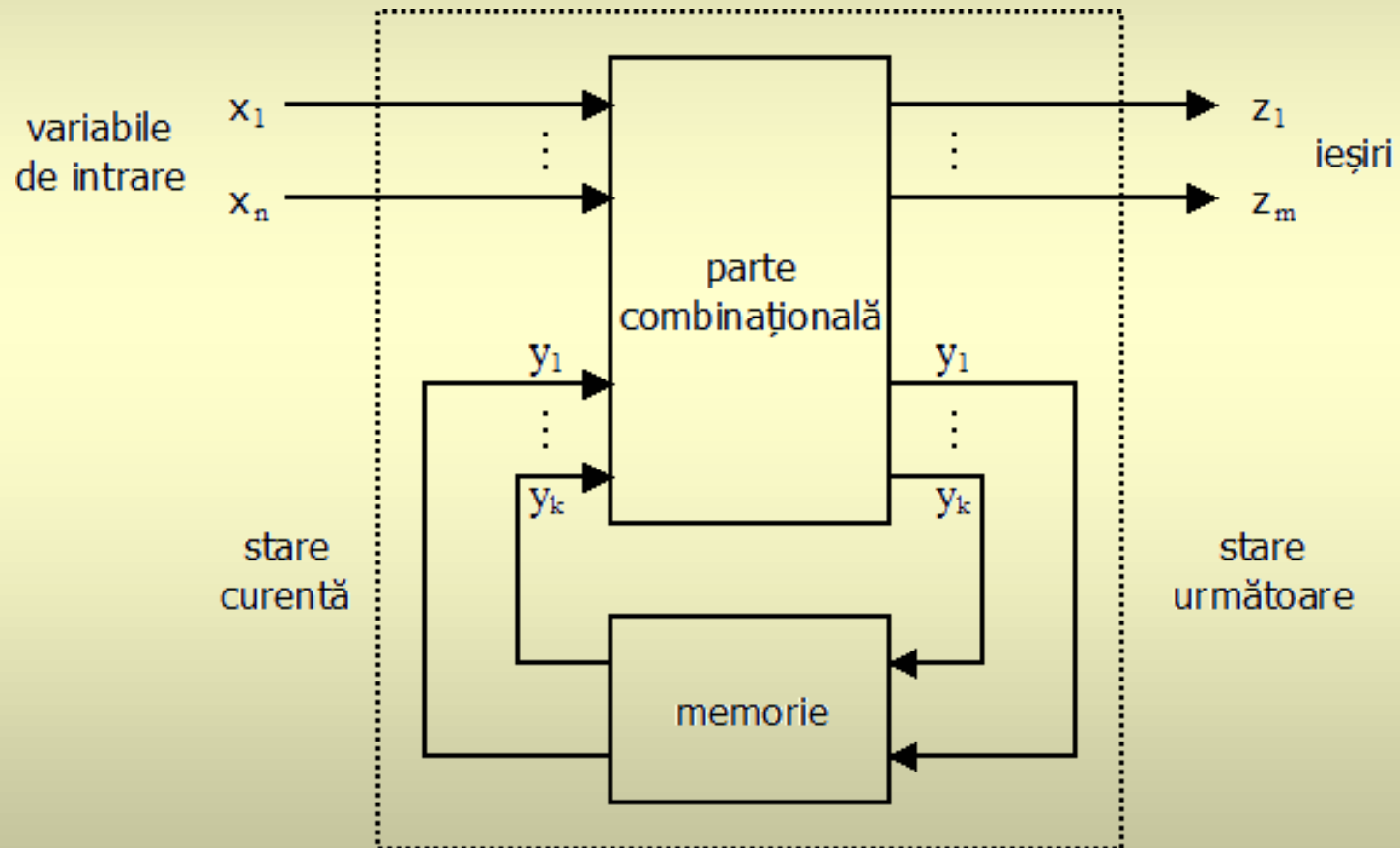


# III. Circuite secvențiale

# Circuit secvențial

- ieșirea la orice moment depinde de
  - intrare
  - starea internă
- deci pentru aceeași intrare se pot obține valori diferite la ieșire, la momente diferite
- starea internă
  - este memorată de către circuit
  - evoluează în timp

# Diagrama bloc



## Evoluția stării

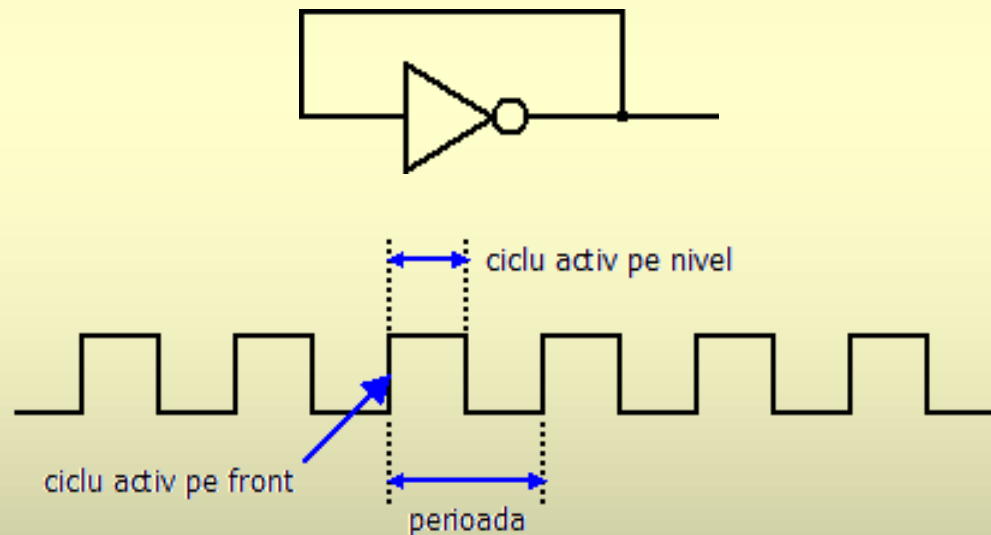
- starea se modifică la anumite momente
  - sincron: la intervale regulate de timp
    - date de un semnal special (ceas)
  - asincron: la momentul apariției unui eveniment
    - evenimentele sunt definite în funcție de activitatea circuitului
  - de ce nu se modifică permanent?
    - transmiterea semnalului prin porți și prin liniile de comunicare se face cu întârzieri
    - semnalele - luate în considerare după stabilizare

# Ceasul

- semnal periodic
  - ciclu activ - procentajul din perioadă în care semnalul este activ
  - depinde de ce înseamnă semnal activ
    - pe nivel (0 sau 1)
    - pe front (trecerea de la 0 la 1 sau invers)
- durata perioadei
  - suficient de mare pentru a avea intrări stabile

# Implementare

- cea mai simplă variantă
  - esențială este conexiunea inversă



- în general se folosesc scheme mai complexe

# Tipuri de circuite secvențiale

- la nivel de bit - circuite bistabile
- după cum este detectat semnalul activ
  - *latch* - activ pe nivel
  - *flip-flop* - activ pe front
- circuite pe mai mulți biți
  - regiștri, numărătoare (contoare)
  - formate din mai multe circuite bistabile

# III.1. Circuite bistabile

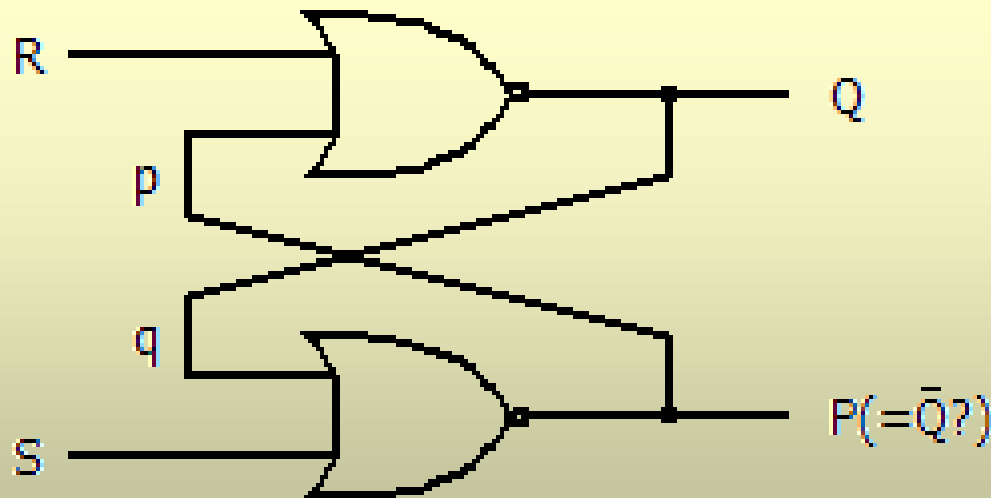


# Bistabil

- cum trebuie să arate un circuit care implementează bitul?
- specificații
  - să se poată scrie în el un 0 sau un 1
  - să memoreze acea valoare până la scrierea alteia
  - să se poată citi ultima valoare scrisă
- nu poate fi circuit combinațional (memorare)

## Bistabil RS fără ceas

- două intrări (R,S), două ieșiri (Q,P), două conexiuni inverse
  - circuitul implementează un singur bit:  $P = \bar{Q}$



# Funcționarea bistabilului RS (1)

- la prima vedere avem simultan

$$q = Q \text{ și } p = P$$

$$Q = \overline{p + R}$$

$$P = \overline{q + S}$$

- de fapt, ieșirile nu se modifică instantaneu la modificarea intrărilor
  - datorită timpilor de propagare prin porți
  - deci putem studia evoluția prin tabele de adevăr

# Funcționarea bistabilului RS (2)

- considerăm (q,p) valorile curente ale ieșirilor
- iar (Q,P) valorile viitoare
  - în funcție de (q,p) și de intrările (R,S)
  - care vor deveni efective după timpii de propagare

Diagrama Karnaugh

ieșirile: QP

qp\RS	00	01	11	10
00	11	10	00	01
01	01	00	00	01
11	00	00	00	00
10	10	10	00	00

$$Q = \overline{p + R}$$

$$P = \overline{q + S}$$

## Stări stabile

- în principiu,  $(Q,P)$  se modifică permanent
- dar atunci când  $(Q,P)=(q,p)$ , avem o stare stabilă
  - dorim identificarea acestor stări
  - circuitul poate fi controlat dacă trece doar prin stările stabile

qp\RS	00	01	11	10
00	11	10	<b>00</b>	01
01	<b>01</b>	00	00	<b>01</b>
11	00	00	00	00
10	<b>10</b>	<b>10</b>	00	00

# Funcționare (1)

stare inițială (q,p)	evoluție (q,p)	concluzie
$R=0, S=1$		
00	00 → 10 stabil	circuitul evoluează întotdeauna spre starea stabilă (Q,P)=(1,0)
01	01 → 00 → 10 stabil	
10	10 stabil	
11	11 → 00 → 10 stabil	
$R=1, S=0$		
00	00 → 01 stabil	circuitul evoluează întotdeauna spre starea stabilă (Q,P)=(0,1)
01	01 stabil	
10	10 → 00 → 01 stabil	
11	11 → 00 → 01 stabil	

# Funcționare (2)

stare inițială (q,p)	evoluție (q,p)	concluzie
$R=0, S=0$		
00	$00 \rightarrow 11 \rightarrow 00 \rightarrow \dots$	pentru $q=p$ , circuitul oscilează la infinit pentru $q \neq p$ , circuitul își păstrează starea (stabilă)
01	01 stabil	
10	10 stabil	
11	$11 \rightarrow 00 \rightarrow 11 \rightarrow \dots$	
$R=1, S=1$		
00	00 stabil	circuitul evoluează întotdeauna spre starea stabilă $(Q,P)=(0,0)$
01	$01 \rightarrow 00$ stabil	
10	$10 \rightarrow 00$ stabil	
11	$11 \rightarrow 00$ stabil	

## Funcționare (3)

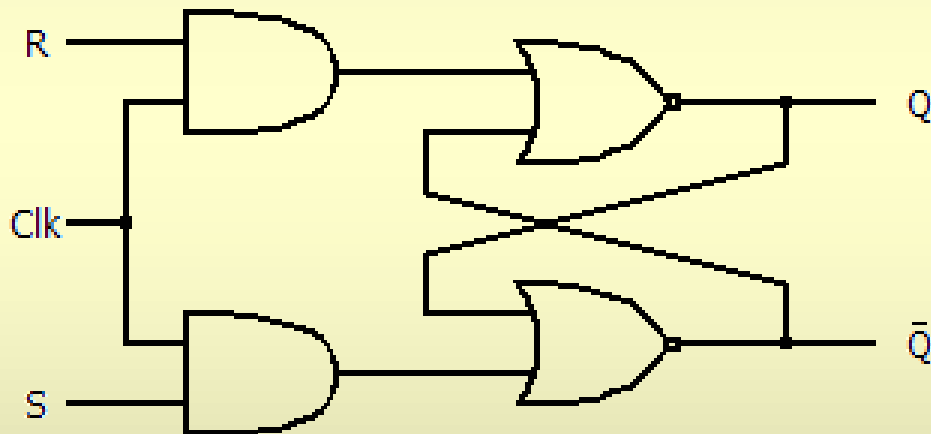
- reamintim condiția  $P = \bar{Q}$
- $(R,S)=(0,0)$ : păstrare stare existentă  
(memorare)
- $(R,S)=(0,1)$ : stabilizare la  $Q=1$  (set)
- $(R,S)=(1,0)$ : stabilizare la  $Q=0$  (reset)
- $(R,S)=(1,1)$ : combinație interzisă
- deoarece  $P=Q$  - nu implementează un bit



# Circuite secvențiale sincrone

- se adaugă bistabilului RS un semnal de sincronizare (ceas)
- pornind de la acesta se pot realiza alte circuite bistabile
  - D, JK, T
- toate sunt de tip latch (active pe nivel)

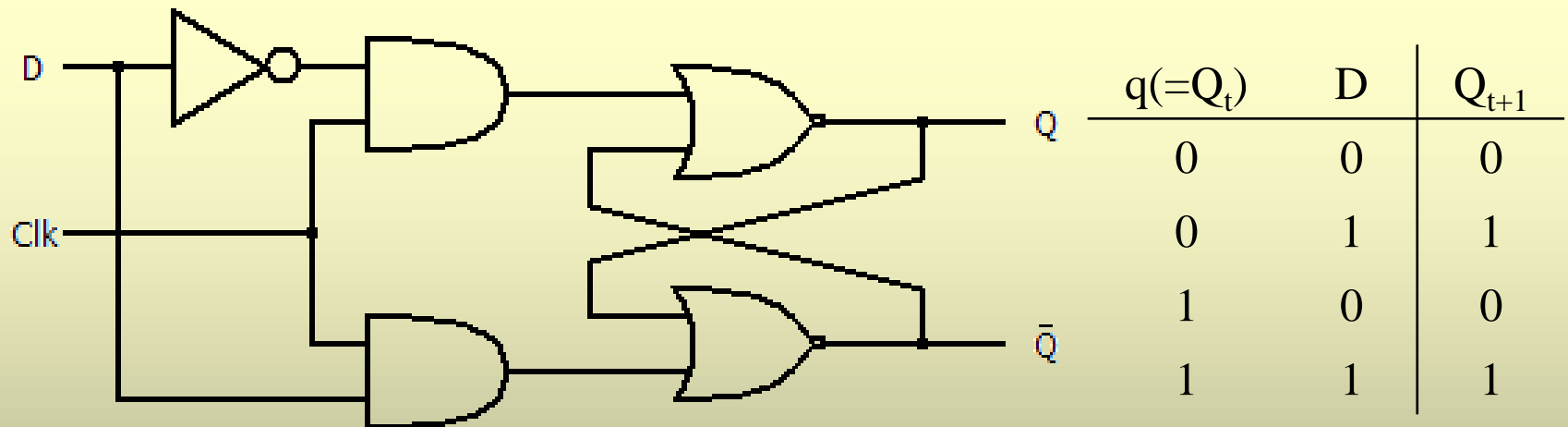
# Latch RS cu ceas



$q(=Q_t)$	R	S	$Q_{t+1}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	*
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	*

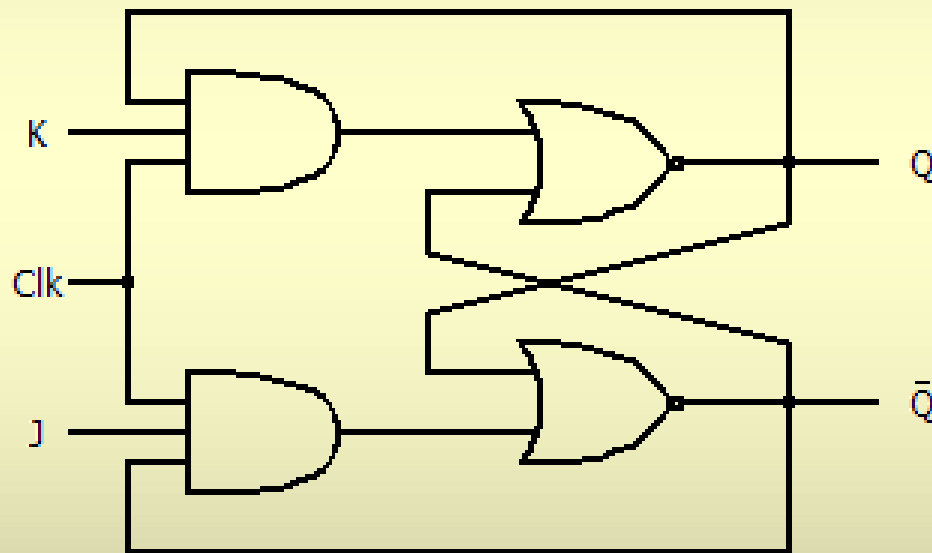
# Latch D

- modelează doar situațiile  $R \neq S$
- elimină combinațiile interzise
- aici ieșirea nu depinde de fapt de starea anterioară



# Latch JK

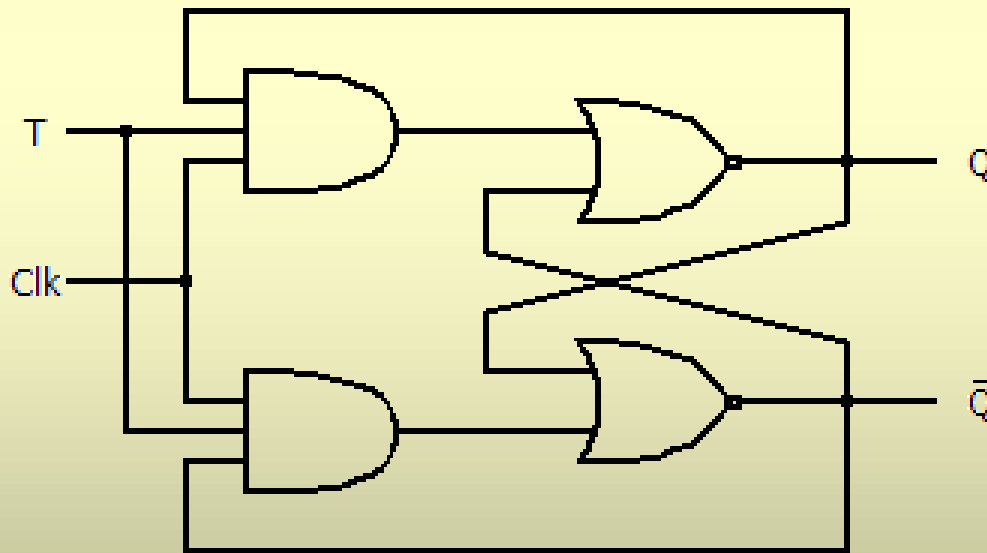
- elimină combinația imposibilă de la bistabilul RS



$q(=Q_t)$	J	K	$Q_{t+1}$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

# Latch T

- derivat din bistabilul JK
- modelează doar situațiile  $J=K$



$q(=Q_t)$	$T$	$Q_{t+1}$
0	0	0
0	1	1
1	0	1
1	1	0

# Evoluția stărilor

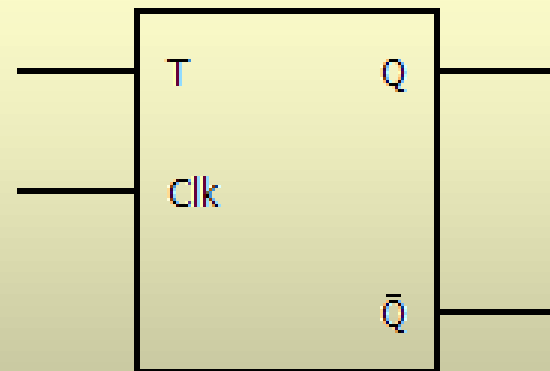
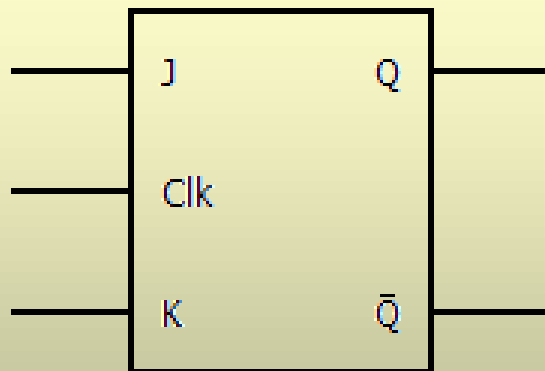
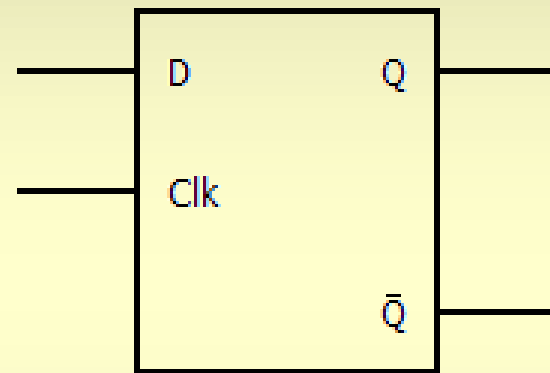
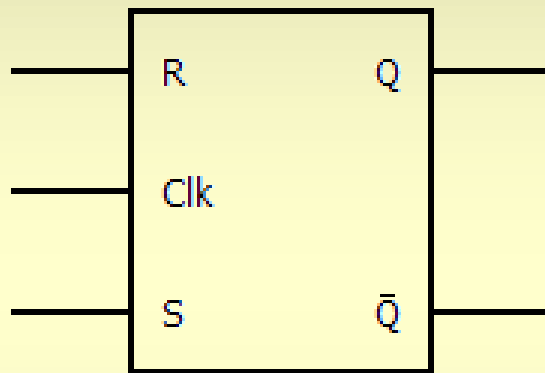
R	S	$Q_{t+1}$	
0	0	$Q_t$	neschimbat
0	1	1	scriere 1
1	0	0	scriere 0
1	1	*	interzis

J	K	$Q_{t+1}$	
0	0	$Q_t$	neschimbat
0	1	0	scriere 0
1	0	1	scriere 1
1	1	$\overline{Q_t}$	inversare

D	$Q_{t+1}$	
0	0	scriere 0
1	1	scriere 1

T	$Q_{t+1}$	
0	$Q_t$	neschimbat
1	$\overline{Q_t}$	inversare

# Diagrame bloc pentru bistabili



# Temă

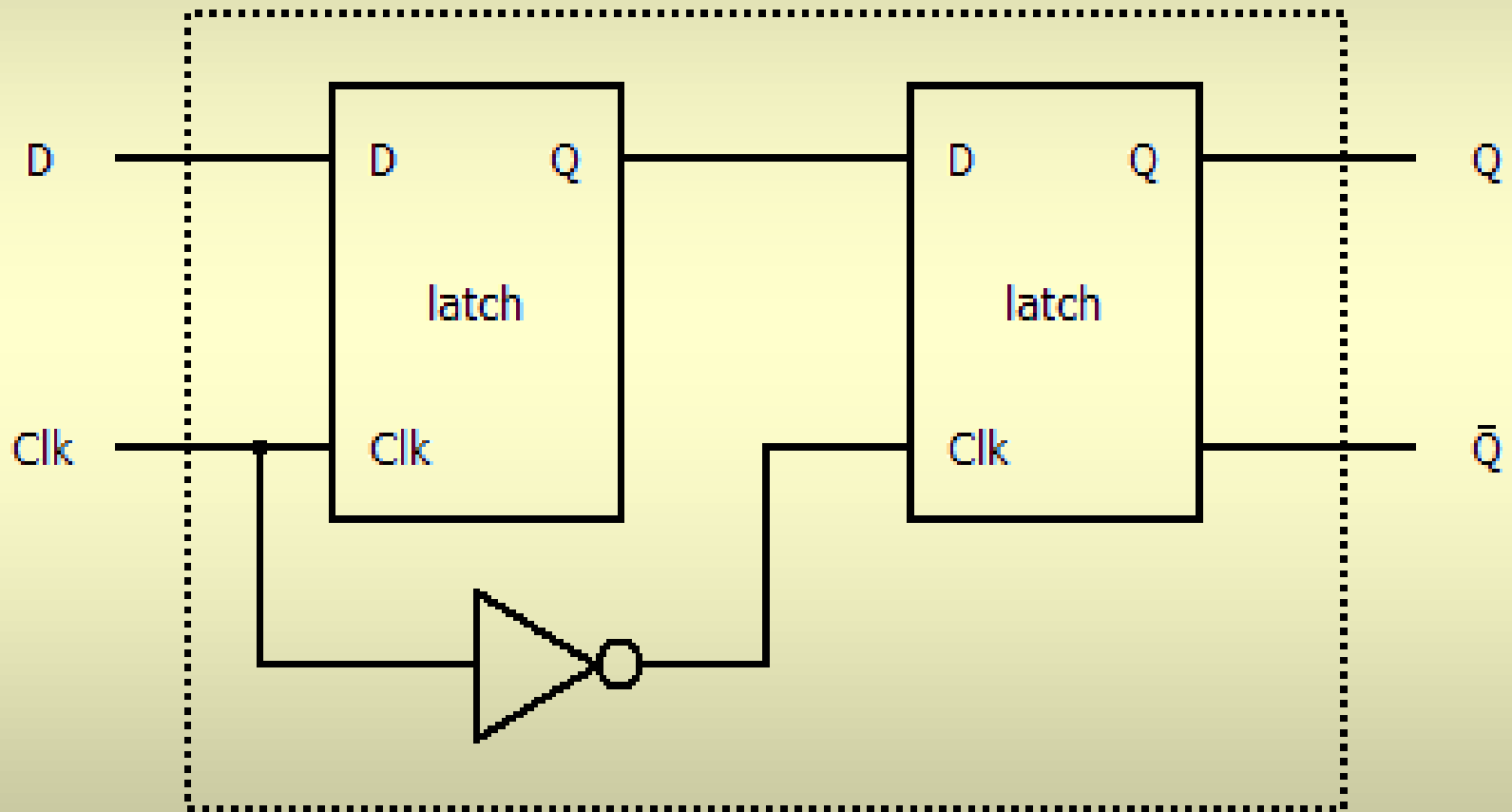
- implementați și analizați comportamentul bistabilului RS (fără ceas) utilizând porți NAND în locul porților NOR
- similar pentru bistabilii latch RS, D, JK, T



# Flip-flop

- intrările sunt luate în considerare doar pe frontul crescător (sau descrescător) al semnalului de ceas
- cum se poate obține un flip-flop
  - electronic - derivarea semnalului de ceas
  - utilizând circuite latch → circuite master-slave

# Flip-flop master-slave D



## Latch vs. flip-flop

- fiecare categorie are utilitatea sa
- circuitele flip-flop - utilizate pentru comanda sistemelor digitale
  - frontul semnalului de ceas este foarte scurt comparativ cu perioada ceasului
  - exact un pas în evoluția sistemului într-o perioadă de ceas
- circuitele latch - sisteme asincrone

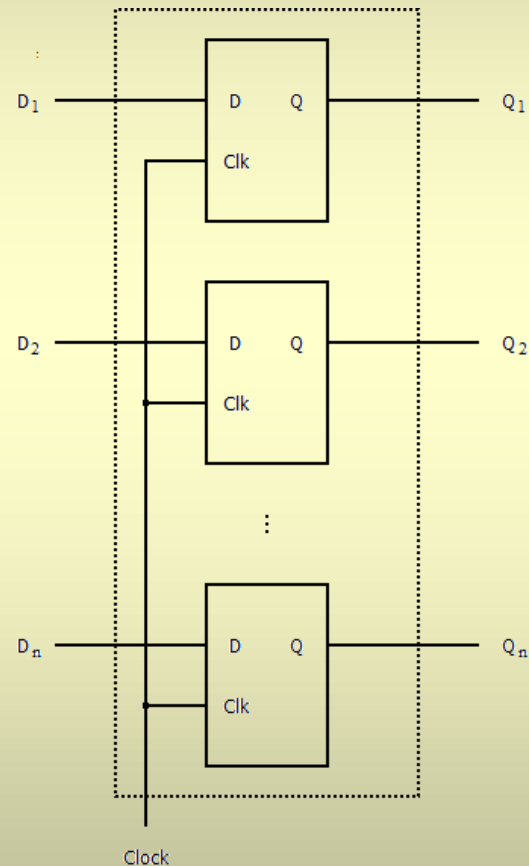
## III.2. Circuite secvențiale complexe

# Regiștri

- un circuit bistabil controlează un singur bit
  - nu foarte util în practică
- putem utiliza mai mulți bistabili simultan
  - toți primind aceeași comandă
  - un asemenea circuit se numește registru
- tipuri de regiștri
  - paraleli
  - cu deplasare (seriali)

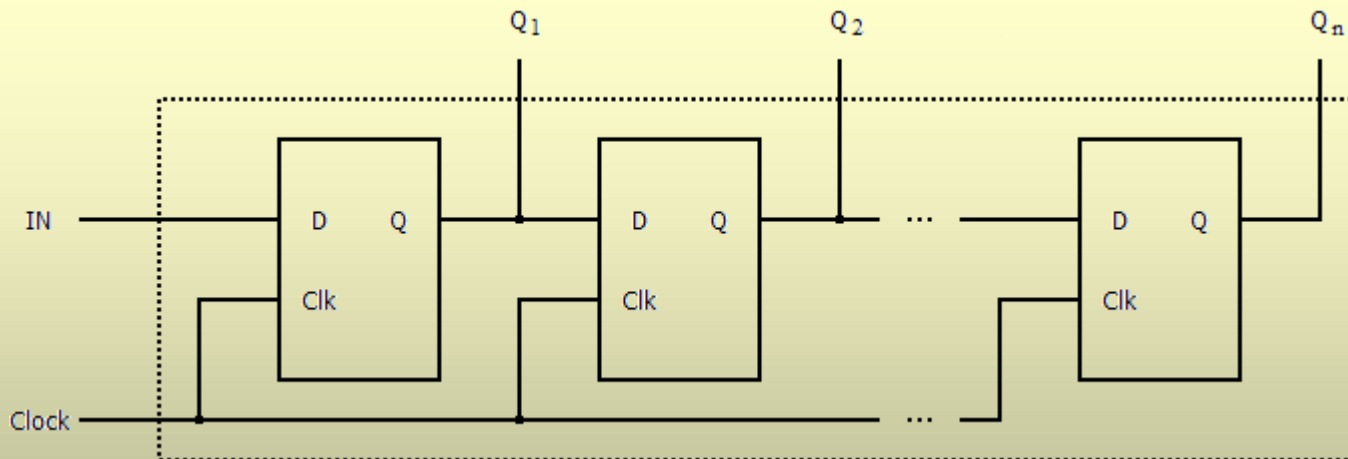
# Registru paralel

- implementare cu bistabili D
  - pot fi latch sau flip-flop, după necesități
- aceeași comandă (ceas)
  - toți bistabilii se modifică la aceleași momente
- extinderea bistabilului

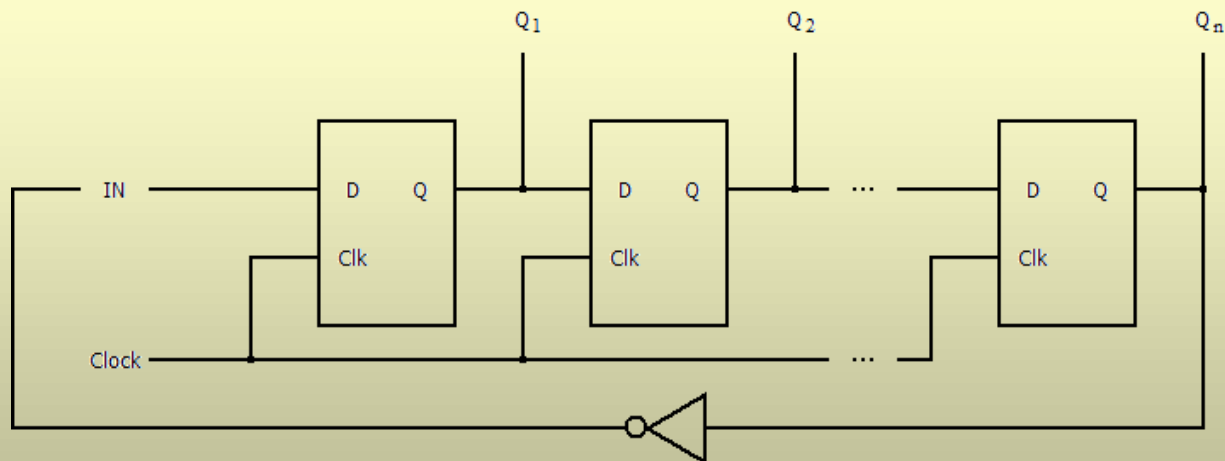
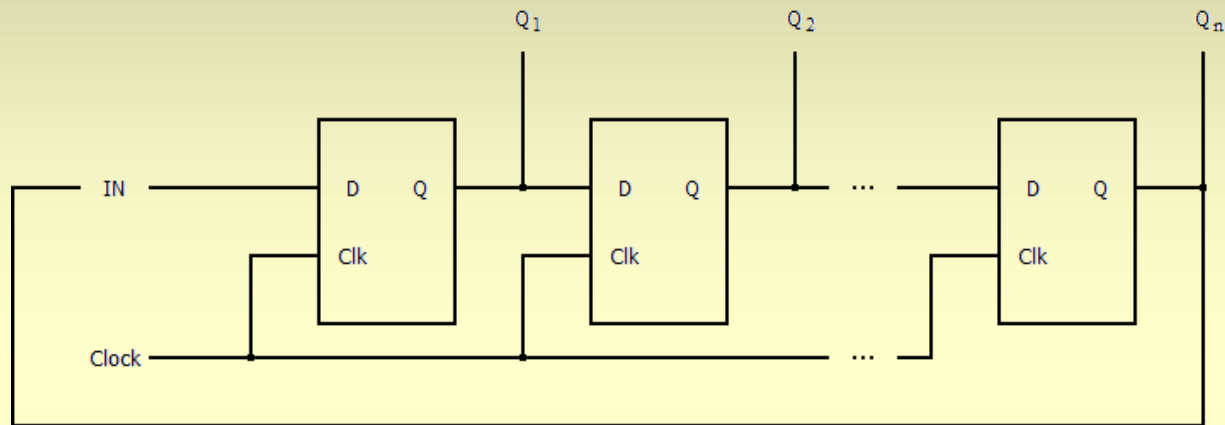


# Registru cu deplasare (clasic)

- memorează ultimele  $n$  valori de la intrare
- poate fi implementat doar cu flip-flop
  - temă: de ce?



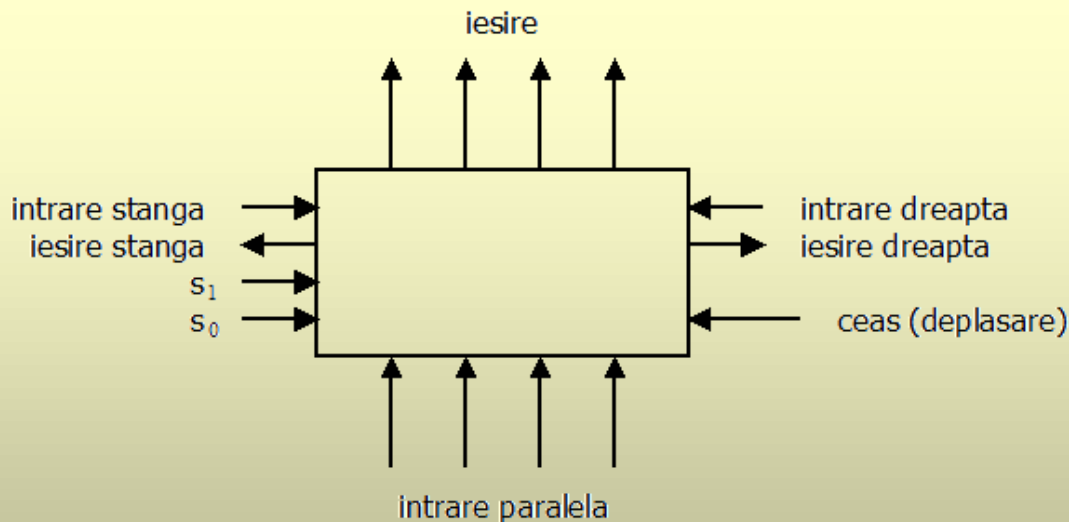
# Alți regiștri cu deplasare





# Registru universal

- intrări și ieșiri seriale sau paralele
- deplasare spre stânga sau spre dreapta
- pot fi folosite cele care sunt necesare la un moment dat



$s_0$	$s_1$	funcție
0	0	nemodificat
0	1	deplasare dreapta
1	0	deplasare stânga
1	1	încărcare paralelă

# Proiectarea unui circuit secvențial

- mașină cu număr finit de stări (automat)
  1. stabilirea stărilor prin care trece circuitul
  2. stabilirea tranzițiilor între stări
    - starea următoare și ieșirile în funcție de intrări și de starea curentă
  3. codificarea stărilor
    - pe numărul de biți necesar
  4. scrierea tabelului de adevăr pentru tranziții

# Proiectarea unui circuit secvențial

5. minimizare

6. implementare

- starea este memorată prin circuite bistabile
- partea combinațională - conform minimizării
  - intrările părții combinaționale (starea curentă) se preiau de la ieșirile circuitelor bistabile și de la variabilele de intrare
  - ieșirile părții combinaționale (starea următoare) se aplică la intrările circuitelor bistabile

# Numărătorul (contorul) binar

- reține la fiecare moment un număr pe  $n$  biți
- la fiecare "bătăie" a ceasului - incrementare
  - poate fi și decrementare
  - după valoarea maximă urmează din nou 0
  - nu are intrări, doar variabile de stare
    - care rețin de fapt numărul curent
  - ieșirile sunt identice cu variabilele de stare

# Exemplu: $n=4$

starea curentă				starea următoare				starea curentă				starea următoare			
$q_3$	$q_2$	$q_1$	$q_0$	$d_3$	$d_2$	$d_1$	$d_0$	$q_3$	$q_2$	$q_1$	$q_0$	$d_3$	$d_2$	$d_1$	$d_0$
0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	1
0	0	0	1	0	0	1	0	1	0	0	1	1	0	1	0
0	0	1	0	0	0	1	1	1	0	1	0	1	0	1	1
0	0	1	1	0	1	0	0	1	0	1	1	1	1	0	0
0	1	0	0	0	1	0	1	1	1	0	0	1	1	0	1
0	1	0	1	0	1	1	0	1	1	0	1	1	1	1	0
0	1	1	0	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0

## Exemplu: $n=4$

- prin minimizare se obțin ecuațiile

$$d_0 = \bar{q}_0 = q_0 \oplus 1$$

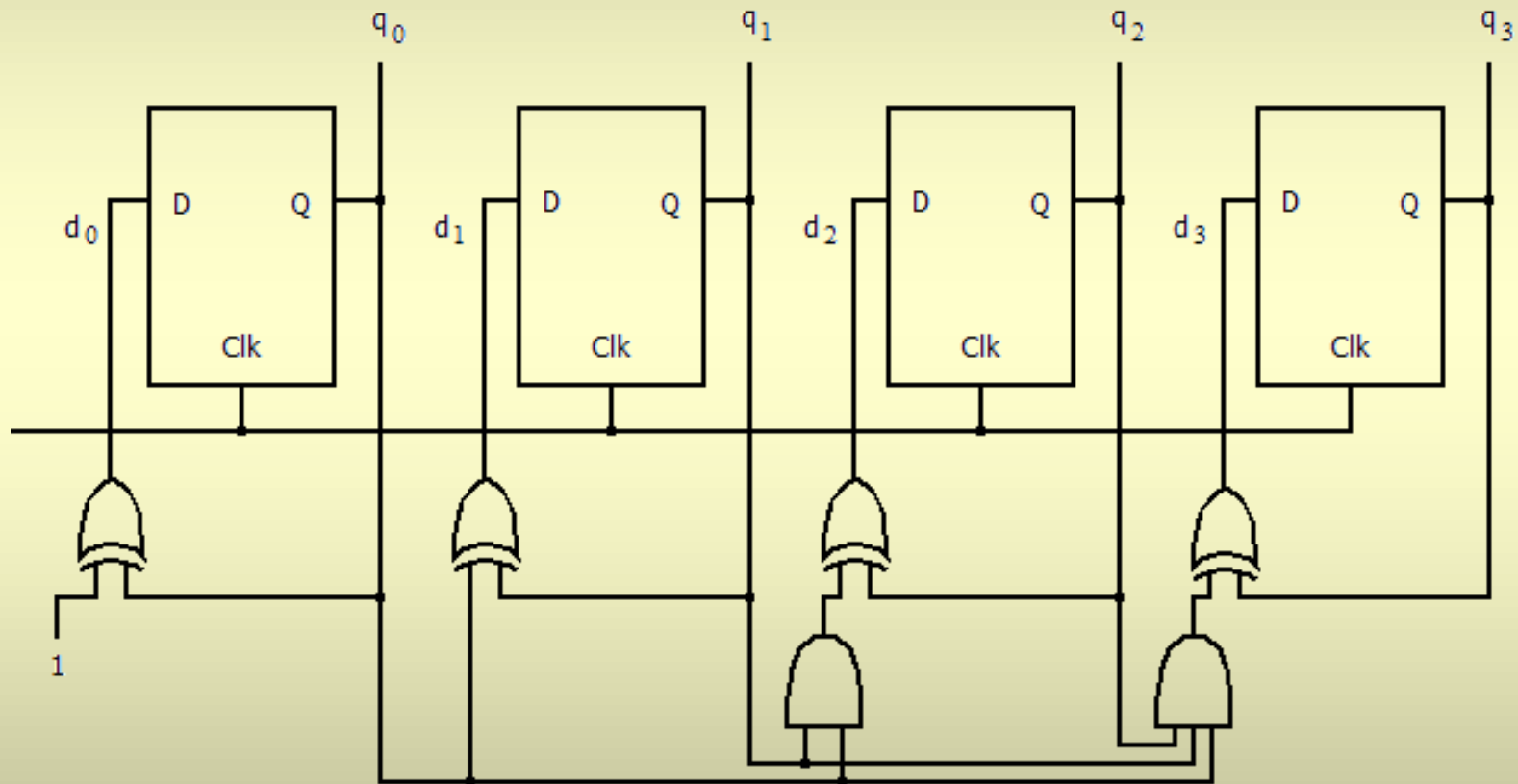
$$d_1 = \bar{q}_1 \cdot q_0 + q_1 \cdot \bar{q}_0 = q_1 \oplus q_0$$

$$d_2 = \bar{q}_2 \cdot q_1 \cdot q_0 + q_2 \cdot \bar{q}_1 + q_2 \cdot \bar{q}_0 = q_2 \oplus (q_1 \cdot q_0)$$

$$\begin{aligned} d_3 &= \bar{q}_3 \cdot q_2 \cdot q_1 \cdot q_0 + q_3 \cdot \bar{q}_2 + q_3 \cdot \bar{q}_1 + q_3 \cdot \bar{q}_0 = \\ &= q_3 \oplus (q_2 \cdot q_1 \cdot q_0) \end{aligned}$$

- implementarea stării - cu flip-flop-uri D

# Implementare



# Microprogramare (1)

- formă alternativă de implementare
  - starea este memorată tot de circuite bistabile
  - partea combinațională - cu ajutorul unei memorii ROM
  - intrările funcțiilor booleene se aplică la intrările de adresă ale circuitului
  - iar ieșirile funcțiilor booleene se colectează de la ieșirile de date



## Microprogramare (2)

- implementarea părții combinaționale
  - se pornește tot de la tabelul de adevăr
  - în fiecare locație se scriu valorile dorite pentru ieșire
- avantaj - flexibilitate
  - orice modificare a automatului implică doar rescrierea conținutului memoriei ROM
- dezavantaj - viteză redusă
  - memoria ROM este mai lentă decât porțile

## Același exemplu

- avem 16 ( $= 2^4$ ) stări
  - codificate pe 4 biți de stare
- deci circuitul ROM va avea
  - $2^4$  adrese  $\rightarrow$  4 biți de adresă
    - 16 locații
  - 4 biți de date  $\rightarrow$  locații de 4 biți
  - în acest exemplu nu avem variabile de intrare și ieșiri ale sistemului care să se adauge la biții de stare

# Conținutul memoriei ROM

adresă	valoare
0	0 0 0 1
1	0 0 1 0
2	0 0 1 1
3	0 1 0 0
4	0 1 0 1
5	0 1 1 0
6	0 1 1 1
7	1 0 0 0

adresă	valoare
8	1 0 0 1
9	1 0 1 0
10	1 0 1 1
11	1 1 0 0
12	1 1 0 1
13	1 1 1 0
14	1 1 1 1
15	0 0 0 0

# Implementare

