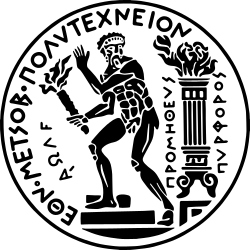
**Εθνικό Μετσόβιο Πολυτεχνείο** 

**Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών**

**Υπολογιστών**

Εξάμηνο: 8ο Ακ. Έτος: 2024- 2025

# «Ψηφιακά Συστήματα VLSI»

Χαράλαμπος Παπαδόπουλος (ΑΜ: 03120199)

Νικόλαος Παπακωνσταντόπουλος (ΑΜ: 03120069)

Ομάδα: 45

# Α\_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 <= "00000001";  **when** "001" =>  dec\_out <= "00000010";  **when** "010" =>  dec\_out <= "00000100";  **when** "011" =>  dec\_out <= "00001000";  **when** "100" =>  dec\_out <= "00010000";  **when** "101" =>  dec\_out <= "00100000";  **when** "110" =>  dec\_out <= "01000000";  **when** "111" =>  dec\_out <= "10000000";  **when** **others** =>  dec\_out <= "00000000";  **end** **case**;  **end** **process**;  **end** **behave**; |

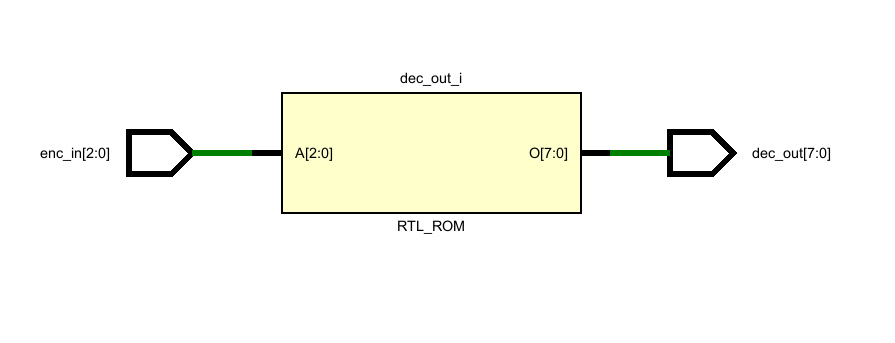
Και η dataflow:

|  |
| --- |
| **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.

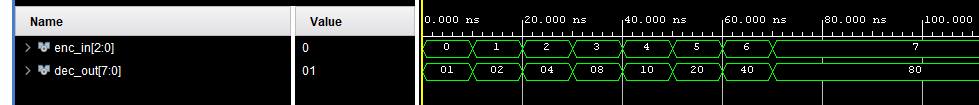
Δεδομένου αυτού του ορισμού, η παραπάνω dataflow αρχιτεκτονική είναι στην πραγματικότητα behavioral. Όμως, στο βιβλίο «free range VHDL» που μας προτάθηκε, αναγράφεται πως η εντολή with -select ανήκει στην κατηγορία dataflow.

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**; |



# B\_2

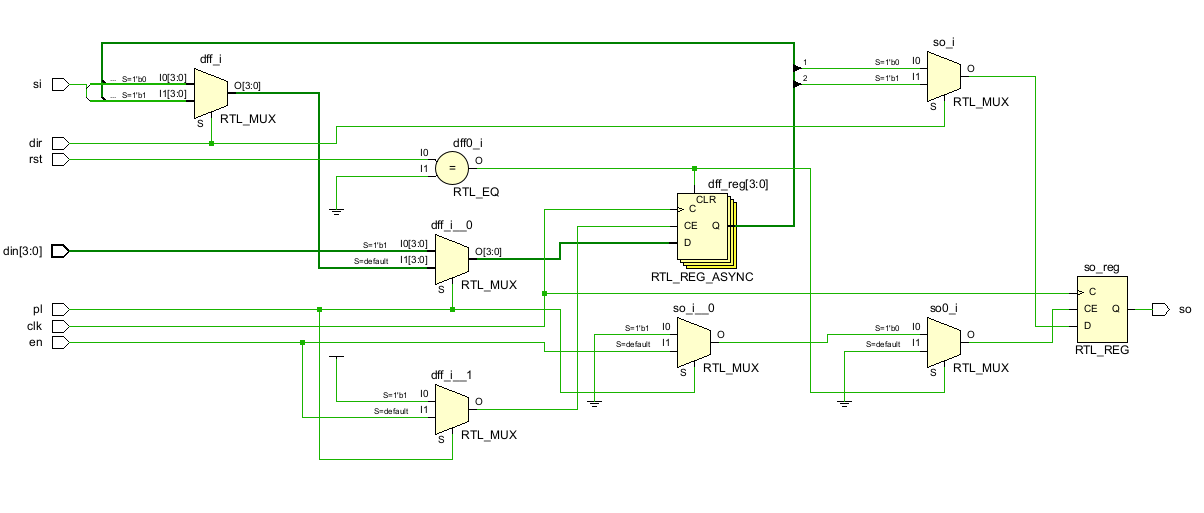
Ζητούμενο της άσκησης ήταν η υλοποίηση ενός καταχωρητή ολίσθησης των 4 bits με παράλληλη φόρτωση.

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

|  |
| --- |
| **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**; |

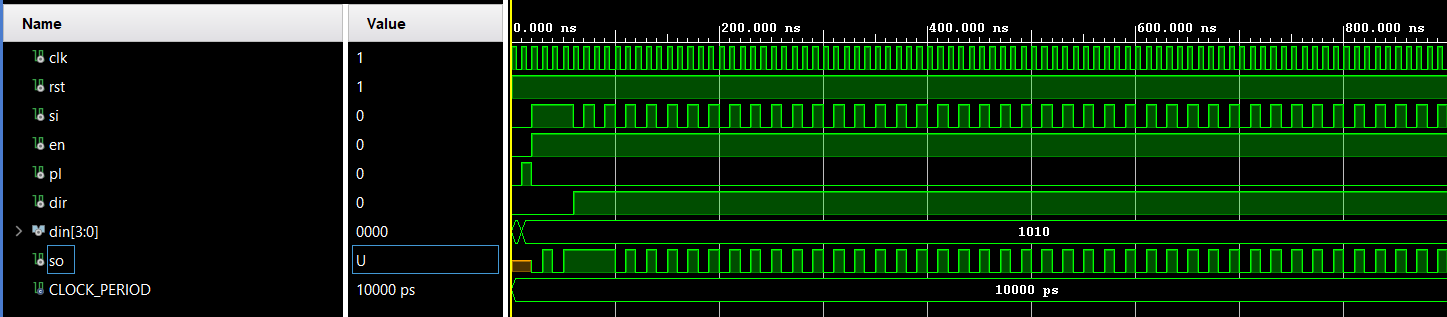
Στον δοσμένο κώδικα η ανάθεση του so γινόταν έξω από το process block, δηλαδή «έτρεχε» παράλληλα (concurrently) με αυτό, το οποίο οδηγούσε σε ανεπιθύμητη συμπεριφορά. Γι’ αυτό, το κάναμε comment out και το προσθέσαμε εντός του process block για να παίρνει την σωστή τιμή.

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**; |



# Β\_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

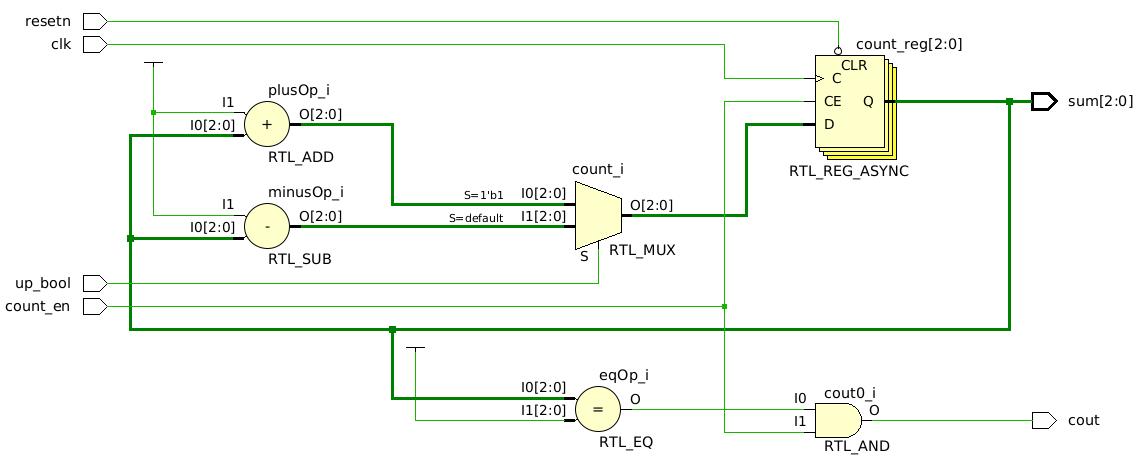
resetn <= '0'; -- reset to 0

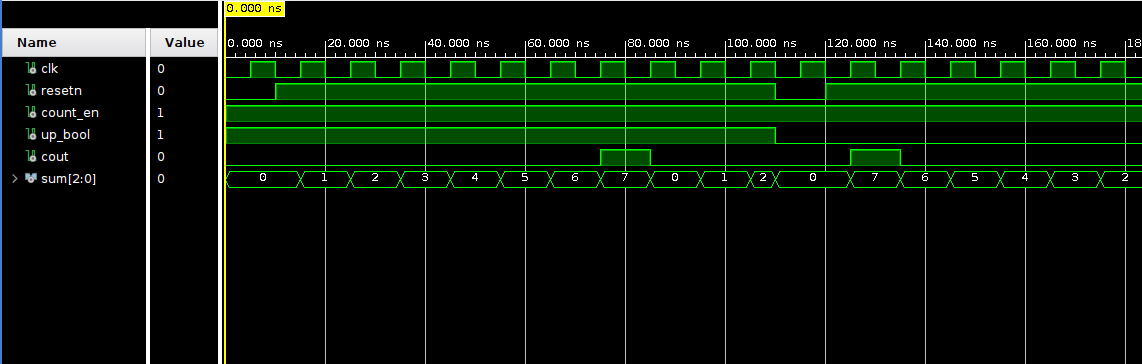
wait for 10 ns; -- for better visualization

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

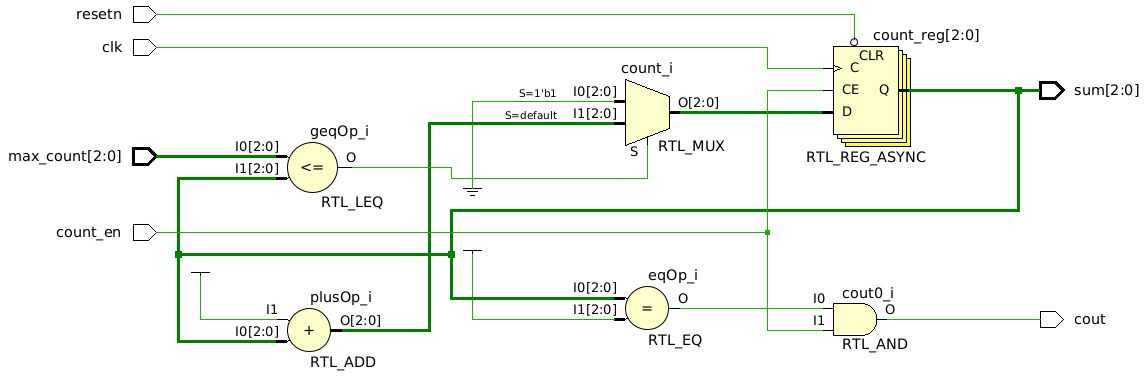
wait for 100 ns; -- count up to max\_count

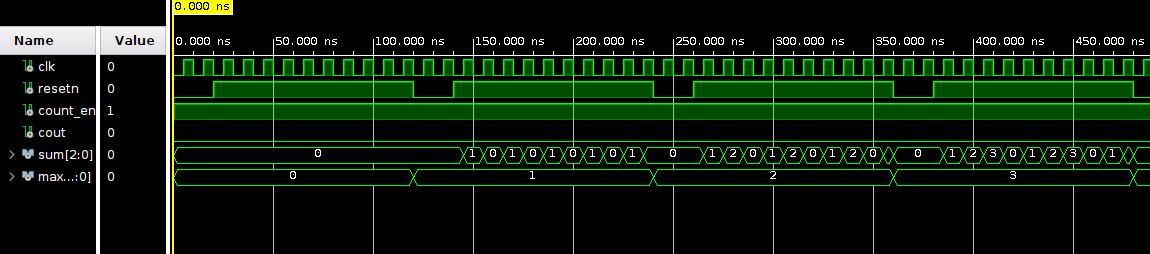
end loop;

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.