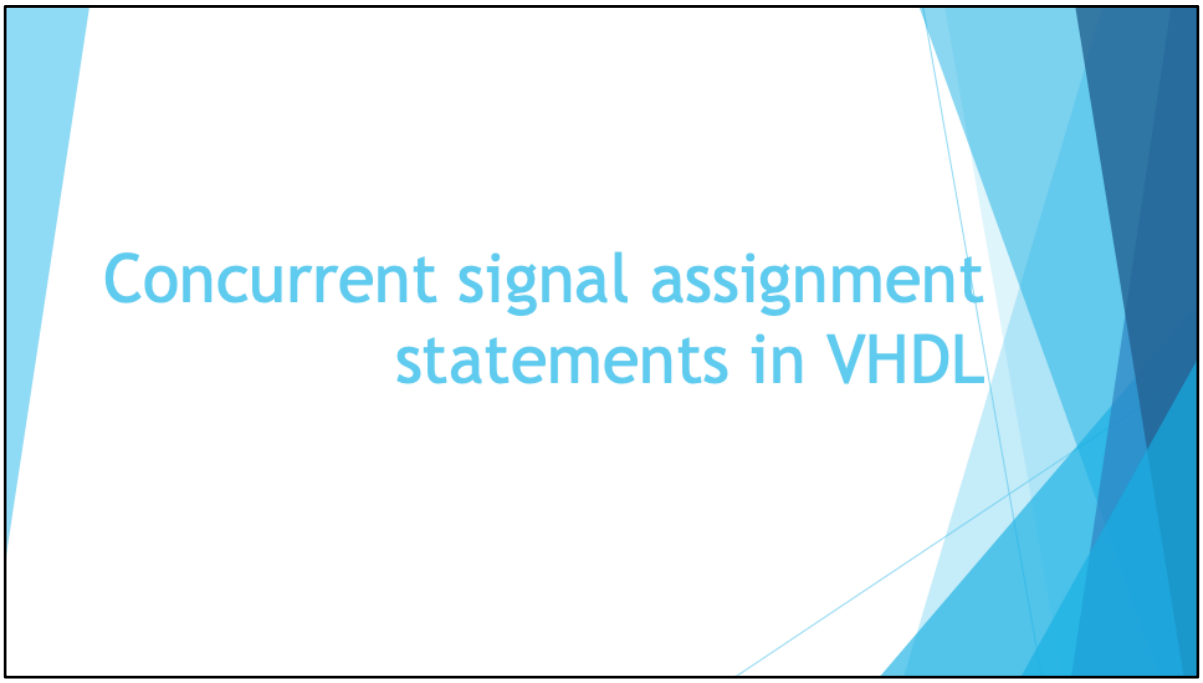
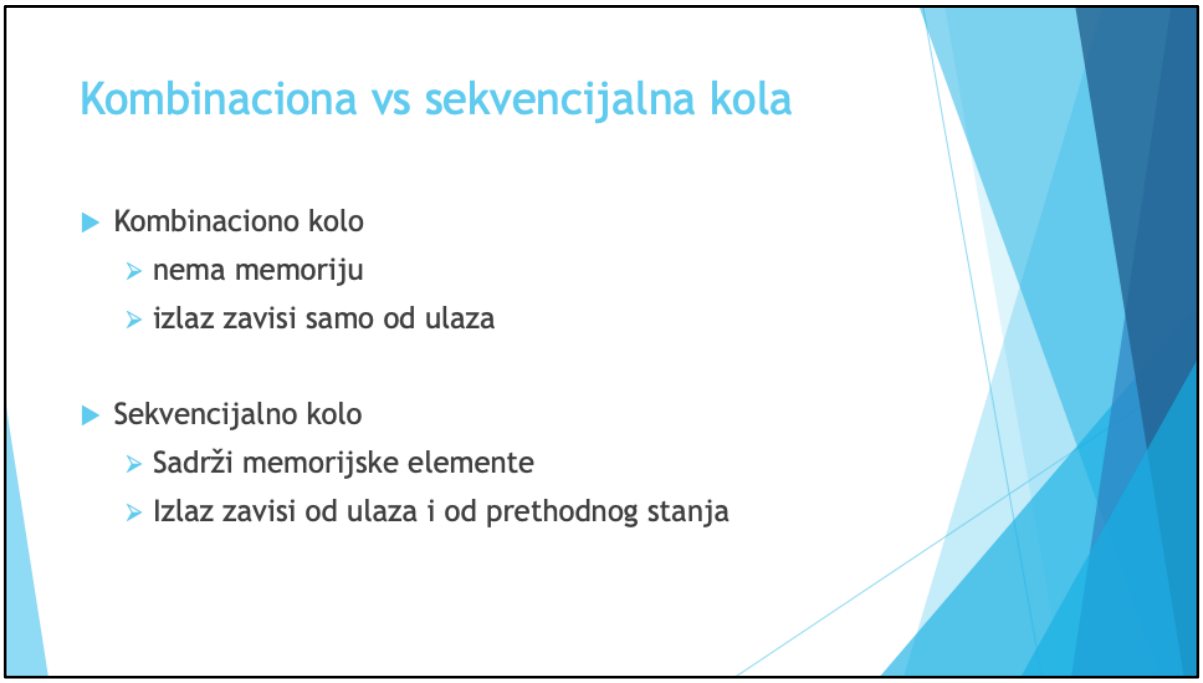


Concurrent signal assignment statements u VHDL-u se prilično jednostavno mogu vizuelizovati u smislu konceptualnog dijagrama s obzirom da postoji hardverska paralela. Ovo svojstvo je veoma korisno prilikom razvoja efikasnog dizajna. Postoje dvije osnovne forme: conditional signal assignment statements i selected signal assignment statements. Osim njih, uvodi se i jednsotavni signal assignement statements, koji je zapravo uslovni pri čemu uslov ne postoji.



Concurrent signal assignment se uglavnom koristi kod dizajna kombinacionih kola

Simple signal assignment statement

```
signal_name <= projected_waveform;
y <= a + b + 1 after 10 ns
```

- U sintezi se kašnjenje ne navodi, već se podrazumijeva:

```
signal_name <= value_expression
```

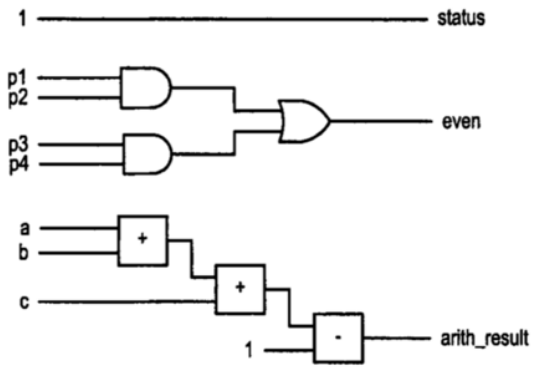
```
status <= '1';
even <= (p1 and p2) or (p3 and p4);
arith_out <= a + b + c - 1;
```

```
status <= '1' after δ;
even <= (p1 and p2) or (p3 and p4) after δ;
arith_out <= a + b + c - 1 after δ;
```

projected_waveform sadrži dvije specifikacije: izraz kojim se zadaje vrijednost signala i vrijeme kada će signal imati zadatu vrjednost. Zadato vrijeme odgovara propagacionom kašnjenju potrebnom za izvršavanje izraza kojim se definiše vrijednost signala. Naravno, nije moguće izvršiti sintezu kola sa tačno odereñenim kašnjenjem (zavisi od komponenti, tehnologije, fabrikacionog procesa, radnog okruženja). Iz tog razloga, u okviru sinteze se eksplicitno ne zadaje vrijeme u VHDL kodu, već se koristi default-no delta kašnjenje.

Konceptualna implementacija

```
status <= '1';
even <= (p1 and p2) or (p3 and p4);
arith_out <= a + b + c - 1;
```



Dijagram je samo konceptualna skica. U toku sinteze se transformiše i značajno pojednostavljuje. Kompleksnost realizacije pojedinih operatora značajno varira

Conditional signal assignment statement

```
signal_name <= value_expr_1 when boolean_expr_1 else
               value_expr_2 when boolean_expr_2 else
                    . . .
               value_expr_n;
```

Primjer: Multiplekser

```
library ieee;
use ieee.std_logic_1164.ALL;

entity multiplekser is
    port(
            a, b, c, d : in std_logic_vector (7 downto 0);
            s : in std_logic_vector (1 downto 0);
            x : out std_logic_vector (7 downto 0) );
end multiplekser;

architecture multiplekser_arc of multiplekser is
begin
    x  <= a when s = "00" else
            b when s = "01" else
            c when s = "10" else
            d;
end multiplekser_arc;
```

Primjer: Binary decoder

```
library ieee;
use ieee.std_logic_1164.ALL;

entity binary_decoder is
    port
        (
            s : in std_logic_vector(1 downto 0);
            x : out std_logic_vector(3 downto 0)
        );
end binary_decoder;

architecture binary_decoder_arc of binary_decoder is
begin
    x <= "0001" when s = "00" else
         "0010" when s = "01" else
         "0100" when s = "10" else
         "1000";
end binary_decoder_arc;
```

Input	Output
s	x
00	0001
01	0010
10	0100
11	1000

Primjer: Priority encoder

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

entity priority_encoder_4in2 is
    port
        (
            r : in std_logic_vector (3 downto 0);
            code : out std_logic_vector (1 downto 0);
            active : out std_logic
        );
end priority_encoder_4in2;

architecture priority_encoder_4in2_arc of
priority_encoder_4in2 is
begin
    code <= "11" when r(3) = '1' else
            "10" when r(2) = '1' else
            "01" when r(1) = '1' else
            "00";
    active <= r(3) or r(2) or r(1) or r(0);
end priority_encoder_4in2_arc;
```

Input	Output	
r	code	active
1---	11	1
01--	10	1
001-	01	1
0001	00	1
0000	00	0

Primjer: Simple ALU

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

entity Simple_ALU is
     port
          (
               src0, src1 : in std_logic_vector (7 downto 0);
               ctrl : in std_logic_vector (2 downto 0);
               result : out std_logic_vector (7 downto 0)
          );
end Simple_ALU;

architecture Simple_ALU_arc of Simple_ALU is
signal sum, diff, inc : std_logic_vector (7 downto 0);

begin
     sum <= std_logic_vector(signed(src0) + signed(src1));
     diff <= std_logic_vector(signed(src0) - signed(src1));
     inc <= std_logic_vector(signed(src0) + 1);
     result <= inc when ctrl(2) = '0' else
               sum when ctrl(1 downto 0) = "00" else
               diff when ctrl(1 downto 0) = "01" else
               (src0 and src1) when ctrl(1 downto 0) = "10" else
               (src0 or src1);
end Simple_ALU_arc;
```

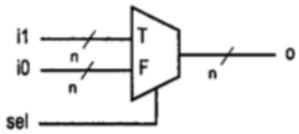
Input	Output
ctrl	result
0--	src0 + 1
100	src0 + src1
101	src0 - src1
110	src0 and src1
111	src0 or src1

Konceptualna implementacija

- Neophodne tri hardverske grupe:
  - Kola za value izraze
  - Kola za boolean izraze
  - Prioritetna routing mreža

```
signal_name <= value_expr_1 when boolean_expr_1 else
               value_expr_2 when boolean_expr_2 else
                        . . .
               value_expr_n;
```

Konceptualni dijagram apstraktnog multipleksera

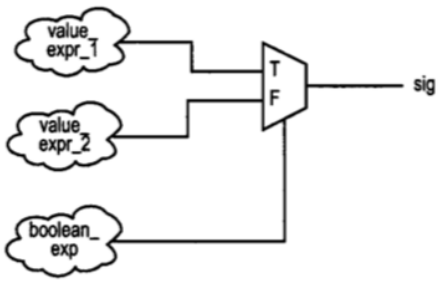


Kako bi se implementirao uslovni signal assignment statement, neophodno je ostvariti izvršavanje izraza u opadajućem redosljedu sve dok se ne dođe do tačnog izraza. Za razliku od tradicionalnih programskih jezika, u okviru sinteze, neophodno je koristiti hardver u tu svrhu. Potrebno je specificirati prioritetnu routing mrežu, jer izraz naveden ranije ima viši prioritet. Kada boolean izraz postane tačan, odgovarajući izraz se prosljeđuje na izlaz. Prioritetna routing mreža se realizuje na prostornom nivou, za razliku od tradicionalnih programskih jezika kod kojih je izvršavanje temporalnog karaktera. Kako se hardver ne može dinamički kreirati, potrebno je za svaki izraz implementirati odgovarajući hardver.

Prioritetna routing mreža je struktura koja sprovodi i kontroliše željenu vrijednost do izlaznog signala. Može se implementirati kao sekvenca 2u1 multipleksera.

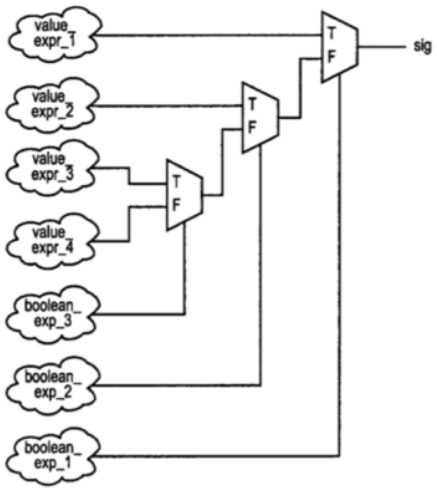
Primjer konstruisanja prioritetne mreže

```
sig <= value_expr_1 when boolean_expr_1 else
       value_expr_2;
```



Primjer konstruisanja prioritetne mreže

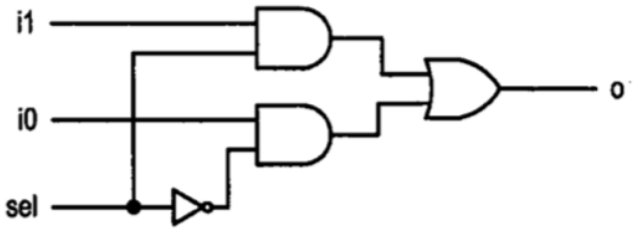
```
sig <= value_expr_1 when boolean_expr_1 else
       value_expr_2 when boolean_expr_2 else
       value_expr_3 when boolean_expr_3 else
       value_expr_4;
```



Iako je koncept prilično jednostavan, softver za sintezu ima relativno težak zadatak ukoliko je u pitanju veliki broj nivoa (when izraza).

Primjeri implementacije

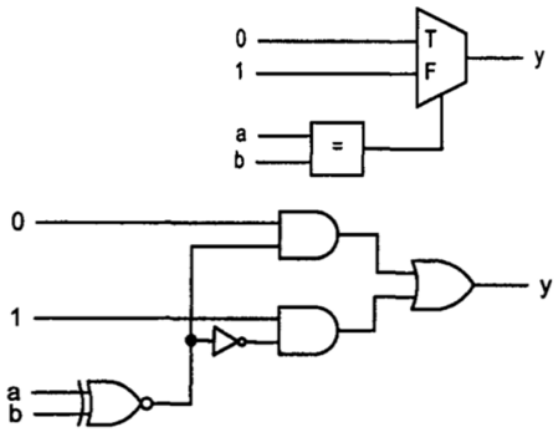
jednobitni 2u1 multiplekser



Konceptualni dijagram je samo prvi korak u sintezi. Potrebno je izvesti mnogo detaljniju implementaciju za multipleksere i “clouds” i eventualno izvršiti mapiranje u skladu sa odabranom tehnologijom. Većinu ovih zadataka obavlja softver za sintezu. Ovdje će se osnovni proces sinteze ilustrovati na jednostavnim primjerima.

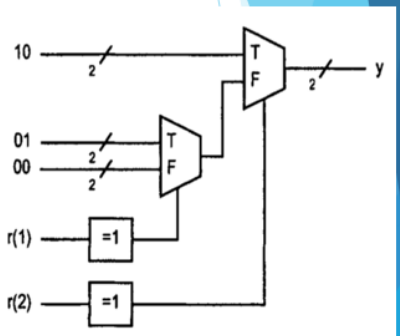
Primjer implementacije

```
. . .
signal a, b, y : std_logic;
. . .
y <= '0' when a = b else
     '1';
. . .
```

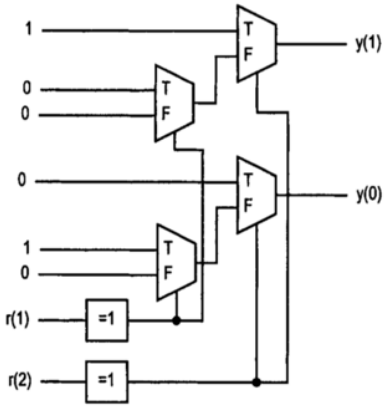


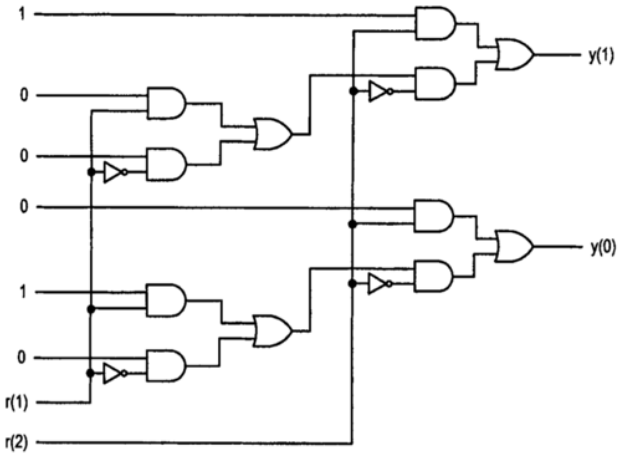
Primjeri implementacije

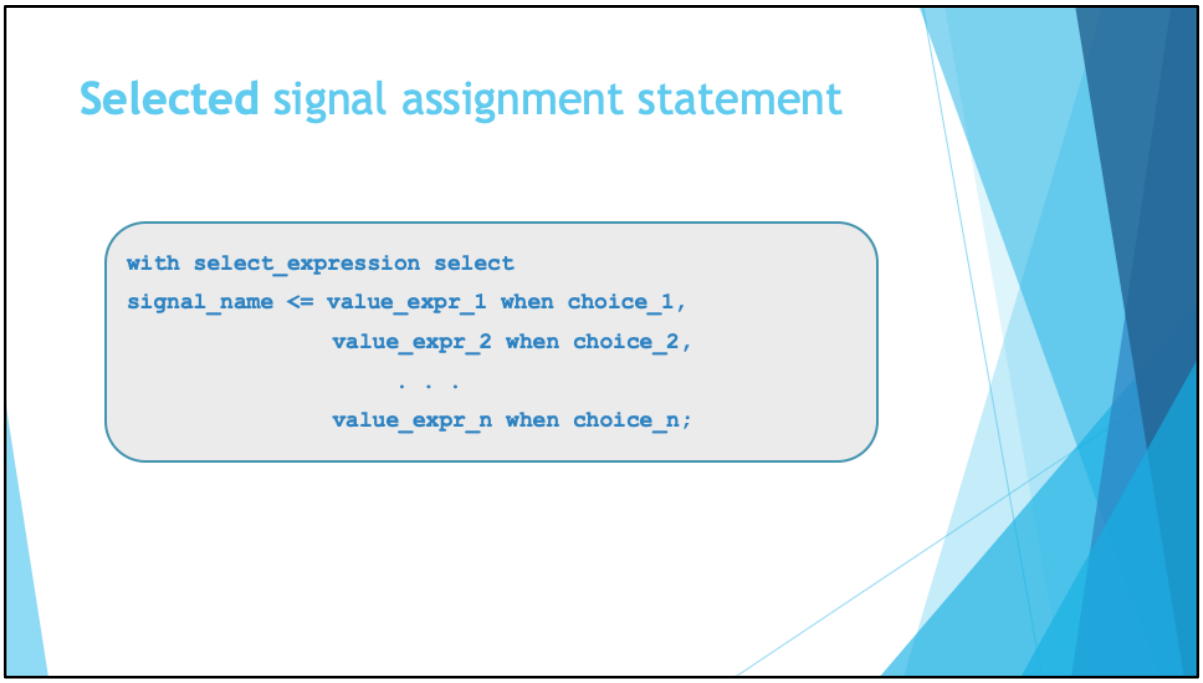
```
. . .
signal r : std_logic_vector (2 downto 0);
signal y : std_logic_vector (1 downto 0);
. . .
y <= "10" when r(2) = 1 else
     "01" when r(1) = 1 else
      "00";
. . .
```



Primjeri implementacije







choice_1 do choice_n mora pokriti sve mogućnosti za select_expression, pri čemu ne smije biti preklapanja.

Primjer: Multiplekser

```
library ieee;
use ieee.std_logic_1164.ALL;

entity multiplekser is
    port(
            a, b, c, d : in std_logic_vector (7 downto 0);
            s : in std_logic_vector (1 downto 0);
            x : out std_logic_vector (7 downto 0) );
end multiplekser;

architecture multiplekser_arc of multiplekser is
begin
with s select
    x  <= a when "00",
          b when "01",
          c when "10",
          d when others;
end multiplekser_arc;
```

Kako std_locig tip može imati jednu od 9 vrijednosti, **others** se odnosi na s=”11”, kao i na ostalih 77 kombinacija (9*9-4).

Primjer: Priority encoder

```
architecture priority_encoder_4in2_arc of
priority_encoder_4in2 is
begin
with r select
    code <= "11" when ”1000”|”1001”|”1010”|”1011”|
                      ”1100”|”1101”|”1110”|”1111”,
            "10" when ”0100”|”0101”|”0110”|”0111”,
            "01" when ”0010”|”0011”,
            "00" when others;
    active <= r(3) or r(2) or r(1) or r(0);

end priority_encoder_4in2_arc;
```

Input	Output	
r	code	active
1---	11	1
01--	10	1
001-	01	1
0001	00	1
0000	00	0

Primjer: Simple ALU

```
architecture Simple_ALU_arc of Simple_ALU is
signal sum, diff, inc : std_logic_vector (7 downto 0);

begin
    sum <= std_logic_vector(signed(src0) + signed(src1));
    diff <= std_logic_vector(signed(src0) - signed(src1));
    inc <= std_logic_vector(signed(src0) + 1);
     with ctrl select
    result <= inc when "000"|"001"|"010"|"011",
              sum when "100",
              diff when "101",
             (src0 and src1) when "110",
             (src0 or src1) when others;
end Simple_ALU_arc;
```

Input	Output
ctrl	result
0--	src0 + 1
100	src0 + src1
101	src0 - src1
110	src0 and src1
111	src0 or src1

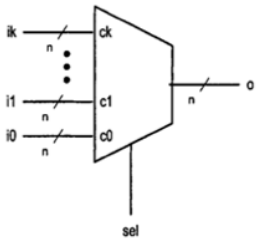
Primjer: Truth table implementation

```
library ieee;
use ieee.std_logic_1164.all;
entity truth_table is
    port
    (
     a, b : in std_logic;
     y : out std_logic
    );
end truth_table;
architecture truth_table_arc of truth_table is
    signal tmp: std_logic_vector (1 downto 0);
begin
    tmp <= a & b;
    with tmp select
        y <= '0' when "00",
             '1' when "01",
             '1' when "10",
             '1' when others;
end truth_table_arc;
```

Input		Output
a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

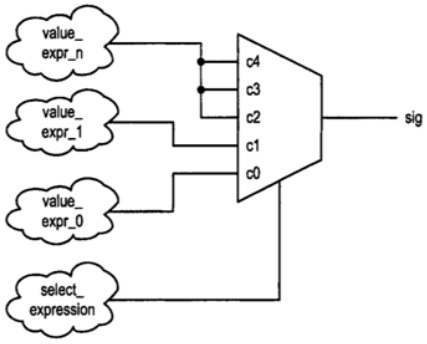
Konceptualna implementacija

```
with select_expression select
signal_name <= value_expr_1 when choice_1,
               value_expr_2 when choice_2,
                    . . .
               value_expr_n when choice_n;
```

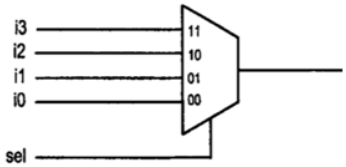


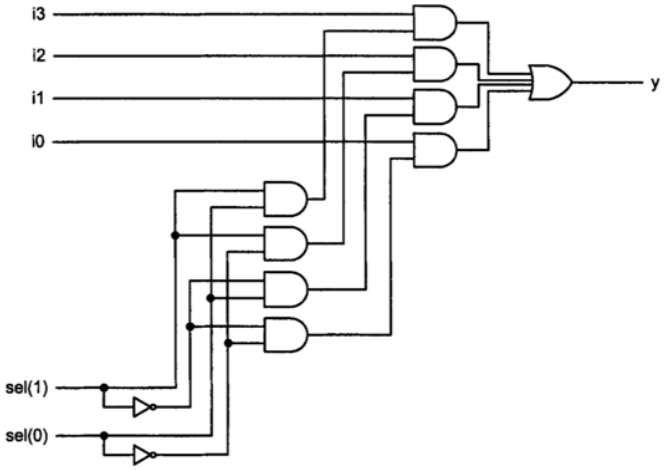
Konceptualna implementacija: primjer

```
with select_expression select
sig <= value_expr_0 when c0,
       value_expr_1 when c1,
       value_expr_n when others;
```



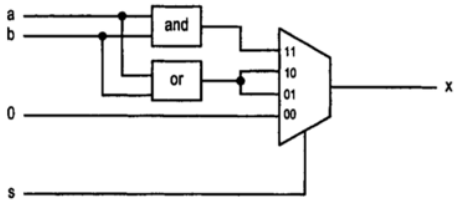
Primjer implementacije: 4u1 multiplekser

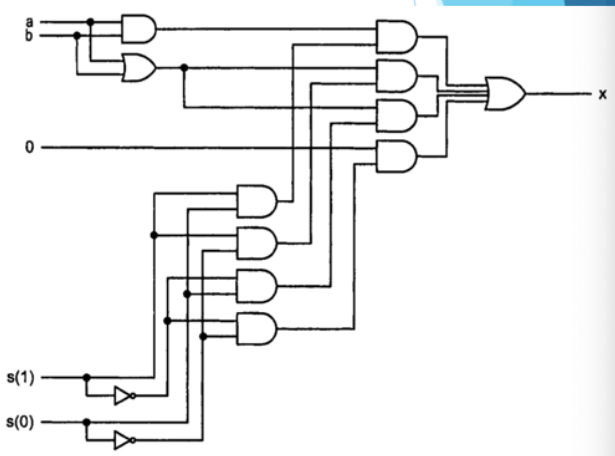




Primjeri implementacije

```
. . .
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;
. . .
```

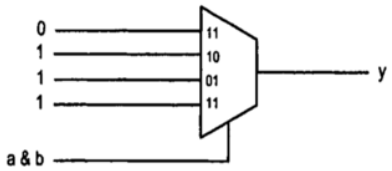




Primjeri implementacije

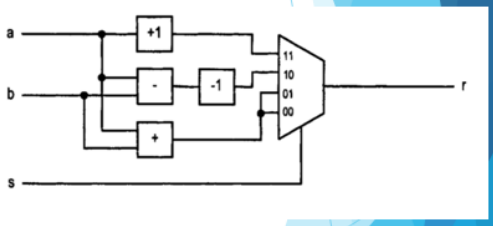
```
tmp <= a & b;
with tmp select
    y <= '0' when "00",
         '1' when "01",
         '1' when "10",
         '1' when others;
```

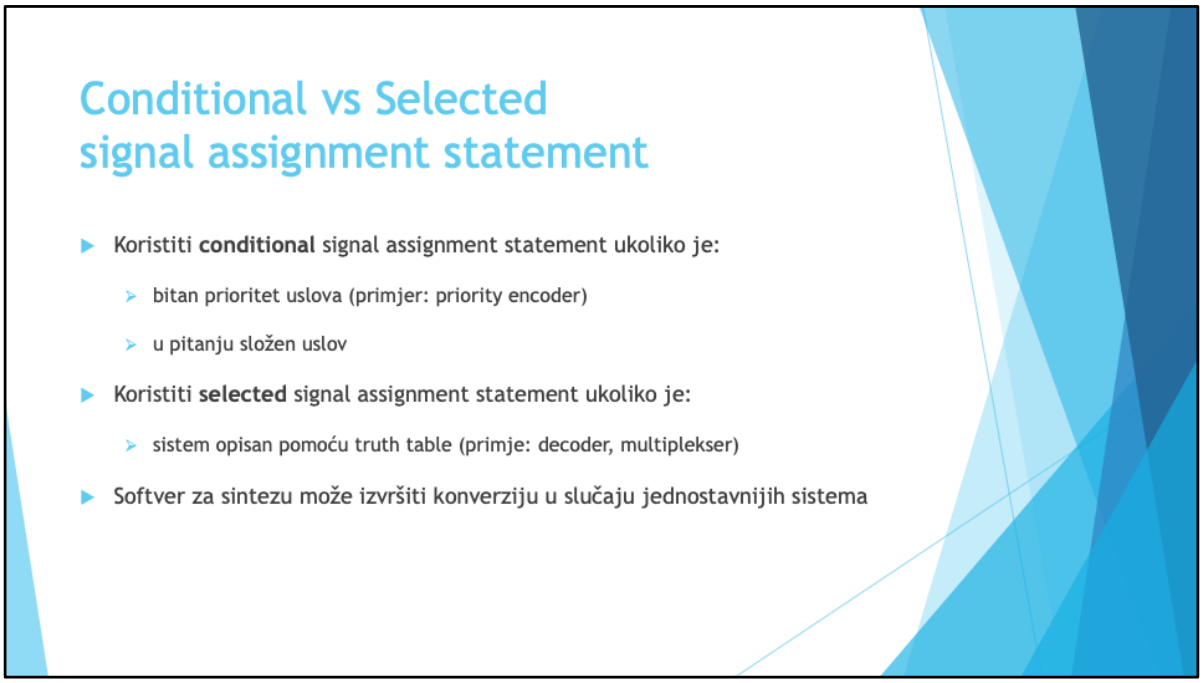
Input		Output
a	b	y
0	0	0
0	1	1
1	0	1
1	1	1



Primjeri implementacije

```
. . .
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;
    . . .
```





Sa aspekta sinteze ova dva pristupa se u potpunosti razlikuju.