Laboratory Report

COMBINATIONAL LOGIC DIGITAL SYSTEM DESIGNING AND IMPLEMENTATION

EEX5351

Digital Electronic System

LAB 01

By

S.A.P. Kavinda

617143597

Submitted to

Department of Electrical & Computer Engineering

Faculty of Engineering Technology

The Open University of Sri Lanka

On

31/08/2022

COMBINATIONAL LOGIC DIGITAL SYSTEM DESIGNING AND IMPLEMENTATION

1. Learning Outcome

- Apply the concepts of basic timing issues, including clocking, timing constraints, and propagational delays during design process.
- Develop Complex digital systems in a hierarchical fashion using top-down and bottom-up design approaches.
- Verify the digital system design via digital circuit simulation using an HDL.
- Implement digital system design using a programmable platform.
- Explain the types and characteristics of common fault that occur a in digital electronic systems.
- Discuss different computer-aid testing tools for circuit simulation, fault diagnosis and Automatic Test Pattern Generator.
-

2. Apparatus

- Anvyls Spartan 6 FPGA board
- Xlinx ISE Design Suite 14.7

3. Introduction

Digital Electronic Systems

We can call the physical systems as the set of interconnected objects or elements that realize some functions using a set of input and output signals. The systems where all input and outputs are digital signals are called Digital Systems. In modern days digital systems are presents in everywhere. We can say the simplicity of design approaches, the higher efficiency, high accuracy, and the modeling capability using algorithms are the key points of digital electronic systems that involve to this success. Therefore, the design and implementation is an important part for Engineers for developing sustainable systems and analysis. Digital Electronic systems use two levels of voltages or currents for represents the logic 0 or 1 that in binary number system. These are the basic components of digital systems. Also, the switches are the main controlling elements of digital electronic systems. Normally MOS Transistors are uses for develop this switches. By using MOS transistors, we can implement Inverters, NAND ga,tes and NOR gates. These NAND and NOR gates are called as universal gate because other all logic gates (AND, OR, XOR, XNOR) could implement using one of these.

Why use NAND or NOR gates instead of AND and OR gates? Mainly there are three reasons for that. In the laboratory, we only need one kind of gate. Because we can implement all kinds of gate using sing NAND or NOR gates. Also, nMOS switches are good for transmitting 0V. pMOS switches are transmitting 1V. So in COMS technology, AND gates are implemented using NAND gates and OR gates are Implemented using NOR gates. The 3rd reason is within an IC, NAND and NOR gates are cheaper than AND and OR gates.

Delay Timing

In reality, the output of a piece of real hardware does not change instantaneously when an input changes. Inevitably, there is a delay. This is due to various things such as capacitance of transistors, Transmission line capacitance etc… In VHDL digital system modelling, there are three prescribed delays can be identified. Any signal assignment will incur a delay of one the three types of below.

1. <u>Transport Delay</u>

In this delay type consider propagation delay only. This signal will assume the new value after specific time.

$$output <= TRANSPORT\ x\ AFTER\ 10ns;$$

This Transport delay is like a infinite bandwidth Transmission line.

2. Inertial Delay

This Delay deal with the minimum input pulse with and propagation delay. Inertial delay acts like a real gate, It absorbs pulses narrow than in width than the propagation delay. This is the default in VHDL statements which contain “AFETER” clause.

$$output <= x\ AFETR\ 10ns$$

Also, In VHDL we can define the Pulse Reject time specifically also. This REJECT keyword can be used only with the keyword INTERNAL.

$$output = REJECT\ 5ns\ INTERNAL\ x\ AFETR\ 10ns;$$

3. Delta Delay

This type represents the zero-delay time, that’s means ideal components. In VHDL not included any keyword in the signal assignment.

Anvyl FPGA Development board

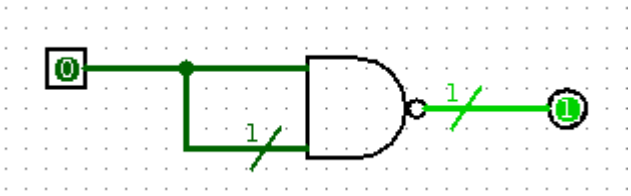
FPGA (Field Programmable Gate Arrays) are the semiconductor devices that are based on a matrix of configurable logic block (CLBs) connected via programmable interconnects. Although there are various types of FPGA available, SRAM based FPGAs are dominant.

The Anvyl FPGA development platform is a complete, ready to use digital circuit development platform based on a speed grade -3 Xilinx Spaetan-6 LX45 FPGA. In this laboratory session we are used this demonstration board to demonstrate our designs.

4. Preparatory Tasks

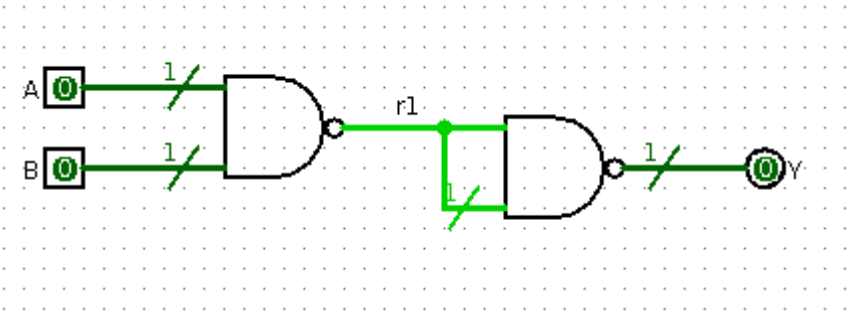
4.1 Design an INVERTER, AND Gate, OR Gate, XOR Gate, Multiplexer and D-Flip Flop using NAND Gates.

4.1.1 Inverter:



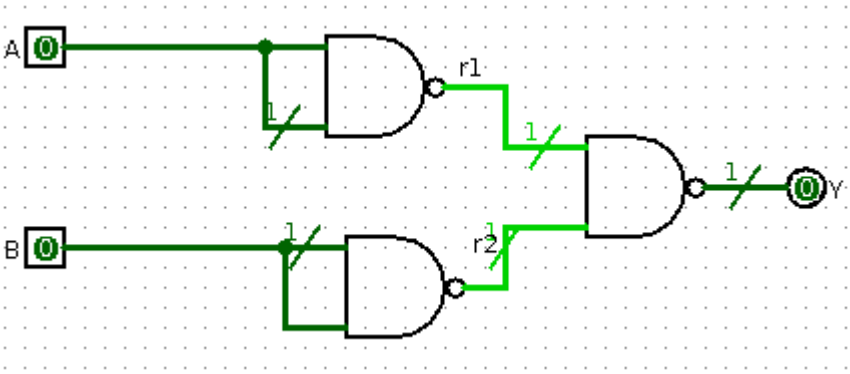
$$Y = \overline{A\,.A} \;=\; \bar{A}$$

4.1.2 AND Gate:



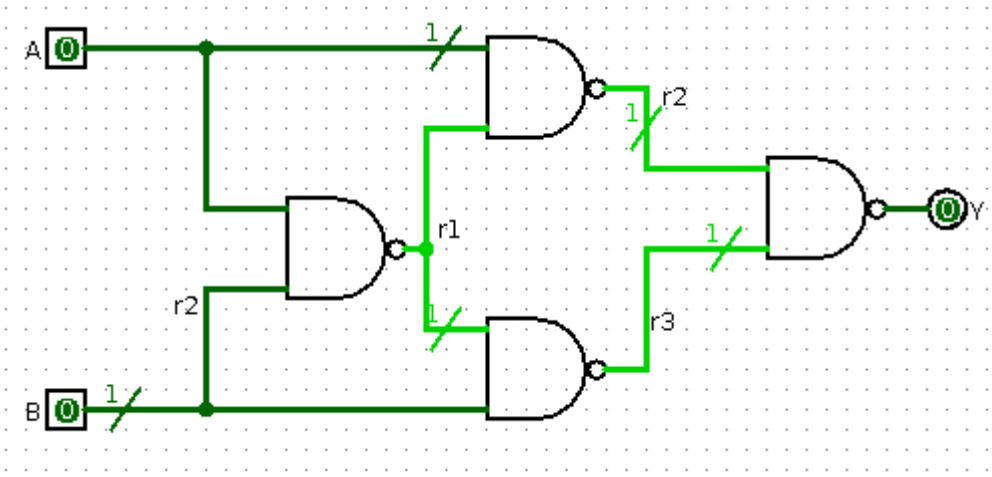
$$Y = \overline{\overline{A\,.B}} \;= A.B$$

4.1.3 OR Gate:



$$Y = A + B = \overline{\bar{A}.\bar{B}}$$

4.1.4 XOR Gate:



$$Y = \overline{\bar{A}.\bar{B} + A.B} = (A + B).(\bar{A} + \bar{B}) = A.\bar{B} + \bar{A}.B$$

4.1.5 Multiplexer

Table 1

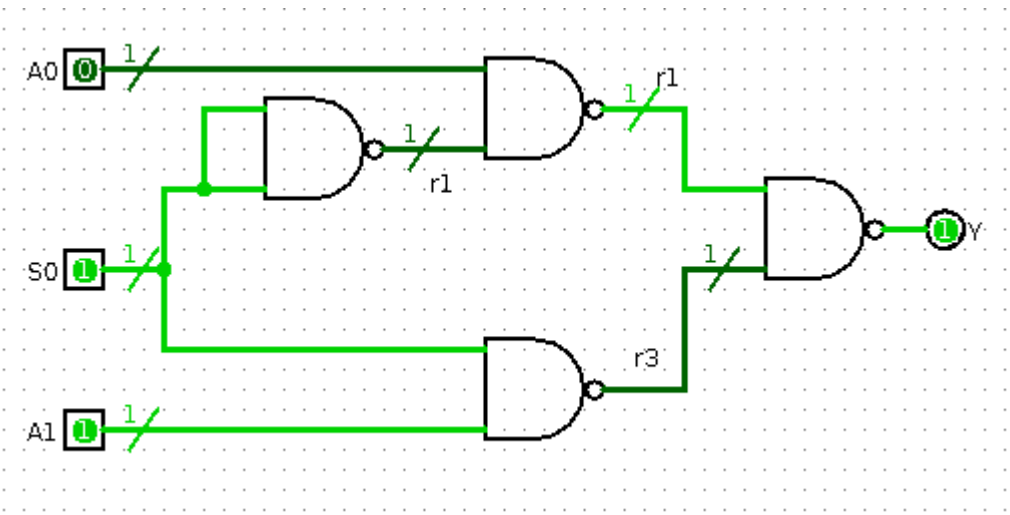
A0	A1	S0	Y
0	X	0	0
1	X	0	1
X	0	1	0
X	1	1	1

Y1 = A0 S0’

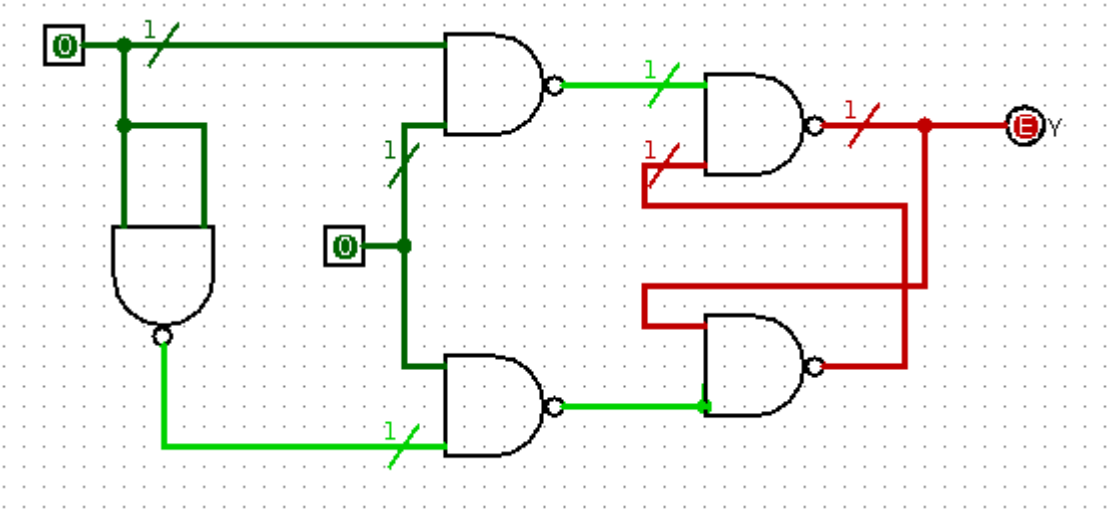
Y1 = A1 S0

$$Y = A_0.\bar{S_0} + A_1 .S_0$$

$$Y = \overline{A_0 .\bar{S_0}}.\overline{A_1 .S_0}$$

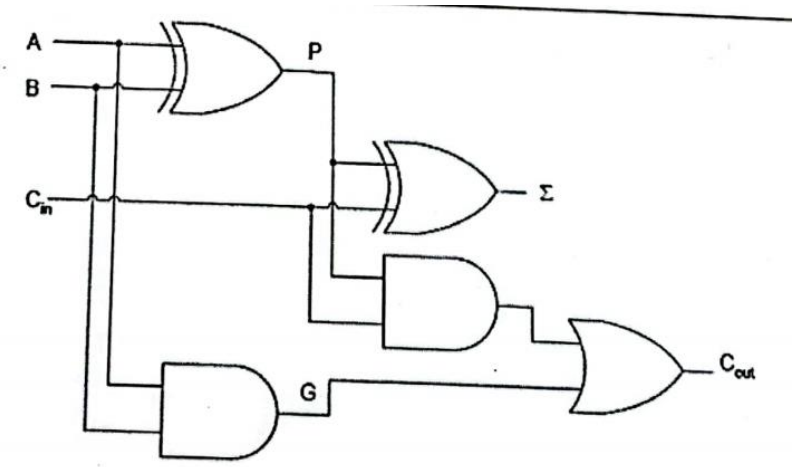


4.1.6 D-Flip Flop

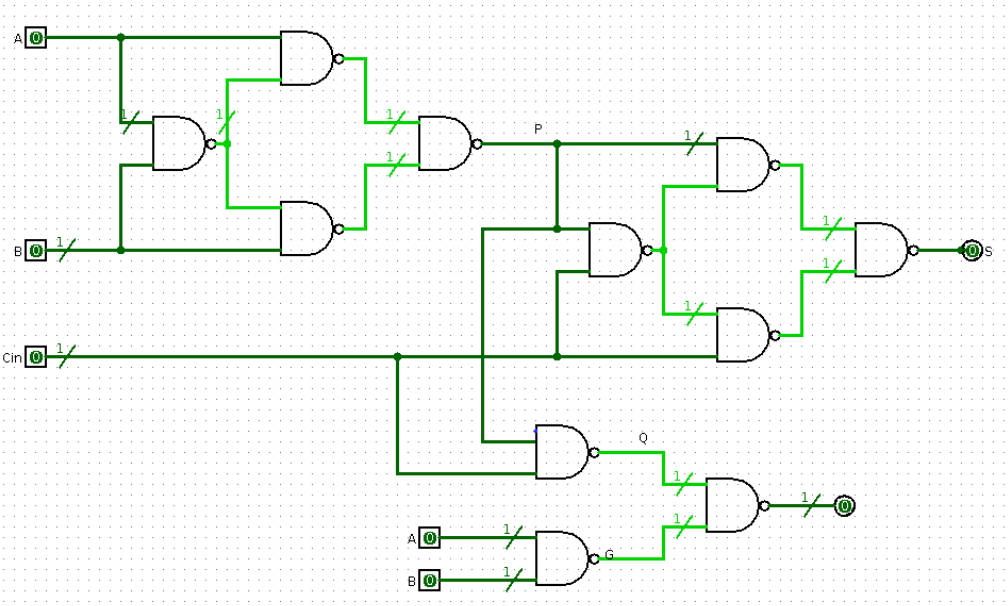


4.2 Design given system using NAND Gates

Given System:

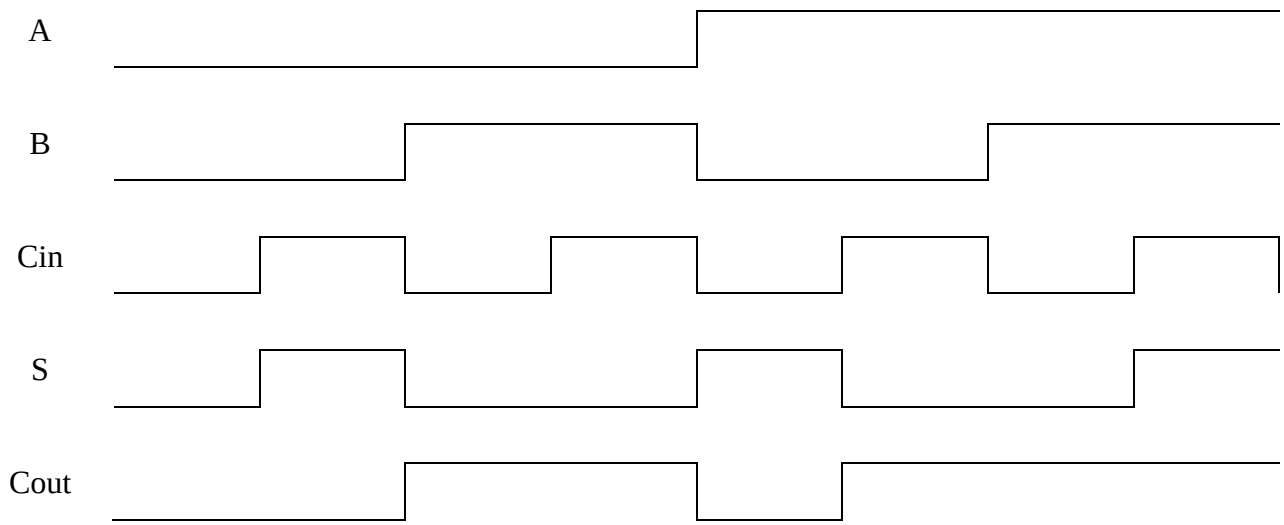


Proposed System:



4.3 Draw a timing diagram proposed design

Table 2



5. Procedure

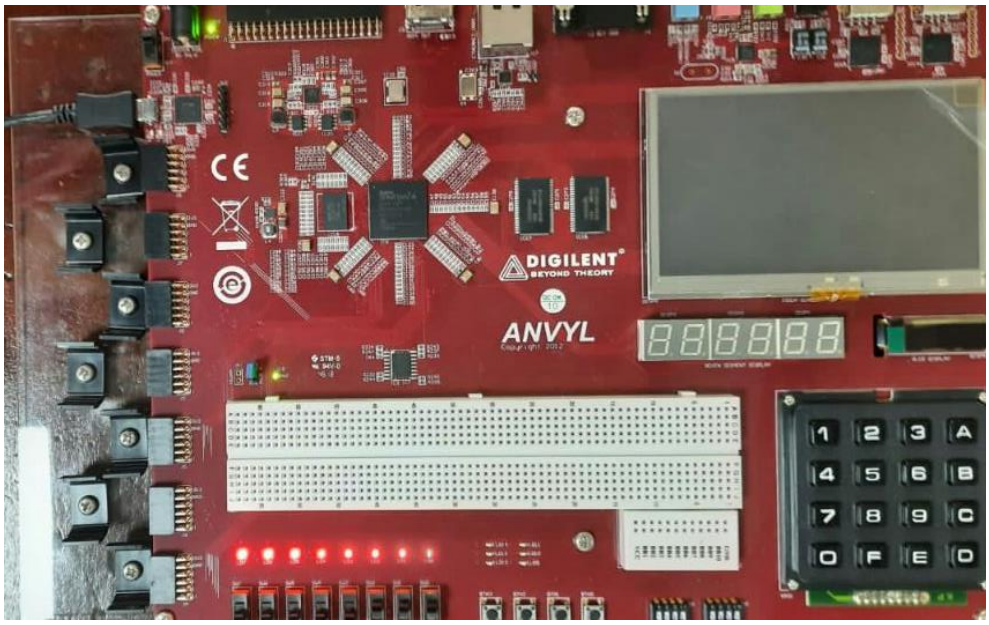
1. Interfaced the Anvyl FPGA demonstration system with computer and open the Xilinx Software, and program uploaded utility software.
2. Read the Anvyl FPGA Demonstration manual and Test demo, Counter demo, Keypad demo and touch pad demo with Anvyl FPGA.
3. Designed Inverter, AND Gate, OR Gate, XOP Gate, Multiplexer and D Flip flop using NAND Gates with VHDL.
4. Simulated the proposed VHDL program and verify the results with Timing Diagram which included in 4.3.
5. Assigned input, output pins to available buttons and uploaded the VHDL program to Anvyl FPGA.
6. Tested the design and identify the faults or Errors.

6. Observations/Results

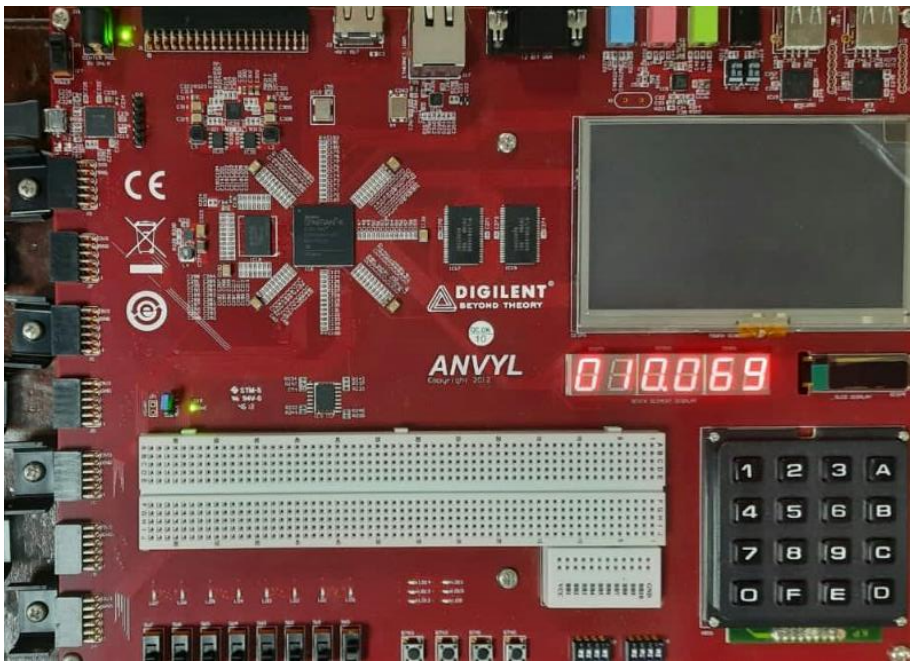
6.1 Demo Program Testing results

LED Demo:

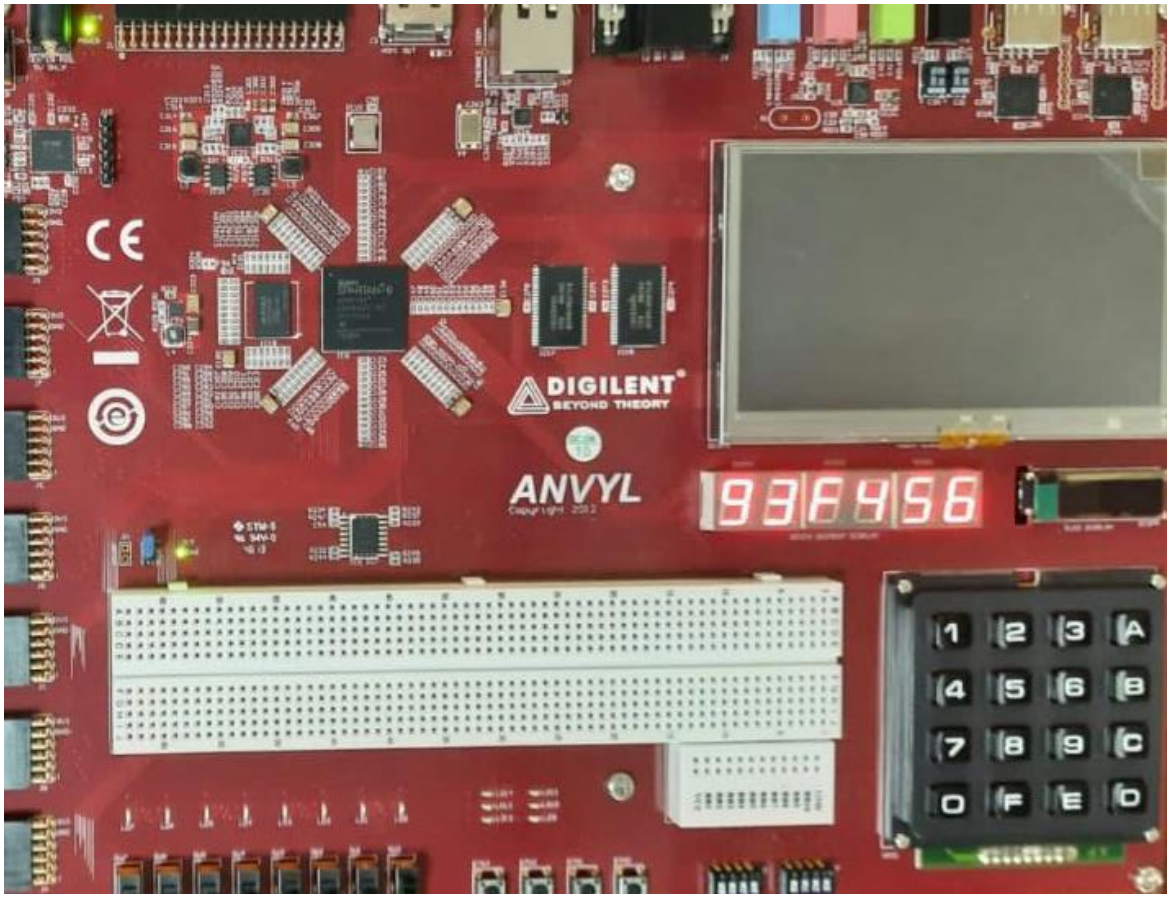
The Speed of lights could change using the sliders.



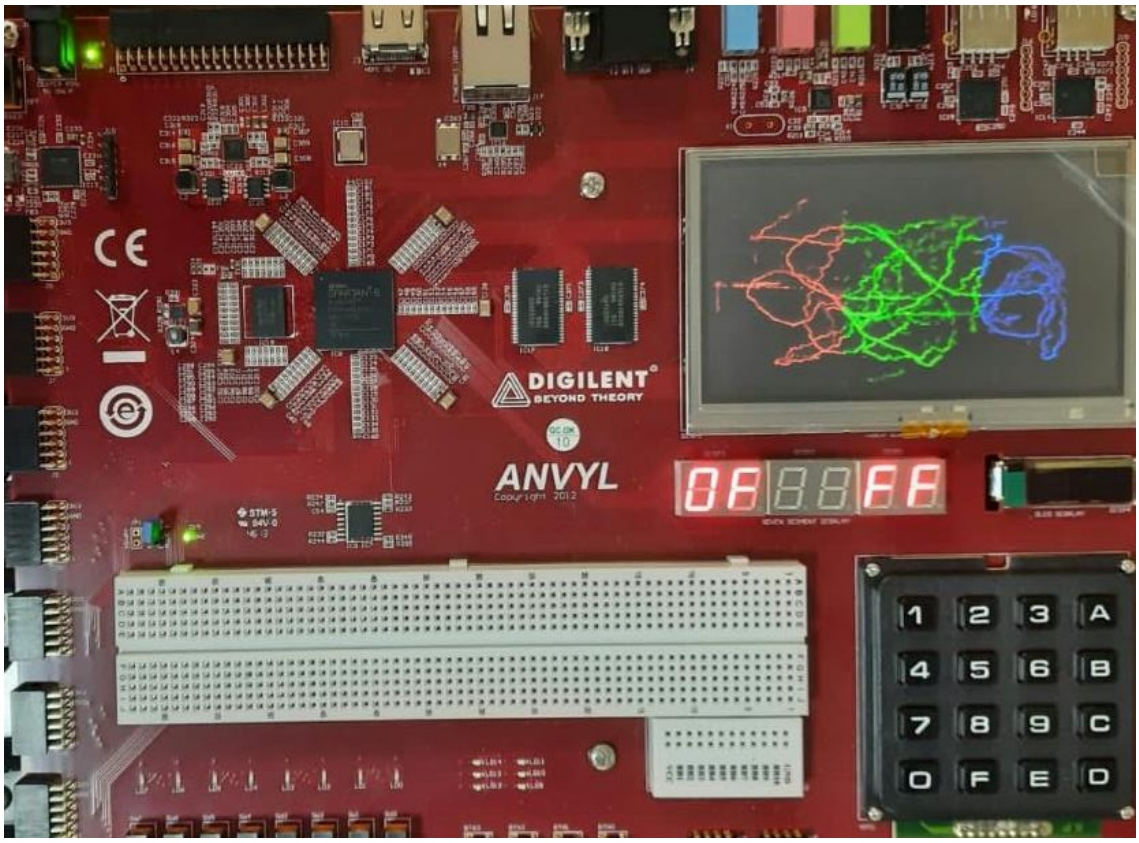
Counter Demo



Keypad Demo:

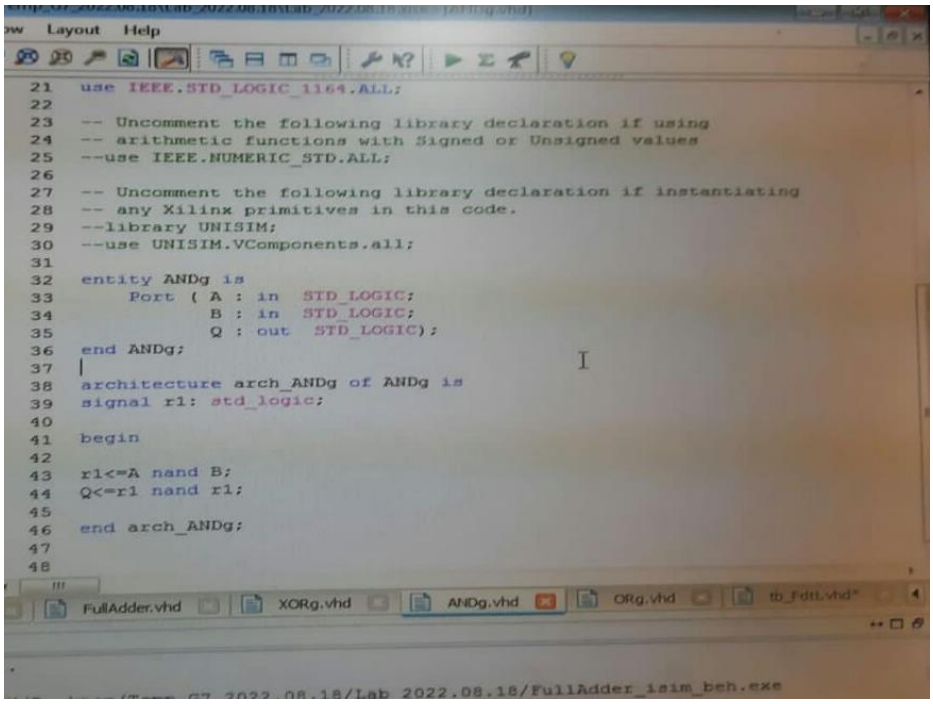


Touch Pad Demo



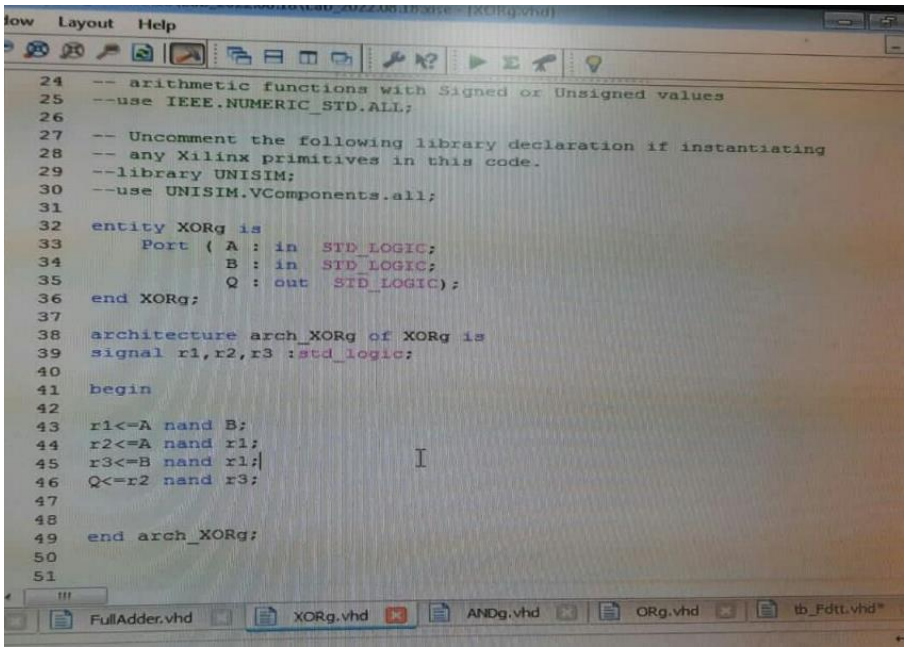
6.2 Implemented Components using NAND Gate

AND Gate



```
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ANDg is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           Q : out  STD_LOGIC);
end ANDg;

architecture arch_ANDg of ANDg is
signal r1: std_logic;

begin

r1<=A nand B;
Q<=r1 nand r1;

end arch_ANDg;
```

XOR Gate



```
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity XORg is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           Q : out  STD_LOGIC);
end XORg;

architecture arch_XORg of XORg is
signal r1,r2,r3 :std_logic;

begin

r1<=A nand B;
r2<=A nand r1;
r3<=B nand r1;
Q<=r2 nand r3;


end arch_XORg;
```

OR Gate

```
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ORg is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           Q : out  STD_LOGIC);
end ORg;

architecture arch_ORg of ORg is
signal r1,r2 :std_logic;
begin

r1<=A nand A;
r2<=B nand B;

Q<=r1 nand r2;


end arch_ORg;
```

2Bit Multiplexer

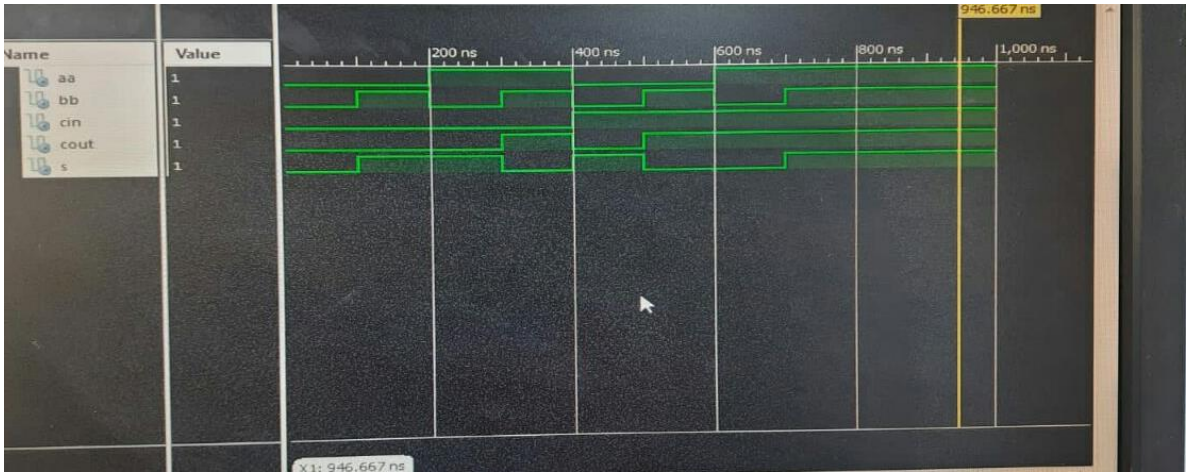
```
-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity MUX is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           S : in  STD_LOGIC;
           Q : out  STD_LOGIC);
end MUX;

architecture arch_MUX of MUX is
signal r1,r2,r3 :std_logic;

begin

r1<=S nand S;
r2<=A nand r1;
r3<= B nand S;
Q<= r2 nand r3;


end arch_MUX;
```

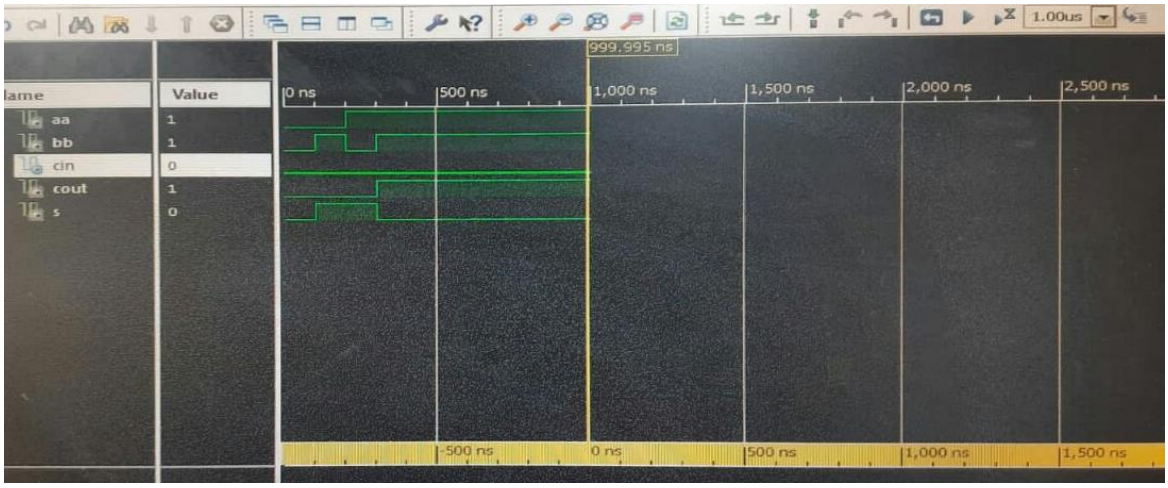
Full Adder

```
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity FullAdder is
    Port ( aa : in  STD_LOGIC;
           bb : in  STD_LOGIC;
           Cin : in  STD_LOGIC;
           Cout : out  STD_LOGIC;
           S : out  STD_LOGIC);
end FullAdder;



architecture arch_FullAdder of FullAdder is

component XORg is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           Q : out  STD_LOGIC);
end component;

component ANDg is
    Port ( A : in  STD_LOGIC;
```

```
42  architecture arch_FullAdder of FullAdder is
43
44  component XORg is
45      Port ( A : in  STD_LOGIC;
46             B : in  STD_LOGIC;
47             Q : out  STD_LOGIC);
48  end component;
49
50
51  component ANDg is
52      Port ( A : in  STD_LOGIC;
53             B : in  STD_LOGIC;
54             Q : out  STD_LOGIC);
55  end component;
56
57
58  component ORg is
59      Port ( A : in  STD_LOGIC;
60             B : in  STD_LOGIC;
61             Q : out  STD_LOGIC);
62  end component;
63
64  signal r1,r2,r3: std_logic;
65  begin
66
67  x1uu: XORg
68  port map( a=>aa,b=>bb,Q=>r1);
69
```

Full Adder Simulation Results





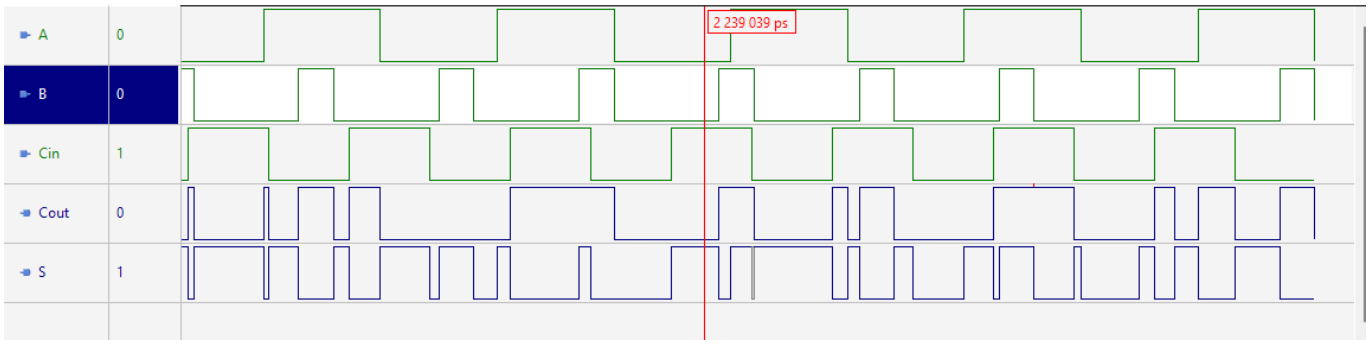
7. Analysis of Results

1. In first task we experienced various applications and functionalities of Anvyl FPGA. Also, we identified the main components and software requirements.
2. In second task, we implemented some basic logic elements using NAND Gates. First we implemented a NAND gate using VHDL and we implement another AND, OR, XOR, D flip flop and Multiplexer in hierarchical manner. Also we try to simulate those elements using Xilinx software.
3. Then for the task 3, we design and implemented a given full adder circuit by using implemented basic elements. Also, we wrote a test bench program for test the deign from simulation.
4. We got the simulation timing diagram for implemented design and refer it.
5. Finally, we upload the program to Anvyl FPGA Demonstration board. But unfortunately, we could not saw the results.

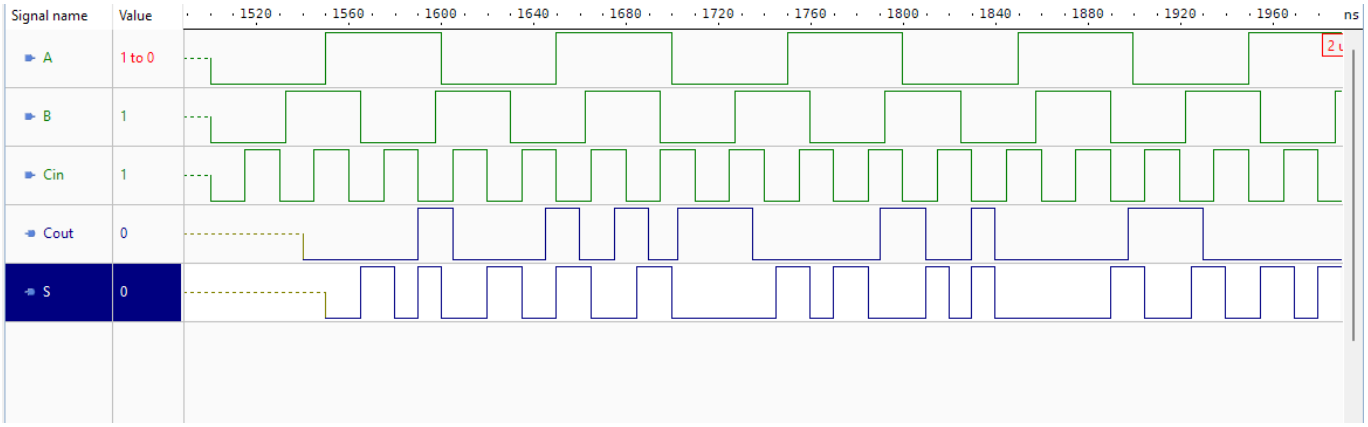
So although some VHDL programs are working in simulations without any issue, in real time it may not working.

After lab, I simulated my codes with and without considering delay timing. These are the simulation results that I obtained.

Without considering delay timing:



Considering delay timing:



8. Conclusion

- Every digital electronic system could implement using a universal gate. In practically it has raise time, fall time, propagational delay time and minimum pulse width can deal.
- We can separate complex design tasks into small blocks and implements those and interconnect them.
- Same digital system behaves differently with different timing parameters. Therefore, timing is an important aspect the in-design process.
- Although simulation results are working without any error, reality may be show several errors. For examples,
    - Digital Design don’t work if the time between clock pulse isn’t sufficient for all of the logic take place.
    - When switches are translation, switch contacts create glitches on the line. We can avoid the affect from those by implement a debounce module.

9. Appendix

9.1 VHDL Code without considering timing

9.1.1 NAND Gate:

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

entity NANDg is
    port(A : in std_logic;
    B : in std_logic;
    Y : out std_logic);
end NANDg;


architecture arc_NANDg of NANDg is
begin
    Y <= A nand B;

end arc_NANDg;
```

9.1.2 Inverter Code:

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

entity inverter is
    port(A : in std_logic;
    Y : out std_logic);
end inverter;


architecture arc_inverter of inverter is
component NANDg is
    port(A : in std_logic;
    B : in std_logic;
    Y : out std_logic);
end component;

begin
    nand1 : NANDg
    port map(A => A, B =>A, Y=>Y);

end arc_inverter;
```

9.1.3 AND GATE CODE

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

entity ANDg is
    port(A : in std_logic;
    B : in std_logic;
    Y : out Std_logic);
end ANDg;


architecture arc_ANDg of ANDg is

component NANDg is
    port(A : in std_logic;
    B : in std_logic;
    Y : out std_logic);
end component;


signal r1 : std_logic;

begin

    NAND1 : NANDg
    port map(A => A , B => B , Y => r1);

    NAND2: NANDg
    port map(A => r1 , B=>r1, Y => Y);

end arc_ANDg;
```

9.1.4 OR GATE CODE

```
Library IEEE;
use IEEE.Std_logic_1164.ALL;

entity ORg is
    port(A : in std_logic;
    B : in std_logic;
    Y : out std_logic);
end ORG;

architecture arc_ORg of ORg is
component NANDg is
    port(A : in std_logic;
    B : in std_logic;
    Y : out std_logic);
end component;

signal r1,r2 :std_logic;

begin
    NAND1:NANDg
    port map(A => A,B => A , Y => r1);

    NAND2:NANDg
    port map(A => B,B => B , Y => r2);

    NAND3:NANDg
    port map(A => r1,B => r2 , Y => Y);

end arc_ORg;
```

9.1.5 XOR GATE CODE

```
Library IEEE;
use IEEE.Std_logic_1164.ALL;

entity XORg is
    port(A : in std_logic;
    B : in std_logic;
    Y : out std_logic);
end XORg;

architecture arc_XORg of XORg is
component NANDg is
    port(A : in std_logic;
    B : in std_logic;
    Y : out std_logic);
end component;

signal r1,r2,r3,r4 :std_logic;

begin
    NAND1:NANDg
    port map(A => A,B => A , Y => r1);

    NAND2:NANDg
    port map(A => B,B => B , Y => r2);

    NAND3:NANDg
    port map(A => r1,B => B , Y => r3);

    NAND4:NANDg
    port map(A => A,B => r2 , Y => r4);

    NAND5:NANDg
    port map(A => r3,B => r4 , Y => Y);

end arc_XORg;
```

9.1.6 D FLIP FLOP

```
Library IEEE;
use IEEE.Std_logic_1164.ALL;

entity DFLIP is
    port(D : in std_logic;
    CLK : in std_logic;
    Q : inout std_logic;
    Q_i : inout std_logic);
end DFLIP;

architecture arc_DFLIP of DFLIP is
component NANDg is
    port(A : in std_logic;
    B : in std_logic;
    Y : out std_logic);
end component;

signal r1,r2,r3 :std_logic;

begin
    NAND1:NANDg
    port map(A => D,B => D , Y => r1);

    NAND2:NANDg
    port map(A => D,B => CLK , Y => r2);

    NAND3:NANDg
    port map(A => CLK,B => r1 , Y => r3);

    NAND4:NANDg
    port map(A => r2,B => Q_i , Y => Q);

    NAND5:NANDg
    port map(A => r3,B => Q , Y => Q_i);

end arc_DFLIP;
```

9.1.7 MUX CODE

```
Library IEEE;
use IEEE.Std_logic_1164.ALL;

entity MUX is
    port(A : in std_logic;
    B : in std_logic;
    S :in std_logic;
    Y : out std_logic);
end MUX;

architecture arc_MUX of MUX is
component NANDg is
    port(A : in std_logic;
    B : in std_logic;
    Y : out std_logic);
end component;

signal r1,r2,r3 :std_logic;

begin
    NAND1:NANDg
    port map(A => S,B => S , Y => r1);

    NAND2:NANDg
    port map(A => A,B => r1 , Y => r2);

    NAND3:NANDg
    port map(A => B,B => S , Y => r3);

    NAND4:NANDg
    port map(A => r2,B =>r3 , Y => Y);

end arc_MUX;
```

9.1.8 Full Adder Code

```
Library IEEE;
use IEEE.Std_logic_1164.ALL;

entity Full_Adder is
    port(A : in std_logic;
    B : in std_logic;
    Cin :in std_logic;
    Cout : out std_logic;
    S : out std_logic);
end Full_Adder;

architecture arc_Full_Adder of Full_Adder is

entity XORg is
    port(A : in std_logic;
    B : in std_logic;
    Y : out std_logic);
end XORg;

entity ANDg is
    port(A : in std_logic;
    B : in std_logic;
    Y : out Std_logic);
end ANDg;

entity ORg is
    port(A : in std_logic;
    B : in std_logic;
    Y : out std_logic);
end ORG;


signal P,G,R :std_logic;

begin
    XOR1:XORg
    port map(A => S,B => S , Y => r1);

    XOR2:XORg
    port map(A => A,B => r1 , Y => r2);

    AND1:ANDg
    port map(A => B,B => S , Y => r3);

    AND2:ANDg
    port map(A => r2,B =>r3 , Y => Y);

    OR1:ORg
    port map(A => r2,B =>r3 , Y => Y);

end arc_Full_Adder;
```

9.2 VHDL Code with considering timing

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

entity NANDg is
    port(A : in std_logic;
    B : in std_logic;
    Y : out std_logic);
end NANDg;


architecture arc_NANDg of NANDg is
begin
    Y <= A nand B after 10ns;

end arc_NANDg;
```