CSCI 355 - Lab 2 Report

Student: Luka Karanovic
Student #: 665778833
Instructor: Ajay Shrestha

1 - Pre-Lab Answers

4.1 NAND Equivalent for Sum-of-Products

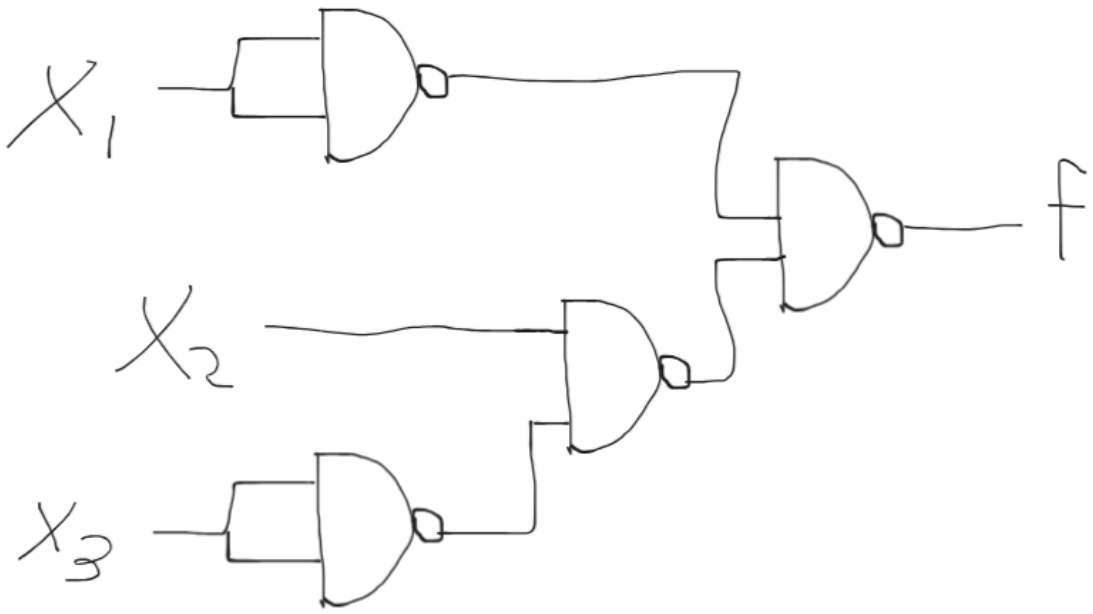
Truth Table:

x1	x2	x3	f(x1, x2, x3)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Boolean Algebra:

$$x_1 + (x_2\overline{x_3}) = \overline{\overline{x_1 + (x_2\overline{x_3})}} = \overline{\overline{x_1}\ \overline{(x_2\overline{x_3})}}$$

Circuit using only NAND Gates:



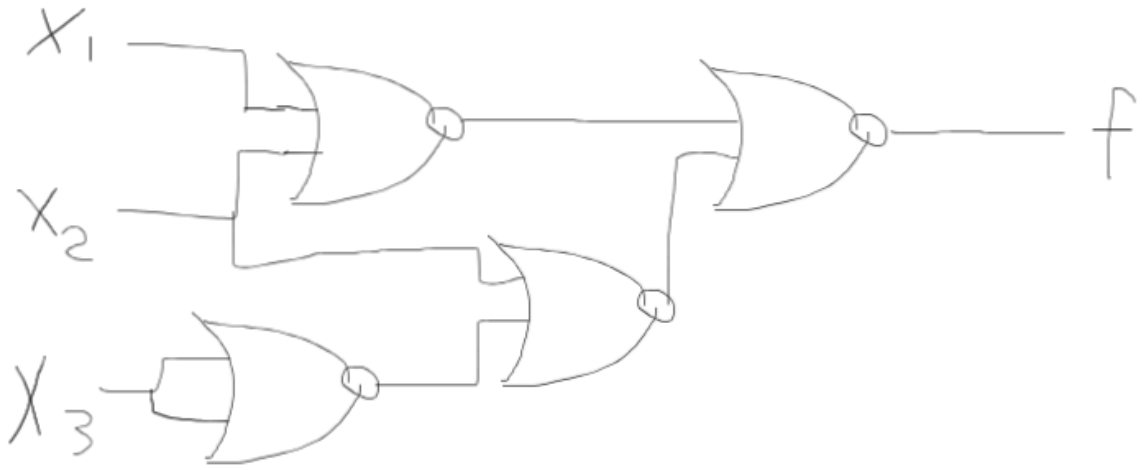
4.2 NOR Equivalent of Product-of-Sums
Truth Table:

x1	x2	x3	f(x1, x2, x3)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

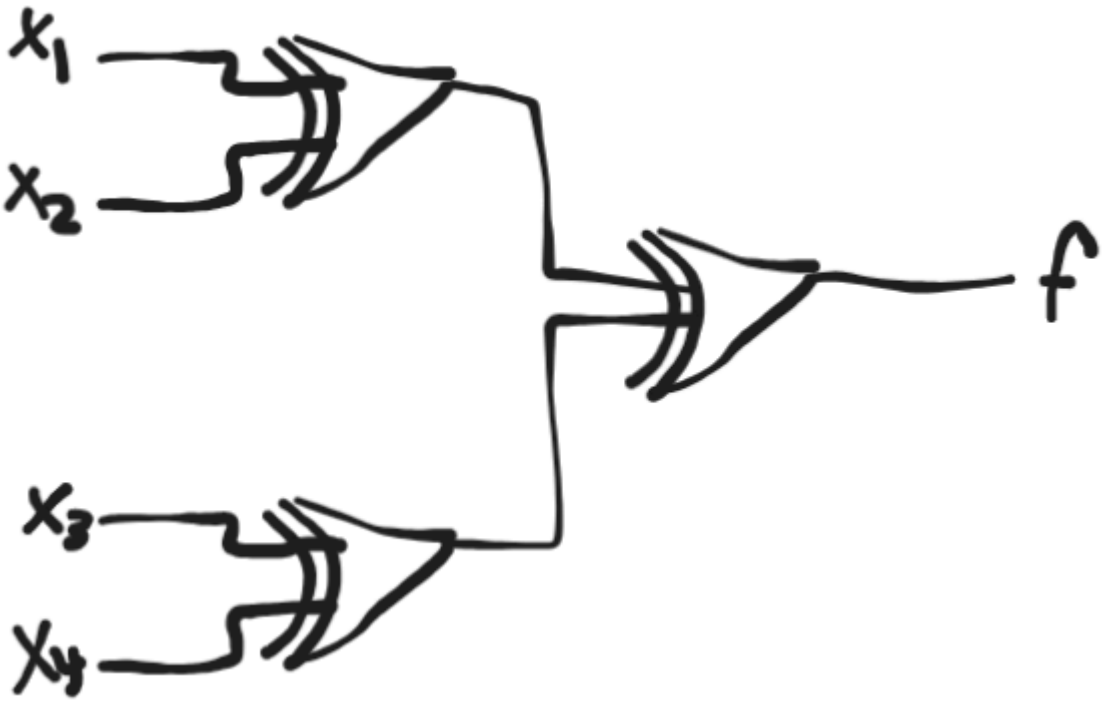
Boolean Algebra:

$$(X_1+X_2)(X_2+\overline{X_3}) = \overline{\overline{(X_1+X_2)(X_2+\overline{X_3})}} = \overline{\overline{(X_1+X_2)} + \overline{(X_2+\overline{X_3})}}$$

Circuit using only NOR Gates:



4.3 A 4-input XOR using 2-input XOR Gates
Diagram:



Truth Table:

x1	**x2**	**x3**	**x4**	**f**	**ODD/EVEN number of 1s**
0	0	0	0	0	even
0	0	0	1	1	odd
0	0	1	0	1	odd
0	0	1	1	0	even
0	1	0	0	1	odd
0	1	0	1	0	even
0	1	1	0	0	even
0	1	1	1	1	odd
1	0	0	0	1	odd
1	0	0	1	0	even
1	0	1	0	0	even
1	0	1	1	1	odd
1	1	0	0	0	even

1	1	0	1	1	odd
1	1	1	0	1	odd
1	1	1	1	0	even

You can tell that if the output is 1, the number of logic 1 inputs was odd. If the output is 0, the number of logic 1 inputs was even. This is because a 2-input XOR gate is 1 if exactly one of its inputs is 1, and because our 3rd 2-input XOR gate takes two 2-input XOR gates as input.

Let's walk through it:

- If we have exactly one 1 logic 1 input, one of the XOR gates will evaluate to 1 and the other to 0, making the 3rd XOR gate evaluates to 1.
- If we have exactly two logic 1 inputs, we have two cases
    - Both logic 1 inputs feed into 1 XOR gate -> that XOR gate becomes 0, the other is 0 (as its inputs are 0 and 0). Both XOR gates are 0 so the 3rd XOR gate evaluates to 0.
    - One logic 1 input feeds into 1 XOR gate -> that XOR gate becomes 1. The other logic 1 input feeds into the other XOR gate -> that XOR gate becomes 1. Both XOR gates are 1 so the 3rd XOR gate evaluates to 0.
- If we have exactly three logic 1 inputs, one XOR gate will have two logic 1 inputs -> that XOR gate will be 0. The other XOR gate will have exactly one logic 1 input -> that XOR gate will be 1. No matter what, one gate is 0 and the other is 1, meaning the 3rd XOR gate evaluates to 1.
- If we have exactly four logic 1 inputs, both XOR gates will evaluate to 0 and therefore the 3rd XOR gate (inputs 0 and 0) will evaluate to 0.

Using this logic backwards, if we know whether the output f is either 0 or 1, we know if we have an even/odd amount of logic 1 inputs.

4.4 XOR Gate using only NAND logic and only NOR logic

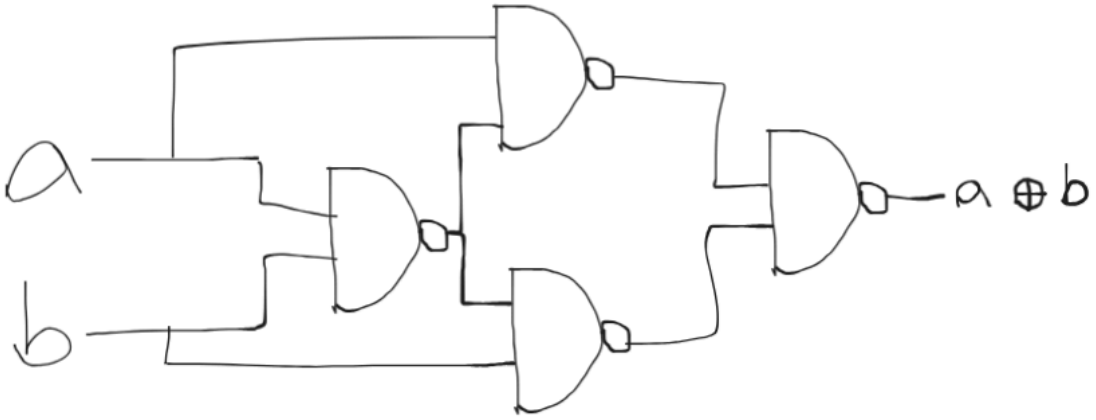
Boolean Algebra:

$$XOR = A\bar{B} + \bar{A}B = A\bar{B} + \bar{A}B + 0 + 0$$

$$= A\bar{B} + \bar{A}B + A\bar{A} + B\bar{B} = A(\bar{A}+\bar{B}) + B(\bar{A}+\bar{B})$$

$$= A(\overline{AB}) + B(\overline{AB}) = \overline{\overline{A(\overline{AB})} + \overline{B(\overline{AB})}} = \left(\overline{A(\overline{AB})}\right)\left(\overline{B(\overline{AB})}\right)$$

XOR Gate using only NAND Gates:

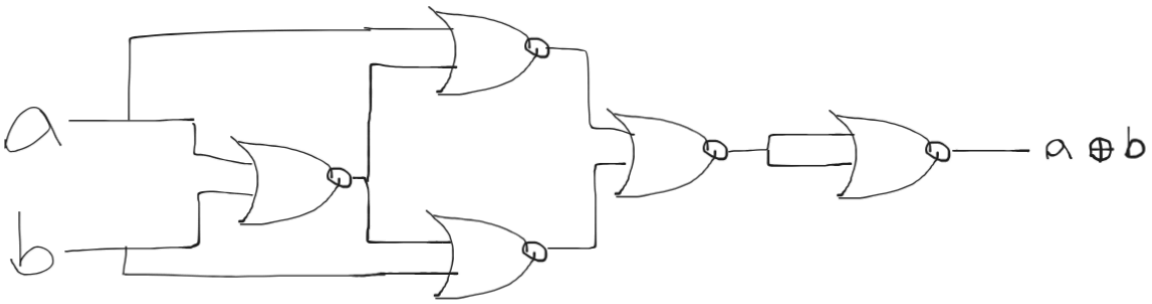


Boolean Algebra:

$$XOR = A\bar{B} + \bar{A}B = A\bar{A} + A\bar{B} + \bar{A}B + B\bar{B} = A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B})$$

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

XOR Gate using only NOR Gates:



2 - Lab Procedures

7.1 NAND Equivalent for Sum-of-Products (from Pre-Lab 4.1)

i) Construct NAND-only version of the Sum-of-Products circuit developed in Prelab 4.1.

Filename: sop_nand_only.v

```
module sop_nand_only(f, x1, x2, x3);
        input x1, x2, x3;
        output f;

        wire n1, n2, n3;
        // n1 = ~(x1x1) = ~x1
        nand(n1, x1, x1);
        // n2 = ~(x3x3) = ~x3
        nand(n2, x3, x3);
```

```
        // n3 = ~(x2~x3)
        nand(n3, x2, n2);
        // ~(~(x2~x3)~x1)
        nand(f, n3, n1);
endmodule
```

ii) Create an excerpt of the XDC file showing the relevant constraints.

File name: lab2_constraints.xdc

```
## SWITCH (SW0) -> Port x1
set_property PACKAGE_PIN V17 [get_ports x1] ;# SW0
set_property IOSTANDARD LVCMOS33 [get_ports x1]
## SWITCH (SW1) -> Port x2
set_property PACKAGE_PIN V16 [get_ports x2] ;# SW1
set_property IOSTANDARD LVCMOS33 [get_ports x2]
## (Only if your module uses x3)
## SWITCH (SW2) -> Port x3
set_property PACKAGE_PIN W16 [get_ports x3] ;# SW2
set_property IOSTANDARD LVCMOS33 [get_ports x3]
## LED (LED0) -> Port f
set_property PACKAGE_PIN U16 [get_ports f] ;# LED0
set_property IOSTANDARD LVCMOS33 [get_ports f]
```

iii) Fill in the appropriate Truth Table and comment if the circuit is working as expected.

x1	**x2**	**x3**	**f(x1, x2, x3)**
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Circuit is working as expected!

7.2 NOR Equivalent of Product-of-Sums

i) Construct the NOR-only version of the Products-of-Sums circuit developed in Prelab 4.2.

Filename: pos_nor_only.v

```
module pos_nor_only(f, x1, x2, x3);

        input x1, x2, x3;
        output f;
        wire n1, n2, n3;
        // n1 = ~(x1 + x2)
        nor(n1, x1, x2);
        // n2 = ~(x3 + x3) = ~x3
        nor(n2, x3, x3);
        // n3 = ~(x2 + ~x3)
        nor(n3, x2, n2);
        // f = ~(~(x2 + ~x3) + ~(x1 + x2)) = (x1 + x2)(x2 + ~x3)
        nor(f, n3, n1);
endmodule
```

ii) XDC file showing the relevant constraints.

File name: lab2_constraints.xdc

```
## SWITCH (SW0) -> Port x1
set_property PACKAGE_PIN V17 [get_ports x1] ;# SW0
set_property IOSTANDARD LVCMOS33 [get_ports x1]
## SWITCH (SW1) -> Port x2
set_property PACKAGE_PIN V16 [get_ports x2] ;# SW1
set_property IOSTANDARD LVCMOS33 [get_ports x2]
## (Only if your module uses x3)
## SWITCH (SW2) -> Port x3
set_property PACKAGE_PIN W16 [get_ports x3] ;# SW2
set_property IOSTANDARD LVCMOS33 [get_ports x3]
## LED (LED0) -> Port f
set_property PACKAGE_PIN U16 [get_ports f] ;# LED0
set_property IOSTANDARD LVCMOS33 [get_ports f]
```

iii) Fill in the appropriate Truth Table and comment if the circuit is working as expected.

x1	**x2**	**x3**	**f(x1, x2, x3)**
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0

1	1	0	1
1	1	1	1

Circuit is working as expected!

7.3 XOR using NAND logics only

i) Construct the NAND-only version of the XOR equivalent circuit developed in Prelab 4.4.

Filename: xor_from_nand.v

```
module xor_from_nand(f, x1, x2);
	input x1, x2;
	output f;
	wire n1, n2, n3;
	// n1 = ~(x1x2)
	nand(n1, x1, x2);
	// n2 = ~(~(x1x2)x1)
	nand(n2, n1, x1);
	// n3 = ~(~(x1x2)x2)
	nand(n3, n1, x2);
	// f = ~( (~(x1x2)x1) (~(x1x2)x2) ) = (x1~x2) + (~x1x2)
	nand(f, n2, n3);
endmodule
```

ii) XDC file showing the relevant constraints.

File name: lab2_xor_constraints.xdc

```
## SWITCH (SW0) -> Port x1
set_property PACKAGE_PIN V17 [get_ports x1] ;# SW0
set_property IOSTANDARD LVCMOS33 [get_ports x1]
## SWITCH (SW1) -> Port x2
set_property PACKAGE_PIN V16 [get_ports x2] ;# SW1
set_property IOSTANDARD LVCMOS33 [get_ports x2]
## LED (LED0) -> Port f
set_property PACKAGE_PIN U16 [get_ports f] ;# LED0
set_property IOSTANDARD LVCMOS33 [get_ports f]
```

iii) Fill in the appropriate Truth Table and comment if the circuit is working as expected.

x1	**x2**	**f(x1, x2)**
0	0	0
0	1	1
1	0	1
1	1	0

Circuit works as expected!

7.4 XOR using NOR logics only

i) Construct the NOR-only version of the XOR equivalent circuit developed in Prelab 4.4.

Filename: xor_from_nor.v

```
module xor_from_nor(f, x1, x2);
	input x1, x2;
	output f;
	wire n1, n2, n3, n4;
	// n1 = ~(x1 + x2)
	nor(n1, x1, x2);
	// n2 = ~(~(x1 + x2) + x1)
	nor(n2, n1, x1);
	// n3 = ~(~(x1 + x2) + x2)
	nor(n3, n1, x2);
	// n4 = ~( (~(~(x1 + x2) + x1)) + (~(~(x1 + x2) + x2)) )
	nor(n4, n3, n2);
	// f = ~( ~( (~(~(x1 + x2) + x1)) + (~(~(x1 + x2) + x2))) ) = (x1~x2) + (~x1x2) <-
confusing i know
	nor(f, n4, n4);
endmodule
```

ii) XDC file showing the relevant constraints.

File name: lab2_xor_constraints.xdc

```
## SWITCH (SW0) -> Port x1
set_property PACKAGE_PIN V17 [get_ports x1] ;# SW0
set_property IOSTANDARD LVCMOS33 [get_ports x1]
## SWITCH (SW1) -> Port x2
set_property PACKAGE_PIN V16 [get_ports x2] ;# SW1
set_property IOSTANDARD LVCMOS33 [get_ports x2]
## LED (LED0) -> Port f
set_property PACKAGE_PIN U16 [get_ports f] ;# LED0
set_property IOSTANDARD LVCMOS33 [get_ports f]
```

iii) Fill in the appropriate Truth Table and comment if the circuit is working as expected.

x1	**x2**	**f(x1, x2)**
0	0	0
0	1	1
1	0	1
1	1	0

Circuit is working as expected!

3 - Short Discussion

a. How you applied the NAND-only SOP and NOR-only POS patterns to your specific functions from the pre-lab.

I built the truth tables and diagrams from the prelab which gave me the solution. I went through each gate and made a separate wire variable for the output. I implemented my Verilog code 1 gate at a time. This was easiest as the gate was meant to be implemented using structural specification.

b. Any constraints or port-naming issues you corrected.

No constraint or port naming issues. Only thing I had to correct was commenting out the port naming code for x3 when I was not using it. This was for the XOR gate implementations, as they only have two inputs x1 and x2.

I solved this by making two separate constraints files: lab2_constraints.xdc and lab2_xor_constraints.xdc.