



6. Standardni kombinacijski moduli

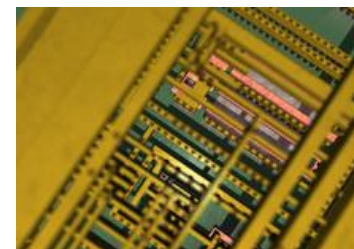


Sadržaj predavanja

- **kombinacijski moduli**
- dekodeer
- multipleksor
- prioritetni koder
- pretvornik koda
- komparator

Kombinacijski moduli

- dekompozicija sustava:
 - identifikacija češće korištenih podsustava/sklopova
~ moduli
 - *kombinacijski* moduli:
 - $izlazi = f(ulazi)$
 - ostvarivanje *složenije* funkcije
(složenije od I, ILI, NE)
 - tipične izvedbe:
 - tehnologija MSI i LSI: komponente
 - tehnologija VLSI: čipovi/dijelovi čipa





Kombinacijski moduli

- općenita podjela kombinacijskih modula:
 - specijalni:
 - ciljano projektirani za zadani sustav
 - *optimalna* izvedba
 - standardni:
 - "opće namjene" (engl. general purpose)
 - široko korištene funkcije
 - proizvodnja u velikim serijama
~ niska cijena (→ "*economy of scale*")
 - univerzalni
~ ostvarivanje proizvoljne Booleove funkcije:
potrebno *programirati* radi obavljanja specifične funkcije



Kombinacijski moduli

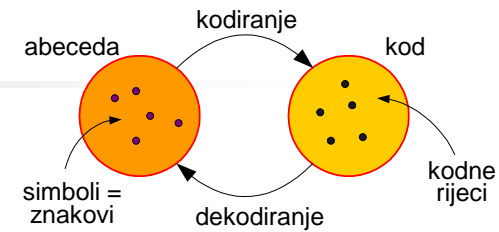
- *standardni* kombinacijski moduli
~ ostvarivanje složenijih kombinacijskih funkcija:
 - dekoderi, koderi, pretvornici koda
 - sklopovi za odabir/usmjeravanje podataka
 - komparatori
 - *aritmetički sklopovi* *



Sadržaj predavanja

- kombinacijski moduli
- **dekoder**
 - **binarni dekodeer**
 - **dekodersko stablo**
 - **ostvarivanje Booleovih funkcija dekodeerom**
 - **dekadski dekodeer**
- multipleksor
- prioritetni koder
- pretvornik koda
- komparator

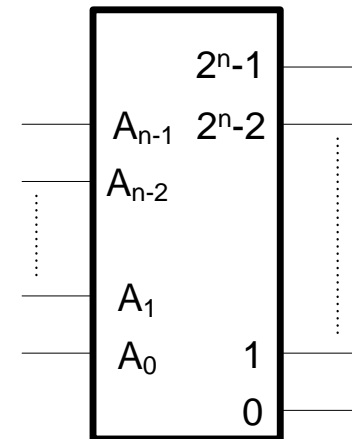
Dekoder



- funkcija *dekodiranja*
~ identificiranje kodne riječi nekog (binarnog!) koda
- *dekoder*
~ aktivan *samo jedan* izlaz,
onaj koji "odgovara" narinutoj kodnoj riječi
 n ulaza $\rightarrow 2^n$ izlaza

$$Z_i = \begin{cases} 1 & \text{za } i = A_{n-1} \dots A_0 \\ 0 & \text{za } i \neq A_{n-1} \dots A_0 \end{cases}$$

- tipična oznaka:
 $\langle \text{broj adresa} \rangle / 2^{\langle \text{broj adresa} \rangle}$





Dekoder

- podjela dekodera:
 - *potpuno* dekodiranje:
 - $n = 2, 3, 4, \dots$ ulaza \rightarrow "1-od- 2^n " izlaza
 \sim *binarni* dekoderi
 - *nepotpuno* dekodiranje:
 - broj izlaza $< 2^n$ (neki put i $\ll 2^n$)
 \sim npr. adresiranje funkcijskih blokova *unutar* digitalnog sustava
 - $n = 4$ ulaza \rightarrow "1-od-10" izlaza
 \sim *dekadski* dekoderi:
dekodiranje binarnih kodova za prikaz dekadskih znamenki, npr. BCD, XS-3

Binarni dekode

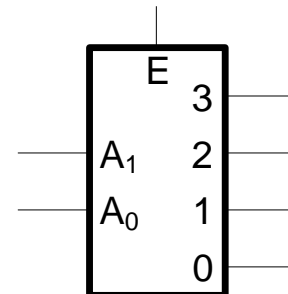
Primjer: binarni dekode 2/4

$$\overline{A_1}\overline{A_0} \equiv 00_2 = 0_{10}$$

$$\overline{A_1}A_0 \equiv 01_2 = 1_{10}$$

$$A_1\overline{A_0} \equiv 10_2 = 2_{10}$$

$$A_1A_0 \equiv 11_2 = 3_{10}$$



- izlazi su *mintermi* funkcije $f(A_1, A_0)$
- upravljanje sklopom
~ *ulaz za omogućavanje*
E (engl. enable)

E	A ₁	A ₀	"0"	"1"	"2"	"3"
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

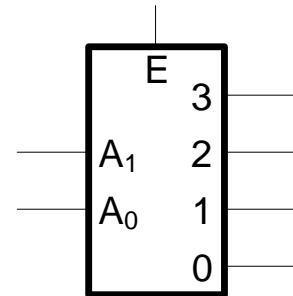
Binarni dekodler

- VHDL *ponašajni* model binarnog dekodera 2/4

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

entity dekod24e is
    port (d: out std_logic_vector(0 to 3);
          a: in std_logic_vector(1 downto 0);
          e: in std_logic);
end dekod24e;
```

```
architecture ponasajna of dekod24e is
begin
    process (a,e)
    begin
        if(e = '0')
            then d <= "0000";
        else case a is
            when "00"    => d <= "1000";
            when "01"    => d <= "0100";
            when "10"    => d <= "0010";
            when "11"    => d <= "0001";
            when others => d <= "0000";
        end case;
        end if;
    end process;
end ponasajna;
```



Binarni dekode

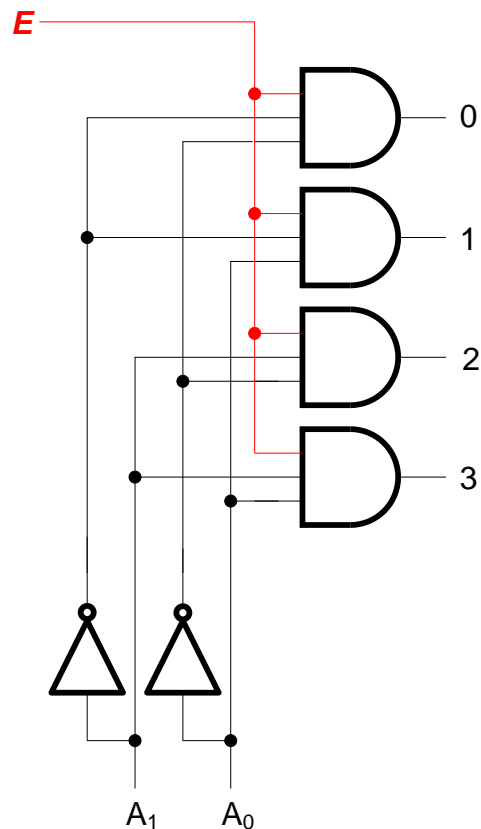
- *struktura* dekodera 2/4
(građa komponentama "niže složenosti"):

$$\overline{A_1}\overline{A_0} \equiv 00_2 = 0_{10}$$

$$\overline{A_1}A_0 \equiv 01_2 = 1_{10}$$

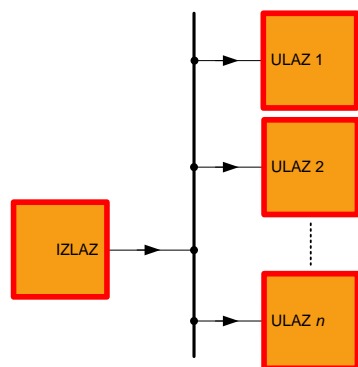
$$A_1\overline{A_0} \equiv 10_2 = 2_{10}$$

$$A_1A_0 \equiv 11_2 = 3_{10}$$

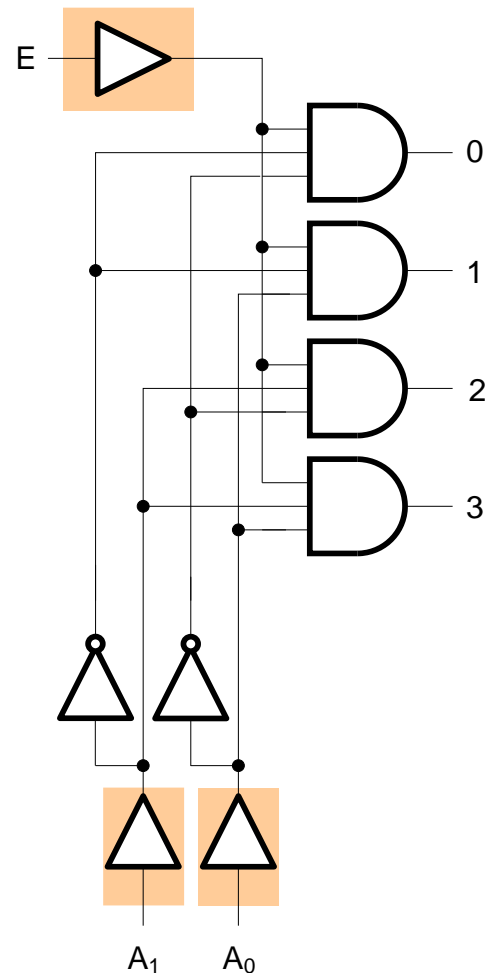


Binarni dekodler

- *stvarna* izvedba
~ postoje električka ograničenja:
svaki ulaz sklopa *opterećuje* izlaz
na koji se priključuje



- ograničiti ovo opterećenje ulazima
- odvojni sklopovi
~ *jedinično* opterećenje
- funkcijski ekvivalentna izvedba



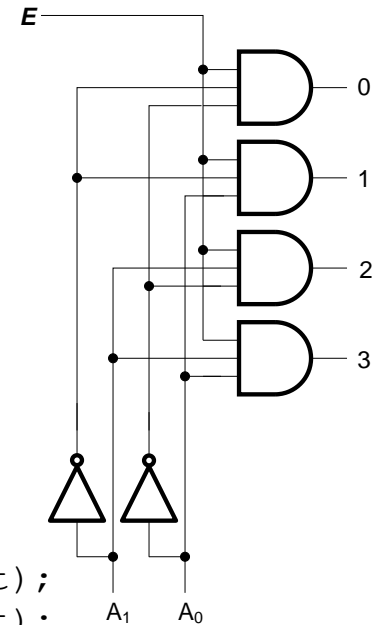
Binarni dekodler

- VHDL *strukturni* model binarnog dekodera 2/4
~ *dva tipa* komponenti (inverter, sklop I s 3 ulaza)

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

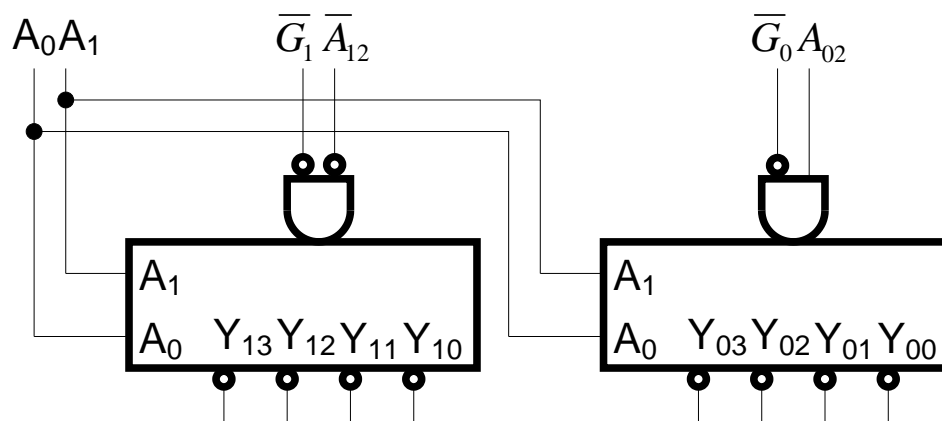
entity dekoder24e is
    port (d: out std_logic_vector(0 to 3);
          a: in std_logic_vector(1 downto 0);
          e: in std_logic);
end dekoder24e;

architecture strukturna of dekoder24e is
    signal a1_komplement, a0_komplement: std_logic;
begin
    sklop1: entity work.sklopNOT port map (a(1), a1_komplement);
    sklop2: entity work.sklopNOT port map (a(0), a0_komplement);
    sklop3: entity work.sklopAND3 port map (e, a1_komplement, a0_komplement, d(0));
    sklop4: entity work.sklopAND3 port map (e, a1_komplement, a(0), d(1));
    sklop5: entity work.sklopAND3 port map (e, a(1), a0_komplement, d(2));
    sklop6: entity work.sklopAND3 port map (e, a(1), a(0), d(3));
end strukturna;
```



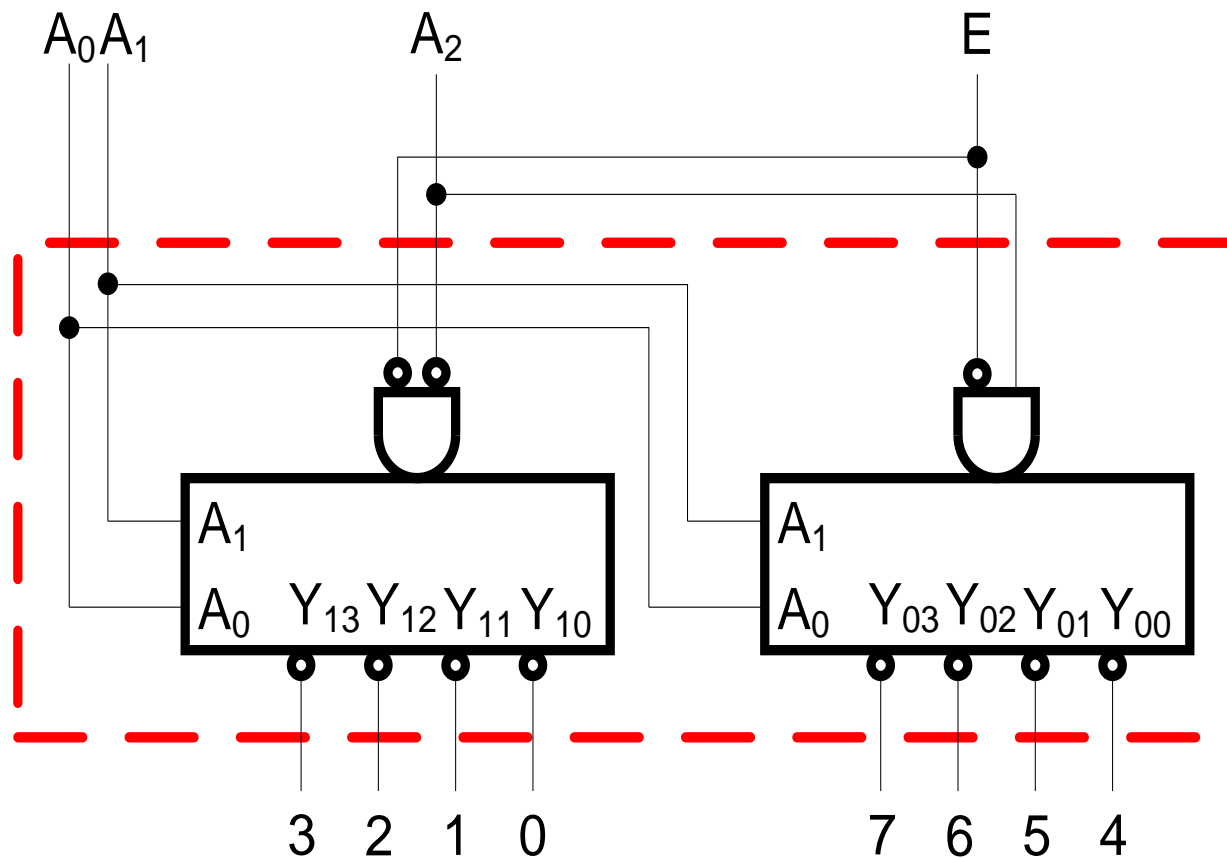
Binarni dekode

- izvedbe dekodera
 - ~ MSI modul; npr. 74155 (*dvostruki* četveroizlazni):
 - *zajedničke* adrese A_1, A_0
 - *izdvojeni* ulazi za omogućavanje s ugrađenom logičkom funkcijom: $\overline{G}_1, \overline{G}_0$
 - jednostavna izvedba dekodera 3-bitnih riječi:
 $A_{02} = \overline{A}_{12} = A_2$



Binarni dekode

Primjer: binarni dekode 3/8 na temelju 74155

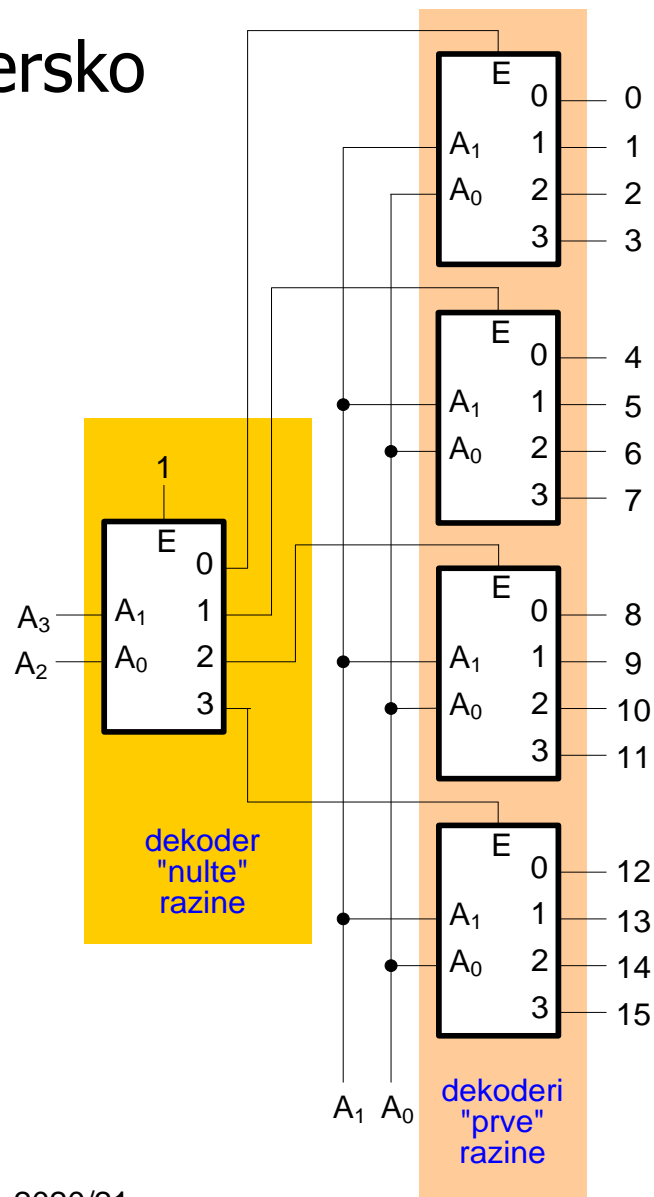


- dekoderi s većim ($n > 16$) brojem izlaza:
 - izvedba jednim modulom *nepraktična*
~ presloženi MSI modul
 - radije *kaskadiranje* više jednostavnijih modula
~ *dekodersko stablo* (engl. decoder tree):
 - općenita metoda
 - vrijedi za proizvoljno složeni modul;
npr. izvedba dekodera sklopovima I

Binarni dekode

Primjer: dekode 4/16 kao dekodersko stablo dekodera 2/4

$A_3 \ A_2$		A_1	A_0	
0	0	0	0	D_0^1
		0	1	
		1	0	
		1	1	
0	1	0	0	D_1^1
		0	1	
		1	0	
		1	1	
1	0	0	0	D_2^1
		0	1	
		1	0	
		1	1	
1	1	0	0	D_3^1
		0	1	
		1	0	
		1	1	



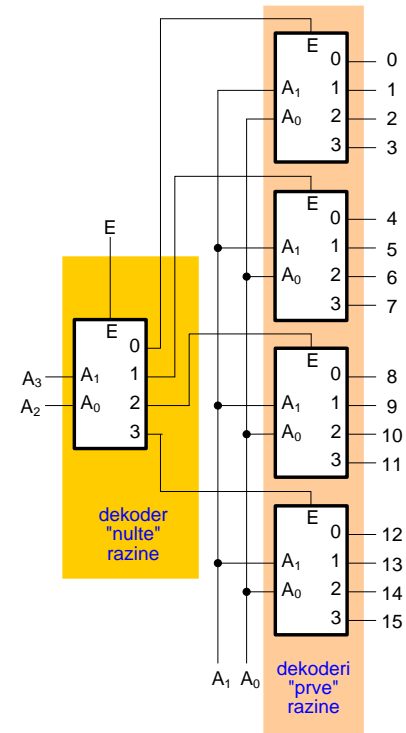
Binarni dekodler

- VHDL *strukturni* model dekoderskog stabla 4/16

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

```
entity dekod416e is  
  port (d: out std_logic_vector(0 to 15);  
        a: in  std_logic_vector(3 downto 0);  
        e: in  std_logic);  
end dekod416e;
```

```
architecture strukturna of dekod416e is  
  signal e_tmp: std_logic_vector(0 to 3);  
begin  
  sklop1: entity work.dekoder24e port map (e_tmp, a(3 downto 2), e);  
  sklop2: entity work.dekoder24e port map (d(0 to 3), a(1 downto 0), e_tmp(0));  
  sklop3: entity work.dekoder24e port map (d(4 to 7), a(1 downto 0), e_tmp(1));  
  sklop4: entity work.dekoder24e port map (d(8 to 11), a(1 downto 0), e_tmp(2));  
  sklop5: entity work.dekoder24e port map (d(12 to 15), a(1 downto 0), e_tmp(3));  
end strukturna;
```



Ostvarivanje funkcija dekomerom

- zapažanje:

- izlazi dekomera = *potpuno dekodirane* kodne riječi
~ *mintermi*

$$\overline{A_1}\overline{A_0} \equiv 00_2 = 0_{10}$$

$$\overline{A_1}A_0 \equiv 01_2 = 1_{10}$$

$$A_1\overline{A_0} \equiv 10_2 = 2_{10}$$

$$A_1A_0 \equiv 11_2 = 3_{10}$$

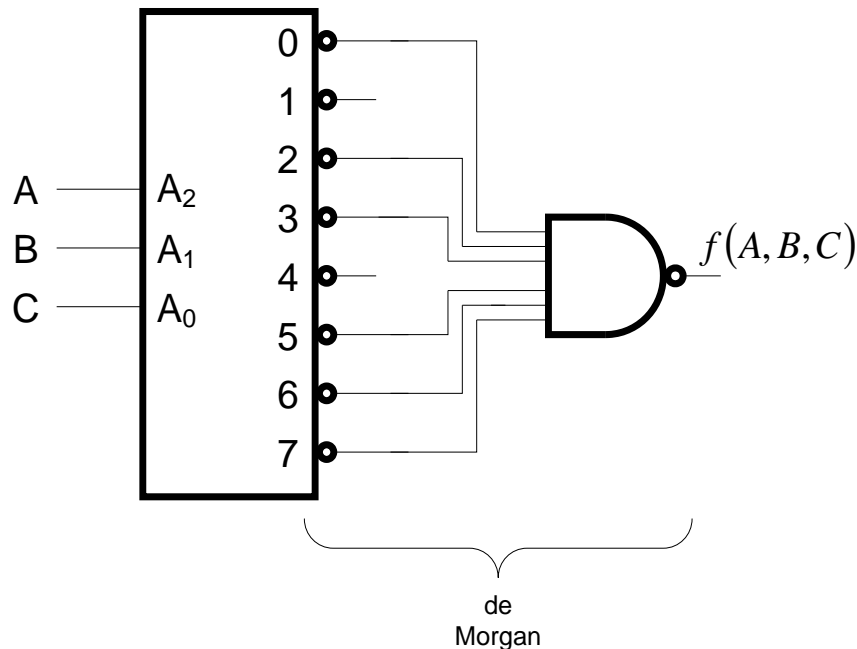
- *ostvarivanje logičkih funkcija* dekomerom:

- funkcija u kanonskom *disjunktivnom* obliku
- (inkluzivna) disjunkcija (funkcija ILI) izlaza koji odgovaraju mintermima zastupljenim u definiciji funkcije

Ostvarivanje funkcija dekomerom

Primjer: $f(A, B, C) = \sum m(0, 2, 3, 5, 6, 7)$

- $f(A, B, C) \rightarrow$ dekomer 3/8
- dekomer s *invertiranim* izlazima
- uočiti: $ILI \circ I = (NI \circ NE) \circ I$





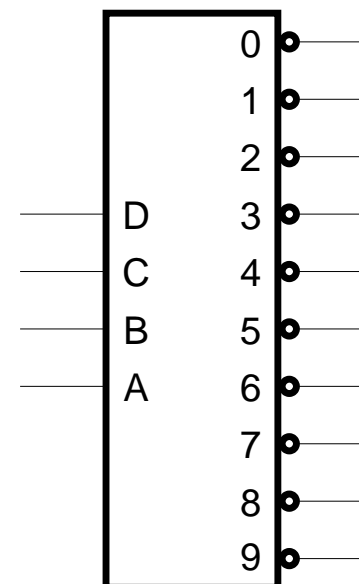
Dekadski dekodler

- dekodiranje koda s $N < 2^n$
~ *nepotpuno* dekodiranje
 - broj izlaza $< 2^{\text{broj ulaza}}$
 - tipični slučaj
~ dekodiranje binarno kodiranih dekadskih znamenki:
 - BCD
 - XS-3
 - 2421
 - itd.

Dekadski dekodler

Primjer: BCD-dekadski dekodler 7442

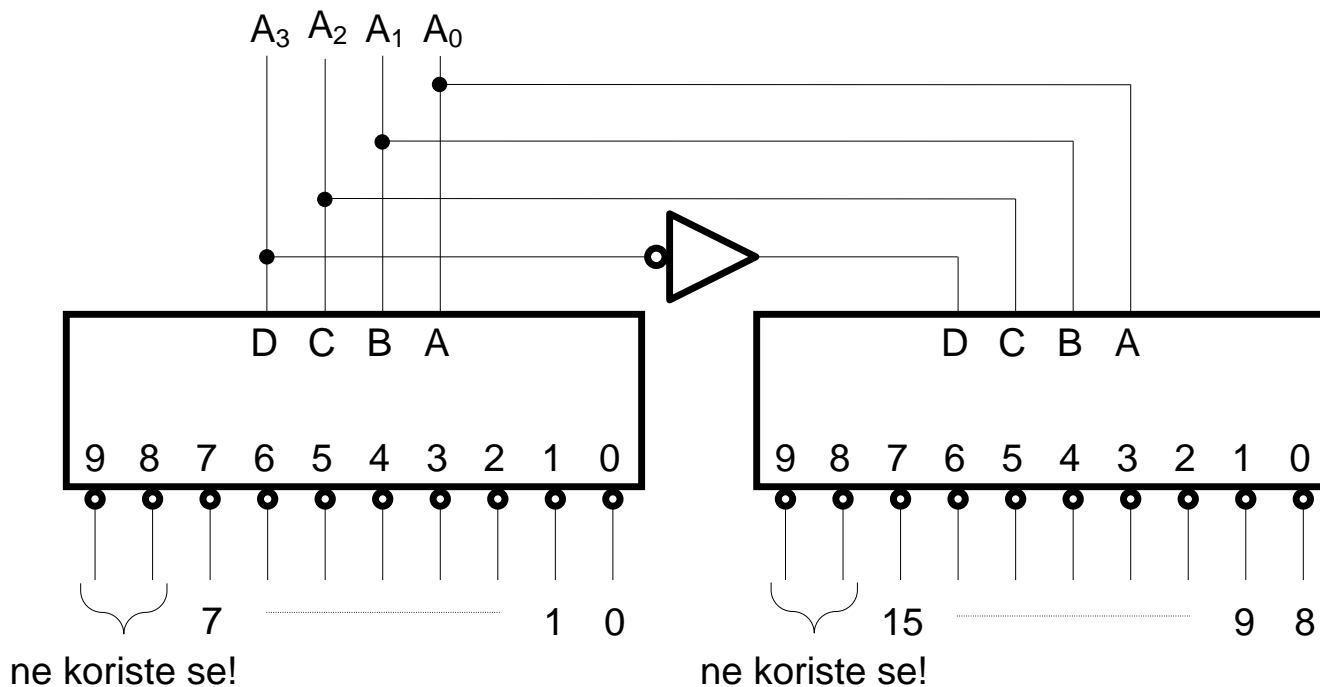
- oznake: D ~ najviša težina (2^3)
- izlazi *invertirani*:
zgodno kod kombiniranja sklopova
- uzorci 0 i 1 koji nisu kodne riječi:
 - ne prepoznaju se!
 - minimizacija sklopa
- varijante:
7443: XS-3
7444: XS-3 Gray



Dekadski dekoder

Primjer: izvedba dekodera "1-od-16" od dva 7442 :

- koristi se samo prvih 8 izlaza
~ dekodiraju se adresne varijable nižih težina B, C, A
- adresni ulaz D
~ odabir jednog od dva dekodera 3/8





Sadržaj predavanja

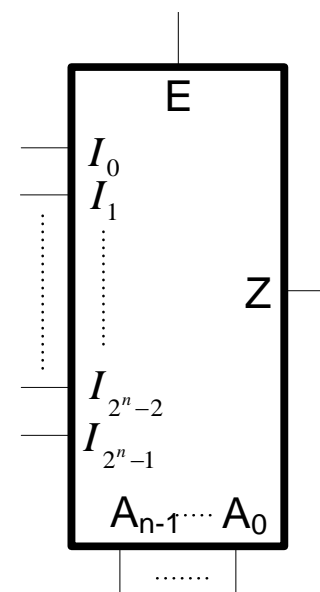
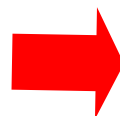
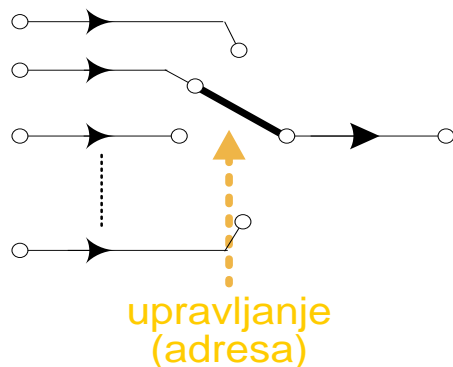
- kombinacijski moduli
- dekodeer
- **multiplesor**
 - **funkcionalnost multiplesora**
 - **multiplesorsko stablo**
 - **ostvarivanje Booleovih funkcija multiplesorom**
- prioritetni koder
- pretvornik koda
- komparator

Multiplesor

- multiplesor:
 - odabir podatka koji će se proslijediti na izlaz
~ "multipleksiranje"
 - funkcija *upravljanje preklopke*
(engl. rotary switch):



$$Z = \begin{cases} I_i & \text{za } i = A_{n-1} \dots A_0 \quad (\wedge E = 1) \\ 0 & \text{za } E = 0 \end{cases}$$

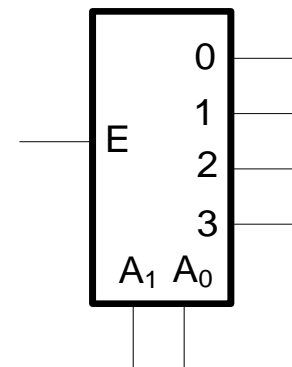
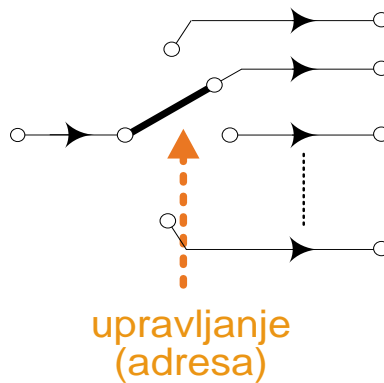


Multiplexsor

- *demultiplexsor*:

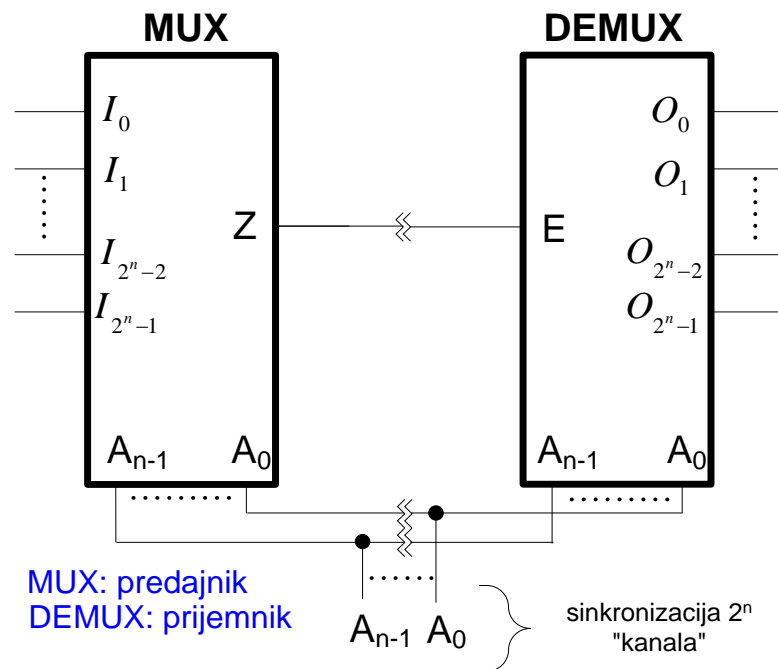
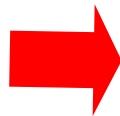
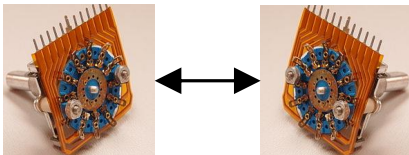
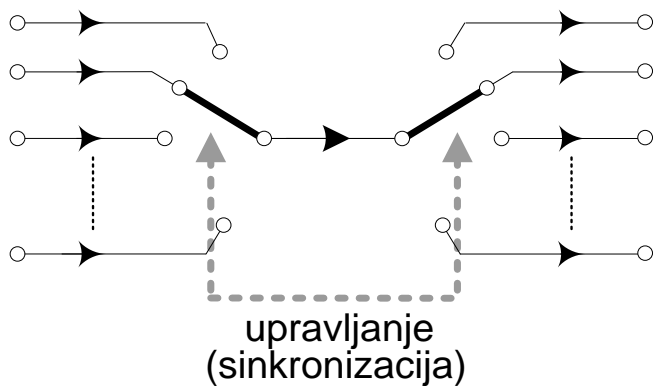
- dekodeer kojem se ulaz za omogućavanje E interpretira kao *ulaz za podatke*
- "usmjeravanje"/"raspodjela" ulaza na odabrani izlaz ~ "demultipleksiranje"

A ₁	A ₀	"0"	"1"	"2"	"3"
0	0	E	0	0	0
0	1	0	E	0	0
1	0	0	0	E	0
1	1	0	0	0	E



Multiplesor

- funkcija multipleksiranja:
 - ~ višestruko iskorištenje spojnih puteva:
 - prijenos različitih podataka *istim* fizičkim spojnim putem
 - ~ "više logičkih kanala preko jedne fizičke linije"
 - *vremenska podjela* (vremenski multipleks)

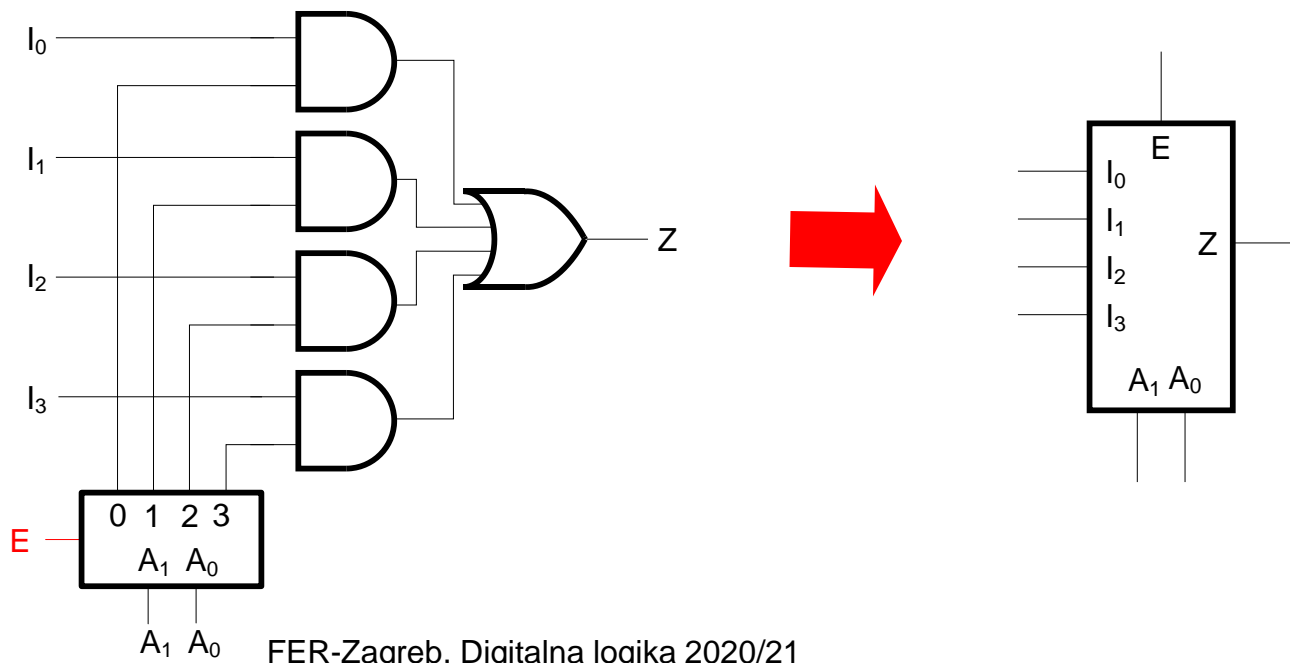


Multiplexsor

Primjer: multipleksor 4/1

- adrese se dekodiraju
- izlazi iz dekodera *koincidiraju* s ulazima I_i
~ propuštaju *samo jedan* od ulaza
na izlazni ILI sklop

E	A ₁	A ₀	Z
0	X	X	0
1	0	0	I_0
1	0	1	I_1
1	1	0	I_2
1	1	1	I_3



Multipleksor

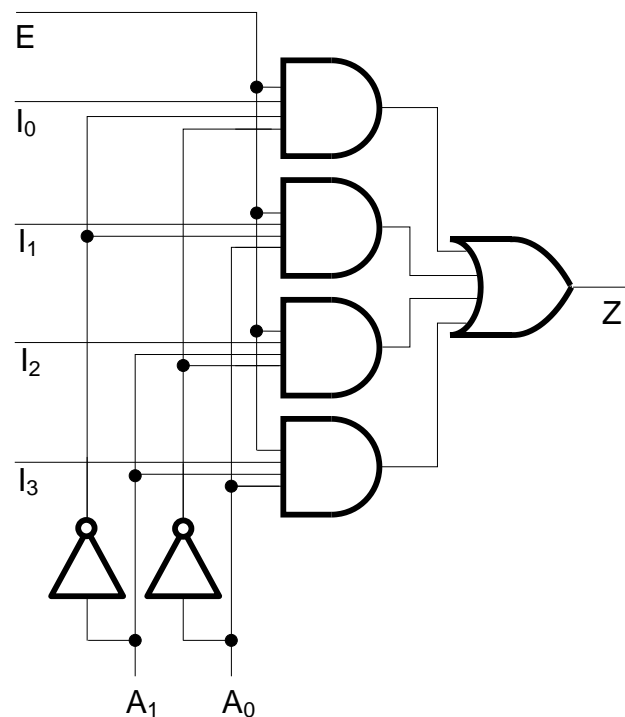
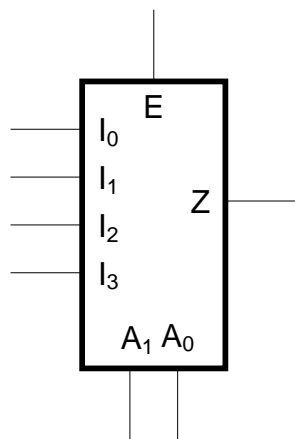
- *VHDL ponašajni model* multipleksora 4/1

```
entity mux41e is
  port (i: in std_logic_vector(0 to 3);
        a: in std_logic_vector(1 downto 0);
        e: in std_logic;
        z: out std_logic);
end mux41e;
```

```
architecture ponasajna of mux41e is
begin
  process (i,a,e)
  begin
    if(e = '0')
    then z <= '0';
    else
      case a is
        when "00" => z <= i(0);
        when "01" => z <= i(1);
        when "10" => z <= i(2);
        when "11" => z <= i(3);
        when others => z <= '0';
      end case;
    end if;
  end process;
end ponasajna;
```

Multipleksor

- funkcija I je *asocijativna*
~ "grupirati" I sklopove iz dekodera i koncidenciju s I_i

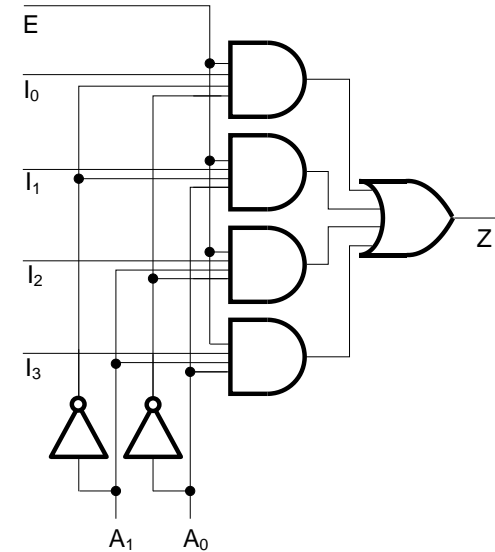


Multipleksor

- *VHDL strukturni model* multipleksora 4/1

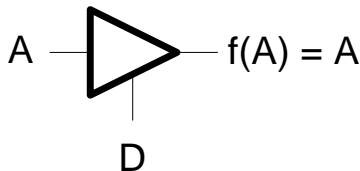
```
entity mux41e is
  port (i: in std_logic_vector(0 to 3);
        a: in std_logic_vector(1 downto 0);
        e: in std_logic;
        z: out std_logic);
end mux41e;

architecture strukturna of mux41e is
  signal a1_komplement, a0_komplement: std_logic;
  signal rez: std_logic_vector(0 to 3);
  begin
    sklop1: entity work.sklopNOT port map (a(1), a1_komplement);
    sklop2: entity work.sklopNOT port map (a(0), a0_komplement);
    sklop3: entity work.sklopAND4 port map (e, a1_komplement, a0_komplement, i(0), rez(0));
    sklop4: entity work.sklopAND4 port map (e, a1_komplement, a(0), i(1), rez(1));
    sklop5: entity work.sklopAND4 port map (e, a(1), a0_komplement, i(2), rez(2));
    sklop6: entity work.sklopAND4 port map (e, a(1), a(0), i(3), rez(3));
    sklop7: entity work.sklopOR4 port map (rez(0), rez(1), rez(2), rez(3), z);
  end strukturna;
```

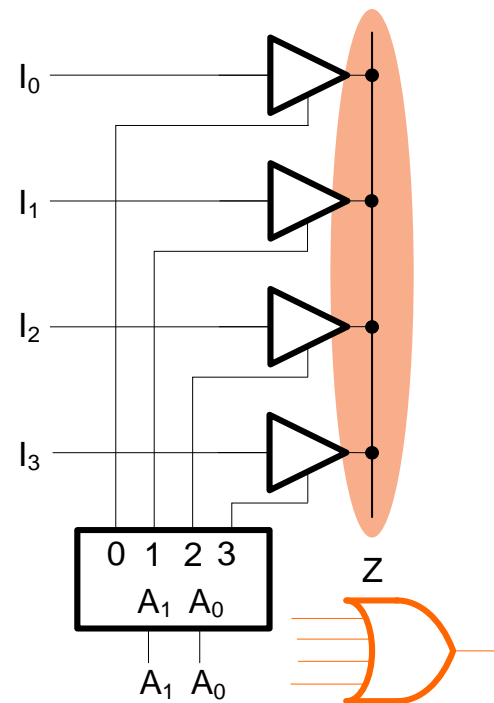
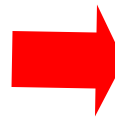
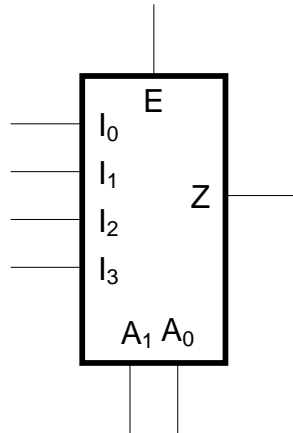


Multiplexsor

- izvedba funkcije multipleksiranja *na sabirničkoj liniji*:
 - izlazni ILI → "upravljani spojeni ILI"
 - ~ *samo jedan* sklop definira vrijednost V/N (~ 1/0)
 - *sklopovi s tri stanja* upravljani izlazima iz dekodera
 - (upravljani) *odvojni sklop*

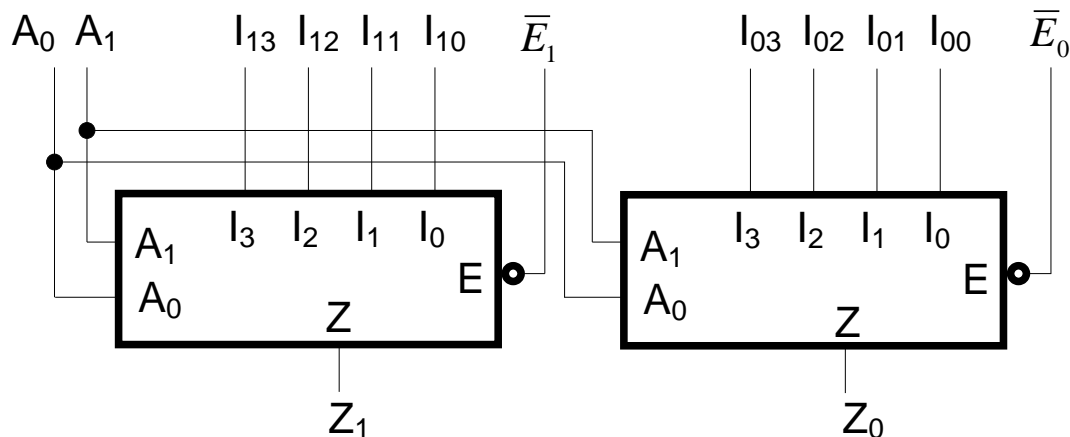


D	A	f
0	X	Z
1	0	0
1	1	1



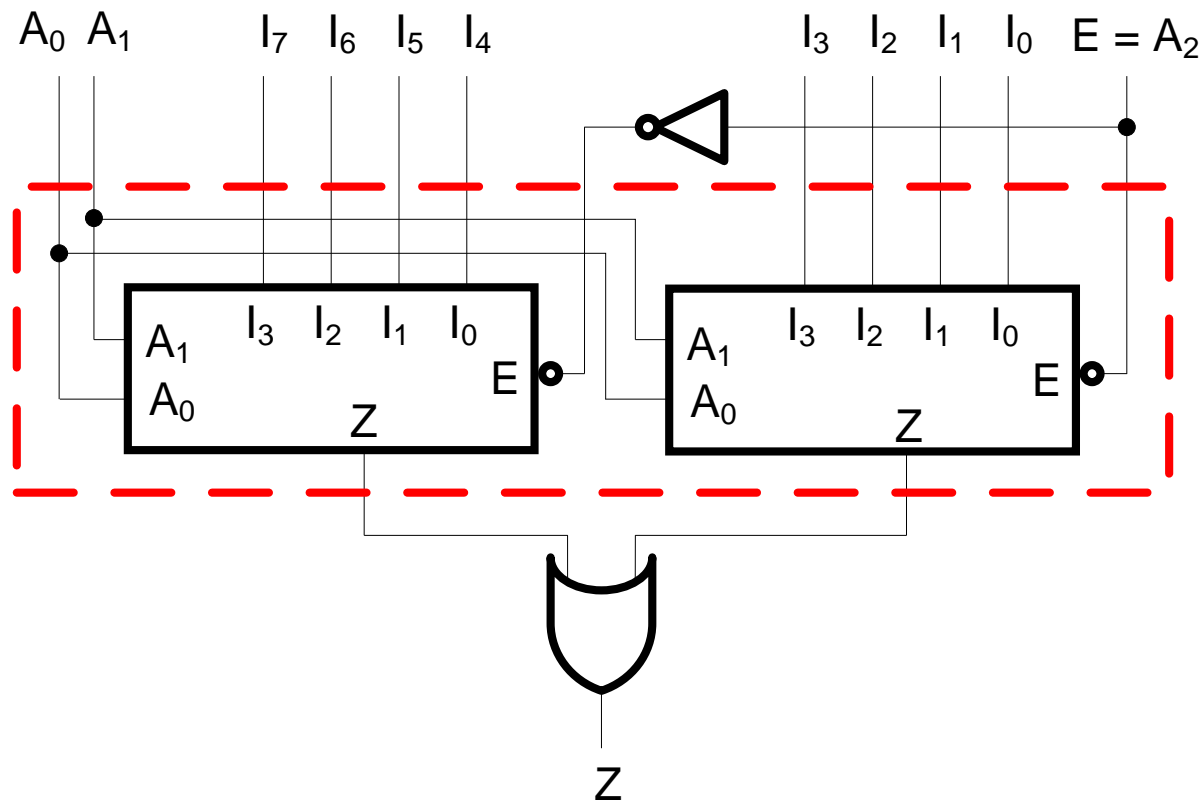
Multiplexsor

- izvedbe multipleksora
~ MSI modul; npr. 74153 (dvostruki četveroulazni)
 - *zajedničke* adrese A_1, A_0
 - *izdvojeni* ulazi za omogućavanje: $\overline{E}_1, \overline{E}_0$



Multiplexsor

Primjer: multiplexsor 8/1 na temelju 74153



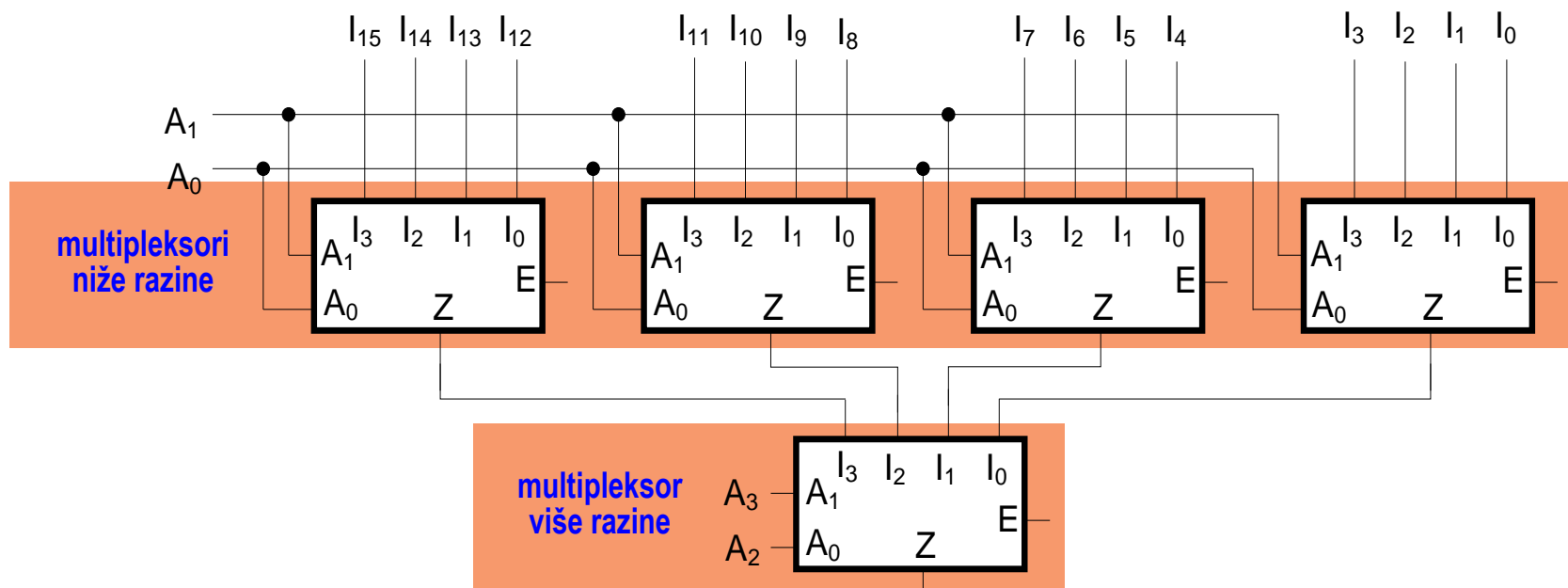


Multiplexsor

- multiplexsori s većim ($n > 16$) brojem ulaza:
 - izvedba jednim modulom *nepraktična*
~ presloženi MSI modul
 - radije *kaskadiranje* više jednostavnijih modula
~ *multiplexorsko stablo* (engl. multiplexer tree)
 - izgradnja multiplexorskog stabla:
 - podjela tablice definicije funkcija u podtablice
~ ulazi u MUX više razine
 - varijable viših težina na MUX "više razine"

Multiplexsor

Primjer: multiplexsor 16/1
kao multiplexsorsko stablo multiplesora 4/1



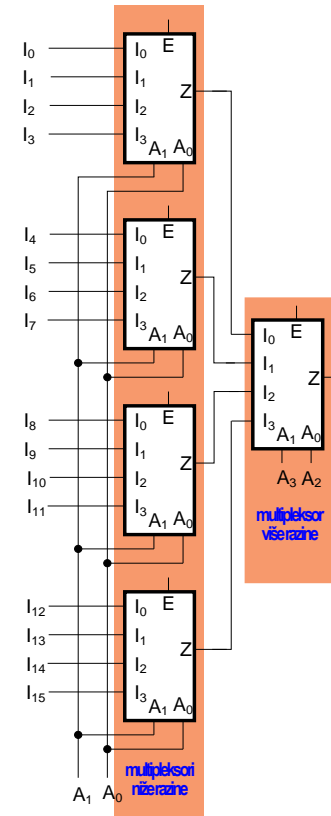
Multipleksor

- VHDL *strukturni* model multipleksorskog stabla 16/1:

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

entity mux16le is
    port (i: in std_logic_vector(0 to 16);
          a: in std_logic_vector(3 downto 0);
          e: in std_logic;
          z: out std_logic);
end mux16le;

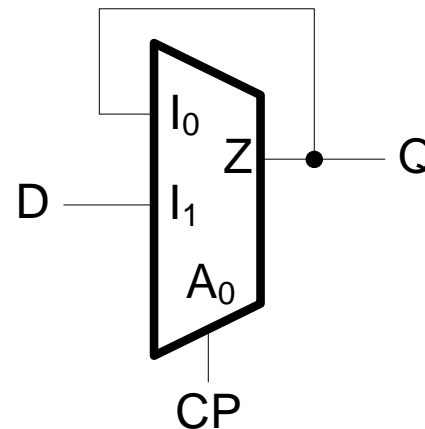
architecture strukturna of mux16le is
    signal rez: std_logic_vector(0 to 3);
begin
    sklop1: entity work.mux4le port map (i(0 to 3), a(1 to 0), e, rez(0));
    sklop2: entity work.mux4le port map (i(4 to 7), a(1 to 0), e, rez(1));
    sklop3: entity work.mux4le port map (i(8 to 11), a(1 to 0), e, rez(2));
    sklop4: entity work.mux4le port map (i(12 to 15), a(1 to 0), e, rez(3));
    sklop5: entity work.mux4le port map (rez(0 to 3), a(3 to 2), e, z);
end strukturna;
```



Multipleksor

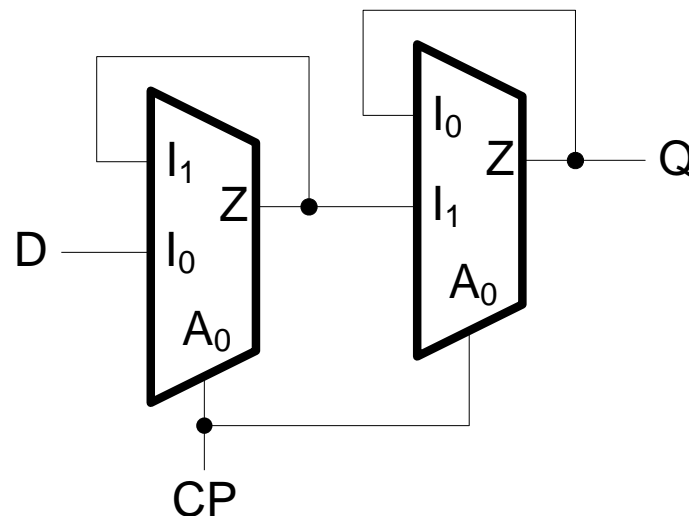
- izvedba *bistabila* multipleksorom:
 - "zabavljanje" podatka u multipleksoru
~ povratna veza s izlaza *na jedan* od ulaza
 - drugi ulaz za vanjski podatak
 - adresni ulaz za impuls takta
~ upravljanje razinom CP:
CP = 1 upis podatka

$A_0 = CP$	$Z = Q^{n+1}$
0	$I_0 = Q^n$
1	$I_1 = D$



Multiplesor

- izvedba *dvostrukog bistabila* multiplesorom:
 - po jedan multiplesor za svaki komponentni bistabil
 - *komplementarni* CP
~ odabir ulaza u multiplesore
 - funkcionalnost dvostrukog bistabila:
 - CP = 0 : podatak zapamćen u *izlaznom* MUX;
veza ulaza D i Z ulaznog MUX
 - CP = 1 : podatak zapamćen u *ulaznom* MUX;
veza Z ulaznog i Z izlaznog MUX



Ostvarivanje funkcija multipleksorom

- ostvarivanje logičkih funkcija multipleksorom:

- funkcija multipleksiranja:

m_i : minterm predstavlja adresu $(A_{n-1}A_{n-2}\dots A_1A_0)$

$$Z = \sum_{i=0}^{2^n-1} I_i \cdot m_i$$

- definicija funkcije od n varijabli u kanonskom disjunktivnom obliku:

$$f(x_{n-1}, \dots, x_0) = \sum_{i=0}^{2^n-1} \alpha_i \cdot m_i$$

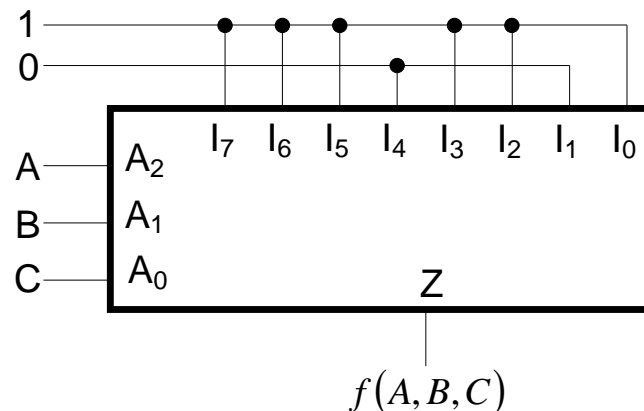
- pridruživanje: $\forall A_i = x_i, I_i = \alpha_i \Rightarrow Z = f(x_{n-1}, \dots, x_0)$

Ostvarivanje funkcija multipleksorom

Primjer: ostvarivanje funkcije tri varijable

$$f(A, B, C) = \sum m(0, 2, 3, 5, 6, 7)$$

- na podatkovne ulaze dovesti funkcijske vrijednosti (0 ili 1)
~ kao da su fiksno postavljeni bitovi:
"simulacija rada permanentne memorije (ROM)"
- neefikasno! ($\forall m_i \exists I_i$)



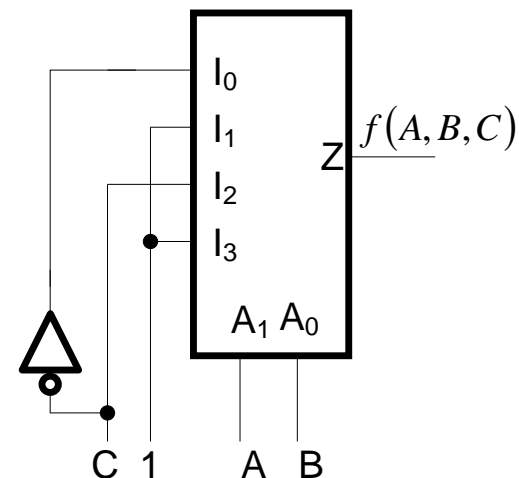
Ostvarivanje funkcija multipleksorom

- efikasnije rješenje
 - ~ primjena *trivijalnih rezidualnih funkcija*:
 - za $f(x_{n-1}, \dots, x_0)$ MUX s $n-1$ adresa
 - ~ 2^{n-1} podatkovnih ulaza:
 - 2 puta manje podatkovnih ulaza!
 - na *podatkovne* ulaze dovoditi funkcije varijable *najmanje* težine (ima ih 2^{n-1}):
$$\varphi(x_0) = \{0, 1, x_0, \bar{x}_0\}$$
 - na *adresne* ulaze dovoditi varijable viših težina x_{n-1}, \dots, x_1
 - $\varphi(x_0)$ su funkcije ostatka, *rezidualne funkcije*
 - rezidualne funkcije od jedne varijable
 - ~ *trivijalne* rezidualne funkcije

Ostvarivanje funkcija multipleksorom

Primjer: $f(A, B, C) = \sum m(0, 2, 3, 5, 6, 7)$

A ₁	A ₀		ADRESIRANI		
A	B	C	ULAZ	f	
0	0	0	I ₀	1	\overline{C}
0	0	1		0	
0	1	0	I ₁	1	1
0	1	1		1	
1	0	0	I ₂	0	C
1	0	1		1	
1	1	0	I ₃	1	1
1	1	1		1	





Ostvarivanje funkcija multipleksorom

- rješenje *netrivijalnim* rezidualnim funkcijama:
 - "netrivijalne" rezidualne funkcije (>1 varijable):
 - multipleksorskim stablom se ostvaruje podjela tablice definicije funkcije
 - rezidualne funkcije se ostvaruju posebno, moguće "drugim sredstvima"
 - obično presloženo rješenje
 - ostvarenje *kanonskog* oblika funkcije
~ *nema* minimizacije
 - pojednostavljivanje rješenja
~ odabir prikladnog pridruživanja varijabli adresnim ulazima

Ostvarivanje funkcija multipleksorom

Primjer: $f(A, B, C, D, E) = \sum m(0, 1, 2, 5, 6, 8, 13, 14, 15, 16, 21, 26, 28, 30, 31)$

- ostvarenje 32-ulaznim MUX
~ simulacija permanentne memorije:
 $ABCDE \rightarrow A_4A_3A_2A_1A_0; I_i \in \{0, 1\}$
- ostvarenje 16-ulaznim MUX
~ standardno rješenje trivijalnim rezidualnim funkcijama:
 $ABCD \rightarrow A_3A_2A_1A_0; I_i = \varphi_i(E)$
- ostvarenje 8-ulaznim MUX
~ rezidualne funkcije od 2 varijable:
 $ABC \rightarrow A_2A_1A_0; I_i = \varphi_i(D, E)$

Ostvarivanje funkcija multipleksorom

- ostvarenje 8-ulaznim MUX
~ rezidualne funkcije od 2 varijable

$$f(A, B, C, D, E) = \sum m(0, 1, 2, 5, 6, 8, 13, 14, 15, 16, 21, 26, 28, 30, 31)$$

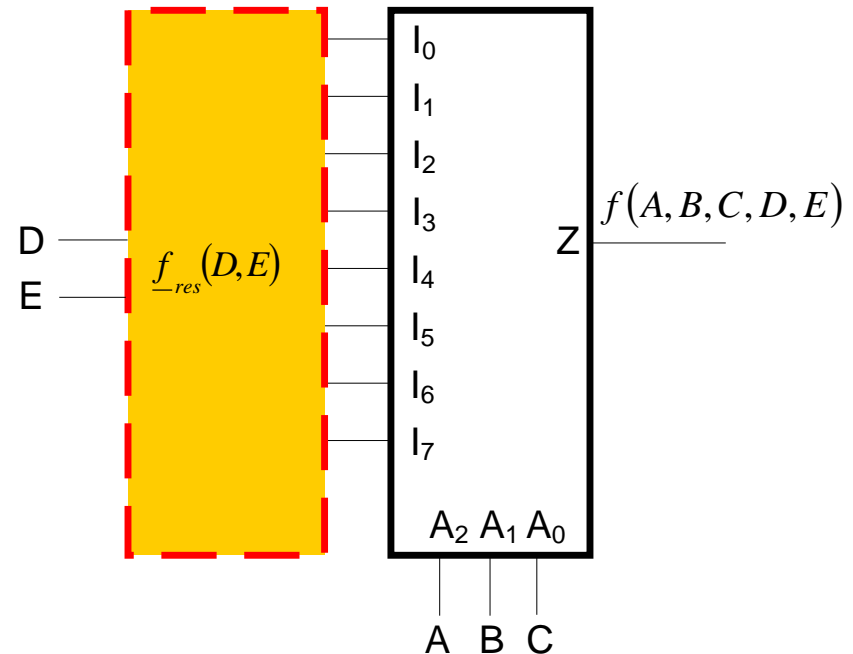
DE	I_0 000	I_2 010	I_6 110	I_4 100	I_1 001	I_3 011	I_7 111	I_5 101	ABC
00	1	1		1			1		
01	1				1	1		1	
11						1	1		
10	1		1		1	1	1		
	I_0	I_2	I_6	I_4	I_1	I_3	I_7	I_5	

Ostvarivanje funkcija multipleksorom

- ostvarenje 8-ulaznim MUX
 \sim rezidualne funkcije od 2 varijable: $ABC \rightarrow A_2A_1A_0$

$$f(A, B, C, D, E) = \sum m(0, 1, 2, 5, 6, 8, 13, 14, 15, 16, 21, 26, 28, 30, 31)$$

i	A	B	C	$f_{\text{res}}=I_i$
	A_2	A_1	A_0	
0	0	0	0	$\overline{D}\overline{E}$
1	0	0	1	$D \oplus E$
2	0	1	0	$\overline{D}\overline{E}$
3	0	1	1	$D + E$
4	1	0	0	$\overline{D}\overline{E}$
5	1	0	1	$\overline{D}E$
6	1	1	0	$D\overline{E}$
7	1	1	1	$D + \overline{E}$

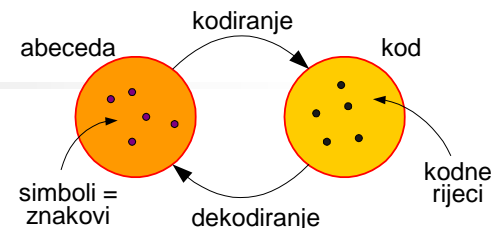




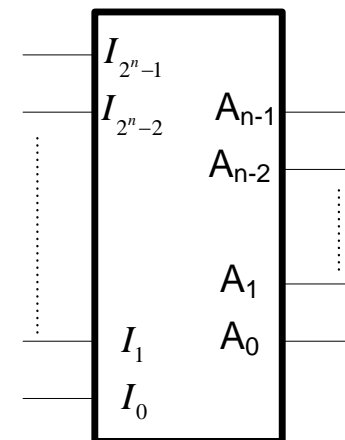
Sadržaj predavanja

- kombinacijski moduli
- dekodeer
- multipleksor
- **prioritetni koder**
- pretvornik koda
- komparator

Prioritetni koder



- funkcija *kodiranja*
 - ~ generiranje *binarne* kodne riječi nekog koda; prethodno osigurati da je *samo jedan* ulaz aktivan!
- koder
 - ~ aktivan *samo jedan* ulaz (npr. $I_i = 1$)
 2^n ulaza $\rightarrow n$ izlaza
- *prioritetni* koder
 - ~ aktivno više ulaza, samo jedan djeluje! (onaj s najvećim indeksom)
- tipična oznaka:
 $2^{\langle \text{broj adresa} \rangle} / \langle \text{broj adresa} \rangle$



Prioritetni koder

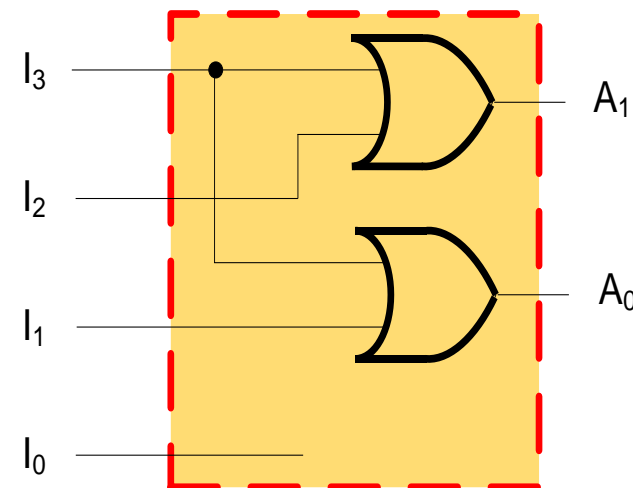
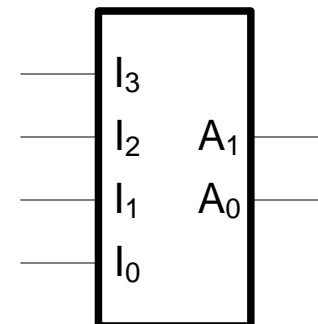
Primjer: koder 4/2

- (još jednom!) ograničenje:
uzorci ulaza s više 1 *ne mogu* se pojaviti
- uvijek jedan ulaz aktivan!
- simbol "0" ($\sim I_0 = 1$) daje $A_1A_0 = 00$
~ ne utječe!

I_3	I_2	I_1	I_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$$A_1 = I_3 + I_2$$

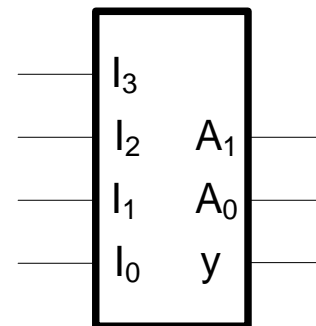
$$A_0 = I_3 + I_1$$



Prioritetni koder

- *prioritetni koder* (engl. priority encoder)
 - ~ rješenje problema više aktivnih ulaza:
 - djeluje ulaz najvišeg prioriteta
 - svi ulazi = 0 ?
 - ~ poseban izlaz pokazuje valjanost očitavanja izlaza:
 $y = \text{if } I_3 \text{ or } I_2 \text{ or } I_1 \text{ or } I_0 \text{ then } 1 \text{ else } 0$

I_3	I_2	I_1	I_0	A_1	A_0	y
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1



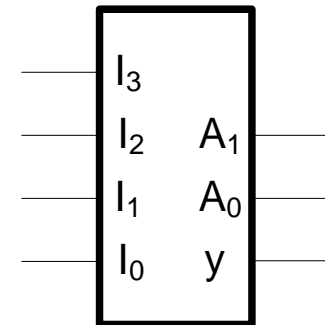
Prioritetni koder

- VHDL *ponašajni* model prioritetnog koder 2/4

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

entity priorityEncoder is
  port (I: in std_logic_vector(3 downto 0);
        A: out std_logic_vector(1 downto 0);
        y: out std_logic);
end priorityEncoder;

architecture ponasajna of priorityEncoder is
begin
  A <= "11" when I(3)='1' else
       "10" when I(2)='1' else
       "01" when I(1)='1' else
       "00";
  y <= '0' when I="0000" else '1';
end ponasajna;
```



Prioritetni koder

Primjer: prioritetni koder 4/2 [Brown i Vranešić, 2000]

I_3	I_2	I_1	I_0	A_1	A_0	y
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1



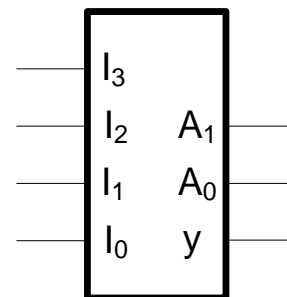
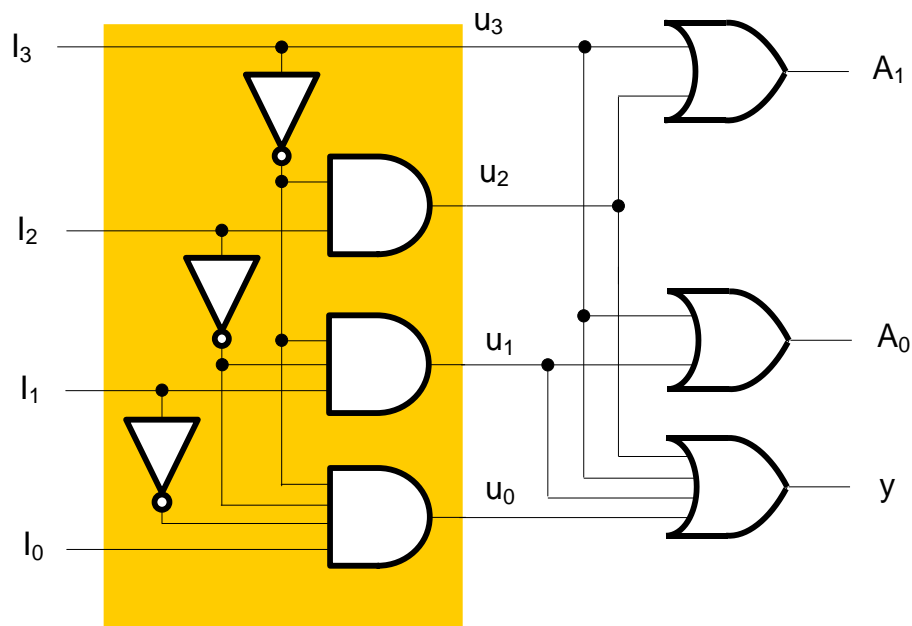
$$u_3 = \overline{I_3}$$

$$u_2 = \overline{I_3} \cdot \overline{I_2}$$

$$u_1 = \overline{I_3} \cdot \overline{I_2} \cdot \overline{I_1}$$

$$u_0 = \overline{I_3} \cdot \overline{I_2} \cdot \overline{I_1} \cdot I_0$$

$$\Rightarrow \begin{aligned} A_1 &= u_3 + u_2 \\ A_0 &= u_3 + u_1 \\ y &= u_3 + u_2 + u_1 + u_0 \end{aligned}$$



Prioritetni koder

- VHDL *strukturni* model prioritetnog koda 2/4

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

```
entity priorityEncoder is  
  port (I: in  std_logic_vector(3 downto 0);  
        A: out std_logic_vector(1 downto 0);  
        y: out std_logic );  
end priorityEncoder;
```

```
architecture strukturna of priorityEncoder is  
  signal n3,n2,n1,u3,u2,u1,u0: std_logic;
```

```
begin
```

```
  sklop1: entity work.sklopNOT port map (I(3),n3);
```

```
  sklop2: entity work.sklopNOT port map (I(2),n2);
```

```
  sklop3: entity work.sklopNOT port map (I(1),n1);
```

```
  u3 <= I(3);
```

```
  sklop4: entity work.sklopAND2 port map (n3,I(2),u2);
```

```
  sklop5: entity work.sklopAND3 port map (n3,n2,I(1),u1);
```

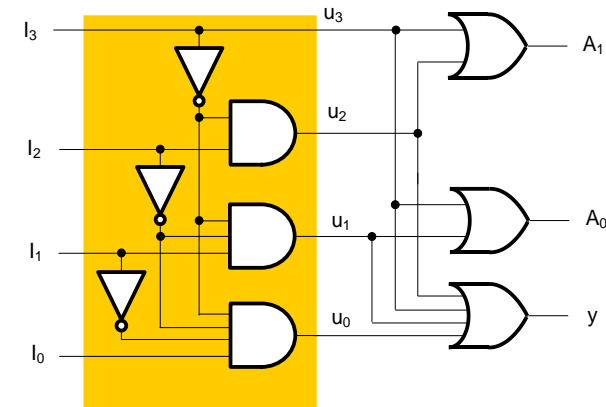
```
  sklop6: entity work.sklopAND4 port map (n3,n2,n1,I(0),u0);
```

```
  sklop7: entity work.sklopOR2 port map (u3,u2,A(1));
```

```
  sklop8: entity work.sklopOR2 port map (u3,u1,A(0));
```

```
  sklop9: entity work.sklopOR4 port map (u3,u2,u1,u0,y);
```

```
end strukturna;
```



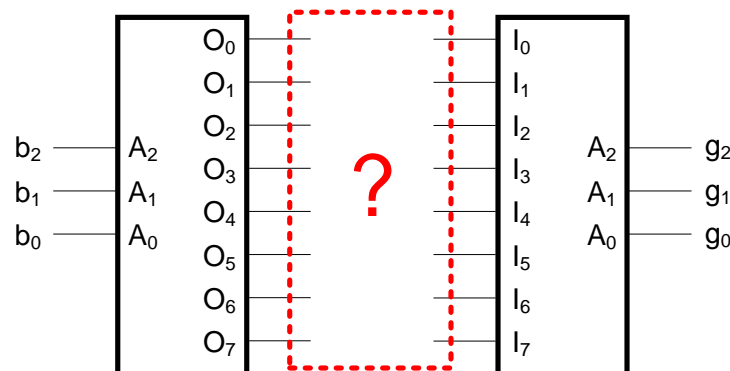


Sadržaj predavanja

- kombinacijski moduli
- dekodeer
- multipleksor
- prioritetni koder
- **pretvornik koda**
- komparator

Pretvornik koda

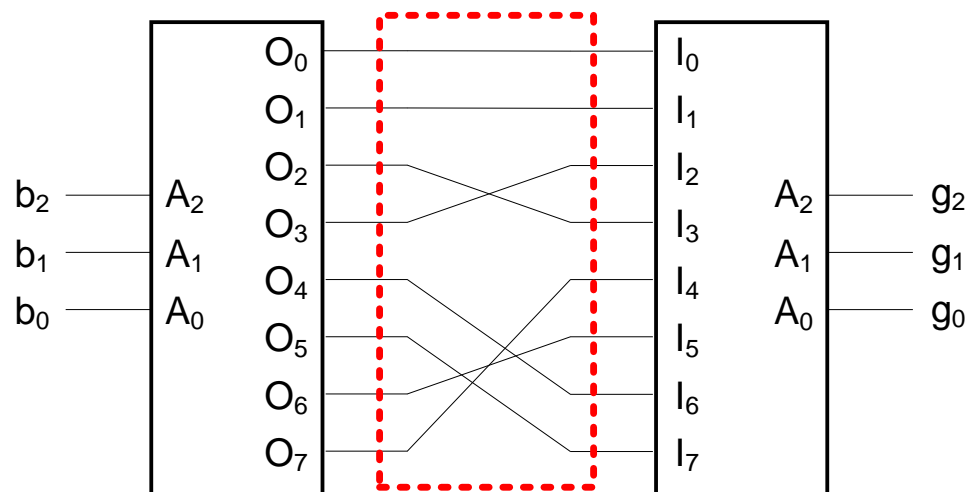
- *pretvornik koda* (engl. code converter):
 - pretvorba kodne riječi *jednog* koda u kodnu riječ *nekog drugog* koda (*različiti* kodovi!)
 - isti princip kao kod dekodera i koder:
 - dekodер
 - ~ kodna riječ
 - 1 aktivni izlaz
 - koder
 - ~ 1 aktivni ulaz
 - kodna riječ
 - različiti tipovi MSI modula
 - + mogućnost *kaskadiranja* (2-dimenzijske strukture)



Pretvornik koda

Primjer: pretvornik 3-bitnog binarnog u Grayev kod

i	$b_2b_1b_0$	$g_2g_1g_0$	j
0	000	000	0
1	001	001	1
2	010	011	3
3	011	010	2
4	100	110	6
5	101	111	7
6	110	101	5
7	111	100	4



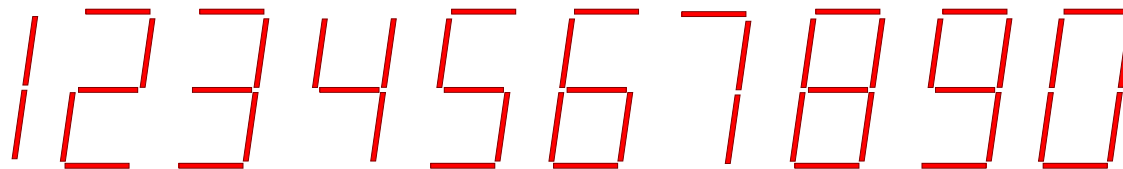
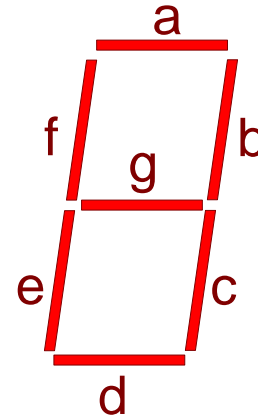
$O_0 \rightarrow l_0; O_1 \rightarrow l_1; O_2 \rightarrow l_3; O_3 \rightarrow l_2;$

$O_4 \rightarrow l_6; O_5 \rightarrow l_7; O_6 \rightarrow l_5; O_7 \rightarrow l_4;$

Pretvornik koda

Primjer: pretvornik BCD koda u 7-segmentni

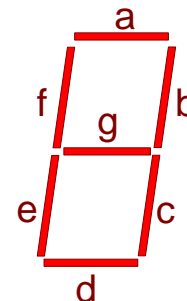
- vrlo raširena primjena
~ prikaz BCD znamenki
- element za prikaz
~ *7-segmentni prikaz*
(engl. 7-segment display)



Pretvornik koda

- tablica pretvorbe BCD u 7-segmentni kod

1234567890

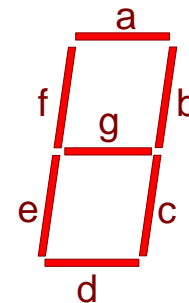


	D_3 2^3 8	D_2 2^2 4	D_1 2^1 2	D_0 2^0 1	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Pretvornik koda

Zadatak:

- napisati minimalne izraze za a, b, ..., g
- nacrtati sklop
- napisati VHDL ponašajni / strukturni model
- ponoviti sve za pretvornik *heksadekadskih brojeva* u 7-segmentni kod



1 2 3 4 5 6 7 8 9 0 A B C d e F

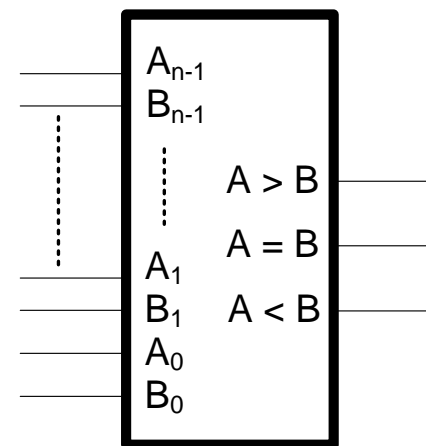


Sadržaj predavanja

- kombinacijski moduli
- dekodeer
- multipleksor
- prioritetni koder
- pretvornik koda
- **komparator**

Komparator

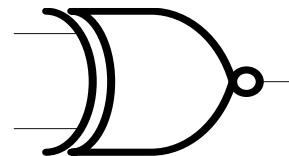
- *komparator*
 - ~ sklop za usporedbu dva n -bitna broja (npr. A i B)
 - obično cijeli brojevi *bez* predznaka
 - mogućnosti ~ izlazi modula:
 - $A = B$
 - $A > B$
 - $A < B$
 - MSI modul
 - ~ 4-bitni
 - + mogućnost *kaskadiranja*



Komparator

- izgradnja komparatora (jedna mogućnost):

- usporedba po bitovima
~ sklop EX-NILI ($\bar{A} \cdot \bar{B} + A \cdot B$)



- izlaz $A = B$
~ I funkcija usporedbi po bitovima
- izlaz $A > B$
~ dominira prvi bit sa svojstvom $A_i > B_i$
(počev od bita najviše težine)
- izlaz $A < B$
~ **not** (($A > B$) **or** ($A = B$))

Komparator

Primjer: 4-bitni komparator

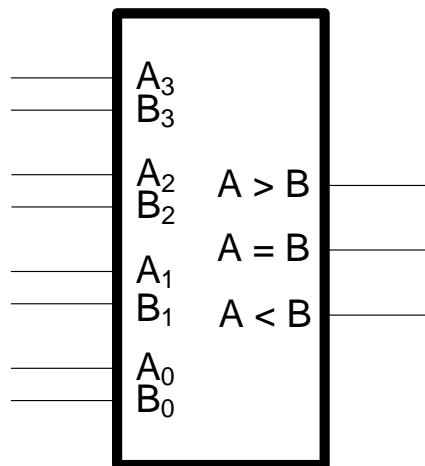
- usporedba po bitovima: $u_i = a_i \otimes b_i = \overline{a_i \oplus b_i}, \quad i = 0, \dots, 3$
- izlaz $A = B$: " $A = B$ " = $u_3 \cdot u_2 \cdot u_1 \cdot u_0$
- izlaz $A > B$
~ *rekurzivno* utvrđivanje $a_i > b_i$, od bita najviše težine:
" $a_3 > b_3$ " = $a_3 \cdot \overline{b_3}$
" $a_2 > b_2$ " = $a_2 \cdot \overline{b_2} \cdot u_3$
" $a_1 > b_1$ " = $a_1 \cdot \overline{b_1} \cdot u_3 \cdot u_2$
" $a_0 > b_0$ " = $a_0 \cdot \overline{b_0} \cdot u_3 \cdot u_2 \cdot u_1$

" $A > B$ " = " $a_3 > b_3$ " + " $a_2 > b_2$ " + " $a_1 > b_1$ " + " $a_0 > b_0$ "
- izlaz $A < B$: " $A < B$ " = $\overline{"A = B" + "A > B"} = \overline{"A = B"} \cdot \overline{"A > B"}$

Komparator

Zadatak:

- nacrtati sklop 4-bitnog komparatora
- napisati tablicu kombinacija
- napisati VHDL ponašajni model sklopa
- napisati VHDL strukturni model sklopa



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

- kombinacijski moduli: str. 253-254
- dekodер: str. 254-259
- multipleksor
(funcionalnost, multipleksorsko stablo):
str. 260-264, 266-267
- ostvarivanje Booleovih funkcija multipleksorom:
str. 264-266



Zadaci za vježbu (1)

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

- dekode: 7.1, 7.3-7.6
- ostvarivanje Booleovih funkcija multipleksorom:
7.7-7.26
- pretvornik koda: 7.47

Zadaci za vježbu (2)

M. Čupić: *Digitalna elektronika i digitalna logika. Zbirka riješenih zadataka*, Cjelina 5: Standardni kombinacijski moduli.

- dekodер:
 - riješeni zadaci: 5.2, 5.3, 5.9, 5.10, 5.11a, 5.12a
 - zadaci za vježbu: 6-12, 15, 19 (VHDL)
- multipleksor (funcionalnost, multipleksorsko stablo):
 - riješeni zadaci: 5.1
 - zadaci za vježbu: 1-5, 13-14
- ostvarivanje Booleovih funkcija multipleksorom:
 - riješeni zadaci: 5.5-5.8, 5.11b, 5.16 (VHDL), 5.17
 - zadaci za vježbu: 16-18, 20-29, 32
- komparator
 - zadaci za vježbu: 30, 31