Digital Design IE1204

(Föreläsningsbilder av William Sandqvist)

F10 Asynkrona sekvensnät (modul 3)

Carl-Mikael Zetterling
bellman@kth.se

Asynkrona sekvensmaskiner

- En asynkron sekvensmaskin är en sekvensmaskin *utan vippor*
- Asynkrona sekvensmaskiner bygger på återkopplade kombinatoriska grindnätverk

<u>Vid analys antar man</u>: Endast EN signal i taget i grindnätet kan förändra sitt värde vid någon tidpunkt

Gyllene regeln

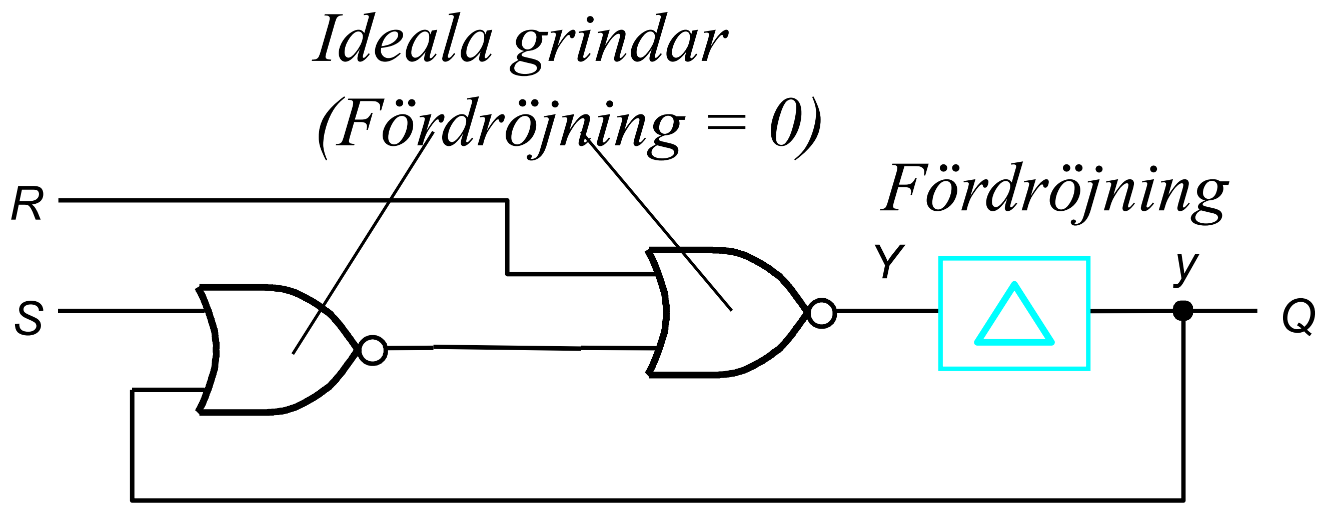
Asynkron tillståndsmaskin

Asynkrona tillståndsmaskiner används då det är nödvändigt att bibehålla ett tillstånd, men då det inte finns någon klocka tillgänglig.

- Alla vippor och latchar är själva asynkrona tillståndsmaskiner
- De är användbara för att synkronisera händelser i situationer där metastabilitet är/kan vara ett problem

SR-latchen med NOR-grindar

För att analyserar beteendet av en asynkron krets så antar man ideala grindar och sammanfattar all fördröjning till ett enda block med fördröjningen **Δ**.

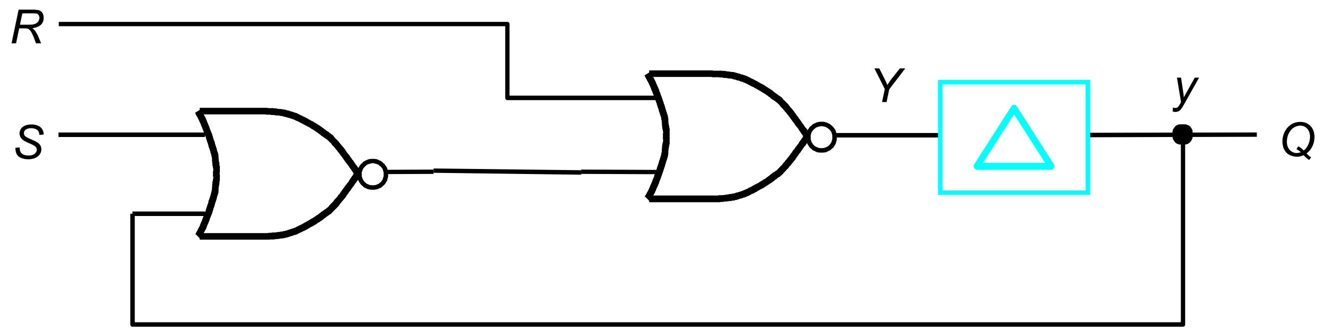


Analys av det asynkrona sekvensnätet

Genom att vi har ett **fördröjningsblock** kan vi betrakta

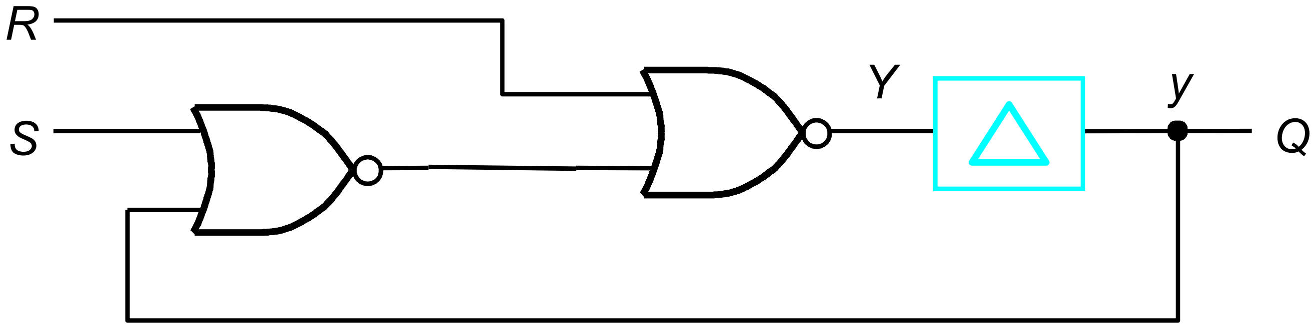
y som nuvarande tillstånd

Y som nästa tillstånd



Tillståndsfunktion

Därmed kan vi ta fram ett funktionssamband hur nästa tillstånd ***Y*** beror på insignalerna ***S*** och ***R*** samt nuvarande tillstånd ***y***



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

Tillståndstabell

Ibland används binärkodsordning

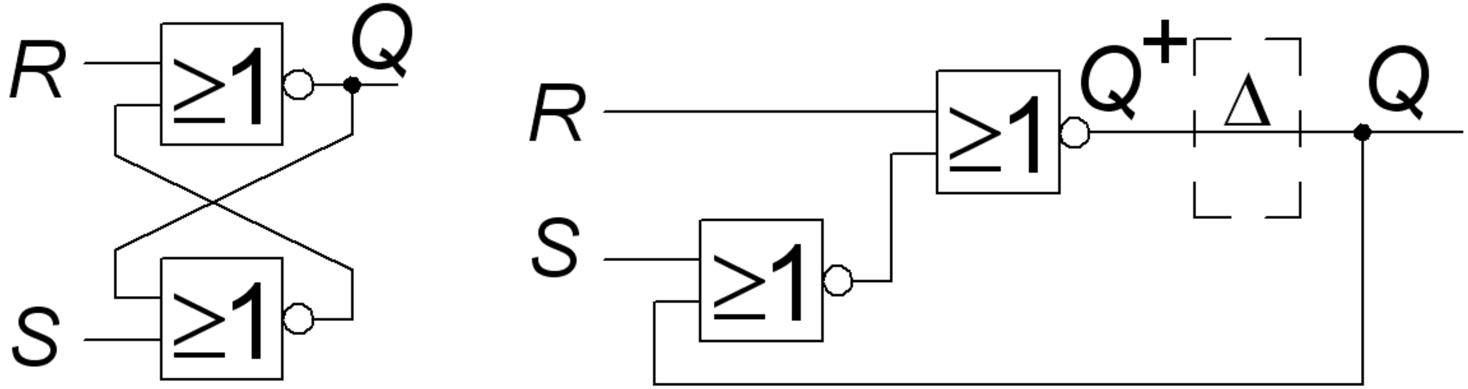
Från tillståndsfunktion till sanningstabell

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	$1 = \overline{0 + (\overline{1 + 0})}$
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	$0 = \overline{1 + (\overline{1 + 1})}$

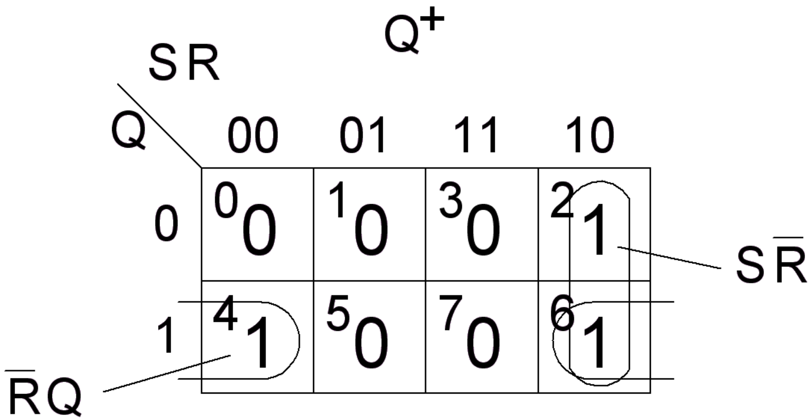
$$Y = \overline{R + (\overline{S + y})}$$

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

SR analys



$$Q^{+} = \overline{R+\overline{S+Q}} = \overline{R}\cdot\overline{\overline{(S+Q)}} = \overline{R}\cdot(S+Q) = S\overline{R}+\overline{R}Q$$



Nuvarande tillstånd *Q*	**Nästa tillstånd *Q*+**			
	Insignaler SR			
	00	01	11	10
0	0	0	0	1
1	1	0	0	1

För binär ordning

Stabila tillstånd

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

- Eftersom vi inte har vippor utan bara kombinatoriska kretsar kan en tillståndsändring medföra ytterligare tillståndsändringar
- Ett tillstånd är
  - stabilt om $Y(t) = y(t + \Delta)$
  - instabil om $Y(t) \neq y(t + \Delta)$

$\boxed{Y = y}$ stabilt

Exitationstabell

Den asynkrona kodade tillståndstabellen kallas för **Excitationstabell**

De stabila tillstånden (de med next state = present state) ”ringas in”

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

$$Y = y$$

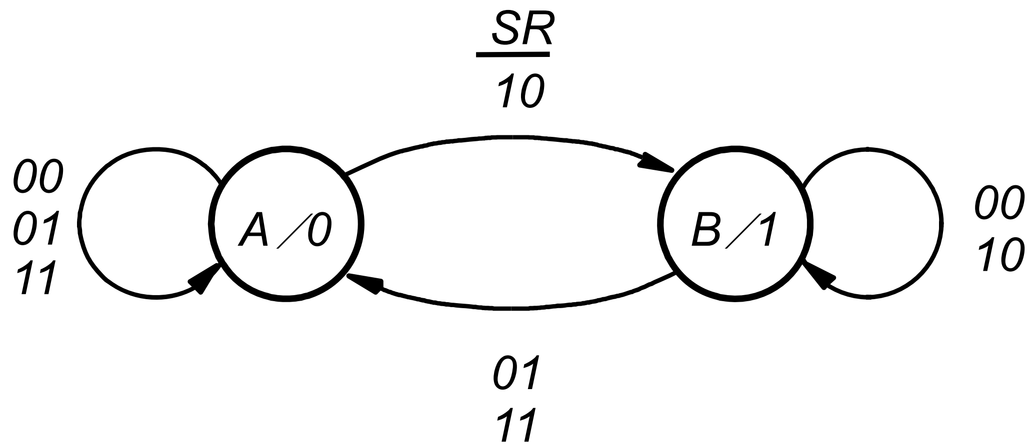
Terminologi

När man arbetar med asynkrona sekvensnät så används det en annan terminologi

- Den asynkrona okodade tillståndstabellen kallas $\boxed{\textbf{flödestabell}}$

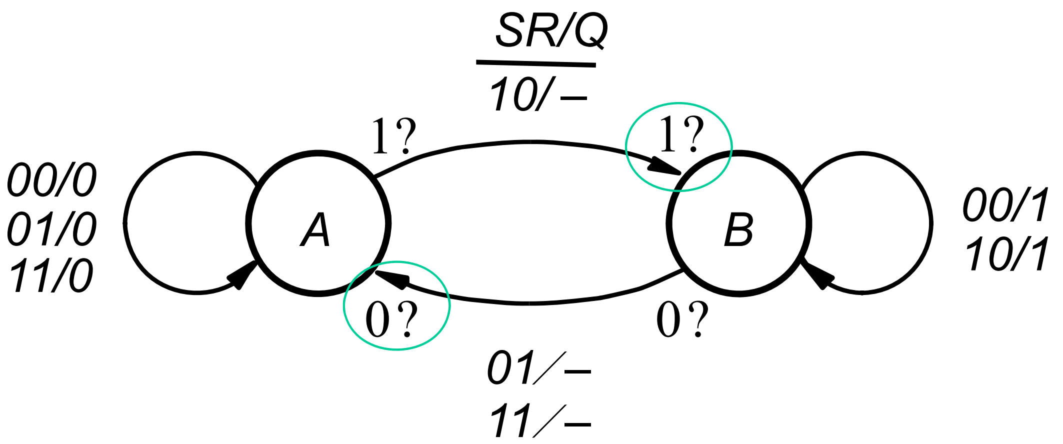
Flödestabell och Tillståndsdiagram (Moore)

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



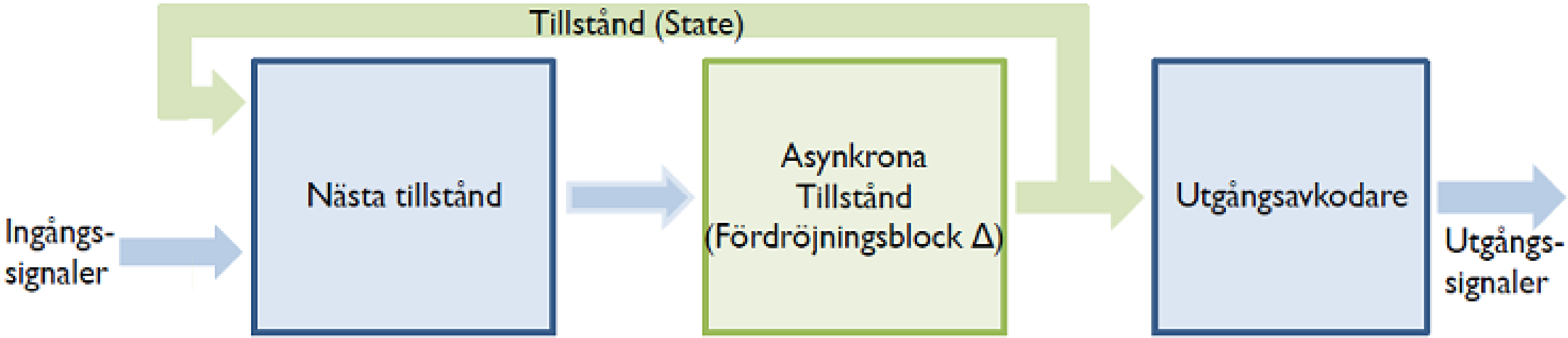
Flödestabell och Tillståndsdiagram (Mealy)

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



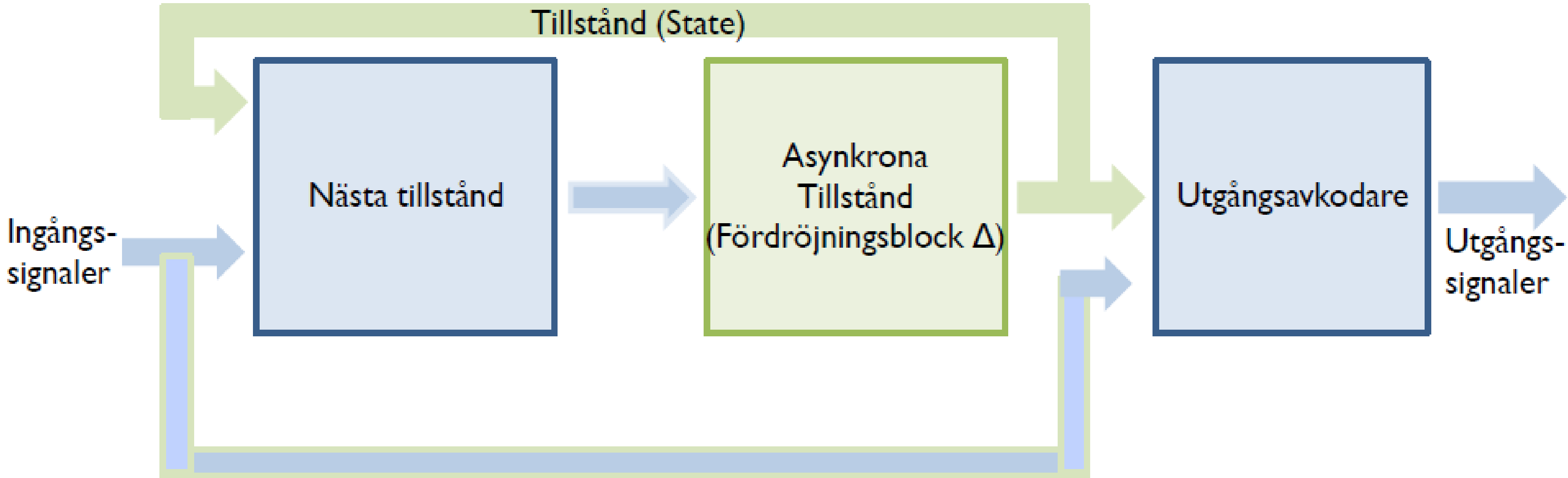
Don't care ('-') har valts för utgångsavkodaren. Det spelar ingen roll om utgången ändras före eller efter tillståndsövergången (= enklare grindnät).

Asynkron Moore kompatibel



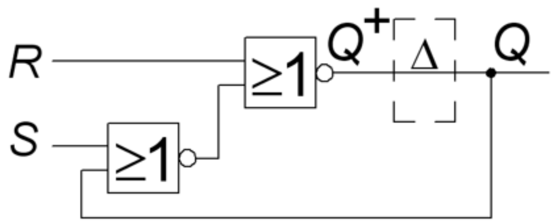
- Asynkrona sekvensnät har liknande uppbyggnad som synkrona sekvensnät
- I stället för vippor har man ”fördröjningsblock”

Asynkron Mealy kompatibel

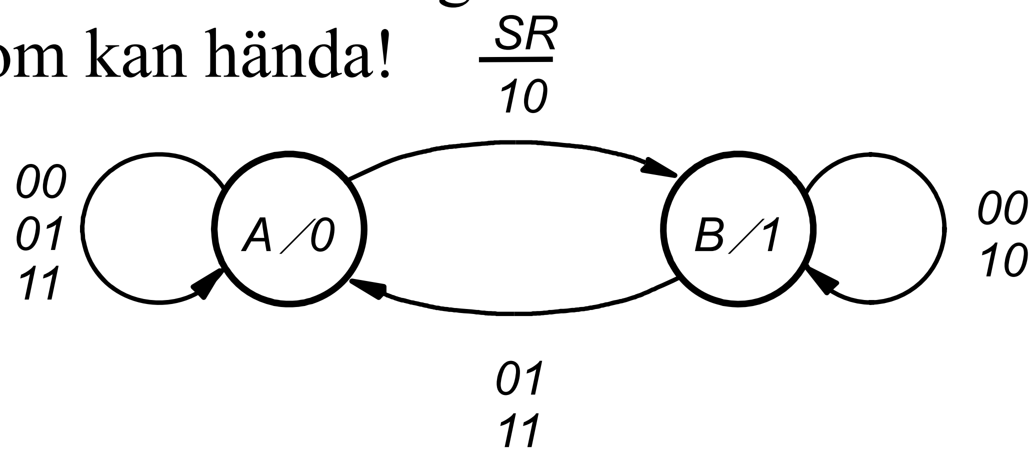


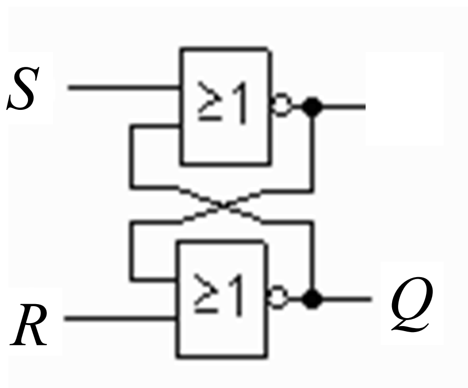
- Asynkrona sekvensnät har liknande uppbyggnad som synkrona sekvensnät
- I stället för vippor har man ”fördröjningsblock”

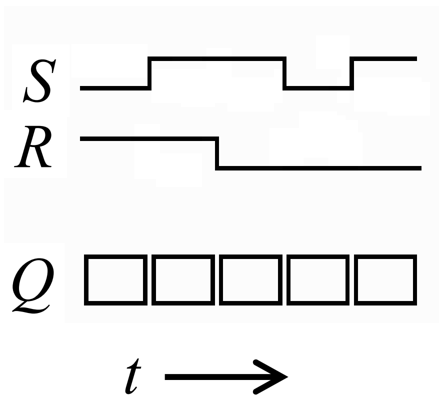
Nu enklare att lösa tidsdiagram!



Med tillståndsdiagrammet vet vi allt som kan hända!

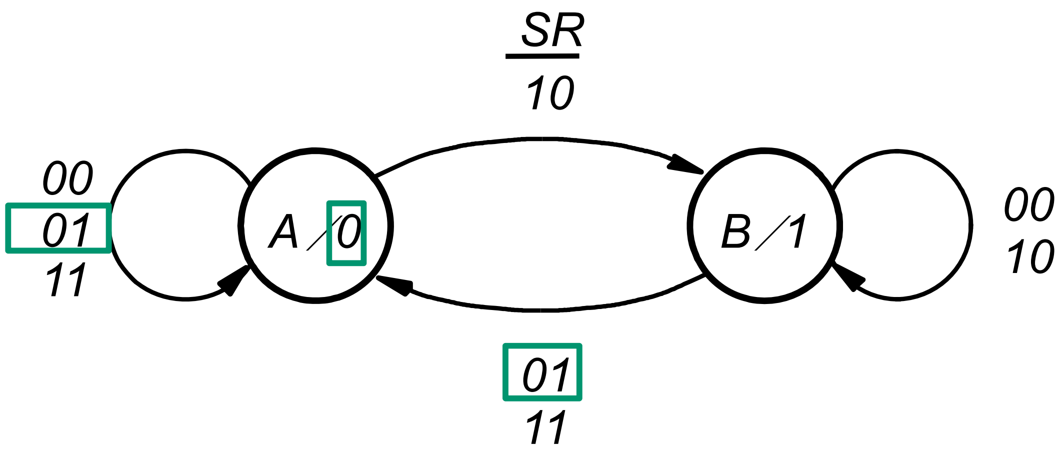


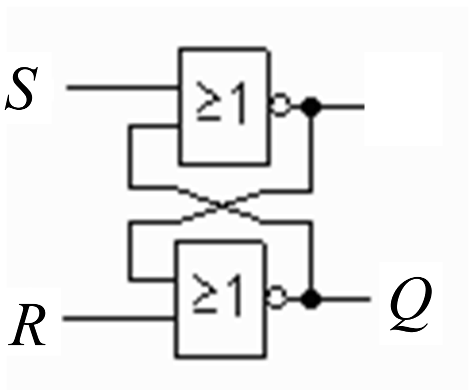


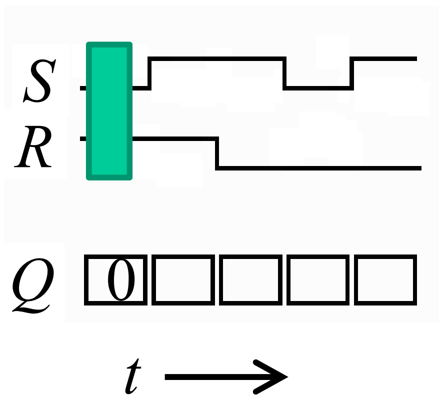


?

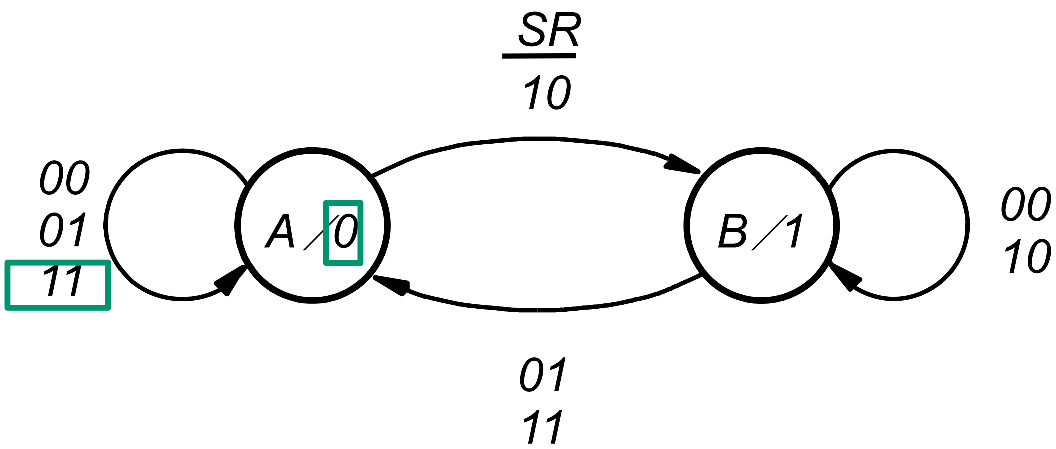
Nu enklare att lösa tidsdiagram!

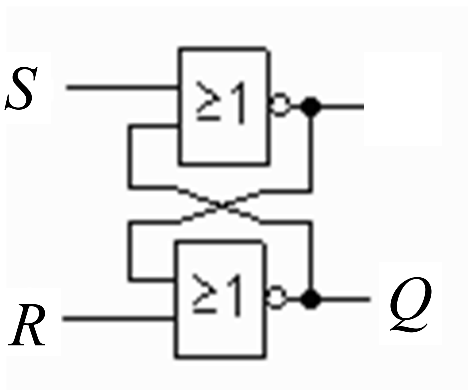


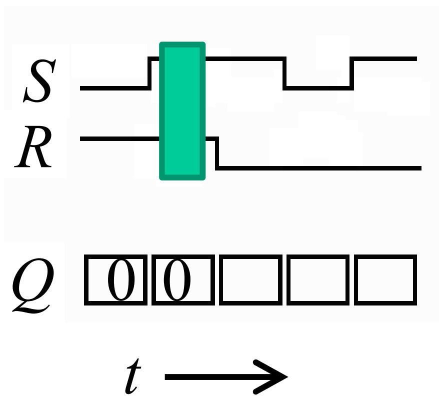




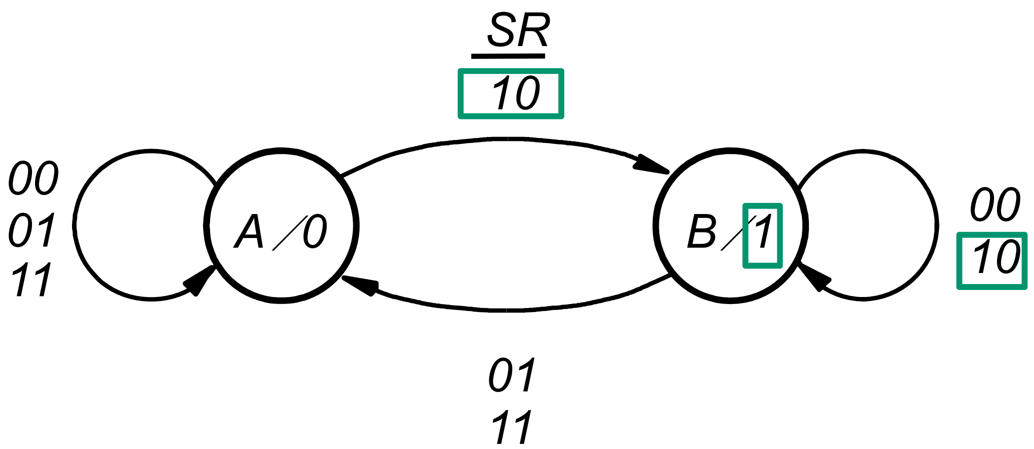
Nu enklare att lösa tidsdiagram!

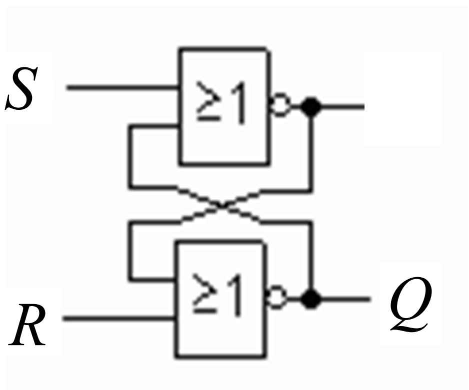


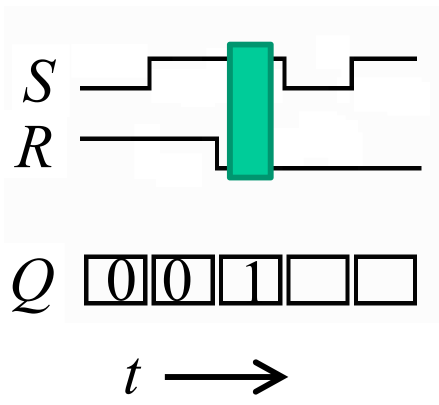




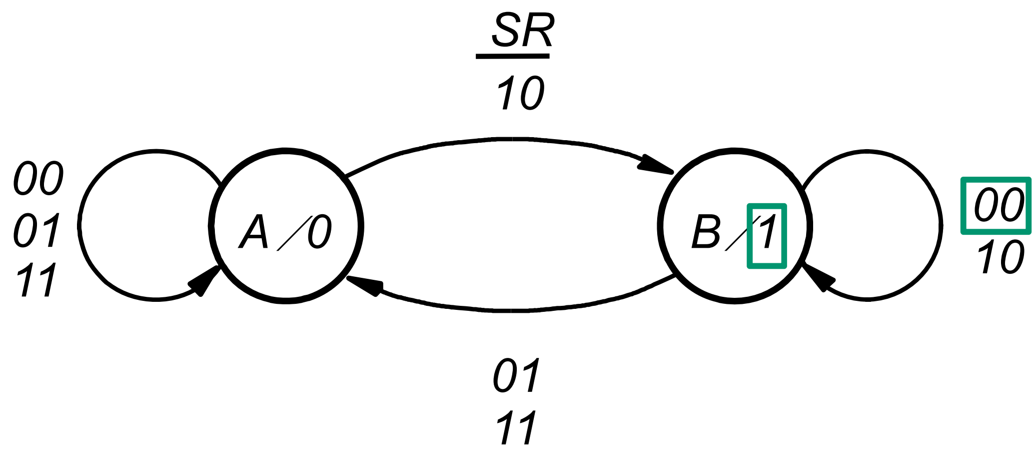
Nu enklare att lösa tidsdiagram!

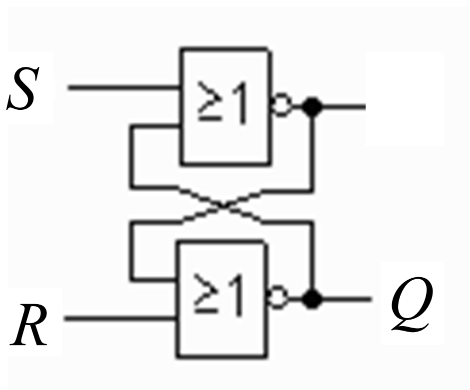


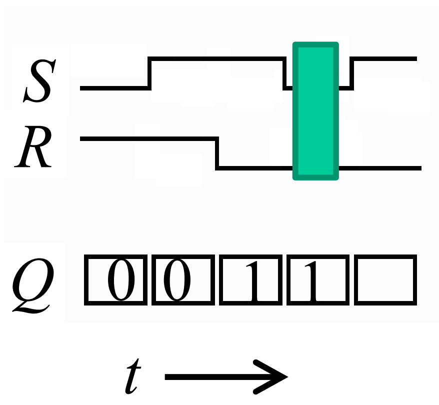




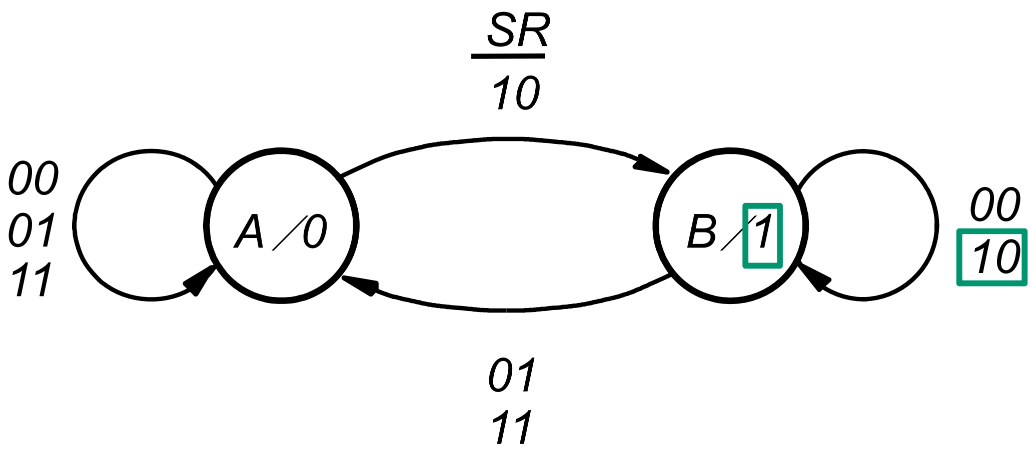
Nu enklare att lösa tidsdiagram!

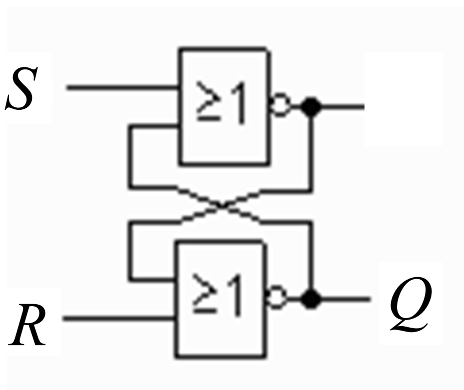


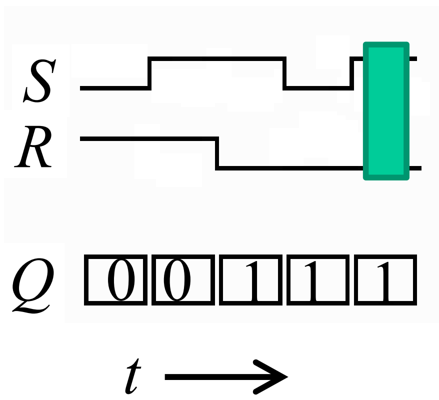




Nu enklare att lösa tidsdiagram!





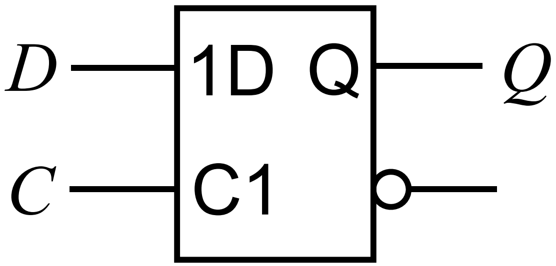


Analys av asynkrona kretsar

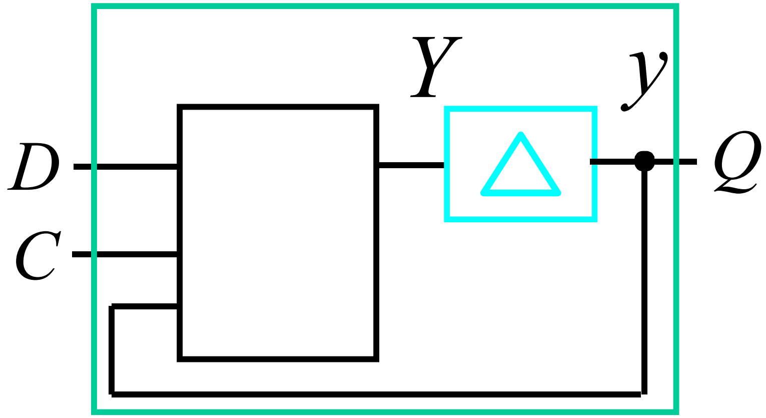
Analysen görs i följande steg:

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 reda på next-state och output uttrycken
3) Ställ upp motsvarande **excitationstabell**
4) Skapa en **flödestabell** genom att byta ut kodade tillstånd mot symboliska
5) Rita ett tillståndsdiagram om så behövs

Först: D-latchens tillståndsfunktion



$$C = follow\,/\,\overline{latch}$$

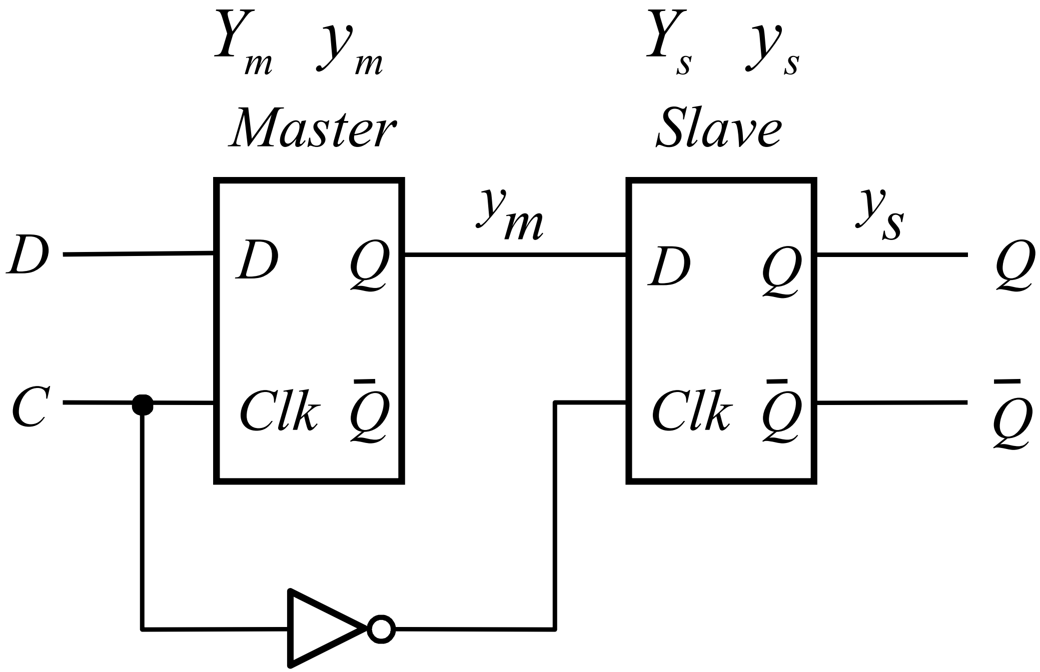


D-latchens tillståndsfunktion. Funktionssambandet mellan nuvarande tillstånd y och nästa tillstånd Y

$$Y = \underset{follow}{D \cdot C} + \underset{\overline{latch}}{y \cdot \overline{C}}$$

Exempel: Master-Slave-vippan

*Master-slave D-vippan är konstruerad av **två** asynkrona **D-latchar**.*



Tillstånds-uttryck:

$$Y_m = D \cdot C + y_m \cdot \overline{C}$$

$$Y_s = y_m \cdot \overline{C} + y_s \cdot C$$

Exitationstabell

*Ur uttrycken kan man **direkt** härleda excitations-tabellen (om man nu kan hålla allt i huvudet?)*

$$Y_m = D \cdot C + y_m \cdot \overline{C}$$

$$Y_s = y_m \cdot \overline{C} + y_s \cdot C$$

Present state $y_m\ y_s$	Next state CD = 00	01	11	10	Output Q
	$Y_m\ Y_s$				
00	00	00	10	00	0
01	00	00	11	01	1
11	11	11	11	01	1
10	11	11	10	00	0

eller med K-map till hjälp …

Y_m

$y_m y_s$ \ CD	00	01	11	10
00			1	
01			1	
11	1	1	1	
10	1	1	1	

$y_m\overline{C}$ DC

Y_s

$y_m y_s$ \ CD	00	01	11	10
00				
01			1	1
11	1	1	1	1
10	1	1		

$y_m\overline{C}$ $y_s C$

$Y_m Y_s$

$y_m y_s$ \ CD	00	01	11	10
00	00	00	10	00
01	00	00	11	01
11	11	11	11	01
10	11	11	10	00

$$Y_m = D \cdot C + y_m \cdot \overline{C} \qquad Y_s = y_m \cdot \overline{C} + y_s \cdot C$$

Present state $y_m\, y_s$	Next state CD = 00	01	11	10	Output Q
	$Y_m\, Y_s$				
00	00	00	10	00	0
01	00	00	11	01	1
11	11	11	11	01	1
10	11	11	10	00	0

Flödestabell

Vi definierar fyra tillstånd S1, S2, S3, S4 och erhåller då flödestabellen

Present state $y_m\,y_s$	Next state CD = 00	01	11	10	Output Q
	$Y_m\,Y_s$				
00	00	00	10	00	0
01	00	00	11	01	1
11	11	11	11	01	1
10	11	11	10	00	0

Present state	Nextstate CD = 00	01	11	10	Output Q
S1	S1	S1	S3	S1	0
S2	S1	S1	S4	S2	1
S4	S4	S4	S4	S2	1
S3	S4	S4	S3	S1	0

Flödestabell

***Kom ihåg**: Bara en insignal kan ändras åt gången*

- *Därmed kommer vissa övergångar **aldrig** att kunna inträffa!*

Present state	Nextstate CD = 00	01	11	10	Output Q
S1	S1	S1	S3	S1	0
S2	S1	S1	S4	S2	1
S4	S4	S4	S4	S2	1
S3	S4	S4	S3	S1	0

Flödestabell – omöjliga övergångar

Present state	Nextstate CD = 00	01	11	10	Output Q
S1	S1	S1	S3	S1	0
S2	S1	S1	S4	S2	1
S4	S4	S4	S4	S2	1
S3	~~S4~~	S4	S3	S1	0

Tillstånd S3

Enda stabila tillståndet för **S3** är när ingångskombinationen är 11

Bara en ingång kan ändras → möjliga ändringar är 11 → 01, 11 → 10

- Dessa kombinationer lämnar S3!
- Ingångskombinationen 00 i S3 är *inte* möjligt!
- Ingångskombinationen 00 sätts därför till **don't care**!

Flödestabell – omöjliga övergångar

Present state	Nextstate CD = 00	01	11	10	Output Q
S1	S1	S1	S3	S1	0
S2	S1	~~S1~~	S4	S2	1
S4	S4	S4	S4	S2	1
S3	~~S4~~	S4	S3	S1	0

Tillstånd S2

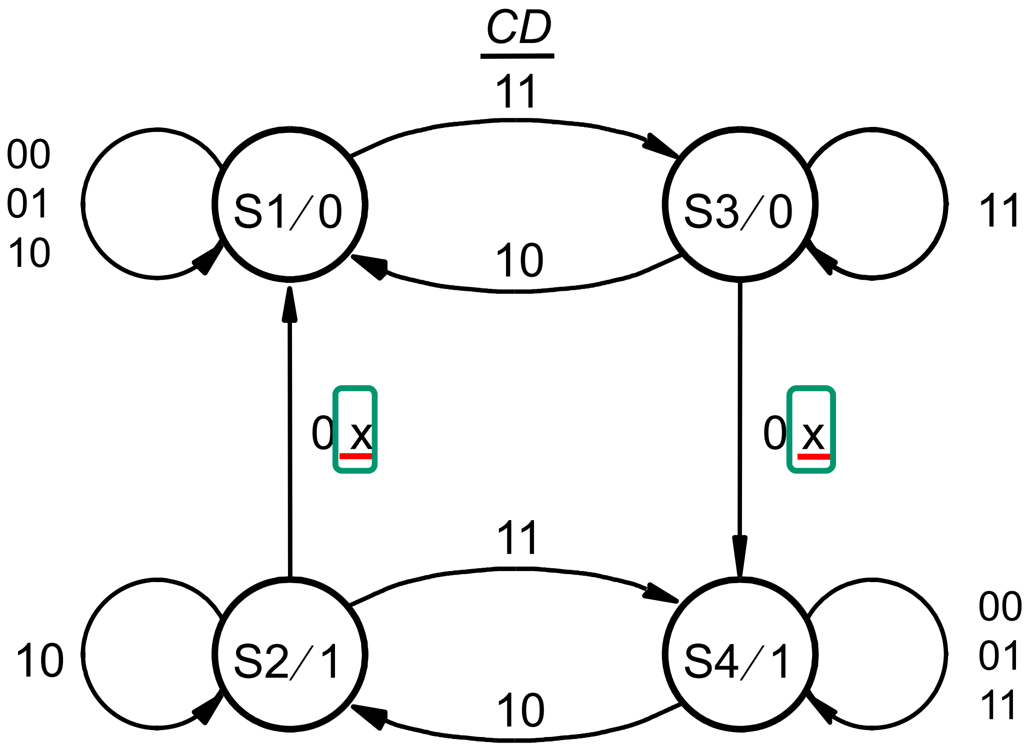
Enda stabila tillståndet för **S2** är när ingångskombinationen är 10

Bara en ingång kan ändras → möjliga ändringar är 10 → 11, 10 → 00

- Dessa kombinationer lämnar S2!
- Ingångskombinationen 01 i S2 är *inte* möjligt!
- Ingångskombinationen 01 sätts därför till **don't care**!

D-vippans tillståndsdiagram

Don't care betecknas här med x



Present state	Next state CD = 00	01	11	10	Output Q
S1	S1	S1	S3	S1	0
S2	S1	~~S1~~	S4	S2	1
S4	S4	S4	S4	S2	1
S3	~~S4~~	S4	S3	S1	0

Don't care kan användas för att förenkla kretsens nästa tillståndsavkodning.

Syntes av asynkrona kretsar

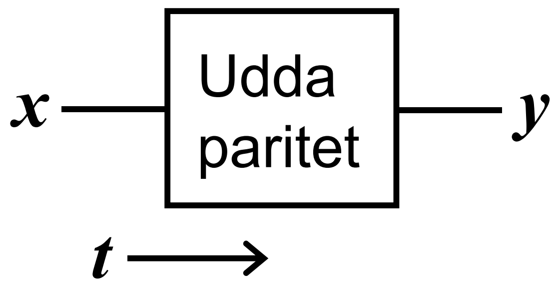
Syntesen genomförs i följande steg:

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

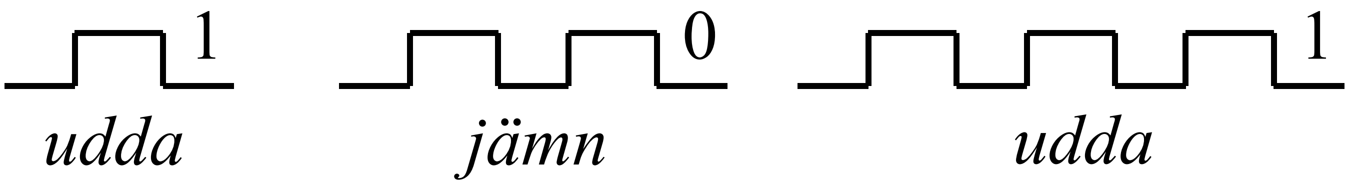
Exempel: seriell paritetskrets

Ingång $\boldsymbol{x}$ Utgång $\boldsymbol{y}$

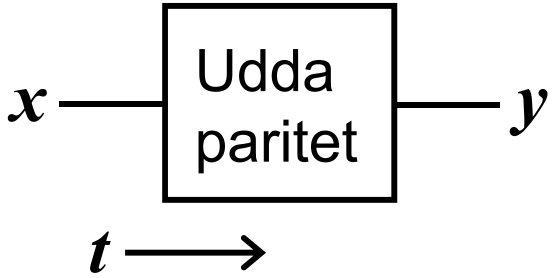
$\boldsymbol{y} = 1$ om antalet pulser på ingången $\boldsymbol{x}$ har varit udda.

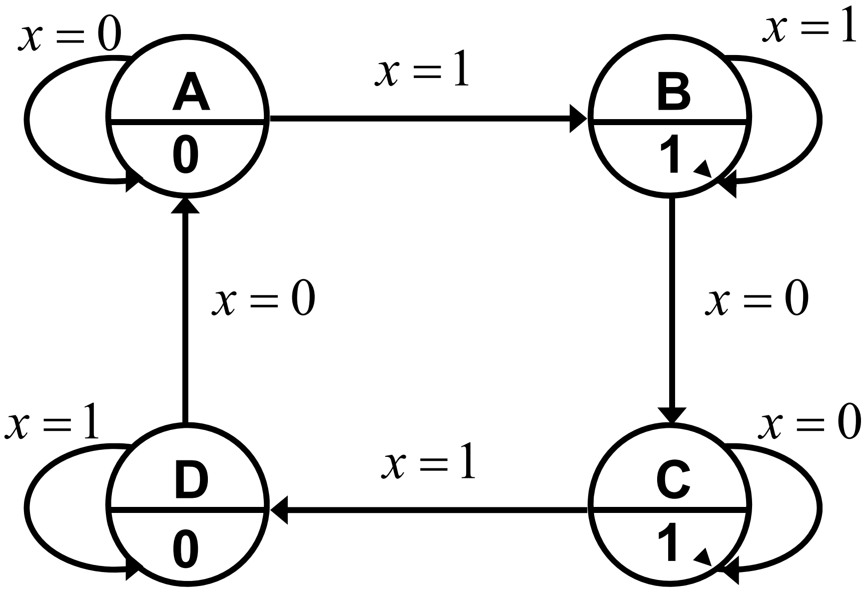


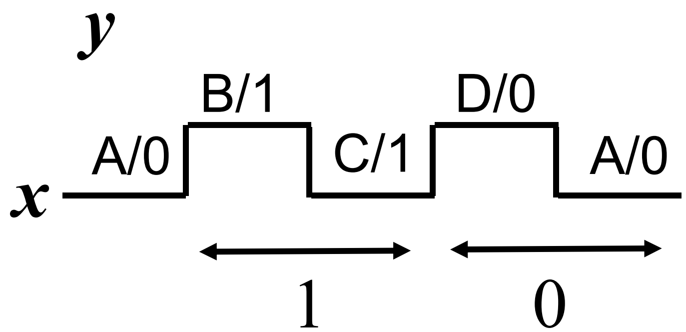
Med andra ord en "varannangång" krets …



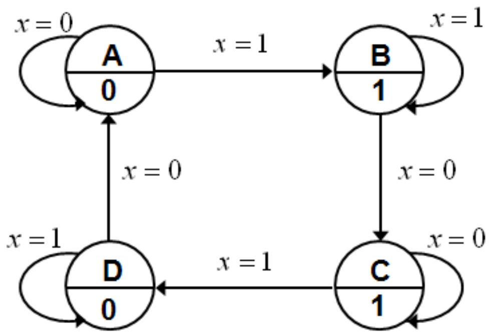
Skapa tillståndsdiagram

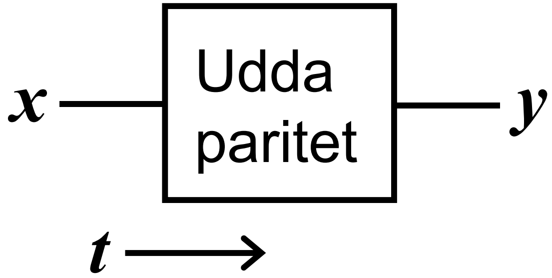






Skapa flödestabellen





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

Dålig kodning (HD=2!)

- *Antag*

$X = 1 \;\; Y_2Y_1 = 11$

- *därefter*

$X \rightarrow 0 \rightarrow Y_2Y_1 = 00?$

$11 \rightarrow 10!$

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

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

Richard Hamming

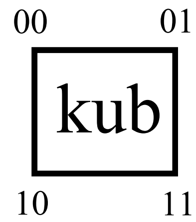
- I asynkrona sekvensnät är det omöjligt att garantera att två **tillståndsvariabler** ändrar värdet samtidigt
  - Därmed kan en övergång 00 → 11 resultera i
    - en övergång 00 → 01 → ???
    - en övergång 00 → 10 → ???
- För att säkerställa funktionen **<u>MÅSTE</u>** alla tillståndsövergångar ha ***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

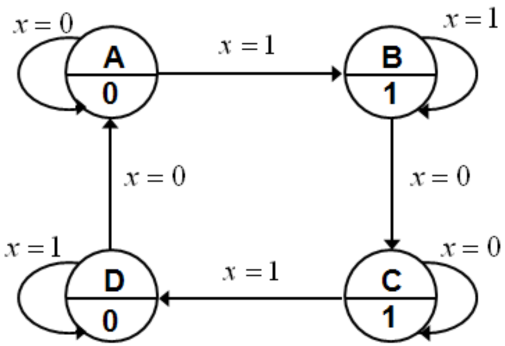
Bra tillståndskodning

- Procedur för att erhålla bra koder:
  1) Rita transitionsdiagram längs kanterna i **hyperkuber** (Graykod) som bildas av koderna
  2) Ta bort eventuella *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

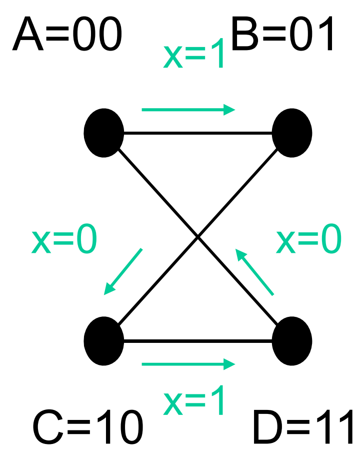
Dålig kodning av paritetskretsen

Den dåliga tillståndskodningen





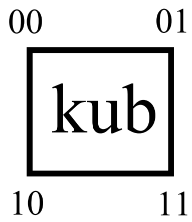
Pres state y_2y_1	Next State Y_2Y_1 X=0	1	Q
A 00	00	01	0
B 01	10	01	1
C 10	10	11	1
D 11	00	11	0

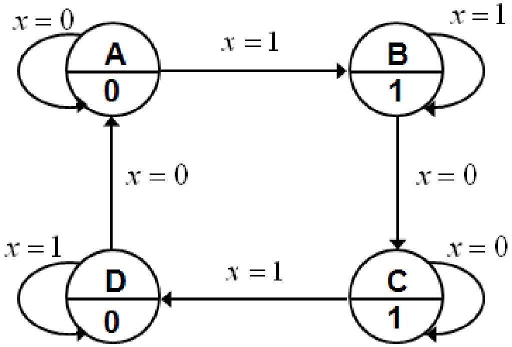


Dålig kodning –
Hamming Distance = 2
(**korsande linjer**)

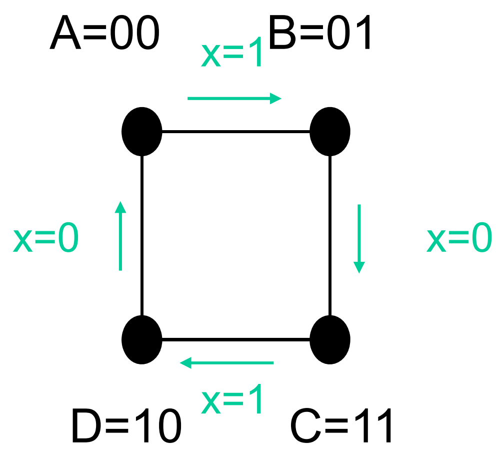
Bra kodning av paritetskretsen

Den bra tillståndskodningen





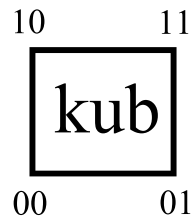
Pres state y_2y_1	Next State Y_2Y_1 X=0	1	Q
A 00	00	01	0
B 01	11	01	1
C 11	11	10	1
D 10	00	10	0



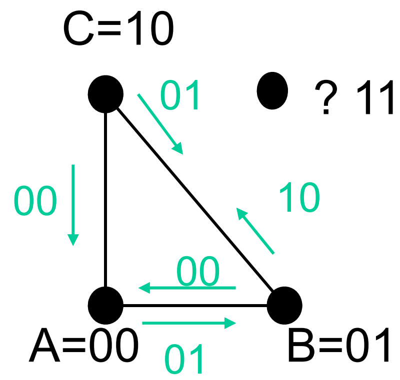
Bra kodning
Hamming Distance = 1
(**inga korsande linjer**)

Problem med icke stabila tillstånd

Ex. en annan krets:



Present state	Nextstate r_2r_1 = 00	01	11	10	Output g_2g_1
A 00	A	B	–	C	00
B 01	A	B	B	C	01
C 10	A	B	C	C	10

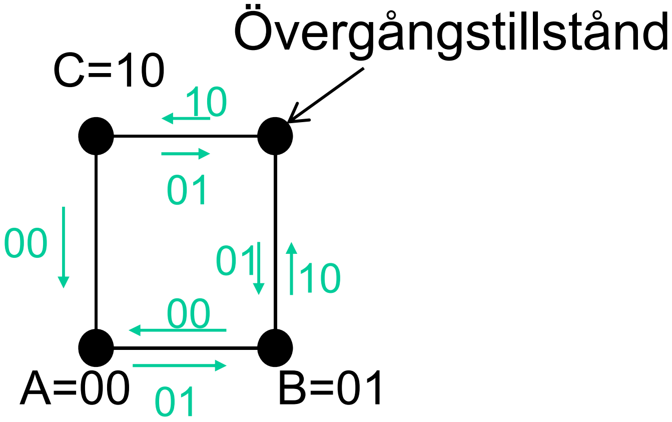


Dålig kodning

Vid övergången från **B** till **C** (eller **C** till **B**) är Hamming distansen 2 (10↔01)!
Risk att man fastnar i ett **ospecificerat** tillstånd (med kod 11)!

Lösning på icke stabila tillstånd

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



Bra kodning

	Present state y_2y_1	Nextstate r_2r_1 = 00	01	11	10	Output g_2g_1
		Y_2Y_1				
A	00	00	01	–	10	00
B	01	00	01	01	11	01
-	11	–	01	–	10	--
C	10	00	11	10	10	10

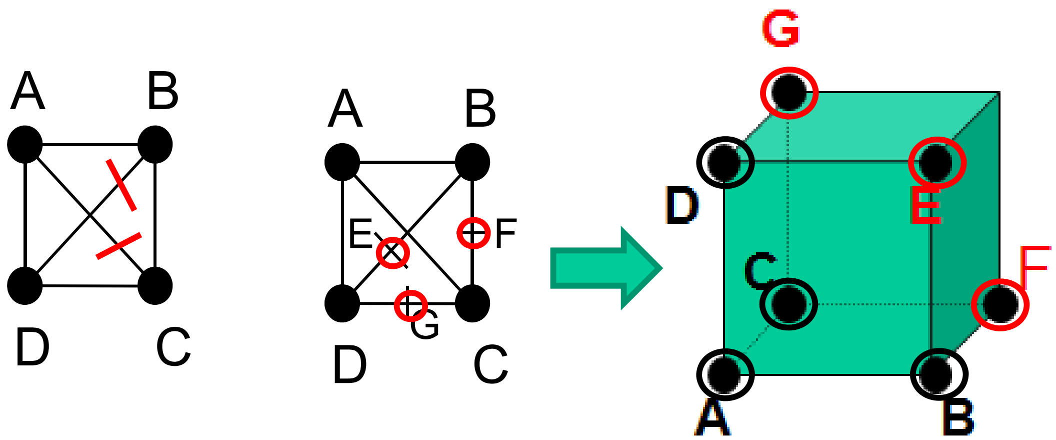
Begära ”efter-sändning”

$$01 \rightarrow 11 \rightarrow 10$$
$$10 \rightarrow 11 \rightarrow 01$$

övergångstillstånd

Extra tillstånd – fler dimensioner

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

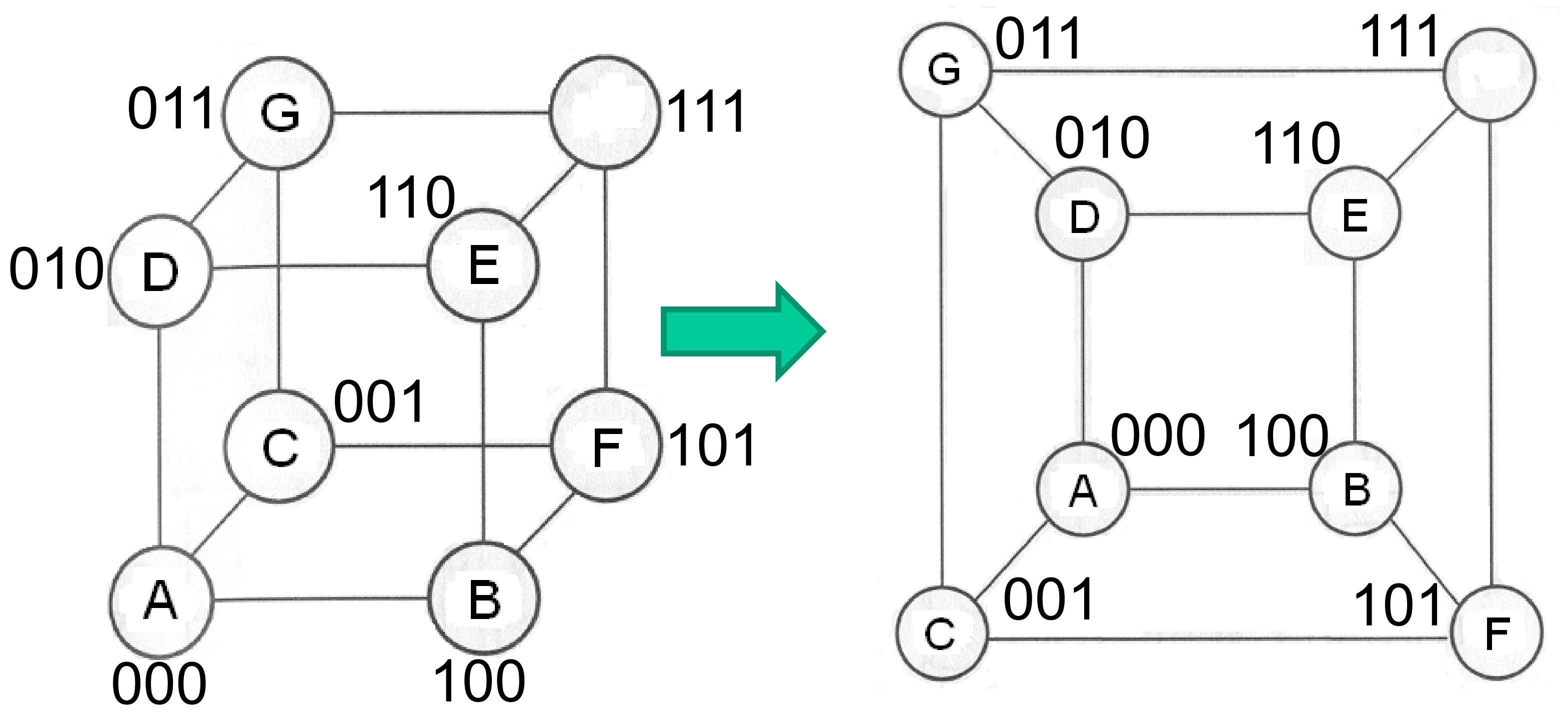


Använd lediga tillstånd som övergångstillstånd ("eftersändning"). **B→D : B→E→D**

- 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 ("eftersändning"). **B→D : B→E→D.**

Extra tillstånd – fler dimensioner

- Det är enklare att rita en ”platt” 3D-kub (perspektivet då rakt framifrån)



Karnaughdiagrammen

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

x \ y_2y_1	00	01	11	10
0	0	1	1	0
1	0	0	1	1

$$Y_2 = \bar{x}y_1 + \boxed{y_2y_1} + xy_2$$

x \ y_2y_1	00	01	11	10
0	0	1	1	0
1	1	1	0	0

$$Y_1 = x\bar{y}_2 + \boxed{\bar{y}_2y_1} + \bar{x}y_1$$

y_2 \ y_1	0	1
0	0	1
1	0	1

$$Q = y_1$$

De röda inringningarna är för att undvika Hazard (se senare avsnitt)!

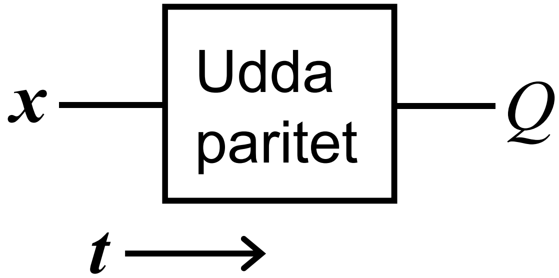
Färdig krets

x \ y_2y_1	00	01	11	10
0	0	1	1	0
1	0	0	1	1

$$Y_2 = \overline{x}y_1 + y_2y_1 + xy_2$$

y_2 \ y_1	0	1
0	0	1
1	0	1

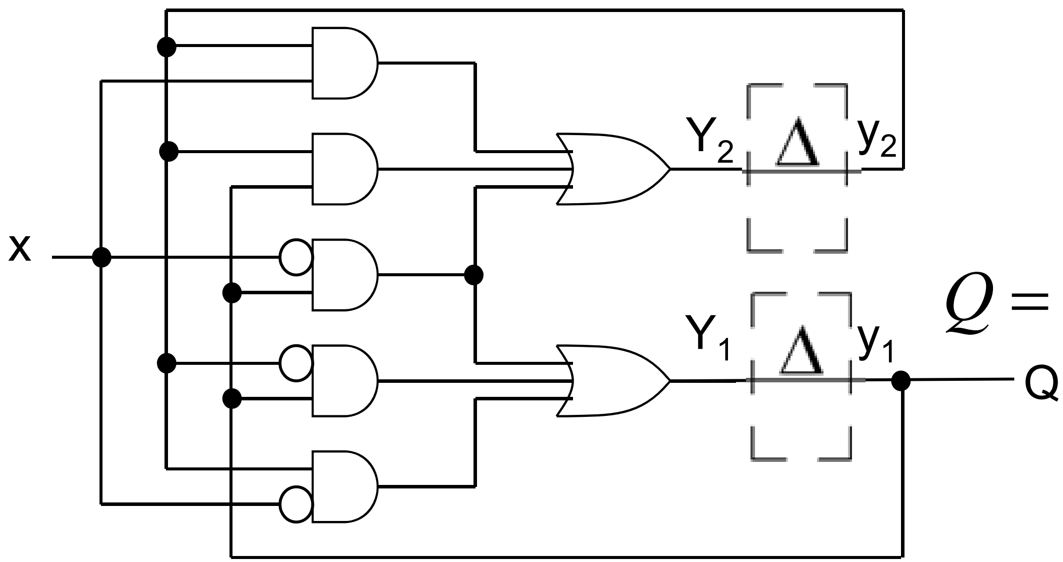
$$Q = y_1$$



x \ y_2y_1	00	01	11	10
0	0	1	1	0
1	1	1	0	0

$$Y_1 = x\overline{y}_2 + \overline{y}_2y_1 + \overline{x}y_1$$

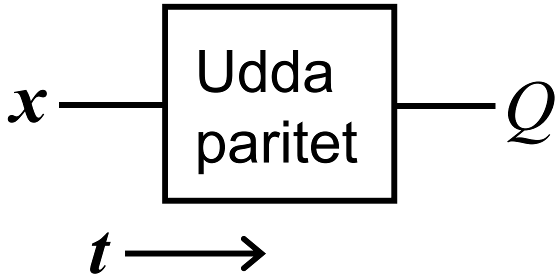
$$Y_2 = \overline{x}y_1 + y_2y_1 + xy_2$$

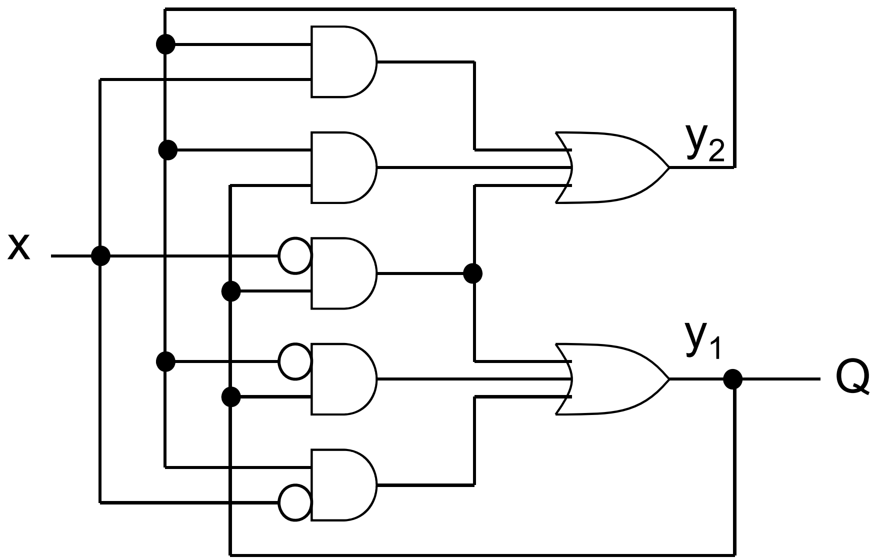


$$Q = y_1$$

$$Y_1 = x\overline{y}_2 + \overline{y}_2y_1 + \overline{x}y_1$$

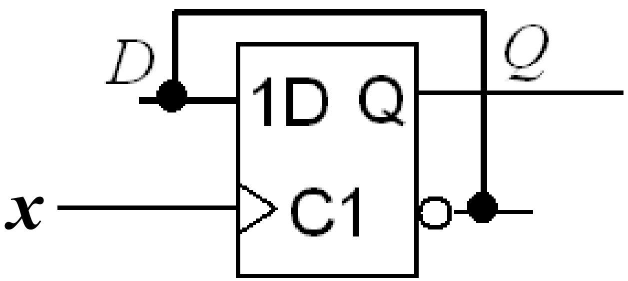
Färdig krets

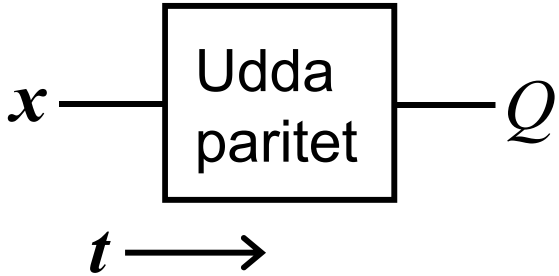




(enklare med D-vippa)

$D = \overline{Q} \quad Q = D$





Vi har gjort en ”varannangångkrets” tidigare i kursen. Då med en D-vippa. Men nu blev det ju mera ”sport”!

Vad är Hazard?
”Glitch” i Harris & Harris, sid 92-95

- 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, signal-kapplöpning.
- För att motverka detta måste man lägga till prim-implikanter för att täcka upp den farliga övergången.

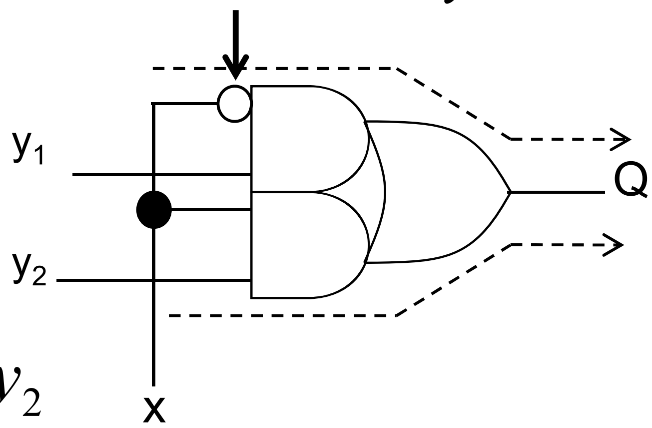
Exempel på Hazard – MUX:en

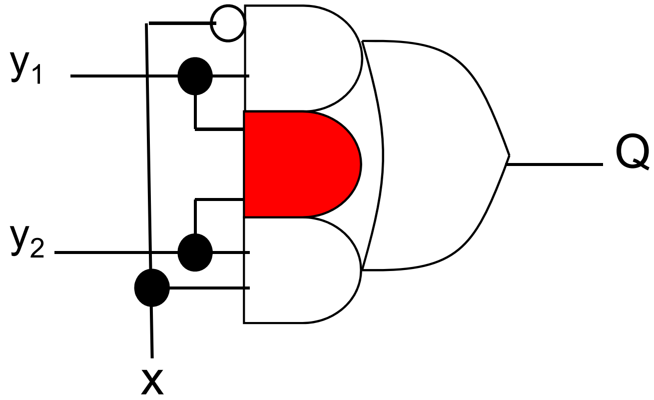
extra delay!

x \ y_2y_1	00	01	11	10
0	0	1	1	0
1	0	0	1	1

$$Y_2 = \bar{x}y_1 + \boxed{y_2 y_1} + xy_2$$

extra delay!



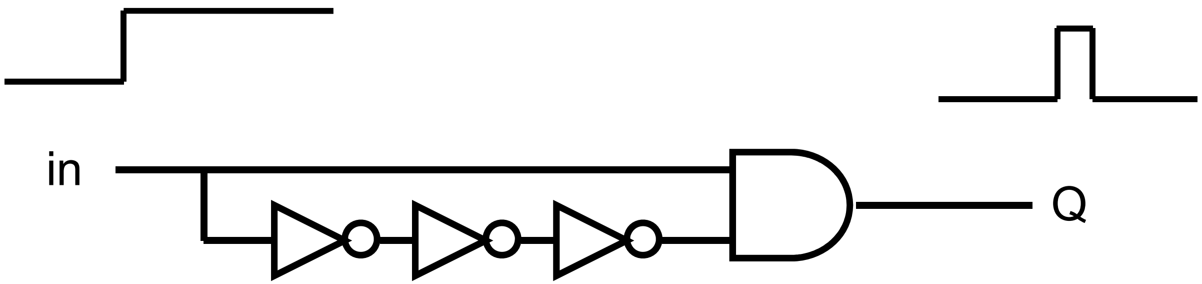


Vid övergång från xy_2y_1=(111) → (011) kan utgången *Q* ***blinka till***, eftersom vägen från *x* till *Q* är längre via den övre AND-grinden än den lägre (kapplöpning).

Hasard ”spikar”

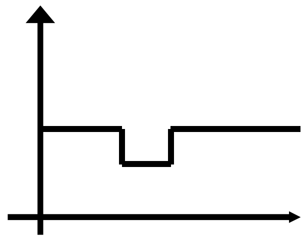
- När man konstruerar asynkrona kretsar så kan det händer att man får spikar (glitches) på signalvärden
- Detta beror på att olika signalvägar har olika fördröjningstider
- Fenomenet kallas för hasard (hazard) och kan elimineras med noggrann konstruktion

(Kort 0-ställningspuls)

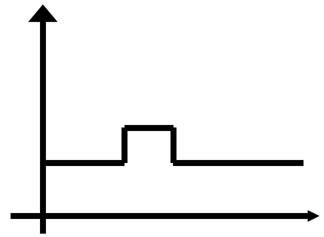


Kretsen används ibland för att generera en kort 0-ställningspuls.

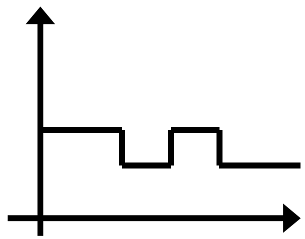
Olika typer av Hasard



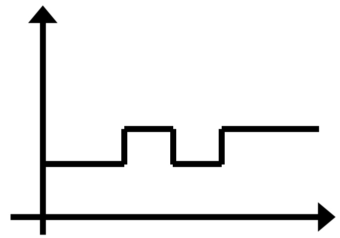
Statisk 1 → 1



Statisk 0 → 0

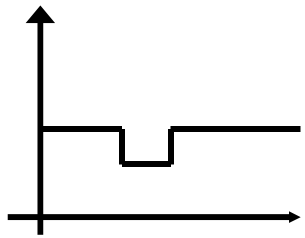


Dynamisk 1 → 0

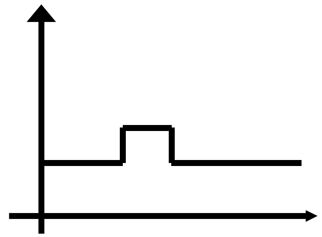


Dynamisk 0 → 1

Statisk Hasard



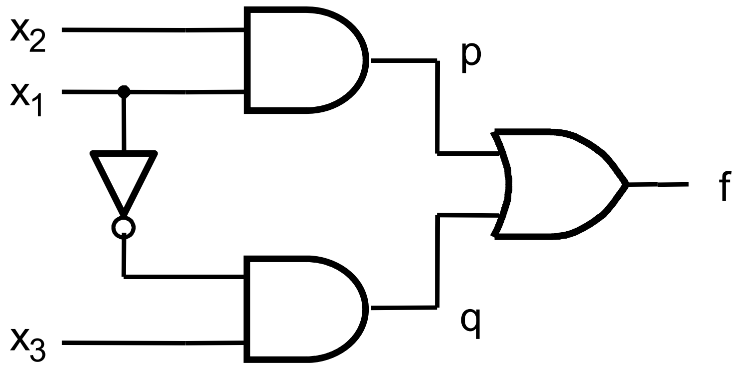
Statisk 1 → 1



Statisk 0 → 0

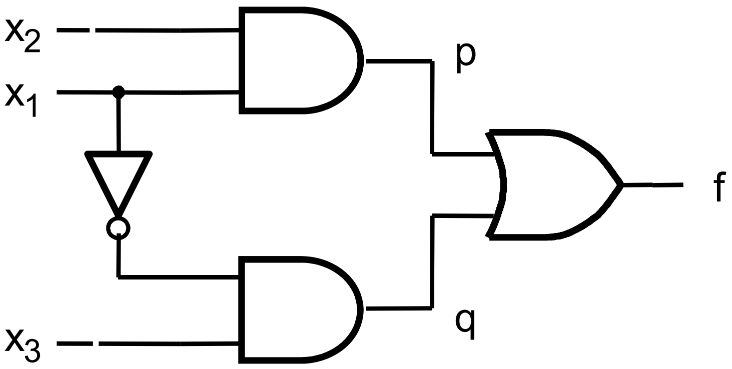
Exempel, Statisk Hasard

- Hasard kan uppträda vid nedanstående krets vid övergången av $x_3x_2x_1$ från 11**1** $\leftrightarrow$ 11**0**

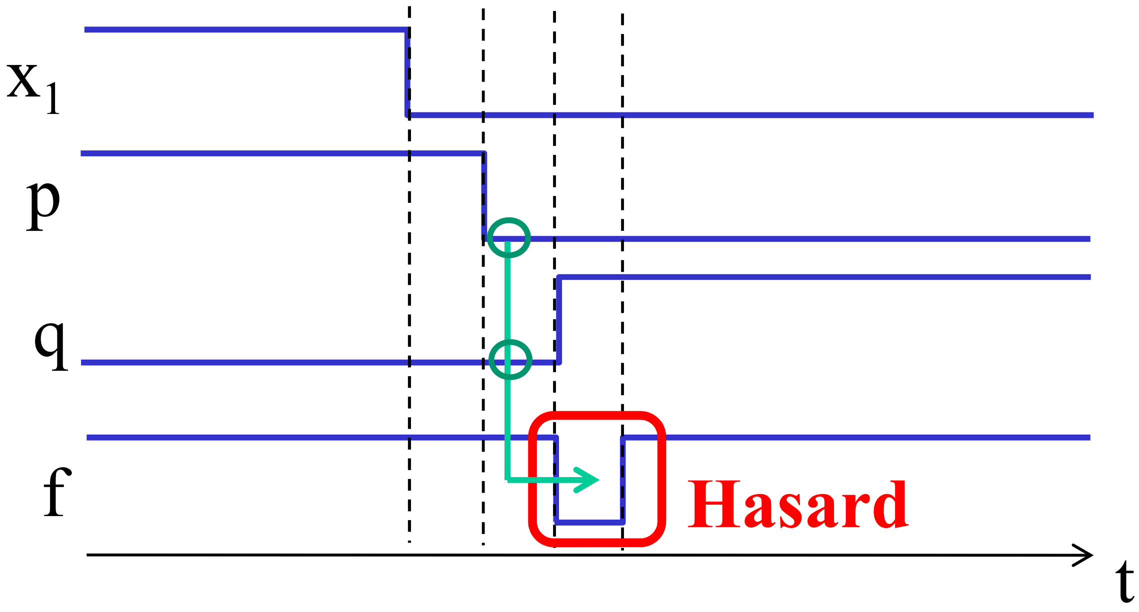


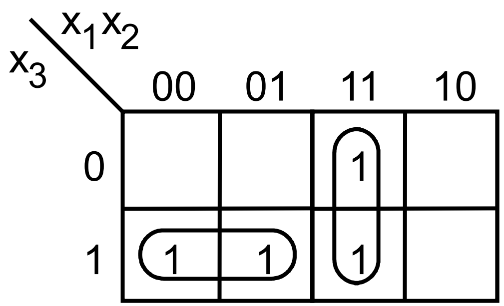
$$f = x_1x_2 + \bar{x}_1x_3$$

Tidsdiagrammet

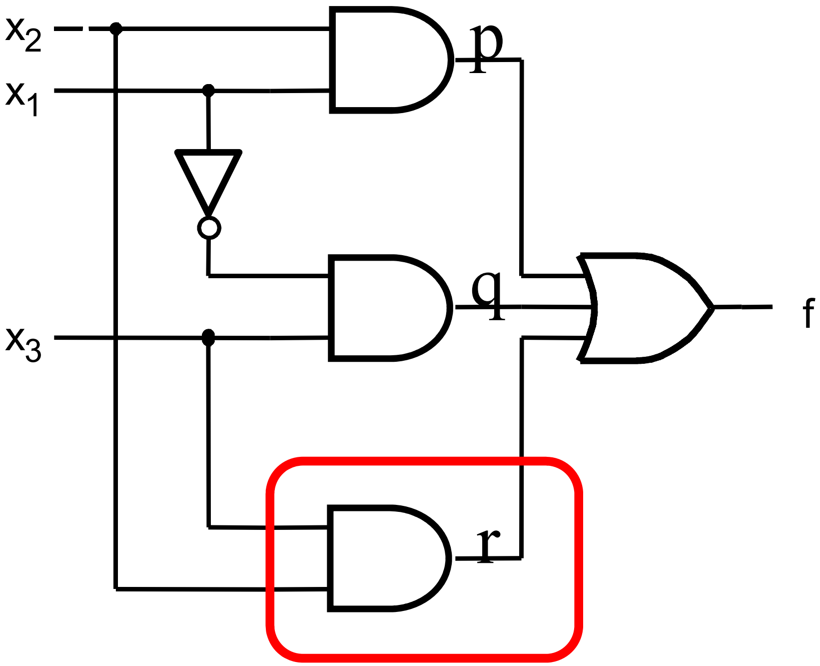


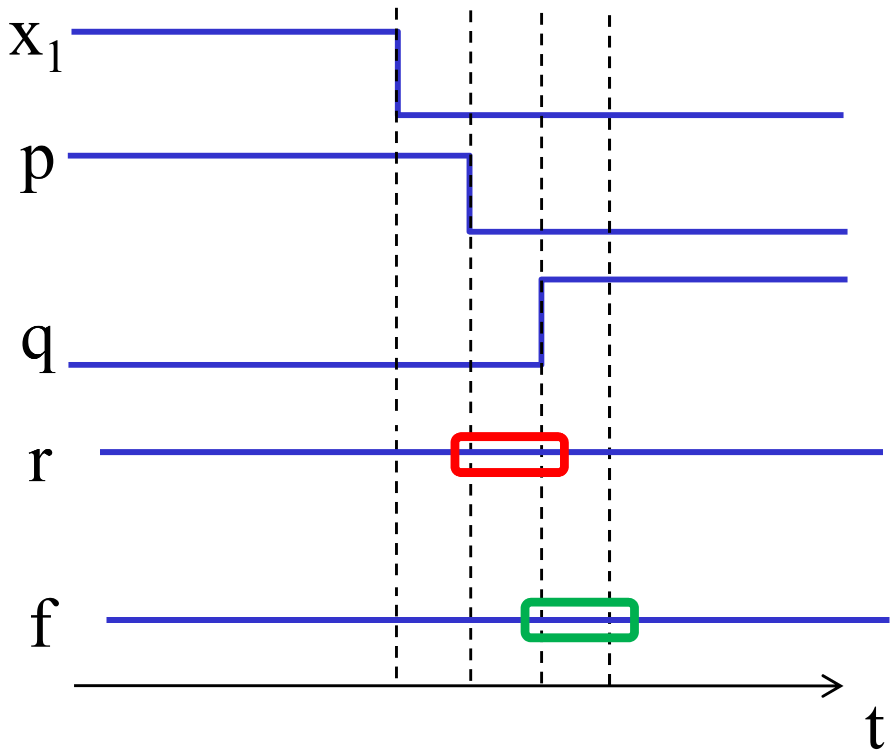
$$f = x_1 x_2 + \bar{x}_1 x_3$$





Hasardfri krets





Hasard-Cover

x_3 \ x_1x_2	00	01	11	10
0			1	
1	1	1	1	

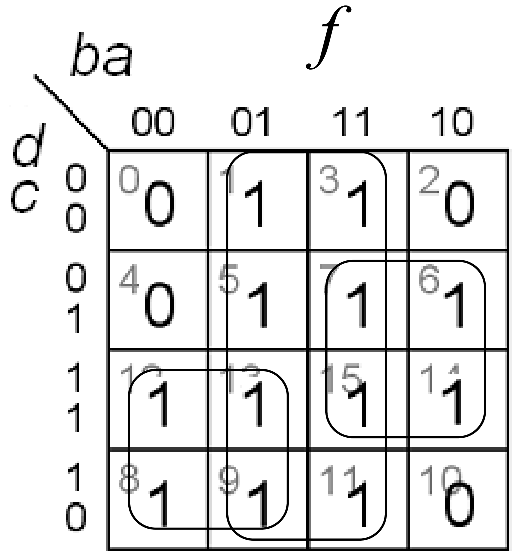
$$f = x_1x_2 + \bar{x}_1x_3 \boxed{+ x_2x_3}$$

Hur undviker man statisk hasard?

- Möjligheten för statisk hasard finns om två intill-liggande 1:or inte är täckta med en egen produktterm vid SOP
- Därmed kan man ta bort risken för statisk hazard genom att lägga till inringningar så att **alla intill-liggande** 1:or är täckta med en egen inringning

Ex. Hasardfria hoptagningar

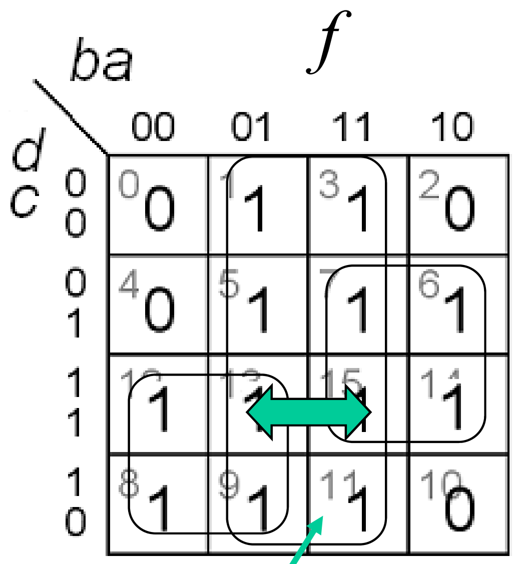
Räcker dessa hoptagningar för hasardfrihet?



$$f = a + d\bar{b} + cb$$

Ex. Hasardfria hoptagningar

Räcker dessa hoptagningar för hasardfrihet?

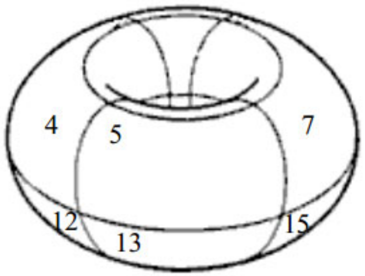


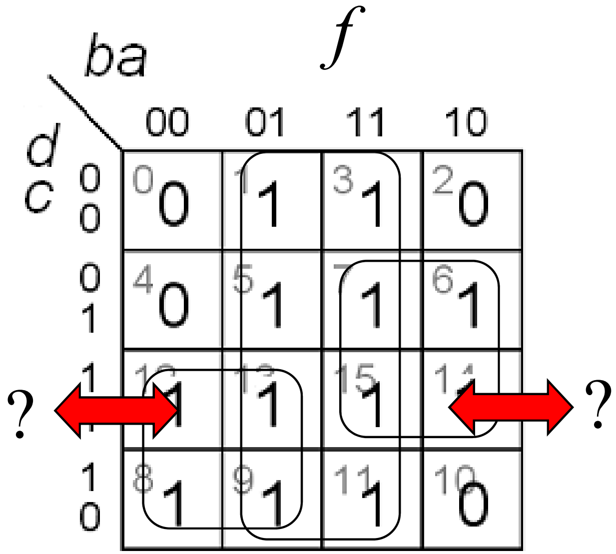
$$f = a + d\overline{b} + cb$$

Hasard cover

Ex. Hasardfria hoptagningar

Räcker dessa hoptagningar för hasardfrihet?



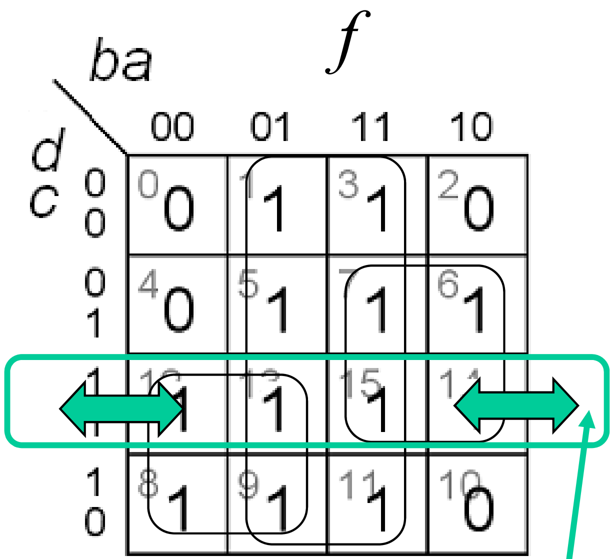


Karnaugh-diagrammet är en donut!

$$f = a + d\overline{b} + cb$$

Ex. Hasardfria hoptagningar

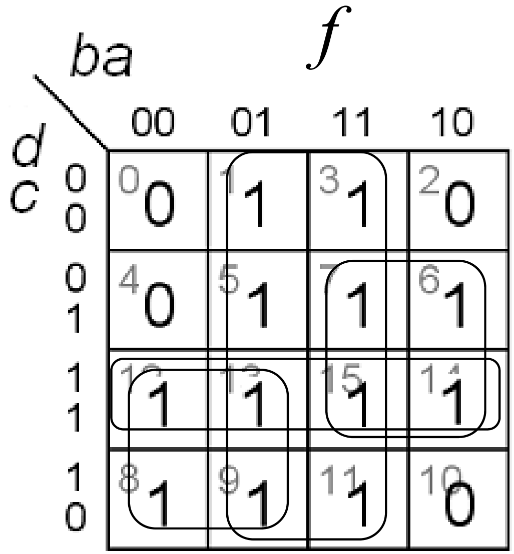
Räcker dessa hoptagningar för hasardfrihet?



$$f = a + d\bar{b} + cb + dc$$

Ex. Hasardfria hoptagningar

Lätt att missa!

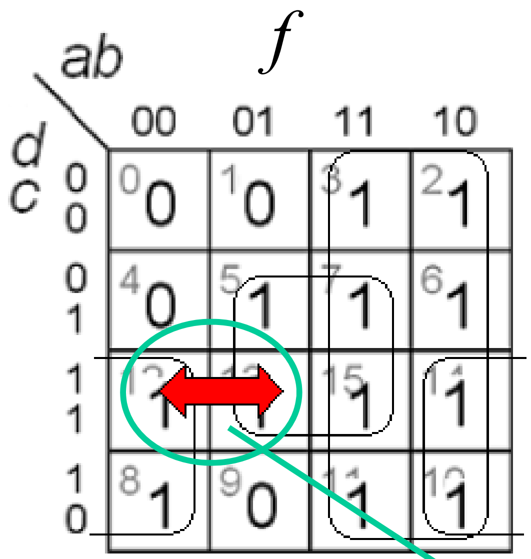


$$f = a + d\bar{b} + cb + dc$$

Hasard cover

Ex. Hasardfria hoptagningar

Med annan variabelordning missar man inte!



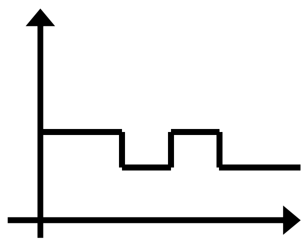
$$f = a + d\bar{b} + cb + cd$$

Statisk hasard vid POS?

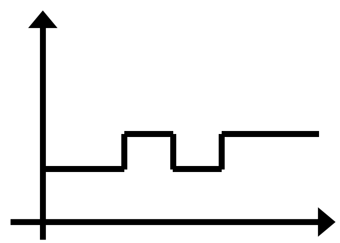
- Har man en POS-implementering så måste man se till att alla bredvidliggande 0:or är täckta av en egen summaterm

Dynamisk Hasard?

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



Dynamisk 1 → 0



Dynamisk 0 → 1

Exempel, Dynamisk Hasard

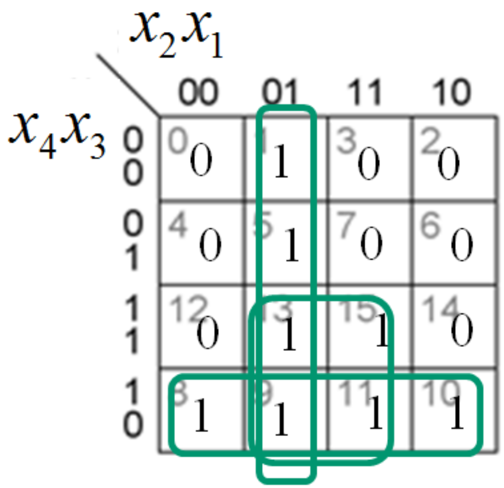
- Följande ekvation orsakar ingen hasard om man implementerar den som en AND-OR-struktur

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

Exempel, Dynamisk Hasard

- Följande ekvation orsakar ingen hasard om man implementerar den som en AND-OR-struktur

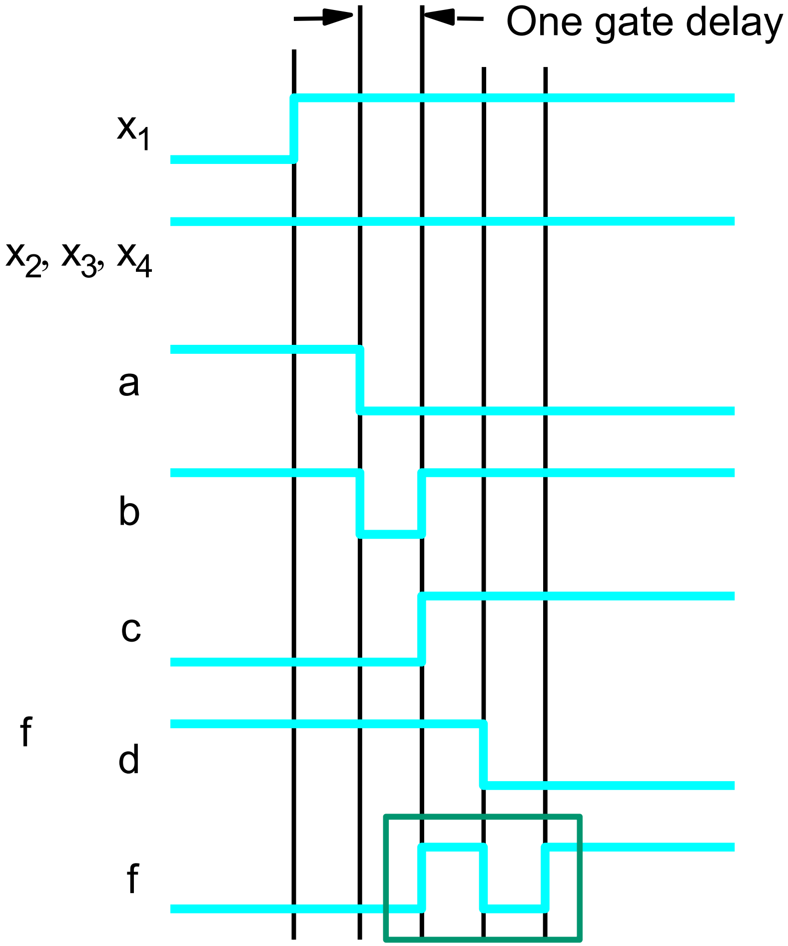
$$f = x_1\bar{x}_2 + \bar{x}_3 x_4 + x_1 x_4$$

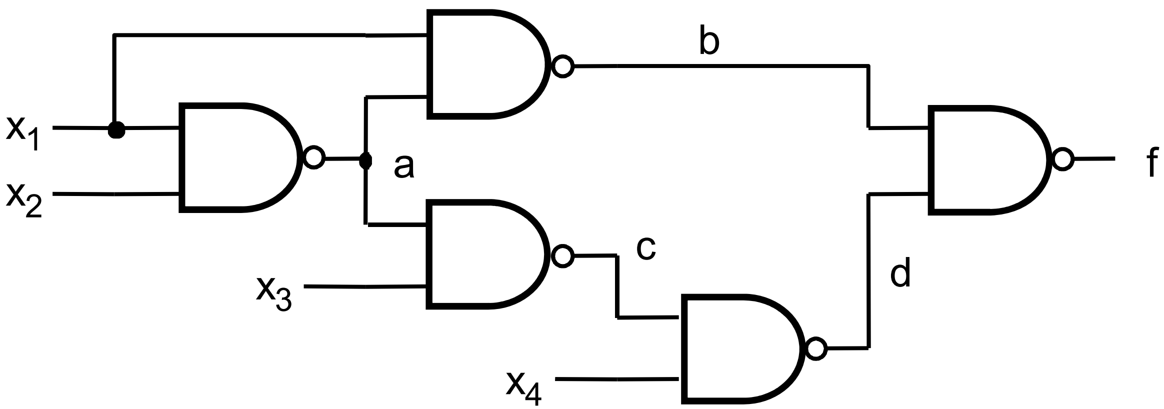


Problemfri med två-nivålogik!

Exempel, Dynamisk Hasard

- Men implementerar man ekvationen med följande **flernivåslogik**, så uppträder dynamisk hasard





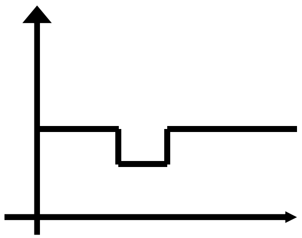
Hur undviks Dynamisk Hasard?

- Dynamisk hasard kan undvikas med **två-nivå-logik**
- Ser man till att en två-nivå krets är fri från statisk hasard, så finns det inte heller någon dynamisk hasard!

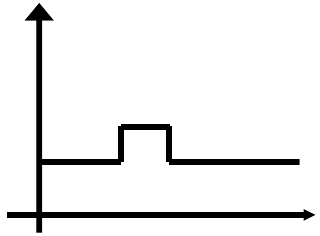
När behöver man ta hänsyn till Hasard?

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

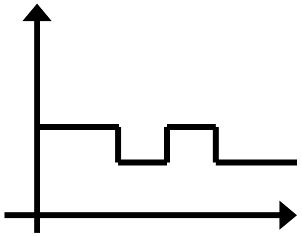
Undvik Hasard



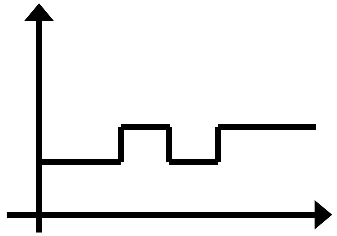
Statisk 1 → 1



Statisk 0 → 0



Dynamisk 1 → 0

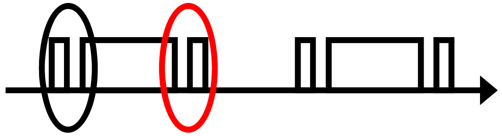


Dynamisk 0 → 1

Statisk hasard orsakas av utelämnade primimplikanter

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

Utgångs-spikar i asynkrona sekvensnät

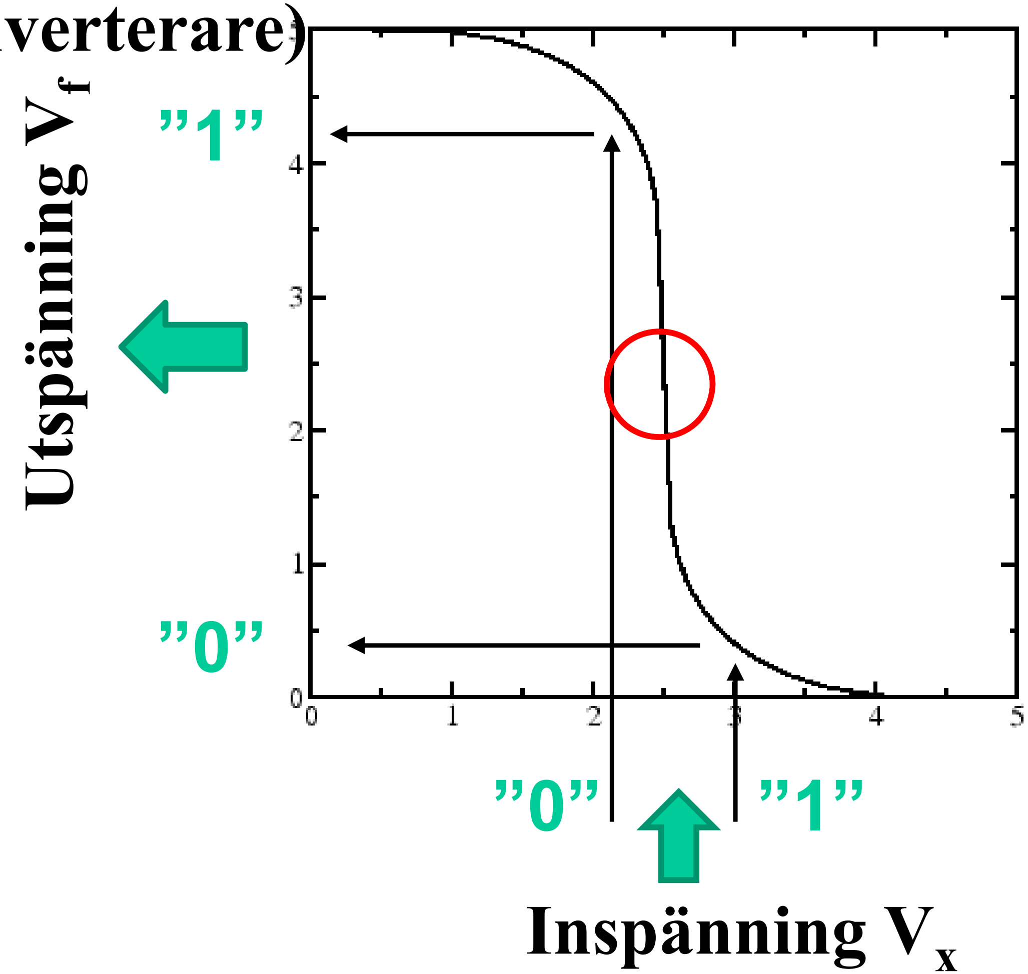


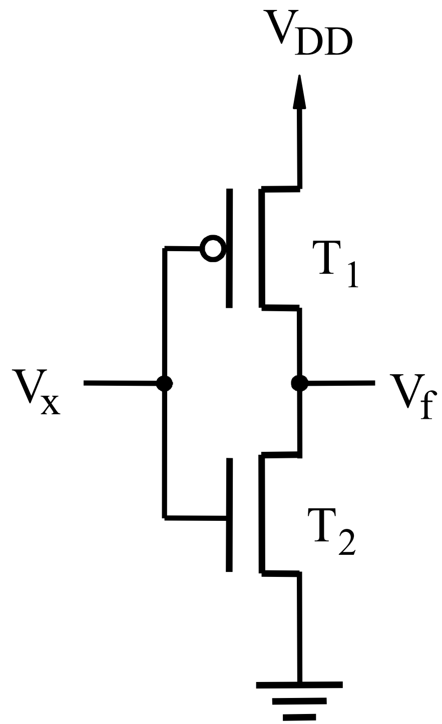
Pres state	Next State		Q
	X=0	1	
y_2y_1	Y_2Y_1		
00	00	01	0
01	00	11	1
11	01	10	0
10	11	10	1

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!).

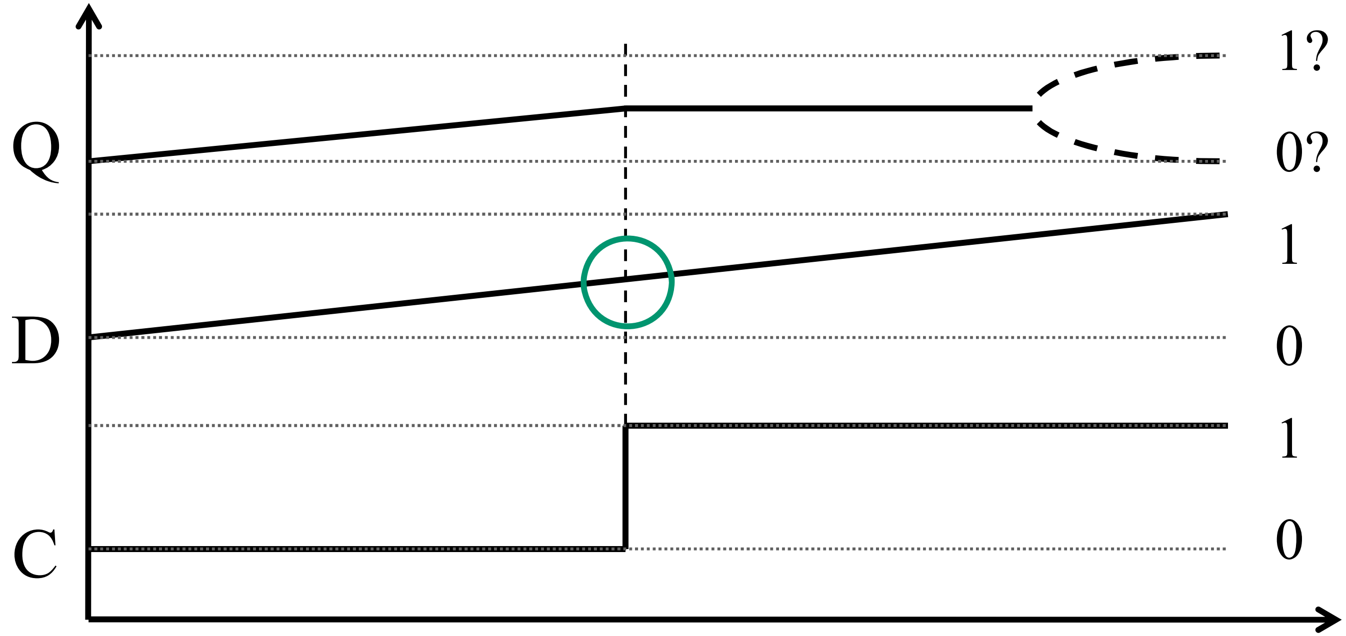
Metastabilitet

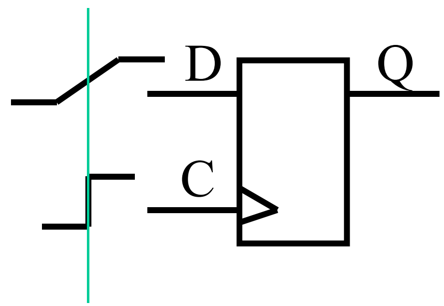
CMOS-kretsens överföringsfunktion (ex. inverterare)





Om metastabilitet





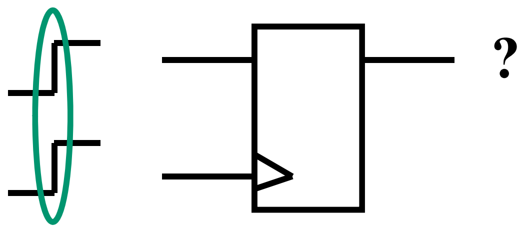
För att förstå vad metastabilitet innebär så kan vi tänka oss att insignalen D till en latch är väldigt belastad och därmed ändrar sig mycket långsamt i förhållande till klockan. Antag vidare att klock-signalen C slår om precis när D är vid $V_{DD}/2$.

Då låser sig latchen vid det spänningsvärde som råkar finnas på D.
Efter en tid slår latchen om till antingen '1' eller '0'.

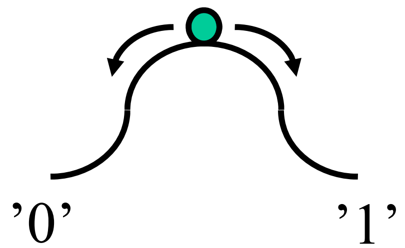
Om metastabilitet …

Denna instabilitet varar tills transistorerna i återkopplingen behagar gå åt ena eller andra hållet – men det kan ta tid, och tiden beror 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.

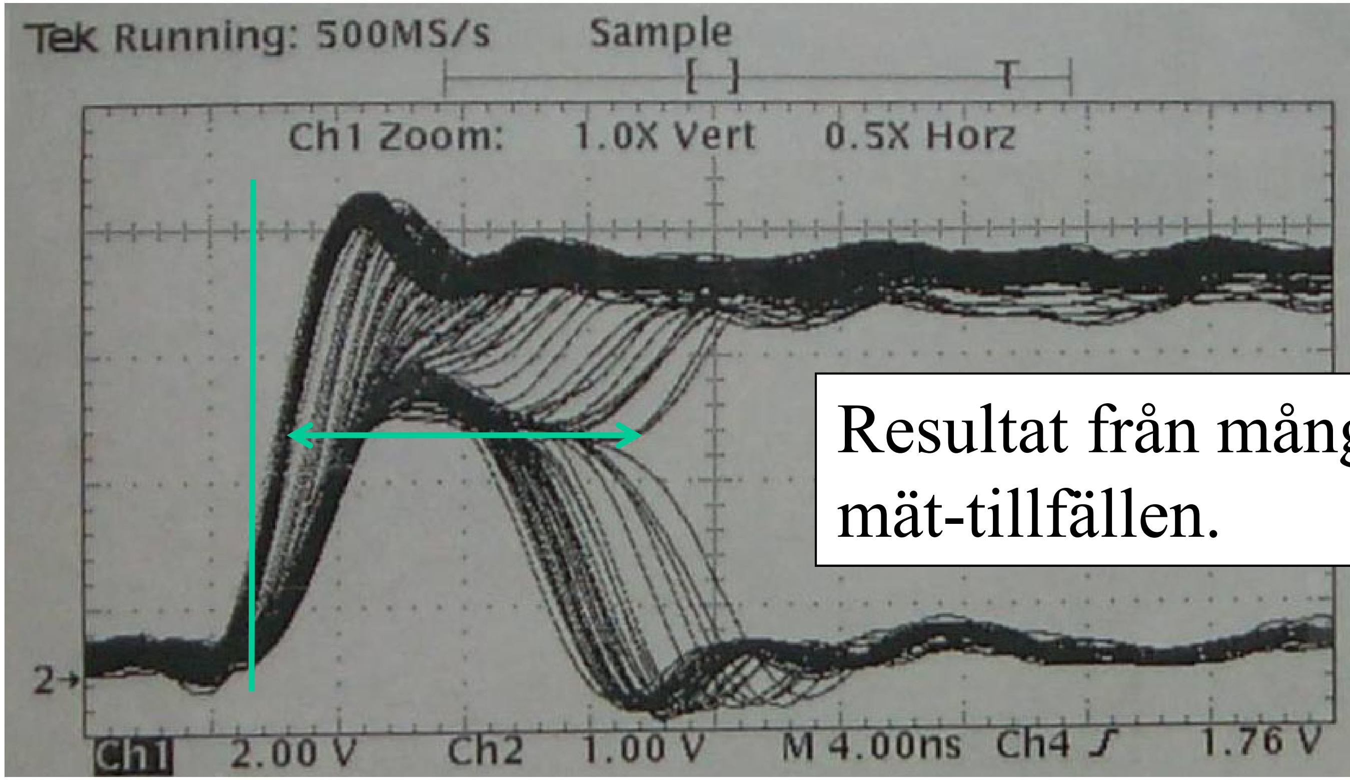


Om Clk och D switchar samtidigt, vilket värde får då Q?



På vilken sida kommer bollen att trilla ner?

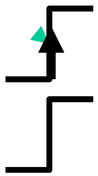
Klockpuls och data samtidigt!



Resultat från många mät-tillfällen.

”1”

”0”



Klocka och data ändras samtidigt!

Hur mycket samtidigt?

∞ exakt samtidigt – oändlig tid …

Tidsfördröjning innan utvärdet blir bestämt

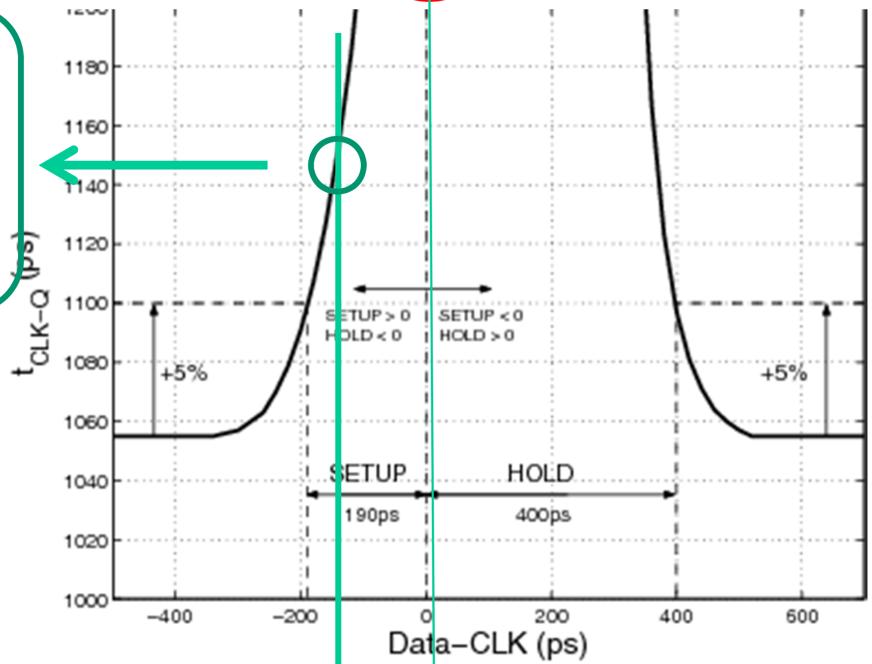


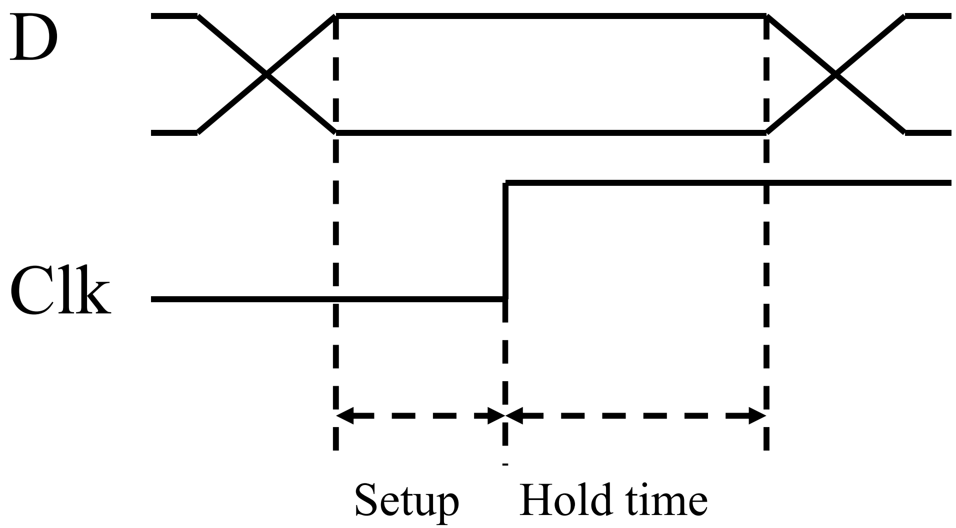
Figure 1. Definitions of setup and hold times.

data klocka

hur nära?

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

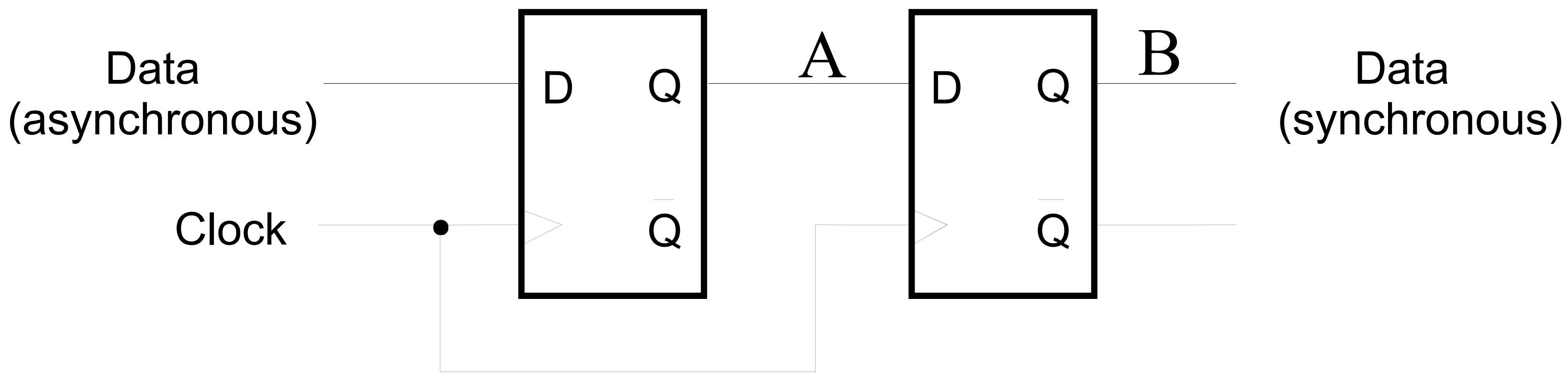
Om Setup and Hold time's är uppfyllda, så kommer vippan (Flip-flop) att garanterat bete sig snällt/deterministiskt!

Asynkrona insignaler?

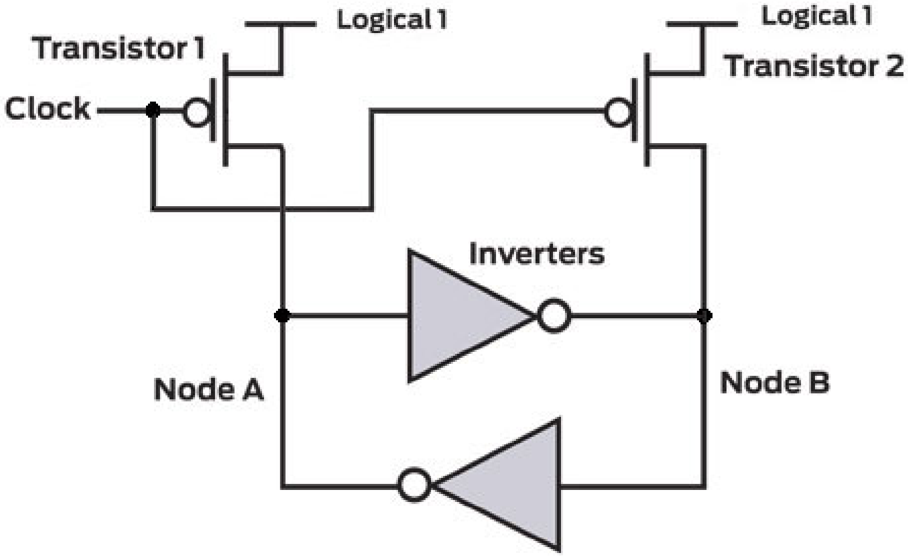
- Dessvärre kan vi inte alltid garantera att en ingång är stabil under hela setup- och hold-tiden
- Antag att du kopplar in en tryckknapp på D-ingången av en vippa
  - Användaren kan trycker knappen **när som helst**, även under setup- och hold-tiden!
  - Risken är att vippan hamnar i ett metastabilt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

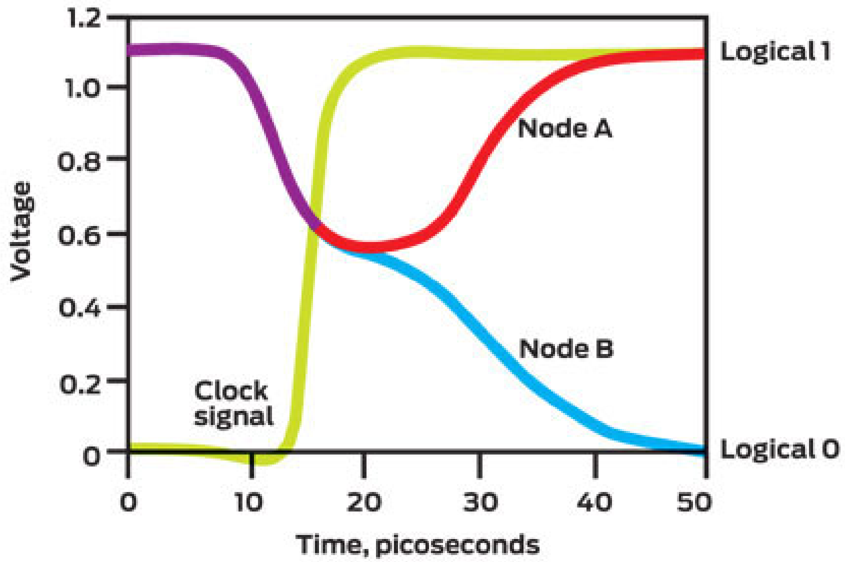
Synkronisering av insignaler

- För att synkronisera asynkrona ingångar använder man en extra vippa på ingången
- Den första vippans utgång (A) kan hamna 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!



(Slumptal med metastabilitet?)





Intelprocessorer ”singlar slant” med följande krets. Innan klockpulsen blir ”1” är både node A och node B logiskt ”1”. När klockpulsen kommer hamnar båda inverterarna i det metastabila tillståndet och **slumpen** avgör sedan vilket tillstånd inverterarna slutgiltigen hamnar i.

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