1η Εργαστηριακή Άσκηση στο μάθημα

Ψηφιακά Συστήματα VLSI

Ομάδα 13 :

Γρίβας Αριστοτέλης – el19889

Αυγουστής Μολτσάνοβ Εμίλ – el17064

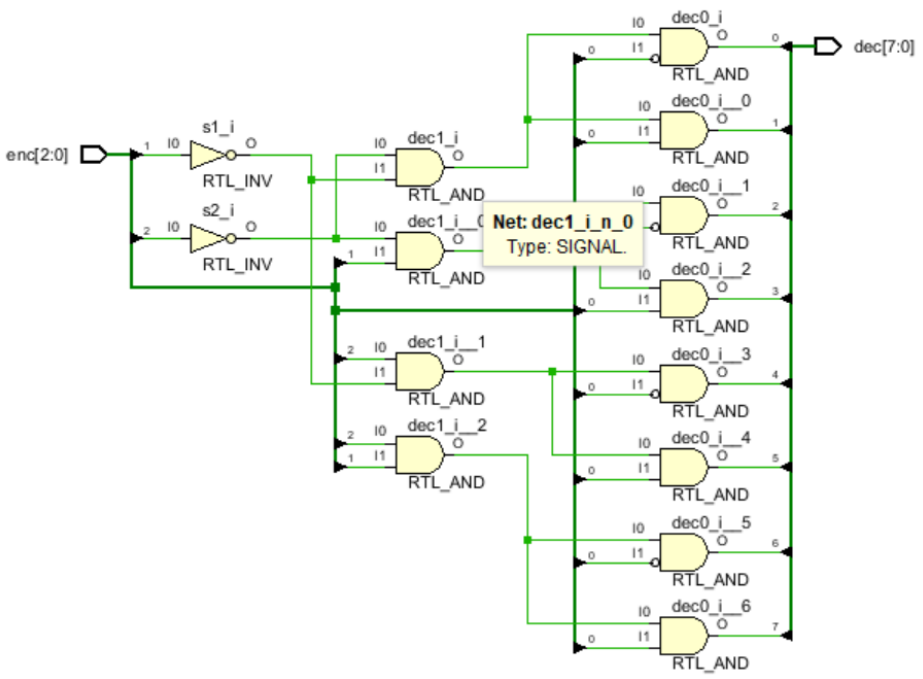
Άσκηση 1 (A2):

Ζητούμενο της άσκησης είναι η υλοποίηση ενός πολυπλέκτη 3 σε 8 με αρχιτεκτονικές dataflow και behavioral.

Αρχικα όσον αφορά στη dataflow,περιγράφουμε τη ροή των δεδομένων μέσω πυλών στον πολυπλέκτη.Ο κώδικας που χρησιμοποιήθηκε είναι ο εξής:

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

entity dec_3to8_dataflow is
port (
enc :in std_logic_vector(2 downto 0);
dec :out std_logic_vector(7 downto 0));
end dec_3to8_dataflow;

architecture data_arch of dec_3to8_dataflow is
signal s0, s1, s2: std_logic;
begin
s0 <= not enc(0);
s1 <= not enc(1);
s2 <= not enc(2);
dec(0) <= s2 and s1 and s0;
dec(1) <= s2 and s1 and enc(0);
dec(2) <= s2 and enc(1) and s0;
dec(3) <= s2 and enc(1) and enc(0);
dec(4) <= enc(2) and s1 and s0;
dec(5) <= enc(2) and s1 and enc(0);
dec(6) <= enc(2) and enc(1) and s0;
dec(7) <= enc(2) and enc(1) and enc(0);
end data_arch;
```

Το RTL διάγραμμα που παράγεται στο vivado μέσω του κώδικα είναι το παρακάτω:

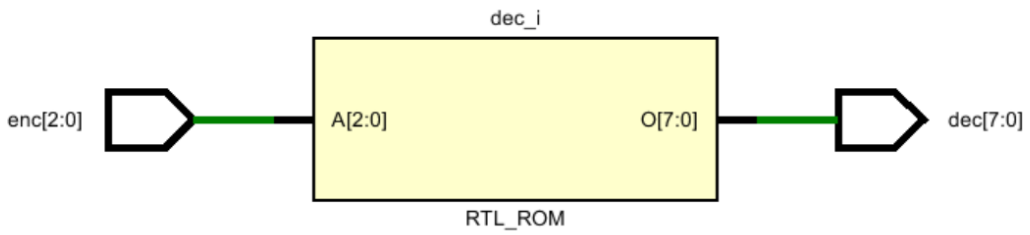


Παρατηρούμε ότι το παραπάνω διάγραμμα συμφωνεί πλήρως με τον τρόπο με τον οποίο θα υλοποιούσαμε έναν πολυπλέκτη στο χαρτί.

Στη συνέχεια υλοποιούμε την behavioral αρχιτεκτονική του κυκλώματος ορίζοντας τις τιμές εξόδου για κάθε μία από τις τιμές εισόδου όπως φαίνεται στον παρακάτω κώδικα:

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


entity mux3to8 is
    PORT ( enc :IN std_logic_vector(2 downto 0);
           dec :OUT std_logic_vector(7 downto 0)
           );
end mux3to8;

architecture Behavioral of mux3to8 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;
```

Το RTL διάγραμμα που παράγεται στο vivado μέσω του κώδικα είναι το παρακάτω:



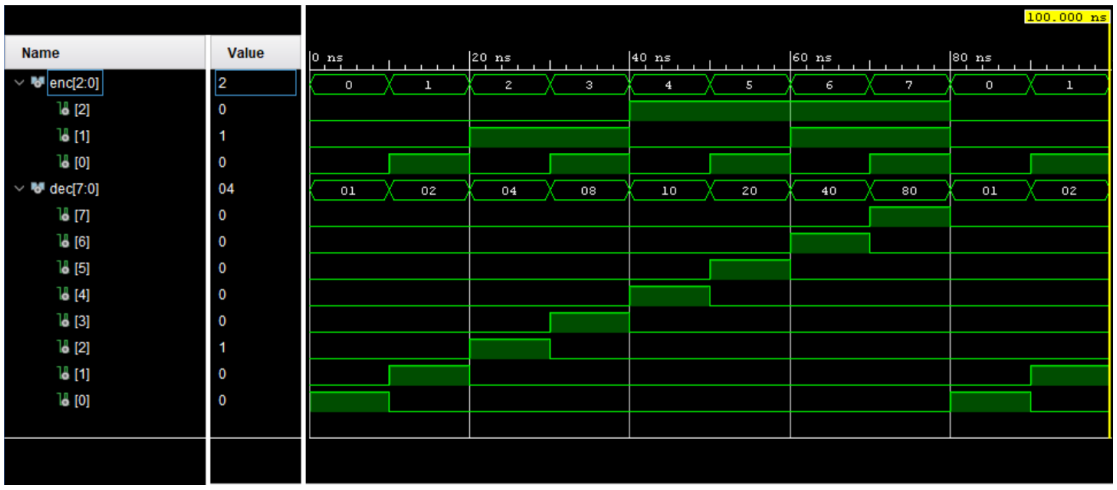
Παρατηρούμε ότι το RTL κύκλωμα αυτή τη φορά είναι διαφορετικό από το προηγούμενο, και αυτό είναι λογικό καθώς δεν έχουμε υλοποίηση με πύλες αλλά μόνο με τιμές εισόδου-εξόδου(έτσι το κύκλωμα μας είναι κάτι σα "μαυρό κουτί").

Το testbench που χρησιμοποιήθηκε και για τις δύο αρχιτεκτονικές δίνεται παρακάτω,όπου εξετάζουμε την έξοδο για κάθε πιθανή είσοδο:

Κώδικας:

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


entity TestBench is
end TestBench;

architecture Behavioral of TestBench is
    SIGNAL enc :  STD_LOGIC_VECTOR(2 downto 0);
    SIGNAL dec :  STD_LOGIC_VECTOR(7 downto 0);
begin

--UUT : entity work.mux3to8 port map(enc,dec);
UUT : entity work.mux3to8DF port map(enc,dec);

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


end Behavioral;
```

Οι κυματομορφές που παράγονται:



Παρατηρούμε ότι τα αποτελέσματα είναι όπως τα περιμέναμε.

Άσκηση 2 (B2):

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

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).

Το κύκλωμα έχει τη δυνατότητα τόσο δεξιάς όσο και αριστερής ολίσθησης. Ειδικότερα με βάση τη μεταβλητή left του κώδικα :

left = 1 : Αριστερή ολίσθηση,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.

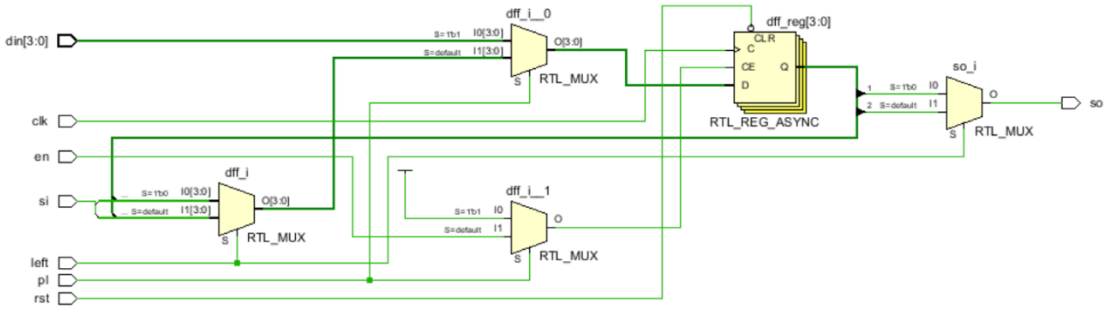
left = 0 : Δεξιά ολίσθηση,το περιεχόμενο του καταχωρητή ολισθαίνει κατά μια θέση προς το LSB(bit 0). Το LSB βγαίνει στην έξοδο so και η είσοδος si περνά στο MSB του καταχωρητή.

Αυτά γίνονται σε κάθε θετικό παλμό του ρολογιού clk. Η ασύγχρονη είσοδος rst (reset) μηδενίζει τα flip-flops του καταχωρητή ολίσθησης.

Ο κώδικας σε VHDL δίνεται παρακάτω:

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

entity shift is
port (
clk,rst,si,en,pl,left: in std_logic;
din: in std_logic_vector(3 downto 0);
so: out std_logic);

end shift;

architecture Behavioral of shift is
signal dff: std_logic_vector(3 downto 0);

begin
edge: process (clk,rst,left)
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 (left = '0') then
               dff<=si&dff(3 downto 1);

            elsif (left = '1') then
                      dff<=dff(2 downto 0)&si;
            end if;
          end if;
      end if;
    end process edge;

so <= dff(0) when left = '0' else dff(3);

end Behavioral;
```

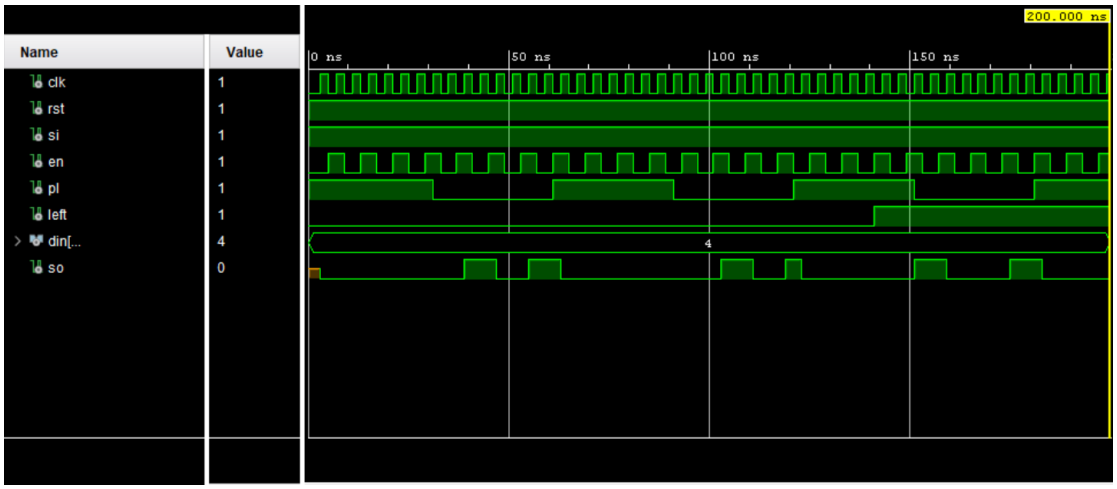
Το RTL σχηματικό που παράγεται :



Ακολουθεί το testbench που χρησιμοποιήθηκε για να επαληθευτεί η ορθή λειτουργία του κυκλώματος:

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

entity shift_tb is
end shift_tb;

architecture Behavioral of shift_tb is

SIGNAL    clk: std_logic := '0';
SIGNAL    rst: std_logic := '1';
SIGNAL    si: std_logic := '1';
SIGNAL    en: std_logic := '0';
SIGNAL    pl: std_logic := '1';
SIGNAL    left: std_logic := '0';
SIGNAL    din:  std_logic_vector(3 downto 0) := "0100";
SIGNAL    so:  std_logic;

begin

UUT : entity work.shift port
map(clk,rst,si,en,pl,left,din,so);

tb: PROCESS
    begin
    wait for 1 ns;
    left <= '1' after 140 ns;
    clk <= not clk after 2 ns;
    pl <= not pl after 30 ns;
    en <= not en after 4 ns;
    rst <= '0' after 50 ns;
    rst <= '1' after 51 ns;
    end process;

end Behavioral;
```

Στο συγκεκριμένο testbench έχουμε ορίσει το si σε τιμή 1(είσοδος αφού το LSB/MSB βγει στην έξοδο),το din είναι "0100".Επιπλέον,το rst ενεργοποιείται μια φορά στην αρχή ώστε να γίνει η αρχικοποίηση του σήματος dff και στη συνέχεια τίθεται σε τιμή ίση με 1 ,ώστε να αρχίσει η λειτουργία της ολίσθησης.Οι κυματομορφές που παράγονται δίνονται παρακάτω:



Το κύκλωμα λειτουργεί όπως θα αναμέναμε και η έξοδος μεταβάλεται σε κάθε θετικό παλμό του ρολογιού εαν pl = 0 και count_en = 1,ενώ για pl = 1 έχουμε φόρτωση του σήματος(και η έξοδος παραμένει ίση με το LSB/MSB σύμφωνα με το σήμα left).Επιπλέον,βλέπουμε τη λειτουργία τόσο για αριστερή όσο και δεξιά ολίσθηση.

Άσκηση 3 (B3) :

Ζητούμενο 1:

Κώδικας VHDL:

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

entity counter_updown is
port(
count_en, clk, resetn, updown : in std_logic;
cout                          : out std_logic;
sum                           : out std_logic_vector(2 downto 0));
end counter_updown;

architecture arch_bhv_updown of counter_updown is
signal Q   : std_logic_vector(2 downto 0);
begin
c_proc : process(count_en,clk,resetn) is
begin
  if (resetn='0') then Q <="000";
    else
      if (rising_edge(clk)) then
        if (count_en='1') then
            case (updown) is
          when '1' =>
            if (Q/=7) then Q <= Q+1;
              else Q <= "000";
```

```
          end if;
         when '0' =>
          if (Q/=0) then Q <= Q-1;
          else Q <= "111";
          end if;
         when others =>
          Q <="000";
         end case;
       end if;
     end if;
  end if;
end process c_proc;


sum <= Q;
with updown select
  cout <= Q(2) and Q(1) and Q(0) and count_en when '1',
        (not Q(2)) and (not Q(1)) and (not Q(0)) and count_en when '0',
        '0' when others;
end arch_bhv_updown;
```

Σχόλια:

Η είσοδος μηδενισμού resetn είναι ασύγχρονη, γι'αυτό και η συνθήκη --if (resetn='0')-- ελέγχεται πρώτη. Η είσοδος ενεργοποίησης count_en είναι σύγχρονη, γι'αυτό και η συνθήκη --if (count_en='1')-- ελέγχεται έπειτα από τη συνθήκη -- if (rising_edge(clk)) -- . Τώρα, αν το count_en='1' , θέλουμε ο μετρητής να μετράει πάνω ή κάτω, ανάλογα με την τιμή που δίνουμε στην είσοδο updown. Αυτό επιτυγχώνεται χρησιμοποιώντας την εντολή -- case (updown) is-- έπειτα από τη συνθήκη ελέγχου --if (count_en='1')-- και πέρνοντας δύο περιπτώσεις, όταν updown=1 το

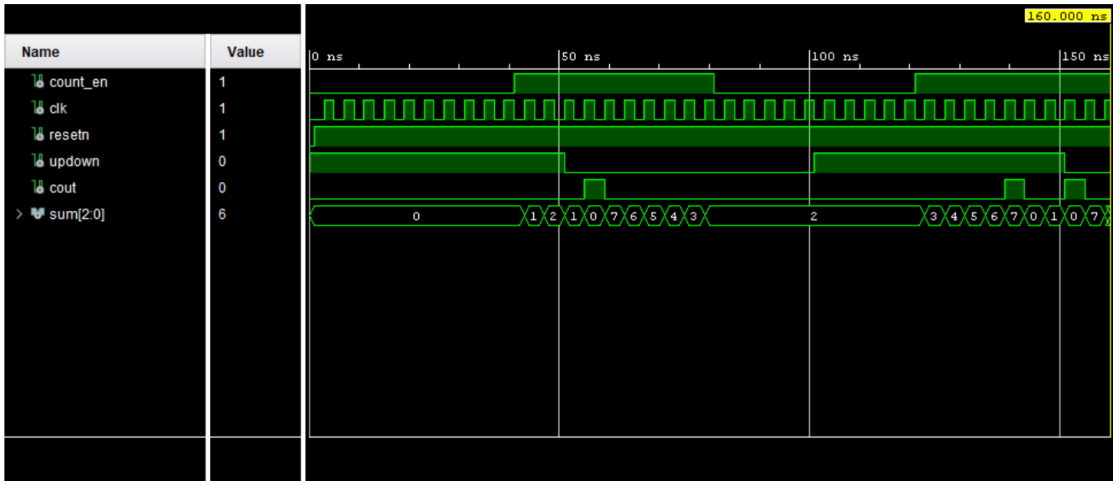
κύκλωμα να μετράει άνω, και όταν updown=0 το κύκλωμα να μετράει κάτω.

Testbench

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



entity testbenchA is
end testbenchA;

architecture Behavioral of testbenchA is

SIGNAL    count_en: std_logic := '0';
SIGNAL    clk: std_logic := '0';
SIGNAL    resetn: std_logic := '0';
SIGNAL    updown: std_logic := '1';
SIGNAL    cout: std_logic := '0';
SIGNAL    sum:  std_logic_vector(2 downto 0);

begin

UUT : entity work.counter_updown port map(count_en, clk, resetn, updown,cout,sum);

tb: PROCESS
   begin
   resetn <= '1';
```

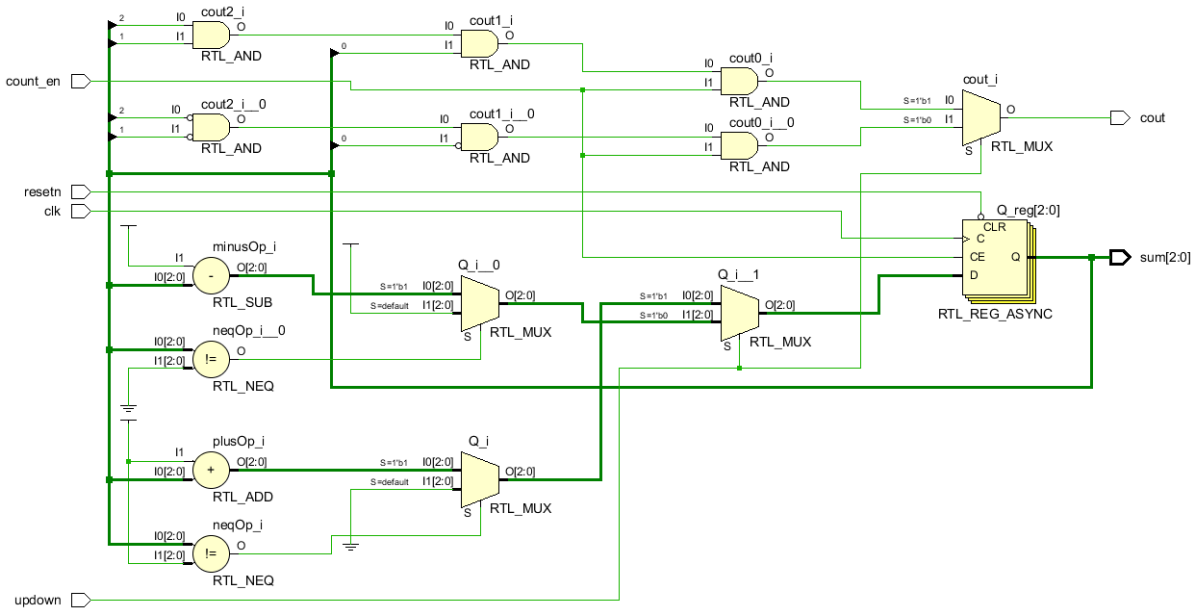
```
 wait for 1 ns;
resetn <= '0';
clk <= not clk after 2 ns;
count_en <= not count_en after 40 ns;
updown <= '0'  after 80 ns;
end process;


end Behavioral;
```

Οι κυματομορφές που παράγονται :



Παρατηρούμε την ορθή λειτουργία του κυκλώματος καθώς για count_en = 0 δεν έχουμε μέτρηση,ενώ όταν count_en = 1,έχουμε αύξηση για updown = 1 και μείωση για updown = 0,με το κρατούμενο να γίνεται 1 στα όρια 7 και 0 αντίστοιχα.

Rtl σχηματικό



Ζητούμενο 2:

Κώδικας VHDL:

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


entity versatile_counter is
port(
clk, count_en, resetn : in std_logic;
modulo                : in std_logic_vector(2 downto 0);
cout                  : out std_logic;
```

```
sum                 : out std_logic_vector(2 downto 0));
end versatile_counter;

architecture arch_vers of versatile_counter is
signal Q : std_logic_vector(2 downto 0);

begin
  vers_proc : process(clk, count_en, resetn, modulo) is
  begin
  if (resetn='0') then Q <= "000";
  else

     if (rising_edge(clk)) then
      if (count_en='1') then
        if (Q=modulo) then Q <="000";
          else Q <= Q+1;
        end if;

        if(Q=modulo-1) then Cout <= '1';
        else Cout <= '0';
        end if;
      end if;
    end if;
  end if;
  end process vers_proc;

sum <= Q;
end arch_vers;
```

Σχόλια:

Η είσοδος μηδενισμού resetn είναι ασύγχρονη, γι'αυτό και η συνθήκη --if (resetn='0')-- ελέγχεται πρώτη. Η είσοδος ενεργοποίησης count_en είναι σύγχρονη, γι'αυτό και η συνθήκη --if (count_en='1')-- ελέγχεται έπειτα από τη συνθήκη -- if (rising_edge(clk)) -- . Τώρα, αν το count_en='1' , θέλουμε ο μετρητής να μετράει πάνω έως ότου Q=modulo, και έπειτα το Q να μηδενίζεται στον επόμενο παλμό του ρολογιού και να ξεκινάει η μέτρηση πάλι από το 0. Αυτό επιτυγχάνεται χρησιμοποιώντας τις εντολές:

```
if (Q=modulo) then Q <="000";
else Q <= Q+1;
end if;
```

έπειτα από τη συνθήκη -- if (count_en='1')-- . Επίσης, θέλουμε Cout=1 όταν Q=modulo, διαφορετικά Cout=0. Για να γίνεται αυτό σύγχρονα με το ρολόι, χρησιμοποιούμε τις εντολές:

```
if(Q=modulo-1) then Cout <= '1';
else Cout <= '0';
end if;
```

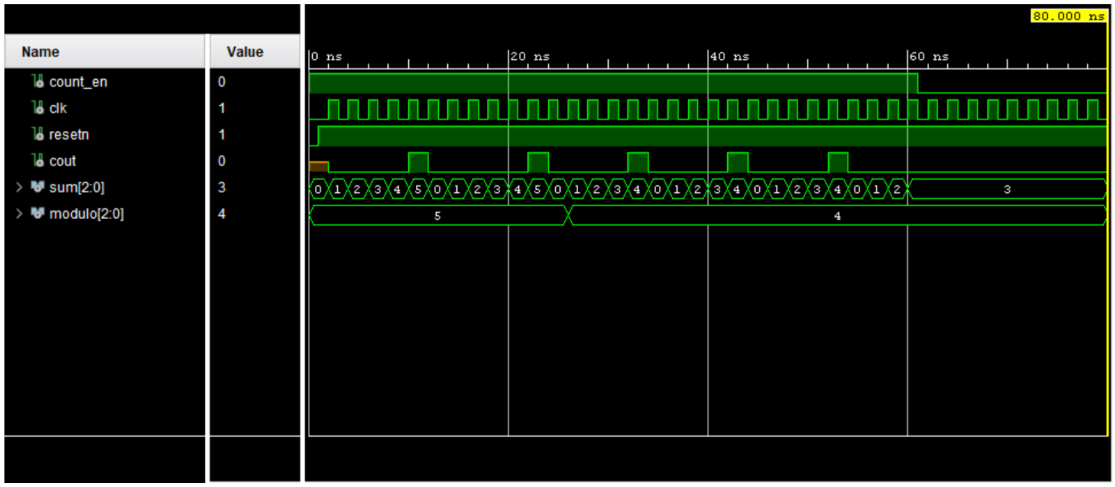
έπειτα από τις συνθήκες -- if (rising_edge(clk)) -- και -- if (count_en='1')-- .

Testbench

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.std_logic_unsigned.all;


entity testbenchB is

end testbenchB;


architecture Behavioral of testbenchB is
```

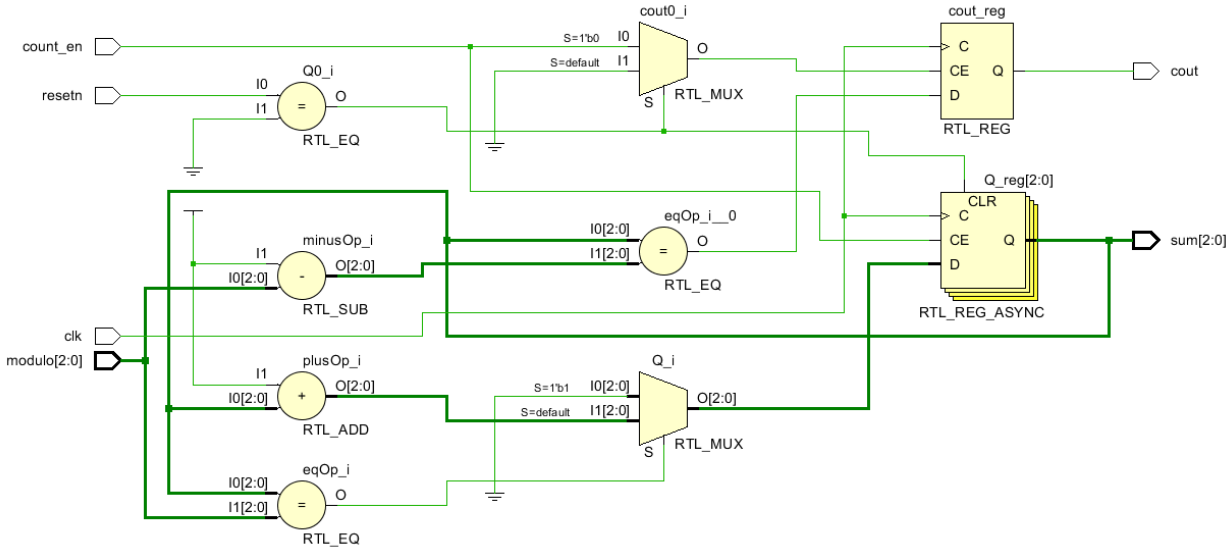
```
SIGNAL    count_en: std_logic := '1';
SIGNAL    clk: std_logic := '0';
SIGNAL    resetn: std_logic := '0';
SIGNAL    cout: std_logic := '0';
SIGNAL    sum:  std_logic_vector(2 downto 0);
SIGNAL    modulo: std_logic_vector(2 downto 0);
begin

UUT : entity work.versatile_counter port map(clk, count_en, resetn, modulo, cout, sum);

tb: PROCESS
     begin
    modulo <= "111";
    resetn <= '1';
    wait for 1ns;
    resetn <= '0';
    clk <= not clk after 1 ns;
    count_en <= not count_en after 55 ns;
    end process;

end Behavioral;
```

Οι κυματομορφές που παράγονται :



Παρατηρούμε ότι το κύκλωμα λειτουργεί όπως θα θέλαμε

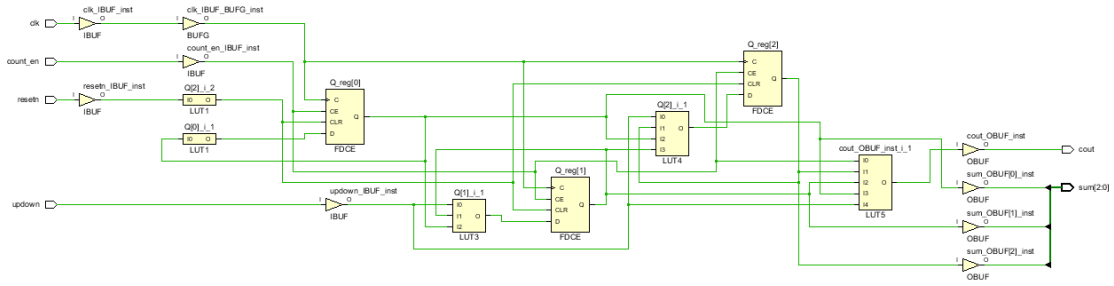
Rtl σχηματικό



Ζητούμενο 3:

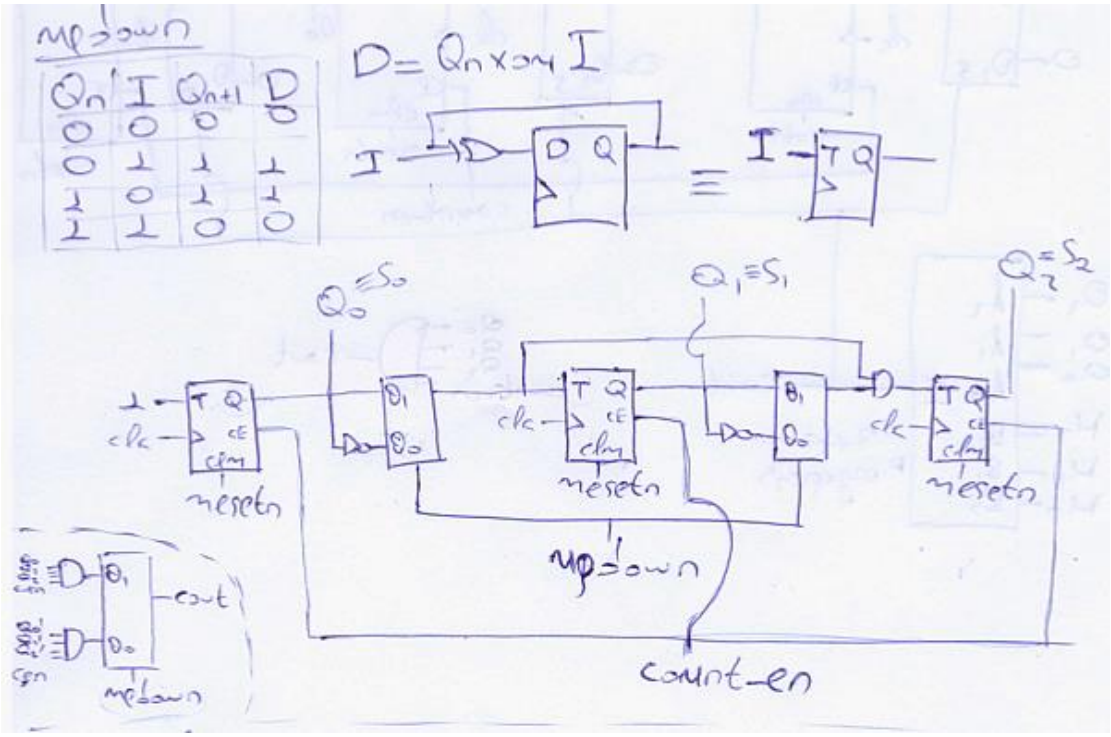
Μετρητής του 1ου ζητούμενου

Κύκλωμα που δίνει ο synthesizer:



<u>Κύκλωμα σχεδιασμένο με το χέρι:</u>

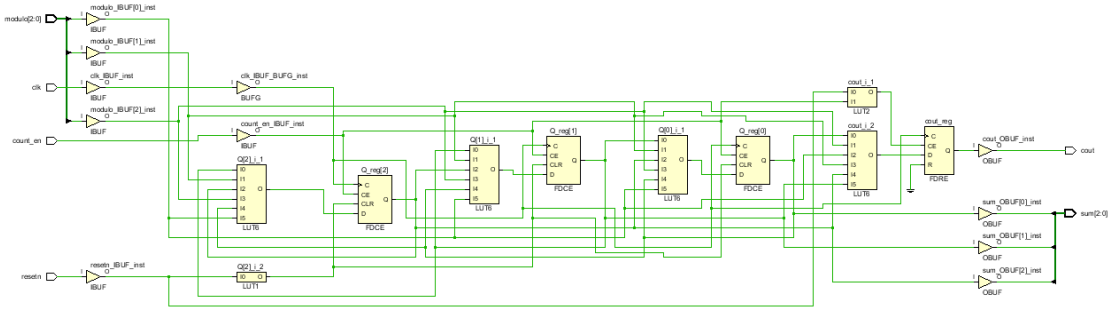
Μπορεί να σχεδιαστεί χρησιμοποιώντας T flip flops(κάθε ένα εκ των οποιών αποτελείται από ένα D flip flop και μια πύλη xor), πολυπλέκτες, πύλες and και αντιστροφείς.



Για να γνωρίζουμε με ακρίβεια, πιο απο τα δύο κυκλώματα, αυτό του synthesizer ή αυτό που έγινε με το χέρι, είναι απλούστερο, χρειάζεται να ξέρουμε την εσωτερίκη δομή των LUT που δίνει ο synthesizer. Έτσι θα μπορούμε να δούμε πιο από τα δύο κυκλώματα είναι οικονομικότερο σε επίπεδο πυλών.

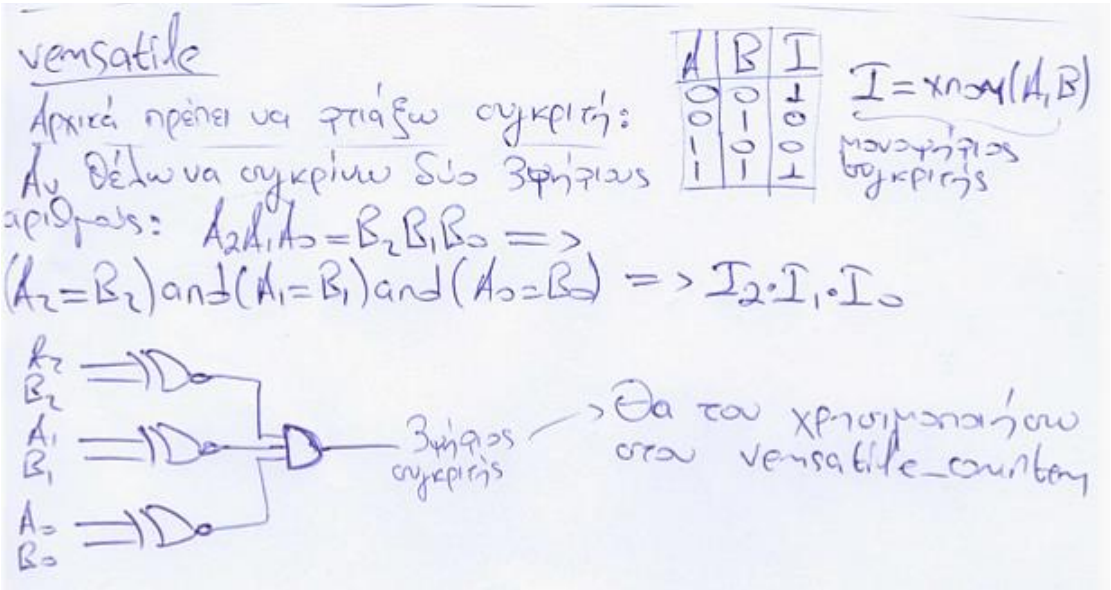
Μετρητής του 2ου ζητούμενου

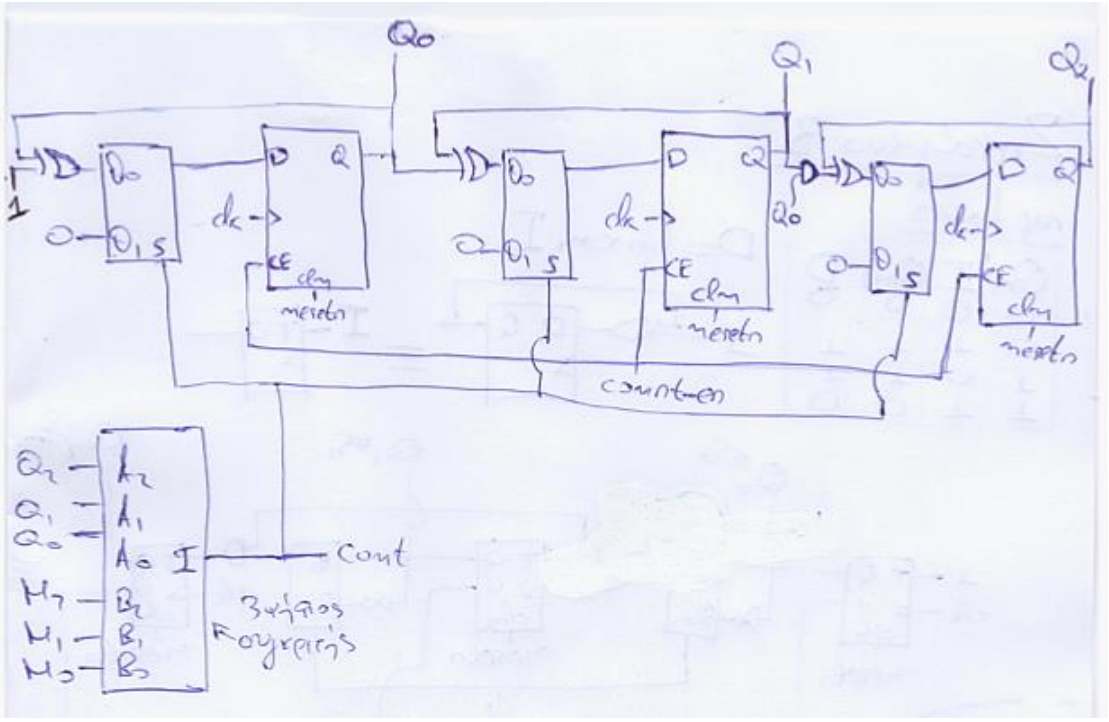
Κύκλωμα που δίνει ο synthesizer:



Κύκλωμα σχεδιασμένο με το χέρι:

Μπορεί να σχεδιαστεί χρησιμοποιώντας έναν 3ψήφιο συγκριτή, D flip flops, πολυπλέκτες, πύλες xor και and.





Για να γνωρίζουμε με ακρίβεια, πιο απο τα δύο κυκλώματα, αυτό του synthesizer ή αυτό που έγινε με το χέρι, είναι απλούστερο, χρειάζεται να ξέρουμε την εσωτερίκη δομή των LUT που δίνει ο synthesizer. Έτσι θα μπορούμε να δούμε πιο από τα δύο κυκλώματα είναι οικονομικότερο σε επίπεδο πυλών.