Konkurentne naredbe dodele

- Tipičan digitalni sistem se sastoji od više podsistema koji rade nezavisno jedan od drugog, tj. istovremeno ili paralelno.
- Iako se funkcija svakog podsistema može opisati sekvencijalnim programom, za modeliranje paralelizma na sistemskom nivou neophodna je posebna podrška u jeziku za opis hardvera.
- U VHDL-u takva podrška postoji u vidu konkurentnih naredbi, koje se slično podsistemima ili kolima izvršavaju (tj. rade) istovremeno.

Konkurentne naredbe dodele

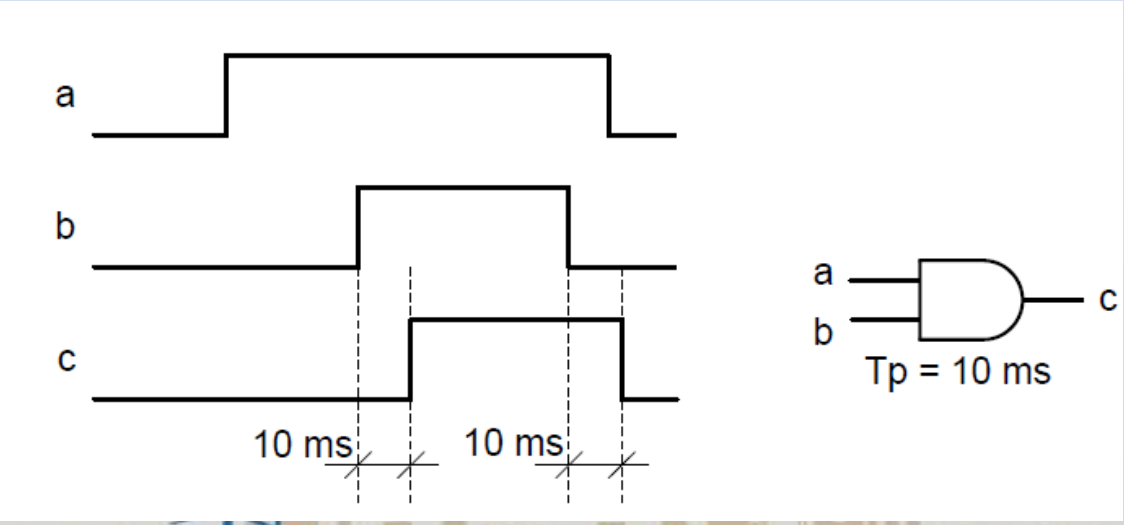
- Kod je sadržan u sekciji ARCHITECTURE:
- **Jednostavna konkurentna naredba dodele**
- **Naredba WHEN -** uslovna naredba dodele
- **Naredba SELECT -** naredba dodele sa izborom vrednosti

Jednostavna konkurentna nareba dodele

- *signal <= izraz;*
- i*zraz* – logički ili aritmetički izraz (sadrži operatore AND, NOT, +, sll i sl.)
- **s <= a AND (b OR c);**
- **enable <= ´1´;**
- **sel <= (r1 AND r2) OR (r3 AND r4);**
- **sum <= a + b + c – 1;**
- Koristi se za opis jednostavnih logičkih i aritmetičkih funkcija.

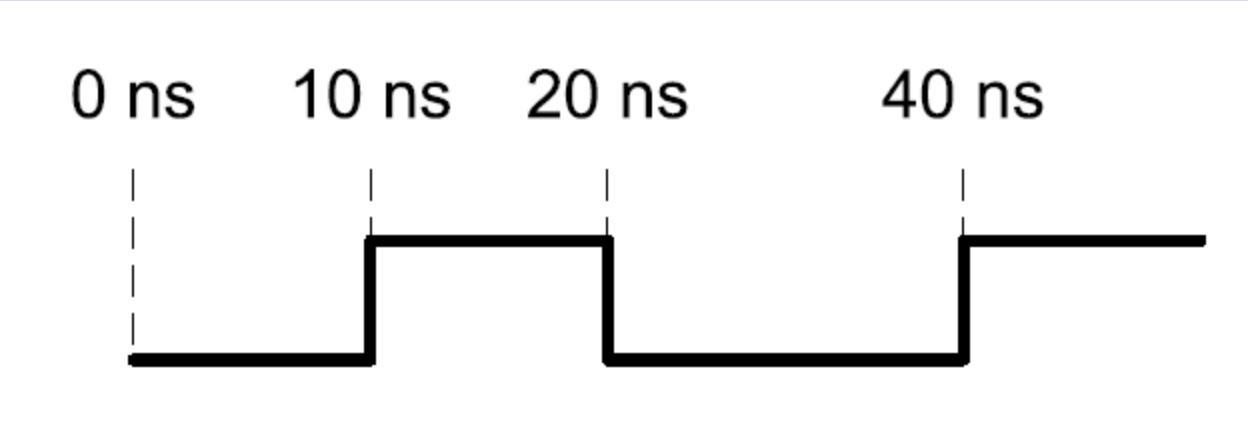
Jednostavna konkurentna nareba dodele

- Jednostavna konkurentna naredba dodele sa klauzulom *after*:
- *signal* <= *izraz* **after** *kašnjenje;*
- **c <= a AND b after 10 ns;**
- Koristi se u simulaciji. U sintezi nijie dozvoljena.



Generisanje talasnog oblika

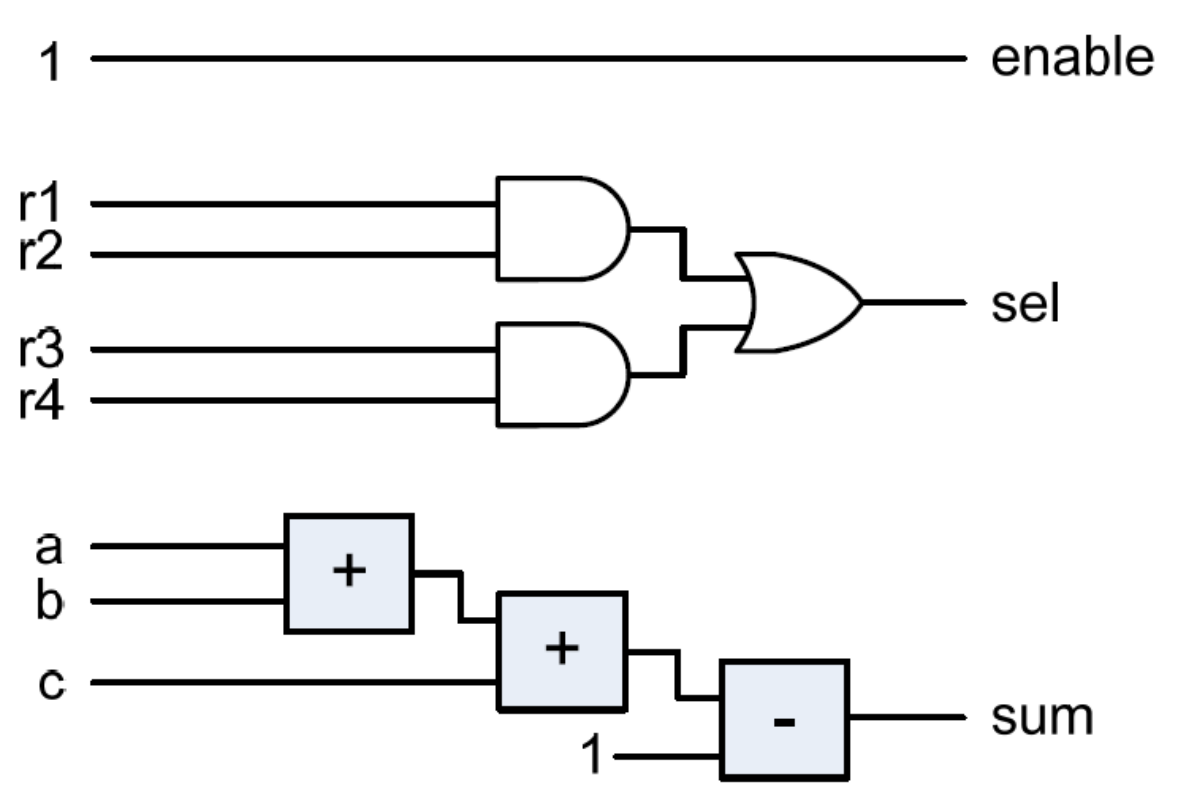
- **sig <= ´0´, ´1´ after 10 ns, ´0´ after 10 ns, ´1´ after 20 ns;**



- Tipična primena je u testbenču za generisanje pobudnih signala složenog talasnog oblika.

Konceptualna implementacija konkurentne naredbe dodele

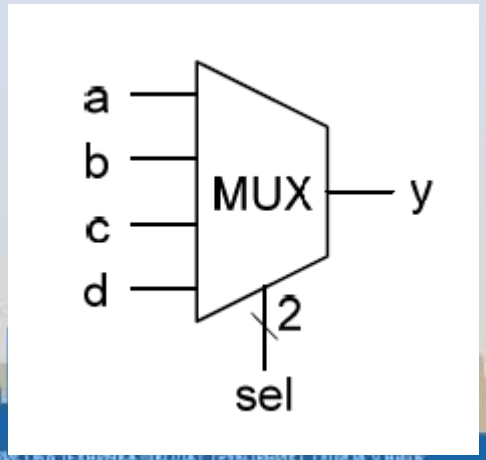
- **enable <= ´1´;**
- **sel <= (r1 AND r2) OR (r3 AND r4);**
- **sum <= a + b + c – 1;**

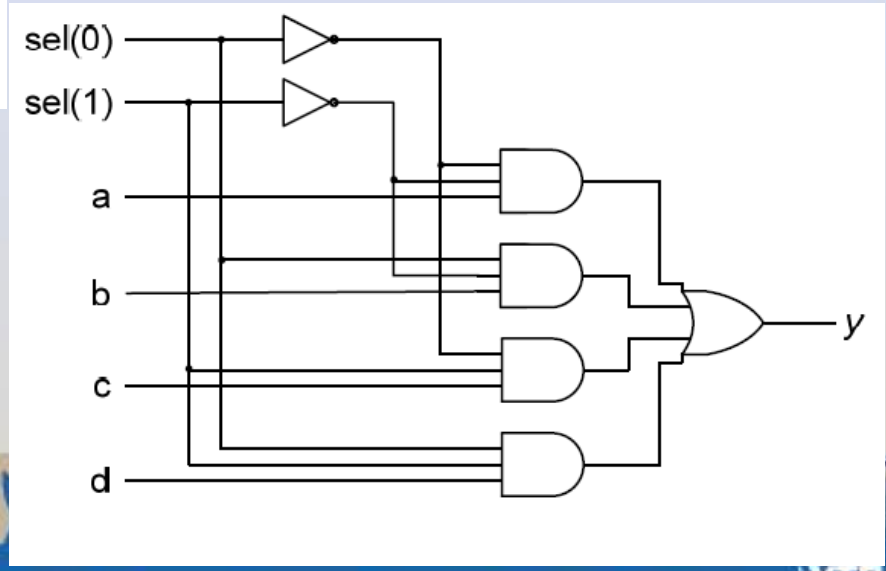


Konkurentna nareba dodele Primer: multiplekser 4-u-1

```
 1  LIBRARY ieee;
 2  USE ieee.std_logic_1164.all;
 3
 4  ENTITY mux IS
 5  PORT ( a,b,c,d,s0,s1 : IN STD_LOGIC;
 6  y : OUT STD_LOGIC);
 7  END mux;
 8
 9  ARCHITECTURE log_funk OF mux IS
10  BEGIN
11  y <= (a AND NOT s1 AND NOT s0) OR
12  (b AND NOT s1 AND s0) OR
13  (c AND s1 AND NOT s0) OR
14  (d AND s1 AND s0);
15  END log_funk;
16
```

"Nizak" nivo opisa





Konkurentna naredba dodele sa zatvorenom petljom

- **q** <= (**q** AND NOT en) OR (d AND en);

Nije zabranjeno, ali nije preporučljivo !
(stvaraju se interna stanja i mogu se izazvati oscilacije u kolu)

Za en='1', q = d.
Međutim, za en = '0', q = q - zadržava svoju vrednost => stvara se memorija.

- **q** <= (NOT **q** AND NOT en) OR (d AND en);

Za en='0', q = NOT q - oscilovanje !

Za modelovanje sekvencijalnog ponašanja koriste se sekvencijalne naredbe !

WHEN - konkurentna naredba uslovne dodele

- Uopštenje konkurentne naredbe dodele.
- S desne strane znaka <= može se naći više od jednog izraza.
- Sintaksa:

sig <= izraz_1 WHEN uslov_1 ELSE

izraz_2 WHEN uslov_2 ELSE

...

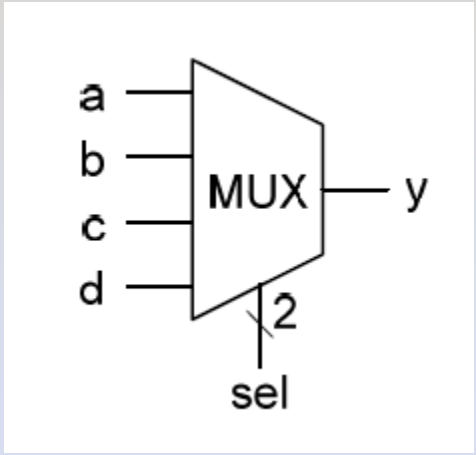
izraz_n-1 WHEN uslov_n-1 ELSE

izraz_n;

- ″uslov″ - logički/relacioni izraz koji može biti tačan ili netačan, npr: (a=’1’ OR b=’0’) ili (a > b)
- **″uslovi″ se ispituju redom**, a signal dobija vrednost prvog izraza čiji je uslov tačan
- Ako ni jedan uslov nije tačan, izvršava se izraz iz poslednje grane.

WHEN Primer: Multiplekser

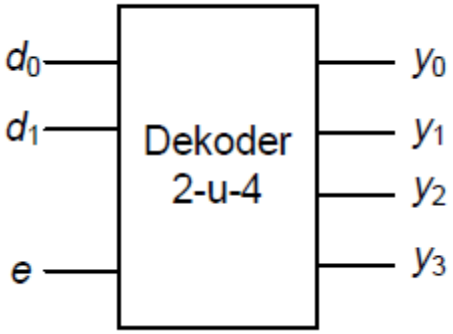
```
 1  LIBRARY ieee;
 2  USE ieee.std_logic_1164.all;
 3
 4  ENTITY mux IS
 5  PORT ( a,b,c,d : IN STD_LOGIC;
 6  sel: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
 7  y : OUT STD_LOGIC);
 8  END mux;
 9
10  ARCHITECTURE when_arch OF mux IS
11  BEGIN
12  y <= a WHEN sel = "00" ELSE
13  b WHEN sel = "01" ELSE
14  c WHEN sel = "10" ELSE
15  d;
16  END when_arch;
```



d pokriva sel=″11″, ali i sve preostale, nebinarne kombinacije, kao što su ″0-″, ″LH″, ″ZW, …

WHEN Primer: Dekoder

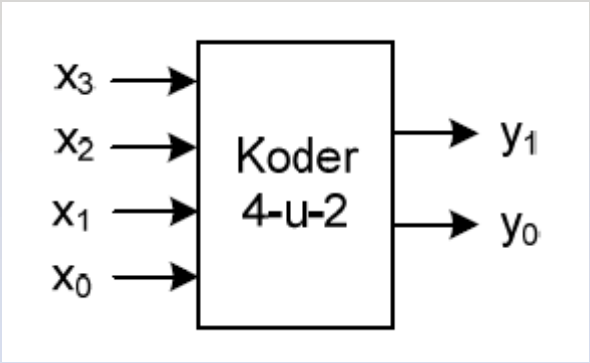
```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY dek2u4 IS
5  PORT (d : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
6  e : IN STD_LOGIC;
7  y : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
8  END dek2u4;
9
10 ARCHITECTURE when_arch OF dek2u4 IS
11 BEGIN
12 y <= "0000" WHEN e = '0' ELSE
13 "0001" WHEN d = "00" ELSE
14 "0010" WHEN d = "01" ELSE
15 "0100" WHEN d = "10" ELSE
16 "1000";
17 END when_arch;
```



e	d_1	d_0	y_3	y_2	y_1	y_0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0
0	X	X	0	0	0	0

WHEN Primer: Koder

```
 1  LIBRARY IEEE;
 2  USE IEEE.STD_LOGIC_1164.ALL;
 3
 4  ENTITY encoder IS
 5  PORT (x : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
 6  y : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
 7  END encoder;
 8
 9  ARCHITECTURE when_arch OF encoder IS
10  BEGIN
11  y <= "00" WHEN x="0001" ELSE
12  "01" WHEN x="0010" ELSE
13  "10" WHEN x="0100" ELSE
14  "11";
15  END
16  when_arch;
```

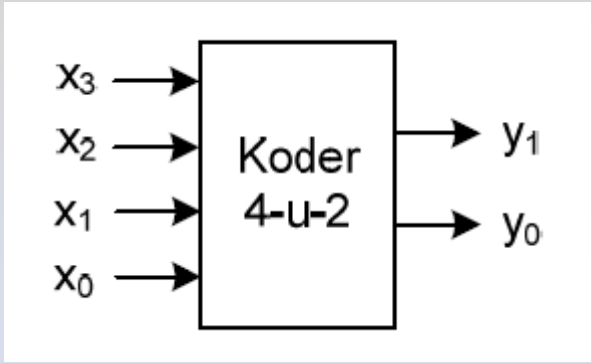


x_3	x_2	x_1	x_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
sve ostale komb.				-	-

- Uočimo da u datom VHDL opisu neregularne ulazne kombinacije nisu tretirane izdvojeno, već su sve one implicitno pripojene jednoj regularnoj ulaznoj kombinaciji (tj. kombinaciji ″1000″). To nije pogrešno, ali se gubi mogućnost za logičku minimizaciju, jer se od hardvera zahteva da za svaku neregularnu ulaznu kombinaciju na izlazu uvek postavi istu vrednost, ″11″.

WHEN Primer: Koder

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY encoder IS
5  PORT (x : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
6  y : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
7  END encoder;
8
9  ARCHITECTURE when_arch OF encoder IS
10 BEGIN
11 y <= "00" WHEN x="0001" ELSE
12 "01" WHEN x="0010" ELSE
13 "10" WHEN x="0100" ELSE
14 "11" WHEN x="1000" ELSE
15 "--";
16 END
17 when_arch;
```

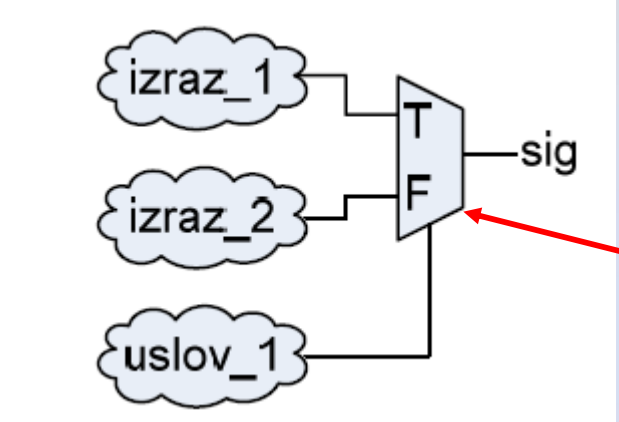


x_3	x_2	x_1	x_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
sve ostale komb.				-	-

- U VHDL-u je moguće postaviti signale tipa *std_logic* na proizvoljnu vrednost, tj. "--". Sada je opis u potpunosti usklađen sa specifikacijom kodera, a softveru za sintezu se pruža mogućnost da proizvoljnost u opisu iskoristi za optimizaciju (minimizaciju) hardvera. Korišćenje proizvoljnih vrednosti, iako podržano u jeziku VHDL, nije univerzalno podržano do strane svih softvera za sintezu.

Konceptualna implementacija WHEN naredbe

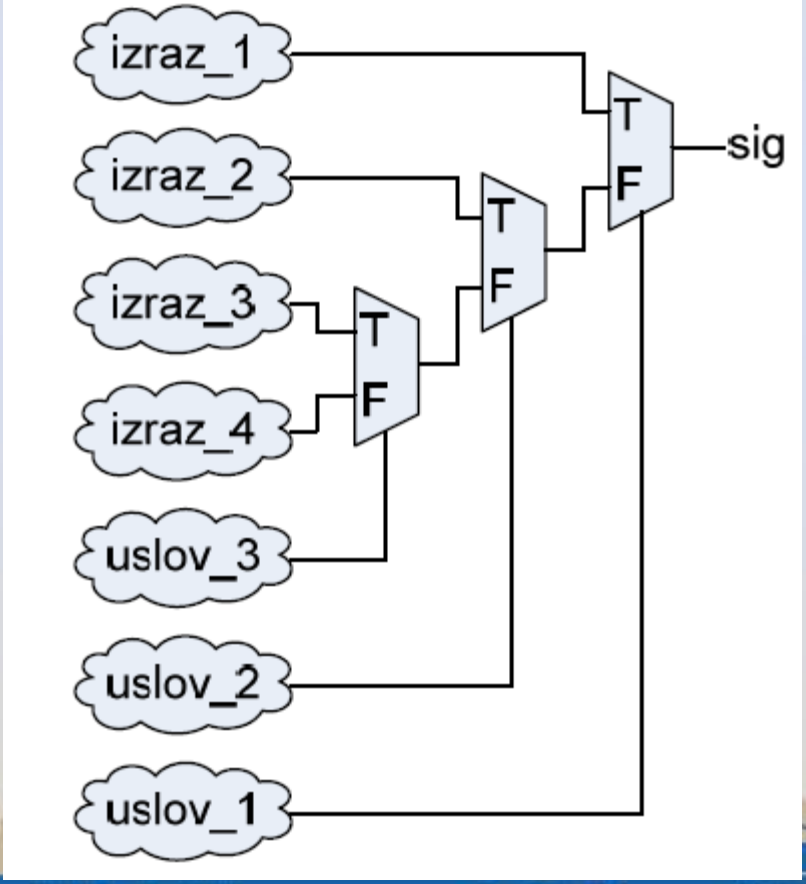
- **sig <= izraz_1 WHEN uslov_1 ELSE izraz_2**;



Apstraktni multiplekser:
T (true) - uslov je tačan
F (false) - uslov je netačan

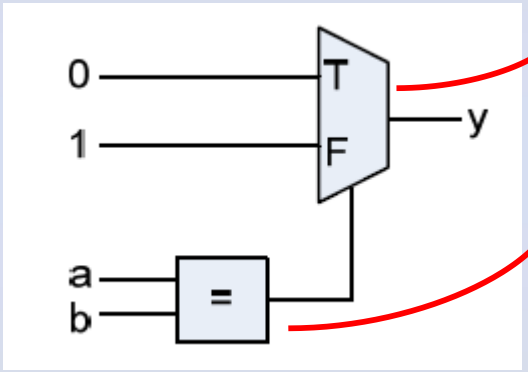
Konceptualna implementacija WHEN naredbe

sig <= izraz_1 WHEN uslov_1 ELSE
izraz_2 WHEN uslov_2 ELSE
izraz_3 WHEN uslov_3 ELSE
izraz_4;

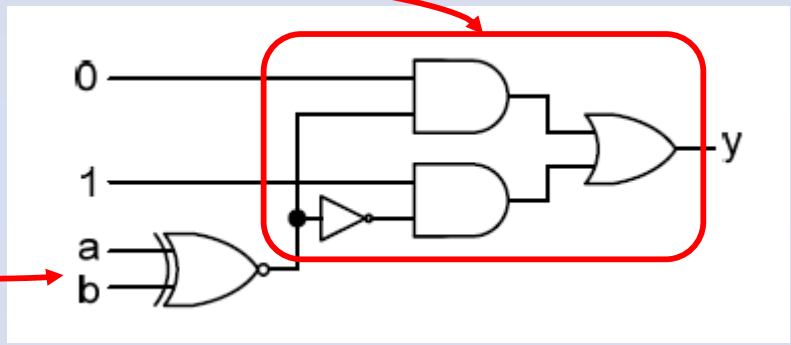


Sinteza WHEN naredbe

```
SIGNAL a,b,y : STD_LOGIC;
…
y <= ´0´ WHEN a=b ELSE ´1´;
```



a	b	a=b
0	0	1
0	1	0
1	0	0
1	1	1



Sinteza WHEN naredbe

SIGNAL a,b,f : UNSIGNED(7 DOWNTO 0);

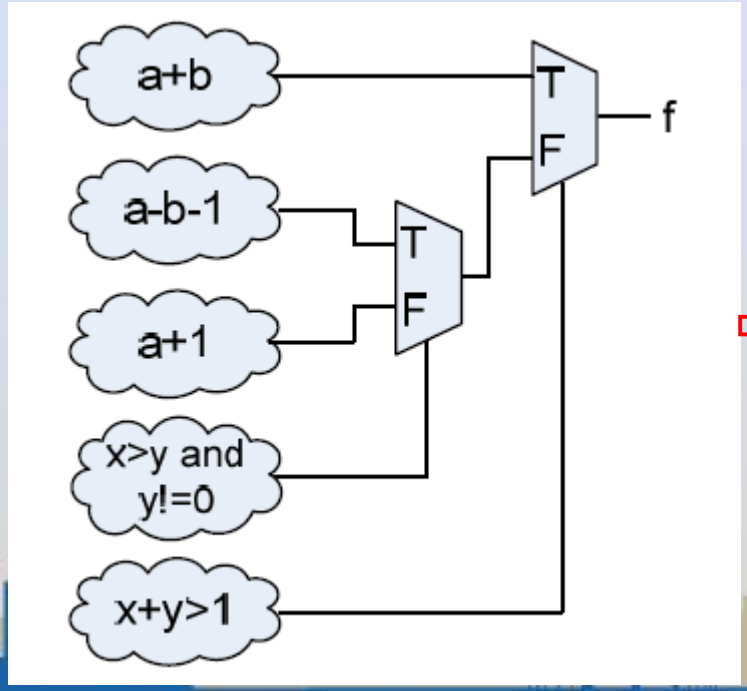
SIGNAL x,y : UNSIGNED(3 DOWNTO 0);

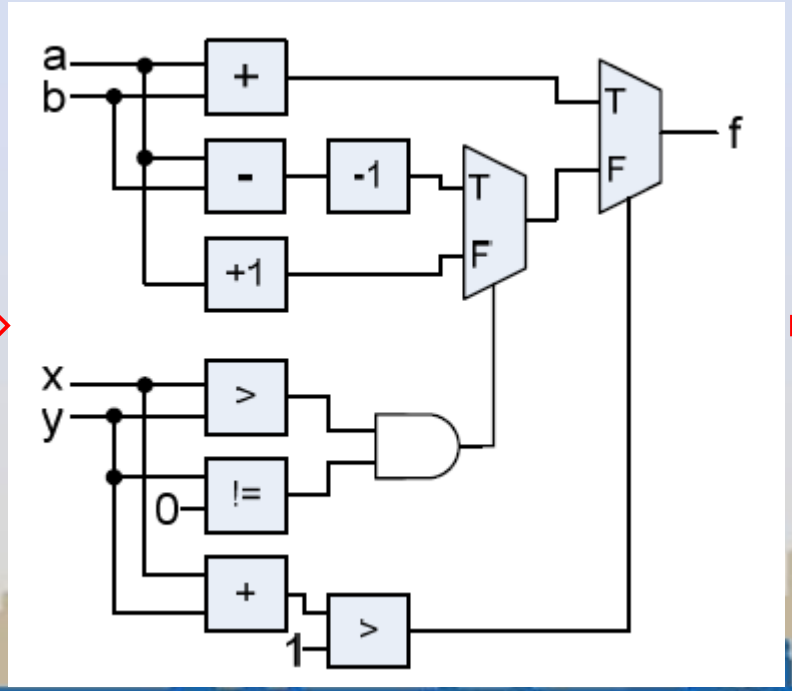
. . .

f <= a+b WHEN x+y>1 ELSE

a-b-1 WHEN (x>y AND y!=0) ELSE a+1;

Nije kraj sinteze !
Sledi zamena apstraktnih blo
odgovarajućim modulima
realizovanim
pomoću logičkih kola!





SELECET - naredba dodele sa izborom vrednosti

- Sintaksa:

WITH selekcioni_izraz SELECT

sig <= izraz_1 WHEN vrednost_1,

izraz_2 WHEN vrednost_2,

...

izraz_n WHEN vrednost_n;

- *selekcioni_izraz* - aritmetički ili logički izraz
- Izračunata vrednost selekcioni_izraz, poredi se sa vrednostima iz svih grana, a signal dobija vrednost izraza iz grane gde se javilo slaganje.
- **Zahtev: svaka moguća vrednost selekcionog_izraza mora biti pokrivena tačno jednom granom.**
- ˝WHEN vrednost˝ može biti:

WHEN vrednost; -- jedinstvena vrednost

WHEN vrednost_1 TO vrednost_2; -- opseg vrednosti

WHEN vrednost_1 | vrednost_2 | vrednost_3 ... -- više vrednosti

SELECET - naredba dodele sa izborom vrednosti

Dozvoljeno korišćenje reči OTHERS:

WITH selekcioni_izraz SELECT

sig <= izraz_1 WHEN vrednost_1,

izraz_2 WHEN vrednost_2,

...

izraz_n WHEN OTHERS;

Bira se ako vrednost selekcionog izraza nije jednaka ni jednoj navedenoj vrednosti.

- **Redosled grana u *select* naredbi nije od značaja. To je zato što se uslovi ne ispituju redom (kao kod naredbe *when*) već svi u isto vreme (paralelno).**

Realizacija tabele istinitosti

```
2 LIBRARY IEEE;
3 USE IEEE.STD_LOGIC_1164.ALL;
4 ----------------------------------------------------
5 ENTITY tabela_istinitosti IS
6 PORT (a,b : IN STD_LOGIC;
7 y : OUT STD_LOGIC);
8 END tabela_istinitosti;
9 ----------------------------------------------------
10 ARCHITECTURE select_arch OF tabela_istinitosti IS
11 SIGNAL s : STD_LOGIC_VECTOR(1 DOWNTO 0);
12 BEGIN
13 s <= a & b;
14 WITH control SELECT
15 y <= '0' WHEN "00",
16 '1' WHEN "01",
17 '0' WHEN "10",
18 '1' WHEN OTHERS;
19 END select_arch;
20 ----------------------------------------------------
```

a	b	y
0	0	0
0	1	1
1	0	0
1	1	1

Kompaktniji zapis:

```
1 y <= '1' WHEN "01" | "11",
2 '0' WHEN OTHERS;
```

ili

```
1 y <= '0' WHEN "00" | "10",
2 '1' WHEN OTHERS;
```

OTHERS grana je neophodna kad se radi s tipom *std_logic* !

a	b	y
0	0	0
0	1	1
1	0	0
1	1	1

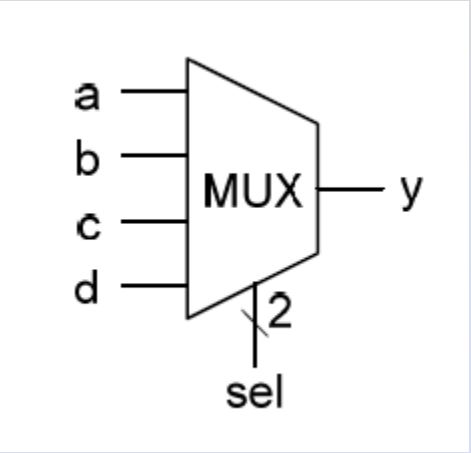
⇨

```
1  tmp <= a & b;
2  WITH tmp SELECT
3  y <= '0' WHEN "00",
4  '1' WHEN "01",
5  '0' WHEN "10",
6  '1' WHEN "11";
```

- Sve moguće vrednosti selekcionog signala moraju biti pokrivene. Za tip *std_logic* to nisu samo ″00″, ″01″, ″10″, ″11″, već i ″0X″, ″HL″, … (ukupno 64).

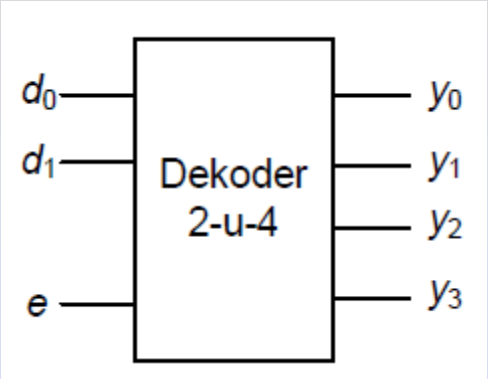
SELECT Primer: Multiplekser

```
1  ARCHITECTURE select_arch OF mux4u1 IS
2  BEGIN
3  WITH sel SELECT
4  y <= a WHEN "00", -- "," umesto ";"
5  b WHEN "01",
6  c WHEN "10",
7  d WHEN OTHERS; -- ne moze d WHEN "11"
8  END select_arch;
```



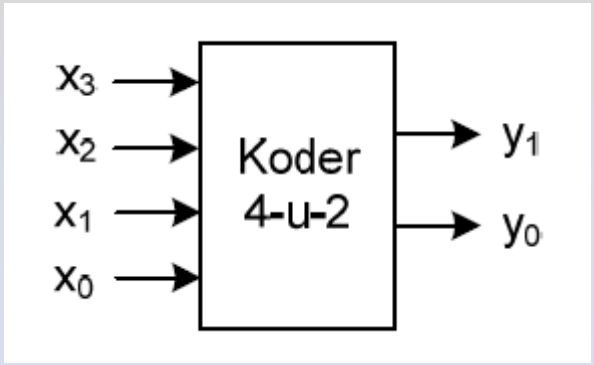
SELECT Primer: Dekoder

```
 1  ARCHITECTURE select_arch OF dek2u4 IS
 2  SIGNAL ed : STD_LOGIC_VECTOR(2 DOWNTO 0);
 3  BEGIN
 4  ed <= e & d;
 5  WITH ed SELECT
 6  y <= "0001" WHEN "100",
 7  "0010" WHEN "101",
 8  "0100" WHEN "110",
 9  "1000" WHEN "111",
10  "0000" WHEN OTHERS;
11  END select_arch;
```



SELECT Primer: Koder

```
1  ARCHITECTURE select_arch OF encoder IS
2  BEGIN
3  WITH x SELECT
4  y <= "00" WHEN "0001",
5  "01" WHEN "0010",
6  "10" WHEN "0100",
7  "11" WHEN OTHERS;
8  END select_arch;
```

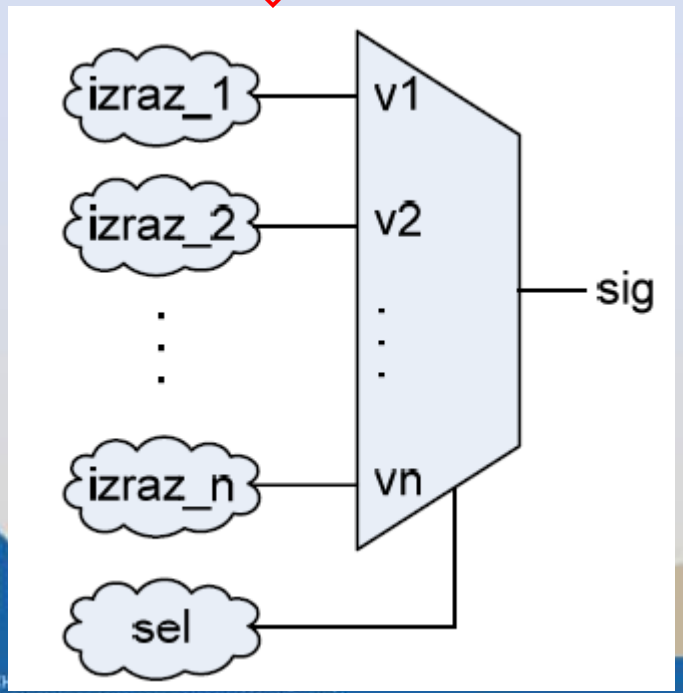



```
1  ARCHITECTURE select_arch OF encoder IS
2  BEGIN
3  WITH x SELECT
4  y <= "00" WHEN "0001",
5  "01" WHEN "0010",
6  "10" WHEN "0100",
7  "11" WHEN "1000",
8  "--" WHEN OTHERS;
9  END select_arch;
```

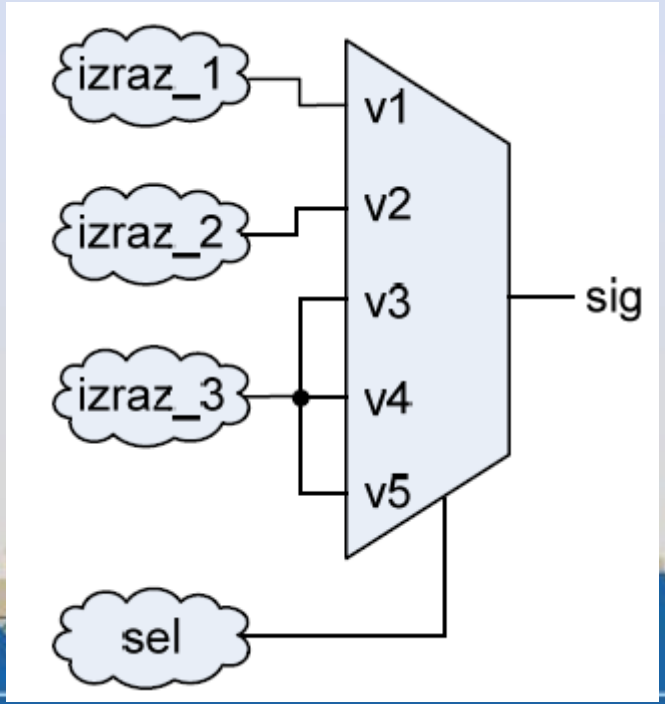
x_3	x_2	x_1	x_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
sve ostale komb.				-	-

Konceptualna implementacija naredbe SELECT

```
WITH sel SELECT
sig <= izraz_1 WHEN
v1,
izraz_2 WHEN v2,
...
izraz_n WHEN vn;
```

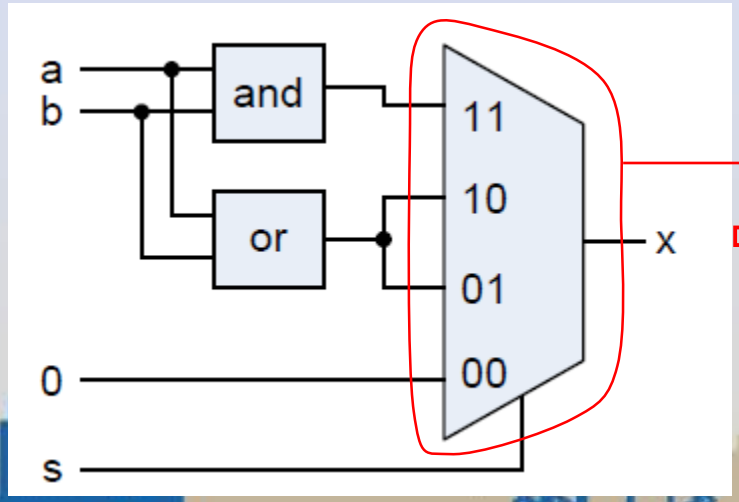


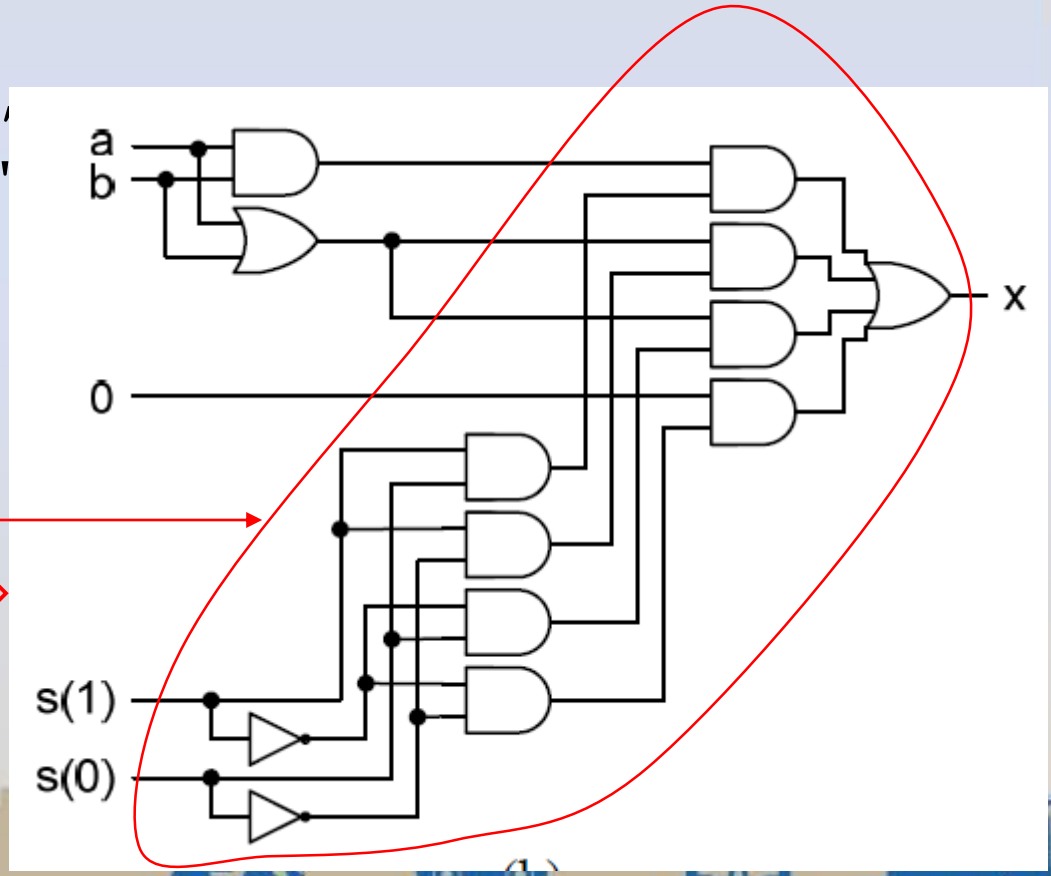

```
WITH sel SELECT
sig <= izraz_1 WHEN
v1,
izraz_2 WHEN v2,
izraz_3 WHEN OTHERS;
```



Konceptualna implementacija naredbe SELECT - primer

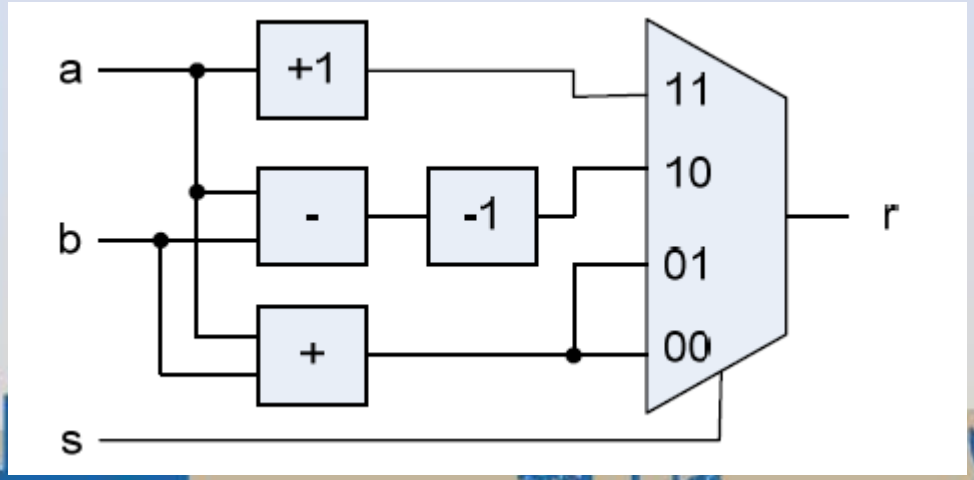
```
SIGNAL s: STD_LOGIC_VECTOR(1 DOWNTO
0);
. . .
WITH s SELECT
x <= (a AND b) WHEN "11",
(a OR b) WHEN "01" | "10"
'0' WHEN OTHERS;
```





Konceptualna implementacija naredbe SELECT - primer

```
SIGNAL a,b,r: UNSIGNED(7 DOWNTO 0);
SIGNAL s: STD_LOGIC_VECTOR(1 DOWNTO 0);
. . .
WITH s SELECT
r <= a+1 WHEN "11",
a-b-1 WHEN "10",
a+b WHEN OTHERS;
. . .
```



⇨ …

SELECT vs WHEN

SELECT

- Kada je kolo opisano tabelom istinitosti ili nekom formom funkcionalne tabele (npr. dekoder, multiplekser, ...)

WHEN

- Kada se nekim ulazima ili operacijama daje viši prioritet (npr. prioritetni koder)
- Koncizno opisivanje složenih uslova, kao na primer:

r <= a+b WHEN (x+y>1 AND x>3) ELSE

a-b-1 WHEN (z>v AND f = ’1’) ELSE

- Manje efikasna kada treba opisati tabelu istinitosti:

```
x<=a WHEN (s="00") ELSE
   b WHEN (s="01") ELSE
   c WHEN (s="10") ELSE
   d;
```

⇔

```
x<=b WHEN (s="01") ELSE
   a WHEN (s="00") ELSE
   c WHEN (s="10") ELSE
   d;
```

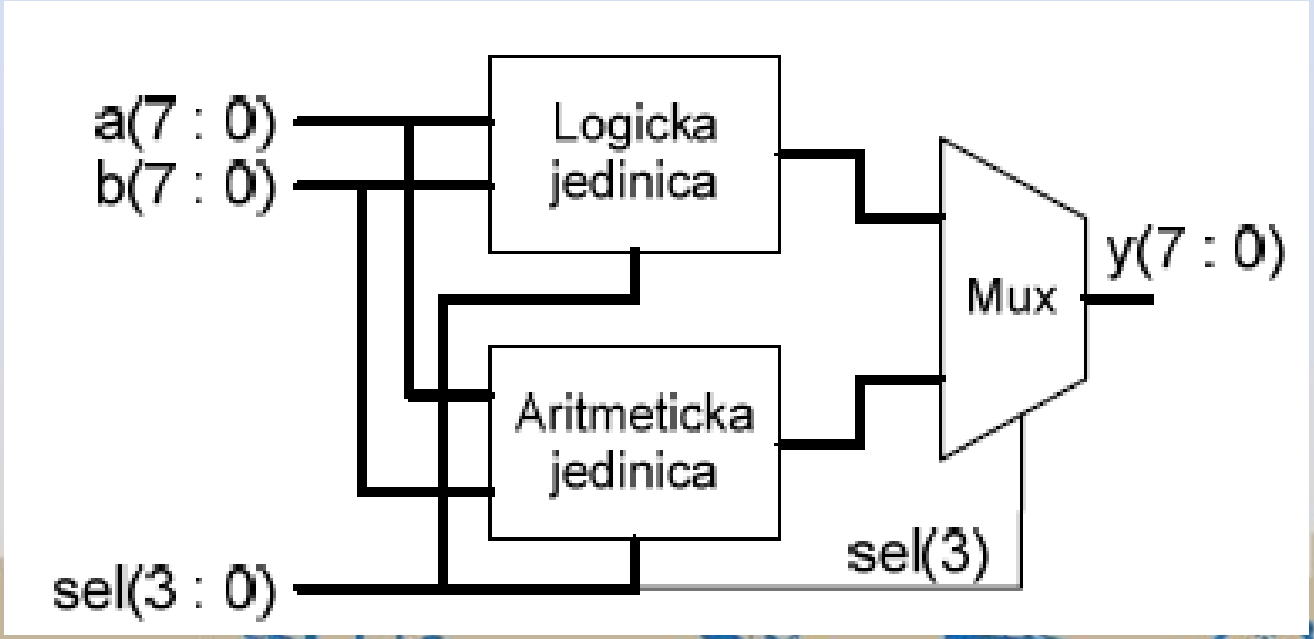
⇔

```
x<=c WHEN (s="10") ELSE
   b WHEN (s="01") ELSE
   a WHEN (s="00") ELSE
   d;
```

Nije pogrešno, ali uvodi nepotrebna ograničenja (prioriteti) koji mogu otežati sintezu i uneti nepotreban hardver.

Jednostavna ALU

Aritmetičko-logička jedinica (ili ALU, engl. ***A**rithmetic and **L**ogic **U**nit*) je višefunkcionalno kombinaciono kolo koje može da obavlja više različitih aritmetičkih i logičkih operacija nad parom višebitnih operanada. Na Sl. 4-12(a) je prikazan blok dijagram, a na Sl. 4-12(b) funkcionalna tabela jedne jednostavne 8-bitne ALU koja podržava osam aritmetičkih i osam logičkih operacija. Izbor između aritmetičkih i logičkih operacija vrši se bitom najveće težine 4-bitnog selekcionog signala *sel*. Preostala tri bita signala *sel* biraju jednu od osam operacija iz svake grupe.



Jednostavna ALU

sel	Operacija	Funkcija	Jedinica
0000	y <= a	Transfer a	Aritmetička
0001	y <= a + 1	Inkrement a	
0010	y <= a - 1	Dekrement a	
0011	y <= b	Transfer b	
0100	y <= b + 1	Inkrement b	
0101	y <= b - 1	Dekrement b	
0110	y <= a + b	Sabiranje	
0111	y <= a-b	Oduzimanje	
1000	y <= NOT a	Komplement a	Logička
1001	y <= NOT b	Komplement b	
1010	y <= a AND b	I	
1011	y <= a OR b	ILI	
1100	y <= a NAND b	NI	
1101	y <= a NOR b	NILI	
1110	y <= a XOR b	Iskljucivo ILI	
1111	y <= a NXOR b	Iskljucivo NILI	

Jednostavna ALU

```
1  ----------------------------------------------------
2  LIBRARY IEEE;
3  USE IEEE.STD_LOGIC_1164.ALL;
4  USE IEEE.NUMERIC_STD.ALL;
5  ----------------------------------------------------
6  ENTITY ALU IS
7  PORT (a,b : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
8  sel : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
9  y : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
10 END ALU;
11 ----------------------------------------------------
12 ARCHITECTURE select_arch OF ALU IS
13 SIGNAL arith, logic : STD_LOGIC_VECTOR(7 DOWNTO 0);
14 SIGNAL inc_a, dec_a, inc_b,
15 dec_b, sum, dif : STD_LOGIC_VECTOR(7 DOWNTO 0);
16 BEGIN
17 -- Aritmeticka jedinica ---------------------------
18 inc_a <= STD_LOGIC_VECTOR(SIGNED(a) + 1);
19 dec_a <= STD_LOGIC_VECTOR(SIGNED(a) - 1);
20 inc_b <= STD_LOGIC_VECTOR(SIGNED(b) + 1);
21 dec_b <= STD_LOGIC_VECTOR(SIGNED(b) - 1);
22 sum <= STD_LOGIC_VECTOR(SIGNED(a) + SIGNED(b));
23 dif <= STD_LOGIC_VECTOR(SIGNED(a) - SIGNED(b));
24 WITH sel(2 DOWNTO 0) SELECT
25 arith <= a WHEN "000",
26 inc_a WHEN "001",
27 dec_a WHEN "010",
28 b WHEN "011",
29 inc_b WHEN "100",
30 dec_b WHEN "101",
31 sum WHEN "110",
32 dif WHEN OTHERS;
33 -- Logicka jedinica -------------------------------
```

Jednostavna ALU

```
33 -- Logicka jedinica ---------------------------------
34 WITH sel(2 DOWNTO 0) SELECT
35 logic <= NOT a WHEN "000",
36 NOT b WHEN "001",
37 a AND b WHEN "010",
38 a OR b WHEN "011",
39 a NAND b WHEN "100",
40 a NOR b WHEN "101",
41 a XOR b WHEN "110",
42 NOT (a XOR b) WHEN OTHERS;
43 -- Mux ----------------------------------------------
44 WITH sel(3) SELECT
45 y <= arith WHEN '0',
46 logic WHEN OTHERS;
47 END select_arch;
48 -------------------------------------------------------
```

Optimizacija konkurentnog koda

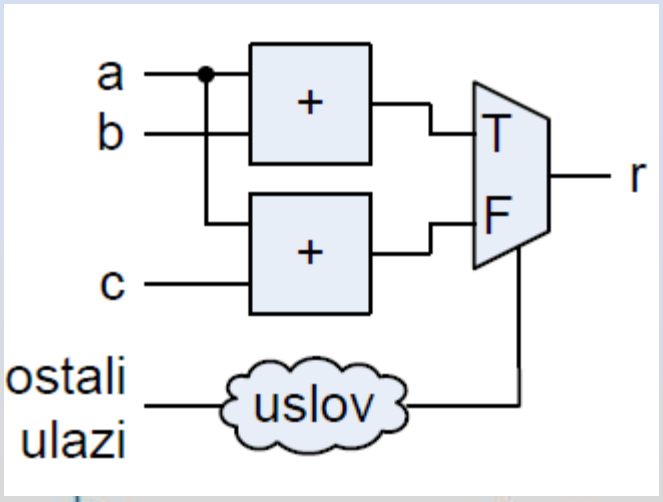
- Cilj: sa što manje hardverskih resursa realizovati željenu funkciju.
- Minimizacija broja aritmetičkih i relacionih operatora u kodu.
- Dve tehnike:

1. Deoba operatora i
2. Deoba funkcija

Deoba operatora

- Kako smanjiti broj aritmetičkih operatora u kodu?
- Preurediti kod tako da se se isti operator može iskoristiti za obavljanje više različitih operacija.

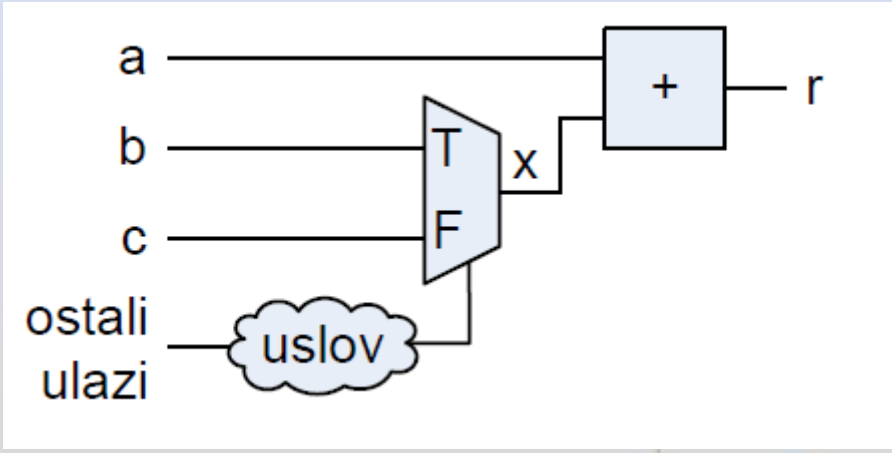
```
r <= a + b WHEN uslov ELSE
a + c;
```



Složenost: 2 sabirača + multiplekser

Kašnjenje: $max\{T_{sab}, T_{uslov}\} + T_{mux}$

```
x <= b WHEN uslov ELSE
c;
r <= a + x;
```

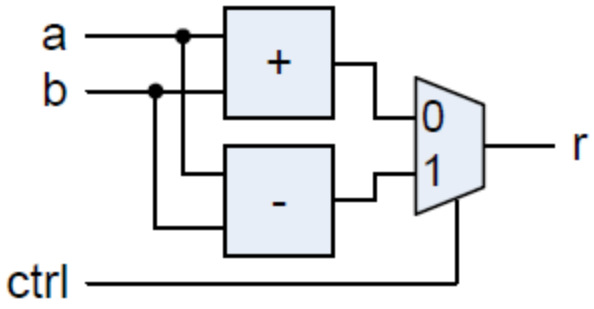


Složenost: 1 sabirač + multiplekser

Kašnjenje: $T_{sab} + T_{uslov} + T_{mux}$

Deoba funkcija

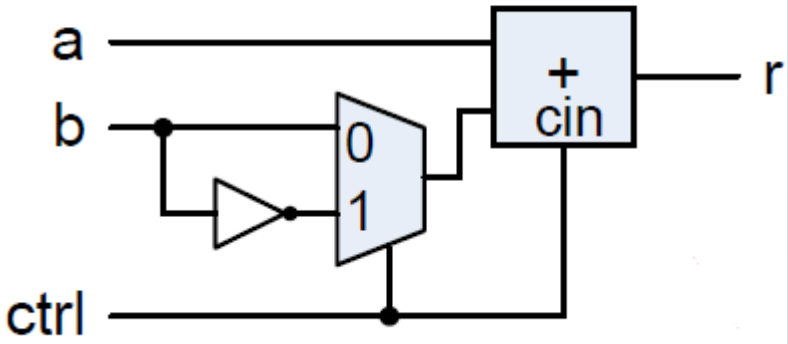
- Više funkcija realizuju se tako da dele neke zajedničke delove ili se jedna funkcija koristi za realizaciju neke druge funkcije.



Funkcionalna tabela

ctrl	operacija
0	a + b
1	a - b

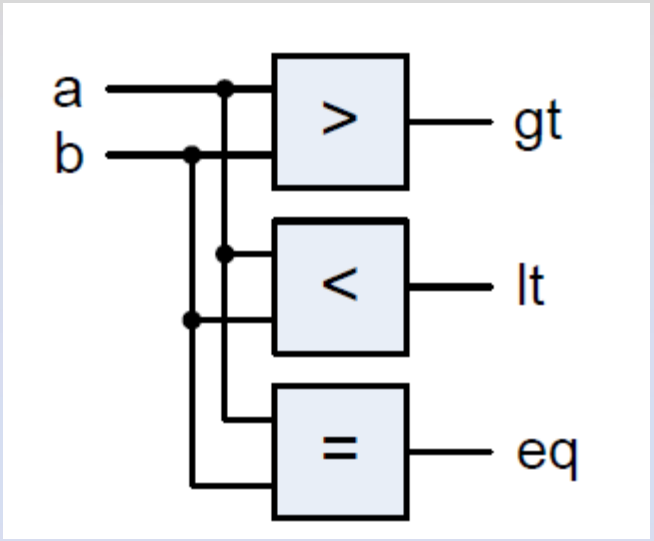
$$a - b = a + b' + 1$$



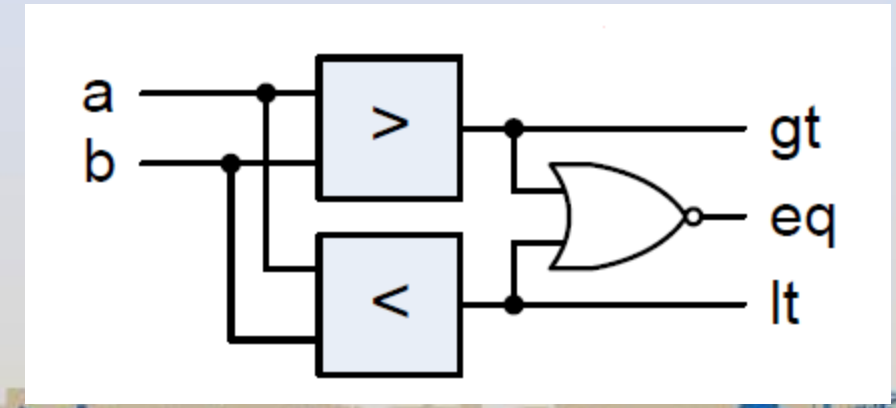
Deoba funkcija

- Potpuni komparator:

```
gt <= '1' WHEN a > b ELSE
'0';
lt <= '1' WHEN a < b ELSE
'0';
eq <= '1' WHEN a = b ELSE
'0';
```



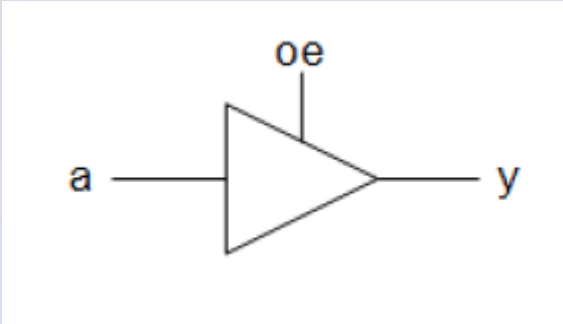

```
xgt <= '1' WHEN a > b ELSE
'0';
xlt <= '1' WHEN a < b ELSE
'0';
gt <= xgt;
lt <= xlt;
eq <= xgt NOR xlt;
```



´Z´

- ´Z´ nije logička vrednost (u smislu Bulove algebre), već predstavlja električnu osobinu (visoka imedansa) - karakteristika *trostičkih bafera*.

***oe*=´1´**: bafer prenosi logičku vrednost ulaza na izlaz -deluje kao ″zatvoreno kolo″.
***oe*=´0´:** izlaz bafera je postavljen u stanje visoke impedanse - deluje kao ″otvoreno kolo″.



oe	y
0	Z
1	a

```
y <= a WHEN oe='1' ELSE 'Z';
```

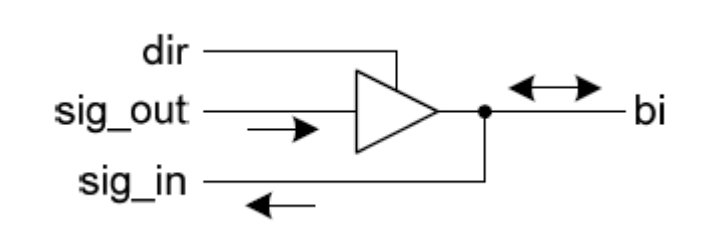
- Ne može se koristiti kao ulazna vrednost, niti se s ovom vrednošću može manipulisati na način kao sa logičkim vrednostima ´0´ i ´1´.
- Sledeće dve naredbe se ne mogu sintetizovati:

```
r <= 'Z' AND a;
g <= d - c WHEN a = 'Z' ELSE d - b;
```

Bidirekcioni (ulazno/izlazni) port

- Port koji se po potrebi može koristiti bilo kao ulaz bilo kao izlaz.

```
1  ENTITY bi_port IS
2  PORT(...
3  bi : INOUT STD LOGIC;
4  ...);
5  BEGIN
6  ...
7  bi <= sig_out WHEN dir = '1' ELSE
8  'Z';
9  sig_in <= bi;
10 ...
```

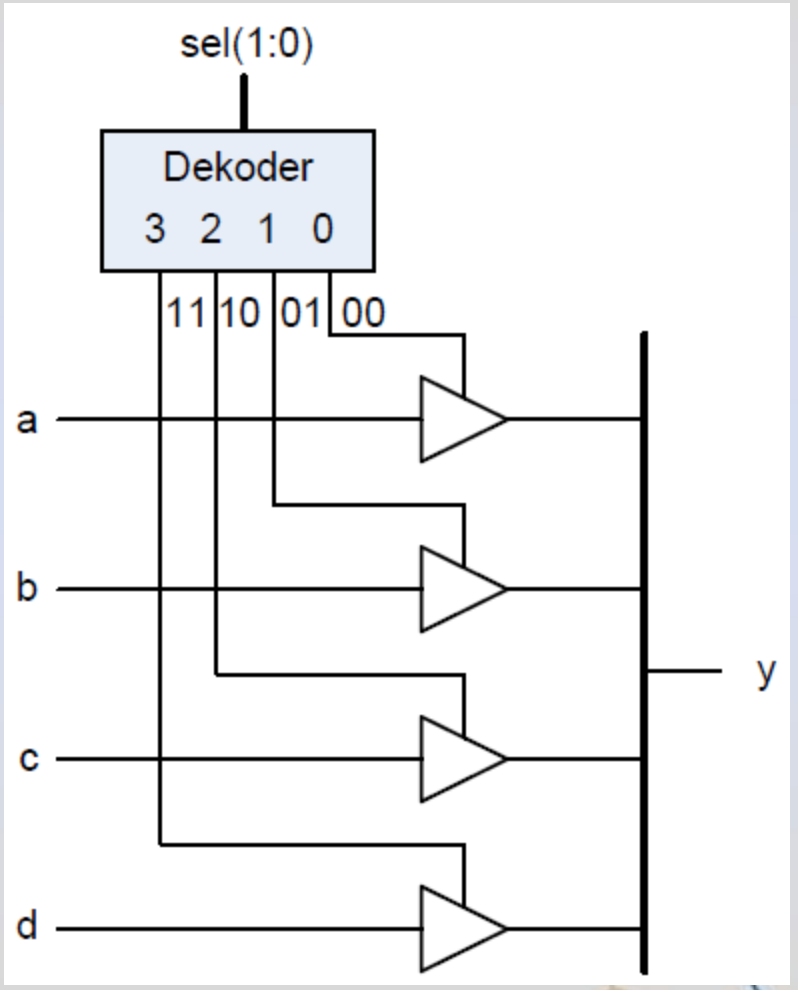


Za **dir=´1´**, **bi** je izlaz

Za **dir=0**, **bi** je ulaz

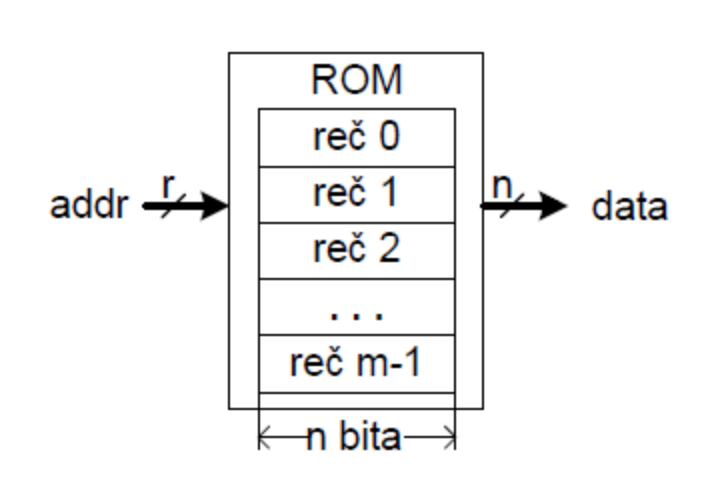
Trostatička magistrala

```
1  ----------------------------------
2  LIBRARY IEEE;
3  USE IEEE.STD_LOGIC_1164.ALL;
4  ----------------------------------
5  ENTITY bus IS
6  PORT ( a,b,c,d : IN STD_LOGIC;
7  s: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
8  y : OUT STD_LOGIC);
9  END bus;
10 ----------------------------------
11 ARCHITECTURE arch OF bus IS
12 BEGIN
13 y <= a WHEN s = "00" ELSE 'Z';
14 y <= b WHEN s = "01" ELSE 'Z';
15 y <= c WHEN s = "10" ELSE 'Z';
16 y <= d WHEN s = "11" ELSE 'Z';
17 END arch;
18 ----------------------------------
```



ROM

- ROM (prema eng. ***R**ead-**O**nly **M**emory*) je memorija sa konstantnim sadržajem - memorijska komponenta koja može samo da se čita.



ROM

```
1  ----------------------------------------------------------------------
2  LIBRARY IEEE;
3  USE IEEE.STD_LOGIC_1164.ALL;
4  USE IEEE.NUMERIC_STD.ALL;
5  ----------------------------------------------------------------------
6  ENTITY ROM IS
7  PORT (addr : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
8  data : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
9  END ROM;
10 ----------------------------------------------------------------------
11 ARCHITECTURE rom OF ROM IS
12 TYPE mem_array IS ARRAY (0 TO 7) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
13 CONSTANT memory : mem_array := ("00000000",
14 "00000010",
15 "00000100",
16 "00001000",
17 "00010000",
18 "00100000",
19 "01000000",
20 "10000000");
21 BEGIN
22 data <= memory(TO_INTEGER(UNSIGNED(addr)));
23 END rom;
24 ----------------------------------------------------------------------
```

Indeks polja može biti samo **integer**