Ψηφιακά Συστήματα VLSI

8ο εξάμηνο, Ακαδημαϊκή περίοδος 2024-2025

1η Εργαστηριακή Άσκηση

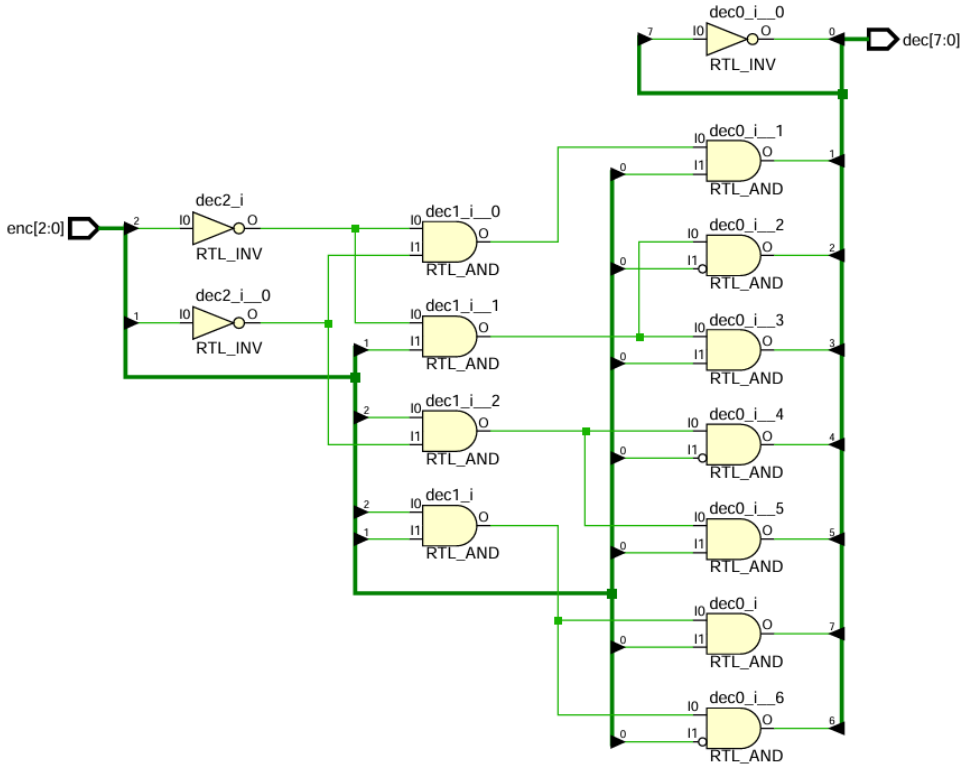
Δημήτρης Καμπανάκης: 03121012
Αγγελική Ζέρβα: 03121101

Θέμα Α.2: Δυαδικός αποκωδικοποιητής 3 σε 8

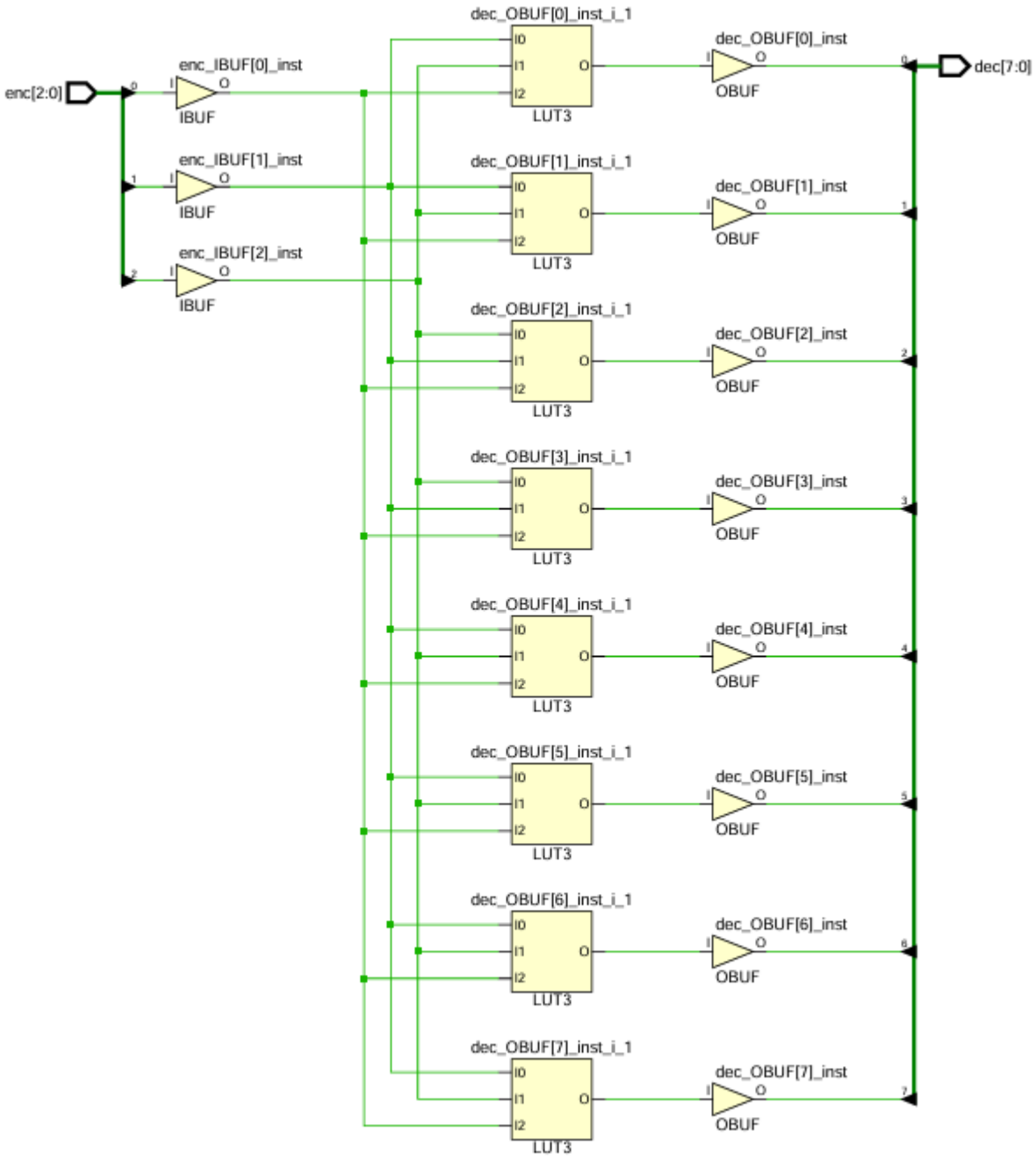
Ο κώδικας σε vhdl για την περιγραφή του αποκωδικοποιητή και της αρχιτεκτονικής του σε dataflow vhdl είναι ο εξής:

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

entity decoder_3_to_8_dataflow is
    port(
        enc : in std_logic_vector(3-1 downto 0);
        dec : out std_logic_vector(8-1 downto 0)
    );
end decoder_3_to_8_dataflow;

architecture dataflow_arch of decoder_3_to_8_dataflow is

begin
    dec(7) <= enc(2) and enc(1) and enc(0);
    dec(6) <= enc(2) and enc(1) and (not enc(0));
    dec(5) <= enc(2) and (not enc(1)) and enc(0);
    dec(4) <= enc(2) and (not enc(1)) and (not enc(0));
    dec(3) <= (not enc(2)) and enc(1) and enc(0);
    dec(2) <= (not enc(2)) and enc(1) and (not enc(0));
    dec(1) <= (not enc(2)) and (not enc(1)) and enc(0);
    dec(0) <=  (not enc(2)) and (not enc(1)) and (not enc(0));

end dataflow_arch;
```

Με τη χρήση του elaborate design παίρνουμε το εξής σχηματικό:



Με τη χρήση του synthesis design παίρνουμε το εξής σχηματικό:

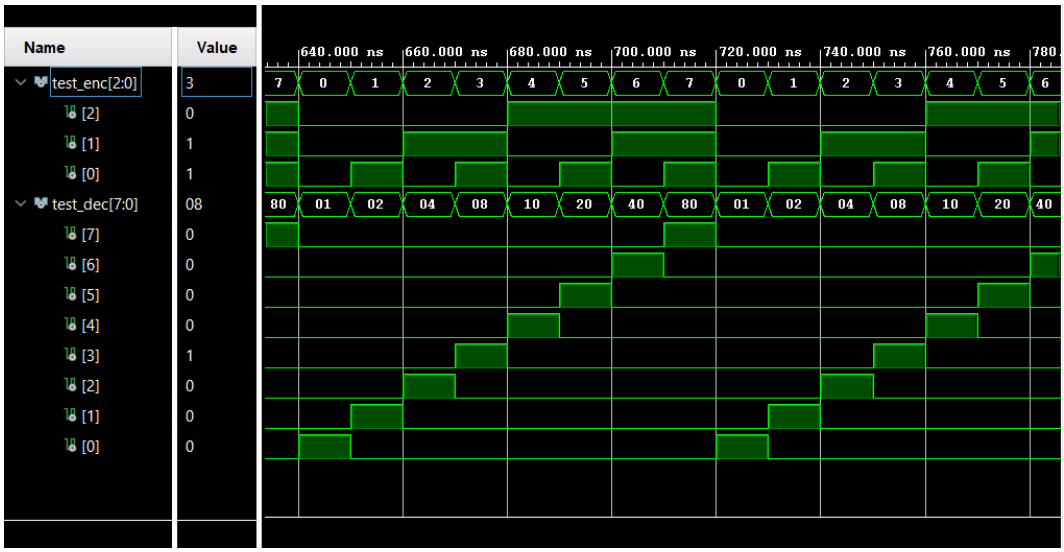


Για να μπορούμε να ελέγξουμε την ορθή λειτουργία του αποκωδικοποιητή παράγουμε όλα τα πιθανά σήματα εισόδου και παρατηρούμε τις εξόδους σε καθένα από αυτά:

```
    library IEEE;

use IEEE.std_logic_1164.all;

entity decoder_3_to_8_tb_dataflow is
end decoder_3_to_8_tb_dataflow;
```

```
architecture decoder_3_to_8_tb_dataflow_arch of decoder_3_to_8_tb_dataflow is

component decoder_3_to_8_dataflow

port(
   enc : in std_logic_vector(3-1 downto 0);
   dec : out std_logic_vector(8-1 downto 0)
);

end component;

-- stimulus signals
signal test_enc : std_logic_vector(3-1 downto 0);
signal test_dec : std_logic_vector(8-1 downto 0);

begin

uut: decoder_3_to_8_dataflow
   port map(
       enc => test_enc,
       dec => test_dec
   );

testing: process

begin
   test_enc <= "000";
   wait for 10 ns;

   test_enc <= "001";
   wait for 10 ns;

   test_enc <= "010";
   wait for 10 ns;

   test_enc <= "011";
   wait for 10 ns;

   test_enc <= "100";
   wait for 10 ns;

   test_enc <= "101";
   wait for 10 ns;

   test_enc <= "110";
   wait for 10 ns;

   test_enc <= "111";
   wait for 10 ns;

end process;

end decoder_3_to_8_tb_dataflow_arch;
```

Τα αποτελέσματα της προσομοίωσης φαίνονται παρακάτω:

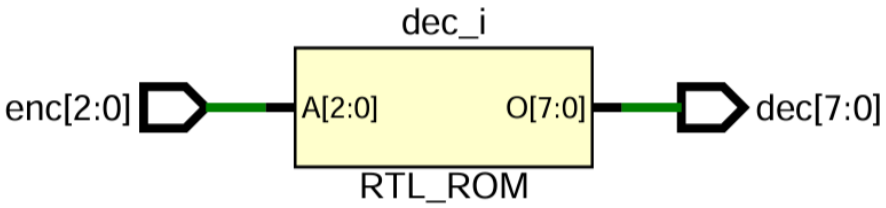


Παρατηρούμε την ορθή σχέση μεταξύ κάθε εισόδου και αντίστοιχης εξόδου.
Ο κώδικας σε vhdl για την περιγραφή του αποκωδικοποιητή και της αρχιτεκτονικής του σε behavioral vhdl είναι ο εξής:

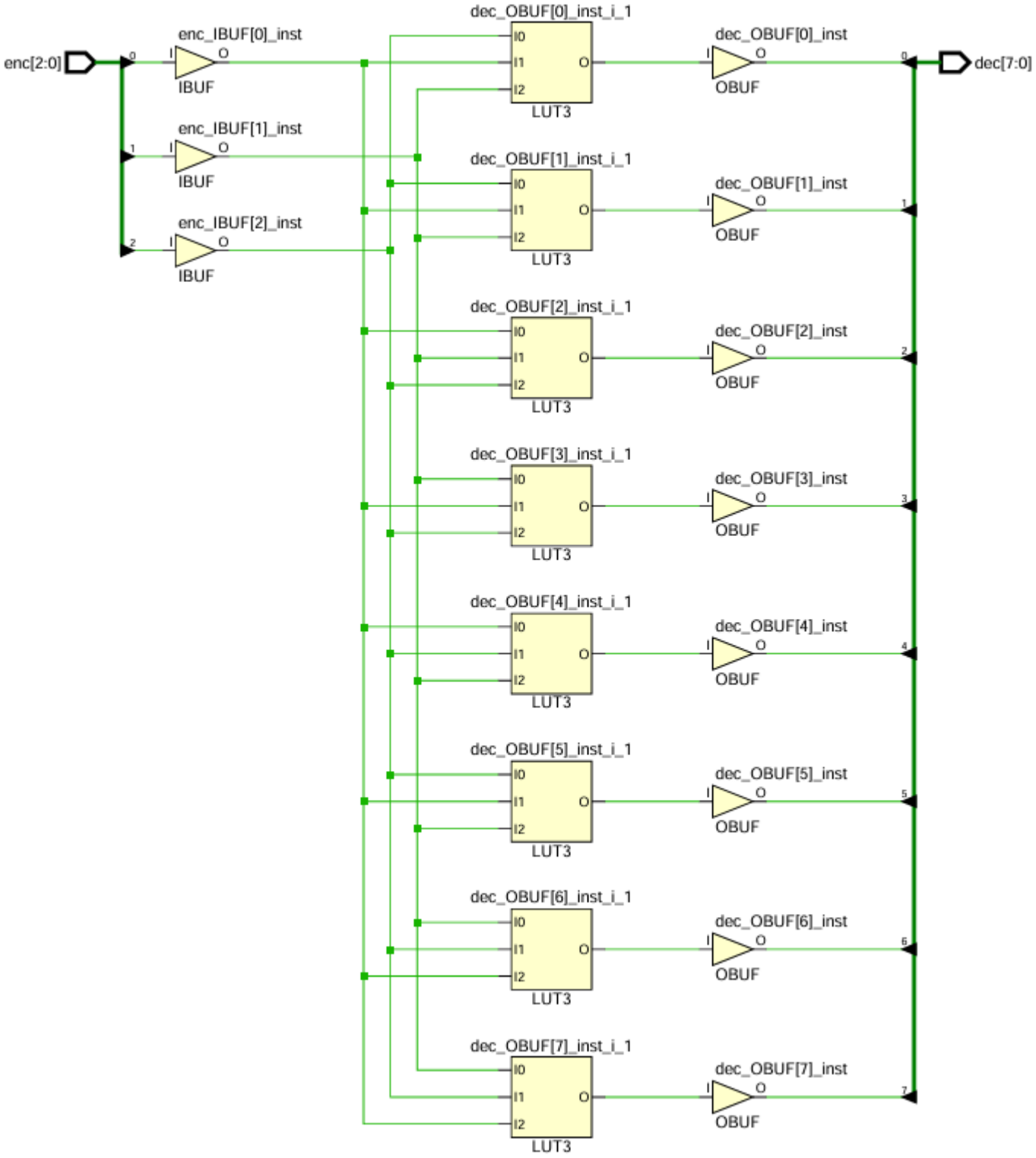
```
library IEEE;
use IEEE.std_logic_1164.all;

entity decoder_3_to_8_behavioral is
    port(
        enc : in std_logic_vector(3-1 downto 0);
        dec : out std_logic_vector(8-1 downto 0)
    );
end decoder_3_to_8_behavioral;

architecture behavioral_arch of decoder_3_to_8_behavioral is

begin

    decoder_3_to_8_module : process(enc)
    begin

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

```
end behavioral_arch;
```

Με τη χρήση του elaborate design παίρνουμε το εξής σχηματικό:



Με τη χρήση του synthesis design παίρνουμε το εξής σχηματικό:


enc[2:0]
enc_IBUF[0]_inst
IBUF
enc_IBUF[1]_inst
IBUF
enc_IBUF[2]_inst
IBUF
dec_OBUF[0]_inst_i_1
LUT3
dec_OBUF[1]_inst_i_1
LUT3
dec_OBUF[2]_inst_i_1
LUT3
dec_OBUF[3]_inst_i_1
LUT3
dec_OBUF[4]_inst_i_1
LUT3
dec_OBUF[5]_inst_i_1
LUT3
dec_OBUF[6]_inst_i_1
LUT3
dec_OBUF[7]_inst_i_1
LUT3
dec_OBUF[0]_inst
OBUF
dec_OBUF[1]_inst
OBUF
dec_OBUF[2]_inst
OBUF
dec_OBUF[3]_inst
OBUF
dec_OBUF[4]_inst
OBUF
dec_OBUF[5]_inst
OBUF
dec_OBUF[6]_inst
OBUF
dec_OBUF[7]_inst
OBUF
dec[7:0]

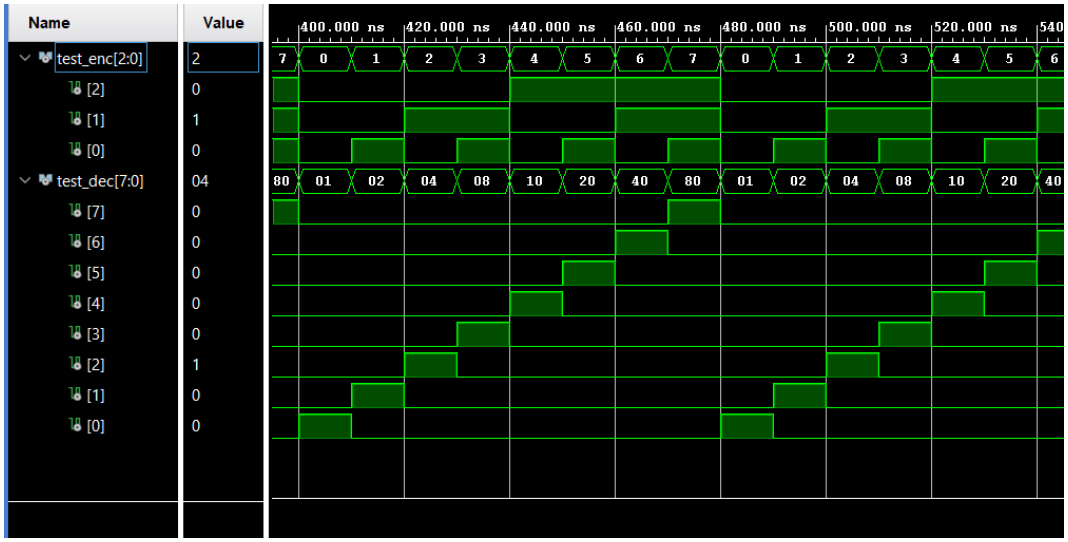
Μπορούμε να παρατηρήσουμε για τις δύο διαφορετικές αρχιτεκτονικές ότι ενώ το σχηματικό που προκύπτει από το elaborate design είναι διαφορετικό, καθώς εξαρτάται από τον τρόπο περιγραφής της αρχιτεκτονικής, το σχηματικό που προκύπτει μετά το synthesis είναι και στις δύο περιπτώσεις το ίδιο. Αυτό σημαίνει ότι αν και έχει περιγραφεί με δύο διαφορετικές αρχιτεκτονικές στον κώδικα ο αποκωδικοποιητής, μετά το synthesis και στις δύο περιπτώσεις θα έχει την ίδια δομή.

Για να μπορούμε να ελέγξουμε την ορθή λειτουργία του αποκωδικοποιητή παράγουμε όλα τα πιθανά σήματα εισόδου και παρατηρούμε τις εξόδους σε καθένα από αυτά:

```
library IEEE;

use IEEE.std_logic_1164.all;

entity decoder_3_to_8_tb_behavioral is
end decoder_3_to_8_tb_behavioral;

architecture decoder_3_to_8_tb_behavioral_arch of decoder_3_to_8_tb_behavioral is

component decoder_3_to_8_behavioral

port(
    enc : in std_logic_vector(3-1 downto 0);
    dec : out std_logic_vector(8-1 downto 0)
);

end component;

-- stimulus signals
signal test_enc : std_logic_vector(3-1 downto 0);
signal test_dec : std_logic_vector(8-1 downto 0);

begin

uut: decoder_3_to_8_behavioral
    port map(
        enc => test_enc,
        dec => test_dec
    );

testing: process

begin
    test_enc <= "000";
    wait for 10 ns;

    test_enc <= "001";
    wait for 10 ns;

    test_enc <= "010";
    wait for 10 ns;

    test_enc <= "011";
    wait for 10 ns;
```

```
    test_enc <= "100";
    wait for 10 ns;

    test_enc <= "101";
    wait for 10 ns;

    test_enc <= "110";
    wait for 10 ns;

    test_enc <= "111";
    wait for 10 ns;

end process;

end decoder_3_to_8_tb_behavioral_arch;
```

Τα αποτελέσματα της προσομοίωσης φαίνονται παρακάτω:



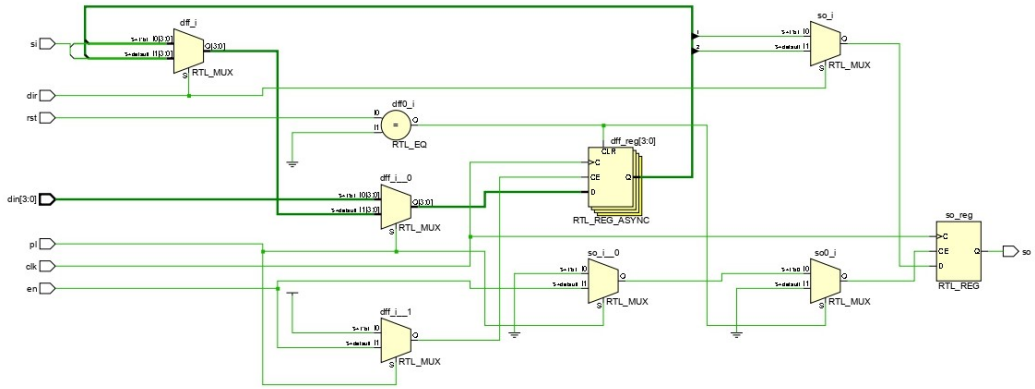
Παρατηρούμε και σε αυτήν την περίπτωση την ορθή σχέση μεταξύ κάθε εισόδου και αντίστοιχης εξόδου.

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

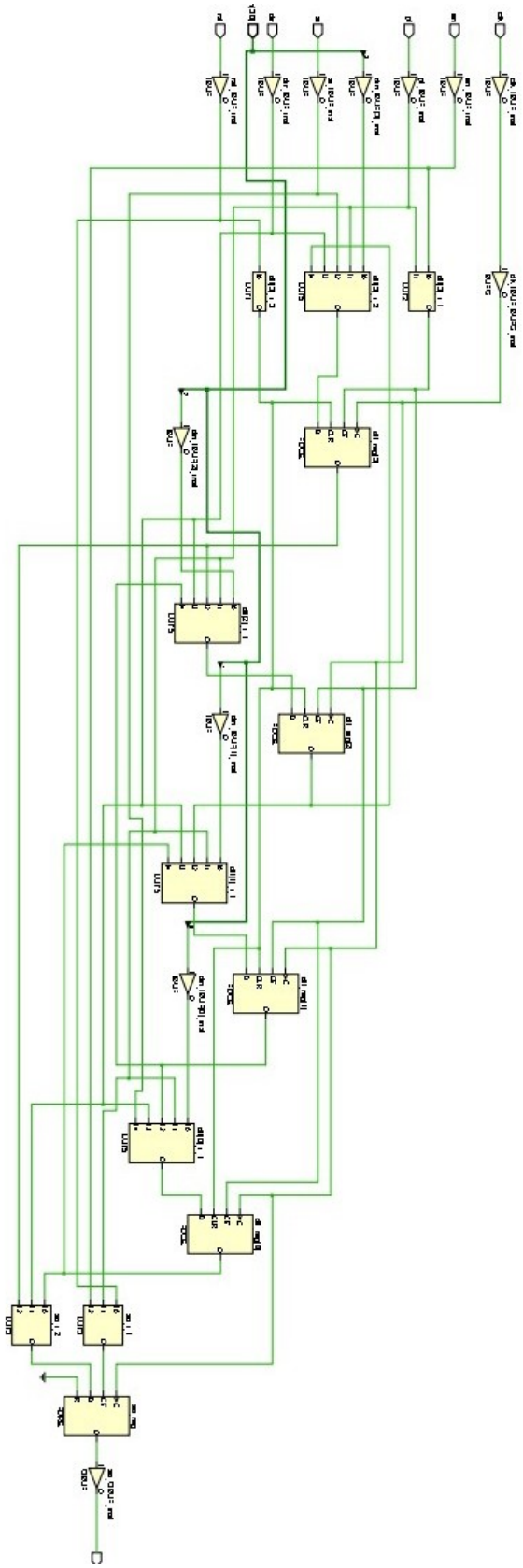
Ο κώδικας σε vhdl για την περιγραφή του καταχωρητή ολίσθησης με μια επιπλέον είσοδο (std_logic) η οποία θα επιλέγει ανάμεσα σε αριστερή και δεξιά ολίσθηση είναι ο εξής:

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

entity shift_reg4 is
    port (
        clk, rst, en, pl, dir : in std_logic;
        din : in std_logic_vector(3 downto 0);
        si : in std_logic;
        so : out std_logic
        -- q : out std_logic_vector(3 downto 0)
    );
end shift_reg4;

architecture rtl of shift_reg4 is
    signal dff : std_logic_vector(3 downto 0);

begin
    process (clk, rst)
    begin
        if rst = '0' then
            dff <= (others => '0');
        elsif rising_edge(clk) then
            if pl = '1' then
                dff <= din;
            elsif en = '1' then
                if dir = '1' then -- Right Shift
                    dff <= si & dff(3 downto 1);
                    so <=dff(0);
                else -- Left Shift
                    dff <= dff(2 downto 0) & si;
                    so <=dff(3);
                end if;
            end if;
        end if;
    end process;
end rtl;
```

Με τη χρήση του elaborate design παίρνουμε το εξής σχηματικό:



Με τη χρήση του synthesis design παίρνουμε το εξής σχηματικό:

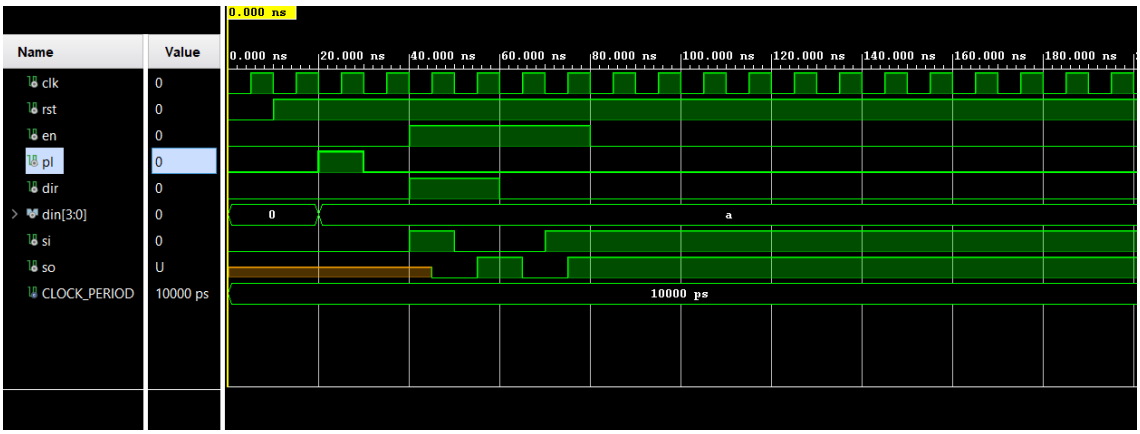


Παρατηρούμε πως παραπάνω σχηματικό έχει, όπως και αυτό του καταχωρητή ολίσθησης χωρίς επιλογή κατεύθυνσης, 4 D Flip Flops. Η είσοδος dir, η οποία είναι υπεύθυνη για την επιλογή κατεύθυνση, περνάει στην έξοδο so μέσω των LUT. Για την εκτέλεση της προσομοίωσης υλοποιήθηκε το παρακάτω testbench:

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

entity shift_reg4_tb is
end entity;

architecture tb of shift_reg4_tb is
    component shift_reg4 is
        port (
            clk, rst, en, pl, dir : in std_logic;
            din : in std_logic_vector(3 downto 0);
            si : in std_logic;
            so : out std_logic
        );
    end component;

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

    constant CLOCK_PERIOD : time := 10 ns;

begin
    DUT: shift_reg4
        port map (
            clk => clk,
            rst => rst,
            en  => en,
            pl  => pl,
            dir => dir,
            din => din,
            si  => si,
            so  => so
        );

    STIMULUS: process
    begin
        -- Apply reset
        rst <= '0';
        wait for CLOCK_PERIOD;
        rst <= '1';
        wait for CLOCK_PERIOD;

        -- Load parallel data
        pl <= '1';
        din <= "1010";
        wait for CLOCK_PERIOD;
        pl <= '0';
        wait for CLOCK_PERIOD;
```

```
        -- Right shift with serial input '1'
        en <= '1';
        dir <= '1';
        si <= '1';
        wait for CLOCK_PERIOD;
        si <= '0';
        wait for CLOCK_PERIOD;

        -- Left shift with serial input '0'
        dir <= '0';
        si <= '0';
        wait for CLOCK_PERIOD;
        si <= '1';
        wait for CLOCK_PERIOD;

        -- Disable shifting
        en <= '0';
        wait for CLOCK_PERIOD;

        wait;
    end process;

    GEN_CLK: process
    begin
        while true loop
            clk <= '0';
            wait for CLOCK_PERIOD / 2;
            clk <= '1';
            wait for CLOCK_PERIOD / 2;
        end loop;
    end process;

end architecture;
```

Για να ελέγξουμε τη λειτουργία την ορθή λειτουργία του καταχωρητή, ενεργοποιούμε αρχικά τη φόρτωση δεδομένων μέσω του din και το σήμα en για την ολίσθηση. Στη συνέχεια, δοκιμάζουμε και τις δύο κατευθύνσεις ολίσθησης με διαφορετική τιμή εισόδου και si, ώστε να παρατηρήσουμε τις αλλαγές στο so. Σημειώνουμε ότι έχουμε υλοποίηση επίσης process που προσομοιώνει το ρολόι. Τα αποτελέσματα της προσομοίωσης είναι τα εξής:



Επομένως, μπορούμε να παρατηρήσουμε ότι οι τιμές τις εξόδου και η μορφή των σημάτων επαληθεύουν την ορθή λειτουργία του καταχωρητή που υλοποιήσαμε.

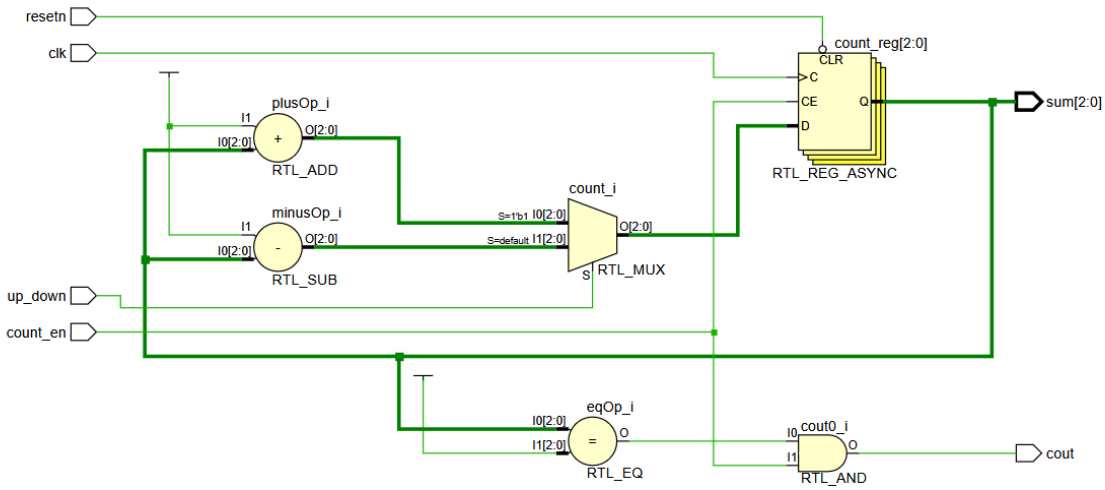
Θέμα B.3: Μετρητής 3 bit με είσοδο ενεργοποίησης και κρατούμενο εξόδου

1. Up/Down μετρητής χωρίς όριο

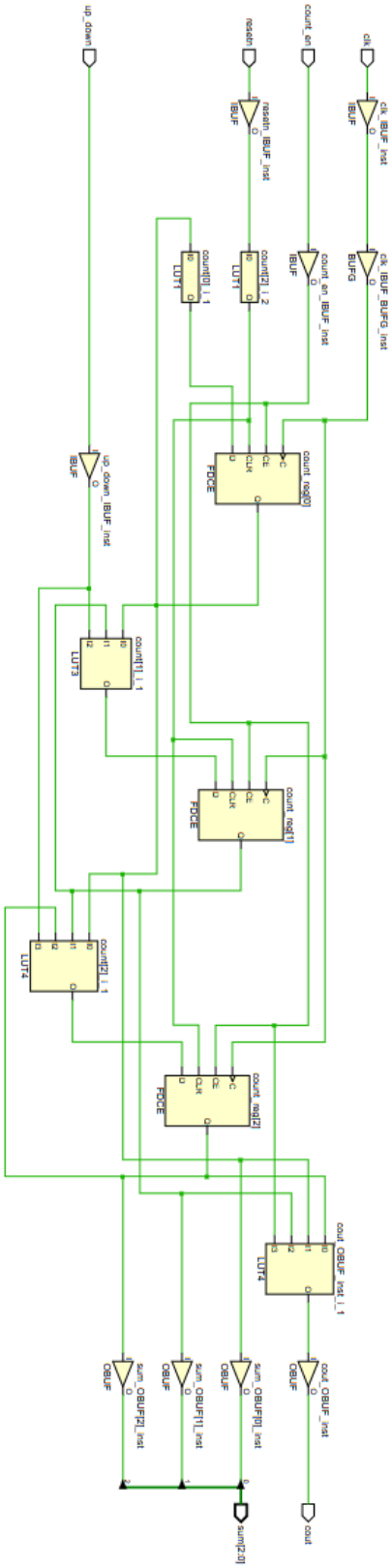
Ο κώδικας σε vhdl για την περιγραφή του up/down μετρητή χωρίς όριο είναι ο εξής:

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

entity count3_up_down 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 count3_up_down;

architecture rtl of count3_up_down is
    signal count: std_logic_vector(2 downto 0):= "000";

begin
    process(clk, resetn)
    begin
        if resetn='0' then
            -- Κώδικας για την περίπτωση του reset (ενεργό χαμηλά)
            count <= (others=>'0');
        elsif clk'event and clk='1' then
            -- Χρήση case για περίπτωση up/down
            case(up_down) is
                when '1' =>
                    if count_en='1' then
                    -- Μέτρηση μόνο αν count_en = 1
                        count<=count+1;
                    end if;
                when others =>
                    if count_en='1' then
                    -- Μέτρηση μόνο αν count_en = 1
                        count<=count-1;
                    end if;
            end case;
        end if;
    end process;
    -- Ανάθεση τιμων στα σήματα εξόδου

    sum <= count;
    cout <= '1' when count=7 and count_en='1' else '0';
end rtl;
```

Με τη χρήση του elaborate design παίρνουμε το εξής σχηματικό:



Με τη χρήση του synthesis design παίρνουμε το εξής σχηματικό:

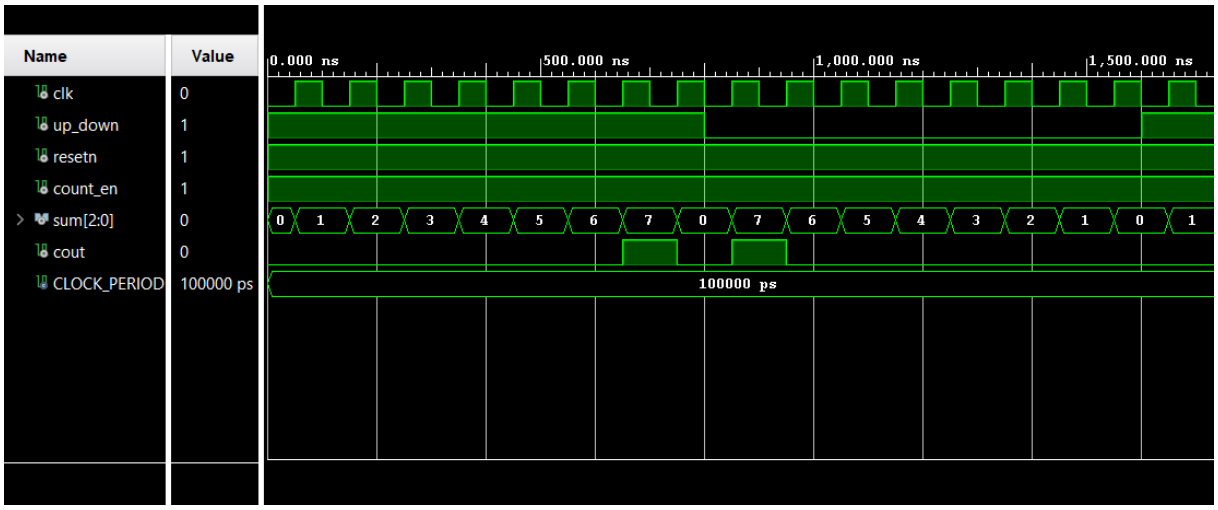


Για την εκτέλεση της προσομοίωσης υλοποιήθηκε το παρακάτω testbench:

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

entity count3_up_down_tb is
end entity;

architecture tb of count3_up_down_tb is
    component count3_up_down 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;


    -- inputs
    signal clk: std_logic;
    signal up_down : std_logic;
    signal resetn :  std_logic := '1';
    signal count_en : std_logic := '1';
    -- outputs
    signal sum : std_logic_vector(2 downto 0);
    signal cout : std_logic;

    constant CLOCK_PERIOD : time := 100 ns;

begin
    DUT: count3_up_down
        port map(
            clk => clk,
            resetn => resetn,
            count_en => count_en,
            up_down => up_down,
            sum => sum,
            cout => cout
    );

    process
    begin
        up_down <= '1';

        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;

        up_down <= '0';

        wait for CLOCK_PERIOD;
```

```
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;

    end process;

    GEN_CLK: process
    begin
        while true loop
            clk <= '0';
            wait for CLOCK_PERIOD / 2;
            clk <= '1';
            wait for CLOCK_PERIOD / 2;
        end loop;
    end process;

end architecture;
```

Για την προσομοίωση επιλέγουμε να θέσουμε την λειτουργία του μετρητή σε up για 8 κύκλους και στη συνέχεια σε down για άλλους 8 κύκλους. Παρακάτω φαίνεται η εκτέλεση της προσομοίωσης.



Από την παραπάνω εικόνα μπορούμε να παρατηρήσουμε ότι ο counter αυξάνεται για τους πρώτους 8 κύκλους και όταν φτάσει το 7 μηδενίζεται (με βάση τη λειτουργία του +). Στη συνέχεια για τους επόμενους 8 κύκλους μειώνεται μέχρι το 0. Αξίζει να σημειώσουμε επίσης ότι κάθε φορά που ο counter φτάνει την τιμή 7 το cout παίρνει την τιμή 1.

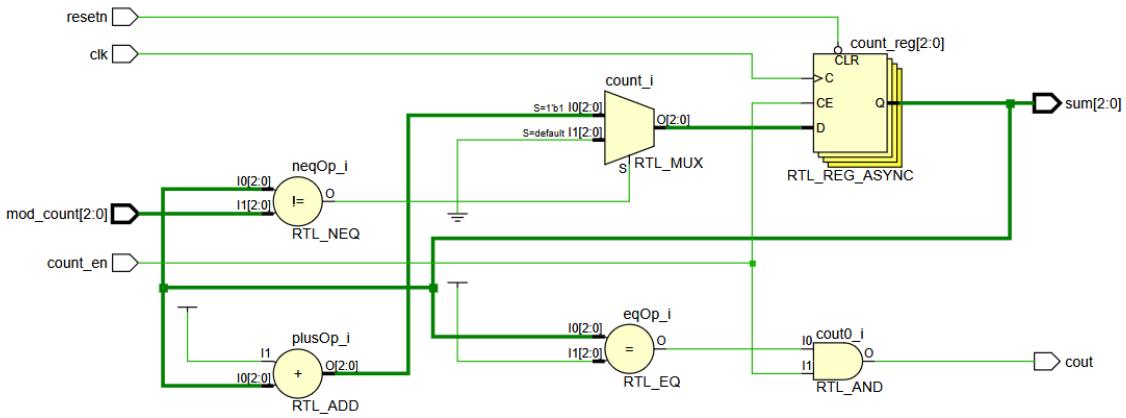
1. Up μετρητής με όριο

Ο κώδικας σε vhdl για την περιγραφή του up μετρητή με όριο (έχουμε επιλέξει το 5) είναι ο εξής:

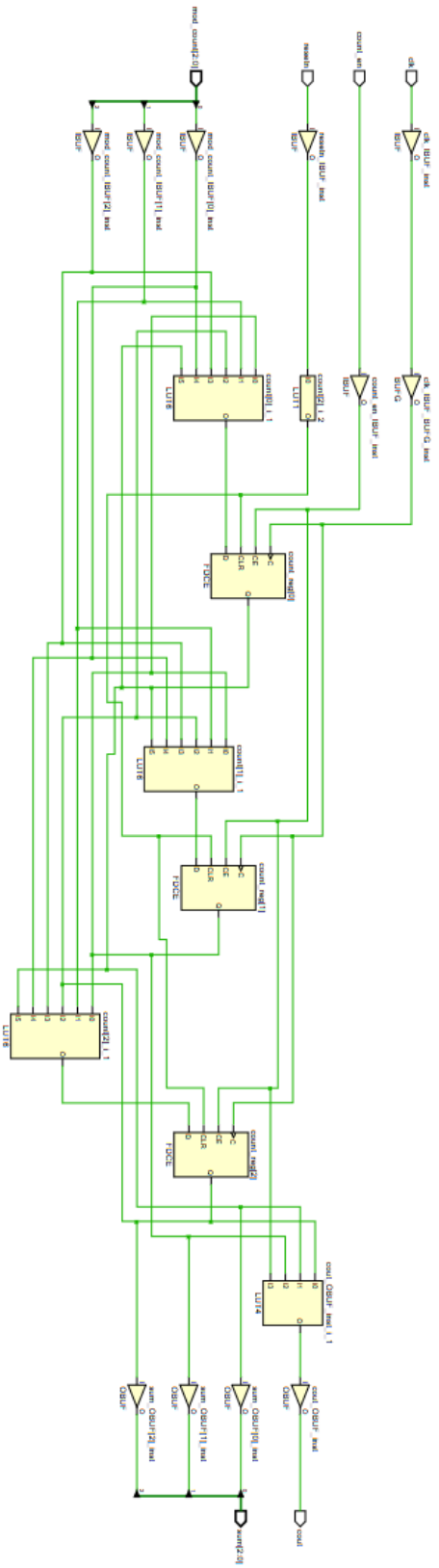
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity count3_up_down_mod is
    port(
        clk: in std_logic;
        resetn: in std_logic;
```

```
        count_en : in std_logic;
        mod_count : in std_logic_vector(2 downto 0):= "101";
        sum : out std_logic_vector(2 downto 0);
        cout : out std_logic

        );
end count3_up_down_mod;

architecture rtl of count3_up_down_mod is
    signal count: std_logic_vector(2 downto 0):= "000";

begin
    process(clk, resetn)
    begin

        if resetn='0' then
            -- Κώδικας για την περίπτωση του reset (ενεργο χαμηλά)
            count <= (others=>'0');

        elsif clk'event and clk='1' then
            -- Χρήση case για περίπτωση up/down
                if count_en='1' then
                -- Μέτρηση μονο αν count_en = 1
                    if count/=mod_count then
                        count<=count+1;
                    else
                        count <= "000";
                    end if;
                end if;
        end if;
    end process;
    -- Ανάθεση τιμων στα σήματα εξόδου

    sum <= count;
    cout <= '1' when count=7 and count_en='1' else '0';
end rtl;
```

Με τη χρήση του elaborate design παίρνουμε το εξής σχηματικό:



Με τη χρήση του synthesis design παίρνουμε το εξής σχηματικό:

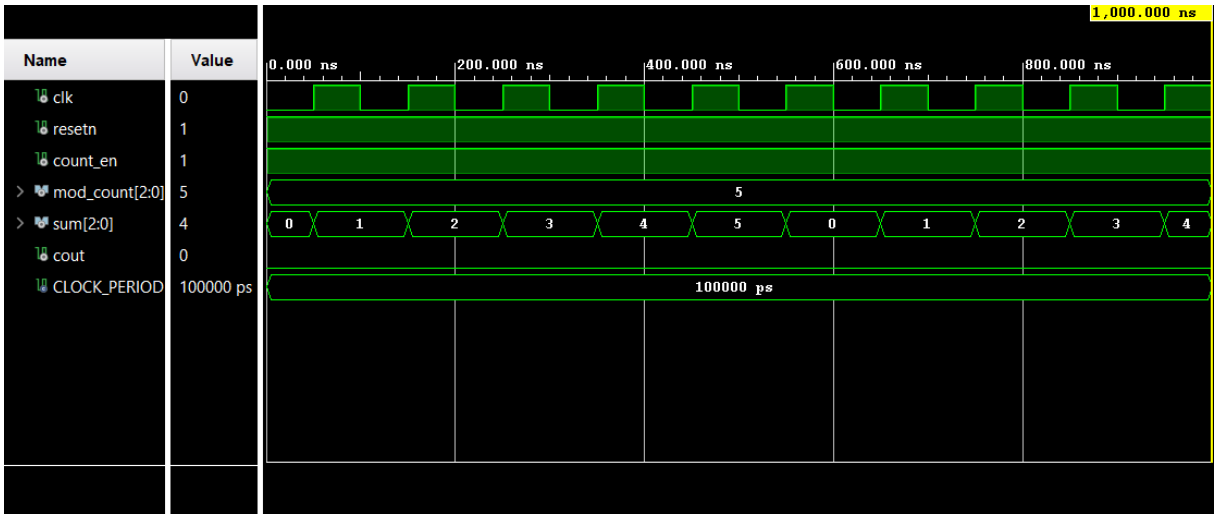


Για την εκτέλεση της προσομοίωσης υλοποιήθηκε το παρακάτω testbench:

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

entity count3_up_down_mod_tb is
end entity;

architecture tb of count3_up_down_mod_tb is
    component count3_up_down_mod is
        port (
            clk : in std_logic;
            resetn : in std_logic;
            count_en: in std_logic;
            mod_count : in std_logic_vector(2 downto 0);
            sum : out std_logic_vector(2 downto 0);
            cout : out std_logic
        );
    end component;

    -- inputs
    signal clk: std_logic;
    signal resetn :  std_logic := '1';
    signal count_en : std_logic := '1';
    signal mod_count : std_logic_vector(2 downto 0) := "101";

    -- outputs
    signal sum : std_logic_vector(2 downto 0);
    signal cout : std_logic;

    constant CLOCK_PERIOD : time := 100 ns;

begin
    DUT: count3_up_down_mod
        port map(
            clk => clk,
            resetn => resetn,
            count_en => count_en,
            mod_count => mod_count,
            sum => sum,
            cout => cout
    );

    process
    begin
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;

        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
```

```
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;
        wait for CLOCK_PERIOD;

    end process;

    GEN_CLK: process
    begin
        while true loop
            clk <= '0';
            wait for CLOCK_PERIOD / 2;
            clk <= '1';
            wait for CLOCK_PERIOD / 2;
        end loop;
    end process;

end architecture;
```

Για την εκτέλεση της προσομοίωσης επιλέγουμε να τρέξουμε για 16 κύκλους ρολογιού.Το αποτέλεσμα της προσομοίωσης είναι το εξής:



Μπορούμε πράγματι να παρατηρήσουμε ότι ο μετρητής φτάνει έως την τιμή 5, που είναι και το limit του και στη συνέχεια μηδενίζεται ώστε να αρχίσει ξανά την άνω μέτρηση.