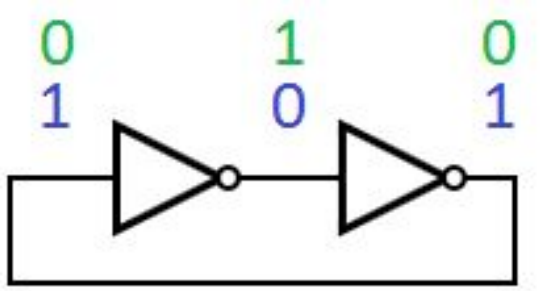
Univerzitet u Novom Sadu
Fakultet tehničkih nauka
Odsek za računarsku tehniku i računarske komunikacije

Osnovi logičkog projektovanja sekvencijalnih mreža

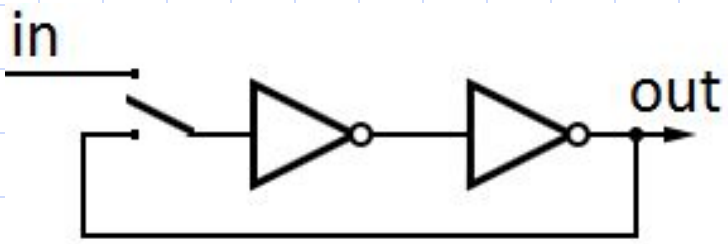
Automati opšteg tipa

Kako memorisati 1 bit informacije?

- Koncepcijski dva invertora povezana u krug pamte stanje



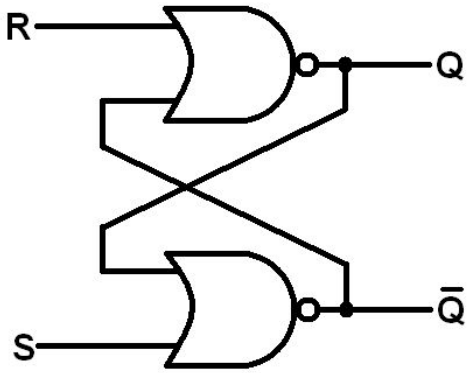
- Ako bi im se dodao idealan preklopnik, stanje bi se moglo menjati



- Preklopnik
  - gore: upiši
  - dole: pamti

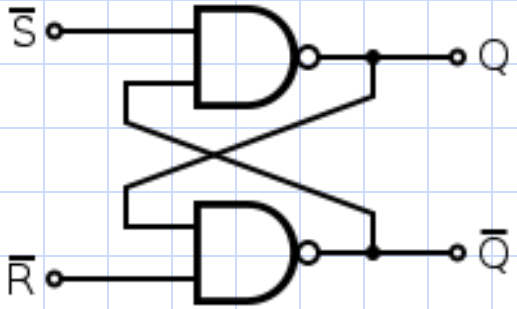
SR leč (latch)

NILI



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

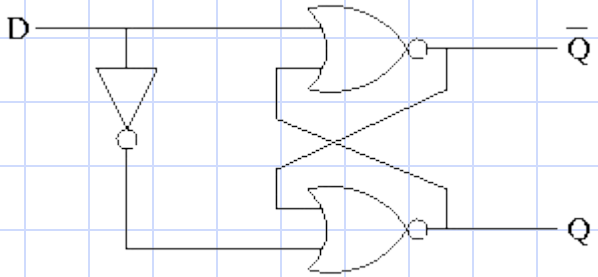
NI



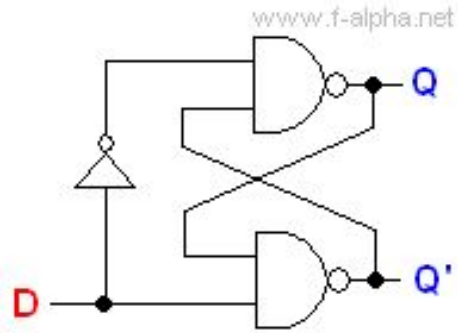
S	R	Q	$\bar{Q}$
0	0	Q-	$\bar{Q}$-
0	1	0	1
1	0	1	0
1	1	1	1

D leč (latch)

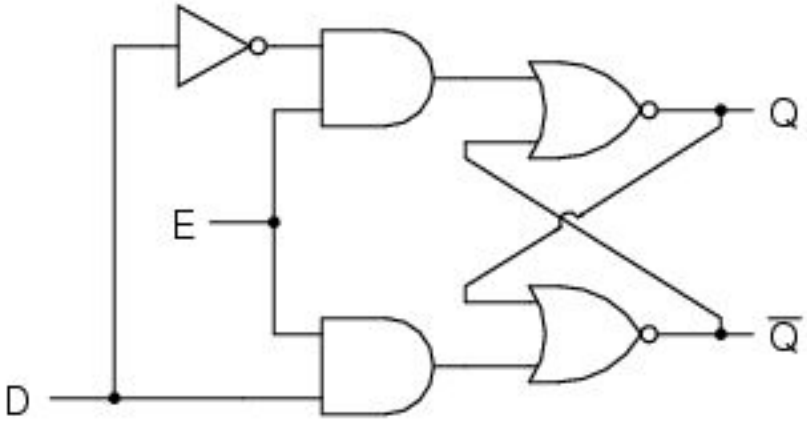
NILI



NI

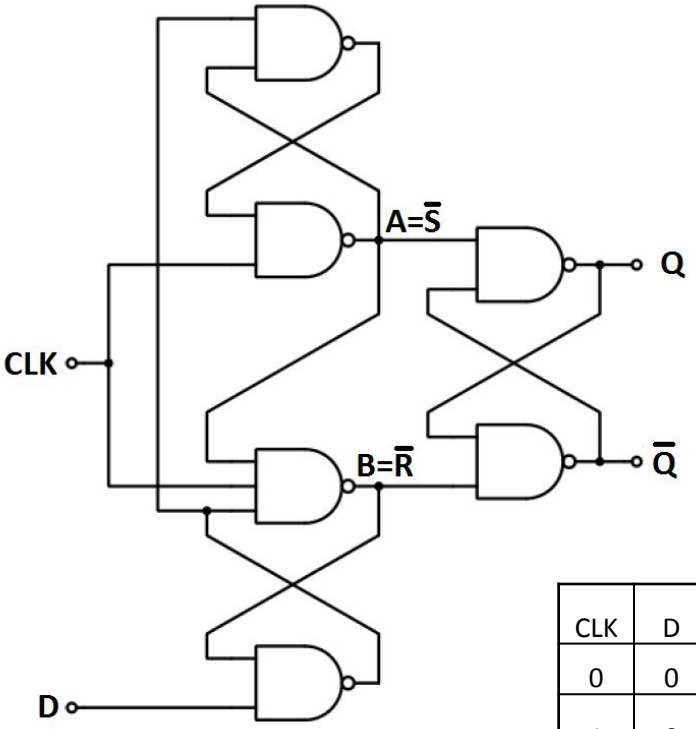


Gejtovani D (latch)
Leč sa dozvolom upisa



E	D	Q	$\overline{Q}$
0	**0**	latch	latch
0	1	latch	latch
1	**0**	**0**	1
1	1	1	**0**

Taktovani D flip-flop sa NI kolima



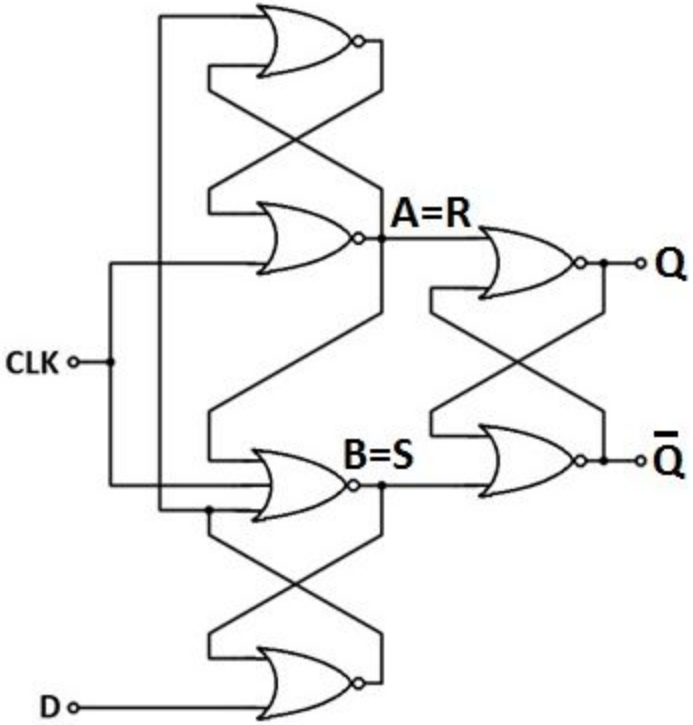
Samo pri tranziciji CLK (0-1) može doći do promene A i B!

Ako je CLK=0 (A,B)=(1,1) izlazni leč pamti.

Ako je CLK=1 promena na D ne afektira (A,B)

CLK	D	A	B	S	R	Q	
0	0	1	1	0	0	Q-	donji leč u zabr stanju
1	0	1	0	0	1	0	
0	1	1	1	0	0	Q-	gornji leč u zabr stanju
1	1	0	1	1	0	1	

Taktovani D flip-flop sa NILI kolima



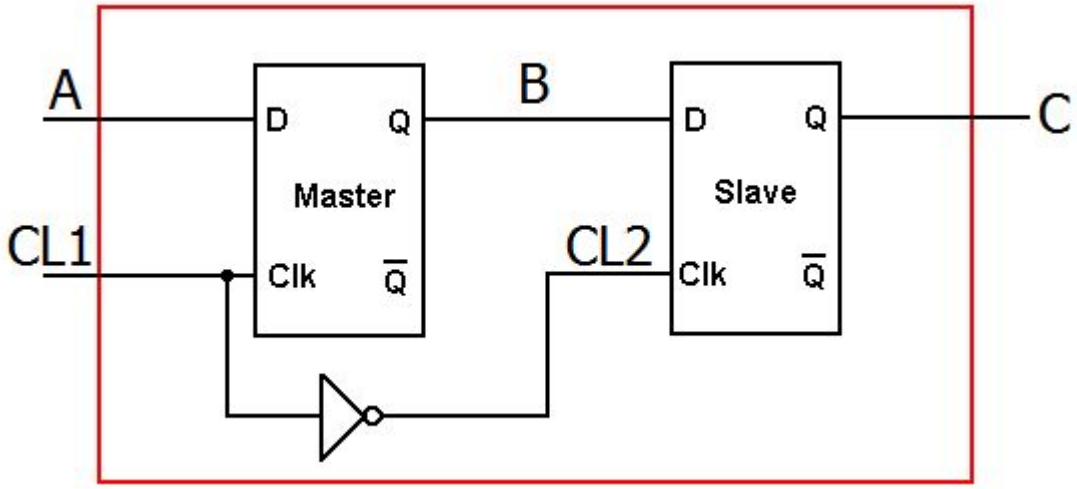
Samo pri tranziciji CLK (1-0) može doći do promene A i B!

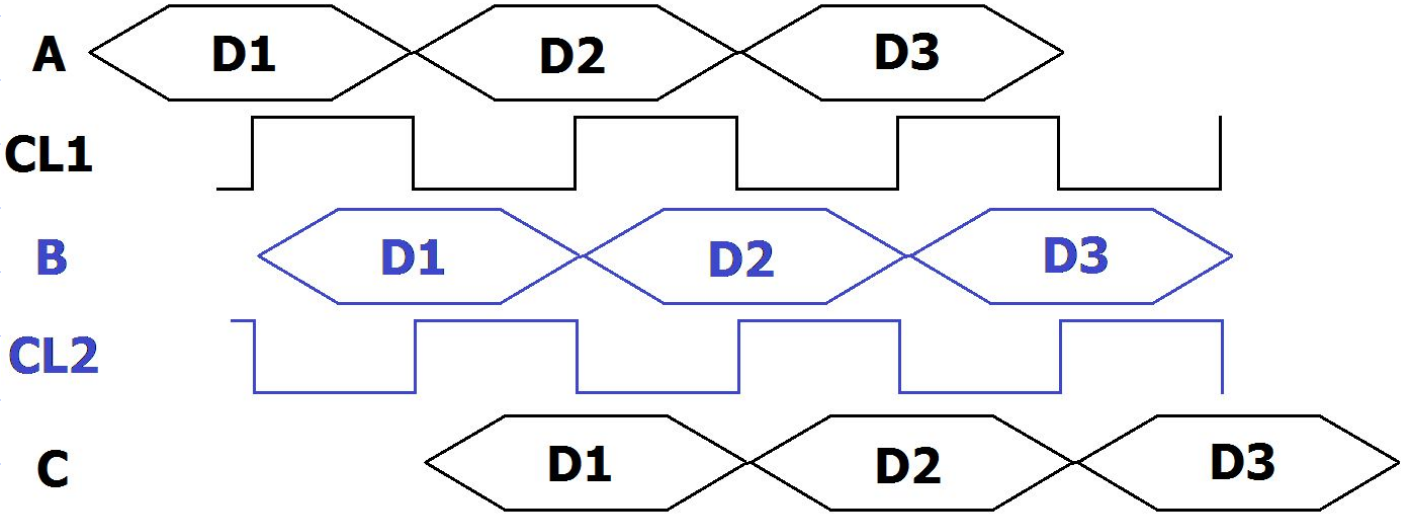
Ako je CLK=1 (A,B)=(0,0) izlazni leč pamti.

Ako je CLK=0 promena na D ne afektira (A,B)

CLK	D	A	B	S	R	Q	
1	0	0	0	0	0	Q-	gornji leč u zabr stanju
0	0	1	0	0	1	0	reset
1	1	0	0	0	0	Q-	donji leč u zabr stanju
0	1	0	1	1	0	1	set

Memorijski element tipa vodeći-prateći (engl. *Master-Slave*)





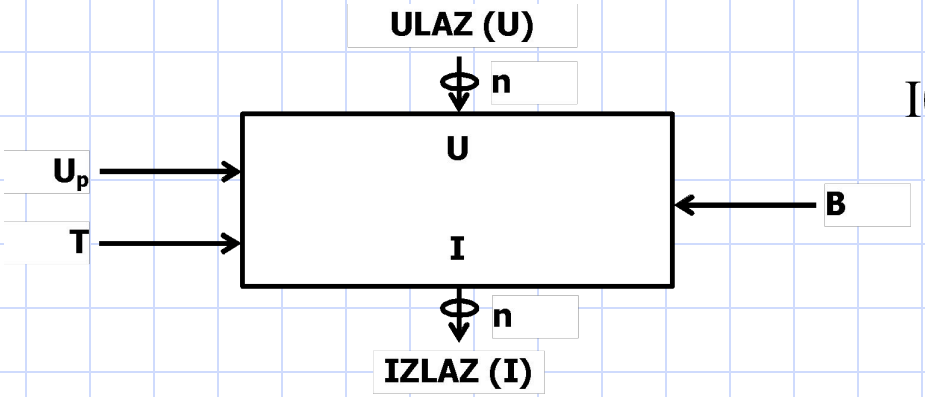
Univerzitet u Novom Sadu
Fakultet tehničkih nauka
Odsek za računarsku tehniku i računarske komunikacije

Standardne sekvencijalne mreže

Registri i brojači

Registri

- Registri predstavljaju skup elementarnih automata (flip-flopova) i kombinacionih mreža koje omogućuju pamćenje reči i u opštem slučaju omogućuju izvršavanje sledećih elementarnih operacija:
  - postavljanje registara na nulu;
  - prijem reči iz drugog registra, kombinacione mreže ili brojača;
  - prenos reči u drugi registar, kombinacionu mrežu ili brojač;
  - pretvaranje direktnog koda u komplementarni kod i obrnuto;
  - pomeraj reči u levo i desno za dati broj razreda;
  - pretvaranje tipa serijsko/paralelno i paralelno/serijsko.



$$I(t+1)=\begin{cases} U(t) & \text{za } U_p = 1 \\ I(t) & \text{u protivnom} \end{cases}$$

Sinhrono vs. asinhrono

reset **asinhron** u odnosu na takt signal

reset **sinhron** u odnosu na takt signal

reset = inicijalizacija stanja flip-flopa

Registar sa sinhronim postavljanjem početnog stanja

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY REG IS
  GENERIC (
    -- pretpostavljeni broj bita je 4
    pWIDTH: integer := 4
  );
  PORT (
    iCLK,
    iCLR : IN  STD_LOGIC;
    iCE  : IN  STD_LOGIC;
    iD   : IN  STD_LOGIC_VECTOR
                (pWIDTH-1 DOWNTO 0);
    oQ   : OUT STD_LOGIC_VECTOR
                (pWIDTH-1 DOWNTO 0)
  );
END REG;
```

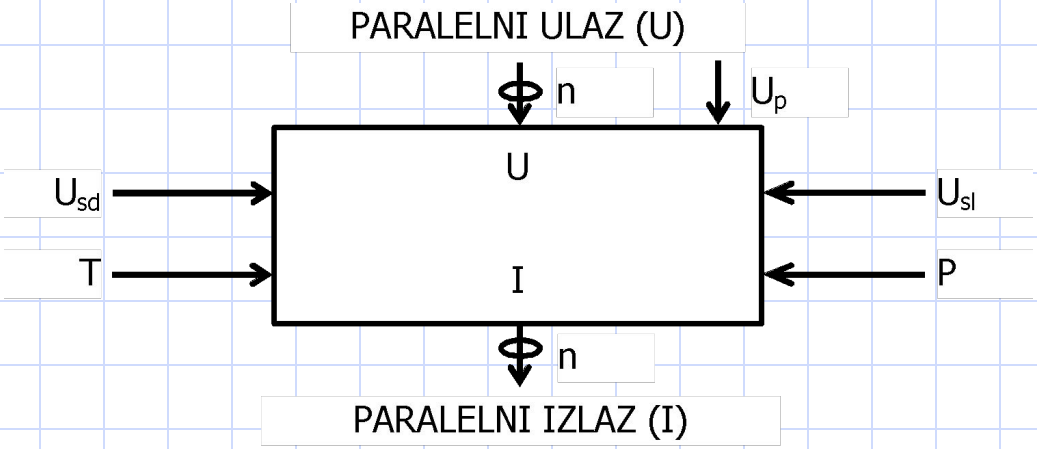
```
ARCHITECTURE ARH_REG OF REG IS
  -- stanje registra
  SIGNAL sREG : STD_LOGIC_VECTOR
                (pWIDTH-1 DOWNTO 0);
BEGIN

  PROCESS (iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      IF (iCLR = '1') THEN
        -- sinhrono postavljanje
        -- pocetne vrednosti
        sREG <= (OTHERS => '0');
      ELSE
        IF (iCE = '1') THEN
          -- upis u registar
          sREG <= iD;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- preslikavanje stanja registra
  -- na izlazni vektor
  oQ <= sREG;

END ARH_REG;
```

Registar sa asinhronim postavljanjem početnog stanja

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY REG IS
  GENERIC (
    -- pretpostavljeni broj bita je 4
    pWIDTH: integer := 4
  );
  PORT (
    iCLK,
    iCLR : IN  STD_LOGIC;
    iCE  : IN  STD_LOGIC;
    iD   : IN  STD_LOGIC_VECTOR
                (pWIDTH-1 DOWNTO 0);
    oQ   : OUT STD_LOGIC_VECTOR
                (pWIDTH-1 DOWNTO 0)
  );
END REG;
```

```
ARCHITECTURE ARH_REG OF REG IS
  -- stanje registra
  SIGNAL sREG : STD_LOGIC_VECTOR
                (pWIDTH-1 DOWNTO 0);
BEGIN

  PROCESS (iCLK, iCLR) BEGIN
    IF (iCLR = '1') THEN
      -- asinhrono postavljanje
      -- pocetne vrednosti
      sREG <= (OTHERS => '0');
    ELSIF (iCLK'EVENT AND iCLK = '1') THEN
      IF (iCE = '1') THEN
        -- upis u registar
        sREG <= iD;
      END IF;
    END IF;
  END PROCESS;

  -- preslikavanje stanja registra
  -- na izlazni vektor
  oQ <= sREG;

END ARH_REG;
```

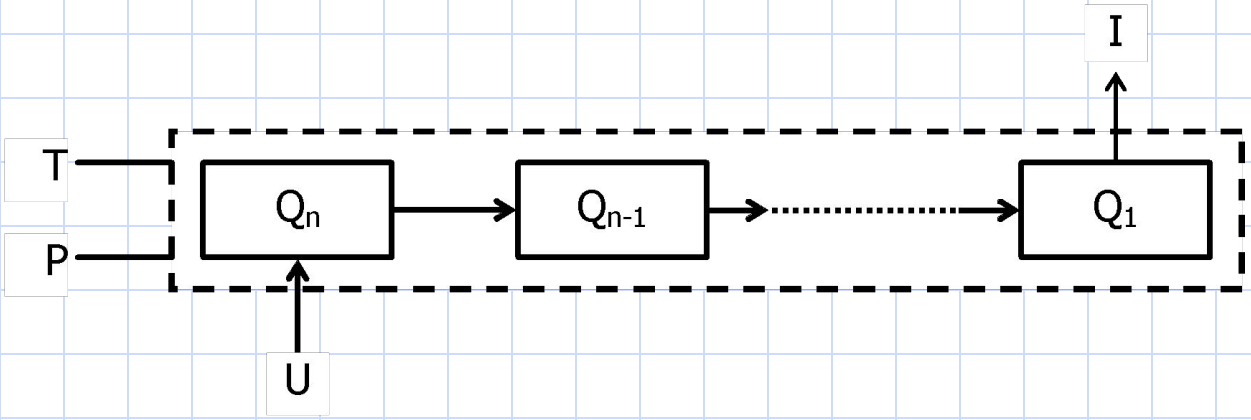
Pomerački registri



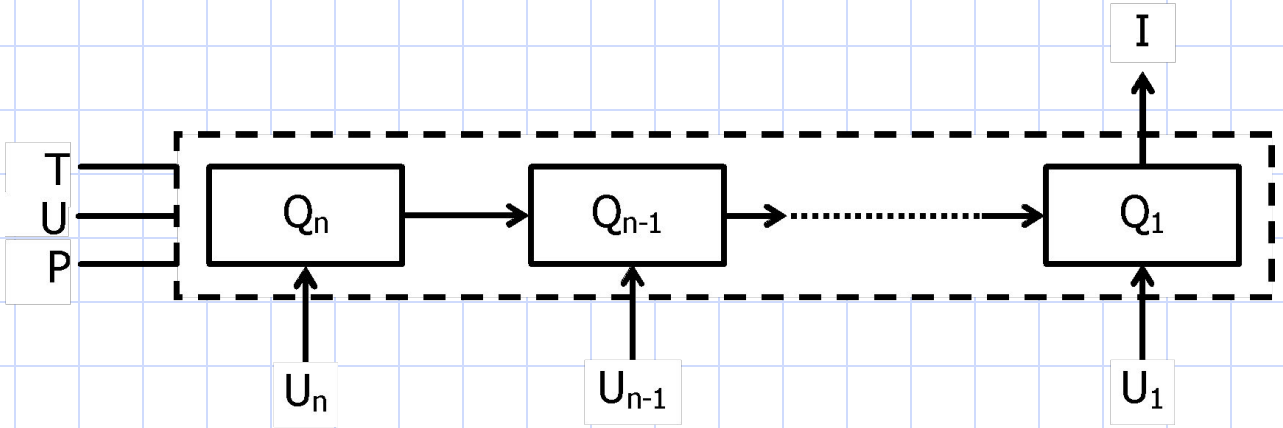
$$I(t+1)=\begin{cases} I(t) & za\ p=0 \\ U(t) & za\ p=U_p \\ (I_{n-1}, I_{n-2}, \ldots, I_1, U_{sl}) & za\ p=pomeranje\ u\ levo \\ (U_{sd}, I_n, \ldots, I_3, I_2) & za\ p=pomeranje\ u\ desno \end{cases}$$

Primeri pomeračkih registara

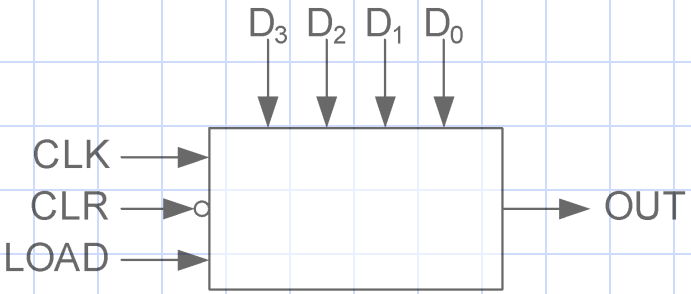
Primer 1 Pomerački registar sa serijskim ulazom/izlazom



Primer 2 Pomerački registar sa paralelnim ulazom i serijskim izlazom

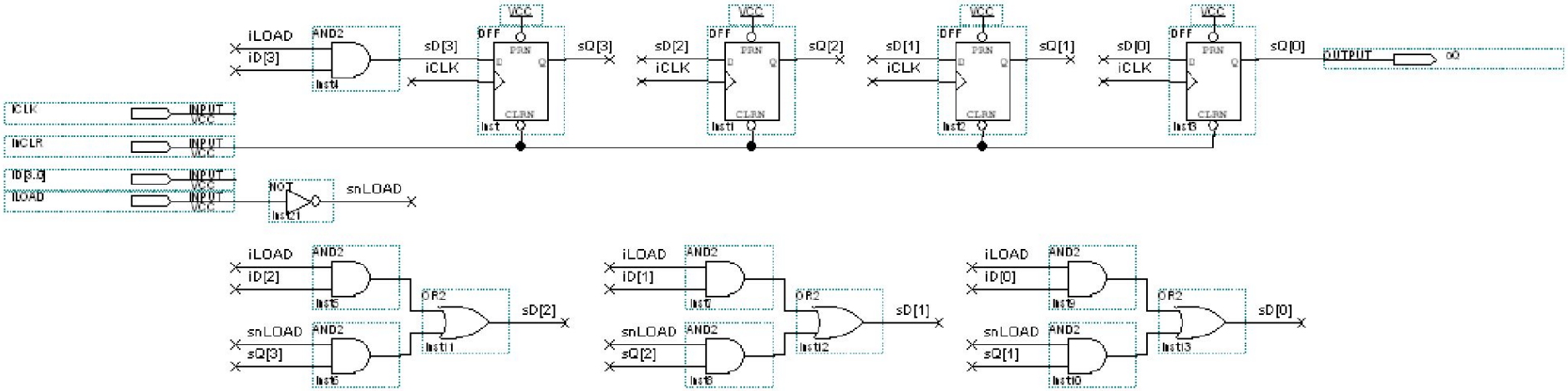


Primer: pomerački registar sa paralelnim ulazom i serijskim izlazom



LOAD	F_i
0	Q_{i+1}
1	D_i

$$F_i = \overline{LOAD} \cdot Q_{i+1} + LOAD \cdot D_i$$



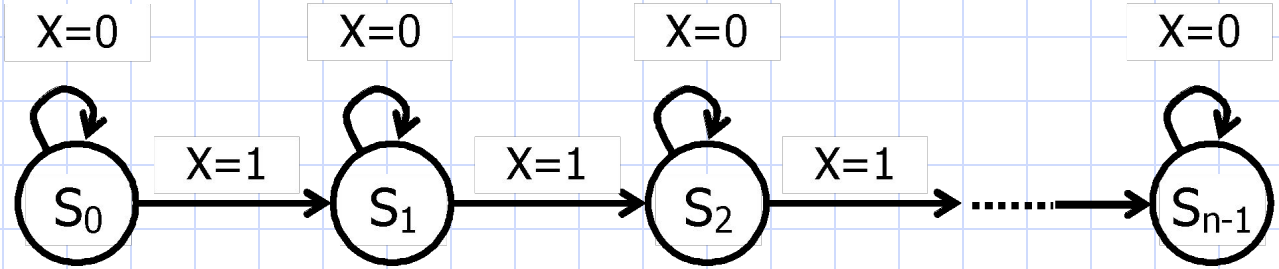
Pomerački registar u VHDL-u (serijski ulaz, paralelni izlaz)

```
PROCESS (iCLK) BEGIN
  IF (iCLK'EVENT AND iCLK = '1') THEN
    IF (inRESET = '0') THEN
      -- sinhroni reset
      sREG <= "00000";
    ELSE
      -- dozvoljeno pomeranje?
      IF (iSE = '1') THEN
        -- LSB bit ulazi prvi
        sREG <= ( iD      &
                  sREG(4) &
                  sREG(3) &
                  sREG(2) &
                  sREG(1)
                );
      END IF;
    END IF;
  END IF;
END PROCESS;
```

Operator konkatenacije

Brojači

- Brojači su sekvencijalne mreže sa jednim binarnim ulazom X (brojački impuls), čiji dijagram stanja predstavlja repetitivni ciklus.



- Broj različitih stanja u ciklusu se naziva modul ili osnova brojača (brojač modula N je brojač sa N stanja).

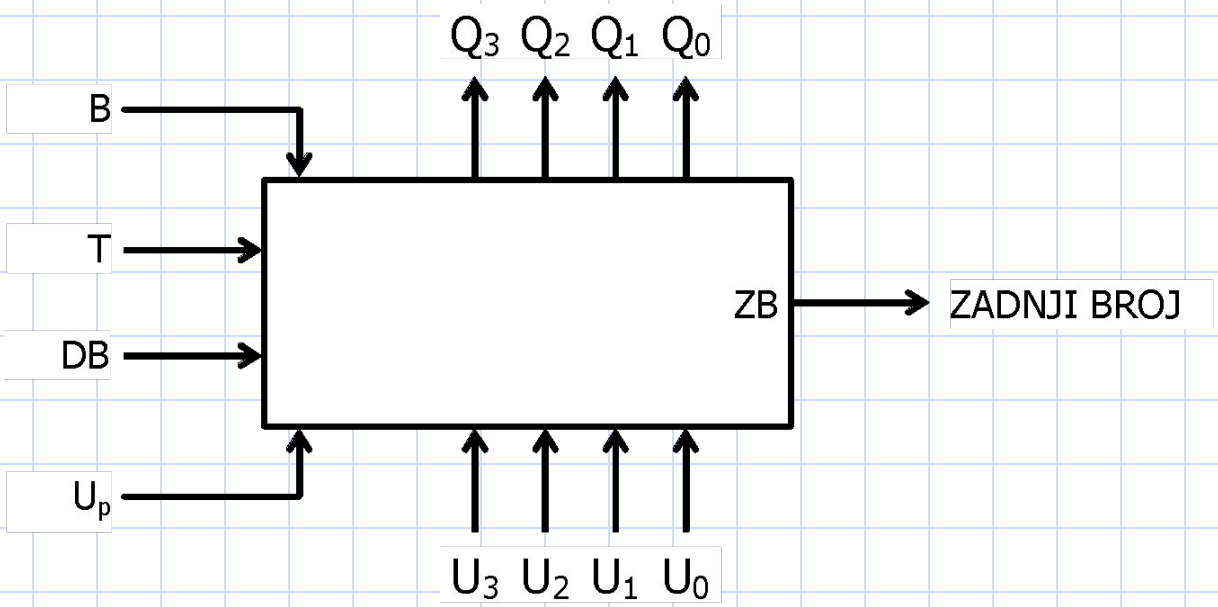
- Ako se stanja označe celim brojevima 0,1,...,N-1, funkcija prelaza brojača može se analitički izraziti u obliku.

$$S(t+1) = (S(t) + x) \bmod N$$

- Broj elementarnih automata potrebnih za realizaciju brojača sa N stanja je

$$m \geq \log_2 N$$

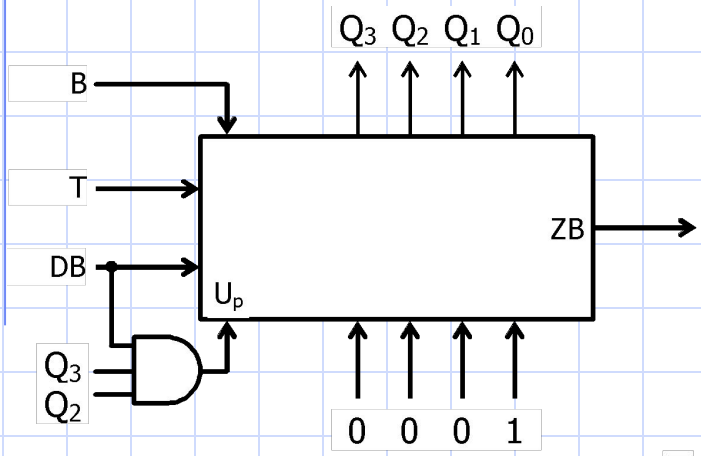
Tipičan brojač

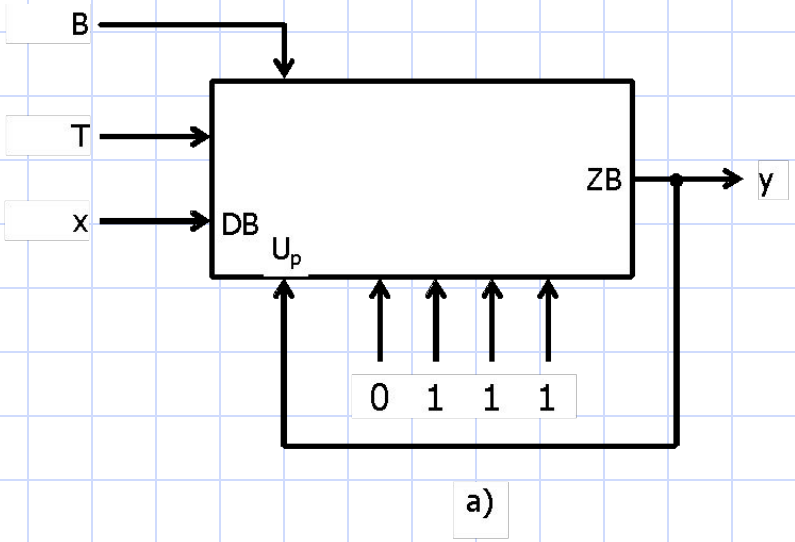


perioda N=16, n=4

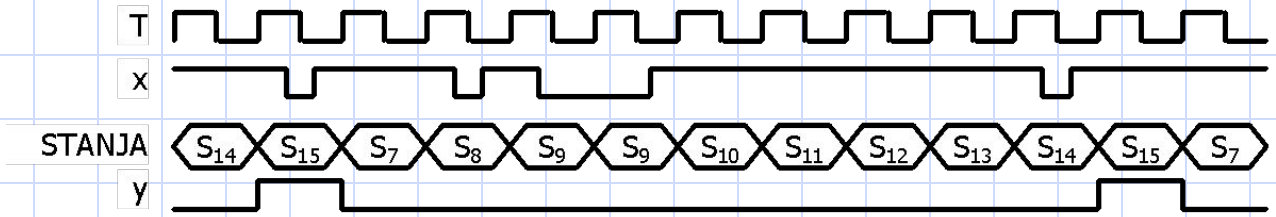
Primeri brojača

Brojač od 1 do 12





a)



b)

a) Delitelj po modulu 9
b) Deo vremenskih signala

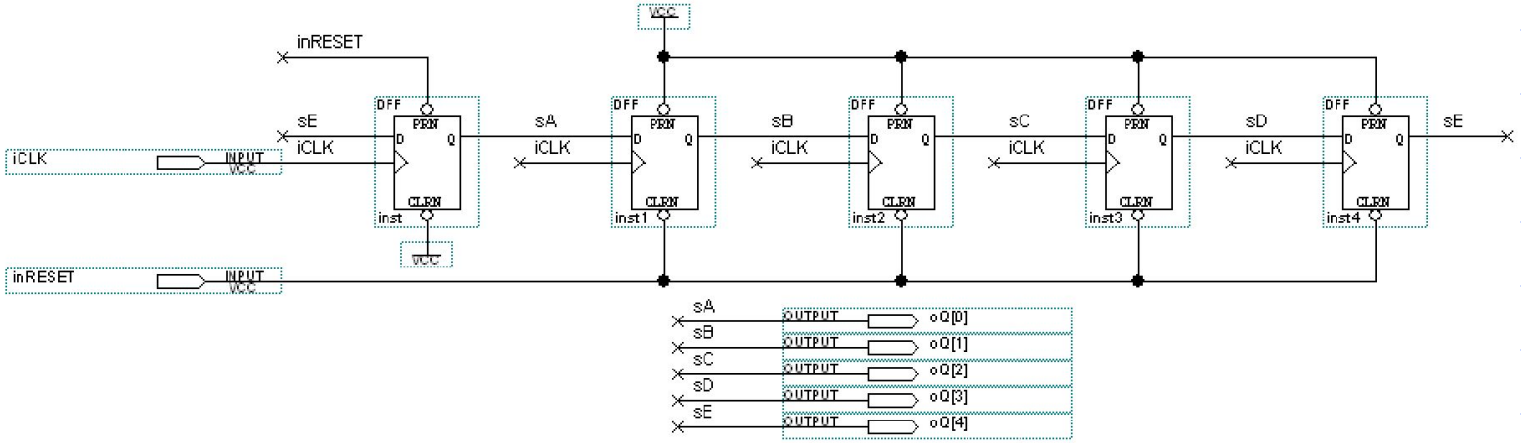
Specijalni brojači

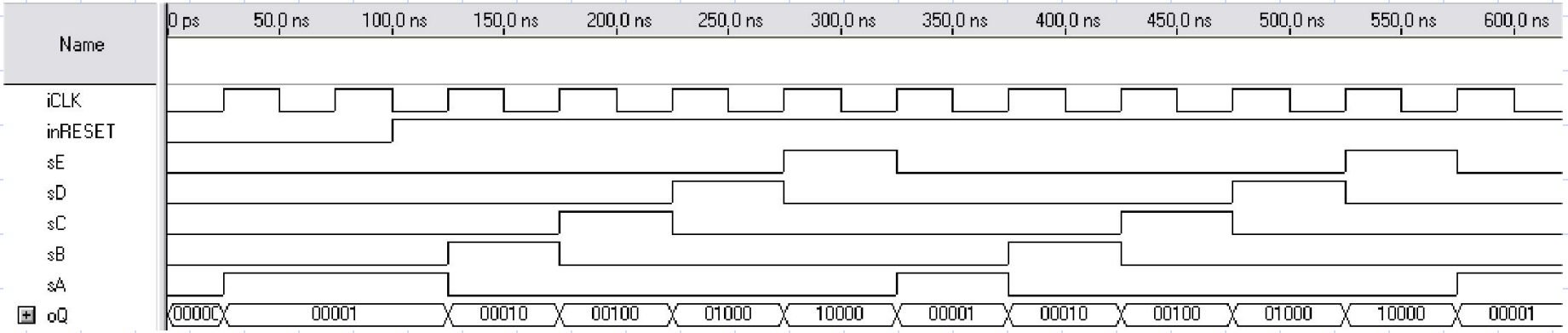
- **Decimalni brojač:**
  na izlazu se dobijaju vrednosti decimalnog brojnog sistema

- **Redni (kružni) brojač:**
  poseduje onoliko memorijskih elemenata koliko ima simbola
  (modul brojača = broj flip-flopova)

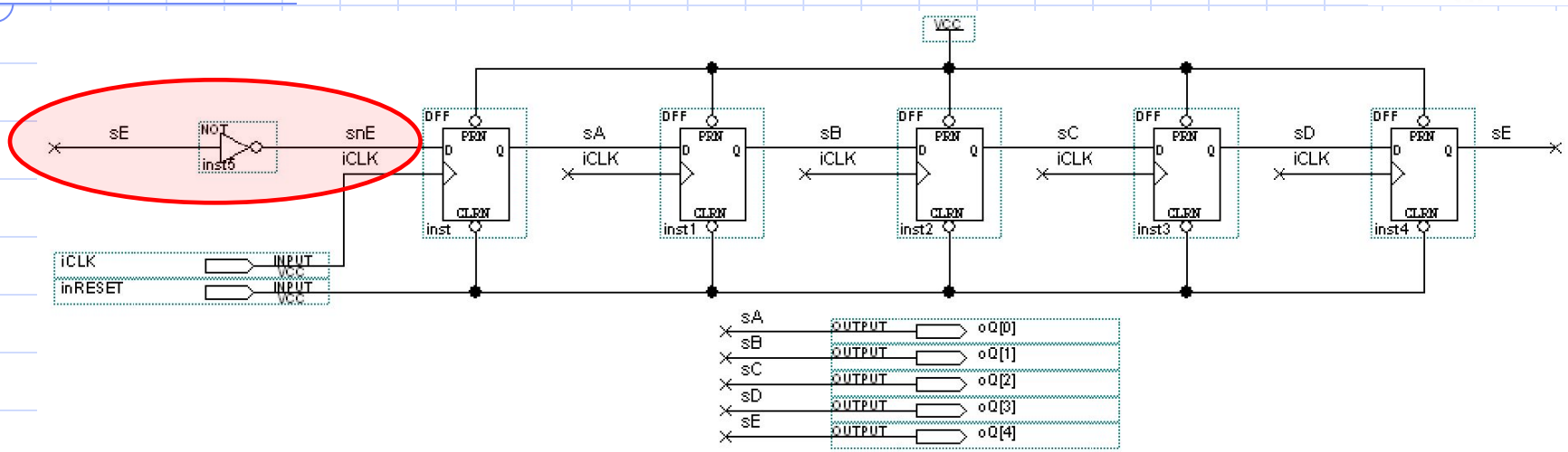
- **Džonsonov brojač:**
  Modifikacija kružnog brojača
  (modul brojača = 2 x broj flip-flopova)

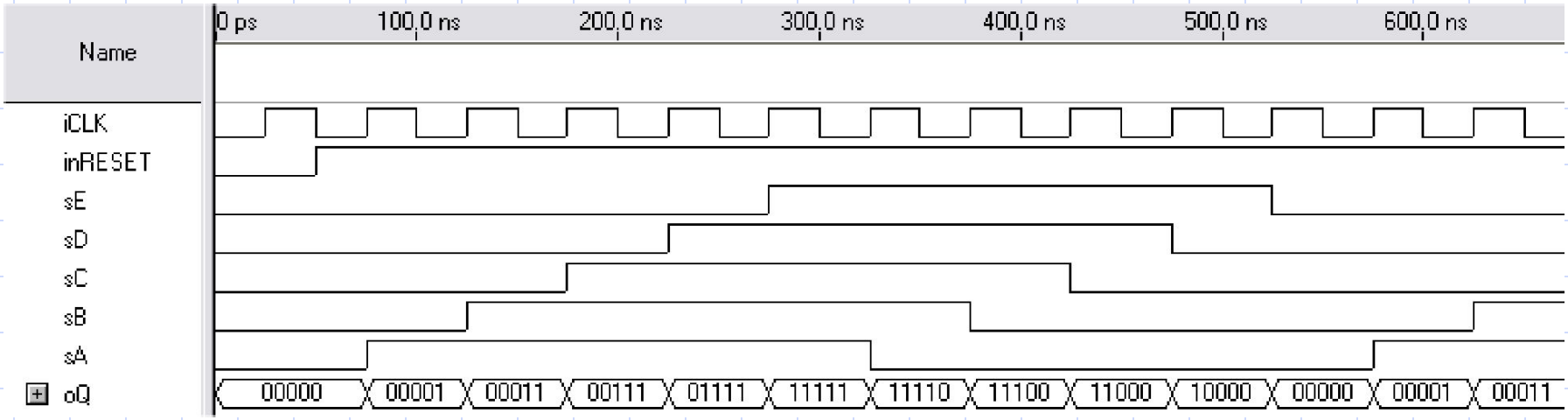
Redni brojač





Džonsonov brojač





Primer realizacije u VHDL-u

```
library IEEE;
  use IEEE.STD_LOGIC_1164.ALL;
  use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity BROJAC is
  port (
    iCLK       : in  STD_LOGIC;
    inRESET    : in  STD_LOGIC;
    iDIRECTION : in  STD_LOGIC;
    oCNT       : out STD_LOGIC_VECTOR
                     (3 downto 0)
  );
end BROJAC;
```

```
architecture BROJAC_ARH of BROJAC is
  SIGNAL sCOUNTER : STD_LOGIC_VECTOR
                    (3 DOWNTO 0) :=
                    "0000";
begin

  PROCESS (iCLK, inRESET)
  BEGIN
    IF inRESET='0' THEN
      sCOUNTER <= "0000";
    ELSIF iCLK='1' AND iCLK'event THEN
      IF iDIRECTION='1' THEN
        sCOUNTER <= sCOUNTER + 1;
      ELSE
        sCOUNTER <= sCOUNTER - 1;
      END IF;
    END IF;
  END PROCESS;

  oCNT <= sCOUNTER;

end BROJAC_ARH;
```

Preklapanje operatora tipa `std_logic_vector` (`sCOUNTER`) i tipa `integer` (+1 i -1)

Detektor rastuće ivice

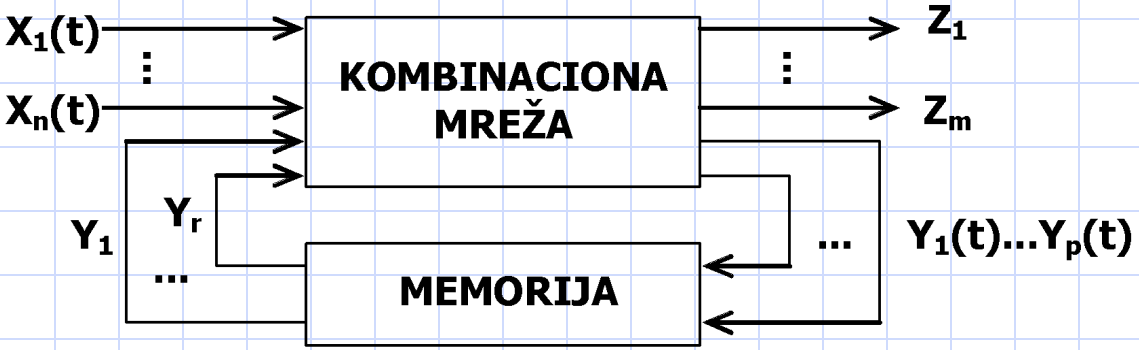
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity detektor_rast is
    Port ( inRST : in  STD_LOGIC;
           inCLK : in  STD_LOGIC;
           iX    : in  STD_LOGIC;
           oDETECT : out STD_LOGIC);
end detektor_rast;

architecture Behavioral of detektor_rast is

signal niz : std_logic_vector(3 downto 0);

begin
```

```
process (inCLK,inRST)
begin
   if inRST = '0' then
        niz <= "0000";
   elsif (inCLK'event and inCLK = '1') then
       niz <= niz(2 downto 0) & iX;
   end if;
end process;

process (niz)
begin
      case niz is
        when "0111" => oDETECT <= '1';
        when others => oDETECT <= '0';
      end case;
end process;

end Behavioral;
```

Definicija sekvencijalnih mreža

- Opšti model digitalnog sistema se definiše funkcijom u vremenu sa n ulaznih promenljivih i p izlaznih promenljivih

- Ukoliko vrednosti izlaznih promenljivih zavise ne samo od trenutnih vrednosti ulaznih promenljivih nego i od prošlih vrednosti (parova ulaza-izlaza) za digitalni sistem se kaže da je **sekvencijalni** sistem ili **automat**



Vremensko ponašanje automata

- Izlaz automata u trenutku t, označen sa z(t), zavisi od ulazne vremenski zavisne funkcije u intervalu (-∞, t)
- Vrednosti ulazne vremenske funkcije x(-∞, t) grupiše u klase tako da sve vremenske funkcije koje imaju isti uticaj na izlaz u trenutku t, pripadaju istoj klasi.
- Klase predstavljaju **STANJA** koja se označavaju promenljivom S. Njima se izražava uticaj prošlih ulaza na trenutne i buduće vrednosti izlaza
- U praktičnim sistemima broj klasa je konačan.
- Ukoliko su ulazna i izlazna azbuka (alfabet) stanja konačne, automat se naziva **KONAČNIM**

Formalni matematički opis sekvencijalnih sistema

- Apstraktni automat je matematički model prekidačkog upravljačkog automata koji se zadaje skupom iz šest elemenata:

$W=(X, Y, S, \delta, \lambda, S_0)$

$X = (x_1, x_2, ..., x_n)$ - skup ulaznih signala ili ulazna azbuka/alfabet
(ulazna reč automata)

$Y = (y_1, y_2, ..., y_m)$ - skup izlaznih signala ili izlaznih azbuka/alfabet
(izlazna reč automata)

$S = (s_1, s_2, ..., s_k)$ - skup stanja ili azbuka/alfabet stanja

S_0 - početno stanje

δ - funkcija prelaza koja realizuje azbučno preslikavanje skupa $S \times X \rightarrow S$

λ - funkcija izlaza koja realizuje preslikavanje skupa $S \times X \rightarrow Y$

Vremensko modelovanje automata

- Zavisnost izlaza u trenutku t od ulaza i stanja u istom vremenskom trenutku izražava se tzv. **FUNKCIJOM IZLAZA**

  $Z(t) = \lambda(X(t), S(t))$

- Uticaj ulazne vremenske funkcije se izražava i u odnosu na promenu stanja, odnosno, novo stanje zavisi od trenutnog stanja i ulaza. U tom slučaju govori se o **FUNKCIJI PRELAZA**

  $S(t+\Delta) = \delta(S(t), X(t))$

  pri čemu je S(t) trenutno (sadašnje), a S(t+Δ) sledeće (naredno) stanje

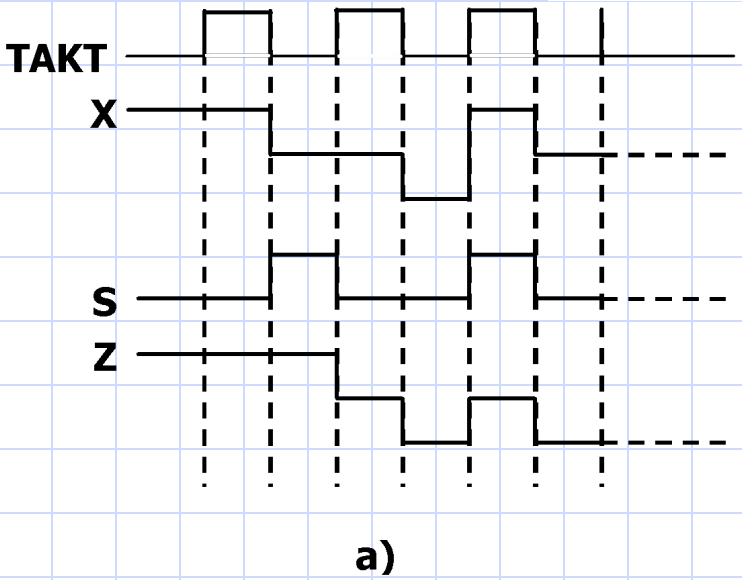
- Pošto sinhroni sekvencijalni sistemi mogu menjati stanje u diskretnim trenucima kontinualna promenljiva t se zamenjuje diskretnom promenljivom definisanom pozitivnim celim brojem

- Sistem je u stanju S(i) u vremenskom intervalu (t-1=i-1, t=i). Sinhrona sekvencijalna mreža se može opisati kao

  $Z(t) = \lambda(S(t), X(t))$
  $S(t+1) = \delta(S(t), X(t))$

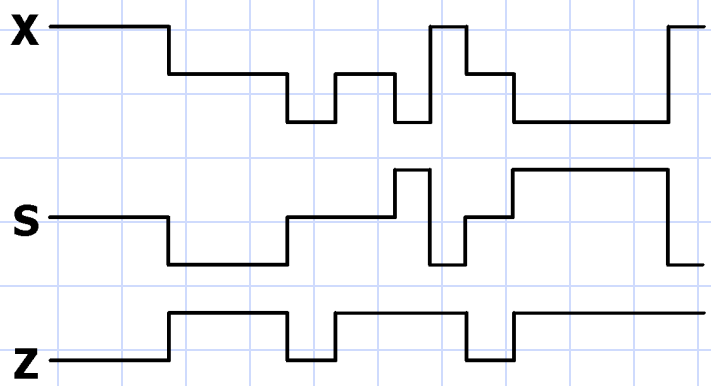
Sinhrone i asinhrone sekvencijalne mreže

- Kod sinhronih mreža ulazi, izlazi i interna stanja se menjaju u diskretnim vremenskim trenucima, definisanim preko sinhronizacionog ulaza osnovnom frekvencijom takta sistema, tj. taktom



a)

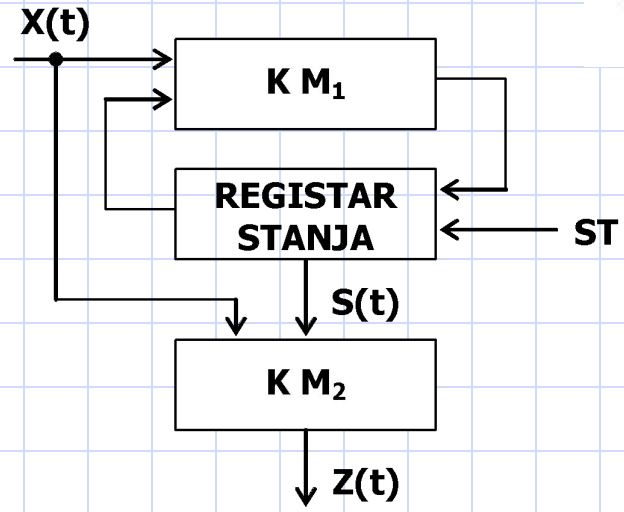
- Kod asinhronih sekvencijalnih mreža stanja se mogu menjati u bilo koje vreme, a ulazi mogu biti signali različitih nivoa koji se javljaju u proizvoljnom intervalu vremena



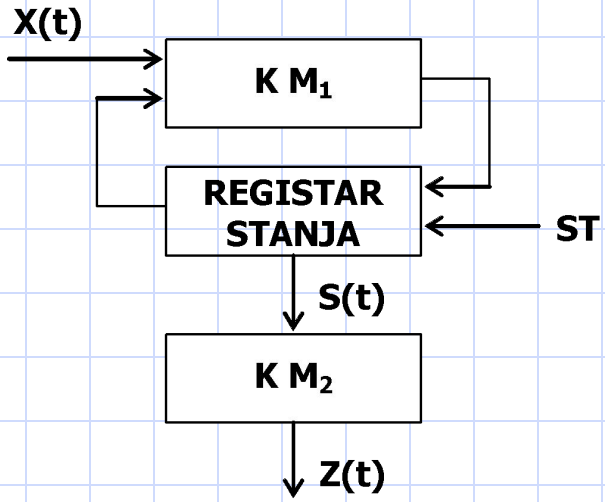
b)

Milijev i Murov automat

- U odnosu na funkciju izlaza u praksi se sreću dva slučaja
  - Automati prve vrste ili **Milijevi** (*Mealy*) automati definišu funkciju izlaza u obliku
    $Z(t)= \lambda(S(t), X(t))$
  - Automati druge vrste ili automati **Mura** (*Moore*) definišu funkciju izlaza
    $Z(t)= \lambda(S(t))$



Milijev automat



Murov automat

Zadavanje konačnog automata tabličnom metodom 1/2

- Milijev automat se opisuje tablicama prelaza i izlaza

tablica prelaza

X/S	S_0	$S_1 \dots S_k$
X_1	$\delta(S_0, X_1)$	$\delta(S_1, X_1) \dots \delta(S_k, X_1)$
…	…	. . .
X_m	$\delta(S_0, X_m)$	$\delta(S_1, X_m) \dots \delta(S_k, X_m)$

tablica izlaza

X/S	S_0	$S_1 \dots S_k$
X_1	$\lambda(S_0, X_1)$	$\lambda(S_1, X_1) \dots \lambda(S_k, X_1)$
…	…	. . .
X_m	$\lambda(S_0, X_m)$	$\lambda(S_1, X_m) \dots \lambda(S_k, X_m)$

Primer 1: automat prve vrste (Milijev automat)

A_1 tabela prelaza

	S_0	S_1	S_2
X_1	S_2	S_0	S_0
X_2	S_0	S_2	S_1

A_1 tabela izlaza

	S_0	S_1	S_2
X_1	Y_1	Y_1	Y_2
X_2	Y_1	Y_2	Y_1

Zadavanje konačnog automata tabličnom metodom 2/2

Primer 2: nepotpuno definisan automat

A_2 tabela prelaza

	S_0	S_1	S_2	S_3
X_1	S_1	S_2	S_3	-
X_2	S_2	-	S_1	S_1

A_2 tabela izlaza

	S_0	S_1	S_2	S_3
X_1	Y_1	Y_3	Y_3	-
X_2	Y_2	-	Y_1	Y_2

Primer 3: Murov automat (automat druge vrste)

	$\lambda(S_0) \ldots \lambda(S_k)$
	$S_0 \ldots S_k$
X_1	$\delta(S_0, X_1) \ldots \delta(S_k, X_1)$
...	...
X_p	$\delta(S_0, X_p) \ldots \delta(S_k, X_p)$

Uopšteni Murov automat

A_3	Y_1	Y_1	Y_3	Y_2	Y_3
	S_0	S_1	S_2	S_3	S_4
X_1	S_1	S_4	S_4	S_2	S_2
X_2	S_3	S_1	S_1	S_0	S_0

Konačan Murov automat

Zadavanje konačnog automata grafom 1/3

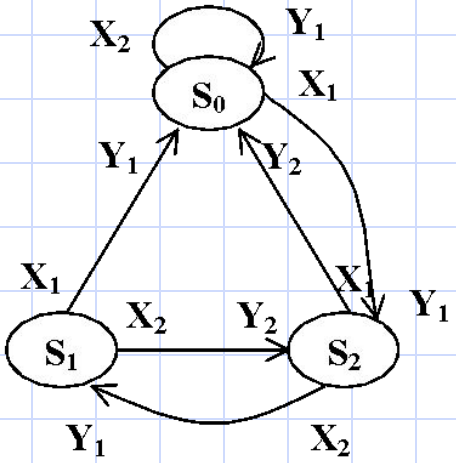
- Graf automata je orijentisani graf čiji čvorovi odgovaraju stanjima, a lukovi prelazima između stanja.
- Dva čvora grafa automata Si i Sj (polazno i stanje prelaza) spajaju se lukovima usmerenim od Si ka Sj ako u automatu postoji prelaz iz Si u Sj, tj. ako je Sj= δ(Si, Xr) pri nekom ulaznom signalu Xr.
- Luku (Si, Sj) grafa automata dodeljuje se ulazni signal X i izlazni signal Y=λ(S, X) ako je on definisan, a u protivnom se stavlja crtica.
- Ako prelaz automata iz stanja Si u Sj proizilazi pod uticajem nekoliko ulaznih signala, luku (Si, Sj) dodeljuju se svi ti ulazni signali i odgovarajući izlazni signali.
- Pri opisu Murovog automata u vidu grafa izlazni signal Ys=λ(Si) zapisuje se unutar čvora Si ili odmah pored njega

Zadavanje konačnog automata grafom 2/3

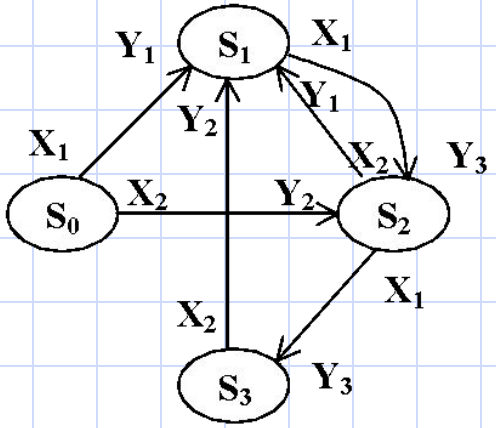
- Luk u nekom čvoru Si može biti:
  - povratni (Si, Si)
  - odlazni (Si, Sj)
  - dolazni (Sj, Si)

- Prema vrsti lukova u čvoru S, stanja automata su:
  - izolovana ako čvor ima samo povratne grane
  - prelazna ako u čvoru nema dolaznih lukova
  - ustaljena ako čvor nema odlaznih lukova.

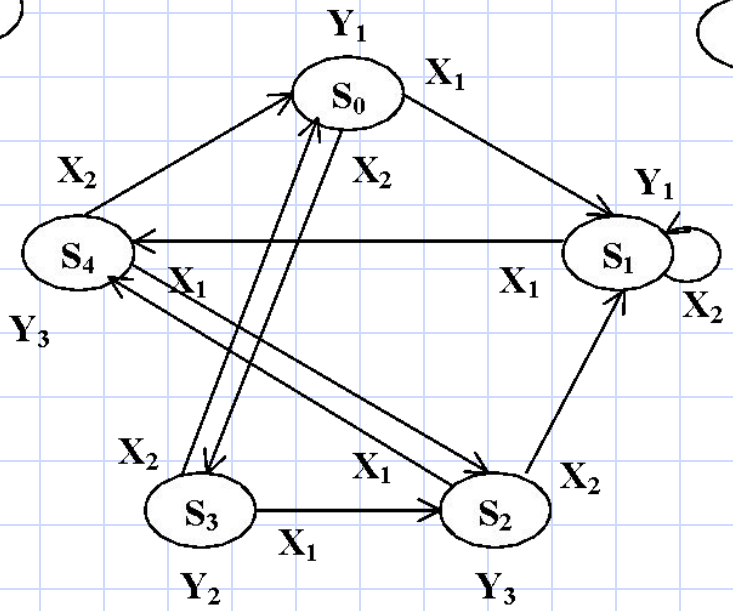
Zadavanje konačnog automata grafom 3/3



automat A_1



automat A_2



automat A_3

Zadavanje konačnog automata matričnom metodom 1/2

- Matrično zadavanje automata vrši se preko kvadratne matrice C=||Cij|| čije vrste odgovaraju polaznim stanjima, a kolone stanjima prelaza.
- Element Cij=Xp/Yq koji stoji na preseku i-te vrste i j-te kolone u slučaju Milijevog automata, odgovara ulaznom signalu Xp koji izaziva prelaz iz stanja Si u Sj i izlaznom signalu Yq, koji se izdaje pri tom prelazu

$$C_1 = \begin{Vmatrix} X_2/Y_1 & & X_1/Y_1 \\ X_1/Y_1 & & X_2/Y_2 \\ X_1/Y_2 & X_2/Y_1 & \end{Vmatrix}$$

Automat A_1

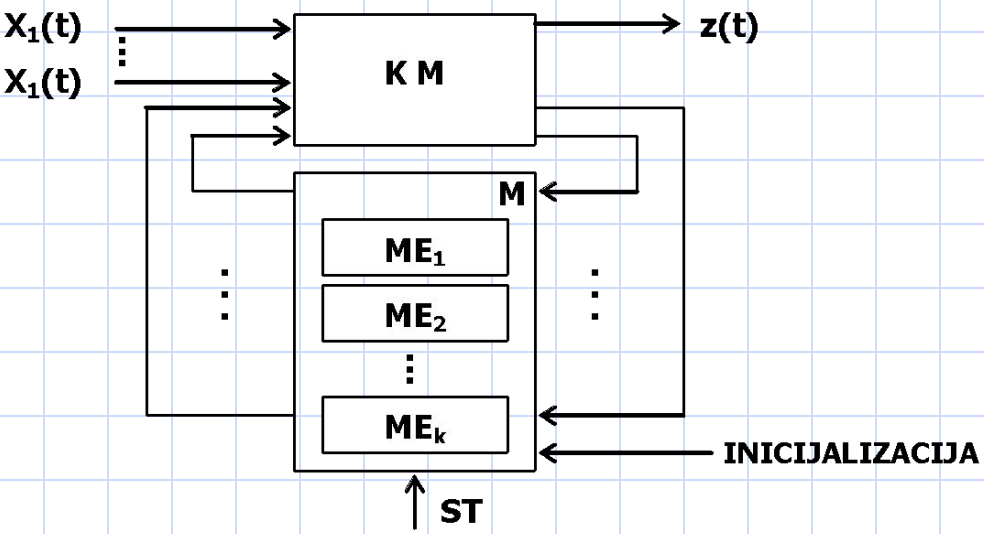
Zadavanje konačnog automata matričnom metodom 2/2

- Primer zadavanja Murovog automata

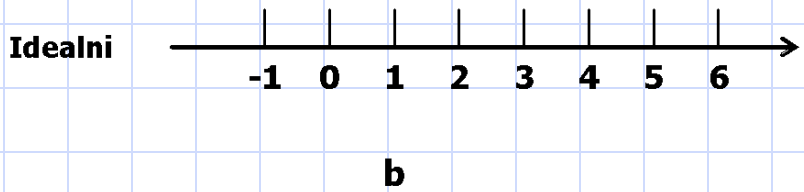
$$C = \begin{Vmatrix} & X_1 & & X_2 & \\ & X_2 & & & X_1 \\ & X_2 & & & X_1 \\ X_2 & & X_1 & & \\ X_2 & & X_1 & & \end{Vmatrix} \qquad Y_q = \begin{Vmatrix} Y_1 \\ Y_1 \\ Y_3 \\ Y_2 \\ Y_3 \end{Vmatrix}$$

Automat A_3

Kanonička struktura automata



a



b

- Pošto je sistem sinhron, signal takta ili sinhronizacije (označen ST) određuje vremenske trenutke u kojima dolazi do formiranja narednog stanja i izdvajanja izlaznog signala.
- U vreme t unosi se novo stanje u memoriju i tu se čuva sve do trenutka t+1.
- Signal takta čine periodični impulsi takta, koji u idealnom slučaju imaju nultu širinu

- Inicijalizacija se realizuje nad delom memorije posebnim signalom inicijalizacije.

Kodirane tablice ulaza, izlaza i stanja

	Q_1	Q_2
S_0	0	0
S_1	0	1
S_2	1	0
S_3	1	1

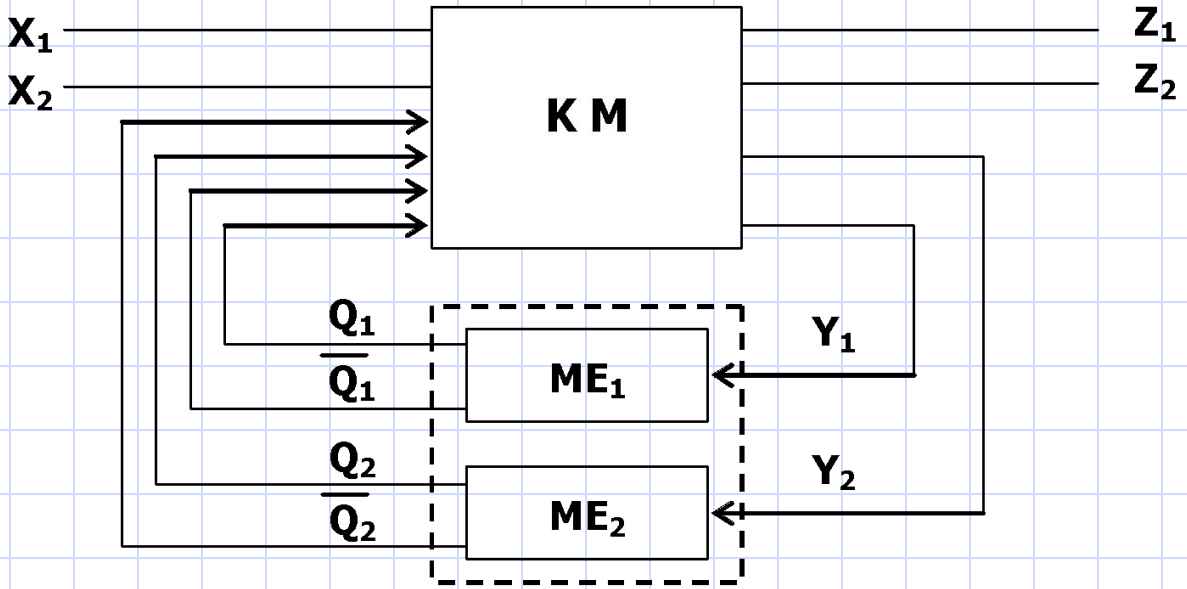
a) stanja

	X_1	X_2
W_0	0	0
W_1	0	1
W_2	1	0

b) ulazni simboli

	Z_1	Z_2
Y_0	0	0
Y_1	0	1
Y_2	1	1
Y_3	1	0

c) izlazni simboli



Jednoprocesni automat u VHDL-u

```
architecture behavioral of sm is
    type state_t is (s1, s2, s3);
    signal state : state_t;
begin
    oneproc: process(rst, clk)
    begin
        if (rst = '0') then
            -- Reset
        elsif(clk'event and clk = '1') then
            case state is
                when s1 =>
                    if (input = '1') then
                        state <= s2;
                    else
                        state <= s1;
                    end if;
                    ...
                    ...
            end case;
        end if;
    end process;
end architecture;
```

Dvoprocesni automat u VHDL-u

```
architecture behavioral of sm is
    type state_t is (s1, s2, s3);
    signal state, next_state : state_t;
begin
    syncproc: process(rst, clk)
    begin
        if (rst = '0') then
          state   <= s1;
        elsif(clk'event and clk = '1') then
       state   <= next_state;
       end if;
end process;

     combproc: process(state, input)
    begin
        case state is
            when s1 =>
                if (input = '1') then
                    next_state <= s2;
                else
                   next_state <= s1;
                end if;
               ...
        end case;
    end process;
end architecture;
```