ECE 281 IN CLASS EXERCISE 3

TOP LEVEL DESIGN

OVERVIEW

So far in this course you have implemented individual components on your FPGA. However, one of the benefits of VHDL is that we can pull multiple components into a single design. To do this, we are introducing the concept of a top_level design. The top level is where you pull in and connect all of the different components you will be using before finally connecting them to the IO of the board. You will need to use the concept of a top_level design in Lab 2 so this is directly applicable.

OBJECTIVES

The objectives of this in-class exercise are for you to:

- Implement and test a full adder with half-adders using VDHL
- Gain more experience using tools (Git, Markdown, Xilinx Vivado)

DESIRED END STATE

You will be able to provide a live demo to your instructor. You are not required to turn in ICE3 however, you will need to demonstrate functionality and ensure the following files are pushed to your repository:

1. VHDL files used (top_basys3, half-adder and testbench) in a **code** folder
2. Constraints (.xdc) file in a **code** folder
3. .bit file used to program board

FULL-ADDER AND DESIGN FLOW

A half-adder takes two single-bit inputs and outputs their sum. A truth table, schematic symbol, and logic equations for a half-adder are shown in Figure 1. We will follow the digital design shown in Figure 3.
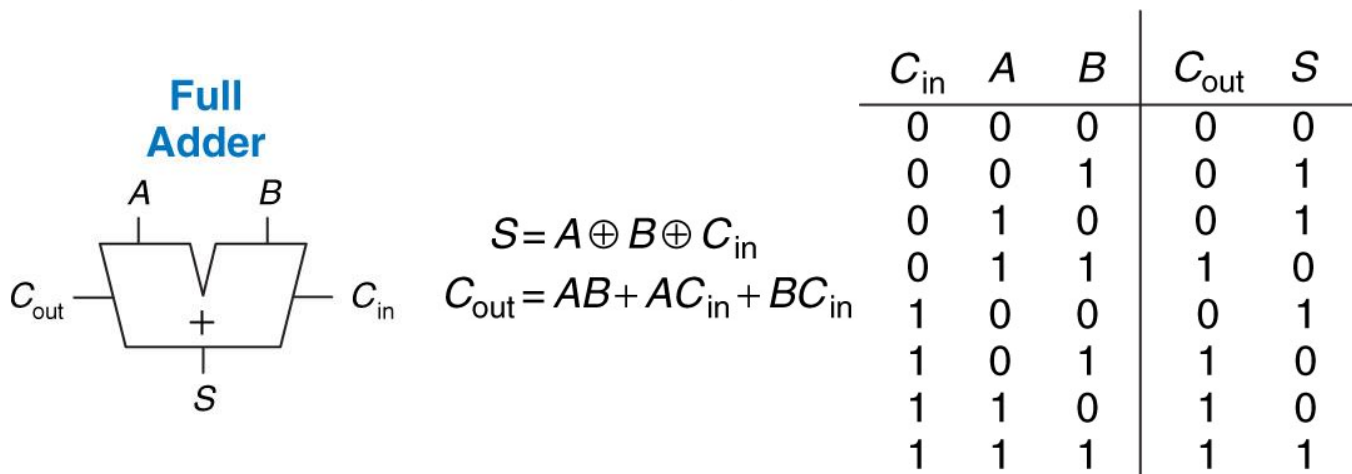

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Figure 1 – Full-adder schematic symbol, truth table, and logic equations

1. CREATE A VIVADO PROJECT

At this point you should NOT have an ICE3 directory in your git repository. Instead, Vivado will create that directory for you when you create a project. Create a new project titled ICE3. The process to do this was covered in ICE2. Refer back to ICE2 instructions if you need help

2. CREATE CODE AND IMAGES DIRECTORIES AND ADD FILES TO THEM

Navigate to your ICE3 folder in Windows Explorer (file manager), create a code folder, and add these provided files to it:

- o halfAdder.vhd and halfAdder_tb.vhd (these are the solutions. You can use yours if desired)
- o top_basys3 and top_basys3_tb
- o Constraints file (Basys3_Master.xdc)

Add/stage and commit your changes

- Prepend all commit statements for ICE3 with "ICE3 – ". For instance, your first commit might be:
  - o "**ICE3 – created initial project files and README**"

3. CREATE DESIGN

ADD SOURCES (DESIGN, SIMULATION, AND CONSTRAINTS)

Add all of the provided files to your project. Instructions were given in ICE2 to do this. The _tb files will be simulation sources, the .xdc file will be a constraint, and the remaining two files will be design sources.

EDIT YOUR TOP_BASYS3

Open up your top_basys3.vhd file in a text editor.

FILE HEADER
Modify the file header to reflect your work. Do not leave the documentation statement blank. You may refer all documentation statements to top_basys3 header. (ex: your top_basys3_tb file header will say "See top_basys3.vhd" for documentation.

INSPECT THE TOP_BASYS3 ENTITY
Unlike the half-adder, the full-adder has three inputs though it shares the same two outputs. This has been given to you. Note that we went ahead and used a logic vector of size three for the input switches, and a logic vector of size two for the output LEDs. We could have given each input and output its own name if desired.

```
60  □entity top_basys3 is
61  □    port(
62              -- Switches
63              sw      :   in  std_logic_vector(2 downto 0);
64
65              -- LEDs
66              led     :   out std_logic_vector(1 downto 0)
67          );
68  └end top_basys3;
69
```

Figure 2– Entity for top_basys3 implementing a full-adder

MODIFY THE HALF-ADDER ARCHITECTURE

You need to modify the architecture of your top_basys3 file to reflect the full adder shown below in Figure 3.
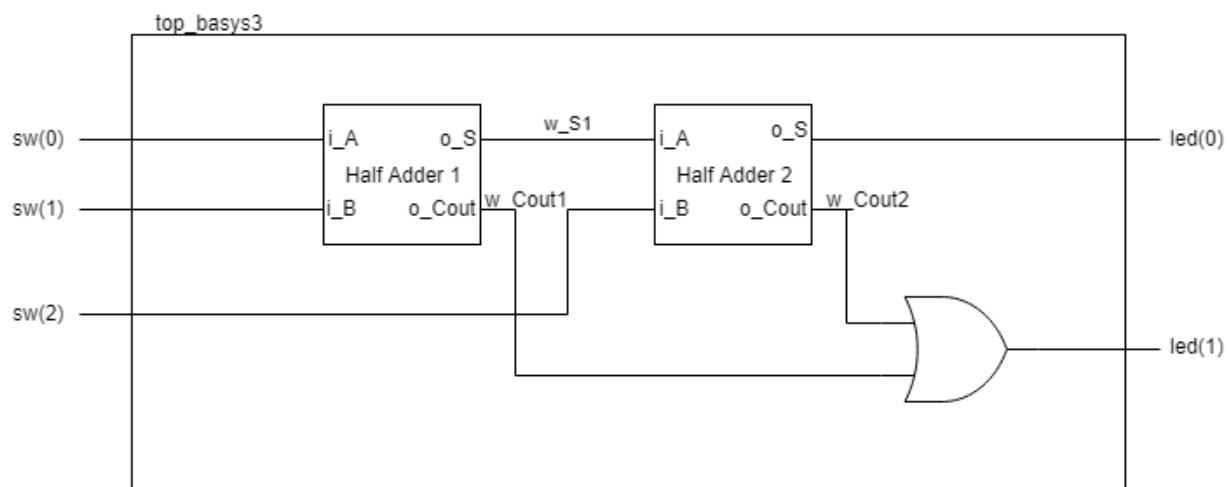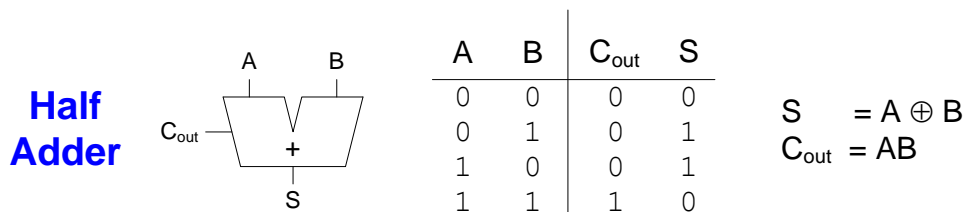

Figure 3 – Entity and architecture for implementing full adder with two half-adders

If you remember from the half adder

This means w_S1 = A xor B. It follows that placing w_S1 into the second half adder as i_A and Cin as i_B would get us o_S= A xor B xor Cin matching the equation in figure 1. Likewise the o_Cout of half adder 1 will be w_Cout1 = AB. W_Cout2 then becomes w_Cout2=Cin(A xor B). Thus led(1)=AB + Cin(A xor B) which simplifies down to led(1)= AB +ACin + BCin.

The first step is to add the half adder component and any signals you will need to implement your design. These declarations go in between the architecture and begin statements as shown in Figure 4 below. See the test bench you developed for ICE2 if you are confused how to do this. The syntax will be identical to the half adder entity, but

will instead start with "component" (see figure 4). Just like the test bench pulls in your component to simulate inputs and outputs for it, a top level design will bring in components and wire them together. If you aren't sure what signals you may need to create, the general rule is to create a signal for any wire not connected directly to an input or an output. See Figure 3 above for some ideas on what to create.

```vhdl
architecture top_basys3_arch of top_basys3 is

    -- declare the component of your top-level design
    component halfAdder is
        port(
            i_A     : in  std_logic; -- 1-bit input port
            i_B     : in  std_logic;
            o_S     : out std_logic;  -- 1-bit output port
            o_Cout  : out std_logic
            );
    end component halfAdder;

    -- declare any signals you will need
    signal |
```

Figure 4 – Component and signal declarations

You will then need to create two instantiations of the half adder wire them up, and add the rest of the logic you need to fully implement the design in Figure 3. Notice that we have TWO instantiations of halfAdder and we have given them unique names (halfadder1_inst and halfAdder2_inst).

```vhdl
-- PORT MAPS ------------------
halfadder1_inst: halfAdder
port map(
i_A => sw(0),
i_B => sw(1),
o_S => w_S1,
o_Cout  => w_Cout1
);

halfAdder2_inst: halfAdder
port map(
i_A =>
i_B =>
o_S =>
o_Cout=>
);

--Combinational Logic
led(1) <= w_Cout2 or w_Cout1;
```

Figure 5 – Half adder instantiations

3. TEST DESIGN

At this point your testbench VHDL file (top_basys3_tb.vhd) should already be added. If not, go back and add it.

EDIT YOUR TESTBENCH

Open your testbench file in a text editor. Note the header has top_basys3.vhd as a REQUIRED FILE. This exercise will not walk you through the details of creating a test bench. If you are unsure, revisit ICE2 for more detailed instructions.

You will need to declare your top_basys3 "component" in the test bench, create signals to simulate the inputs and outputs, and connect them in a port map. Again, revisit ICE2 (or the included half_adder_tb) if you struggle with this.

You will then need to create your test cases within the test process. Figure 6 below includes an example for the first line. Note that we were able to simulate all three bits of the input with a single command. Create all additional test cases you will need. How many test cases will you need?

```
97        begin
98
99            i_sw <= "000"; wait for 10 ns;
```
Figure 6 – Example test case

CHECK SYNTAX AND SIMULATE YOUR PROJECT

4. IMPLEMENT DESIGN

EDIT THE CONSTRAINTS FILE

1. Click on the Project Manager in the Flow Navigator
2. Double click on the Basys3_Master.xdc file in the Sources sub-window to open it.
3. Get your BASYS 3 board out and look at the text surrounding the switches and LEDs. Note, the labels underneath the switches are the physical pin locations on your BASYS 3 board. For example, SW0 is connected to pin V17. Find (use CTRL+f) **V17** in the constraints file. You should see it first on line 12 as shown below.

```
11 : ## Switches
12 : #set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
13 :     #set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
```

4. Unlike our pervious exercise, you will not need to replace the highlighted portions. We made our entity inputs and outputs reflect the default constraint file naming convention.
5. Uncomment all lines needed for this design (sw{0}, sw{1}, sw{2}, led{0}, and led{1}.

IMPLEMENT AND GENERATE BITSTREAM

5. DOWNLOAD ONTO FPGA AND TEST FUNCTIONALITY

IF you are unsure how to proceed, revisit ICE2

7. WRAPPING UP...

- Double check that you have everything **committed** to your git repo.
- Then **push** them to your Bitbucket repo.
- Make sure all your work appears in Bitbucket as you expect.
- Show your simulation waveform to an instructor
- Demo your working board to an instructor
- **Congratulations, you have completed In Class Exercise 3!**