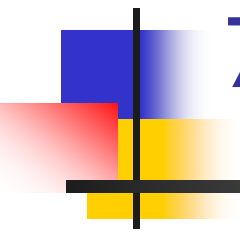


7. Aritmetički sklopovi

Sadržaj predavanja

- **binarno zbrajalo**
  - **poluzbrajalo**
  - **potpuno zbrajalo**
  - **višebitno paralelno zbrajalo**
  - **izdvojeno generiranje prijenosa**
  - **MSI izvedbe**
  - **akumulator**
- zbrajanje u kodu
- binarno oduzimanje
- binarno množenje
- sklop za posmak

Aritmetički sklopovi

- značajna funkcija digitalnog sustava
  ~ "obrada podataka":
  obavljanje aritmetičkih i logičkih operacija
  - važni podsustav
    ~ *procesor*:
    - obavljanje operacija
      - cijeli brojevi (engl. integers)
      - miješani (racionalni) brojevi
    - glavni registri
    - upravljačka jedinica
  - algoritmi digitalne aritmetike

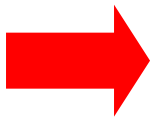
Aritmetički sklopovi

- "radni" dio procesora
  ~ aritmetičko-logička jedinica, ALU
  (engl. Arithmetic-Logic Unit):
  - osnovna izvedba
    ~ operacije nad cijelim brojevima
  - građa ALU:
    - binarno zbrajalo
      ~ zbrajanje, oduzimanje, množenje, dijeljenje
    - jedinica za logičke operacije
      ~ I, ILI, NE, EX-ILI
    - sklop za posmak
      ~ množenje, dijeljenje

Binarno zbrajalo

- osnovni algoritam *binarnog* zbrajanja
  ~ zbrajanje *dva* bita

b_i \ a_i	0	1
0	0	1
1	1	10



		2^0	2^1
A_i	B_i	S_i	C_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

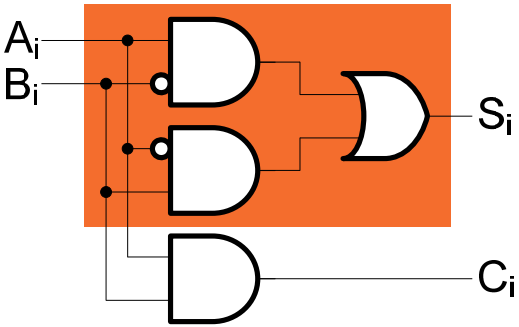
$$S_i = A_i\overline{B}_i + \overline{A}_i B_i = A_i \oplus B_i$$

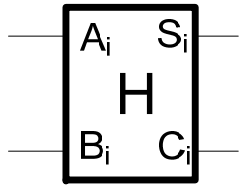
$$C_i = A_i \cdot B_i$$

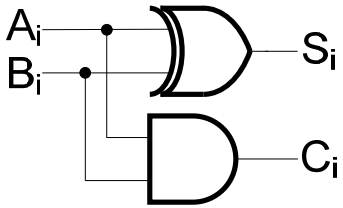
Binarno zbrajalo

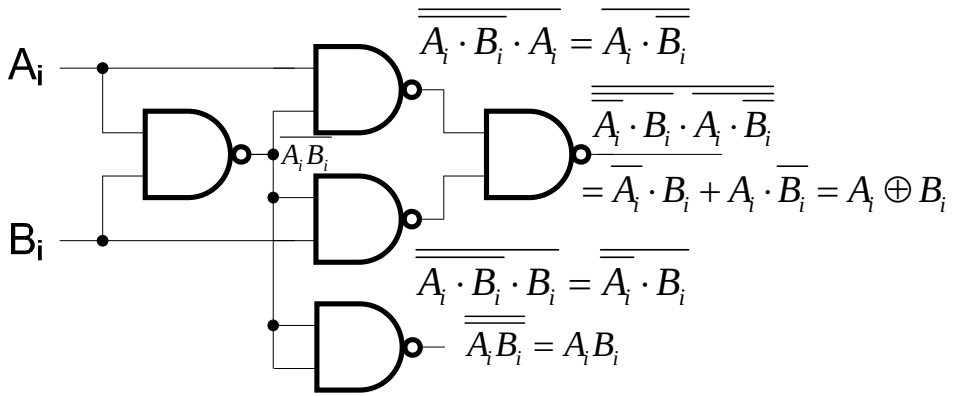
- sklopovska izvedba zbrajanja dva bita
  ~ *poluzbrajalo* (engl. half-adder)

$$S_i = A_i\overline{B}_i + \overline{A}_i B_i$$
$$= A_i \oplus B_i$$
$$C_i = A_i \cdot B_i$$









Binarno zbrajalo

- zbrajanje s prijenosom
  ~ zbrajanje *tri* bita

A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

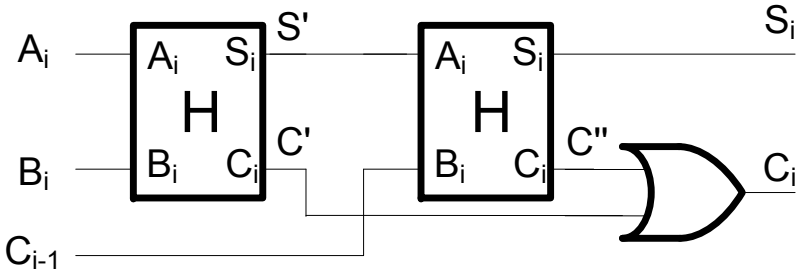
$$
\begin{aligned}
S_i &= \overline{A}_i\overline{B}_i C_{i-1} + \overline{A}_i B_i \overline{C}_{i-1} + A_i\overline{B}_i\overline{C}_{i-1} + A_i B_i C_{i-1} \\
&= \left(\overline{A}_i\overline{B}_i + A_i B_i\right)\cdot C_{i-1} + \left(\overline{A}_i B_i + A_i\overline{B}_i\right)\cdot \overline{C}_{i-1} \\
&= \left(\overline{A_i \oplus B_i}\right)\cdot C_{i-1} + \left(A_i \oplus B_i\right)\cdot \overline{C}_{i-1} \\
&= \left(A_i \oplus B_i\right) \oplus C_{i-1} \\
&= A_i \oplus B_i \oplus C_{i-1}
\end{aligned}
$$

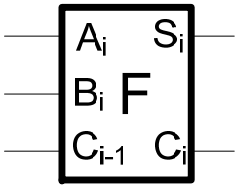
$$
\begin{aligned}
C_i &= \overline{A}_i B_i C_{i-1} + A_i\overline{B}_i C_{i-1} + A_i B_i\overline{C}_{i-1} + A_i B_i C_{i-1} \\
&= \left(\overline{A}_i B_i + A_i\overline{B}_i\right)\cdot C_{i-1} + A_i B_i \\
&= \left(A_i \oplus B_i\right)\cdot C_{i-1} + A_i B_i
\end{aligned}
$$

Binarno zbrajalo

- sklopovska izvedba zbrajanja tri bita
  ~ *potpuno zbrajalo* (engl. full-adder):
  kaskadiranje dva poluzbrajala!

$$S' = A_i \oplus B_i$$
$$C' = A_i \cdot B_i$$
$$S_i = S' \oplus C_{i-1}$$
$$C_i = S' \cdot C_{i-1} + C'$$
$$= C'' + C'$$





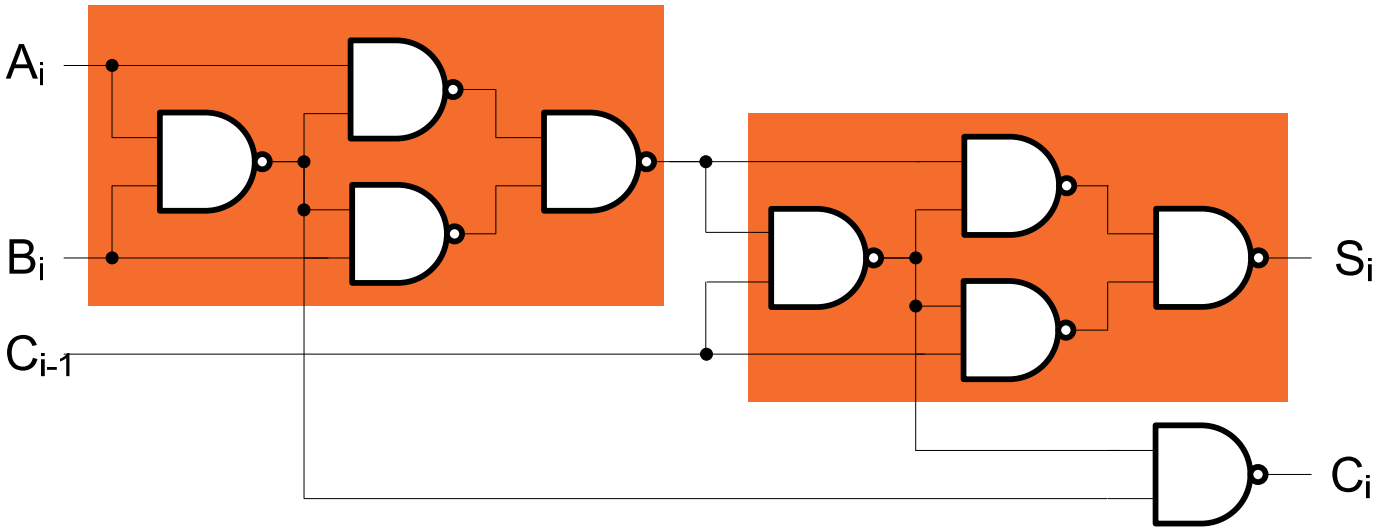
Binarno zbrajalo

- VHDL *ponašajni* model potpunog zbrajala

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

entity potpunoZbrajalo is
  port (a, b, cin: in std_logic;
        s, cout: out std_logic);
end potpunoZbrajalo;

architecture ponasajna of potpunoZbrajalo is
  begin
    s <= a xor b xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end ponasajna;
```

Binarno zbrajalo

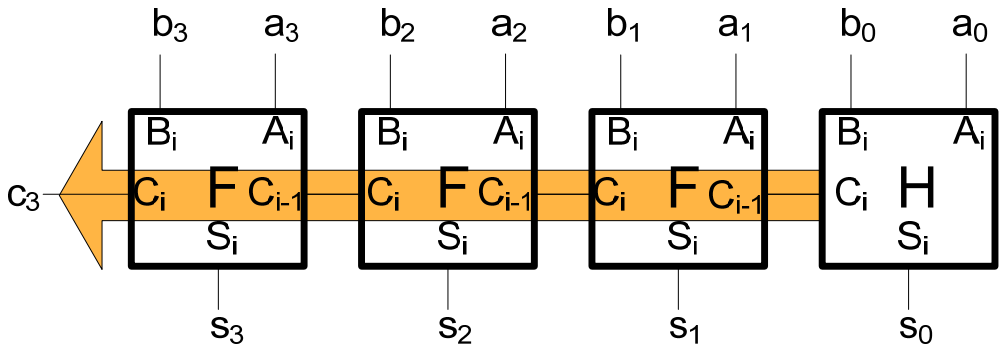
- izvedba potpunog zbrajala
  samo sklopovima NI s 2 ulaza
  ~ C_i samo jednim dodatnim NI s 2 ulaza



$$C_i = \overline{\overline{(A_i \oplus B_i)\cdot C_{i-1}} \cdot \overline{A_i B_i}}$$
$$= (A_i \oplus B_i)\cdot C_{i-1} + A_i B_i$$

Binarno zbrajalo

- zbrajanje *višebitnih* brojeva
  ~ *iteriranje* (jednobitnih) potpunih zbrajala:
  - iteriranje u prostoru
  - *paralelno* izvršavanje operacije zbrajanja
  - ipak se prijenos širi "serijski" (engl. ripple carry)
  - a_0 PLUS b_0
    ~ potpuno zbrajalo (uz $C_{i-1} = 0$)



Binarno zbrajalo

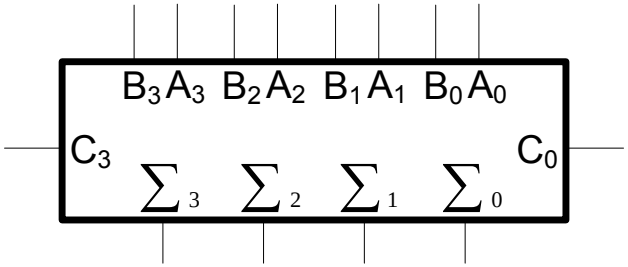
- VHDL *strukturni* model *n*-bitnog zbrajala
  - općeniti model koji pozivom s n = 4 generira 4-bitni modul (parametar definiran s **`generic`**)
  - naredba **`generate`** generira n primjeraka komponente

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

entity nBitZbrajalo is
  generic (n: positive);
  port (a, b: in std_logic_vector(n-1 downto 0);
        cin:  in std_logic;
        s:    out std_logic_vector(n-1 downto 0);
        cout: out std_logic);
end nBitZbrajalo;

architecture strukturna of nBitZbrajalo is
  signal carrys: std_logic_vector(n downto 0);
  begin
    carrys(0) <= cin;
    zbrajala: for i in 0 to n-1 generate
      begin
        zbrajalo: entity work.potpunoZbrajalo
                  port map (a(i),b(i),carrys(i),s(i),carrys(i+1));
      end generate;
  cout <= carrys(n);
end strukturna;
```

Binarno zbrajalo

- *višebitno* paralelno zbrajalo
  ~ MSI modul (obično 4-bitni)
  - mogućnost kaskadiranja
    ~ ostvarivanje *n*-bitnih (n > 4) zbrajala
  - tipični primjer: 7483 (TTL, serija 74)

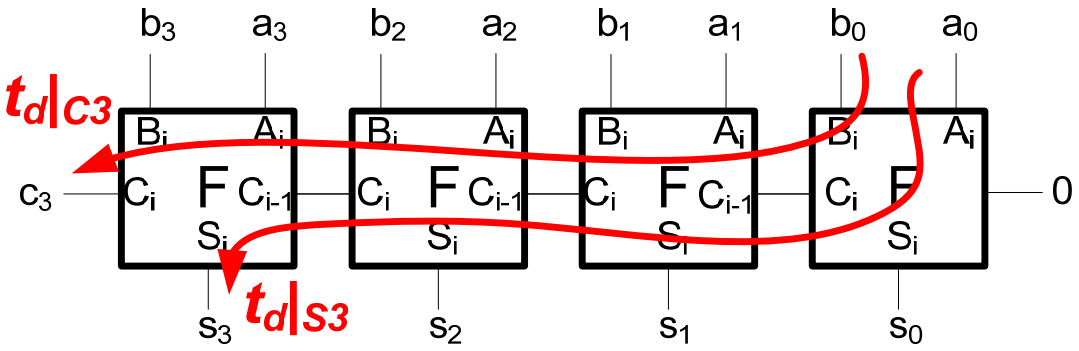
$$A = A_3 A_2 A_1 A_0$$
$$B = B_3 B_2 B_1 B_0$$
$$S = \Sigma_3 \Sigma_2 \Sigma_1 \Sigma_0$$



Binarno zbrajalo

- višebitno paralelno zbrajalo
  ~ *serijsko* rasprostiranje prijenosa:
  usporavanje rada
  - za računanje S_i potreban C_{i-1} , i = 1, …, n-1
  - najviše vremena treba za računanje S_{n-1}, C_{n-1}
    ~ $t_d\big|_{S3} = 3 \cdot t_d\big|_C + 1 \cdot t_d\big|_S$

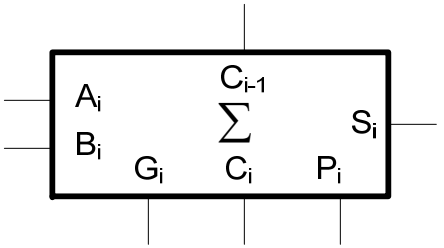
      $t_d\big|_{C3} = 4 \cdot t_d\big|_C$



Binarno zbrajalo

- rješenje problema *serijskog* rasprostiranja prijenosa
  ~ *izdvojeno generiranje prijenosa*, CLA
  (engl. carry look-ahead)

$$C_i = A_i B_i + (A_i \oplus B_i) \cdot C_{i-1}$$
$$= G_i + P_i \cdot C_{i-1}$$
$$G_i = A_i B_i$$
$$P_i = A_i \oplus B_i$$



  - G_i : *generirajući član* (engl. generate)
    $G_i = 1 \Rightarrow C_i = 1$ (bez obzira na $C_{i-1} \cdot (A_i \oplus B_i)$)
    ~ G_i "generira" prijenos
  - P_i : *propagirajući član* (engl. propagate)
    $P_i = 1 \Rightarrow C_i = C_{i-1}$
    ~ P_i "propagira" prijenos s prethodnog bita

Binarno zbrajalo

- *izdvojeno generiranje prijenosa*:
  - za i-ti bit u n-bitnom paralelnom zbrajalu:

    $C_i = G_i + P_i \cdot C_{i-1}$

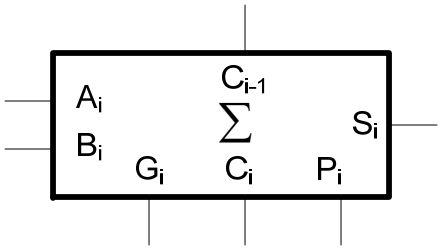
  - prvih nekoliko prijenosa: npr. za 4-bita

    $C_0 = G_0$ jer je $C_{-1} = 0$

    $C_1 = G_1 + G_0 \cdot P_1$

    $C_2 = G_2 + G_1 \cdot P_2 + G_0 \cdot P_2 \cdot P_1$

    $C_3 = G_3 + G_2 \cdot P_3 + G_1 \cdot P_3 \cdot P_2 + G_0 \cdot P_3 \cdot P_2 \cdot P_1$



  - razvoj za i-ti bit:

    $C_i = G_i + G_{i-1} \cdot P_i + G_{i-2} \cdot P_i \cdot P_{i-1} + \ldots + G_0 \cdot P_i \cdot P_{i-1} \cdot \ldots \cdot P_2 \cdot P_1$

Binarno zbrajalo

- ostvarivanje izdvojenog generiranja prijenosa sklopom drugog reda (tipa ILI-I) $\forall$ C_i:

$$C_i = G_i + G_{i-1} \cdot P_i + G_{i-2} \cdot P_i \cdot P_{i-1} + \ldots + G_0 \cdot P_i \cdot P_{i-1} \cdot \ldots \cdot P_2 \cdot P_1$$

  - kašnjenje pri generiranju C_i: $2 \cdot t_{dls}$
  - uračunati t_d za generiranje G_i i P_i: još $2 \cdot t_{dls}$
  - sveukupno kašnjenje: $4 \cdot t_{dls}$

Binarno zbrajalo

- "malo drugačije" ostvarivanje C_i
  ~ *manje* kašnjenje pri generiranju C_i:
  $2 \cdot t_{dls} + \cdot t_{dls} = 3 \cdot t_{dls}$

$$C_i = A_i B_i + (A_i \oplus B_i) \cdot C_{i-1} = A_i B_i + \overline{A_i} B_i C_{i-1} + A_i \overline{B_i} C_{i-1} =$$
$$= A_i B_i + (A_i B_i + \overline{A_i} B_i C_{i-1}) + (A_i B_i + A_i \overline{B_i} C_{i-1}) =$$
$$= A_i B_i + B_i C_{i-1} + A_i C_{i-1} = A_i B_i + (A_i + B_i) \cdot C_{i-1}$$

$$G_i = A_i B_i$$

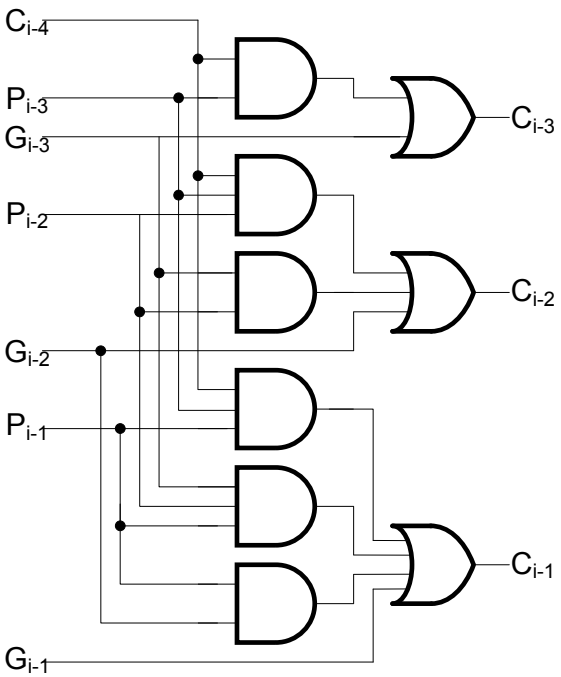
$P_i = A_i + B_i$ umjesto $P_i = A_i \oplus B_i$

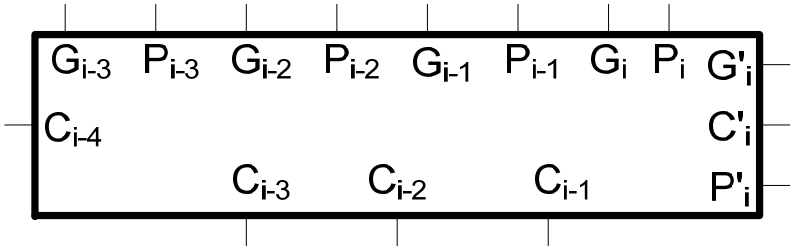
Binarno zbrajalo

- MSI sklop za izdvojeno generiranje prijenosa
  ~ tipično 4-bitni moduli:
  - tehnološki problemi u ostvarenju I sklopova
    s (pre)velikim brojem ulaza!
    ~ *kaskadiranje* izdvojenog generiranja prijenosa
    za manji broj bitova
    - broj bitova zbrajala ↗ → kašnjenje ↗
    - ipak kašnjenje ne raste toliko brzo kao
      kod zbrajala sa serijskim prijenosom!

Binarno zbrajalo

- *MSI sklop za izdvojeno generiranje prijenosa* (engl. carry look-ahead generator, CLA generator), 74182:





$$C_{i-4} = G_{i-4}$$

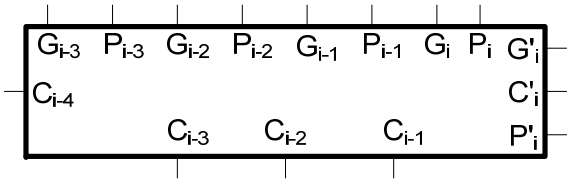
$$C_{i-3} = G_{i-3} + G_{i-4} \cdot P_{i-3}$$

$$C_{i-2} = G_{i-2} + G_{i-3} \cdot P_{i-2} + G_{i-4} \cdot P_{i-4} \cdot P_{i-3}$$

$$C_{i-1} = G_{i-1} + G_{i-2} \cdot P_{i-1} + G_{i-3} \cdot P_{i-1} \cdot P_{i-2} + G_{i-4} \cdot P_{i-1} \cdot P_{i-2} \cdot P_{i-3}$$

Binarno zbrajalo

- *kaskadiranje* MSI sklopova
  za izdvojeno generiranje prijenosa
  ~ P_i', G_i': izlazi za kaskadiranje



$$C_i' = (G_i + G_{i-1} \cdot P_i + G_{i-2} \cdot P_i \cdot P_{i-1} + G_{i-3} \cdot P_i \cdot P_{i-1} \cdot P_{i-2}) + G_{i-4} \cdot (P_i \cdot P_{i-1} \cdot P_{i-2} \cdot P_{i-3})$$

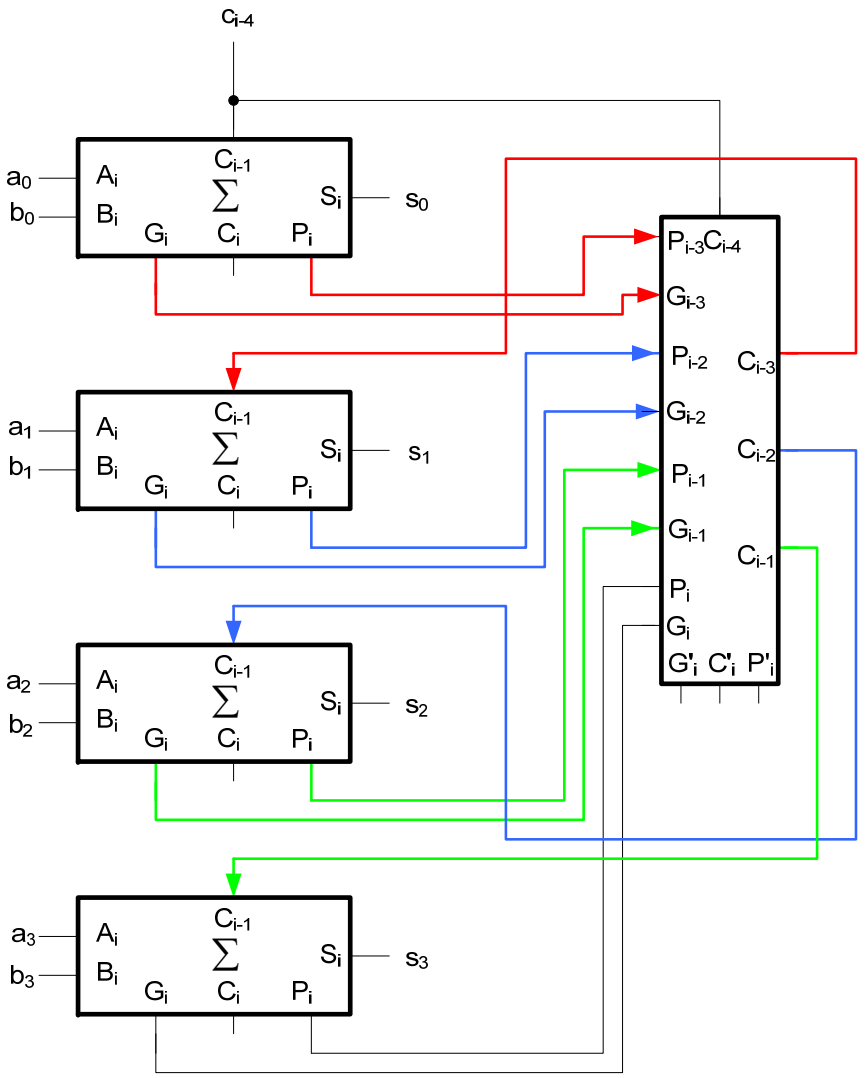
$$C_i' = G_i' + C_{i-4} \cdot P_i'$$

$$P_i' = P_i \cdot P_{i-1} \cdot P_{i-2} \cdot P_{i-3}$$

$$G_i' = G_i + G_{i-1} \cdot P_i + G_{i-2} \cdot P_i \cdot P_{i-1} + G_{i-3} \cdot P_i \cdot P_{i-1} \cdot P_{i-2}$$

Binarno zbrajalo

Primjer: zbrajanje 4-bitnih brojeva s izdvojenim generiranjem prijenosa

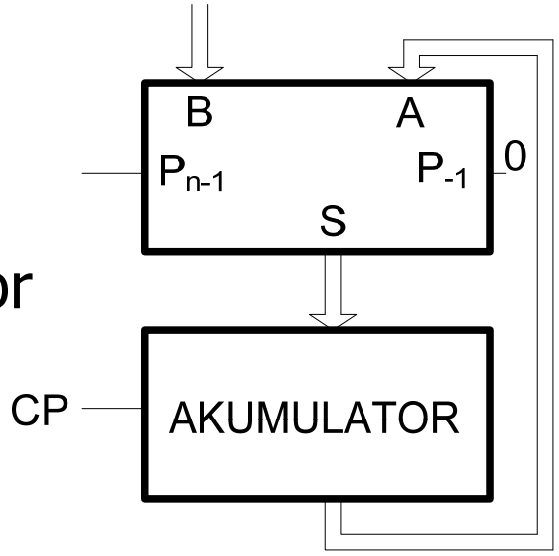


Binarno zbrajalo

- zbroj *više* n-bitnih brojeva
  ~ *pribrajanje* parcijalnoj sumi = "*akumuliranje*"

$$S = x_0 + x_1 + x_2 + \ldots + x_i + \ldots$$
$$= (\ldots(\ldots((((0) + x_0) + x_1) + x_2) + \ldots + x_i) + \ldots)$$

  - uobičajena situacija u računalima
  - mora postojati registar za parcijalnu sumu
    ~ *akumulator* (engl. accumulator)
    - značajni element arhitekture procesora
    - paralelno zbrajalo + akumulator = (primitivna) ALU



Sadržaj predavanja

- binarno zbrajalo
- **zbrajanje u kodu**
  - **BCD zbrajanje**
  - **BCD zbrajalo**
- binarno oduzimanje
- binarno množenje
- sklop za posmak

Zbrajanje u kodu

- BCD kod
  ~ zanimljiv za ljudsku upotrebu (prikaz brojeva!)
  - primijeniti *binarno* zbrajalo
    ~ "ekonomičnost" sklopovlja (standardni modul!)
  - eventualna korekcija rezultata;
    - izbjeći zabranjene kombinacije (1010÷1111)
    - ispravno riješiti pitanje (aritmetičkog) *preljeva*

Zbrajanje u kodu

- analiza BCD zbrajanja
  ~ 3 mogućnosti u ovisnosti o sumi
  - suma < 10
    ~ OK
  - 10 ≤ suma ≤ 15
    ~ generirati ispravnu BCD znamenku + BCD prijenos
  - suma > 15
    ~ generirati ispravnu BCD znamenku + BCD prijenos

```
   5 ≡ 0 1 0 1
 + 3 ≡ 0 0 1 1
-------------
   8 ≡ 1 0 0 0
```

OK.

```
   5 ≡ 0 1 0 1
 + 8 ≡ 1 0 0 0
-------------
 1 3 ≡ 1 1 0 1
```

suma: 3
prijenos: 1

očekuje se BCD rezultat
(2 BCD znamenke)
1 0011

```
   9 ≡   1 0 0 1
 + 8 ≡   1 0 0 0
-----------------
 1 7 ≡ 1 0 0 0 1
```

suma: 7
prijenos: 1
postoji preljev!

1 0111

Zbrajanje u kodu

- rezultat binarnog zbrajanja neispravan u kontekstu BCD koda
  ~ *korekcija* :
  - oduzeti 10 ≡ 1010_2
  - uz 4-bitni prikaz: -10 = -16 + 6
    ~ oduzimanje 10 ≡ pribrajanje 6

```
   5   ≡     0 1 0 1
 + 8   ≡   + 1 0 0 0
------     -------
  13   ≡     1 1 0 1
       +   + 0 1 1 0
       -------------
       1   1 0 0 1 1
```

```
   9   ≡     1 0 0 1
 + 8   ≡   + 1 0 0 0
------     -------
  17   ≡   1 0 0 0 1
       +   + 0 1 1 0
       -------------
       1   1 0 1 1 1
```

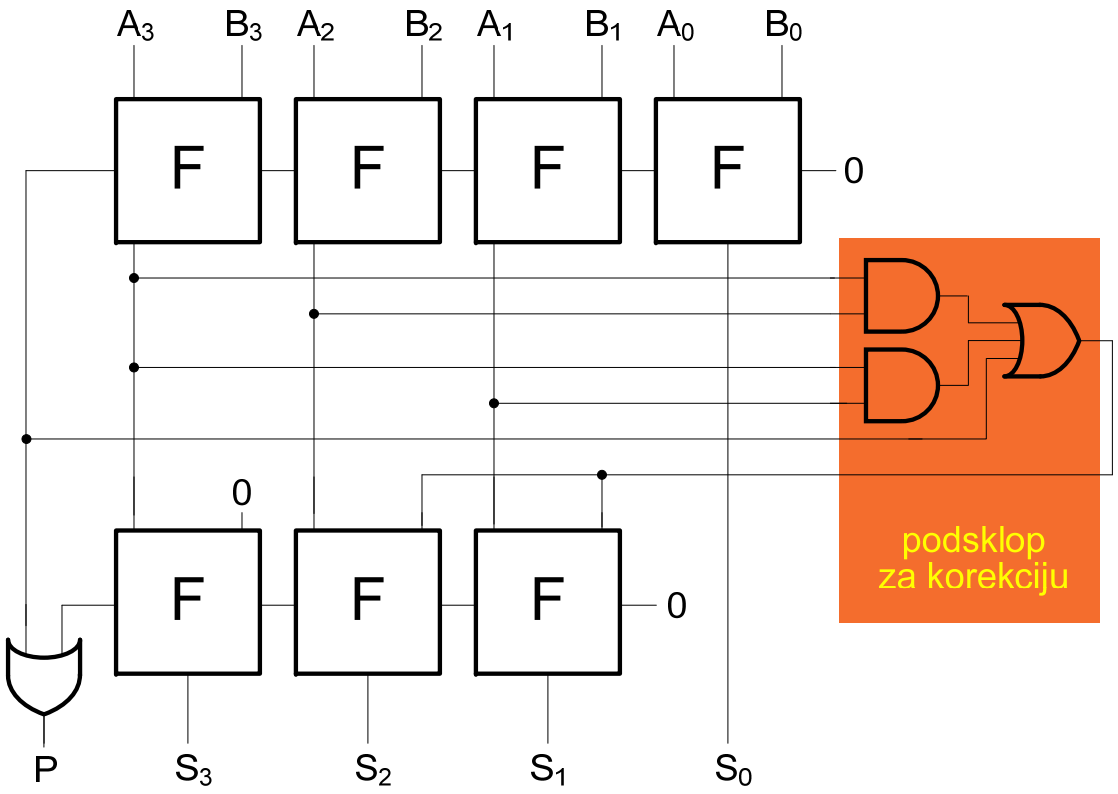
Zbrajanje u kodu

- neformalni algoritam:
  - zbrojiti BCD znamenke po pravilima binarnog zbrajanja
  - **if** (preljev = 0) **and** (suma = kodna riječ iz BCD)
    then rezultat je ispravan
    else dodati $6_{10} = 0110_2$
- sklopovski:

$$korekcija = S_3 \cdot (S_2 + S_1) + C_3$$

Zbrajanje u kodu

- sklop BCD zbrajala:

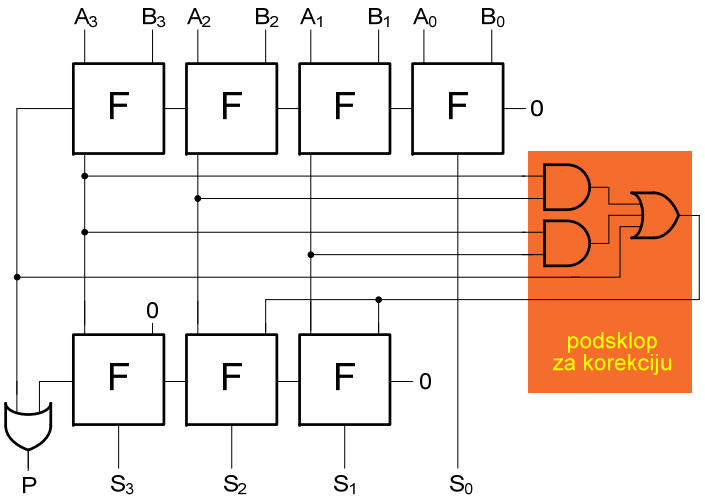


Zbrajanje u kodu

Zadatak: nacrtati sklop za zbrajanje dva 4-znamenkasta BCD broja (uputa: na raspolaganju su moduli 1-znamenkastog BCD zbrajala)

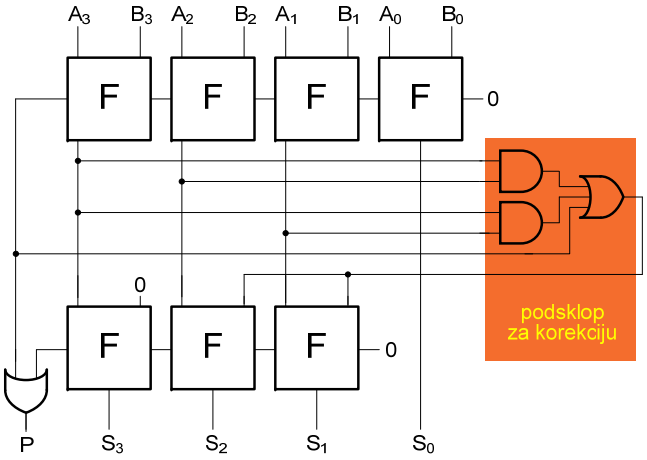
Zbrajanje u kodu

- VHDL *strukturni* model BCD zbrajala




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

entity BCDZbrajalo is
  port (a, b: in std_logic_vector(3 downto 0);
        s:    out std_logic_vector(3 downto 0);
        P:    out std_logic);
end BCDZbrajalo;
```

Zbrajanje u kodu




```
architecture strukturna of BCDZbrajalo is
  signal   carrys1: std_logic_vector(4 downto 0);
  signal   carrys2: std_logic_vector(3 downto 0);
  signal         s1: std_logic_vector(3 downto 0);
  signal korekcija: std_logic;
  signal     ulaz2: std_logic_vector(2 downto 0);
begin
  carrys1(0) <= '0';
  zbrajala1: for i in 0 to 3 generate
    begin
      zbrajalo: entity work.potpunoZbrajalo
                port map (a(i),b(i),carrys1(i),s1(i),carrys1(i+1));
    end generate;
  korekcija <= (s1(3) and s1(2)) or (s1(3) and s1(1)) or carrys1(4);
  ulaz2 <= '0' & korekcija & korekcija;
  carrys2(0) <= '0';
  zbrajala2: for i in 0 to 2 generate
    begin
      zbrajalo: entity work.potpunoZbrajalo
                port map (s1(i+1),ulaz2(i),carrys2(i),s(i+1),carrys2(i+1));
    end generate;
  p <= carrys1(4) or carrys2(3);
  s(0) <= s1(0);
end strukturna;
```

Sadržaj predavanja

- binarno zbrajalo
- zbrajanje u kodu
- **binarno oduzimanje**
  - **potpuno odbijalo**
  - **odbijalo s 2 komplementom**
- binarno množenje
- sklop za posmak

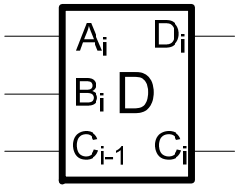
Binarno oduzimanje

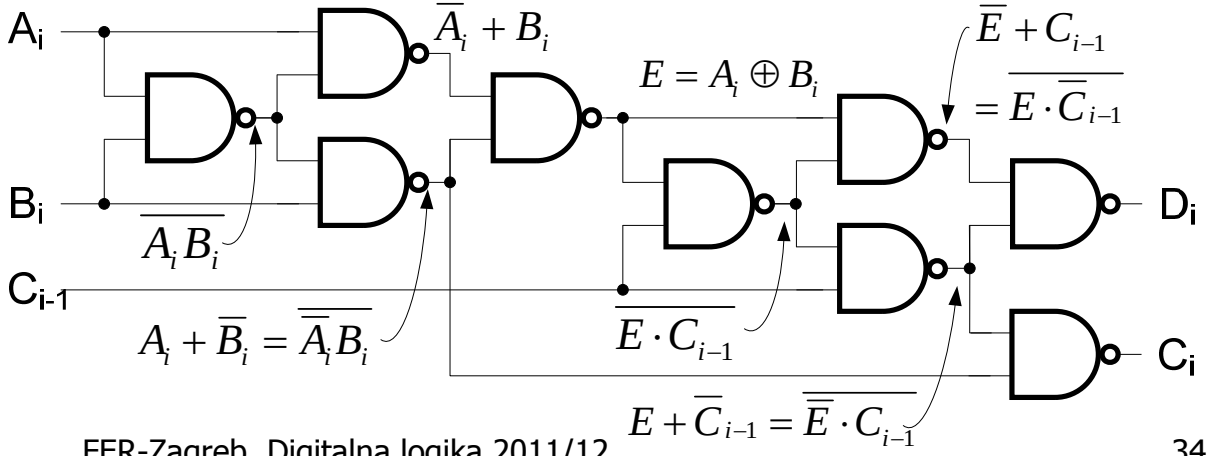
- *potpuno odbijalo* na način potpunog zbrajala:

A_i	B_i	C_{i-1}	D_i	C_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

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

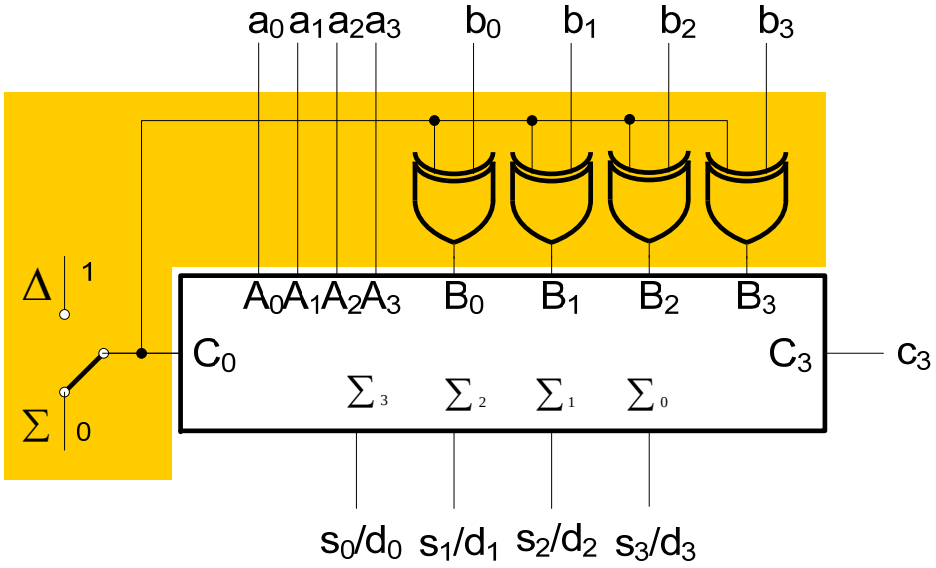
$$C_i = \overline{A}_i B_i + \left(\overline{A}_i \oplus B_i\right) \cdot C_{i-1}$$





Binarno oduzimanje

- potpuno odbijalo *s dvojnim komplementom*,
  ~ binarno zbrajalo + *dodatno sklopovlje*



$$c_0 = \begin{cases} 0 & \Sigma \quad a_3a_2a_1a_0 + b_3b_2b_1b_0 = s_3s_2s_1s_0 \\ 1 & \Delta \quad a_3a_2a_1a_0 - b_3b_2b_1b_0 = d_3d_2d_1d_0 \end{cases}$$

Binarno oduzimanje

- VHDL strukturni model potpunog odbijala s dvojnim komplementom:
  - generiranje *primjerka* parametrizirane komponente
    ~ utvrđivanje vrijednosti parametra n naredbom **`generic map`**

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

entity ZbrajaloOduzimalo is
  port (a, b:      in std_logic_vector(3 downto 0);
        s:         out std_logic_vector(3 downto 0);
        operacija: in std_logic;
        cout:      out std_logic);
end ZbrajaloOduzimalo;

architecture strukturna of ZbrajaloOduzimalo is
  signal b_int: std_logic_vector(3 downto 0);
  begin
    m: entity work.nBitZbrajalo generic map (n => 4) port map
(a,b_int,operacija,s,cout);
    xorovi: for i in 0 to 3 generate
    begin
      b_int(i) <= b(i) xor operacija;
  end generate;
end strukturna;
```

Binarno oduzimanje

Zadatak: ostvariti sklop za oduzimanje u BCD kodu

- izvesti algoritam oduzimanja
- nacrtati sklop za oduzimanje BCD znamenki
- napisati VHDL ponašajni model sklopa
- napisati VHDL strukturni model sklopa

Sadržaj predavanja

- binarno zbrajalo
- zbrajanje u kodu
- binarno oduzimanje
- **binarno množenje**
- sklop za posmak

Binarno množenje

- sklopovska izvedba *sklopa za množenje* binarnih brojeva (*bez* predznaka)
  ~ implementacija Hornerove sheme kombinacijskim sklopovima
  npr. množenje 4-bitnih brojeva

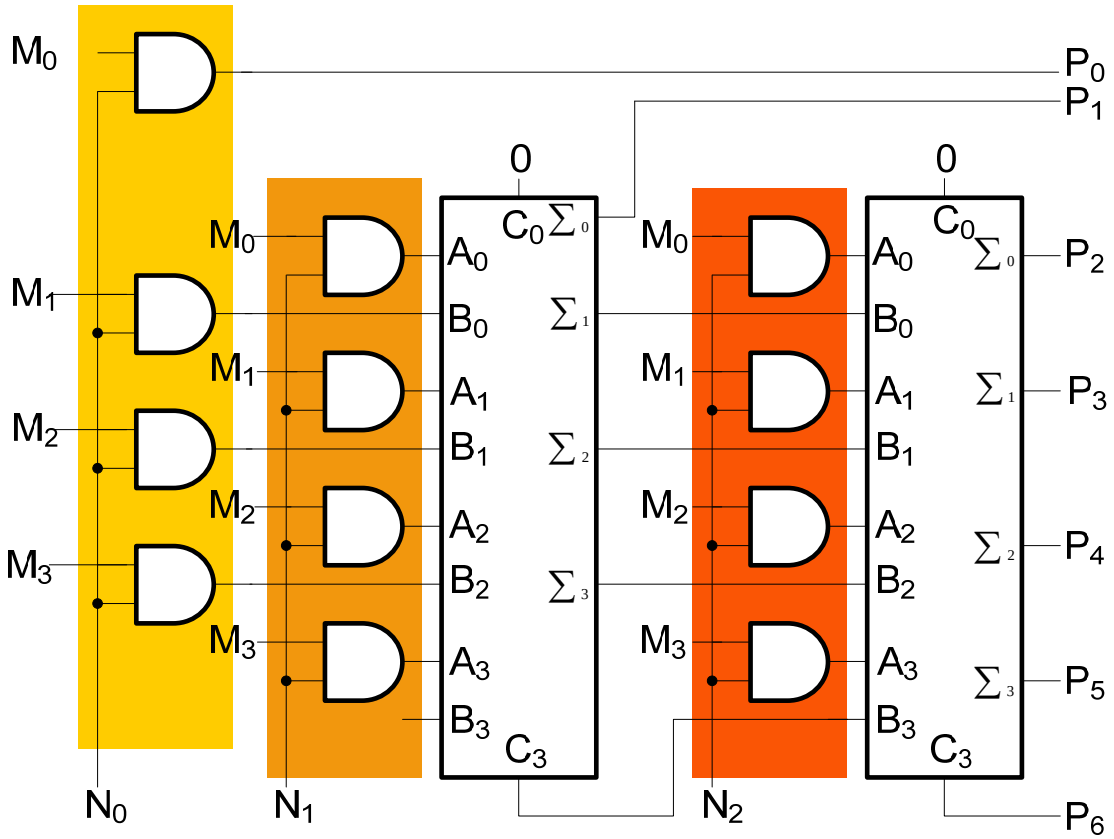
$$M \times N = \left(\left(M \cdot n_3 \cdot 2 + M \cdot n_2\right) \cdot 2 + M \cdot n_1\right) \cdot 2 + M \cdot n_0, n_i \in \{0,1\}$$

  - posmak multiplikanda
    ~ niz *posmaknutih* zbrajala
  - odabir posmaknutog multiplikanda
    ~ niz koincidentnih sklopova (sklopovi I)
  - broj stupnjeva
    ~ broj bitova multiplikatora

Binarno množenje

Primjer: sklop za množenje binarnih brojeva bez predznaka (engl. unsigned array multiplier)

$$P = M \times N = \left(M \cdot n_2 \cdot 2 + M \cdot n_1\right) \cdot 2 + M \cdot n_0, n_i \in \{0,1\}$$

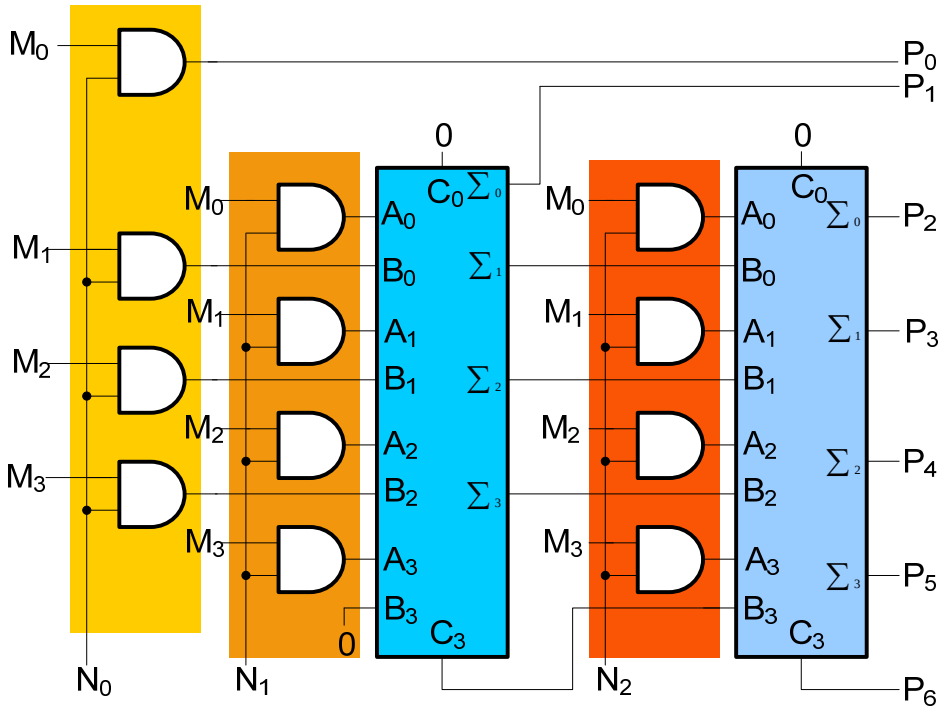


$$P = M \cdot N$$

Binarno množenje

- sklop za množenje binarnih brojeva bez predznaka
  ~ *pridruživanje* operacija – razina sklopovlja

$$m_3m_2m_1m_0 \times n_2n_1n_0 = p_6p_5p_4p_3p_2p_1p_0$$



1110 x 101 = 1000110

			1	1	1	0	
		0	1	1	1	0	
		0	0	0	0		
	0	0	1	1	1		
	1	1	1	0			
1	0	0	0	1	1	0	
P_6	P_5	P_4	P_3	P_2	P_1	P_0	

Binarno množenje

Zadatak: nacrtati sklop za množenje binarnih brojeva M x N:

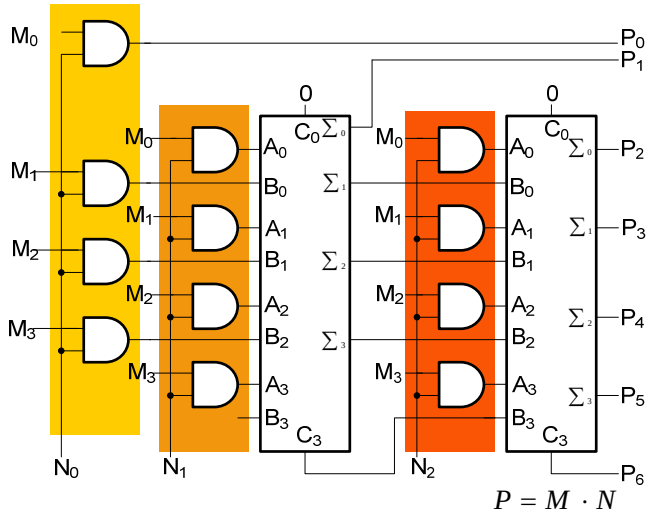
- $M = m_3m_2m_1m_0$, $N = n_4n_3n_2n_1n_0$
- $M = m_7 \ldots m_0$, $N = n_3n_2n_1n_0$

koristiti module 4-bitnih zbrajala

Binarno množenje

- VHDL *strukturni* model sklopa za množenje binarnih brojeva bez predznaka

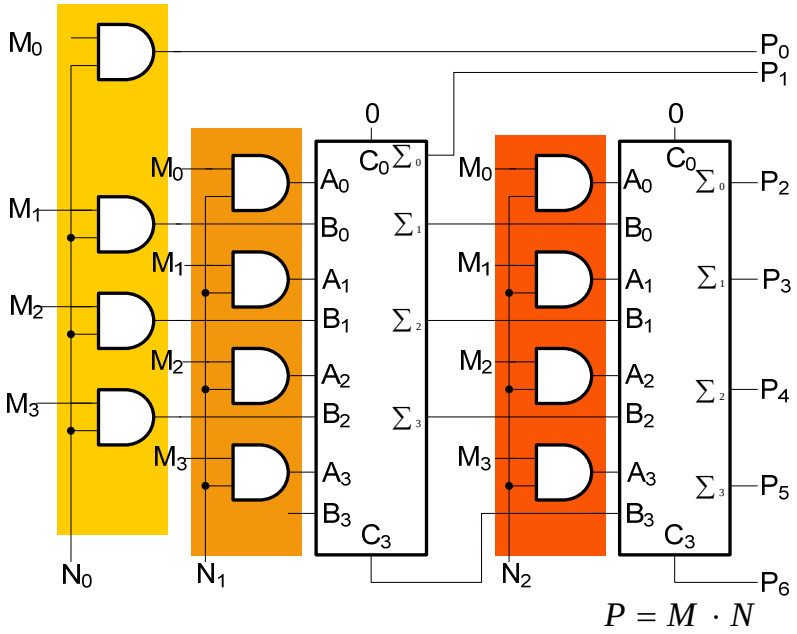
$$M \times N = \left(M \cdot n_2 \cdot 2 + M \cdot n_1\right) \cdot 2 + M \cdot n_0, n_i \in \{0,1\}$$




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

entity Mnozilo4_3 is
  port (m: in std_logic_vector(3 downto 0);
        n: in std_logic_vector(2 downto 0);
        p: out std_logic_vector(6 downto 0));
end Mnozilo4_3;
```

Binarno množenje

```
architecture strukturna of Mnozilo4_3 is
  signal pom1: std_logic_vector(4 downto 0);
  signal pom2: std_logic_vector(3 downto 0);
  signal pom3: std_logic_vector(4 downto 0);
  signal pom4: std_logic_vector(3 downto 0);
  signal pom5: std_logic_vector(4 downto 0);
begin
  prvi: for i in 0 to 3 generate
    begin
      pom1(i) <= m(i) and n(0);
    end generate;
  pom1(4) <= '0';
  p(0) <= pom1(0);
  drugi: for i in 0 to 3 generate
    begin
      pom2(i) <= m(i) and n(1);
    end generate;
  m1: entity work.nBitZbrajalo generic map (n => 4)
      port map (pom2,pom1(4 downto 1),'0',pom3(3 downto 0),pom3(4));
  p(1) <= pom3(0);
  treci: for i in 0 to 3 generate
    begin
      pom4(i) <= m(i) and n(2);
    end generate;
  m2: entity work.nBitZbrajalo generic map (n => 4)
      port map (pom4,pom3(4 downto 1),'0',pom5(3 downto 0),pom5(4));
  p(5 downto 2) <= pom5(3 downto 0);
  p(6) <= pom5(4);
end strukturna;
```



$$P = M \cdot N$$

Binarno množenje

Zadatak: napisati *parametrizirani* VHDL strukturni model sklopa za množenje m-bitnog broja M s n-bitnim brojem N (parametri m i n)

Binarno množenje

- porast broja bitova:
  - multiplikanda:
    ~ povećanje broja bitova zbrajala
  - multiplikatora:
    ~ povećanje broja stupnjeva

Binarno množenje

- sklopovi za množenje unutar ALU:
  - *iskoristiti binarno zbrajalo* !!!
    ~ uzastopno pribrajanje multiplikanda?
  - bolje
    ~ "upravljani posmak" multiplikanda +
    pribrajanje parcijalnih produkata:
    odvijanje u vremenu
    - sporije od n-stupanjskog rješenja (kombinacijskim sklopovima)
    - brže od uzastopnog pribrajanja multiplikanda
  - potreban *sklop za posmak*

Sadržaj predavanja

- binarno zbrajalo
- zbrajanje u kodu
- binarno oduzimanje
- binarno množenje
- **sklop za posmak**

Sklop za posmak

- vrste *posmaka* (engl. shift):
  - logički
    ~ posmak cijelih brojeva *bez predznaka*/uzorka bitova:
    - posmak udesno: umetanje 0 s lijeva
    - posmak ulijevo: umetanje 0 s desna
  - aritmetički
    ~ posmak cijelih brojeva *s predznakom* (2-komplement):
    - posmak udesno: ponavljanje *najznačajnijeg bita* (bit predznaka) radi ispravne interpretacije posmaknutog broja (dijeljenje!)
    - posmak ulijevo: umetanje 0 s desna
  - kružni
    ~ "rotiranje" bitovnog uzorka

Sklop za posmak

Primjer: posmak brojeva udesno

- posmak broja $F0_H = 11110000_2$
  - aritmetički: 11110000 >> 2 ➔ 11111100 (FC)
  - logički: 11110000 >> 2 ➔ 00111100 (3C)
- posmak broja $5050_H = 0101000001010000_2$
  - aritmetički: 0101000001010000 >> 4 ➔ 0000010100000101
  - logički: isto!
- posmak broja $BBCC_H = 1011101111001100_2$
  - aritmetički: 1011101111001100 >> 4 ➔ 1111101110111100
  - logički: 1011101111001100 >> 4 ➔ 0000101110111100

Sklop za posmak

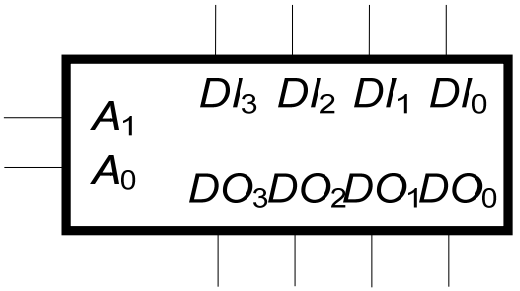
- *sklop za posmak* (engl. shifter):
  - direktno djelovanje na podatak u registru:
    - poseban tip registra ~ sekvencijski sklop
    - posmak se odvija *u vremenu*
  - (posebni) kombinacijski sklop:
    - posmak se ostvaruje iteriranjem sklopova
    - princip: *mreža* multipleksora

Sklop za posmak

Primjer: 4-bitni sklop za *kružni* posmak

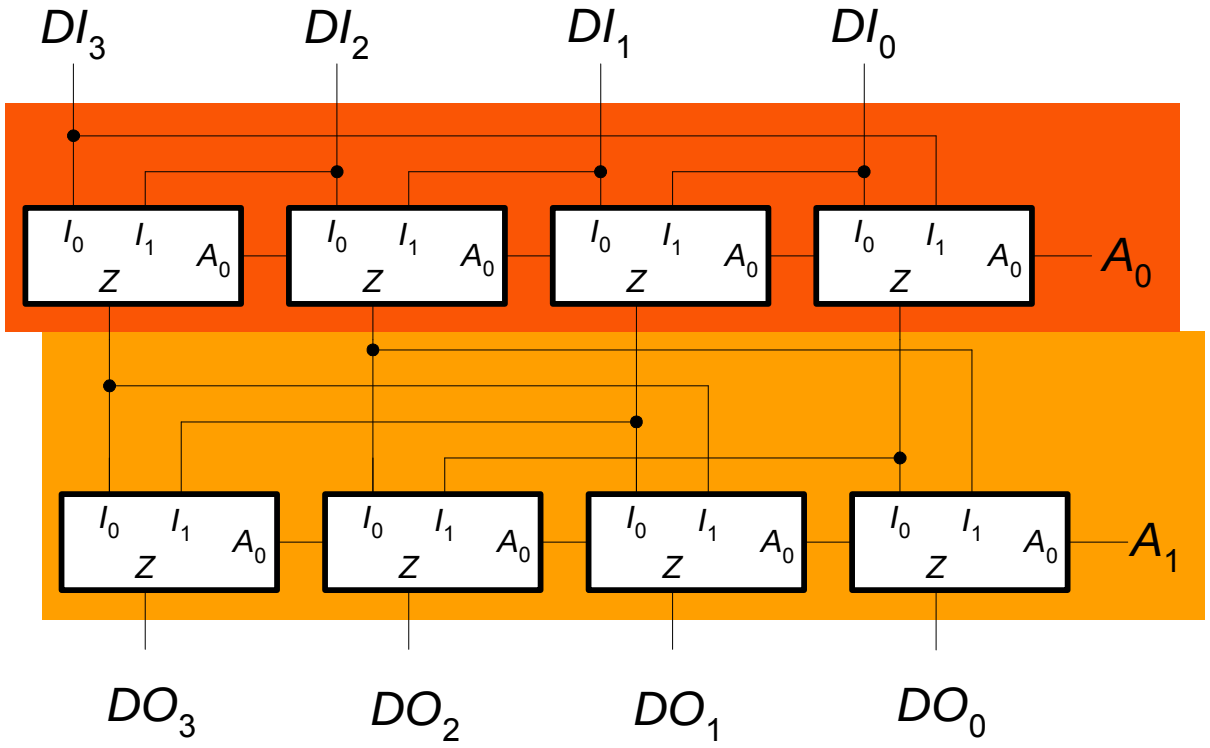
- posmak u smjeru viših težina
  ~ posmak *ulijevo*
- posmak *udesno* za k mjesta
  ~ posmak ulijevo za (n - k) mjesta,
  n: broj bitova riječi

i	A_1	A_0	DO_3	DO_2	DO_1	DO_0
0	0	0	DI_3	DI_2	DI_1	DI_0
1	0	1	DI_2	DI_1	DI_0	DI_3
2	1	0	DI_1	DI_0	DI_3	DI_2
3	1	1	DI_0	DI_3	DI_2	DI_1



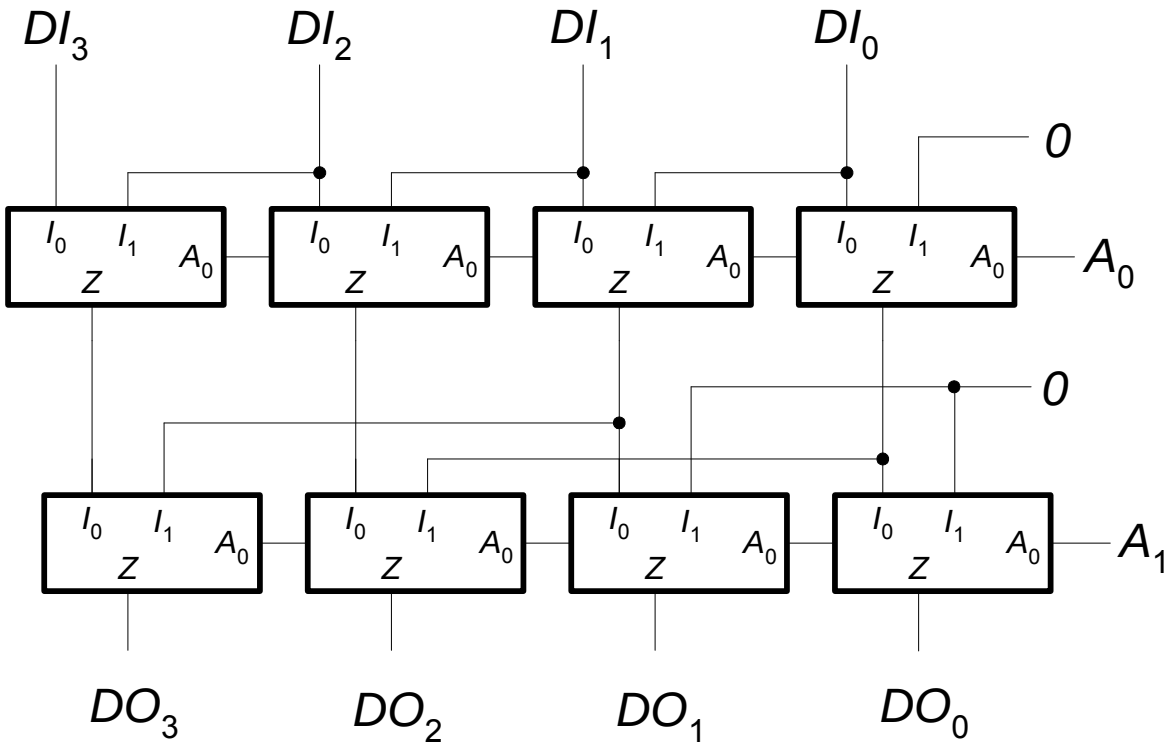
Sklop za posmak

- izvedba sklopa za kružni posmak:
  - prva razina (A_0)
    ~ posmak ulijevo za *jedno* mjesto ($A_0 = 1$)
  - druga razina (A_1)
    ~ posmak ulijevo za *dva* mjesta ($A_1 = 1$)



Sklop za posmak

- sklop za posmak *ulijevo*:
  - nema "povratne veze" na rubovima sklopa
  - s desna umetati 0
  - posmak za 1, 2 ili 3 mjesta



Sklop za posmak

Zadaci:

- napisati VHDL *strukturni* model sklopa za posmak ulijevo (uputa: pročitati str. 322-327 u knjizi Peruško, Glavinić: *Digitalni sustavi*)
- nacrtati 4-bitni sklop za posmak udesno za 1 do 3 mjesta; sklop ostvaruje logički *i* aritmetički posmak
- napisati VHDL *strukturni* model sklopa za posmak udesno
- nacrtati 4-bitni sklop za posmak koji ostvaruje:
  - posmak ulijevo za jedno mjesto
  - posmak udesno za jedno mjesto
  - bez posmaka
  - postavljanje konstante 0000

  (uputa: koristiti multipleksore 4/1)

Literatura

U. Peruško, V. Glavinić: *Digitalni sustavi*, Poglavlje 7: Standardni kombinacijski moduli. Poglavlje 8: Postupak projektiranja s osvrtom na jezik VHDL.

- binarno zbrajalo: str. 287-294
- binarno množenje: str. 295-296
- sklop za posmak: str. 322-327

Zadaci za vježbu (1)

U. Peruško, V. Glavinić: *Digitalni sustavi*, Poglavlje 7: Standardni kombinacijski moduli.

- binarno zbrajalo: 7.35-7.37, 7.48-7.50, 7.53
- zbrajanje u kodu: 7.38, 7.42, 7.45, 7.52,
- binarno oduzimanje: 7.39, 7.40, 7.41
- binarno množenje: 7.43, 7.44, 7.51

Zadaci za vježbu (2)

M. Čupić: *Digitalna elektronika i digitalna logika. Zbirka riješenih zadataka*, Cjelina 8: Digitalna aritmetika.

- binarno zbrajalo:
  - riješeni zadaci: 6.12, 8.7, 8.15, 8.16, 8.17, 8.19, 8.21-8.25
  - zadaci za vježbu: 6 (VHDL)
- zbrajanje u kodu:
  - riješeni zadaci: 8.8, 8.13, 8.14
  - zadaci za vježbu: 9-12
- binarno oduzimanje:
  - riješeni zadaci: 8.10-8.12, 8.18
  - zadaci za vježbu: 1-5, 13-14
- binarno množenje:
  - riješeni zadaci: 8.20, 8.23,
  - zadaci za vježbu: 1-5, 13-14