LAB 5:MULTIPLIER AND INTRODUCTION TO SCANCHAIN

ARYA AGARWAL
Roll Number - 210070012

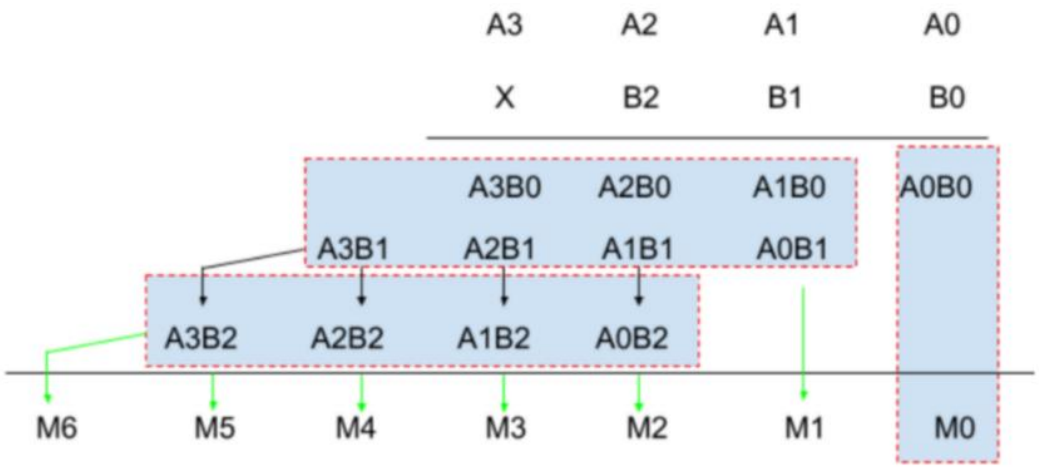
1 September 2022

1. Overview of the Experiment / Assignment:

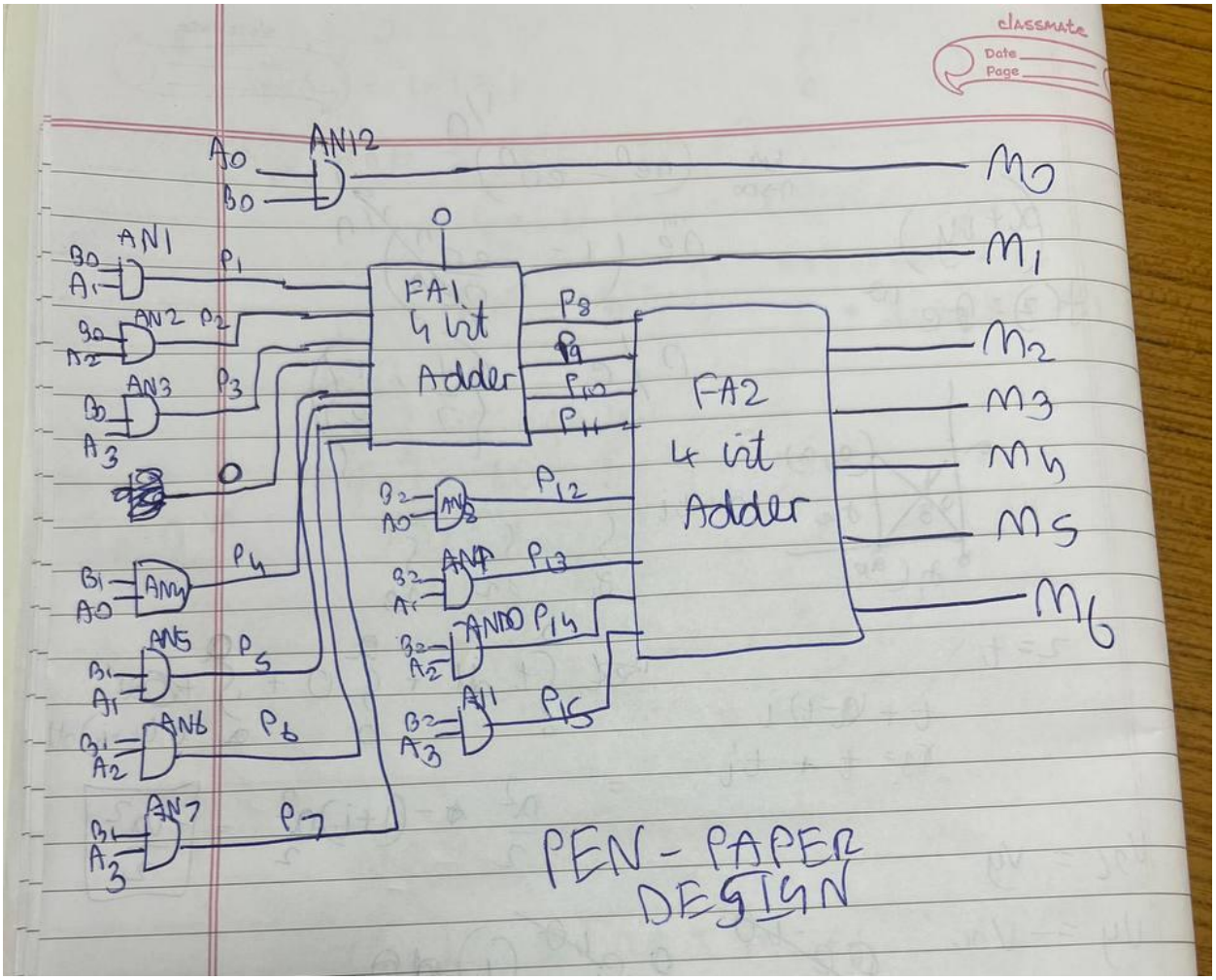
- In this experiment, we were first given the introduction of scanchain.
- Then we were introduced to our lab 5. It was a MULTIPLIER.
- Then using Quartus and tracefile provided we have to write our VHDL code for our design.
- After writing the VHDL code , we have to check our design using Scanchain.

2. Experiment Setup or Approach to the Assignment:

- By Using the tracefile given and the logic design, I used two 4 bit adder which we created in our last experiment.I also used 12 And gates to get the multiplied bits. Below is the pen paper Design which I used and the logic of the design which were given to us.



- Therefore, the possible figure of circuit is given below



2.1 Design Code and Documentation:

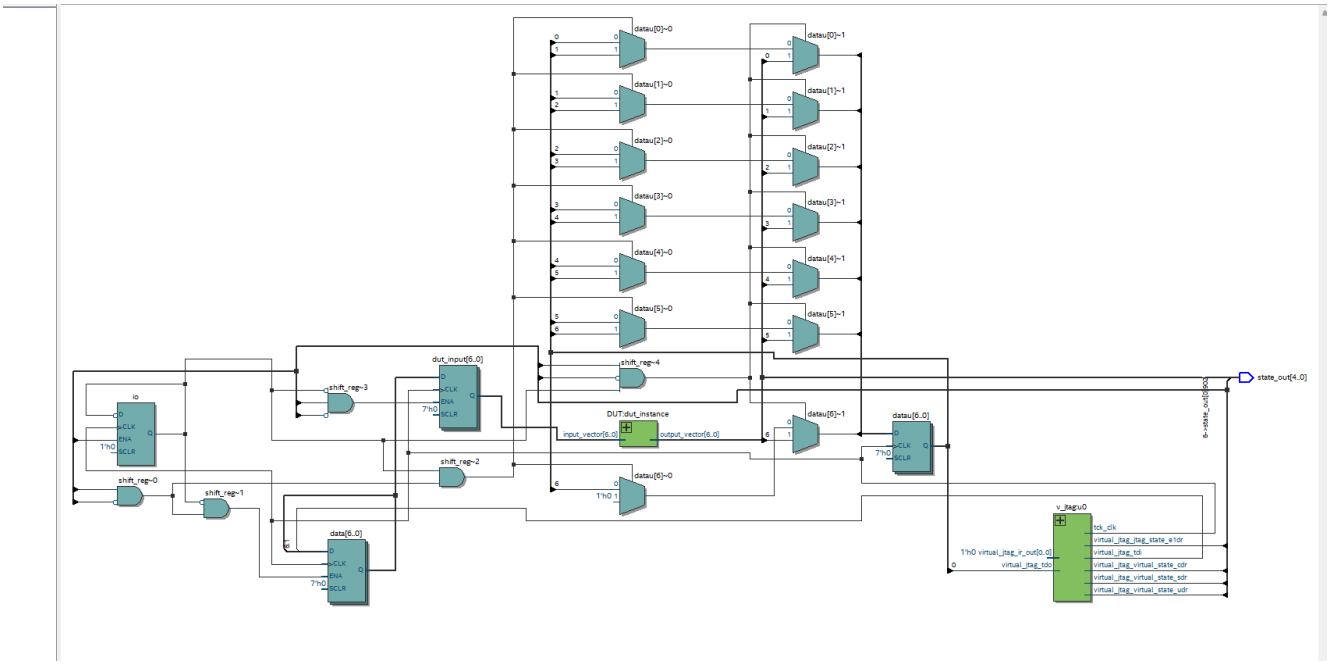
Now once we have drawn the circuits on paper, its time to describe the circuit using code in vhdl language in Quartus. The dut, gates files will almost be same except some minor changes in both case. We have to add BINARY ADDER component before using it in the design.

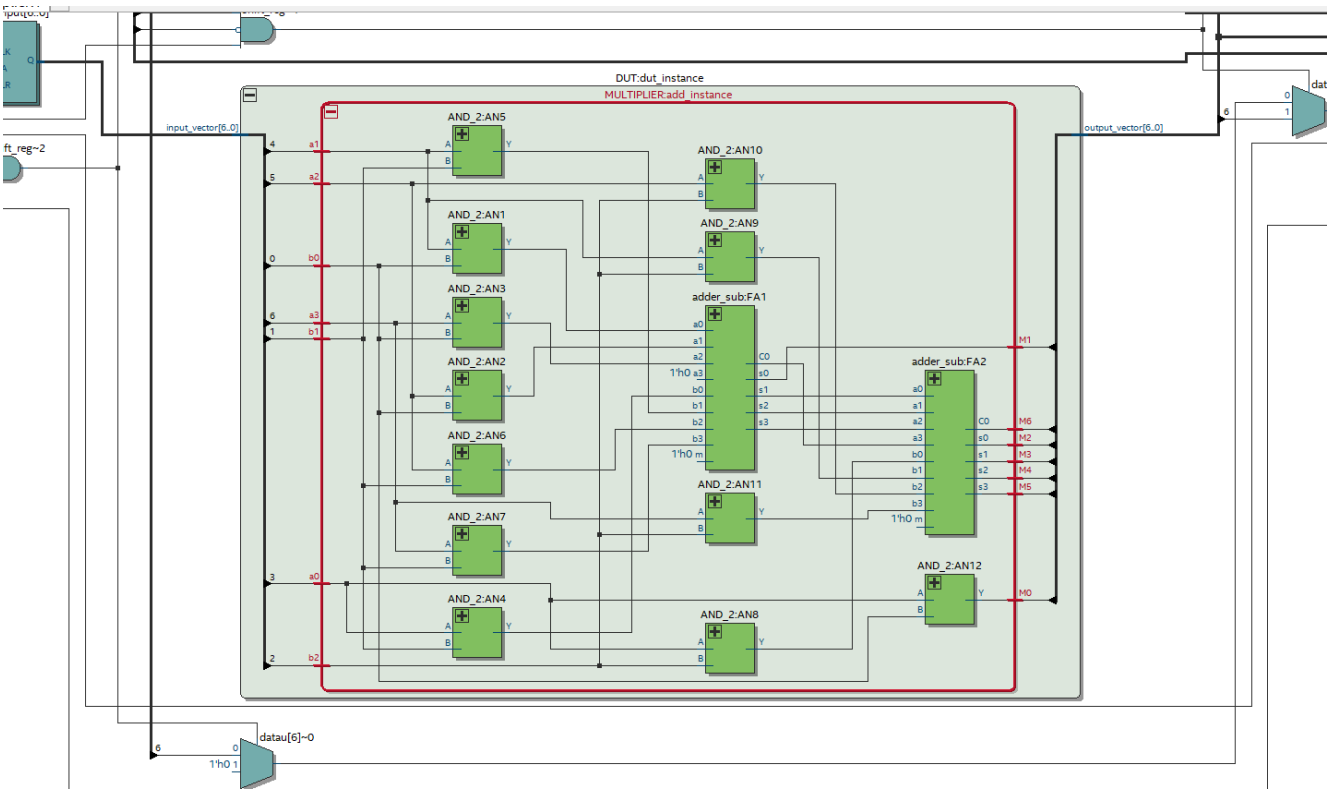
The code for the main file is written below which is used to describe the circuit. You can refer to the pen paper design for instance and signal names.

```
library ieee;
use ieee.std_logic_1164.all;

library work;
use work.Gates.all;

entity MULTIPLIER is
port (a0,a1,a2,a3,b0,b1,b2:in std_logic;
      M0,M1,M2,M3,M4,M5,M6:out std_logic);
end entity MULTIPLIER;

architecture struct of MULTIPLIER is
component adder_sub is
port (a0,a1,a2,a3,b0,b1,b2,b3,m:in std_logic;
      s0,s1,s2,s3,C0:out std_logic);
  end component;
signal p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15:std_logic;
begin
AN1: AND_2
port map(a=>a1,b=>b0,y=>p1);
AN2: AND_2
port map(a=>a2,b=>b0,y=>p2);
AN3: AND_2
port map(a=>a3,b=>b0,y=>p3);
AN4: AND_2
port map(a=>a0,b=>b1,y=>p4);
AN5: AND_2
port map(a=>a1,b=>b1,y=>p5);
AN6: AND_2
port map(a=>a2,b=>b1,y=>p6);
AN7: AND_2
port map(a=>a3,b=>b1,y=>p7);
AN8: AND_2
port map(a=>a0,b=>b2,y=>p12);
AN9: AND_2
```

```
port map(a=>a3,b=>b1,y=>p7);
AN8: AND_2
port map(a=>a0,b=>b2,y=>p12);
AN9: AND_2
port map(a=>a1,b=>b2,y=>p13);
AN10: AND_2
port map(a=>a2,b=>b2,y=>p14);
AN11: AND_2
port map(a=>a3,b=>b2,y=>p15);
AN12: AND_2
port map(a=>a0,b=>b0,y=>M0);
FA1: adder_sub
port map(a0=>p1,a1=>p2,a2=>p3,a3=>'0',b0=>p4,b1=>p5,b2=>p6,b3=>p7,m=>'0',s0=>M1,s1=>p8,s2=>p9,s3=>p10,c0=>p11);
FA2: adder_sub
port map(a0=>p8,a1=>p9,a2=>p10,a3=>p11,b0=>p12,b1=>p13,b2=>p14,b3=>p15,m=>'0',s0=>M2,s1=>M3,s2=>M4,s3=>M5,c0=>M6);
END STRUCT;
```

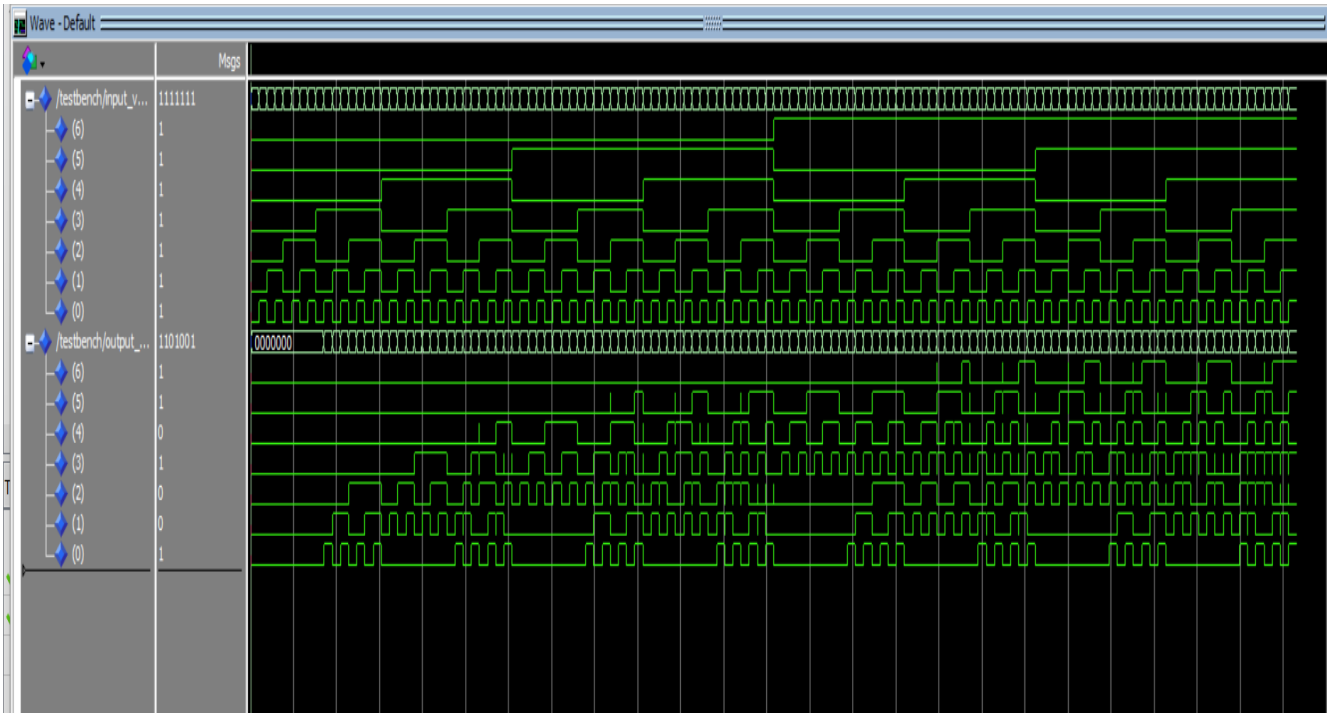
2.2 RTL VIEW





3. Observations:

After you run the analysis of code with no error. The next step is to run the simulation. For this we again use the Modelsim - Altera Software. We also need the Testbench and the tracefile (already provided) to run the simulation. The simulation is shown below.



3.1 TRANSCRIPT(all test cases passed successfully)

```
Transcript
# Errors: 0, Warnings: 0
#
# vcom -93 -work work {C:/Users/ARYA AGARWAL/Desktop/LAB5/Testbench.vhdl}
# Model Technology ModelSim - Intel FPGA Edition vcom 2020.1 Compiler 2020.02 Feb 28 2020
# Start time: 15:49:47 on Sep 01,2022
# vcom -reportprogress 300 -93 -work work C:/Users/ARYA AGARWAL/Desktop/LAB5/Testbench.vhdl
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Compiling entity Testbench
# -- Compiling architecture Behave of Testbench
# End time: 15:49:47 on Sep 01,2022, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
#
# vsim -t 1ps -L altera -L lpm -L sgate -L altera_mf -L altera_lnsim -L fiftyfivenm -L rtl_work -L work -voptargs="+acc"  Testbench
# vsim -t 1ps -L altera -L lpm -L sgate -L altera_mf -L altera_lnsim -L fiftyfivenm -L rtl_work -L work -voptargs=""+acc"" Testbench
# Start time: 15:49:47 on Sep 01,2022
# Loading std.standard
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading work.testbench(behave)
# Loading work.dut(dutwrap)
# Loading work.gates
# Loading work.multiplier(struct)
# Loading work.and_2(equations)
# Loading work.adder_sub(struct)
# Loading work.xor_2(equations)
# Loading work.full_adder_using_nand(trivial)
# Loading work.nand_2(equations)
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 2432 ns  Iteration: 0  Instance: /testbench

VSIM 2>
Transcript  Wave  Dataflow  Objects  Processes  Library  sim
```

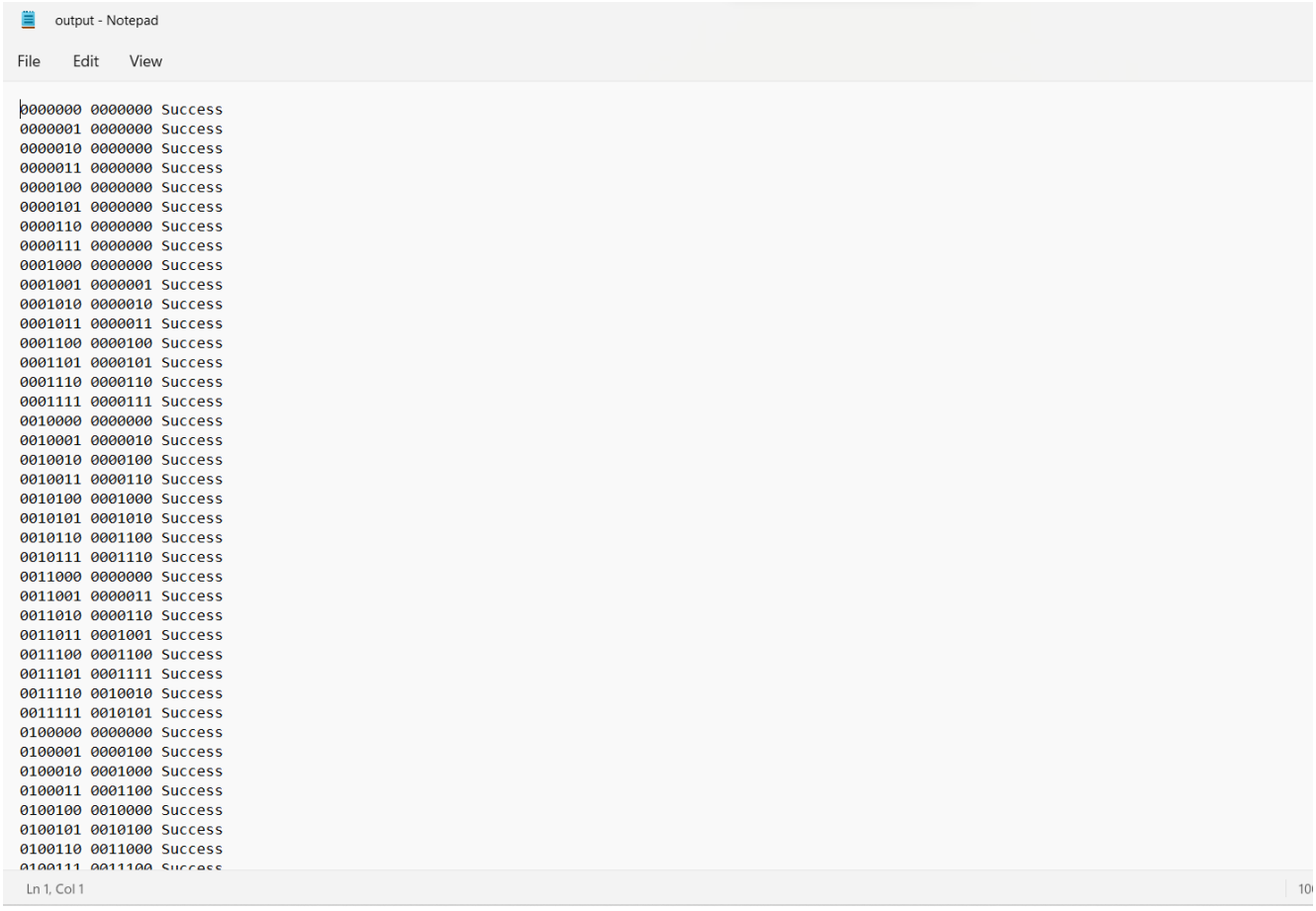
4. SCANCHAIN

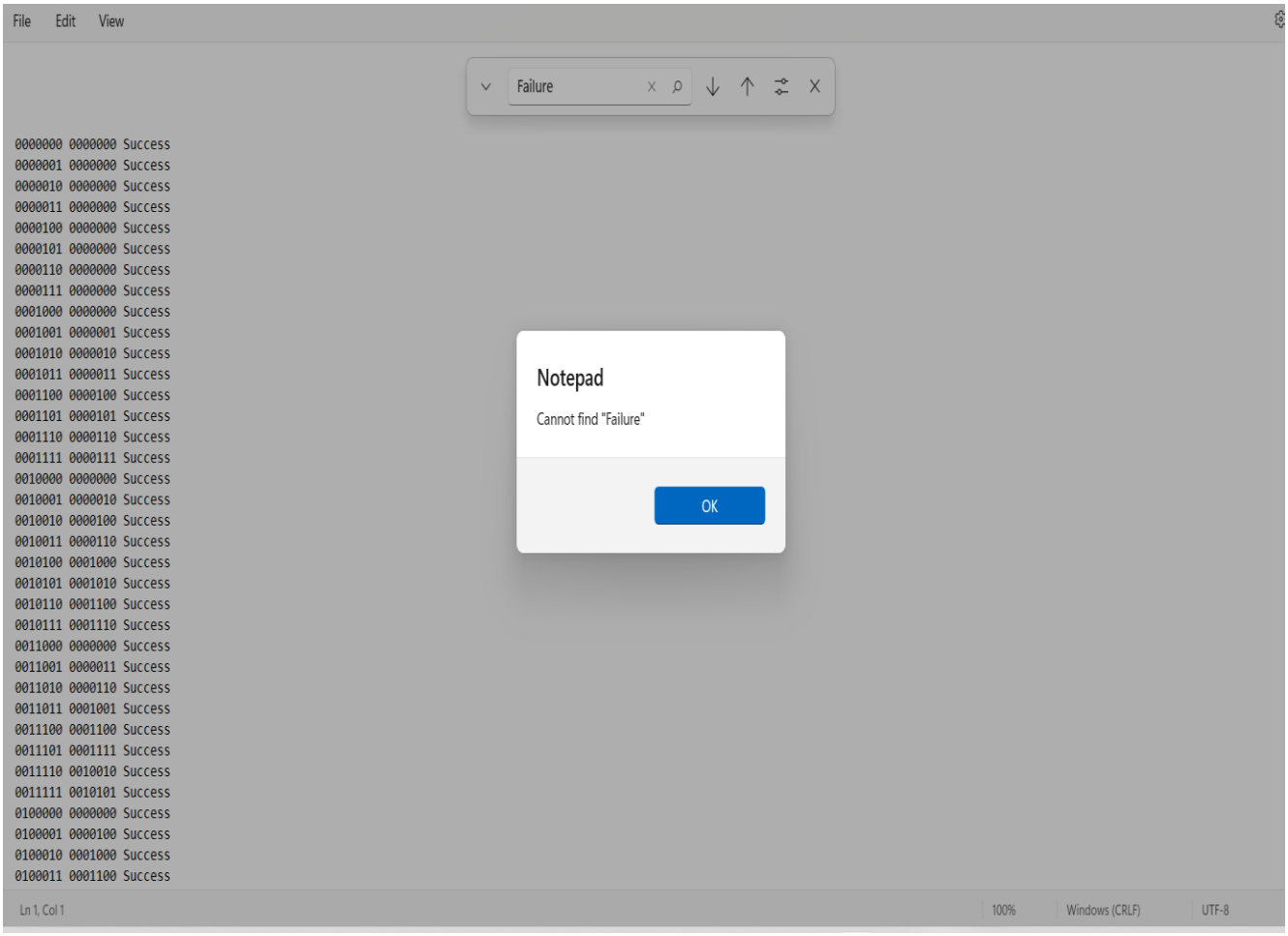
After running the code successfully, we have to connect our design to Xenon. Then we were provided with the scanchain files. After adding the Top-level file from the scanchain files to our main code file. Then making some minor changes in the top level file and then set it as top level entity. Then Copile the whole design. Once the design is compiled make the svf file and throw it in our Urjtag after connecting the Xen10 board to our PC. Before Using the Xenon Board, it should be checked thoroughly. We have to perform all the tests before we use it. Once the Testing is done we can throw our svf file in our Xen10 board using UrJTAG. Then using scanchain we can check our logic.

The benefit of using SCANCHAIN over Xen10 is that if we want to check an 9 output bit device we can't check it using Xen10 board because it only contains 8 LEDs. Another benefit is that scanchain is much more faster than Xen10.

After doing all the necessary steps scanchain will create an output file which will show how many cases passed.

Here is my output file. My design did not contain any test case failure.




File Edit View
Failure
0000000 0000000 Success
0000001 0000000 Success
0000010 0000000 Success
0000011 0000000 Success
0000100 0000000 Success
0000101 0000000 Success
0000110 0000000 Success
0000111 0000000 Success
0001000 0000000 Success
0001001 0000001 Success
0001010 0000010 Success
0001011 0000011 Success
0001100 0000100 Success
0001101 0000101 Success
0001110 0000110 Success
0001111 0000111 Success
0010000 0000000 Success
0010001 0000010 Success
0010010 0000100 Success
0010011 0000110 Success
0010100 0001000 Success
0010101 0001010 Success
0010110 0001100 Success
0010111 0001110 Success
0011000 0000000 Success
0011001 0000011 Success
0011010 0000110 Success
0011011 0001001 Success
0011100 0001100 Success
0011101 0001111 Success
0011110 0010010 Success
0011111 0010101 Success
0100000 0000000 Success
0100001 0000100 Success
0100010 0001000 Success
0100011 0001100 Success
Notepad
Cannot find "Failure"
OK
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8