CS 211, Fall 2022
Laboratory Exercise 4:
Advanced MUX

Instructor: Professor Izidor Gertner
Due October 16, 2022, by 11:59 PM

Zi Xuan Li

Professor Gertner

CS 211

October 16th, 2022

<u>(DID NOT KNOW IF PROFESSOR GERTNER WANTED THIS EXERCISE SUBMITTED WITH THE PHOTOS ALONG WITH ANOTHER LAB REPORT, SO I INCLUDED THE TYPICAL LAB REPORT CONTENTS WHERE THE LAB EXERCISE ENDS)</u>

1. Objective

The objective of this lab exercise was to transition to more advanced work related to the "Schematic Capture" phase of digital circuit design as it is executed in the Quartus II software by closely studying and implementing multiplexers of various size and other related circuits. In doing so, we were able to more firmly grasp the concept of expressing a specification in three different ways (Boolean function, truth table, and schematic diagram). In addition, the simulation technique used in previous labs in which a Vector Waveform File was created as input for the built-in simulation tool was found to be meaningfully useful for the first time, because the circuits explored here a little more complex. In short, the lab demonstrates the practice of Schematic Capture, waveform simulation, and verification using the Quartus II software through the study and design of the following digital circuits:

- 4-to-1 multiplexer
- 2-to-1 8-bit vector multiplexer

2. Functionality and Specifications

2.1 4-to-1 Multiplexer

A 4-to-1 multiplexer is, most basically, a device that "selects" from 4 input pins. The selected signal is forwarded to the output pin and is chosen by the values of the selector pins. Note that in a previous lab we only worked with a 2-to-1 multiplexer, which requires just one selector input. Since the 4-to-1 multiplexer selects from 4 inputs, 2 bits of information are required for the selection, meaning there are two selector inputs, for a total of 6 input ports and one output port.

2.1.1 Logic Function

Assuming our four inputs are I_0, I_1, I_2, and I_3 and our two selectors are S_0 and S_1, a 4-to-1 multiplexer can be expressed with the following Boolean function:

CS 211, Fall 2022

Laboratory Exercise 4: Advanced MUX

Instructor: Professor Izidor Gertner
Due October 16, 2022, by 11:59 PM

$$f(I_0, I_1, I_2, I_3, S_0, S_1) = (I_0 * \overline{S_0} * \overline{S_1}) + (I_1 * S_0 * \overline{S_1}) + (I_2 * \overline{S_0} * S_1) + (I_3 * S_0 * S_1)$$

This is somewhat difficult to digest, so let's put a few examples in a truth table to better see what is going on (we're not going to include the comprehensive table in this manner as it would require 2^6=64 rows):

I_0	I_1	I_2	I_3	S_0	S_1	$(I_0 * \overline{S_0} * \overline{S_1})$	$(I_1 * S_0 * \overline{S_1})$	$(I_2 * \overline{S_0} * S_1)$	$(I_3 * S_0 * S_1)$	f
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0
0	0	1	0	0	1	0	0	0	0	0
0	0	1	1	1	1	0	0	0	1	1
…	…	…	…	…	…	…	…	…	…	…

Figure 1. Partial truth table for 4-to-1 multiplexer (fill in the table)

What we see here is that in each of the terms of the functions, the selectors (or their inversions) are "ANDed" in such a way that the result is the input's value only if the input is "selected," otherwise it is always "0". When the terms are added (a four way OR is applied, it is only possible that the selected input's value is "1", meaning that value shows up in the output. The above partial truth table is useful for understanding the Boolean function. Due to the nature of what a multiplexer does, it is possible to create a much simpler table (this will be useful to us when simulating and verifying):

S_0	S_1	f
0	0	$(I_0 * \overline{S_0} * \overline{S_1})$
0	1	$(I_1 * S_0 * \overline{S_1})$
1	0	$(I_2 * \overline{S_0} * S_1)$
1	1	$(I_3 * S_0 * S_1)$

Figure 2. Simplified but comprehensive truth table for 4-to-1 multiplexer. (fill in the table)

2.1.1 Logic Function

Having thoroughly specified the 4-to-1 multiplexer in theory using a Boolean function and a truth table, it's time to design the circuit using the Quartus II Block Diagram Editor. As an exercise, we designed two versions of the 4-to-1 multiplexer: one using only 2-input NAND gates, and the other using the 2-to-1 multiplexer symbol we created in the previous lab.

In order to design the multiplexer using only 2-input NAND gates, it was useful to first design 3 and 3 input NAND gates. After creating block diagrams for those, we use Quartus II to generate symbols that we can later import into the block diagram for the multiplexer.

CS 211, Fall 2022
Laboratory Exercise 4: Advanced MUX

Instructor: Professor Izidor Gertner
Due October 16, 2022, by 11:59 PM

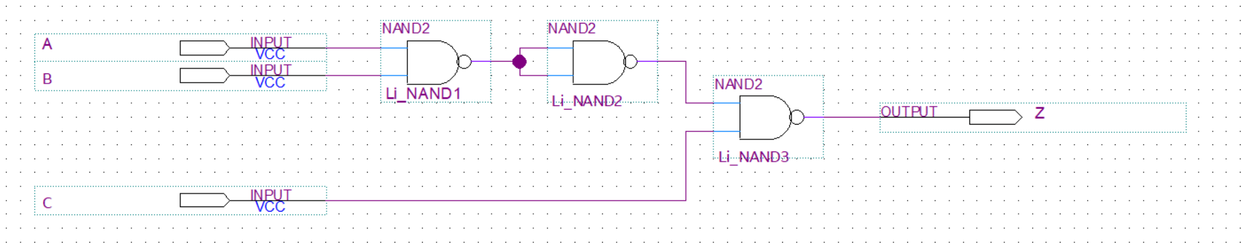


Figure 3. 3-input NAND block created using only 2-input NAND gates.

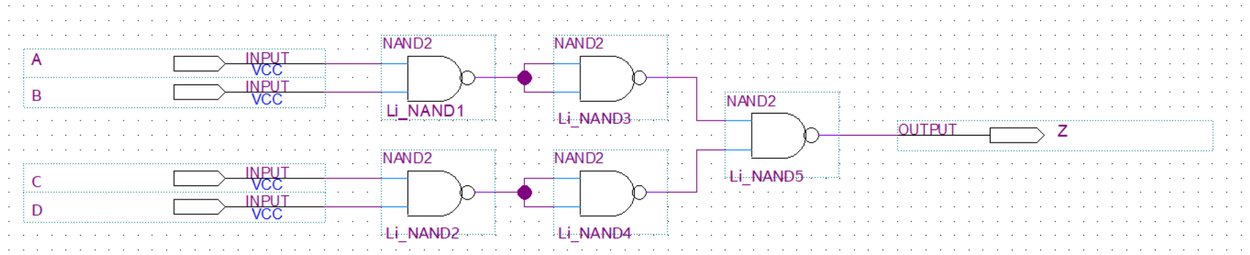


Figure 4. 4-input NAND block created using only 2-input NAND gates.

When designing these, it's important to go through the entire process (Schematic Capture, simulation, verification) for each component. Knowing that these modules work correctly will make it easier to debug circuits that use if something goes wrong.

The 3 and 4 input NAND blocks were created as symbols. We can import them into any Block Diagram the same way as we add any other symbol—symbols created within the project will appear in the Project Directory when using the Symbol Tool. A screenshot of the completed 4-to-1 multiplexer, using only NAND gates, is attached below.

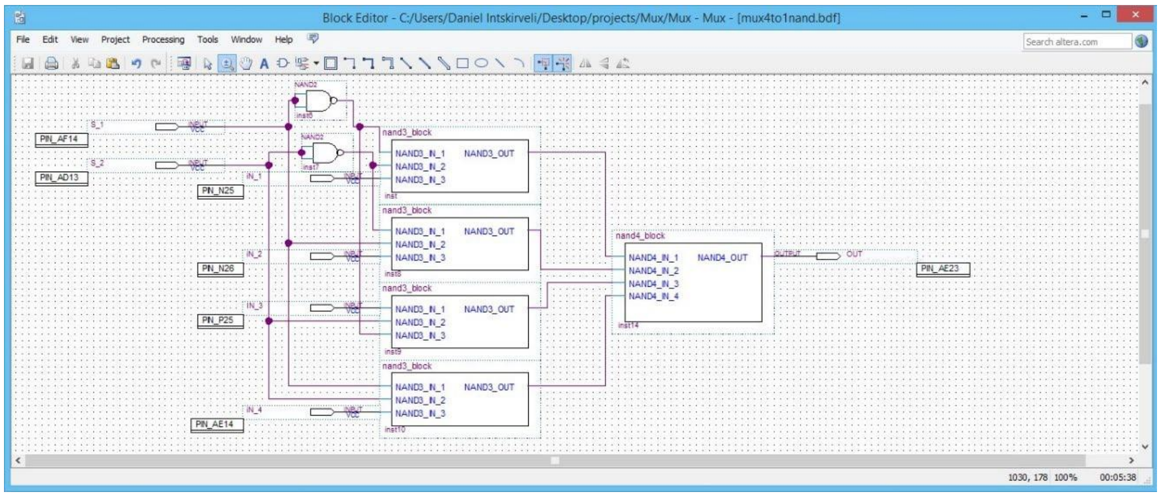


Figure 5. The 4-to-1 multiplexer using only 2-input NAND gates. Note the use of the 3 and 4 input NAND blocks we created earlier, shown in Figures 3, and Figure 4) as well as NAND gates taking the place of NOT gates.

CS 211, Fall 2022

Laboratory Exercise 4: Advanced MUX

Instructor: Professor Izidor Gertner
Due October 16, 2022, by 11:59 PM

In order to create the second version, we must first create a Symbol for the 2-to-1 multiplexer Block Diagram created in the previous lab. Once this is complete, we can use the 2-to-1 multiplexer (which we already simulated and verified) in the new 4-to-1 multiplexer block diagram.

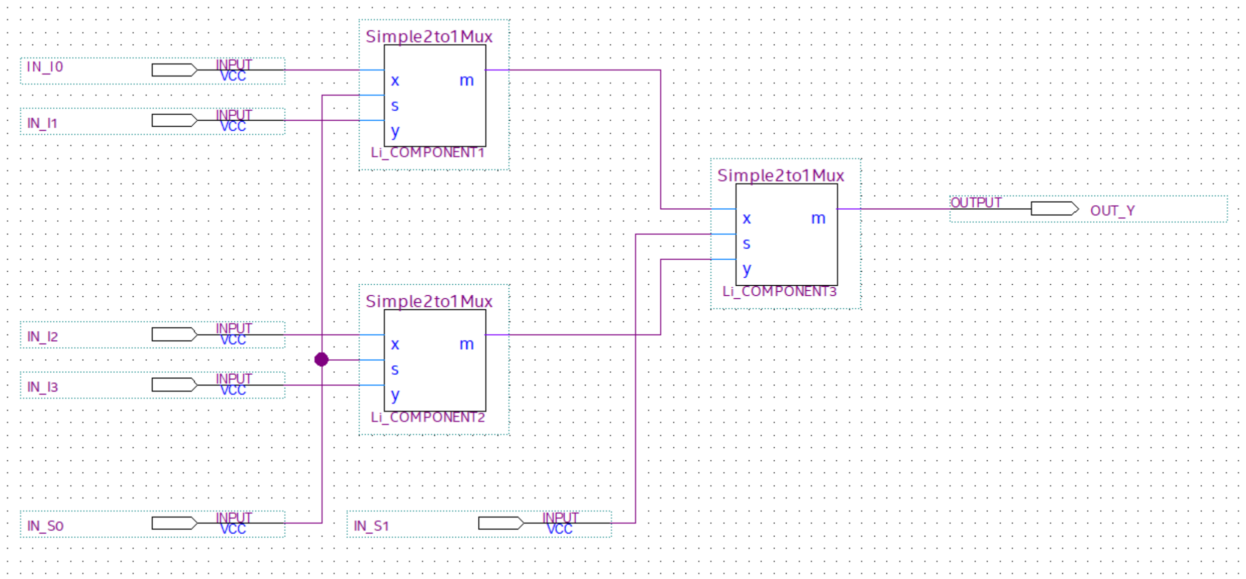


Figure 6. 4-to-1 multiplexer block diagram using the symbol for the 2-to-1 multiplexer.

2.2 2-to-1 8-Bit Vector Multiplexer

A 2-to-1 8-bit vector multiplexer has two vector inputs of 8 bits each, and the selector signal choose amongst them to output to an 8-bit vector output. Therefore, we have 17 input ports (1 selector, 8 inputs ports for the first vector, and 8 input ports for the second vectors) and 8 output ports (for the 8-bit output vector).

2.2.1 Logic Function

In order to simplify the function, we treat each vector as one variable. This will make our lives easier for the theoretical portion of the specification, and we'll overcome the fact that each vector has 8 bits by using 8 2-to-1 multiplexers in the Block Diagram. Put this way, the Boolean function is the same as what we came up with for the 2-to-1 multiplexer in the previous lab, except that it is important to understand that the inputs are vectors, not single bits. Here's the function, with *s* as the selector and *X* and *Y* as the input vectors:

$$\boldsymbol{f(x, y, s) = (x * \bar{s}) + (s * y)}$$

CS 211, Fall 2022
Laboratory Exercise 4:
Advanced MUX

Instructor: Professor Izidor Gertner
Due October 16, 2022, by 11:59 PM

In addition, we note our simplified truth table once more, as it will be useful to us in testing:

S	*f*
0	X
1	Y

Figure 7. Simple truth table for a 2-to-1 multiplexer.

To put it simply, our specification says that if the value of the selector is “0”, the output vector should be the same as the first input vector. If the value of the selector is “1”, the output vector should be the same as the second input vector.

2.2.2 Block Diagram

Let’s create the block diagram. Just as we did when creating the second version of the 4-to 1multiplexer, we will use the 2-to-1 multiplexer symbol. In this case, we’re simply extending the functionality of the 2-to-1 multiplexer to 8 bits by using 8 multiplexers and feeding each of them the same selector signal.

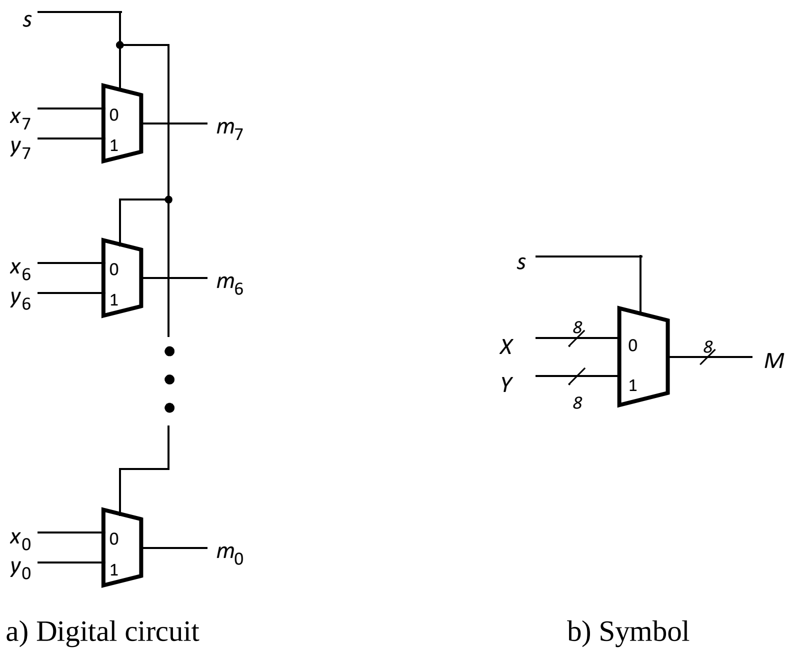


Figure 8. The width of vector input X is 8 bits. The width of vector input Y is 8 bits an eight-bit.

1. Create all truth tables, Boolean functions, and BDF files for all designs.
2. Compile.
3. Simulate to verify correctness.
4. Write a report including all block diagrams and waveforms in simulation. Prove that your design is correct.

CS 211, Fall 2022

Laboratory Exercise 4:

Advanced MUX

Instructor: Professor Izidor Gertner

Due October 16, 2022, by 11:59 PM

3. SIMULATION

3-Input NAND gate:

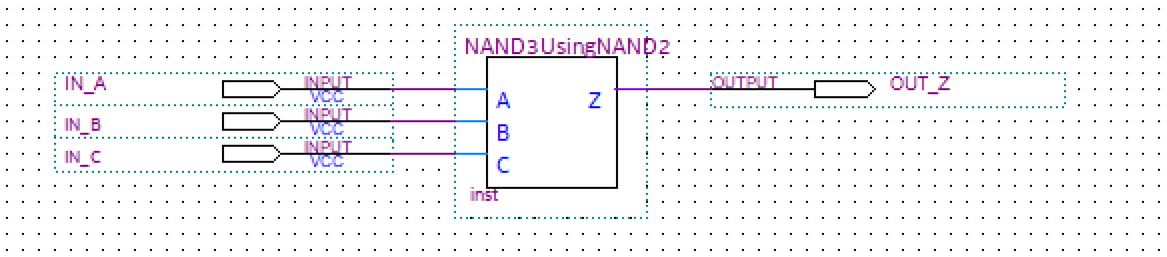
- Truth Table

A	*B*	*C*	*Z*
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

- Boolean Expression

$$Z = \overline{ABC}$$

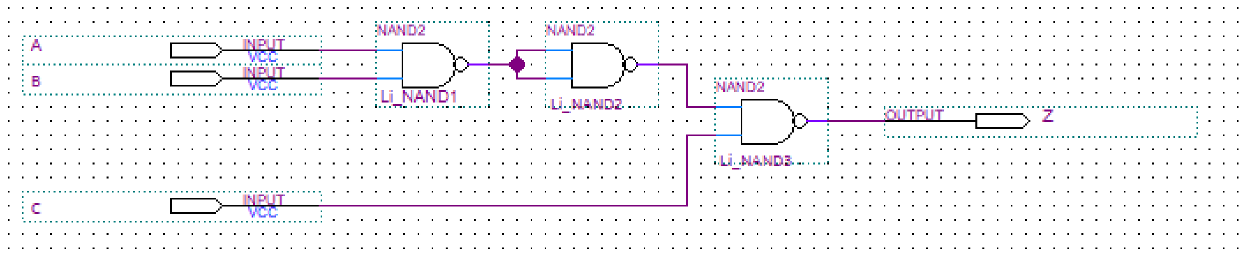
- BDF File



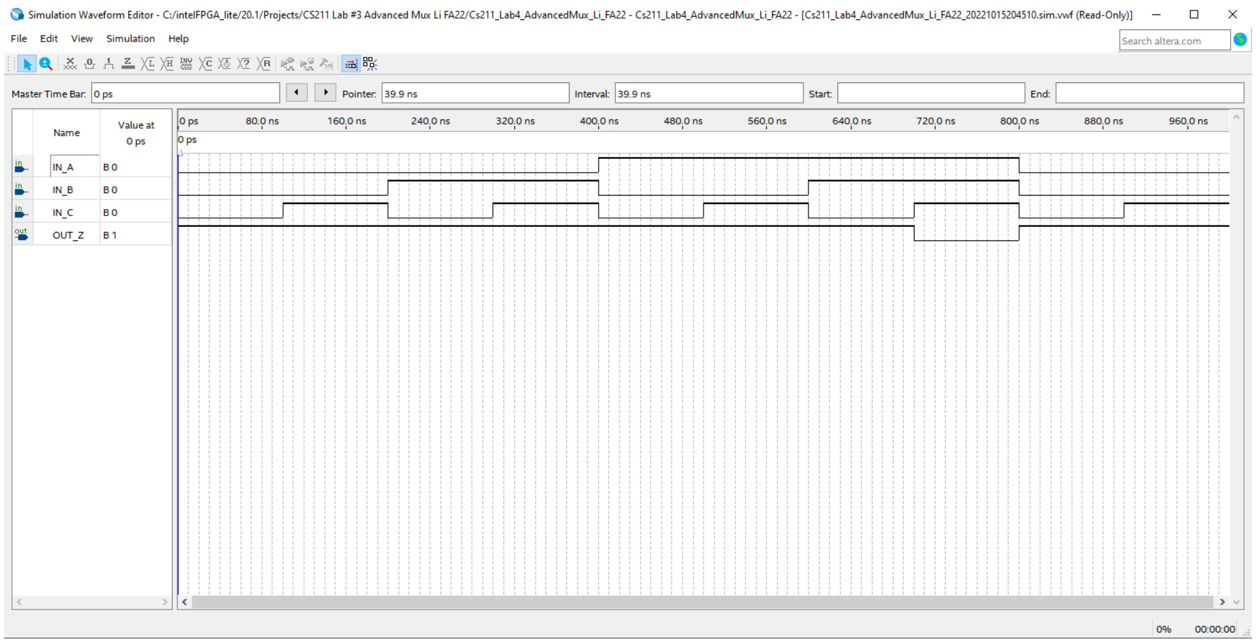
CS 211, Fall 2022

Laboratory Exercise 4: Advanced MUX

Instructor: Professor Izidor Gertner
Due October 16, 2022, by 11:59 PM



- Waveform Simulation



4-Input NAND gate:

- Truth Table

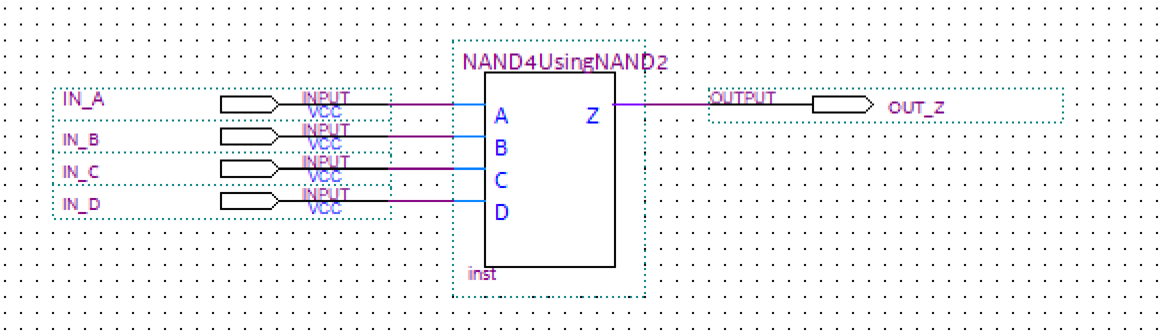
A	*B*	*C*	*D*	*Z*
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1

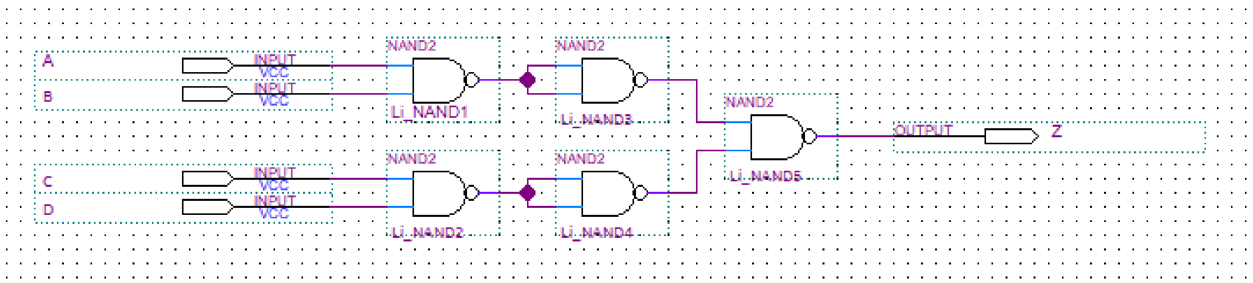
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

- Boolean Expression

$$Z = \overline{ABCD}$$

- BDF File





- Waveform Simulation

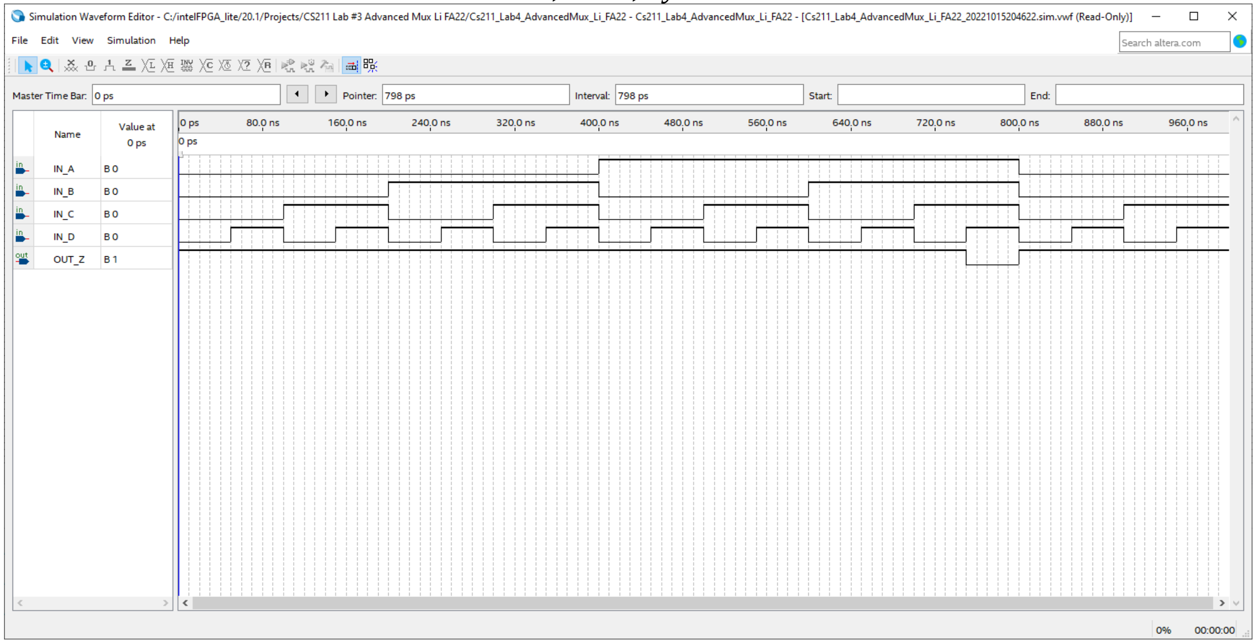
CS 211, Fall 2022

Laboratory Exercise 4:

Advanced MUX

Instructor: Professor Izidor Gertner

Due October 16, 2022, by 11:59 PM



Simulation Waveform Editor - C:/intelFPGA_lite/20.1/Projects/CS211 Lab #3 Advanced Mux Li FA22/Cs211_Lab4_AdvancedMux_Li_FA22 - Cs211_Lab4_AdvancedMux_Li_FA22 - [Cs211_Lab4_AdvancedMux_Li_FA22_20221015204622.sim.vwf (Read-Only)]

Master Time Bar: 0 ps Pointer: 798 ps Interval: 798 ps Start: End:

Name	Value at 0 ps
IN_A	B 0
IN_B	B 0
IN_C	B 0
IN_D	B 0
OUT_Z	B 1

4-to-1 Multiplexor:

- Truth Table

S_0	S_1	I_0	I_1	I_2	I_3	Z
0	0	0	X	X	X	0
0	0	1	X	X	X	1
0	1	X	0	X	X	0
0	1	X	1	X	X	1
1	0	X	X	0	X	0
1	0	X	X	1	X	1
1	1	X	X	X	0	0
1	1	X	X	X	1	1

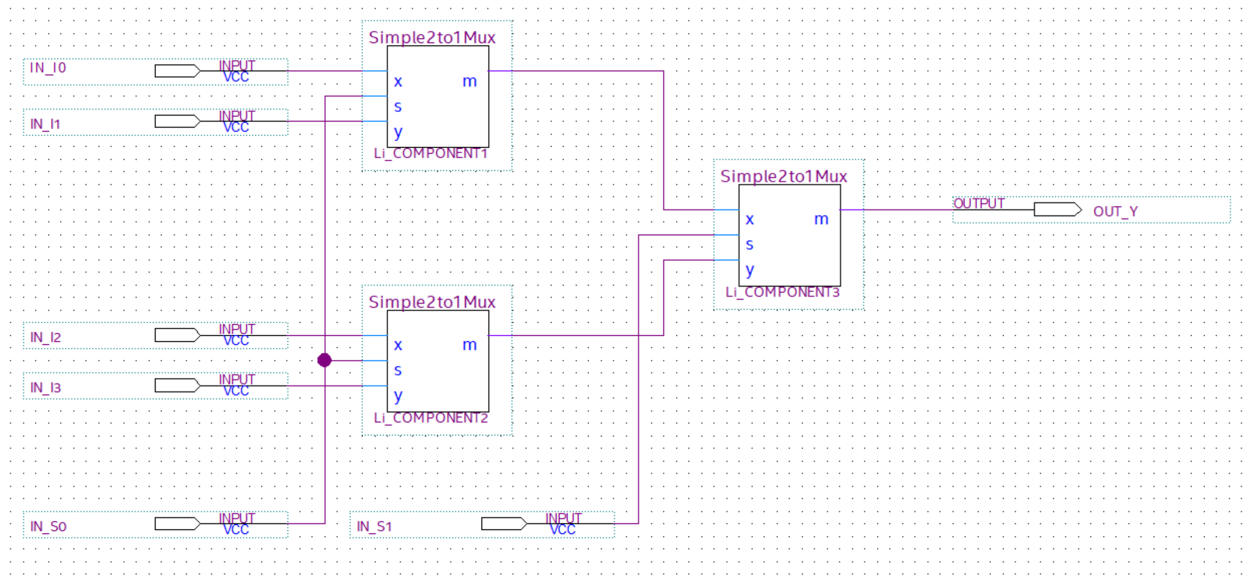
- Boolean Expression

$$\boldsymbol{Z = (I_0 * \overline{S_0} * \overline{S_1}) + (I_1 * S_0 * \overline{S_1}) + (I_2 * \overline{S_0} * S_1) + (I_3 * S_0 * S_1)}$$

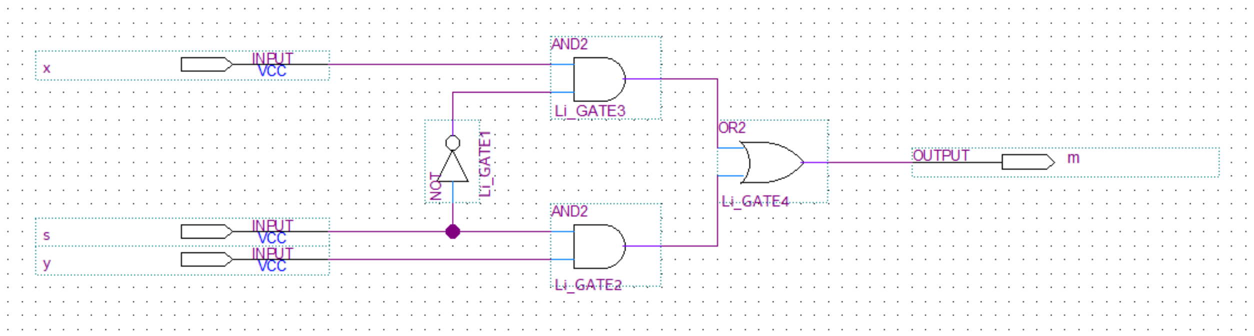
- BDF File

CS 211, Fall 2022
Laboratory Exercise 4:
Advanced MUX

Instructor: Professor Izidor Gertner
Due October 16, 2022, by 11:59 PM



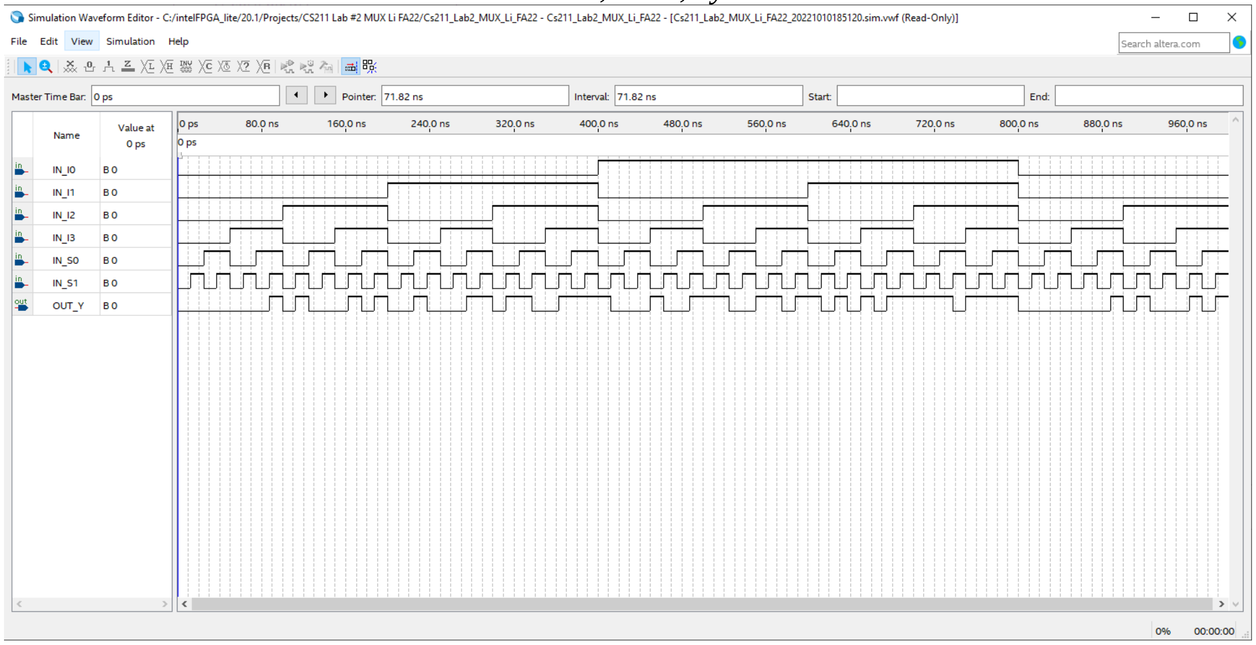
Below is the 2-to-1 Mux Symbol that is being used in the figure above



- Waveform Simulation

CS 211, Fall 2022
Laboratory Exercise 4: Advanced MUX

Instructor: Professor Izidor Gertner
Due October 16, 2022, by 11:59 PM



8-bit 2-to-1 Multiplexor:

- Truth Table

S	x_0	y_0	x_1	y_1	x_2	y_2	x_3	y_3	x_4	y_4	x_5	y_5	x_6	y_6	x_7	y_7	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8
0	0	x	0	x	0	x	0	x	0	x	0	x	0	x	0	x	0	0	0	0	0	0	0	0	0
0	1	x	1	x	1	x	1	x	1	x	1	x	1	x	1	x	1	1	1	1	1	1	1	1	1
1	x	0	x	0	x	0	x	0	x	0	x	0	x	0	x	0	0	0	0	0	0	0	0	0	0
1	x	1	x	1	x	1	x	1	x	1	x	1	x	1	x	1	1	1	1	1	1	1	1	1	1

- Boolean Expression

$$\begin{aligned} \boldsymbol{Z} = \boldsymbol{f}(x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, x_5, y_5, x_6, y_6, x_7, y_7, s) \\ = [(x_0 * \bar{s}) + (s * y_0)] + [(x_1 * \bar{s}) + (s * y_1)] + [(x_2 * \bar{s}) + (s * y_2)] \\ + [(x_3 * \bar{s}) + (s * y_3)] + [(x_4 * \bar{s}) + (s * y_4)] + [(x_5 * \bar{s}) + (s * y_5)] \\ + [(x_6 * \bar{s}) + (s * y_6)] + [(x_7 * \bar{s}) + (s * y_7)] \end{aligned}$$

- BDF File

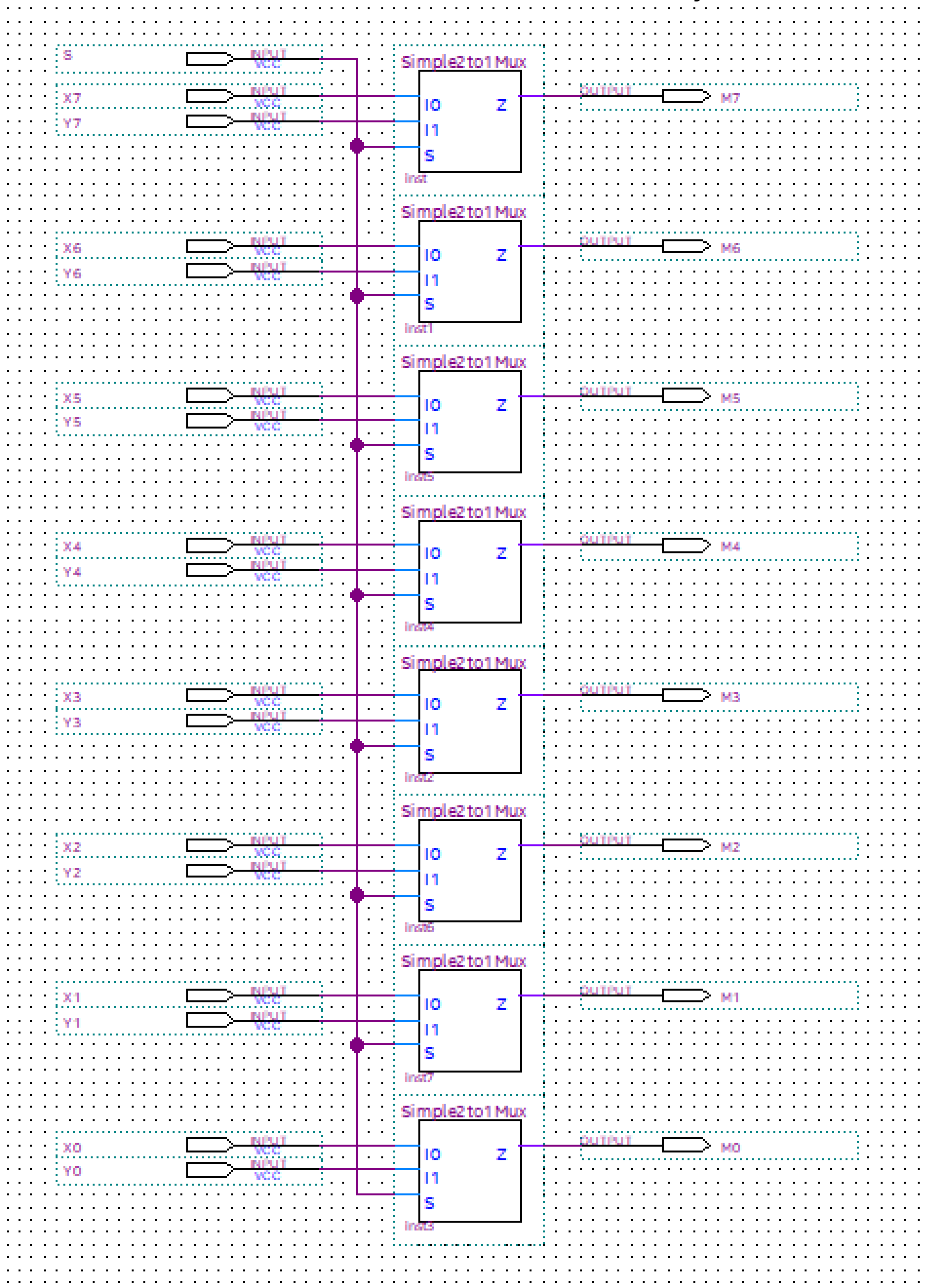
CS 211, Fall 2022

Laboratory Exercise 4:

Advanced MUX

Instructor: Professor Izidor Gertner

Due October 16, 2022, by 11:59 PM



- Waveform Simulation

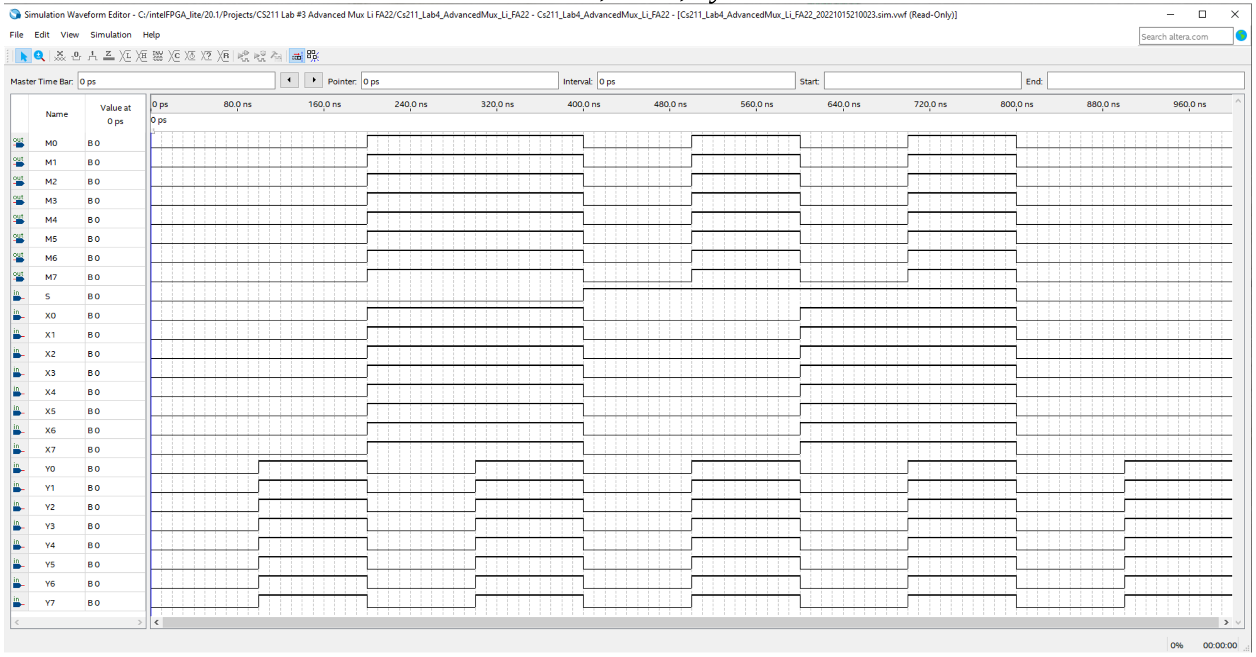
CS 211, Fall 2022

Laboratory Exercise 4:

Advanced MUX

Instructor: Professor Izidor Gertner

Due October 16, 2022, by 11:59 PM



Simulation Waveform Editor - C:/intelFPGA_lite/20.1/Projects/CS211 Lab #3 Advanced Mux Li FA22/Cs211_Lab4_AdvancedMux_Li_FA22 - Cs211_Lab4_AdvancedMux_Li_FA22 - [Cs211_Lab4_AdvancedMux_Li_FA22_20221015210023.sim.vwf (Read-Only)]

Master Time Bar: 0 ps   Pointer: 0 ps   Interval: 0 ps   Start: End:

Name	Value at 0 ps
M0	B 0
M1	B 0
M2	B 0
M3	B 0
M4	B 0
M5	B 0
M6	B 0
M7	B 0
S	B 0
X0	B 0
X1	B 0
X2	B 0
X3	B 0
X4	B 0
X5	B 0
X6	B 0
X7	B 0
Y0	B 0
Y1	B 0
Y2	B 0
Y3	B 0
Y4	B 0
Y5	B 0
Y6	B 0
Y7	B 0

<u>4. CONCLUSION</u>

Aside from learning how to create a 3 & 4 input NAND gate from 2 input NAND gates and how to create 8 bit 2-to-1 multiplexer, I honed my skills in using the Quartus application and refined my knowledge on solving for Boolean expressions, truth tables, and logic diagrams.