An-Najah National University
Faculty of Engineering and IT

جامعة النجاح الوطنية
كلية الهندسة وتكنولوجيا المعلومات

Computer Engineering Department

Course Name: Digital design Lab 2 **Number: 10636391**

Lab Report Grading Sheet

Instructor: Dr Ashraf Armoush	Experiment #: 6
Academic Year: 2020/2021	Experiment Name: IP core
Semester: Summer	

Students	
1-Ashraf Habromman	2-Ra'ed Khwayerh
Performed on: 15/7/2021	Submitted on: 15/7/2021

Report's Outcomes

ILO __ =( ) %	ILO __ =( ) %	ILO __ =( ) %	ILO __ =( ) %	ILO __ =( ) %

Evaluation Criterion	Grade	Points
Abstract answers of the questions: "What did you do? How did you do it? What ?"did you find	0.5	
Introduction and Theory Sufficient, clear and complete statement of objectives. In addition to Presents sufficiently the theoretical basis.	1.5	
Apparatus/ Procedure Apparatus sufficiently described to enable another experimenter to identify the equipment needed to conduct the experiment.Procedure .sufficiently described	2	
(Experimental Results and Discussion (In-Lab Worksheet Crisp explanation of experimental results. Comparison of theoretical predictions to experimental results, including discussion of accuracy .and error analysis in some cases	4	
Conclusions and Recommendations Conclusions summarize the major findings from the experimental results with adequate specificity. Recommendations appropriate in .light of conclusions. Correct grammar	1	
Appearance Title page is complete, page numbers applied, content is well organized, correct spelling, fonts are consistent, good visual appeal.	1	
Total	10	

IP cores (Multiplier)

Introduction:

As we know, in the previous experiment we build an ASM chart to implement multiple operations because the software (vivado) can't implement and synthesize * multiple, but in this experiment, we want to implement multiple in a different way.

So we used IP core. An IP (Intellectual Property) core is a block of HDL code that other engineers have already written to perform a specific function. As with any engineering tool, IP cores have their advantages and disadvantages. Although they may simplify a given design, the engineer has to design the interfaces to send and receive data from this "black box".

Abstract:

We'll learn how to implement multiple without building an asm chart, by used IP core that let us use a black box that implements multiple and we'll instantiate it in our block then we'll send data for it then we check the results, we'll first simulate it, then synthesize and generate it in our kit to check results.

Apparatus:

1-Vivado Design Suite HL WebPACK™ Edition.
2-ZedBoard.

Procedure:

1. In Vivado, create a new project.
2. From the Flow Navigator, under PROJECT MANAGER, select IP Catalog. The Xilinx IP Catalog displays in a new tab. The IP catalog contains categories of IP that you can filter and search.
3. You can work with the IP Catalog in a variety of ways. You can search using keywords in the search box or browse through the catalog in the various categories.
4. Type Multiplier in the search box. The search results narrow the list of IP definitions displayed in the catalog.
5. From the Math Functions > Multipliers group select Multiplier, as shown in the figure
6. Right-click to open the popup menu, and select Customize IP, or double-click on the selected IP.
7. The Multiplier customizes window opens.
8. Select the options to match those shown in the figure:
• Component Name: mult_gen
• Multiplier Type: Parallel Multiplier
• Data Type: Unsigned
• Width: 4 9. Then click OK to add the IP customization to your design.

10. The Generate Output Product dialogue box opens as shown in the next figure.
11. Click Generate to generate the required output products.
12. The mult_gen now appears in the Sources view.
13. In the IP Sources tab of the Sources window, you can examine the output products produced for the Multiplier Generator customization.
14- From the Project Manager click Add Source and instantiate mult_gen

Using “mult_gen.vho” file to implement the top-level entity:

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

entity Main is
    Port ( clk : in STD_LOGIC;
           A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           P : out STD_LOGIC_VECTOR (7 downto 0));
end Main;

architecture Behavioral of Main is

COMPONENT mult_gen
  PORT (
    CLK : IN STD_LOGIC;
    A : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    B : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    P : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END COMPONENT;
begin
mult1 : mult_gen
  PORT MAP (
    CLK => CLK,
    A => A,
    B => B,
    P => P
  );

end Behavioral;
```

Simulate part:

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


entity multiplier_tb is
--  Port ( );
end  multiplier_tb;

architecture Behavioral of multiplier_tb is
component Main
    Port ( clk : in STD_LOGIC;
           A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           P : out STD_LOGIC_VECTOR (7 downto 0));
end component;

signal  CLK : std_logic:='0';
signal A, B : STD_LOGIC_VECTOR (3 downto 0);
signal P : STD_LOGIC_VECTOR (7 downto 0);
constant period : time := 10 ns;
begin
u1: Main
port map(clk=>clk,
 A=>A,
 B=>B,
 P=>P);

clk_gen: process
begin
clk<='0';
wait for period/2;
clk<='1';
wait for period/2;
end process clk_gen;

stimulus: process
begin

A<="0000";
B<="0000";
```

```
wait for period*20;

        for i in 0 to 15 loop
        for j in 0 to 15 loop
            A<= A + '1';
            wait for period*20 ;
                end loop;
                B<= B + '1';
                wait for period*20;
        end loop;
wait;
end process stimulus;
end Behavioral;
```

Constraint file:

```
set_property IOSTANDARD LVCMOS18 [get_ports {A[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {A[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {A[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {A[0]}]
set_property PACKAGE_PIN F21 [get_ports {A[3]}]
set_property PACKAGE_PIN H22 [get_ports {A[2]}]
set_property PACKAGE_PIN G22 [get_ports {A[1]}]
set_property PACKAGE_PIN F22 [get_ports {A[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {B[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {B[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {B[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {B[0]}]
set_property PACKAGE_PIN M15 [get_ports {B[3]}]
set_property PACKAGE_PIN H17 [get_ports {B[2]}]
set_property PACKAGE_PIN H18 [get_ports {B[1]}]
set_property PACKAGE_PIN H19 [get_ports {B[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN Y9 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports {P[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {P[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {P[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {P[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {P[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {P[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {P[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {P[0]}]
set_property PACKAGE_PIN U14 [get_ports {P[7]}]
set_property PACKAGE_PIN U19 [get_ports {P[6]}]
```

```
set_property PACKAGE_PIN W22 [get_ports {P[5]}]
set_property PACKAGE_PIN V22 [get_ports {P[4]}]
set_property PACKAGE_PIN U21 [get_ports {P[3]}]
set_property PACKAGE_PIN U22 [get_ports {P[2]}]
set_property PACKAGE_PIN T21 [get_ports {P[1]}]
set_property PACKAGE_PIN T22 [get_ports {P[0]}]
```

16- Synthesize and generate the bitstream.

Conclusion:

In the experiment, we learned how to use an IP config that let us use a black box that implements multiple operations. and then simulate it and connect it with ZedBoard to check results.