Experiment 1 Lab Report

EEE3342C - 00012

Yousef Awad

January 2025

Contents

Equipment

For this experiment a Windows 11 computer was used alongside the Xilinx Vivado 2024.2 software, alongside an FPGA board, the BASYS 3 development board. The board specifically only used to ensure the simulation by the Vivado software was accurate in the real world.

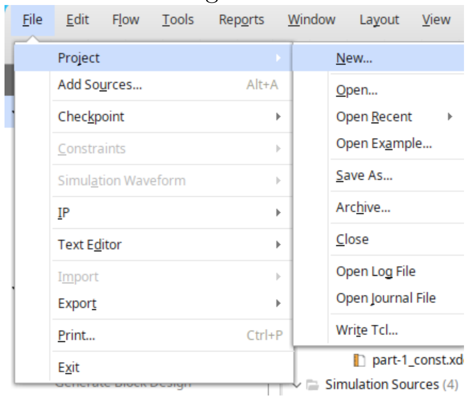
Objective

The objective that was given in the experiment was to introduce the Vivado software as well as the following binary operators: AND, OR, XOR, and the unary operator of the Inverter. Alongside this two circuits were made, one with 2 inputs (of which variations for all the binary operators were made) as well as with 5 inputs (of which all the operators were used). They were then simulated, as well as pushed to the FPGA board for testing and confirmation respectively.

Design Steps (General/Part 1)

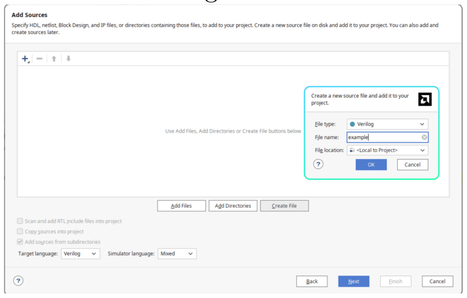
To begin with all of the schematics (listed below in Section 4), I first had to create a project via opening the Vivado software. After opening the software I then went to the top left of the screen, under the "File" tab and then clicked on "Project" and then "New" (as shown on Figure 1).

Figure 1:



After clicking on "New", I then went through the prompts, defining the name of the project (for example "Experiment-1"), as well as its location (for example "C:/user/yousef/Desktop/experiment-1/"). After this I selected the type of project that I am creating a "RTL Project" of which is a full circuit design alongside its constraints as well as allows for simulation. After this, I then create a "source file", which is a Verilog file (ending with .v) that defines the inputs and outputs of a circuit, alongside what it does specifically (See figure 2 for specifics).

Figure 2:



Add Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

Create a new source file and add it to your project.

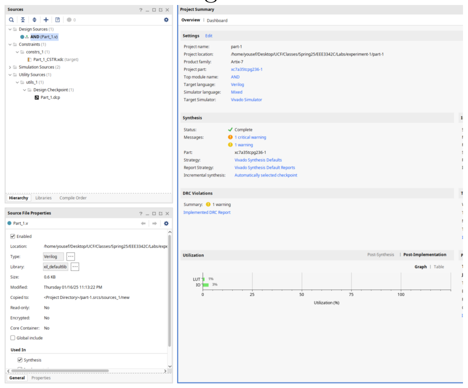
File type: Verilog
File name: example
File location: <Local to Project>

☐ Scan and add RTL include files into project
☐ Copy sources into project
☑ Add sources from subdirectories

Target language: Verilog Simulator language: Mixed

After this I then ignored the rest of the prompts and opened the Verilog file I created during the setup. What opened up for me then, was the project summary screen (example shown in Figure 3), with to the left of it being the Sources window showing the "Design Sources" (where we design the circuit), the "Constraints" (where we define the ports on the FPGA as well as how it should actually simulate it), as well as the "Simulation Sources" (which let's us create a test-bench to test the circuit we made in the Design Sources).

Figure 3:



Design Sources (1)
AND (Part_1.v)
Constraints (1)
constrs_1 (1)
Part_1_CSTR.xdc (target)
Simulation Sources (2)
Utility Sources (1)
utils_1 (1)
Design Checkpoint (1)
Part_1.dcp

Part_1.v
☑ Enabled
Location: /home/yousef/Desktop/UCF/Classes/Spring25/EEE3342C/Labs/expe
Type: Verilog
Library: xil_defaultlib
Size: 0.6 KB
Modified: Thursday 01/16/25 11:13:22 PM
Copied to: <Project Directory>/part-1.srcs/sources_1/new
Read-only: No
Encrypted: No
Core Container: No
☐ Global include
Used In
☑ Synthesis

Settings
Project name: part-1
Project location: /home/yousef/Desktop/UCF/Classes/Spring25/EEE3342C/Labs/experiment-1/part-1
Product family: Artix-7
Project part: xc7a35tcpg236-1
Top module name: AND
Target language: Verilog
Simulator language: Mixed
Target Simulator: Vivado Simulator

Synthesis
Status: ✓ Complete
Messages: 1 critical warning
1 warning
Part: xc7a35tcpg236-1
Strategy: Vivado Synthesis Defaults
Report Strategy: Vivado Synthesis Default Reports
Incremental synthesis: Automatically selected checkpoint

DRC Violations
Summary: 1 warning
Implemented DRC Report

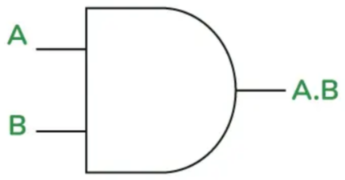
Utilization

Design Steps (Specific)

AND Gate

After this, I then began to create an AND gate, of which looks like the following, at its most basic form:

Figure 4: AND Gate



After inputting the following code for the Verilog module, of which is shown below in Figure 5, I then compiled the schematic to ensure that it is correctly modeled.

Figure 5: Verilog Code for AND Gate

```
module AND(
    input In1, In2,
    output Out
    );

    assign Out = In1 & In2;

endmodule
```

And from that code it compiled this schematic (Figure 6) which has 2 inputs, being A and B, as well as one output, being Out. Alongside this, the device floor plan is shown as well, in Figure 7, as properly implemented according to the basic form shown in Figure 4, as well as the constraints for the simulation to take place (shown in Figure 8).

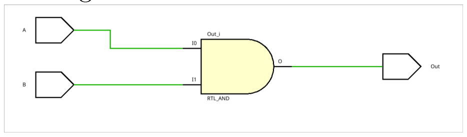


Figure 6: AND Gate Schematic

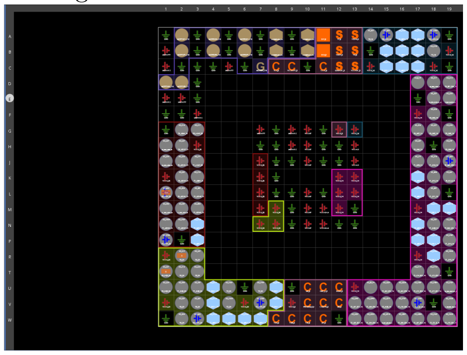
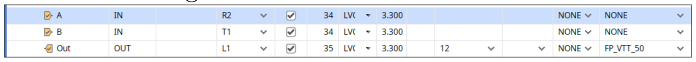


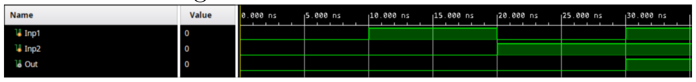
Figure 7: AND Gate Floor Plan



A	IN		R2	✓	34	LV(	3.300				NONE	NONE
B	IN		T1	✓	34	LV(	3.300				NONE	NONE
Out	OUT		L1	✓	35	LV(	3.300		12		NONE	FP_VTT_50

Figure 8: AND Gate Constraints

After this, I then ran the waveform simulation properly, as the circuit was properly implemented, as well as having the correct schematic. The waveform returned the following, as shown in Figure 9, as well as shown in Table 1 with its truth table.



Name	Value
Inp1	0
Inp2	0
Out	0

Figure 9: AND Gate Waveform

Truth Table 1: AND Gate Waveform

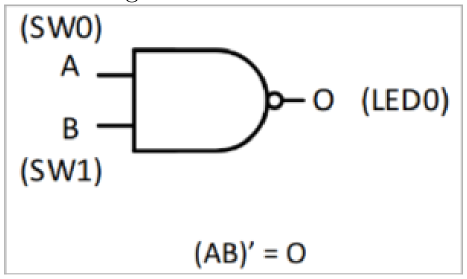
$In1$	$In2$	$In1 \wedge In2$
F	F	F
F	T	F
T	F	F
T	T	T

Now, for context, all the waveform is explaining to us is that, as expected, when both of the Inputs are true, the output shall be true, otherwise the output is false.

NAND Gate

Now to create an NAND gate, of which looks like the following, at its most basic form:

Figure 10: NAND Gate



After inputting the following code for the Verilog module, of which is shown below in Figure 11, I then compiled the schematic to ensure that it is correctly modeled.

Figure 11: Verilog Code for NAND Gate

```
module NAND(
    input A, B,
    output Out
    );

    assign Out = ~(A & B);

endmodule
```

And from that code it compiled this schematic (Figure 12) which has 2 inputs, being A and B, as well as one output, being Out. Alongside this, the device floor plan is shown as well, in Figure 13, as properly implemented according to the basic form shown in Figure 10, as well as the constraints for the simulation to take place (shown in Figure 14).

Figure 12: NAND Gate Schematic

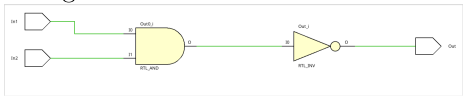


Figure 13: NAND Gate Floor Plan

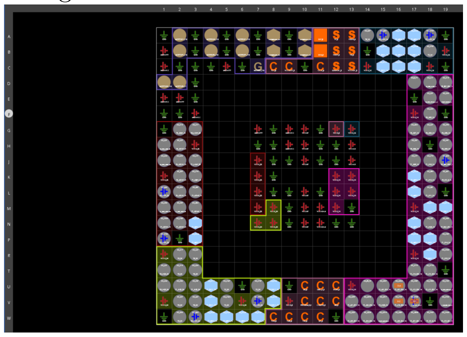
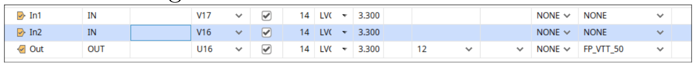


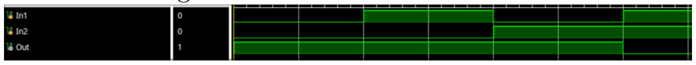
Figure 14: NAND Gate Constraints



In1	IN		V17	✓	14	LV(	3.300				NONE	NONE
In2	IN		V16	✓	14	LV(	3.300				NONE	NONE
Out	OUT		U16	✓	14	LV(	3.300		12		NONE	FP_VTT_50

After this, I then ran the waveform simulation properly, as the circuit was properly implemented, as well as having the correct schematic. The waveform returned the following, as shown in Figure 15, as well as shown in Table 2 with its truth table.

Figure 15: NAND Gate Waveform



In1	0
In2	0
Out	1

Truth Table 2: NAND Gate Waveform

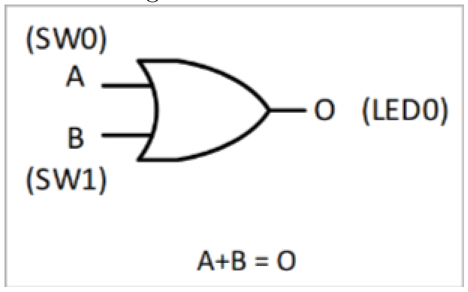
$In1$	$In2$	$\neg(In1 \land In2)$
F	F	T
F	T	T
T	F	T
T	T	F

Now, to explain the waveform, all it is showing is that when both inputs are not enabled, or true as shown in the truth table, the output result shall be true/enabled.

OR Gate

Now to create an OR gate, of which looks like the below figure, at its most basic form.

Figure 16: OR Gate



After inputting the following code for the Verilog module, of which is shown below in Figure 17, I then compiled the schematic to ensure that it is correctly modeled.
And from that code it compiled this schematic (Figure 18) which has 2 inputs,

Figure 17: Verilog Code for OR Gate

```
module OR(
    input A, B,
    output Out
    );

    assign Out = A | B;

endmodule
```

being A and B, as well as one output, being Out. Alongside this, the device floor plan is shown as well, in Figure 19, as properly implemented according to the basic form shown in Figure 16, as well as the constraints for the simulation to take place (shown in Figure 20).

Figure 18: OR Gate Schematic

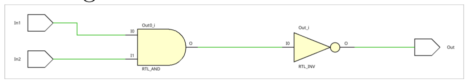


Figure 19: OR Gate Floor Plan

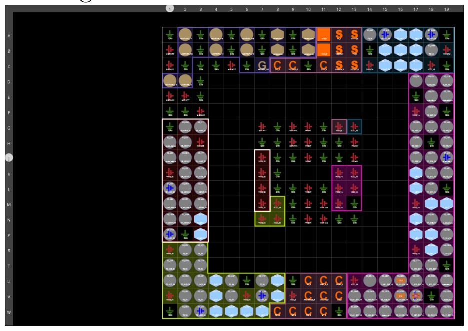
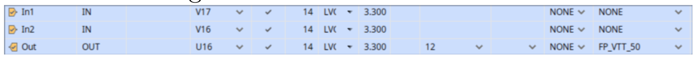


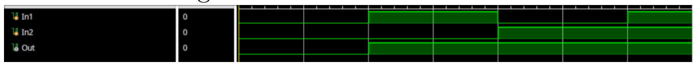
Figure 20: OR Gate Constraints



In1	IN		V17	✓	14	LV(	3.300			NONE	NONE
In2	IN		V16	✓	14	LV(	3.300			NONE	NONE
Out	OUT		U16	✓	14	LV(	3.300	12		NONE	FP_VTT_50

After this, I then ran the waveform simulation properly, as the circuit was properly implemented, as well as having the correct schematic. The waveform returned the following, as shown in Figure 21, as well as shown in Table 3 with its truth table.

Figure 21: OR Gate Waveform



Truth Table 3: OR Gate Waveform

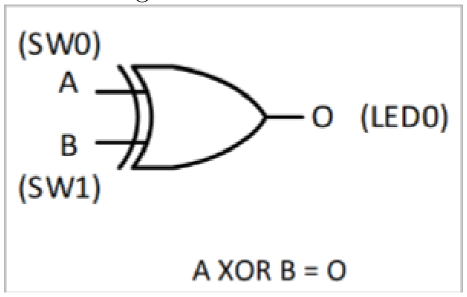
$In1$	$In2$	$In1 \vee In2$
F	F	F
F	T	T
T	F	T
T	T	T

The waveform, in this case, is specifically showing that, when translated to a proper truth table, the output is On/True only if at least one input is enabled/true.

XOR Gate

Now to create an XOR gate, of which looks like the following, at its most basic form.

Figure 22: XOR Gate



After inputting the following code for the Verilog module, of which is shown below in Figure 23, I then compiled the schematic to ensure that it is correctly modeled.

Figure 23: Verilog Code for XOR Gate

```
module XOR(
    input A, B,
    output Out
    );

    assign Out = A ^ B;

endmodule
```

And from that code it compiled this schematic (Figure 24) which has 2 inputs, being A and B, as well as one output, being Out. Alongside this, the device floor plan is shown as well, in Figure 25, as properly implemented according to the basic form shown in Figure 22, as well as the constraints for the simulation to take place (shown in Figure 26).

Figure 24: XOR Gate Schematic

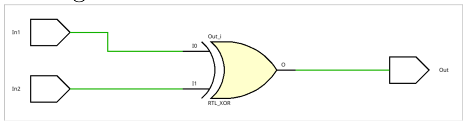


Figure 25: XOR Gate Floor Plan

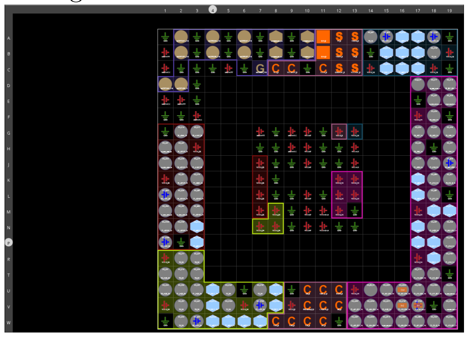
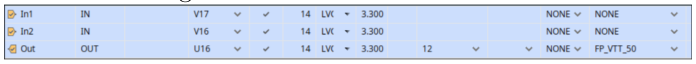


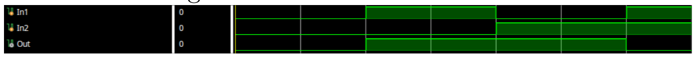
Figure 26: XOR Gate Constraints



In1	IN	V17	✓	14	LV(	3.300			NONE	NONE
In2	IN	V16	✓	14	LV(	3.300			NONE	NONE
Out	OUT	U16	✓	14	LV(	3.300	12		NONE	FP_VTT_50

After this, I then ran the waveform simulation properly, as the circuit was properly implemented, as well as having the correct schematic. The waveform returned the following, as shown in Figure 27, as well as shown in Table 4 with its truth table.

Figure 27: XOR Gate Waveform



In1	0
In2	0
Out	0

Truth Table 4: XOR Gate Waveform

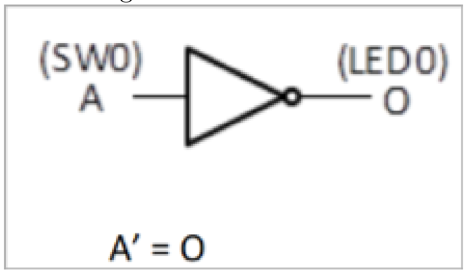
$In1$	$In2$	$In1 \oplus In2$
F	F	F
F	T	T
T	F	T
T	T	F

The waveform in this instance shows that the output is only true/enabled only when ONE, not both, input is enabled/true.

Inverter Gate

Now to create an Inverter gate, of which looks like the following, at its most basic form.

Figure 28: Inverter Gate



After inputting the following code for the Verilog module, of which is shown below in Figure 29, I then compiled the schematic to ensure that it is correctly modeled.

Figure 29: Verilog Code for Inverter Gate

```
module INVERTER(
    input A,
    output Out
    );

    assign Out = ~A;

endmodule
```

And from that code it compiled this schematic (Figure 30) which has 1 input, being A, as well as one output, being Out. Alongside this, the device floor plan is shown as well, in Figure 31, as properly implemented according to the basic form shown in Figure 28, as well as the constraints for the simulation to take place (shown in Figure 32).

Figure 30: Inverter Gate Schematic

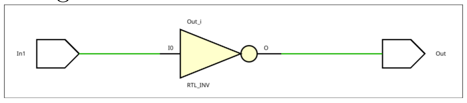


Figure 31: Inverter Gate Floor Plan

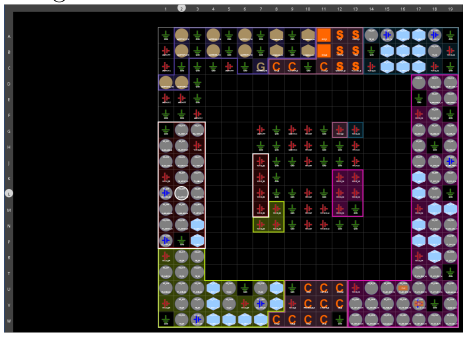
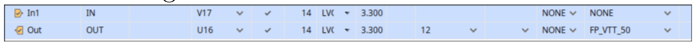


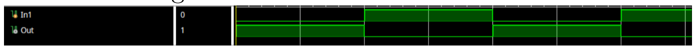
Figure 32: Inverter Gate Constraints



In1	IN		V17	✓	14	LV(	3.300				NONE	NONE
Out	OUT		U16	✓	14	LV(	3.300		12		NONE	FP_VTT_50

After this, I then ran the waveform simulation properly, as the circuit was properly implemented, as well as having the correct schematic. The waveform returned the following, as shown in Figure 33, as well as shown in Table 5 with its truth table.

Figure 33: Inverter Gate Waveform



Truth Table 5: Inverter Gate Waveform

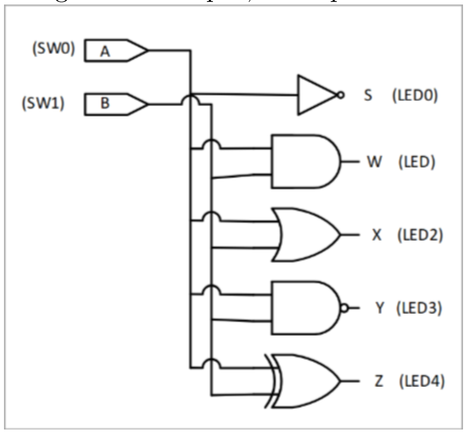
$In1$	$\neg(In1)$
F	T
F	T
T	F
T	F

The waveform here shows that the inverter simply turns the current state to the opposite type, for example from true to false and vice versa.

2 Input 5 Output Gate

Now to create the 2 Input, 5 Outputs gate, of which looks like the following, at its most basic form:

Figure 34: 2 Input, 5 Outputs Gate



After inputting the following code for the Verilog module, of which is shown below in Figure 35, I then compiled the schematic to ensure that it is correctly modeled.

Figure 35: Verilog Code for 2 Input, 5 Output Gate

```
module 2_Input_5_Output(
    input A, B,
    output S, W, X, Y, Z
    );

    assign S = ~A;
    assign W = A & B;
    assign X = A | B;
    assign Y = ~(A & B);
    assign Z = A ^ B;

endmodule
```

And from that code it compiled this schematic (Figure 36) which has 2 inputs, being A and B, as well as 5 outputs, being S, W, X, Y, and Z. Alongside this, the device floor plan is shown as well, in Figure 37, as properly implemented according to the basic form shown in Figure 34, as well as the constraints for the simulation to take place (shown in Figure 38).

Figure 36: 2 Input, 5 Output Gate Schematic

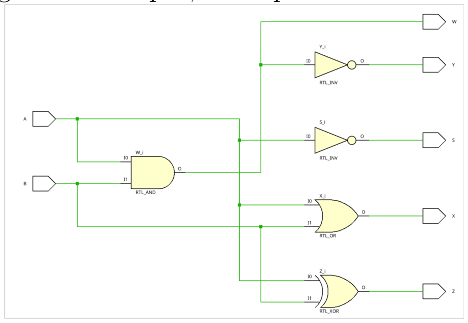


Figure 37: Inverter Gate Floor Plan

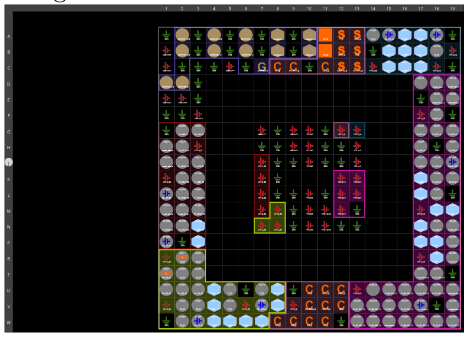
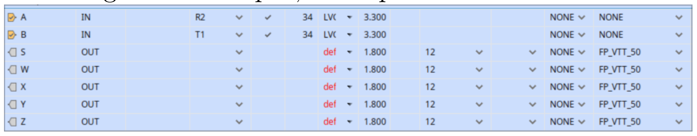


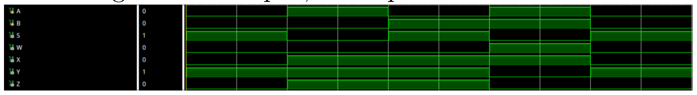
Figure 38: 2 Input, 5 Output Gate Constraints



A	IN		R2	✓	34	LV(	3.300				NONE	NONE
B	IN		T1	✓	34	LV(	3.300				NONE	NONE
S	OUT					def	1.800		12		NONE	FP_VTT_50
W	OUT					def	1.800		12		NONE	FP_VTT_50
X	OUT					def	1.800		12		NONE	FP_VTT_50
Y	OUT					def	1.800		12		NONE	FP_VTT_50
Z	OUT					def	1.800		12		NONE	FP_VTT_50

After this, I then ran the waveform simulation properly, as the circuit was properly implemented, as well as having the correct schematic. The waveform returned the following, as shown in Figure 39, as well as shown in Table 6 with its truth table.

Figure 39: 2 Input, 5 Output Gate Waveform



Truth Table 6: 2 Input, 5 Output Gate Waveform

A	B	S	W	X	Y	Z
F	T	T	F	F	T	F
F	T	T	F	T	T	T
T	F	F	F	T	T	T
T	F	F	T	T	F	F

The waveform here is showing that all S does is simply INVERT the value of A, and is independent from the rest. W is simply an AND gate between the 2 inputs, and also is independent of the rest of the operations. X is an OR gate between the 2 inputs, and, again, is independent of the rest of the operations being done. Y is a NAND gate and, again, is independent of the rest of the operations. And Z is a XOR gate, and like the rest before it, is independent of the operations that go on in parallel with it.

Test Steps

The designs were tested by first running them through a simulation based off of a test bench (of which you see in the waveform figures in Section 4). Each input was set to cycle so that every single permutation was shown in the test bench. After testing, and proving correct from hand calculations, the design was then programmed onto the Basys 3 board, and all of the switches were then tested manually, so that they can be confirmed for a third time.

Conclusion

The entire experiment was a glaring success. The results were perfectly as expected, and all tests/checks agreed with one another harmoniously. Alongside this, the testing of the gate logic helped with my understanding of how they function as well as cemented them with the lab reports summarization.