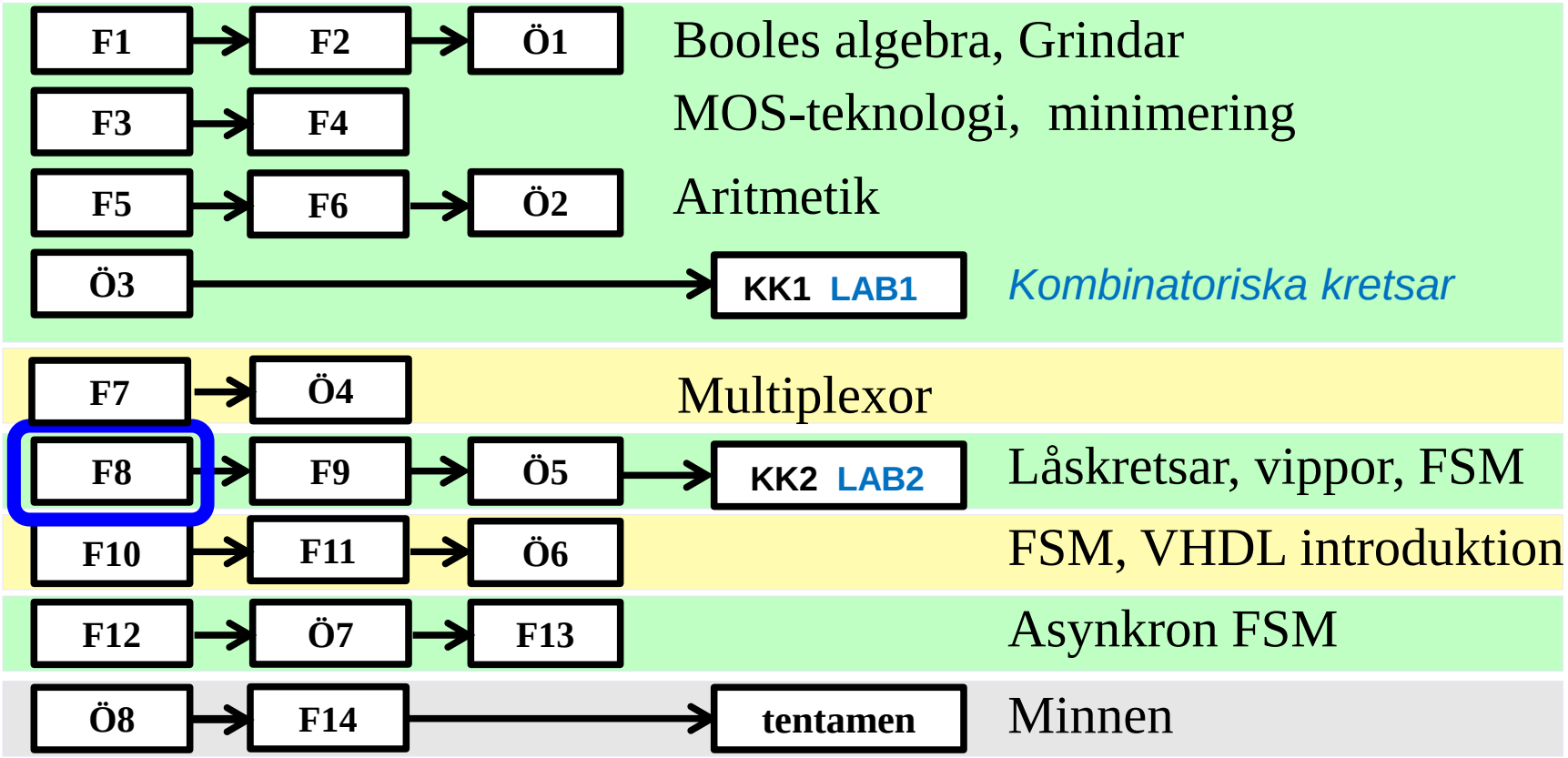
IE1205 Digital Design:

F8: Minneselement:
Latchar och Vippor. Räknare

IE1205 Digital Design



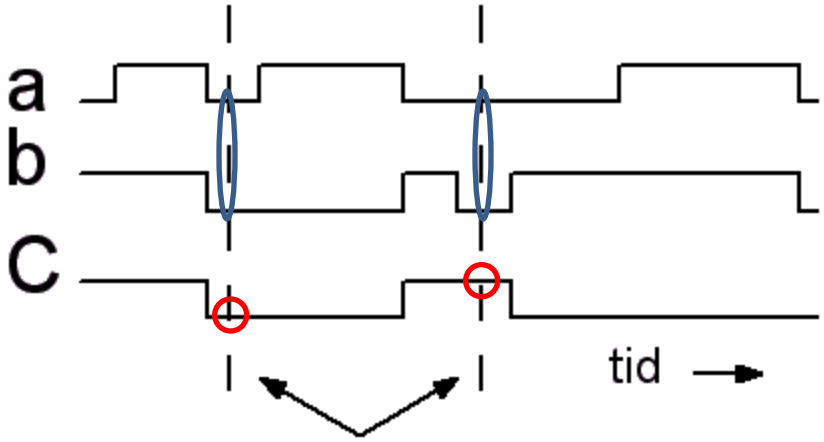
Föreläsningar och övningar bygger på varandra! Ta alltid igen det Du missat!
Läs på i förväg – delta i undervisningen – arbeta igenom materialet efteråt!

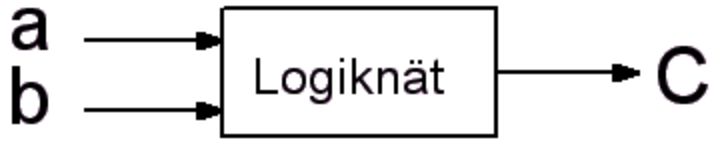
Detta har hänt…

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

Sekvensnät



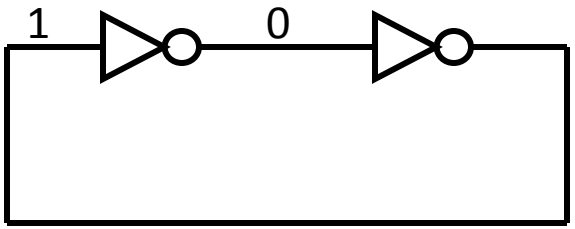


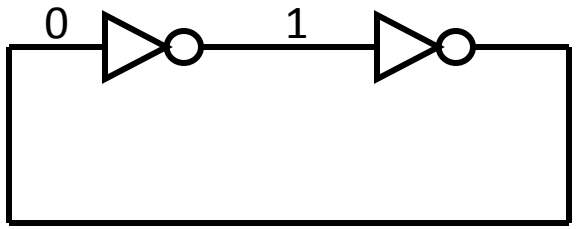
Om en och samma insignal kan ge upphov till olika utsignal, är logiknätet ett sekvensnät.
Det måste då ha ett ***inre minne*** som gör att utsignalen påverkas av både nuvarande och föregående insignaler!

Sekvensmaskiner

- Alla system i naturen har (ofta) ett dynamiskt beteende och förändrar sig ständigt, ibland med återkommande mönster...
- Att bygga digitala system förutsätter att vi kan beskriva dynamiska beteenden, dvs att beräkna, behandla och skapa sekvenser...

Hur får vi hårdvara att ”minnas”?

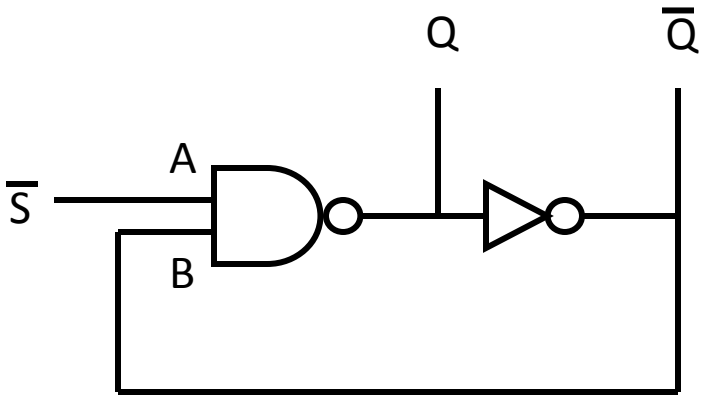




Två återkopplade inverterare har stabila tillstånd

Hur får vi hårdvara att ”minnas”?

Hur kan vi sätta värdet?



NAND grind

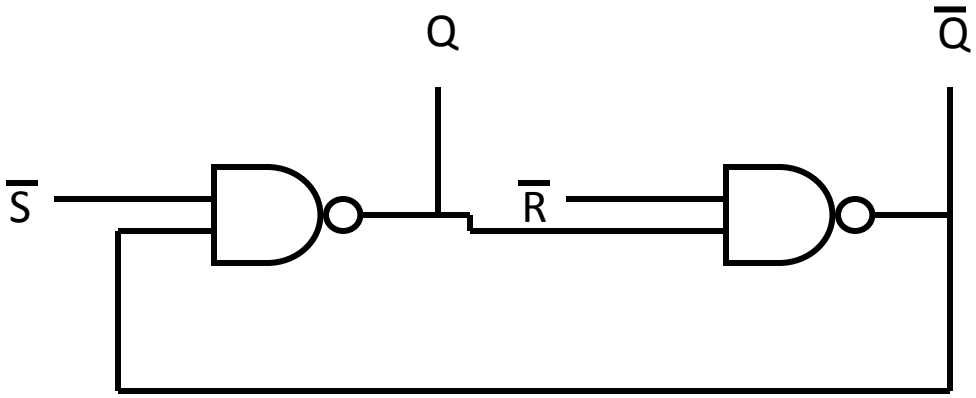
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

Om A ingången är 0 blir utgången 1

Om A ingången är 1 blir B ingången inverterad, värdet behålls

Byt ut ena inverteraren mot en NAND grind

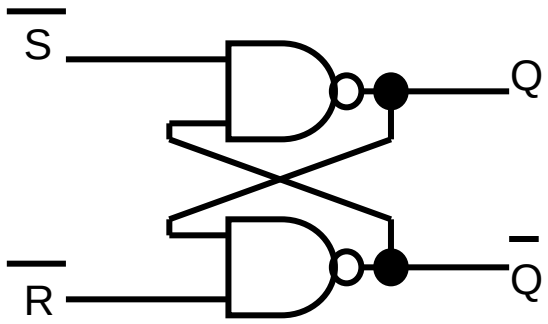
Hur får vi hårdvara att ”minnas”?



Byt ut båda inverterarna för att kunna sätta och återställa (Set and Reset)

SR-Latchen (NAND-grindar)

- SR-latchen har aktiv låga SET och RESET ingånger
- SET och RESET ska inte vara aktiva samtidigt!



Kort återkoppling (~1T)

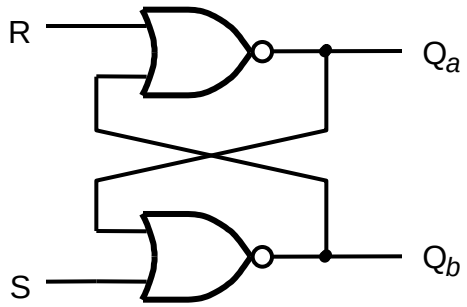
”Förbjudet läge”

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

”Characteristic table”

SR-latchen (NOR-grindar)

- SR-latchen kan även implementeras med NOR-grindar
- Skillnaden mot förra implementeringen är att SET och RESET är aktiv höga



(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

”Förbjudet” tillstånd

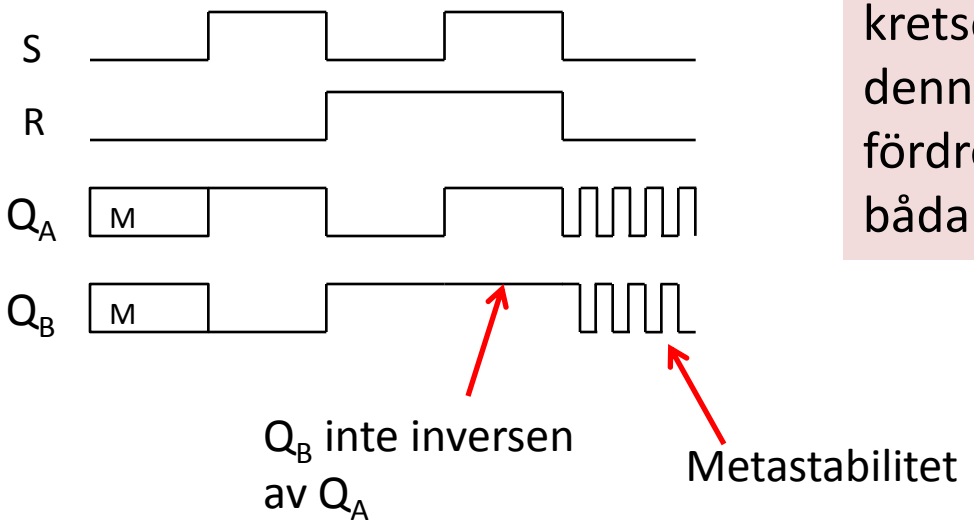
- Varför är vissa tillstånd börbjudna?

Simulering av SR-latchen (BV Figur 7.5 - sida 384)

- Ibland kan vippor låsa sig (pga meta-stabilitet – mer om det i F12):

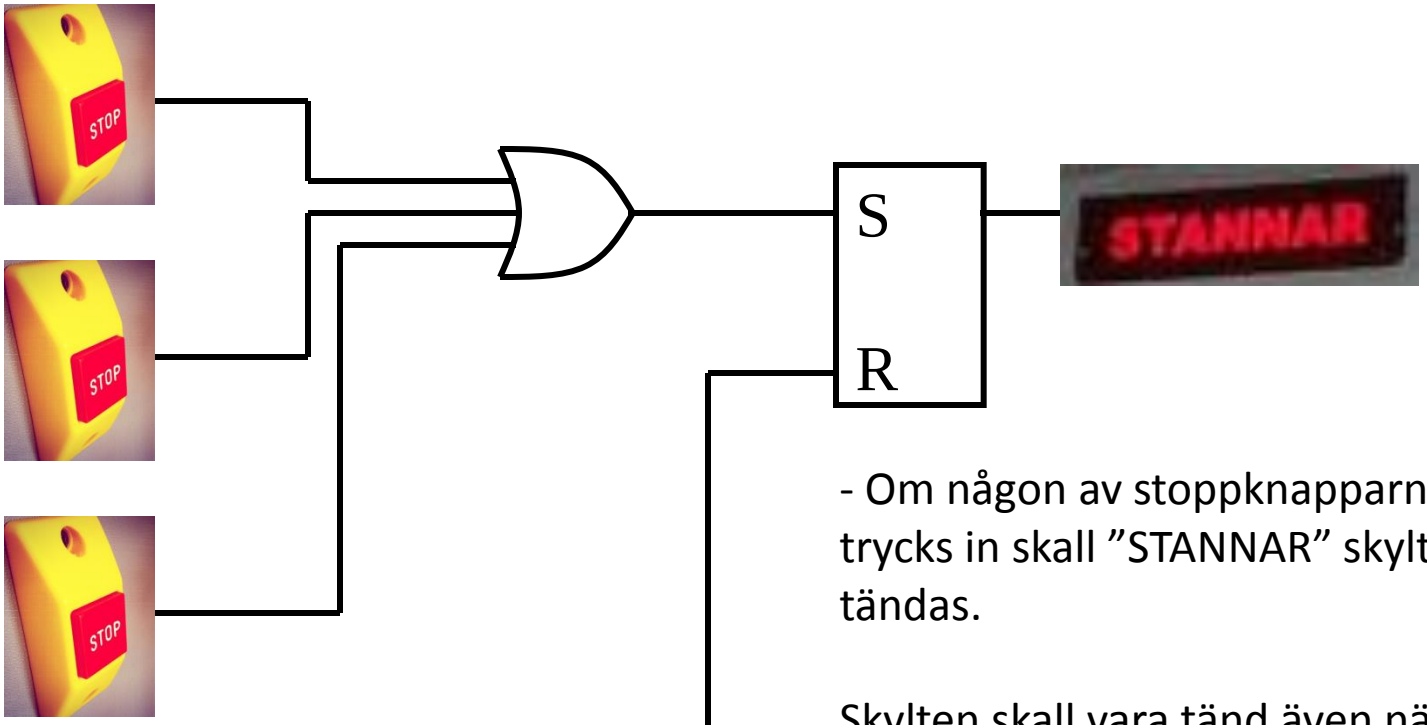
```
latch:    SR_latch port map (S=>S,R=>R, Q=>Q, Q=>Q_inv);

          S<=not(S) after 10 ns;
          R<=not(R) after 20 ns;
```



Om SR-latchen är uppbyggd av NOR-grindar så oscillerar kretsen efter 40 ns med denna insignal-kombination (om fördröjningen är exakt lika i båda NOR-grindar).

Exempel: Stoppknapp i buss

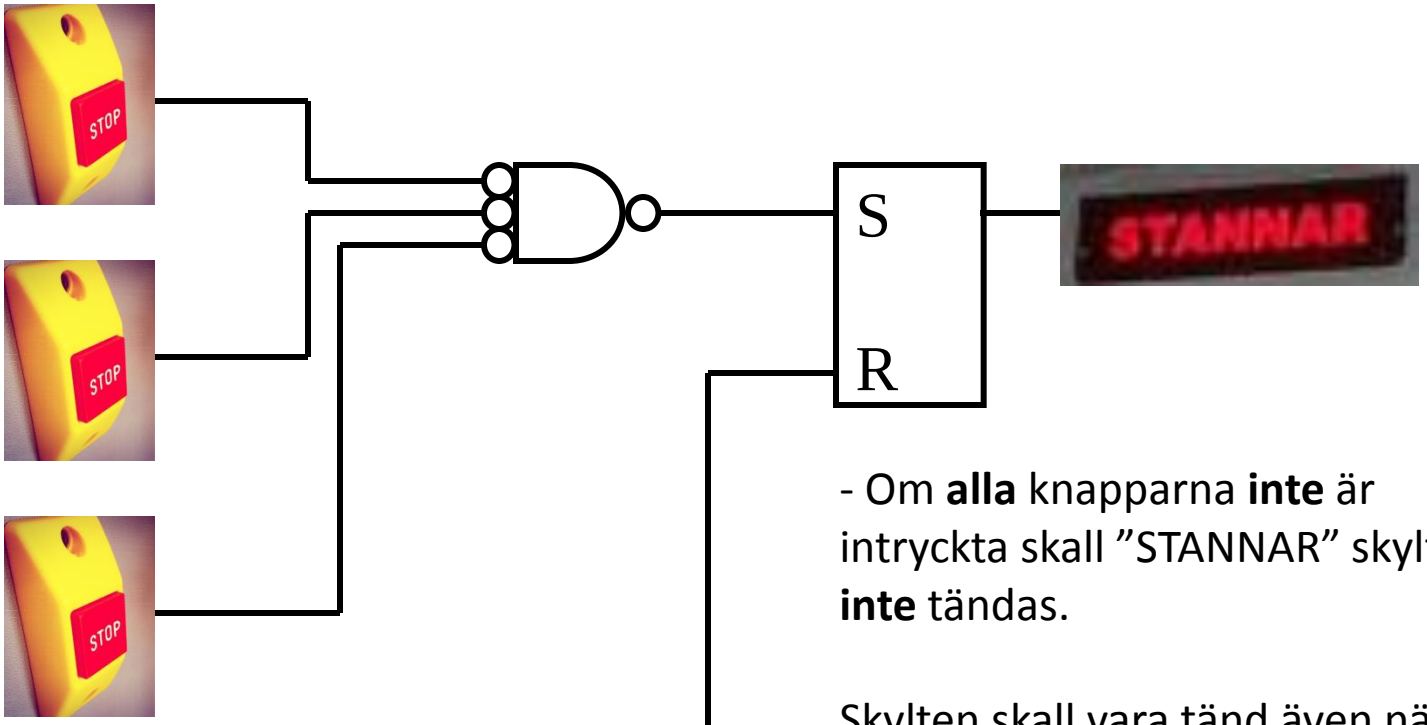


- Om någon av stoppknapparna trycks in skall ”STANNAR” skylten tändas.

Skylten skall vara tänd även när knappen släpps.
Vi behöver en krets med minne

- När chauffören stannar vid hållplats kan han släcka skylten.

Rep: DeMorgan



- Om **alla** knapparna **inte** är intryckta skall ”STANNAR” skylten **inte** tändas.

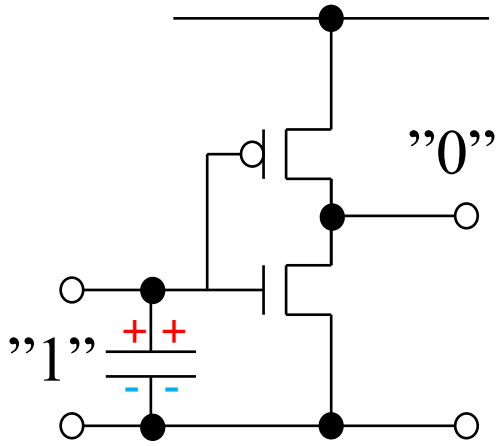
Skylten skall vara tänd även när knappen släpps.
Vi behöver en krets med minne

- När chauffören stannar vid hållplats kan han släcka skylten.

Hur minns hårdvara?

- För att minnas någonting, så måste vi på något sätt hålla kvar informationen.

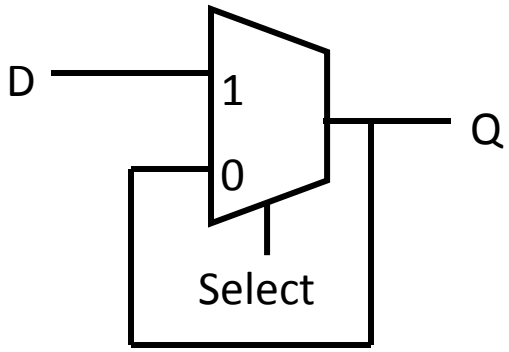
- Ett sätt är att lagra information i form av en laddning på en kapacitans (DRAM).



Det finns andra möjligheter …

Hur får vi hårdvara att minnas?

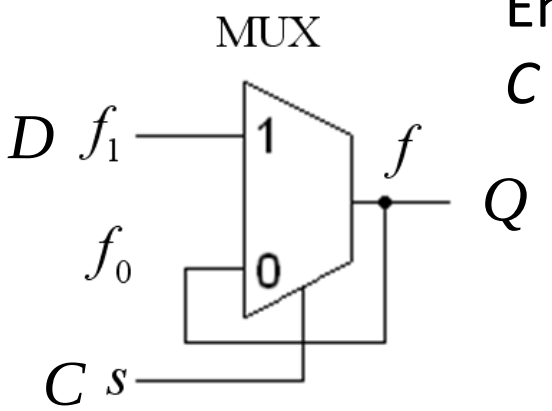
Select	D	Q
0	-	M
1	D	D

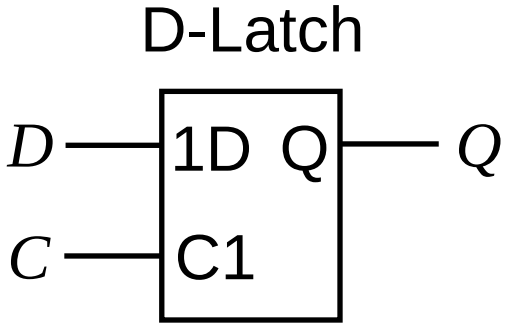


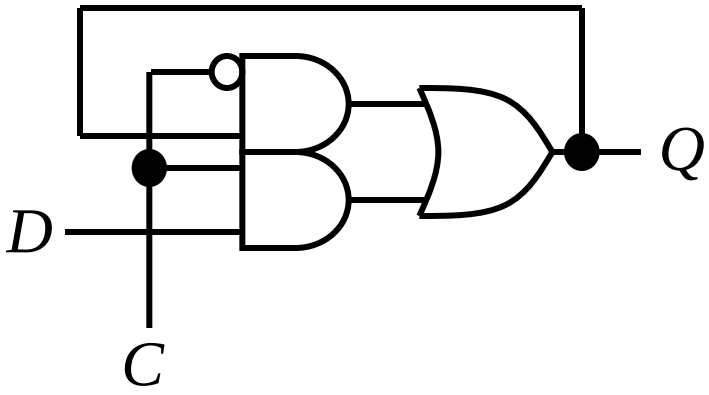
Om vi återkopplar och kopplar utgången f till en av ingångarna så kommer Muxen att få ett nytt värde när den andra ingången selekteras (Select=1), och behålla detta värde när återkopplingen selekteras (Select=0)

D-Latch

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



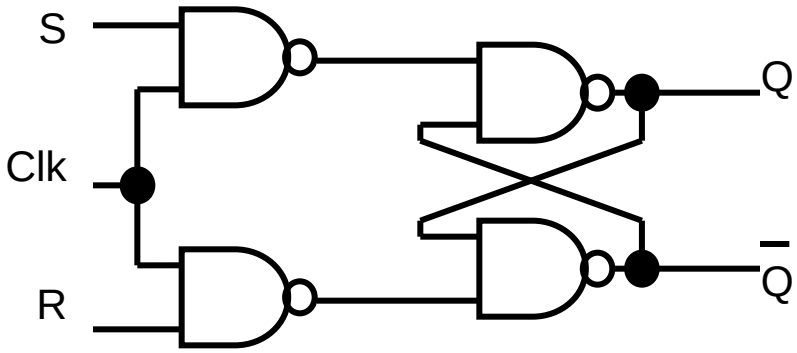




C *follow* / $\overline{latch}$	D	Q
0	–	*M latch*
1	D	*D follow*

Gated SR-Latch

Stoppknapparna kan stängas av om det blir stökigt på bussen

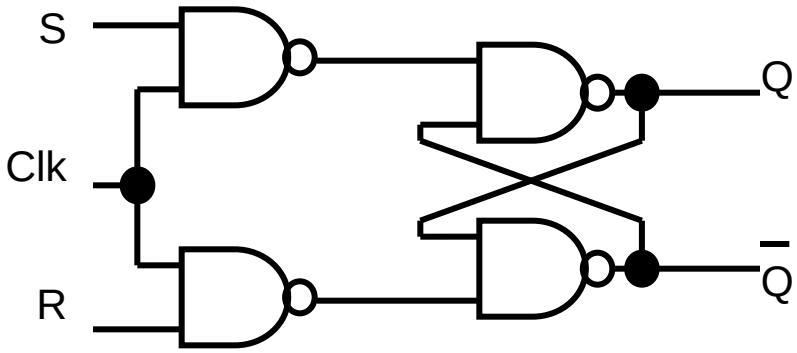


”Förbjudet läge”

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

(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.

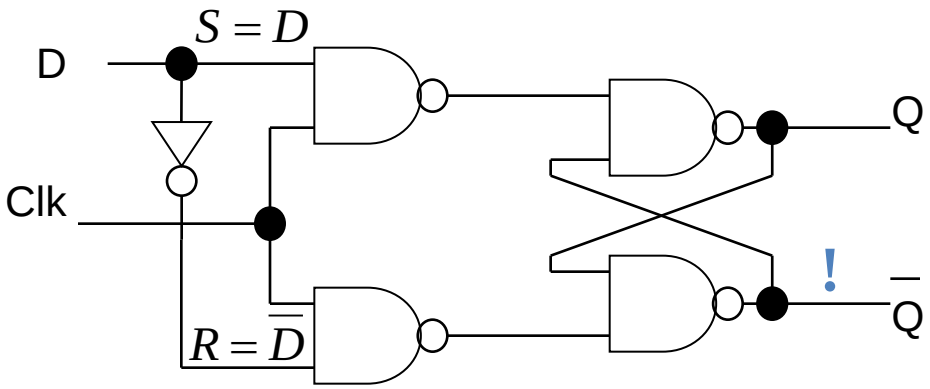


Clk	S	R	Q	$\overline{Q}$
1	0	0	M	M
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1
0	-	-	M	M

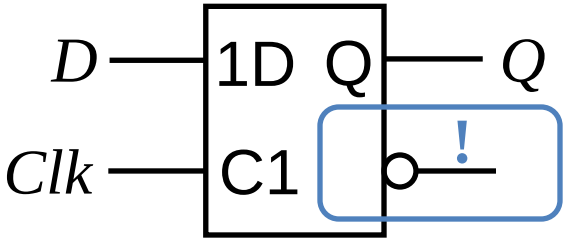
Förbjudet tillstånd

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***.

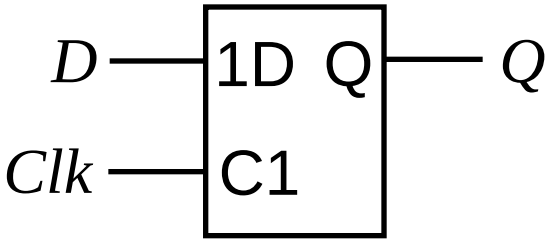


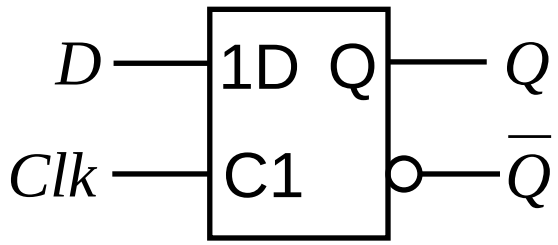
$$Clk = follow / \overline{latch}$$



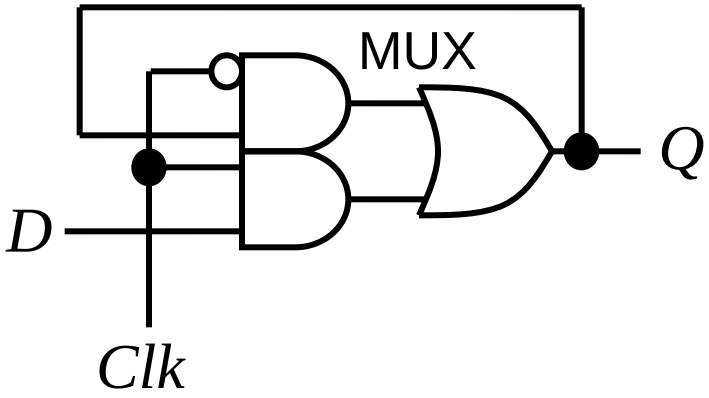
Clk	D	Q	$\overline{Q}$
1	0	0	1
1	1	1	0
0	-	M	M

Två olika D-latchar

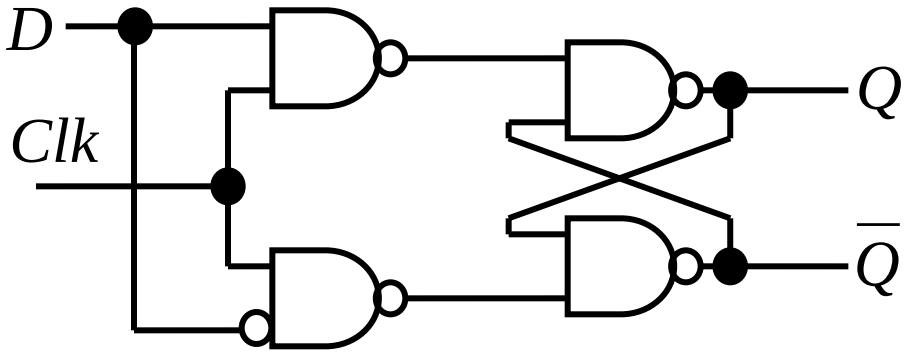




Lång återkoppling (~4T)

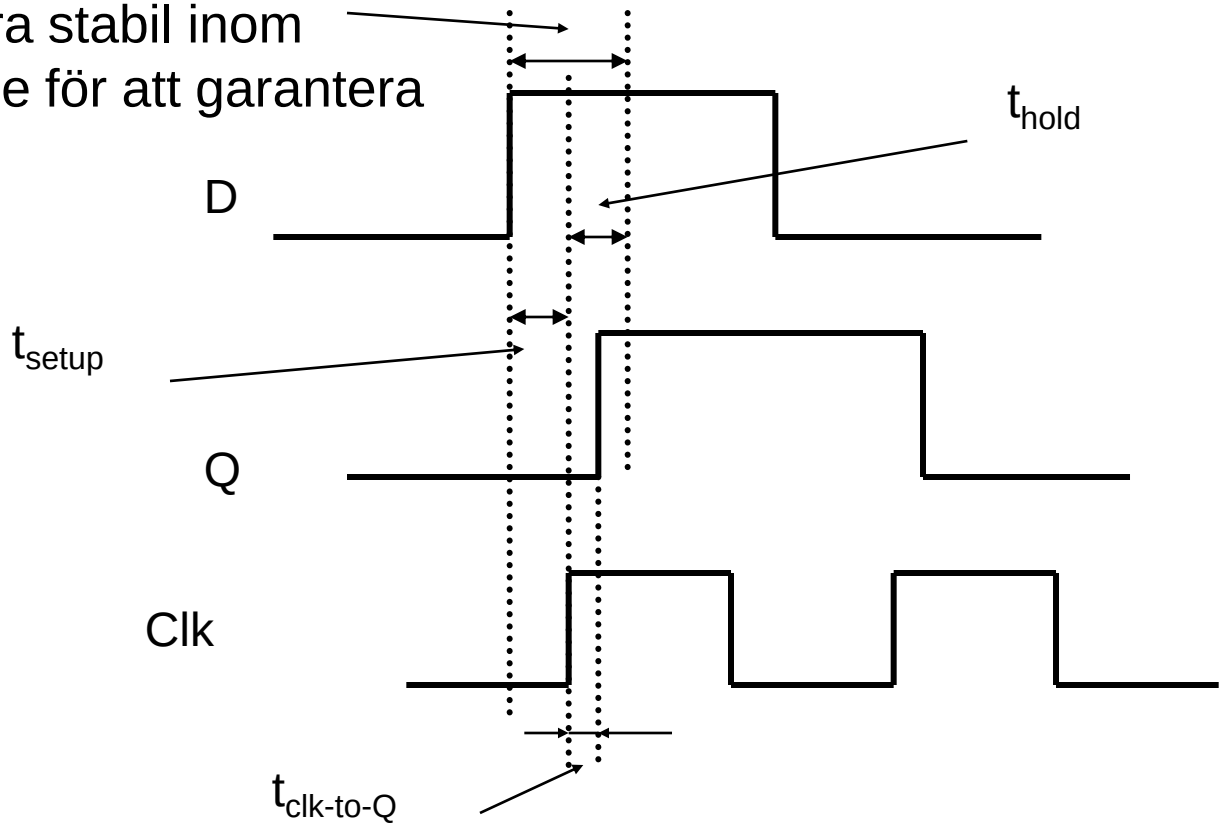


Kort återkoppling (~1T)



Setup- & Hold-time

D måste vara stabil inom detta område för att garantera funktionen



Hur skapar vi en sekvens?

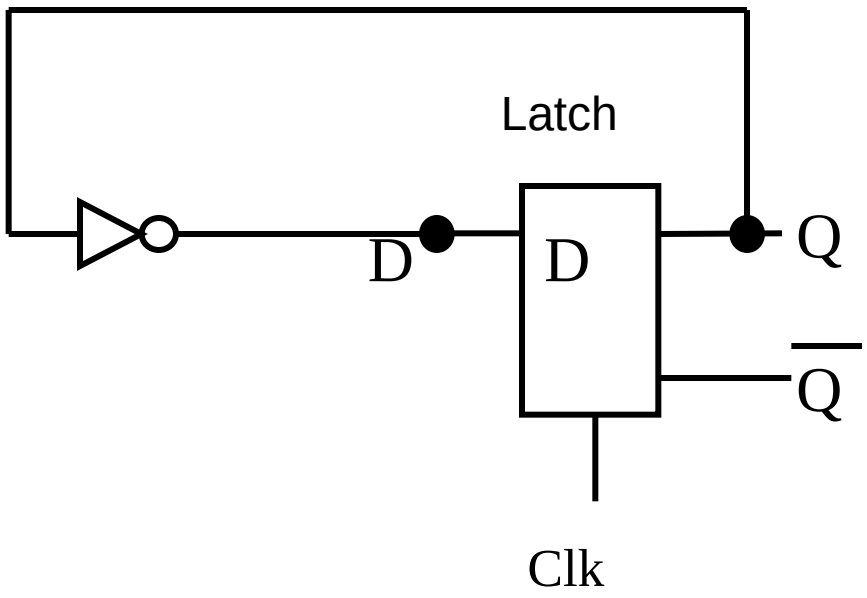
- En sekvens får vi om vi tar ett värde och sedan bestämmer nästa värde utifrån nuvarande värde.

Ex: 0,1,0,1,…

nästa värde = NOT (nuvarande värde)

- Om vi återkopplar nuvarande värde, behandlar det (inverterar) och därigenom skapar nästa värde samt kommer ihåg det tills nästa värde skall beräknas

Sekvensmaskiner (forts.)

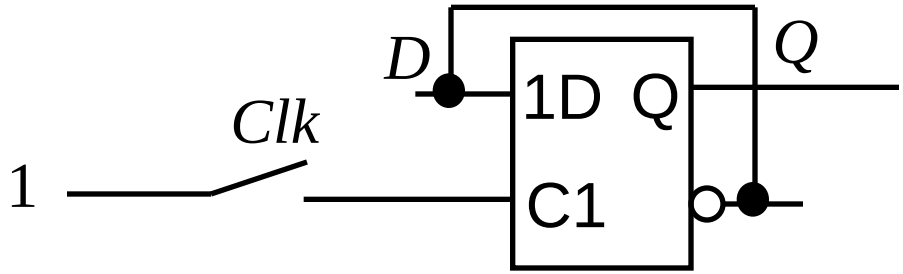


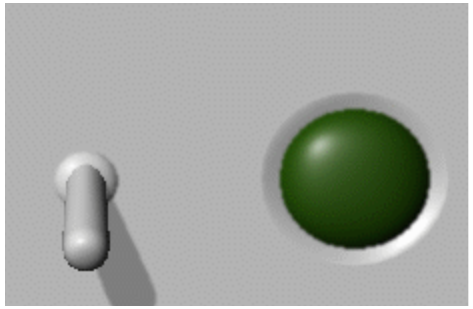
Problem!!! Om CLK är 1 för länge så snurrar värdena bara runt med en period av $T_{latch}+T_{incr}$

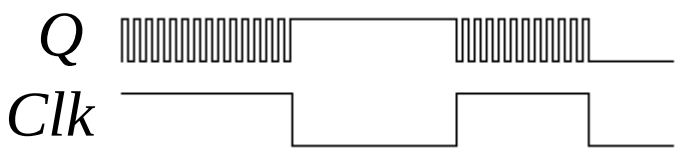
Går ej med enkel låskrets …

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

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



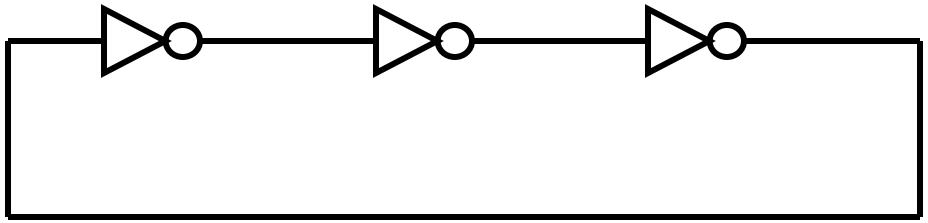




- När *Clk* = 1 följer utsignalen insignalen – därför byter utgången 1/0 så fort som möjligt! Kretsen blir en oscillator!

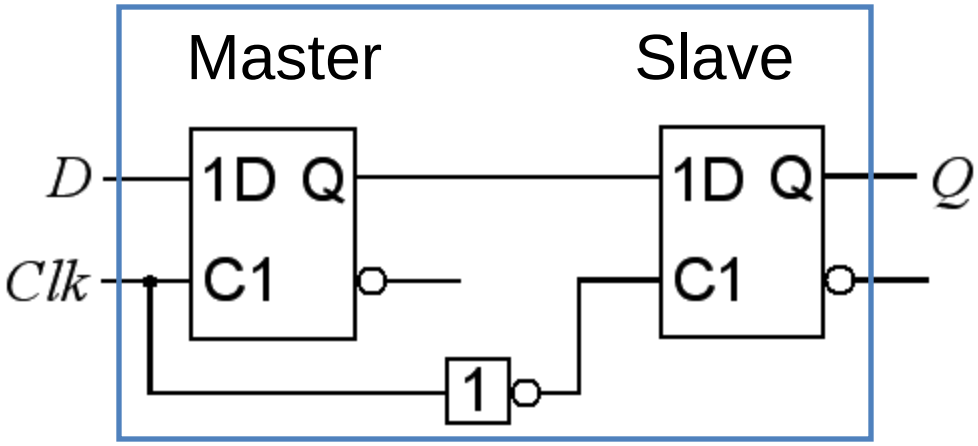
- När sedan *Clk* = 0 behåller utsignalen sitt värde 1/0 allt efter vad den senast stod på. (= Slumpgenerator?)

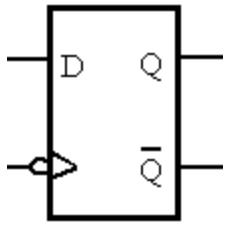
Instabila kretsar



Exempel ringoscillator

Klockade vippor
Master-Slave vippan

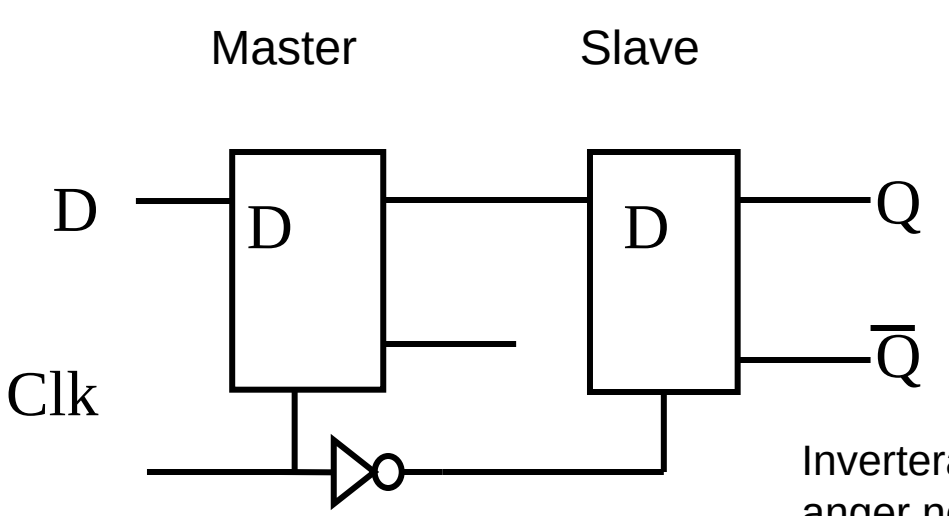




Problemet är att den **enkla låskretsen** är öppen för förändring ända fram tills den ska låsa sitt värde.

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

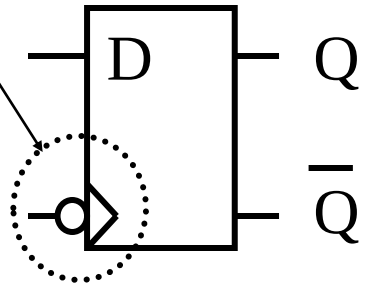
D-vippan (eng. Flip-flop)



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

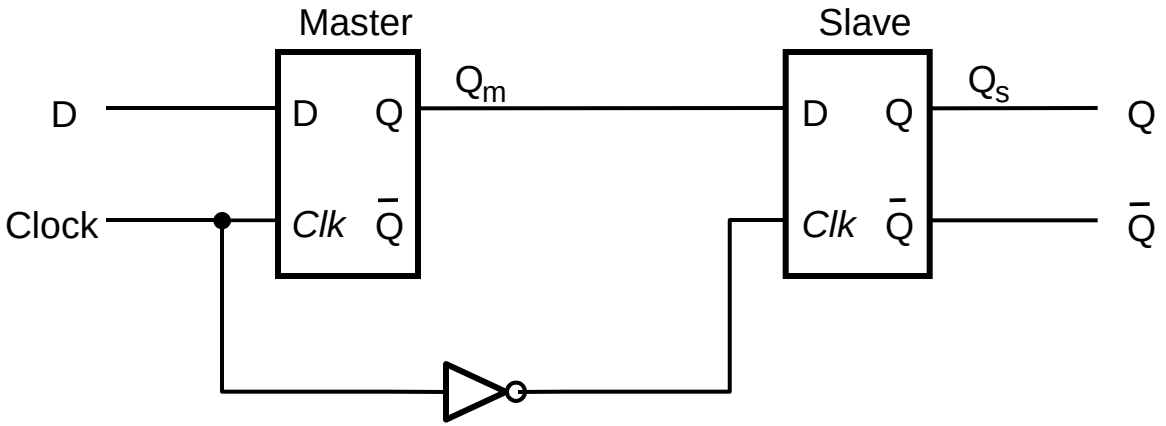
Inverterar-ring på clk anger negativ flank.

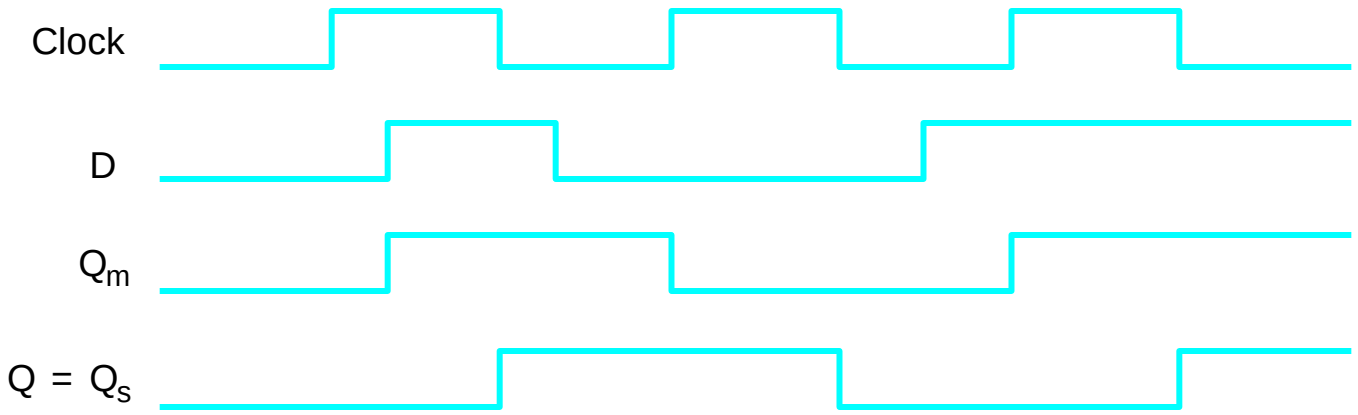
D-vippa



Lösning: Koppla två D-latchar efter varandra!

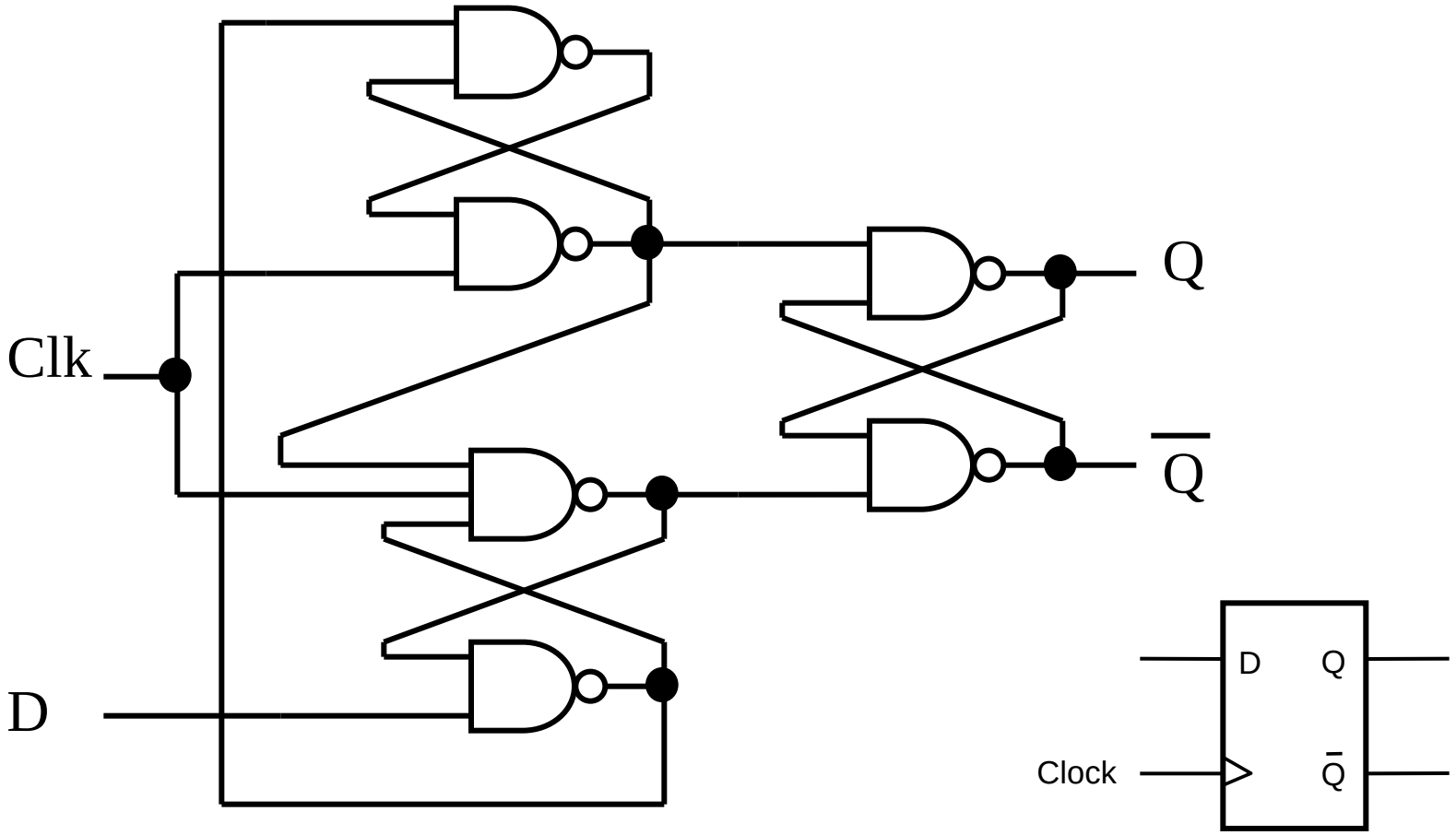
Tidsdiagram Master-Slave





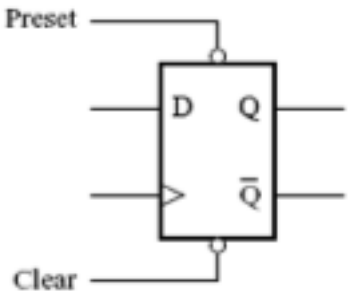
Flank-triggad D-vippa

Flank-triggad: Låser värdet då klockan **ändras** i viss riktning (tex 0 till 1)



Vippor med Clear och Preset ingång

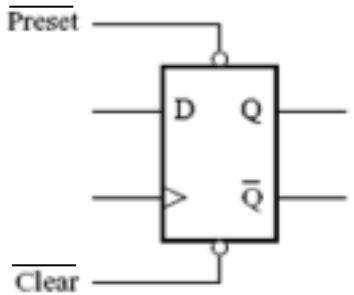
- Det är viktigt för konstruktionen av en sekvenskrets att man kan sätta vippor i ett förbestämt läge
  - Preset: Sätt vippan till 1
  - Clear: Sätt vippan till 0



Reset-knappen

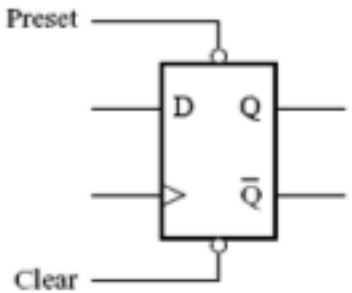
De flesta digitala system behöver kunna startas i ett känt tillstånd. Det kan innebära att en del vippor ska vara ”1” medan andra ska vara ”0”. En resetfunktion kan därför behöva anslutas till antingen **Preset** eller **Clear** på de ingående vipporna.

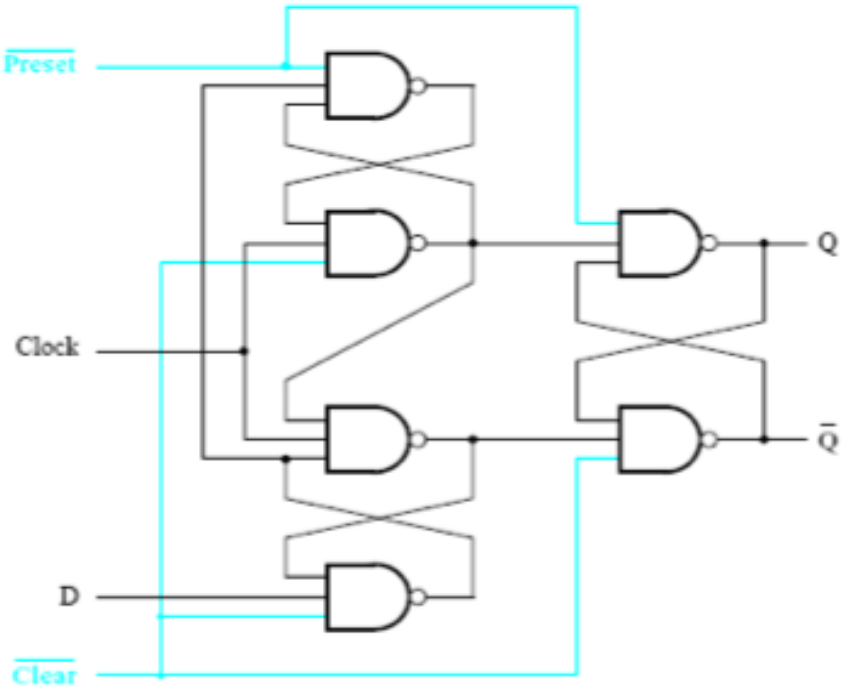
Preset och **Clear** är asynkrona ingångar – vipporna ändrar sig omedelbart oavsett klockpuls.



Asynkron Reset

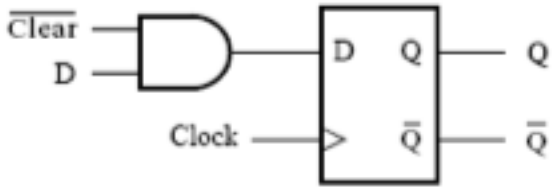
- En asynkron reset (clear) betyder att vippan ändra tillståndet till 0 omedelbart när reset är aktiv





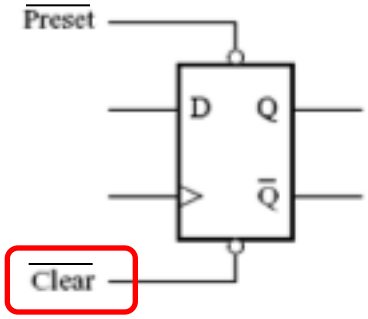
Synkron Reset

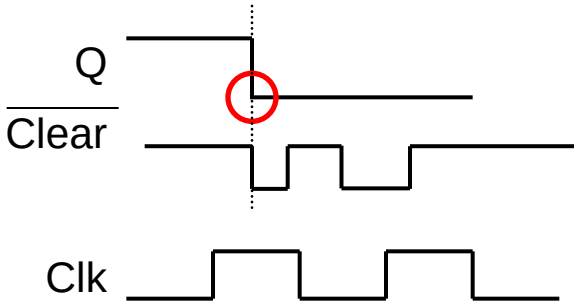
- En synkron reset orsaker att vippan går till läge 0 vid nästa klockflank
- Synkron reset implementeras med extra logik



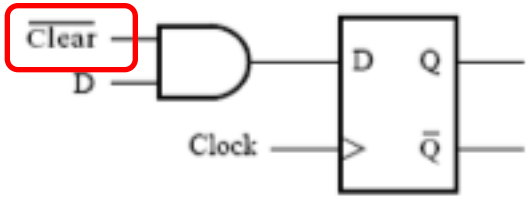
Asynkron/Synkron Reset

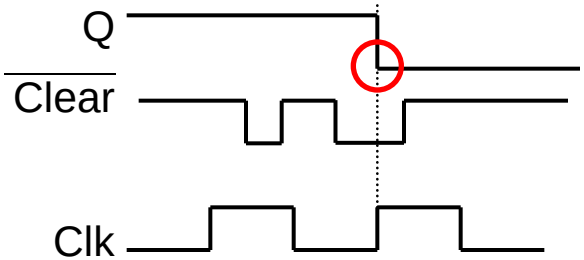
Asynkron reset





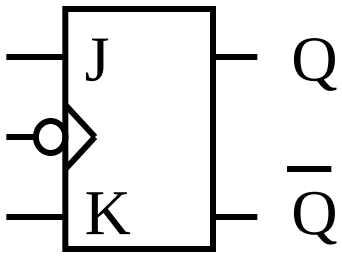
Synkron reset





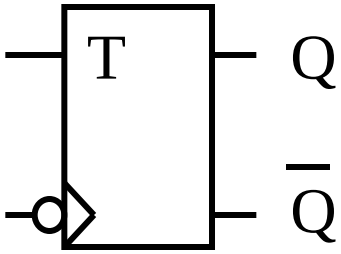
Andra vanliga typer av vippor

JK-vippa (av Jack Kilby - Nobelpriset 2000)



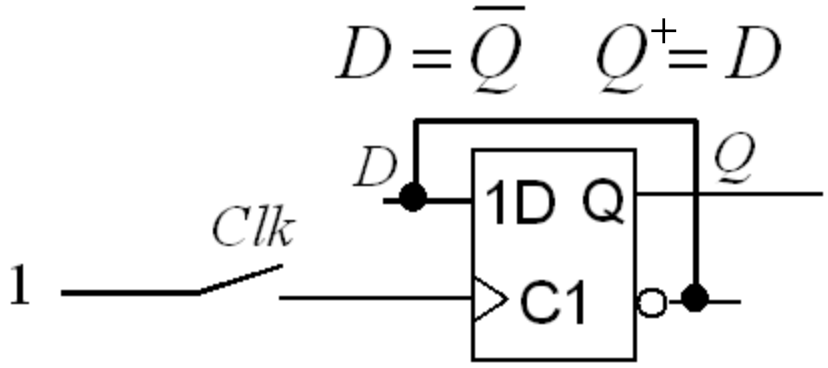
Clk	J	K	Q	$\overline{Q}$
↓	0	0	M	M
↓	0	1	0	1
↓	1	0	1	0
↓	1	1	Toggle	Toggle

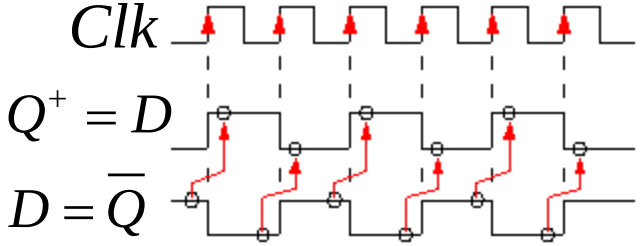
T-vippa (T=Toggle)

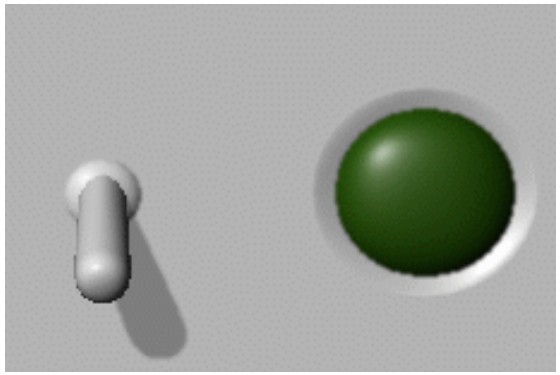


Clk	T	Q	$\overline{Q}$
↓	0	M	M
↓	1	Toggle	Toggle

Varannan gång (vippa)?





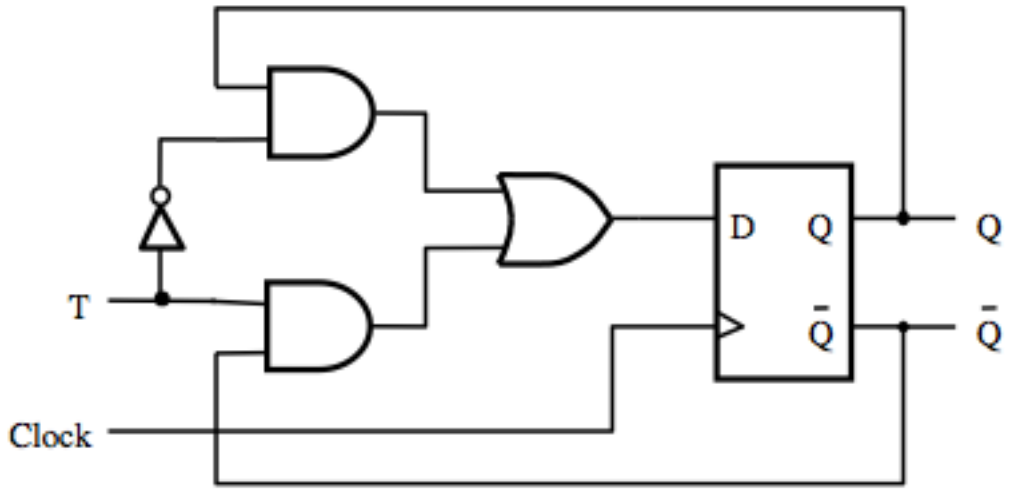


Nu fungerar ”varannan gång” precis som tänkt!

Till sekvensnät använder man i allmänhet flanktriggade vippor som minneselement!

Konstruktion av T-vippan mha D-vippan

- Man kan konstruera nya vippor baserad på en existerande typ

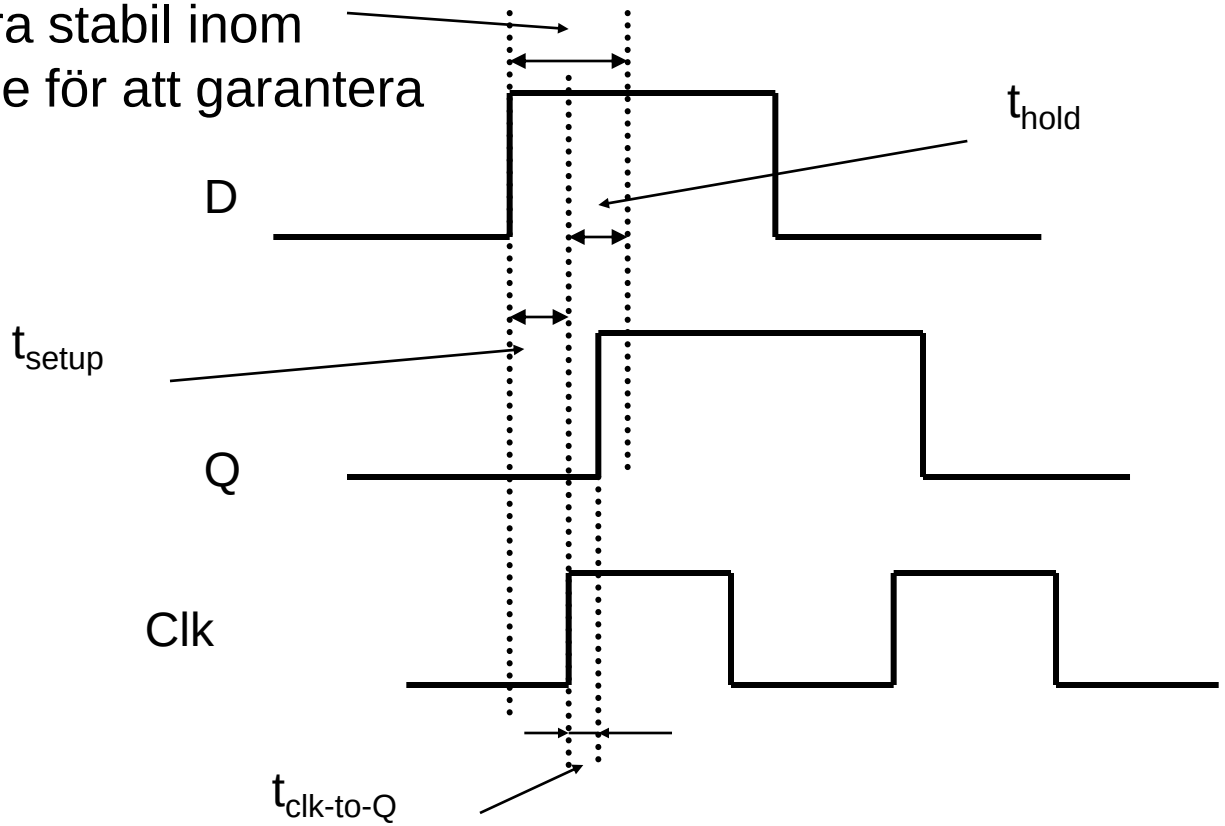


Timing Analysis

- 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

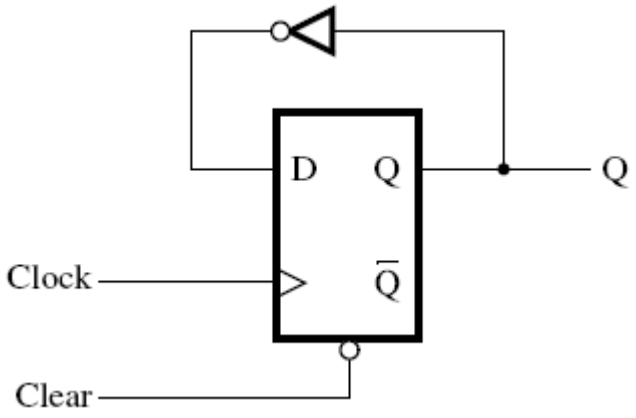
Setup- & Hold-time

D måste vara stabil inom detta område för att garantera funktionen



Vad är den maximala frekvensen?

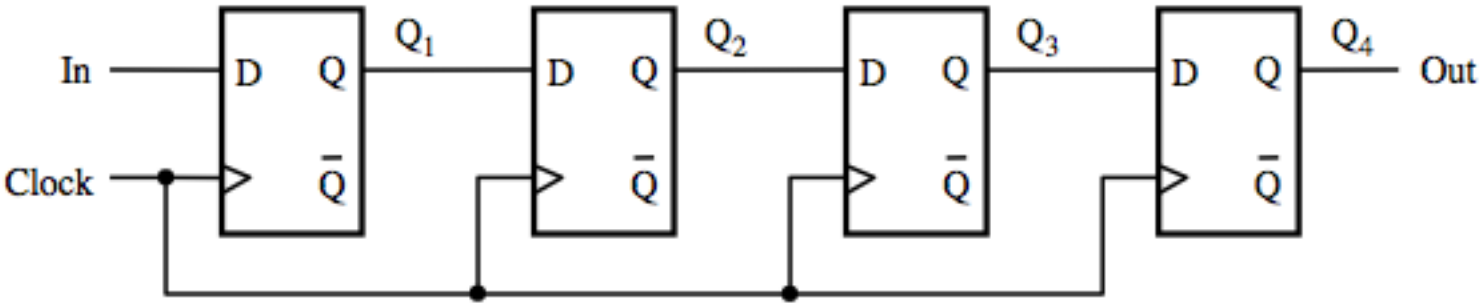
- Grindfördröjningar
  - $t_{logic} = t_{NOT} = 1.1$ ns
- Setup-tid
  - $t_{su} = 0.6$ ns
- Hold-tid
  - $t_h = 0.4$ ns
- Clock-to-utgång
  - $t_{cQ} = 1.0$ ns



$T = t_{su} + t_{cQ} + t_{logic} = 2.7$ ns
$F = 1/T = 370$ MHz

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, …, Q_1 som ingångsvärden till andra komponenter



Vanliga typer av Shift-register

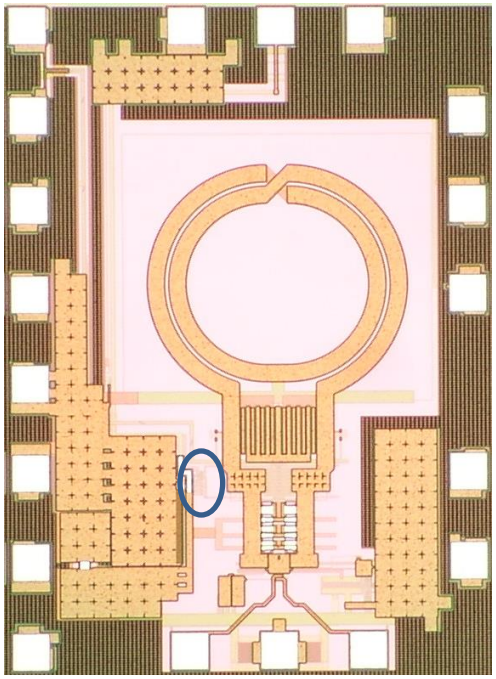
- Parallel-In/Parallel-Out (PIPO)
- Parallel-In/Serial-Out (PISO)
- Serial-In/Parallel-Out (SIPO)
- Serial-In/Serial-Out (SISO)

- Användningsområden
  - Köer, tex First-In/First-Out (FIFO)
  - Mönsterigenkänning (eng. Pattern recognizers)

Exempel shift-register

Clk

Data

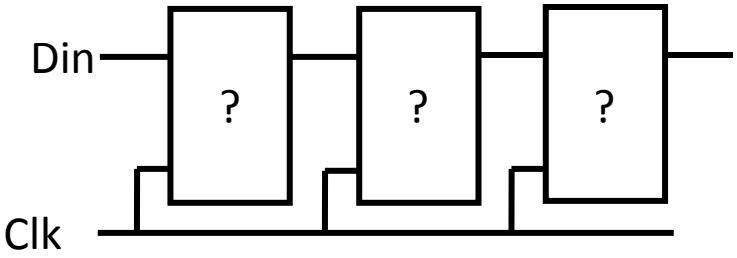


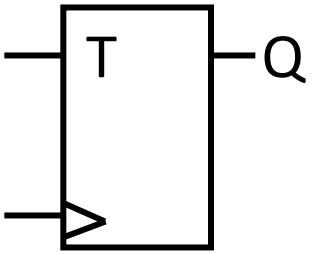
Problem:
I ett experiment skulle ett flertal kontrollsignaler konfigureras.
(Enable, frekvens, ström etc.)
Endast ett fåtal ben fanns tillgängliga i kretsen

Lösning:
Ett skiftregister möjliggör seriell konfigurering med enbart två ben

Snabbfråga

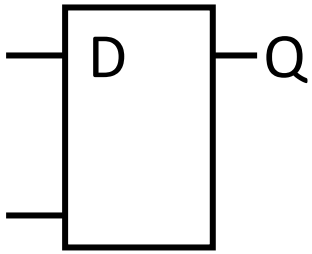
Vilken typ av vippa skall vi använda för att konstruera ett skift-register?





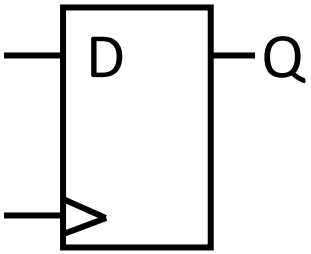
T - vippa

Alt: 1



D - latch

Alt: 2



D – vippa
(flank triggad)

Alt: 3

Räknare

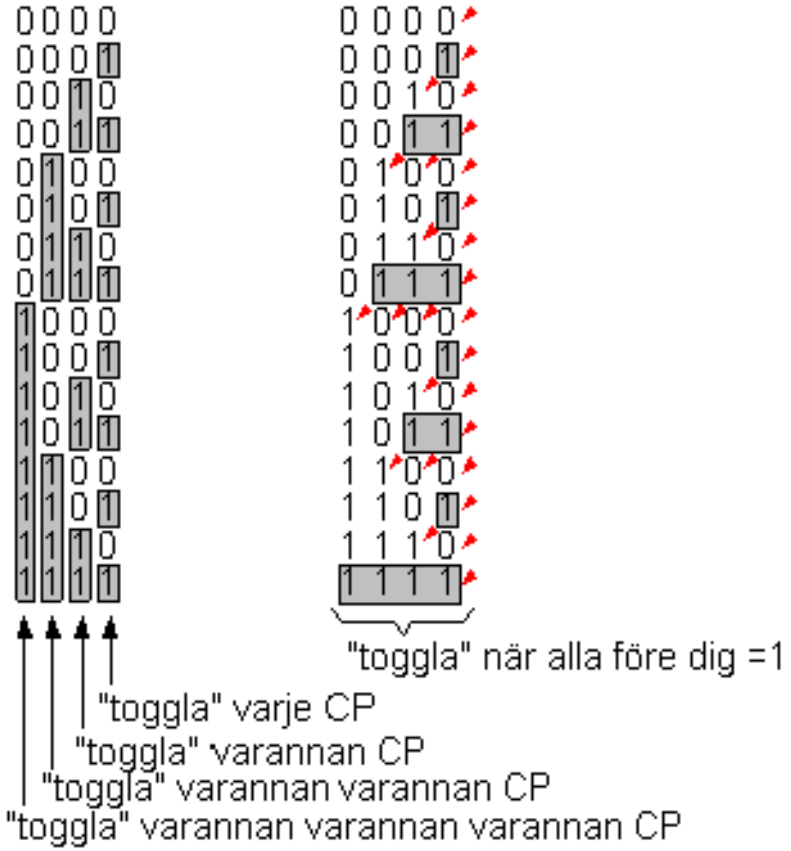
En räknare är en speciell typ av sekvensnät som registrerar antalet inkommande klockpulser. Registreringen sker efter någon kod, oftast **binärkod**. Efter ett visst antal pulser tar räknarens tillstånd slut och den börjar om från början igen.

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

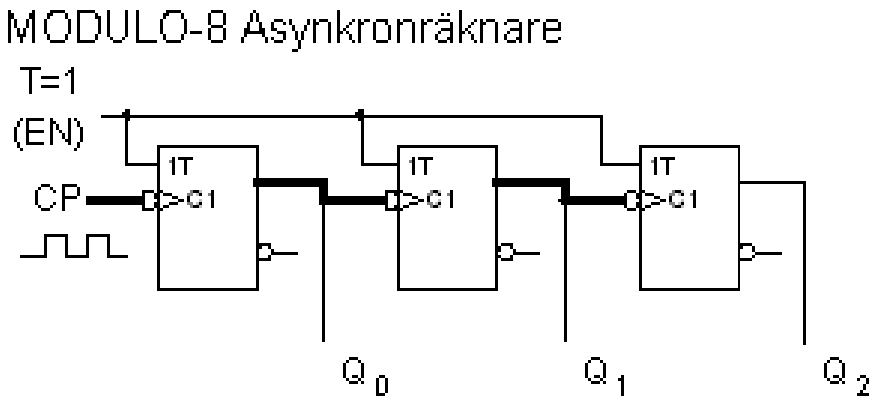
Räknaren behöver inte ha någon insignal utom klockpulserna (som då kan ses som insignalen).
Ett sådant sekvensnät kallas för **autonomt**.

Binärkodens räkneegenskaper

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



Toggla varannan gång …

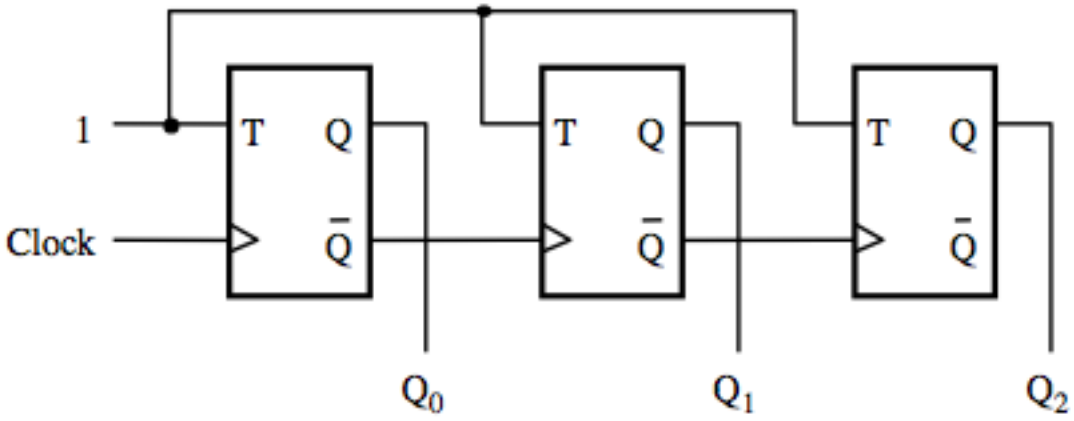


varannan, varannan varannan, varannan varannan varannan …

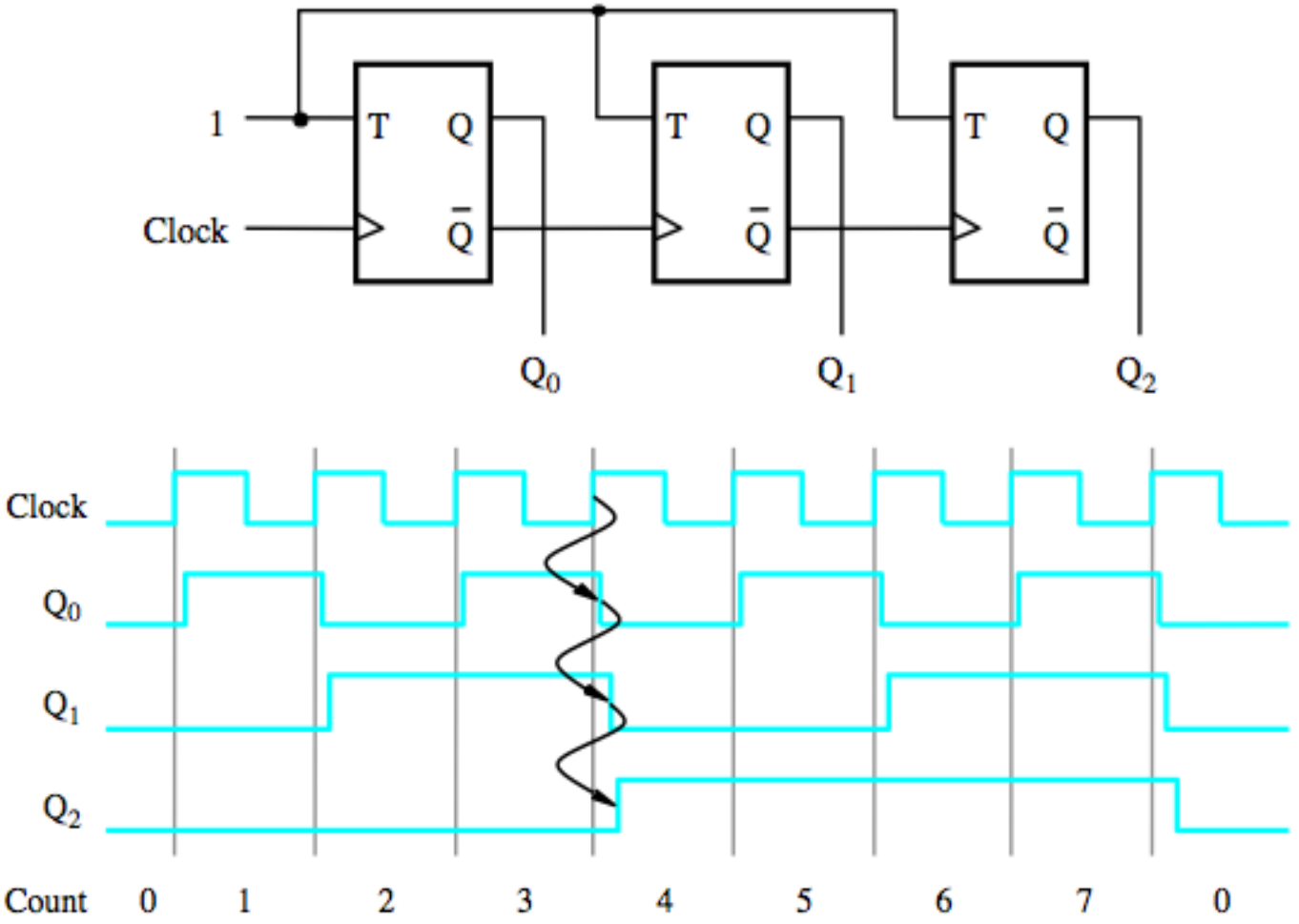
Räknaren är uppbygd av T-vippor, de har alla T=1 och "togglar" därför vid varje klockpuls. Den första vippan Q_0 "togglar" för varje klockpuls. Vippan därefter Q_1 klockas av den första vippan. Den kommer därför bara att "toggla" för varannan klockpuls. Den tredje vippan Q_2 kommer "toggla" för varannan varannan klockpuls.
Enligt binärtabellen kommer räknaren därför att räkna i binärkod.
($Q_2Q_1Q_0$: 000 001 010 011 100 101 110 111 000 ...).

Asynkron räknare

- Man kan realisera räknare mha vippor
- Nedanstående exempel visa en *asynkron* räknare
- Några klockingångar är kopplade till Q'-utgångarna från föregående vippa

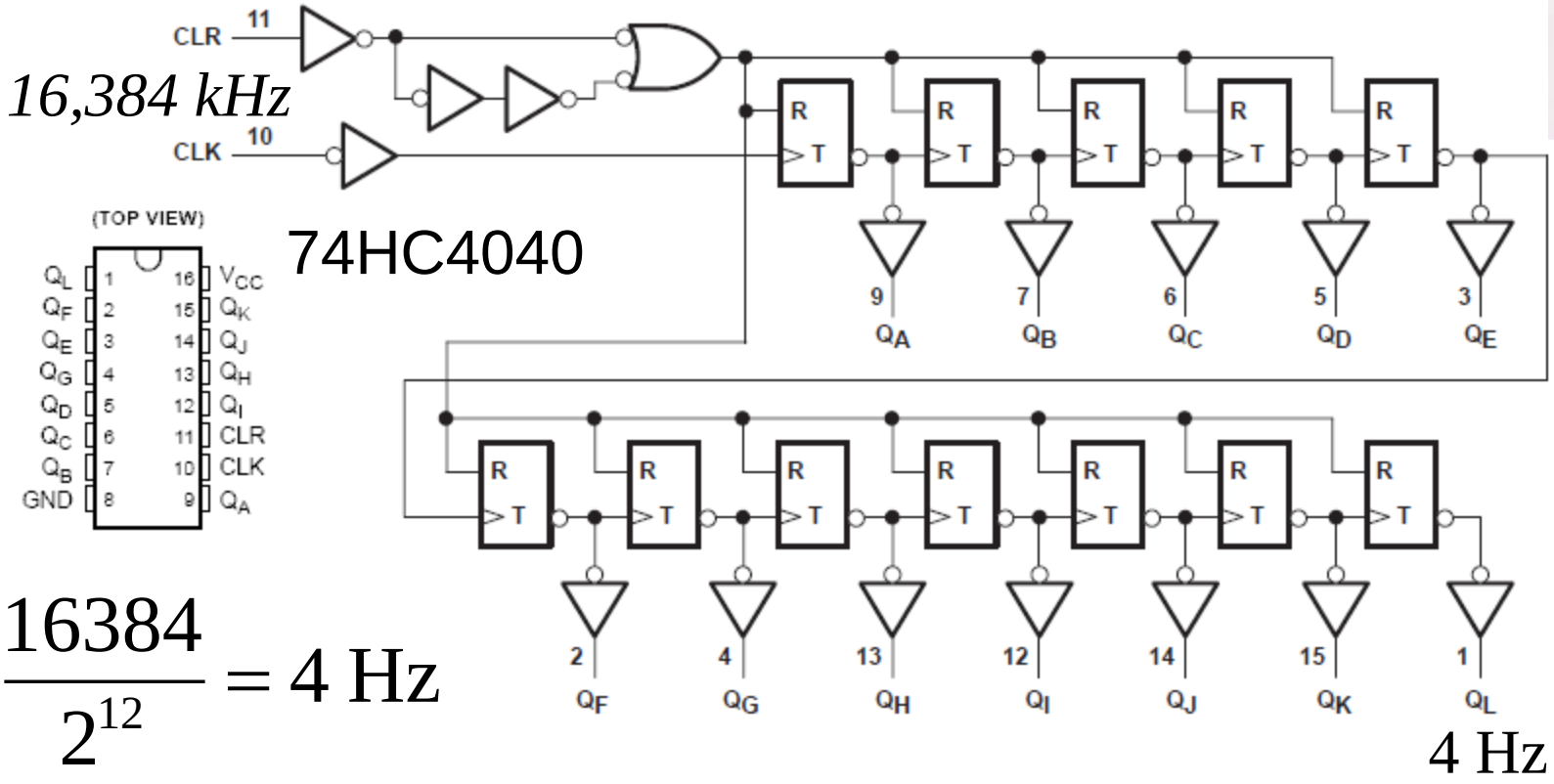


Asynkron räknare



En räknarkrets

16,384 kHz



(TOP VIEW)

74HC4040

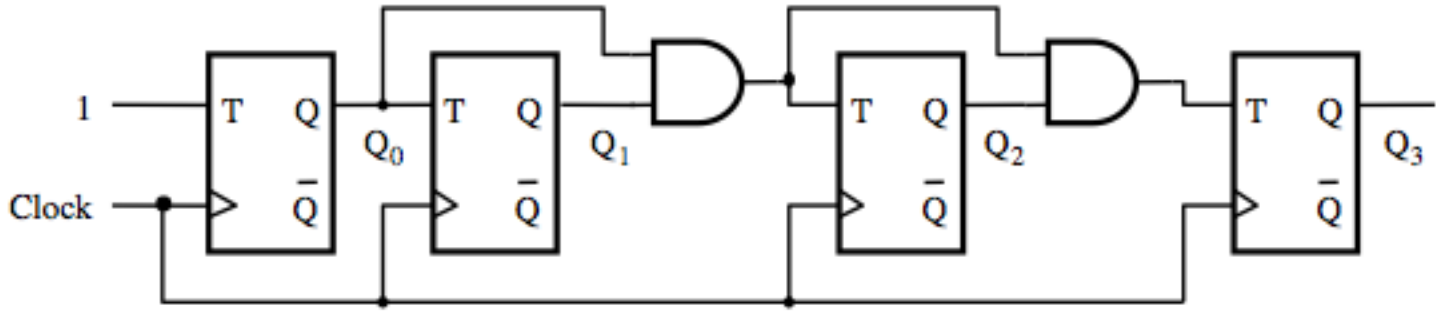
$$\frac{16384}{2^{12}} = 4\ \text{Hz}$$

4 Hz

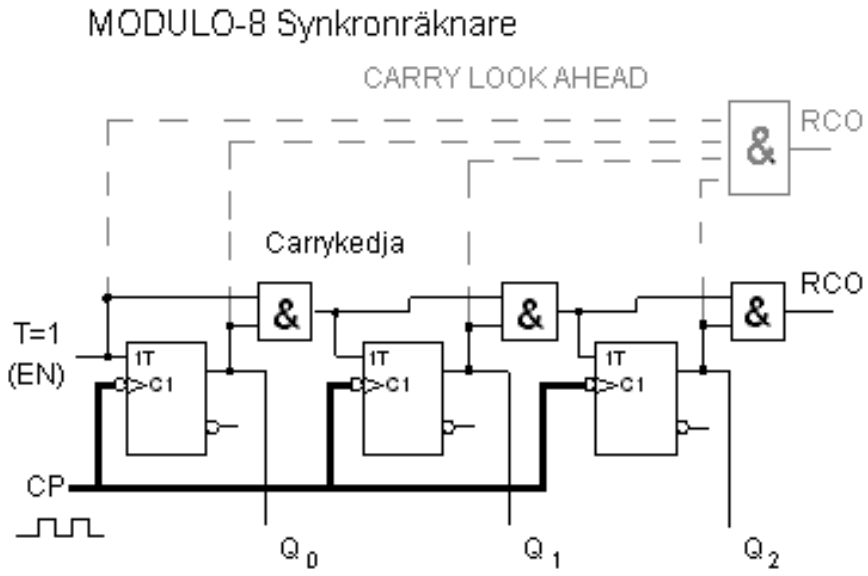
Hur man får 1 sekund får Du räkna ut själv …

Synkron räknare

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



Toggla om alla före dig är 1…

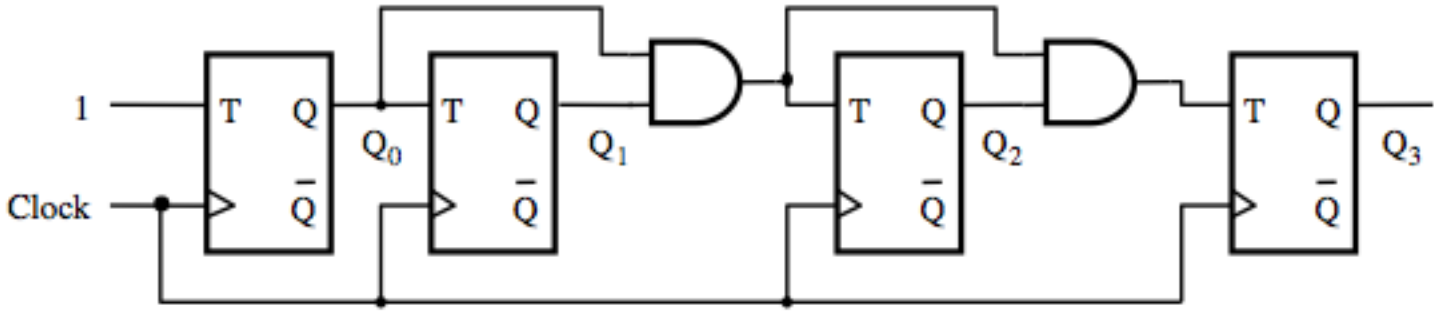


En snabbare räknare kan man få om man tar fram Carry parallellt (jfr. med adderaren).

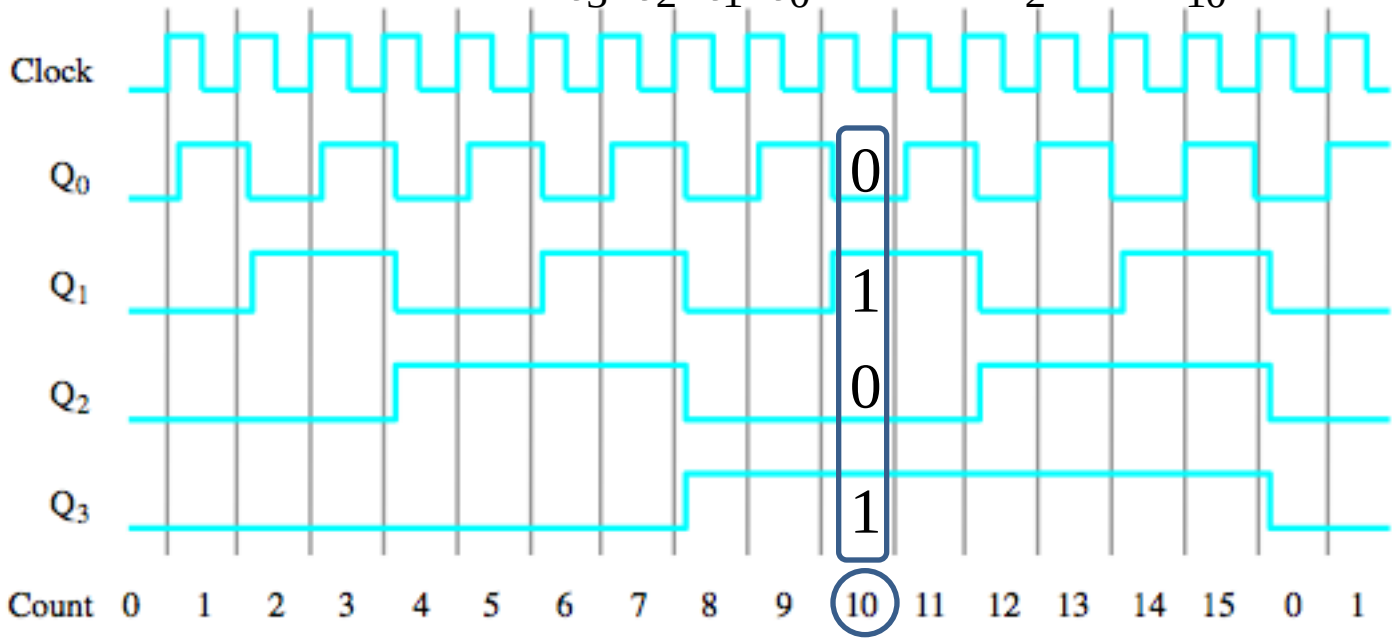
Vill man utöka räknaren sker det med en vippa och en AND-grind per steg.

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

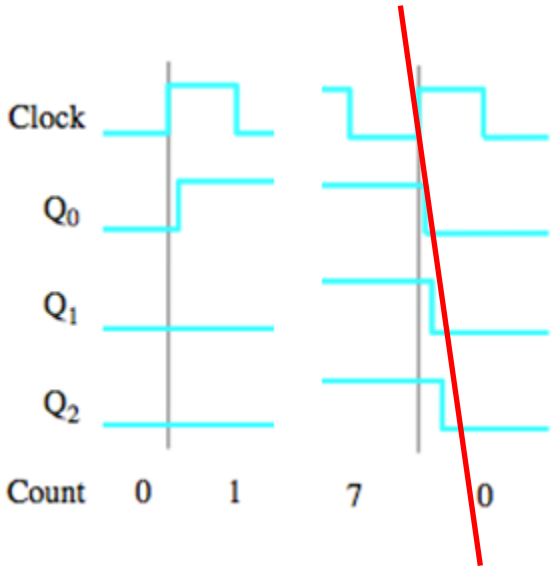


$$Q_3Q_2Q_1Q_0 = 1010_2 = 10_{10}$$



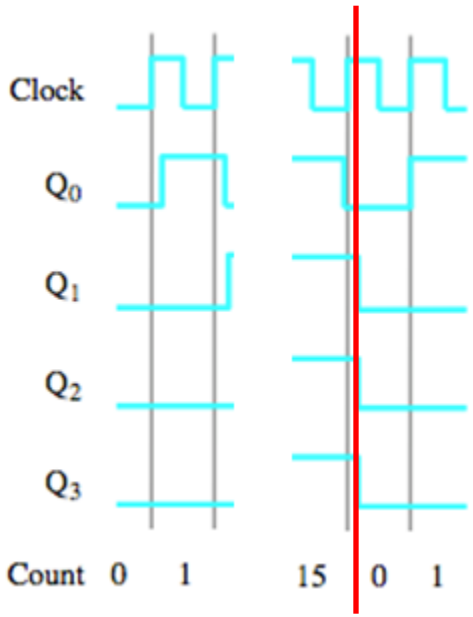
Asynkron eller Synkron räknare

Asynkron räknare



Utgångssignalerna fördröjs mer och mer för varje räknarsteg

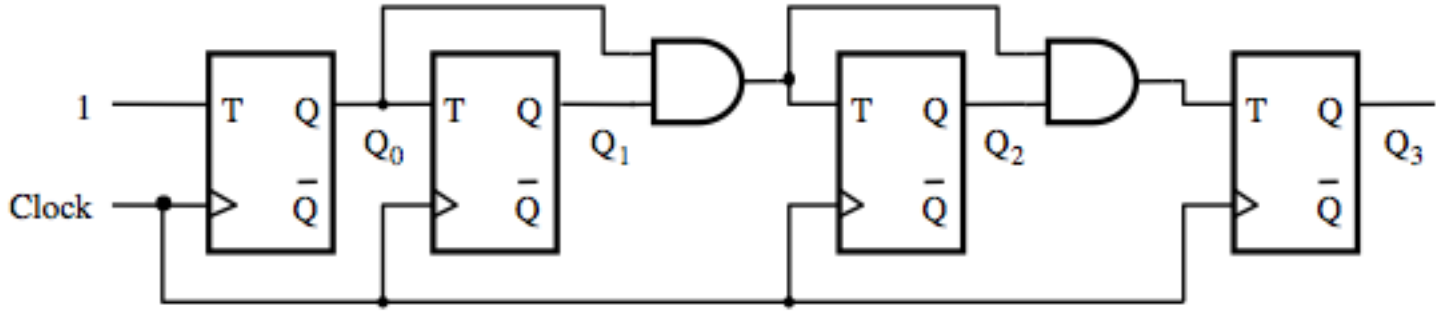
Synkron räknare



Utgångssignalerna har samma fördröjning

William Sandqvist william@kth.se

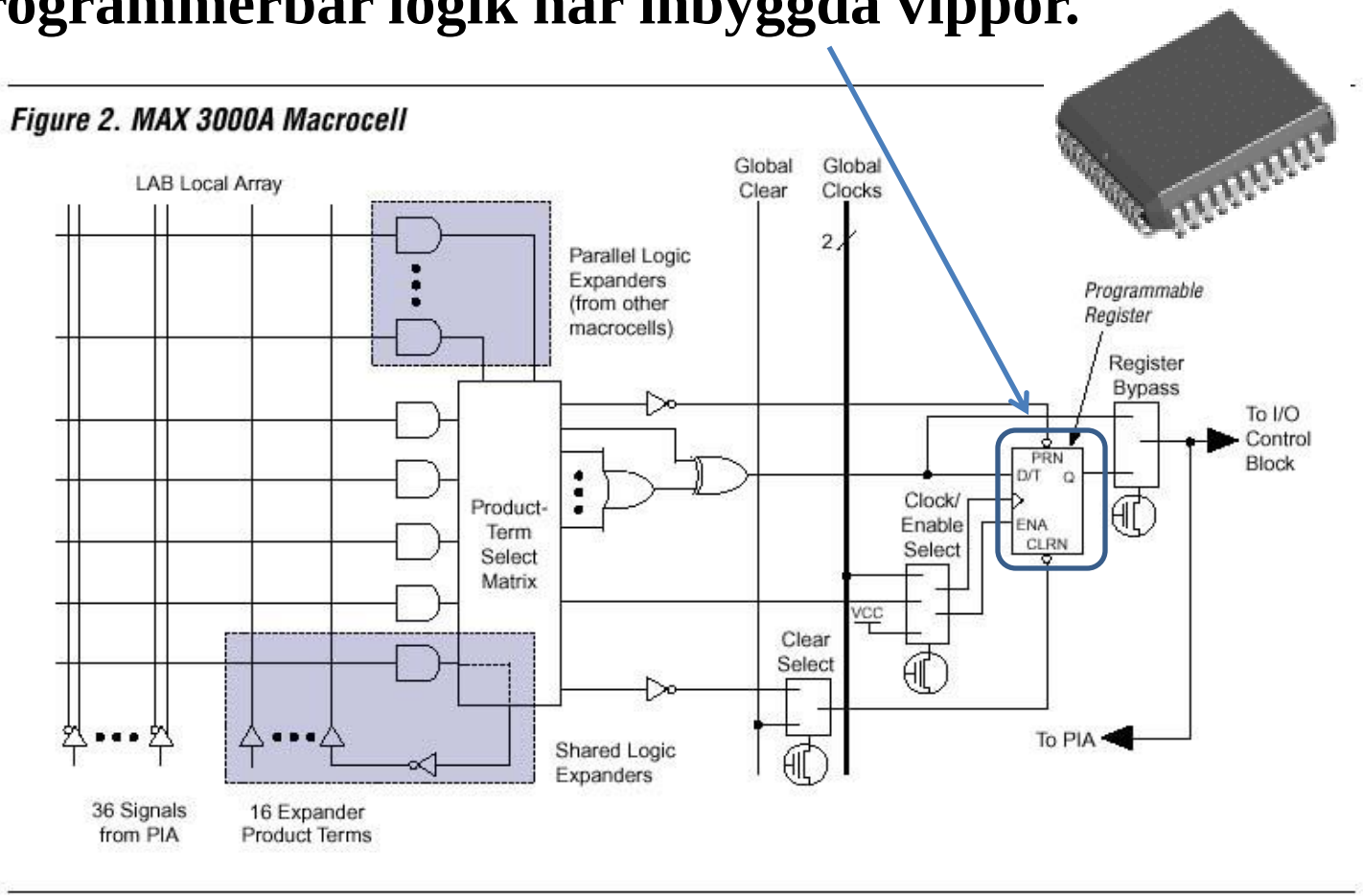
Vad är den maximala frekvensen?



- Den kritiska vägen bestämmer den maximala frekvensen!
- Här är den längsta kombinatoriska vägen från Q_0 via två AND-grindar till ingången av vippan som beräknar Q_3
  - t_{logic} motsvarar alltså fördröjningen av två NAND-grindar

VHDL för vippor och låskretsar

Programmerbar logik har inbyggda vippor.



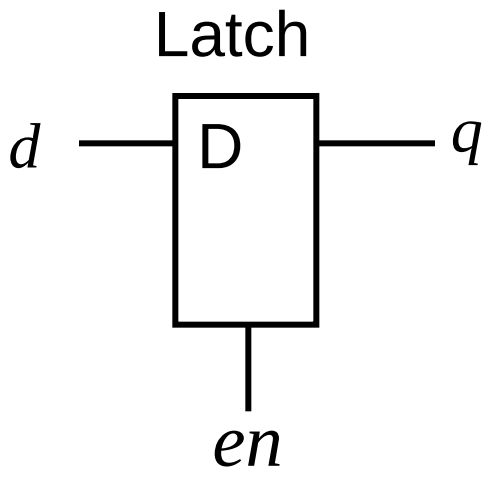
VHDL för vippor och låskretsar

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
           q <= d;
        END IF;
     END PROCESS;
END ARCHITECTURE RTL;
```



Enable	D	Q
0	-	M
1	D	D

Latch som process

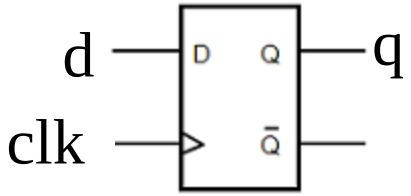
```
PROCESS(en, d)
    BEGIN
        IF en = '1' THEN
            q <= d;
        END IF;
END PROCESS;
```

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

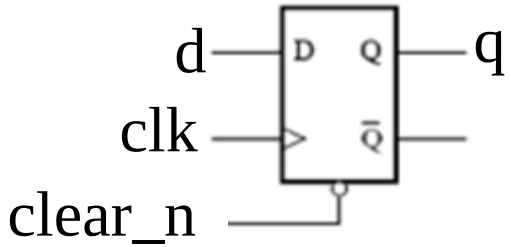
I stället för funktionen **`rising_edge(clk)`** kan man skriva **`clk'event and clk=1`**

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

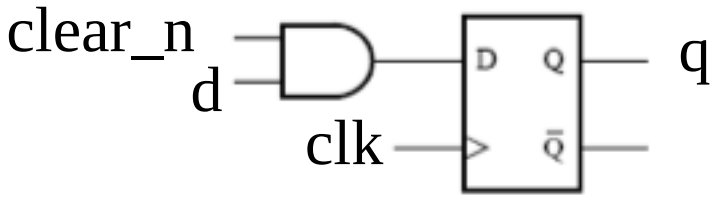
Med asynkron RESET

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 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;
```

Summering vippor

Basic latch (SR)

Gated latch

- Gated SR latch
- Gated D latch

Flip-Flop

- Edge triggered flip-flop
- Master slave flip-flop
- Med och utan reset

Andra vippor

- JK vippa
- T vippa