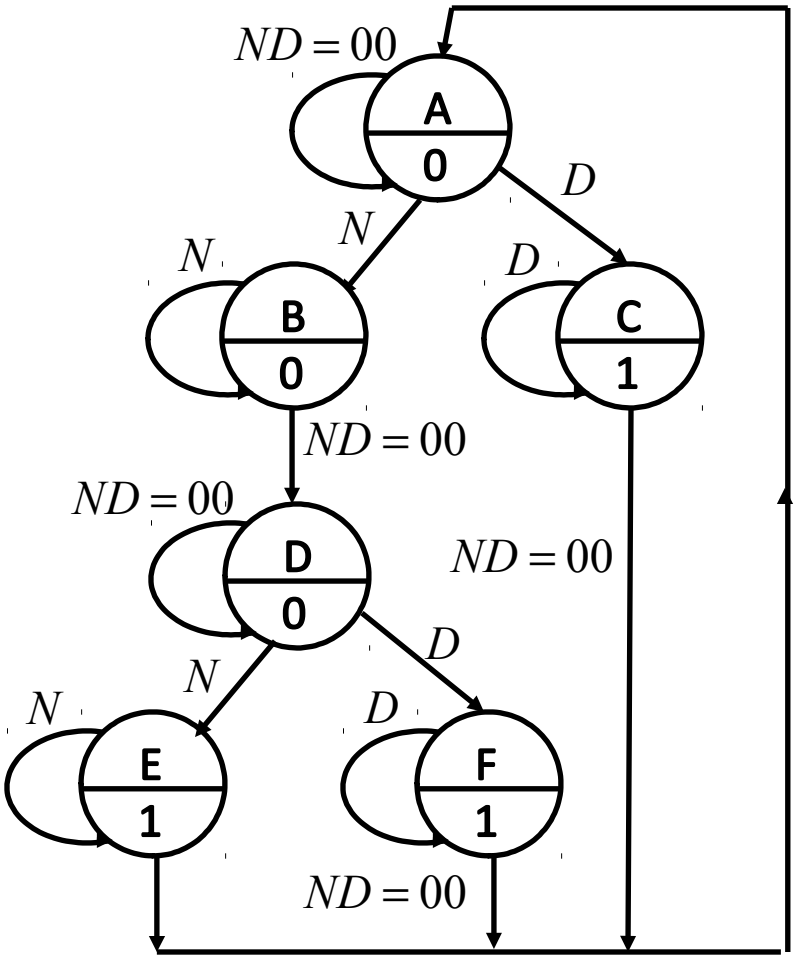
1. Bilda **ekvivalensgrupper**.
   För att vara i samma grupp ska följande gälla:
   - Utgångar måste ha samma värde
   - Stabila tillstånd måste finnas på samma plats (kolumn)
   - Don't cares för next state måste finnas på samma plats (kolumn)
2. Minimera ekvivalensgrupperna (state-reduktion)
3. Bilda **sammanslagningsdiagram**, olika för Mealy eller för Moore.
4. Slå ihop kompatibla tillstånd i grupper. Minimera samtidigt antalet grupper. Varje tillstånd får endast ingå i en grupp.
5. Konstruera den reducerade flödestabellen genom att slå samman raderna i de valda grupperna
6. Repetera steg 3-5 för att se om fler minimeringar kan göras

Godisautomat (BV sid 610)



- Godismaskinen har två ingångar:
  - ***N***: Nickel (5 cent)
  - ***D***: Dime (10 cent)
- En godisbit kostar 10 cent
- Maskinen returnerar inga pengar om det finns 15 cent i automaten (en godisbit returneras)
- Utgången ***z*** är aktiv när det finns tillräckligt med pengar för en godisbit

Tillståndsdiagram, Flödestabell



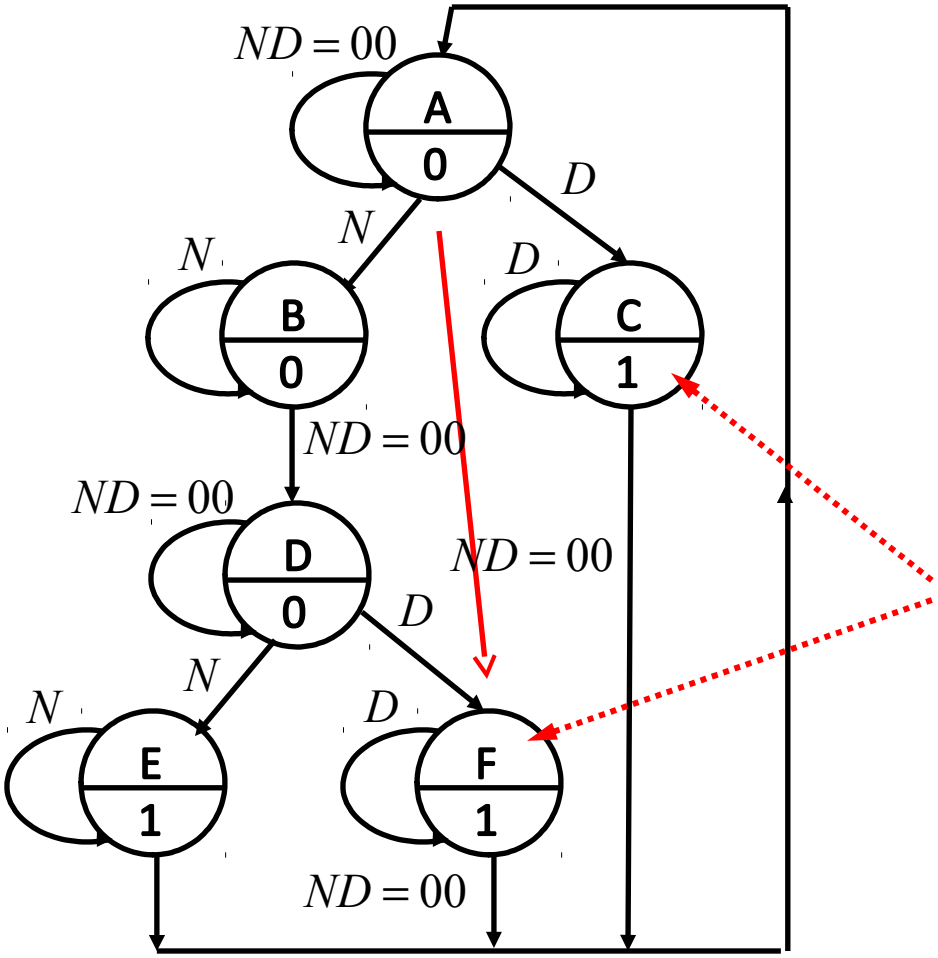
- *Inga ”dubbeländringar” av insignalerna!*
- *Tiden ny går inte att stoppa i samtidigt!*

Pres state	Next State X=00	01	10	11	Q
A	(A)	B	C	-	0
B	D	(B)	-	-	0
C	A	-	(C)	-	1
D	(D)	E	F	-	0
E	A	(E)	-	-	1
F	A	-	(F)	-	1

(X = ND, Q = z)

En flödestabell som endast innehåller ett stabilt tillstånd per rad kallas för en ***primitiv flödestabell***.

Tillståndsminimering



Tillståndsminimering innebär att **två** tillstånd kan vara ekvivalenta, och i så fall ersättas av **ett** tillstånd för att förenkla tillståndsdiagrammet, och nätet.

Man kan lätt inse att tillstånd **C** och **F** kommer att kunna ersättas med **ett** tillstånd eftersom godis *alltid* ska matas ut efter en Dime oavsett tidigare tillstånd.

Bilda/minimera ekvivalensgrupper

1. **Bilda ekvivalensgrupper. För att vara i samma grupp ska följande gäller:**
   - Utgångar måste ha samma värde
   - Stabila tillstånd måste finnas på samma plats (kolumn)
   - Don’t cares för next state måste finnas på samma plats (kolumn)
2. **Minimera ekvivalensgrupperna (state reduction).**

Ekvivalensgrupper

Pres state	Next State				Q
	X=00	01	10	11	
A	(A)	B	C	-	0
B	D	(B)	-	-	0
C	A	-	(C)	-	1
D	(D)	E	F	-	0
E	A	(E)	-	-	1
F	A	-	(F)	-	1

($X = ND$, $Q = z$)

Tillstånden delas i block efter utsignal.
ABD har utsignal **0**, **CEF** har utsignal **1**.
P_1 = (ABD)(CEF)

Stabila tillstånd måste finnas för samma insignal (kolumn), don't care måste finnas för samma kolumn.

AD har stabilt tillstånd för 00. **B** har stabilt för 01. **CF** har stabilt tillstånd för 10. **E** har stabilt för 01. **AD** och **CF** har don't care för motsvarande insignaler.

P_2 = (AD)(B)(CF)(E)

Slå ihop ekvivalensgrupper

*Två rader kan ”slås ihop” om det **inte** innebär någon **konflikt** för deras efterföljartillstånd*

Pres state	Next State X=00	01	10	11	Q
A	(A)	B	C	-	0
B	D	(B)	-	-	0
C	A	-	(C)	-	1
D	(D)	E	F	-	0
E	A	(E)	-	-	1
F	A	-	(F)	-	1

(X = ND, Q = z)

P_2=(AD)(B)(CF)(E)

P_3=(A)(D)(B)(C)(E)

P_4=P_3.

Raderna **C** och **F** kan slås ihop med ny samlingsbeteckning **C**, medan **A** och **D** som har efterföljare i olika grupper *inte* kan slås ihop.

$CF_{00} \rightarrow (A)$

$CF_{01} \rightarrow (--)$

$CF_{10} \rightarrow (CF)$

$CF_{11} \rightarrow (--)$

$AD_{00} \rightarrow (AD)$

$AD_{01} \rightarrow (B)(E)$

$AD_{10} \rightarrow (CF)$

$AD_{11} \rightarrow (--)$

Resulterande flödestabell

Pres state	Next State X=00	01	10	11	Q
A	(A)	B	C	-	0
B	D	(B)	-	-	0
C	A	-	(C)	-	1
D	(D)	E	C	-	0
E	A	(E)	-	-	1

Kompatibilitetsgrupper

3. Bilda sammanslagningsdiagram ***antingen*** för **Mealy** eller **Moore**
4. Slå ihop kompatibla tillstånd i grupper. Minimera samtidigt antalet grupper. Varje tillstånd får endast ingå i en grupp.
5. Konstruera den reducerade flödestabellen genom att slå samman raderna i de valda grupperna
6. Repetera steg 3-5 för att se om fler minimeringar kan göras

Sammanslagningsregler

- **Två tillstånd är ”kompatibla” och kan slås ihop om följande gäller**
  1. åtminstone ***ett*** av följande villkor gäller för alla ingångskombinationer
     - både S_i och S_j har samma följdtillstånd, eller
     - både S_i och S_j är stabila, eller
     - följdtillståndet av S_i eller S_j eller båda är ospecifierade
  2. Sedan gäller följande om man vill konstruera en Moore-kompatibel automat
     - både S_i och S_j har samma **utgångsvärde** (gäller ju inte när man konstruerar en Mealy-kompatibel automat)

Sammanslagningsdiagram

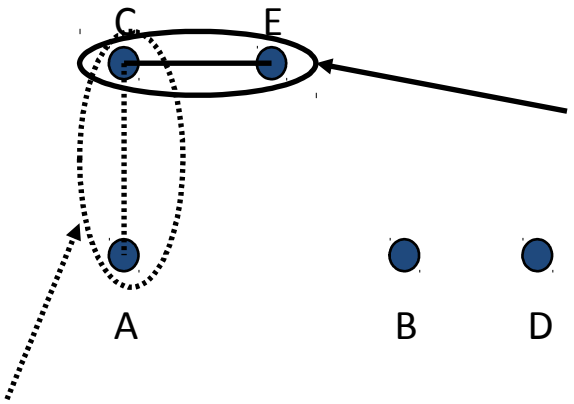
Resulterande flödestabell

Pres state	Next State X=00	01	10	11	Q
→A	(A)	B	C	-	0
B	D	(B)	-	-	0
→C	A	-	(C)	-	1
D	(D)	E	C	-	0
→E	A	(E)	-	-	1

Varje rad blir en punkt i kompatibilitetsgrafen.

- *När det finns flera möjligheter …*

Kompatibilitetsgraf



Moore-kompatibla

C(1):	A–**C**–
E(1):	A**E**––

Mealy-kompatibla: I tillstånd **A** (X = 00) är utgången 0, i tillstånd **C** är utgången 1

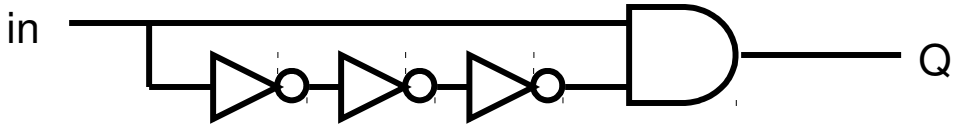
C(1):	A–**C**–
A(0):	**A**BC–

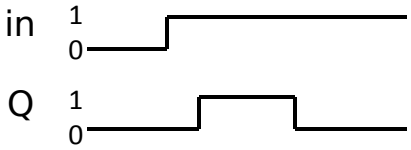
Hasard: Spikar på signalvärden

- När man konstruerar asynkrona kretsar så kan det händer att man få spikar (glitches) på signalvärden
- Detta beror på att olika signalvägar har olika fördröjningstider
- Fenomenet kallas för *hasard* (hazard) och kan elimineras med noggrann konstruktion

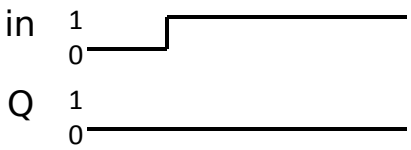
Snabbfråga

Vilket tidsdigaram motsvarar bäst den signal som genereras av följande grindnät vid stigande flank?

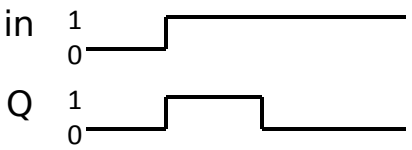




Alt: A



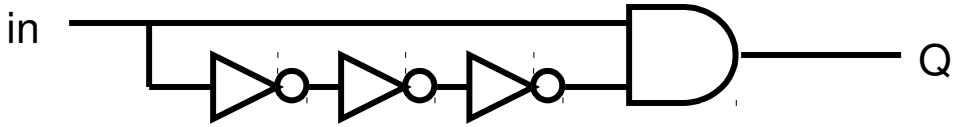
Alt: B

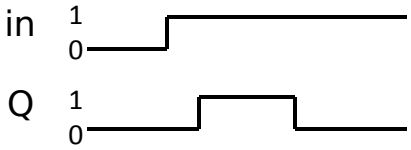


Alt: C

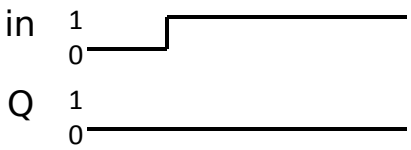
Snabbfråga

Vilket tidsdigaram motsvarar bäst den signal som genereras av följande grindnät vid stigande flank?

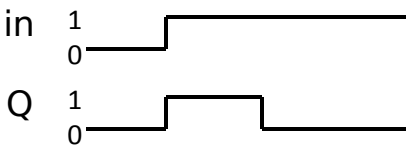




Alt: A

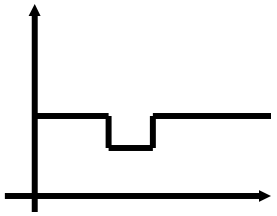


Alt: B

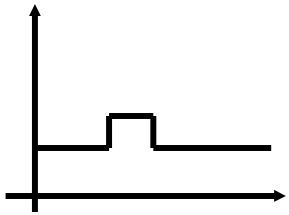


Alt: C

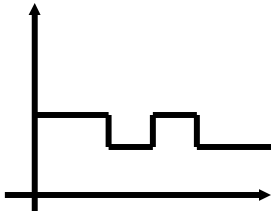
Olika typer av Hasard



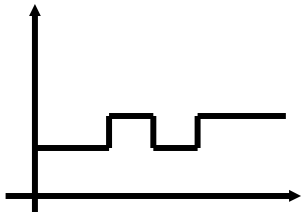
Statisk 1 ⟶ 1



Statisk 0 ⟶ 0

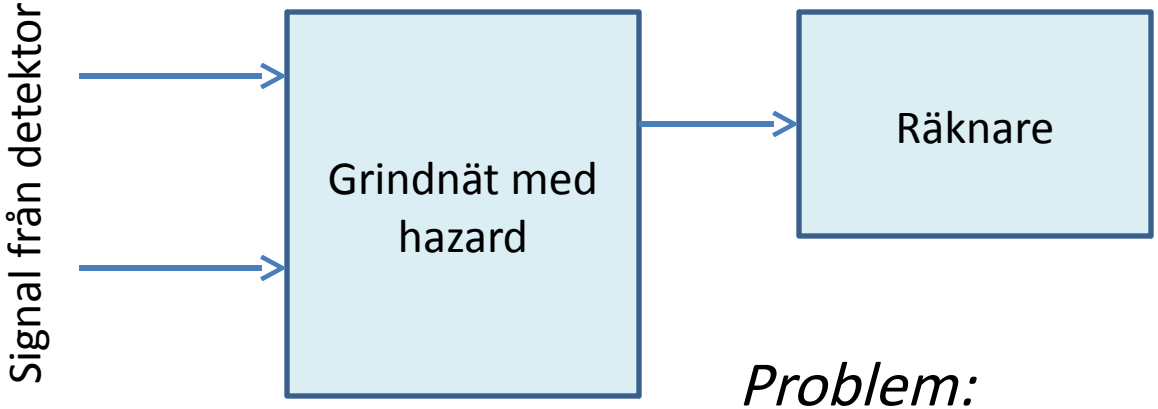


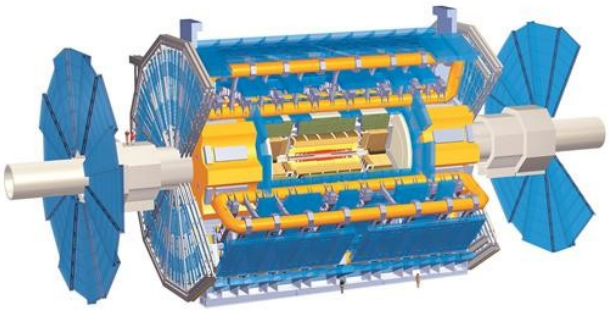
Dynamisk 1 ⟶ 0



Dynamisk 0 ⟶ 1

Exempel på möjliga problem



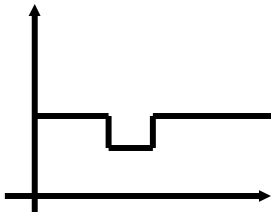


ATLAS particle detector

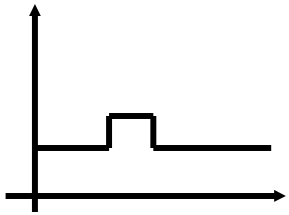
Problem:
Om grindnätet genererar spikar kommer räknaren visa felaktigt antal detekterade partiklar

I askynkrona sekvensnät måste vi undvika hazard

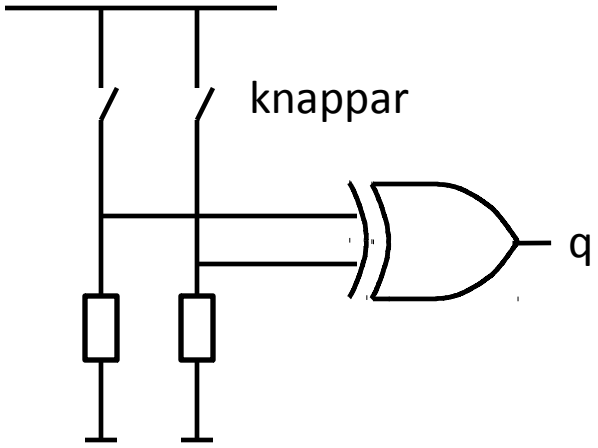
Statisk Hasard



Statisk 1 ⟶ 1



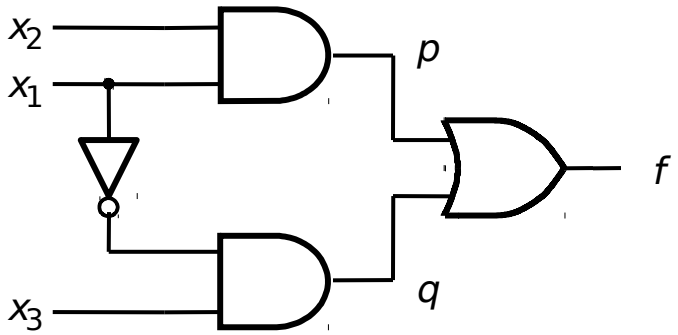
Statisk 0 ⟶ 0



Exempel:
Trycker vi ned
båda knapparna
genereras alltid en kort
puls på ugången eftersom
vi inte kan trycka ned knapparna
EXAKT samtidigt

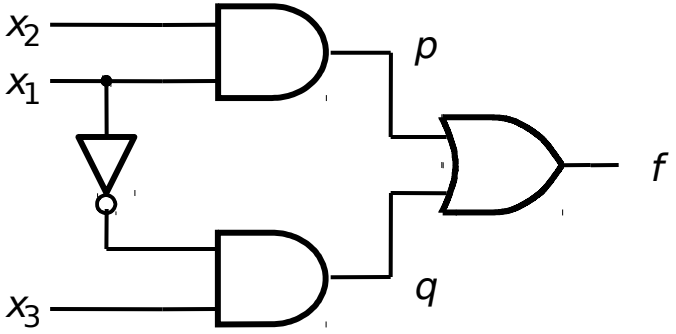
Exempel Statisk Hasard

- Hasard kan uppträda vid nedanstående krest om vid övergången av $x_3x_2x_1$ från 111 => 110

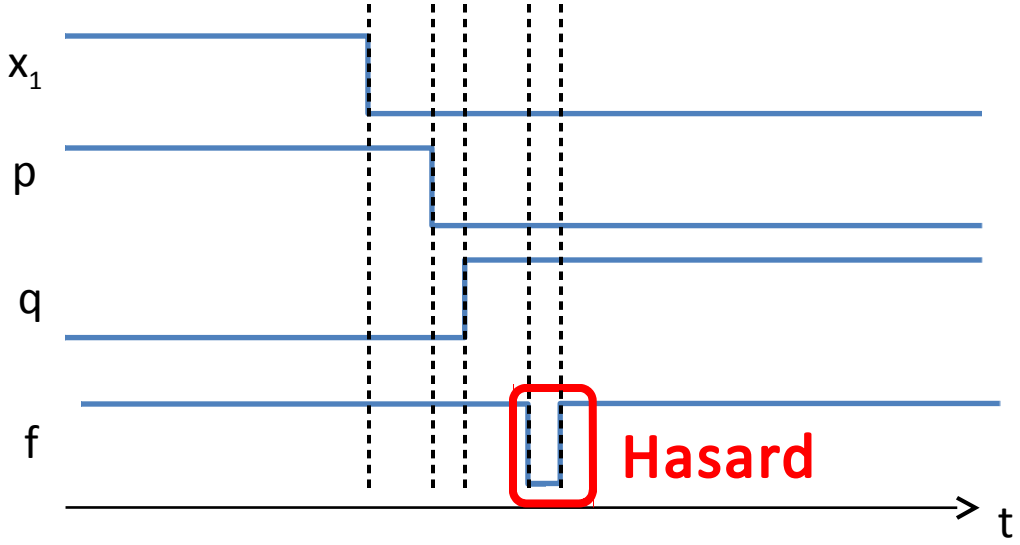


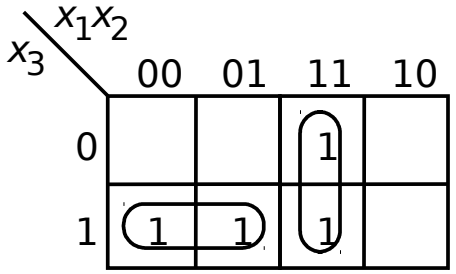
$$f = x_1x_2 + \bar{x}_1x_3$$

Tidsdiagram



$$f = x_1 x_2 + \bar{x}_1 x_3$$

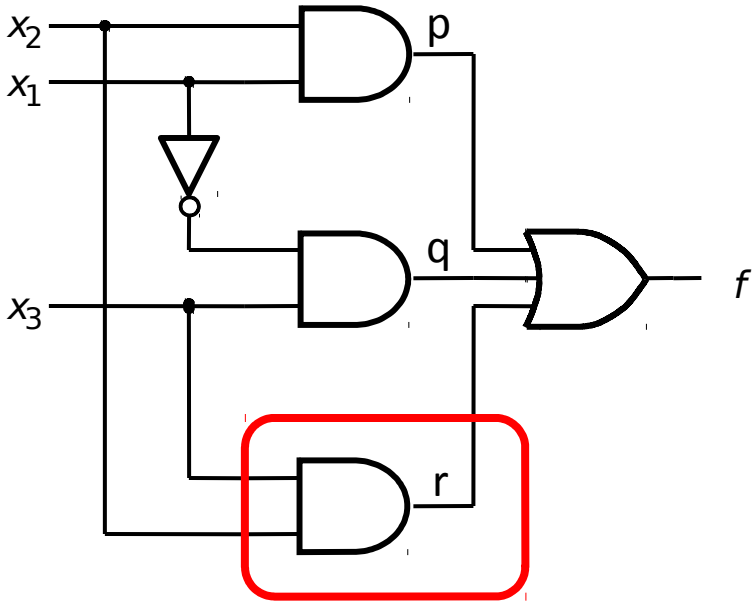


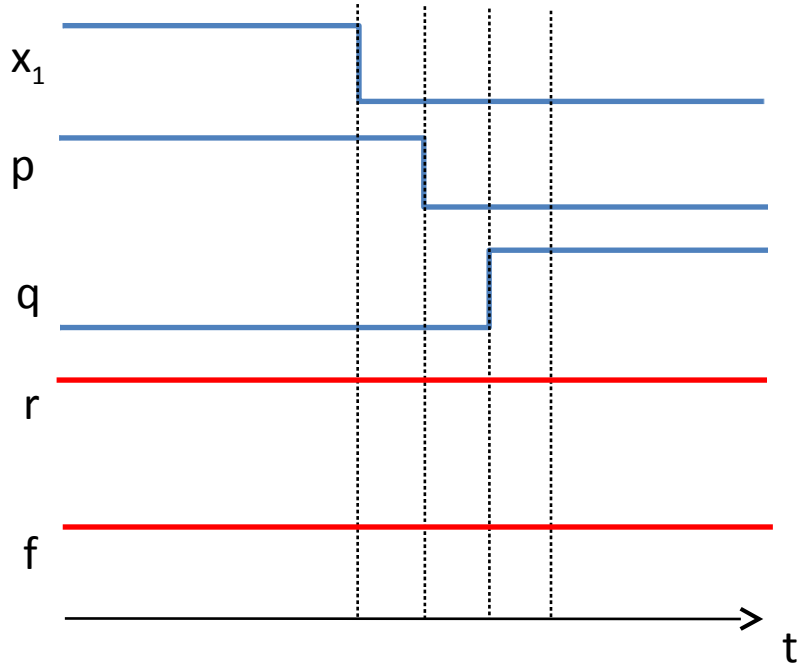


Hur undviker man statisk Hasard?

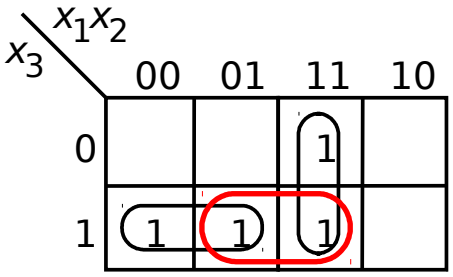
- Möjligheten för statisk hasard finns om två intill-liggande 1:or inte är täckta med en egen produkter i Sum of Products
- Därmed kan man ta bort risken för statisk hazard genom att lägga till en inringningar så att alla intill-liggande 1:or är täckta med en egen inringning

Hasardfri krets (SOP)





Hasard-Cover



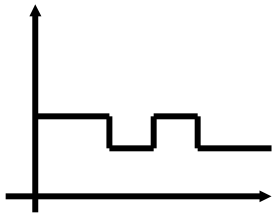
$$f = x_1x_2 + \bar{x}_1x_3 + x_2x_3$$

Statisk Hasard i en POS-circuit

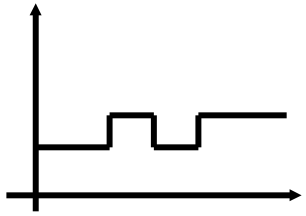
- Har man en Product of Sums (POS)-implementering så måste man se till att man alla bredvidliggande 0:or är täckta av en egen produktterm

Dynamisk Hasard

- En dynamisk hasard orsaker flera spikar på utgången
- En dynamisk hasard orsakas av kretsens struktur



Dynamisk 1 ⟶ 0



Dynamisk 0 ⟶ 1

Exempel: Dynamisk Hasard

- Följande ekvation osaker ingen hasard om man implementera det som AND-OR-struktur

$$f = x_1\bar{x}_2 + \bar{x}_3 x_4 + x_1 x_4$$

Exempel: Dynamisk Hasard

- Följande ekvation osaker ingen hasard om man implementera det som AND-OR-struktur

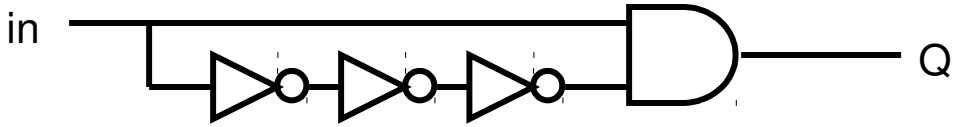
$$f = x_1\bar{x}_2 + \bar{x}_3 x_4 + x_1 x_4$$

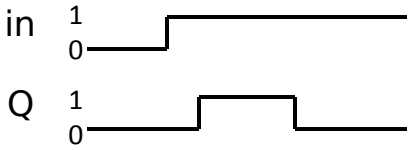
Hur kan man undvika Dynamisk Hasard?

- Dynamisk hasard kan undvikas med två-nivås-logik
- Man måste vid minimering se till att kretsen är fri från statisk hasard, då finns det inte heller en dynamisk hasard!

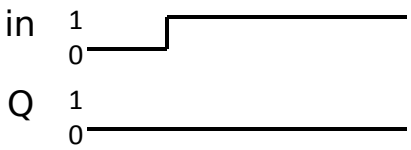
Snabbfråga

Vilket tidsdigaram motsvarar bäst den signal som genereras av följande grindnät vid stigande flank?

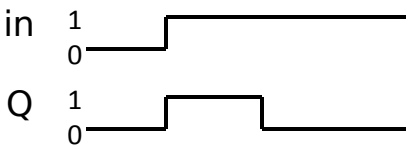




Alt: A



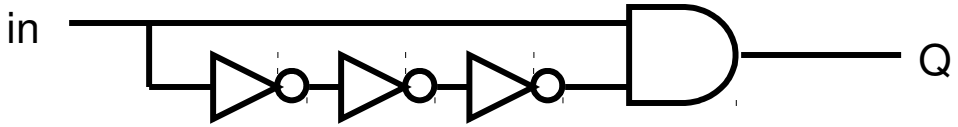
Alt: B

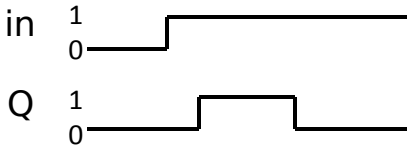


Alt: C

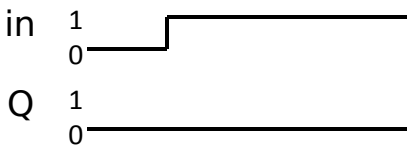
Snabbfråga

Vilket tidsdigaram motsvarar bäst den signal som genereras av följande grindnät vid stigande flank?

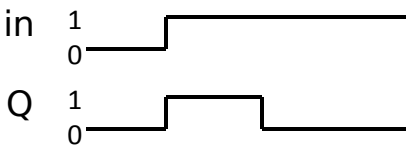




Alt: A



Alt: B

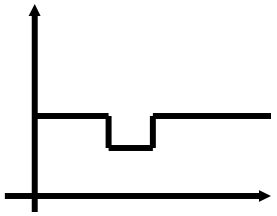


Alt: C

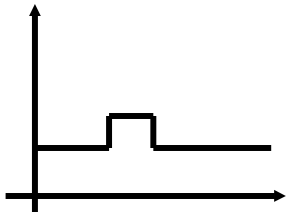
När ska man tänker på hasard?

- I ett *asynkront sekvensnät* måste avkodaren för nästa-tillstånd vara hasardfri!
  - Annars kan man hamnar i ett inkorrekt tillstånd
- För *kombinatoriska kretsar* är hasard inte viktigt eftersom utgången kommer efter ett kort tag stabiliserar sig
- I ett *synkront sekvensnät* är hasard inget problem, så länge man respekterar setup- och hold-tider (under dessa tider får hasard inte uppträda!)

Undvik Hasard!

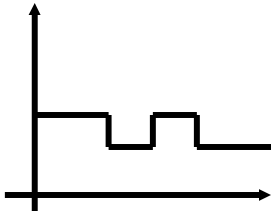


Statisk 1 ⟶ 1

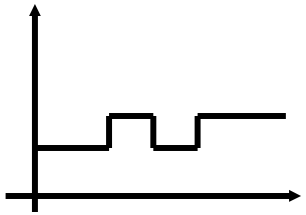


Statisk 0 ⟶ 0

Statisk hasard orsakas av utelämnade prim-implikanter



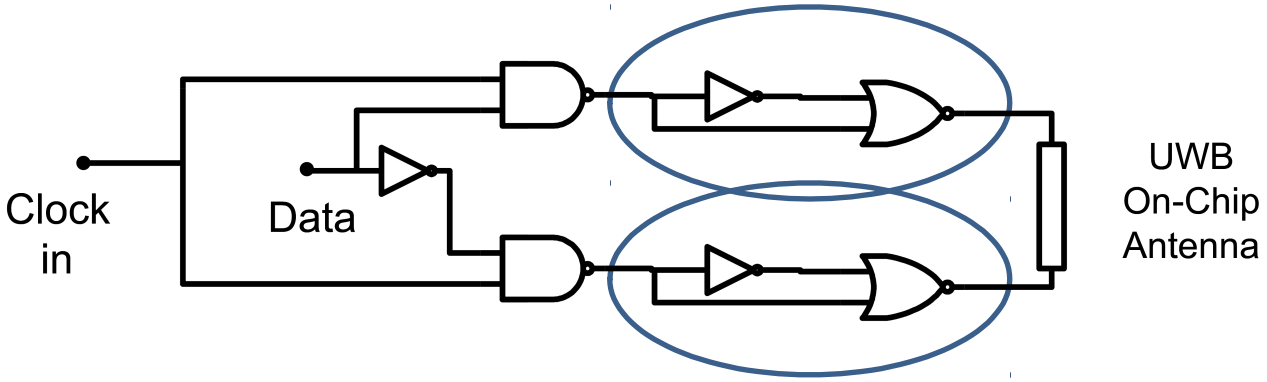
Dynamisk 1 ⟶ 0

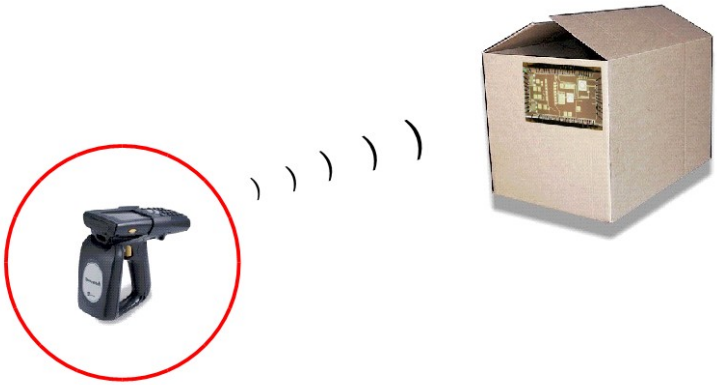


Dynamisk 0 ⟶ 1

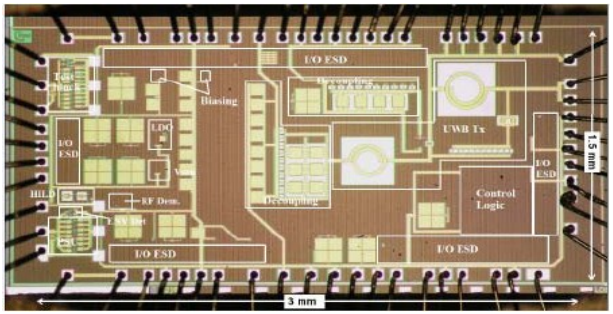
Dynamisk hasard kan uppstå när man implementera kretsar med flernivåslogik. Två- nivåslogikkretsar som är fria från statisk hasard är också fria från dynamisk hasard.

Exempel på önskvärd hazard





Exempel: Kommunikation med Pulsad Ultra Wideband



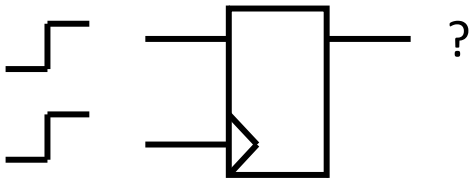
Chipfoto

Spikar på utgången i ett asynkront sekvensnät

Pres state	Next State		Q
	X=0	1	
y_2y_1	Y_2Y_1		
00	00	01	0
01	00	11	1
11	01	10	0
10	11	10	1

Man kan få utgångsspikar I ett asynkront sekvensnät när man byter från ett stabilt tillstånd till ett annat genom att passerar flera instabila tillstånd (Fenomenet är ingen hasard!).

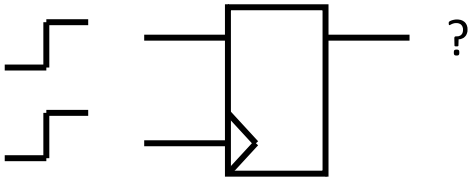
Metastabilitet



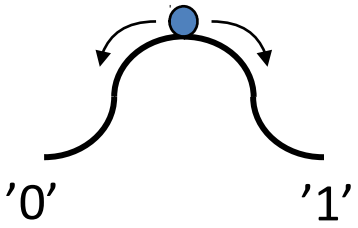
Om Clk och D slår om samtidigt, vilket värde får då Q?

Metastabilitet (forts.)

- Denna instabilitet varar tills transistorerna i återkopplingen behagar gå åt ena eller andra hållet – men det kan ta tid, och tiden beror bla på hur nära $V_{DD}/2$ som låsningen skedde.
- Man kan likna situationen vid en boll som ligger på toppen en kulle eller en penna som balanserar på sin spets. Minsta störning kommer att få bollen eller pennan att falla åt ena eller andra hållet.

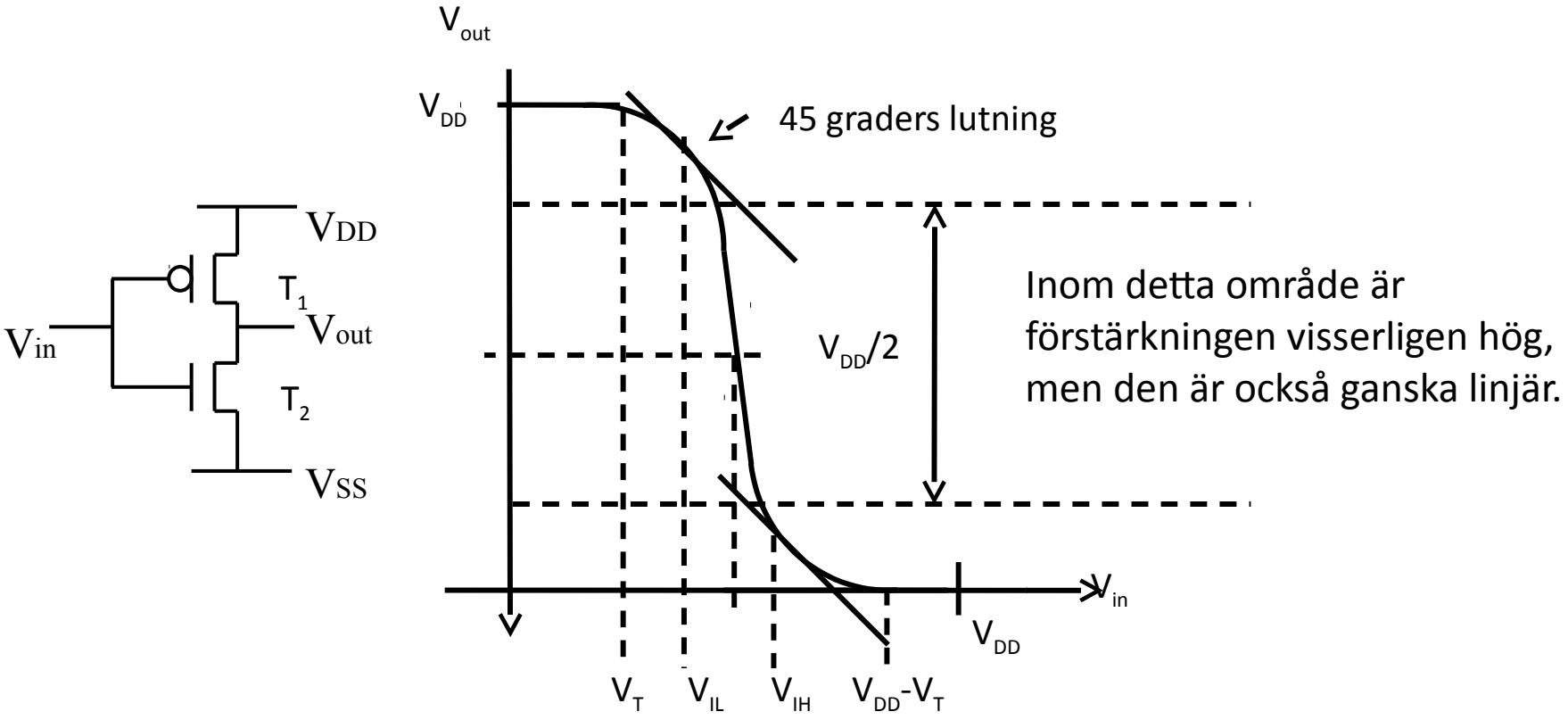


Om Clk och D switchar samtidigt, vilket värde får då Q?



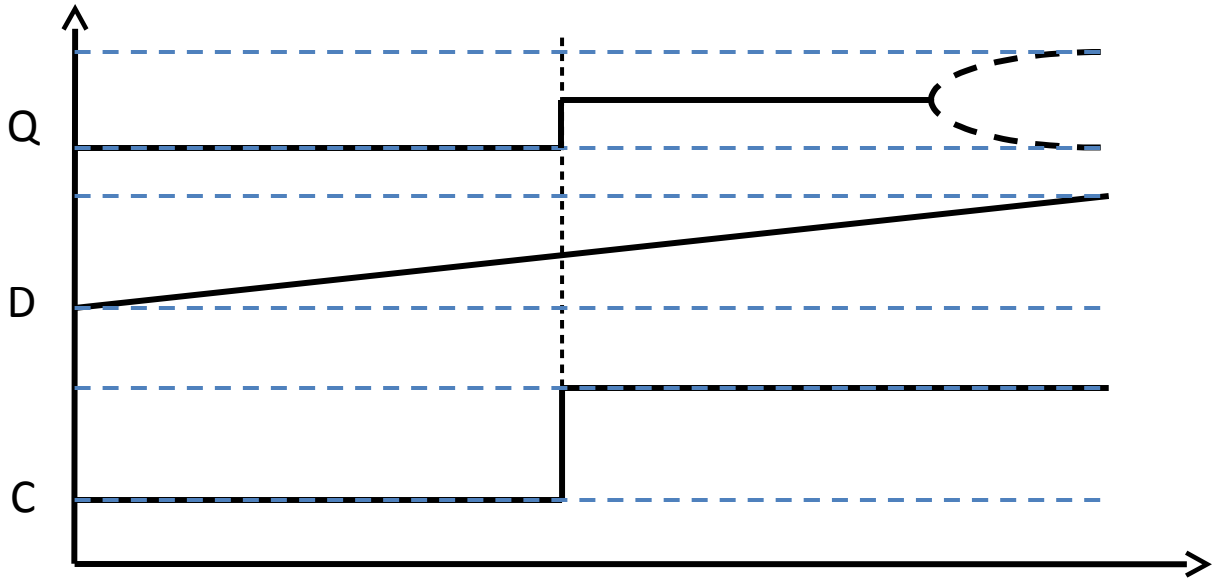
På vilken sida kommer bollen att trilla ner?

Inverterarens överföringsfunktion



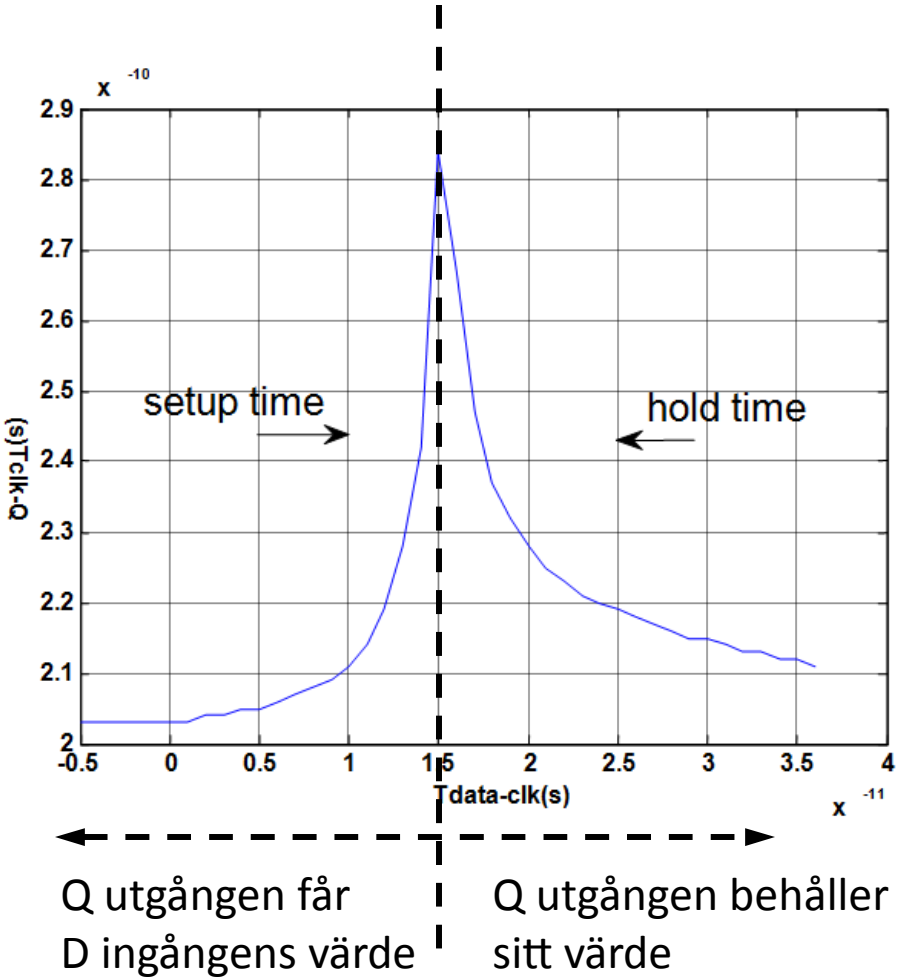
För att förstå metastabilitet krävs att grindar analyseras som analoga byggblock

Om metastabilitet...



För att förstå vad metastabilitet innebär så kan vi tänka oss att insignalen D till latchen är väldigt belastad och därmed slår om mycket långsamt i förhållande till klockan. Antag vidare att klock-signalen C slår om precis när D är vid $V_{DD}/2$. Då låser sig latchen vid det spänningsvärde som råkar finnas på D. Efter en tid slår latchen om till antingen ’1’ eller ’0’.

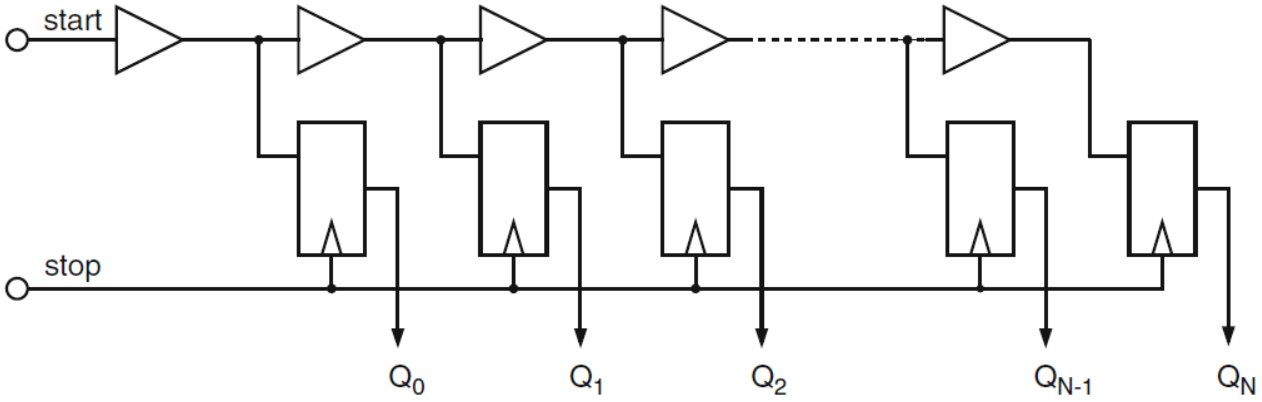
Exempel Setup and Hold time

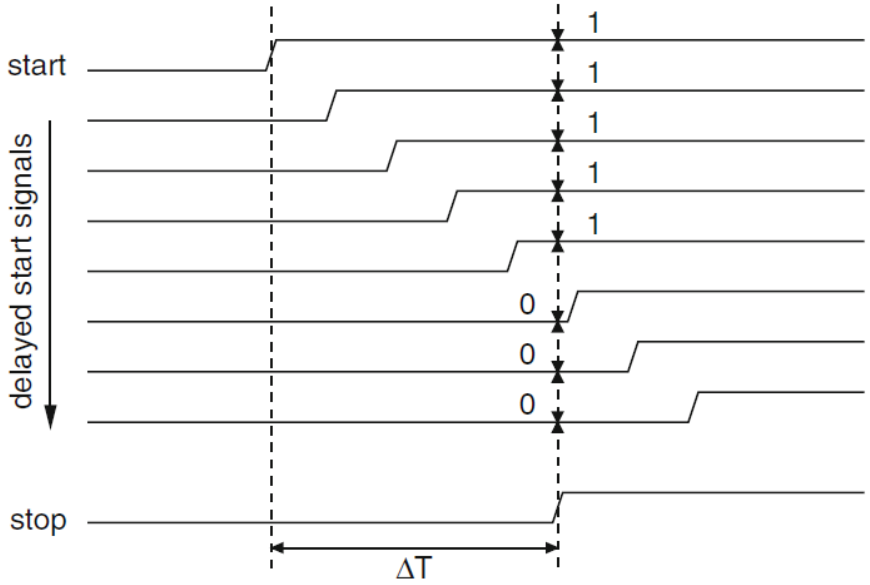


Utgången stabiliseras
alltid till ett logiskt värde,
men det tar längre
tid om setup och hold time
är litet.

På grund av process variationer
är det också osäkert exakt
vid vilken tidpunkt övergången
mellan setup och hold sker.

Exempel: Time to Digital converter



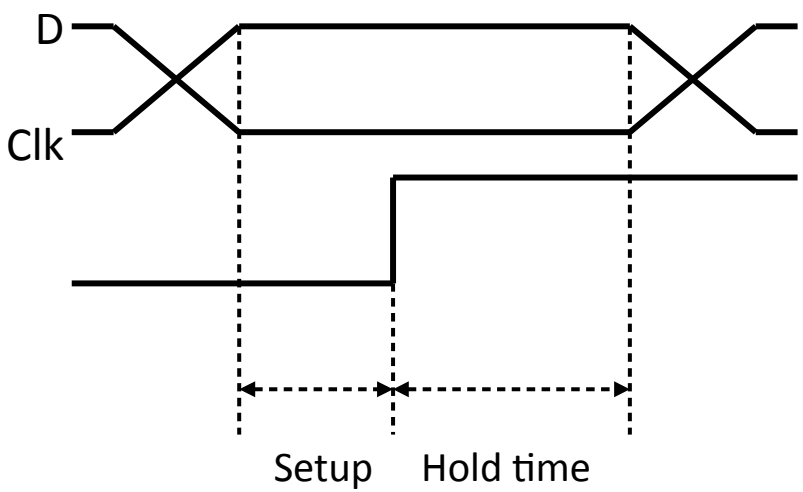


Exempel vågform

Används för att detektera tidsskillnad mellan två signaler, tex vid frekvenslåsning och i avståndsmätning

Setup and Hold time (= metastabilitets-skydd)

- För att undvika samtidigt omslag/switchning, så måste setup and hold times garanteras:



Setup time är den tid D måste vara stabil innan Clk ändrar värde
Hold time är den tid D måste vara stabil efter Clk har ändrat värde

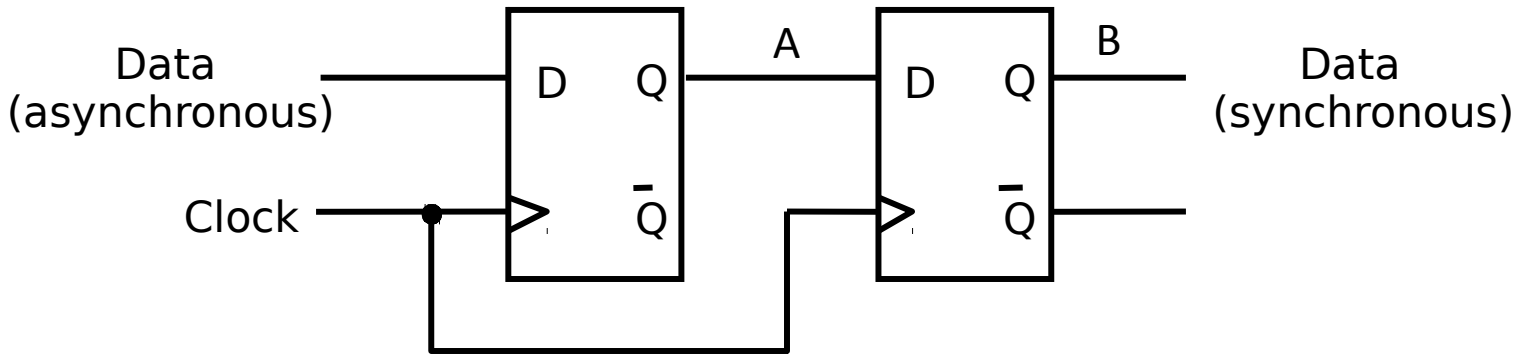
Om Setup and Hold times är uppfyllda, så kommer vippan (Flip-flop) att garanterat bete sig snällt/deterministiskt!

Asynkrona ingångar

- Dessvärre kan vi inte alltid garantera att en ingång är stabil under hela setup- och hold-tiden
- Antag att du kopplar in en tryckknapp på D-ingången av en vippa
  - Användaren kan trycker knappen när som helst, även under setup- och hold-tiden!
  - Risken är att vippan hamnar i ett metastabilt tillstånd!

Synkronisering av asynkrona utgångar

- För att synkronisera asynkrona ingångar användar man en extra vippa på ingången
- Den första vippans utgång (A) kan hamnar i ett metastabilt läge
- Men om klockperioden är tillräckligt lång, så kommer den att stabiliseras innan nästa klockflank, så att B inte hamnar I ett metastabilt läge!



Sammanfattning

- Asynkrona tillståndsmaskiner
  - Bygger på analys av återkopplade kombinatoriska nät
  - Alla vippor och latchar är asynkrona tillståndsmaskiner
- En liknande teori som för synkrona tillståndsmaskiner kan appliceras
  - Bara en ingång eller tillståndsvariabel kan ändras åt gången!
  - Man får även ta hänsyn till kapplöpningsproblem

Gyllene regeln