

## Programmable Logic Devices: Stage B

**Acknowledgements:** Developed by Bassam Matar, Faculty at Chandler-Gilbert Community College, Chandler, Arizona.

**Lab Summary:** This lab continues the lab completed in Lab A and presents design entry, simulation, and prototyping with tools that are provided by Altera for this purpose. We will show how a complete NOT/NEG, ADD/OR, and ALU can be directly entered into Quartus® II for synthesis, post synthesis simulation, timing analysis, and device programming. We will show the process of creating a device or symbol for each of the circuits. You will be using this lab as the basis for the two labs in the WRE Embedded Controller Module.

**Lab Goal:** The goal of this lab is to understand how to create an Arithmetic Logic Unit circuit and determine its truth table.

### **Learning Objectives**

1. Create, compile, and simulate the first component of the Arithmetic Logic Unit (ALU), NEG/NOT circuit and device in Quartus® II.
2. Create, compile, and simulate the second component of the ALU, ADD/OR circuit and device in Quartus® II.
3. Create, compile, and simulate the ALU circuit and device in Quartus® II.
4. Program the design of the ALU on a CPLD test board to determine its truth table.

**Grading Criteria:** Your grade will be determined by your instructor.

**Time Required:** 6 - 7 hours

### **Special Safety Requirements**

Static electricity can damage the CPLD device used in this lab. Use appropriate ESD methods to protect the devices. Be sure to wear a grounded wrist-strap at all times while handling the electronic components in this circuit. The wrist strap need not be worn after the circuit construction is complete.

No serious hazards are involved in this laboratory experiment, but be careful to connect the components with the proper polarity to avoid damage.

### **Lab Preparation**

- Read the WRE PLD Narrative Module.
- Read this document completely before you start on this experiment.

NOTE: This lab uses the information provided in the Introduction of Lab A. You will be referred to that section to review the needed information as required.



### **Equipment and Materials**

Each team of students will need the test equipment, tools, and parts specified below. Students should work in teams of two.

<b>Test Equipment and Power Supplies</b>	<b>Quantity</b>
The following items from the UP2 Educational Kit: <ul style="list-style-type: none"> <li>• Altera UP-2 circuit board with ByteBlaster Download Cable</li> <li>• Quartus® II Web Edition software</li> <li>• AC adapter, minimum output: 7 VDC, 250 mA DC</li> </ul>	1
ESD Anti-static Wrist Strap	1
#22 Solid-core wire	As needed
Wire Strippers	1

### **Additional References:**

1. *Digital Design and Implementation with Field Programmable Devices* textbook found with UP2 educational kit.
2. UP2 educational kit data sheet found on Altera web site: [www.altera.com](http://www.altera.com)

## **INTRODUCTION**

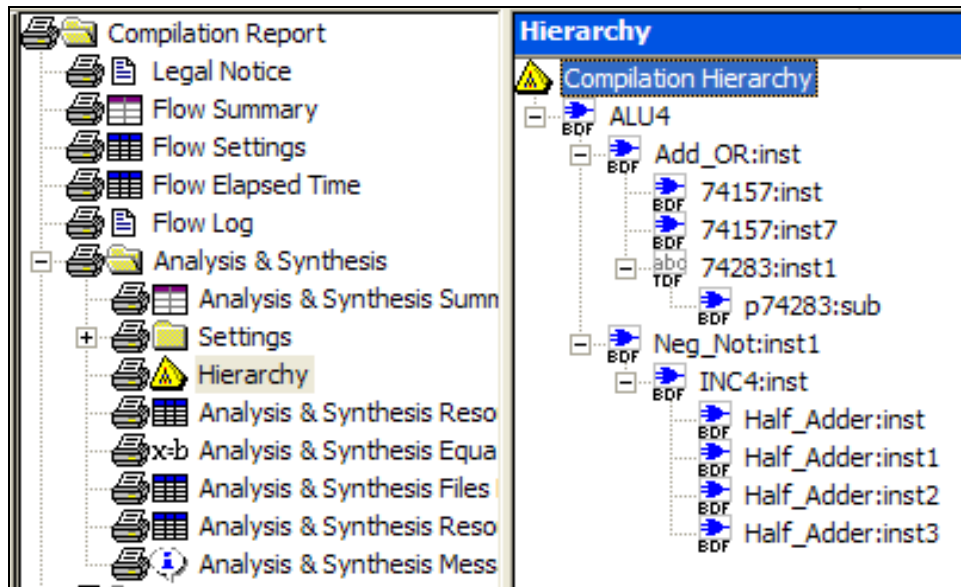
In stage A, we built combinational circuits of increasing complexity. In stage B, we will continue this to its conclusion - the Arithmetic Logical Unit (ALU). An ALU is a combinational logic circuit. It has two data inputs, each of which will be a 4-bit binary string. These are the operands. The ALU generates its output by operating on these operands with some arithmetic or logical operation. The actual operation is selected by a set of control signals which form the remaining inputs to the ALU. The central thing that a microprocessor does is to deliver the correct two binary operands to the results in the appropriate memory location.

In a microprocessor, we obviously must have circuits that somehow translate the commands provided to the microprocessor by the user into the control signals fed into the ALU. These are the instruction decode circuits. This often is done by storing the ALU control signals in a memory. The user's command then contains the address of the memory location where the appropriate controls are found. Additionally, in a real system, we often need to store the result generated by the ALU into some specified memory location. Therefore, we need circuits that take a binary bit string and use it to activate only one output line; the line addressed is the binary input. Such circuits are one type of decoder.

The ALU is used to process data by performing binary addition, subtraction, AND's, OR's, one's and two's complements, etc. A typical ALU has two data inputs (in our case, each 4-bit wide), some assorted control inputs that tell the ALU what type of operation to perform, and an output (again 4-bit wide). There is also often a carry input, a carry output, and sometimes outputs that tell if the 4-bit output is zero, etc. The inputs and outputs of an ALU are only limited by design considerations (such as size, speed, and expense) and by what the designer thinks might be useful for instructions in the microprocessor. In our case, we will try to keep the design relatively simple. We will use the ALU we build here in the microprocessor we build in the embedded controller module.



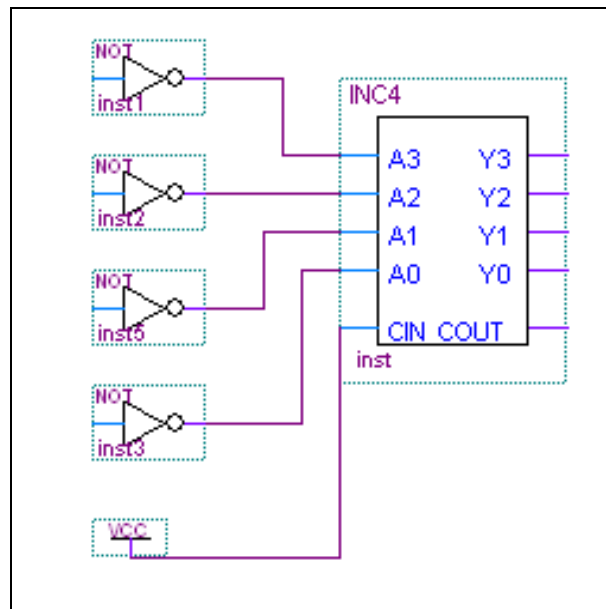
Here is a breakdown of the ALU4 Hierarchy.



In the lab, we will first build, compile, and test the NEG/NOT circuit in Altera. The next step will be to build, compile, and test the ADD/OR circuit. When we have both of these complete, we will combine them to make the ALU. Finally, we will program the target PLD and determine the truth table. In order to build these circuits' we must first understand how they work together.

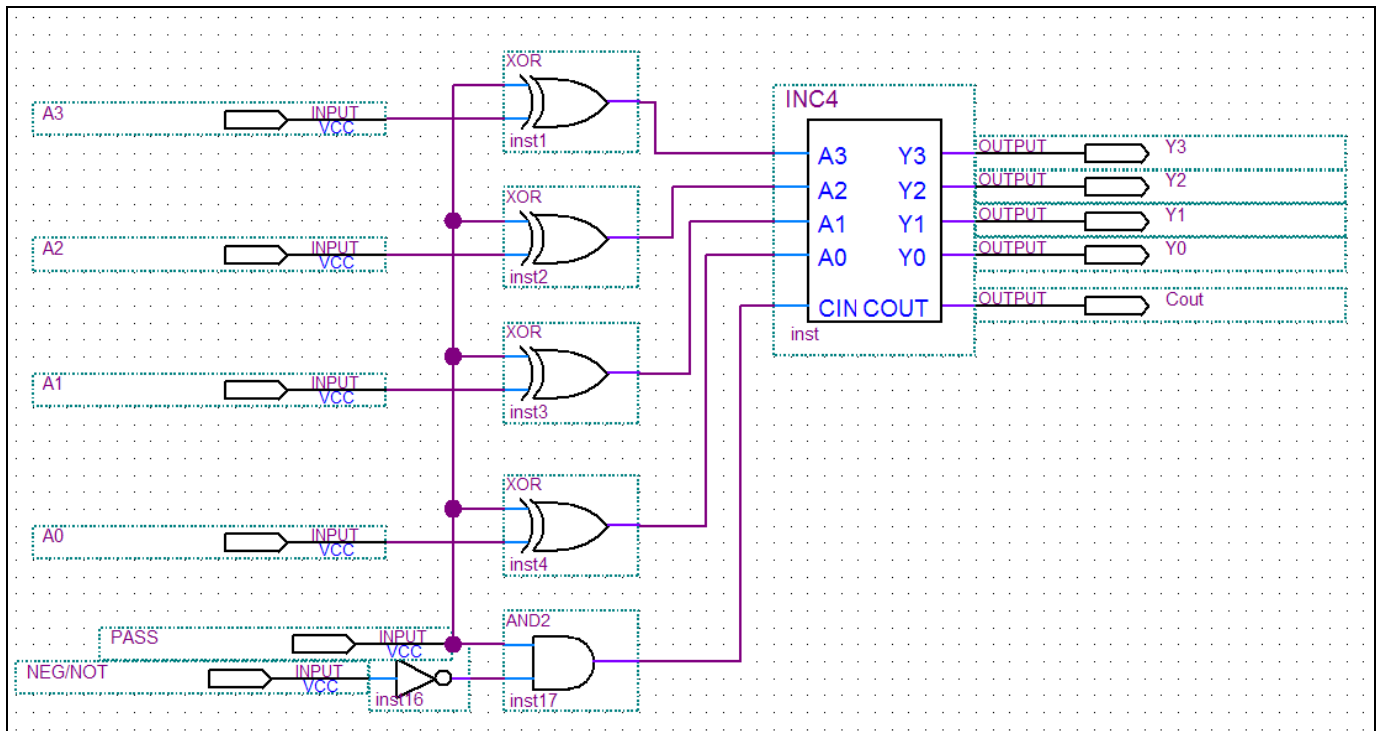
### Building the NEG/NOT Circuit

We will start by making the two's complement circuit (shown here) a little more useful.





Making the following changes can make this circuit better for our purposes:



Now there is a single 4-bit wide data input and there are two control inputs that tell us what we can do with the inputs. If the PASS control input is high, each exclusive-or (XOR) gate acts like an inverter on A0, A1, A2, or A3. If NEG/NOT is high, the result is a one's complement operation. If NEG/NOT is low, the result is a two's complement operation. The pin is called NEG/NOT because if the pin is 0, a negate operation is performed (negate is another name for two's complement), but if it is a 1, a NOT is performed (one's complement). There are thus two ways to look at the pin, as NEG or as NOT. Both are indicated in NEG/NOT. If PASS is low, the exclusive OR gates have their output equal to their other input, and no incrementing is done. We will verify this by checking the truth table of the exclusive-OR gate. It is one of the exclusive-OR's most useful properties. This is summarized in the following function table:

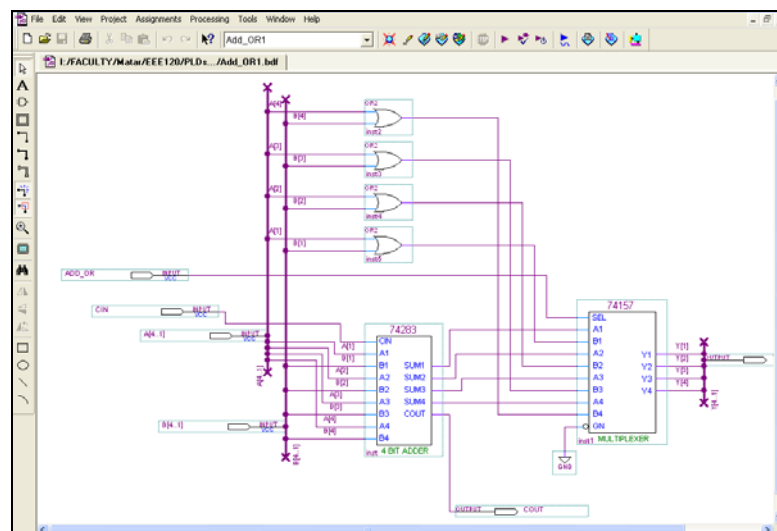
NEG/NOT	PASS	FUNCTION
0	0	PASS-THROUGH
0	1	TWO'S COMPLEMENT
1	0	PASS-THROUGH
1	1	ONE'S COMPLEMENT



## Build the ADD/OR Circuit

The NEG/NOT circuit is an extremely simple version of an ALU, so simple that no one would actually call it an ALU. Yet, it will do either an arithmetic operation, the negate, or a logical operation, the NOTing of 4 bits. Which operation is to be performed is selected by a user provided control signal. An ALU is a circuit that is supposed to perform many different functions as inputs.

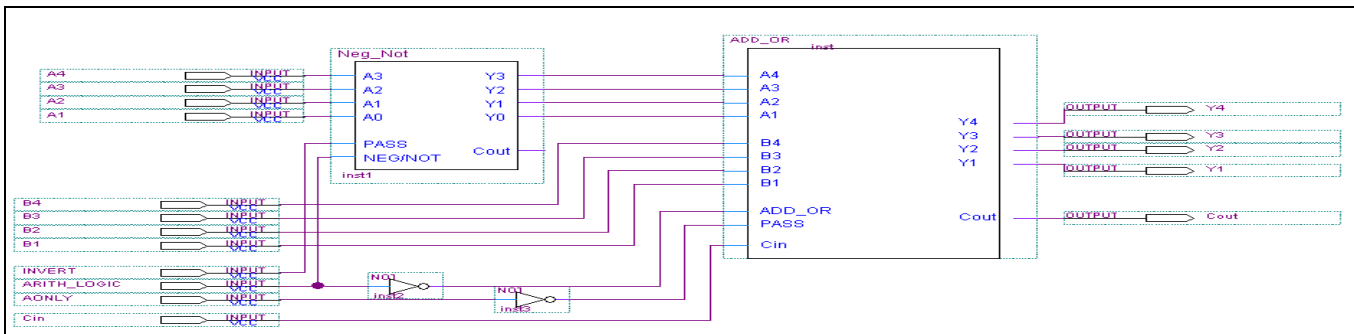
However, for a true ALU we would like to perform more functions than just the three given above. In particular, we would like to be able to do an addition and an OR of two four-bit numbers. We already have everything we need to do this. We now take our input data and feed it to two different circuits. One is our **FULLADDER-4** while the other is a set of OR gates. The OR gates have two input data streams. We then use a MUX to choose which of the two resulting outputs: the sum or the OR, is to be connected to the output of our overall circuit. This is an excellent example of how multiplexers are used in digital streams. Our circuit is:



The circuit above does an addition and it does an OR logic. A multiplexer is used to select which we want. There is only one thing that it does not do, and that is passing just one of the values. Pass-through is useful, so we can add this in by adding another control input and a final MUX.

Instead of creating the 4-bit adder (74283) and the multiplexer (74157), we will use the software library to create the above circuit.

Finally, using Altera Software, we can combine the NEG/NOT and the ADD/OR as follows, so that we have a circuit that can perform enough elementary functions that we could call it an ALU. This will be the ALU that we use in our microprocessor.



## Discussion of the ALU Design Methodology

It is extremely important to remember how to produce this ALU. The design approach we will use is to build up simple circuit blocks, and then build up bigger blocks from the simple ones, and so on to greater levels of complexity. This has made design of the ALU easy. Think for a minute how it would be to design an ALU out of AND's and OR's. Doing it by blocks in stages is much easier. We do not need to worry about too many changes to a circuit when we build it out of blocks. It is still likely that we could make the design smaller and faster if we build it right out of the AND's and OR's using an optimal technique.

This reliance on a hierarchy of well-defined modules is commonly used in computer programming as well as in digital circuit design. Structured programming and structured design techniques are like the block-design approach that we have used in these labs. It is easier to use these structured techniques, and it is much easier to make modifications to what we have done and for other people to understand what we have done. However, unstructured techniques are usually better in terms of performance.

The current trend is for more and more work to be done in a structured fashion. However, for combinational digital logic there is another option. POS and SOP designs can be used to generate **any** logic function. So we could go with one of these designs, instead of the block design we have here. There are also Karnaugh map methods (for small circuits) and computer optimization routines (for big circuits) to minimize the circuitry needed for a function. These could be applied to the ALU above to give us an optimum circuit. In a real design, this would be done because it is really no more difficult to understand or maintain the truth table of the combinational circuit than it is to understand the blocks we have developed. Computer programs exist which will automatically generate an optimal combinational circuit from its truth table.

For our ALU, we will use the MAX7000s device family. From this family of devices, we will use the EPM7128SLC84-7 CPLD. We have selected this device because it is one of the two programmable devices on Altera's UP2 development board.

## Programming the Target PLD

The last step in developing the ALU design for a PLD is programming the target PLD. The necessary files for this purpose are generated in Quartus® II after the design has been successfully compiled.

MAX 7000S devices use the *pof* (Programmer Object File) programming file format and the FLEX 10K device use the *sof* (SRAM Object File). These files are generated by the compiler and they include all necessary configuration data for the appropriate PLDs.



## Configure Devices

The next step in our lab is to configure the devices. This was discussed in the Introduction to Lab A. Review that section as needed before you work on this section of the lab.

## Testing the ALU Circuit in the Altera UP2 Development Board

The final step in our lab is to test the ALU circuit in the development board. Since the switch and LED connections to the MAX device are not pre-assigned in UP2, we will need to make the hard-wire connections. For a complete description of the pin numbers and LEDs, use the tables contained in the Introduction of Lab A.

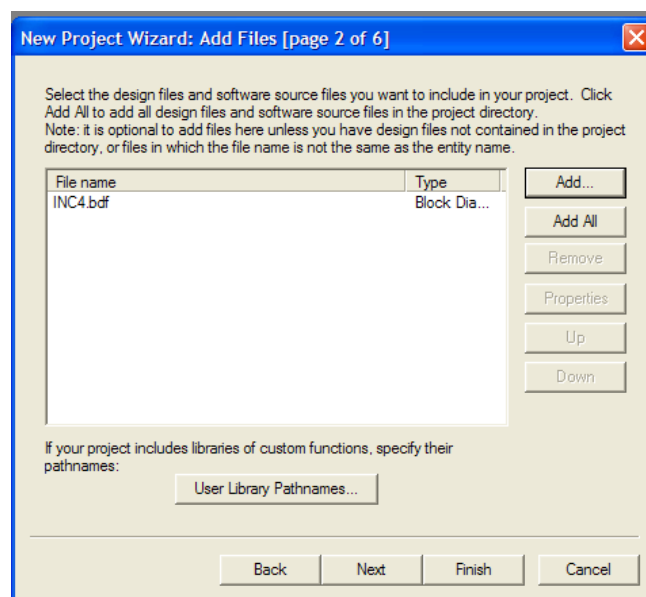
## Device Programming

In this section, we will set up the UP2 development board for its EPM7128S device to be programmed with the ALU circuit of this lab.

After performing the required steps, our ALU project implemented on the MAX 7000S device of the UP2 board will be ready to be tested.

## Lab Procedure: Build the NEG/NOT Circuit

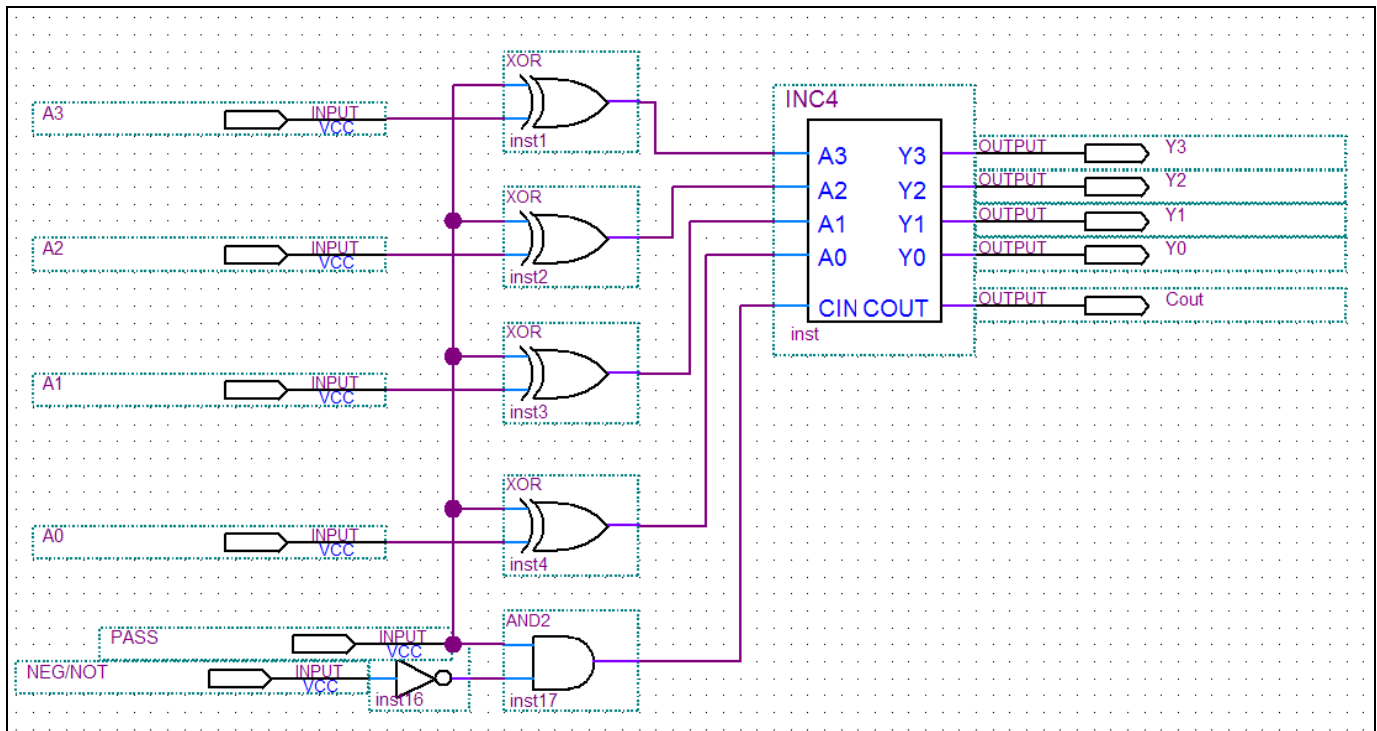
1. Build the circuit:
  - a. Select File→New Project Wizard and enter the name for the project (NEG\_NOT)
  - b. Click Next and **Add** INC-4 as part of your project as shown below.
  - c. Click Finish.



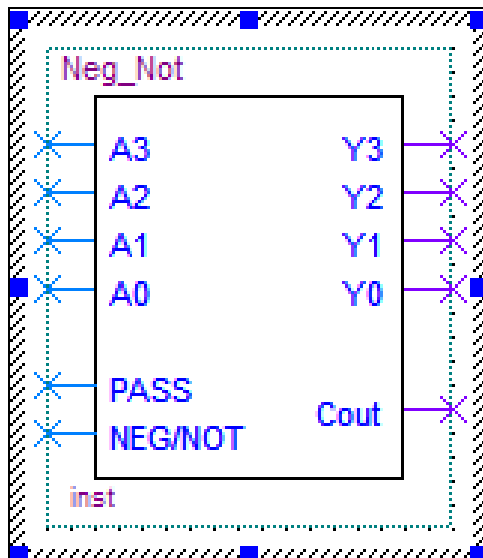
2. Now, click **File New**.
3. Select **Block Diagram/Schematic File**.



4. Create the NEG/NOT circuit shown here.



5. Call this circuit the NEG\_NOT circuit and give it the following symbol:

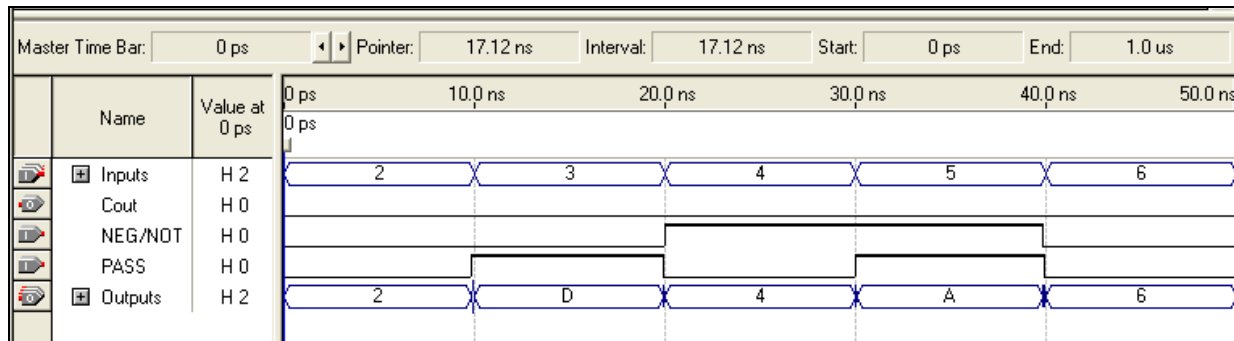


6. Build this circuit in Altera, compile it, test it, box it, and add it to your library. If you have forgotten how, refer the steps you used in Lab A.
7. After it is in your library, test it again. This is a circuit you will be using again, so ensure it is correct now.





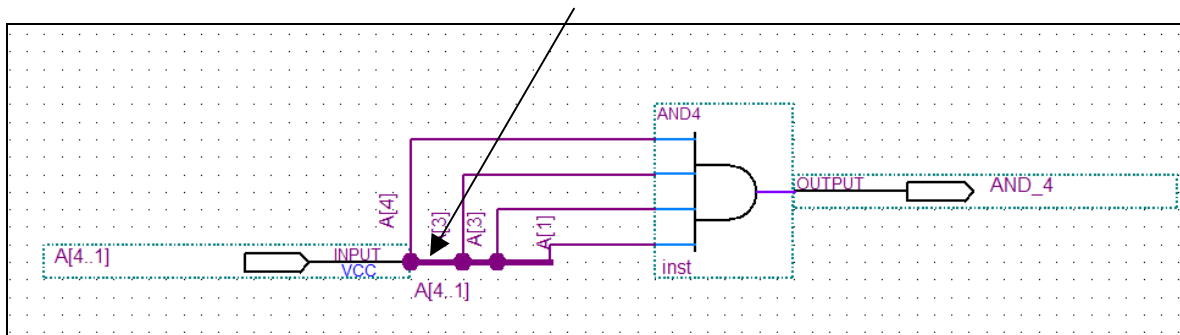
8. You should get the following results for the Neg\_Not Symbol Waveform Diagram test:



NEG/NOT	PASS	FUNCTION of the Above Waveform Diagram
0	0	PASS-THROUGH <i>the value of 2</i>
0	1	TWO'S COMPLEMENT <i>of 3 = D</i>
1	0	PASS-THROUGH <i>the value of 4</i>
1	1	ONE'S COMPLEMENT <i>of 5 = A</i>

9. Create a bus.

a. A “bus” is a single line on the schematic that represents a group of related signals as shown here.



b. To draw a bus, click the **Orthogonal Bus Tool** button from the tool bar.

c. Name the bus. The following are examples of legal bus names: A[4..1], the name defines a single-range bus whose identifier is **A** and contains 4 bits. The individual bits of this bus are **A4**, **A3**, **A2**, and **A1** (or A[4], A[3], A[2] and A[1]). The MSB is A4 (or A[4]); the LSB is A1 (or A[1]).

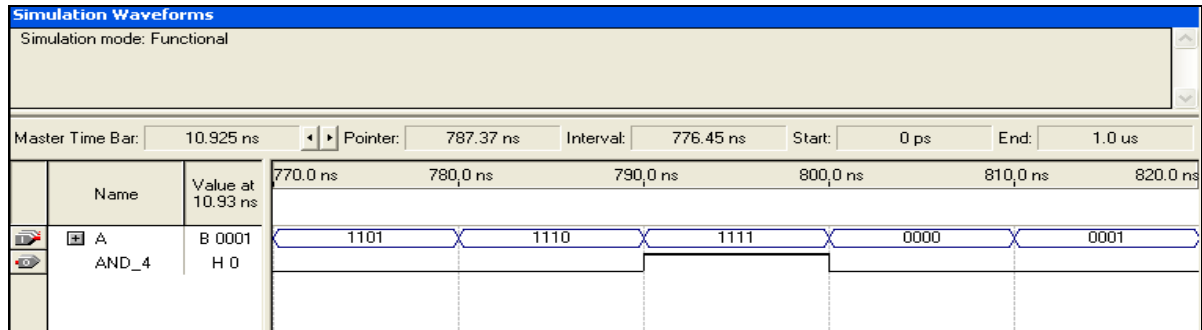
NOTE: Make sure you name the actual bus line and not just changing the name to A[4..1].

d. Specifying the line of the bus a wire is connected to by selecting the wire.

e. In its node properties window, set its name to the name of the bus indexed by the bus-line it is connected to. For example, naming a wire as A[1] connects the line to bit 1 of bus A[1..4].

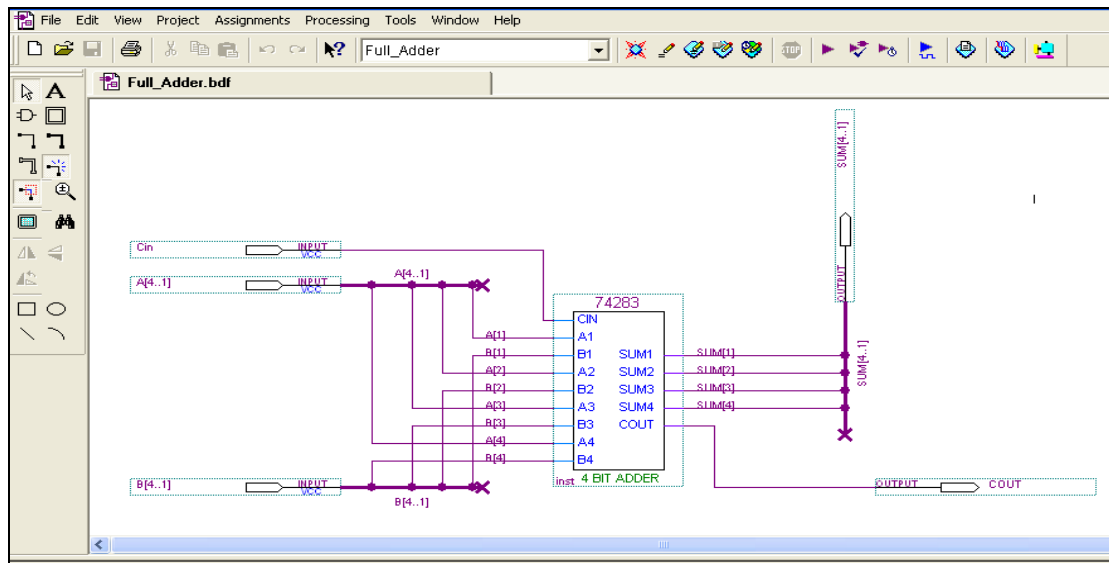


10. Run the AND-4 gate timing circuit. Your results should look like this.

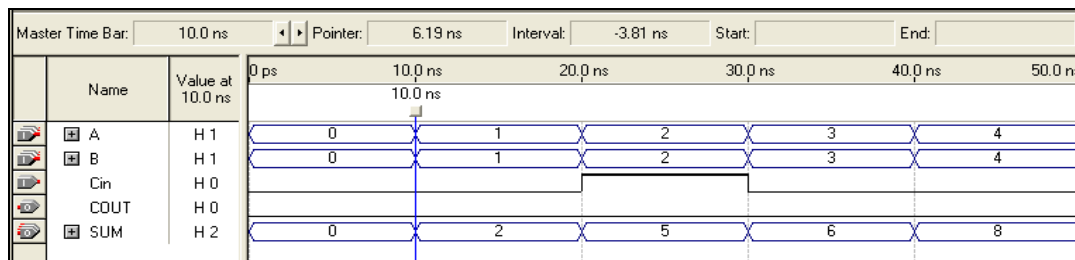


11. Apply the same method to make the rest of the connection of the **Full\_Adder** circuit.

- Select 74283 device (4 bit adder) from *others directory*.
- Select **MaxPlus2**.

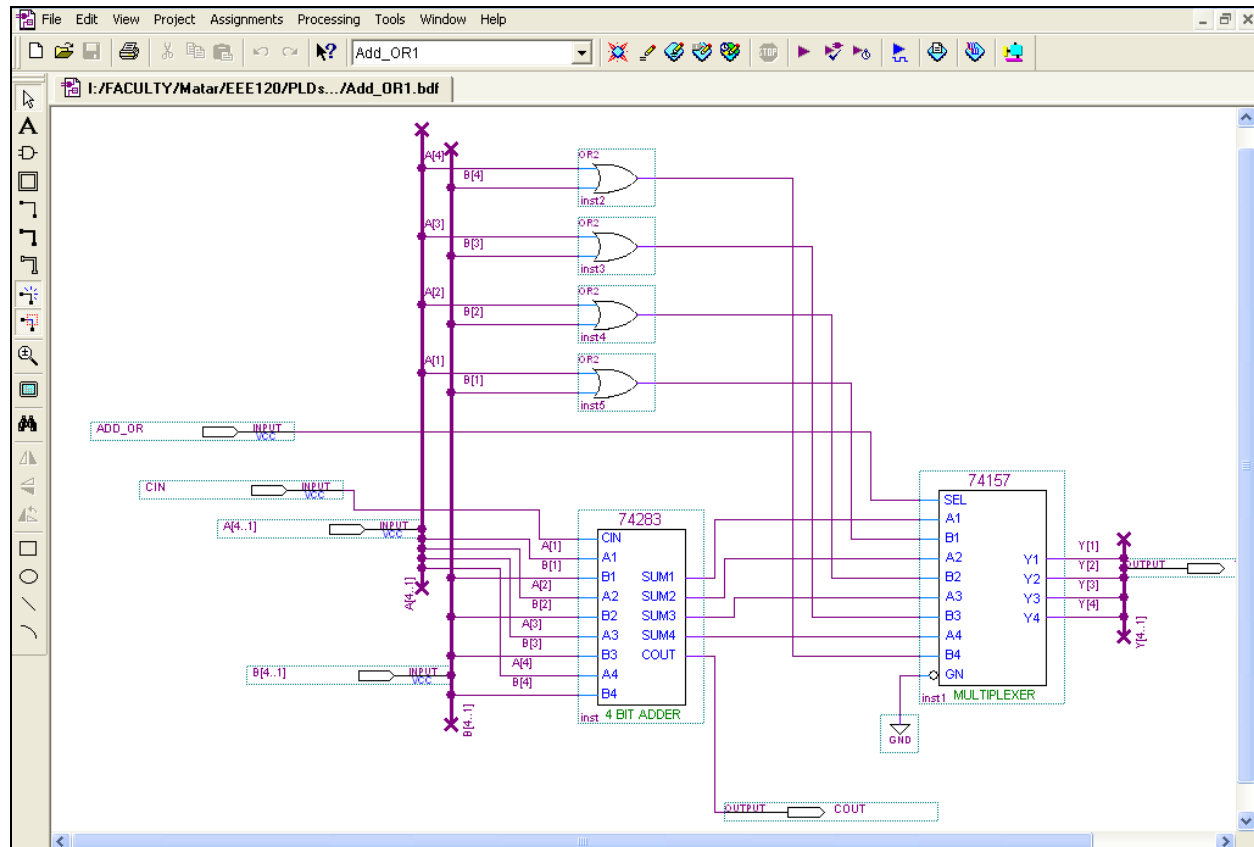


12. Test the circuit. Your results should look like this:

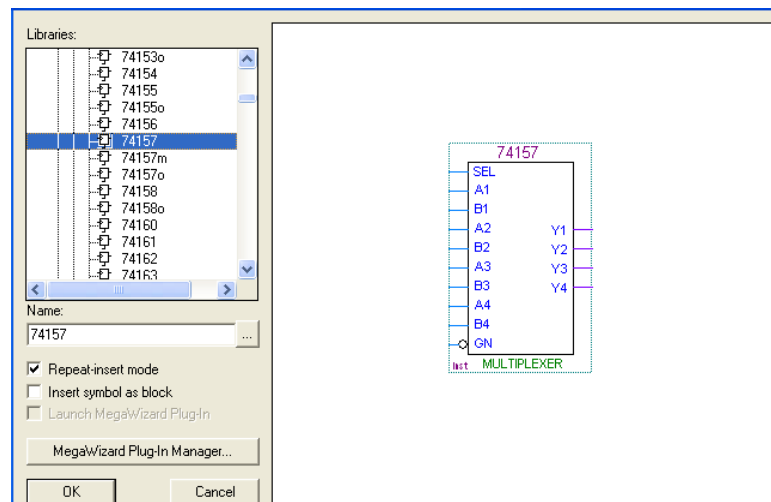




## Lab Procedure: Build the ADD/OR Circuit

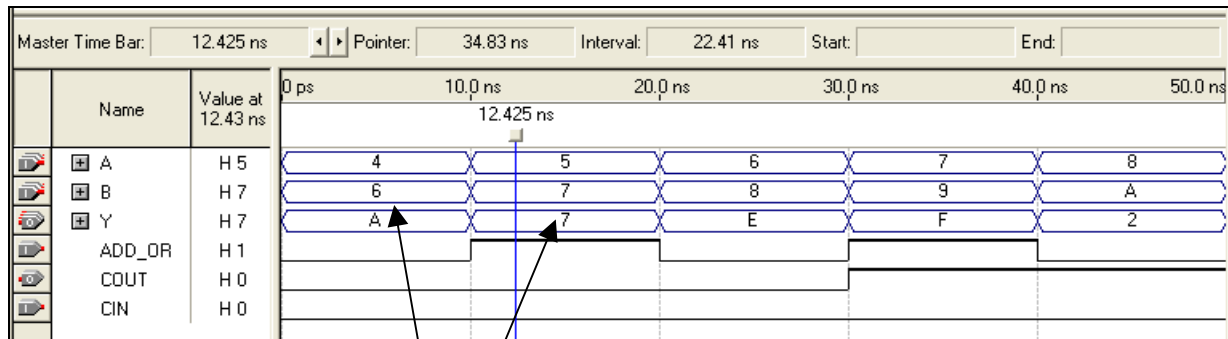


13. Create the circuit above from the **Others** directory and then MaxPlus2.



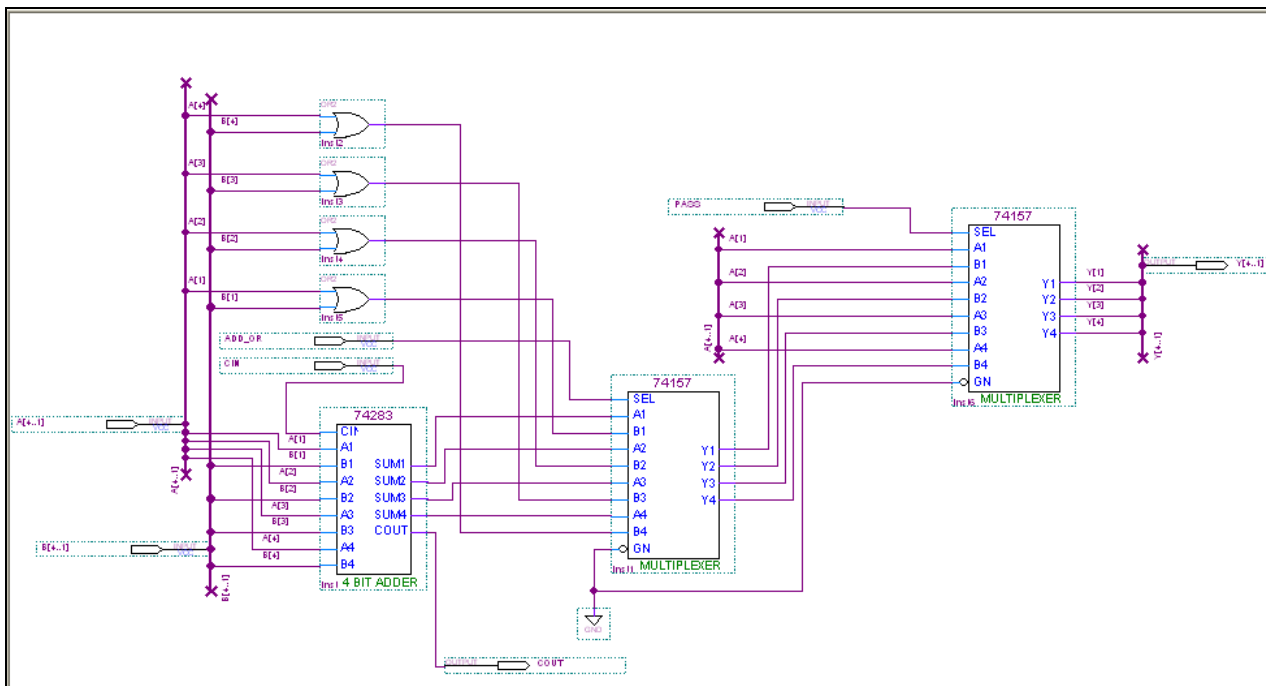


14. Select the MUX 74157. In the 74157, when the data select is 1, the Bs inputs are selected. In order to pass the A's values, we need to connect a NOT gate to the selector which will allow us to route the As inputs when the data selector is 1.



ADD/OR	FUNCTION
0	ADD (4+6=A)
1	OR (5 ORed with 7 = 7)

- a. This expanded circuit is shown here:

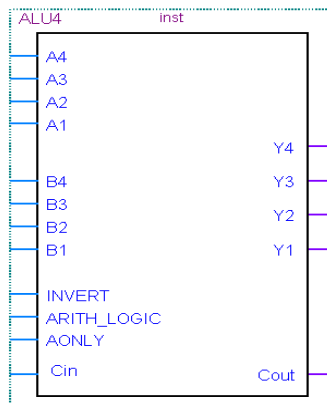




b. The following function table describes this circuit:

ADD/OR	PASS	FUNCTION
0	0	PASS-THROUGH (the values of the As)
0	1	ADD
1	0	PASS-THROUGH (the values of the As)
1	1	OR

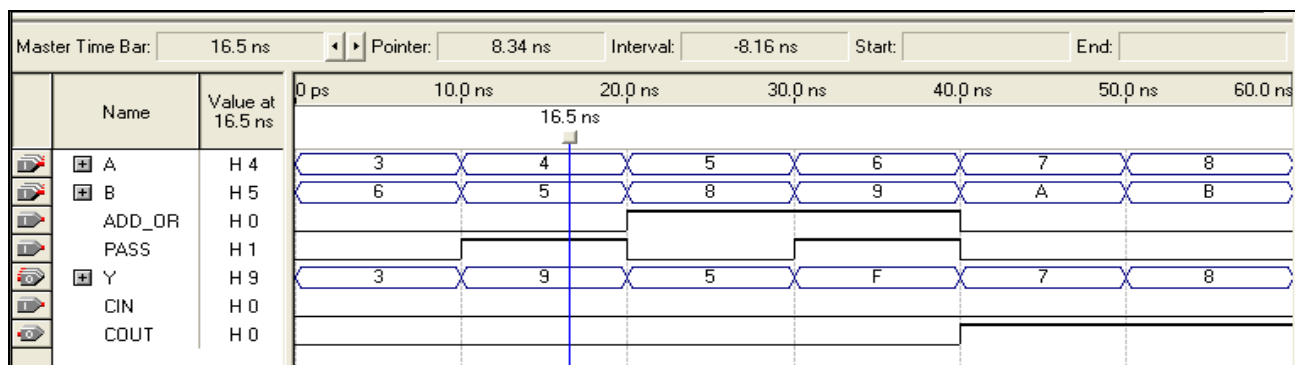
c. Give the ADD/Or circuit the following picture:



15. Build this circuit in Altera software.

16. Build the device for it and add it to your Library.

17. Test the circuit. Your test results should look like this:

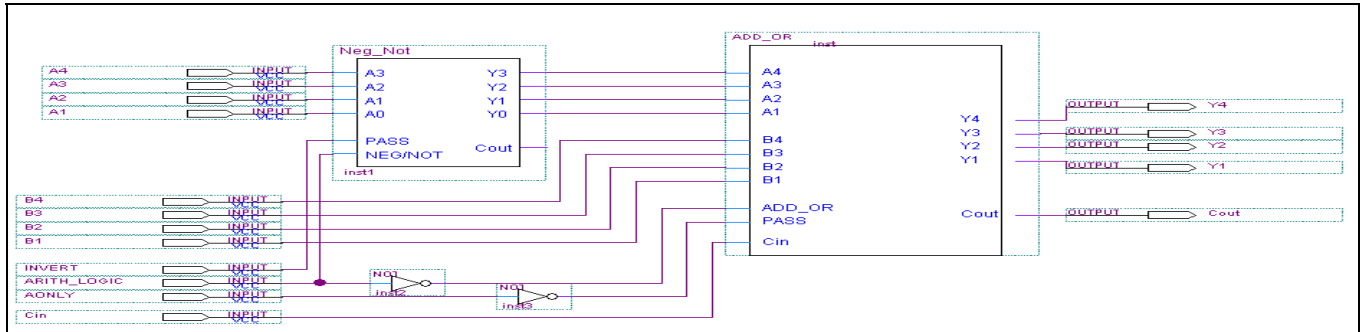


18. Be prepared to describe the procedure you have used to test it and how the test procedure fully tests the circuit.

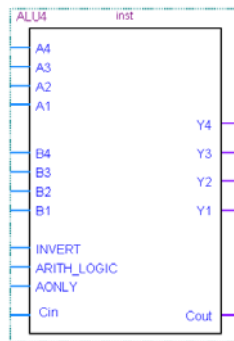


## Lab Procedure: Building the ALU Circuit

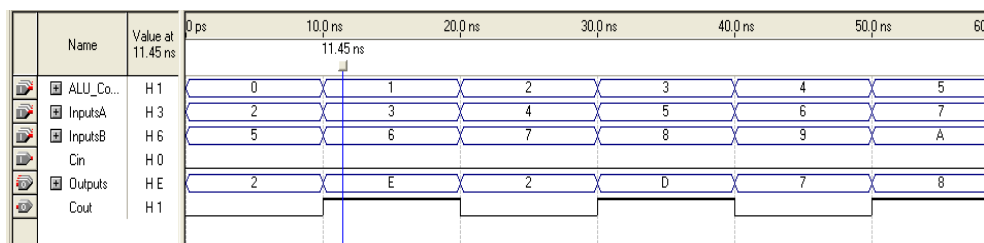
NOTE: A larger drawing of this circuit is shown at the end of this lab procedure.



19. Build this circuit in Altera software.
20. Build the device for it and add it to your Library.
21. Give the ALU the following picture:



22. Test the circuit. Your test results should look like this:



23. According to our Altera software compilation, the final Time Propagation Delay (tpd is 19.9 ns.)

NOTE: As you can see in the final timing simulation result above, there are a couple of glitches in the output. Remember, the target device is not specified yet and therefore, Altera software uses a generic model that does expose the potential for a glitch in the design.

24. Write a short description of how you have tested it. Include the following information:
  - a. What is the function table for this circuit?
  - b. What are the control inputs combinations?



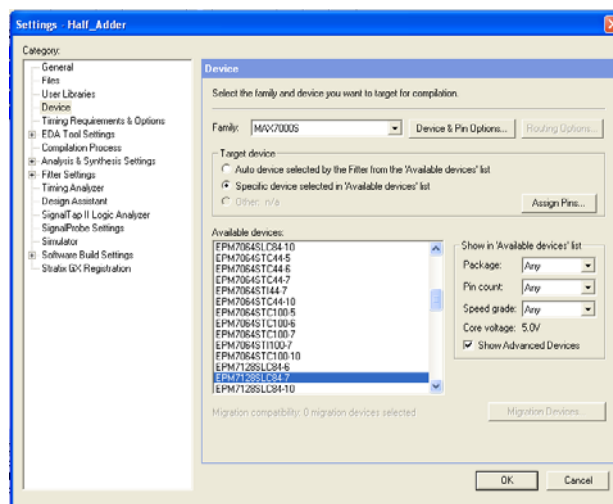
- c. Create the following table in your report and find the function of the ALU for each of the control combinations (*Aonly, Arith/Logic and Invert*)

Function Table of the ALU

A ONLY'	ARITH'/LOGIC	INVERT	FUNCTION	HEX INPUT A	HEX INPUT B	HEX OUTPUT
0	0	0	PASS-THROUGH	2	5	2
0	0	1	2's COMPLIMENT of A	2	5	E
0	1	0	PASS-THROUGH	2	5	2
0	1	1	1's COMP of A	2	5	D
1	0	0	A plus B	2	5	7
1	0	1	(2's COMP A) plus B	2	5	3
1	1	0	A OR B	2	5	7
1	1	1	(1's COMP of A) OR B	2	5	D
0	0	0	PASS-THROUGH	A	3	A
0	0	1	2's COMPLIMENT of A	A	3	6
0	1	0	PASS-THROUGH	A	3	A
0	1	1	1's COMP of A	A	3	5
1	0	0	A plus B	A	3	D
1	0	1	(2's COMP A) plus B	A	3	9
1	1	0	A OR B	A	3	B
1	1	1	(1's COMP of A) OR B	A	3	7

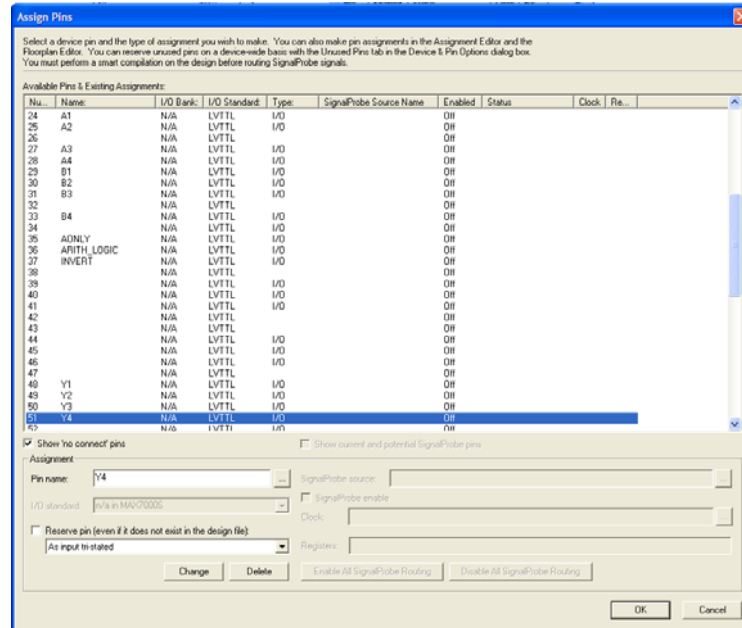
### Lab Procedure: Device Programming

25. For our ALU, we will use the MAX7000s device family. From this family of devices, we will use the EPM7128SLC84-7 CPLD. We have selected this device because it is one of the two programmable devices on Altera's UP2 development board. The figure below shows the project definition page in which the project is defined.



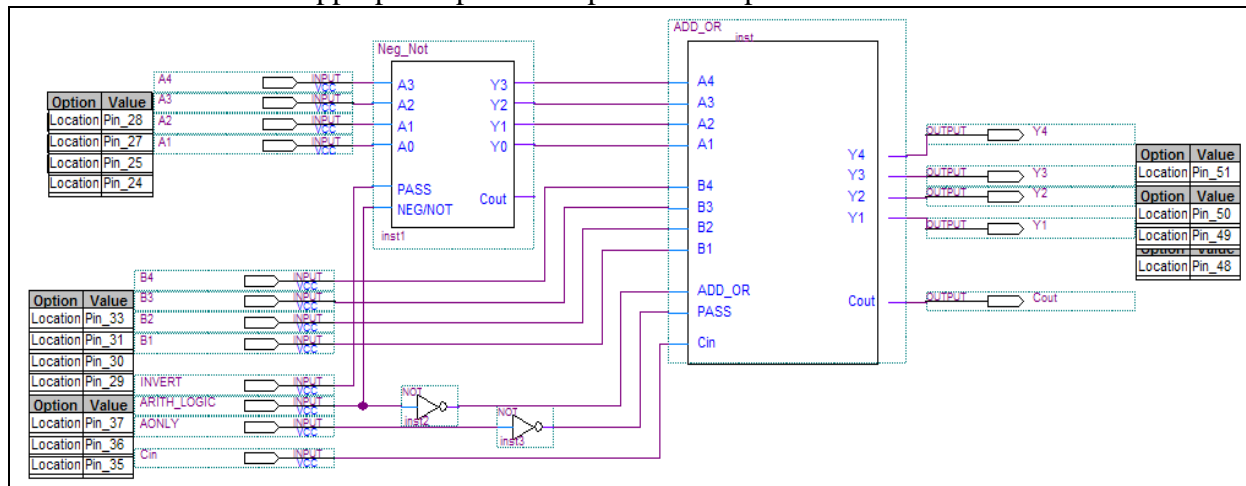


26. Select **Assign Pins** from **Assignments** menu to assign Pins to our inputs and outputs ports of our ALU circuit to the pins of the CPLD we are using. The corresponding window is shown below:



- Scroll down the device's pin list.
- Click on the pin that is being assigned to a circuit node.
- Type the name of the node of your circuit in the assignment area, next to the Pin. In our ALU circuit, we type the name of the node in association of the highlighted pin number. We assign A1, A2, A1, A2, B1, B2, B3, B4 AONLY, ARITH\_LOGIC, and INVERT inputs to pins 24, 25, 27, 28, 29, 30, 31, 33, 35, 36, and 37 as respectively as shown above. Also, the outputs Y1, Y2, Y3, and Y4 to pins 48, 49, 50 and 51. The reason for this selection becomes clear after we discuss the UP2 board in the next section. In assigning pins, make sure reserved pins, such as those of the JTAG, are not used. For example, pin 26 is reserved and not used as I/O.

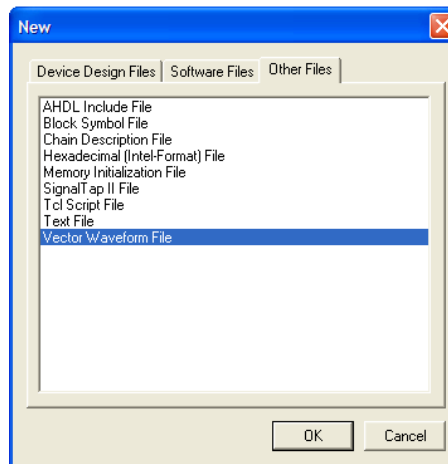
The circuit now shows the appropriate pins for inputs and outputs as shown below:



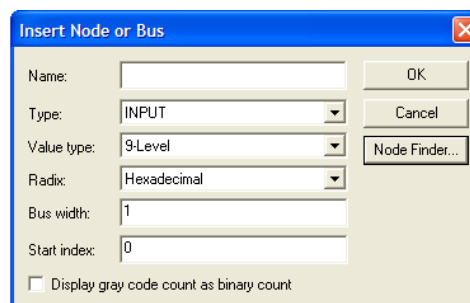




27. Compile your circuit one more time to make sure there are no errors.
28. Complete the following steps before running a simulation. This simulation is often referred to as post-synthesis, because it simulates the actual gates and cells of the target device (the device that will be programmed).
- Before we start our simulation, we have to specify input values for the Half Adder (A and B). To do this, from the file menu, select **New** and then **Vector Waveform File** in the **Other Files** folder to open the waveform editor window as shown below and click OK.



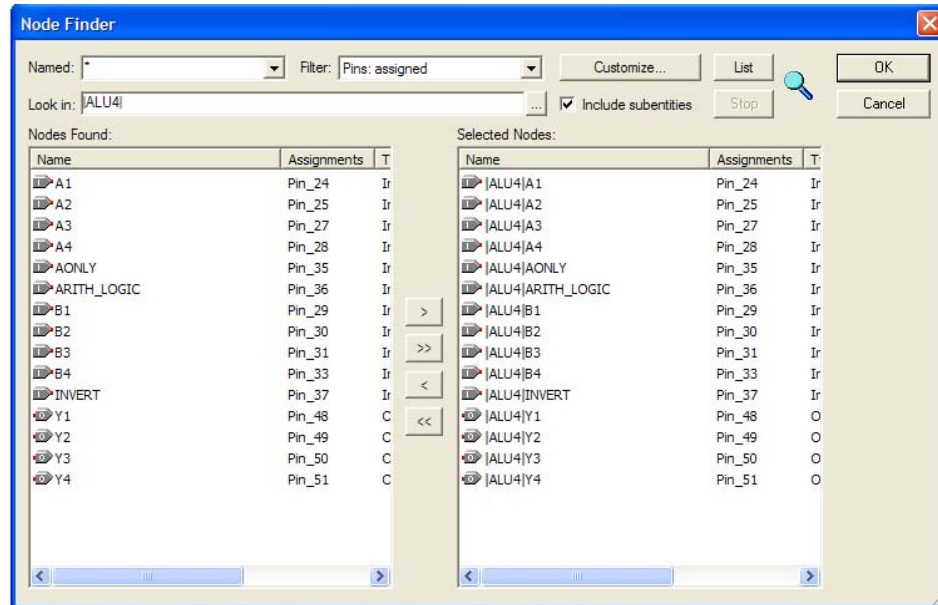
- The waveform file is initially blank and we need to enter our node names and their associated waveforms. Save the waveform as ALU.vwf.
- Use the right mouse button and select the **Insert Node or Bus**. This brings up the **Insert Node or Bus window** as shown below:



- Click on Node Finder and select **Pins: Assigned** and click List to view all the nodes that apply to our circuit.



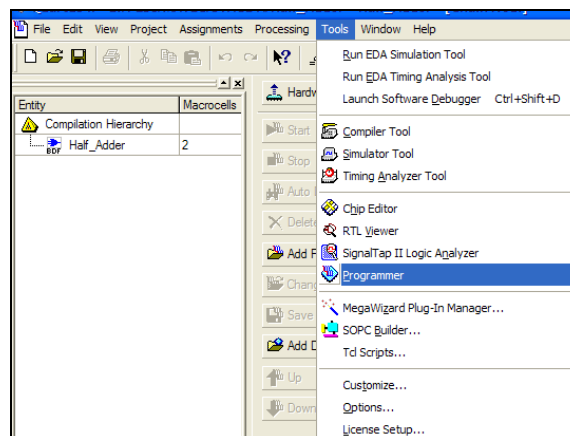
- e. Next highlight all the nodes and click the >> symbol button to copy all nodes to the Selected Nodes list as shown below. Click OK to accept all the nodes inputs and outputs.



29. Run a functional simulation for sub circuits. Run a timing simulation.

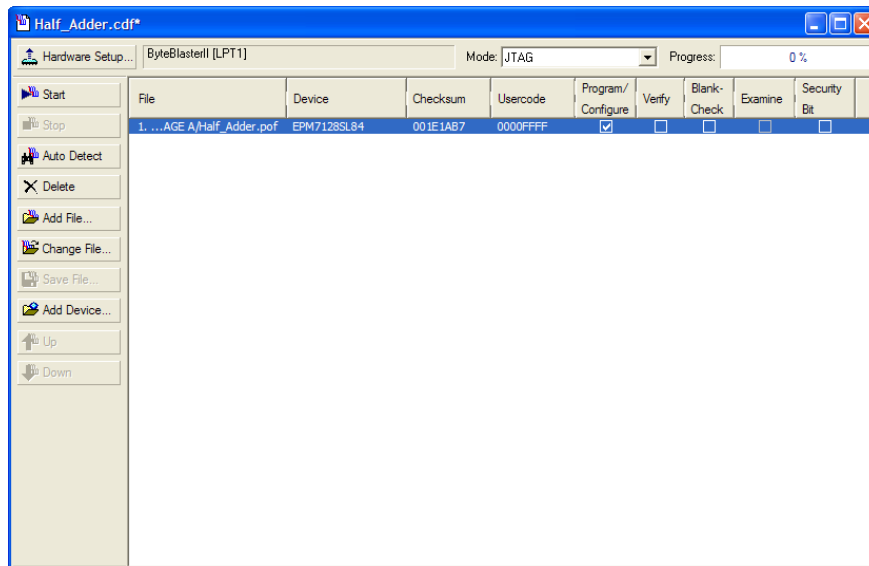
### Lab Procedure: Programming the Target PLD

30. To start the device programming process, select the Programmer from the **tool** menu as shown below.





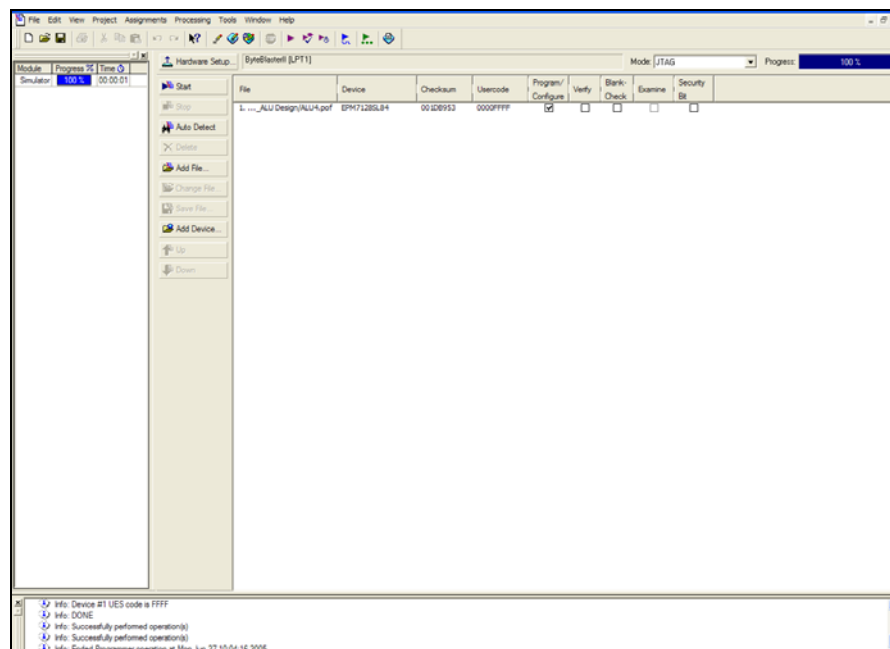
31. This brings up the device configuration window as shown below.



32. Before you click start, make sure that the development board containing the target device has been properly configured for programming. For our ALU circuit, the ByteBlasterII has been selected and assigned to port LPT1 using JTAG (Joint Test Action Group) mode.

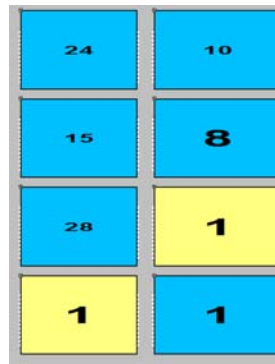
NOTE: If the Quartus® II environment is being used for the first time, you should setup its programmer hardware. Refer to Lab A for the proper process to set up the hardware.

33. After you have completed your setup in the configuration window, select Start to download. *Program and configure* are terms synonymous with *download*.

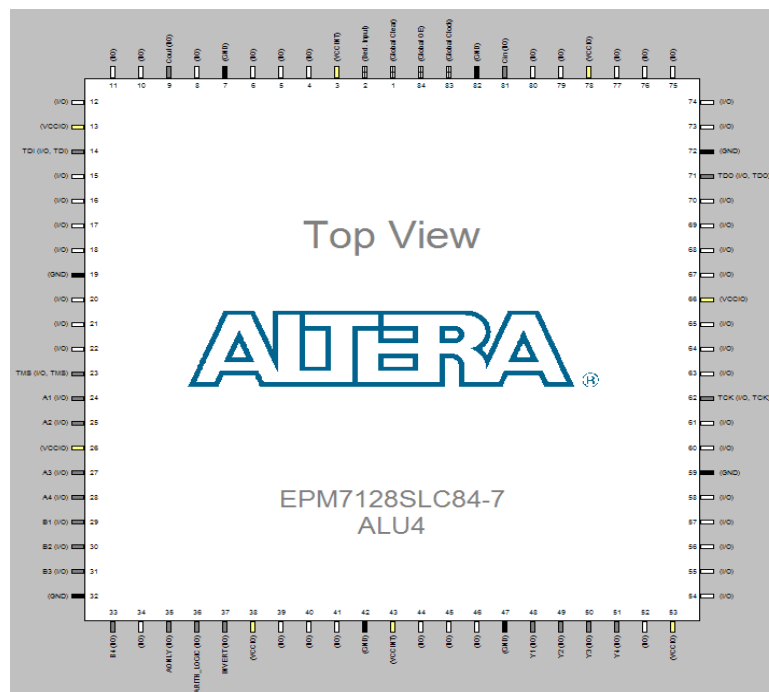




34. After the downloading is completed, as indicated in the progress bar by 100% on the blue bar graph, the object design is ready for testing on the circuit board. Keep in mind that the relatively simple ALU logic circuit will use much more space of the total capacity of the target device compared to our previous simpler circuit (Half\_Adder).
35. View the interior LABs by selecting **Interior Labs** from **View** of the main menu.



36. View the top of your ICs package EPM7128SLC84-7 CPLD. Notice the pins locations for our inputs, outputs, and ALU controls.





### Lab Procedure: Testing the ALU Circuit in the ALTERA UP2 Development Board



37. Connect MAX\_SW1 (8 inputs, 1, 2, 3, 4, 5, 6, 7, 8) switches to device pins 24, 25, 27, 28, 29, 30, 31, and 33 as shown in the picture above.
38. Hard-wire the ALU controls (Aonly, Arith\_Logic and Invert) pins 35, 36, and 37 to MAX\_SW2 (6, 7, 8).
39. Hard-wire the ALU output pins 48, 49, 50 and 51 to LEDs D1, D2, D3, and D4.
40. Use the table below to make the connections to the eight switches that provide logic-level signals for MAX\_SW1 and MAX\_SW2 by inserting one end of the hook-up wire into the female header aligned with the appropriate switch. Insert the other end of the hook-up wire into the appropriate female header assigned to the I/O pin of the EPM7128S device. The switch output is set to logic 1 when the switch is open and set to logic 0 when the switch is closed. The power, ground, and JTAG signal pins are not accessible through these headers.

P1		P2		P3		P4	
Outside	Inside	Outside	Inside	Outside	Inside	Outside	Inside
75	76	12	13	33	34	54	55
77	78	14	15	35	36	56	57
79	80	16	17	37	38	58	59
81	82	18	19	39	40	60	61
83	84	20	21	41	42	62	63
1	2	22	23	43	44	64	65
3	4	24	25	45	46	66	67
5	6	26	27	47	48	68	69
7	8	28	29	49	50	70	71
9	10	30	31	51	52	72	73
11	X	32	X	53	X	74	X

**Pin Numbers for Each Prototyping Header**

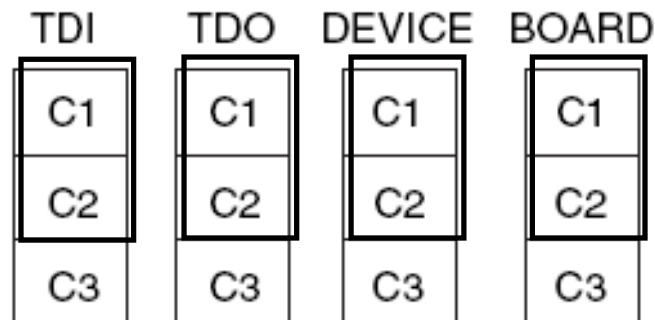


41. Use the table below to connect the LEDs D9 through D16 in the same sequence to the female headers (i.e., D9 is connected to position 1, and D10 is connected to position 2, etc.).

Female Header Position		LEDs		Female Header Position		LEDs	
1	○	D1	●	1	○	D9	●
2	○		D5	2	○		D13
3	○	D2	●	3	○	D10	●
4	○		D6	4	○		D14
5	○	D3	●	5	○	D11	●
6	○		D7	6	○		D15
7	○	D4	●	7	○	D12	●
8	○		D8	8	○		D16

**LED Corresponding Female Headers**

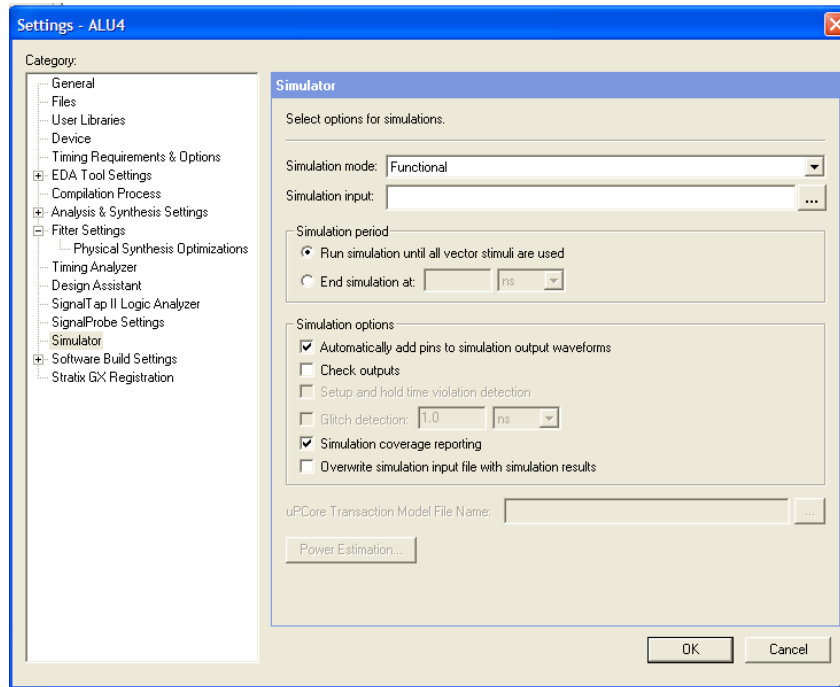
42. Set the board jumpers as shown in the figure below in order to program this device.



Desired Action	TDI	TDO	DEVICE	BOARD
Program EPM7128S device only	C1 & C2	C1 & C2	C1 & C2	C1 & C2



43. Click on the Start Programming icon of the Programming tool bar. Within a few seconds, the EPM7128S devices will be programmed with the ALU.pof file that corresponds to our ALU design circuit. Make sure the selection says *Functional* under *Assignments, Settings*.



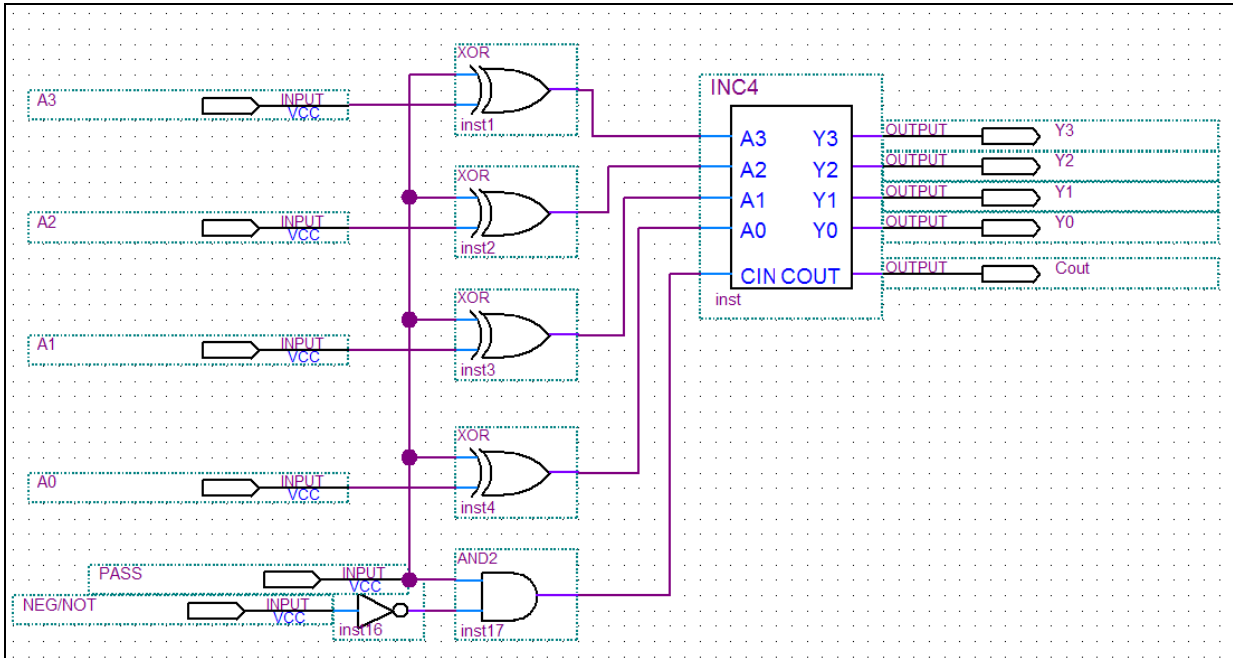
44. After all connections are made, test the EPM7128S implementation of our ALU design.

A ONLY'	ARITH'/LOGIC	INVERT	F	HEX INPUT A	HEX INPUT B	HEX OUTPUT
0	0	0	PASS-THROUGH	2	5	2
0	0	1	2's COMPLIMENT of A	2	5	E
0	1	0	PASS-THROUGH	2	5	2
0	1	1	1's COMP of A	2	5	D
1	0	0	A plus B	2	5	7
1	0	1	(2's COMP A) plus B	2	5	3
1	1	0	A OR B	2	5	7
1	1	1	(1's COMP of A) OR B	2	5	D
0	0	0	PASS-THROUGH	A	3	A
0	0	1	2's COMPLIMENT of A	A	3	6
0	1	0	PASS-THROUGH	A	3	A
0	1	1	1's COMP of A	A	3	5
1	0	0	A plus B	A	3	D
1	0	1	(2's COMP A) plus B	A	3	9
1	1	0	A OR B	A	3	B
1	1	1	(1's COMP of A) OR B	A	3	7



## Lab Questions

1. Complete the table below for the following NOT/NEG circuit.



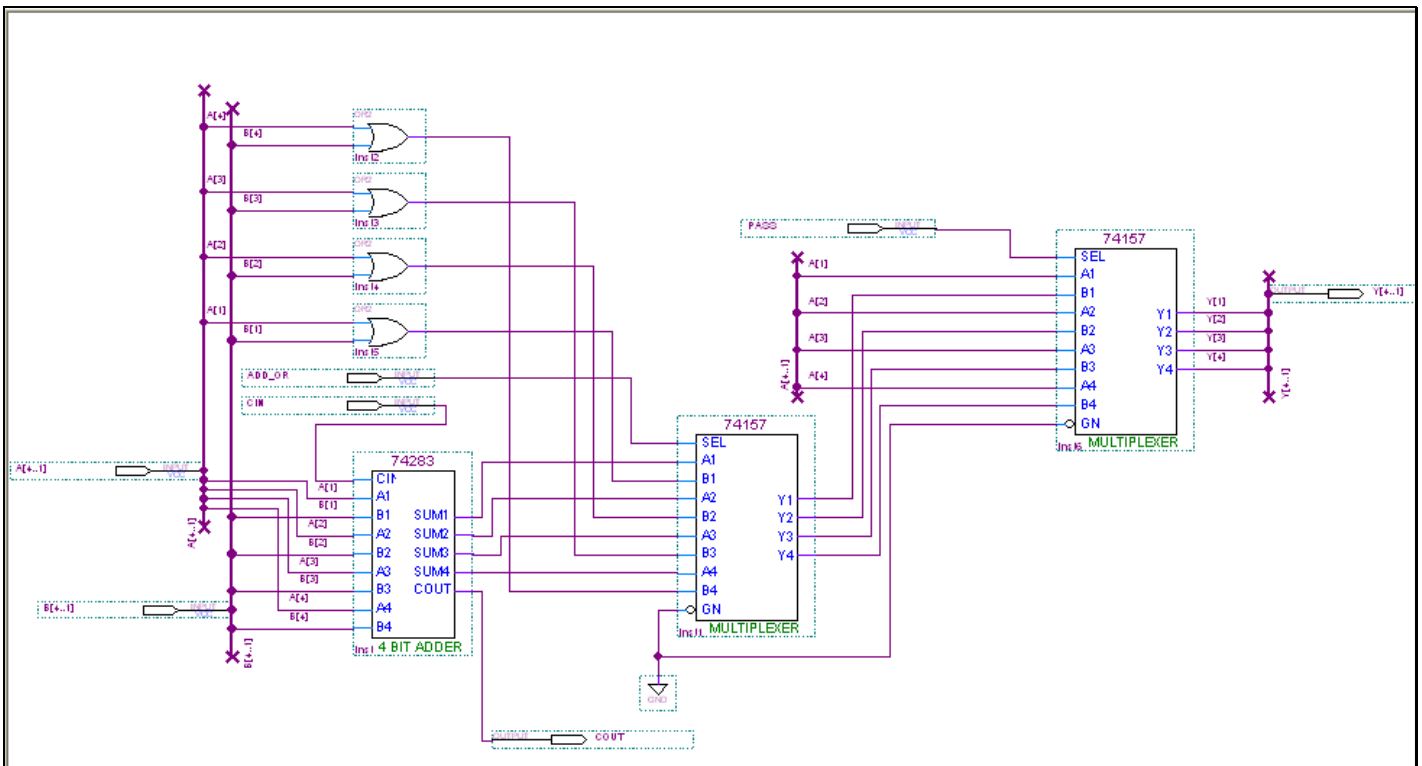
NEG/NOT	PASS	Input values of A3 A2 A1 A0	Output values of Y3 Y2 Y1 Y0	Indicate the function of the output
0	0	5 (0101)		
0	1	A (1010)		
1	0	C (1100)		
1	1	9 (1001)		





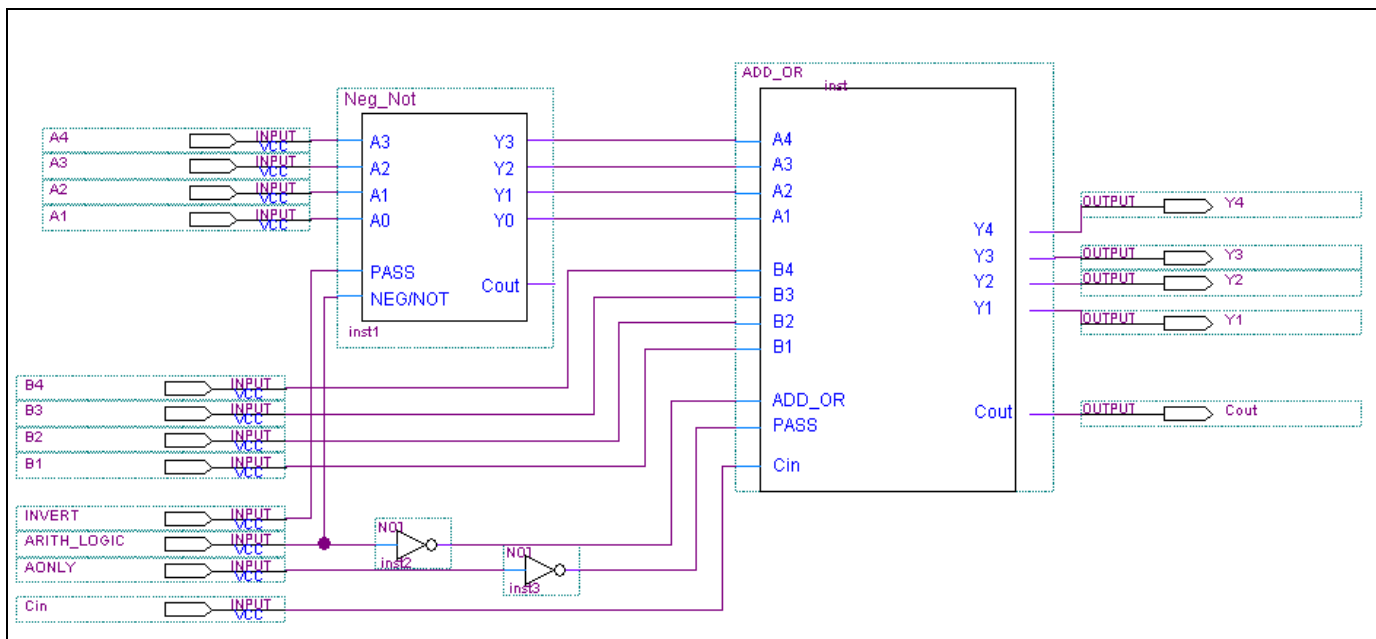
2. Complete the table below for the following ADD/OR circuit.

ADD/OR	PASS	Input values of A3 A2 A1 A0	Input values of B3 B2 B1 B0	Output values of Y3 Y2 Y1 Y0	Function of the Output
0	0	5 (0101)	9 (1001)		
0	1	A (1010)	C (1100)		
1	0	C (1100)	A (1010)		
1	1	9 (1001)	5 (0101)		





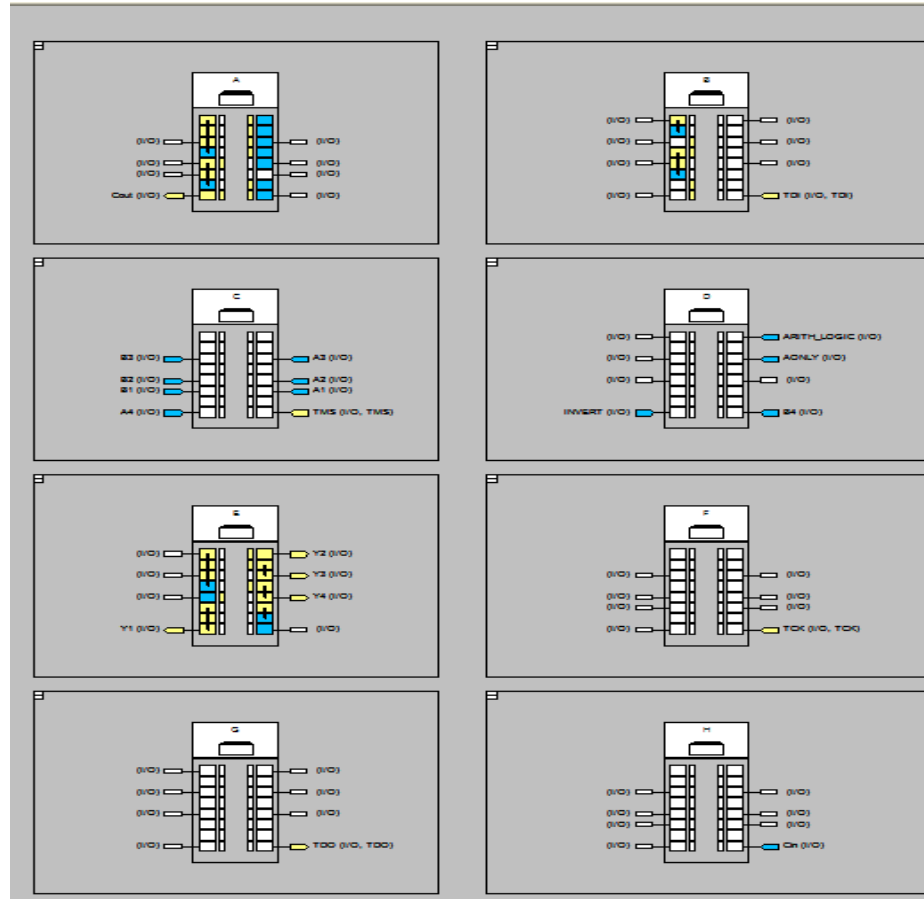
### 3. Complete the table below for following ALU circuit.



A ONLY'	ARITH'/LOGIC	INVERT	HEX INPUT A's	HEX INPUT B's	HEX OUTPUT Y's	Function
0	0	0	2	5		
0	0	1	3	5		
0	1	0	A	2		
0	1	1	5	4		
1	0	0	5	2		
1	0	1	4	6		
1	1	0	A	B		
1	1	1	F	C		



4. Explain the following floor plans of our ALU circuit targeting the EPM7128SLC84-CPLD.





## ALU Circuit

