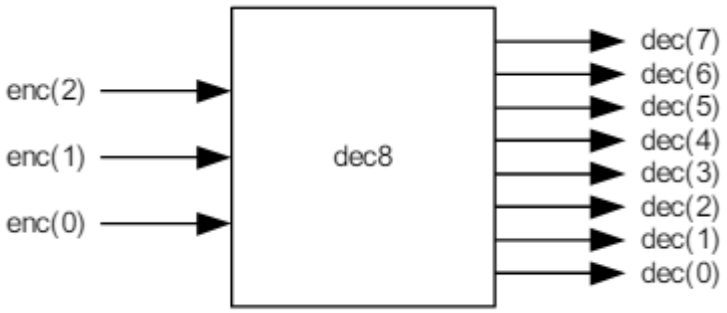
Εθνικό Μετσόβιο Πολυτεχνείο, ΣΗΜΜΥ
Ψηφιακά Συστήματα VLSI

1η Εργαστηριακή Άσκηση

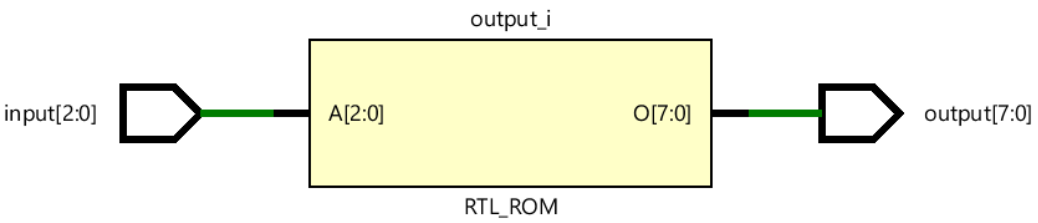
Ομάδα 19
Παναγιώτης Μπέλσης, ΑΜ: 03120874
Θεοδώρα Εξάρχου, ΑΜ : 03120865

Θέμα Α.2: Δυαδικός αποκωδικοποιητής 3 σε 8

Ζητούμενο: Να δοθεί η περιγραφή της οντότητας του δυαδικού αποκωδικοποιητή 3 σε 8 και της αρχιτεκτονικής του σε dataflow και behavioral VHDL. Κατόπιν να γίνει προσομοίωση και στις δύο αρχιτεκτονικές



Το RTL σχηματικό του δυαδικού αποκωδικοποιητή 3 σε 8 είναι:



Ο κώδικας VHDL για την dataflow αρχιτεκτονική:

```
entity decoder3_8 is
    port (
        input : in std_logic_vector(2 downto 0);
        output : out std_logic_vector(7 downto 0)
    );
end entity decoder3_8;

architecture Dataflow of decoder3_8 is

    -- Dataflow implementation of the decoder using with select

    begin
        with input select
            output <= "00000001" when "000",
                      "00000010" when "001",
                      "00000100" when "010",
                      "00001000" when "011",
                      "00010000" when "100",
                      "00100000" when "101",
                      "01000000" when "110",
                      "10000000" when "111",
                      "00000000" when others;  -- Default output


end Dataflow;
```

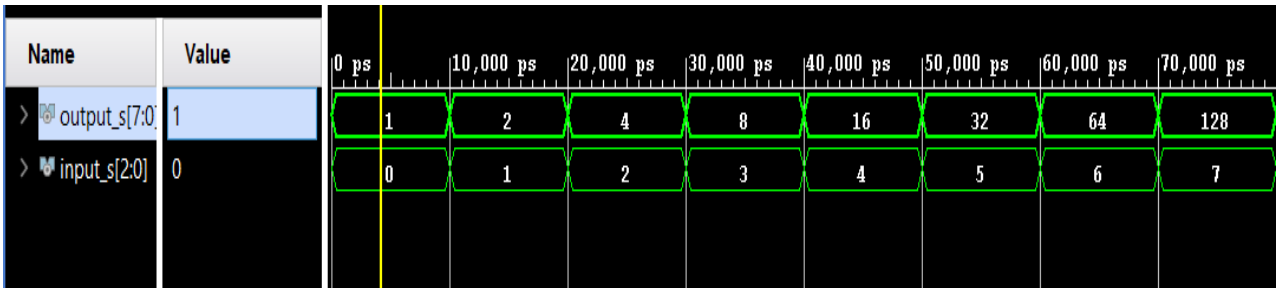
Ο κώδικας VHDL για την behavioral αρχιτεκτονική:

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

entity decoder3_8 is
    port (
        input : in std_logic_vector(2 downto 0);
        output : out std_logic_vector(7 downto 0)
    );
end entity decoder3_8;

architecture behavioral of decoder3_8 is
begin
    process(input)
    begin
        case input is
            when "000" =>
                output <= "00000001";
            when "001" =>
                output <= "00000010";
            when "010" =>
                output <= "00000100";
            when "011" =>
                output <= "00001000";
            when "100" =>
                output <= "00010000";
            when "101" =>
                output <= "00100000";
            when "110" =>
                output <= "01000000";
            when "111" =>
                output <= "10000000";
            when others =>
                output <= (others => '0');
        end case;
    end process;
end architecture behavioral;
```

Ο κώδικας VHDL για το testbench της dataflow /behavioral αρχιτεκτονικής:

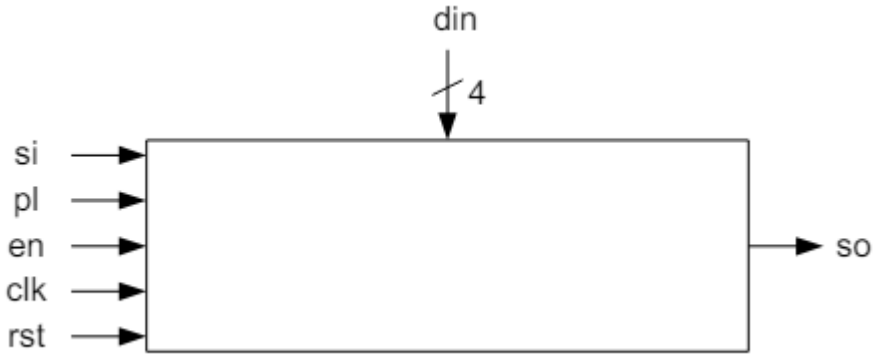
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decoder3_8_tb is
--  Port ( );
end decoder3_8_tb;

architecture testbench of decoder3_8_tb is
  component decoder3_8
    port (
      input: in std_logic_vector(2 downto 0); --inputs
      output: out std_logic_vector(7 downto 0) --outputs
    );
  end component;

  signal input_s : std_logic_vector(2 downto 0);  -- signals
  signal output_s : std_logic_vector(7 downto 0);  -- output signals

begin
  uut: decoder3_8
  port map (input => input_s, output => output_s);

  simulation : process
  begin
    input_s <= "000";
    wait for 10 ns;
    input_s <= "001";
    wait for 10 ns;
    input_s <= "010";
    wait for 10 ns;
    input_s <= "011";
    wait for 10 ns;
    input_s <= "100";
    wait for 10 ns;
    input_s <= "101";
    wait for 10 ns;
    input_s <= "110";
    wait for 10 ns;
    input_s <= "111";
    wait for 10 ns;
  end process;

end testbench;
```

Η προσομοίωση που επιβεβαιώνει την ορθή λειτουργία του κυκλώματός μας:

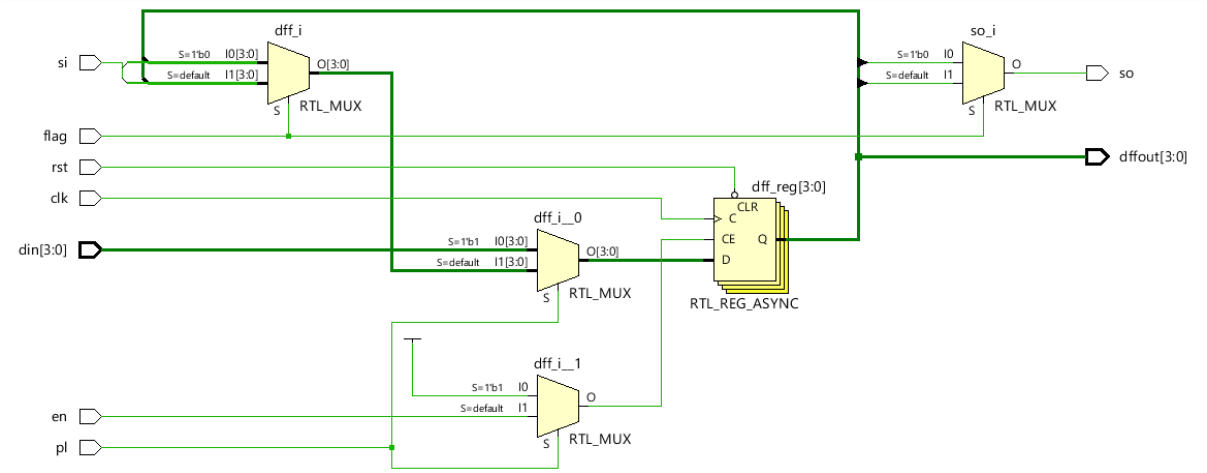


Θέμα Β.2: Καταχωρητής ολίσθησης των 4 bits με παράλληλη φόρτωση

Ζητούμενο: Να περιγραφεί η οντότητα του καταχωρητή ολίσθησης με μια επιπλέον είσοδο (std_logic) η οποία θα επιλέγει ανάμεσα σε αριστερή και δεξιά ολίσθηση. Υπενθυμίζεται ότι στην αριστερή ολίσθηση η έξοδος είναι το MSB του καταχωρητή και η σειριακή είσοδος γίνεται από το LSB. Για την περιγραφή την οποία θα φτιάξετε να ελέγξετε και το κύκλωμα που προκύπτει από τον synthesizer, παρατηρώντας τις διαφορές που έχει από το κύκλωμα που δίνεται στο σχήμα 2.8



Το RTL σχηματικό του καταχωρητή που περιγραφεται παραπάνω:



Ο κώδικας VHDL για την behavioral αρχιτεκτονική:

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

entity lab01_ex02 is
port (
clk,rst,si,en,pl,flag: in std_logic;
```

```
din: in std_logic_vector(3 downto 0);
so: out std_logic;
dffout: out std_logic_vector(3 downto 0));
end lab01_ex02;

architecture rtl of lab01_ex02 is
signal dff: std_logic_vector(3 downto 0);
--signal temp: std_logic;


begin
edge: process (clk,rst)

begin


if rst='0' then
dff<=(others=>'0');

elsif clk'event and clk='1' then
if pl='1' then
dff<=din;

elsif en='1' then
  if flag = '0' then
  dff<=si&dff(3 downto 1) ;

  elsif flag ='1' then
  dff<=dff(2 downto 0)&si;


end if;
end if;
end if;

end process;

dffout <= dff;
so <= dff(0) WHEN flag = '0' ELSE dff(3) WHEN flag = '1';
---end process;
end rtl;
```

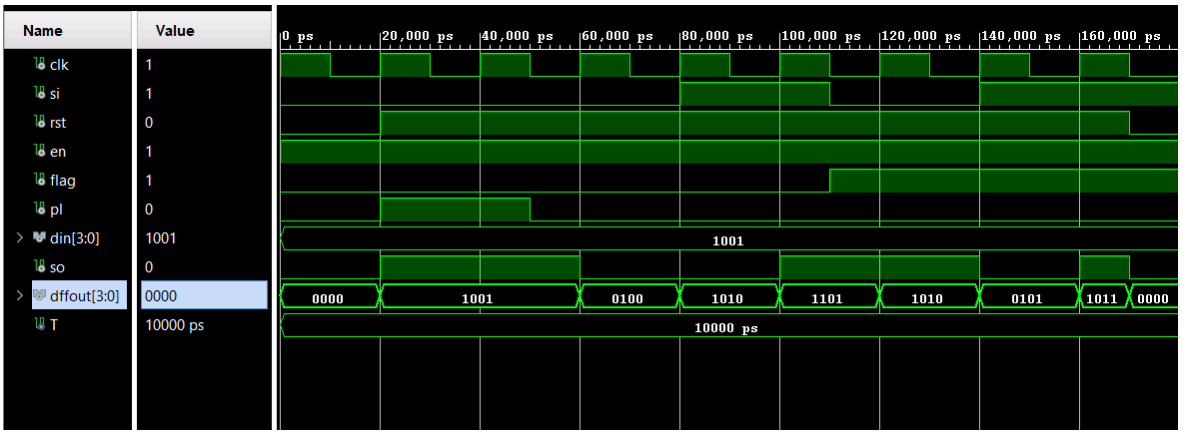
Ο κώδικας VHDL για το testbench της behavioral αρχιτεκτονικής:

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

entity lab01_ex02_tb is
end lab01_ex02_tb;

architecture Behavioral of lab01_ex02_tb is

component lab01_ex02
port(
clk,rst,si,en,pl,flag: in std_logic;
din: in std_logic_vector(3 downto 0);
so: out std_logic;
dffout: out std_logic_vector(3 downto 0));
end component;

signal clk : STD_LOGIC := '0';
signal si : STD_LOGIC := '0';
signal rst : STD_LOGIC := '0';
signal en : STD_LOGIC := '0';
signal flag : STD_LOGIC := '0';


signal pl: STD_LOGIC := '0';

signal din : std_logic_vector(3 downto 0) := "1001";

signal so :  std_logic ;
signal dffout: std_logic_vector(3 downto 0);

constant T : time := 10 ns ; -- clock period

begin
test: lab01_ex02

 port map(
 clk => clk,
 rst => rst,
 si => si,
 en=>en,
 pl =>pl,
 din =>din,
 flag =>flag,
```

```
so =>so,
dffout => dffout
);

stimulus:
process

begin

-- reset off
en <= '1';
rst <= '0';
for i in 0 to 1 loop
clk <= not clk;
wait for T;
end loop;

-- parallel load
 rst <= '1';
 pl <= '1';
 for i in 0 to 2 loop
 clk <= not clk;
 wait for T;
 end loop;

    -- enable on and si=0
   rst <= '1';
   pl <= '0';
   en <= '1';
   si <= '0';
   for i in 0 to 2 loop
   clk <= not clk;
   wait for T;
   end loop;

     -- enable on and si=1
     rst <= '1';
     pl <= '0';
     en <= '1';
     si <= '1';
     for i in 0 to 2 loop
     clk <= not clk;
     wait for T;
     end loop;
```

```
    ---- flag = 1
    flag <= '1';

      -- enable on and si=0
      rst <= '1';
      pl <= '0';
      en <= '1';
      si <= '0';
      for i in 0 to 2 loop
      clk <= not clk;
      wait for T;
      end loop;


        -- enable on and si=1
        rst <= '1';
        pl <= '0';
        en <= '1';
        si <= '1';
        for i in 0 to 2 loop
        clk <= not clk;
        wait for T;
        end loop;

          -- reset = 0
                  rst <= '0';
                  for i in 0 to 2 loop
                  clk <= not clk;
                  wait for T;
                  end loop;
wait;
end process;

end  Behavioral;
```

Η προσομοίωση που επιβεβαιώνει την ορθή λειτουργία του κυκλώματός μας:



Ο καταχωρητής ολίσθησης) είναι ένα κύκλωμα το οποίο δέχεται μια παράλληλη είσοδο din (Data in) η οποία φορτώνεται μέσω του σύγχρονου σήματος ενεργοποίησης παράλληλης φόρτωσης pl (Parallel Load). Η ενεργοποίηση της ολίσθησης γίνεται μέσω του σήματος en (Enable). Ο καταχωρητής έχει και μια σειριακή είσοδο si (Serial Input), καθώς και μια σειριακή έξοδο so (Serial Output). Επιπλέον, έχει μια είσοδο **flag** η οποία καθορίζει την κατεύθυνση της ολίσθησης.

Κατά την δεξιά ολίσθηση (flag = '0'), το περιεχόμενο του καταχωρητή ολισθαίνει κατά μια θέση προς το LSB (bit 0). Το LSB βγαίνει στην έξοδο so και η είσοδος si περνά στο MSB του καταχωρητή. Κατά την αριστερή ολίσθηση (flag = '1'), το περιεχόμενο του καταχωρητή ολισθαίνει κατά μια θέση προς το MSB (bit 3). Το MSB βγαίνει στην έξοδο so και η είσοδος si περνά στο LSB του καταχωρητή. Αυτές οι ολισθήσεις γίνονται σε κάθε θετικό παλμό του ρολογιού clk. Η ασύγχρονη είσοδος rst (reset) μηδενίζει τα flip-flops του καταχωρητή ολίσθησης.

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. Υπόδειξη: Χρησιμοποιήστε μια είσοδο επιλογής κατεύθυνσης: 1 για μέτρηση προς τα πάνω, 0 για μέτρηση προς τα κάτω και την εντολή case.

Ο κώδικας VHDL της behavioral αρχιτεκτονικής:

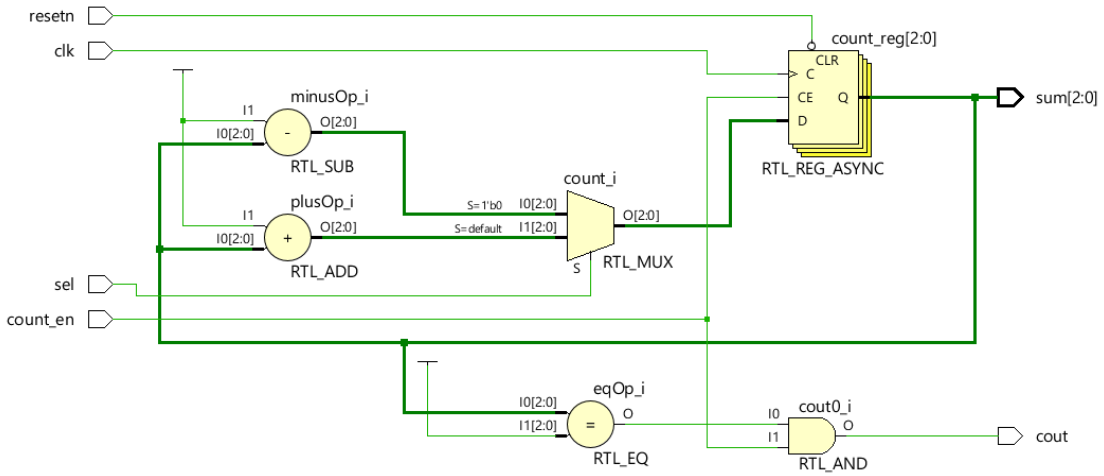
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

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

architecture Behavioral of counter_updown is
signal count: std_logic_vector(2 downto 0);
begin
    process(clk, resetn)
    begin
        if resetn='0' then
            count <= (others=>'0'); -- Code for reset
        elsif clk'event and clk='1' then --clock rising edge
            if count_en='1' then
                if sel='0' then  --down_count with zero
                    count<=count-1;
                elsif sel='1' then  --up_count with one
                    count<=count+1;
                else
                    count <= (others=>'0');
                end if;
            end if;
        end if;
    end process;
    sum <= count; -- Output signals
    cout <= '1' when count=7 and count_en='1' else '0';
end Behavioral;
```

Ο κώδικας VHDL για το testbench της behavioral αρχιτεκτονικής:

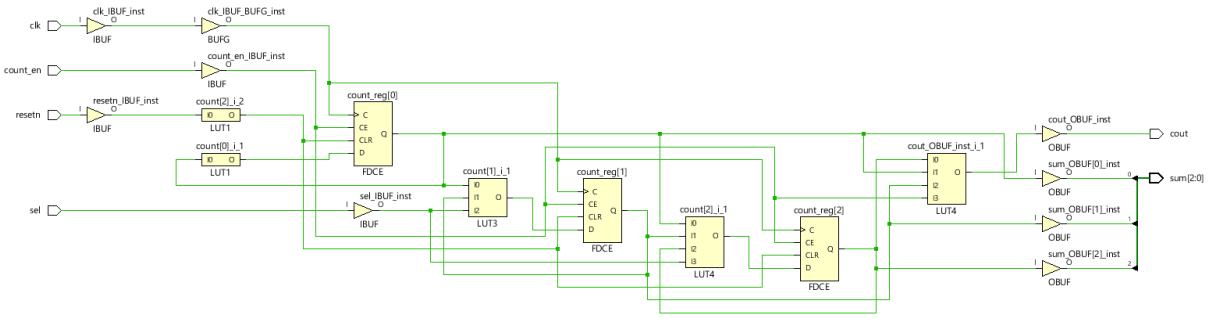
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity counter_updown_tb is
end counter_updown_tb;

architecture Behavioral of counter_updown_tb is
component counter_updown is
        port ( clk : in STD_LOGIC;
               count_en : in STD_LOGIC;
               resetn : in STD_LOGIC;
               sel : in STD_LOGIC;
               cout : out STD_LOGIC;
               sum : out STD_LOGIC_VECTOR (2 downto 0));
    end component;

    signal clk : STD_LOGIC := '0';
    signal count_en : STD_LOGIC := '0';
    signal resetn : STD_LOGIC := '0';
    signal sel : STD_LOGIC := '1';
    signal cout : STD_LOGIC;
    signal sum : STD_LOGIC_VECTOR (2 downto 0);

    constant CLOCK_PERIOD : time := 10 ns;

begin
    test : counter_updown
        port map ( clk => clk,
                   count_en => count_en,
                   resetn => resetn,
                   sel => sel,
                   cout => cout,
                   sum => sum
                   );

     stimulus : process
     begin
        --dummy start counter
         resetn<='1';
         count_en <= '1';
         clk <='1';
         sel <= '1';
         for i in 0 to 4 loop
```

```
            clk <= not clk;
            wait for CLOCK_PERIOD;
         end loop;

        --Test count_en=0
         count_en <= '0';
         for i in 0 to 9 loop
           clk <= not clk;
           wait for CLOCK_PERIOD;
         end loop;

        --Test count_up
        sel <= '1';
        count_en <= '1';
        for i in 0 to 17 loop
           clk <= not clk;
           wait for CLOCK_PERIOD;
        end loop;

        --Test count_en=0
         count_en <= '0';
         for i in 0 to 9 loop
           clk <= not clk;
           wait for CLOCK_PERIOD;
         end loop;

        --Test count_down
        count_en <= '1';
        sel <= '0';
        for i in 0 to 17 loop
           clk <= not clk;
           wait for CLOCK_PERIOD;
        end loop;

        --Test resetn=0
        resetn <= '0';
        for i in 0 to 9 loop
           clk <= not clk;
           wait for CLOCK_PERIOD;
        end loop;
        wait;
     end process;
end Behavioral;
```

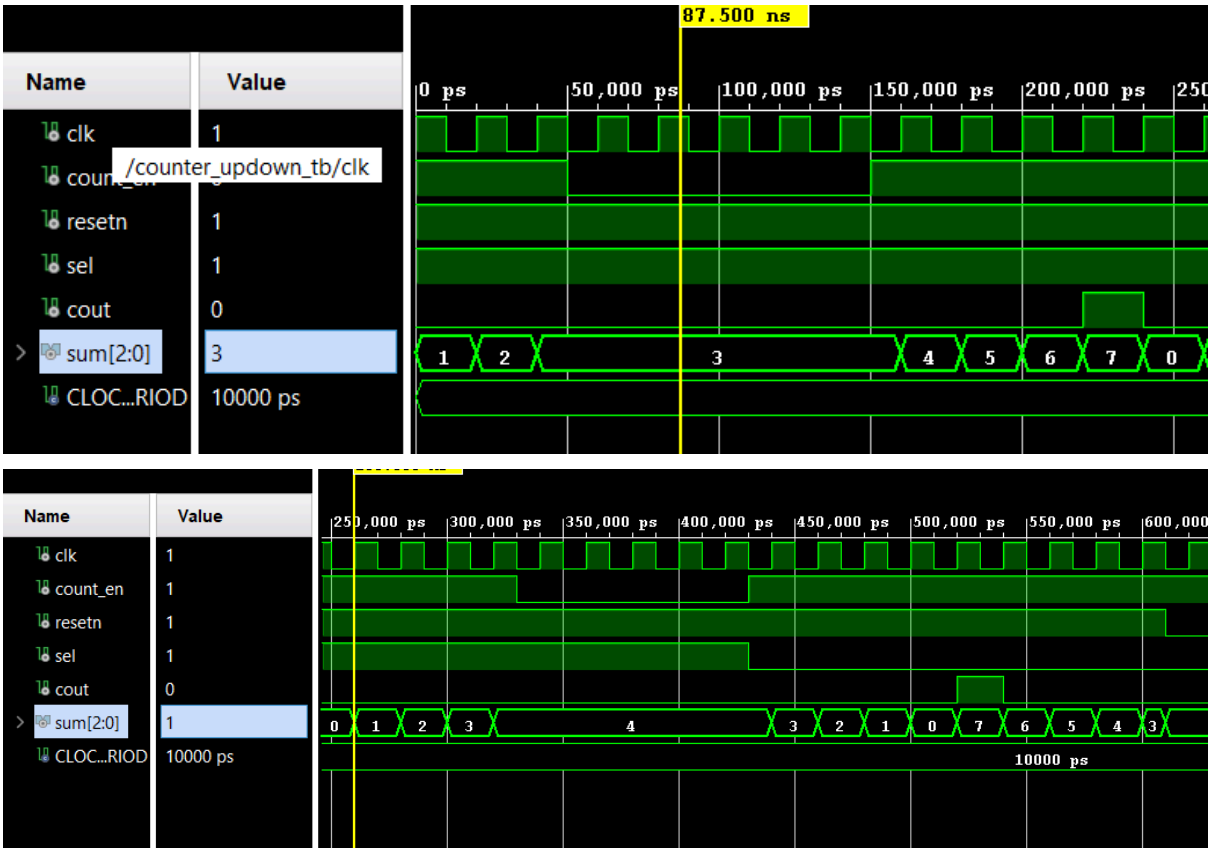
Το RTL σχηματικό του μετρητή :



Το Synthesis του παραπάνω μετρητή:



Μέρος της προσομοίωσης που επιβεβαιώνει την ορθή λειτουργία του κυκλώματός:



Ζητούμενα: 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. Υπόδειξη: Χρησιμοποιήστε μια είσοδο std_logic_vector την οποία θα συγκρίνετε με την τιμή του μετρητή.

Ο κώδικας VHDL της behavioral αρχιτεκτονικής:

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

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

```
architecture Behavioral of counter_up_limit is
signal count: std_logic_vector(2 downto 0);
begin
    process(clk, resetn)
    begin
        if resetn='0' then
            count <= (others=>'0'); -- Code for reset
        elsif clk'event and clk='1' then --clock rising edge
            if count_en='1' then  -- Count up when count_en=1
               if count/=parall then
                    count<=count+1;
               else  --set to zero when modulo input value is reached
                    count<=(others=>'0');
               end if;
            end if;
        end if;
    end process;
    sum <= count; -- Output signals
    cout <= '1' when count=7 and count_en='1' else '0';
end Behavioral;
```

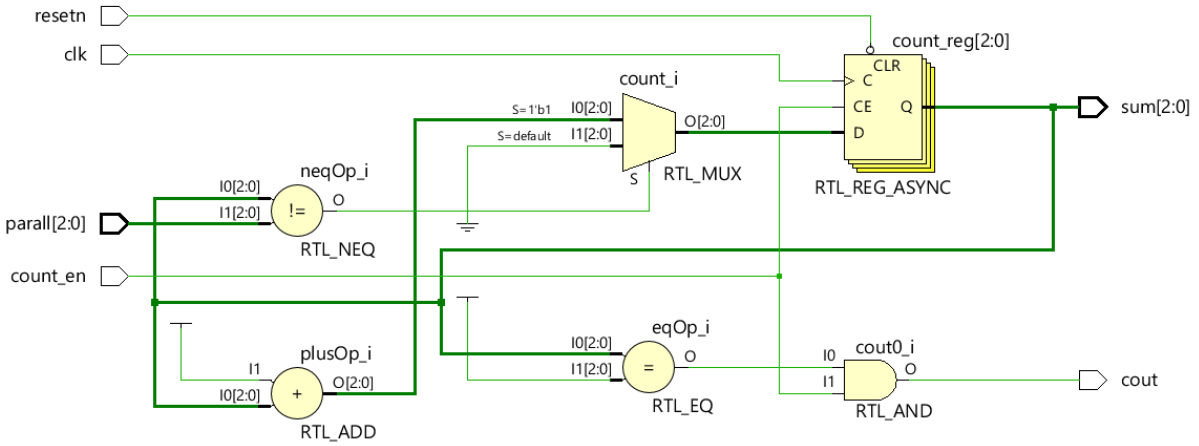
Ο κώδικας VHDL για το testbench της behavioral αρχιτεκτονικής:

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

entity counter_up_limit_tb is
end counter_up_limit_tb;

architecture Testbench of counter_up_limit_tb is
component counter_up_limit is
        port ( clk : in STD_LOGIC;
               count_en : in STD_LOGIC;
               resetn : in STD_LOGIC;
               parall : in STD_LOGIC_VECTOR(2 downto 0);
               cout : out STD_LOGIC;
               sum : out STD_LOGIC_VECTOR (2 downto 0));
    end component;
```

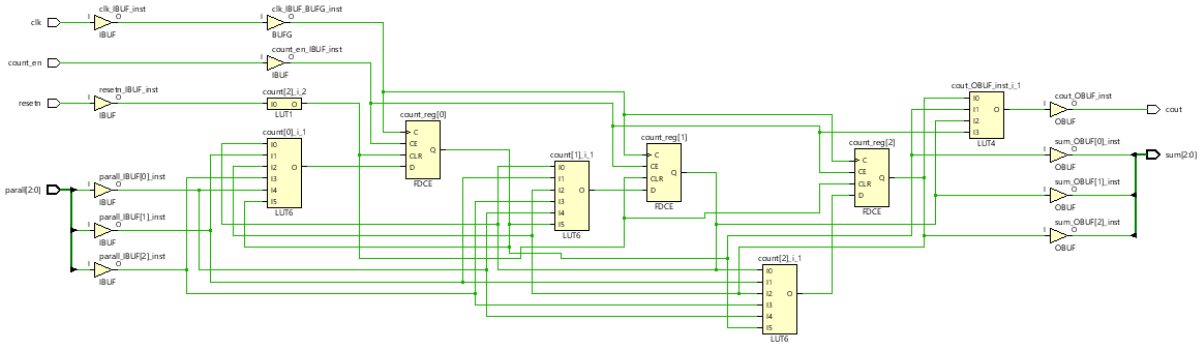
```
    signal clk : STD_LOGIC := '0';
    signal count_en : STD_LOGIC := '0';
    signal resetn : STD_LOGIC := '0';
    signal parall : STD_LOGIC_VECTOR (2 downto 0) := (others => '0');
    signal cout : STD_LOGIC;
    signal sum : STD_LOGIC_VECTOR (2 downto 0);
    constant CLOCK_PERIOD : time := 10 ns;
begin
test : counter_up_limit
        port map ( clk => clk,
                   count_en => count_en,
                   resetn => resetn,
                   parall => parall,
                   cout => cout,
                   sum => sum
                   );

     stimulus : process
     begin
        --Start dummy count
        count_en <= '1';
        resetn <= '1';
        parall <= "111";
        clk <='0';
        for i in 0 to 4 loop
          clk <= not clk;
          wait for CLOCK_PERIOD;
        end loop;

        --Test count_en = 0
        count_en <= '0';
        for i in 0 to 9 loop
          clk <= not clk;
          wait for CLOCK_PERIOD;
        end loop;

        count_en <= '1';
        --Test count up to 6
        for i in 0 to 17 loop
           clk <= not clk;
           wait for CLOCK_PERIOD;
        end loop;

        resetn <= '0';
        --Test reset='0'
```

```
        for i in 0 to 9 loop
            clk <= not clk;
            wait for CLOCK_PERIOD;
        end loop;
        wait;
    end process;
end Testbench;
```

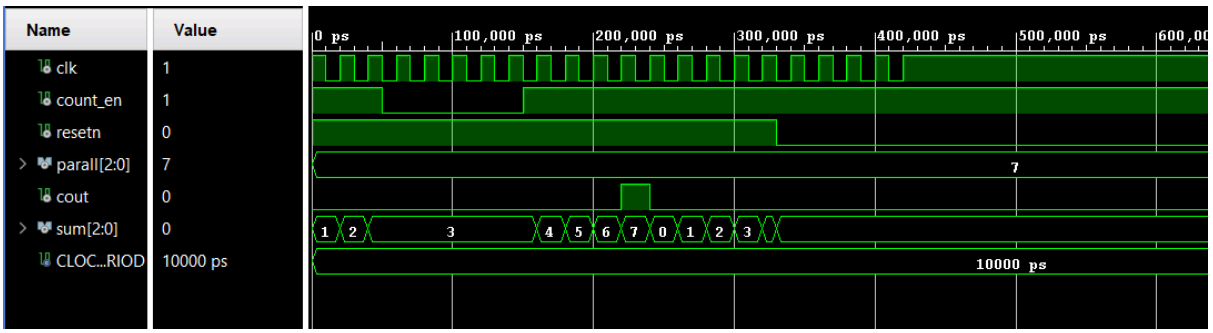
Το RTL σχηματικό του παραπάνω μετρητή:



Το Synthesis του παραπάνω μετρητή:



Μέρος της προσομοίωσης που επιβεβαιώνει την ορθή λειτουργία του κυκλώματός:



Ζητούμενα: 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.

Τα κυκλώματα που προκύπτουν από τον synthesizer για τα ζητούμενα 1 και 2 είναι ορθά και έχουν βελτιστοποιηθεί με τους ενσωματωμένους αλγόριθμους βελτιστοποίησης του Vivado, δεδομένου ότι χρησιμοποιήθηκε behavioral περιγραφή. Για να γίνει ακόμα πιο αποδοτική η περιγραφή του κυκλώματος μας θα έπρεπε να επιλέξουμε Structural περιγραφή, δηλαδή σχεδίαση με χρήση απευθείας λογικών πυλών ή ακόμα και transistor. Το Vivado στην behavioral περιγραφή χρησιμοποιεί κυρίως multiplexers κάτι που δεν είναι πάντα αποδοτικό.

Σχεδίαση απευθείας με λογικές πύλες και κάποια J-K Flip-Flop :