Elektronski fakultet u Nišu

Smer: R

Predmet: Projektovanje ugrađenih računarskih sistema

BOUNDARY-SCAN TEST

Studenti:

Darko Filipović 9551
Goran Ristić 10067

Boundary-Scan Test

1. Uvod u Boundary-Scan tehniku

Kako integrisana kola tako i stampane ploče napretkom tehnologije postaju sve kompleksnije, pa potreba za temeljnim testiranjem postaje sve izraženija. VLSI integrisana kola su sve manja, broj pinova sve veći, pa je zbog toga klasične metode testiranja sve teže implementirati. Boundary-Scan test (BST) je jedna od tehnika koja nudi mogućnost efikasnog testiranja komponenti kako na štampanim pločama, tako i pojedinačno. Ova tehnika omogućava dovodjenje test signala na ulaze kola serijskim pomeranjem, bez narušavanja normalnog rada kola ili sistema, procesiranje tih podataka dovođenjem na ulaz kola/sistema, i konačno, serijskim pomeranjem dobijenog rezultata u jedinicu u kojoj se vrši provera tačnosti dobijenog rezultata.

1.1 Elementi Boundary-Scan arhitekture

Na slici 1.a prikazana je struktura na blok-šematskom nivou tehnike BST. Blok DUT (Device Under Test) predstavlja logika (integrisano kolo ili sistem) koji se testira. Na svim ulazima i izlazima su povezani u lanac BST ćelija. Ćelije označene sa UC_i , i=1,..,n su ulazne, a ćelije označene sa IC_j j=1,..,m su izlazne. Blok TAP (Test Access Point) predstavlja upravljacku logiku kojom se definiše režim rada sistema, normalni rad ili testiranje. Na paralelnim ulazima se dovodi informacija koja se procesira u normalnom režimu rada, a na paralelnim izlazima se dobija rezultat. Test sekvenca se dovodi preko serijskog ulaza, a rezultat testiranja se prihvata na serijskom izlazu.

Na slici 1.b prikazan je izgled jedne ćelije BST strukture sa aspekta ulaza i izlaza.

1.2 Struktura Boundary-Scan ćelije

Na slici 2 je prikazana struktura BST ćelije. Ćeliju čini jedan flip-flop FF, i dva multipleksera, MUX1 i MUX2.

Funkcija ulazno-izlaznih pinova je sledeća:

PARALELNI ULAZ - Dovodi se jednobitna informacija :

a) koja će biti procesirana od strane DUT-a kada je ćelija tipa UC_i

a) koja je procesirana od strane DUT-a kada je ćelija tipa IC_j

SERIJSKI ULAZ –Dovodi se jednobitna informacija koja se koristi za potrebe testiranja DUT-a.

PARALELNI IZLAZ-Ovi signali se interpretiraju na sledeća dva načina:

a) za ćelije tipa UC_i signali na ovim izvodima su pobudni signali DUT-a

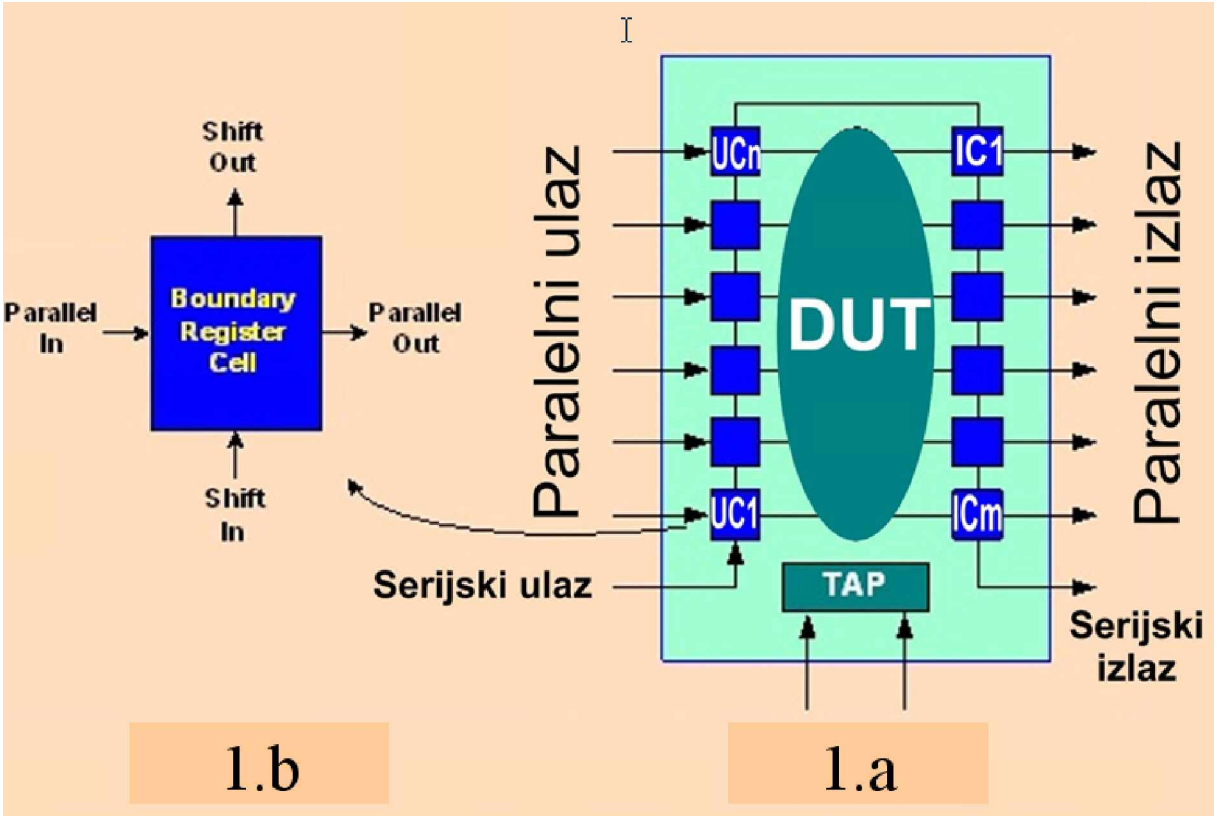
b) za ćelije tipa IC_j signali na ovim izvodima su izlazni signali sistema

SERIJSKI IZLAZ-Prosleđuje jednobitnu informaciju narednoj ćeliji u lancu BST-a

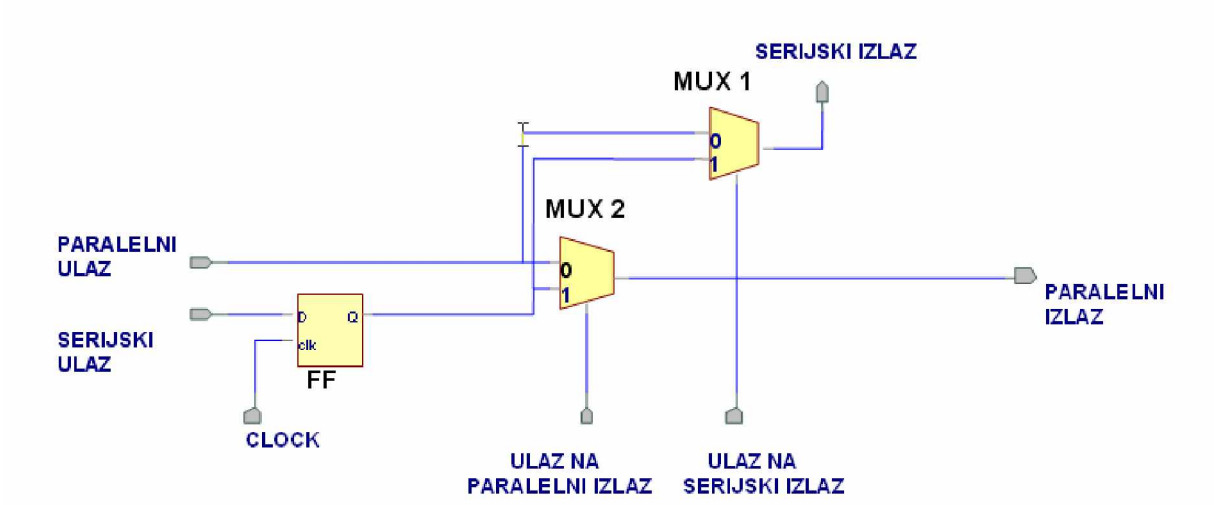
CLOCK- Signal taktne pobude.

ULAZ NA PARELELNI IZLAZ – Upravljački signal kojim se vrši selekcija pobude DUT-a (za UC_i), ili izbor izlaza DUT-a (za IC_j)

ULAZ NA SERIJSKI IZLAZ- Upravljački signal kojim se vrši selekcija signala ka serijskom izlazu/

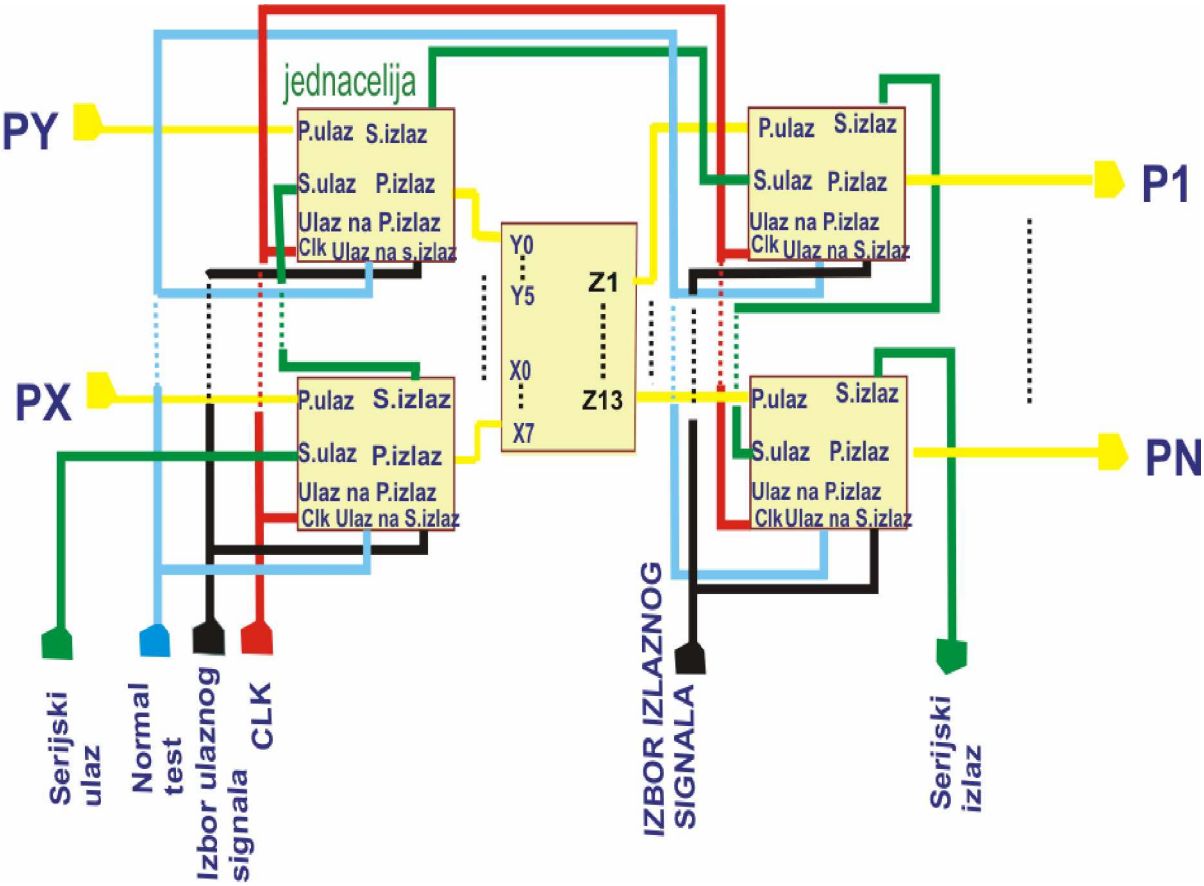


Slika 1. Princip realizacija Boundary-Scan-a



Slika 2. Struktura Boundary-Scan ćelije

1.3 Blok šema testiranja sistema



Slika 3 Blok šema sistema za testiranje

Na slici 3 prikazana je blok šema sistema za testiranje, DUT&BST. U konkretnom slučaju DUT može biti implementiran kao množač ili ALU. DUT prihvata dva operanda koji se dovode na ulaze Y0-Yn i X0-Xm, respektivno, a rezultat se generiše na izlazima Z0-Zp. Kada je aktivna ALU važeći su Y0-Y7, X0-X7 i Z0-Z8, a kada je aktivan množač važeći su ulazoi Y0-Y5, X0-X7, a izlazi su Z0-Z12.

PX_i i=1,..,m – Dovodi se operand X
PY_j j=1,..,n – Dovodi se operand Y
P_k k=1,..,p – Izazni rezultat Z

Serijski ulaz - Ulaz preko koga se unose podaci kojima proveravamo rad kola. Ti podaci se unose bit bo bit

Normal/Test – Bira režim-rada (mod) tj. da li se kolu koje proveravamo prosleđuju podaci sa paralelnog ulaza ili iz flip-flopova u koje smo prethodno serijski uneli podatke

Izbor ulaznog signala – Treba da bude 1 ukoliko želimo serijski da ušiftamo podatak

Izbor izlaznog signala – Treba da bude 1 ako želimo serijski da izšiftamo podatke

Serijski izlaz – Dobijeni rezultat se serijski prosleđuju TAP-u

CLK – Clock

2. Upoznavanje sa konceptom testiranja

1.Upoznati se sa strukturom osnovnih gradivnih blokova Boundary-Scan arhitekture na nivou šeme i na nivou VHDL koda

2. Upoznati se sa Boundary-Scan arhitekturom na nivou VHDL koda

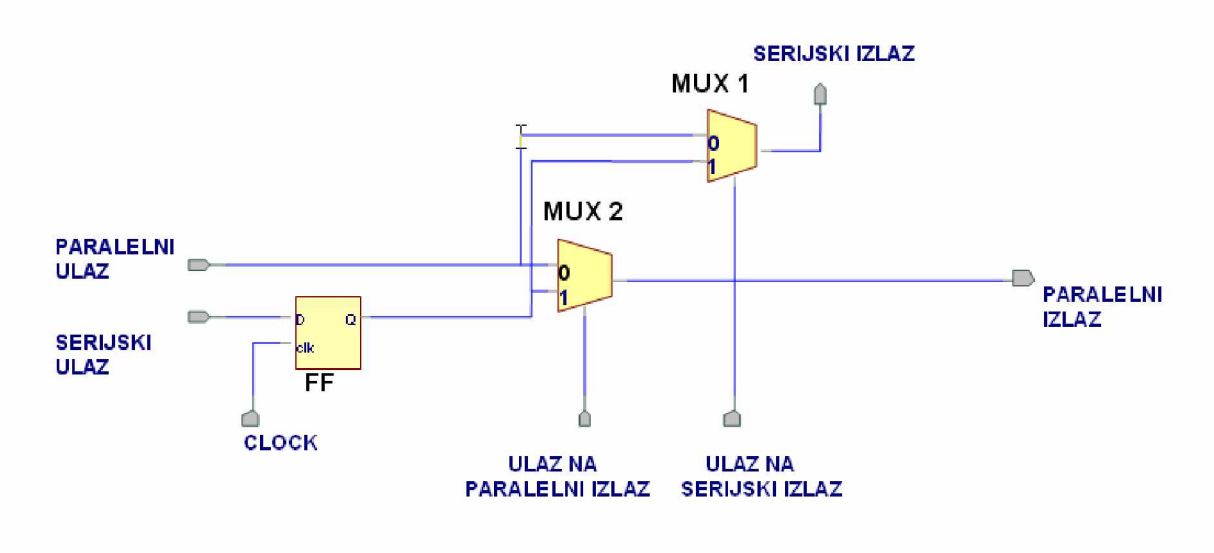
3. Upoznati se sa konceptom testiranja:

a) Množača dva neoznačenih brojeva obima 8 i 6 bitova, respektivno

b) Aritmetičko logičke jedinice (ALU) koja operiše sa dva 8-bitna operanda

Boundary-Scan ćelija se sastoji od jednog master-slave D flip-flopa i dva multipleksera.

Šema Boundary-Scan ćelijeje data na slici 4.a), a odgovarajući VHDL kod na slici 4.b).



a) Boundary-Scan ćelija

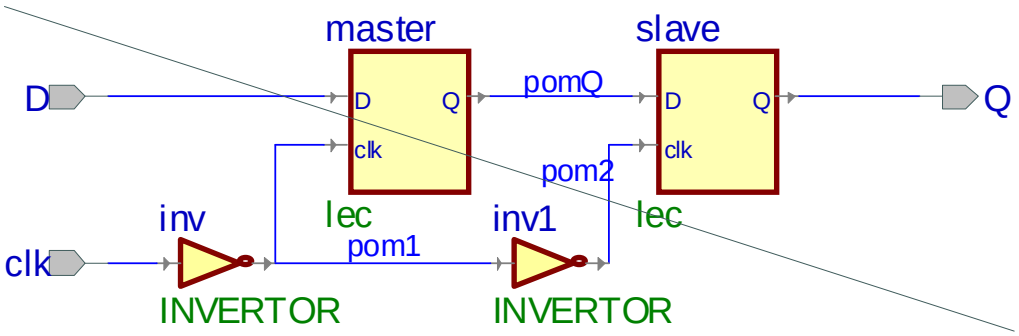
```
library IEEE;
use IEEE.std_logic_1164.all;

entity jednacelija is
  port(
      clk : in STD_LOGIC;
      paralelni_ulaz : in STD_LOGIC;
      seriski_ulaz : in STD_LOGIC;
      ulaz_na_paralelni_izlaz : in STD_LOGIC;
      ulaz_na_seriski_izlaz : in STD_LOGIC;
      paralelni_izlaz : out STD_LOGIC;
      seriski_izlaz : out STD_LOGIC
  );
end jednacelija;
architecture jednacelija of jednacelija is
---- Component declarations -----
component flipflop
  port (
      D : in STD_LOGIC;
      clk : in STD_LOGIC;
      Q : out STD_LOGIC
  );
```

```
end component;
component mux
  port (
       izbor : in STD_LOGIC;
       ulaz0 : in STD_LOGIC;
       ulaz1 : in STD_LOGIC;
       izlaz : out STD_LOGIC
  );
end component;
---- Signal declarations used on the diagram ----
signal pom_sulaz : STD_LOGIC;

begin
----  Component instantiations  ----
ff : flipflop
  port map(
       D => seriski_ulaz,
       Q => pom_sulaz,
       clk => clk
  );
mux_p_izlaz : mux
  port map(
       izbor => ulaz_na_paralelni_izlaz,
       izlaz => paralelni_izlaz,
       ulaz0 => paralelni_ulaz,
       ulaz1 => pom_sulaz
  );
mux_s_izlaz : mux
  port map(
       izbor => ulaz_na_seriski_izlaz,
       izlaz => seriski_izlaz,
       ulaz0 => paralelni_ulaz,
       ulaz1 => pom_sulaz
  );
```

b) VHDL kod

Slika 4 Boundary-Scan ćelija i VHD kod

Šema D flip-flopa je data na slici 5.a), odgovarajući VHDL kod na slici 5.b)



a) Šema D flip-flopa

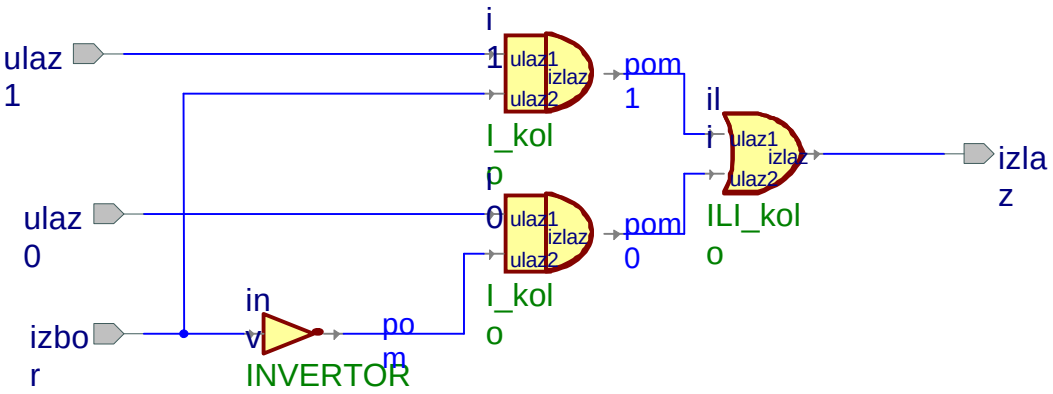
```
library IEEE;
use IEEE.std_logic_1164.all;

entity flipflop is
  port(
       D : in STD_LOGIC;
       clk : in STD_LOGIC;
       Q : out STD_LOGIC
  );
end flipflop;
```

```
architecture flipflop of flipflop is

---- Component declarations -----

component INVERTOR
  port (
       ulaz : in STD_LOGIC;
       izlaz : out STD_LOGIC
  );
end component;
component lec
  port (
       D : in STD_LOGIC;
       clk : in STD_LOGIC;
       Q : out STD_LOGIC
  );
end component;

---- Signal declarations used on the diagram ----
signal pom1 : STD_LOGIC;
signal pom2 : STD_LOGIC;
signal pomQ : STD_LOGIC;

begin

----  Component instantiations  ----

inv : INVERTOR
  port map(
       izlaz => pom1,
       ulaz => clk
  );

inv1 : INVERTOR
  port map(
       izlaz => pom2,
       ulaz => pom1
  );

master : lec
  port map(
       D => D,
    Q => pomQ,
       clk => pom1
  );

slave : lec
  port map(
       D => pomQ,
       Q => Q,
       clk => pom2
  );

end flipflop;
```

b) VHDL kod

Slika 5 D flip-flop i VHDL kod

Šema multipleksera je prikazana na slici 6.a), a VHDL kod na slici 6.b).



a) Šema Multipleksera

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux is
  port(
      izbor : in STD_LOGIC;
      ulaz0 : in STD_LOGIC;
      ulaz1 : in STD_LOGIC;
      izlaz : out STD_LOGIC
  );
end mux;
architecture mux of mux is

---- Component declarations -----

component ILI_kolo
  port (
      ulaz1 : in STD_LOGIC;
      ulaz2 : in STD_LOGIC;
      izlaz : out STD_LOGIC
  );
end component;
component INVERTOR
  port (
      ulaz : in STD_LOGIC;
      izlaz : out STD_LOGIC
  );
end component;
component I_kolo
  port (
      ulaz1 : in STD_LOGIC;
      ulaz2 : in STD_LOGIC;
      izlaz : out STD_LOGIC
  );
end component;

---- Signal declarations used on the diagram ----
signal pom : STD_LOGIC;
signal pom0 : STD_LOGIC;
signal pom1 : STD_LOGIC;

begin
```

```
----  Component instantiations  ----
i0 : I_kolo
  port map(
       izlaz => pom0,
       ulaz1 => ulaz0,
       ulaz2 => pom
  );
i1 : I_kolo
  port map(
       izlaz => pom1,
       ulaz1 => ulaz1,
       ulaz2 => izbor
  );
ili : ILI_kolo
  port map(
       izlaz => izlaz,
       ulaz1 => pom1,
       ulaz2 => pom0
  );
inv : INVERTOR
  port map(
       izlaz => pom,
       ulaz => izbor
  );
end mux;
```

b) VHDL kod

Slika 6 Multipleksor i VHDL kod

2.2.1 VHDL kod Boundary-Scan arhitekture

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
entity kolozatest is
            generic(a : integer ;   -- broj ulaznih signala za X
            b : integer ;           -- broj ulaznih signala za Y
          c : integer     );        -- broj izlaznih signala za Z
            port(paralelni_ulaz :in  unsigned (a+1+b+1+c downto 0) ; -- paralelni ulaz za siulazne sinale X i Y kao i za signal Zpom koji
je izlaz iz testiranog kola
                paralelni_izlaz :out  unsigned (a+1+b+1+c downto 0) ;       -- rezultat operacija testiranog kola
                clk , normal_test , izbor_ulaznog_siftanja , izbor_izlaznog_siftanja , seriski_ulaz : in STD_LOGIC;
                        -- clk
                        -- normal_test omogucava da kolo radi sa vrednostima paralelnog ulaza (stanje 0) dok se unose seriski podaci
kada se zavrsi unos omogucava njihov prikaz (stanje 1)
                        -- izborr_ulaznog_siftanja odredjuje sta da se sifta na ulazu (uvek je 1 da bi seriska informacija kruzila )
                        -- izbor_izlaznog_siftanja       omogucava citanje test rezultata iz testiranog kola i prenos u serisku informaciju
(stanje 0) i siftovanje postojece seriske informacije (stanje 1)
                        -- seriski_ulaz za testiranje
                        seriski_izlaz : out STD_LOGIC );          -- rezultat testiranja
end kolozatest ;

architecture kolozatest of kolozatest is
component jednacelija is        --  kolozatest se sastoji od a+1+b+1+c+1 odgovarajuce povezanih jedinicnih celija
  port(
       clk : in STD_LOGIC;
       paralelni_ulaz : in STD_LOGIC;
       seriski_ulaz : in STD_LOGIC;
       ulaz_na_paralelni_izlaz : in STD_LOGIC;
       ulaz_na_seriski_izlaz : in STD_LOGIC;
       paralelni_izlaz : out STD_LOGIC;
       seriski_izlaz : out STD_LOGIC
  );
end component jednacelija;
signal i : integer := 0;              -- signal petlje
```

```
signal pom :  STD_LOGIC_vector (a+1+b+1+c+1 downto 0);   -- signal za povezivanje celija kroz koji kruzi seriska iformacija
begin
            pom(0) <=  seriski_ulaz ;

            ulaznapetlja : for i in 0 to a+1+b
                  generate
                              u :   jednacelija
                    port map( clk => clk ,
                    paralelni_ulaz => paralelni_ulaz (i) ,
                    ulaz_na_paralelni_izlaz => normal_test ,
                    ulaz_na_seriski_izlaz =>     izbor_ulaznog_siftanja ,
                    seriski_ulaz => pom (i) ,
                    paralelni_izlaz => paralelni_izlaz(i) ,
                    seriski_izlaz => pom (i+1) );
                                          end generate ulaznapetlja ;

                  izlaznapetlja : for i in a+1+b+1 to a+1+b+1+c
                  generate
                              u :   jednacelija
                    port map( clk => clk ,
                    paralelni_ulaz => paralelni_ulaz (i) ,
                    ulaz_na_paralelni_izlaz => normal_test ,
                    ulaz_na_seriski_izlaz =>     izbor_izlaznog_siftanja ,
                    seriski_ulaz => pom (i) ,
                    paralelni_izlaz => paralelni_izlaz(i) ,
                    seriski_izlaz => pom (i+1) );
                                          end generate izlaznapetlja ;


      seriski_izlaz <= pom (a+1+b+1+c+1) ;
end kolozatest ;
```

2.2.2 Integracija Boundary-Scan arhitekture i DUT-a

U konkretnom slučaju razmatraćemo povezivanje Boundary-Scan arhitekture sa sledeća dva tipa DUT-a:

a) Množača dva neoznačena broja (X=8, Y=6, Z=14)

b) ALU (X=8, Y=8, Z=8, C_{out}=1)

2.2.2.a Integracija Boundary-Scan arhitekture i množača

```
library IEEE;
use IEEE.std_logic_1164.all, IEEE.NUMERIC_STD.all;

entity testmultipliers is
      port (
            X: in unsigned (7 downto 0);
            Y: in unsigned (5 downto 0);
            Z: out unsigned (13 downto 0) ;
        clk , normal_test , izbor_ulaznog_siftanja , izbor_izlaznog_siftanja , seriski_ulaz : in STD_LOGIC;
        seriski_izlaz : out STD_LOGIC          );
end entity testmultipliers;
architecture testmultipliers of testmultipliers is
component multipliers is
      port (
            X: in unsigned (7 downto 0);
            Y: in unsigned (5 downto 0);
            Z: out unsigned (13 downto 0)
      );
end component multipliers;
component kolozatest is
      generic(a : integer ;
      b : integer ;
    c : integer     );
      port(paralelni_ulaz :in  unsigned (a+1+b+1+c downto 0) ;
          paralelni_izlaz :out  unsigned (a+1+b+1+c downto 0) ;
          clk , normal_test , izbor_ulaznog_siftanja , izbor_izlaznog_siftanja , seriski_ulaz : in STD_LOGIC;
```

```
                        seriski_izlaz : out STD_LOGIC );
end component kolozatest ;
signal      Xpom : unsigned (7 downto 0);
signal      Ypom :    unsigned (5 downto 0);
signal      Zpom : unsigned (13 downto 0)              ;
signal      inpom , outpom :  unsigned (27 downto 0);
signal   i : integer ;
begin
                        petlja13 : for i in 0 to 13
                        generate
                                inpom (i+14) <= Zpom (i) ;
                                        Z (i) <= outpom (i+14) ;
                        end generate petlja13 ;
                petlja7 : for i in 0 to 7
                        generate
                                inpom (i) <= X (i) ;
                                        Xpom (i) <= outpom (i) ;
                        end generate petlja7 ;

                        petlja5 : for i in 0 to 5
                        generate
                                inpom (i+8) <= Y (i) ;
                                        Ypom (i) <= outpom (i+8) ;
                        end generate petlja5 ;



                        test : kolozatest
                    generic map (a => 7 ,
                     b => 5 ,
                     c => 13 )
                    port map ( paralelni_ulaz => inpom ,
                              paralelni_izlaz => outpom ,
                              clk => clk ,
                                        normal_test => normal_test ,
                                izbor_ulaznog_siftanja => izbor_ulaznog_siftanja ,
                                                izbor_izlaznog_siftanja          => izbor_izlaznog_siftanja ,
                                        seriski_ulaz => seriski_ulaz ,
                                seriski_izlaz => seriski_izlaz );

                        mnozac : multipliers
                                 port map (
                        X => Xpom ,
                        Y => Ypom ,
                                Z => Zpom
                          );

end testmultipliers;
```

2.2.2.b Integracija Boundary-Scan arhitekture i ALU

```
library IEEE;
use IEEE.std_logic_1164.all, IEEE.NUMERIC_STD.all;

entity testalu is
            port (
                        X: in unsigned (7 downto 0);
                        Y: in unsigned (7 downto 0);
                        Z: out unsigned (8 downto 0) ;
                        Sel:                    in unsigned(4 downto 0);

                        CarryIn:    in std_logic;
                clk , normal_test , izbor_ulaznog_siftanja , izbor_izlaznog_siftanja , seriski_ulaz : in STD_LOGIC;
                seriski_izlaz : out STD_LOGIC           );
end entity testalu;
architecture testalu of testalu is
component alu is
            port (Sel:              in unsigned(4 downto 0);
                 CarryIn:           in std_logic;
                 X,Y:               in unsigned(7 downto 0);
```

```
                Z:                  out unsigned(7 downto 0);
                CarryOut: out std_logic);
end component alu ;
component kolozatest is
            generic(a : integer ;
            b : integer ;
        c : integer     );
            port(paralelni_ulaz :in  unsigned (a+1+b+1+c downto 0) ;
                paralelni_izlaz :out  unsigned (a+1+b+1+c downto 0) ;
                clk , normal_test , izbor_ulaznog_siftanja , izbor_izlaznog_siftanja , seriski_ulaz : in STD_LOGIC;
                        seriski_izlaz : out STD_LOGIC );
end component kolozatest ;
signal      Xpom , Ypom  : unsigned (7 downto 0);
signal      Zpom : unsigned (8 downto 0);
signal      inpom  :  unsigned (24 downto 0);
signal      outpom :  unsigned (24 downto 0);
signal  i : integer ;
begin
                        petlja13 : for i in 0 to 8
                        generate
                                inpom (i+16) <= Zpom (i) ;
                                        Z (i) <= outpom (i+16) ;
                        end generate petlja13 ;
                petlja7 : for i in 0 to 7
                        generate
                                inpom (i) <= X (i) ;
                                        Xpom (i) <= outpom (i) ;
                        end generate petlja7 ;
                        petlja5 : for i in 0 to 7
                        generate
                                inpom (i+8) <= Y (i) ;
                                        Ypom (i) <= outpom (i+8) ;
                        end generate petlja5 ;
                        test : kolozatest
                    generic map (a => 7 ,
                    b => 7 ,
                    c => 8 )
                    port map ( paralelni_ulaz => inpom ,
                            paralelni_izlaz => outpom ,
                            clk => clk ,
                                    normal_test => normal_test ,
                                izbor_ulaznog_siftanja => izbor_ulaznog_siftanja ,
                                                        izbor_izlaznog_siftanja         => izbor_izlaznog_siftanja ,
                                    seriski_ulaz => seriski_ulaz ,
                                izbor_izlaznog_siftanja => seriski_izlaz );
                  alu1 : alu
                                 port map (
                        X => Xpom ,
                        Y => Ypom ,
                                Z => Zpom( 7 DOWNTO 0) ,
                                                        Sel => Sel ,
                                                        CarryOut => Zpom(8) ,
                                CarryIn => CarryIn
                                );
end testalu ;
```

2.3 Princip rada

Kod kola sa slike 3 mogu se identifikovati sledeći ulazi i izlazi :

a) CLOCK (ulazni) - 1-bit
b) NORMAL / TEST (ulazni) - 1-bit
c) IZBOR ULAZNOG SIGNALA (ulazni) - 1-bit
d) IZBOR IZLAZNOG SIGNALA (ulazni) - 1-bit
e) SERIJSKI ULAZ (ulazni) - 1-bit
f) SERIJSKI IZLAZ (izlazni) - 1-bit
g) PARALELNI ULAZ (ulazni) – *n* bitova

h) PARALELNI IZLAZ (izlazni) – *m* bitova

U režimu rada **Normal** (kada ne testiramo kolo) Boundary-Scan arhitektura radi transparentno tj. paralelno prenosi signale sa ulaza na izlaz.
Za testiranje kola neophodne su sledeće aktivnosti:

a) Na **Serijski ulaz** *(vidi sliku 3)* dovode se test sekvenca u obliku serijske povorke bitova kojom se testira DUT. Za vreme unosa test sekvence DUT i dalje radi u normalnom režimu, tako da je unos serijskog podatka potpuno transparentan na rad kola.

b) Kada se kompletna test sekvenca unese, Boundary-Scan arhitekturi se izdaje nalog za prelazak u **Test** režim rada. Po automatizmu se prekida sa režimom rada **Normal** i paralelno se prosleđuje iz UC_i ćelija *(vidi sliku 1)* prethodno serijski uneta test sekvenca na ulaze DUT-a sa prednjom ivicom **Test** impulsa. Sa zadnjom impulsom Test impulsa prihvata se odziv DUT-a od strane IC_j ćelija. **Test** režim traje jedan taktni interval. Nakon toga ponovo se prelazi u **Normal** režim. Signale Boundary-Scan lanca (UC_i i IC_j) pomeramo na **Serijski izlaz** radi potrebe analize. Analizu obavlja logika za analizu rezultata. Upoređivanjem odziva DUT-a sa očekivanim utvrđuje se korektnost rada DUT-a.

2.3.a Testiranje množača 8x6

Proceduru testiranja sprovesti na sledeći način:

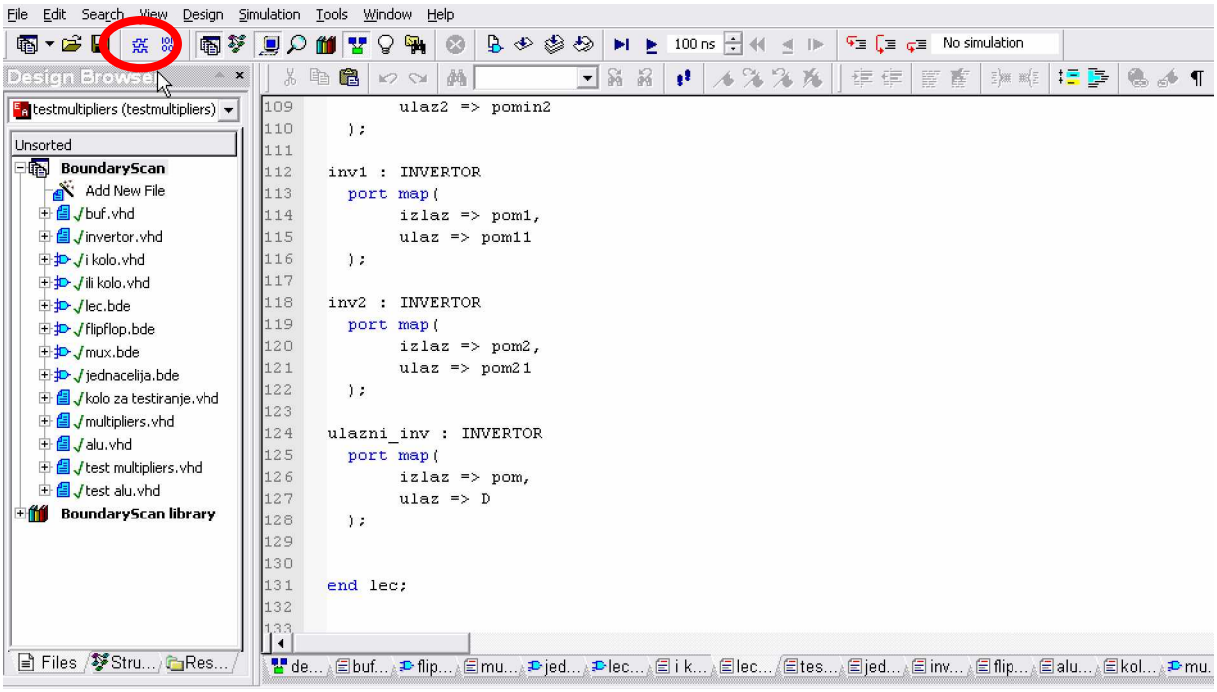
1)Pokrenuti VHDL i učitati Boundary-Scan projekat

2)Selektovati ime koda na levoj strani ekrana koji želimo da pokrenemo (u ovom slučaju, to je test multipliers.vhd)

3)Kliknuti na ikonicu koja je zaokružena crvenim na slici 8, da bi pokrenuli waveform za testiranje

4)Desnim klikom na waveform - add signals dodati signale sa tabele 1

5)Proceduru testiranja sprovesti prema Tabeli 1



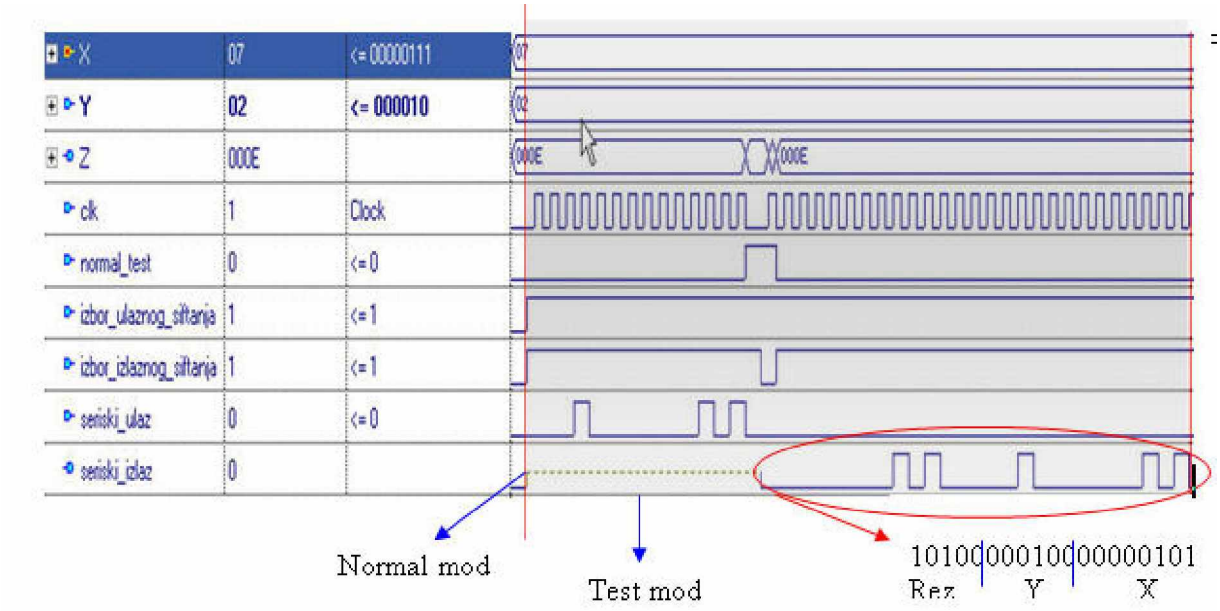
Slika 8 Interface u VHDL-u u pokretanje waveform-a

Tabela 1-Definicija test signala množača(ulazi X(8 bitova),Y(6 bitova))

ULAZI	NORMANI REŽIM	TEST REŽIM			
		takt 1-14	takt 15	takt 16	takt 17+
CLK	NA	A	NA	A	A
NORMAL_TEST	0	0	1	1	0
IZBOR_ULAZNOG	0	1	1	1	1
IZBOR_IZLAZNOG	0	1	1	0	1
SERIJSKI_ULAZ	0	bit	0	0	0

Napomena: Simbol **A** *se odnosi na aktivan signal; simbol* **NA** *na neaktivan; skraćenica* **bit** *se odnosi na 14-bitnu testnu sekvencu*

Dobijeni waveform za konkretan slučaj je prikazan na slici 9



Slika 9 testiranje množača – simulacija rada u VHDL-u

Postavili smo stanje kola kao na slici (dijagram pre vertikalne crvene crte), i uneli za X 00000111 i za Y 000010. Posle prvog kloka množač je izračunao rezultat (stanje na Z). Signali su dovedeni paralelno na množač i isto tako prosleđeni na izlaz kao da nema kola za testiranje.
U drugom koraku testiramo kolo preko serijskog ulaza. Serijski smo uneli prvo šestobitni Y 000100 i osmobitni X 00000101. Proizvod ova dva broja je 10100. Na obeleženom delu na slici je predstavljen ovaj niz u konkretnom slučaju koji se nalazi na serijskom izlazu.

2.3.b Testiranje ALU 8x8

Proceduru testiranja sprovesti na sledeći način:

1)Pokrenuti VHDL i učitati Boundary-Scan projekat

2)Selektovati ime koda na levoj strani ekrana koji želimo da pokrenemo (u ovom slučaju, to je test test alu.vhd

3)Kliknuti na ikonicu koja je zaokružena crvenim na slici 8, da bi pokrenuli waveform za testiranje

4)Desnim klikom na waveform - add signals dodati signale sa tabele 2

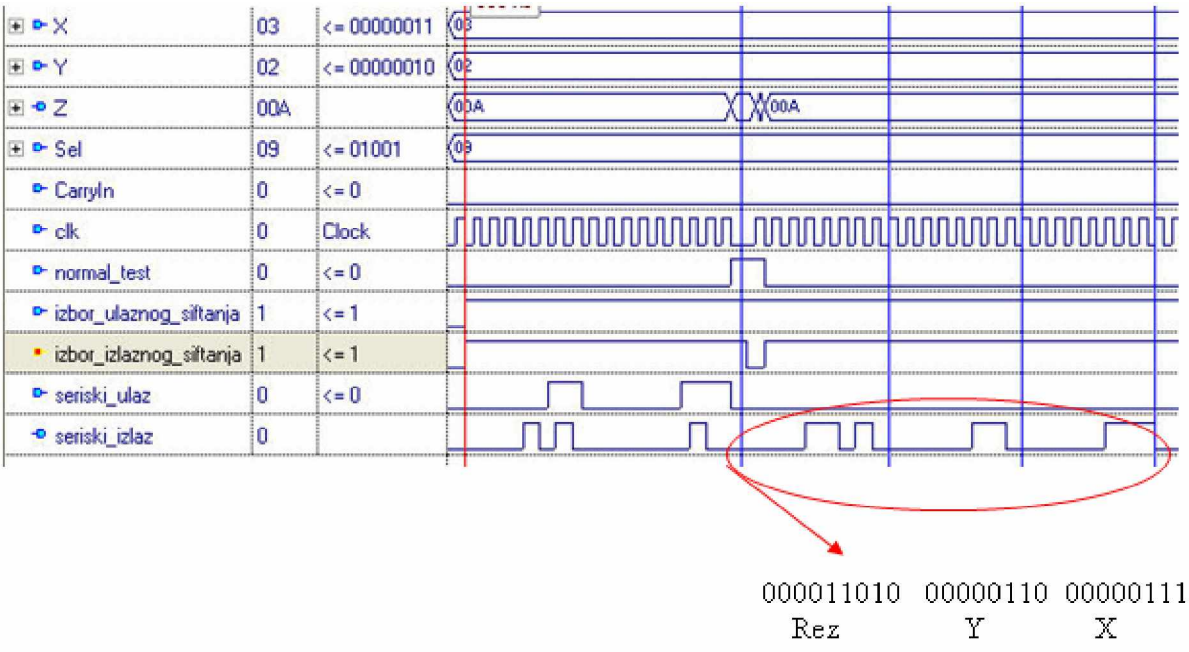
5)Proceduru testiranja sprovesti prema Tabeli 2

Tabela 2 Testiranja kola za množenje (ulazi X (8 bitova) Y (8 bitova))

ULAZI	NORMANI REŽIM	TEST REŽIM			
		takt 1-16	takt 17	takt 18	takt 18+
CLK	NA	A	NA	A	A
NORMAL_TEST	0	0	1	1	0
IZBOR_ULAZNOG	0	1	1	1	1
IZBOR_IZLAZNOG	0	1	1	0	1
SERIJSKI_ULAZ	0	bit	0	0	0

Napomena: Simbol **A** *se odnosi na aktivan signal; simbol* **NA** *na neaktivan; skraćenica* **bit** *se odnosi na 16-bitnu testnu sekvencu*

Dobijeni waveform za konkretan slučaj je prikazan na slici 10



Slika 10 testiranje ALU-a – simulacija rada u VHDL-u

Rad scan kola za testiranje ALU je u principu isti kao i scan kola za testiranje množača. Pri testiranju, ALU smo postavili za rad u aritmetičkom modu i obavljanje sabiranja ulaza i šiftovanje rezultata u levo za 1. Pošto na ulazima kola imamo X (8 bitova) i Y (8 bitova), a na izlazu Z sa carry-jem (9 bitova), razlika u odnosu na testiranje množača je jedino u tome što nam serijsko unošenje X i Y ne traje 14 taktova kao u prethodnom primeru, već, logično, 16. Takođe, dobijanjem rezultata obrade serijski unešenih podataka, izlazno šiftovanje traje dok se TAP-u serijski ne prosledi prvo izlaz, a odmah zatim i Y i X (8+8+9=25 taktova).

DODATAK A

U ovom dodatku dati su VHDL kodovi za sledeće osnovne gradivne blokove ćelije Boundary-Scan-a:

1) Bafera
2) Invertora
3) I kola
4) ILI kola
5) Leč kola

A.1 Kod bafera

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity BUF is
        port ( ulaz : in std_logic;          -- ulaz kola
               izlaz : out std_logic );      -- izlaz kola
end entity BUF ;
architecture BUF of BUF is
begin
        izlaz <=  ulaz;                      -- funkcija kola
end architecture BUF ;
```

A.2 Kod invertora

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity INVERTOR is
        port ( ulaz : in std_logic;        -- ulazni signal kola
               izlaz : out std_logic );    -- izlazni signal kola
end entity INVERTOR ;
architecture INVERTOR of INVERTOR is
begin
        izlaz <=  not ulaz;                -- funkcija kola
end architecture INVERTOR;
```

A.3 Kod I kola

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity I_kolo is
        port ( ulaz1 : in std_logic;       -- ulaz kola
               ulaz2 : in std_logic;       -- ulaz kola
               izlaz : out std_logic );    -- izlaz kola
end entity I_kolo ;
architecture I_kolo of I_kolo is
begin
        izlaz <= ulaz1 and ulaz2;          -- funkcija kola
end architecture I_kolo;
```

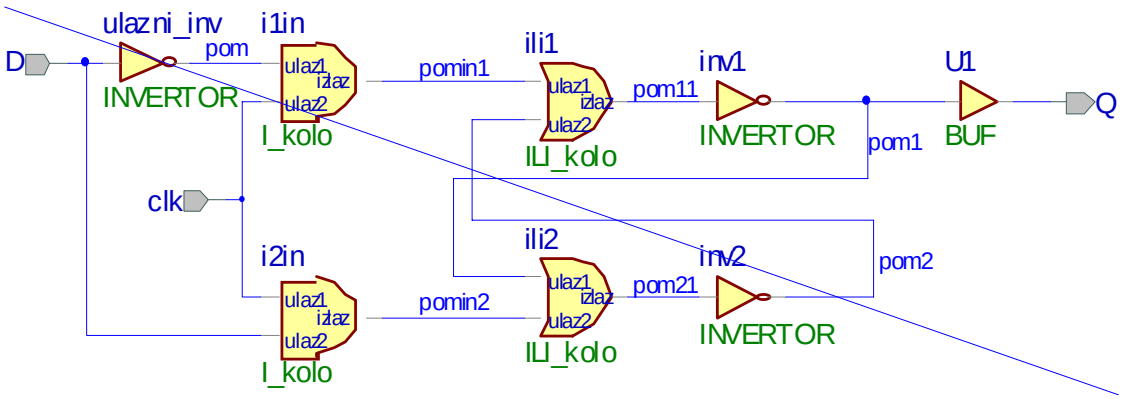
A.4 Kod ILI kola

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.all;
entity ILI_kolo is
        port ( ulaz1 : in std_logic;        -- ulaz kola
      ulaz2 : in std_logic;                 -- ulaz kola
             izlaz : out std_logic );       -- izlaz kola
end entity ILI_kolo ;
architecture ILI_kolo of ILI_kolo is
begin
        izlaz <= ulaz1 or ulaz2;
end architecture ILI_kolo;                  -- funkcija kola
```

A.5 Kod leč kola

Šema leč kola je data na slici A.5.a), odgovarajući VHDL kod na slici A.5.b)



a) Šema Leč kola

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;
entity lec is
  port(
      D : in STD_LOGIC;
      clk : in STD_LOGIC;
      Q : out STD_LOGIC
  );
end lec;
architecture lec of lec is
---- Component declarations -----
component BUF
  port (
      ulaz : in STD_LOGIC;
      izlaz : out STD_LOGIC
  );
end component;
component ILI_kolo
  port (
      ulaz1 : in STD_LOGIC;
      ulaz2 : in STD_LOGIC;
      izlaz : out STD_LOGIC
  );
end component;
component INVERTOR
  port (
      ulaz : in STD_LOGIC;
      izlaz : out STD_LOGIC
  );
end component;
component I_kolo
  port (
      ulaz1 : in STD_LOGIC;
      ulaz2 : in STD_LOGIC;
```

```
      izlaz : out STD_LOGIC
  );
end component;
---- Signal declarations used on the diagram ----
signal pom : STD_LOGIC;
signal pom1 : STD_LOGIC;
signal pom11 : STD_LOGIC;
signal pom2 : STD_LOGIC;
signal pom21 : STD_LOGIC;
signal pomin1 : STD_LOGIC;
signal pomin2 : STD_LOGIC;
begin
----  Component instantiations  ----
U1 : BUF
  port map(
      izlaz => Q,
      ulaz => pom1
  );
i1in : I_kolo
  port map(
      izlaz => pomin1,
      ulaz1 => pom,
      ulaz2 => clk
  );
i2in : I_kolo
  port map(
      izlaz => pomin2,
      ulaz1 => clk,
      ulaz2 => D
  );
ili1 : ILI_kolo
  port map(
      izlaz => pom11,
      ulaz1 => pomin1,
      ulaz2 => pom2
  );
ili2 : ILI_kolo
  port map(
      izlaz => pom21,
      ulaz1 => pom1,
      ulaz2 => pomin2
  );
inv1 : INVERTOR
  port map(
      izlaz => pom1,
      ulaz => pom11
  );
inv2 : INVERTOR
  port map(
      izlaz => pom2,
      ulaz => pom21
  );
ulazni_inv : INVERTOR
  port map(
      izlaz => pom,
      ulaz => D
  );
end lec;
```

Slika A.5 Leč kolo I VHDL kod

ZADATAK

Proveriti ispravnost rada množača i ALU koristeći navedene test operande, i napisati dobijenu izlaznu serijsku sekvencu (u binarnom i dekadnom obliku). Iščitavanje vršiti tako da se obavi provera i unešenih operanada. Kod množača se serijski prvo unosi šestobitni pa osmobitni broj, a kod ALU su oba ulazna operanda osmobitna.

Grupa 1

operand 1 = 6
operand 2 = 28

Dobijena test sekvenca:

množač: _____ _____
sekvenca u binarnom obliku | sekvenca u dekadnom obliku

ALU: **OR** _____ _____
funkcija ALU | sekvenca u binarnom obliku | sekvenca u dekadnom obliku

Grupa 2

operand 1 = 43
operand 2 = 8

Dobijena test sekvenca:

množač: _____ _____
sekvenca u binarnom obliku | sekvenca u dekadnom obliku

ALU: **inkrement A** _____ _____
funkcija ALU | sekvenca u binarnom obliku | sekvenca u dekadnom obliku

Grupa 3

operand 1 = 8
operand 2 = 9

Dobijena test sekvenca:

množač _____ _____
sekvenca u binarnom obliku | sekvenca u dekadnom obliku

ALU: **A+B** _____ _____
funkcija ALU | sekvenca u binarnom obliku | sekvenca u dekadnom obliku

Grupa 4

operand 1 = 12
operand 2 = 13

Dobijena test sekvenca:

množač _____ _____
sekvenca u binarnom obliku | sekvenca u dekadnom obliku

ALU: **dekrement A** _____ _____
funkcija ALU | sekvenca u binarnom obliku | sekvenca u dekadnom obliku

Grupa 5

operand 1 = 3
operand 2 = 27

Dobijena test sekvenca:

množač _____ _____
sekvenca u binarnom obliku | sekvenca u dekadnom obliku

ALU: $\overline{\mathbf{A+B}}$ _____ _____
funkcija ALU | sekvenca u binarnom obliku | sekvenca u dekadnom obliku

Grupa 6

operand 1 = 8
operand 2 = 9

Dobijena test sekvenca:

množač _____ _____
sekvenca u binarnom obliku | sekvenca u dekadnom obliku

ALU: **A+B** _____ _____
funkcija ALU | sekvenca u binarnom obliku | sekvenca u dekadnom obliku

Grupa 7

operand 1 = 17
operand 2 = 5

Dobijena test sekvenca:

množač _____ _____
sekvenca u binarnom obliku | sekvenca u dekadnom obliku

ALU: **A-B** _____ _____
funkcija ALU | sekvenca u binarnom obliku | sekvenca u dekadnom obliku

Grupa 8

operand 1 = 15
operand 2 = 7

Dobijena test sekvenca:

množač _____ _____
sekvenca u binarnom obliku | sekvenca u dekadnom obliku

ALU: **prenos A** _____ _____
funkcija ALU | sekvenca u binarnom obliku | sekvenca u dekadnom obliku

Grupa 9

operand 1 = 4
operand 2 = 9

Dobijena test sekvenca:

množač _____ _____
sekvenca u binarnom obliku | sekvenca u dekadnom obliku

ALU: **AND** _____ _____
funkcija ALU | sekvenca u binarnom obliku | sekvenca u dekadnom obliku

Grupa 10

operand 1 = 5
operand 2 = 3

Dobijena test sekvenca:

množač _____ _____
sekvenca u binarnom obliku | sekvenca u dekadnom obliku

ALU: **XOR** _____ _____
funkcija ALU | sekvenca u binarnom obliku | sekvenca u dekadnom obliku

Grupa 11

operand 1 = 5
operand 2 = 21

Dobijena test sekvenca:

množač _____ _____
sekvenca u binarnom obliku | sekvenca u dekadnom obliku

ALU: $\overline{\mathbf{A}}$ _____ _____
funkcija ALU | sekvenca u binarnom obliku | sekvenca u dekadnom obliku

Grupa 12

operand 1 = 7
operand 2 = 12

Dobijena test sekvenca:

množač _____ _____
sekvenca u binarnom obliku | sekvenca u dekadnom obliku

ALU: **shift left A** _____ _____
funkcija ALU | sekvenca u binarnom obliku | sekvenca u dekadnom obliku

Grupa 13

operand 1 = 4
operand 2 = 2

Dobijena test sekvenca:

množač _____ _____
sekvenca u binarnom obliku | sekvenca u dekadnom obliku

ALU: **shift right A** _____ _____
funkcija ALU | sekvenca u binarnom obliku | sekvenca u dekadnom obliku

Grupa 14

operand 1 = 7
operand 2 = 9

Dobijena test sekvenca:

množač _____ _____
sekvenca u binarnom obliku | sekvenca u dekadnom obliku

ALU: **prenos 0** _____ _____
funkcija ALU | sekvenca u binarnom obliku | sekvenca u dekadnom obliku

Grupa 15

operand 1 = 2
operand 2 = 4

Dobijena test sekvenca:

množač _____ _____
sekvenca u binarnom obliku | sekvenca u dekadnom obliku

ALU: **A+B** _____ _____
funkcija ALU | sekvenca u binarnom obliku | sekvenca u dekadnom obliku