Digital Systems
Laboratory Exercise 4
First VHDL Coding

NAME: ____

LAB OBJECTIVES

1. Write and debug an SOP design using VHDL
2. Create a testbed and simulate the SOP Design
3. Derive a truth table for the SOP design using the simulation
4. Configure your FPGA (on Basys 3 board) to implement SOP

PRE-LAB

1. Read the **laboratory** thoroughly.
2. Create a truth table for a full adder with outputs of Sum and Carry. (The truth table is included in the lab and needs to be completed.)

Deliverables:

At the end of lab, turn in a PDF file showing all of your pre-lab and lab work as required below

Within the lab:

1. Fill in the truth tables provided within the lab. (You can scan the lab or just truth tables as your solution.)

Include the following with your lab:

2. Screenshot showing full zoomed-in simulation of the SOP design for all possible input combinations
3. Copies of your VHDL code including the testbed and relevant sections of the constraints file.

NOTE: Failure to turn-in the required material by Friday (at 5pm) will result in a grade of 0 for the lab. **(No late labs or extensions will be accepted.)**

NOTE: You are to write your own VHDL code. The same code turned in by two or more students will result in a grade of 0 for all involved students.

INTRODUCTION

At this point in the semester, you should be able to

1. Use Vivado to create VHDL code
2. Setup a testbed,
3. Run a simulation,
4. Interpret the simulation (create a truth table from the simulation),
5. Setup constraints for the Artix 7 FPGA (on your Basys 3 board),
6. Configure your Artix 7, and
7. Demonstrate a working fully configured solution.

Word Problem to Solve:

In today's lab, you will be writing your first VHDL code. The component that you will be creating is what is called a "Full Adder." A "Full Adder" or "FA" takes three inputs (Cin, A1, and B1 and adds them together to create two outputs. The least significant output is called "Sum" and the most significant bit of the output is called "Cout" where C is shorthand for "Carry." A "Full Adder" takes a Carry input (Cin) from another location and adds it to two bits (A1 and B1). There are 8 possible results.

Output (Cout, Sum) = Cin + A1 + B1 (Note in this instance + means add not OR)

NOTE: It is HIGHLY advisable to work on your VHDL code prior to coming to lab.

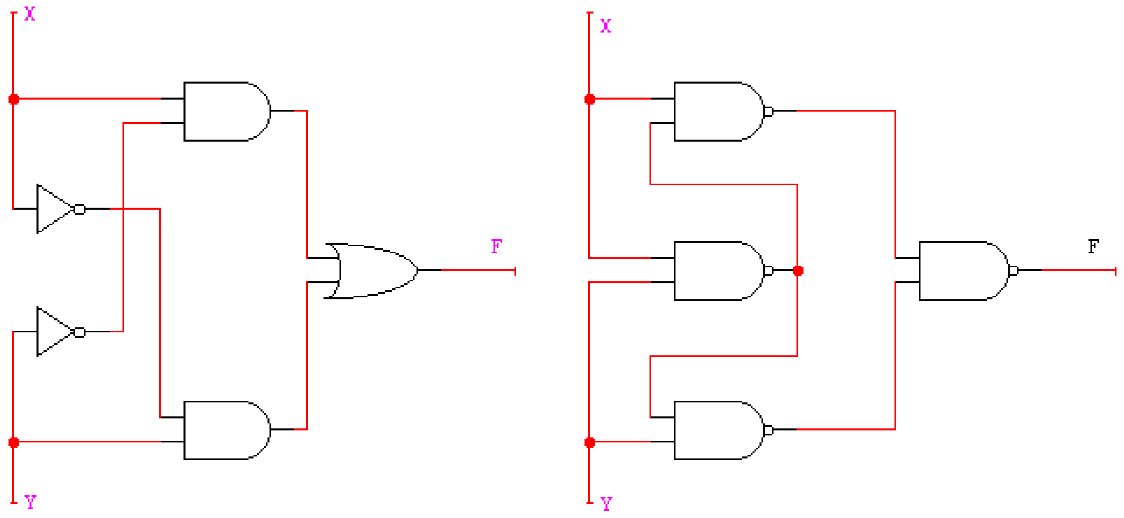
FOR YOUR PRELAB, create the truth table for the "Full Adder"

Inputs			Outputs	
Ain	Bin	Cin	Cout	Sum
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

NOTE: BEFORE PROCEEDING WITH IMPLEMENTATION OF YOUR FULL ADDER, REVIEW THE EXAMPLE THAT FOLLOWS:

VHDL Example:

In Lab 1 you were presented with the following SOP circuit:



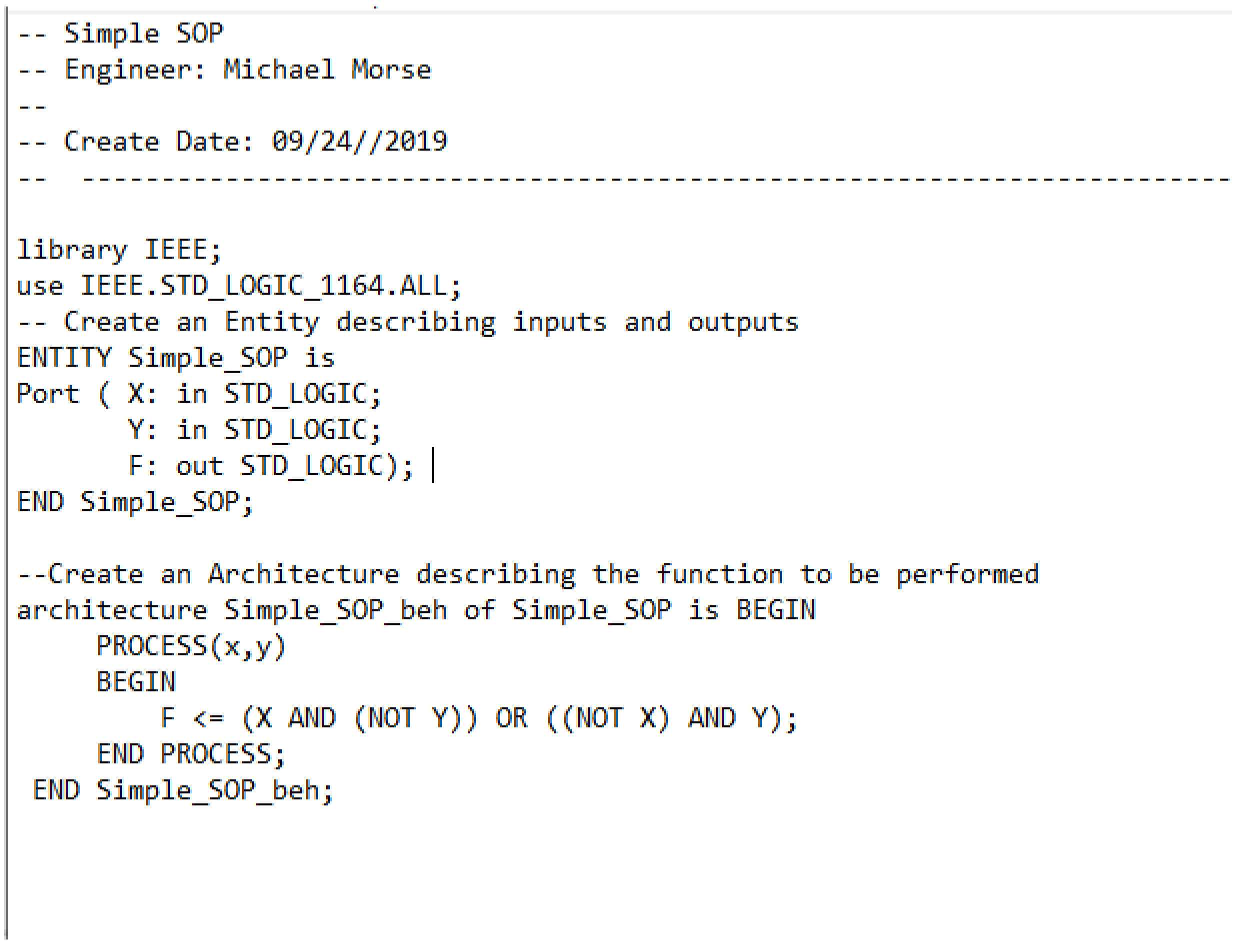
F = X exclusive-or Y

At this point in the semester, you should recognize the circuit on the left as an SOP design for function F and the circuit to the right is a NAND-NAND implementation of the same Function.

The Boolean equation derived from the circuit on the left is:

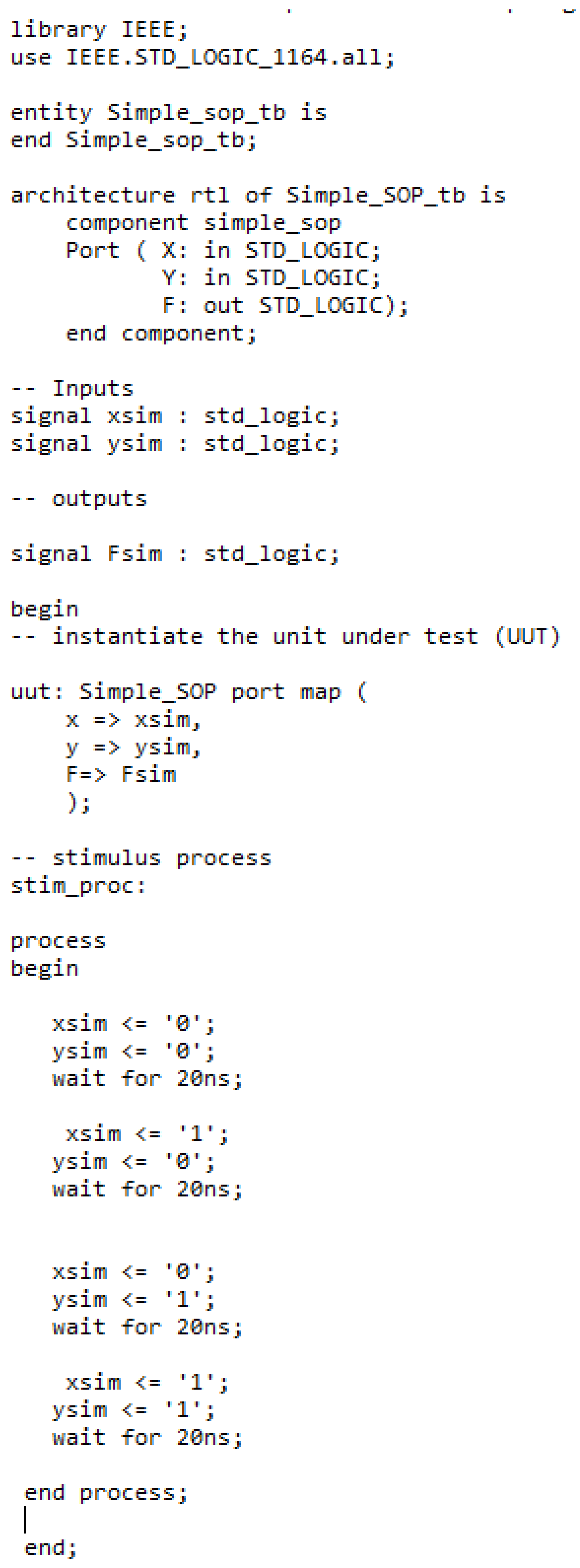
F=XY'+X'Y

The VHDL code to describe (and implement) this circuit is:



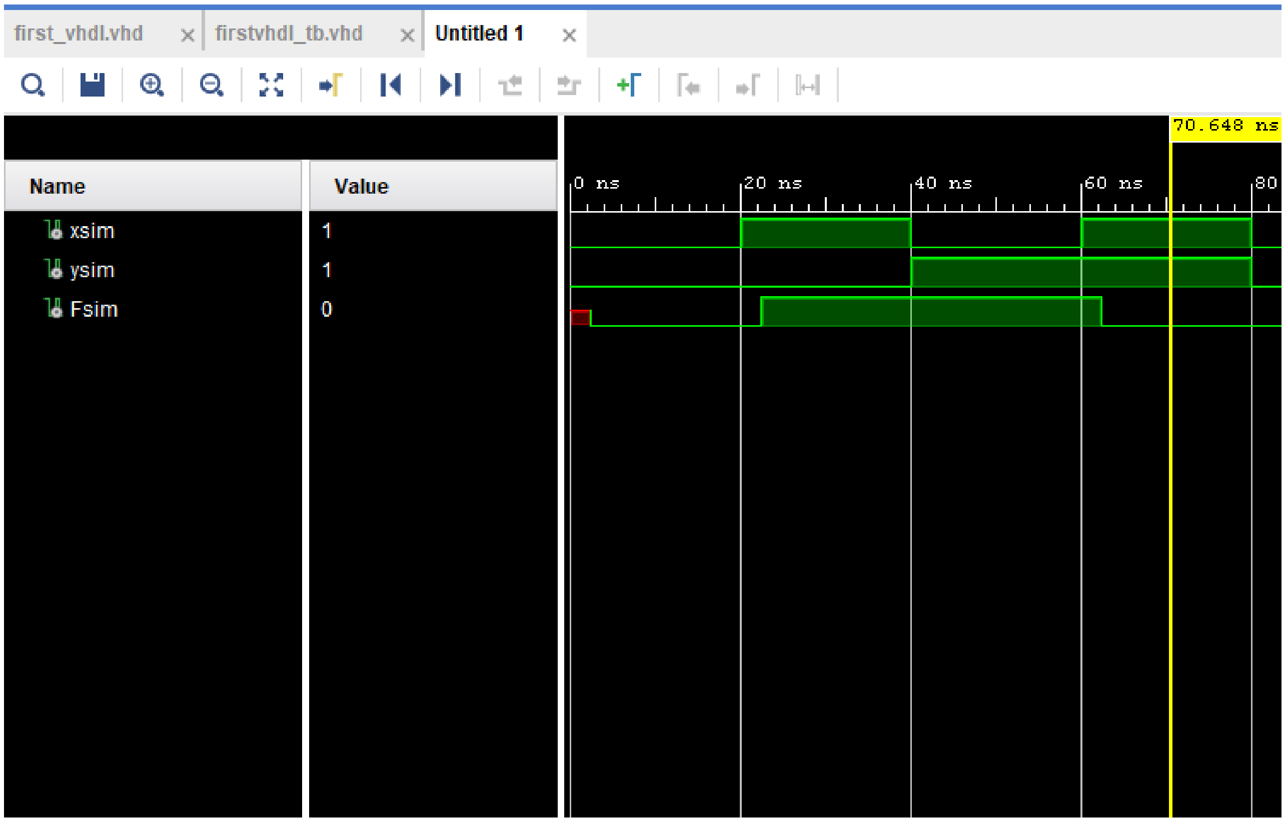
```
-- Simple SOP
-- Engineer: Michael Morse
--
-- Create Date: 09/24//2019
--  ------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- Create an Entity describing inputs and outputs
ENTITY Simple_SOP is
Port ( X: in STD_LOGIC;
       Y: in STD_LOGIC;
       F: out STD_LOGIC);
END Simple_SOP;

--Create an Architecture describing the function to be performed
architecture Simple_SOP_beh of Simple_SOP is BEGIN
     PROCESS(x,y)
     BEGIN
         F <= (X AND (NOT Y)) OR ((NOT X) AND Y);
     END PROCESS;
 END Simple_SOP_beh;
```

The testbed for the simulation would look as follows:

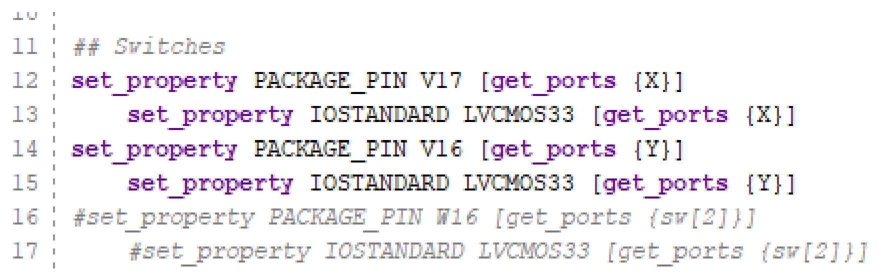


```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Simple_sop_tb is
end Simple_sop_tb;

architecture rtl of Simple_SOP_tb is
    component simple_sop
    Port ( X: in STD_LOGIC;
           Y: in STD_LOGIC;
           F: out STD_LOGIC);
    end component;

-- Inputs
signal xsim : std_logic;
signal ysim : std_logic;

-- outputs

signal Fsim : std_logic;

begin
-- instantiate the unit under test (UUT)

uut: Simple_SOP port map (
    x => xsim,
    y => ysim,
    F=> Fsim
    );

-- stimulus process
stim_proc:

process
begin

   xsim <= '0';
   ysim <= '0';
   wait for 20ns;

    xsim <= '1';
   ysim <= '0';
   wait for 20ns;


   xsim <= '0';
   ysim <= '1';
   wait for 20ns;

    xsim <= '1';
   ysim <= '1';
   wait for 20ns;

 end process;

 end;
```

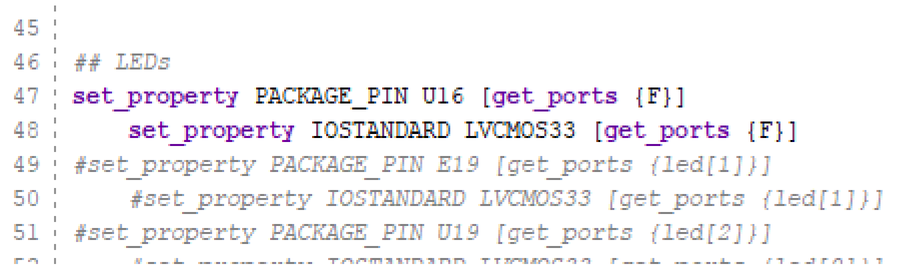
The simulation looks as follows:



Note: To implement on the Basys3 board you will need to delete the testbed VHDL program, modifications have to be made to the XDC constraints file to match the literals used in the VHDL program. For the example, this is as follows:



```
## Switches
set_property PACKAGE_PIN V17 [get_ports {X}]
    set_property IOSTANDARD LVCMOS33 [get_ports {X}]
set_property PACKAGE_PIN V16 [get_ports {Y}]
    set_property IOSTANDARD LVCMOS33 [get_ports {Y}]
#set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
```



```

## LEDs
set_property PACKAGE_PIN U16 [get_ports {F}]
    set_property IOSTANDARD LVCMOS33 [get_ports {F}]
#set_property PACKAGE_PIN E19 [get_ports {led[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
#set_property PACKAGE_PIN U19 [get_ports {led[2]}]
```

<u>**Although it is not required for the pre-lab, it would be prudent to try to reproduce the above example before implementing your Full Adder.**</u>

Part 1.

For part one of the lab, write the VHDL code and VHDL testbed for the "Full Adder" SOP design. Synthesize then simulate. **(It would be advisable to work on this before coming to lab.)**

Capture (and attach) a screenshot of the simulation for all possible values of the inputs:

Fill in the truth table below. NOTE: Your truth table MUST match with your simulation

Inputs			Outputs	
Ain	Bin	Cin	Cout	Sum
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

PART 2:

Delete the testbed from your project or create a new project without the testbed.

1. Insert and modify the proper constraints file.
2. Synthesize and implement.
3. Generate the Bitstream file.
4. Program the Basys board.

Complete the truth table below based on the performance of your Basys board.

Inputs			Outputs	
Ain	Bin	Cin	Cout	Sum
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Demonstrate your configured Basys board in a brief video where you:

1. **Show yourself and state your name**
2. **Aim your camera at the Basys board**
3. **Describe which Switches correspond to specific inputs and which LEDs correspond to specific outputs.**
4. **Step through all input combinations and show that the outputs are correct.**

TEXT your video to the instructor.