LAB #2

Explain VHDL programming concepts to design combinational circuit using Altera Quartus tool and implementation on FPGA

Objective

- To understand different architectures in VHDL language
- To understand the designing of combinational circuit
- To understand the implementation of a truth table

Pre-Lab

ARCHITECTURE contains a description of how the circuit should function, from which the actual circuit is inferred. A simplified syntax is shown below.

```
ARCHITECTURE architecture_name OF entity_name IS
  [architecture_declarative_part]
BEGIN
  architecture_statements_part
END [ARCHITECTURE] [architecture_name];
```

The architecture body defines, the internal working, or behavior, of the entity. We can choose between one of three "ways" to describe the architecture body:

i. Structural (see Lab 1)
ii. Dataflow
iii. Behavioral descriptions.

The main difference between these approaches is the level of detail required (or, conversely, the level of abstraction allowed).

Consider an example of a black box,

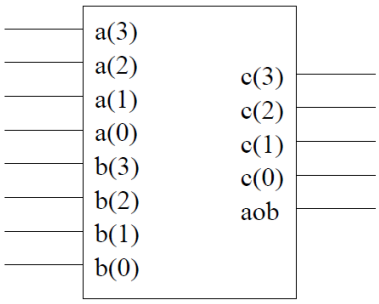

Figure 1 Black Box Description of the circuit

The entity declaration for the black box is

```
entity bbox is port(
     a,b : in std_logic_vector(3 downto 0);
     c   : out std_logic_vector(3 downto 0);
     aob : out std_logic);
end bbox;
```

1. Behavioral Description

Consider the architecture body shown in Figure 1. This behavior description is quite reminiscent of a C-language programmed in many ways. Behavioral descriptions are high-level, just as C is a high-level language. The architectural description is bounded by the architecture and end **arch_bbox** statements. When declaring the architecture **arch_bbox.** We define which entity the architecture belongs to (of bbox is). The **process** statement is used to enclose an algorithm.

The process in Figure 2 is named **comp** and the sensitivity list of **comp** is declared as (a,b). The sensitivity list identifies the signals that will cause the process to execute. In this case, the circuit is sensitive to changes in the two input signals, **a** and **b**. This means that whenever a or b changes, the **comp** process changes

```
architecture arch_bbox of bbox is
begin
  comp: process (a,b) begin
    c <= b;
    if a = b then
        aob <= '1';
     else
       aob <= '0';
     end if
  end process comp;
end arch_bbox;
```

Figure 2 Behavioral Description of Black Box Circuit

```
architecture arch_bbox of bbox is
begin
  comp: process (a,b) begin
    c <= b;
    aob <= '0';
    if a = b then
        aob <= '1';
    end if
  end process comp;
end arch_bbox;
```

Figure 3 An Alternate Behavioral Description of Black Box Circuit

2. DataFlow Description

A dataflow description is very similar to a behavioral description. In fact, the two are often both referred to as behavioral. The main difference is that a dataflow description does not use the **process** construct. Clearly the dataflow description is easy to understand for a simple example, such as the one we are looking at. With a more complicated algorithm is required, such as one with nested sequential statements, a behavioral description will probably make more sense.

```
architecture dataflow of bbox is
begin
   aob <= '1' when (a=b) else '0';
end dataflow;
```

Figure 4: Dataflow Description of Black Box

The big difference between behavioral and dataflow can be seen when we consider a circuit where the inputs may change at any time, but where we only want these (possibly changed) inputs to be noticed when a clock pulse triggers the circuit. We can easily describe this using a behavioral description where **process(clk)** identifies the clock signal as causing the circuit to activate. With a dataflow description, we cannot as easily or neatly control "when" the circuit activates.

3. Combinational logic

Combinational logic can be written with both concurrent and. sequential statements. Concurrent statements may be executed in parallel (concurrently) and are found in dataflow and structural descriptions of a circuit. Sequential statements must be executed in a given sequential order and are used in behavioral descriptions (hint: what is the big difference between behavioral and dataflow descriptions?)

3.1 Concurrent Statements

Concurrent statements fall outside of the process statement (and hence fit nicely with dataflow descriptions).

3.1.1 Boolean Statements

The most "obvious" of concurrent statements are Boolean statements. As an example, suppose we wish to build a circuit that will produce the logical-and and logical-or of two bits. We can accomplish this using Boolean statements, as shown in Figure 5.

```
library ieee;
use ieee.std_logic_1164.all;
entity cct1 is port(
   a,b      : in std_logic;
   land,lor : out std_logic);
end cct1;
architecture archcct1 of cct1 is
begin
   land <= a and b;
   lor  <= a or b;
end archcct1;
```

Figure 5: Concurrent Boolean Statement Circuit

The output of this circuit is the two signals, land and lor, produced concurrently (simultaneously).

The Boolean statements that are available in the **ieee 1164** library are: **and**, **or**, **nand**. **not**, **xor** and **xnor**

These data types can be used with bit and Boolean variables (std logic) and with one-dimensional arrays of bits and Boolean variables (such as std logic vector(3 downto 0)), where both variables have the same length.

If you have an equation with multiple Boolean operations, you must use parenthesis to force VHDL into order of operations (otherwise you will get a compile-time error).

3.1.2 With-Select-When Statements

There may be cases where a signal value is assigned based on the value of another signal (a selection signal). In this case, the **with-select-when** statements come in handy.

For example, consider a circuit where the output, **z**. will be assigned the value of signals **a** or **b**, depending on the value of a selection signal. **s**. We can represent this in VHDL as shown in Figure 6.

```
ARCHITECTURE cct OF testcct IS
BEGIN
with s select
  z <= a when '0',
       b when '1';
END cct;
```

Figure 6: Code Fragment: With-Select-When Statements

3.1.3 With-Select-When-When Others

Because of how std logic is defined, a single bit does not necessarily have only two values (unless it is explicitly Boolean). Other possible values include high impedance, unspecified,

low impedance, and so on. For this reason, we have the choice of specifying the case when others as a catch-all for all other, not already specified, values of the selection signal. This idea is similar to the use of default in a C-language case statement. Figure 7 shows the when-others "equivalent" of Figure 6,

```
ARCHITECTURE cct OF testcct IS
BEGIN
with s select
  z <= a when '0',
       b when '1',
       '0' when others;
```

Figure 7: Code: With-Select-When-When Others Statements

The code in Figure 7 states that for any value of **s** other than "0", the signal **z** will have the same value as the signal **b**.

3.1.4 When-Else Statements

The **when-else** statements are a version of the when-select statements where assignment is based on a condition that may or may not revolve around a single signal. The condition evaluated in this type of statement may be based on a single signal, like the when-else statements. or on multiple signals, or multiple conditions involving different signals. For this reason, there is an order of preference within a when-else statement; once a successful, or true, condition is encountered, the assignment specified by the when-else statement is executed and the entire clause is "exited".

For example, the **when-else** equivalent of the code of Figures 6 and 7 is shown in Figure 8.

```
ARCHITECTURE cct OF testcct IS
BEGIN
  z <= a when (s='0') else
       b;
END cct;
```

Figure 8: Code Fragment: When-Else Statements

Suppose we only want **z** to be assigned the value of **a** or **b** based on the select signal **s** and some other condition. We can create compound conditional statements within a when-else statement. All that we have to do enclose the compound statement in a set of parentheses, to "create" a simple statement, as seen in Figure 9.

```
ARCHITECTURE cct OF testcct IS
BEGIN
  z <= a when (s='0' and ocond1=true) else
       b when (s='1' and ocond2=true) else
       z;
END cct;
```

Figure 9: Code Fragment: When-Else Statements

Pre-Lab Task

Task 1: Implementation of Truth Table: Step-by-Step Practice

Let's start by creating a truth table which is the basic information for any combinatorial logic circuit. We will choose a 2-to-4 binary decoder, which would be described by the following table.

Input		Output			
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

A binary decoder is a circuit that simply translates binary to a position of bit information. Using the syntax given at the intro, this table can be coded as below, assuming input as a two-bit vector and the output as a four-bit vector:

```
output <= "0011" when (input = "00") else
          "0110" when (input = "01") else
          "1100" when (input = "10") else
          "1001";
```

Next, we are going to use this code to implement the **2-to-4 binary decoder**.
Please follow the steps provided in lab # 1
You should see the code window that contains the basic template of a VHDL module which should look like below, excluding the comments.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity dec2to4 is
end dec2to4;

architecture Behavioral of dec2to4 is

begin

end Behavioral;
```

Next, complete the entity and architecture code as shown below.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dec2to4 is
  port ( input :in std_logic_vector(1 downto 0);
         output :out std_logic_vector(3 downto 0));
end dec2to4;

architecture Behavioral of dec2to4 is
begin
  output <= "0011" when (input = "00") else
            "0110" when (input = "01") else
            "1100" when (input = "10") else
            "1001";

end Behavioral;
```

Assign the pins as seen in the table below.

Input		**Output**			
Input (1)	Input (0)	Output (3)	Output (2)	Output (1)	Output (0)
SW1	**SW0**	**LED3**	**LED2**	**LED1**	**LED0**
PIN_AC28	PIN_AB28	PIN_F21	PIN_E19	PIN_F19	PIN_G19

In-Lab Tasks

When you test the implementation using slide switches as the input like **2-to-4 binary decoder** example, always gingerly move the switch handles. Because it is a mechanical switch, it generates bouncing signals (noise) when you move the switch from on-to-off or vice versa, often giving errors or damaging input ports. The best contact condition of slide switches is achieved when you apply just a right amount of force; if you push too hard, you will get bad contact conditions.

Lab Task 1: Implement the 3-to-8 decoder

Complete the truth table of 3-to-8 decoder and show its results on the FPGA

input	**output**
000	00000001
001	00000010
010	00000100
011	
100	
101	
110	
111	

Lab Task 2: Seven Segment Display (SSD)

The DE2-115 Board has eight 7-segment displays. These displays are arranged into two pairs and a group of four, behaving the intent of displaying numbers of various sizes. As indicated in the schematic in Figure 10, the seven segments (common anode) are connected to pins on Cyclone IV E FPGA. Applying a low logic level to a segment will light it up and applying a high logic level turns it off.
Each segment in a display is identified by an index from 0 to 6, with the positions given in Figure 10.

Figure

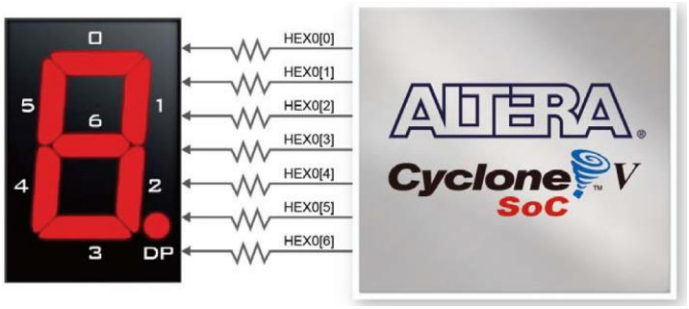

Figure 10: Connections between the 7-segment display HEX0 and Cyclone IV E FPGA

Next you are going to write a VHDL code to control the SSD. Please follow the steps provided below.

Step 1: Create a new project

Step 2: Using a New Source wizard, create a VHDL module

Step 3: Write the entity

```
library ieee;
use ieee.std_logic_1164.all;
entity seven is
port(
        cathodes: out std_logic_vector(6 downto 0);
        bininput: in std_logic_vector(3 downto 0)
);
end seven;
```

The port "**cathodes**" is a seven-bit vector that would be connected to the cathodes of LED segments. The port **binInput** provides binary input patterns and connected to the four slide switches.

Step 4: Complete the binary to SSD conversion table and finish the decoder that can display all 4bit binaries to SSD, i.e., display of patterns, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. and write the architecture code.

bininput	**cathodes**
0000	1000000
0001	1111001
0010	
0011	
0100	
0101	
0110	
0111	
1000	
1001	
1010	
1011	
1100	
1101	
1110	

1111	

Step 5: Now write the dataflow architecture for the truth table generated in step 4,

```
architecture dataflow of seven is
begin
        cat <= "1000000"    when (bininput ="0000") else
                "1111001"   when (bininput ="0001") else
                "0100100"   when (bininput ="0010") else
                "0110000"   when (bininput ="0011") else
```

*Hint: Write the remaining patterns for **bininput** variable*

Step 6: Assign the pins as seen in the table below

Pin Assignments for bininput	
SW0	PIN_AB28
SW1	PIN_AC28
SW2	PIN_AC27
SW3	PIN_AD27

Pin Assignments for 7-segment Displays	
HEX0[0]	PIN_G18
HEX0[1]	PIN_F22
HEX0[2]	PIN_E17
HEX0[3]	PIN_L26
HEX0[4]	PIN_L25
HEX0[5]	PIN_J22
HEX0[6]	PIN_H22

Step 7: Generate the bit file, download it to the board, and test the circuit. Remember that by changing the **bininput** slide switches, the value of 7-segment will change.

Challenge:

a) Can you put the same digit on another 7-segment? How
b) Lets say your switches are set to **ON OFF OFF ON** , then digit "9" should be displayed on one of the 7-segment, now you want to put digit "8" (one less than the digit on 7-segment) on another 7-segment. What changes do you need to do in your code in order to achieve this?

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date:_____