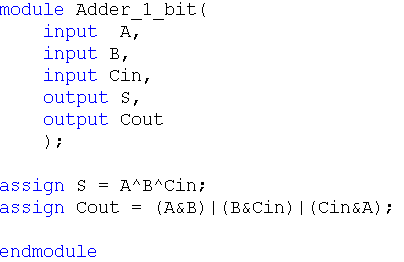
**Experiment No 2: Synthesis of Adders and ALU**

**Exercise 2.1 Synthesis of 1-bit Adder.**

Synthesis is a process by which the RTL Verilog implementation is converted in to actual circuit. The aim of this experiment is to learn how to view and analyze the synthesized circuit Xilinx. Please follow the steps as shown below:

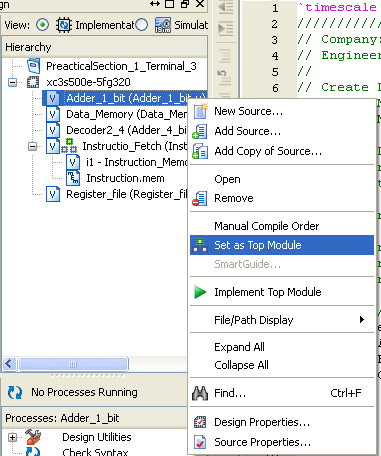
1. **Implement a simple 1-bit adder:** Adder\_1\_bit.v **(Please use the code implemented in Experiment 1)**



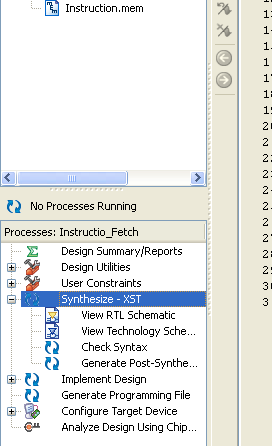
1. **Change the view to implementation mode**



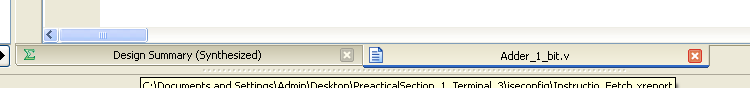
1. **Right click on the Adder\_1\_bit.v and set it as Top Module**

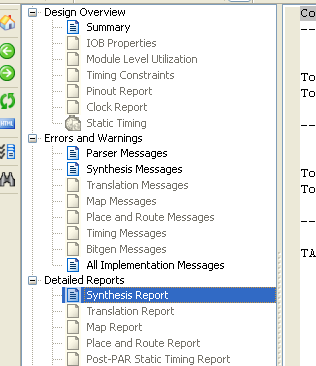


1. **Double click on the Synthesize-XST in the processes window. If synthesis fails then it means that the Verilog code you have written is not synthesizable. You have to rewrite the Verilog code by removing non synthesizable parts. (e.g. initial block is used only in test bench if you write an initial block in design file, then that design will not synthesize)**

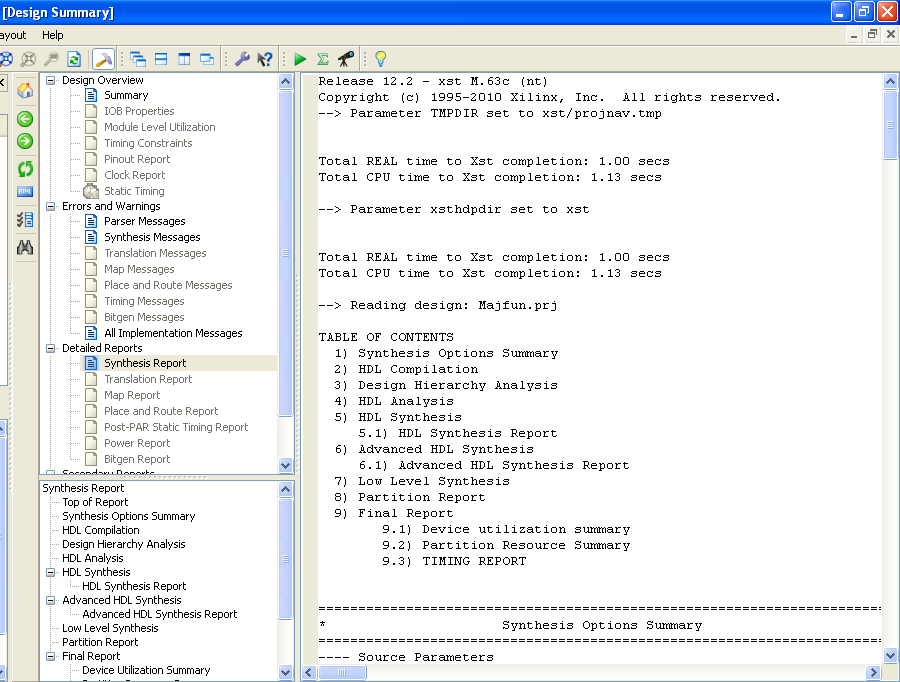


1. **Once the synthesis is done you will be able to check the synthesis report by clicking on Design summary tab and then double clicking on the synthesis report. Synthesis report will have the information about the delay and the resources used. Since the Xilinx is mainly used for FPGA based implementation the results will be for a specific FPGA.**

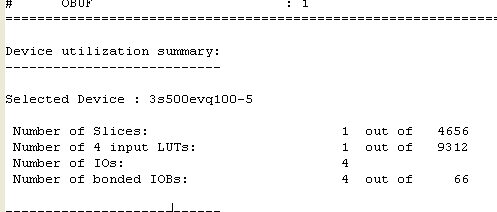




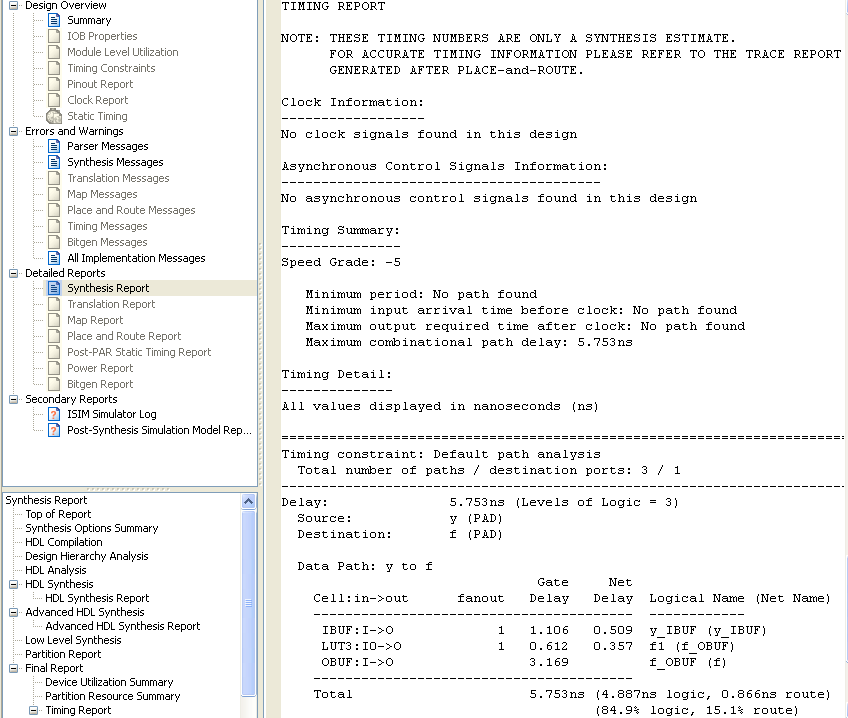
**A sample report is shown below**



**If you scroll down the report you will get the “Device utilization summary:” which will give you the LUT usage details. Also there will be a section called “Timing Report” which will give you the timing details including total delay of the circuit.**

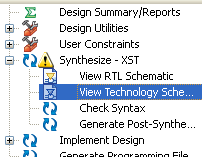


**Device utilization summary**

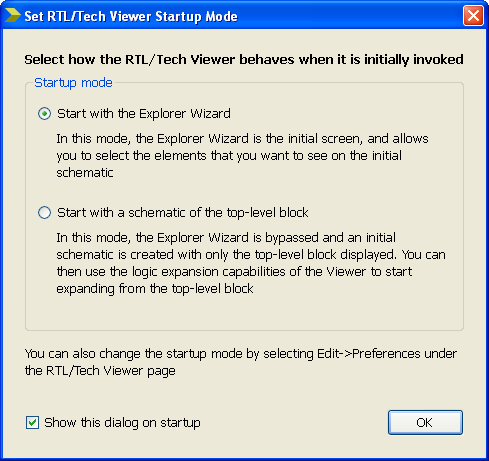


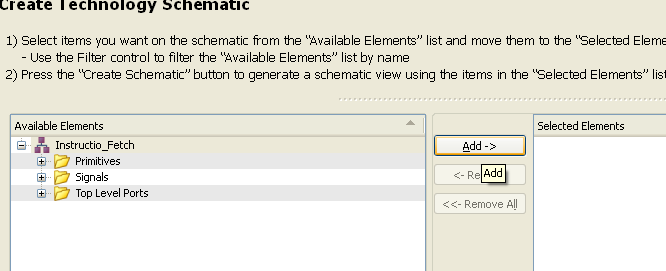
**Timing Report**

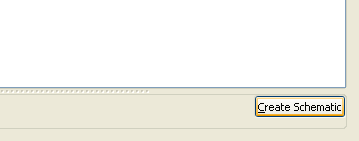
1. **You can also check the synthesized circuit by double clicking on view technology schematic 🡪 click ok 🡪 Add 🡪 Create schematic**



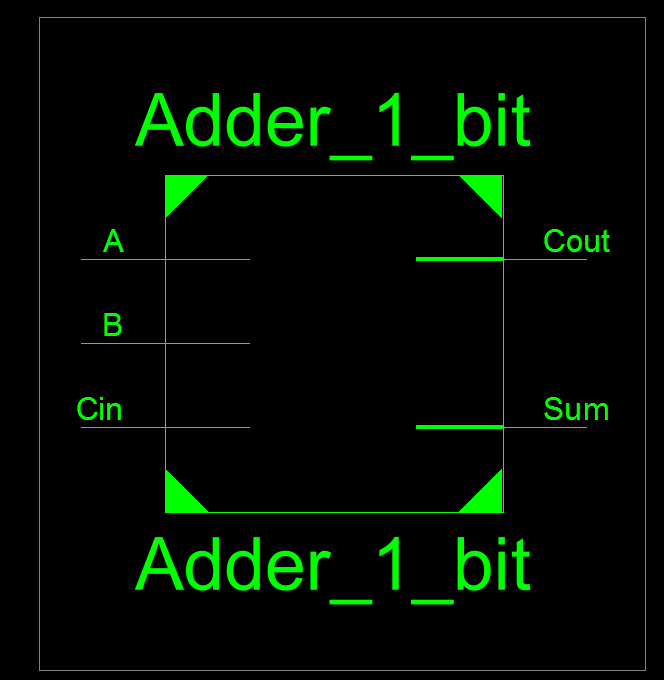
**Technology schematic shows the circuit implemented using LUTs (FPGA implementation). You can also check RTL schematic which shows circuit implemented using basic gates.**

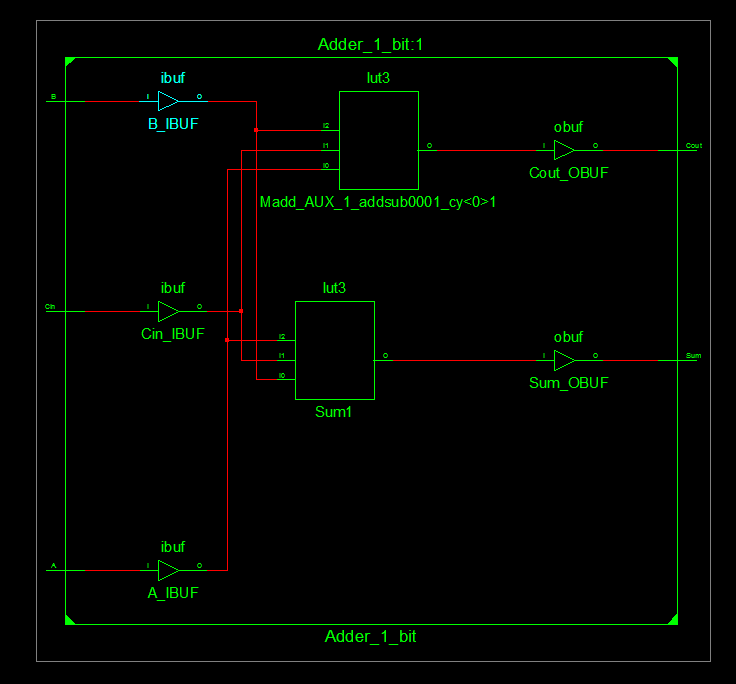




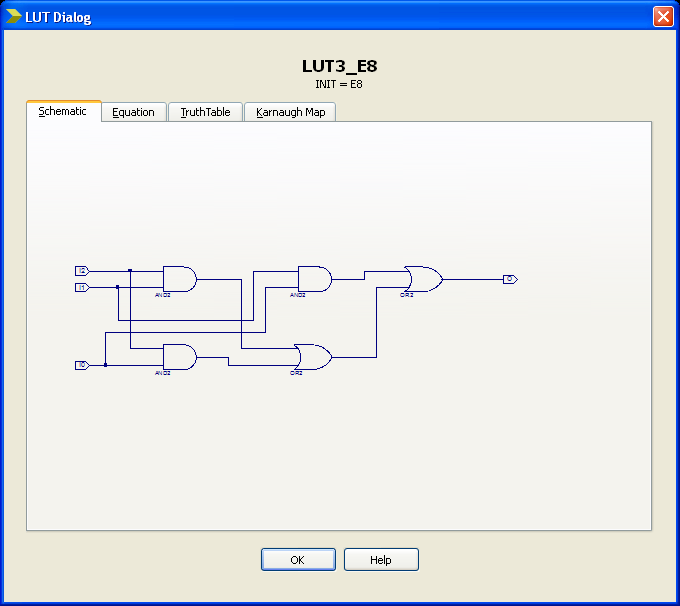


1. **The following window will appear. Double click on module to see the implementation in terms of LUTS.**

****



1. **You can double click on the LUTs to check the functions that they implement. You can also check the equation, truth table and k-Map that LUT implements by clicking on different tabs.**

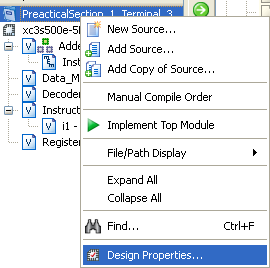


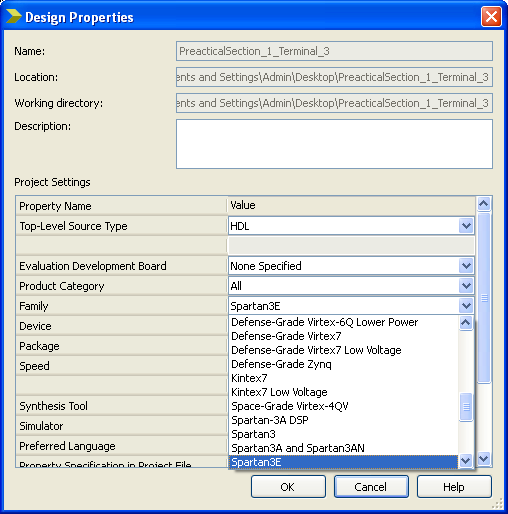
**From the synthesis report find the Delay and the resource information. Note down synthesis details in your observation book.**

Delay: 5.776ns

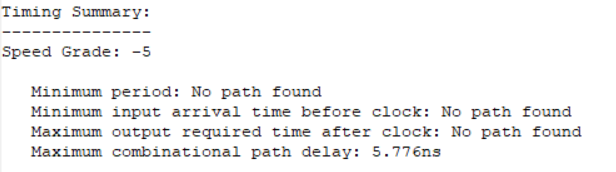
Number of LUTs Used: 2

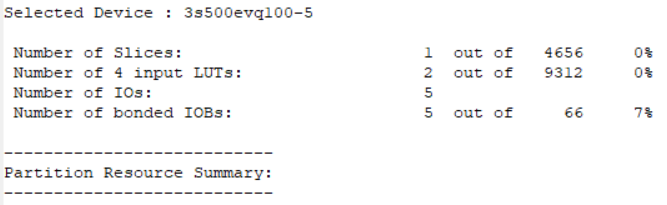
**Change the Target FPGA in Design properties window find the Delay and resource information. You can change the target FPGA by right clicking on the Project🡪 select Design properties window 🡪 Then select the FPGA family and device type.**

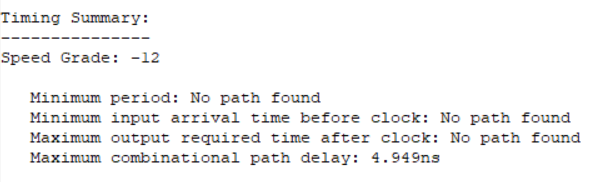
****

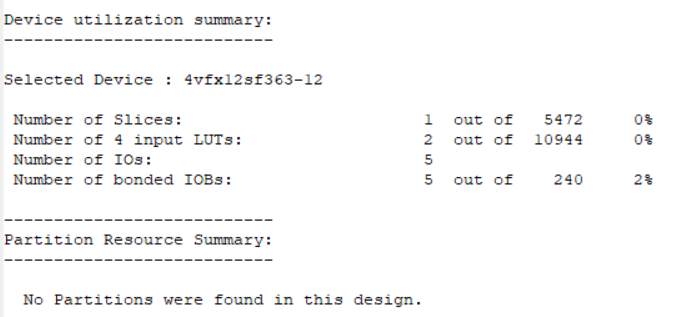


**From the synthesis report find the Delay and the resource information for 1-bit adder on Spartan3E XC3S500E and Virtex4 XC4VFX12. Note down synthesis details in your observation book.**

Delay for Spartan3E XC3S500E: 5.776ns

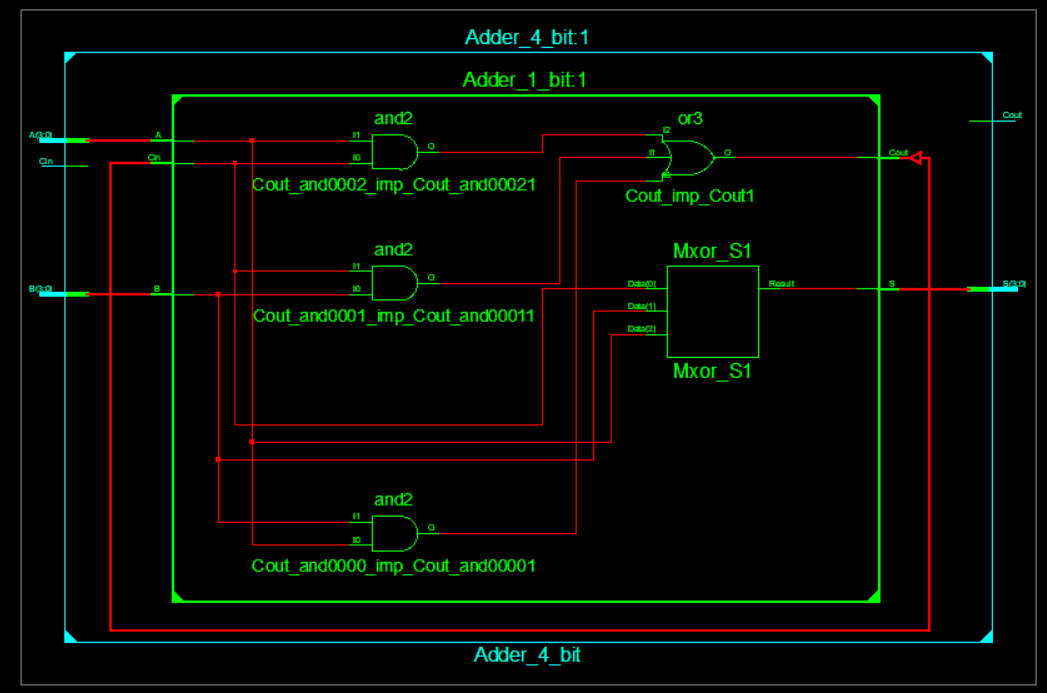
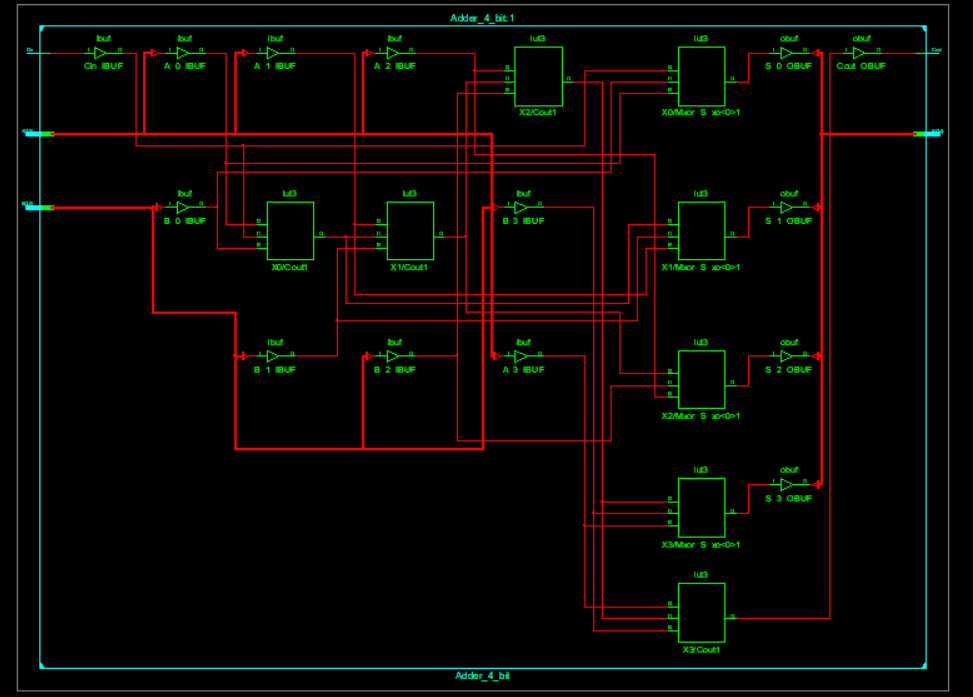
Number of LUTs Used for Spartan3E XC3S500E: 2

Delay for Virtex4 XC4VFX12: 4.949ns

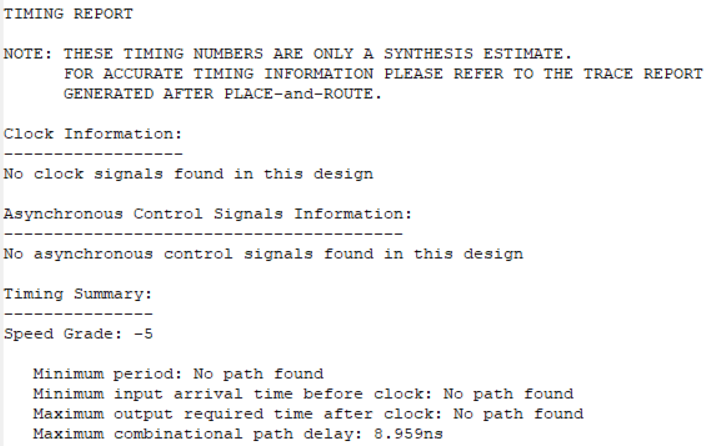
Number of LUTs Used for Virtex4 XC4VFX12: 2

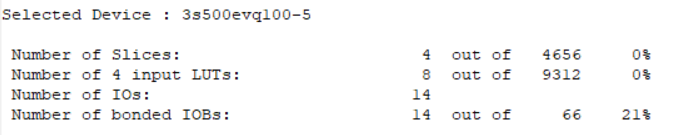
**Exercise 2.2 Synthesis of 4-bit Adder.**

**Use similar process as explained above (for 1-bit adder) to synthesize 4-bit adder, which was implemented in Experiment 1.**

Copy the image of synthesized Technology schematic and RTL schematic: RTL Schematic: Technology schematic: 

**From the synthesis report find the Delay and the resource information for 4-bit adder on Spartan3E XC3S500E. Note down synthesis details in your observation book.**

Delay for Spartan3E XC3S500E: 8.959ns 

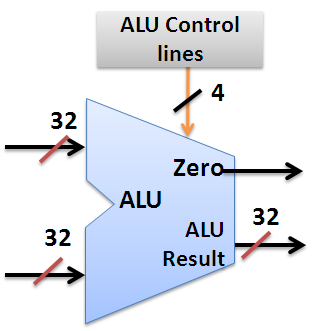
Number of LUTs Used for Spartan3E XC3S500E: 8 

**Exercise 2.3 Design and Synthesis of simple ALU.**

ALU (Arithmetic and Logical unit) is an important component of any processor. ALU performs different arithmetic and logical operations depending on the control lines. In this section you will be implementing a simple ALU for a given set of specifications.

**ALU Specifications:**

This ALU has two 32-bit operands and 4 control lines as inputs. It has two outputs a 32-bit ALU result and a Zero indicator which becomes logic 1 if and only if the 32-bit ALU result is Zero. The ALU has to perform different functions according to the value of 4 control lines. Block diagram of ALU along with the mapping of control lines to functions performed is shown below.



**B**

**A**

|  |  |
| --- | --- |
| **ALU Control lines** | **Function** |
| 0000 | Bitwise-AND |
| 0001 | Bitwise-OR |
| 0010 | Add (A+B) |
| 0100 | Subtract (A-B) |
| 1000 | Set on less than |

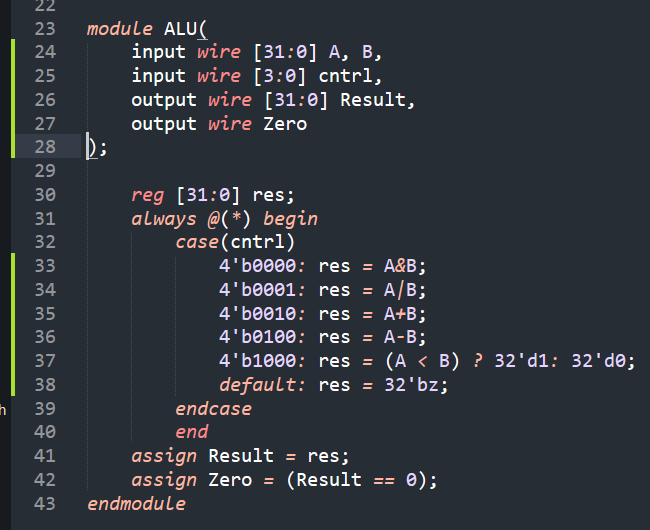
\*Set on less than (ALU Result = 1 if A<B

else ALU Result =0)

**Implement the ALU for above specifications in Verilog using behavioral modeling (Hint: Use Case statement).**

**Note: Partial code for ALU will be available as hint in CMS at time specified by instructor. This code will be available only for limited amount of time.**

**Paste the image of your Verilog code.**

Answer:  **code: **

**Write the test bench to test the ALU. Your test bench should have 7 different test patterns as mentioned below. (Assume test pattern changes after every 20 time units)**

Test case 1: A = 23, B = 42, ALUContol = 4`b0000.

Test case 2: A = 23, B = 42, ALUContol = 4`b0001.

Test case 3: A = 23, B = 42, ALUContol = 4`b0010.

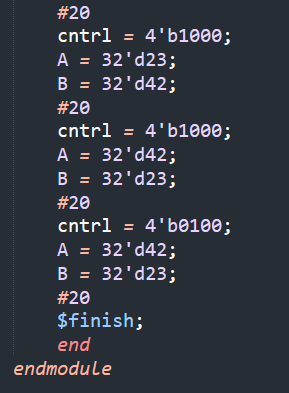
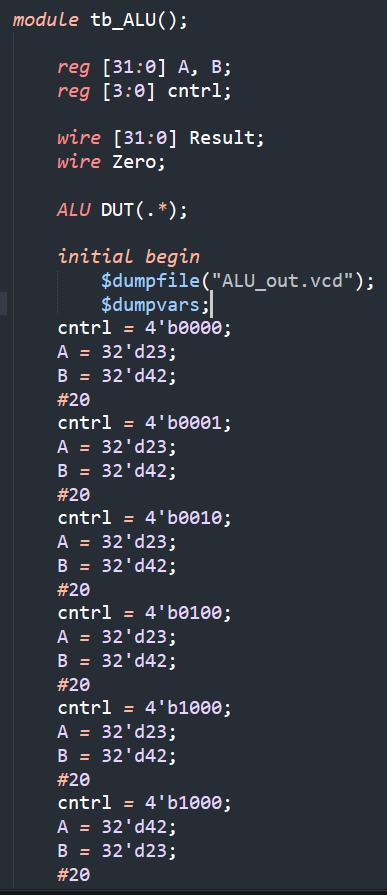
Test case 4: A = 23, B = 42, ALUContol = 4`b0100.

Test case 5: A = 23, B = 42, ALUContol = 4`b1000.

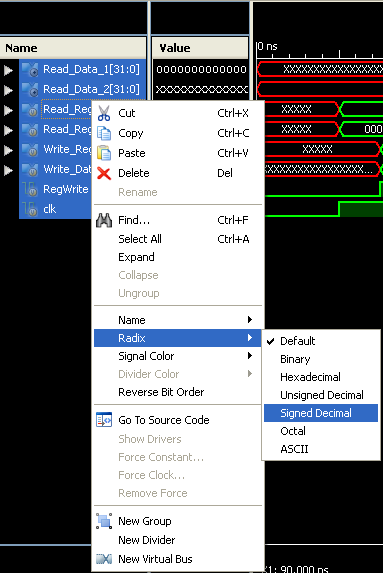
Test case 6: A = 42, B = 23, ALUContol = 4`b1000.

Test case 7: A = 42, B = 23, ALUContol = 4`b0100.

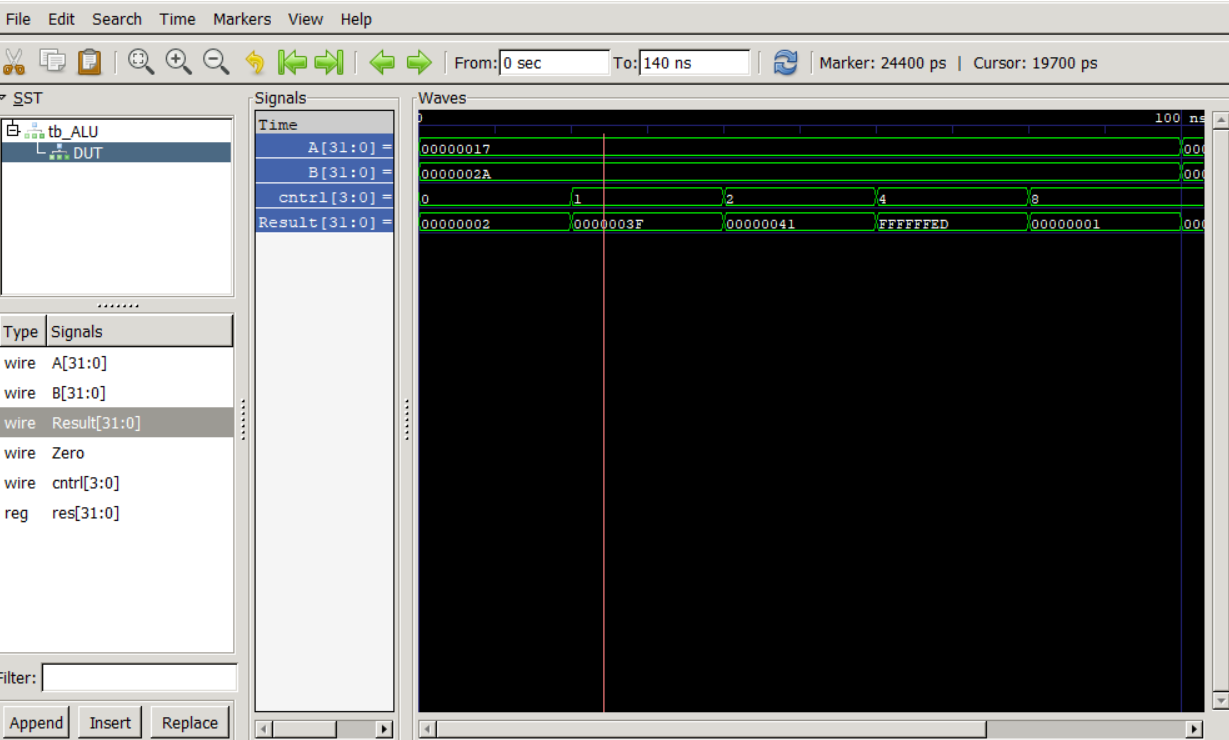
**Paste the image of your test bench with above test cases.**

Answer:  ****

For easy viewing of the testbench waveforms where signals are multi bit, you can change the view in simulation window to decimal/hexadecimal mode. Right click on all the input output signals select Radix and then click on signed decimal/hexadecimal.



**Copy the image of waveform window that is generated for your Testbench? (Change display radix to Hexadecimal).**

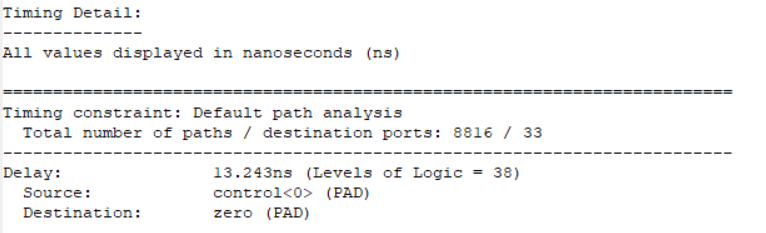
Answer: 

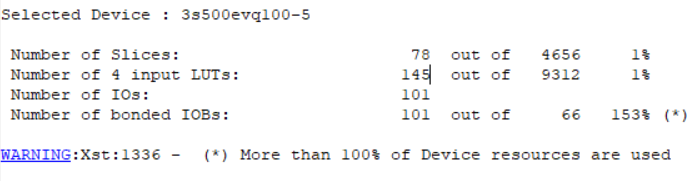
**How many different functions can be implemented by ALU with 4-control lines?**

Answer: 65536

**From the synthesis report find the Delay and the resource information for ALU on Spartan3E XC3S500E.**

Answer

**Delay: Maximum combinational path delay: 13.243ns**

**Number of LUTs Used:**  145 

**List the concepts you learnt from this experiment (Conclusions/Observations)**

Answer:We learned how to generate and analyze post-synthesis reports in vivado

W