UNITED STATES
AIR FORCE
ACADEMY

ECE 281

Lesson 12 Notes

Objectives:

- Demonstrate the ability to use basic VHDL constructs such as an **entity, architecture, and signals** to model combinational circuits using VHDL behavioral modelling
- Given a VHDL behavioral entity model, draw a schematic that shows the complete entity and architecture

Terminology Review:

VHDL – VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

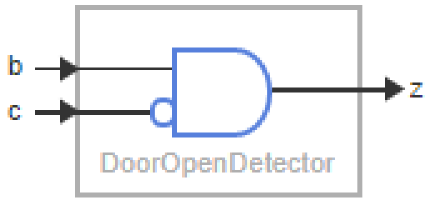
Entity – Defines a new component

Entity Declaration – Specifies the entity's external interface

Architecture Body – specifies the entity's behavior or structure

From zyBooks 7.2

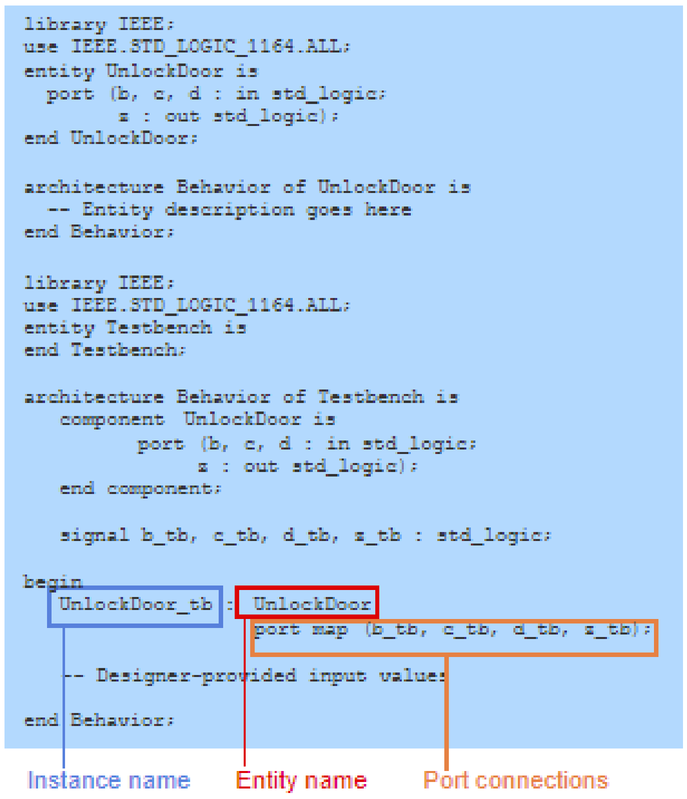
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity DoorOpenDetector is
   port (b, c : in std_logic;
          z : out std_logic);
end DoorOpenDetector;

architecture Behavior of DoorOpenDetector is
begin
   process(b, c) begin
      z <= b and not c;
   end process;
end Behavior;
```



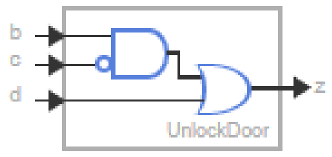
Testbench

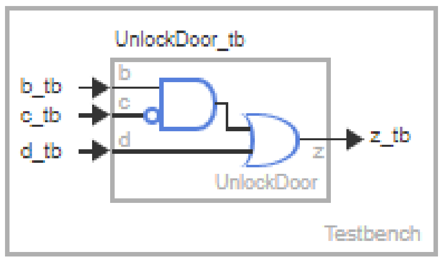
The test bench provides a sequence of input values to test an entity. A testbench is itself an entity with no inputs or outputs, and consists of two main elements:

1. A component declaration and instantiation that creates an instance of the entity being tested
2. A process for generating the sequence of input values to test the entity



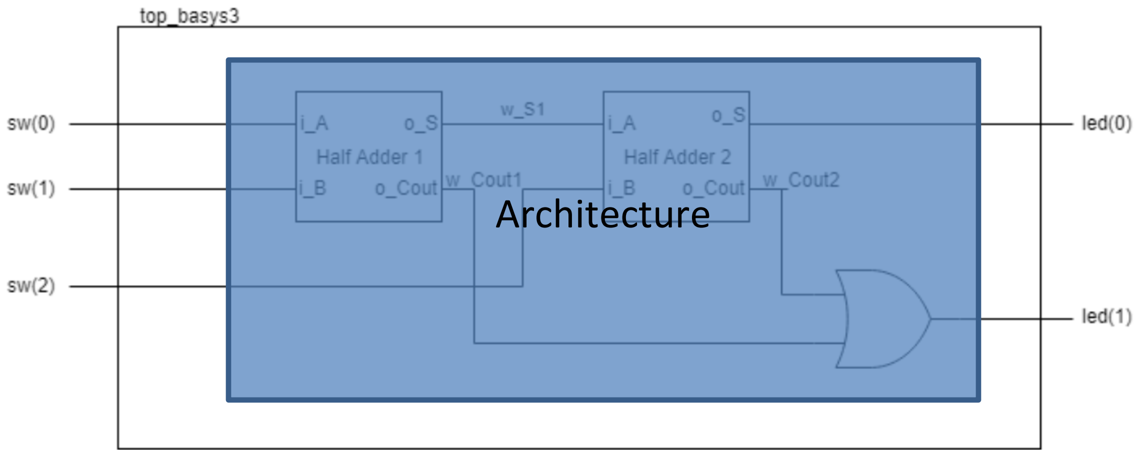
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity UnlockDoor is
  port (b, c, d : in std_logic;
        z : out std_logic);
end UnlockDoor;

architecture Behavior of UnlockDoor is
  -- Entity description goes here
end Behavior;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Testbench is
end Testbench;

architecture Behavior of Testbench is
   component UnlockDoor is
         port (b, c, d : in std_logic;
              z : out std_logic);
   end component;

   signal b_tb, c_tb, d_tb, z_tb : std_logic;

begin
   UnlockDoor_tb : UnlockDoor
                   port map (b_tb, c_tb, d_tb, z_tb);

   -- Designer-provided input values

end Behavior;
```



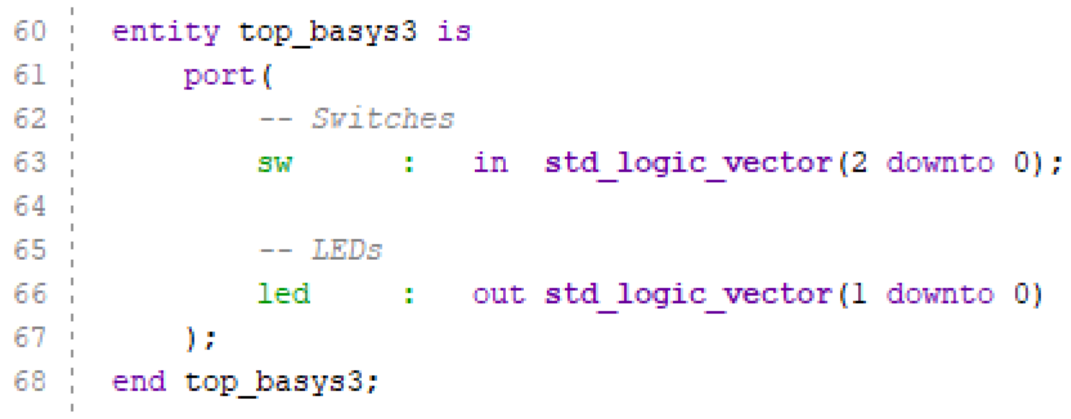


Previous Terminology Applied to ICE3 – The ICE3 handout walks you though each of the major steps to create a full-adder using your previously created half-adder (ICE2). This document will attempt to provide a bit more background on exactly what is required in conjunction with your prior lesson materials.

Entity: declaration of a hardware module's inputs and outputs. In this case you are creating an entity called *top_basys3:*



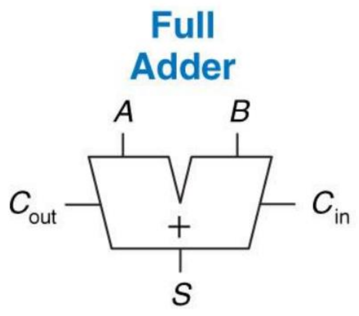
The entity is merely defining the physical interface (i.e. inputs and outputs) to your custom designed architecture. In the case of the full adder, we wish to be able to add three bits together and give the appropriate output. Therefore, we will need three switches defined (inputs) and two LEDs defined (outputs). This is already done for your in the top_basys3.vhd template:



```
entity top_basys3 is
    port(
        -- Switches
        sw      :   in  std_logic_vector(2 downto 0);

        -- LEDs
        led     :   out std_logic_vector(1 downto 0)
    );
end top_basys3;
```

Architecture: the internal logic of the entity

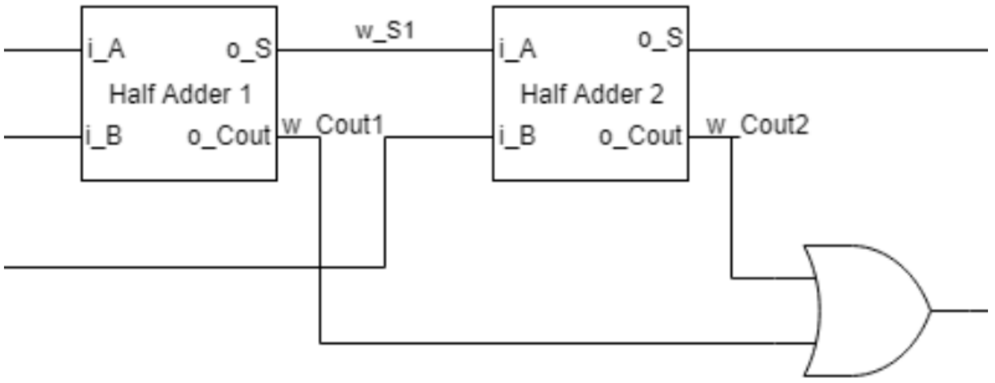
In this case, we are going to implement the logic for the following truth table.



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

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

It would be possible to simply implement the logic equations shown above, but the intent of this In Class Exercise is to instantiate previously designed components. This skill will pay dividends on future Labs or exercises. Therefore, the architecture we are going to implement is as follows:



As you can see above, we essentially have 3 logical pieces (2x Half Adders, and one OR gate). The architecture will have 3 inputs (corresponding to the 3 switches defined in your entity) and 2 outputs (corresponding to the 2 LEDs defined in your entity).

Now your job is to add each of these three pieces in, and also add in the signals required to “wire” everything up. Figure 4 of the ICE, gives you the code to incorporate the previously designed Half-Adder component to your new design. At the bottom of Figure 4, you see the hint that you will need to declare additional signals to wire things up. **Notice you essentially have 3 internal signals required (w_Cout1, w_S1, w_Cout2).**

Now you just need to connect everything up. We will do this with the **PORT MAP.** Whether you recognize it or not, you have been using the PORT map to instantiate components all along in your test bench. Now we are going to do it inside of the architecture of your design.

Figure 5 of the ICE3 walks you through the bulk of the port map, but leaves it for you to finalize the port map for the instantiation of the 2nd Half-Adder. Notice at the bottom of Figure 5 the final step is to connect the Carry-out of each half to an OR gate.

The remainder of this ICE just requires you to complete the design for your Test Bench and identify the mapping of inputs / outputs in your basys3 constraints file.