ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΗΜΜΥ

Ψηφιακά Συστήματα VLSI

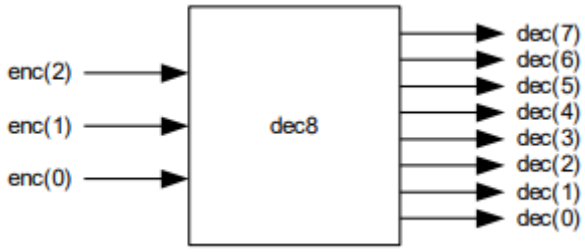
Ακαδημαϊκό έτος 2020-21

1η Εργαστηριακή Άσκηση

Αιμιλία Σιόκουρου – 03117703

Νάταλυ Πεγειώτη – 03117707

Θέμα Α.2: Δυαδικός αποκωδικοποιητής 3 σε 8



Ζητούμενα: Η περιγραφή της οντότητας του δυαδικού αποκωδικοποιητή 3 σε 8 και της αρχιτεκτονικής του σε αρχιτεκτονική dataflow και behavioral VHDL μαζί με τα αντίστοιχα testbench που εξετάζουν την ορθή λειτουργία του κυκλώματος φαίνονται παρακάτω.

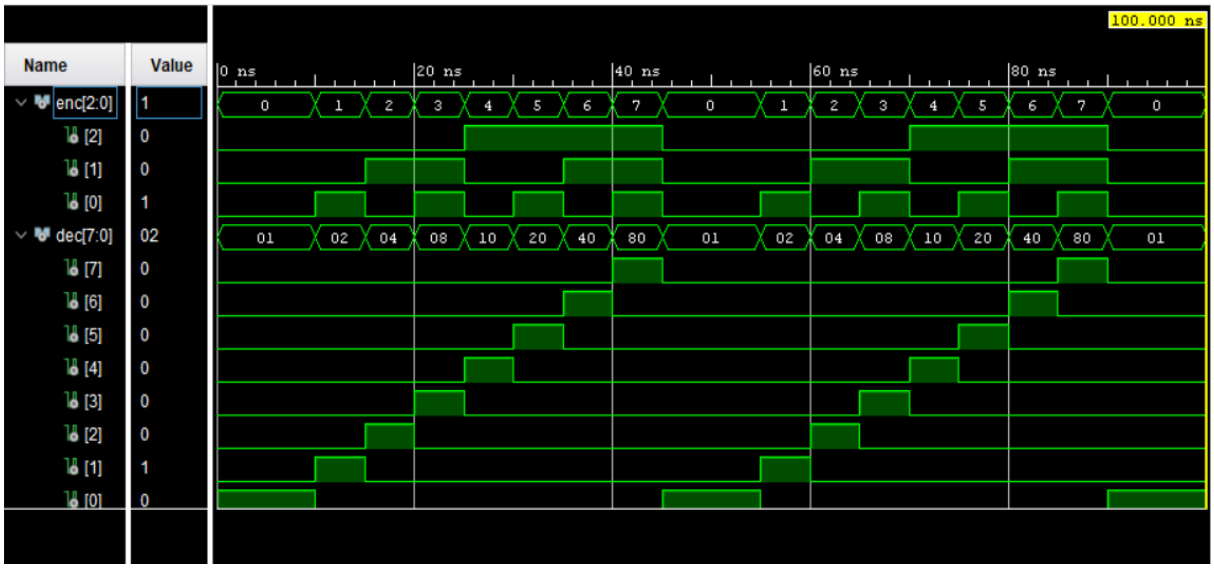
Dataflow:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dec3x8_dataflow is
	Port ( enc : in STD_LOGIC_VECTOR (2 downto 0);
		dec : out STD_LOGIC_VECTOR (7 downto 0));
end dec3x8_dataflow;

architecture Dataflow of dec3x8_dataflow is

begin
	with enc select
	dec <=	"00000001" when "000", --dec(0) ON
		"00000010" when "001", --dec(1) ON
		"00000100" when "010", --dec(2) ON
		"00001000" when "011", --dec(3) ON
		"00010000" when "100", --dec(4) ON
		"00100000" when "101", --dec(5) ON
		"01000000" when "110", --dec(6) ON
		"10000000" when "111", --dec(7) ON
		"00000000" when others; --in other situation dec<=0

end Dataflow;
```

και το αντίστοιχο TestBench:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dec3x8_dataflow_tb is

end dec3x8_dataflow_tb;

architecture Bench of dec3x8_dataflow_tb is
   component dec3x8_dataflow is
   port (
         enc: in std_logic_vector(2 downto 0);
         dec: out std_logic_vector(7 downto 0));
   end component;
signal enc: std_logic_vector(2 downto 0) := (others => '0');
signal dec: std_logic_vector(7 downto 0);

constant delay : time := 5 ns;

begin
 uut : dec3x8_dataflow
      port map (
               enc => enc,
               dec => dec);
stimulus : process

  begin
     enc <= (others => '0');
     wait for delay;
     enc <= "000";
     wait for delay;
     enc <= "001";
     wait for delay;
     enc <= "010";
     wait for delay;
     enc <= "011";
     wait for delay;
     enc <= "100";
     wait for d3.elay;
     enc <= "101";
     wait for delay;
     enc <= "110";
     wait for delay;
     enc <= "111";
     wait for delay;
end process;
end Bench;
```

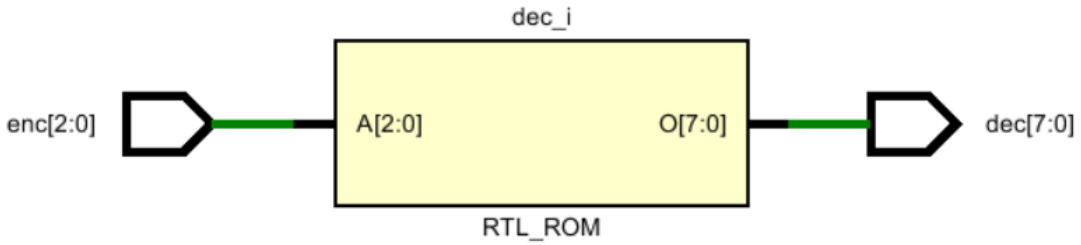
Αφού έχουμε εκτελέσει το testbench τρέχουμε την προσομοίωση(run simulation) η οποία φαίνεται παρακάτω:



Καθώς γνωρίζουμε ότι ο παρακάτω είναι ο πίνακας αληθείας ενός δυαδικού αποκωδικοποιητή μπορούμε να επαληθεύσουμε την λειτουργία του κυκλώματος.

Inputs			Outputs							
x	y	z	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Το **RTL schematic** του dec3x8_dataflow παρουσιάζεται παρακάτω:



όπου επιβεβαιώνουμε ότι το πρόγραμμα έχει καταλάβει ότι δημιουργούμε ένα αποκωδικοποιητή/ROM με είσοδο enc[2:0] και έξοδο dec[7:0].

Behavioral:

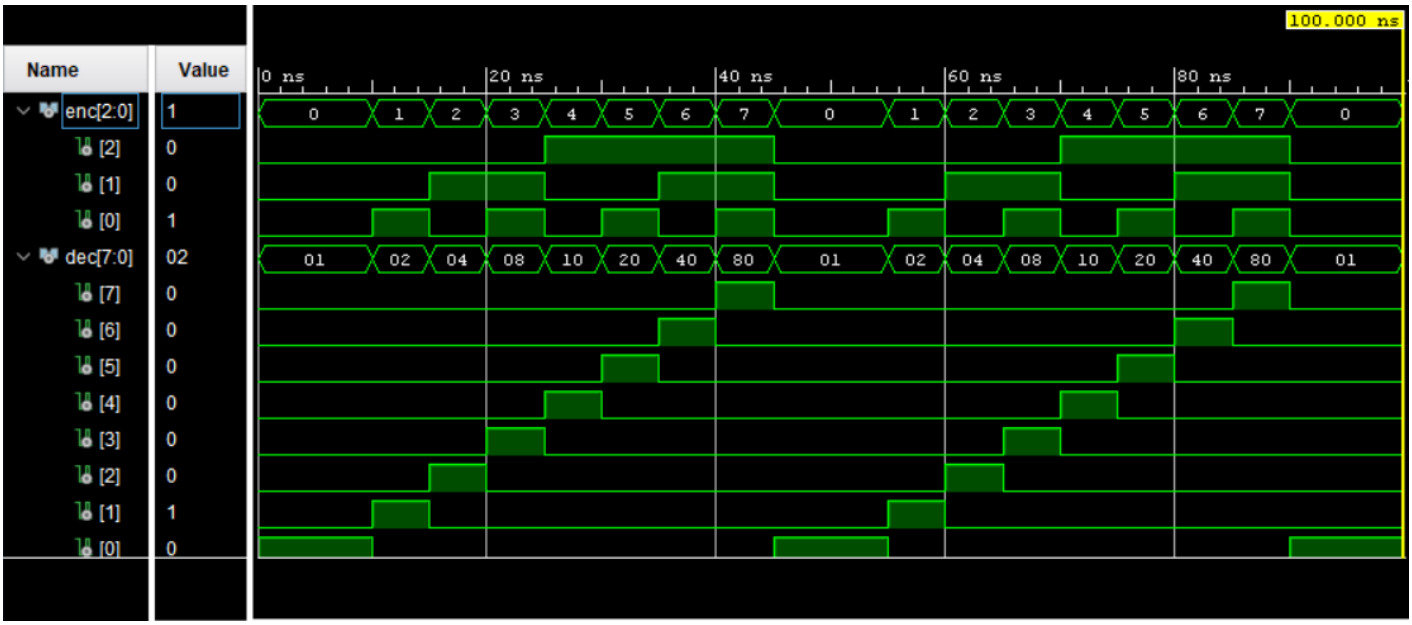
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dec3x8 is
   Port ( enc : in STD_LOGIC_VECTOR (2 downto 0);
          dec : out STD_LOGIC_VECTOR (7 downto 0));
end dec3x8;

architecture Behavioral of dec3x8 is

begin
   process (enc)
      begin
         case enc is
            when "000" => dec <= "00000001";
            when "001" => dec <= "00000010";
            when "010" => dec <= "00000100";
            when "011" => dec <= "00001000";
            when "100" => dec <= "00010000";
            when "101" => dec <= "00100000";
            when "110" => dec <= "01000000";
            when "111" => dec <= "10000000";
            when others => dec <= "00000000";
         end case;
   end process;

end Behavioral;
```

και το αντίστοιχο TestBench:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dec3x8_tb is
end dec3x8_tb;

architecture Bench of dec3x8_tb is
   component dec3x8 is
   port (
         enc: in std_logic_vector(2 downto 0);
         dec: out std_logic_vector(7 downto 0));
   end component;
signal enc: std_logic_vector(2 downto 0) := (others => '0');
signal dec: std_logic_vector(7 downto 0);

constant delay : time := 5 ns;
```

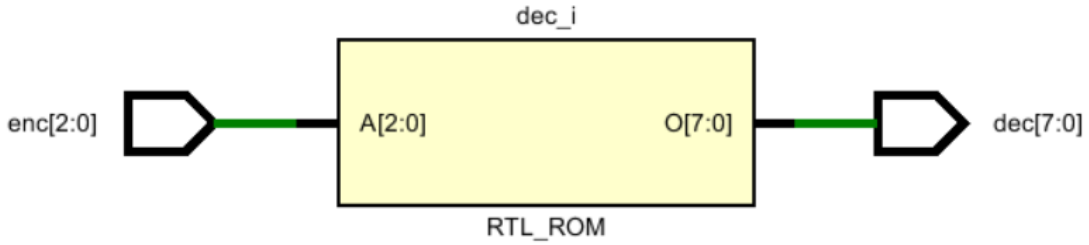
```
begin
   uut : dec3x8
      port map (
               enc => enc,
               dec => dec);
stimulus : process

   begin
      enc <= (others => '0');
      wait for delay;

      enc <= "000";
      wait for delay;
      enc <= "001";
      wait for delay;
      enc <= "010";
      wait for delay;
      enc <= "011";
      wait for delay;
      enc <= "100";
      wait for delay;
      enc <= "101";
      wait for delay;
      enc <= "110";
      wait for delay;
      enc <= "111";
      wait for delay;
end process;
end Bench;
```

Και έπειτα τρέχοντας την προσομοίωση(run simulation) παίρνουμε το παρακάτω:



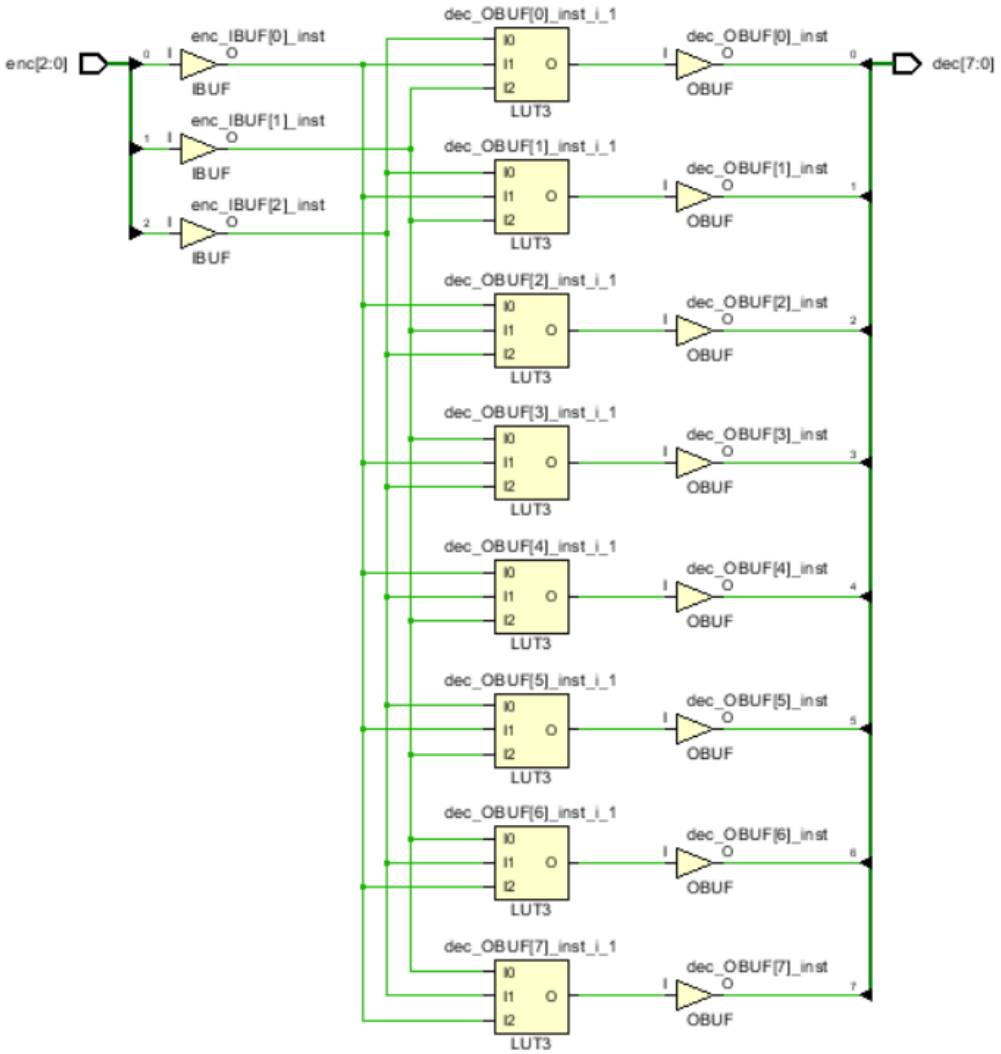
Όπου βλέπουμε ότι λειτουργεί σωστά και ότι είναι ίδια με την προσομοίωση σε αρχιτεκτονική dataflow.

Το **RTL schematic** του dec3x8 (behavioral) φαίνεται παρακάτω:



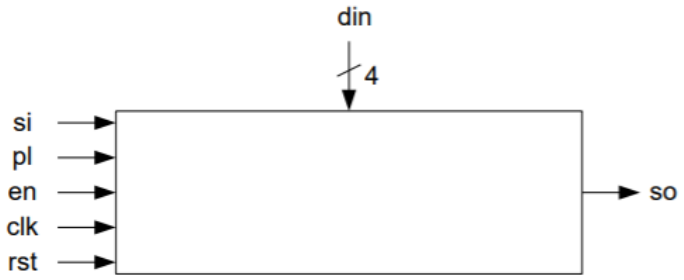
Όπου παρατηρούμε και πάλι ότι το RTL schematic είναι ίδιο και για τις δύο αρχιτεκτονικές. Αυτό συμβαίνει παρά το γεγονός ότι χρησιμοποιούμε διαφορετικές αρχιτεκτονικές, περιγράφουμε το ίδιο κύκλωμα.

Από το **run synthesis,** το σχηματικό σύνθεσης φαίνεται τα παρακάτω:



Το οποίο μετατρέπει τον RTL κώδικα στο netlist. Δηλαδή είναι η διαδικασία της μετατροπής των λογικών πράξεων υψηλού επιπέδου FPGA σε πύλες.

Θέμα Β.2: Καταχωρητής ολίσθησης των 4 bits με παράλληλη φόρτωση



Ζητούμενα: Να περιγραφεί η οντότητα του καταχωρητή ολίσθησης με μια επιπλέον είσοδο (std_logic) η οποία θα επιλέγει ανάμεσα σε αριστερή και δεξιά ολίσθηση.

Ο κώδικας που χρησιμοποιήθηκε για την περιγραφή του καταχωρητή ολίσθησης 4 bit με αριστερή και δεξιά ολίσθηση φαίνεται παρακάτω:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity shift_register is
   Port ( clk : in STD_LOGIC;
          rst : in STD_LOGIC;
          si : in STD_LOGIC;
          en : in STD_LOGIC;
          pl : in STD_LOGIC;
          left_right : in STD_LOGIC;
          din : in STD_LOGIC_VECTOR (3 downto 0);
          so : out STD_LOGIC);
end shift_register;

architecture Behavioral of shift_register is
   signal dff: std_logic_vector(3 downto 0);
   signal output: std_logic;
begin
   edge: process (clk,rst,left_right)
   begin
      if rst='0' then
         dff<=(others=>'0');
         output<=dff(0);
      elsif clk'event and clk='1' then
         if left_right='0' then --right shift
            if pl='1' then
               dff<=din;
            elsif en='1' then
               dff<=si&dff(3 downto 1);
            end if;
            output <= dff(0);
```

```
        else                --left shift
          if pl='1' then
            dff<=din;
          elsif en='1' then
            dff<=dff(2 downto 0)&si;
          end if;
          output <= dff(3);
        end if;
      end if;

  end process;
  so <= output;

end Behavioral;
```

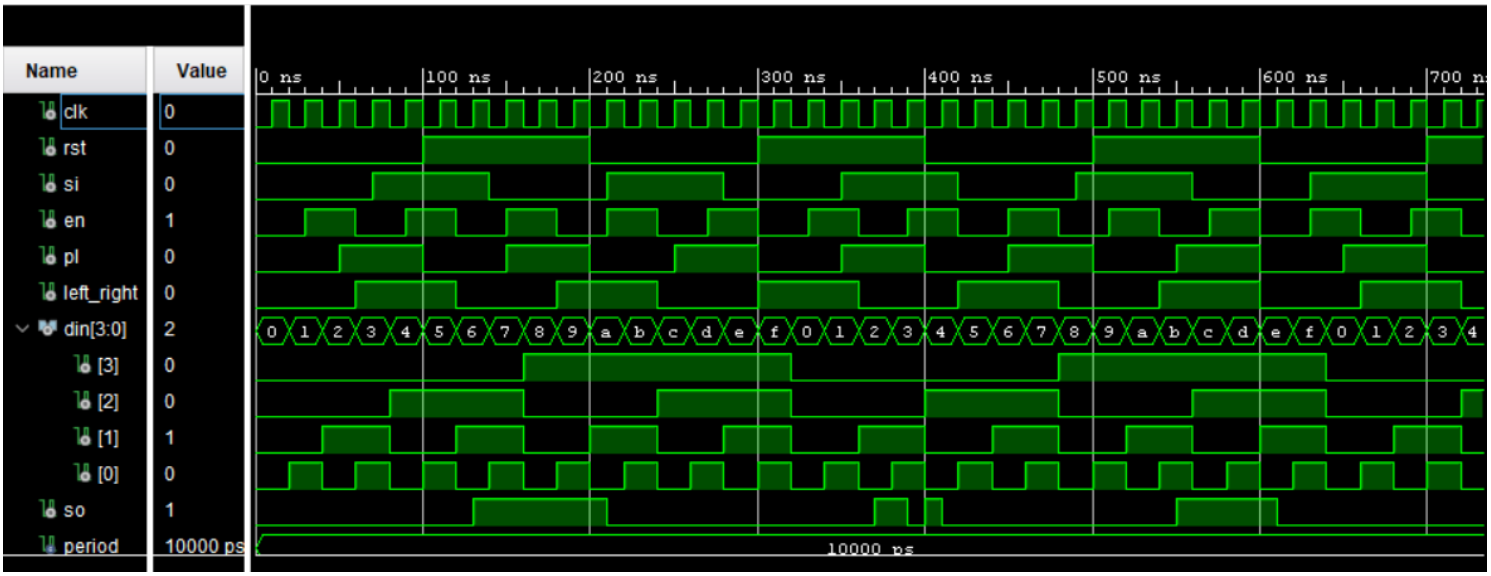
(Με μπλε έχουν σημειωθεί οι γραμμές κώδικα που έχουν προστεθεί από τον αρχικό κώδικα που μας έχει δοθεί έτσι ώστε ο καταχωρητής να εκτελεί και αριστερή ολίσθηση πέρα από δεξιά)

και το **TestBench** που χρησιμοποιήθηκε για την επαλήθευση της λειτουργίας του φαίνεται παρακάτω:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity shift_register_tb is

end shift_register_tb;

architecture Bench of shift_register_tb is

    component shift_register is
        port (
            clk : in STD_LOGIC;
            rst : in STD_LOGIC;
            si : in STD_LOGIC;
            en : in STD_LOGIC;
            pl : in STD_LOGIC;
            left_right : in STD_LOGIC;
            din : in STD_LOGIC_VECTOR (3 downto
0);
            so : out STD_LOGIC);
    end component;

signal clk : std_logic;
signal rst : std_logic :='0';
signal si : std_logic;
signal en : std_logic;
signal pl : std_logic;
signal left_right : std_logic;
signal din : std_logic_vector(3 downto 0) :=
(others => '0');
signal so : std_logic;

constant period : time := 10 ns;

begin
    uut : shift_register
        port map (
                clk => clk,
                rst => rst,
                si => si,
                en => en,
                pl => pl,
                left_right => left_right,
                din => din,
                so => so);
    generate_reset :process
    begin
        -- reset every 100ns
        rst <= '0';
        --din <= (others => '0');
        wait for period*10;
```

```
   rst <= '1';
   wait for period*10;
end process;

generate_left_or_right_shift : process
begin
   --left_right every 60ns
   left_right <= '0';
   wait for period*6;
   left_right <= '1';
   wait for period*6;
end process;

generate_si : process
 begin
      --si every 70ns
      si <= '0';
      wait for period*7;
      si <= '1';
      wait for period*7;
end process;

generate_enable : process
begin
   --enable shift every 30ns
   en <= '0';
   wait for period*3;
   en <= '1';
   wait for period*3;
end process;

generate_parallel : process
begin
   --parallel every 50ns
   pl <= '0';
   wait for period*5;
   pl <= '1';
   wait for period*5;
 end process;

 generate_din : process
 begin
   din <= "0000";
      wait for period*2;
      din <= "0001";
      wait for period*2;
      din <= "0010";
      wait for period*2;
      din <= "0011";
      wait for period*2;
      din <= "0100";
      wait for period*2;
      din <= "0101";
      wait for period*2;
      din <= "0110";
      wait for period*2;
      din <= "0111";
      wait for period*2;
      din <= "1000";
      wait for period*2;
      din <= "1001";
      wait for period*2;
      din <= "1010";
      wait for period*2;
      din <= "1011";
      wait for period*2;
      din <= "1100";
      wait for period*2;
      din <= "1101";
      wait for period*2;
      din <= "1110";
      wait for period*2;
      din <= "1111";
      wait for period*2;
end process;

   generate_clock : process
   begin
       --clock every period
         clk <= '0';
         wait for period;
         clk <= '1';
         wait for period;
    end process;

    end Bench;
```

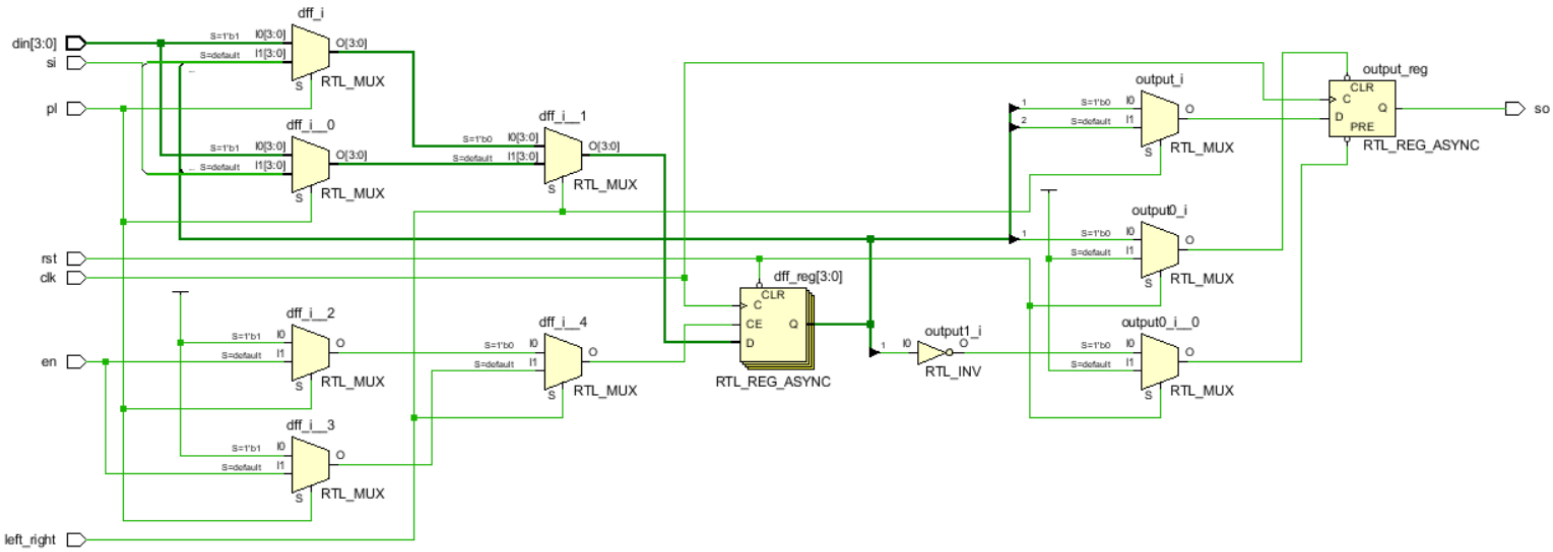
Όταν τρέξουμε το simulation παίρνουμε το παρακάτω:



Όπου βλέπουμε τα παρακάτω:

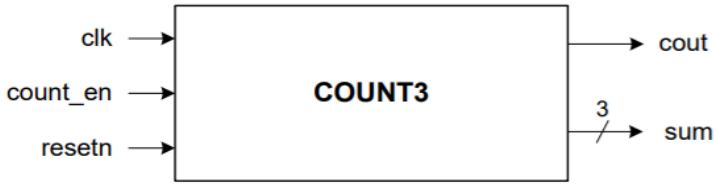
- Όταν rst=0, η έξοδος so=0
- Όταν clk στην θετική του ακμή, rst=1, left_right=1(left shift), pl=0, en=1 => έχουμε αριστερή ολίσθηση.
- Όταν clk στην θετική του ακμή, rst=1, left_right=0(right shift), pl=0, en=1 => έχουμε δεξιά ολίσθηση.
- Όταν clk στην θετική του ακμή, rst=1, left_right=0(right shift), pl=1 => έχουμε παράλληλη φόρτωση του lsb
- Όταν clk στην θετική του ακμή, rst=1, left_right=1(right shift), pl=1 => έχουμε παράλληλη φόρτωση του msb

Στο **RTL schematic** παίρνουμε το παρακάτω:



Όπου βλέπουμε τις λογικές πράξεις και τους καταχωρητές του κυκλώματος.

Θέμα Β.3: Μετρητής 3 bit με είσοδο ενεργοποίησης και κρατούμενο εξόδου



Ζητούμενα:

1. Βασιζόμενοι στην περιγραφή του μετρητή των 3 bits, να περιγράψετε έναν μετρητή up/down των 3 bits.

   Ο **κώδικάς** για την περιγραφή του μετρητή up/down των 3 Bits παρουσιάζεται παρακάτω:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity counter3_nolimit is
   Port ( clk : in STD_LOGIC;
          resetn : in STD_LOGIC;
          count_en : in STD_LOGIC;
          sum : out STD_LOGIC_VECTOR (2 downto 0);
          cout : out STD_LOGIC;
          up_down : in STD_LOGIC);
end counter3_nolimit;

architecture Behavioral of counter3_nolimit is
   signal count: std_logic_vector(2 downto 0);
   signal output: std_logic;
begin
   process(clk, resetn, up_down)
   begin
   if resetn='0' then -- Κώδικας για την περίπτωση του reset (ενεργό χαμηλά)
           count <= (others=>'0');
           output<= '0';
   elsif clk'event and clk='1' then
      case up_down is
         when '1' => --up counting
         if count_en='1' then
            count<=count+1;
          if count=7 then
           output<='1';
          else
           output<='0';
         end if;
         end if;
```

```
            when others => --down counting
            if count_en='1' then
               count<=count-1;
                if count=0 then
                  output<='1';
                 else
                  output<='0';
               end if;
            end if;
          end case;
        end if;
        end process;

        sum <= count;
        cout<=output;

    end Behavioral;
```

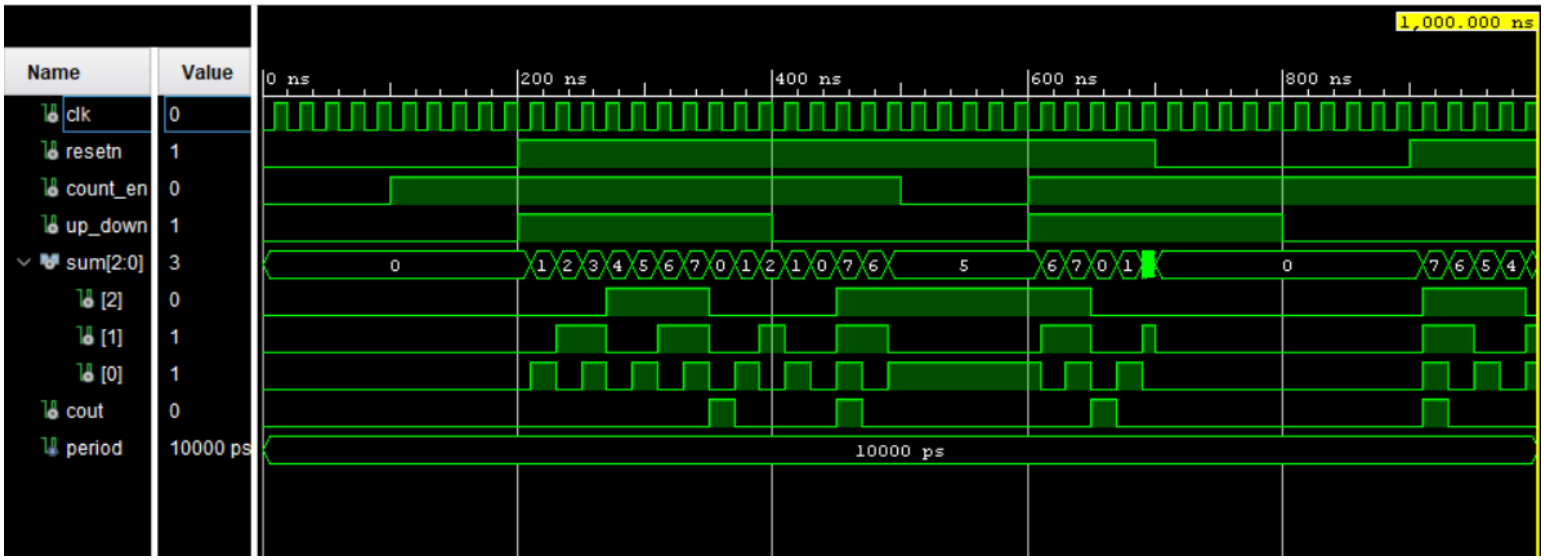
(Με μπλε σημειωμένες οι αλλαγές που κάναμε στον αρχικό κώδικα έτσι ώστε να γίνεται και πάνω αλλά και κάτω μέτρηση)

και το **TestBench** που φτιάξαμε φαίνεται παρακάτω:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity counter3_nolimit_tb is
end counter3_nolimit_tb;

architecture Bench of counter3_nolimit_tb is
component counter3_nolimit is
      port(
         clk : in std_logic;
         resetn : in std_logic;
         count_en : in std_logic;
         up_down : in std_logic;
         sum : out std_logic_vector(2 downto 0);
         cout : out std_logic);
   end component;

signal clk : std_logic;
signal resetn : std_logic;
signal count_en : std_logic;
signal up_down : std_logic;
signal sum : std_logic_vector(2 downto 0);
signal cout : std_logic;

constant period : time := 10ns;
```

```
begin
uut : counter3_nolimit
      port map (
            clk => clk,
            resetn => resetn,
            count_en => count_en,
            up_down => up_down,
            sum => sum,
            cout => cout);

generate_reset :process
   begin
      resetn <= '0';
      wait for period*20;
      resetn <= '1';
      wait for period*50;
   end process;


   generate_count_en : process
    begin
        count_en <= '0';
        wait for period*10;
        count_en <= '1';
        wait for period*40;
   end process;

generate_clock : process
      begin
         --clock every period
           clk <= '0';
           wait for period;
           clk <= '1';
           wait for period;
       end process;

generate_up_down : process
          begin
              --si every 200ns
              up_down <= '0';
              wait for period*20;
              up_down <= '1';
              wait for period*20;
          end process;
end Bench;
```

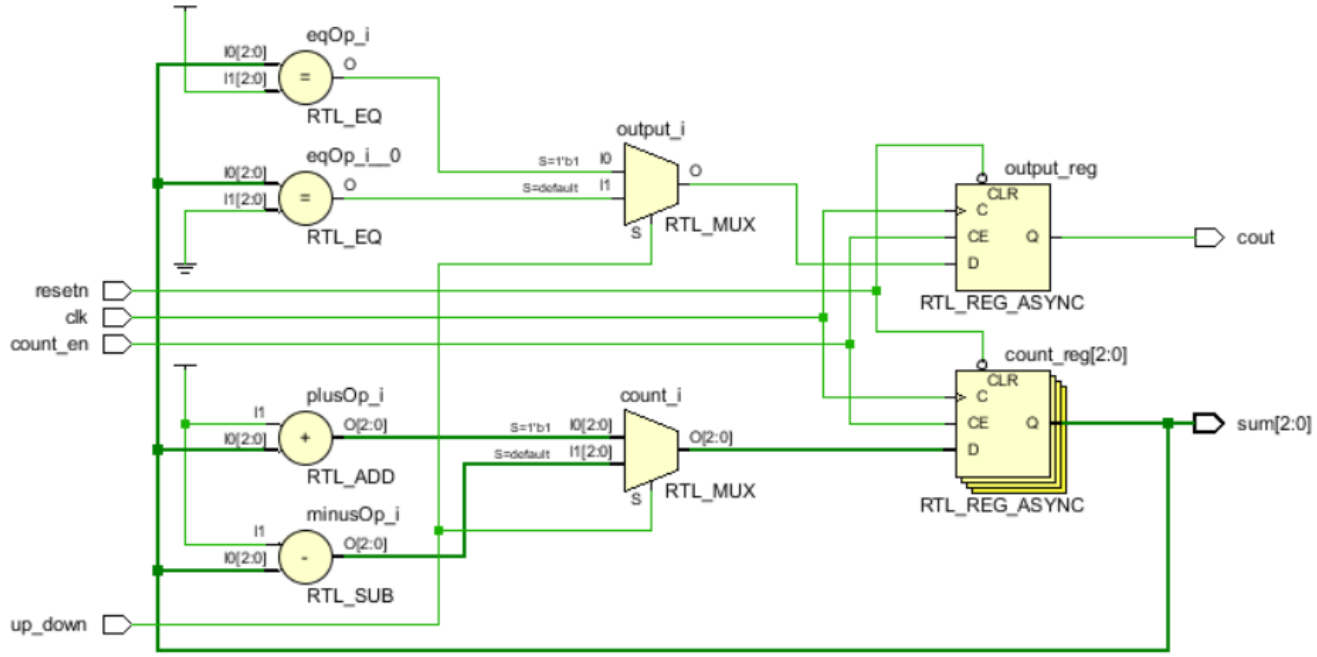
Και από το **run simulation** παίρνουμε τα παρακάτω:



Όπου παρατηρούμε ότι:

- Όταν resetn=0 => sum,cout=0
- Όταν resetn=1, count_en=1, up_down=1 => Μέτρηση πάνω
- Όταν resetn=1, count_en=1, up_down=0 => Μέτρηση κάτω
- Όταν resetn=1, count_en=0 => Σταματάει η μέτρηση εκεί που ήταν

Και στο **RTL schematic** λαμβάνουμε τα παρακάτω:



2. Βασιζόμενοι στην περιγραφή του μετρητή των 3 bits να περιγράψετε έναν up counter των 3 bits με παράλληλη είσοδο modulo (όριο μέτρησης) των 3 bits.

Ο κώδικας για τον μετρητη πάνω με όριο, είναι ο παρακάτω:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity counter3_limit is
   Port ( clk : in STD_LOGIC;
          resetn : in STD_LOGIC;
          count_en : in STD_LOGIC;
          sum : out STD_LOGIC_VECTOR (2 downto 0);
          cout : out STD_LOGIC);
end counter3_limit;

architecture Behavioral of counter3_limit is
signal count : std_logic_vector(2 downto 0);
signal modulo : std_logic_vector(2 downto 0);

begin
   process(clk, resetn)
   begin
   modulo <= std_logic_vector(to_unsigned(5,3)); --Ορίζουμε το όριο ίσο με 5 για να ελέγξουμε
την ορθή λειτουργία του στο simulation
      if resetn='0' then  -- Ασύγχρονος μηδενισμός
         count <= (others=>'0');
      elsif clk'event and clk='1' then
         if count_en = '1' then-- Μέτρηση μόνο αν count_en=’1’
            if count/=modulo then  -- Αυξάνουμε το μετρητή μόνο αν  δεν είναι modulo
                count <= count+1;
            else   -- Αλλιώς τον μηδενίζουμε
                 count<=(others=>'0');
            end if;
         end if;
      end if;
end process;
sum<= count;
cout <= '1' when count=modulo and count_en='1' else '0';

end Behavioral;
```

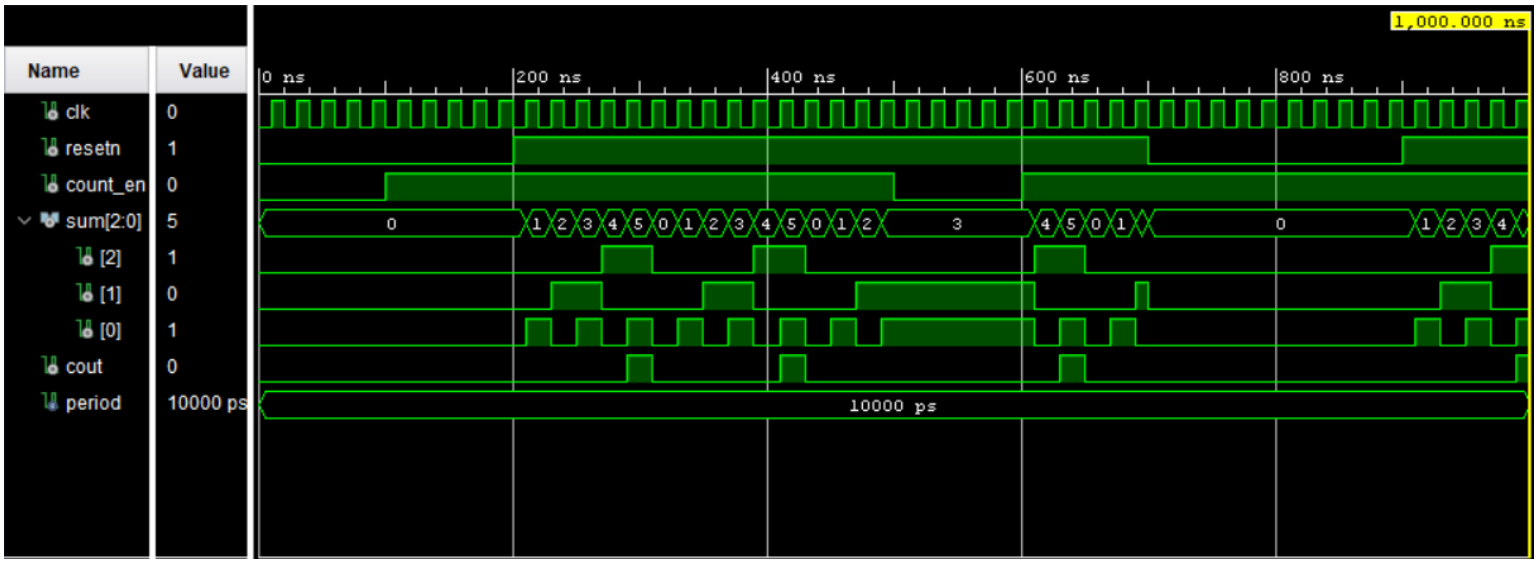
(Με μπλε σημειωμένες οι γραμμές που αλλάξαμε ή προσθέσαμε στον αρχικό κώδικα)
Και το αντίστοιχο **TestBench** που φτιάξαμε είναι:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity counter3_limit_tb is
end counter3_limit_tb;

architecture Bench of counter3_limit_tb is
component counter3_limit is
      port(
         clk : in std_logic;
         resetn : in std_logic;
         count_en : in std_logic;
         sum : out std_logic_vector(2 downto 0);
         cout : out std_logic);
   end component;

signal clk : std_logic;
signal resetn : std_logic;
signal count_en : std_logic;
signal sum : std_logic_vector(2 downto 0);
signal cout : std_logic;

constant period : time := 10ns;

begin
uut : counter3_limit
      port map (
            clk => clk,
            resetn => resetn,
            count_en => count_en,
            sum => sum,
            cout => cout);

generate_reset :process
   begin
      resetn <= '0';
      wait for period*20;
      resetn <= '1';
      wait for period*50;
   end process;

generate_count_en : process
    begin
        count_en <= '0';
        wait for period*10;
        count_en <= '1';
        wait for period*40;
   end process;

generate_clock : process
```

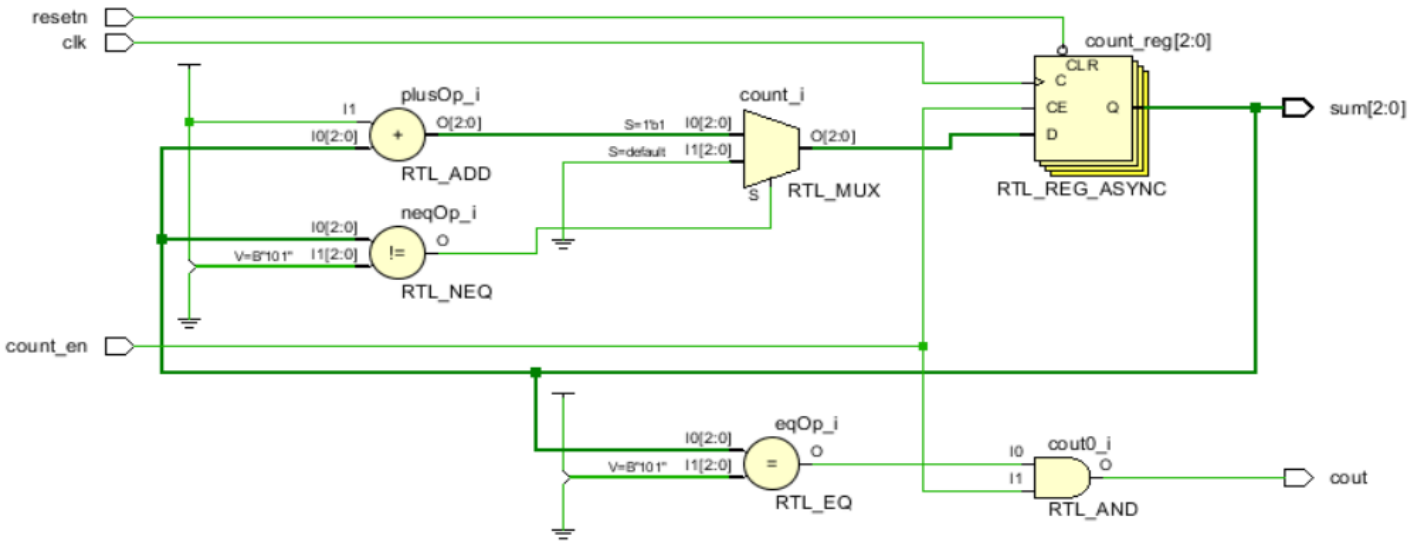
```
        begin --clock every period
            clk <= '0';
            wait for period;
            clk <= '1';
            wait for period;
        end process;
end Bench;
```

Το **run simulation** του μετρητή με όριο(Modulo=ίσο με 5 το θέσαμε εμείς όπως αναφέραμε και στα σχόλια στον κώδικα) φαίνεται παρακάτω:



Όπου βλέπουμε ότι λειτουργεί σωστά με όριο(Modulo)=5.'

Το **RTL schematic** φαίνεται παρακάτω:



Όπου βλέπουμε ότι διαφέρει με τον μετρητή πάνω κάτω έχοντας μια παραπάνω λογική πράξη που ελέγχει την ανισότητα με το modulo.

3. Ελέγξτε την ορθότητα λειτουργίας του κυκλώματος που δίνει ο synthesizer για τα ζητούμενα 1 και 2 και αν θα μπορούσε να σχεδιαστεί “με το χέρι” σε απλούστερη μορφή.

Στα σημεία που επισημαίνονται πιο κάτω παρατηρούμε ότι υπάρχουν περιττοί ακροδέκτες οι οποίοι συνδέονται με την γείωση και την τροφοδοσία ταυτόχρονα, κάτι το οποίο δεν εξυπηρετεί σε κάτι και θα μπορούσε να παραληφθεί. Επομένως, το κύκλωμα θα μπορούσε να σχεδιαστεί καλύτερα “με το χέρι”.