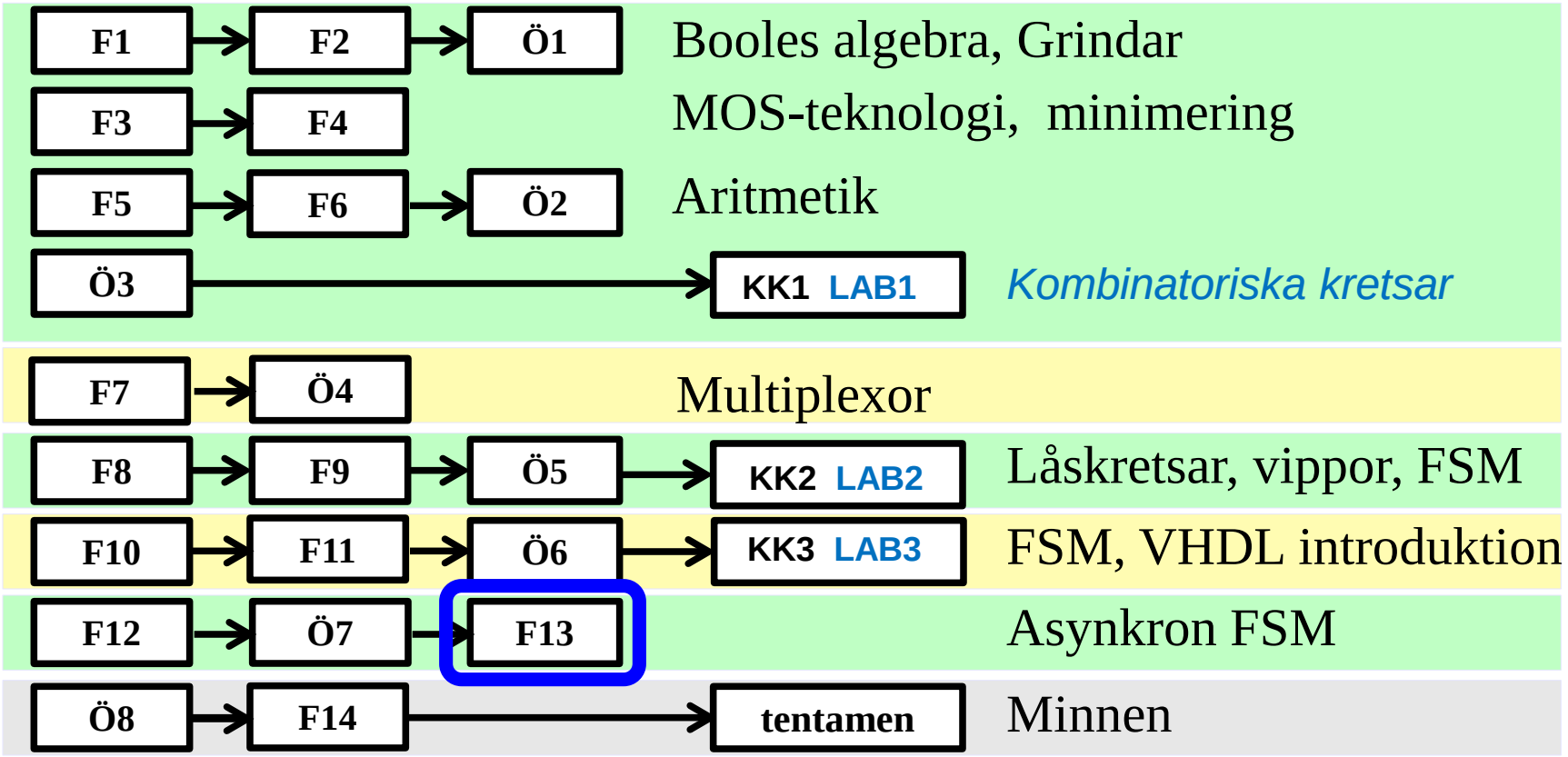
Digital Design IE1204

F13 Asynkrona sekvensnät
del 2

william@kth.se

IE1204 Digital Design



Föreläsningar och övningar bygger på varandra! Ta alltid igen det Du missat!
Läs på i förväg – delta i undervisningen – arbeta igenom materialet efteråt!

William Sandqvist william@kth.se

Detta har hänt i kursen …

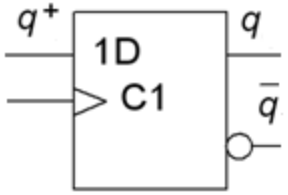
Decimala, hexadecimala, oktala och binära talsystemen
AND OR NOT EXOR EXNOR Sanningstabell, mintermer Maxtermer PS-form Booles algebra SP-form deMorgans lag Bubbelgrindar Fullständig logik NAND NOR CMOS grindar, standardkretsar Minimering med Karnaugh-diagram 2, 3, 4, 5, 6 variabler
Registeraritmetik tvåkomplementrepresentation av binära tal
Additionskretsar Multiplikationskrets Divisionskrets
Multiplexorer och Shannon dekomposition Dekoder/Demultiplexor Enkoder Prioritetsenkoder Kodomvandlare
VHDL introduktion
Vippor och Låskretsar SR-latch D-latch D-vippa JK-vippa T-vippa Räknare Skiftregister Vippor i VHDL Moore-automat Mealy-automat Tillståndskod Oanvända tillstånd Analys av sekvensnät Tillståndsminimering Tillståndsmaskiner i VHDL

Asynkrona sekvensnät flödestabell exitationstabell tillståndskodning

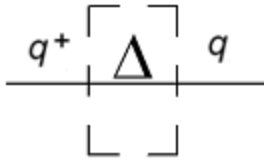
William Sandqvist william@kth.se

Repetition delay-element

Synkront sekvensnät
Klockad vippa



Asynkront sekvensnät
ett konstgrepp:
Delay-element



Andra beteckningar: *Y* och *y*

William Sandqvist william@kth.se

Gyllene regeln

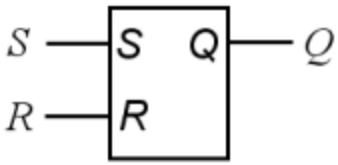
William Sandqvist william@kth.se

Exitationstabell

Den asynkrona kodade tillståndstabellen kallas för ***Excitationstabell***

De stabila tillstånden

(*de med next state = present state*) *”ringas in”*



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

$Y = y$

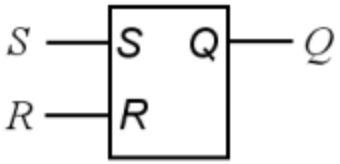
William Sandqvist william@kth.se

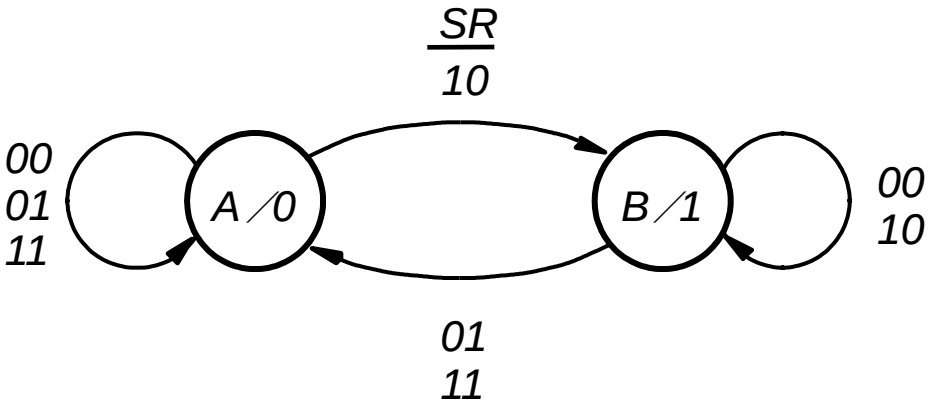
Flödestabell och Tillståndsdiagram

Den asynkrona okodade tillståndstabellen kallas för

Flödestabell

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





William Sandqvist william@kth.se

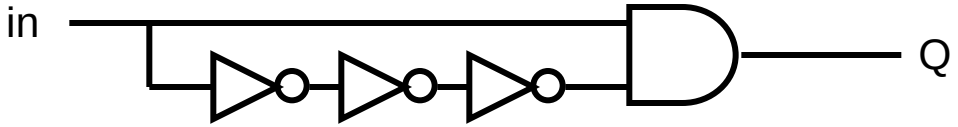
William Sandqvist william@kth.se

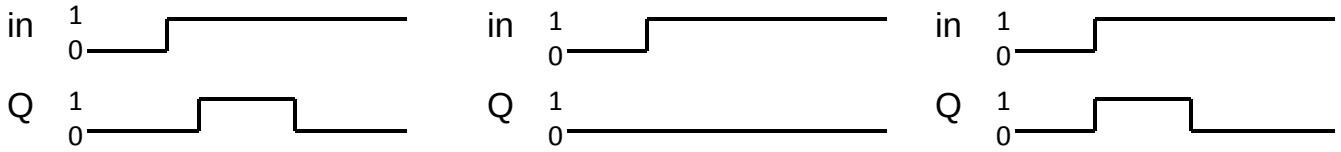
Hasard ”spikar”

- 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

Snabbfråga

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





Alt: A

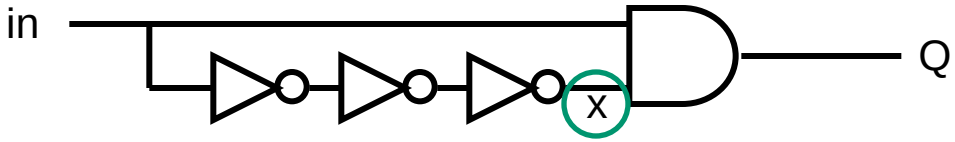
Alt: B

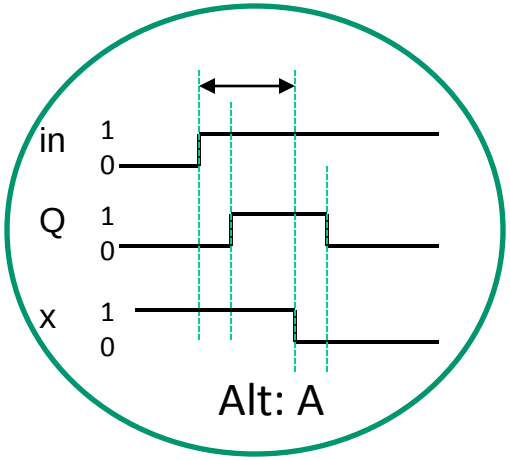
Alt: C

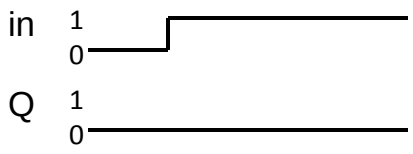
William Sandqvist william@kth.se

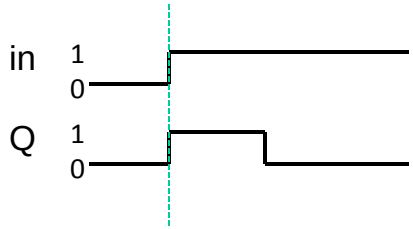
Snabbfråga

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







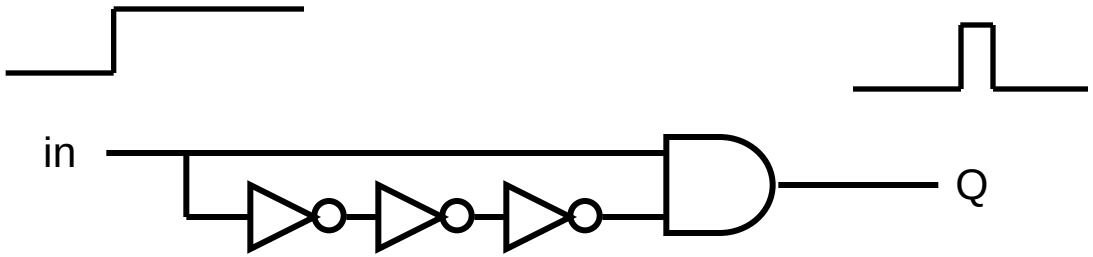


PGA fördröjning i inverterare blir båda ingångarna till AND grinden 1 ett kort tag

Alt C tar ej hänsyn till fördröjning i AND grind

William Sandqvist william@kth.se

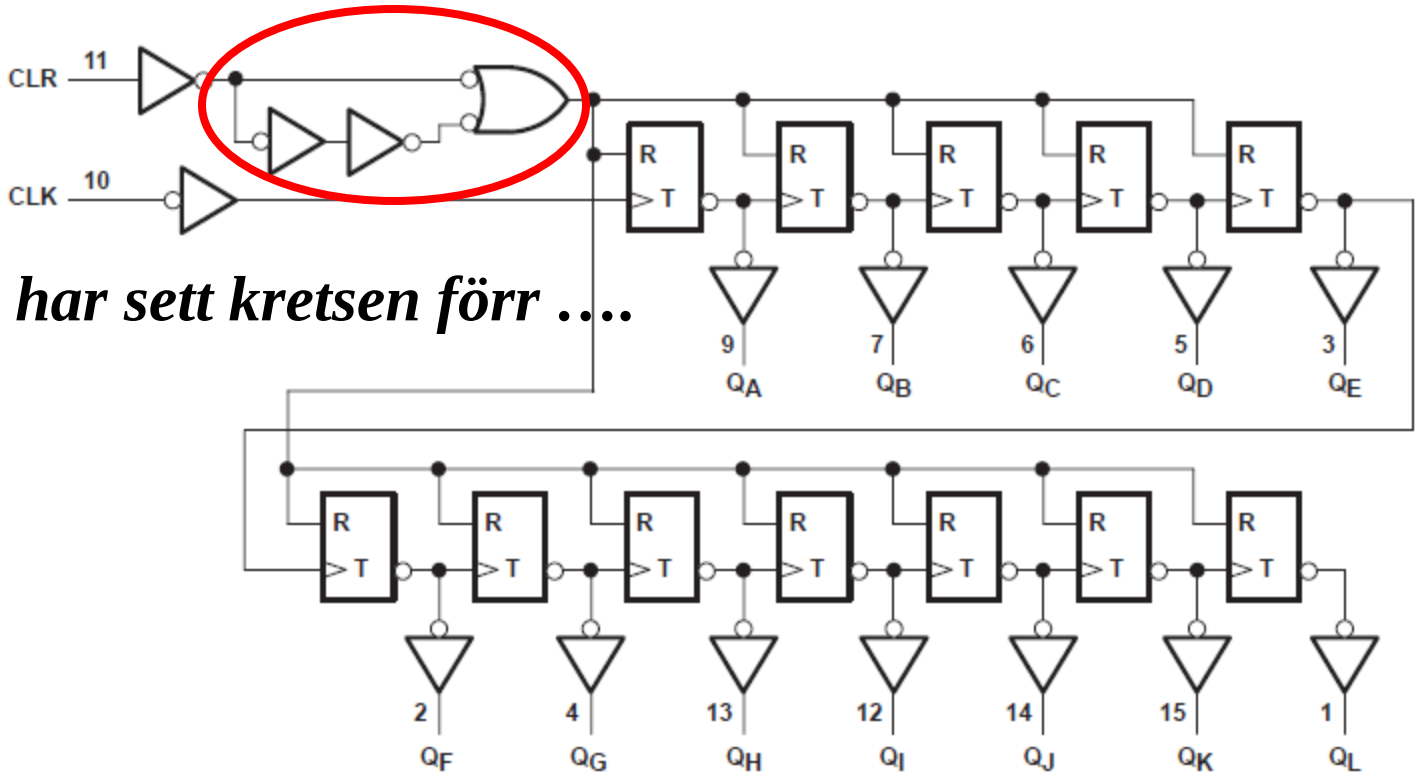
(Kort 0-ställningspuls)



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

William Sandqvist william@kth.se

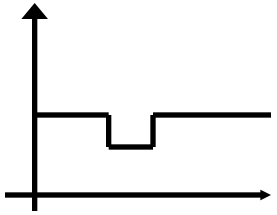
(Kort 0-ställningspuls)



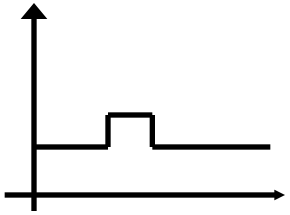
Du har sett kretsen förr ….

William Sandqvist william@kth.se

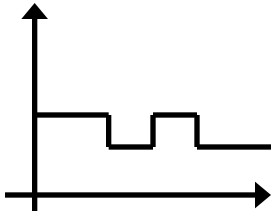
Olika typer av Hasard



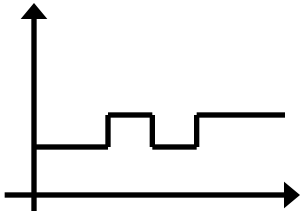
Statisk $1 \rightarrow 1$



Statisk $0 \rightarrow 0$



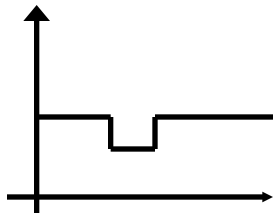
Dynamisk $1 \rightarrow 0$



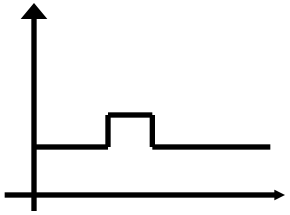
Dynamisk $0 \rightarrow 1$

William Sandqvist william@kth.se

Statisk Hasard



Statisk 1 $\rightarrow$ 1

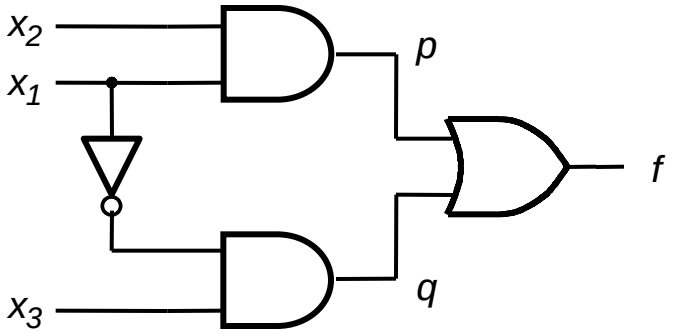


Statisk 0 $\rightarrow$ 0

William Sandqvist william@kth.se

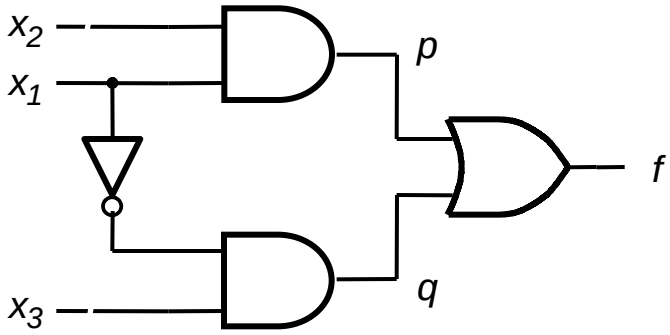
Exempel, Statisk Hasard

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

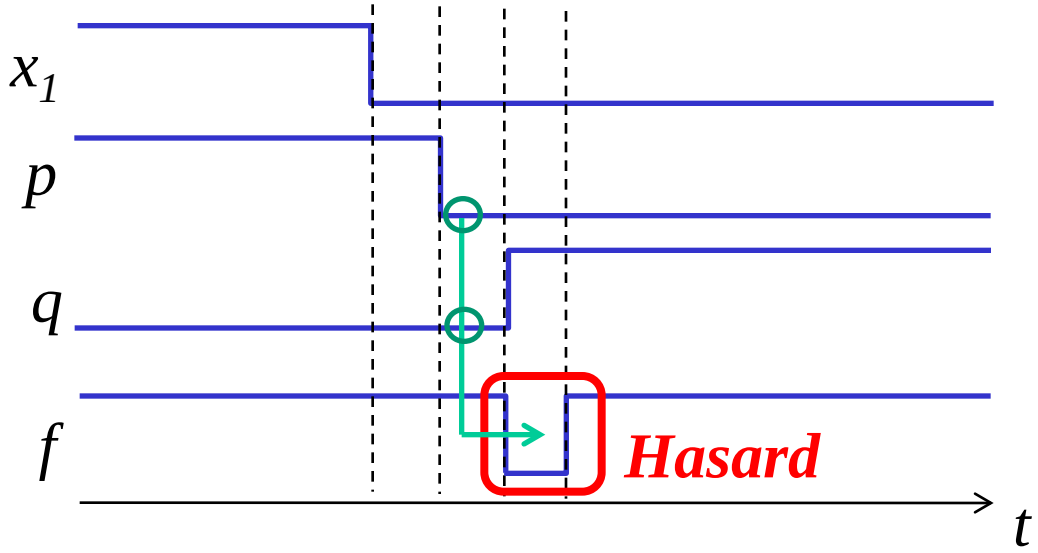


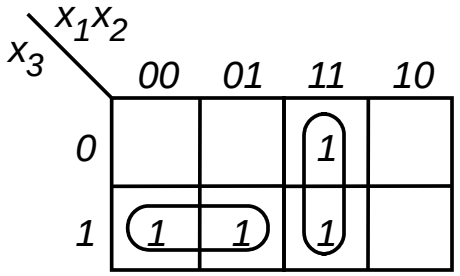
$$f = x_1x_2 + \overline{x}_1x_3$$

Tidsdiagrammet



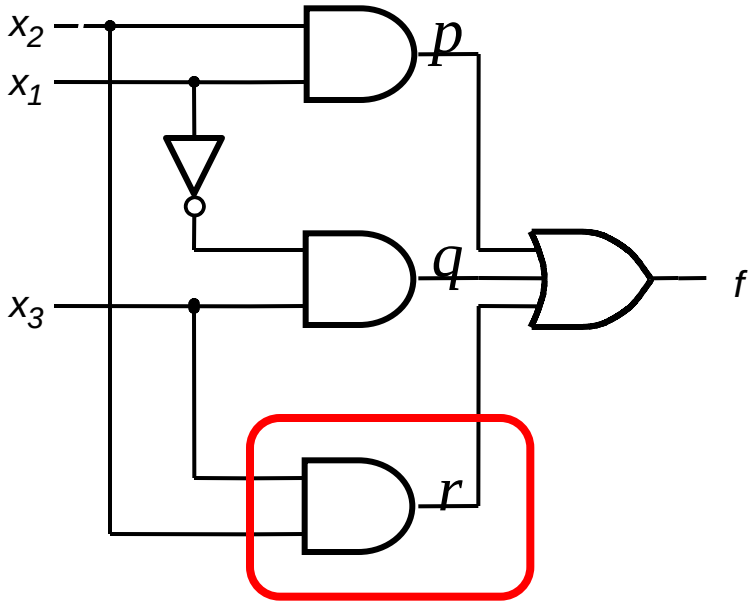
$$f = x_1 x_2 + \bar{x}_1 x_3$$



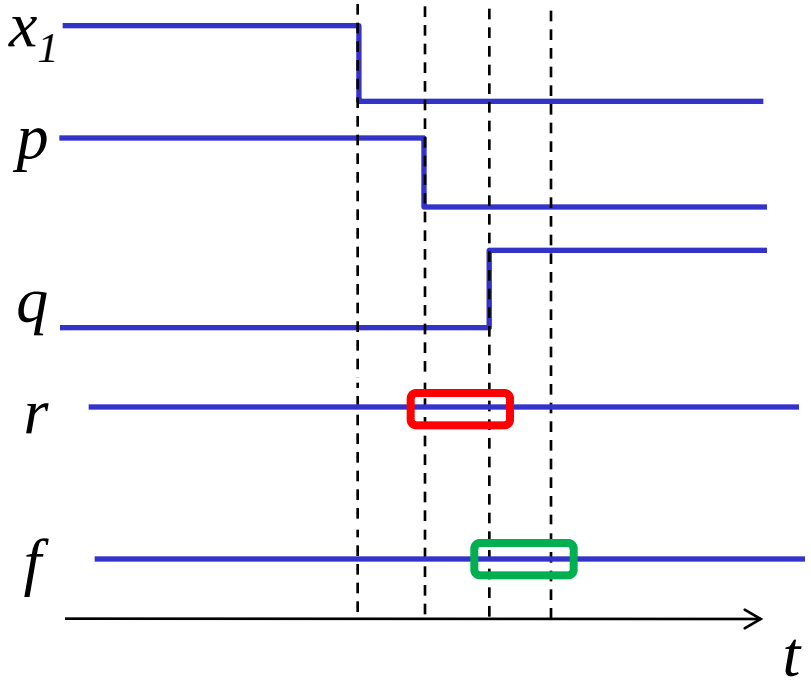


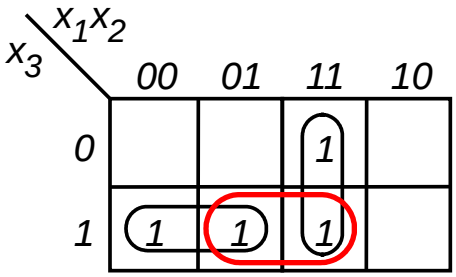
William Sandqvist william@kth.se

Hasardfri krets



Hasard-Cover





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

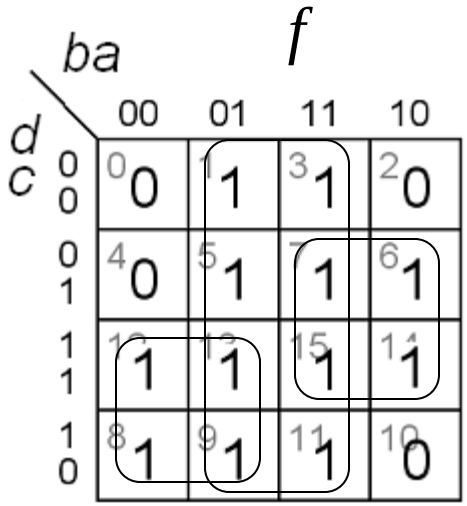
William Sandqvist william@kth.se

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?

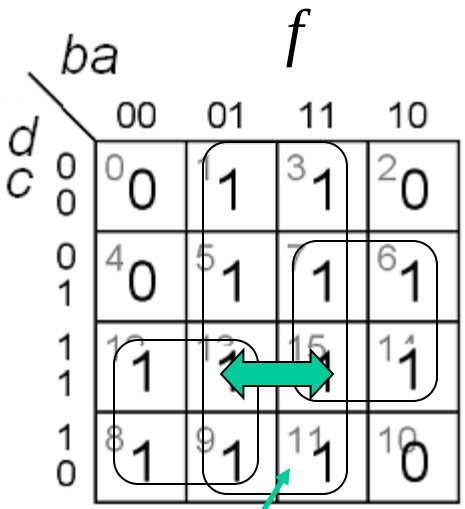


$$f = a + d\bar{b} + cb$$

William Sandqvist william@kth.se

Ex. Hasardfria hoptagningar

Räcker dessa hoptagningar för hasardfrihet?



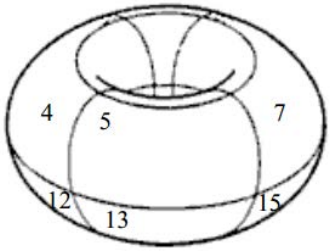
$$f = a + d\bar{b} + cb$$

Hasard cover

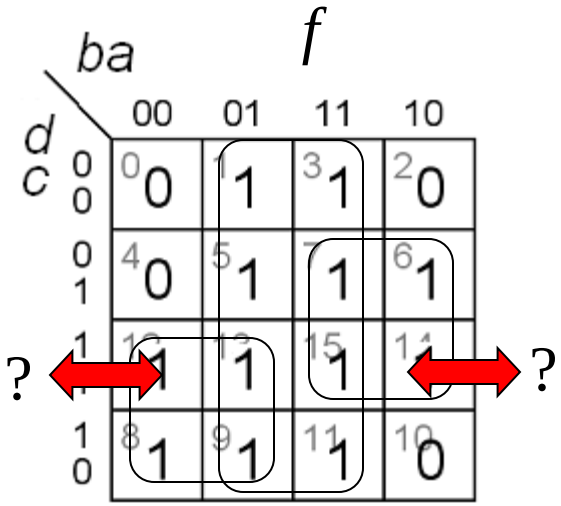
William Sandqvist william@kth.se

Ex. Hasardfria hoptagningar

Räcker dessa hoptagningar för hasardfrihet?



Karnaugh-diagrammet är en doughnut!

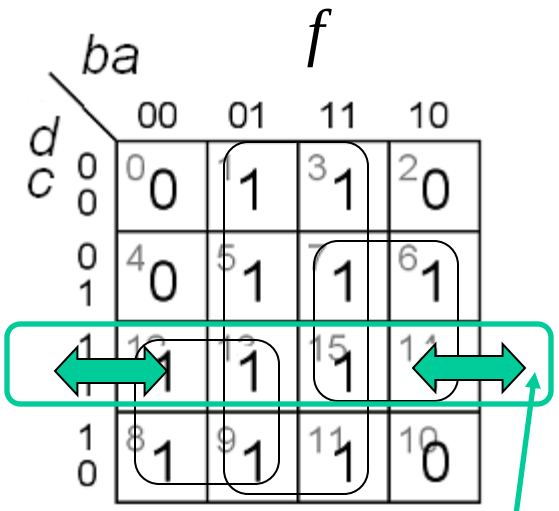


$$f = a + d\bar{b} + cb$$

William Sandqvist william@kth.se

Ex. Hasardfria hoptagningar

Räcker dessa hoptagningar för hasardfrihet?

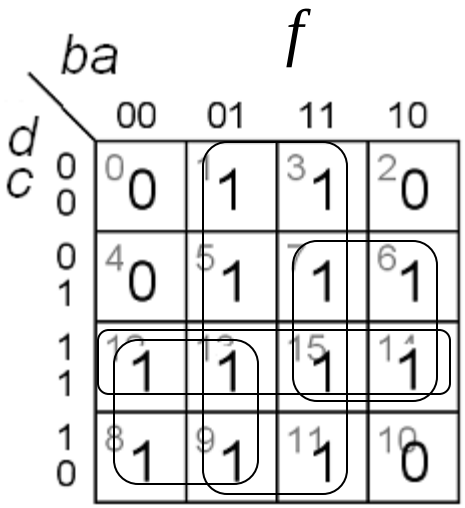


$$f = a + d\bar{b} + cb + dc$$

William Sandqvist william@kth.se

Ex. Hasardfria hoptagningar

Lätt att missa!

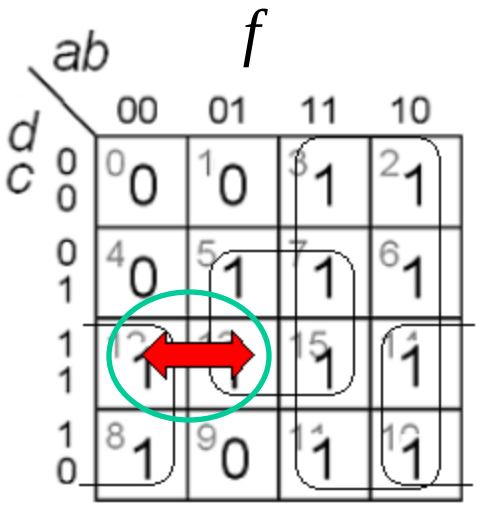


$$f = a + d\bar{b} + cb + dc$$

Hasard cover

Ex. Hasardfria hoptagningar

Med annan variabelordning missar man inte!



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

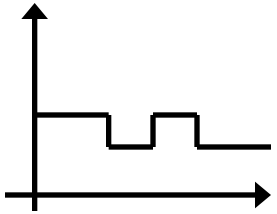
William Sandqvist william@kth.se

Statisk hasard vid POS?

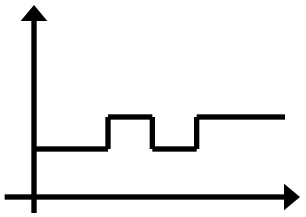
- Har man en POS-implementering så måste man se till att alla bredvidliggande 0:or är täckta av en egen summaterm

Dynamisk Hasard?

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



Dynamisk $1 \rightarrow 0$



Dynamisk $0 \rightarrow 1$

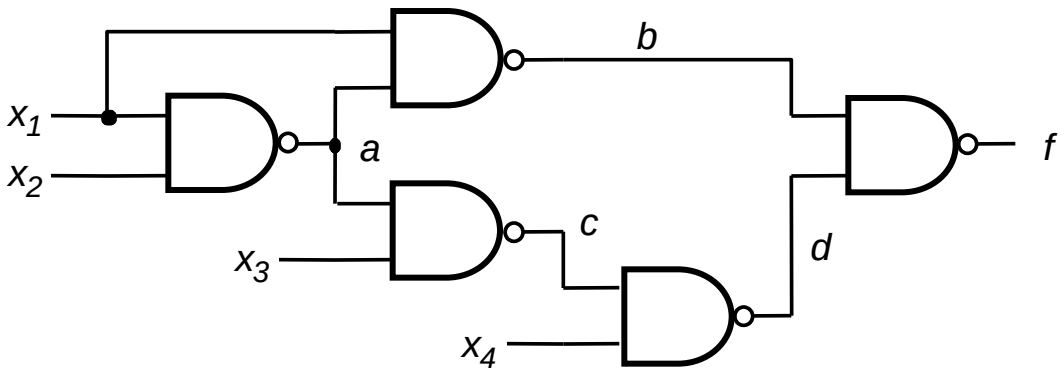
Exempel, Dynamisk Hasard

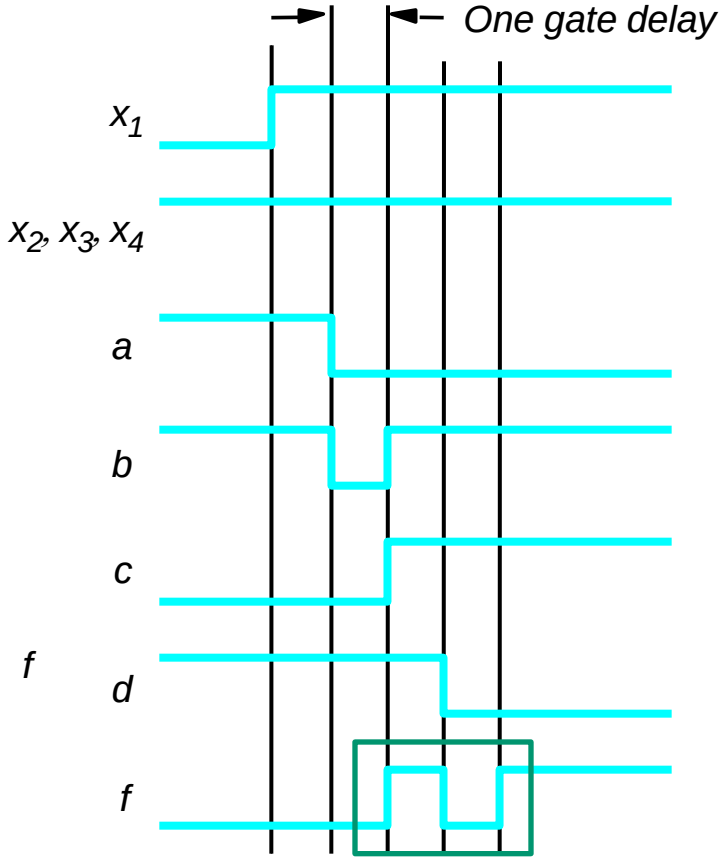
- Följande ekvation osakar ingen hasard om man implementerar den som en AND-OR-struktur

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

Exempel, Dynamisk Hasard

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



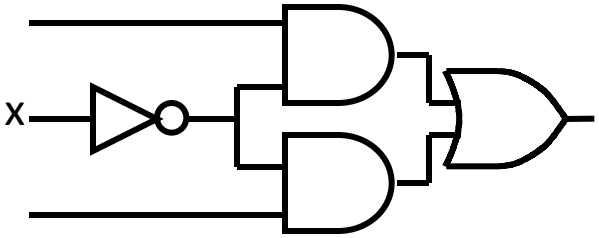


Hur undviks Dynamisk Hasard?

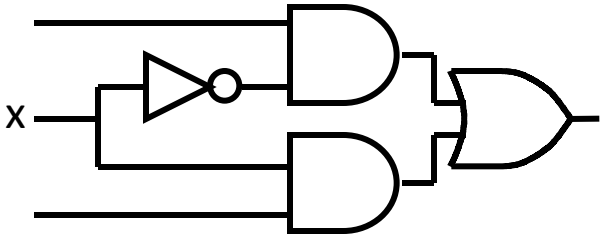
- Dynamisk hasard kan undvikas med **två-nivå-logik**
- Ser man till att en två-nivå krets är fri från statisk hasard, så finns det *inte* heller någon dynamisk hasard!

Snabbfråga

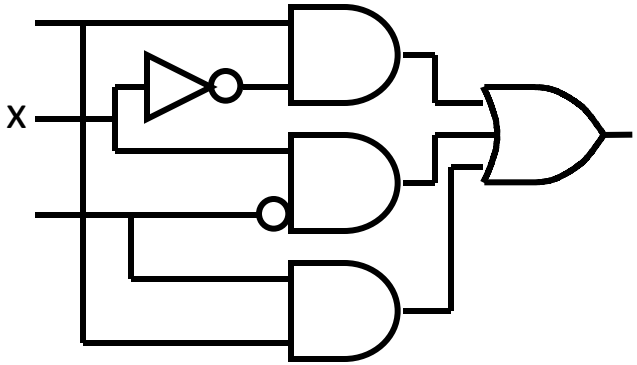
Vilket/vilka av följande grindnät kan ge upphov till hazard då ***x*** *ändras ?*



Alt: B



Alt: A



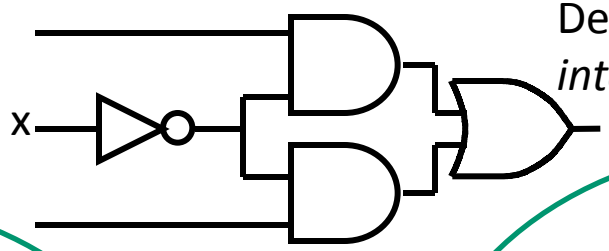
Alt: C

William Sandqvist william@kth.se

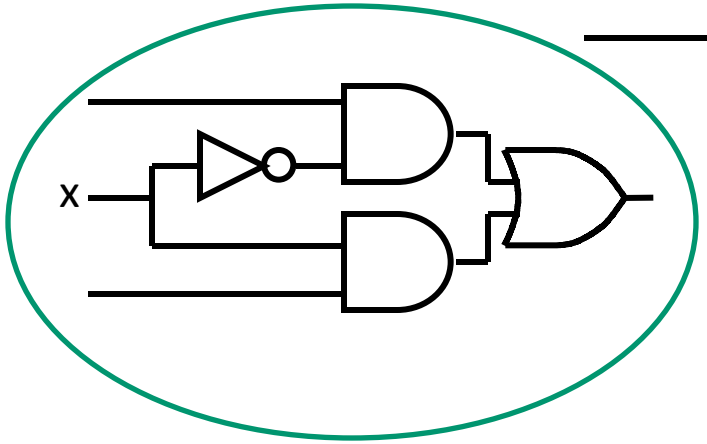
Snabbfråga

*Vilket/vilka av följande grindnät kan ge upphov till hazard då **x** ändras ?*

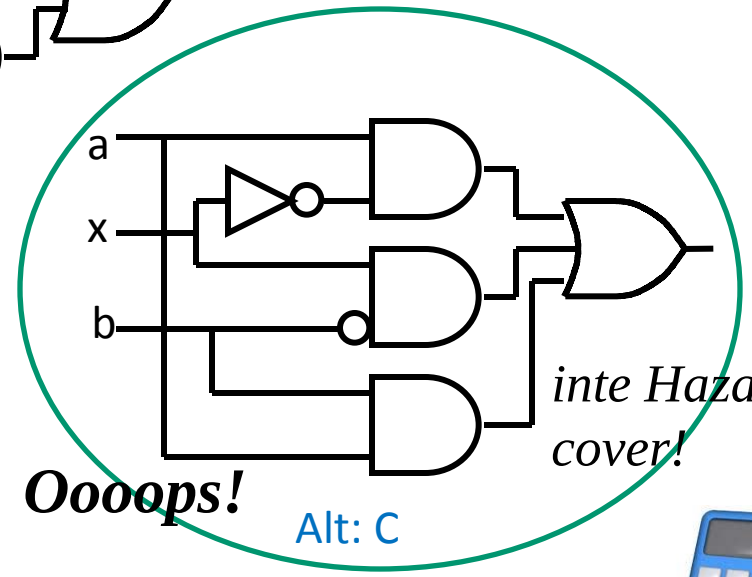
Risk för hazard då a=1 och b=0
Den extra grinden täcker
inte detta fall (utan a=1 och b=1)



Alt: B



Alt: A



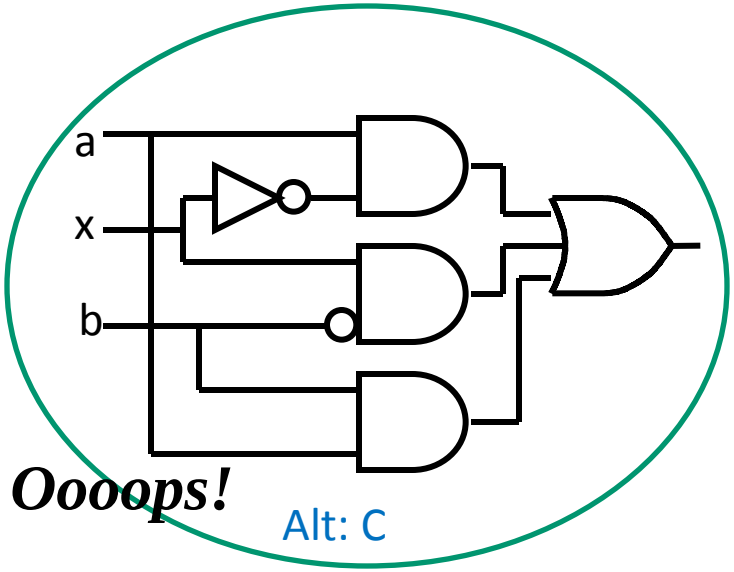
Oooops!

Alt: C

William Sandqvist william@kth.se

Snabbfråga

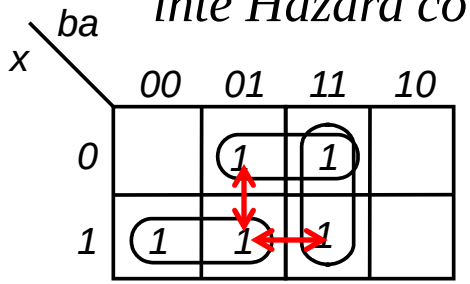
Risk för hazard då a=1 och b=0
Den extra grinden täcker
inte detta fall (utan a=1 och b=1)



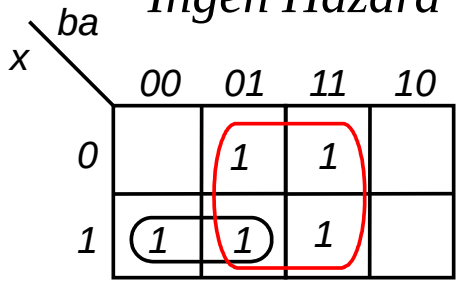
Oooops!

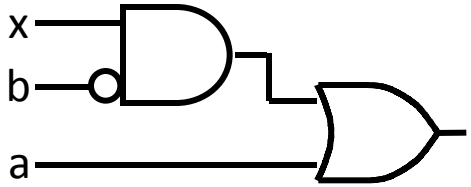
Alt: C

inte Hazard cover!



Ingen Hazard



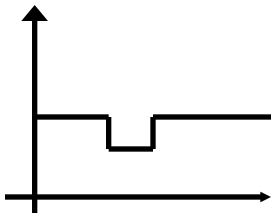


William Sandqvist william@kth.se

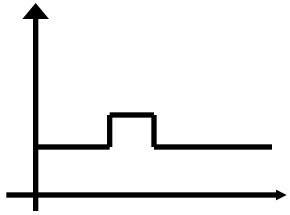
När behöver man ta hänsyn till Hasard?

- I ett ***asynkront sekvensnät*** måste 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

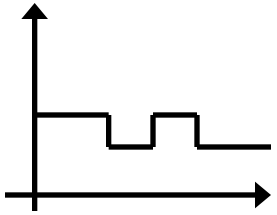


Statisk 1 → 1

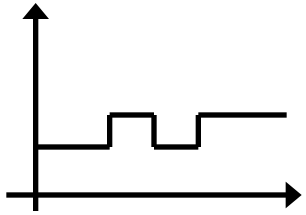


Statisk 0 → 0

Statisk hasard orsakas av utelämnade primimplikanter



Dynamisk 1 → 0

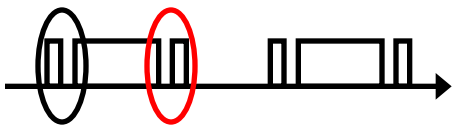


Dynamisk 0 → 1

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

William Sandqvist william@kth.se

Utgångs-spikar i asynkrona sekvensnät



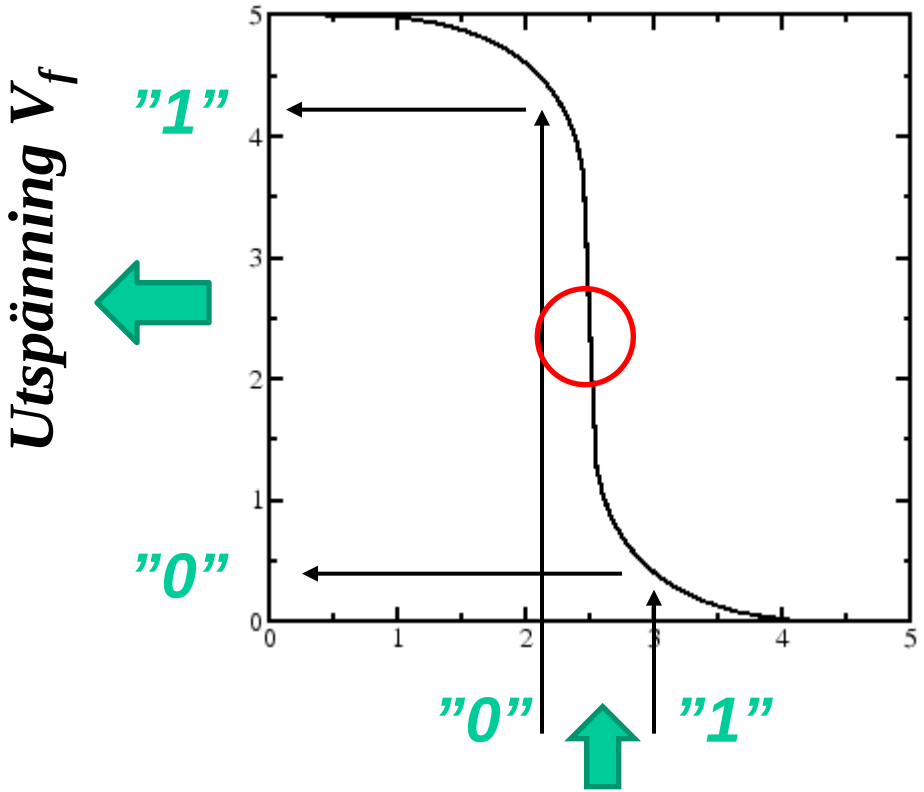
Pres state	Next State		Q
	X=0	1	
y_2y_1	Y_2Y_1		
00	00	01	0
01	00	11	1
11	01	10	0
10	11	10	1

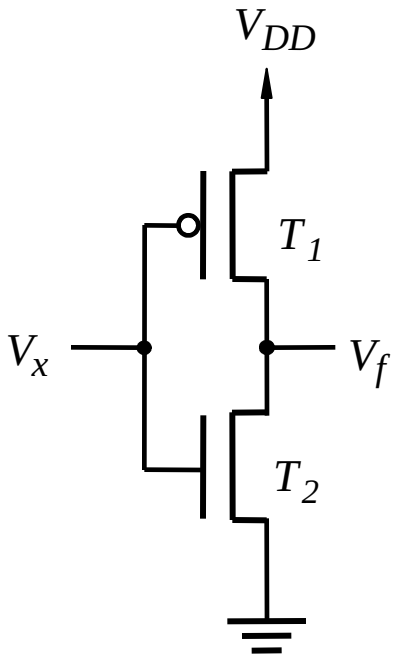
Man kan få utgångsspikar i ett asynkront sekvensnät när man byter från ett stabilt tillstånd till ett annat genom att passerar flera instabila tillstånd (Fenomenet är *ingen* hasard!).

William Sandqvist william@kth.se

Metastabilitet

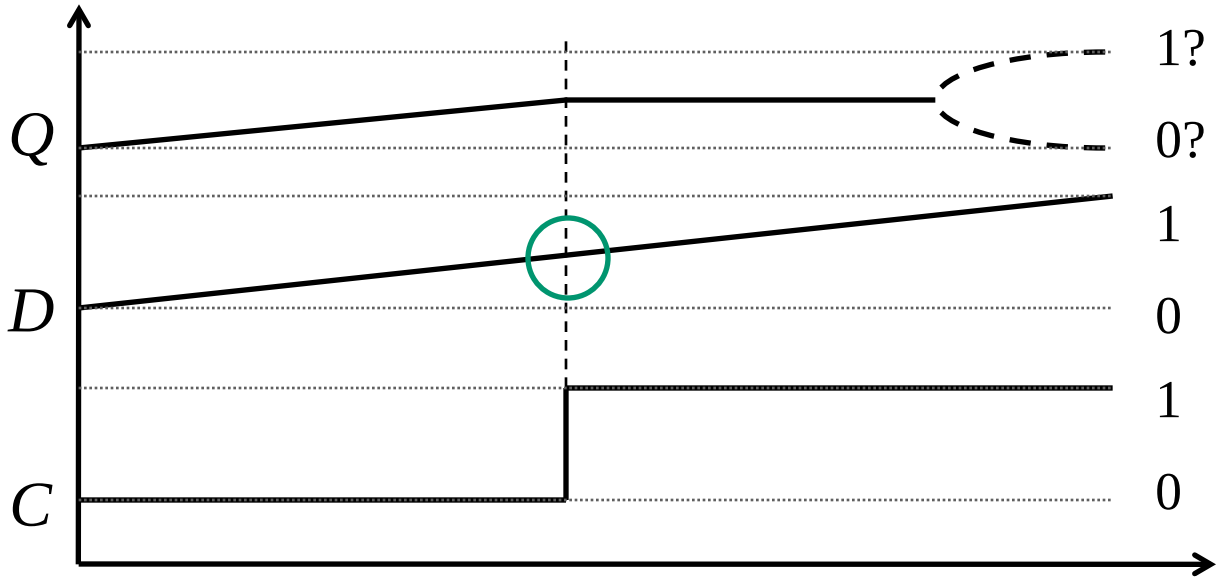
CMOS-kretsens överföringsfunktion (ex. inverterare)





William Sandqvist william@kth.se

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 $V_{DD}/2$.

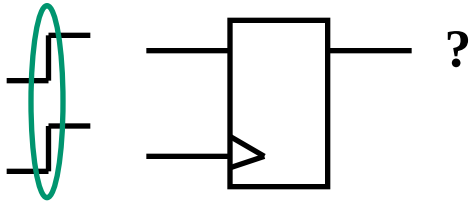
Då låser sig latchen vid det spänningsvärde som råkar finnas på *D*.
Efter en tid slår latchen om till antingen ’1’ eller ’0’.

William Sandqvist william@kth.se

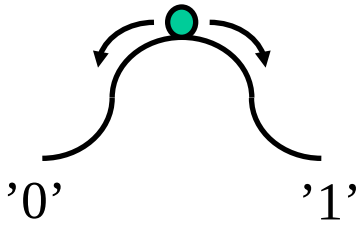
Om metastabilitet …

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.



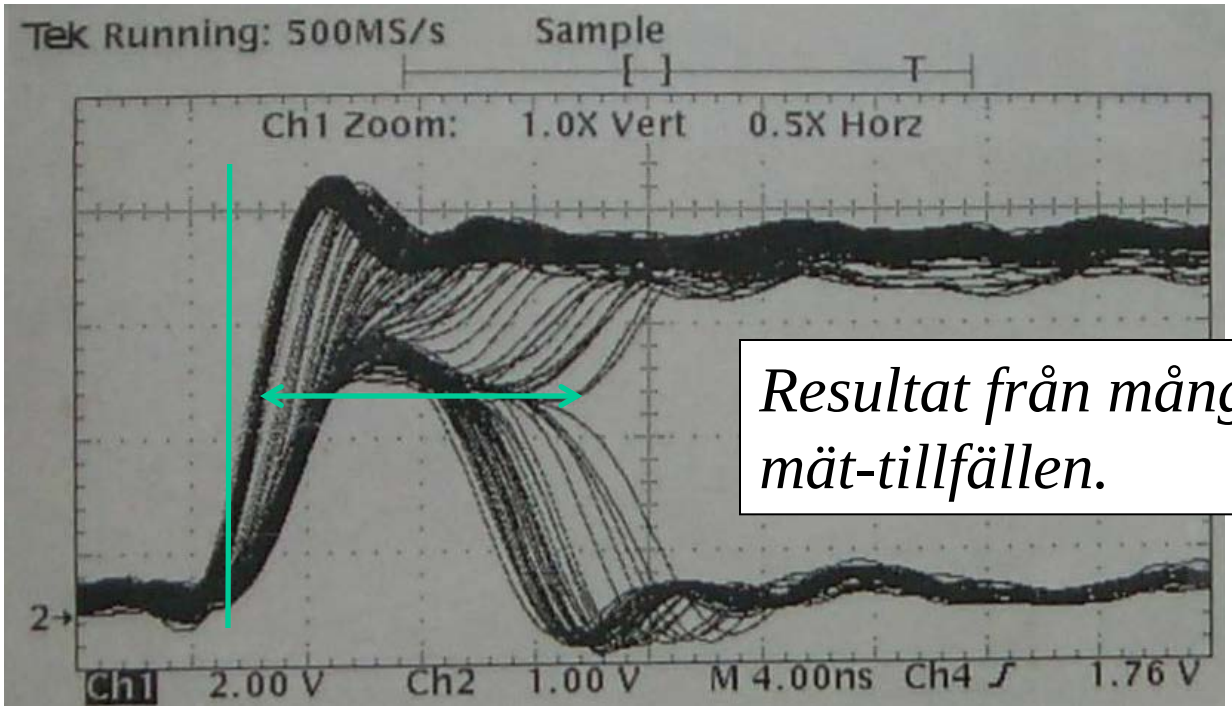
Om *Clk* och *D* switchar samtidigt, vilket värde får då *Q*?



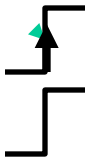
På vilken sida kommer bollen att trilla ner?

William Sandqvist william@kth.se

Klockpuls och data samtidigt!



Resultat från många mät-tillfällen.



Klocka och data ändras samtidigt!

William Sandqvist william@kth.se

Hur mycket samtidigt?

∞ *exakt samtidigt – oändlig tid …*

Tid innan utvärdet blir bestämt

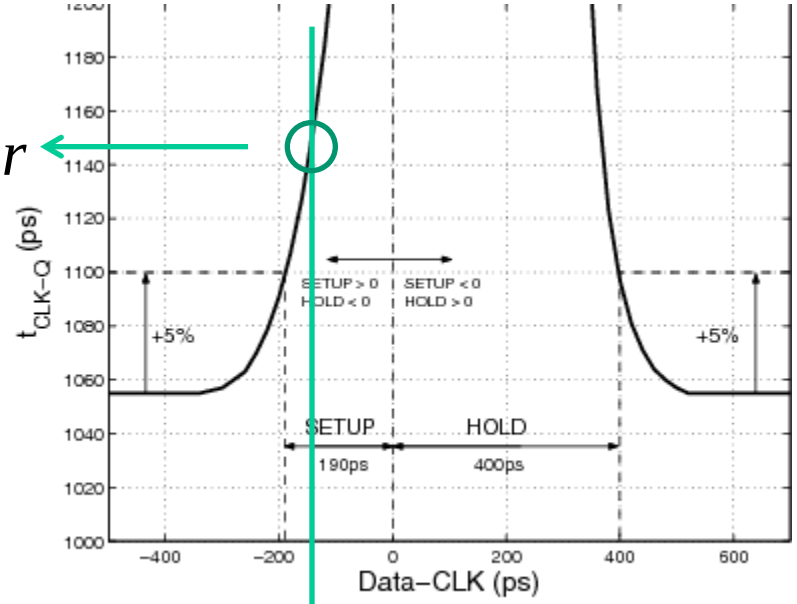


Figure 1. Definitions of setup and hold times.

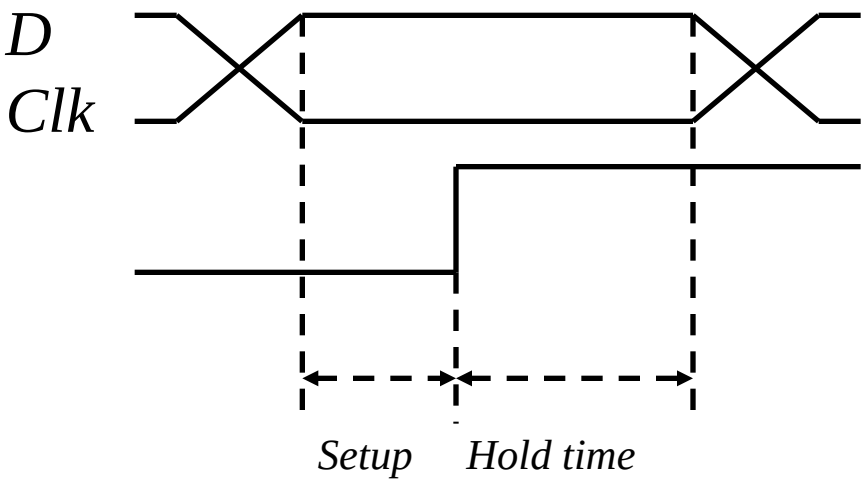
data ⎍‾⎍ *klocka*

hur nära?

William Sandqvist william@kth.se

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

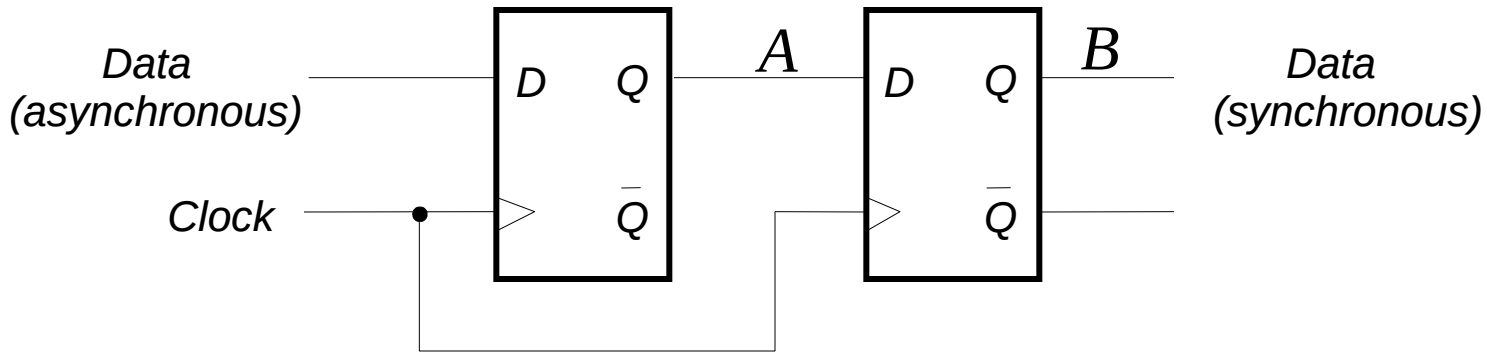
Om Setup and Hold time's är uppfyllda, så kommer vippan (Flip-flop) att garanterat bete sig snällt/deterministiskt!

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!

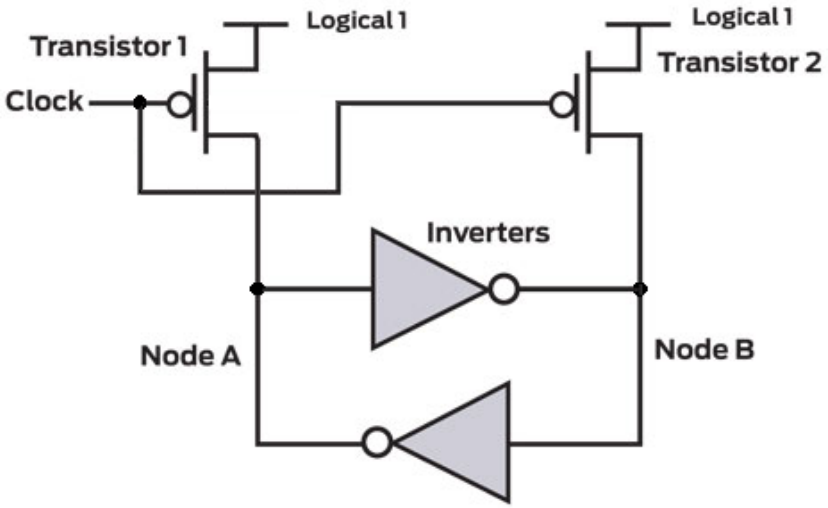
Synkronisering av insignaler

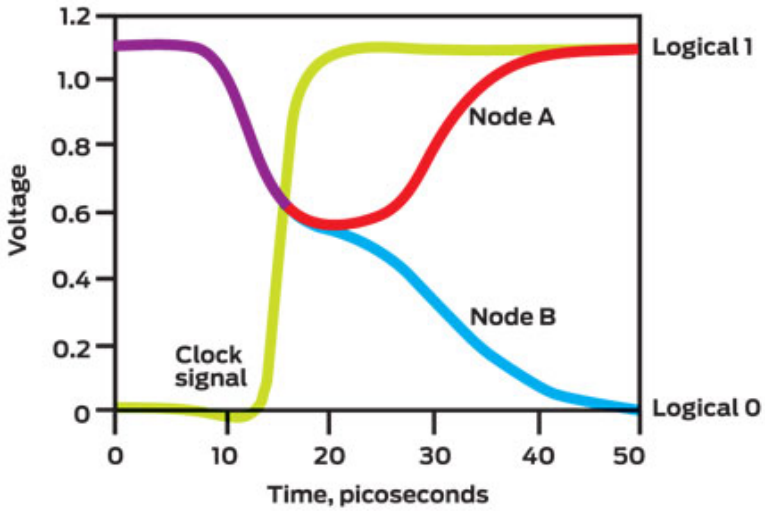
- För att synkronisera asynkrona ingångar användar man en extra vippa på ingången
- Den första vippans utgång (A) kan hamna i ett metastabilt läge
- Men om klockperioden är tillräckligt lång, så kommer den att stabiliseras innan nästa klockflank, så att B inte hamnar i ett metastabilt läge!



William Sandqvist william@kth.se

(Slumptal med metastabilitet?)



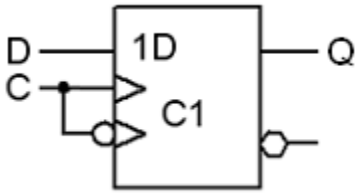


Intelprocessorer ”singlar slant” med följande krets. Innan klockpulsen blir ”1” är både node A och node B logiskt ”1”. När klockpulsen kommer hamnar båda inverterarna i det metastabila tillståndet och ***slumpen*** *avgör sedan vilket tillstånd inverterarna slutgiltigen hamnar i.*

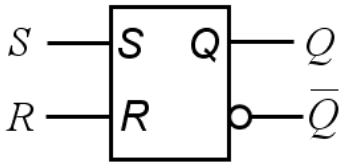
William Sandqvist william@kth.se

Avancerade byggelement

De asynkrona vipporna och låskretsarna används som säkra byggelement vid digital design. Nya byggelement utvecklas hela tiden.



- Vid övningen kommer vi att konstruera en dubbel-flankvippa – en vipptyp som kan komma att ge framtidens datorkretsar högre (dubblerade) prestanda



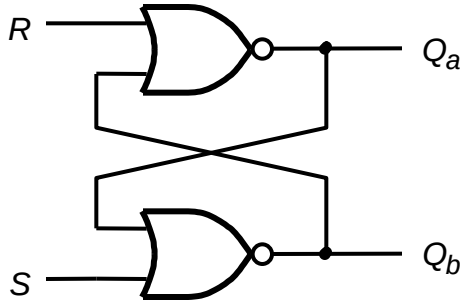
- Vid föreläsningen förfinar vi nu den enkla SR-låskretsen

Exempel – förbättrad SR-latch

- **Konstruktion av set-dominant SR-latch**
- **Specifikation**
  - Konstruktionen är en speciell typ av SR-latch (det finns ***inte*** ett förbjudet läge 11)
    - Om *S* och *R* är 1 så går latchen i SET-läge (*Q* = 1)
    - Latchen kan först gå till RESET-läge om
      1. både *S* och *R* först sätts till *S*=0 och *R*=0 (*Q* = 1)
      2. *R* aktiveras (*S* = 0, *R* = 1) $\rightarrow$ *Q* = 0 !

Källa: "Fletcher: Engineering Approach to Digital Design", Prentice-Hall, 1980. Exempel 10.5 (pp 670).

Repris: SR-latch



(a) Circuit

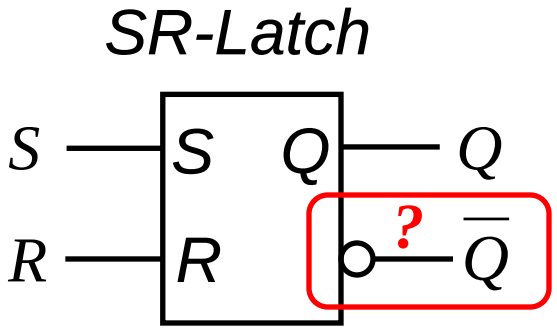
S	R	Q_a	Q_b	
0	0	0/1	1/0	(no change)
0	1	0	1	
1	0	1	0	
1	1	0	0	

(b) Truth table

Förbjuden insignal $S=R=1$

$Q_a \neq \overline{Q}_b$

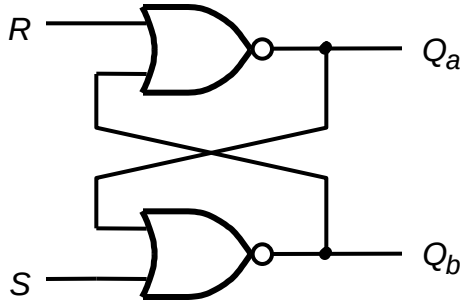
Så länge man **undviker** insignalen $S = R = 1$ (= förbjudet tillstånd) kommer utgångarna Q_a och Q_b att vara varandras inverser. Man kan då använda symbolen till höger.



Tar man signaler från låskretsar finns det således alltid inverser att tillgå!

William Sandqvist william@kth.se

Mer problem med SR-latchen



(a) Circuit

S	R	Q_a	Q_b	
0	0	0/1	1/0	(no change)
0	1	0	1	
1	0	1	0	
1	1	0	0	

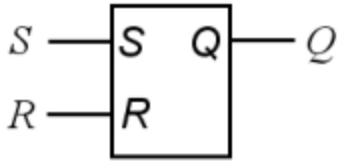
(b) Truth table

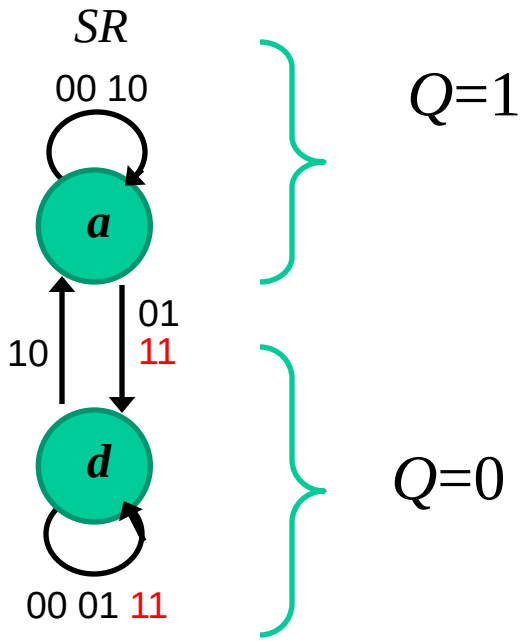
Om man vill gå från $SR = 11$ till $SR = 00$ är det en **dubbeländ-ring** av insignalerna. Därför hamnar vi antingen i $Q = 0$ eller i $Q = 1$ ingen kan veta!

- Detta är ytterligare ett skäl till att utesluta $SR = 11$.

William Sandqvist william@kth.se

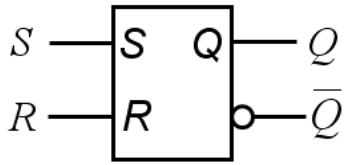
SR-latch

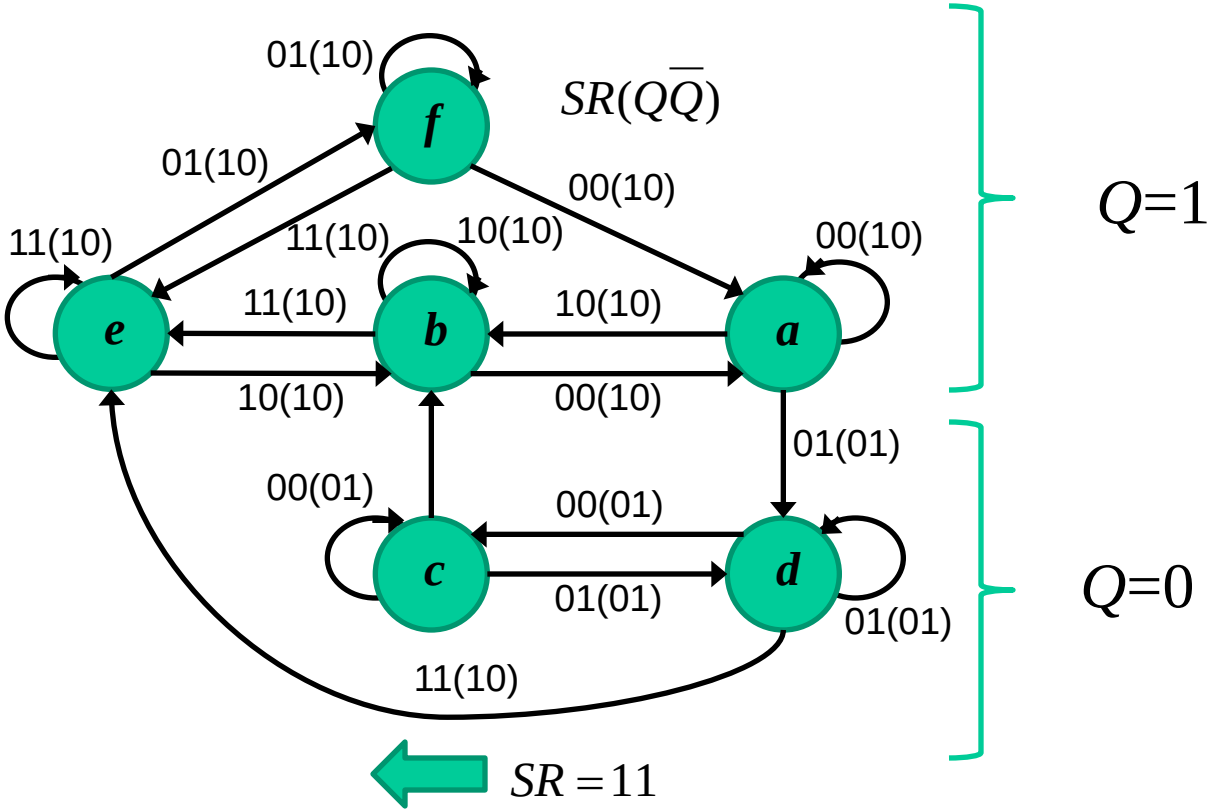




William Sandqvist william@kth.se

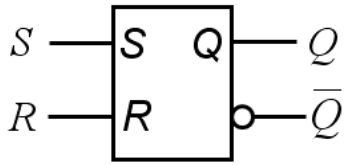
SET-dominant SR-latch

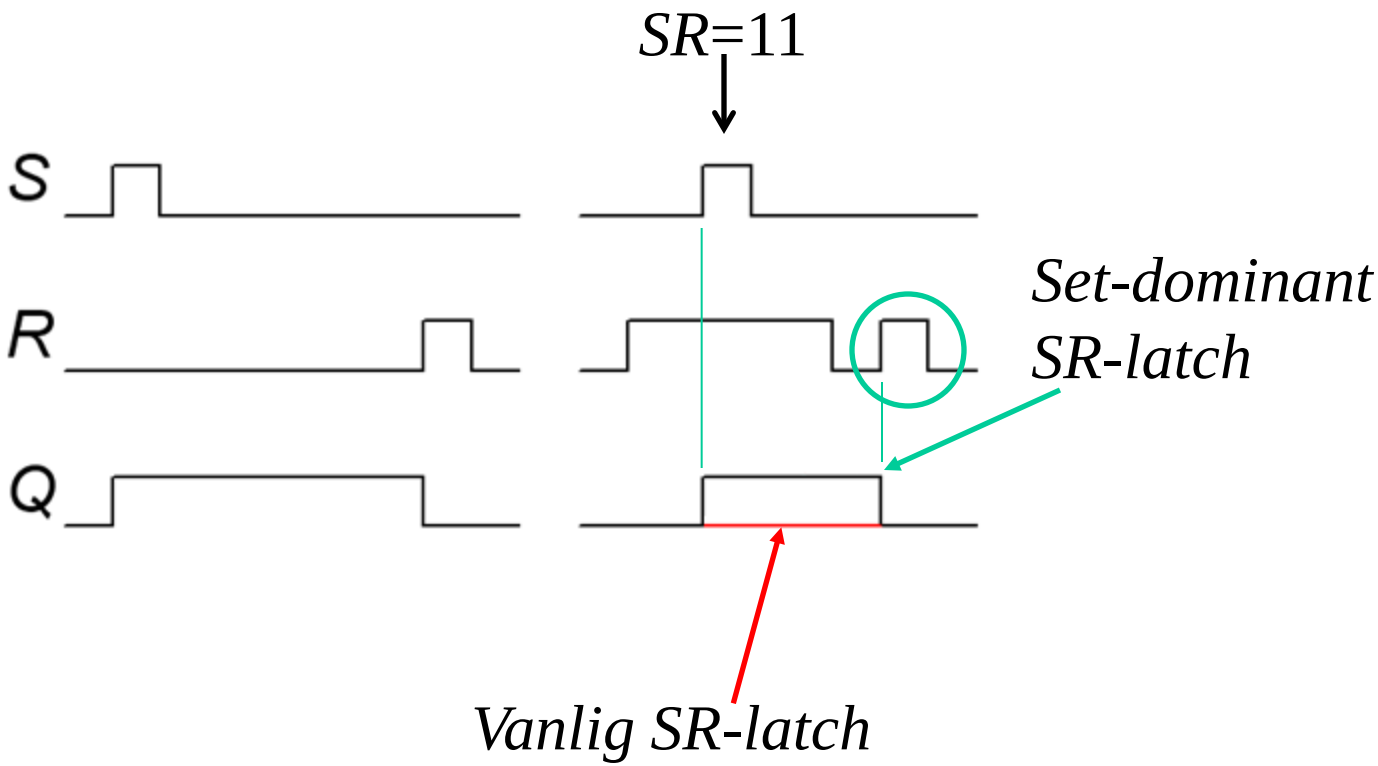




William Sandqvist william@kth.se

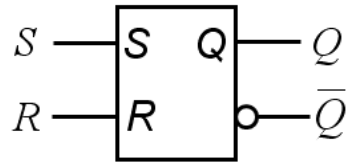
Önskat beteende

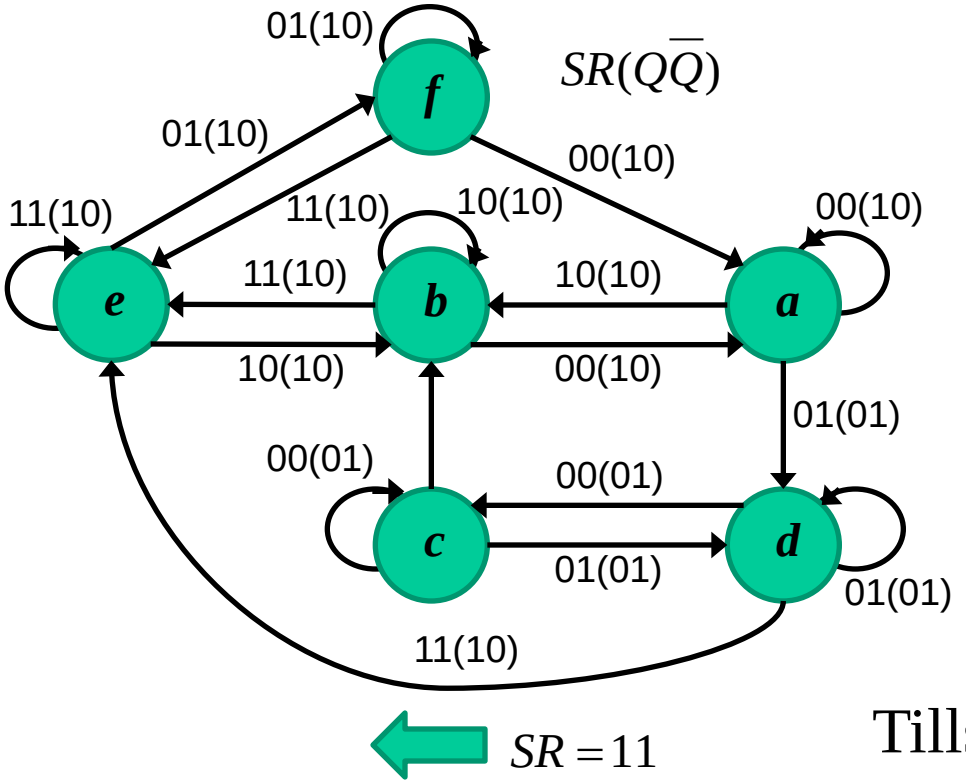




William Sandqvist william@kth.se

SET-dominant SR-latch





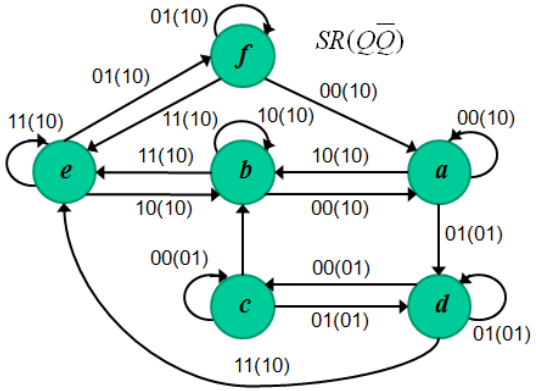
$SR(Q\overline{Q})$

	00	01	11	10	Q	$\overline{Q}$
a	(a)	d	–	b	1	0
b	a	–	e	(b)	1	0
c	(c)	d	–	b	0	1
d	c	(d)	e	–	0	1
e	–	f	(e)	b	1	0
f	a	(f)	e	–	1	0

Tillståndet ***e*** tar ”hand om” fallet $SR = 11$

William Sandqvist william@kth.se

Kompatibilitet



$SR(Q\bar{Q})$

	00	01	11	10	Q	$\bar{Q}$
a	(a)	d	–	b	1	0
b	a	–	e	(b)	1	0
c	(c)	d	–	b	0	1
d	c	(d)	e	–	0	1
e	–	f	(e)	b	1	0
f	a	(f)	e	–	1	0

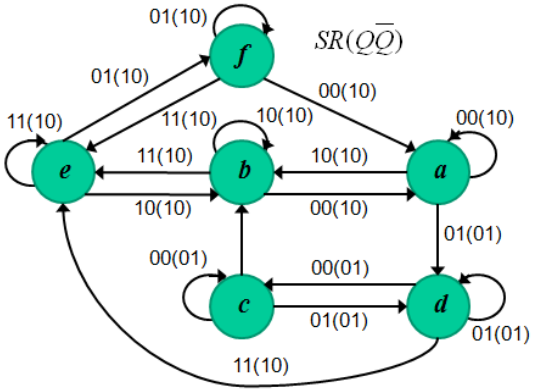
a (10) ***b*** (10) ***c*** (01) ***d*** (01) ***e*** (10) ***f*** (10)

***Det finns inga* ekvivalenta *tillstånd, finns det några Moore-* kompatibla *tillstånd* … ?**

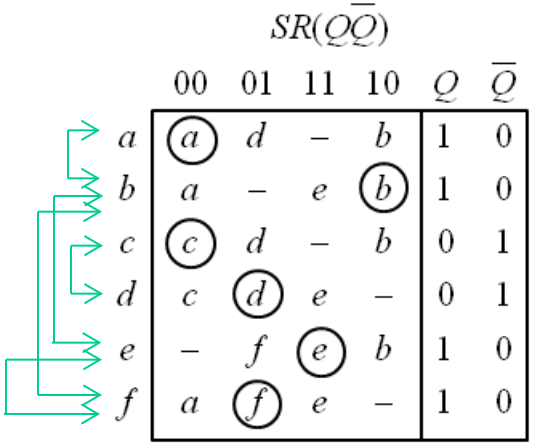
William Sandqvist william@kth.se

• Kompatibilitet

Använd ospecificerade tillstånd!



a (10) **b** (10) **c** (01) **d** (01) **e** (10) **f** (10)



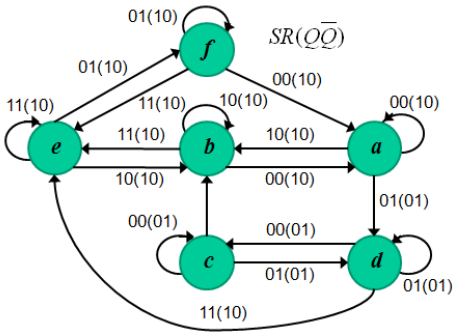
Många valmöjligheter …

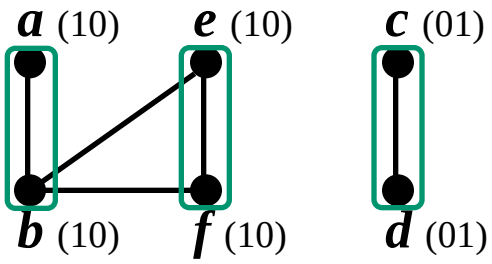
```
a(10): ad-b
b(10): a-eb

b(10): a-eb
f(10): afe-

b(10): a-eb
e(10): -feb

e(10): -feb
f(10): afe-

c(01): cd-b
d(01): cde-
```

William Sandqvist william@kth.se

Kompatibilitetsgraf

Många valmöjligheter …





$SR(Q\overline{Q})$

	00	01	11	10	Q	$\overline{Q}$
a	(a)	d	–	b	1	0
b	a	–	e	(b)	1	0
c	(c)	d	–	b	0	1
d	c	(d)	e	–	0	1
e	–	f	(e)	b	1	0
f	a	(f)	e	–	1	0

Nya beteckningar
a (ab), e (ef), c (cd)
Tre tillstånd kräver
två tillståndsvariabler
Y_2 och Y_1

$SR(Q\overline{Q})$

	00	01	11	10	Q	$\overline{Q}$
a	(a)	c	e	(a)	1	0
c	(c)	(c)	e	a	0	1
e	a	(e)	(e)	a	1	0
–	–	–	–	–	–	–

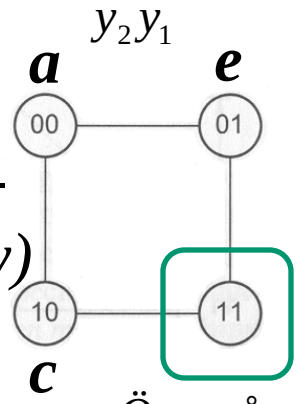
Reducerad flödestabell

William Sandqvist william@kth.se

Tillståndskodning

$SR(Q\bar{Q})$	00	01	11	10	Q	$\bar{Q}$
a	(a)	c	e	(a)	1	0
c	(c)	(c)	e	a	0	1
e	a	(e)	(e)	a	1	0
–	–	–	–	–	–	–

Vald tillstånds-kod (Gray)



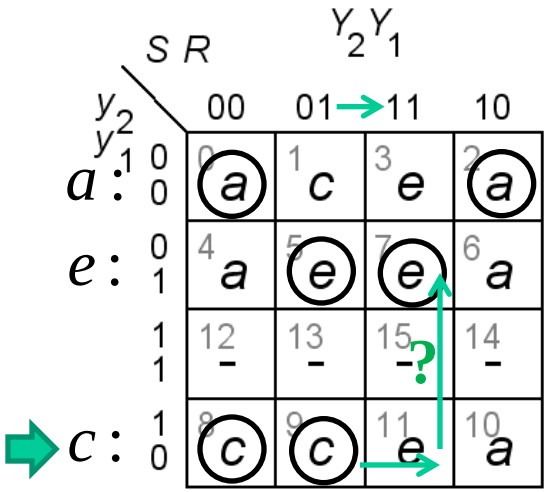
Övergångs-tillstånd

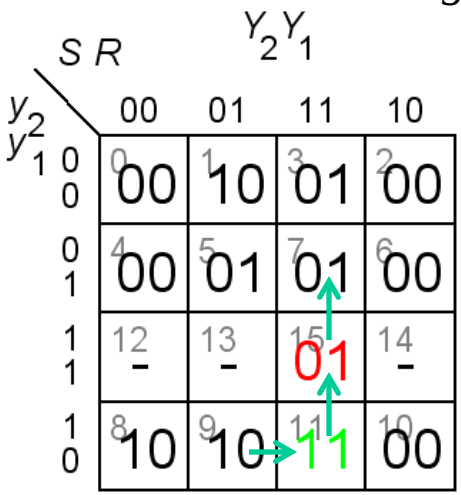
Utgångsavkodning

$a, e \quad Q = 1$

$c \quad Q = 0$

$Q = \bar{y}_2$

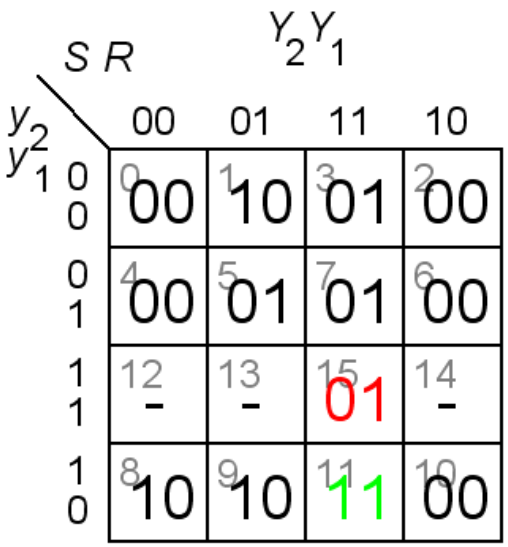


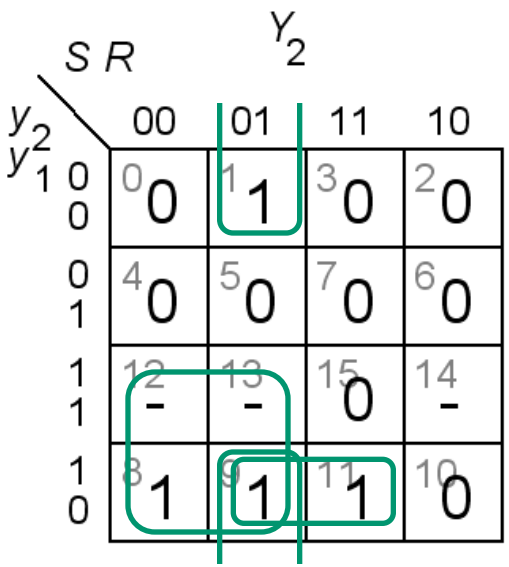


Från ***c*** till ***e*** krävs det en ”dubbeländring” av Y_2Y_1 detta ändras med hjälp av **övergångstillståndet** till två ”enkeländringar”

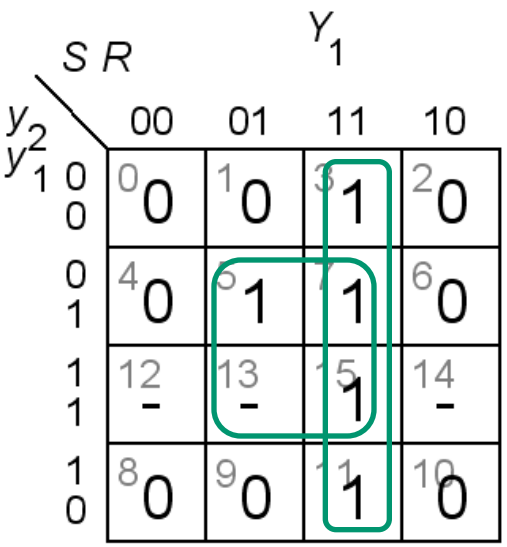
William Sandqvist william@kth.se

Karnaughdiagram





$$Y_2 = \overline{S}y_2 + \overline{S}R\overline{y}_1 + Ry_2\overline{y}_1$$

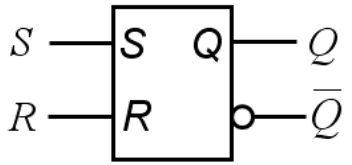


$$Y_1 = Ry_1 + SR$$

Hasardfria nät direkt!

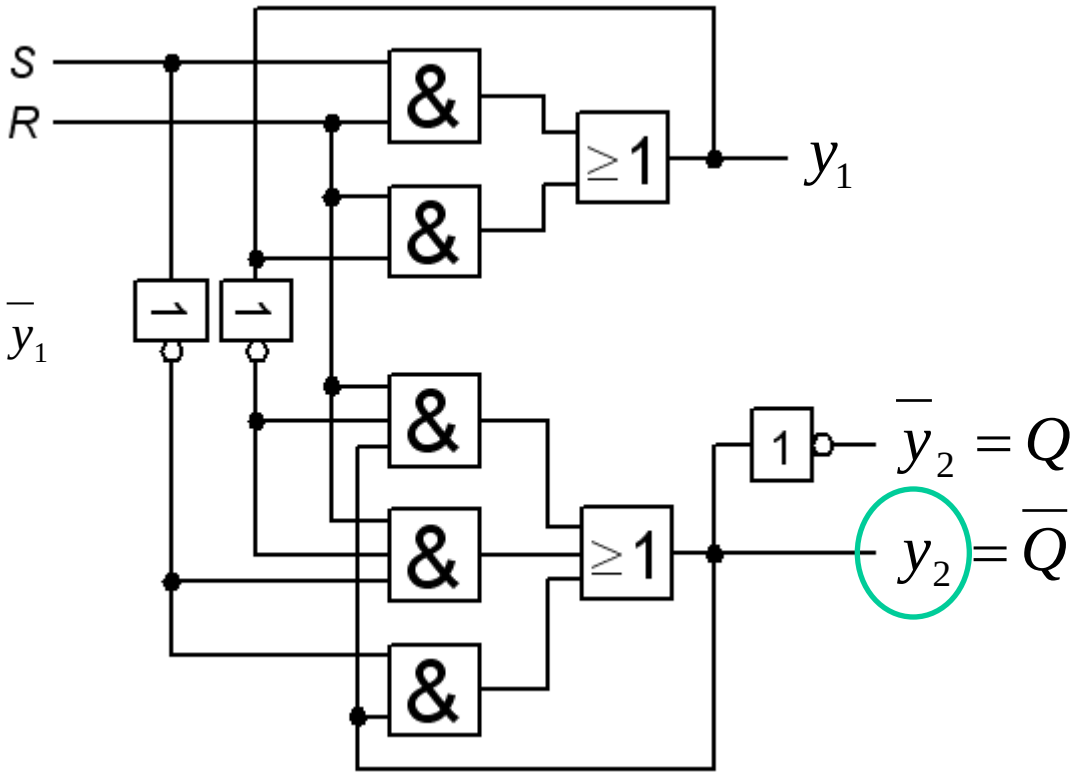
William Sandqvist william@kth.se

Krets-schema



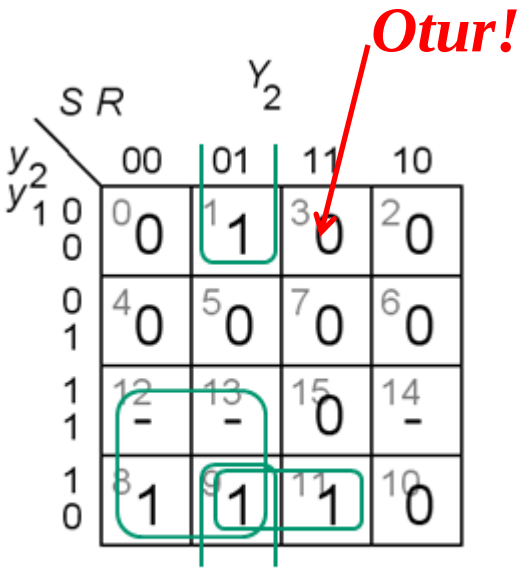
$$Y_2 = \overline{S}y_2 + \overline{S}R\overline{y}_1 + Ry_2\overline{y}_1$$

$$Y_1 = Ry_1 + SR$$



Här har vi vår ”idiotsäkra” SR-låskrets!

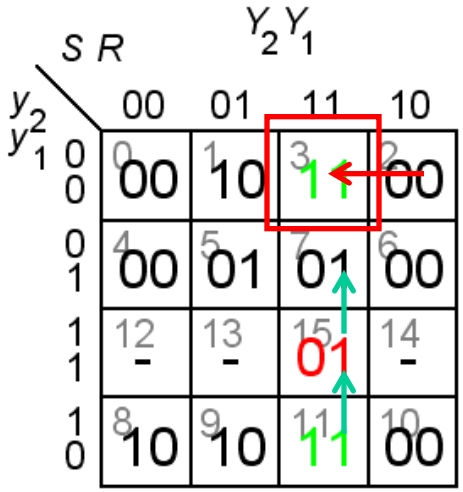
William Sandqvist william@kth.se



$$Y_2 = \overline{S}y_2 + \overline{S}R\overline{y}_1 + Ry_2\overline{y}_1$$

En annan lösning?

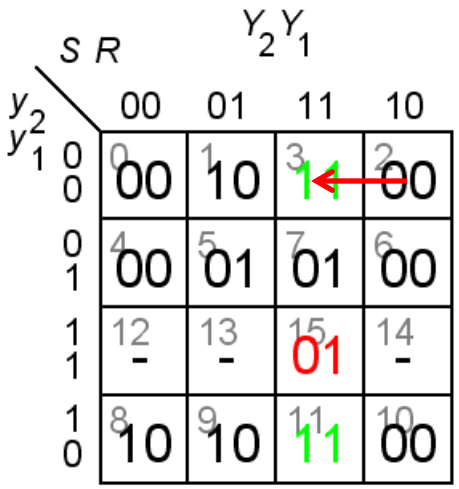
Förutom att lösa problemet med dubbeländringen, som vi redan löst, vill vi få så enkla nät som möjligt!

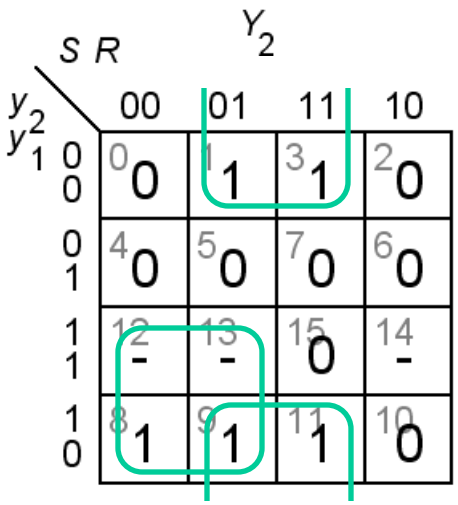


- Vad händer om vi skriver 11 (i stället för 01) som instabilt tillstånd i ruta 3, på ren spekulation att detta kommer att ge oss ett enklare nät?

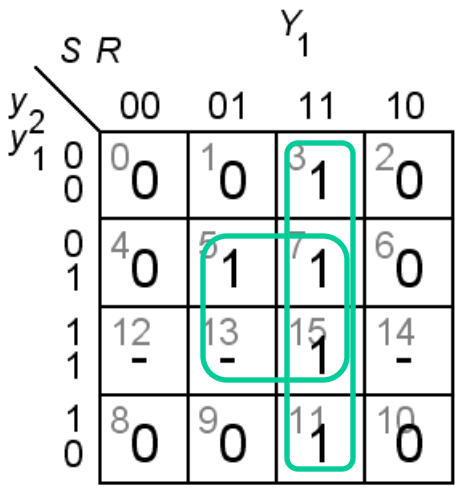
Från 00 i ruta 2 till 11 i ruta 3 är en *ofarlig* dubbeländring. Blir det 01 hamnar man stabilt i **01**, blir det 10 går man till 11 och till sist stabilt till **01**.

Nya Karnaughdiagram





$$Y_2 = \overline{S}y_2 + R\overline{y}_1$$



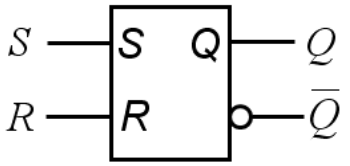
$$Y_1 = Ry_1 + SR$$

• Från 00 till 11 innebär en *ofarlig* dubbeländring av tillståndsvariablerna som till sist alltid leder till stabilt 01.

Enklare nät! Vi introducerade en icke-kritisk Hasard och det gav oss större hoptagningar och ett enklare nät!

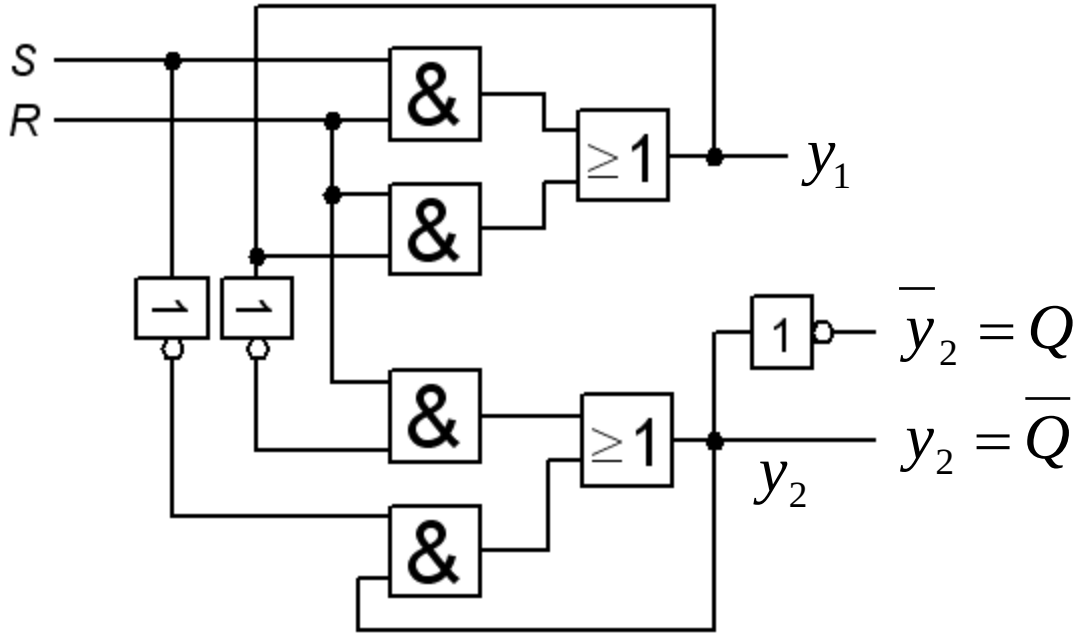
William Sandqvist william@kth.se

Idiotsäker *och* kompakt



$$Y_2 = \overline{S} y_2 + R \overline{y_1}$$

$$Y_1 = R y_1 + SR$$



William Sandqvist william@kth.se

Asynkrona nät är byggstenar

Eftersom de asynkrona sekvensnäten används som byggstenar vid all annan Digital Design är det vanligt att stor möda har lagts på att göra dom så optimala som möjligt.

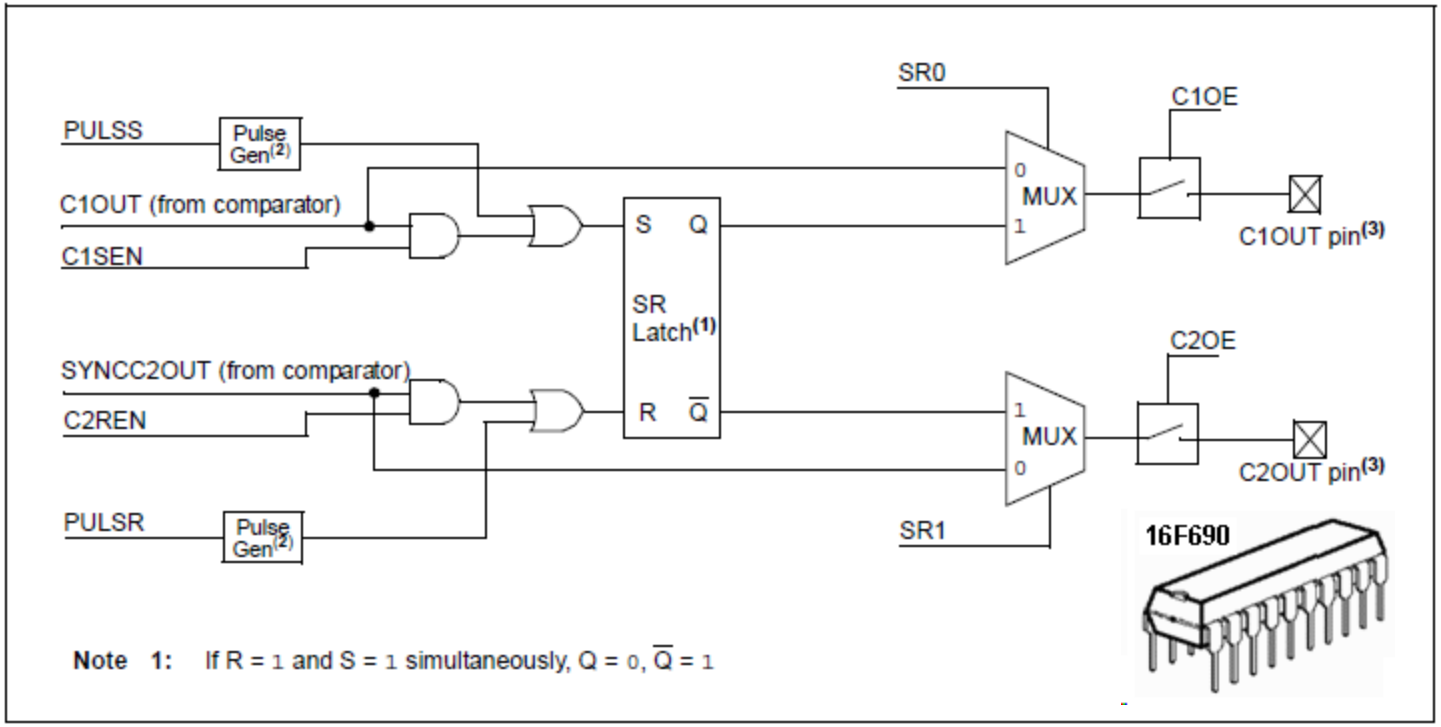
De används oftare i tusental i en konstruktion än styckvis. Varje ingående grind ”kostar” och räknas!

Används dominanta SR-latchar?

William Sandqvist william@kth.se

PIC16F690 IO-enhet, **SR-latch**

FIGURE 8-7: SR LATCH SIMPLIFIED BLOCK DIAGRAM

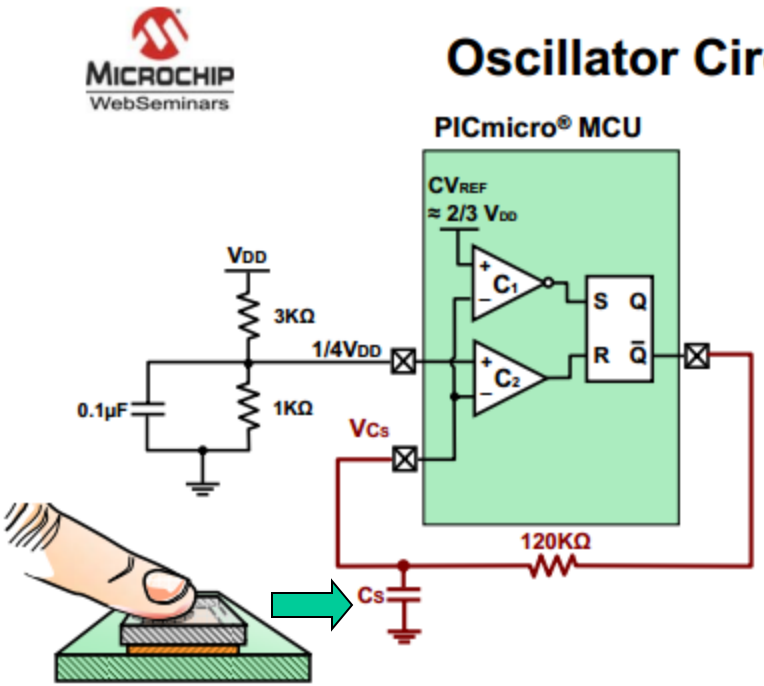


*Låskretsen är **RESET-dominant** SR 11* $\rightarrow Q = 0 \quad \overline{Q} = 1$

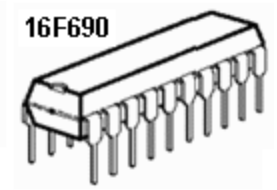
William Sandqvist william@kth.se

PIC16F690 IO-enhet, SR-latch

Touch control. *Användningsområdet för SR-latchen är en kapacitivt styrd oscillator. Den ändrar frekvens vid en ”touch” med fingret.*



$$f = f_0 - \Delta f$$



William Sandqvist william@kth.se

William Sandqvist william@kth.se