CSc 21100 - Fundamentals of Computer Systems

Instructor: Prof. Zheng Peng

Project **02**

Name **Mohamed Kharma**

Section **CC2**

Date **3/2021**

IMPORTANT!

Please follow the **submission guidelines** below or your submission will be rejected.

1. You are expected to submit two files (a PDF lab report and a ZIP file containing the source files) to Blackboard. Note that both files must be upload in the same submission attempt.
2. Each task must have its own source file(s). For VHDL assignments, all source files of this assignment must be in a single Xilinx VHDL project.
3. For VHDL assignments, you must use the export function of the Xilinx ISE Design Suite to create the ZIP file properly. Please refer to Blackboard -> Content -> Assignments -> Exporting_VHDL_project_files
4. Naming convention:

   Report: "FirstName_LastName_Project_**XX**_CC**Y**.pdf"*

   Project: "FirstName_LastName_Project_**XX**_CC**Y**.zip"*

   *Replace "XX" and "Y" with the actual project number (two digits) and section number, respectively.
5. After the due day, all submissions are final. You cannot change it for any reasons. Double check before you make the submission.

Task 1: Inhibit gate (5 points)

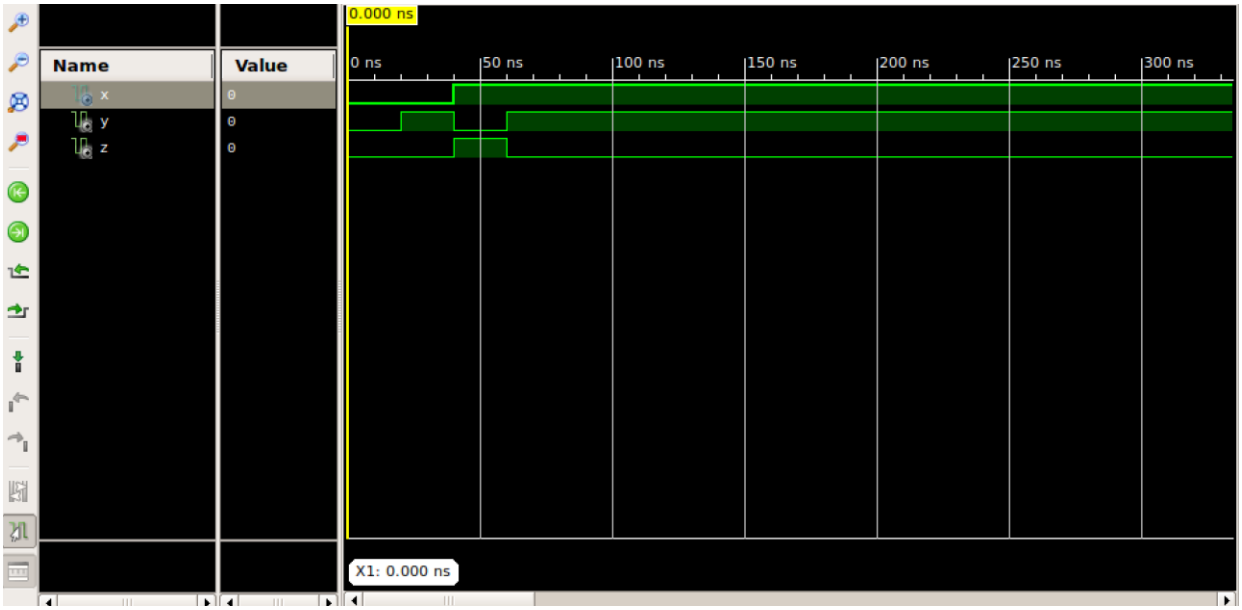
1.1 – The implemented code with explanation:

```
entity Inhibit is -- Name of the gate is declared as "Inhibit"
  port (X,Y: in BIT; -- Input X,Y as bits
        Z:   out BIT); -- Output Z as bit
end Inhibit; -- Entity has been fully declared

architecture Inhibit_arch of Inhibit is  -- Architecture "Inhibit_arch" is created for entity "Inhibit"
begin
  Z <= '1' when X='1' and Y='0' else '0'; -- Return true for Z if and only if X=1 and Y=0
end Inhibit_arch;    -- Architecture is ended.
```

1.2 – Test Bench code with explanation:

```
LIBRARY ieee;   --Calling library ieee.
USE ieee.std_logic_1164.ALL;    --we are using all the definitions from library ieee package.

ENTITY inhibit test IS  -- Name of the gate test is declared as "Inhibit test"
END inhibit_test;   -- The entity has been declared

ARCHITECTURE structural OF inhibit_test IS  -- Architecture "structural" is created for entity "Inhibit_test"
   COMPONENT inhibit   -- Concurrent statement of type component with the name "inhibit."
   PORT (X : IN  BIT;   -- X is assigned as an input
         Y : IN  BIT;   -- Y is assigned as an input
         Z : OUT BIT);  -- Z is assigned as an Output
   END COMPONENT;
   signal X, Y, Z: BIT; -- Local signal is used to create the names X,Y,Z of type Bit.
BEGIN
  uut: inhibit PORT MAP (X, Y, Z); -- Using the component inhibit with signals X,Y,Z to test UUT.
  stim_proc: process
  begin
     -- insert stimulus here
     X <= '0'; Y <= '0';  -- provide input combinations as X=0 and Y=0.
     wait for 20 ns;  -- stop for 20 nano seconds to be able to see the output.
     X <= '0'; Y <= '1';  -- provide input combinations as X=0 and Y=1.
     wait for 20 ns;  -- stop for 20 nano seconds to be able to see the output.
     X <= '1'; Y <= '0';  -- provide input combinations as X=0 and Y=1.
     wait for 20 ns;  -- stop for 20 nano seconds to be able to see the output.
     X <= '1'; Y <= '1';  -- provide input combinations as X=0 and Y=1.
     wait; -- To stop the process
   end process;
END;    -- Architecture is ended.
```

1.3 - Simulation Results:



Explanation:

Looking at the picture we can see that:

1. When the input of X = ‘0’ and Y = ‘0’, The output of Z = ‘0’
2. When the input of X = ‘0’ and Y = ‘1’, The output of Z = ‘0’
3. When the input of X = ‘1’ and Y = ‘0’, The output of Z = ‘1’
4. When the input of X = ‘1’ and Y = ‘1’, The output of Z = ‘0’

Task 2: Prime-number detector (7 points)

2.1 – The implemented code with explanation:

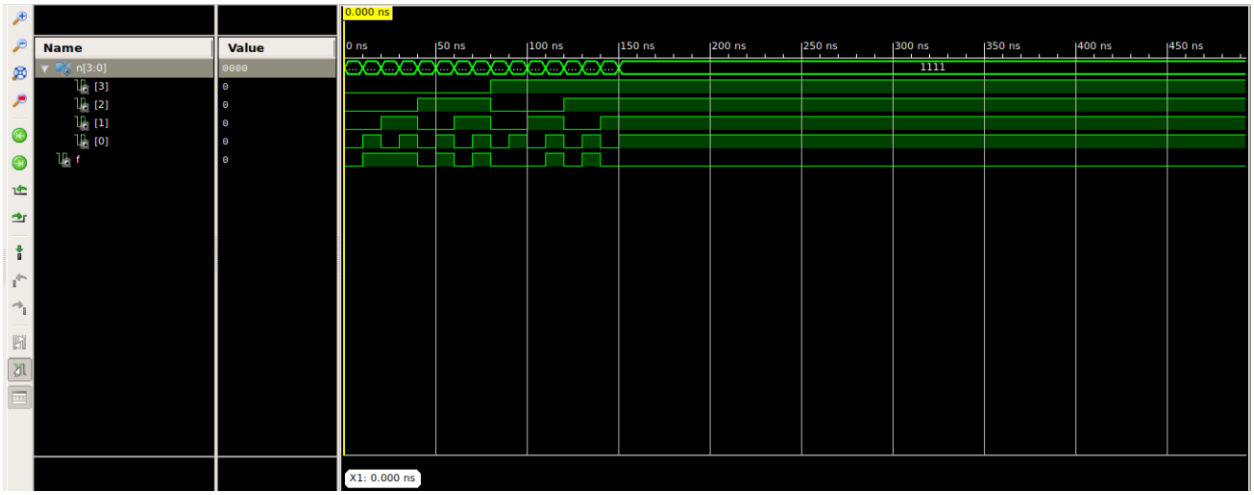
```
library IEEE;   --Calling library IEEE.
use IEEE.std_logic_1164.all; --we are using all the definitions from library IEEE package.
library unisim;  --Calling library unisim.
use unisim.vcomponents.all; --we are using all the definitions from library unisim package.

entity prime is -- Name of the gate is decleared as "prime"
   port (N: in STD_LOGIC_VECTOR (3 downto 0);  -- N is input of type vector and it has 4 element
         F: out STD_LOGIC);    --F is output of type std_logic.
end prime;

architecture prime1_arch of prime is -- Architecture "prime1_arch" is created for entity "prime"
signal N3_L, N2_L, N1_L: STD_LOGIC; -- Local signal is used to create the names N3_L, N2_L and N1_L of type
STD_LOGIC as connections between the gates.
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC; -- Local signal is used tp create the names N3L_N0,
N3L_N2L_N1, N2L_N1_N0 and N2_N1L_N0 of type STD_LOGIC as connections between the gates.
component INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component; -- The component invrese gate takes one
input and give one output of type STD_LOGIC
component AND2 port (I0,I1: in STD_LOGIC; O: out STD_LOGIC); end component; -- The component AND2 gate takes
two inputs and give one output of type STD_LOGIC
component AND3 port (I0,I1,I2: in STD_LOGIC; O: out STD_LOGIC); end component;  -- The component AND3 gate takes
three inputs and give one output of type STD_LOGIC
component OR4 port (I0,I1,I2,I3: in STD_LOGIC; O: out STD_LOGIC); end component;    -- The component OR4 gate takes
four inputs and give one output of type STD_LOGIC
begin
  U1: INV port map (N(3), N3_L);    -- U1 is used to inverse N3 gate to N3_L.
  U2: INV port map (N(2), N2_L);    -- U2 is used to inverse N2 gate to N2_L.
  U3: INV port map (N(1), N1_L);    -- U3 is used to inverse N1 gate to N1_L.
  U4: AND2 port map (N3_L, N(0), N3L_N0);   -- U4 is taking N3_L and N0 as inputs and it output N3L_N0 using an AND2
gate.
  U5: AND3 port map (N3_L, N2_L, N(1), N3L_N2L_N1); -- U5 is taking N3_L, N2_L and N1 as inputs and it output
N3L_N2L_N1 using an AND3 gate.
  U6: AND3 port map (N2_L, N(1), N(0), N2L_N1_N0);  -- U6 is taking N2_L, N0 and N1 as inputs and it output N2L_N1_N0
using an AND3 gate.
  U7: AND3 port map (N(2), N1_L, N(0), N2_N1L_N0);  -- U7 is taking N1_L, N2 and N0 as inputs and it output N2_N1L_N0
using an AND3 gate.
  U8: OR4 port map (N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0, F);   -- U7 is taking N3L_N0, N3L_N2L_N1,
N2L_N1_N0 and N2_N1L_N0 as inputs and it output the final results using an OR4.
end prime1_arch;
```

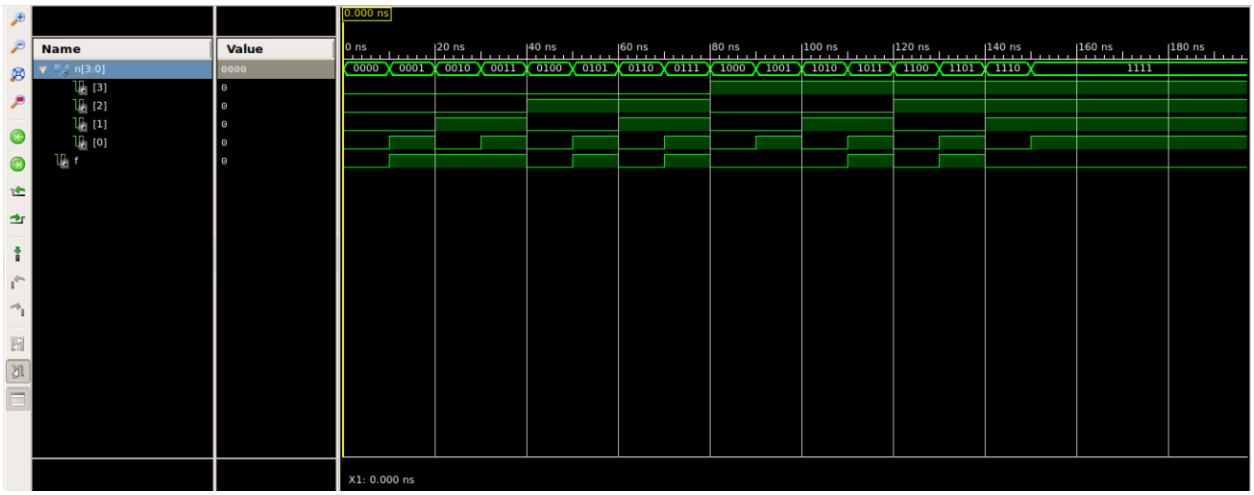
Test Bench code with explanation:

```
LIBRARY ieee; --Calling library IEEE.
USE ieee.std_logic_1164.ALL; --we are using all the definitions from library IEEE package.

ENTITY primel_tb IS -- Name of the gate test is decleared as "prime_tb"
END primel_tb;  -- The entity has been decleared

ARCHITECTURE behavior OF primel_tb IS   -- Architecture "behavior" is created for entity "prime_tb"
   COMPONENT primel --A concureent statement of type component with the name "primel"
   PORT (
         N: IN std_logic_vector (3 downto 0); -- N is input of type vector and it has 4 element
         F: OUT std_logic  -- F is output of type std_logic.
   END COMPONENT;
  signal N : std_logic_vector (3 downto 0) := (others => '0'); -- Input
  signal F : std logic; --  Output
```

```
BEGIN
  uut: primel PORT MAP ( -- Instantiate the Unit Under Test (UUT)
        N => N,
        F => F
      );
  stim proc: process
  begin
            -- Provide input combination for N and stop for 10ns to see the output
     N <= "0000"; wait for 10 ns;
     N <= "0001"; wait for 10 ns;
     N <= "0010"; wait for 10 ns;
     N <= "001l"; wait for 10 ns;
     N <= "0100"; wait for 10 ns;
     N <= "0101"; wait for 10 ns;
     N <= "0110"; wait for 10 ns;
     N <= "0111"; wait for 10 ns;
     N <= "1000"; wait for 10 ns;
     N <= "1001"; wait for 10 ns;
     N <= "1010"; wait for 10 ns;
     N <= "1011"; wait for 10 ns;
     N <= "1100"; wait for 10 ns;
     N <= "1101"; wait for 10 ns;
     N <= "ll10"; wait for 10 ns:
     N <= "1111"; wait for 10 ns;
     wait; -- To stop the process
  end process;
END;
```

2.2 - Simulation Results:





Explanation:

As shown in the picture, the value of F changes to 1 if and only if the binary combination is a prime number. Thus 4-digit binary combinations such as 0101 and 0111 are considered prime combinations because when we convert them to decimal, we get 5 and 7, which are indeed prime numbers.

Thus, looking at this picture, we can see that the binary combinations: 0001, 0010, 0011,0101, 0111, 1011, 1101 all output F as 1. Hence, they are prime numbers, and the rest are not.

Task 3: Excess-3 code detector (8 points)

3.2 - The implemented code with explanation:

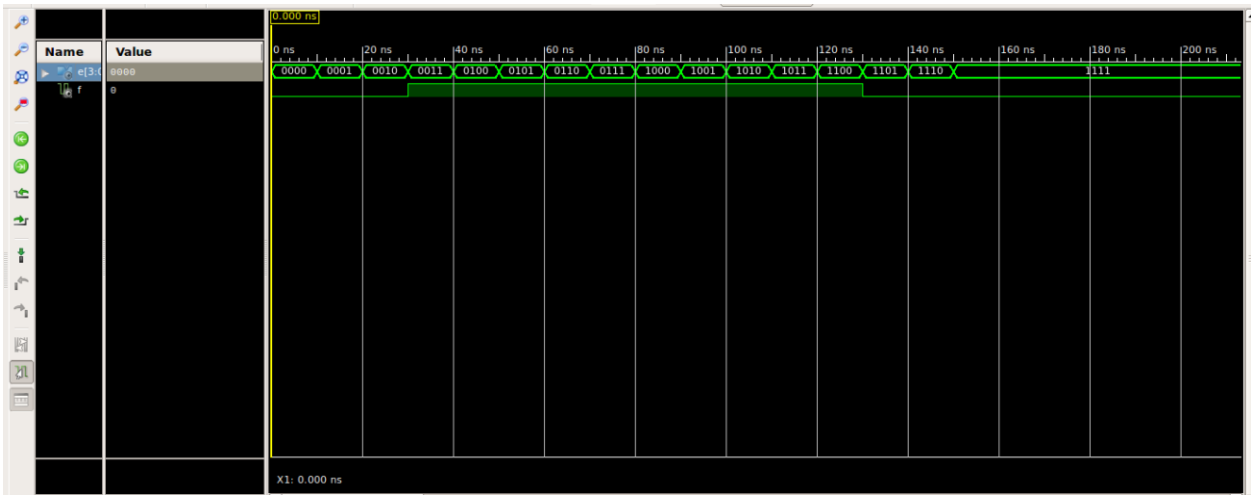
```
library IEEE;
use IEEE.std_logic_1164.aLL; --we are using all the definitions from library IEEE package.
Library unisim;
use unisim.vcomponents.all; --needed for the INV, NAND.

entity ex3_detector is -- Name of the gate is decleared as "ex3_detector"
   Port ( E : in  STD_LOGIC_VECTOR (3 downto 0); -- E is input of type vector and it has 4 element
          F : out  STD_LOGIC); -- F is output of type std_logic.
end ex3_detector;

architecture Behavioral of ex3_detector is -- Architecture "Behavioral" is created for entity "ex3_detector"
signal E3_L, E2_L, E1_L, E0_L: STD_LOGIC; -- Local signal is used to create E3_L, E2_L, E1_L, E0_L of type STD_LOGIC
as connections between the gates.
signal E2L_E3, E3L_E2, E3L_E1_E0, E3_E1L_E0L: STD_LOGIC; -- Local signal is used tp create E2L_E3, E3L_E2,
E3L_E1_E0, E3_E1L_E0L of type STD_LOGIC as connections between the gates.
component INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component; -- The component invrese gate takes one
input and give one output of type STD_LOGIC
component NAND2 port (I0, I1: in STD_LOGIC; O: out STD_LOGIC); end component; -- The component NAND2 gate
takes two inputs and give one output of type STD_LOGIC
component NAND3 port (I0, I1, I2: in STD_LOGIC; O: out STD_LOGIC); end component; -- The component NAND3 gate
takes three inputs and give one output of type STD_LOGIC
component NAND4 port (I0, I1, I2, I3: in STD_LOGIC; O: out STD_LOGIC); end component; -- The component NAND4
gate takes four inputs and give one output of type STD_LOGIC
begin
  U1: INV port map (E(3), E3_L); -- U1 is used to inverse E3 gate to E3_L.
  U2: INV port map (E(2), E2_L); -- U2 is used to inverse E2 gate to E2_L.
  U3: INV port map (E(1), E1_L); -- U3 is used to inverse E1 gate to E1_L.
  U4: INV port map (E(0), E0_L); -- U4 is used to inverse E0 gate to E0_L.
  U5: NAND2 port map (E2_L, E(3), E2L_E3); -- U5 takes 2 inputs and it give an output using NAND2 gate.
  U6: NAND2 port map (E3_L, E(2), E3L_E2); -- U6 takes 2 inputs and it give an output using NAND2 gate.
  U7: NAND3 port map (E3_L, E(1), E(0), E3L_E1_E0); -- U7 takes 3 inputs and it give an output using NAND3 gate.
  U8: NAND3 port map (E(3), E1_L, E0_L, E3_E1L_E0L); -- U8 takes 3 inputs and it give an output using NAND3 gate.
  U9: NAND4 port map (E2L_E3, E3L_E2, E3L_E1_E0, E3_E1L_E0L, F); -- U9 takes 4 inputs and it give an output using
NAND4 gate.

end Behavioral;
```

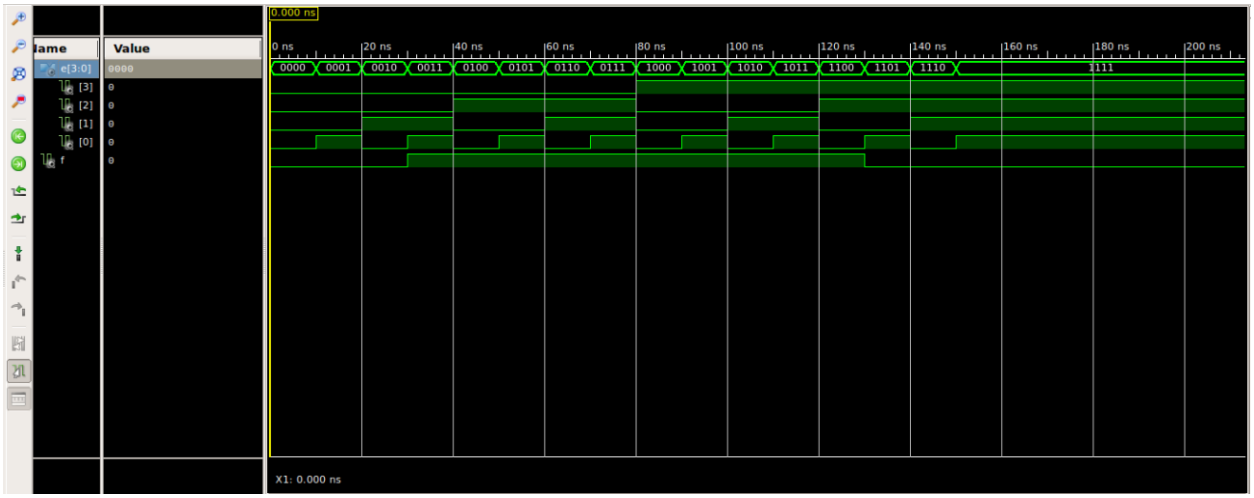
Test Bench code with explanation:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL; --we are using all the definitions from library IEEE package.

ENTITY ex3_testbench IS  -- Name of the gate test is decleared as "ex3_testbench"
END ex3_testbench;

ARCHITECTURE behavior OF ex3_testbench IS -- Architecture "behavior" is created for entity "ex3_testbench"
   -- Component Declaration for the Unit Under Test (UUT)
   COMPONENT ex3_detector
   PORT(
       E : IN  std_logic_vector(3 downto 0); -- E is input of type vector and it has 4 element
       F : OUT  std_logic -- F is output of type std_logic.
      );
   END COMPONENT;

  --Inputs
  signal E : std_logic_vector(3 downto 0) := (others => '0');
```

```
   --Outputs
  signal F : std_logic;
BEGIN
   -- Instantiate the Unit Under Test (UUT)
  uut: ex3_detector PORT MAP (
         E => E,
         F => F
       );
  -- Stimulus process
  stim_proc: process
  begin
    -- insert stimulus here
    -- Provid input combinations for N and stop 10ns to see the output
    E <= "0000"; wait for 10 ns;
    E <= "0001"; wait for 10 ns;
    E <= "0010"; wait for 10 ns;
    E <= "0011"; wait for 10 ns;
    E <= "0100"; wait for 10 ns;
    E <= "0101"; wait for 10 ns;
    E <= "0110"; wait for 10 ns;
    E <= "0111"; wait for 10 ns;
    E <= "1000"; wait for 10 ns;
    E <= "1001"; wait for 10 ns;
    E <= "1010"; wait for 10 ns;
    E <= "1011"; wait for 10 ns;
    E <= "1100"; wait for 10 ns;
    E <= "1101"; wait for 10 ns;
    E <= "1110"; wait for 10 ns;
    E <= "1111"; wait for 10 ns;
   wait;
  end process;
END;
```

3.3 Simulation Results:





Explanation:

In order to explain the wave, we need to understand why some certain combination gives an excess 3. Thus, as we see in the screenshot, F is 1 from 30ns to130ns for the following combinations: 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100 and the rest output F as 0.

I believe this is mainly has to do with the algorithm of turning a binary into decimal by adding 3 to each digit. For instance, if we looked at 1011, which represent 3 in decimal, we see that from 110ns to 120ns it gives excess 3 because F value is 1 and therefore if we add 3, we get 3+3 = 6. However, for a combination like 1110, which represent 14, we see that from 140ns to 150ns the value of F is zero because when we add 14+3 = 17 and the value 17 does not exist in 4-digit binary number.