VHDL Programming 2
Write-Up Documentation

Aubrey McKinney

UIN: 01243380

ECE 341: Digital System Design

Spring 2025

17 February 2025

ODU Honor pledge

"I pledge to support the Honor System of Old Dominion University. I will refrain from any form of academic dishonesty or deception, such as cheating or plagiarism. I am aware that as a member of the academic community it is my responsibility to turn in all suspected violation of the Honor Code. I will report to a hearing if summoned."

Introduction

Continuing on with learning VHDL and the Aldec workspace, we will be applying what we have learned in lecture so far to design and create a set of two-input VHDL models for each of the basic gates, and one-input for the NOT gate. In the second part of our design, we will be writing a VHDL description of the combinational circuit using concurrent statements.

Data Analysis: Part I

For creating each of the models for the basic logic gates, a new Aldec workspace is created using version 11.1. A design is then created in the workspace so that we can add our VHDL source code for each of the gates. For testing each of the different scenarios the following test bench is implemented:

```
process
begin

    A <= '0';
    B <= '0';
    wait for 100 ns;
    A <= '1';
    B <= '0';
    wait for 100 ns;
    A <= '0';
    B <= '1';
    wait for 100ns;
    A <= '1';
    B <= '1';
    wait for 100ns;
    A <= '1';
    B <= 'X';
    wait for 100ns;
    A <= 'X';
    B <= '1';
    wait for 100ns;
    A <= '0';
    B <= 'X';
    wait for 100ns;
    A <= 'X';
    B <= '0';
    wait for 100ns;
    A <= 'X';
    B <= 'X';
    wait for 100ns;
    end process;
```

Figure 2.1: Test bench code testing for each case.

Using each of these cases, we can compare our results against that of the truth table for each of the gates.

AND Gate

Starting with the AND gate, we have the following truth table and waveform:

AND Gate Truth Table		
Inputs		**Output**
A	***B***	***F***
0	0	0
0	1	0
1	0	0
X	X	X
1	1	1
1	X	X
X	1	X

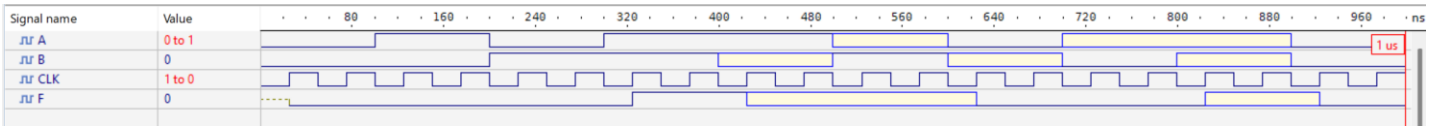

Figure 2.2: AND gate waveform results.

We can see that for all the different inputs we should expect a result of 0 except when both A and B are 1 or a don't care. As we can see by the above results, our waveform concurs with our truth table. When CLK is 0 the gate will not change when CLK is 1 the output can then change depending on the inputs.

OR Gate

OR Gate Truth Table		
Inputs		**Output**
A	***B***	***F***
0	0	0
0	1	1
1	0	1
1	1	1
0	X	X
X	0	X
1	X	1
X	1	1
X	X	X

Figure 2.3: OR gate waveform results.

Again, as we can see by the above waveform, our results concur with our truth table. When the clock is on the output can change and the only time the gate outputs a zero is when both inputs our zero. We can see that this is the expected behavior of the OR gate.

NAND Gate

NAND Gate Truth Table		
Inputs		**Output**
A	***B***	***F***
0	0	1
0	1	1
1	0	1
1	1	0
0	X	1
X	0	1
1	X	X
X	1	X
X	X	X

Figure 2.4: NAND gate waveform results.

Looking at the truth table we have made for the NAND gate, comparing it to our waveform we can see that the results concur. We can see that when one of the inputs is X and the other is 0, because it's a NAND gate the output will be a 1 no matter what X is because of the 0. This result further supports the accuracy of our results.

NOR Gate

NOR Gate Truth Table		
Inputs		**Output**
A	***B***	***F***
0	0	1
0	1	0
1	0	0
1	1	0
0	X	X
X	0	X
1	X	0
X	1	0
X	X	X

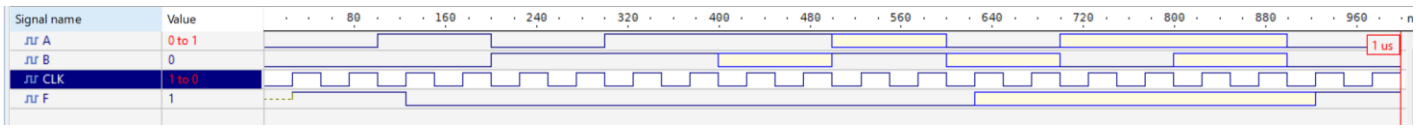

Figure 2.5: NOR gate waveform results.

Comparing the waveform against our truth table, we can again see that our logic for the NOR gate concurs with the truth table. We can see that the only value in which its “on” is when the inputs are 0. The only time the value is X is when the inputs are either 0 and X or X and X which supports our logic.

XOR Gate

XOR Gate Truth Table		
Inputs		**Output**
A	***B***	***F***
0	0	0
0	1	1
1	0	1
1	1	0
0	X	X
X	0	X
1	X	X
X	1	X
X	X	X

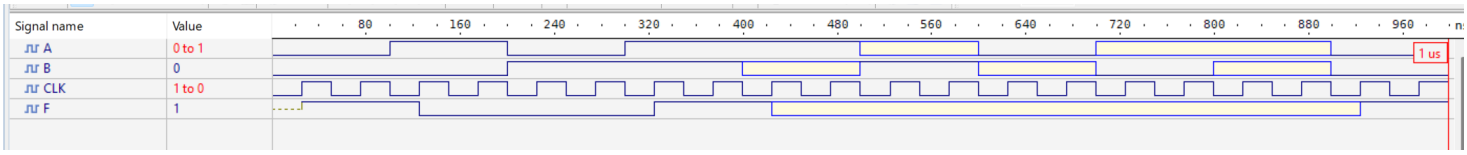

Figure 2.6: XOR gate waveform results.

We can see that for our waveform, the output is 0 when the inputs are both 0,0 and 1,1. We can also see that the output is X for every pair of inputs that contain X. This is expected behavior of the XOR gate and supports our truth table.

XNOR Gate

XNOR Gate Truth Table		
Inputs		**Output**
A	***B***	***F***
0	0	1
0	1	0
1	0	0
1	1	1
0	X	X
X	0	X
1	X	X
X	1	X
X	X	X

Figure 2.7: XNOR gate waveform results.

Again, for the XNOR we can see that if any of the pairs of inputs is X then the output is X. We can also see that when the inputs are 0,0 or 1,1 then we can see that the output is 1. This supports our truth table and is expected behavior of an XNOR gate.

NOT Gate

NOT Gate Truth Table	
Input	**Output**
A	***F***
0	1
1	0
X	X

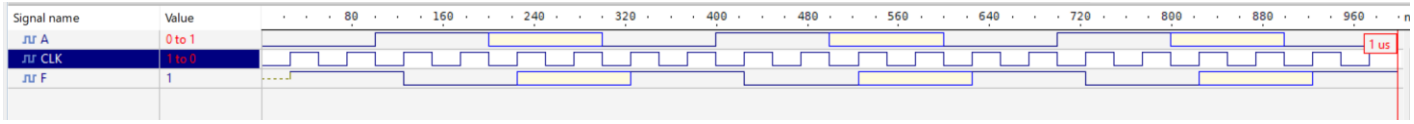

Figure 2.8: NOT gate waveform results.

Finally, we can see that the NOT gate outputs the opposite of the input. We can also see that if the input is an X, then the output is also an X. Our truth table supports our results.

Data Analysis: Part II

For the Second part of our assignment, we will be implementing the following combinational circuit using concurrent statements. Each gate will have a 5-ns delay, except the inverter will have a 2-ns delay.

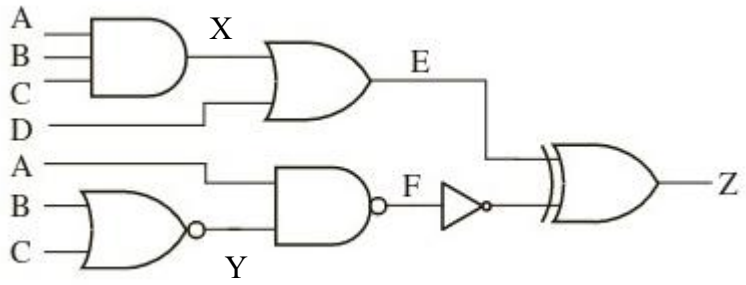

Figure 2.9: Problem 2.4 Combinational Circuit.

Letting the output of the AND and NOR gate be X and Y we can write the following program and Truth table:

Combinational Circuit Results									
Inputs				Outputs					
A	B	C	D	X	Y	E	F	nF	Z
0	0	0	0	0	1	0	1	0	0
0	0	0	1	0	1	1	1	0	1
0	0	1	0	0	0	0	1	0	0
0	0	1	1	0	0	1	1	0	1
0	1	0	0	0	0	0	1	0	0
0	1	0	1	0	0	1	1	0	1
0	1	1	0	0	0	0	1	0	0
0	1	1	1	0	0	1	1	0	1
1	0	0	0	0	1	0	0	1	1
1	0	0	1	0	1	1	0	1	0
1	0	1	0	0	0	0	1	0	0
1	0	1	1	0	0	1	1	0	1
1	1	0	0	0	0	0	1	0	0
1	1	0	1	0	0	1	1	0	1
1	1	1	0	1	0	1	1	0	1
1	1	1	1	1	0	1	1	0	1

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity CombCircuit is
5      port(A, B, C, D:in bit;
6      Z: buffer bit);
7  end CombCircuit;
8
9  architecture CombArch of CombCircuit is
10 signal X,Y,E,F,nF: bit;
11
12 begin
13     X <=(A AND B AND C) after 5 ns;
14     E <=(X OR D) after 5 ns;
15     Y <=(B NOR C) after 5 ns;
16     F <=(A NAND Y) after 5 ns;
17     nF <=(NOT F) after 2 ns;
18     Z <=(nF XOR E) after 5 ns;
19 end architecture CombArch;
```

Figure 2.10: Problem 2.4 Combinational Circuit Assembly Program.

Having confirmed the truth table using an online logic simulator, we can now create the following test bench to test our combinational circuit:

```
    -- Add your stimulus here ...
process
    begin

        A <= '0';
        B <= '0';
        C <= '0';
        D <= '0';
        wait for 50 ns;
        A <= '0';
        B <= '0';
        C <= '0';
        D <= '1';
        wait for 50 ns;
        A <= '0';
        B <= '0';
        C <= '1';
        D <= '0';
        wait for 50 ns;
        A <= '0';
        B <= '0';
        C <= '1';
        D <= '1';
        wait for 50 ns;
        A <= '0';
        B <= '1';
        C <= '0';
        D <= '0';
        wait for 50 ns;
        A <= '0';
        B <= '1';
        C <= '0';
        D <= '1';
        wait for 50 ns;
        A <= '0';
        B <= '1';
        C <= '1';
        D <= '0';
        wait for 50 ns;
        A <= '0';
        B <= '1';
        C <= '1';
        D <= '1';
        wait for 50 ns;
        A <= '1';
        B <= '0';
        C <= '0';
        D <= '0';
        wait for 50 ns;
        A <= '1';
        B <= '0';
        C <= '0';
        D <= '1';
        wait for 50 ns;
        A <= '1';
        B <= '0';
        C <= '1';
        D <= '0';
        wait for 50 ns;
        A <= '1';
        B <= '0';
        C <= '1';
        D <= '1';
        wait for 50 ns;
        A <= '1';
        B <= '1';
        C <= '0';
        D <= '0';
        wait for 50 ns;
        A <= '1';
        B <= '1';
        C <= '0';
        D <= '1';
        wait for 50 ns;
        A <= '1';
        B <= '1';
        C <= '1';
        D <= '0';
        wait for 50 ns;
        A <= '1';
        B <= '1';
        C <= '1';
        D <= '1';
        wait for 50 ns;
        end process;
end TB ARCHITECTURE;
```

Figure 2.11: Problem 2.4 Combinational Circuit Test Bench.

Using the above test bench to test all of the different cases, we can now generate our waveform to compare to the truth table made earlier:

Figure 2.12: Problem 2.4 Combinational Circuit Waveform.

As we can see by our above waveform, looking at when the output is 1 and comparing that to the inputs, we can see that the waveform agrees. The only discrepancy to note is that there is sometimes a glitch as we can see in the waveform. The glitch doesn't effect the accuracy of the results and could be due to how fast the inputs are changing.

Conclusion

For the first part of our assignment, we were successfully able to use if statements to successfully implement the seven different gates. The results from our waveforms support our truth tables, and we even made sure to include the outcome of if the input is an X or a "don't care". We were able to verify the results using a testbench for each logic program. For the second part of the assignment, we were successfully able to implement the combinational circuit in Aldec. After creating a truth table and verifying it, we were able to ensure our waveform agrees with the truth table. Besides the small glitch in our combinational circuit waveform, our experiments were very successful.