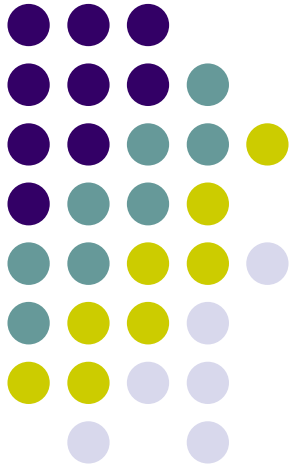
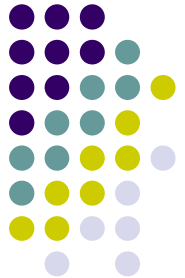
Projektovanje procesorskih resursa

Projektovanje RISC procesora

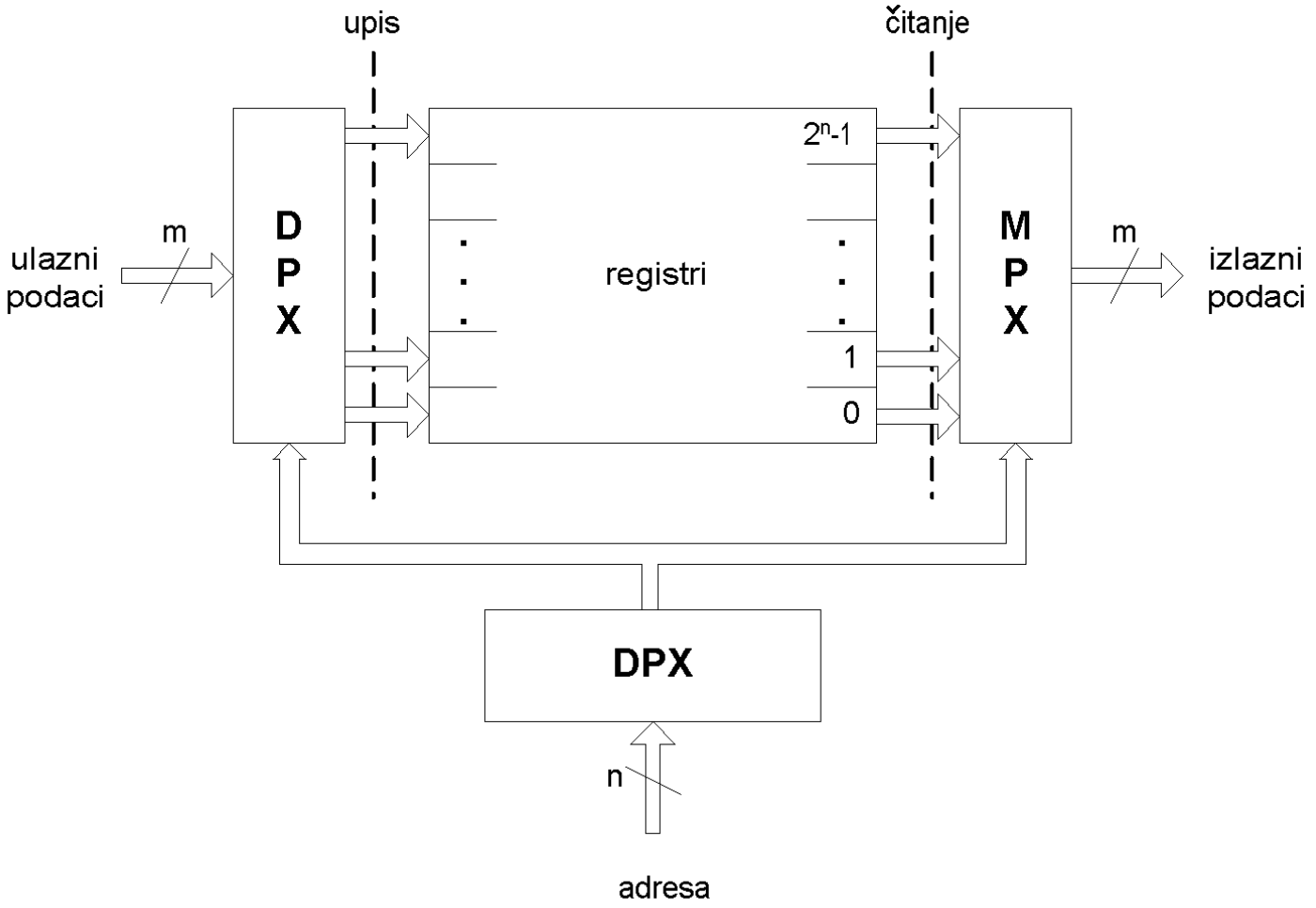


Osnovni resursi

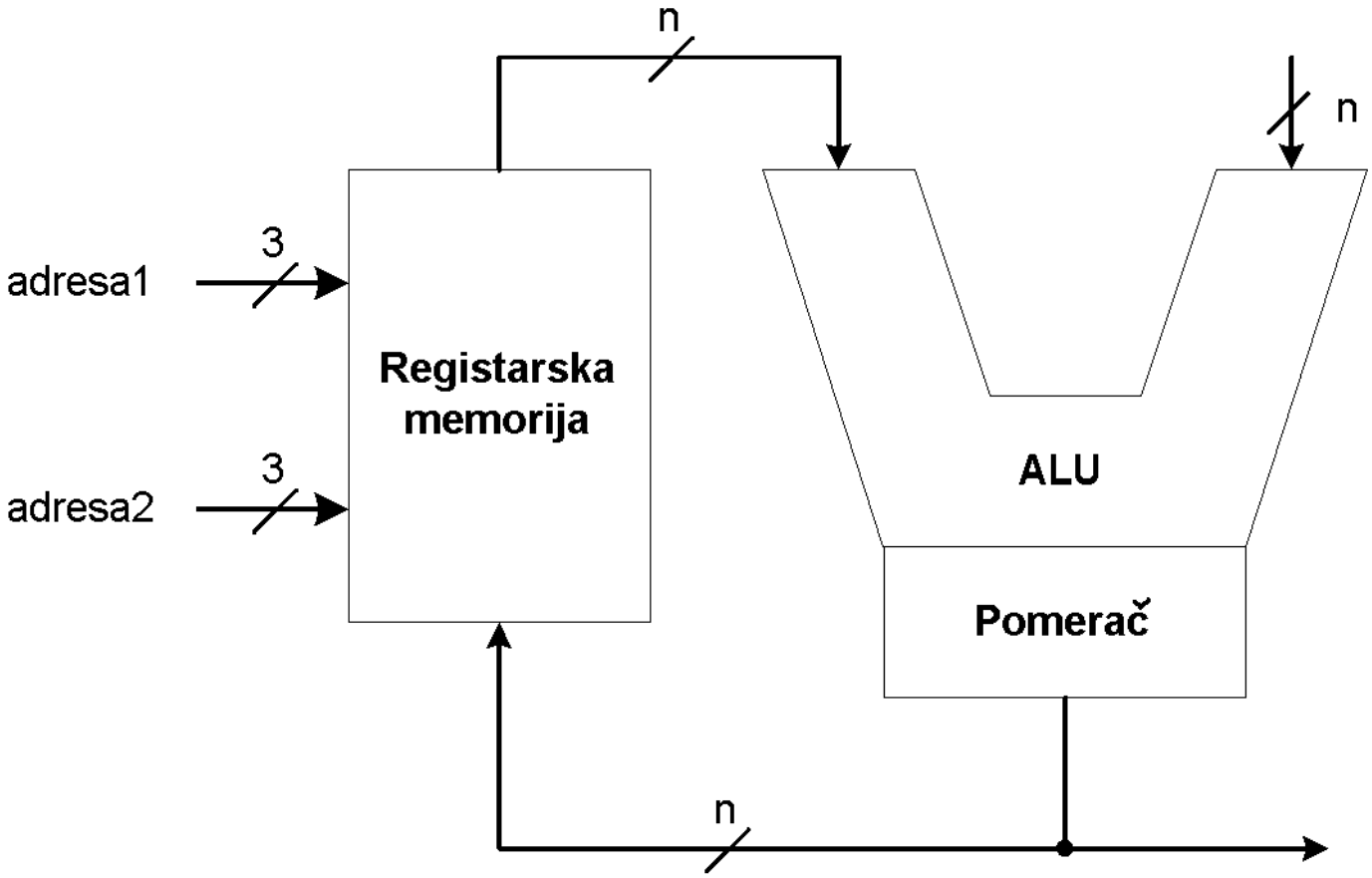


- Registarske memorije
- Brojači
- Sabirači
- Pomerači
- Množači

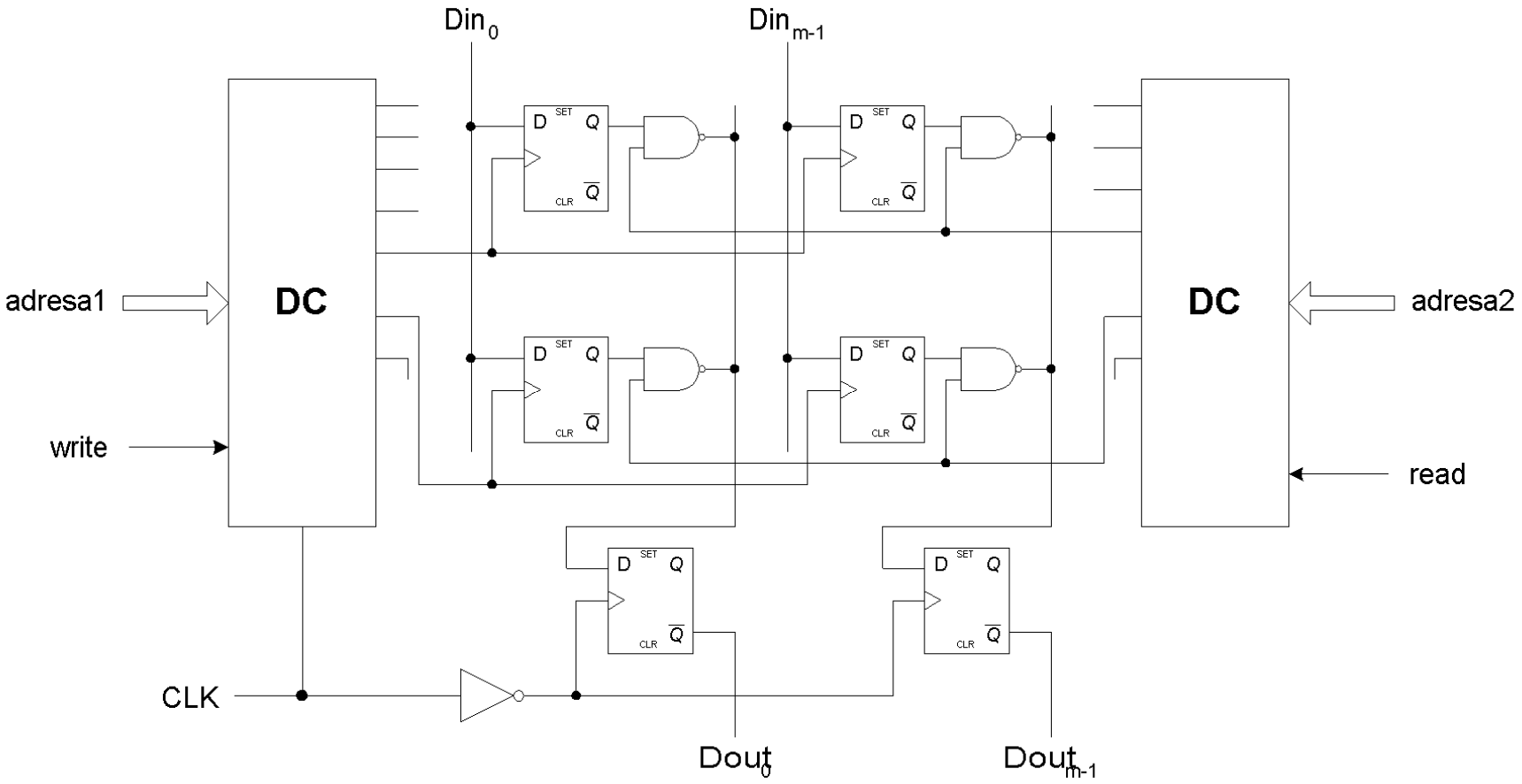
Registarske memorije – blok šema



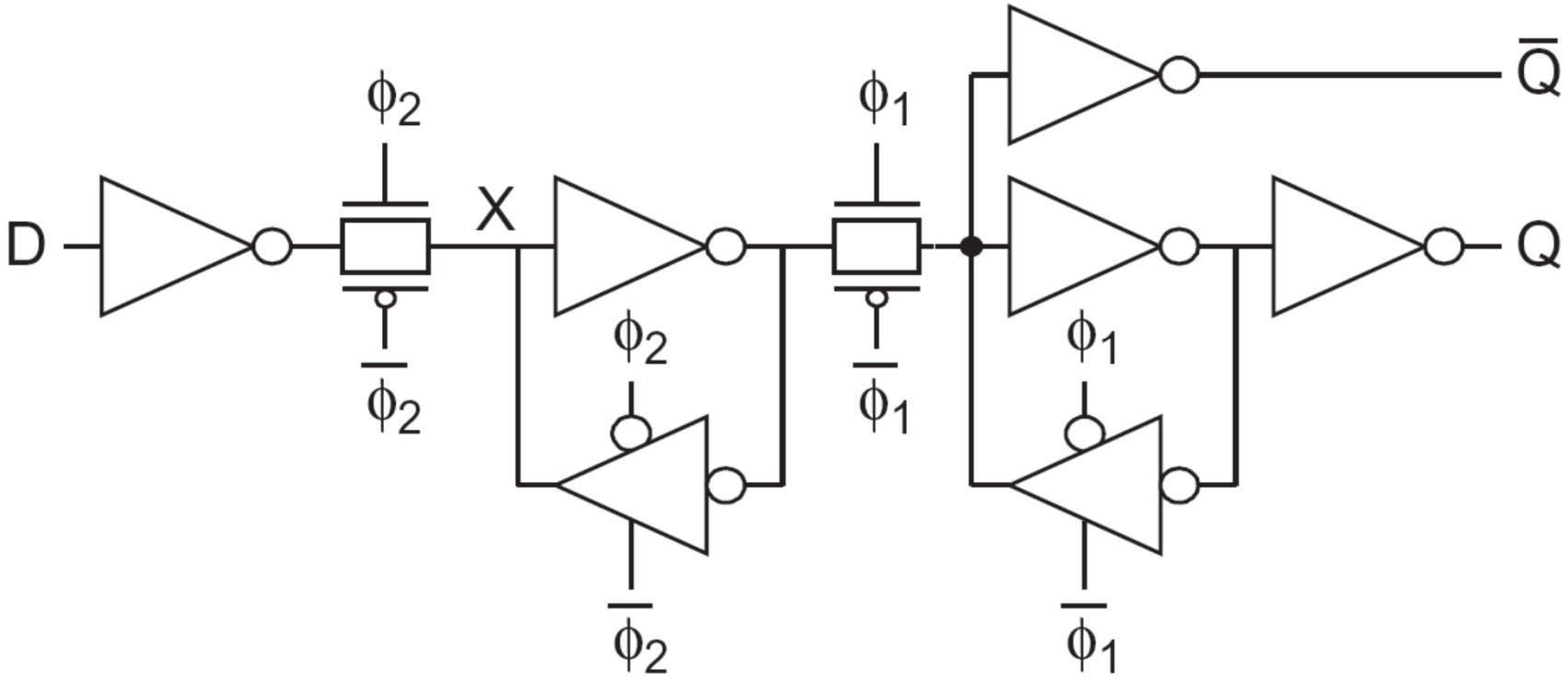
Registarske memorije - istovremeni upis i čitanje



Registarske memorije - istovremeni upis i čitanje

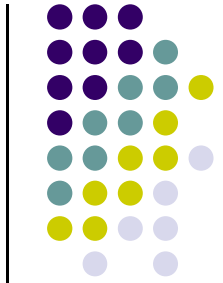


Registarske memorije – D FF

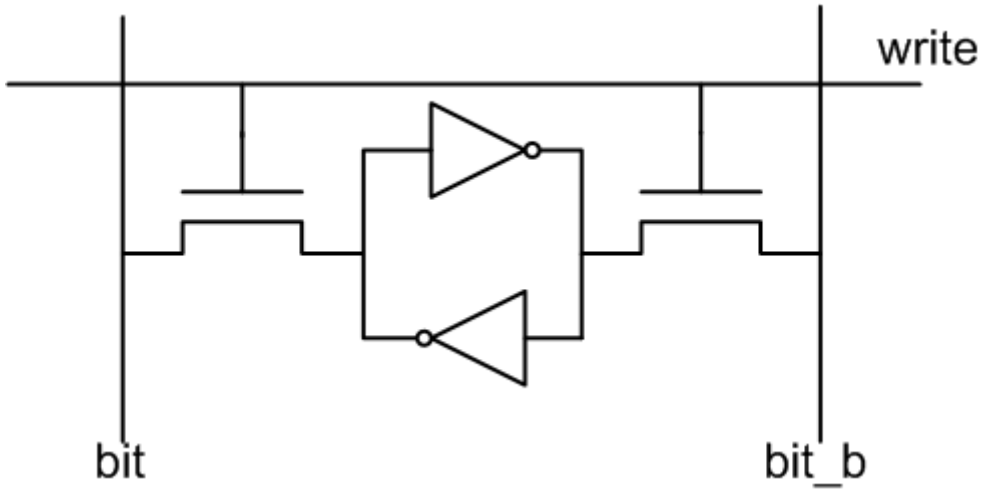


- D flip-flop bez reseta
- Koliko tranzistora sadrži?

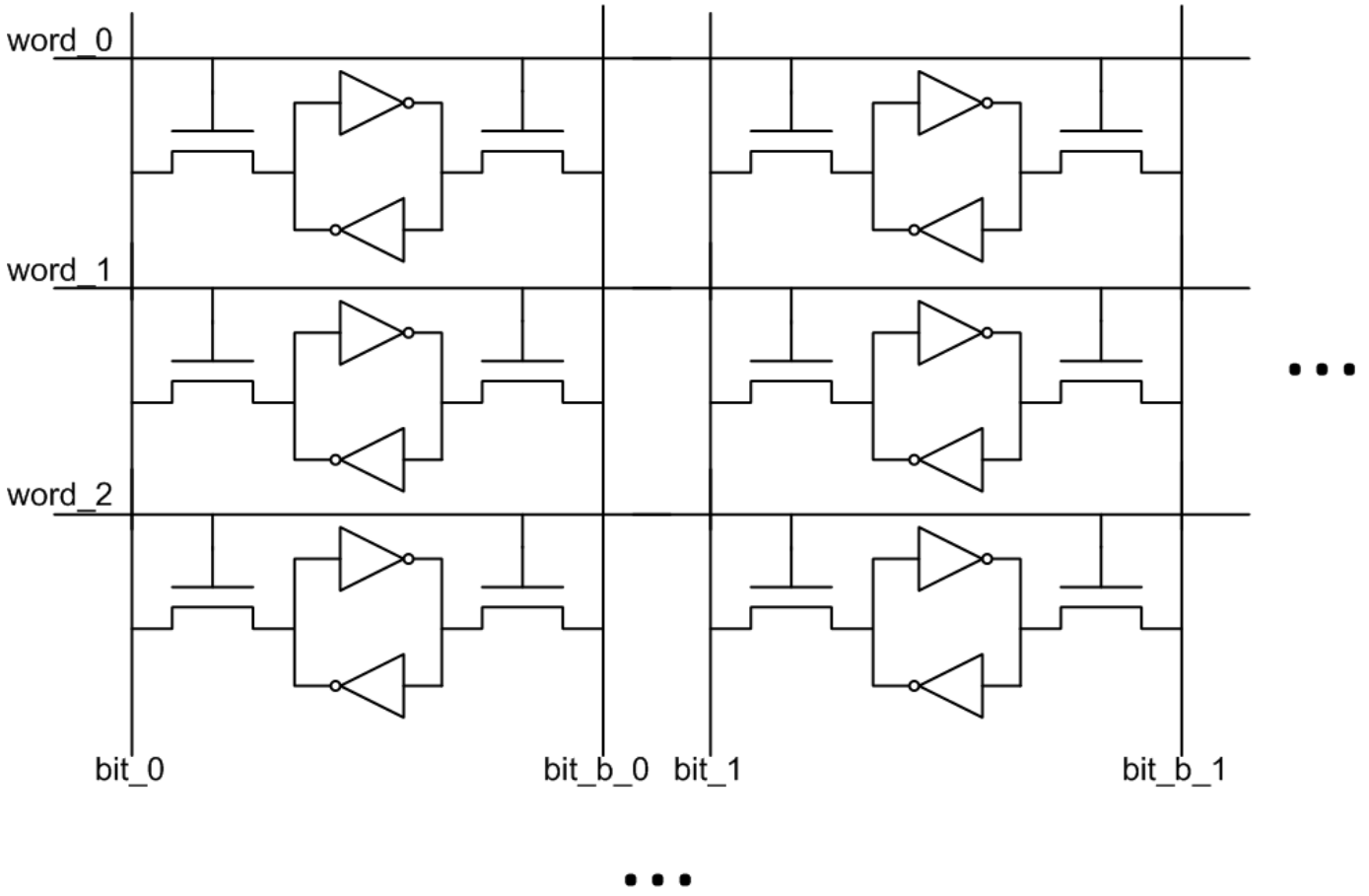
Registarske memorije – SRAM



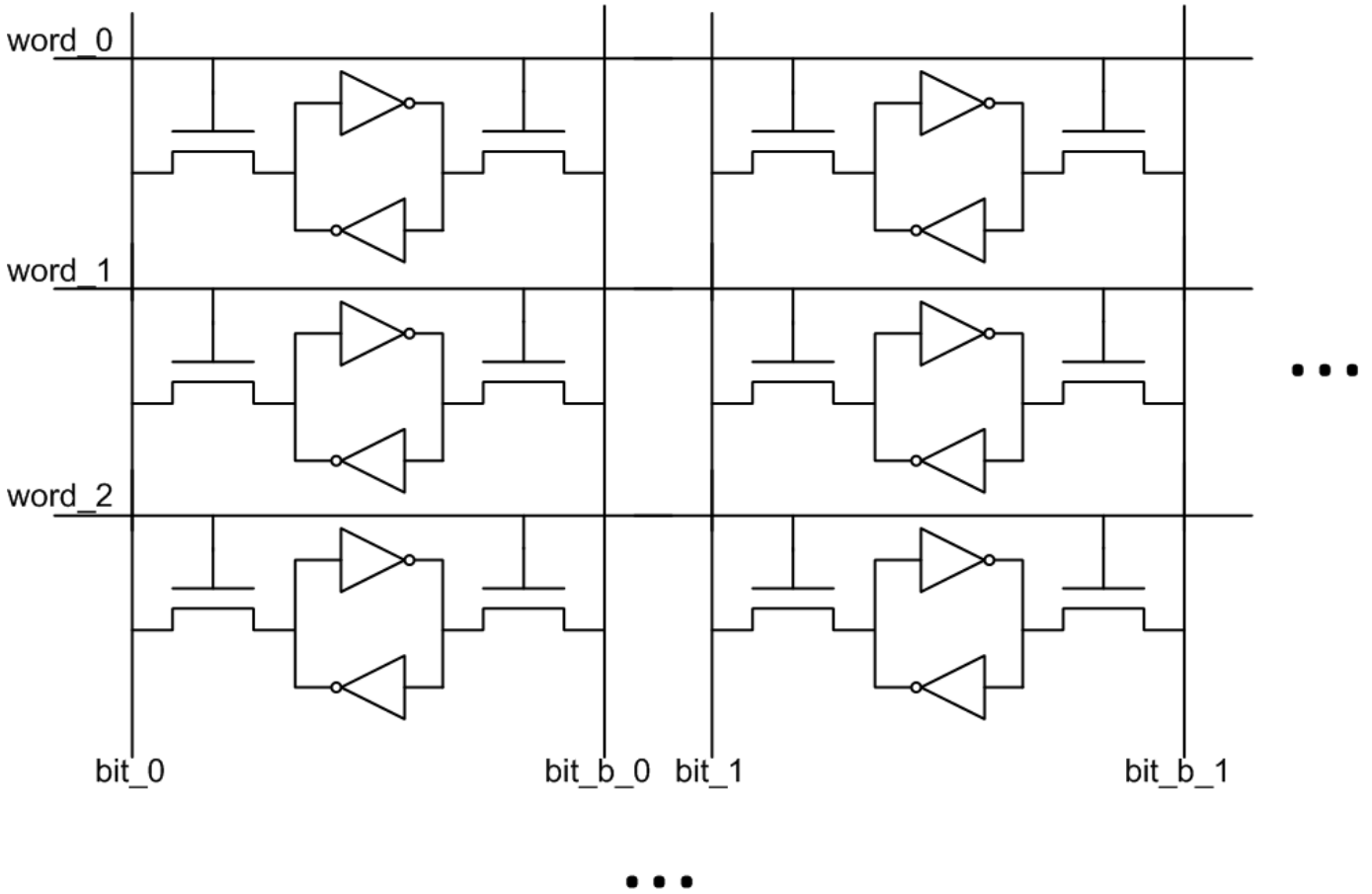
- Static RAM
- Najčešće se koristiti 6T ćelija za pamćenje jednog bita



Registarske memorije – SRAM

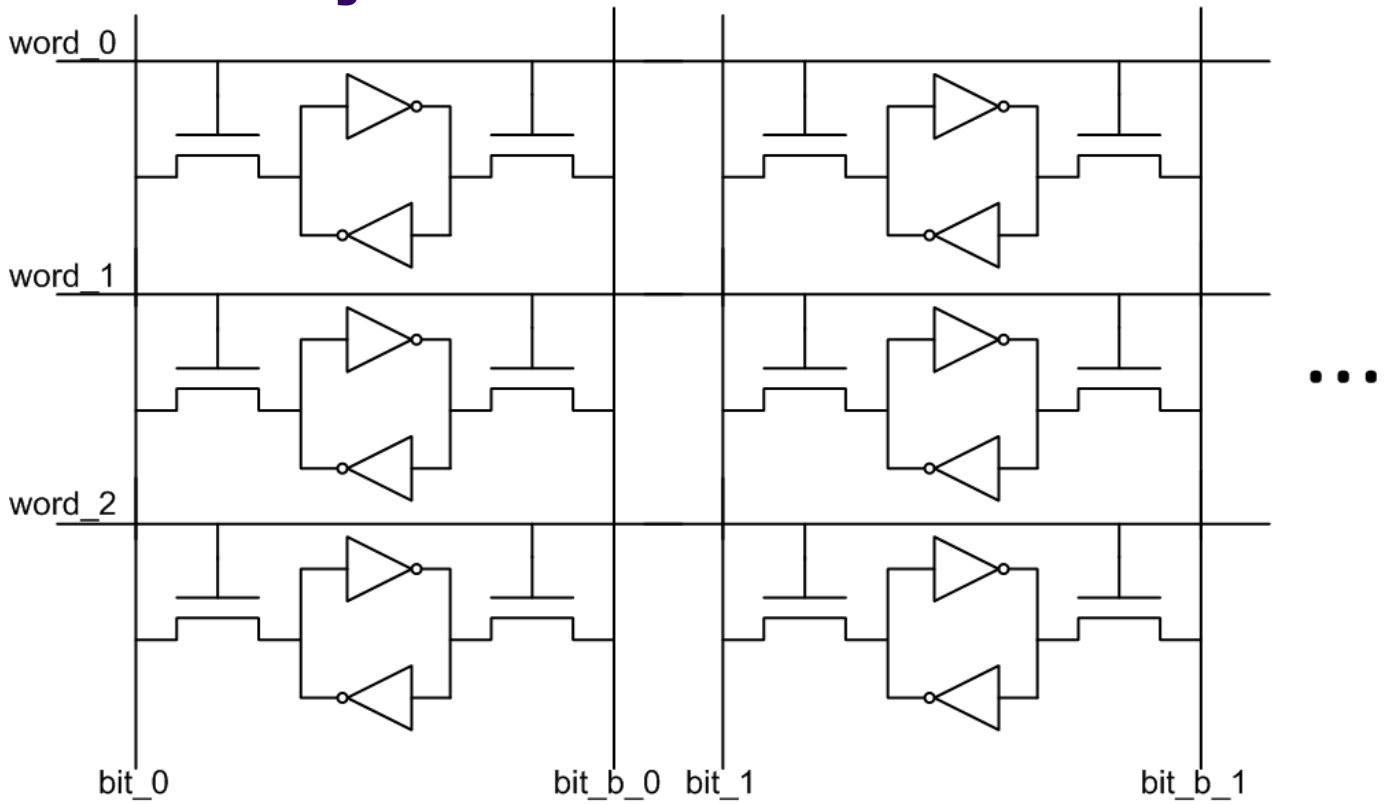


Registarske memorije – SRAM upis



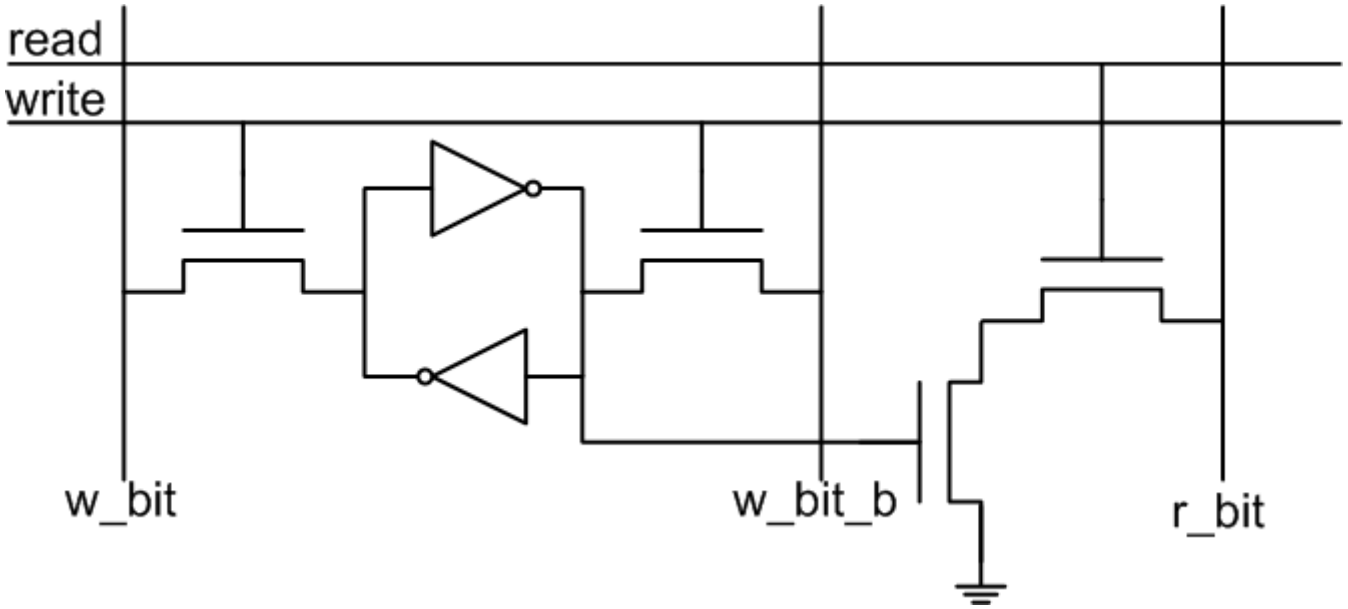
- Na bitove kolona se postave komplementarne vrednosti (npr. bit = 0 i bit_b = 1)

Registarske memorije – SRAM čitanje



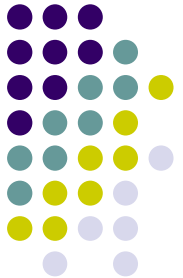
- Na bitove kolona se postave jedinice
- Bit reda se postavi na jedan
- Jedan bit kolone se spušta na nulu

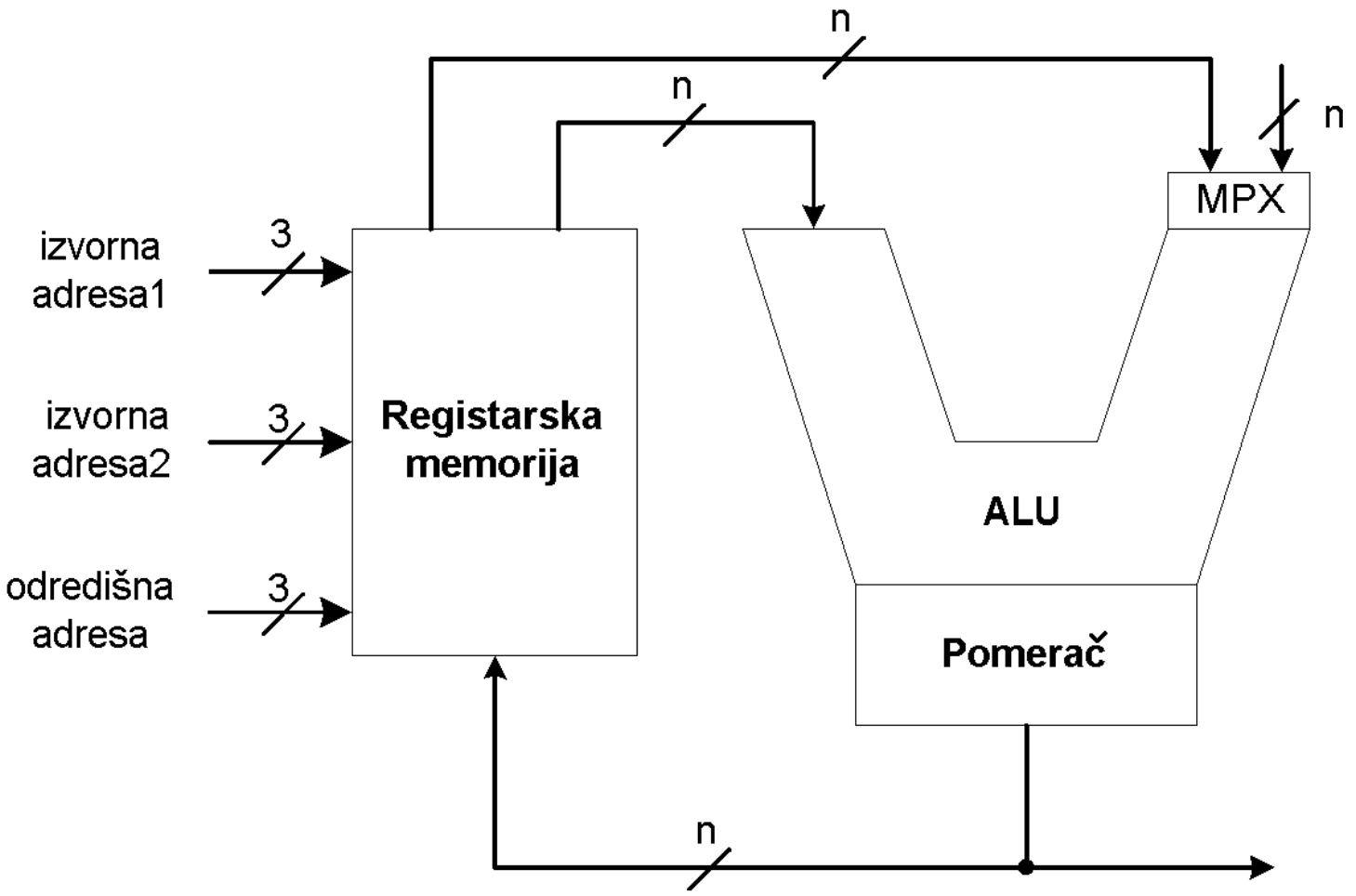
Registarske memorije – SRAM 8T ćelija



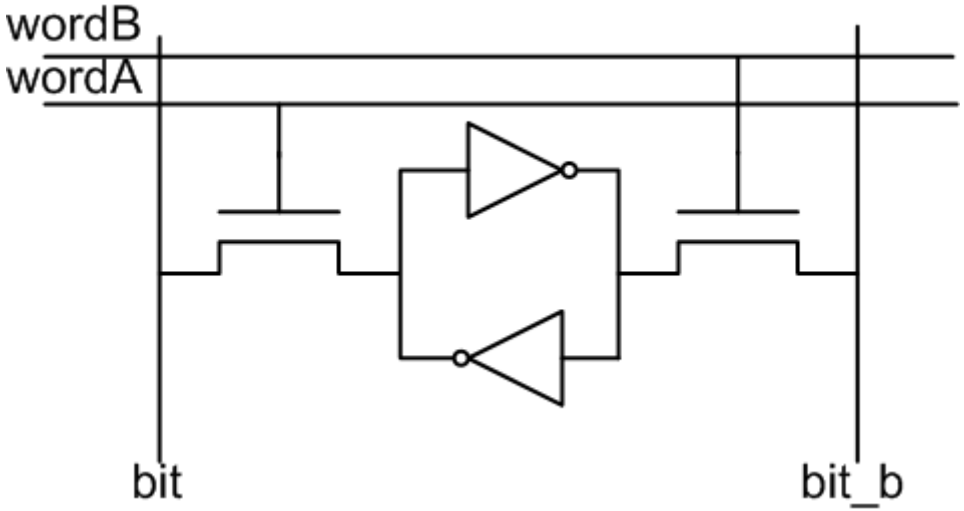
- Upis i čitanje istovremeno

Registarske memorije - istovremeni upis i čitanje





Registarske memorije - istovremeni upis i čitanje

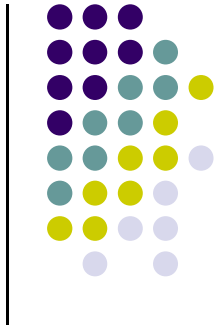


- Dva čitanja
  - Istovremeno
- Jedan upis
- Precizan tajming omogućava dva čitanja i jedan upis u jednoj periodi signala takta

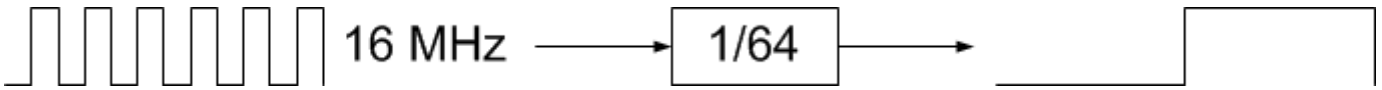
Brojači

- Sekvencijalna kola (KA) koja na svojim izlazima ciklično ponavljaju neki niz vrednosti
- Primeri:
  - Binarni brojač: 000, 001, 010, ...
  - Grejev kod: 000, 010, 110, 100 ...
  - Kružni brojač: 0001, 0010, 0100, ...
  - Pseudoslučajan brojač: 10, 00, 11, 01, 10, ...
  - ...
- Murova mreža sa prstenastom strukturom

Brojači - Primena



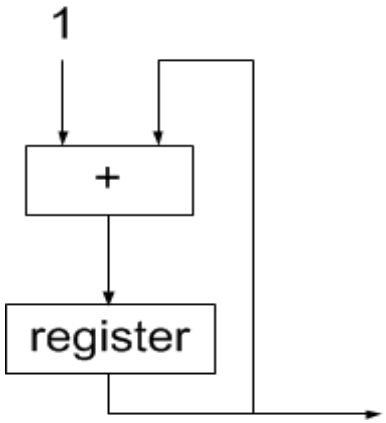
- Računanje koje se vrši u hardveru obično sadrže petlje
  - Množači
  - Bit-serijska komunikacija
- Deljenje frekvencije



- Inspecija nekih struktura podataka
  - npr. Parser IP paketa
- Uprošćenje kontrolnih jedinica
  - Omogućavanje da se akcija izvršava određeni broj ciklusa
  - Generisanje vremenski kontrolisanih signala

Brojači - Šema

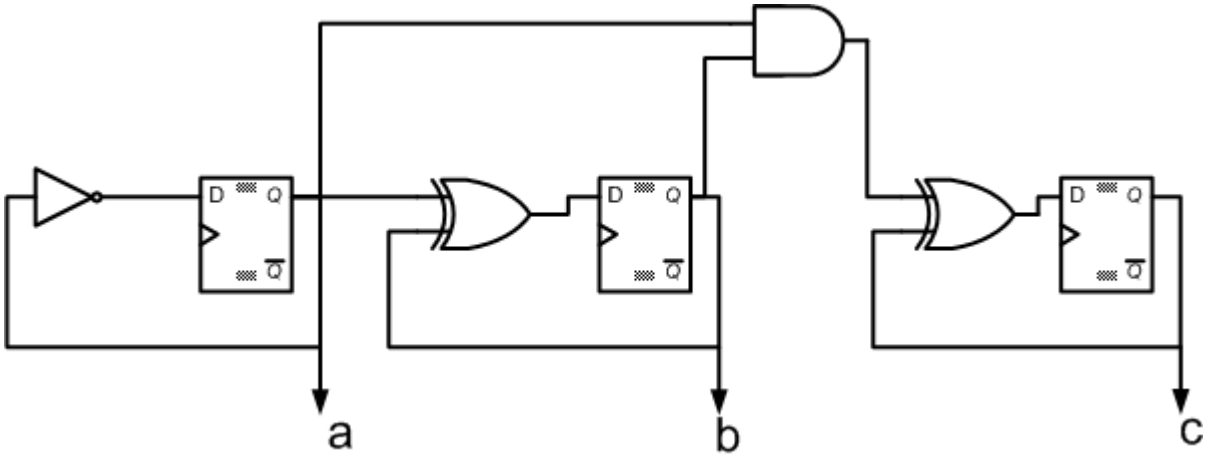
- Binarne brojač - kolo za inkremeniranje i registar



- U VHDL-u, brojač se specificira kao: x <= x + 1;
  - Ne generiše se potpuni sabirač
  - Kolo za inkrementiranje je prostije od sabirača
  - Brojač može biti još prostiji
- Može se koristiti optimizacija za dizajniranje brojača

Brojači – Binarni brojač

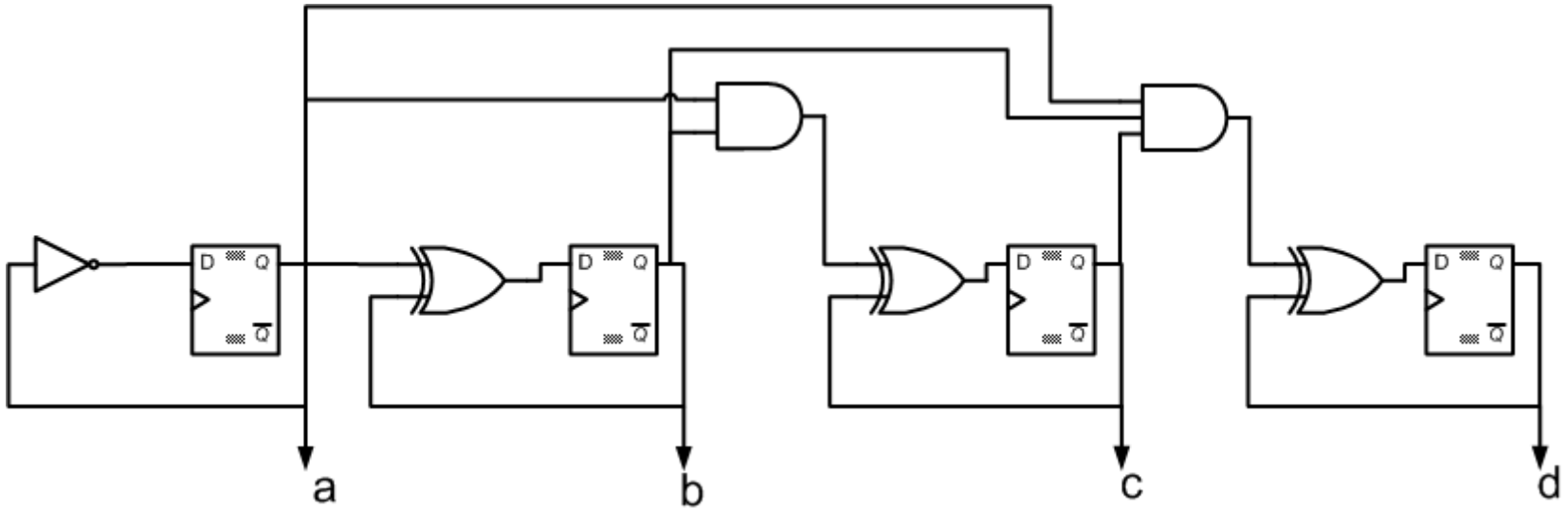
- Dizajn:
  - Početi sa 3-bitnom verzijom i generalizovati
  - a’ = not a
  - b’ = a xor b
  - c’ = c xor (a and b)



c	b	a	c’	b’	a’
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Brojači – Binarni brojač

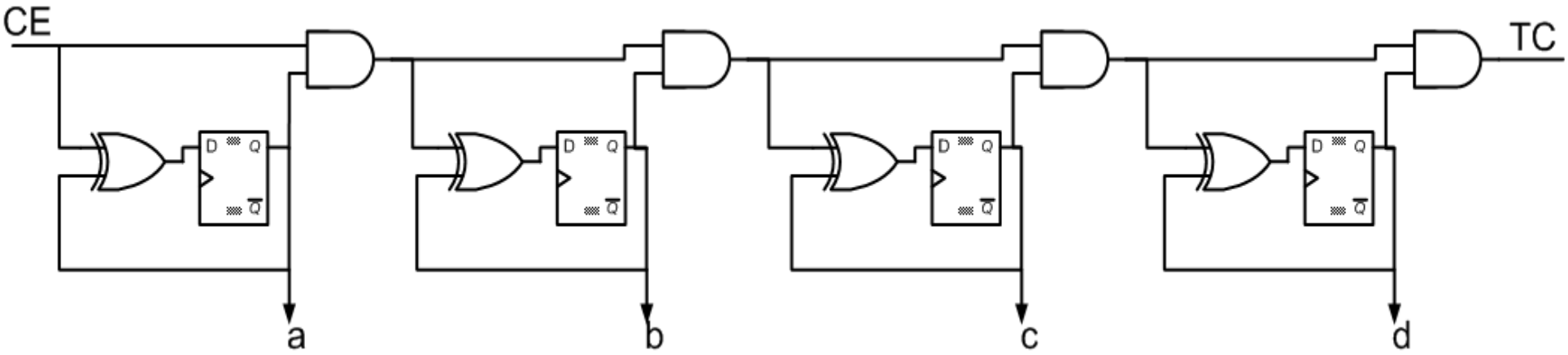
- Kako se proširuje na N bita?
  - Ekstrapolacijom:
  - d’ = d xor (c and b and a)
  - e’ = e xor (d and c and b and a)
  - ...



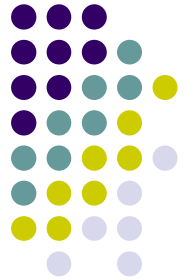
- Dobra strana: kritični put je N-bitni AND i XOR
- Loša strana: veličina hardvera raste relativno brzo sa N

Brojači – Binarni brojač

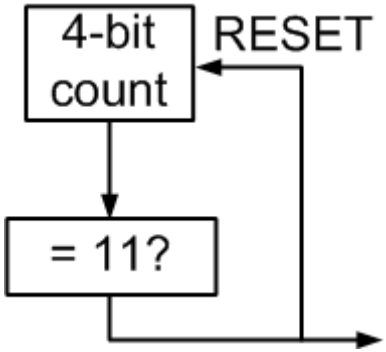
- CE – count enable, dozvoljava brojanje
- TC – terminal count, na aktivnoj vrednosti kada se u brojaču nalazi najveća vrednost
- Prednosti i mane?



Brojači – Binarni brojač



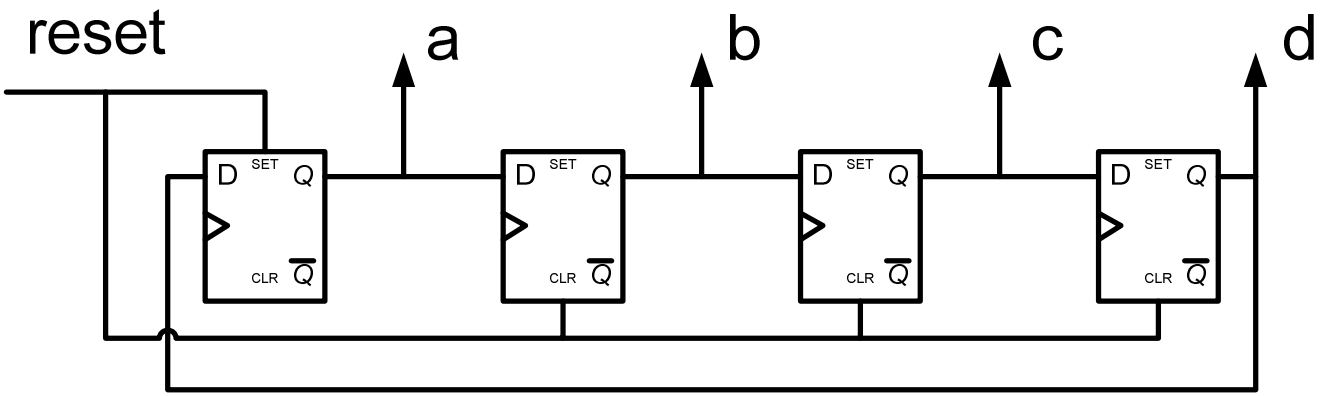
- Dodatna kombinaciona logika može da omogući završetak brojanja pre maksimalne vrednosti
- Npr.: brojač do 12



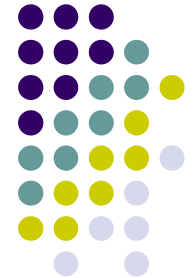
- Alternativa?

Brojači – Kružni brojač

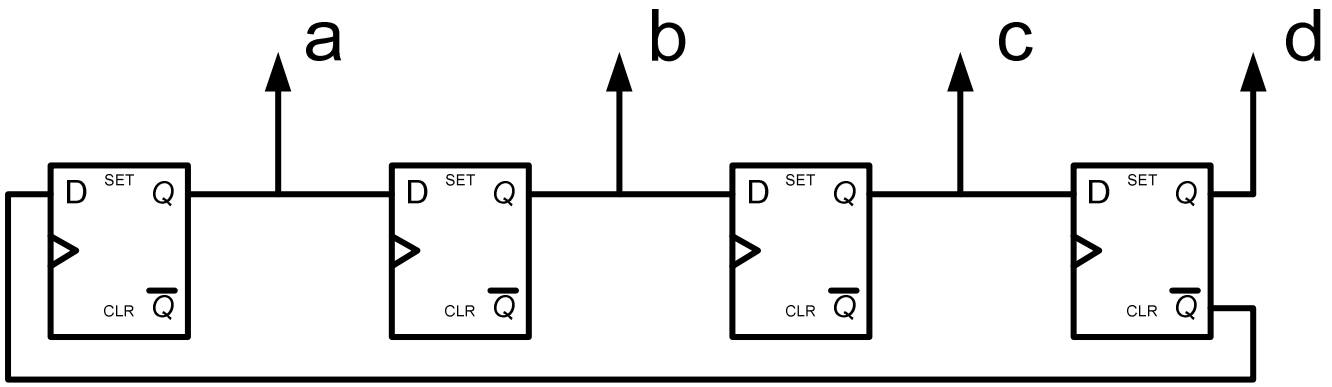
- Sekvenca: 0001, 0010, 0100, 1000, 0001
- Namena?



Brojači – Johnson brojač

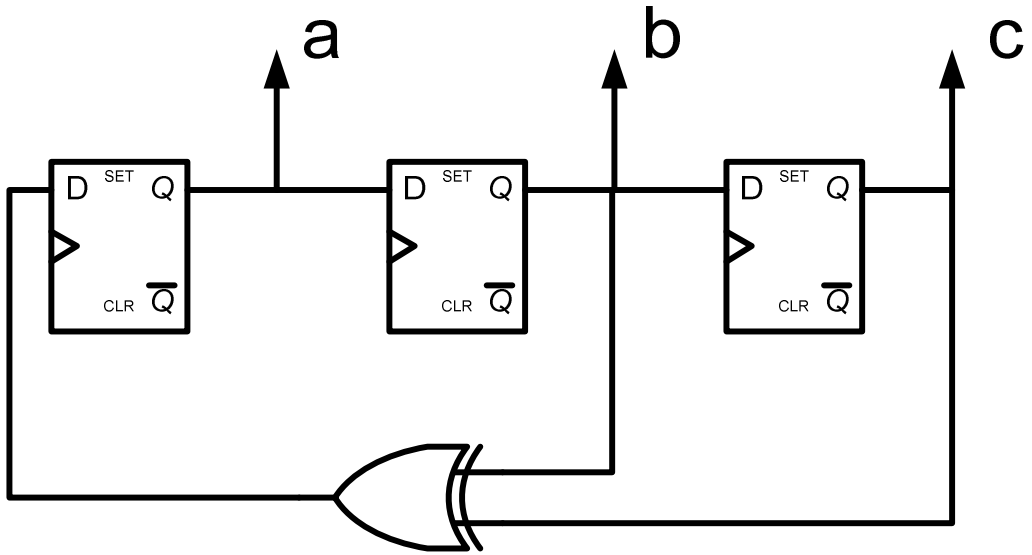


- Sekvenca: 0000, 0001, 0011, 0111, 1111, 1110, 1100, 1000, 0000
- Namena?

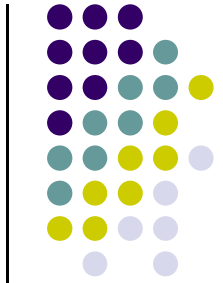


Brojači – Šift registar sa linearnom povratnom spregom

- Sekvenca: 111, 110, 100, 001, 010, 101, 011, 111
- Skup od $2^n - 1$ vrednosti
- Registar resetovati na nenultu vrednost



Sabirači



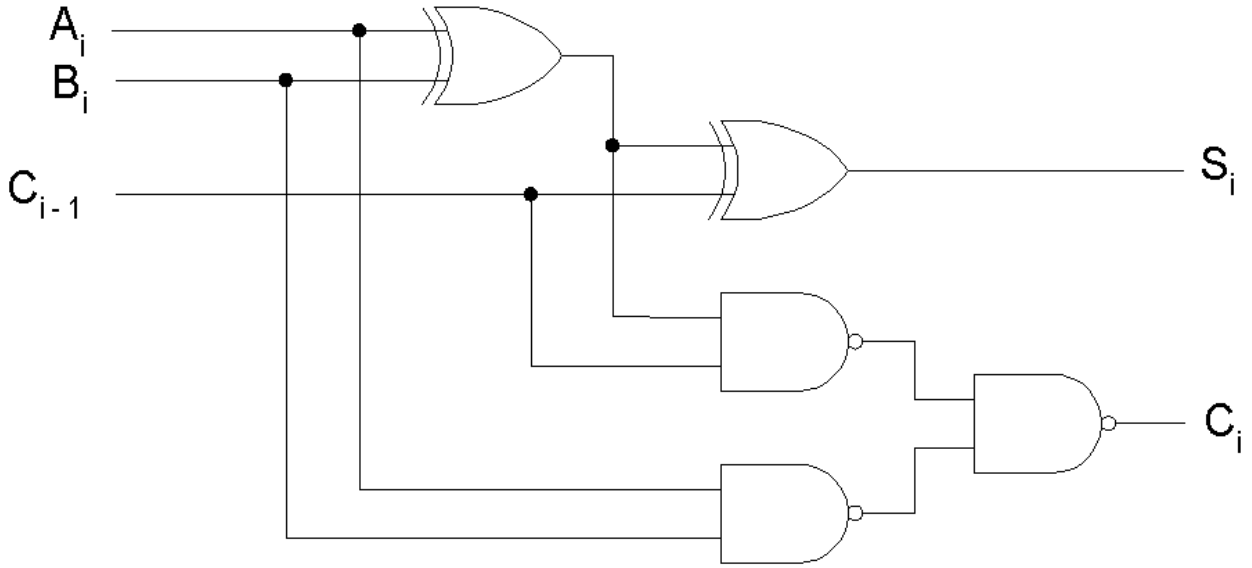
- Sa serijskim prenosom –
  Ripple Carry, RCA
- Sa uslovnim zbirovima –
  Conditional Sum Addres, CSumA
- Na osnovu bita prenosa –
  Carry Select Adder, CSelA
- Sa paralelnim prenosom –
  Carry Lookahead Adder, CLA
- Sa čuvanjem prenosa –
  Carry Save Addrer, CSavA

Ripple Carry Adder

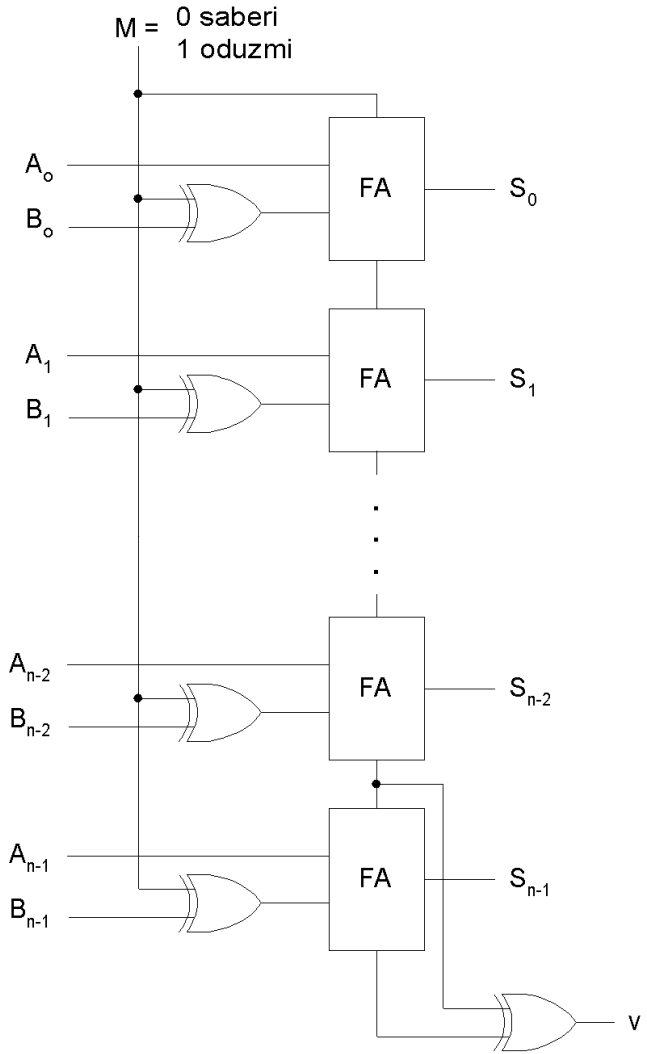
- Osnovne formule za sabiranje dva binarna broja A i B (potpun sabirač):

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

$$C_i = A_iB_i + C_{i-1}(A_i + B_i)$$



Ripple Carry Adder



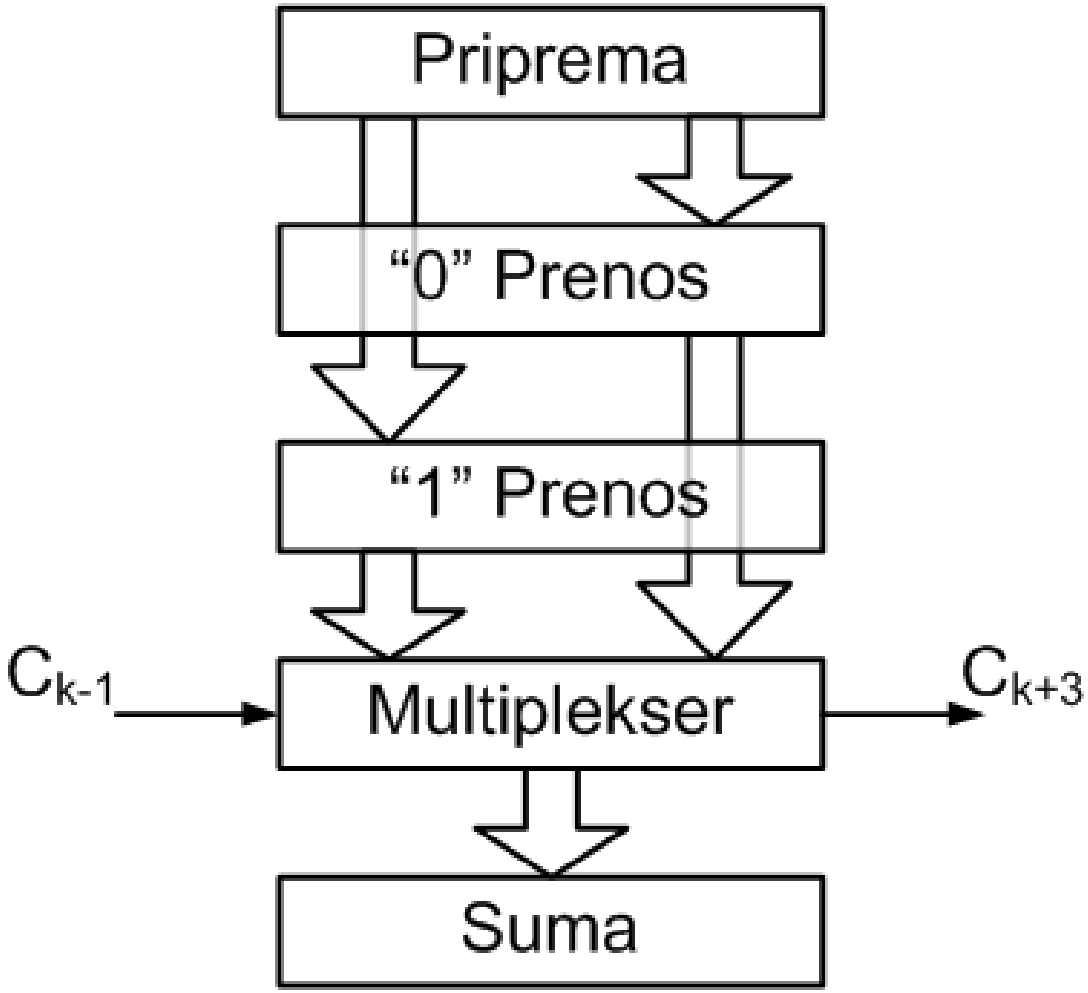
Prednosti:

- Kola sa malim brojem ulaza
- Jednostavna i regularna struktura

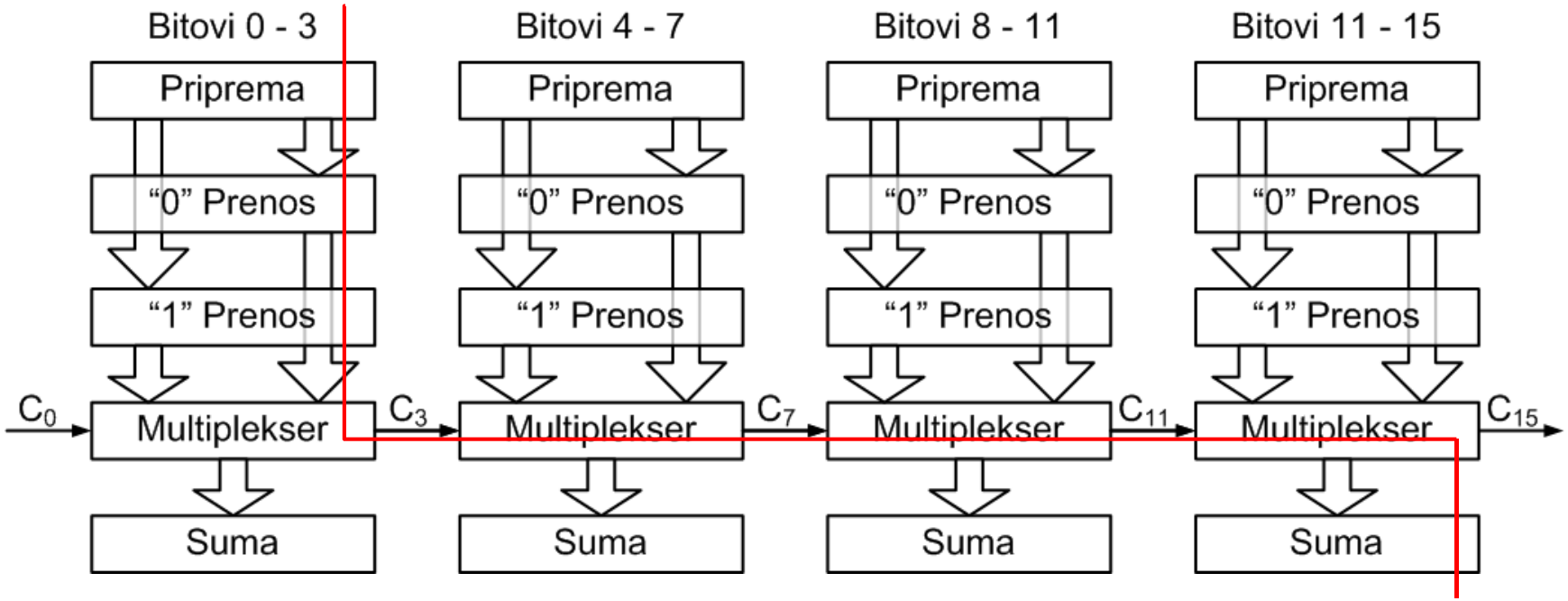
Mane:

- Veliko kašnjenje, srazmerno broju razreda

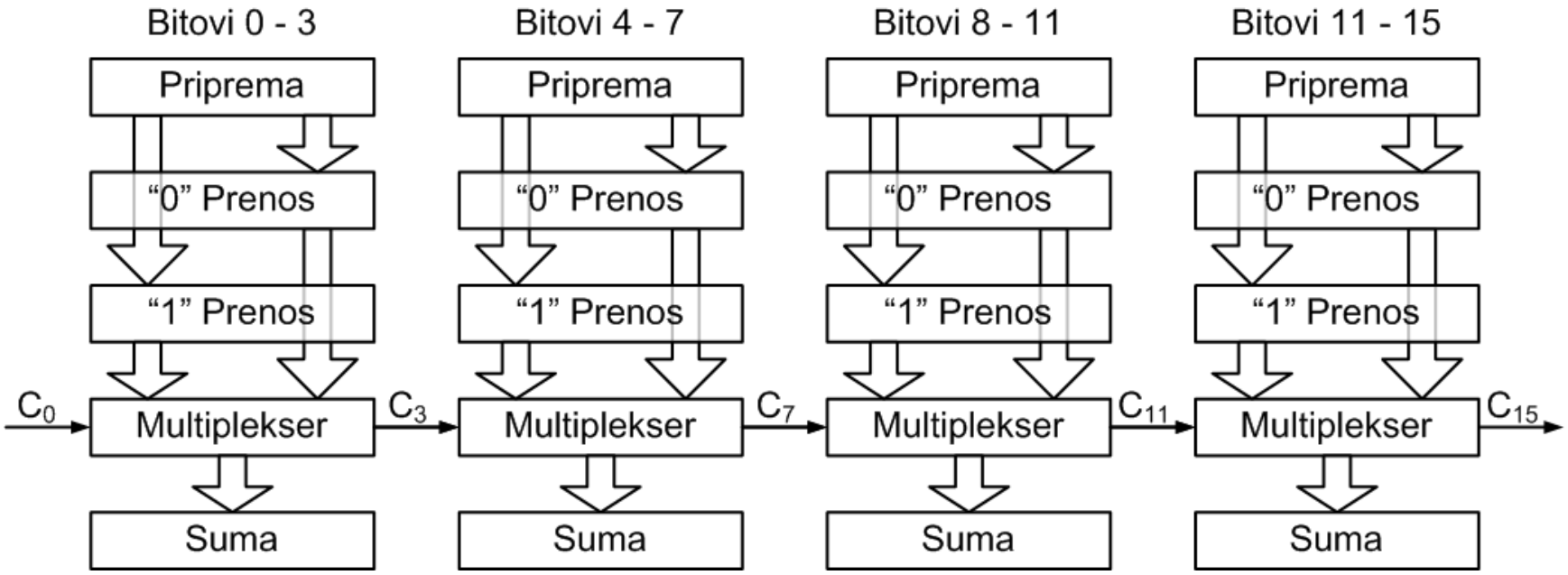
Carry Select Adder



Carry Select Adder – Kritični put

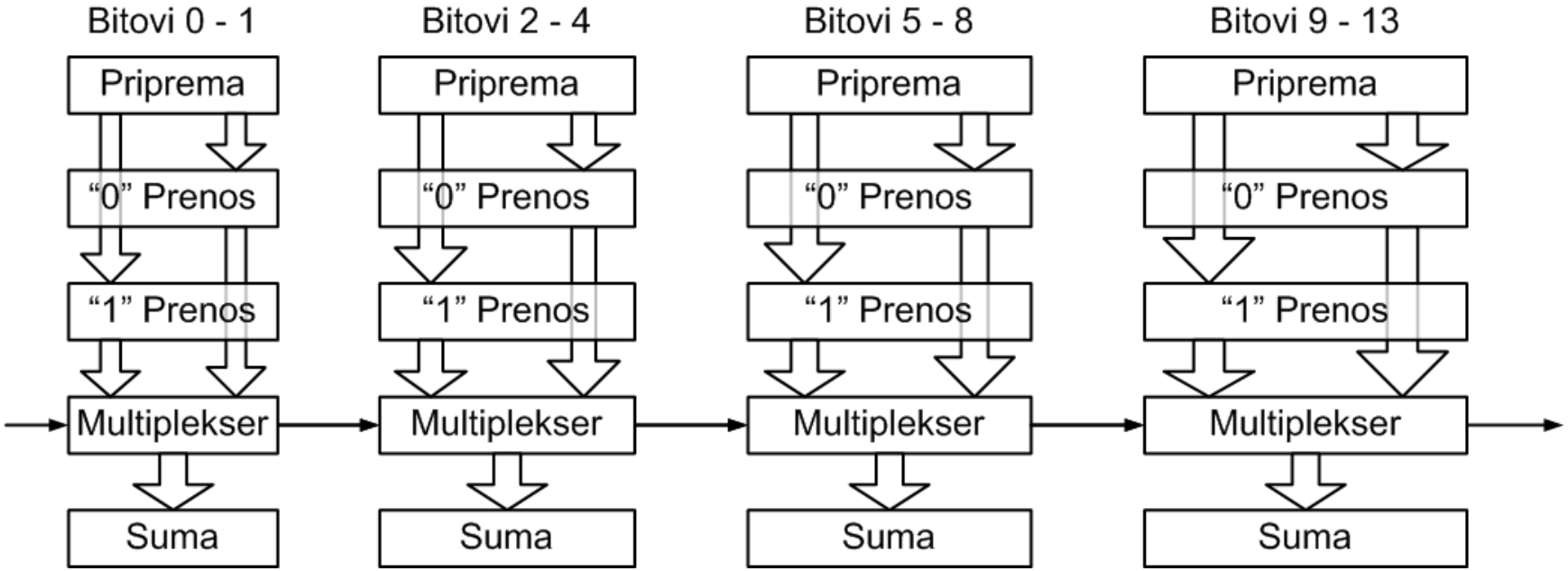


Carry Select Adder – Kašnjenje



- $t_d = t_p + (\frac{N}{M})t_{fa} + M_{tmux} + t_{sum}$
  - N – broj ulaza
  - M – broj blokova

Carry Select Adder – Blokovi promenljive dužine

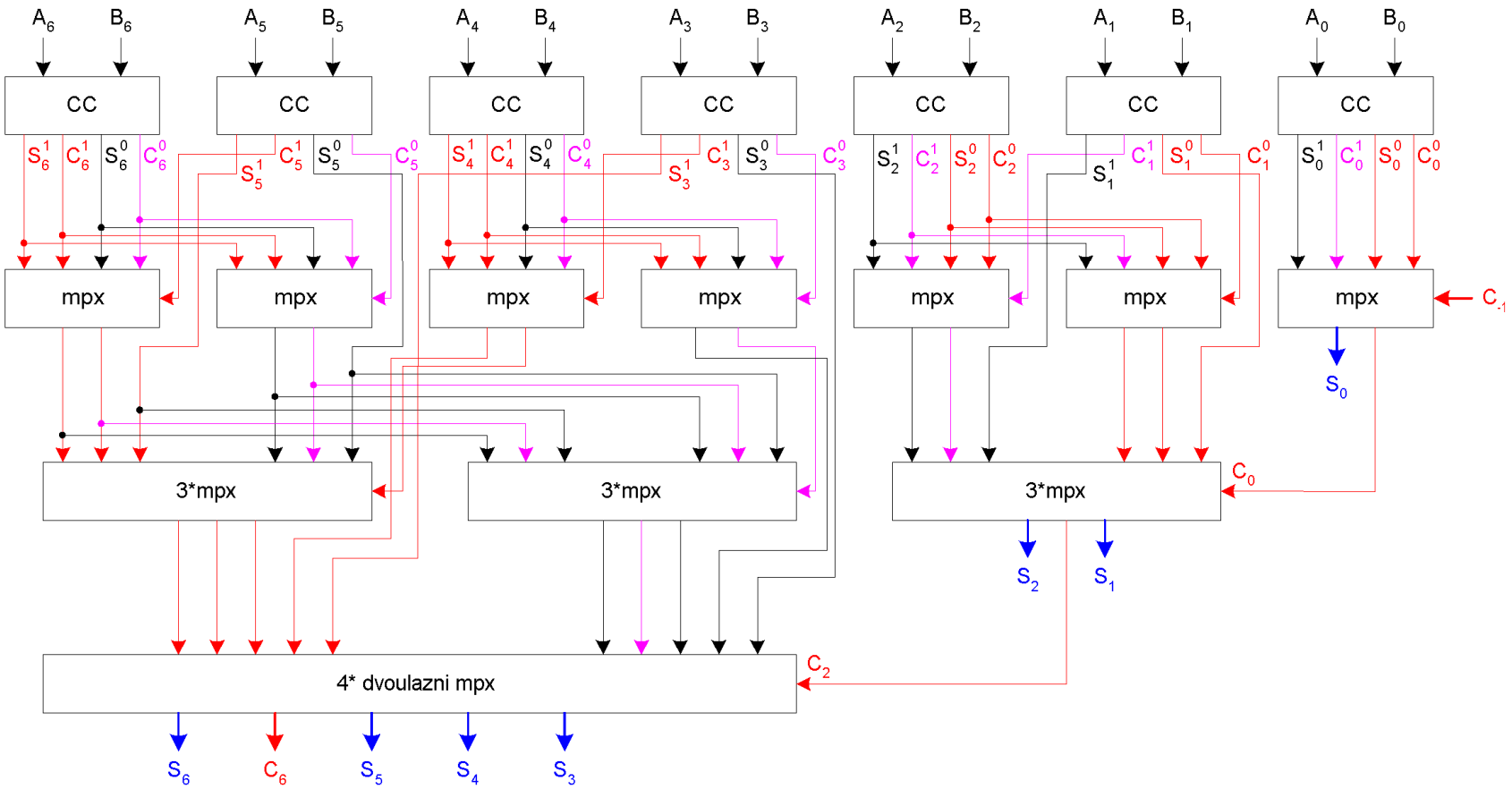


- $t_d = t_p + 2t_{fa} + (\sqrt{2N})t_{mux} + t_{sum}$
  - N – broj ulaza
  - M – broj blokova

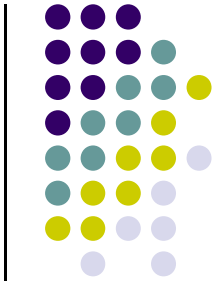
Conditional Sum Adder

- Formiraju se dva zbira sa prenosima za svaki razred:
  - jedan pod pretpostavkom da je ulazni prenos 0
  - drugi pod pretpostavkom da je ulazni prenos 1
- Grupišu se zbirovi po nivoima

Conditional Sum Adder



Conditional Sum Adder



i	6	5	4	3	2	1	0	prenos	korak
A_i	1	1	0	1	1	0	1		
B_i	0	1	1	0	1	1	0		
S_i	1	0	1	1	0	1	1	0	1
C_{i+1}	0	1	0	0	1	0	0		
S_i	0	1	0	0	1	0	0	1	
C_{i+1}	1	1	1	1	1	1	1		
S_i	0	0	1	1	0	1	1	0	2
C_{i+1}	1		0		1		0		
S_i	0	1	0	0	1	0		1	
C_{i+1}	1		1		1				
S_i	0	1	1	1	0	1	1	0	3
C_{i+1}	1				1				
S_i	0	1	0	0				1	
C_{i+1}	1								
Sum	0	1	0	0	0	1	1		

Uslovni sabirač (CC)

- Osnovne formule za sabiranje dva binarna broja A i B (potpun sabirač):

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

$$C_i = A_iB_i + C_{i-1}(A_i + B_i)$$

- Rezultat kada je prenos ulazni prenos 0:

$$S_i^0 = A_i \oplus B_i = (A_i + B_i)\,(\overline{A}_i + \overline{B}_i) = (A_i + B_i)\,\overline{(A_iB_i)}$$

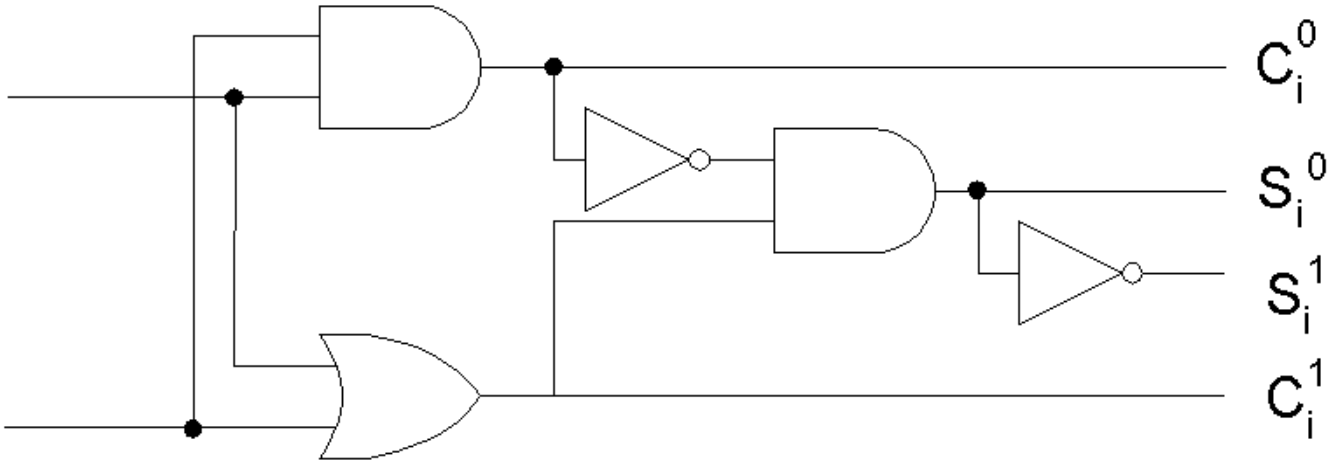
$$C_i^0 = A_iB_i$$

- Rezultat kada je prenos ulazni prenos 1:

$$S_i^1 = A_i \oplus B_i \oplus 1 = \overline{A_i \oplus B_i} = \overline{S_i^0}$$

$$C_i^1 = A_iB_i + (A_i + B_i) = A_i + B_i$$

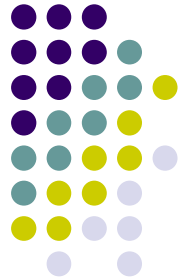
Uslovni sabirač (CC)



Conditional Sum Adder

- **Prednosti**:
  - Kola sa malim brojem ulaza
  - Manje kašnjenje nego kod RCA
- **Mane**:
  - Veliki broj elemenata

Carry Lookahead Adder



- Svi biti prenosa formiraju se paralelno:

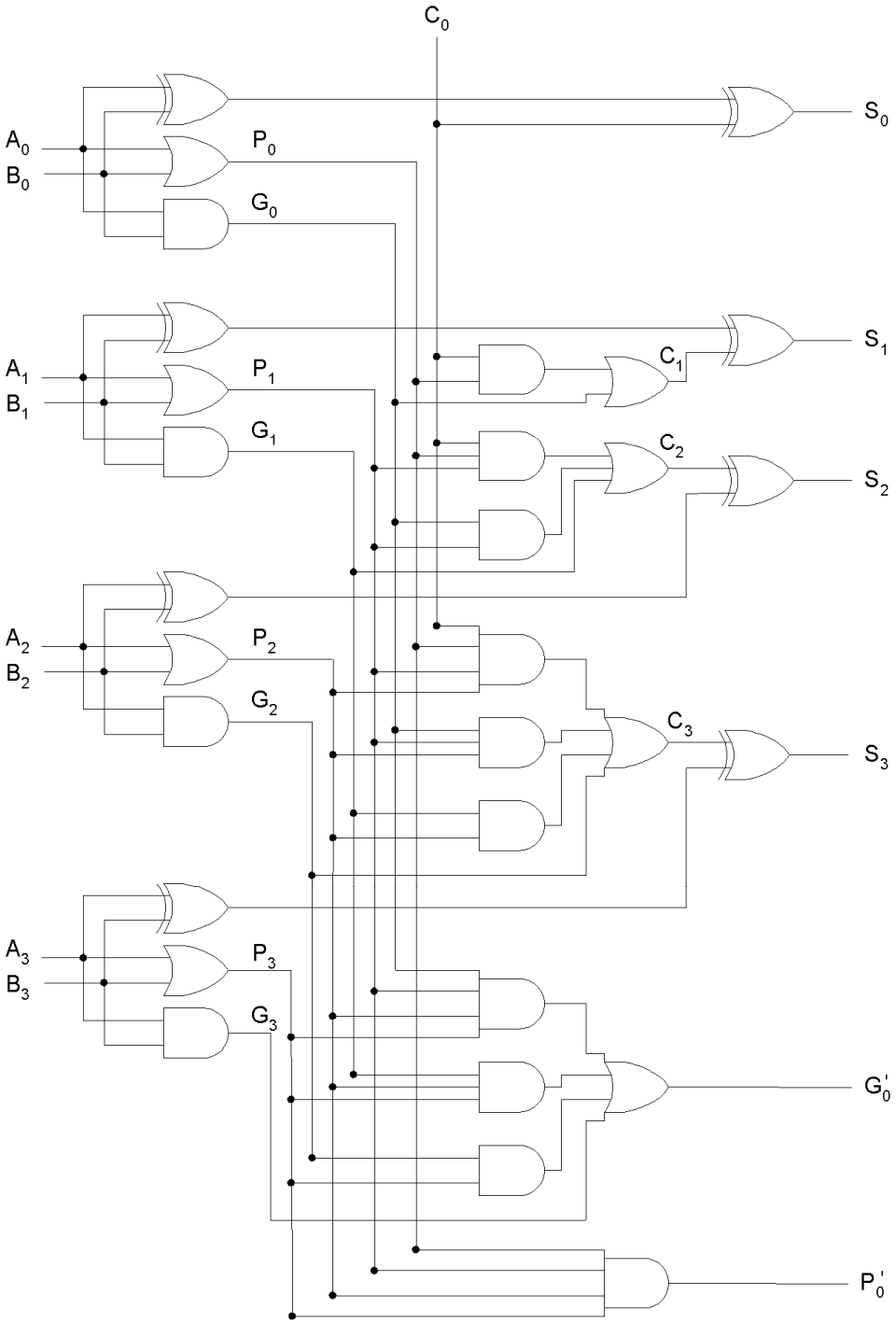
$$C_{i+1} = A_iB_i + C_i(A_i + B_i) = G_i + C_iP_i$$

$$G_i = A_iB_i$$

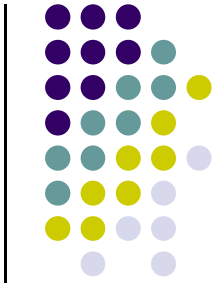
$$P_i = A_i + B_i$$

- Rekurzivnim razvojem formule:

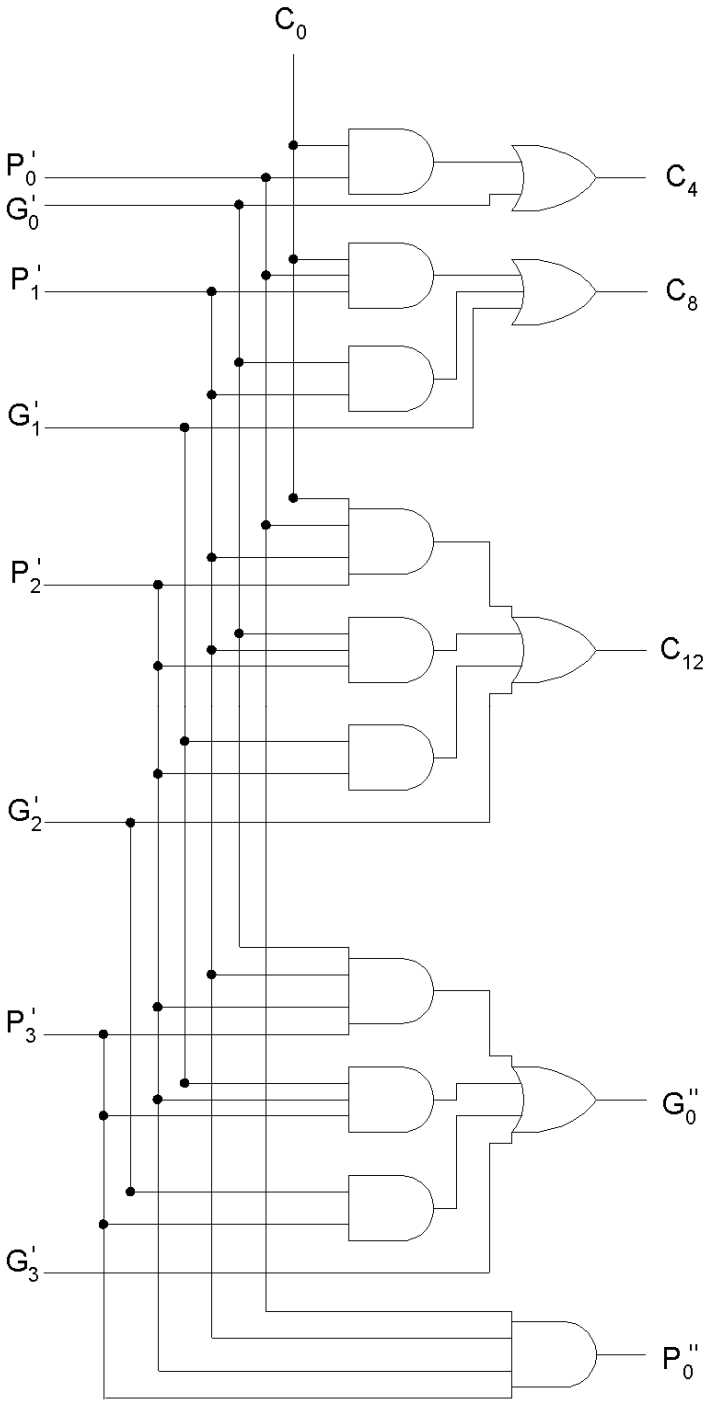
$$C_{i+1} = G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + \ldots + P_iP_{i-1}\ldots P_iC_0$$



Carry Lookahead Adder



- **Prednosti**:
  - Malo vreme propagacije
- **Mane**:
  - Za veći broj bita veliki je broj ulaza kod I i ILI kola
  - Da bi se otklonio ovaj nedostatak koristi se generator prenosa



Generator prenosa

- Prenosi:

$C_4 = G_0 + P_0C_0$

$C_8 = G_1 + P_1G_0 + P_1P_0C_0$

$C_{12} = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

- Prenos za sledeći nivo (nije dio ovog generatora, već se generiše u narednom nivou):

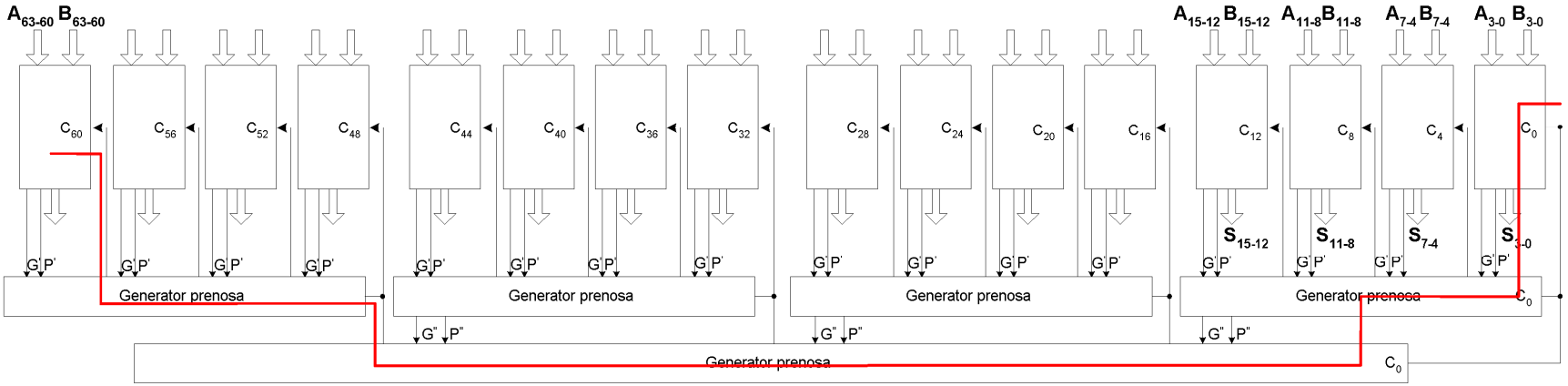
$C_{16} = (G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0) + (P_3P_2P_1P_0)C_0$

- Za dalje računanje:

$G'' = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$

$P'' = P_3P_2P_1P_0$

CLA 64 bita

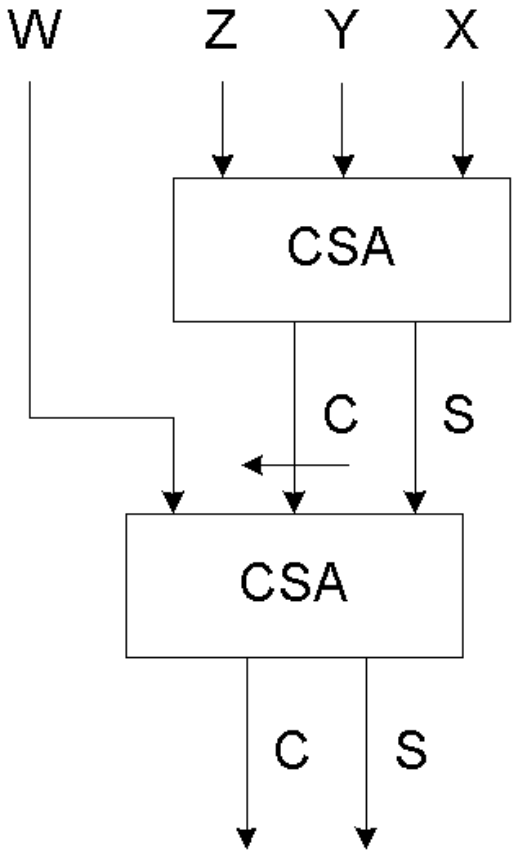


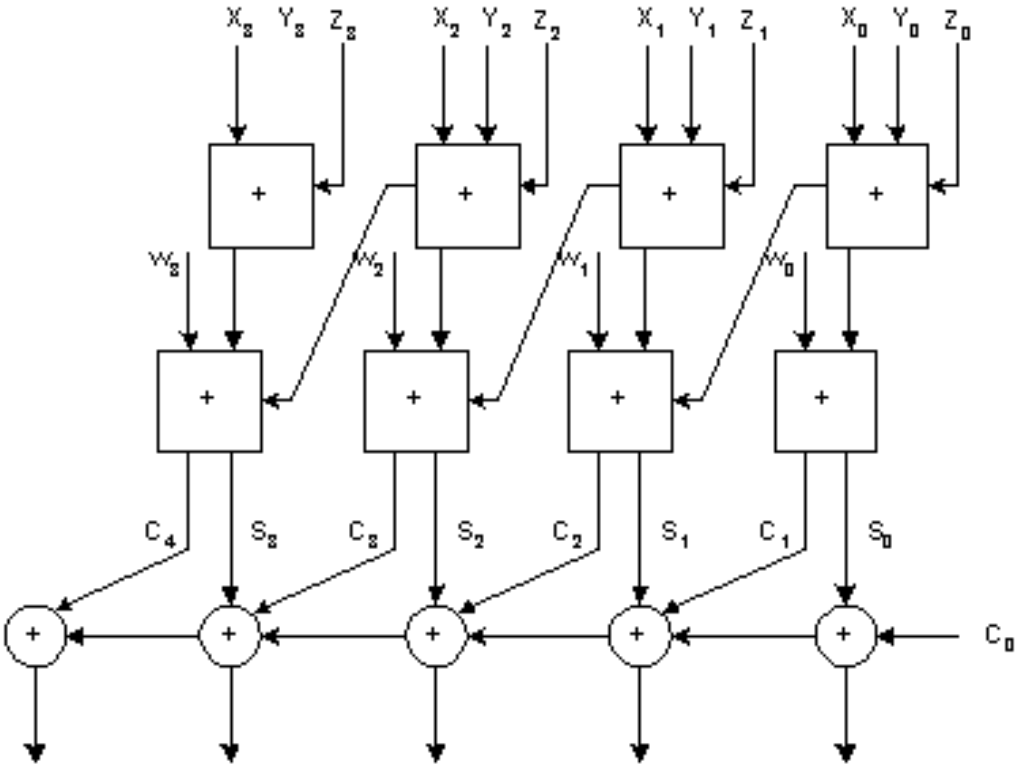
- Ukupno kašnjenje:

$$T = (3 + 2(2N - 3) + 3)t = 4Nt = 4t*\log_r n$$

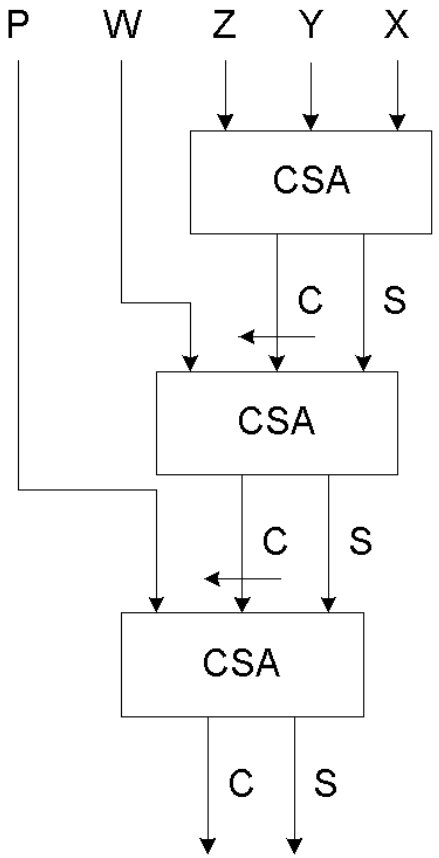
- N – broj CLA nivoa = $\log_r n$
- t – kašnjenje jednog logičkog kola
- r – broj razreda CLA elementa (u ovom slučaju 4)
- n – broj razreda operanada (u ovom slučaju 64)

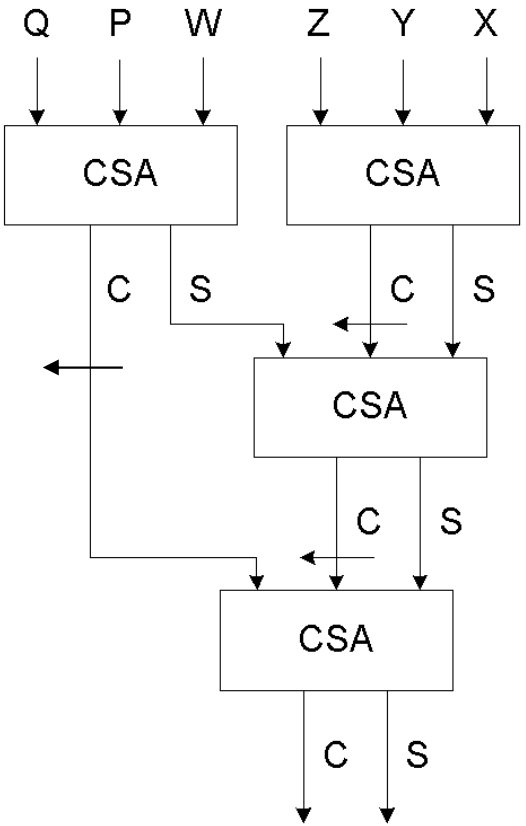
Carry Save Adder

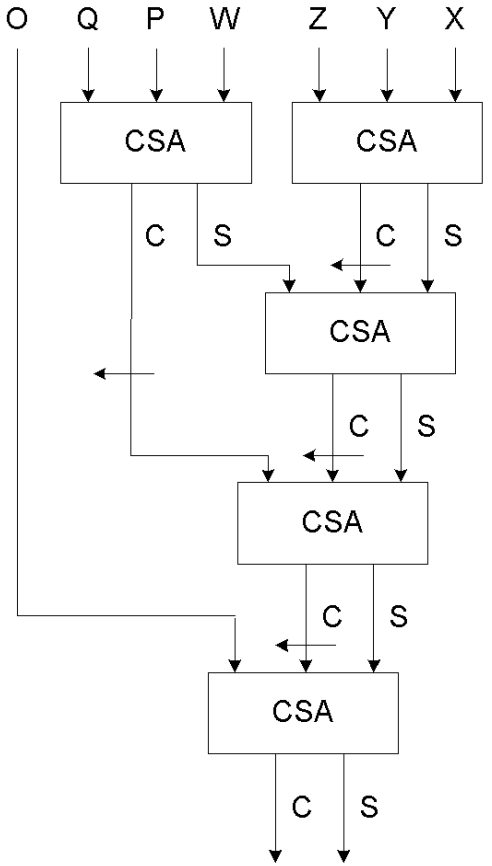




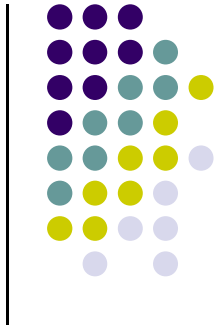
Carry Save Adder







Pomerači

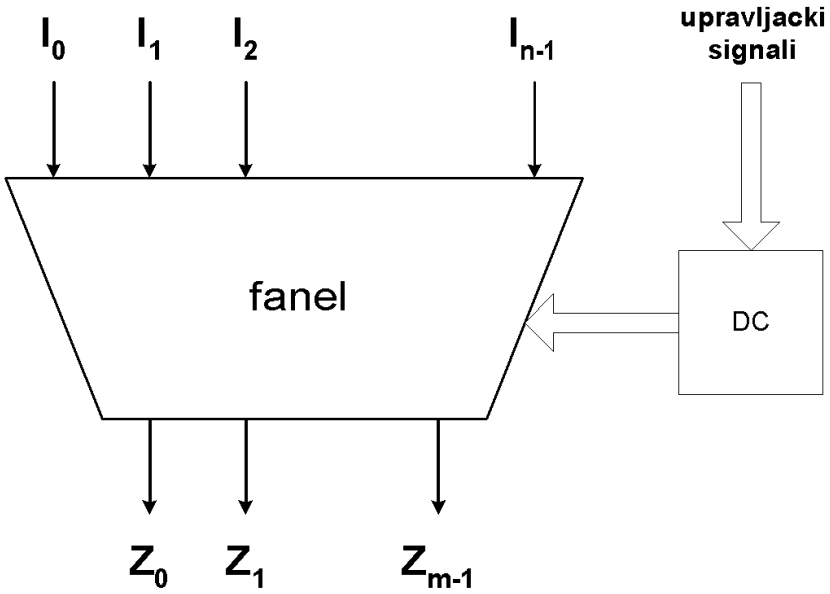


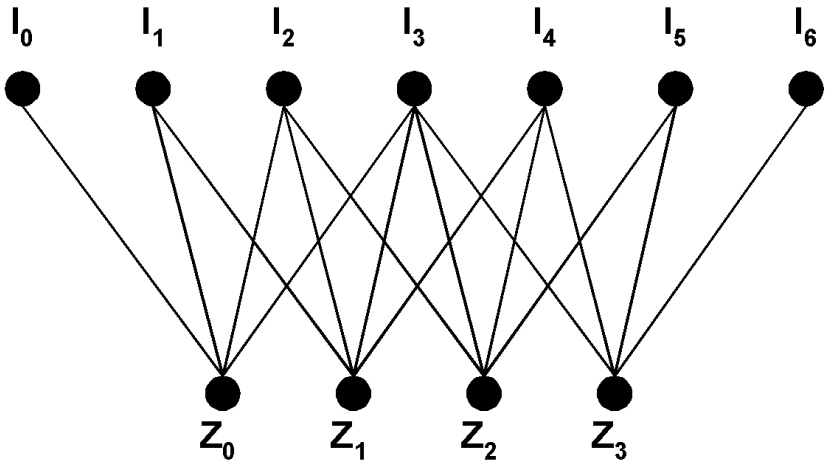
- **Sekvencijalni**:
  - Koriste pomerački registar
  - Najčešće pomera za jedno mesto u jednom takt intervalu
- **Kombinacioni**:
  - Vrše pomeranje za proizvoljan broj mesta u jednom takt intervalu

Kombinacioni pomerači

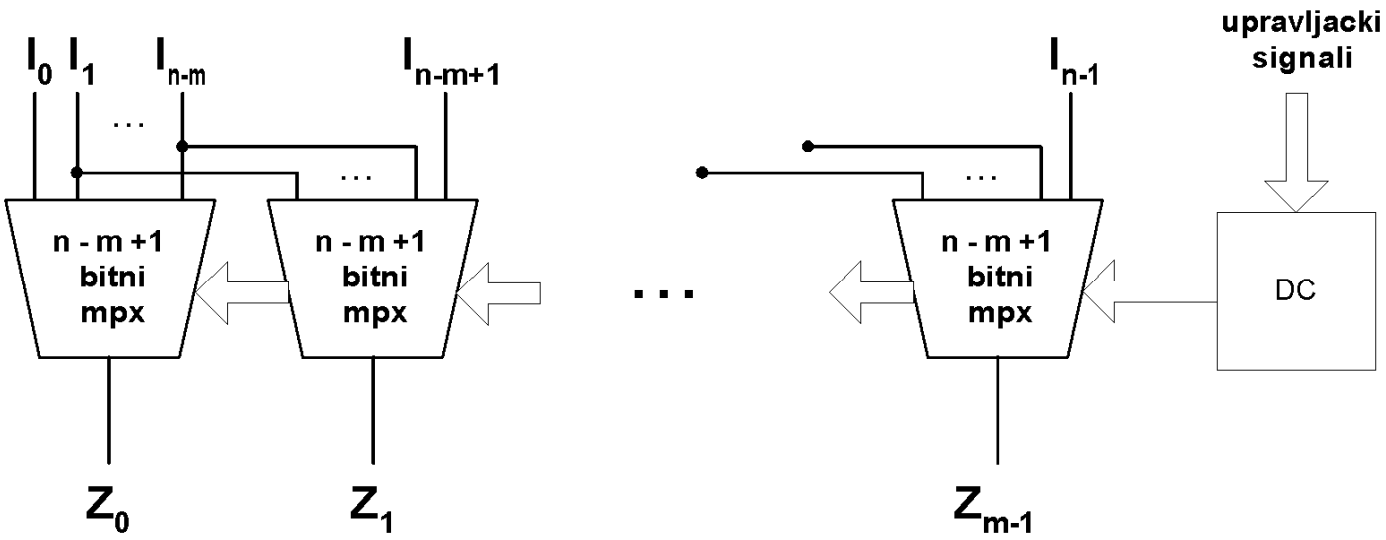
- **Fanel** (*funnel*):
  - Ima n ulaznih i m izlaznih signala, gde je $n > m$
- **Barel** (*barrel*)
  - Pomerač koji može da rotira ulazni podatak za proizvoljan broj mesta
  - Ima isti broj ulaznih i izlaznih signala

Fanel pomerač

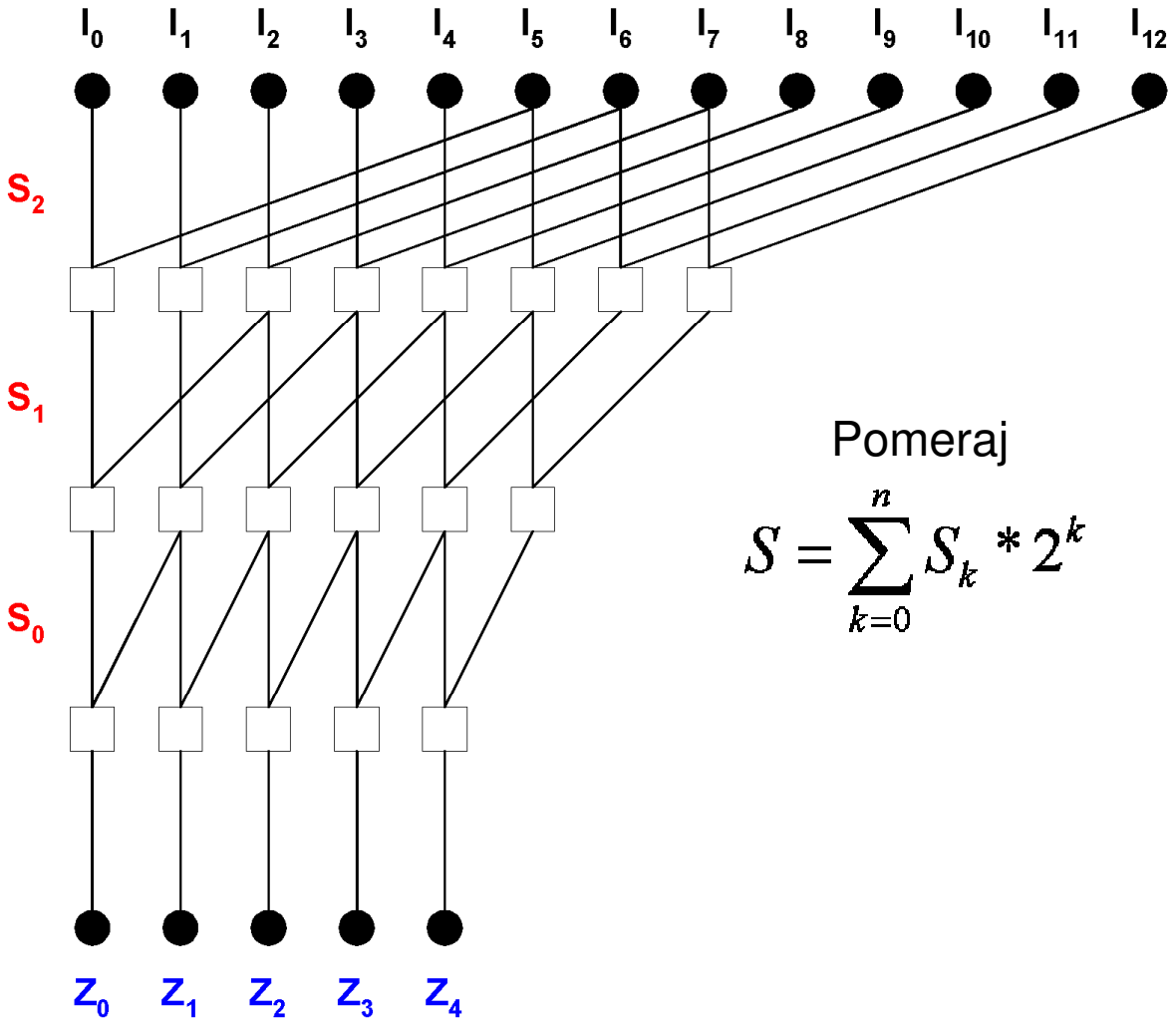




Fanel pomerač



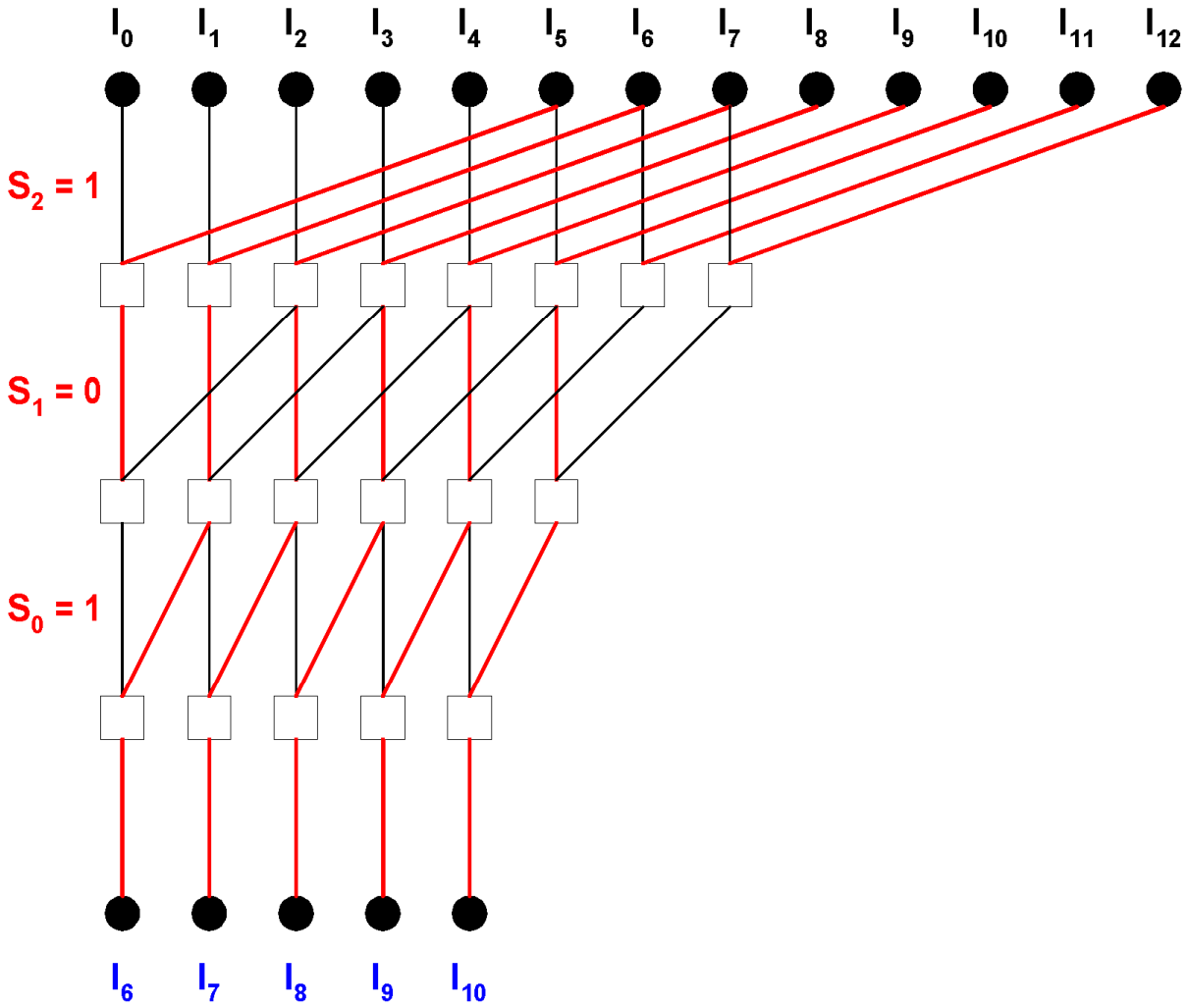
Višestepeni fanel pomerač



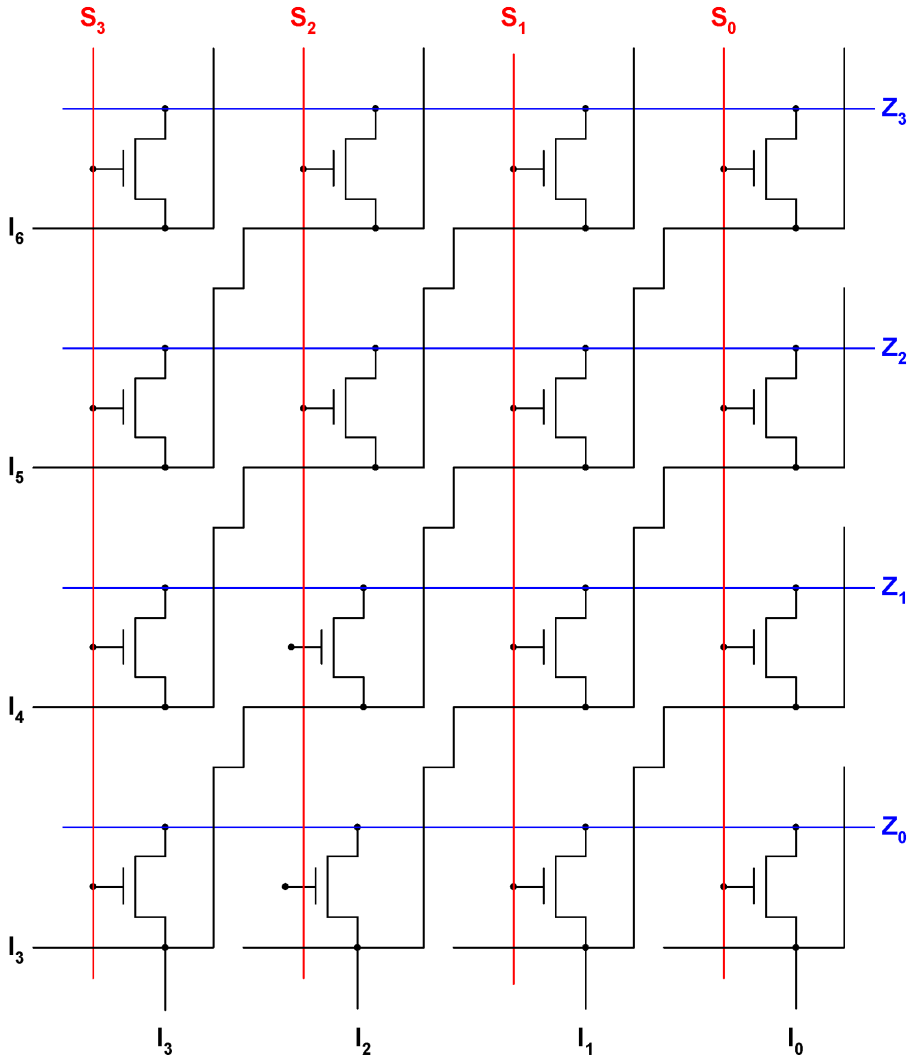
Pomeraj

$$S = \sum_{k=0}^{n} S_k * 2^k$$

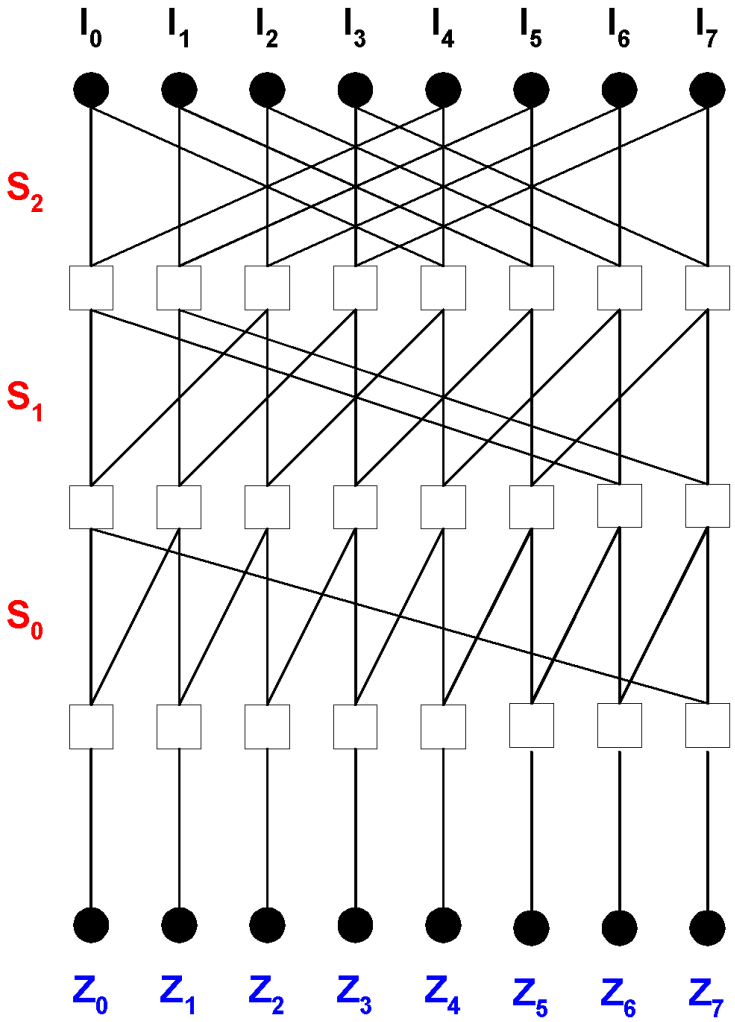
Višestepeni fanel pomerač – pomeranje za 5 mesta



Fanel pomerač – NMOS realizacija



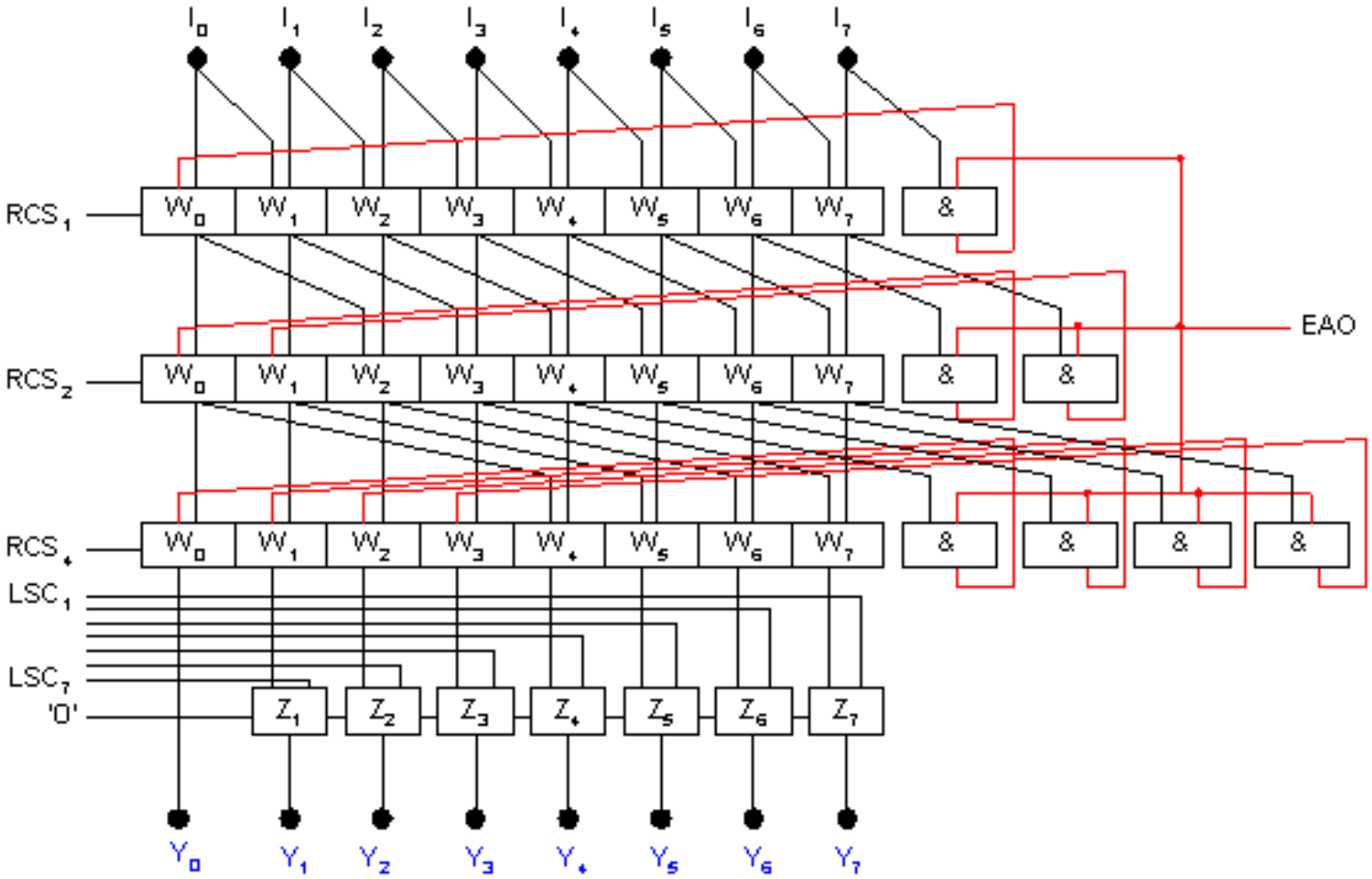
Višestepeni barel pomerač



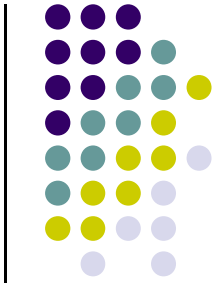
Pomeraj

$$S = \sum_{k=0}^{n} S_k * 2^k$$

Višestepeni Barel pomerač

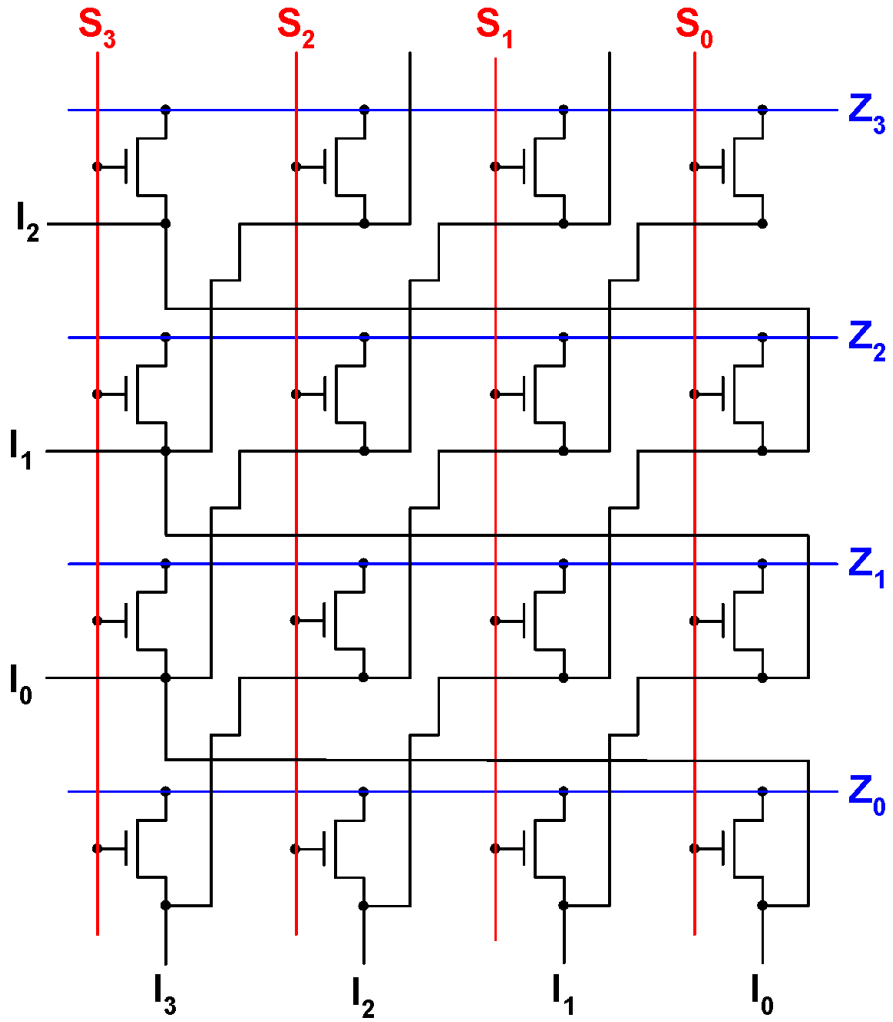


Višestepeni Barel pomerač

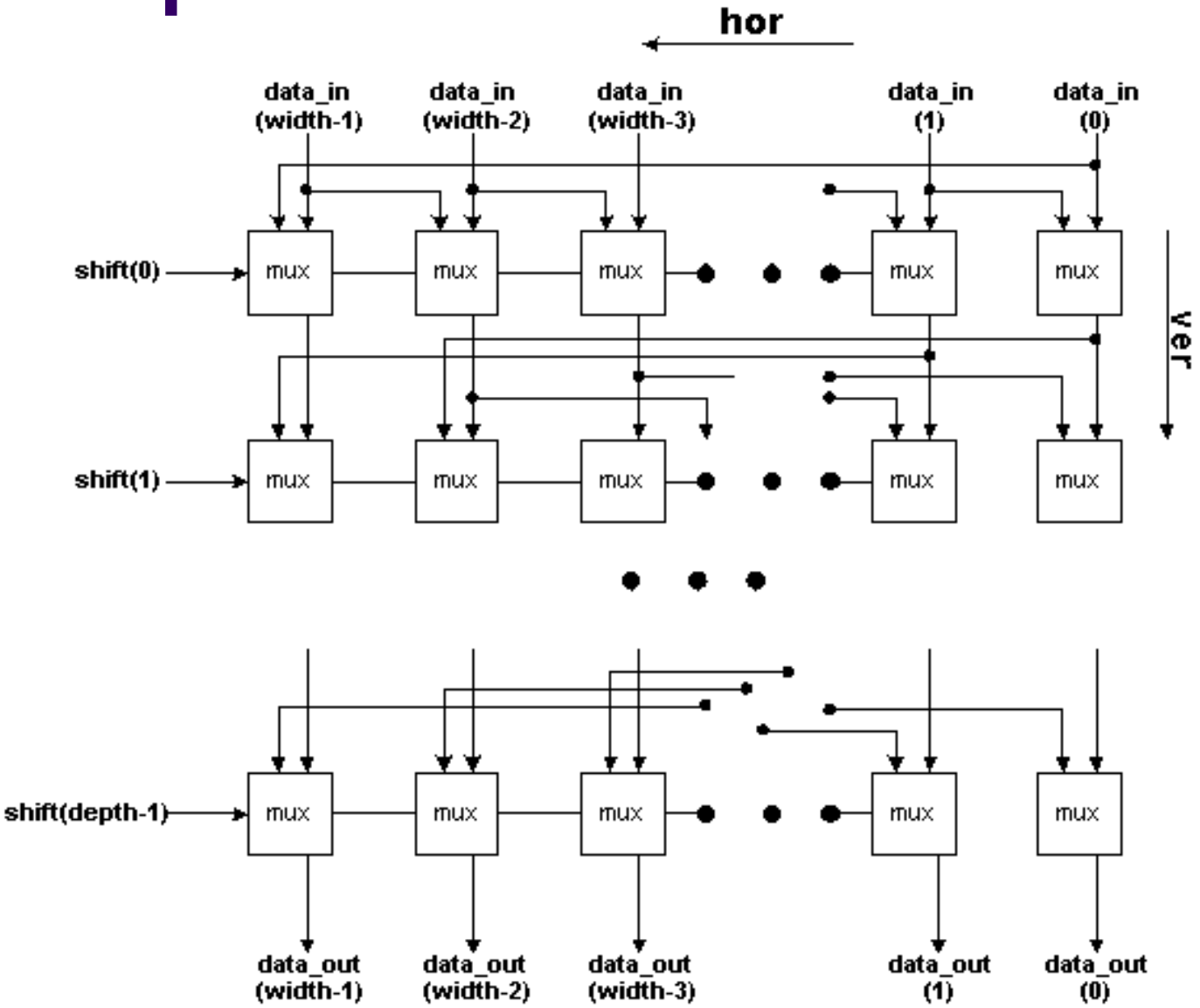


	EAO	LSC	RSC	AR
Rotacija levo	1	0	n	0
Rotacija desno	1	0	8-n	0
Logičko pomeranje levo	0	0	n	0
Logičko pomeranje desno	1	LSCn = 1	8-n	0
Aritmetičko pomeranje levo	0	0	n	1
Aritmetičko pomeranje desno	1	LSCn = 1	8-n	1

Barel pomerač – NMOS realizacija

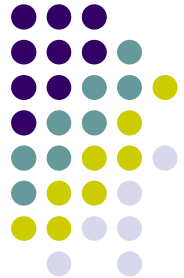


Barel pomerač - blok šema



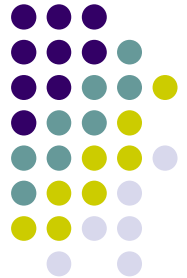
Barel pomerač - VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity barrel is
   generic (
        depth: natural := 3;
        width: natural := 8
        );
   port (
        data_in : in  std_logic_vector (width-1 downto 0);
        shift   : in  std_logic_vector (depth-1 downto 0);
        data_out: out std_logic_vector (width-1 downto 0)
        );
end entity barrel;
```



Barel pomerač - VHDL

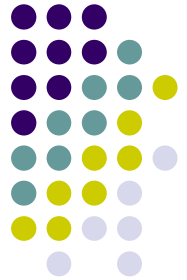
```
architecture structural of barrel is
   component mux_2 is
   port (
        data_in1,
        data_in2,
        sel      : in  std_logic;

        data_out: out std_logic
        );
   end component mux_2;

   type array_of_vectors is array (0 to depth-2) of
        std_logic_vector(width-1 downto 0);

   signal out_ver_hor: array_of_vectors;
```



Barel pomerač - VHDL

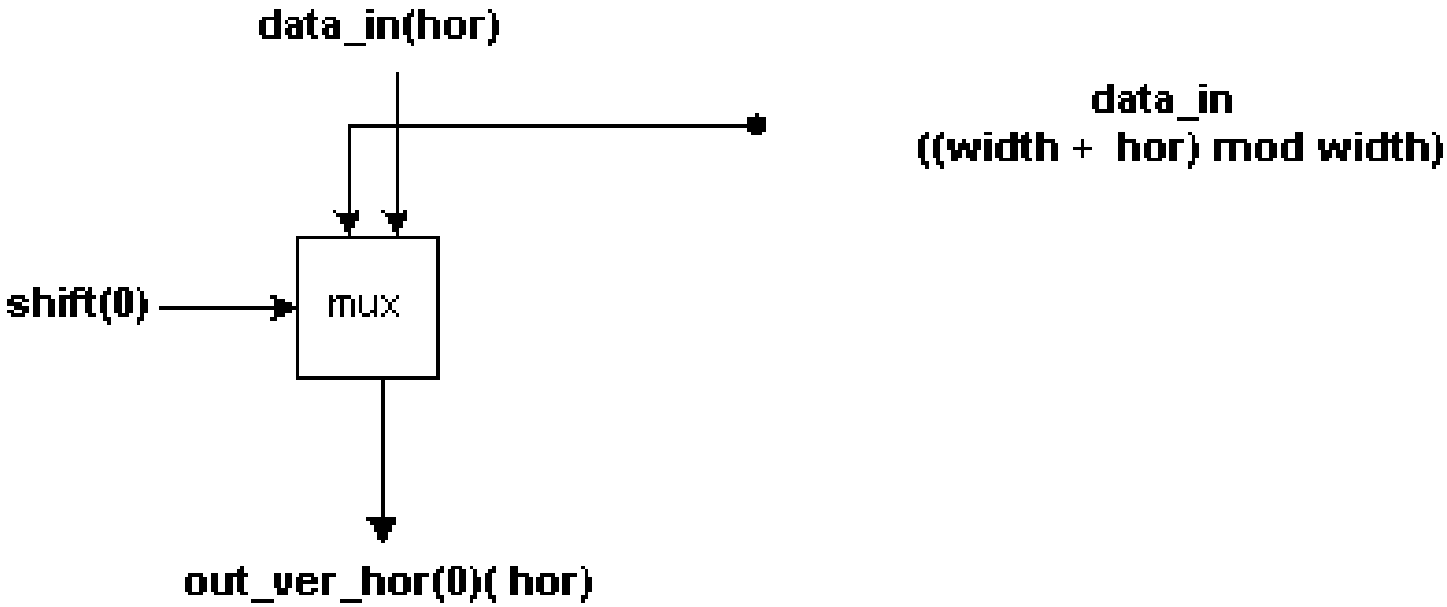
```
begin
   vertical: for ver in 0 to depth-1 generate
   begin
        horizontal: for hor in width-1 downto 0 generate
        begin

        . . .

        end generate horizontal;
   end generate vertical;
end architecture structural;
```

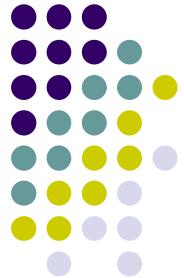
Uslovne strukture



- Razlikuju se prvi, poslednji i redovi u sredini
- Koristi se uslovno generisanje

Barel pomerač – Izgled ćelije u prvom redu

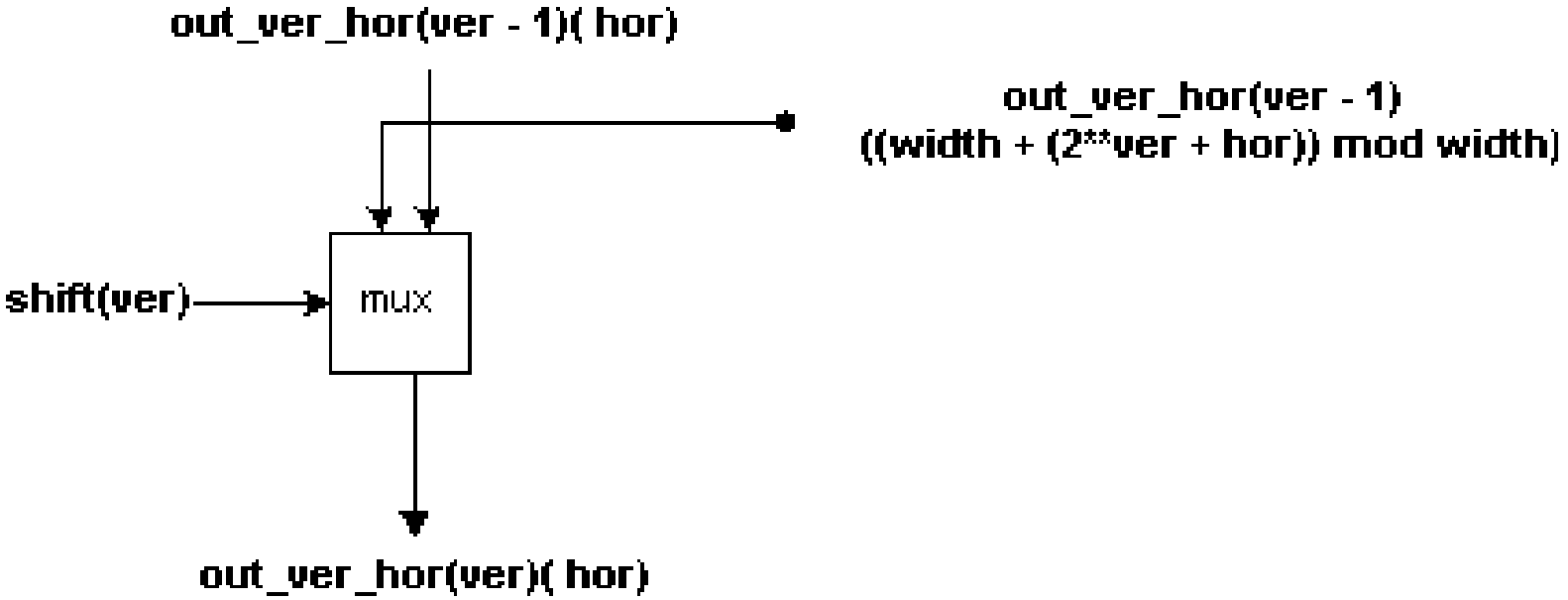


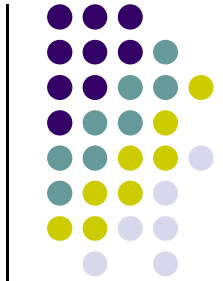


Barel pomerač - VHDL

```
first_line: if ver = 0 generate
begin
   mux: mux_2
   port map (
        data_in1  => data_in(hor),
        data_in2  =>   data_in((width +
                       (1 + hor)) mod width),
        data_out  => out_ver_hor(0)(hor),
        sel       => shift(0)
   );
end generate first_line;
```

Barel pomerač – Izgled ćelije u sredini

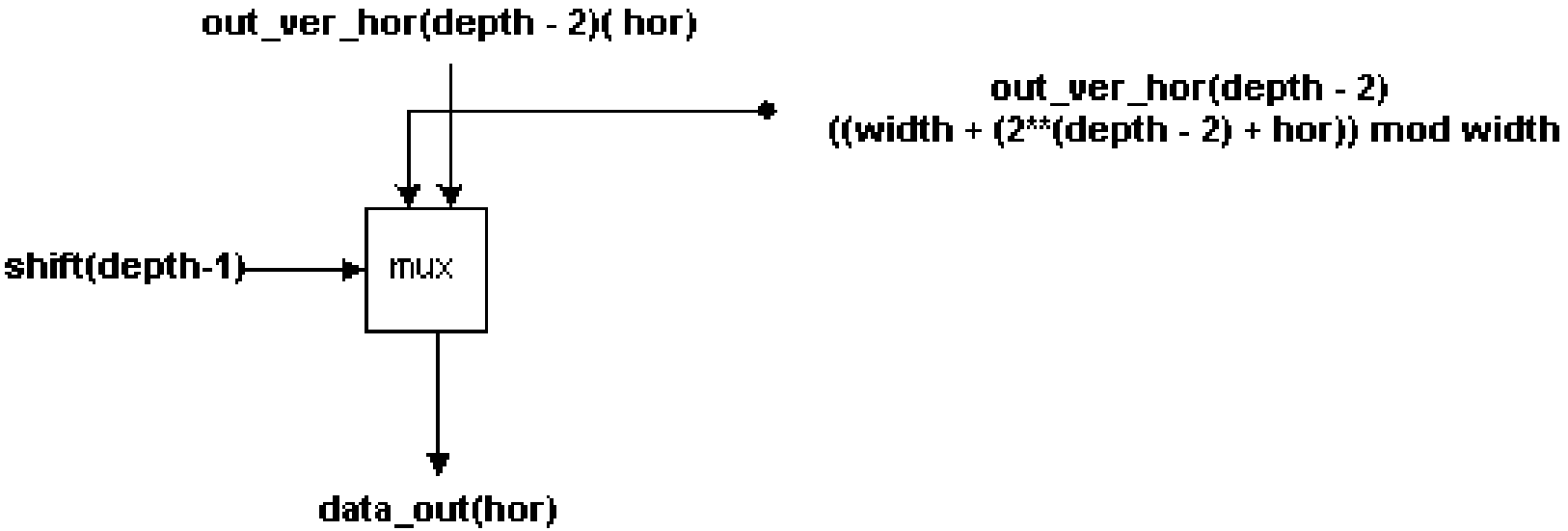


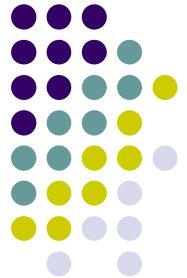


Barel pomerač - VHDL

```
middle_line: if ver > 0 and ver < depth-1
   and depth > 2 generate
begin
   mux: mux_2
   port map (
        data_in1  => out_ver_hor(ver-1)(hor),
        data_in2  => out_ver_hor(ver-1)((width +
                  (2**ver + hor)) mod width),
        data_out  => out_ver_hor(ver)(hor),
        sel       => shift(ver)
   );
end generate middle_line;
```

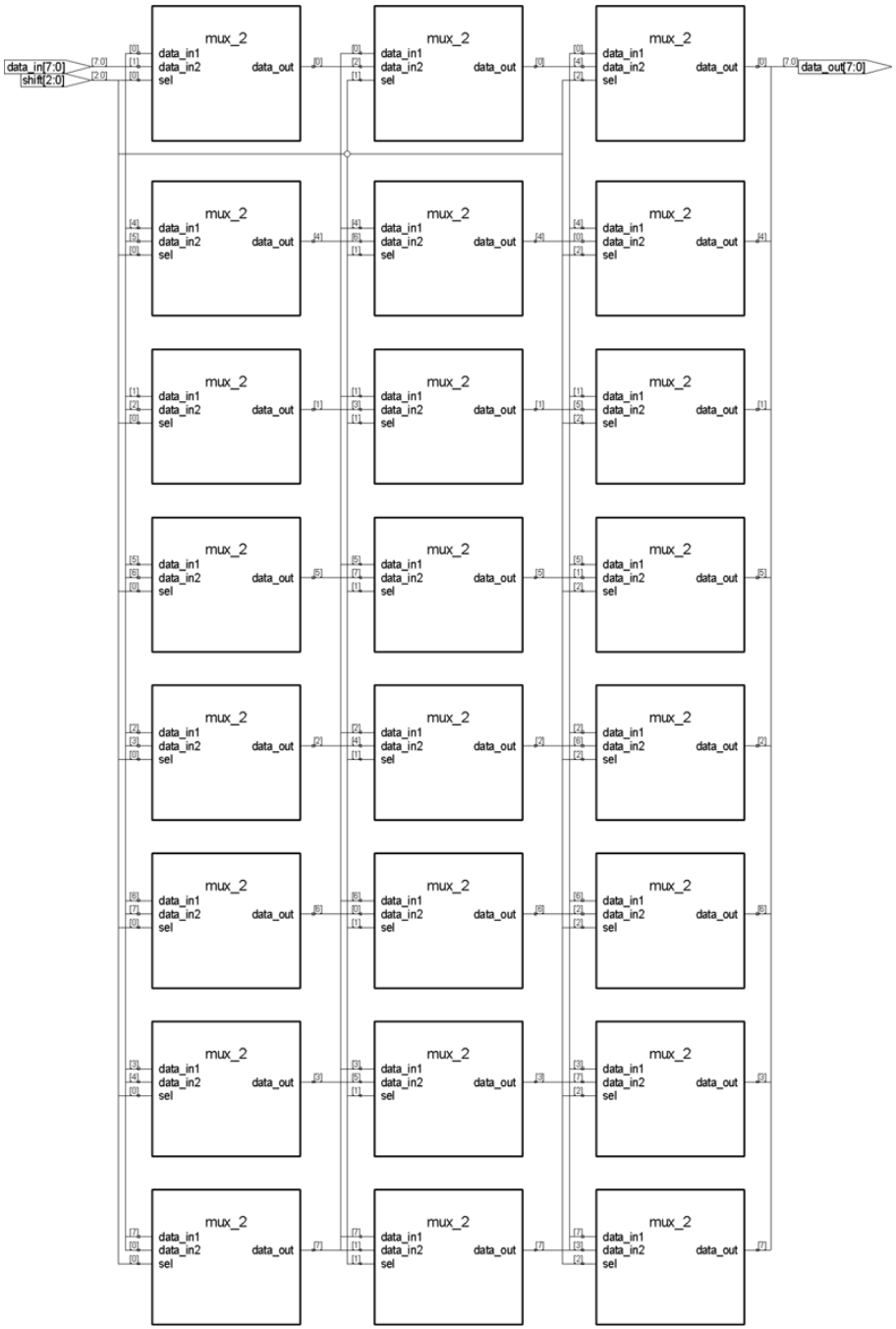
Barel pomerač – Izgled ćelije u poslednjem redu





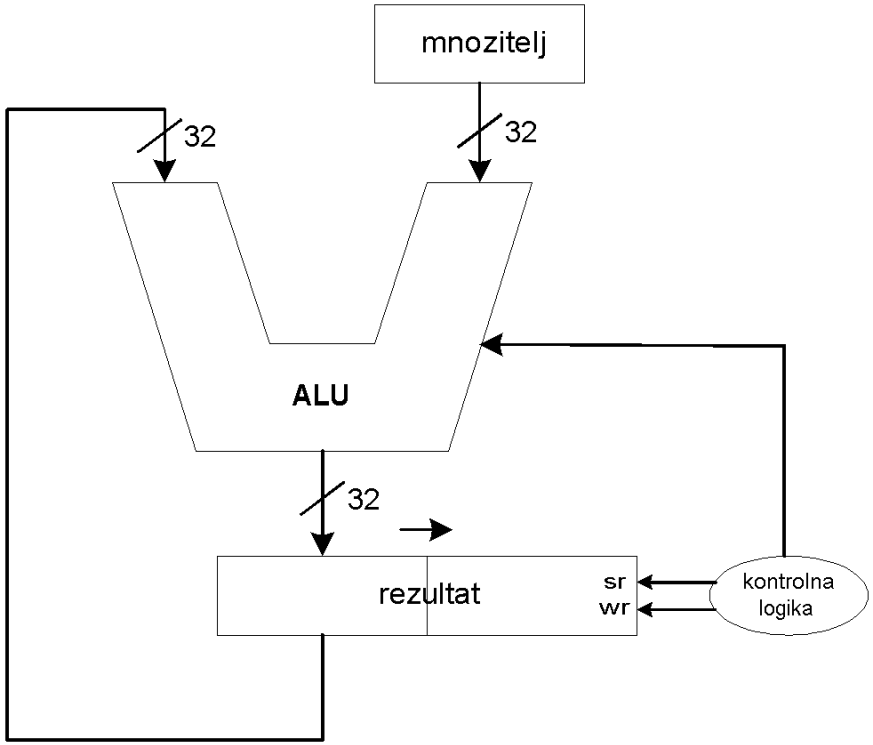
Barel pomerač - VHDL

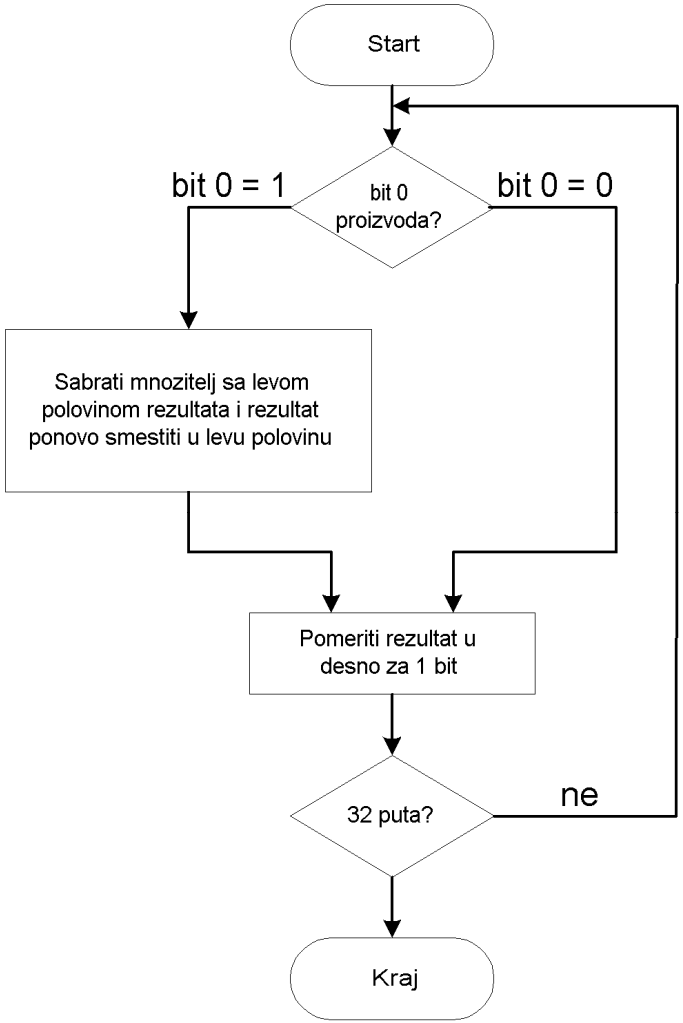
```
last_line: if ver > 0 and ver = depth-1
   and depth > 1 generate
begin
   mux: mux_2
   port map (
        data_in1  => out_ver_hor(ver-1)(hor),
        data_in2  => out_ver_hor(ver-1)((width +
                (2**ver + hor)) mod width),
        data_out  => data_out(hor),
        sel       => shift(ver)
);
end generate last_line;
```



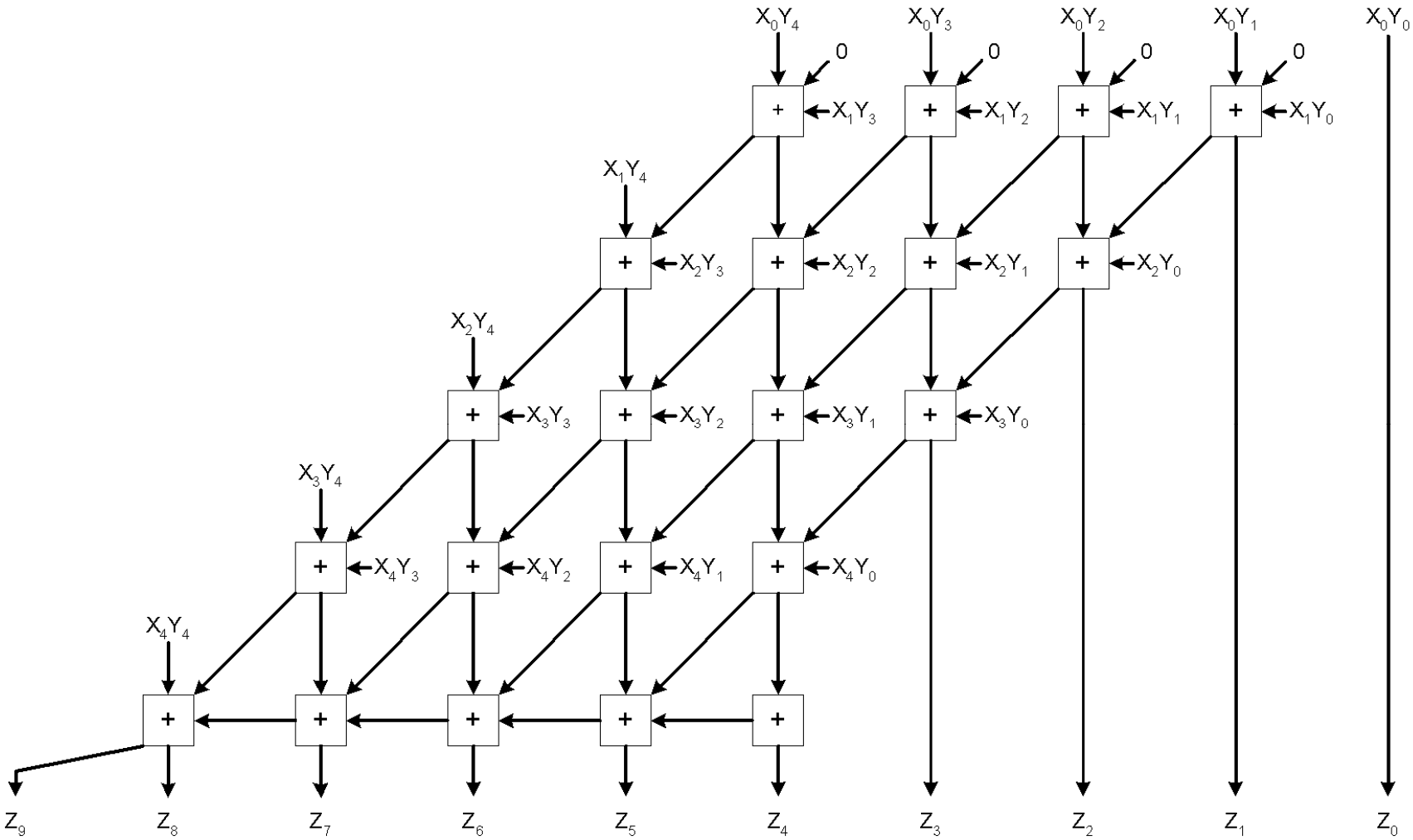
Barel sinteza

Množači

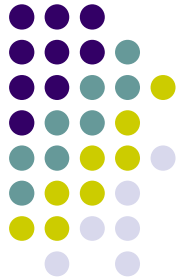

mnozitelj
32
32
ALU
32
rezultat
sr
wr
kontrolna
logika


Start
bit 0 = 1
bit 0
proizvoda?
bit 0 = 0
Sabrati mnozitelj sa levom
polovinom rezultata i rezultat
ponovo smestiti u levu polovinu
Pomeriti rezultat u
desno za 1 bit
32 puta?
ne
Kraj

Iterativni množač

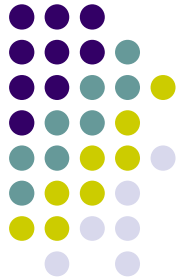


Paralelno množenje



- Osnovna ideja:
  - Formirati sve parcijalne proizvode istovremeno i sabrati ih uz pomeranje, kao kod ručnog množenja

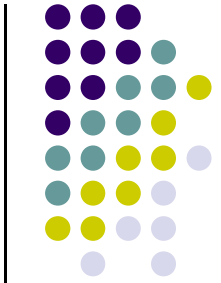
Butov (*Booth*) algoritam



- 6 = -2 + 8 => 0110 = - 0010 + 1000
- 0010 * 0110 =
  0010 * (−0010 + 1000)= +0000
  −0010
  +0000
  +0010

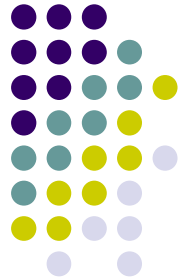
  00001100

Butov algoritam



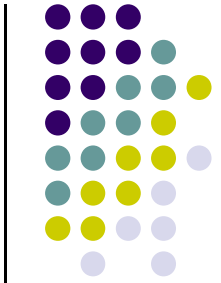
1. U zavisnosti od tekućeg i prethodnog bita:
   - **00**: sredina niza nula =>
     bez operacije
   - **01**: kraj niza jedinica =>
     dodati množilac levoj strani proizvoda
   - **10**: početak niza jedinica =>
     oduzeti množilac od leve strane proizvoda
   - **11**: sredina niza jedinica =>
     bez operacije
2. Šiftuj proizvod za jedan bit

Modifikovan Butov algoritam



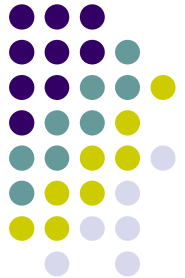
- Da bi se smanjio broj koraka (parcijalnih proizvoda) gledaju se po tri bita
- Proširiti činioce znakom, tako da broj bita bude paran
- Podeliti jedan činilac na grupe od po 3 bita koje se preklapaju za po 1 bit, uz dodatak 0 kao najniže cifre.
- Grupe se formiraju sa desna na levo
- Za svaku grupu formirati parcijalni proizvod
- Sabrati parcijalni proizvod sa levom polovinom rezultata i rezultat pomeriti za dva mesta udesno

Parcijalni proizvodi



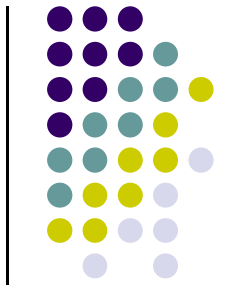
y_{i+1} (-2x)	y_i (x)	y_{i-1} (x)	xy (rezultat)
0	0	0	0
0	0	1	x
0	1	0	x
0	1	1	2x
1	0	0	-2x
1	0	1	-x
1	1	0	-x
1	1	1	0

Modifikovan Butov množač - VHDL



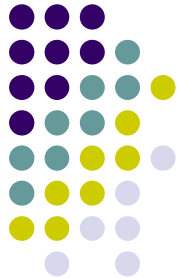
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY Booth IS
   GENERIC (width: POSITIVE:= 6);                    -- sirina mnozenika
   PORT     (X : IN  std_logic_vector(width   - 1 DOWNTO 0); -- mnozenik
             Y : IN  std_logic_vector(width   - 1 DOWNTO 0); -- mnozilac
             O : OUT std_logic_vector(2*width - 1 DOWNTO 0)  -- rezultat
             );
END ENTITY Booth;
```

Modifikovan Butov množač - VHDL



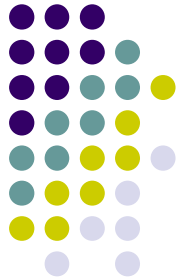
```
ARCHITECTURE behavior OF Booth IS
BEGIN                -- behavior
   MUL: PROCESS (X, Y) IS
      VARIABLE bits : std_logic_vector(2           DOWNTO 0); -- biti operacije
      VARIABLE res  : std_logic_vector(2*width - 1 DOWNTO 0); -- medjurezultat
   BEGIN
      ASSERT (Is_X(X) OR Is_X(Y))                          -- da li su potpuno definisani
         REPORT "Postoje nedefinisane vrednosti" SEVERITY warning;

      res := (OTHERS => '0');                              -- postaviti rezultat na nulu;
           REPORT "X = "    & vector_to_string(X)  &
                  " Y = "   & vector_to_string(Y)  &
                  " res = " & vector_to_string(res);

      FOR index IN 1 TO width/2 LOOP
         IF (index = 1) THEN                               -- ako je index = 1
            bits := Y(1 DOWNTO 0) & '0';                 - prosiriti prva dva bita nulom
         ELSE
            bits := Y(index*2 - 1 DOWNTO index*2 - 3);   -- sledeca grupa bita
         END IF;
```

Modifikovan Butov množač - VHDL



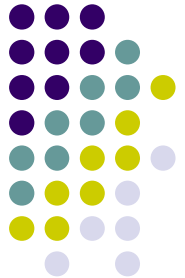
```
STEP: CASE bits IS                                  -- Izvrsavanje operacije
    WHEN "000"  =>  REPORT "Korak: "  & integer'image(index) &
                    " bits = " & vector_to_string(bits)      &
                    " Dodavanje nule(bez niza) "             &
                    " res = "  & vector_to_string(res);

    WHEN "001"  =>  res(2*width - 1 DOWNTO width) :=
                    res(2*width - 1 DOWNTO width) + X;
                    REPORT "Korak: "  & integer'image(index) &
                    " bits = " & vector_to_string(bits)      &
                    " Dodavanje mnozenika (kraj niza) "      &
                    " res = "  & vector_to_string(res);

    WHEN "010"  =>  res(2*width - 1 DOWNTO width) :=
                    res(2*width - 1 DOWNTO width) + X;
                    REPORT "Korak: "  & integer'image(index) &
                    " bits = " & vector_to_string(bits)      &
                    " Dodavanje mnozenika (niza) "           &
                    " res = "  & vector_to_string(res);
```

Modifikovan Butov množač - VHDL



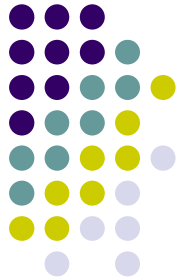
```
WHEN "011"  => res(2*width - 1 DOWNTO width) :=
               res(2*width - 1 DOWNTO width) +
               To_StdLogicVector(To_bitvector(X) SLL 1);
               REPORT "Korak: "  & integer'image(index)  &
               " bits = " & vector_to_string(bits)       &
               " Dodavanje mnozenika 2x (kraj niza) "    &
               " res = "  & vector_to_string(res);

WHEN "100"  => res(2*width - 1 DOWNTO width) :=
               res(2*width - 1 DOWNTO width) -
               To_StdLogicVector(To_bitvector(X) SLL 1);
               REPORT "Korak: "  & integer'image(index)   &
               " bits = " & vector_to_string(bits)        &
               " Oduzimanje mnozenika 2 puta (pocetak niza) " &
               " res = "  & vector_to_string(res);

WHEN "101"  => res(2*width - 1 DOWNTO width) :=
               res(2*width - 1 DOWNTO width) - X;
               REPORT "Korak: "  & integer'image(index)   &
               " bits = " & vector_to_string(bits)        &
               " Oduzimanje mnozenika (-2X + X) "         &
               " res = "  & vector_to_string(res);
```

Modifikovan Butov množač - VHDL

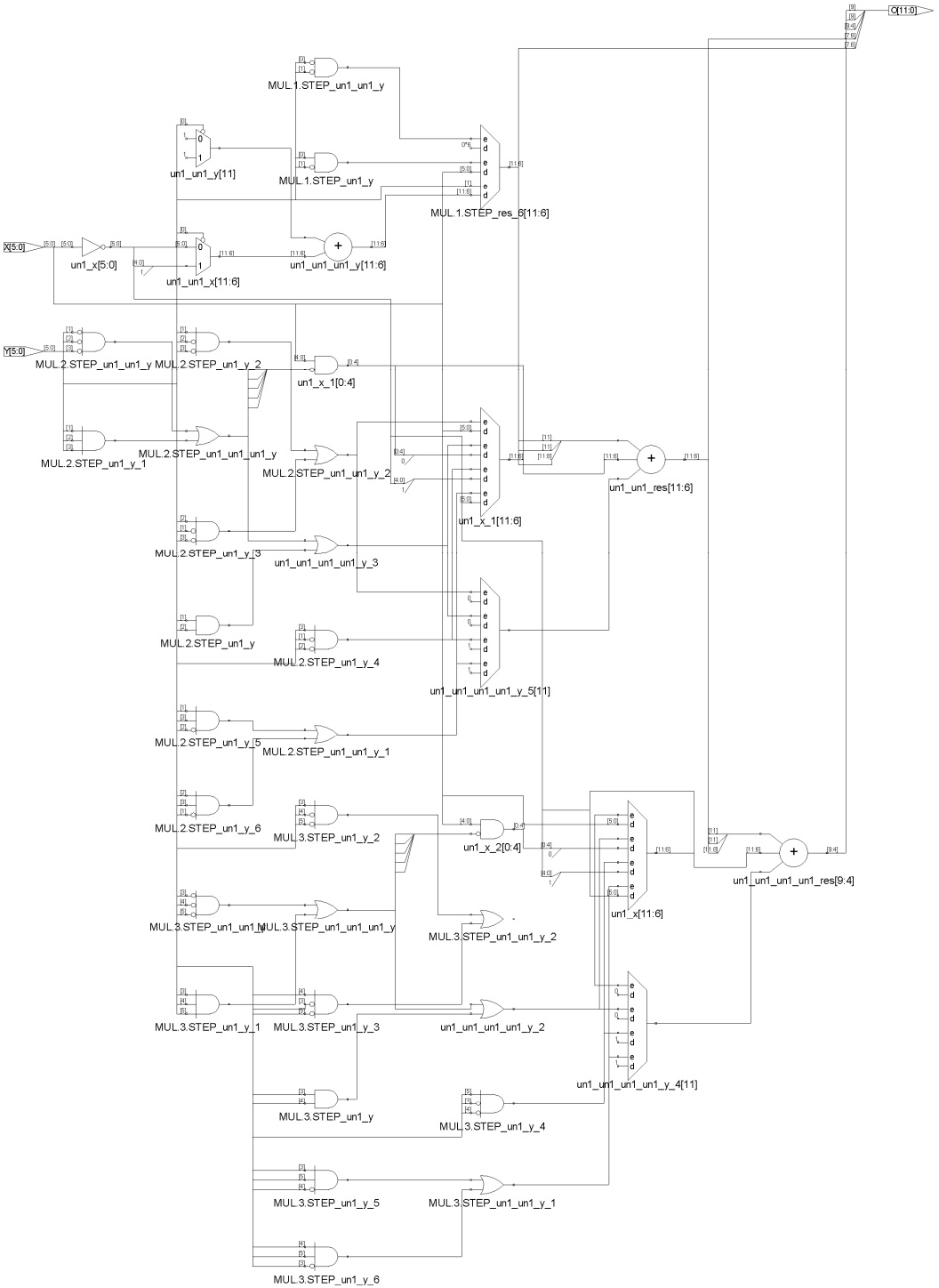


```
        WHEN "110"  => res(2*width - 1 DOWNTO width) :=
                       res(2*width - 1 DOWNTO width) - X;
                       REPORT "Korak: "  & integer'image(index) &
                       " bits = " & vector_to_string(bits)      &
                       " Oduzimanje mnozenika (pozetak niza) "  &
                       " res = "  & vector_to_string(res);

        WHEN "111"  => REPORT "Korak: "  & integer'image(index) &
                       " bits = " & vector_to_string(bits)      &
                       " Oduzimanje nule (sredina niza) "       &
                       " res = "  & vector_to_string(res);

      WHEN OTHERS => NULL;
END CASE STEP;
```

Modifikovan Butov množač - VHDL

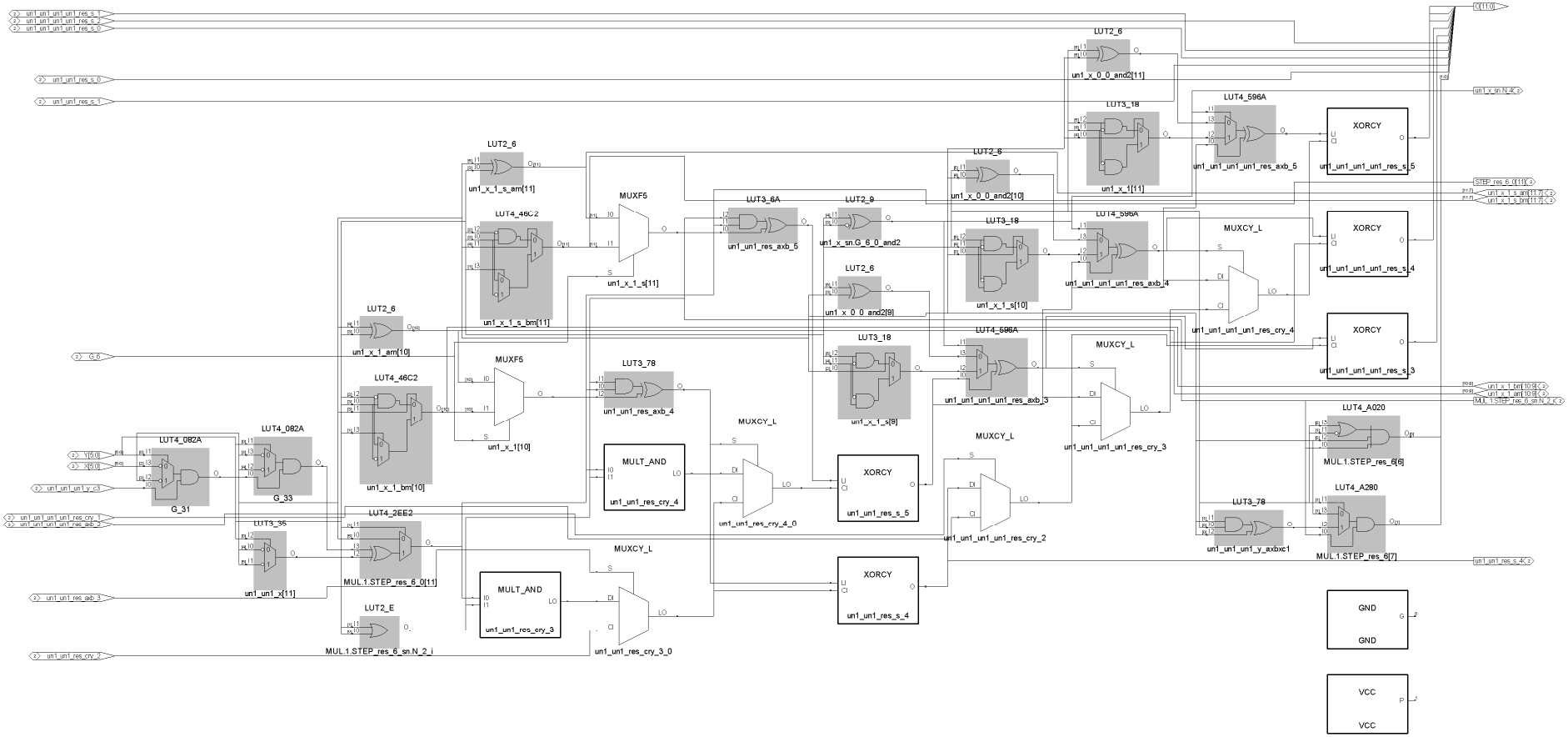


```
            res := To_StdLogicVector(To_bitvector(res) SRA 2);  -- shift right
            REPORT "Korak: "  & integer'image(index) &
                " Pomeranje u desno za dva mesta"    &
                " res = " & vector_to_string(res);

            O <= res;                                   -- Dodela vrednosti izlazu
        END LOOP;
    END PROCESS MUL;
END ARCHITECTURE behavior;
```

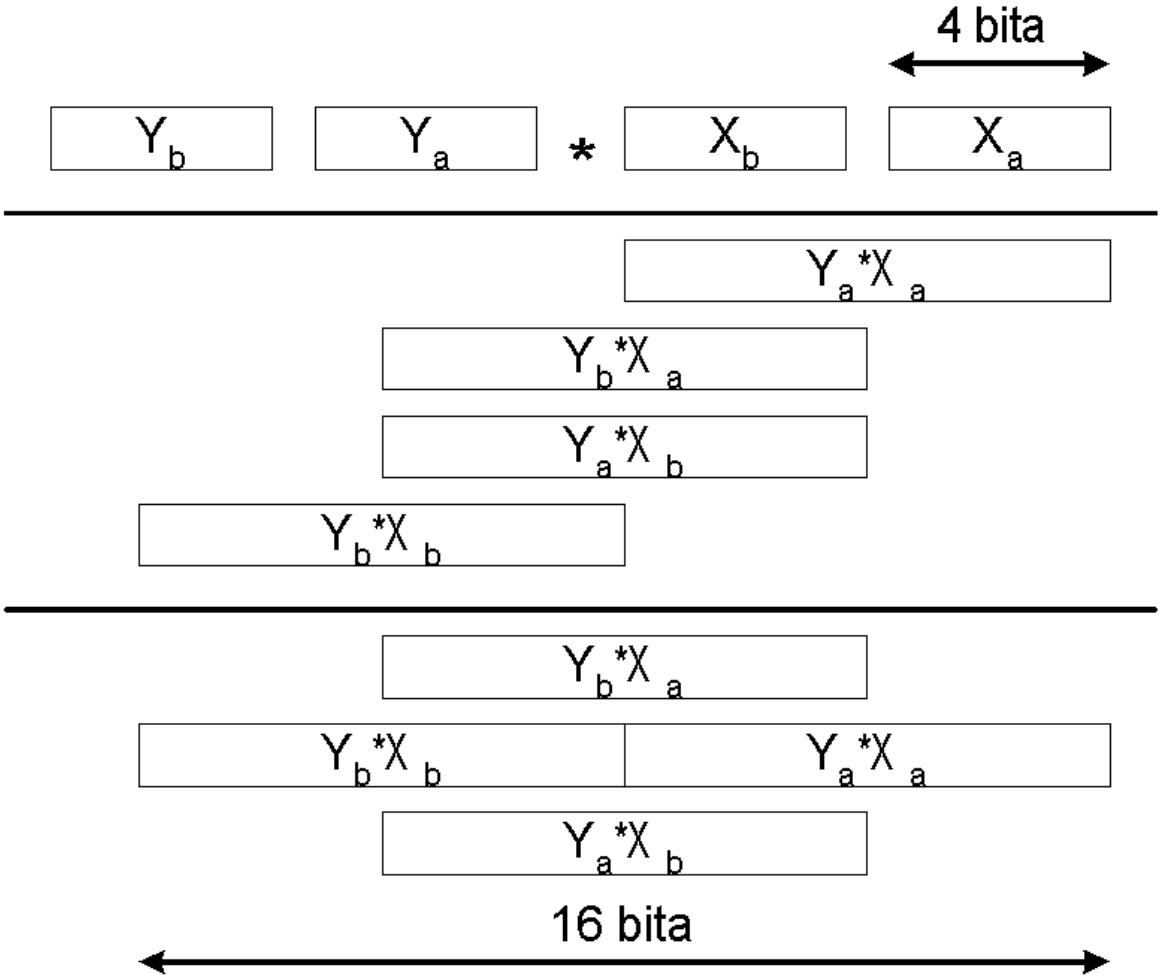


Butov množač sinteza

Butov množač implementacija - Xiliinx



Generisanje parcijalnih proizvoda korišćenjem ROM memorije



Generisanje parcijalnih proizvoda korišćenjem ROM memorije