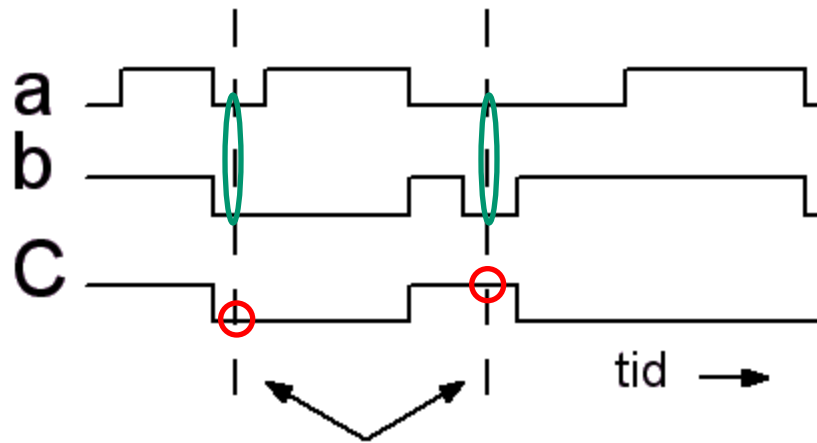
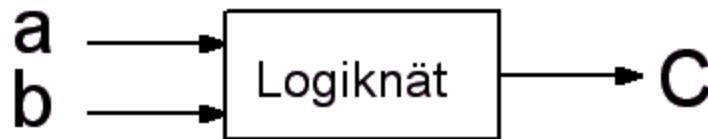


Sekvensnät



Samma insignal
kan ge olika utsignal

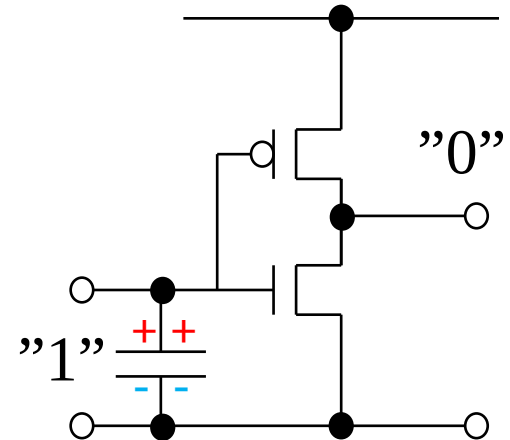


Om en och samma insignal kan ge upphov till olika utsignal, är logiknätet ett sekvensnät.

Det måste då ha ett ***inre minne*** som gör att utsignalen påverkas av både nuvarande och föregående insignaler!

Hur minns hårdvara?

- För att minnas någonting, så måste vi på något sätt hålla kvar informationen.
- Ett sätt är att lagra information i form av en laddning på en kapacitans (DRAM).

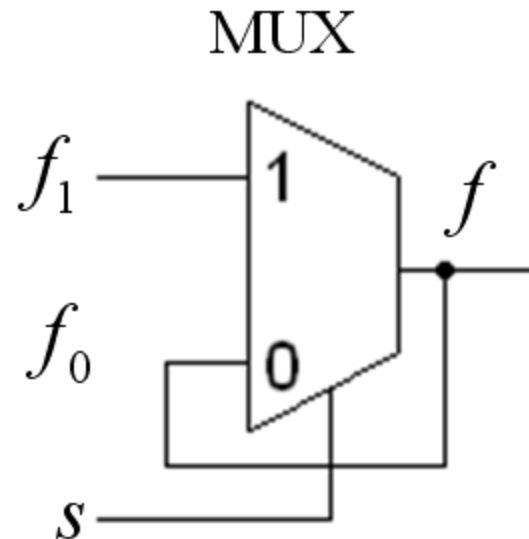


Det finns andra möjligheter ...

”Självhållning”

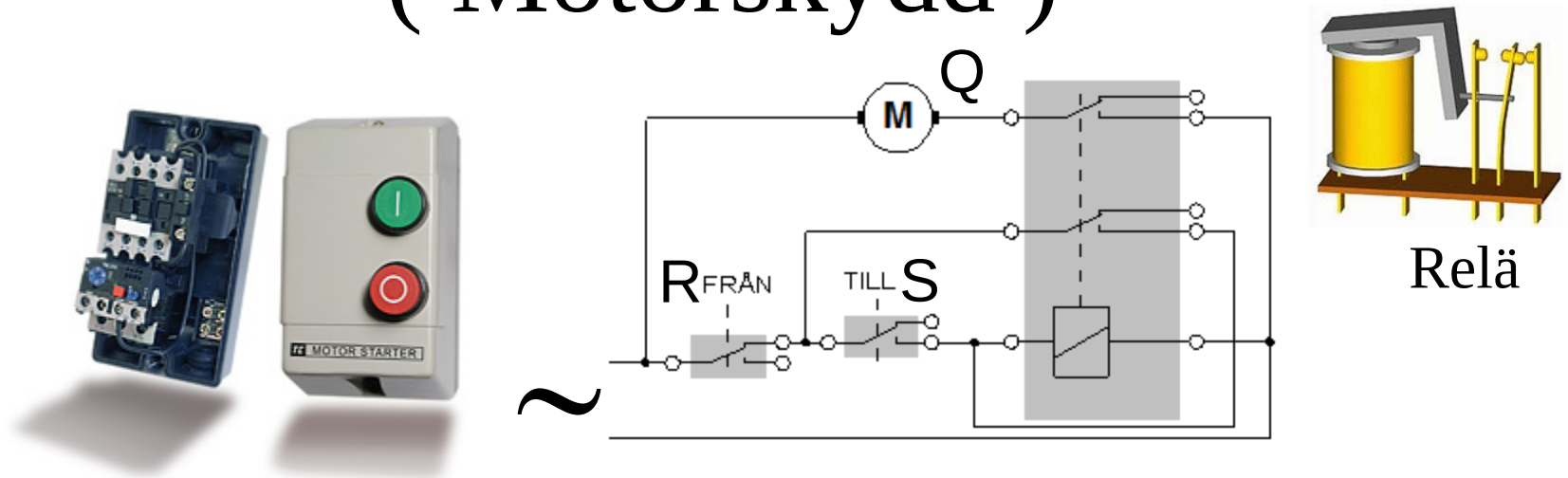
s	f_1	f_0	f
0	—	f_0	f_0
1	f_1	—	f_1

Om $s = 1$ följer utgången f ingången f_1 . När s blir $s = 0$ så ”låser sig” kretsen till det värde f hade i ögonblicket **före** övergången till $s = 0$.



$$s = \textit{follow} / \overline{\textit{latch}}$$

(Motorskydd)



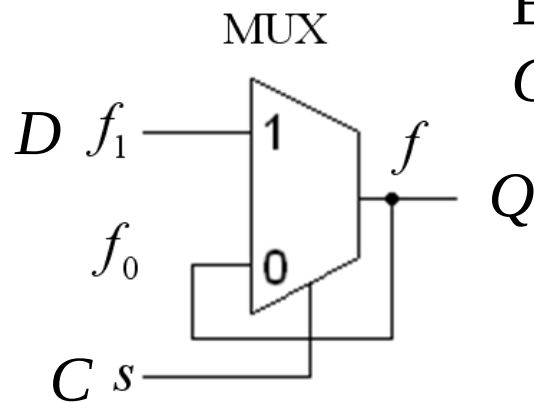
Ett **motorskydd** är ett relä med en **självhållnings**kontakt.

- Man behöver bara trycka kort för att motorn ska starta.
- Blir det strömavbrott så startar *inte* motorn plötsligt av sig själv när strömmen kommer tillbaka – en bra säkerhetsdetalj.
- Lamporna däremot tänds direkt – också det bra.

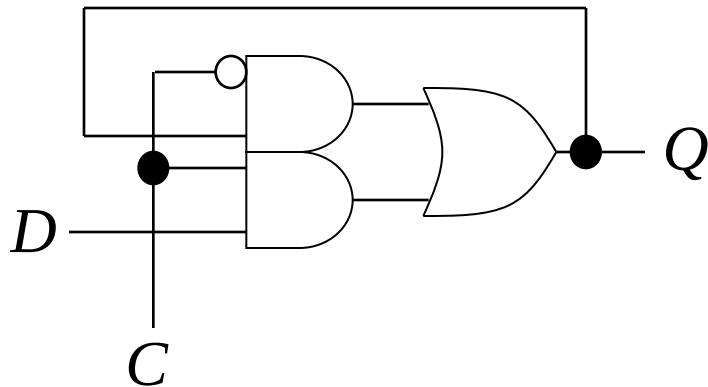
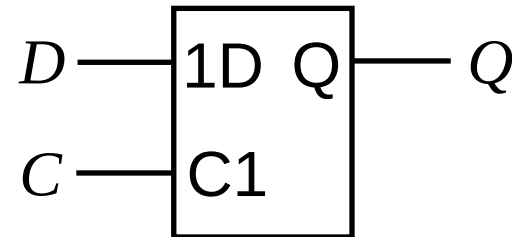
D-Latch



En D-latch är en "återkopplad" MUX. När $C = 0$ låses värdet. (Latch = låsklyka).

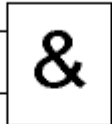
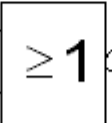


D-Latch



C follow / $\overline{\text{latch}}$	D	Q
0	–	M latch
1	D	D follow

NOR och NAND ”låsande insignal”

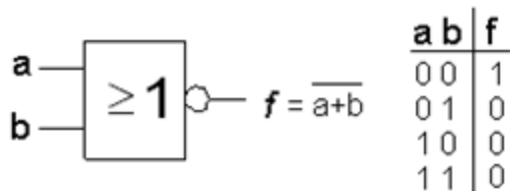
Benämning	Grindfunktion															
NAND	 $f = \overline{a \cdot b}$ <table><tr><th>a</th><th>b</th><th>f</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	a	b	f	0	0	1	0	1	1	1	0	1	1	1	0
a	b	f														
0	0	1														
0	1	1														
1	0	1														
1	1	0														
NOR	 $f = \overline{a + b}$ <table><tr><th>a</th><th>b</th><th>f</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	a	b	f	0	0	1	0	1	0	1	0	0	1	1	0
a	b	f														
0	0	1														
0	1	0														
1	0	0														
1	1	0														

Regel ...

NAND. Om någon ingång är ”0”, så är utgången ”1” oavsett värdet på den andra ingången!

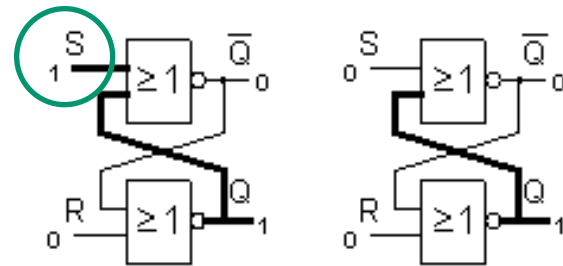
NOR. Om någon ingång är ”1”, så är utgången ”0” oavsett värdet på den andra ingången!

SR-latchen med NOR-grindar

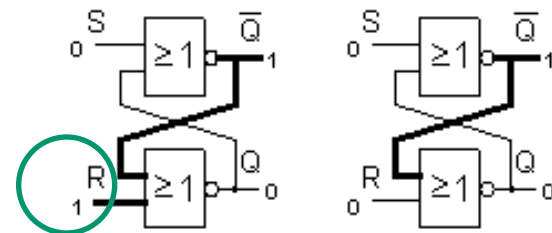


För en NOR-grind är "1" en "låsande" insignal – om någon ingång är "1" har det ingen betydelse vad någon annan ingång har för värde – utgången blir då alltid "0".

$Q=1$

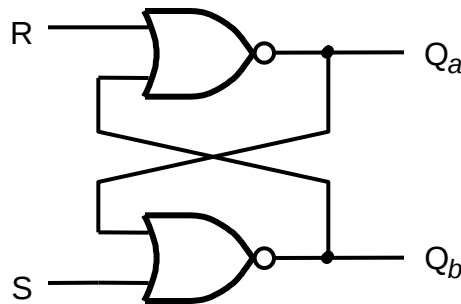


$Q=0$



Det räcker därför med en **kort puls "1"** på S för att kretsen ska hålla $Q = 1$. En kort puls "1" på R ger $Q = 0$.

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

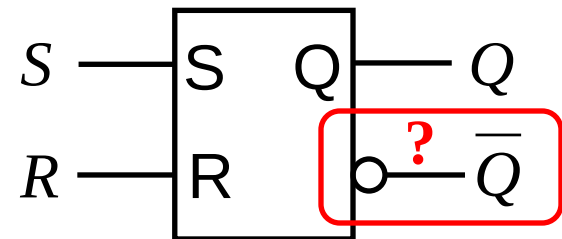
(b) Truth table

Förbjuden
insignal $S=R=1$
 $Q_a \neq \bar{Q}_b$

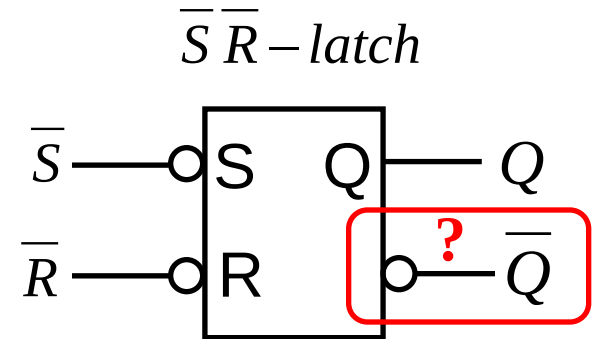
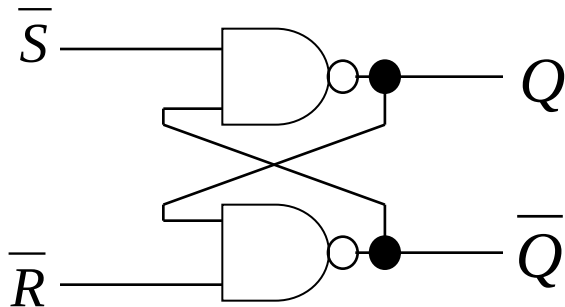
Så länge man **undviker** insignalen $S = R = 1$ (= förbjudet tillstånd) kommer utgångarna Q_a och Q_b att vara varandras inverser. Man kan då använda symbolen till höger.

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

SR-Latch



$\bar{S}\bar{R}$ -latch med NAND-grindar

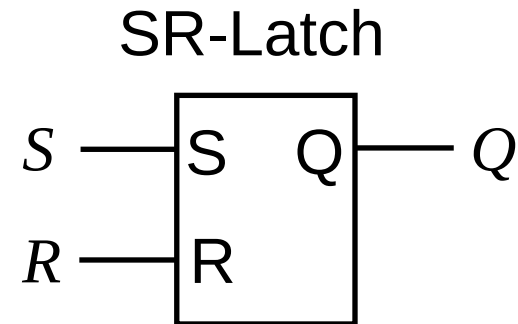
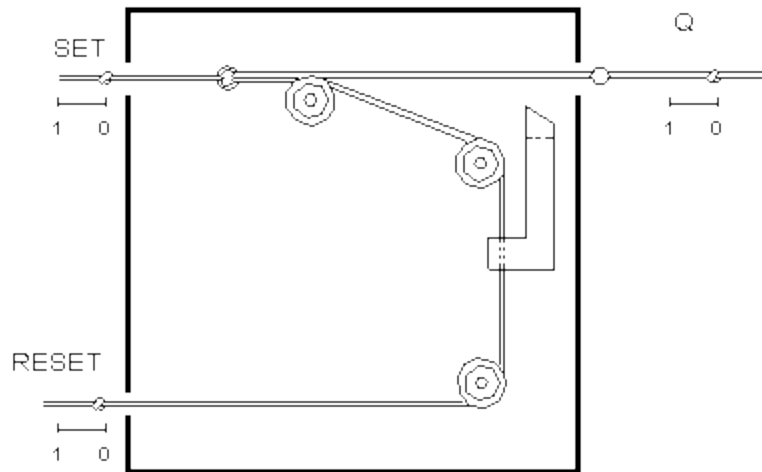


För NAND-grindar är "0" en låsande insignal som tvingar grindens utgång till "1".

Latch med NAND-grindar har **aktiv låga** SET och RESET ingångar. De får *inte* vara "0" samtidigt.

$\bar{S}$	$\bar{R}$	Q	$\bar{Q}$
0	0	1	1
0	1	1	0
1	0	0	1
1	1	M	M

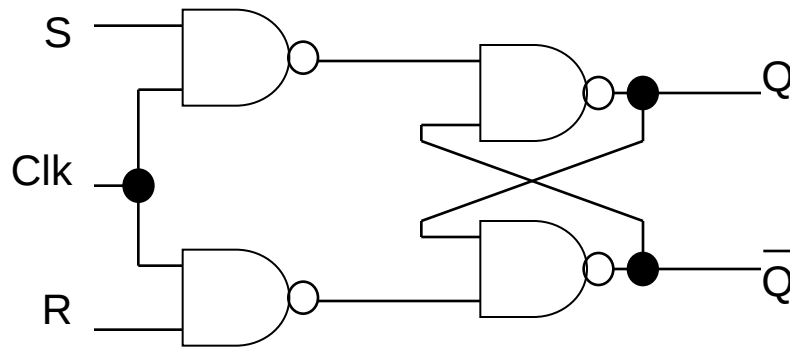
SR-Latch



Till vänster har vi en SR-latch i reptechnik – ett April-skämt från Scientific American! Inte heller här får man dra i SET och RESET repen samtidigt.

(Gated SR-Latch)

Med två extra grindar och en klocksignal Clk kan man styra **när** låskretsen ska få påverkas av insignalerna S och R . När $Clk = 0$ finns **ingen påverkan**, då kan ju till och med $S = R = 1$ tillåtas.



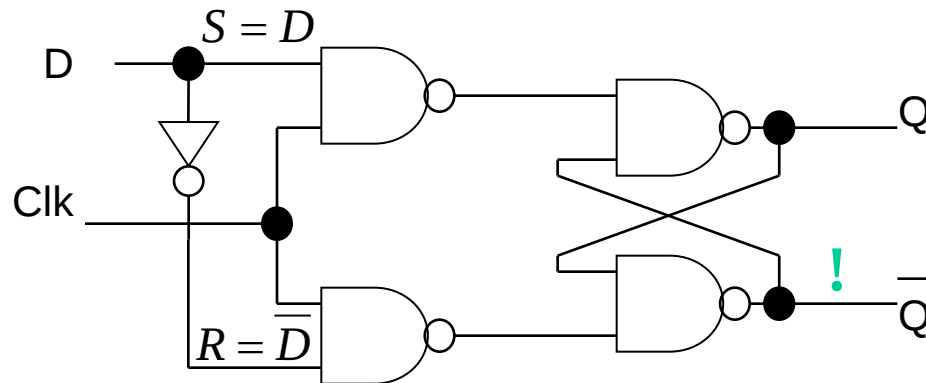
Förbjudet tillstånd

Clk	S	R	Q	$\bar{Q}$
1	0	0	M	M
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1
0	-	-	M	M

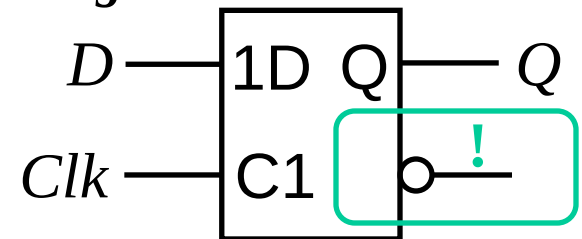
D-latch

En ändå bättre lösning på problemet med det ”förbjudna tillståndet” är D-latchen. Med en **inverterare** säkerställer man att S och R helt enkelt ***alltid har olika värden!***

Låskretsens utgång följer D-ingången när $Clk = 1$ för att låsa värdet när $Clk = 0$. Denna låskrets har samma funktion som den återkopplade MUX-kretsen. Skillnaden ligger i att denna krets har ***snabbare*** återkoppling. Dessutom får vi också tillgång till en ***inverterad utsignal***.

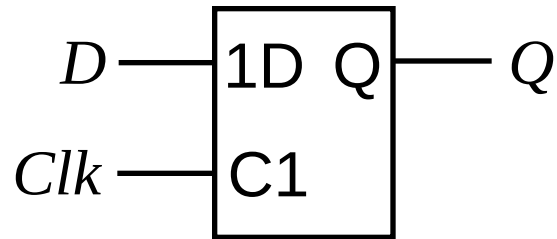


$Clk = \text{follow} / \text{latch}$

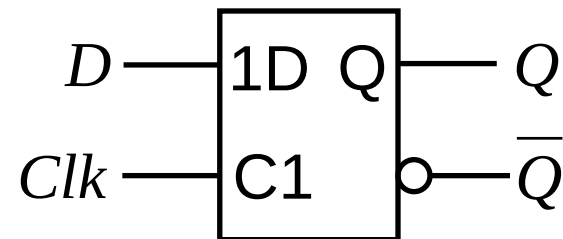
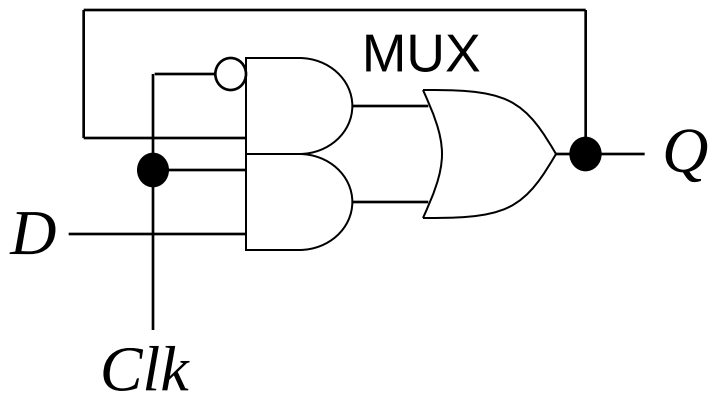


Clk	D	Q	$\overline{Q}$
1	0	0	1
1	1	1	0
0	-	M	M

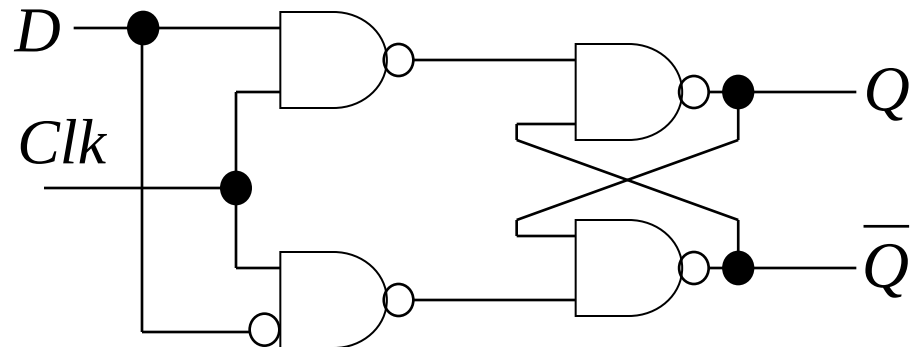
Två olika D-latchar



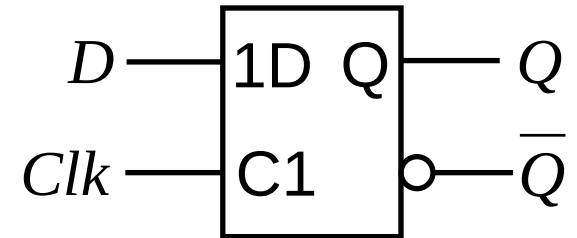
Lång återkoppling ($\sim 4T$)



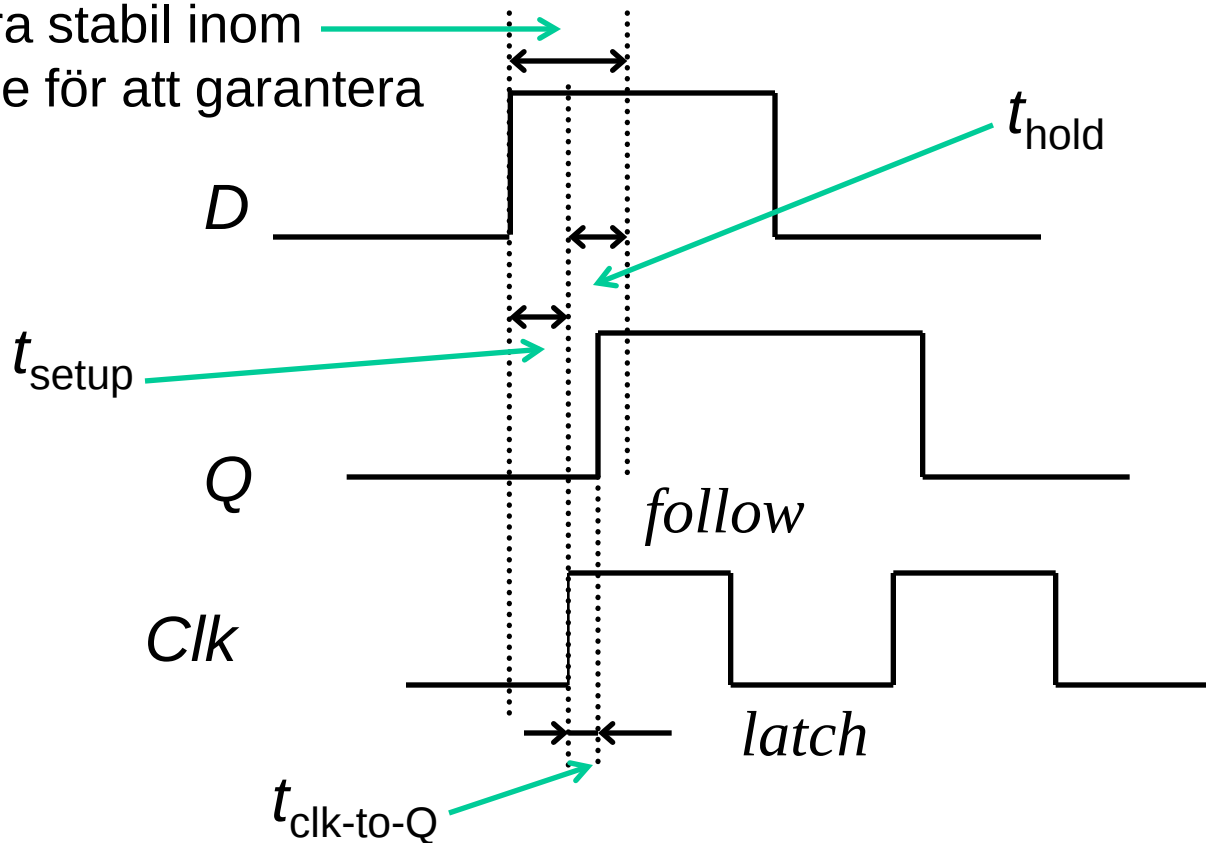
Kort återkoppling ($\sim 1T$)



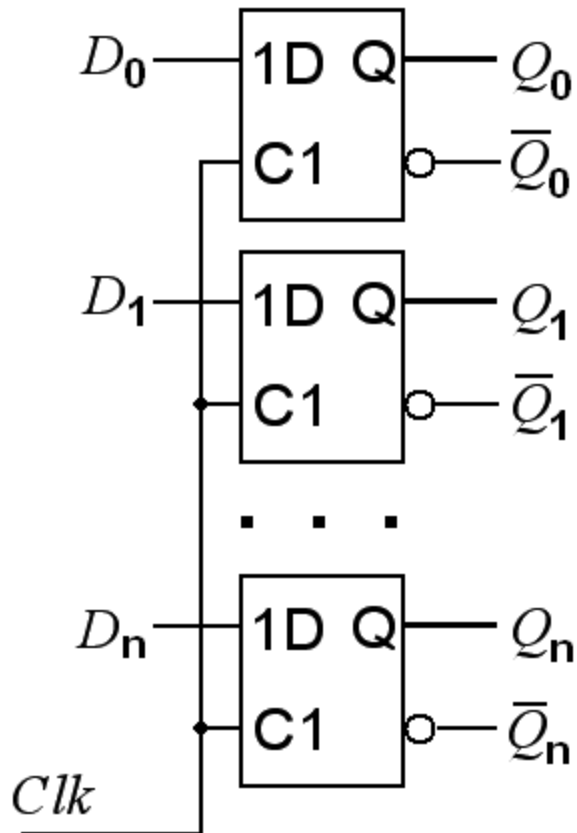
Setup- & Hold-time



D måste vara stabil inom detta område för att garantera funktionen



Register – inverterade utsignaler

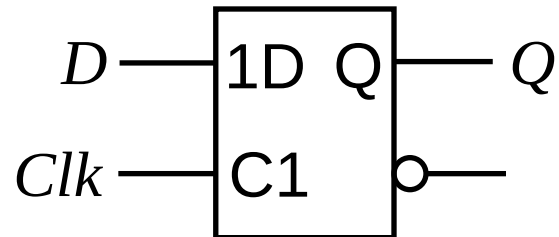


Ett vanligt sätt att konstruera digitala kretsar är att signaler tas via register (= en samling låskretsar eller vippor) till de kombinatoriska nätens ingångar.

D-låskretsar har ”automatiskt” en inversutgång.

Det är därför vi i räkneexemplen oftast utgått ifrån att inverterade signaler finns att tillgå.

Varannan gång?



Hur gör man en **sekvenskrets** som ”byter” värde 1/0 efter varje klockpuls, *Clk* ?

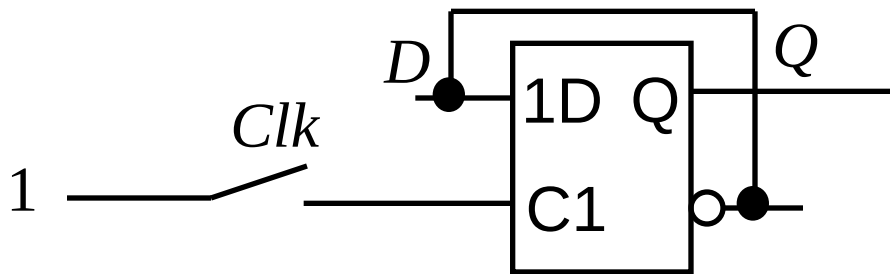
- Kretsen behöver ”minnas” sitt förra värde Q
- och ändra detta till $Q = D = \overline{Q}$.

Låskretsen har ju både ”minne” och en inversutgång – skulle inte den kunna användas?

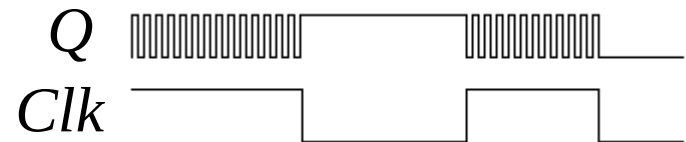
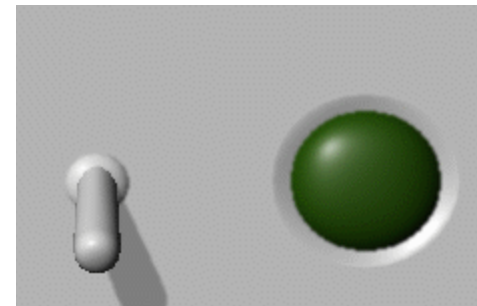
Går ej med enkel låskrets ...

$Clk = follow / \overline{latch}$

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

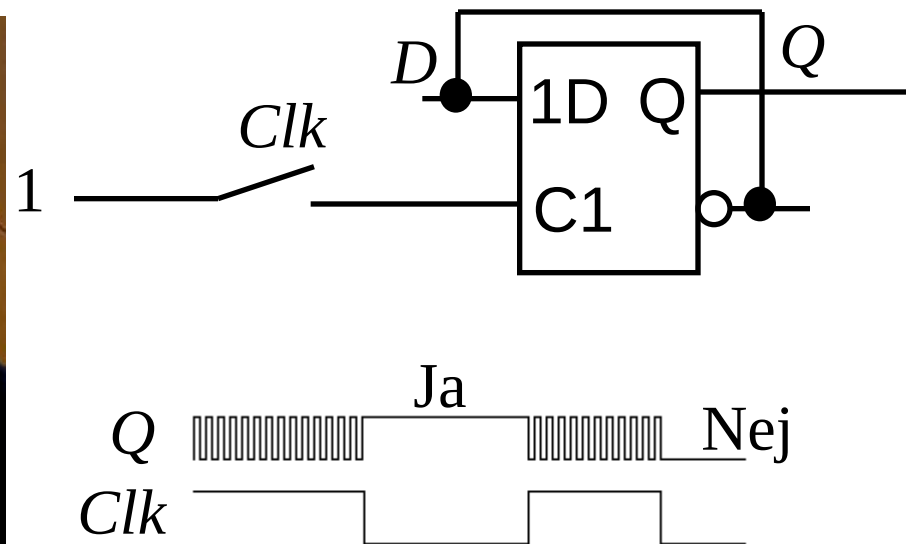


- När $Clk = 1$ följer utsignalen insignalen – därför byter utgången 1/0 så fort som möjligt!
Kretsen blir en oscillator!



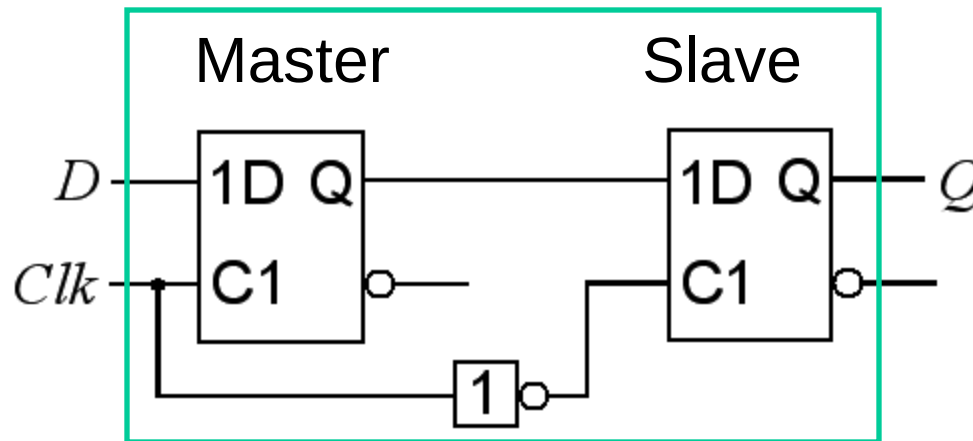
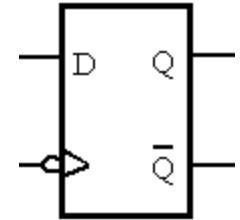
- När sedan $Clk = 0$ behåller utsignalen sitt värde 1/0 allt efter vad den senast stod på. (= Slumpgenerator?)

Voteringshjälp i riksdagen?



Klockade vippor

Master-Slave vippa



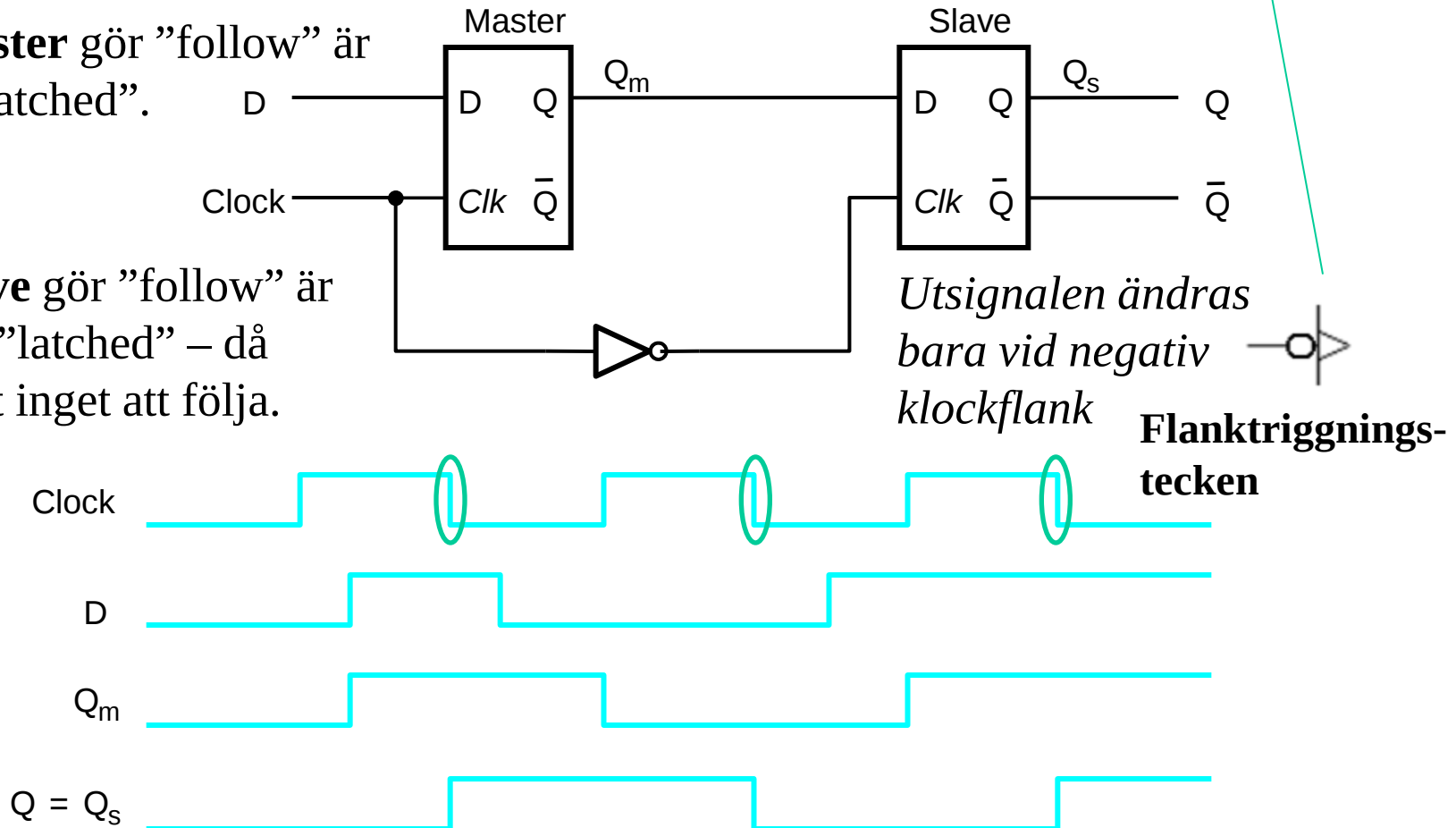
Problemet är att den **enkla låskretsen** är öppen för förändring ända fram tills den ska låsa sitt värde.

Lösningen är den **klockade vippa** som består av flera låskretsar. En låskrets tar emot nya data (Master) medan en annan har kvar det gamla datat (Slave).

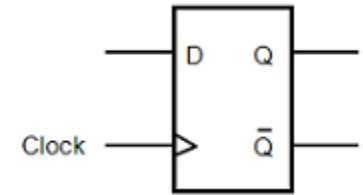
Tidsdiagram Master-Slave

När **Master** gör "follow" är **Slave** "latched".

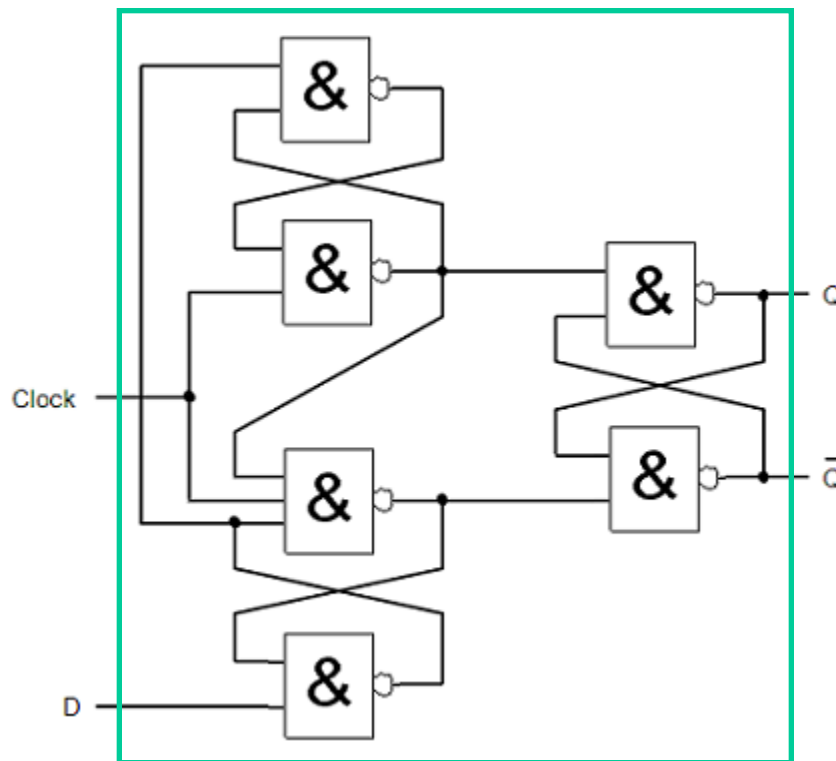
När **Slave** gör "follow" är **Master** "latched" – då finns det inget att följa.



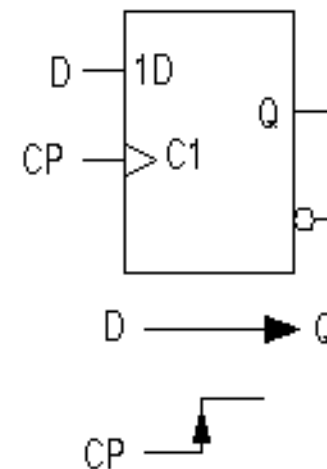
Flanktriggad D-vippa



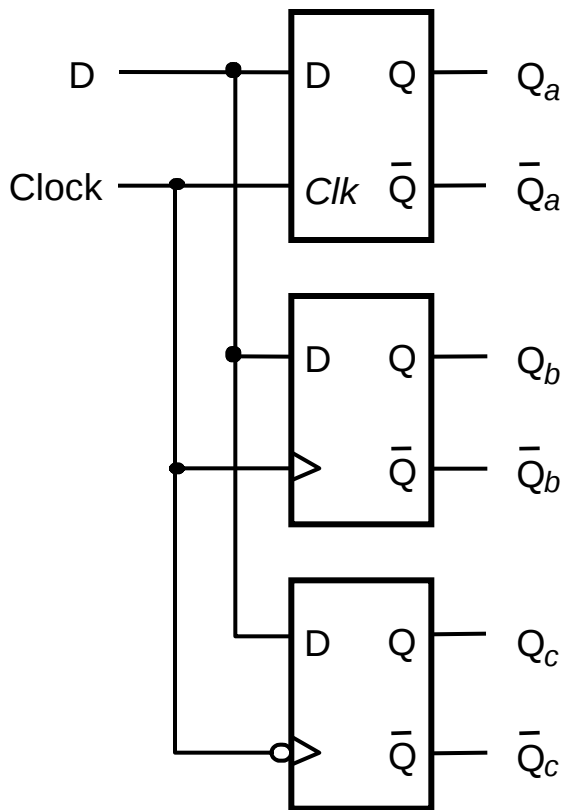
En annan flanktriggad vippa består av **tre** låskretsar. Datavärdet ”kopieras” till utgången **precis** när klocksignalen går från 0 → 1.



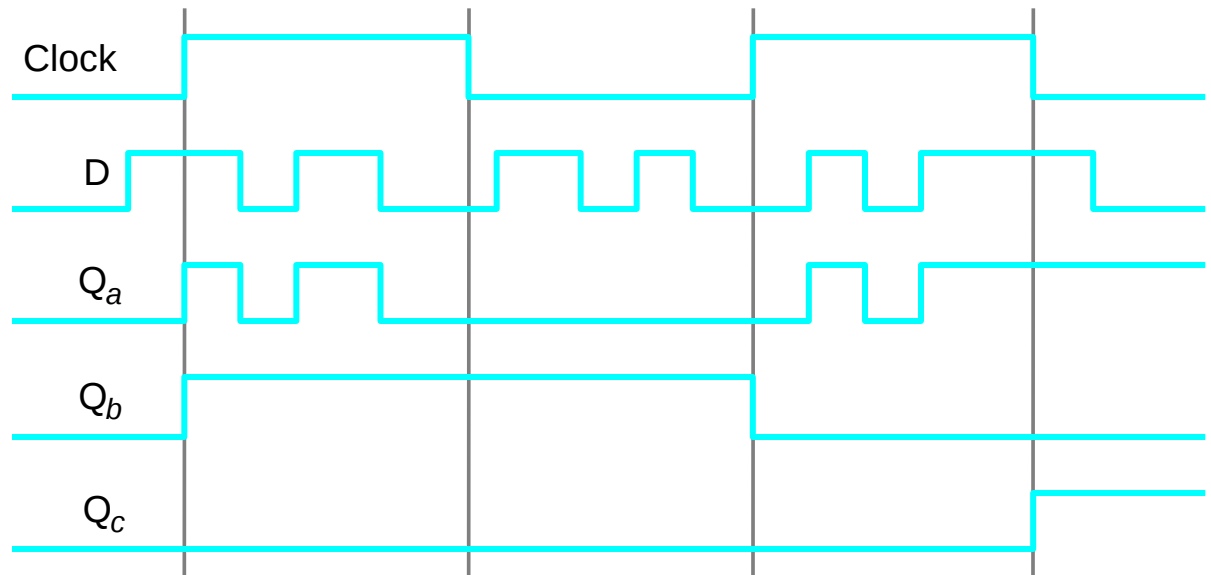
—▶ Positiv flank 0 → 1
—◻▶ Negativ flank 1 → 0



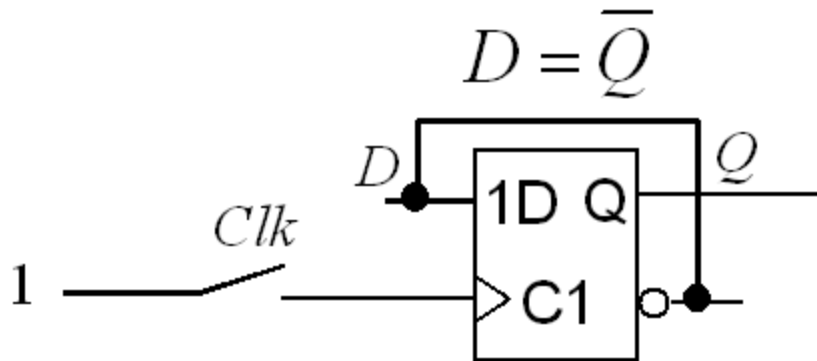
Låskrets eller Vippra?



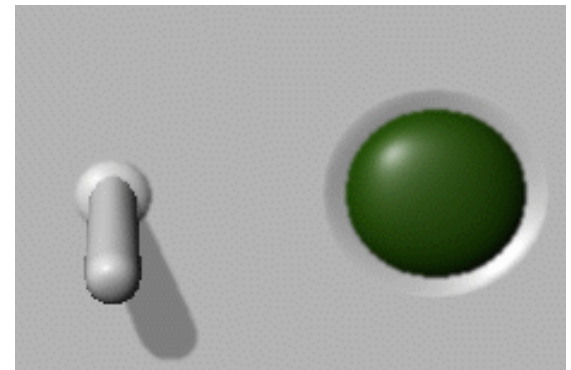
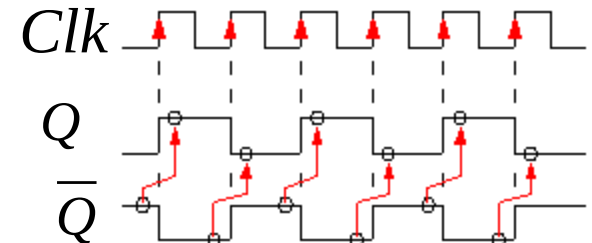
- a) Låskrets – follow/latch
- b) Positivt flanktriggad vippra
- c) Negativt flanktriggad vippra



Varannan gång?



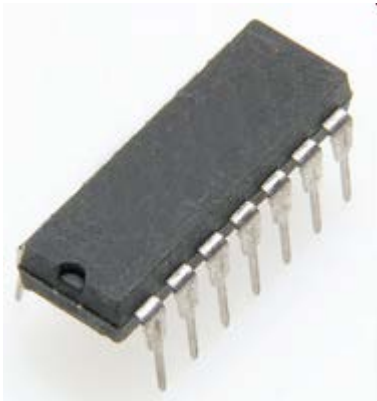
Nu fungerar ”varannan gång”
precis som tänkt!



Till sekvensnät använder man i allmänhet flanktriggade
vippor som minneselement!

Varannan gång med Impulsrelä On-Off-On-Off ...

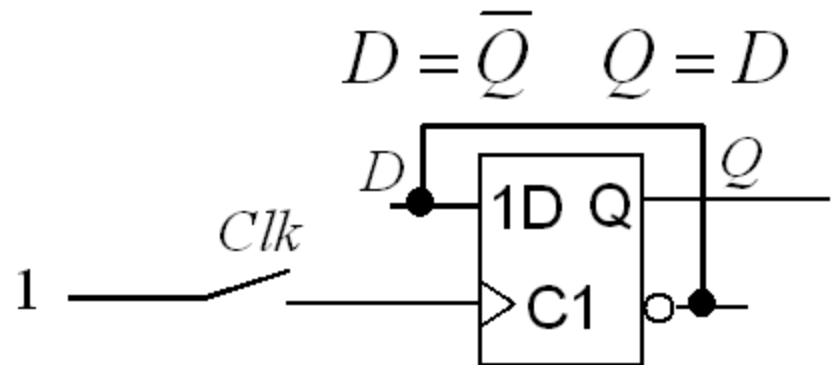
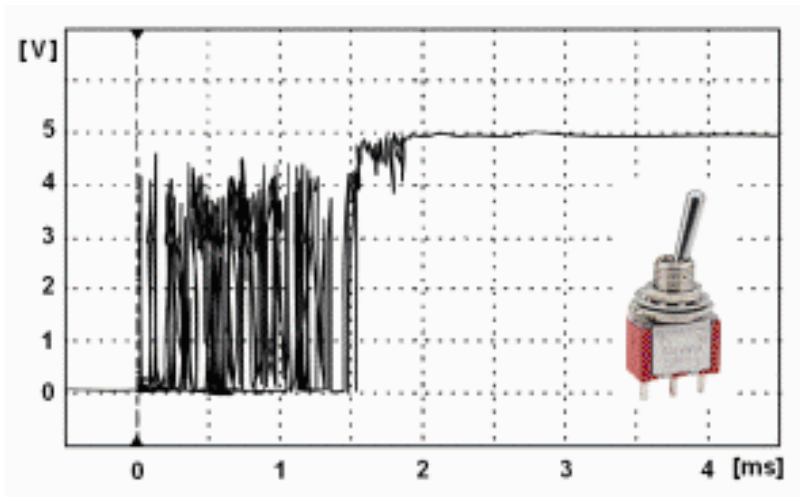
Impulsrelä
Pris: 300:-



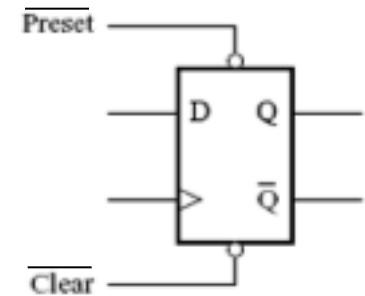
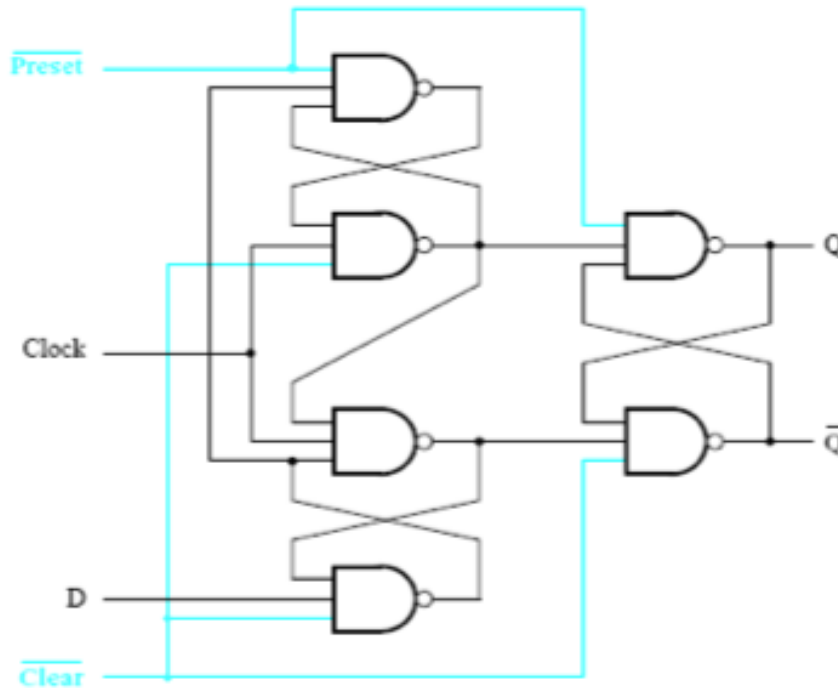
7474 (2st D-vippor)
Pris: 5:- styck

(Kontaktstuds)

Det kan finnas ett annat hot mot ”varannan gång” kretsen, och det är att mekaniska kontakter studsar! Detta får Du pröva på vid laborationen ...



Clear och Preset



D-vippan innehåller tre låskretsar. Signalerna **Preset** och **Clear** går direkt till låskretsarna och kan "låsa" dessa oberoende av klockpulsen. Preset och Clear är aktivt låga.

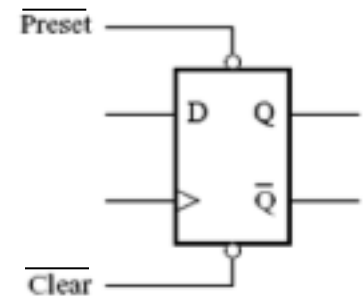
Preset = 0 tvingar $Q = 1$, medan Clear = 0 tvingar $Q = 0$.
Preset = Clear = 1 tillåter vippan att fungera som avsett.

Reset-knappen



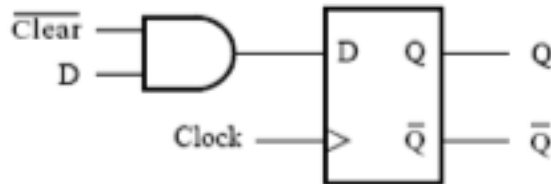
De flesta digitala system behöver kunna startas i ett känt tillstånd. Det kan innebära att en del vippor ska vara "1" medan andra ska vara "0". En resetfunktion kan därför behöva anslutas till antingen **Preset** eller **Clear** på de ingående vipporna.

Preset och **Clear** är asynkrona ingångar – vipporna ändrar sig omedelbart oavsett klockpuls.



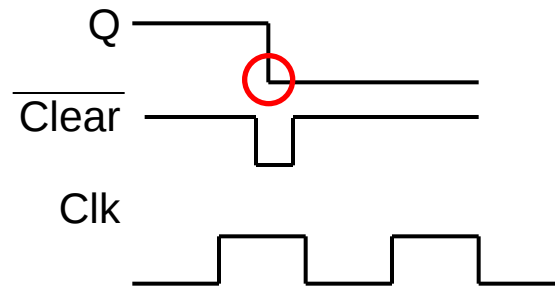
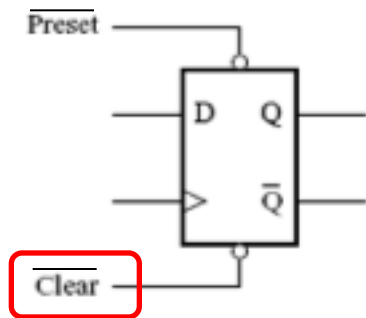
Synkron Reset

Saknar vipporna Preset- och Clear- ingångar, kan reset implementeras med extra logik. Synkron reset orsakar att vippan går till läge 0 vid *nästa* klockflank.

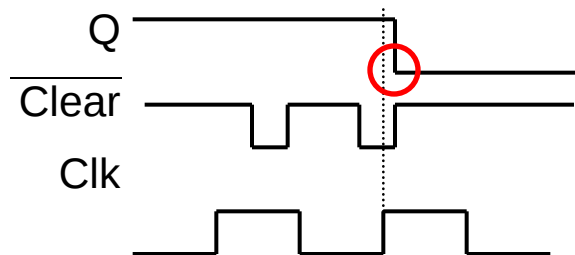
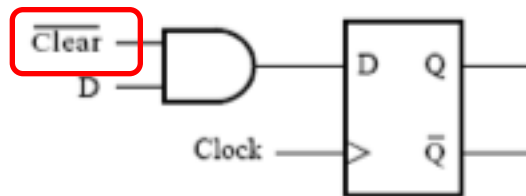


Asynkron/Synkron Reset

Asynkron reset



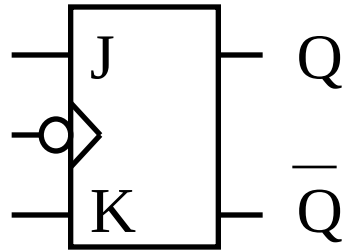
Synkron reset



Andra vanliga typer av vippor

JK-vippa

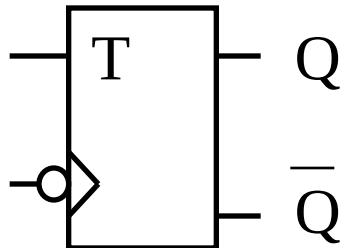
(JK-vippan är en SR-vippa med ”toggle” i stället för förbjudet tillstånd)



Clk	J	K	Q	Q
↓	0	0	M	M
↓	0	1	0	1
↓	1	0	1	0
↓	1	1	Toggle	Toggle

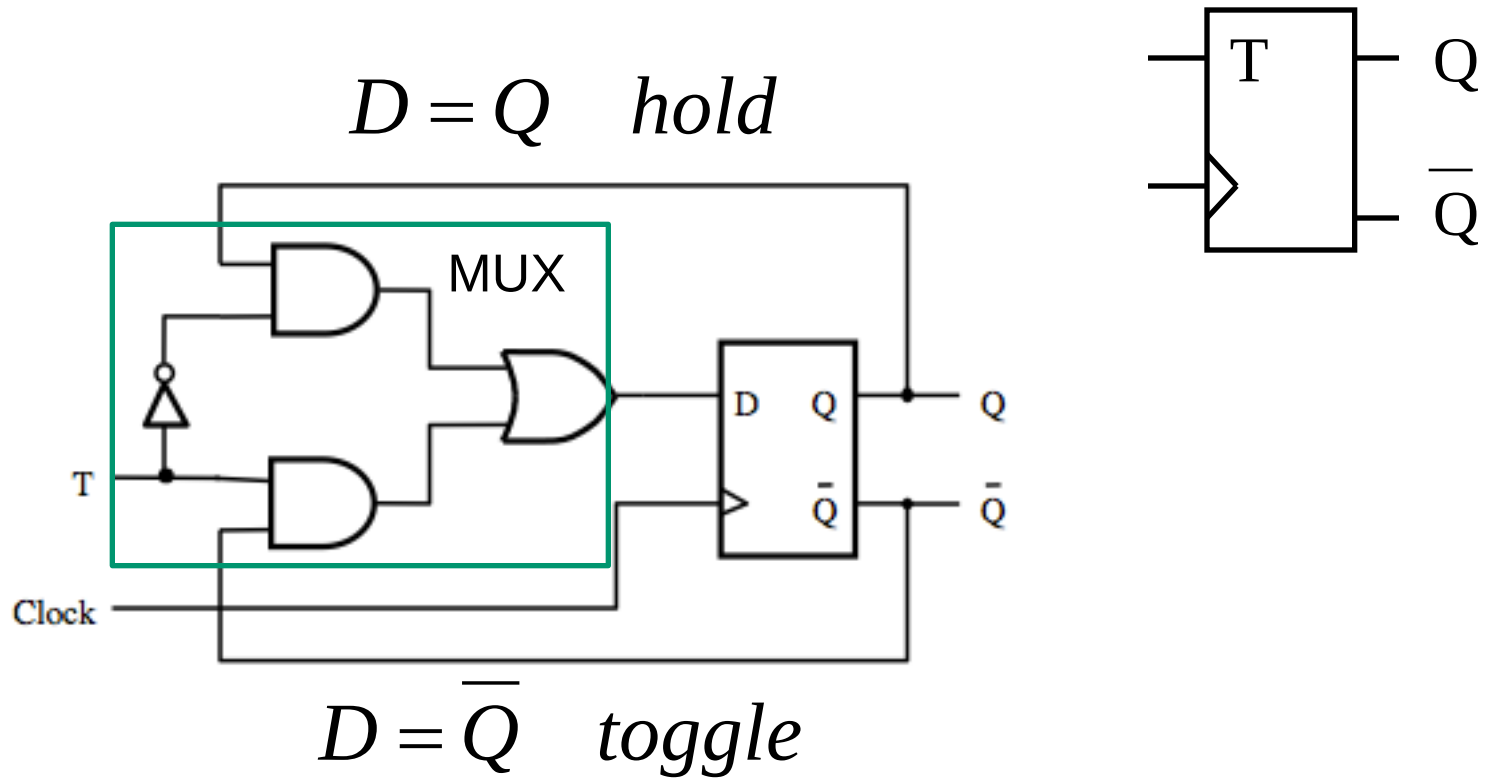
T-vippa (T=Toggle)

(T-vippan är speciellt lämplig för ”räknare”)



Clk	T	Q	Q
↓	0	M	M
↓	1	Toggle	Toggle

Konstruera T-vippa med D-vippa

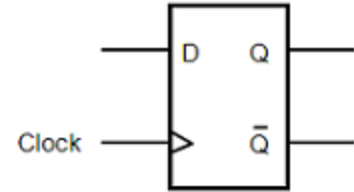


Timing analysis

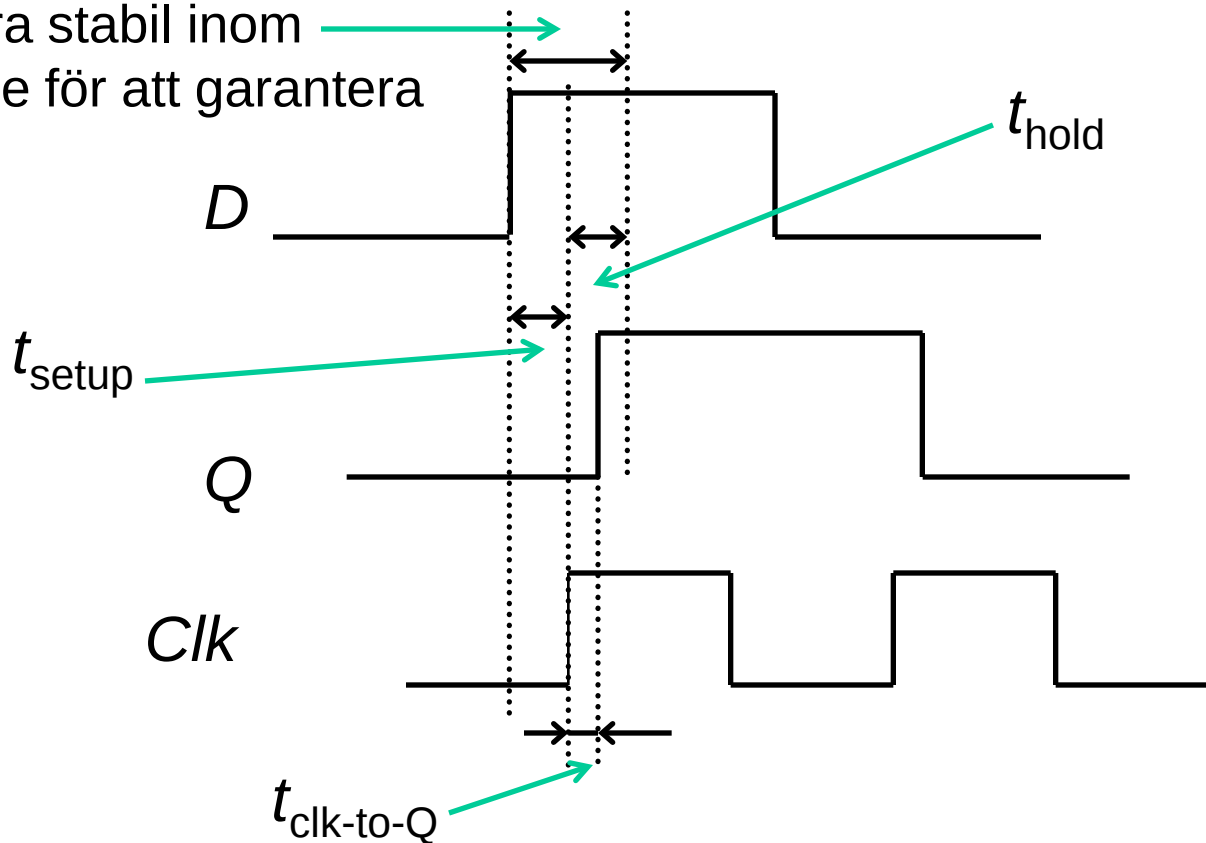
Det är möjligt att kunna bestämma den maximala frekvensen i en sekvensiell krets genom att ha information om

- Grindfördröjningar t_{logic}
- Setup-tid t_{su} för vippan
- Hold-tid t_{h} för vippan
- Clock-to-utgång t_{cQ} tiden

Setup- & Hold-time



D måste vara stabil inom detta område för att garantera funktionen



Vad är den maximala frekvensen?

- Grindfördröjningar

$$t_{\text{logic}} = t_{\text{NOT}} = 1.1 \text{ ns}$$

- Setup-tid

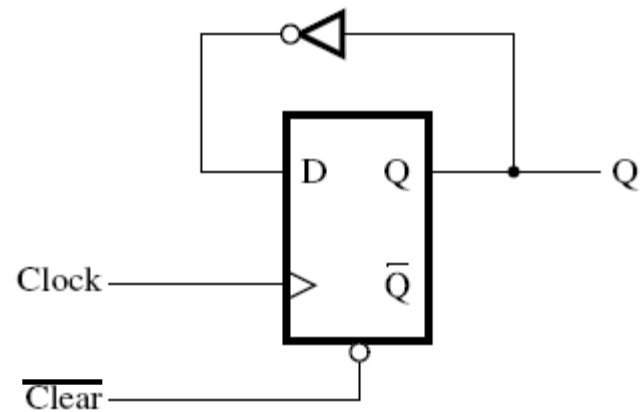
$$t_{\text{su}} = 0.6 \text{ ns}$$

- Hold-tid

$$t_{\text{h}} = 0.4 \text{ ns}$$

- Clock-to-utgång

$$t_{\text{cQ}} = 1.0 \text{ ns}$$

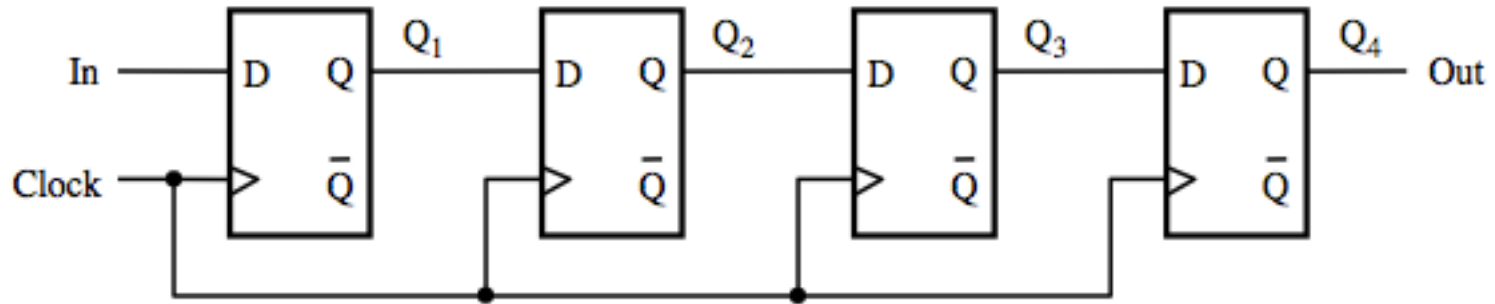


$$T = t_{\text{su}} + \max(t_{\text{h}}, t_{\text{cQ}}) + t_{\text{logic}} = 2.7 \text{ ns}$$
$$f = 1/T = 370 \text{ MHz}$$

0.6 0.4 < 1.0 1.1

↓ ↓ ↓

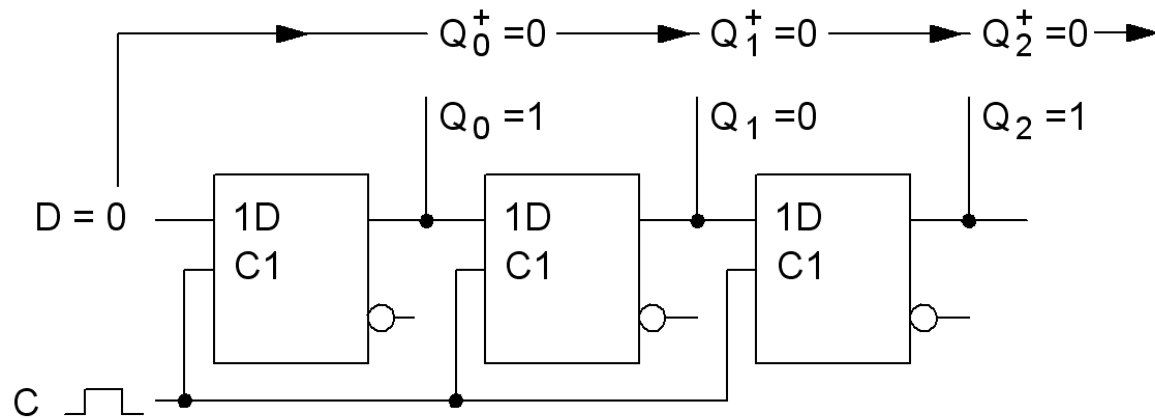
Shiftregister



- En **shiftregister** innehåller flera vippor
För varje klockcykel skiftar man in ett värde från vänster till höger
- Många konstruktioner använder **shiftregister** och värden $Q_4, \dots, Q_1$ som ingångsvärden till andra komponenter

Går inte med låskretsar ...

Det går inte att bygga ett **shiftregister** med låskretsar.



När $C = 1$ *follow* så "rinner" datat igenom alla låskretsarna ...

Vanliga typer av shiftregister

- Parallel-In/Parallel-Out (**PIPO**)
- Parallel-In/Serial-Out (**PISO**)
- Serial-In/Parallel-Out (**SIPO**)
- Serial-In/Serial-Out (**SISO**)
- Användningsområden
 - Köer, tex First-In/First-Out (**FIFO**)
 - Mönsterigenkänning (eng. Pattern recognizers)

Räknare

En räknare är en speciell typ av sekvensnät som registrerar antalet inkommande klockpulser. Registreringen sker efter någon kod, oftast **binärkod**. Efter ett visst antal pulser tar räknarens tillstånd slut och den börjar om från början igen.

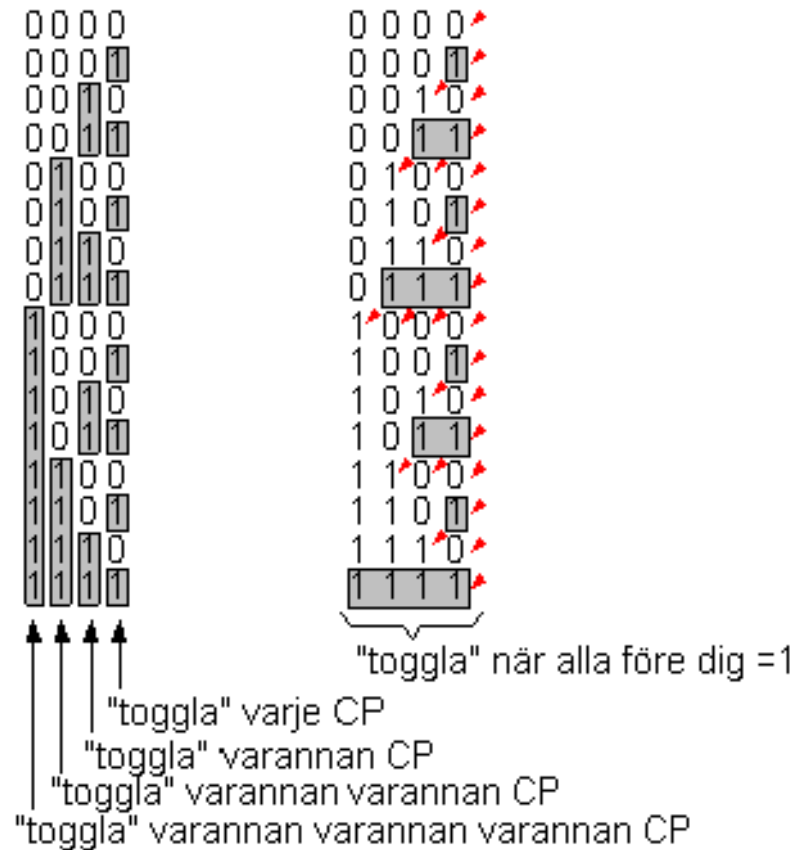
Man talar om räknarens **modul** (dvs. hur många tillstånd räknecykeln innehåller).

Räknaren behöver inte ha någon insignal utom klockpulserna (som då kan ses som signalen).

Ett sådant sekvensnät kallas för **autonomt**.

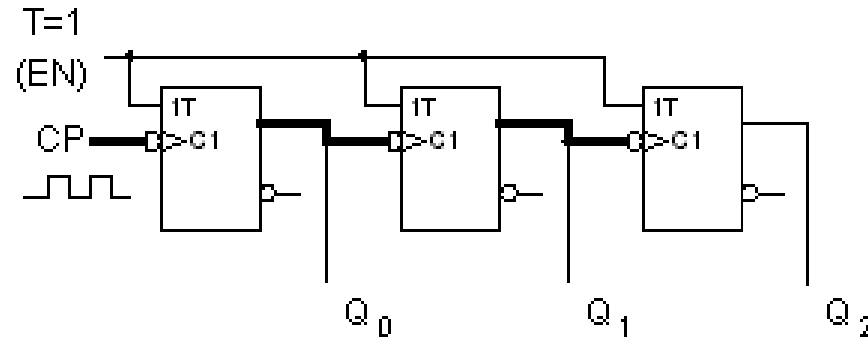
Binärkodens räkne-egenskaper

*Det finns två olika "regler"
för att konstruera binärkoden
ur mindre signifikanta bitar.
Ex. med binärkoden 0 ... 15.*



Togгла varannan gång ...

MODULO-8 Asynkronräknare



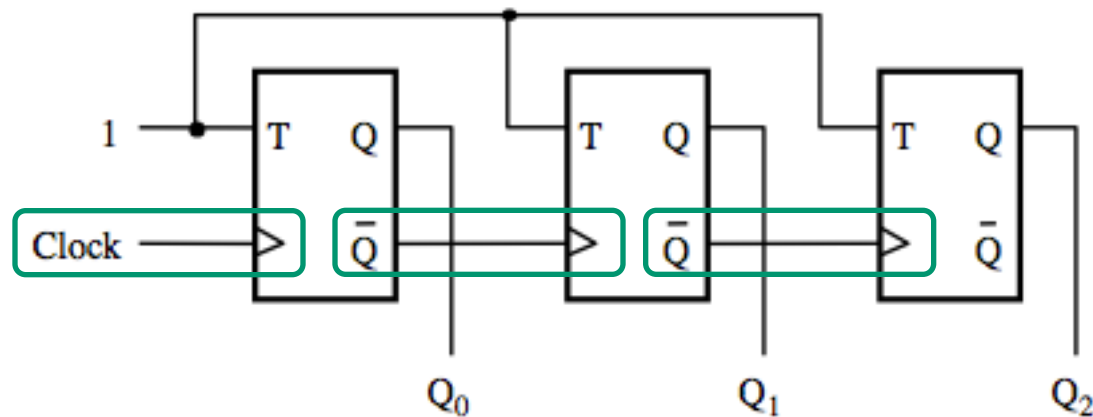
varannan, varannan varannan, varannan varannan varannan ...

Räknaren är uppbyggd av T-vippor, de har alla $T=1$ och "togglar" därför vid varje klockpuls. Den första vippan Q_0 "togglar" för varje klockpuls. Vippan därefter Q_1 klockas av den första vippan. Den kommer därför bara att "togгла" för varannan klockpuls. Den tredje vippan Q_2 kommer "togгла" för varannan varannan klockpuls.

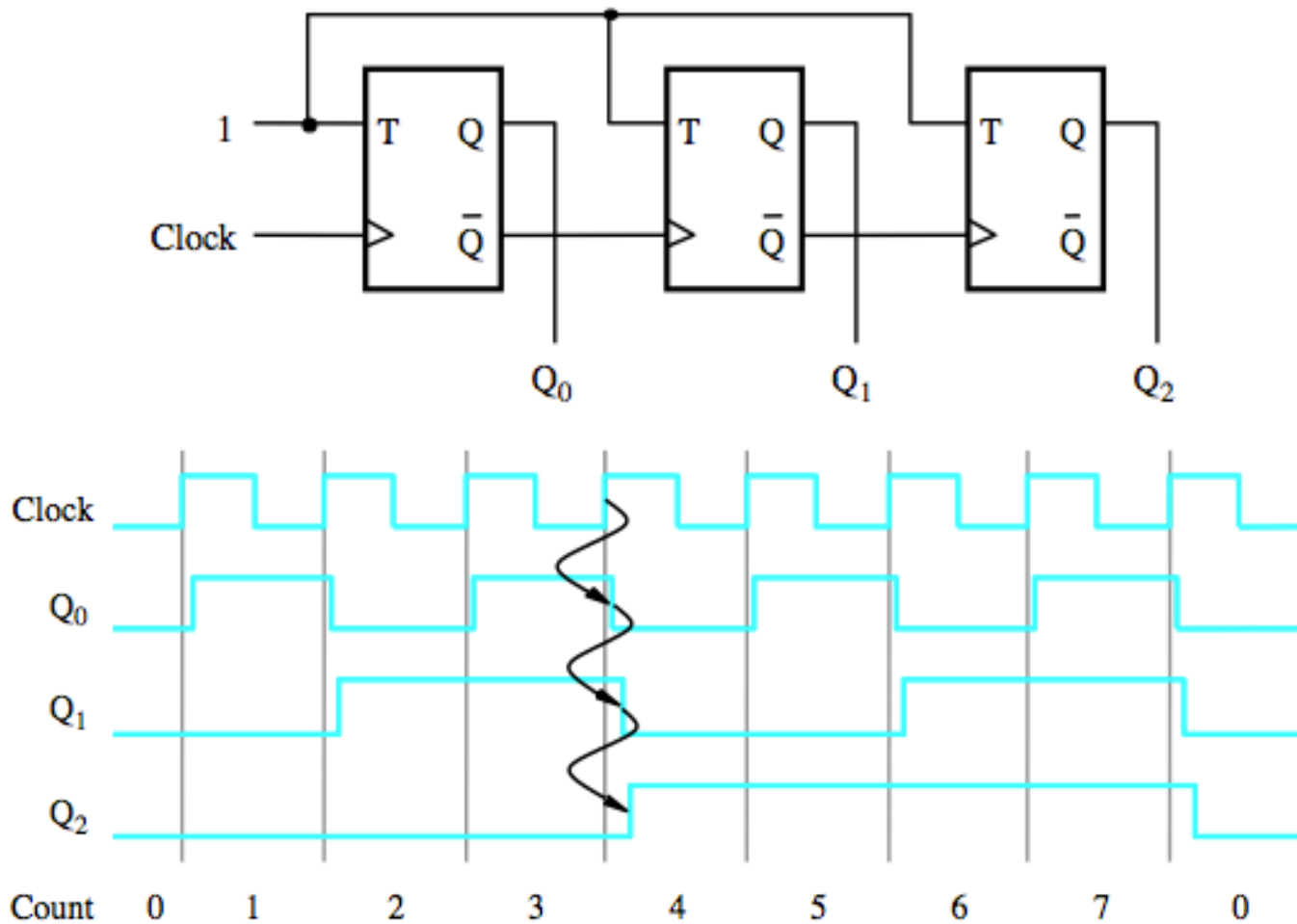
Enligt binärtabellen kommer räknaren därför att räkna i binärkod.

($Q_2Q_1Q_0$: 000 001 010 011 100 101 110 111 000 ...).

Hur räknar den här räknaren?

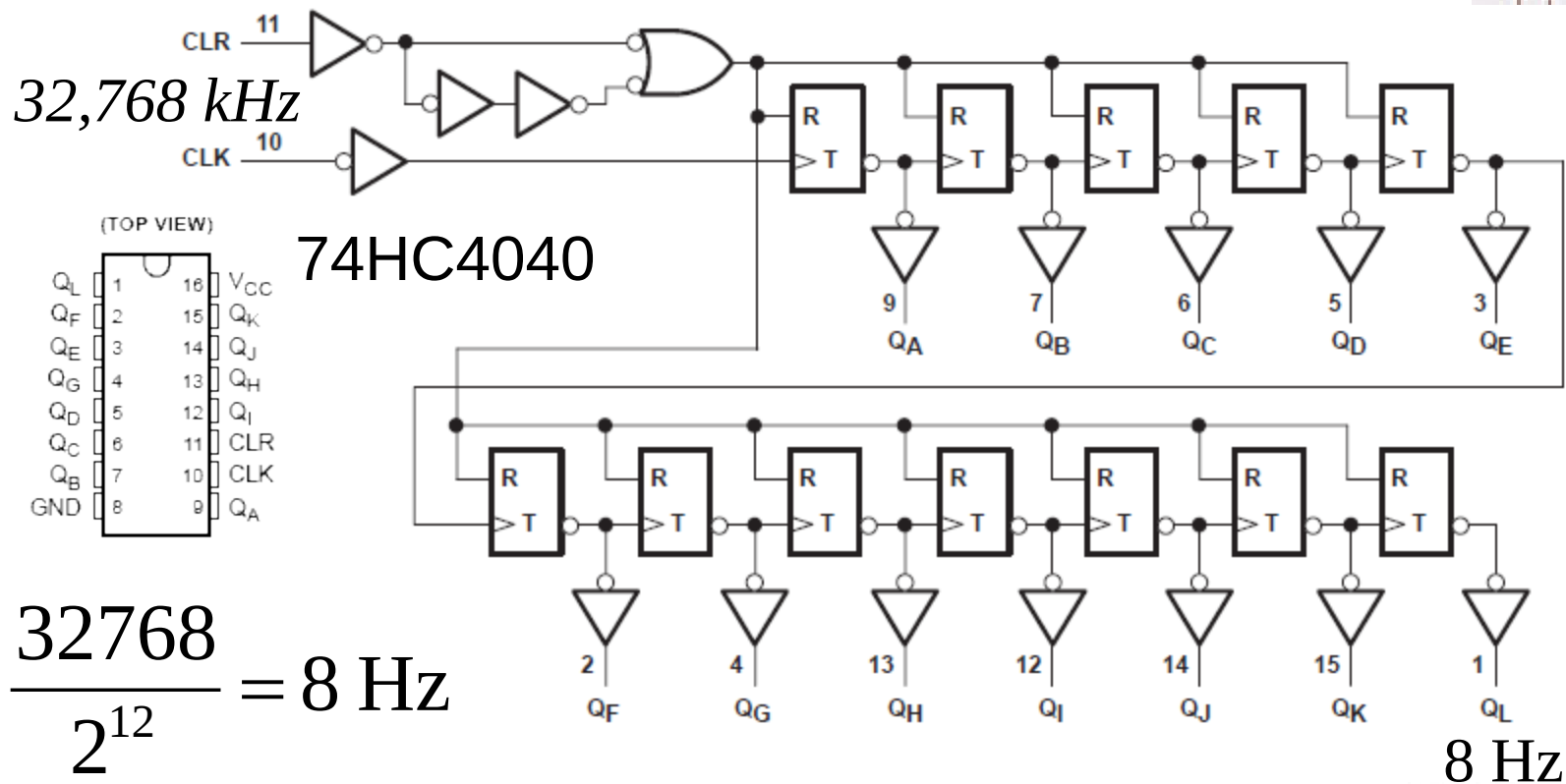


Asynkron räknare



En räknarkrets

32,768 kHz

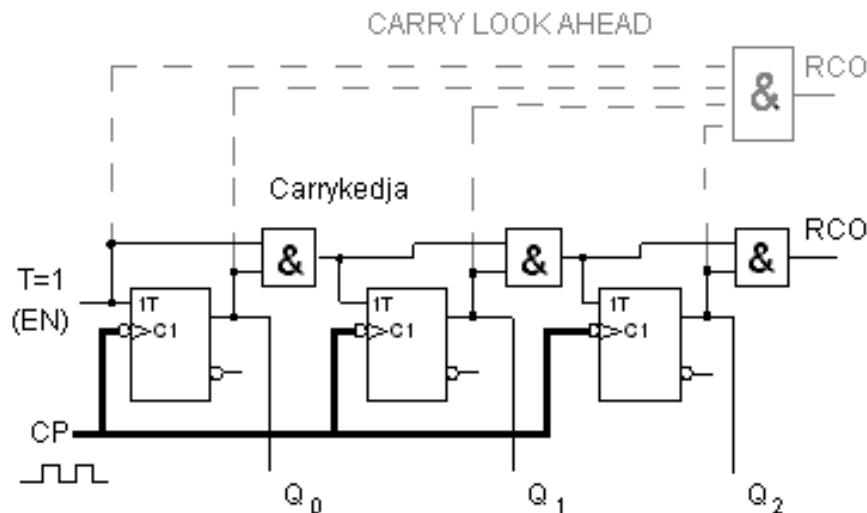


Hur man får 1 sekund får Du räkna ut själv ...



Togglar om alla före dig är 1...

MODULO-8 Synkronräknare



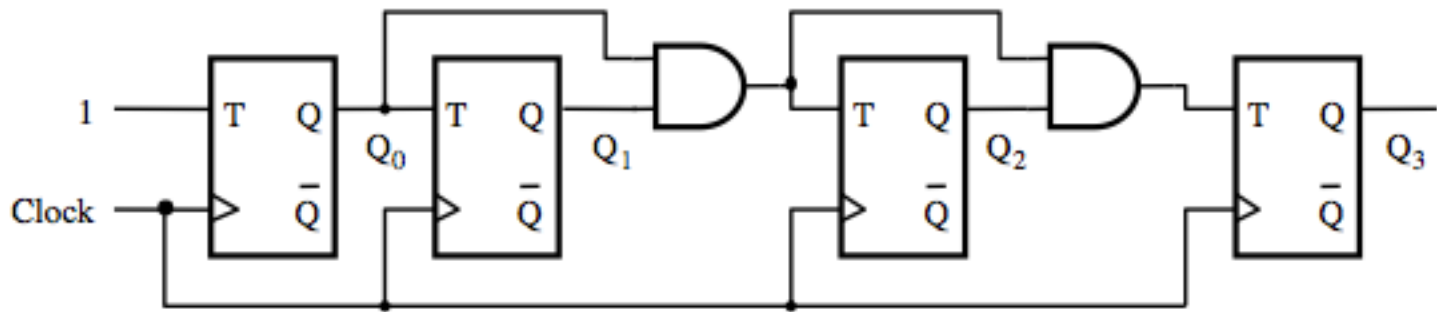
En snabbare räknare kan man få om man tar fram Carry parallellt (jfr. med adderaren).

Vill man utöka räknaren sker det med en vippa och en AND-grind per steg.

Klockpulserna går direkt till alla vippor och därför slår de om samtidigt. Vilka vippor som ska slå om eller ej styrs med T-ingångarna. Den första vippan har T=1 och den slår om för varje klockpuls. En viss vippa ska slå om när alla vippor som är före den står på "1". Det villkoret får man från AND-grindarna i den sk. Carrykedjan och det är dessa som styr T-ingångarna.

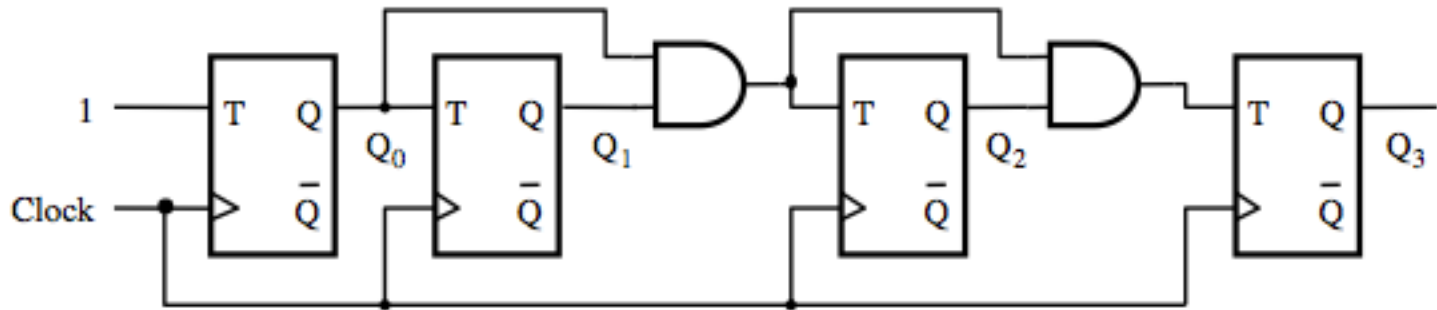
Synkron räknare

I en *synkron* räknare är vippornas klockingångar kopplade till **samma klocksignal**

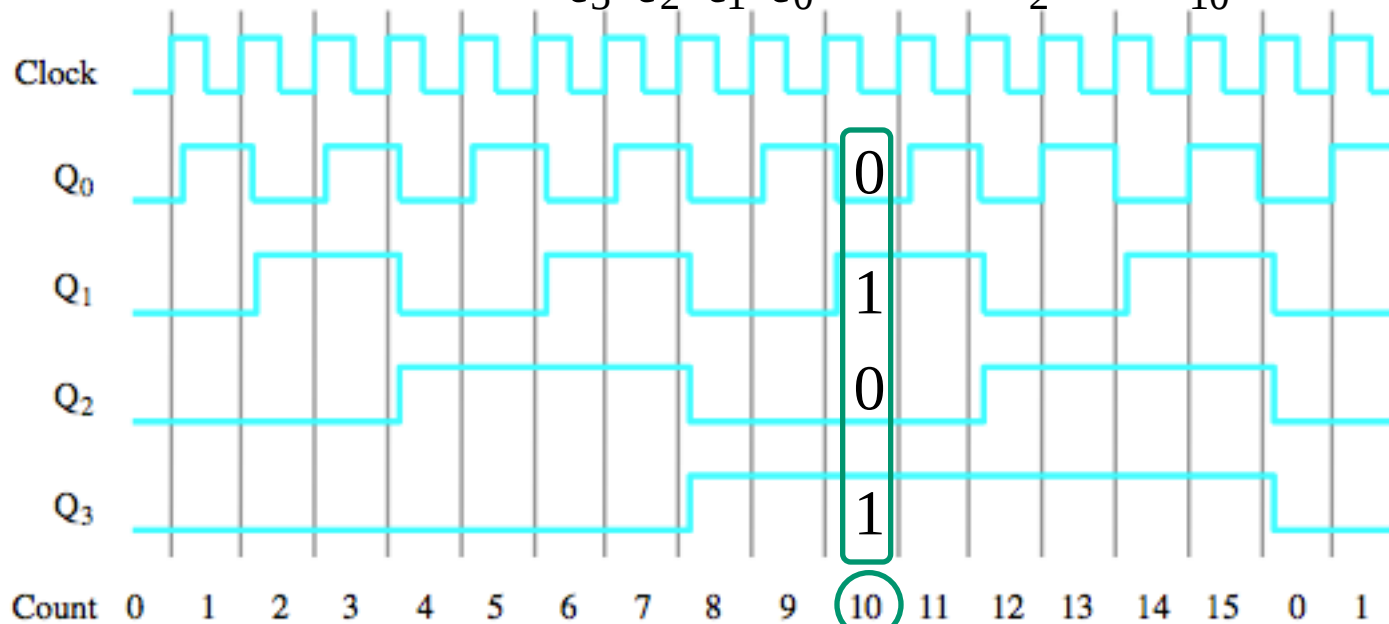


Hur räknar den här räknaren?

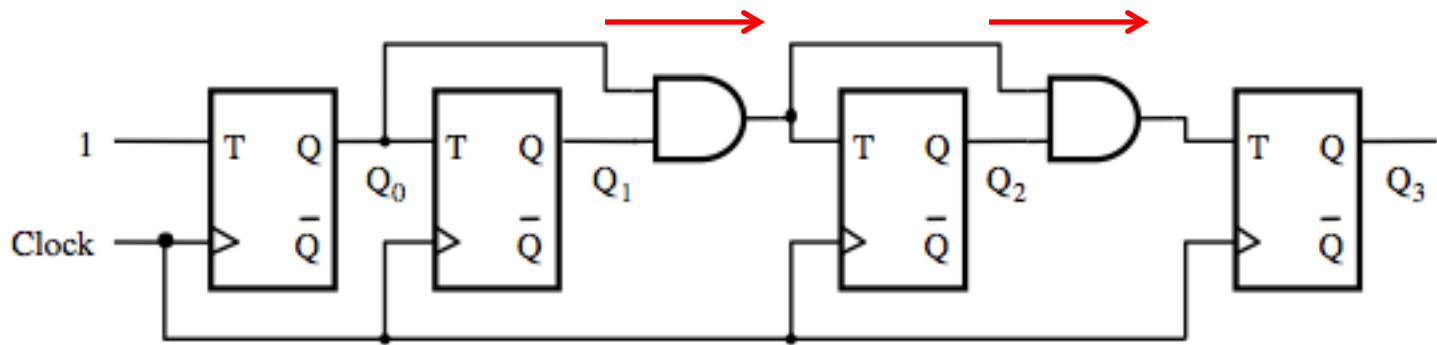
Synkron räknare



$$Q_3 Q_2 Q_1 Q_0 = 1010_2 = 10_{10}$$



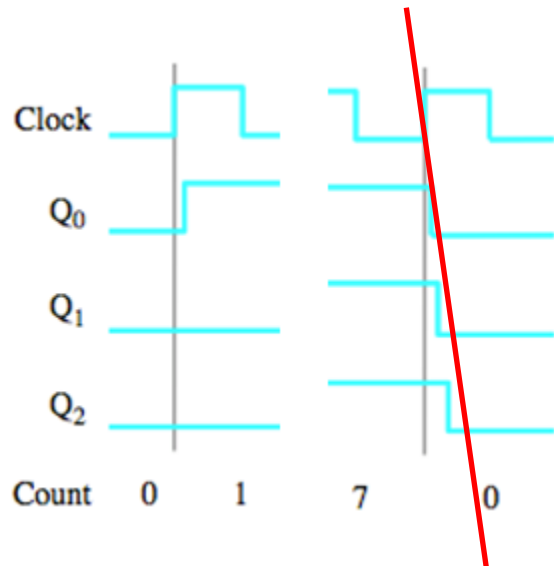
Maximal räknefrekvens?



Den kritiska vägen bestämmer den maximala frekvensen!
Här är den längsta kombinatoriska vägen från Q₀ via två AND-grindar till ingången av vippan som beräknar Q₃
 t_{logic} motsvarar alltså fördröjningen av två AND-grindar.

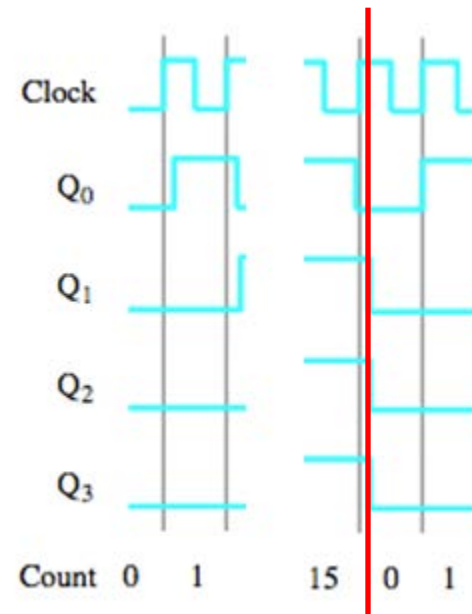
Asynkron eller Synkron räknare

Asynkron räknare



Utgångssignalerna fördröjs mer och mer för varje räknarsteg

Synkron räknare

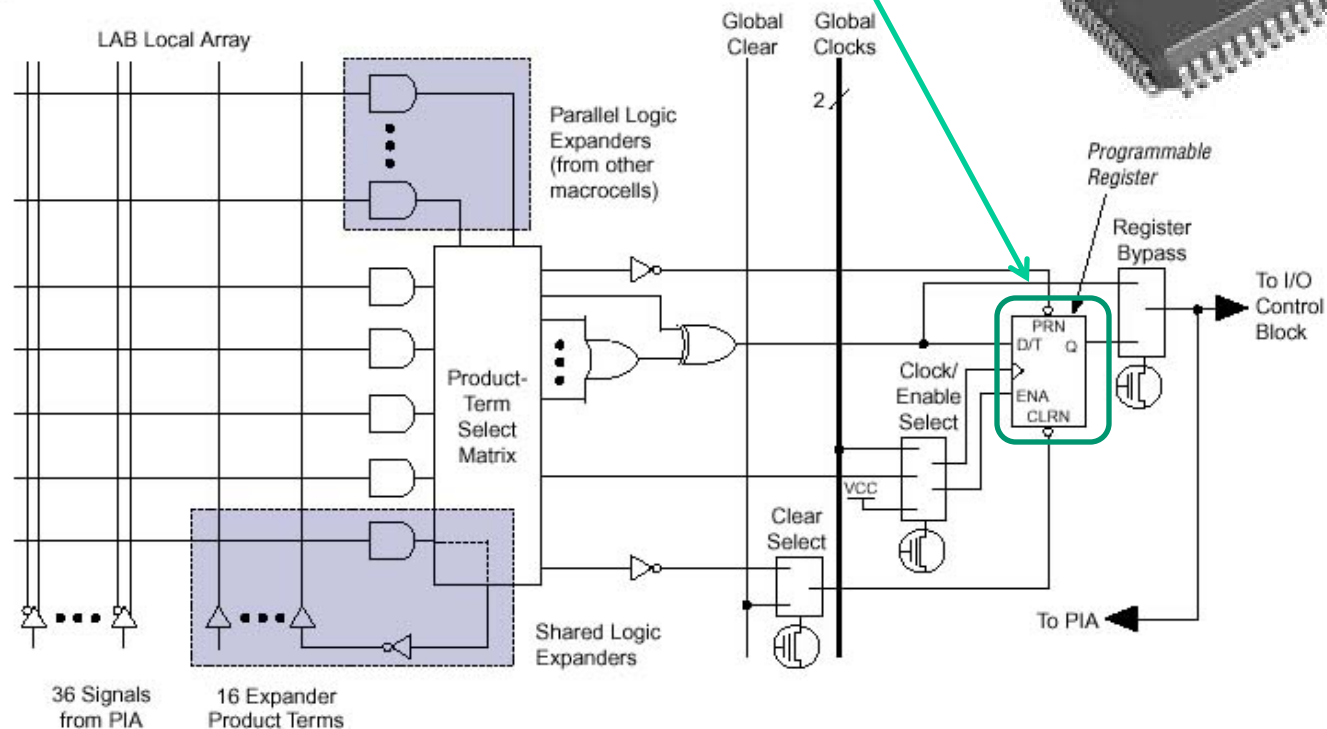


Utgångssignalerna har samma fördröjning

VHDL för vippor och låskretsar

Programmerbar logik har inbyggda vippor.

Figure 2. MAX 3000A Macrocell



VHDL för vippor och låskretsar

Programmerbar logik har inbyggda vippor.

Hur skriver man VHDL-kod som ”talar om” för kompilatorn att man vill använda dom?

En D-latch i VHDL

```

ENTITY D_Latch IS
    PORT(en : IN std_logic;
          d  : IN std_logic;
          q  : OUT std_logic);
END ENTITY D_Latch;
  
```

```

ARCHITECTURE RTL OF D_Latch IS
BEGIN
  
```

```

    PROCESS(en, d)
    BEGIN
        IF en = '1' THEN
  
```

Inget else?  q <= d;

```

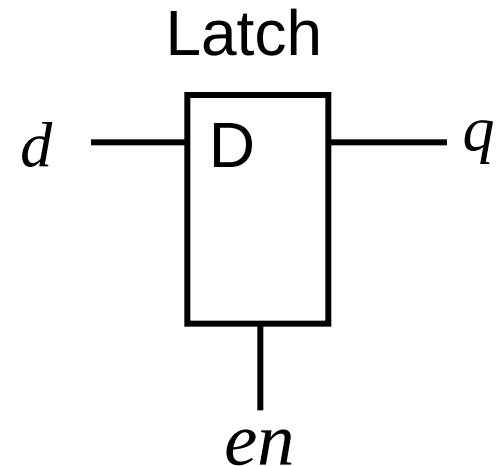
        END IF;
  
```

```

    END PROCESS;
  
```

```

END ARCHITECTURE RTL;
  
```



Enable	D	Q
0	-	M
1	D	D

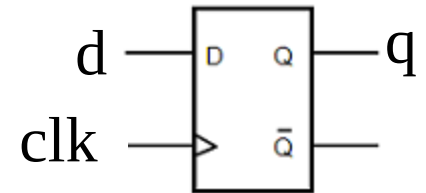
Latch som process

```
PROCESS(en, d)
  BEGIN
    IF en = '1' THEN
      q <= d;
    END IF;
  END PROCESS;
```

Latchar anses generellt vara dåliga ur syntes-synpunkt eftersom de inte alltid är testbara (pga. asynkrona återkopplingar).

Därför undviker man latchar. (Programmerbar logik har inbyggda vippor med asynkron Preset och Clear som man kan använda).

Vippa som process



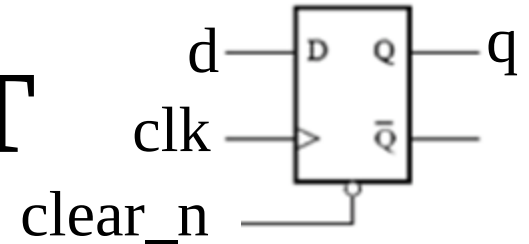
```
PROCESS(clk)
  BEGIN
    IF rising_edge(clk) THEN
      q <= d;
    END IF;
  END PROCESS;
```

*Endast en flank är tillåten
per process*

I stället för funktionen "rising_edge(clk)" kan man skriva "clk'event and clk=1"

Kompilatorn kommer att "förstå" att detta är en vippa och använder någon av de inbyggda vipporna för att implementera processen.

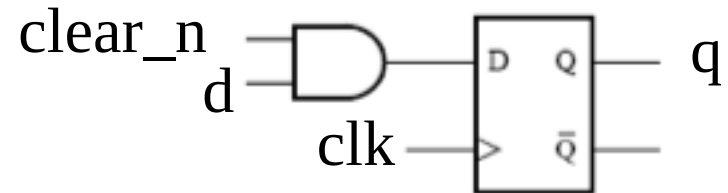
Med asynkron RESET



Clear oberoende av clk

```
PROCESS(clk, clear_n)
BEGIN
    IF clear_n = '0' THEN
        q <= '0';
    ELSE IF rising_edge(clk) THEN
        q <= d;
    END IF;
END PROCESS;
```

Med synkron RESET



```
PROCESS(clk)
  BEGIN
    IF rising_edge(clk) THEN
      IF clear_n = '0' THEN
        q <= '0';
      ELSE
        q <= d;
      END IF;
    END IF;
  END PROCESS;
```

Räknare och andra sekvenskretsar

Vad gör den här ”räknaren” ?

```
bcd:
PROCESS(clk)
BEGIN
    IF rising_edge(clk) THEN
        IF (count = 9) THEN
            count <= 0;
        ELSE
            count <= count+1;
        END IF;
    END IF;
END PROCESS;
```

