



**Εθνικό Μετσόβιο Πολυτεχνείο**  
**Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών**  
**Υπολογιστών**  
Εξάμηνο: 8<sup>ο</sup> Ακ.  
Έτος: 2024- 2025

## **«Ψηφιακά Συστήματα VLSI»**

Χαράλαμπος Παπαδόπουλος (ΑΜ: 03120199)  
Νικόλαος Παπακωνσταντόπουλος (ΑΜ: 03120069)

Ομάδα: 45

## A\_2

Το ζητούμενο αυτής της άσκησης ήταν η υλοποίηση ενός 3 to 8 decoder με χρήση behavioral και dataflow VHDL.

Παρατίθεται αρχικά η behavioral περιγραφή:

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

entity dec is
    Port (enc_in  : in  std_logic_vector(2 downto 0);
          dec_out : out std_logic_vector(7 downto 0) );
end dec;

architecture behave of dec is
begin
    process(enc_in)
    begin
        case enc_in is
            when "000" =>
                dec_out <= "000000001";
            when "001" =>
                dec_out <= "000000010";
            when "010" =>
                dec_out <= "000000100";
            when "011" =>
                dec_out <= "000001000";
            when "100" =>
                dec_out <= "000100000";
            when "101" =>
                dec_out <= "001000000";
            when "110" =>
                dec_out <= "010000000";
            when "111" =>
                dec_out <= "100000000";
            when others =>
                dec_out <= "000000000";
        end case;
    end process;
end behave;
```

Και η dataflow:

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

entity dec is
    Port (enc_in  : in  std_logic_vector(2 downto 0);
          dec_out : out std_logic_vector(7 downto 0) );
end dec;
```

```

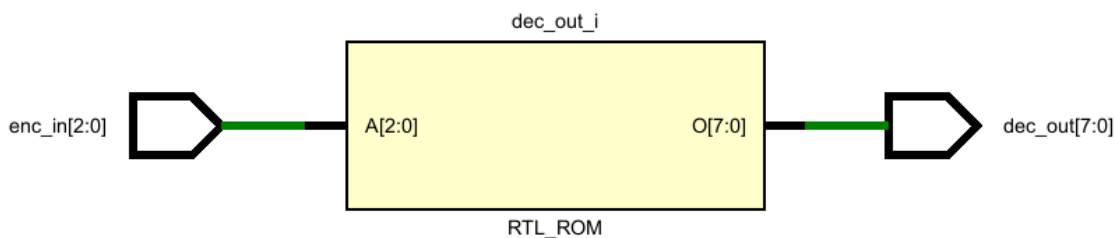
architecture dataflow of dec is
begin
  with enc_in select
    dec_out <= "00000001" when "000",
               "00000010" when "001",
               "00000100" when "010",
               "00001000" when "011",
               "00010000" when "100",
               "00100000" when "101",
               "01000000" when "110",
               "10000000" when "111",
               "00000000" when others;
end dataflow;

```

Αξίζει σε αυτό το σημείο να σημειωθεί πως, θεωρητικά, μία dataflow αρχιτεκτονική περιγράφει την ροή δεδομένων σε επίπεδο λογικών πυλών, ενώ μία behavioral αντιμετωπίζει το σύστημα σαν μαύρο κουτί και δίνει απλά περιγραφή εισόδου εξόδου.

Δεδομένου αυτού του ορισμού, η παραπάνω dataflow αρχιτεκτονική είναι στην πραγματικότητα behavioral. Όμως, στο βιβλίο «free range VHDL» που μας προτάθηκε, αναγράφεται πως η εντολή with - select ανήκει στην κατηγορία dataflow.

Παρακάτω φαίνεται το RTL (κοινό για τις δύο περιγραφές).



Tesbench:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;
use IEEE.NUMERIC_STD.ALL;
use std.textio.ALL;

entity dec_tb is
end dec_tb;

architecture test of dec_tb is
  component dec
    Port (

```

```

        enc_in : in  std_logic_vector(2 downto 0);
        dec_out : out std_logic_vector(7 downto 0)
    );
end component;

signal enc_in      : std_logic_vector(2 downto 0);
signal dec_out     : std_logic_vector(7 downto 0);



begin
    uut: dec
        port map(
            enc_in => enc_in,
            dec_out => dec_out
        );

    process
    begin

        for i in 0 to 7 loop
            enc_in <= std_logic_vector(to_unsigned(i, 3));
            wait for 10 ns;
        end loop;

        wait;
    end process;
end test;

```

Name	Value	0.000 ns	20.000 ns	40.000 ns	60.000 ns	80.000 ns	100.000		
>  enc_in[2:0]	0	0	1	2	3	4	5	6	7
>  dec_out[7:0]	01	01	02	04	08	10	20	40	80

## B\_2

Ζητούμενο της άσκησης ήταν η υλοποίηση ενός καταχωρητή ολίσθησης των 4 bits με παράλληλη φόρτωση.

Μας δόθηκε έτοιμος κώδικας δεξιάς ολίσθησης, τον οποίο εμείς επεκτείναμε με την λειτουργικότητα της επιλογής ανάμεσα σε αριστερή και δεξιά ολίσθηση μέσω του σήματος `dir` και της συμπεριφοράς

$$dir = \{ 0, \quad right\ shift\ 1, \quad left\ shift$$

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

entity shifter is

    port (
        clk, rst, si, en, pl, dir: in std_logic;           -- added direction input
        din: in std_logic_vector(3 downto 0);
        so: out std_logic);

end shifter;

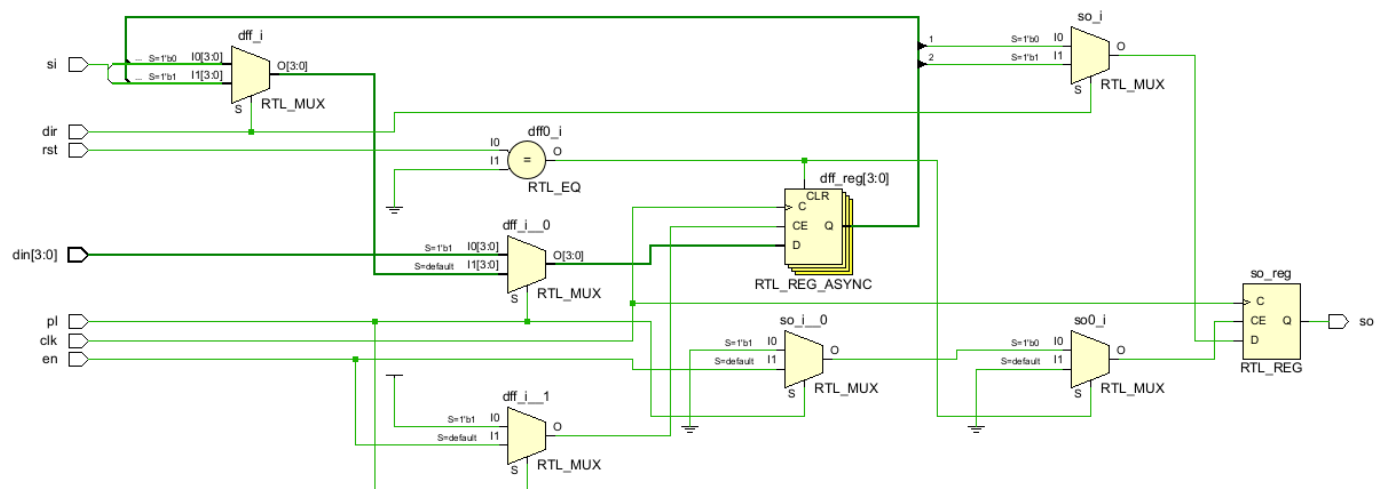
architecture rtl of shifter is
    signal dff: std_logic_vector(3 downto 0);
begin
    edge: process (clk, rst)
    begin
        if rst = '0' then
            dff <= (others => '0');
        elsif rising_edge(clk) then
            if pl = '1' then
                dff <= din;
            elsif en = '1' then
                case dir is
                    when '0' =>
                        so <= dff(0); -- save the LSB to output
                        dff <= si & dff(3 downto 1); -- right shift
                    when '1' =>
                        so <= dff(3);
                        dff <= dff(2 downto 0) & si; -- left shift
                    when others =>
                        so <= '0'; -- default
                        dff <= (others => '0'); -- default
                end case;
            end if;
        end if;
    end process;

    --so <= dff(0);

end rtl;
```

Στον δοσμένο κώδικα η ανάθεση του `so` γινόταν έξω από το `process` block, δηλαδή «έτρεχε» παράλληλα (concurrently) με αυτό, το οποίο οδηγούσε σε ανεπιθύμητη συμπεριφορά. Γι' αυτό, το κάναμε comment out και το προσθέσαμε εντός του `process` block για να παίρνει την σωστή τιμή.

RTL schematic:



Testbench:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;
use std.textio.ALL;

entity shifter_tb is
end shifter_tb;

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

    signal clk          : std_logic;
    signal rst, si, en, pl, dir : std_logic := '0';
    signal din          : std_logic_vector(3 downto 0) :=
"0000";
    signal so           : std_logic;

    constant CLOCK_PERIOD : time := 10 ns;

begin

    uut: shifter
        port map (
            clk => clk,
            rst => rst,
            si  => si,
            en  => en,
            pl  => pl,
            dir => dir,
            din => din,
```

```

        so => so
    );
GEN_CLK : process
begin
    clk <= '1';
    wait for (CLOCK_PERIOD / 2);
    clk <= '0';
    wait for (CLOCK_PERIOD / 2);
end process;

STIMULUS : process
begin
    rst <= '1';
    wait for (1 * CLOCK_PERIOD);

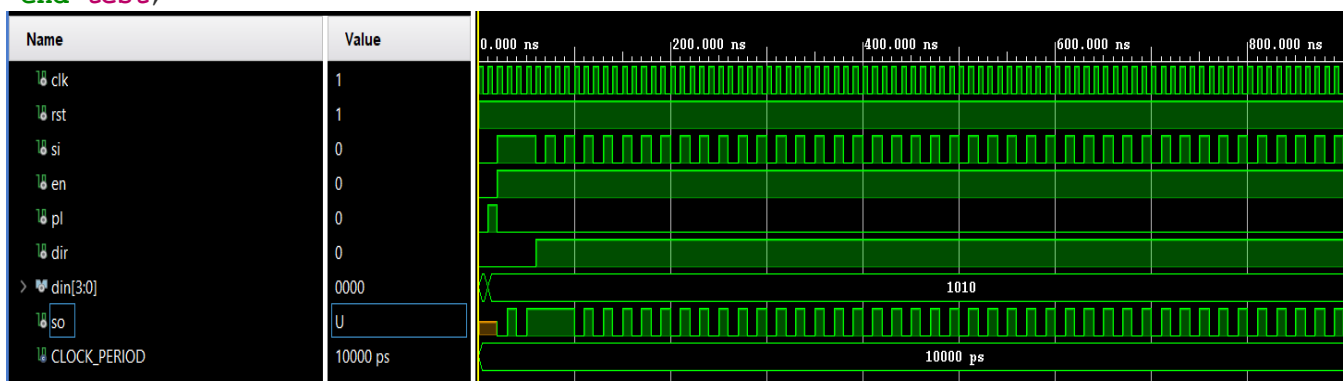
    din <= "1010";
    pl <= '1';
    en <= '0';
    wait for (1 * CLOCK_PERIOD);
    pl <= '0';
    en <= '1';

    -- Right Shift
    dir <= '0';
    si <= '1';
    wait for (4 * CLOCK_PERIOD);

    -- Left Shift
    dir <= '1';
    si <= '0';
    wait for (1 * CLOCK_PERIOD);
    loop
        si <= '1';
        wait for (1 * CLOCK_PERIOD);
        si <= '0';
        wait for (1 * CLOCK_PERIOD);
    end loop;
    wait;
end process;

```

end test;



## B\_3

Το ζητούμενο αυτής της άσκησης είναι η υλοποίηση ενός μετρητή 3 bit με δύο εκδοχές:

- δυνατότητα μέτρησης προς τα πάνω και προς τα κάτω (up down) με χρήση του σήματος up\_bool (όταν είναι 1 γίνεται μέτρηση προς τα πάνω στην ανερχόμενη ακμή του παλμού ρολογιού clk αλλιώς μέτρηση προς τα κάτω)
- δυνατότητα μέτρησης πάνω με όριο κάποιον αριθμό (modulo counter) με χρήση ενός σύγχρονου σήματος (std\_logic\_vector) max\_count που υποδηλώνει τον μέγιστο αριθμό μέχρι τον οποίο φτάνει η μέτρηση

Στην περιγραφή του architecture των entities χρησιμοποιείται process το οποίο θέτει 0 στην έξοδο αν η ασύγχρονη είσοδος resetn είναι 0. Αλλιώς σε κάθε ανερχόμενη ακμή ρολογιού ελέγχονται οι σύγχρονες είσοδοι και αναλόγως πραγματοποιείται αλλαγή της εξόδου σε έναν αριθμό πάνω ή κάτω (για την περίπτωση του up down counter) ή και σε μηδενισμό (στην περίπτωση του modulo counter).

Στα testbenches πραγματοποιούνται η μέτρηση προς τα πάνω και κάτω για κάμποσους ενδεικτικούς κύκλους καθώς και η μέτρηση με πάνω όριο κάποιον αριθμό (δοκιμάζονται διαδοχικά 0, 1, 2, ... ως πάνω όρια)

### up down

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_bool: in STD_LOGIC;
      sum : out STD_LOGIC_VECTOR (2 downto 0);
      cout : out STD_LOGIC);
```

end count3\_up\_down;

architecture rtl\_nolimit of count3\_up\_down is

signal count: std\_logic\_vector(2 downto 0);

begin

process(clk,resetn)

begin

if resetn = '0' then

-- if reset is 0 make the count 0

count <= (others => '0');

elsif clk'event and clk = '1' then

if count\_en='1' then



```

-- increment count only in the rising edge if count_en is 1
if up_bool='1' then
    count <= count + 1;
else
    count <= count - 1;
end if;
end if;
end if;
end process;

-- signal assignment
sum <= count;
cout <= '1' when count=7 and count_en='1' else '0';
end rtl_nolimit;

```

### **test bench**

```

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

entity count_up_down_tb is
end count_up_down_tb;

architecture Behavioral of count_up_down_tb is

    component count3_up_down
    Port ( clk : in STD_LOGIC;
          resetn : in STD_LOGIC;
          count_en : in STD_LOGIC;
          up_bool: in STD_LOGIC;
          sum : out STD_LOGIC_VECTOR (2 downto 0);
          cout : out STD_LOGIC);
    end component;

    signal clk : std_logic := '0';
    signal resetn, count_en, up_bool, cout : std_logic;
    signal sum : std_logic_vector(2 downto 0);
begin
    uut: count3_up_down
        port map ( clk => clk,
                  resetn => resetn,
                  count_en => count_en,
                  up_bool => up_bool,
                  sum => sum,

```

```
    cout => cout);
```

```
clock_process: process
```

```
begin
```

```
    clk <= '0';
```

```
    wait for 5 ns;
```

```
    clk <= '1';
```

```
    wait for 5 ns;
```

```
end process;
```

```
stimulus: process
```

```
begin
```

```
    -- first configure the enable and up_bool for count up
```

```
    count_en <= '1';
```

```
    up_bool <= '1';
```

```
    -- reset to 0 and wait for a clock to set enable and up_bool
```

```
    resetn <= '0'; -- reset is active low so output should be 000
```

```
    wait for 10 ns; -- wait for a clock to set enable and up_bool
```

```
    resetn <= '1'; -- reset gets 1
```

```
    wait for 100 ns; -- enable=1, up=1: count up
```

```
    up_bool <= '0'; -- configure up=0 for count down
```

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

```
    wait for 10 ns; -- wait for a clock to set enable and up_bool
```

```
    resetn <= '1'; -- reset gets 1
```

```
    wait for 100 ns; -- enable=1, up=0: count down
```

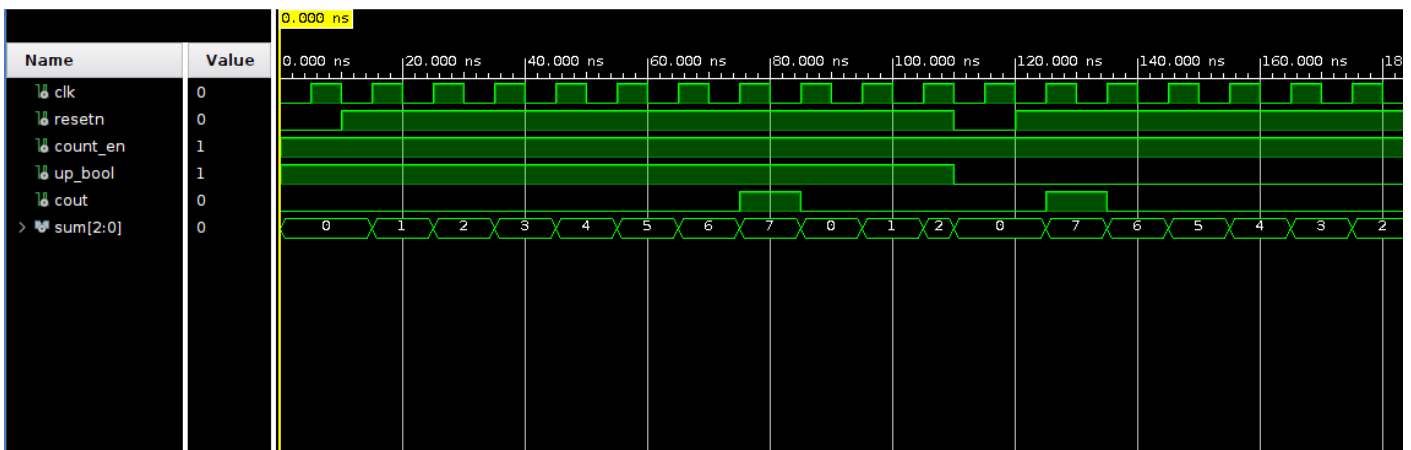
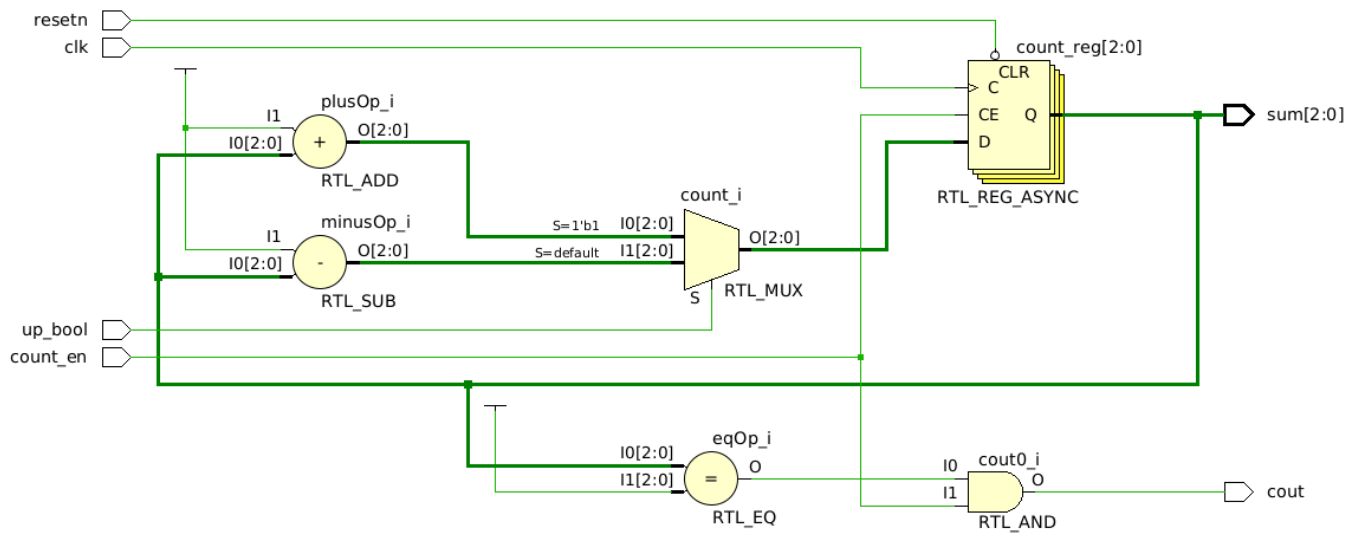
```
    resetn <= '0'; -- reset to 0
```

```
    wait for 10 ns; -- for better visualization
```

```
    wait;
```

```
end process;
```

```
end Behavioral;
```



## modulo

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

```
entity count3_modulo is
  Port ( clk : in STD_LOGIC;
        resetn : in STD_LOGIC;
        count_en : in STD_LOGIC;
        max_count : in STD_LOGIC_VECTOR (2 downto 0);
        sum : out STD_LOGIC_VECTOR (2 downto 0);
        cout : out STD_LOGIC);
end count3_modulo;
```

```
architecture rtl_limit of count3_modulo is
  signal count: std_logic_vector(2 downto 0);
begin
```

```

process(clk,resetn)
begin
    if resetn = '0' then
        -- if reset is 0 make the count 0
        count <= (others => '0');
    elsif clk'event and clk = '1' then
        if count_en='1' then
            -- increment count only in the rising edge if count_en is 1
            if count >= max_count then
                -- if count is equal to max_count then make the count 0
                count <= (others => '0');
            else
                -- increment the count
                count <= count + 1;
            end if;
        end if;
    end if;
end process;

-- signal assignment
sum <= count;
cout <= '1' when count=7 and count_en='1' else '0';
end rtl_limit;

```

### **test bench**

```

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

entity count_modulo_tb is
end count_modulo_tb;

architecture Behavioral of count_modulo_tb is

    component count3_modulo
    Port ( clk : in STD_LOGIC;
          resetn : in STD_LOGIC;
          count_en : in STD_LOGIC;
          max_count : in STD_LOGIC_VECTOR (2 downto 0);
          sum : out STD_LOGIC_VECTOR (2 downto 0);
          cout : out STD_LOGIC);
    end component;

```

```

signal clk : std_logic := '0';
signal resetn, count_en, cout : std_logic;
signal sum, max_count : std_logic_vector(2 downto 0);
begin
    uut: count3_modulo
        port map ( clk => clk,
                    resetn => resetn,
                    count_en => count_en,
                    max_count => max_count,
                    sum => sum,
                    cout => cout);

    clock_process: process
    begin
        clk <= '0';
        wait for 5 ns;
        clk <= '1';
        wait for 5 ns;
    end process;

    stimulus: process
    begin
        -- first configure
        count_en <= '1';

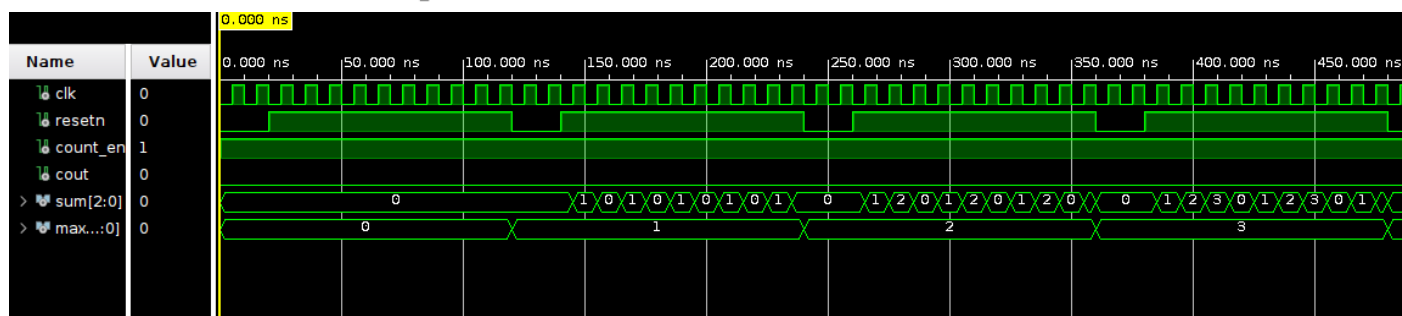
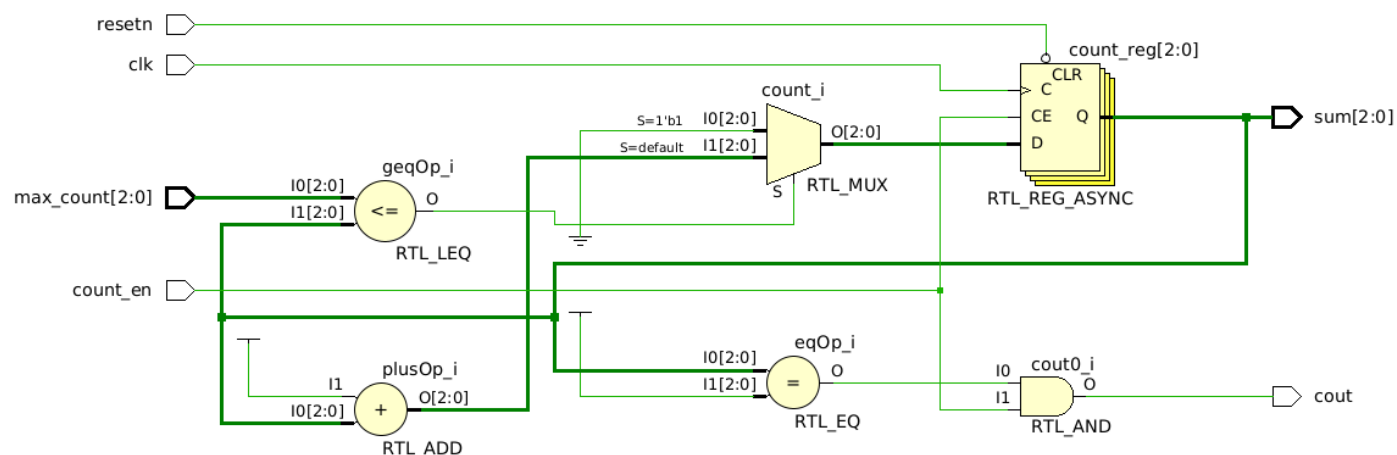
        -- make a for loop for all possible values of max_count
        for i in 0 to 7 loop
            max_count <= std_logic_vector(to_unsigned(i, 3));

            -- reset to 0 and wait for a clock to set values
            resetn <= '0'; -- reset is active low so output should be 000
            wait for 20 ns; -- wait for a clock to set enable and max_count
            resetn <= '1'; -- reset gets 1

            wait for 100 ns; -- count up to max_count
        end loop;

        wait;
    end process;
end Behavioral;

```



Παρατηρώντας την υλοποίηση που προκύπτει από τον synthesizer, βλέπουμε ότι η ασύγχρονη είσοδος resetn συνδέεται με το clr του D flip flop με αποτέλεσμα όταν γίνει μηδέν, κατευθείαν μηδενίζεται και η έξοδος Q του flip flop (οπότε και το επιθυμητό σήμα εξόδου). Επίσης, η είσοδος count\_en συνδέεται με το CE του flip flop και την κάτω πύλη and, οπότε αν γίνει μηδέν απενεργοποιείται προσωρινά η λειτουργία μέτρησης. Τέλος ανάλογα με το κύκλωμα, χρησιμοποιούνται αθροιστές (+1 -1 , up down counter) για μέτρηση πάνω κάτω και συγκριτής για τον έλεγχο με το πάνω όριο (modulo counter). Επίσης χρησιμοποιείται και συγκριτής για τη σύγκριση με το 111 ώστε να προκύψει το κρατούμενο εξόδου.