**COEN 313 – Digital System Design II**  **Jafar Abbas  
LAB 2 – Structural and Concurent VHDL** 26346650  
 Winter 2017 – February 27th 2017

**Objectives**

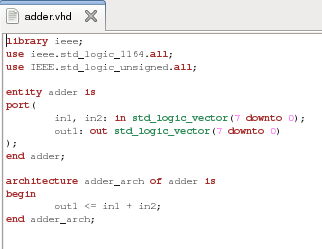
The purpose of this lab is to become acquainted with structural and concurrent VHDL. A secondary purpose is to become acquainted with different VHDL coding styles and to gain insight on the differences between writing VHDL code for simulation versus synthesis. As an example, we will be implementing an **8-bit two’s complementer circuit.**

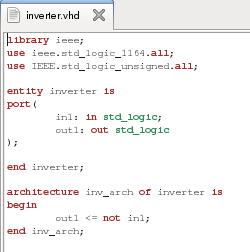
**Procedure**

This lab follows the procedure outlined in Parts I, II, III, and IV of the tutorial “Digital Logic Simulation and Synthesis Using Modelsim, Precision RTL, and Xilinx ISE”. The end-goal is to simulate, synthesize and download to the Xilinx FPGA board the 8-bit two’s complementer circuit shown in the LAB 2 manual.

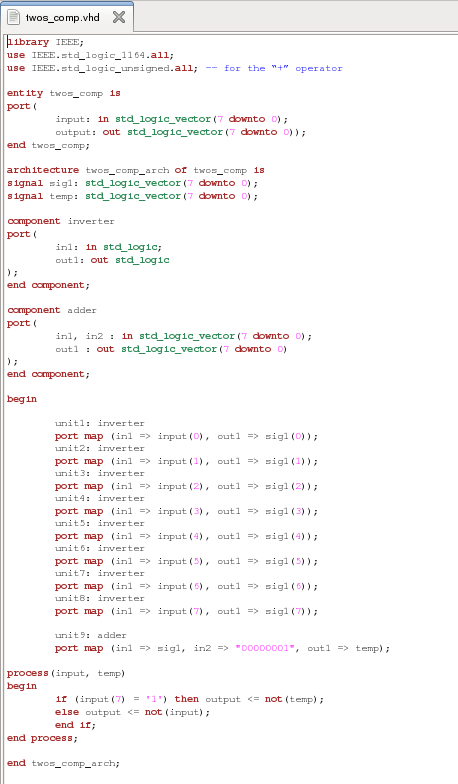
The two’s complement should only apply to numbers that are negative, so if an input is positive then it is unmodified.

**Results**

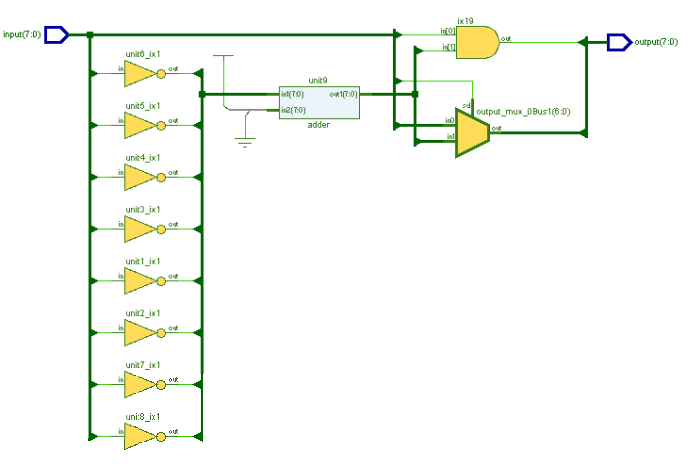
VHDL Code for Inverter: VHDL Code for Adder:



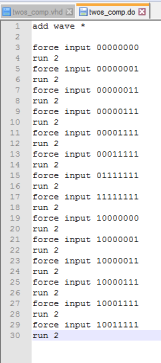
VHDL for Two’s complement using port mapping of the Adder and Inverter previously described:



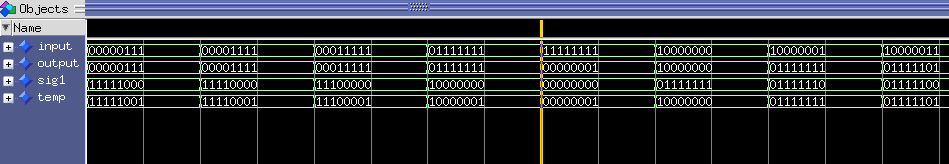
Synthesized circuit using port mapping for the inverters:



DO file to test my circuit:



Corresponding waveform:



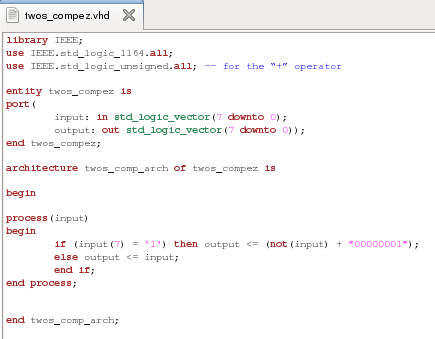
**Conclusion**

This lab experiment was successful in introducing implementation using port map statements. We also observed the advantages of concurrent signal assignment states and vector data types in extension, readability and hardware mapping.

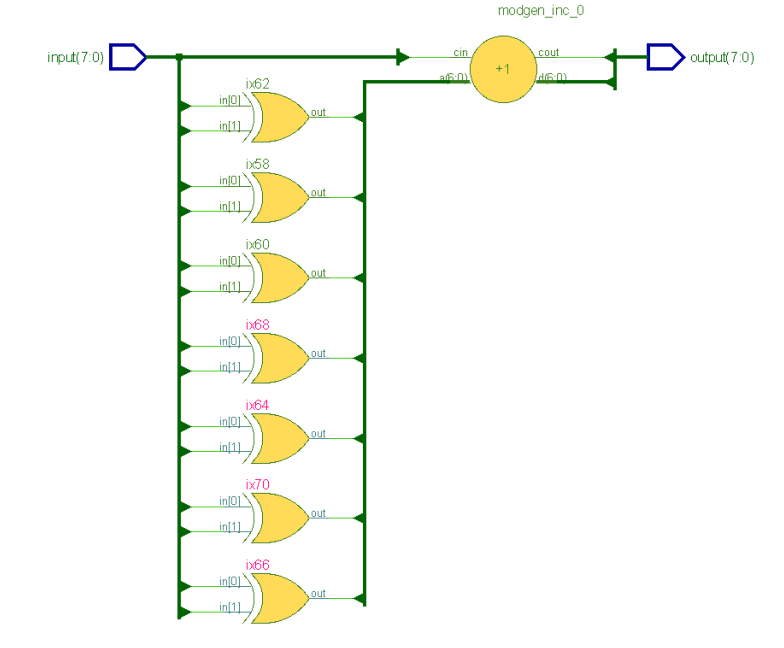
**Questions**

**1. Rewrite the VHDL code for only the “flip the bits” component making use of only CSA statements (no port maps). Re-synthesize your design with Precision RTL and compare the resulting RTL schematic diagram with that of the original design. Comment on any differences/similarities between the two schematics. You do not have to download this version of the circuit to the FPGA board.**

VHDL code for two’s complements using concurrent statements:



Synthesized circuit before adding output negation:



When I added the output negation, the circuit got further simplified.

This circuit is more efficient because it uses less components but it is much harder to read and understand.

**2a. Explain how you would re-write the original code if the input and output to the circuit where changed from std\_logic\_vector(7 downto 0) to std\_logic\_vector(15 downto 0).**

If the input was 16 bits then the entity and components will change accordingly:

entity twos\_comp is

port( input : in std\_logic\_vector(15 downto 0);

output : out std\_logic\_vector(15 downto 0));

end twos\_comp;

The same needs to be done for the adder entity (and the component):

entity adder is

port(

in1, in2: in std\_logic\_vector(15 downto 0);

out1: out std\_logic\_vector(15 downto 0)

);

end adder;

The inverter needs to be port mapped 8 more times (i.e. for bit 0 to 15):

--Pseudo-code follows:

Unit0…15: inverter

port map (in1 => input(0…15), out1 => sig1(0…15));

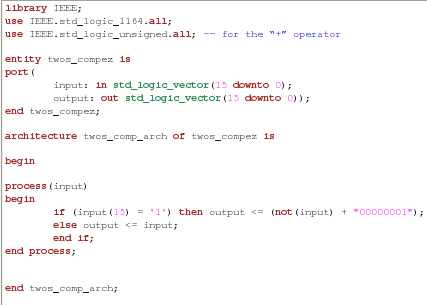
Also we would check bit ‘15’ to see if it’s the number is positive or negative.

**2b. Would it be practical to extend the method as described in Question 2a to vectors of size 256 bits or 512 bits or 1024 bits?**

It would be very impractical to apply 256, 512 or 1024 bits extensions to implementation described in Question 2a because we would have way more lines of code, it would be easy to make mistakes, would take very long to actually do and it will take a lot longer to compile.

**3. Explain how you would use only concurrent signal assignment statements to design a two’s complementer with an input and output of size 16 bits with the following entity specification:**

Simply by changing the input, output and the bit to check for positive:



**4. After having answered Question 3, can you now appreciate the advantages of concurrent signal assignment statements and vector data types over the laborious port map statement?**

I appreciate the advantage of concurrent signal assignment statements, vector data types over laborious port map statements!

In a lot of cases, it’s a lot easier to represent a circuit using concurrent signal assignments and vector data types are very easily extendable and easy to use.

**5. What changes are necessary in your top-level VHDL code (for the twos\_comp entity) in order to be able to simulate the twos\_comp entity which uses the version of the ““flip the bits” component which was designed in Question 1? Were any changes necessary in the top-level twos\_comp entity required in order to synthesize the design as in Question 1? In other words, explain the differences (between a simulation point of view and a logic synthesis point of view) in a component specification statement such as:**

**for bit\_flipper : flip\_the\_bits use entity WORK.flip\_the\_bits(port\_maps);**

I seriously read this question 5 times and I still don’t understand what is being asked.