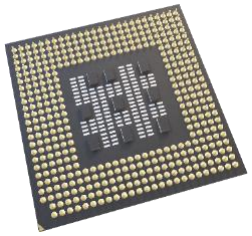
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ



ΨΗΦΙΑΚΗ ΣΧΕΔΙΑΣΗ ΙΙ

Παράδοση 5ης Εργαστηριακής Άσκησης

ΟΜΑΔΑ 4035 – 4123

ΓΑΖΟΣ ΔΗΜΗΤΡΗΣ, ΑΜ : 4035

ΜΠΟΓΡΗΣ ΓΙΩΡΓΟΣ, ΑΜ: 4123

ΜΑΡΤΙΟΣ 2021

Μέρος 1ο

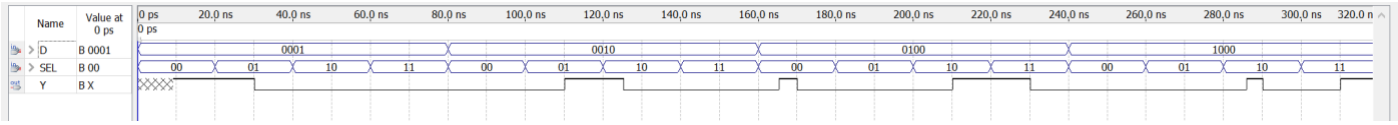
Ερώτημα 1ο

Σχεδίαση Πολυπλέκτη 4 σε 1 (MUX4_1)

Αρχικά σχεδιάζουμε τον πολυπλέκτη 4 σε 1 χρησιμοποιώντας VHDL κώδικα.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity MUX4_1 is
5          port( D:in std_logic_vector(3 downto 0);
6                SEL:in std_logic_vector(1 downto 0);
7                Y:out std_logic);
8
9  end MUX4_1;
10
11 architecture RTL4_1 of MUX4_1 is
12 begin
13
14    Y <= D(0) when SEL="00" else D(1) when SEL="01" else D(2) when SEL="10" else D(3);
15
16 end RTL4_1;
17
18
```

Ακολουθώντας την διαδικασία χρονικής εξομοίωσης αποδεικνύουμε ότι ο πολυπλέκτης λειτουργεί σωστά, σύμφωνα και με τον πίνακα αληθείας.



Βρίσκουμε την μέγιστη χρονική καθυστέρηση του πολυπλέκτη σε Slow Model, όπου πράγματι είναι λίγο μεγαλύτερη από τα 10ns.

	Input Port	Output Port	RR	RF	FR	FF
1	D[0]	Y	9.964	9.964	9.964	9.964
2	D[1]	Y	10.094	10.094	10.094	10.094
3	D[2]	Y	5.500			5.500
4	D[3]	Y	9.481			9.481
5	SEL[0]	Y	10.156	10.156	10.156	10.156
6	SEL[1]	Y	6.046	6.046	6.046	6.046

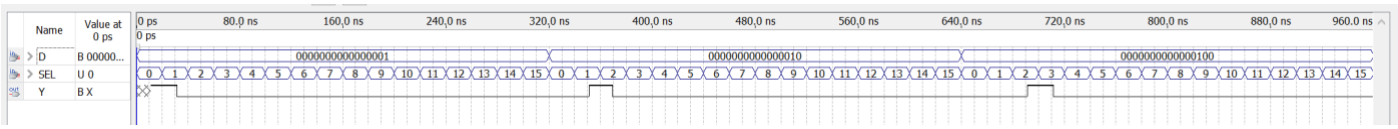
Ερώτημα 2ο

Σχεδίαση Πολυπλέκτη 16 σε 1 (MUX16_1)

Σχεδιάζουμε έναν πολυπλέκτη 16 σε 1 χρησιμοποιώντας VHDL κώδικα και τον πολυπλέκτη 4 σε 1 που σχεδιάσαμε στο προηγούμενο ερώτημα.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity MUX16_1 is
5          port( D:in std_logic_vector(15 downto 0);
6                SEL:in std_logic_vector(3 downto 0);
7                Y:out std_logic );
8
9  end MUX16_1;
10
11 architecture RTL of MUX16_1 is
12 component MUX4_1
13       port( D:in std_logic_vector(3 downto 0);
14             SEL:in std_logic_vector(1 downto 0);
15             Y:out std_logic);
16
17 end component;
18
19 signal F:std_logic_vector(3 downto 0);
20 begin
21          u0:MUX4_1 port map(D => D(3 downto 0),       SEL => SEL(1 downto 0),Y => F(0));
22          u1:MUX4_1 port map(D => D(7 downto 4),       SEL => SEL(1 downto 0),Y => F(1));
23          u2:MUX4_1 port map(D => D(11 downto 8),      SEL => SEL(1 downto 0),Y => F(2));
24          u3:MUX4_1 port map(D => D(15 downto 12),     SEL => SEL(1 downto 0),Y => F(3));
25          u4:MUX4_1 port map(D => F(3 downto 0),       SEL => SEL(3 downto 2),Y => Y);
26
27 end RTL;
```

Ακολουθώντας την διαδικασία χρονικής εξομοίωσης αποδεικνύουμε ότι ο πολυπλέκτης λειτουργεί σωστά, σύμφωνα και με τον πίνακα αληθείας.



Βρίσκουμε την μέγιστη χρονική καθυστέρηση του πολυπλέκτη σε Slow Model, όπου πράγματι περίπου στα 12.25ns.

	Input Port	Output Port	RR	RF	FR	FF
1	D[0]	Y	11.859	11.859	11.859	11.859
2	D[1]	Y	11.451			11.451
3	D[2]	Y	11.673	11.673	11.673	11.673
4	D[3]	Y	11.170			11.170
5	D[4]	Y	11.852	11.852	11.852	11.852
6	D[5]	Y	12.155	12.155	12.155	12.155
7	D[6]	Y	8.937			8.937
8	D[7]	Y	11.743			11.743
9	D[8]	Y	11.621	11.621	11.621	11.621
10	D[9]	Y	11.452			11.452
11	D[10]	Y	11.756			11.756
12	D[11]	Y	11.090			11.090
13	D[12]	Y	11.636	11.636	11.636	11.636
14	D[13]	Y	11.813			11.813
15	D[14]	Y	11.957			11.957
16	D[15]	Y	11.809			11.809
17	SEL[0]	Y	12.247	12.247	12.247	12.247
18	SEL[1]	Y	9.476	9.476	9.476	9.476
19	SEL[2]	Y	11.246	11.246	11.246	11.246
20	SEL[3]	Y	11.221	11.707	11.707	11.221

Ερώτημα 3ο

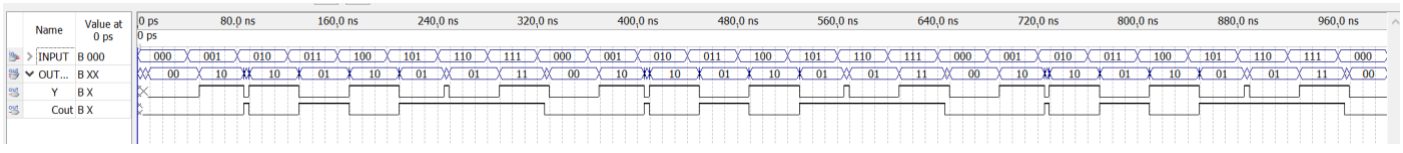
Σχεδίαση Αθροιστή 8 δυαδικών ψηφίων(adder8)

Σχεδίαση Πλήρη αθροιστή(FULL ADDER)

Αρχικά σχεδιάζουμε έναν πλήρη Αθροιστή σε VHDL κώδικα.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity FA is
5           port ( A,B,Cin:in std_logic;
6                  Y,Cout:out std_logic);
7  end FA;
8
9  architecture RTL of FA is
10 begin
11        Y <= A xor B xor Cin;
12        Cout <= (A and B) or (A and Cin) or (B and Cin);
13 end RTL;
14
15
```

Ακολουθώντας την διαδικασία εξομοίωσης αποδεικνύουμε ότι ο πλήρης αθροιστής λειτουργεί σωστά.

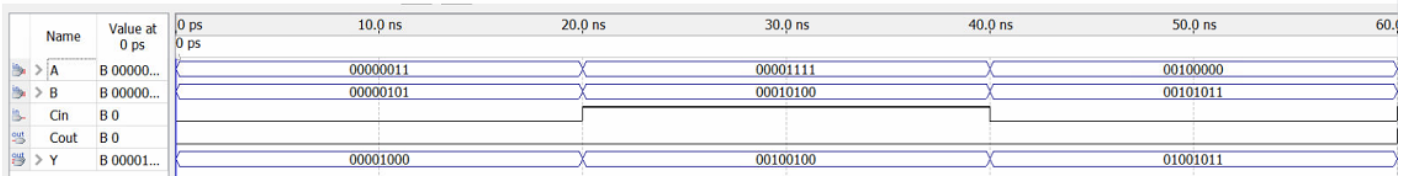


Σχεδίαση Αθροιστή ριπής (Adder8)

Στη συνέχεια, σχεδιάζουμε τον Αθροιστή ριπής χρησιμοποιώντας VHDL κώδικα και χρησιμοποιώντας 8 πλήρη αθροιστές που σχεδιάσαμε νωρίτερα.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity Adder8 is
5           port(    A,B:     in std_logic_vector(7 downto 0);
6                    Cin:     in std_logic;
7                    Y:       out std_logic_vector(7 downto 0);
8                    Cout:    out std_logic     );
9
10 end Adder8;
11
12 architecture RTL1 of Adder8 is
13 component FA
14       port (A,B,Cin: in std_logic;
15             Y,Cout: out std_logic);
16 end component;
17
18 signal C: std_logic_vector(A'length downto 0);
19
20 begin
21 C(0) <= Cin;
22 Cout <= C(A'length);
23 gen : for j in A'range generate
24    fa0: FA port map (A => A(j), B => B(j), Cin => C(j), Y => Y(j), Cout =>C(j+1));
25 end generate;
26
27 end RTL1;
28
```

**Ενώ στην πρώτη δοκιμή λειτούργησε κανονικά, κάναμε το λάθος να μην κρατήσουμε αρχείου από το ModelSim. Λόγω ελλειψης χρονου δεν καταφεραμε να βρουμε το προβλημα.



Μέρος 2ο

Ερώτημα 1ο

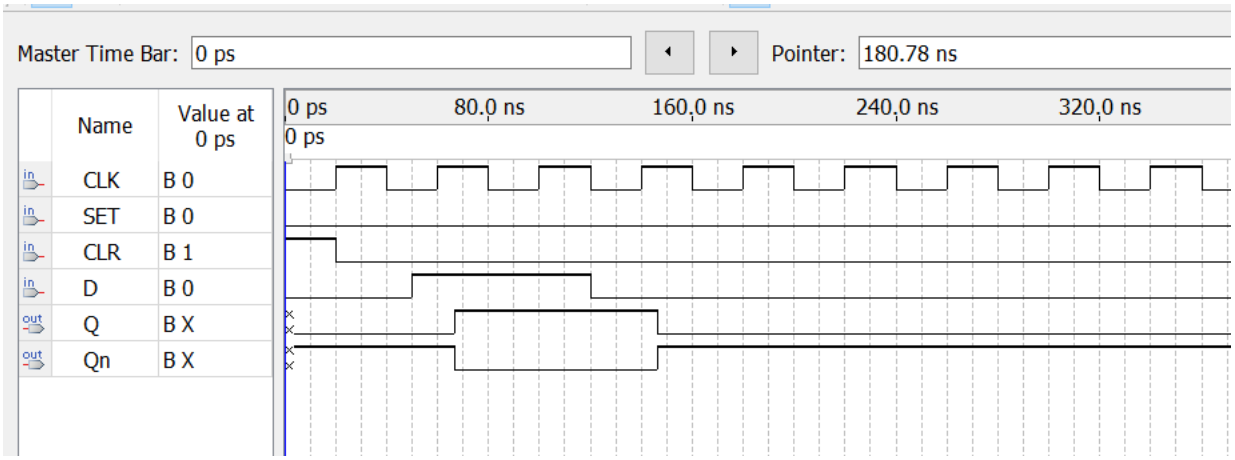
Αρχικά σχεδιάζουμε ένα D FLIP FLOP με τη χρήση VHDL κώδικα όπως και μας δίνεται.

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   entity D_ff is
4         port( CLK,D,CLR,SET: in std_logic;
5               Q,Qn: out std_logic  );
6   end D_ff;
7
8   architecture RTL of D_ff is
9         signal DFF:std_logic;
10
11  begin
12        seq0:process(CLK,CLR,SET)
13        begin
14              if (CLR='1') then DFF <= '0';
15              elsif(SET='1') then DFF <= '1';
16              elsif(CLK'event and CLK='1') then DFF <=D;
17              end if;
18
19        end process;
20        Q <= DFF; Qn <= not DFF;
21  end RTL;
22
23
```

Υπόμνημα

Το CLK είναι το ρολόι,D η είσοδος δεδομένων, το CLR και το SET βρίσκονται ασύγχρονη μηδένιση και θέση αντίστοιχα, και δεν επηρεάζονται από το ρολόι. Αυτό το συμπεραίνουμε διότι είναι εκτός και νωρίτερα του if που περιέχει το ρολόι.

Ακολουθώντας την διαδικασία εξομοίωσης αποδεικνύουμε ότι το D flip flop λειτουργεί σωστά.



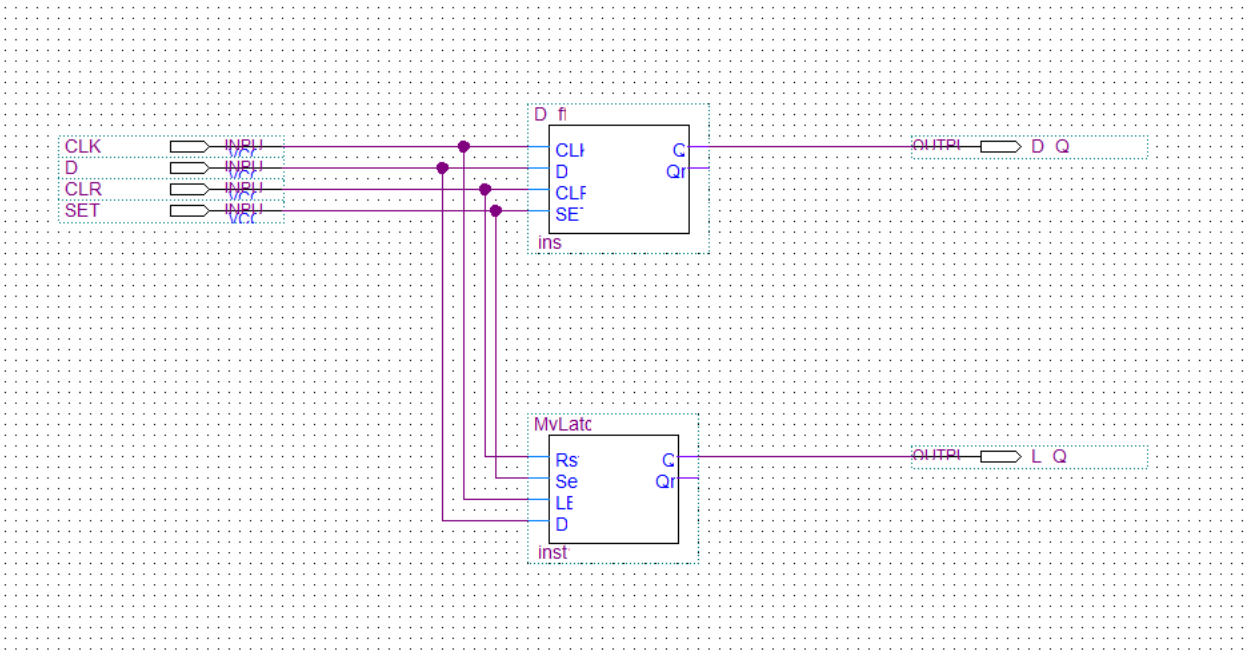
Επίσης η μέγιστη συχνότητα στο Fmax Summary ειναι:

****Μας βγάζει No path to report.**

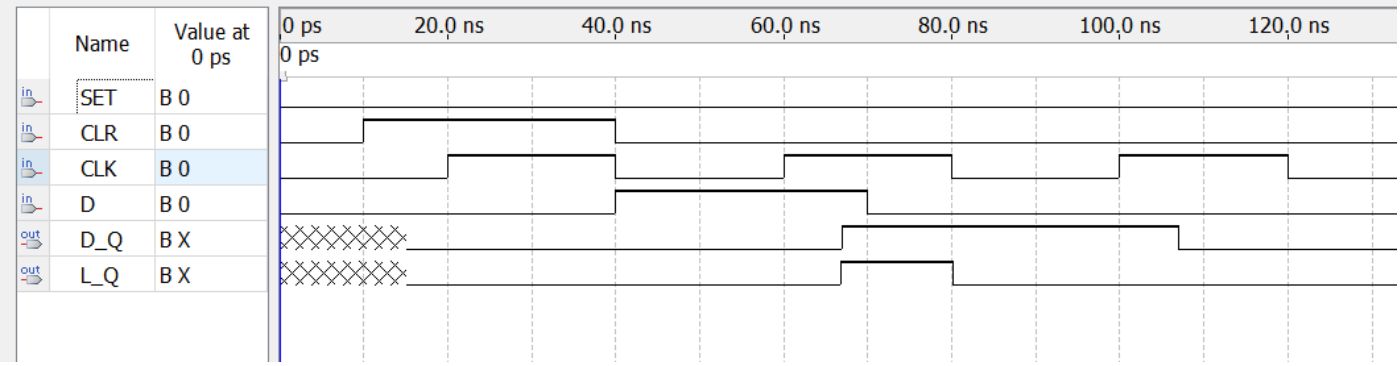
Έπειτα, σχεδιάζουμε ένα Latch με τη χρήση VHDL κώδικα όπως και μας δίνεται.

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

entity MyLatch is
        port (   Rst,Set,LE,D : in std_logic;
                 Q,Qn: out std_logic );

end MyLatch;

architecture RTL of MyLatch is
signal FF: std_logic;
begin
   seq0:process (Rst,Set,D,LE)
   begin
        if Rst='1' then FF <='0';
        elsif Set='1' then FF <='1';
        elsif LE='1' then FF <= D;
        end if;
   end process;
   Q <= FF;
   Qn <= not FF;
end RTL;
```

Μετά δημιουργούμε το σχηματικό με τη χρήση του D flip flop και latch που σχεδιάσαμε νωρίτερα.



Ακολουθώντας την διαδικασία εξομοίωσης αποδεικνύουμε ότι το σχηματικό λειτουργεί σωστά.

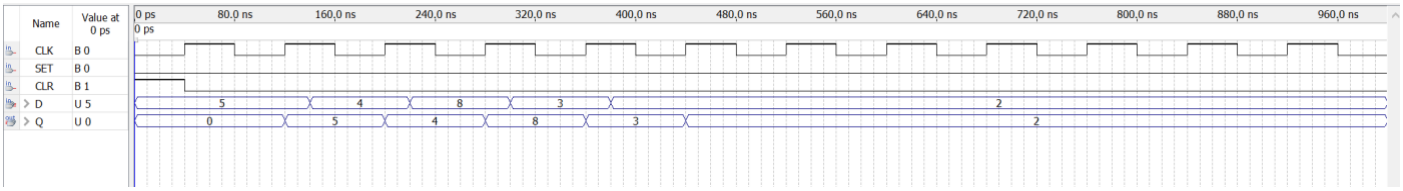


Ερώτημα 2ο

Αρχικά, σχεδιάζουμε με τη χρήση VHDL κώδικα ένα καταχωρητή που στηρίζεται στην λογική των D flip flop. Τροποποιώντας κατάλληλα των κώδικα που συντάξαμε νωρίτερα, προσθέτοντας 8 εισόδους, 8 εξόδους ενώ τα CLR,CLK,SET παραμένουν ως έχουν. Εφόσον έχουμε 8 εισόδους, θέλουμε το process να επαναληφθεί 8 φορές και το επιτυγχάνουμε μέσω των σημάτων DFF.

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
entity Reg8 is
	generic(N : integer:=8);

	port( CLK,CLR,SET: in std_logic;
		D: in std_logic_vector(N-1 downto 0);
		Q: out std_logic_vector(N-1 downto 0)  );
end Reg8;

architecture RTL1 of Reg8 is
	signal DFF:std_logic_vector(N -1 downto 0);

begin
	seq0:process(CLK,CLR,SET)
	begin
		if (CLR='1') then DFF <= (N-1 downto 0 => '0');
		elsif(SET='1') then DFF <= (N-1 downto 0 => '1');
		elsif(CLK'event and CLK='1') then DFF <=D;
		end if;

	end process;
	Q <= DFF;
end RTL1;

```

Ακολουθώντας την διαδικασία εξομοίωσης αποδεικνύουμε ότι ο καταχωρητής λειτουργεί σωστά.



Η μέγιστη συχνοτητα ρολογιου στο Fmax summary είναι:

Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	518.4 MHz	420.17 MHz	clk	limit due to minimum period restriction (max I/O toggle rate)

Ερώτημα 3ο

Σχέδιαζουμε έναν μετρητή των 8 δυαδικών ψηφίων με δυνατότητες μέτρησης προς τα πάνω.

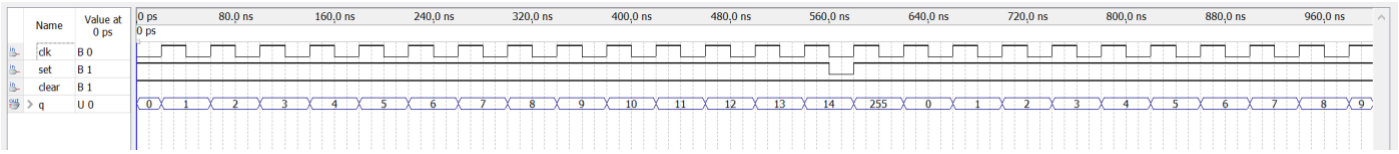
```
1
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.numeric_std.all;
5
6  entity count is
7     port
8     (
9        clk      : in std_logic;
10       set   : in std_logic;
11       clear    : in std_logic;
12       q     : out integer range 0 to 255
13    );
14 end entity;
15
16 architecture rtl of count is
17 begin
18    process (clk)
19       variable   cnt : integer range 0 to 255;
20    begin
21       if (rising_edge(clk)) then
22          if clear = '0' then
23             -- Reset the counter to 0
24             cnt := 0;
25          elsif set = '0' then
26             -- Set the counter to 255
27             cnt := 255;
28          else
29             -- Increment the counter
30             cnt := cnt + 1;
31          end if;
32       end if;
33
34       -- Output the current count
35       q <= cnt;
36    end process;
37
38 end rtl;
39
```

**Προσπαθήσαμε να εκτελέσουμε χρονική εξομοίωση χρησιμοποιώντας test bench, όμως δεν καταφέραμε να το λειτουργήσουμε. Παρακάτω παραθέτουμε τον κώδικα του test bench.

```
1  LIBRARY ieee  ;
2  LIBRARY std  ;
3  USE ieee.NUMERIC_STD.all  ;
4  USE ieee.std_logic_1164.all  ;
5  USE ieee.std_logic_textio.all  ;
6  USE ieee.std_logic_unsigned.all  ;
7  USE std.textio.all  ;
8  ENTITY counter_tb  IS
9  END ;
10
11 ARCHITECTURE counter_tb_arch OF counter_tb IS
12   SIGNAL set   :  STD_LOGIC  ;
13   SIGNAL q   :  INTEGER  ;
14   SIGNAL clk   :  STD_LOGIC  ;
15   SIGNAL clear   :  STD_LOGIC  ;
16   COMPONENT count
17     PORT (
18       set  : in STD_LOGIC ;
19       q  : out INTEGER ;
20       clk  : in STD_LOGIC ;
21       clear  : in STD_LOGIC );
22   END COMPONENT ;
23 BEGIN
24   DUT  : count
25     PORT MAP (
26       set   => set  ,
27       q   => q  ,
28       clk   => clk  ,
29       clear   => clear   ) ;
30
31
32
33 -- "Clock Pattern" : dutyCycle = 50
34 -- Start Time = 0 ns, End Time = 1 us, Period = 40 ns
35   Process
36    Begin
37     clk  <= '0'  ;
38    wait for 20 ns ;
39 -- 20 ns, single loop till start period.
40    for Z in 1 to 24
```

```
41    loop
42        clk  <= '1'  ;
43       wait for 20 ns ;
44        clk  <= '0'  ;
45       wait for 20 ns ;
46  -- 980 ns, repeat pattern in loop.
47    end  loop;
48     clk  <= '1'  ;
49    wait for 20 ns ;
50  -- dumped values till 1 us
51    wait;
52   End Process;
53
54
55  -- "Constant Pattern"
56  -- Start Time = 560 ns, End Time = 1 us, Period = 0 ns
57    Process
58     Begin
59      set  <= '1'  ;
60     wait for 560 ns ;
61      set  <= '0'  ;
62     wait for 20 ns ;
63      set  <= '1'  ;
64     wait for 420 ns ;
65  -- dumped values till 1 us
66     wait;
67   End Process;
68
69
70  -- "Constant Pattern"
71  -- Start Time = 0 ns, End Time = 1 us, Period = 0 ns
72    Process
73     Begin
74      clear  <= '1'  ;
75     wait for 1 us ;
76  -- dumped values till 1 us
77     wait;
78   End Process;
79  END;
80
```

Για να επιβεβαιώσουμε την λειτουργεία του μετρητή, δημιουργήσαμε εν τέλει ένα «κλασσικό» waveform.


Name
Value at 0 ps
0 ps
clk B 0
set B 1
clear B 1
q U 0
0 ps
80,0 ns
160,0 ns
240,0 ns
320,0 ns
400,0 ns
480,0 ns
560,0 ns
640,0 ns
720,0 ns
800,0 ns
880,0 ns
960,0 ns
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 255 0 1 2 3 4 5 6 7 8 9