EXPERIMENT 3

Boolean Laws & Rules and DeMorgan’s Theorem

OBJECTIVES:

- Learn and verify Boolean laws and rules.
- Learn and prove DeMorgan’s theorem
- Use Xilinx simulation tools to test combinational circuits.

MATERIALS:

- Xilinx Vivado software, student or professional edition V2018.2 or higher.
- IBM or compatible computer with Pentium III or higher, 128 M-byte RAM or more, and 8 G-byte Or larger hard drive.
- BASYS 3 Board.

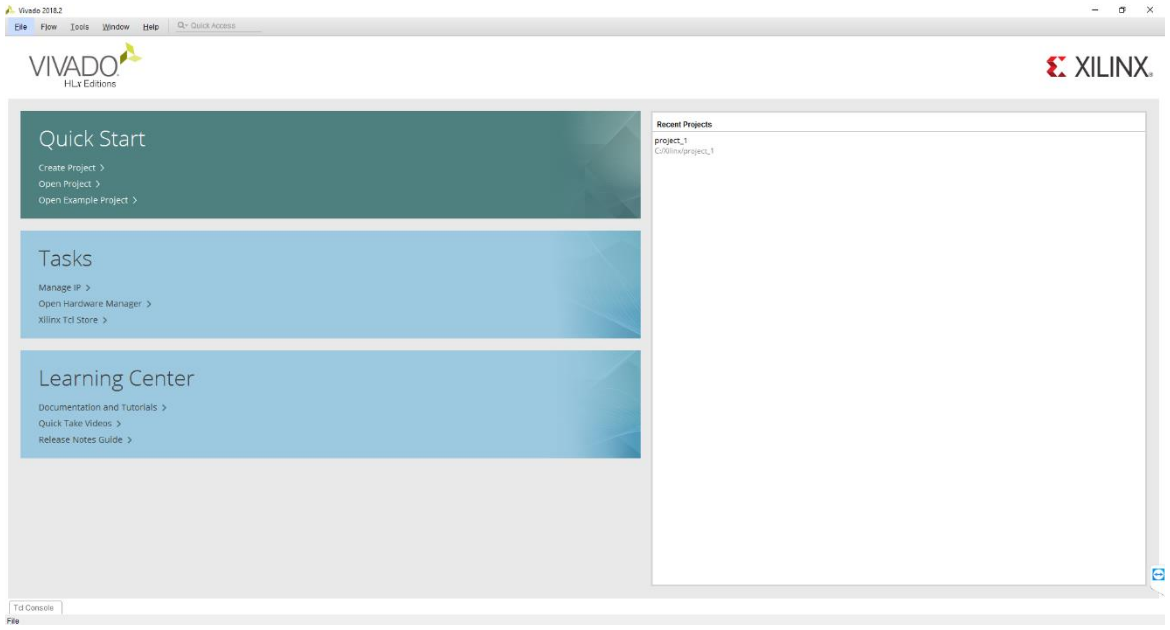
DISCUSSION:

A Boolean equation derived directly from a truth table or from a problem statement usually is not in the simplest form. To have an efficient equivalent logic circuit, the Boolean equation representing the logic design must be in the simplest from. Boolean equations can be simplified using Boolean algebra, DeMorgan’s theorem, or/and Karnaugh maps. In this experiment, we will first present Boolean Laws and rules as well as DeMorgan’s theorem, and then verify them.

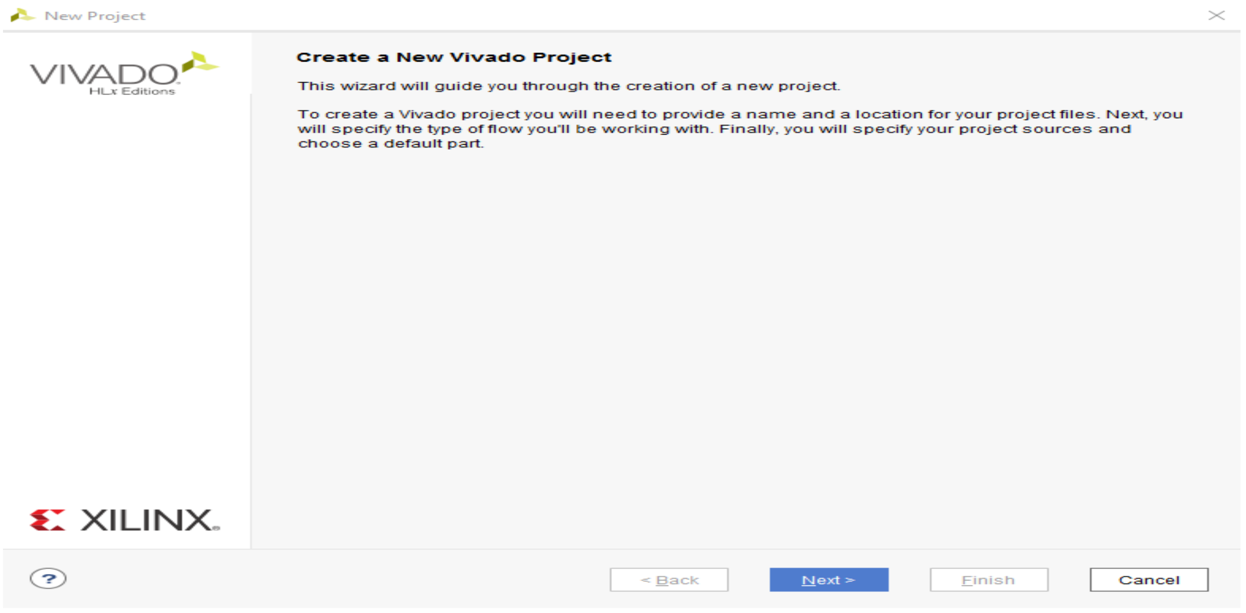
PROCEDURE:

Section I. Boolean Distributive Law

1. Open Xilinix Vivado.



2. In the **Xilinx-Project Navigator** window, Quick start, **New Project**.

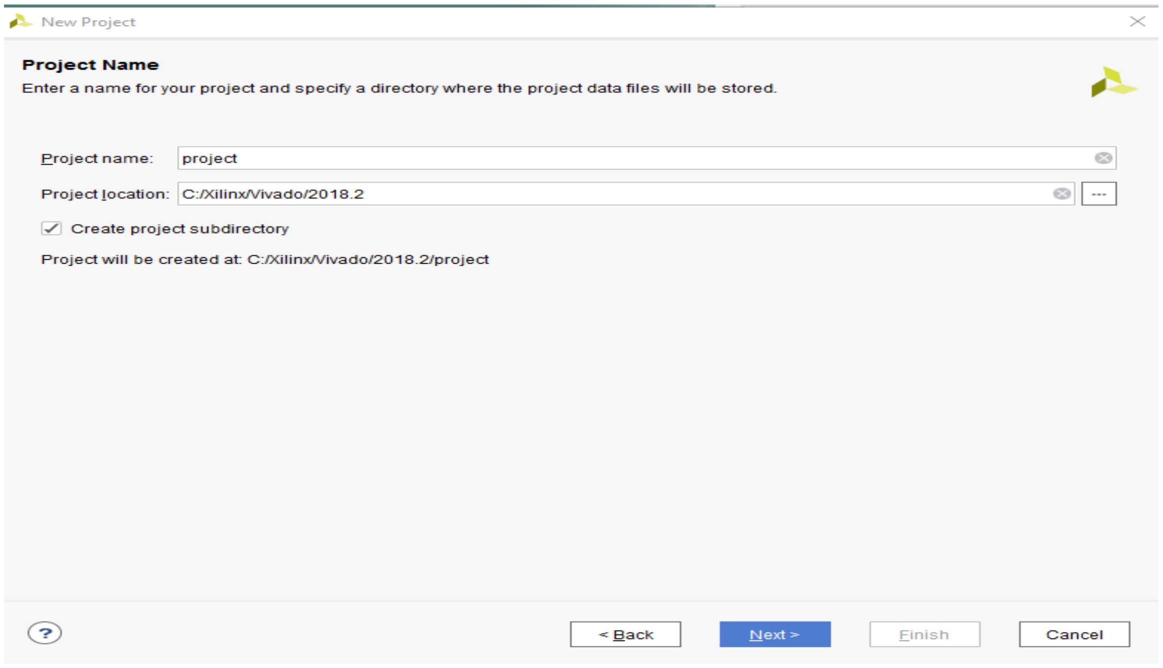


Create a New Vivado Project

This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

3. Name the project.



Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

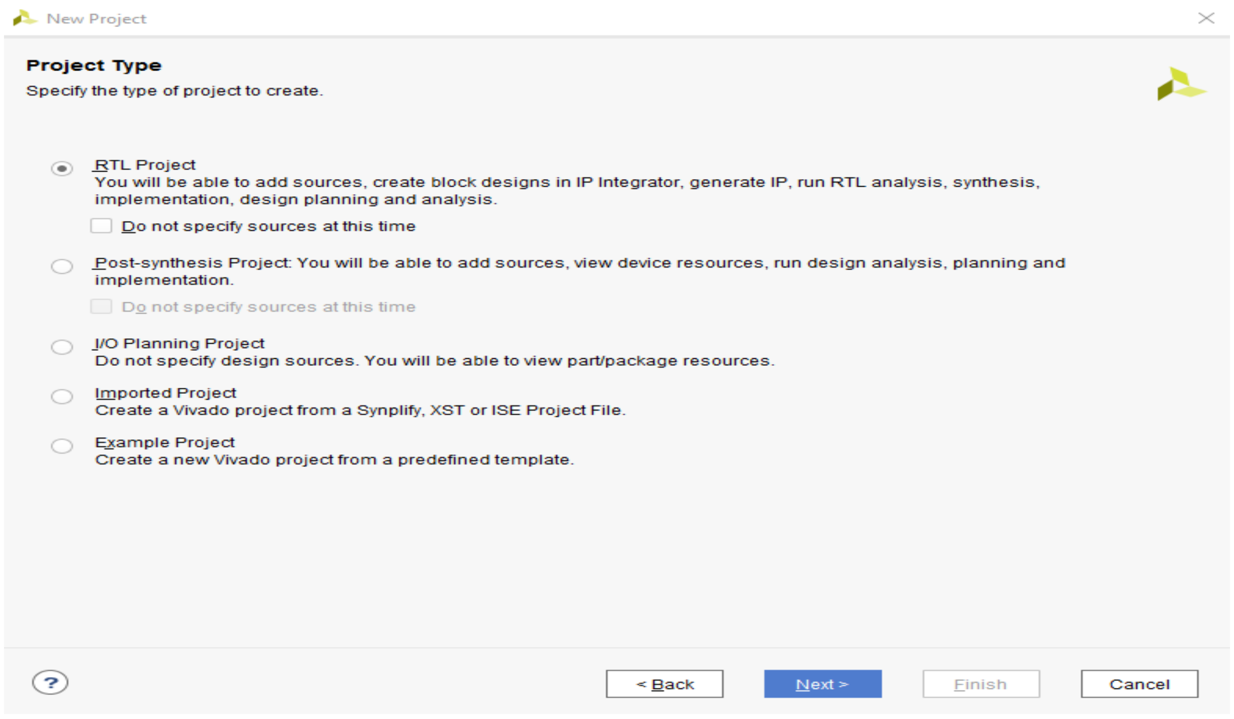
Project name: project

Project location: C:/Xilinx/Vivado/2018.2

[✓] Create project subdirectory

Project will be created at: C:/Xilinx/Vivado/2018.2/project

4. Choose “RTL Project” and check the “Do not specify sources at this time” as we will configure all the settings manually through the navigator from inside the project.



Project Type

Specify the type of project to create.

● RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☐ Do not specify sources at this time

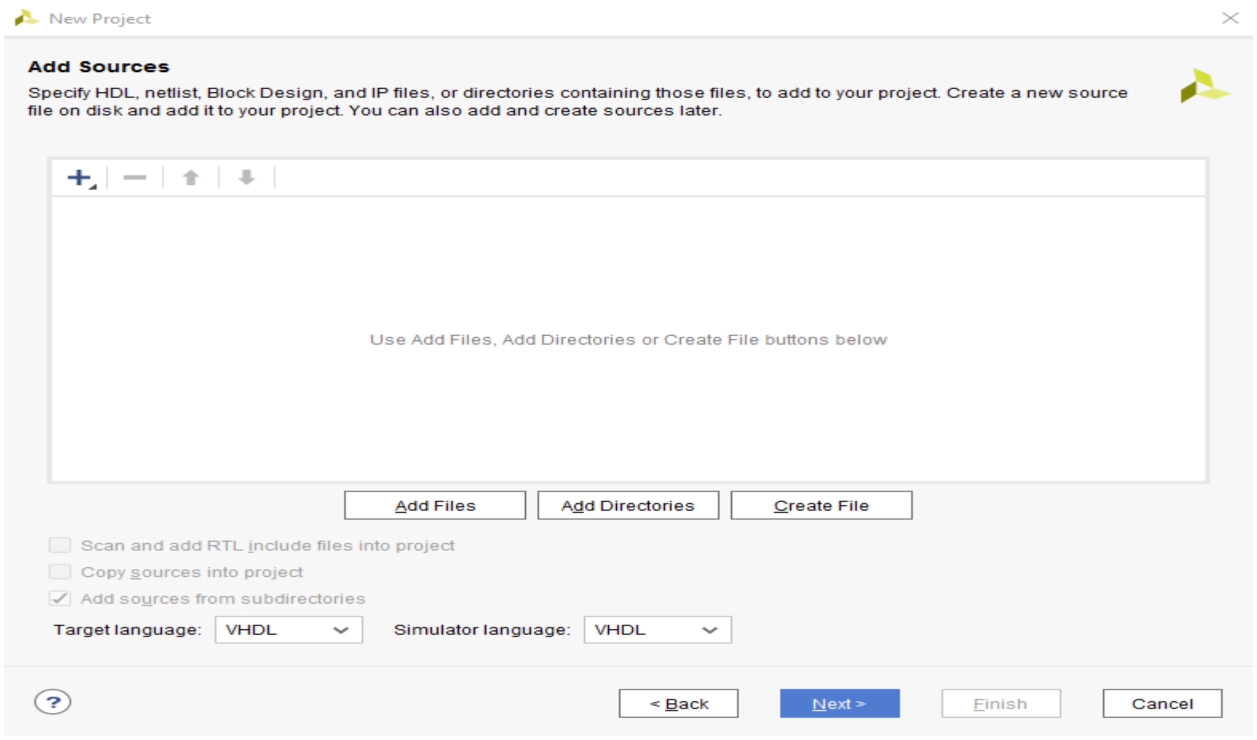
○ Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.
☐ Do not specify sources at this time

○ I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

○ Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

○ Example Project
Create a new Vivado project from a predefined template.

5. Select **New Source…** and the **New** window appears. In the New window, choose Schematic, type your file name (such as ***source_1***) in the File Name editor box, click on OK, and then click on the Next button.


New Project

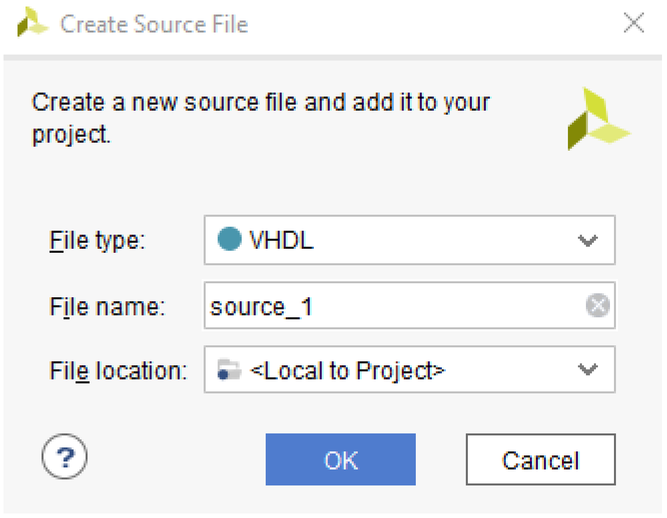
Add Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

Use Add Files, Add Directories or Create File buttons below

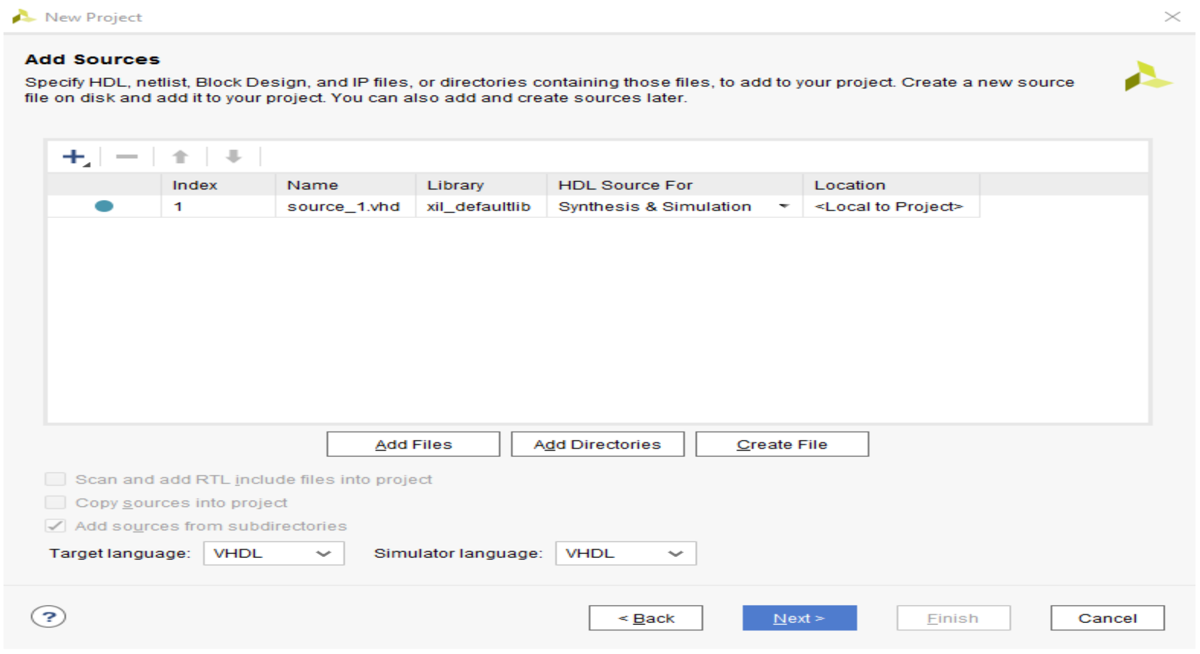
- Scan and add RTL include files into project
- Copy sources into project
- ✓ Add sources from subdirectories

Target language: VHDL
Simulator language: VHDL


Create Source File

Create a new source file and add it to your project.

File type: VHDL
File name: source_1
File location: <Local to Project>


New Project

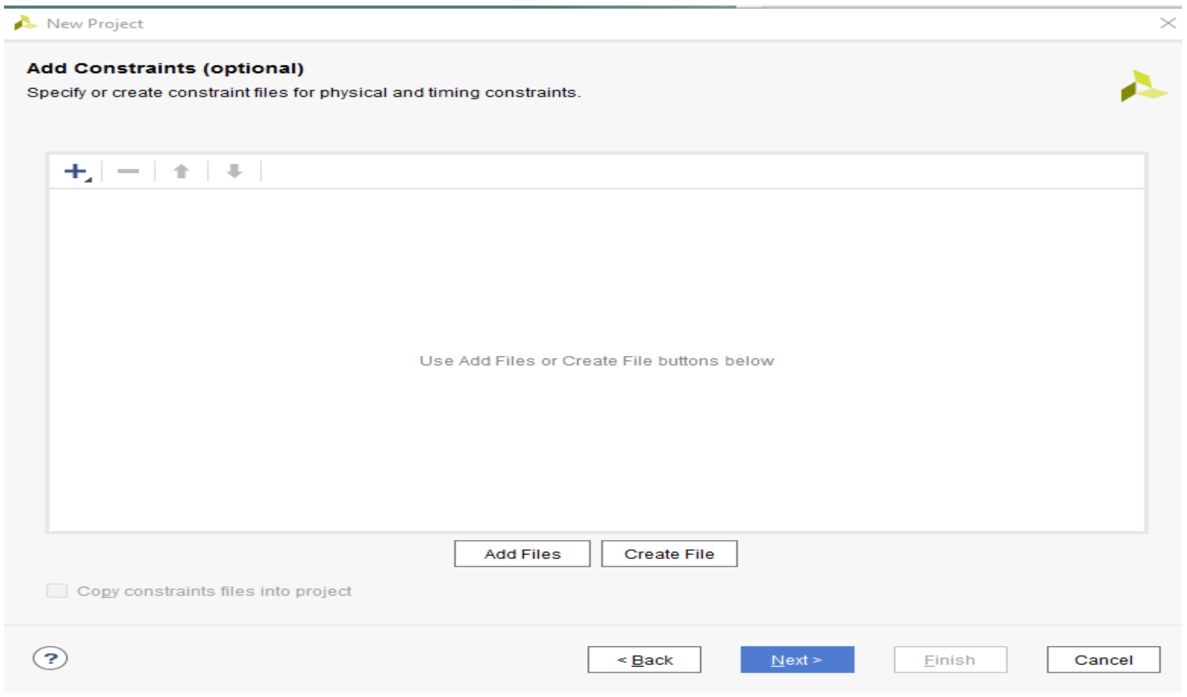
Add Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

	Index	Name	Library	HDL Source For	Location
●	1	source_1.vhd	xil_defaultlib	Synthesis & Simulation	<Local to Project>

- [ ] Scan and add RTL include files into project
- [ ] Copy sources into project
- [✓] Add sources from subdirectories

Target language: VHDL   Simulator language: VHDL


New Project

Add Constraints (optional)

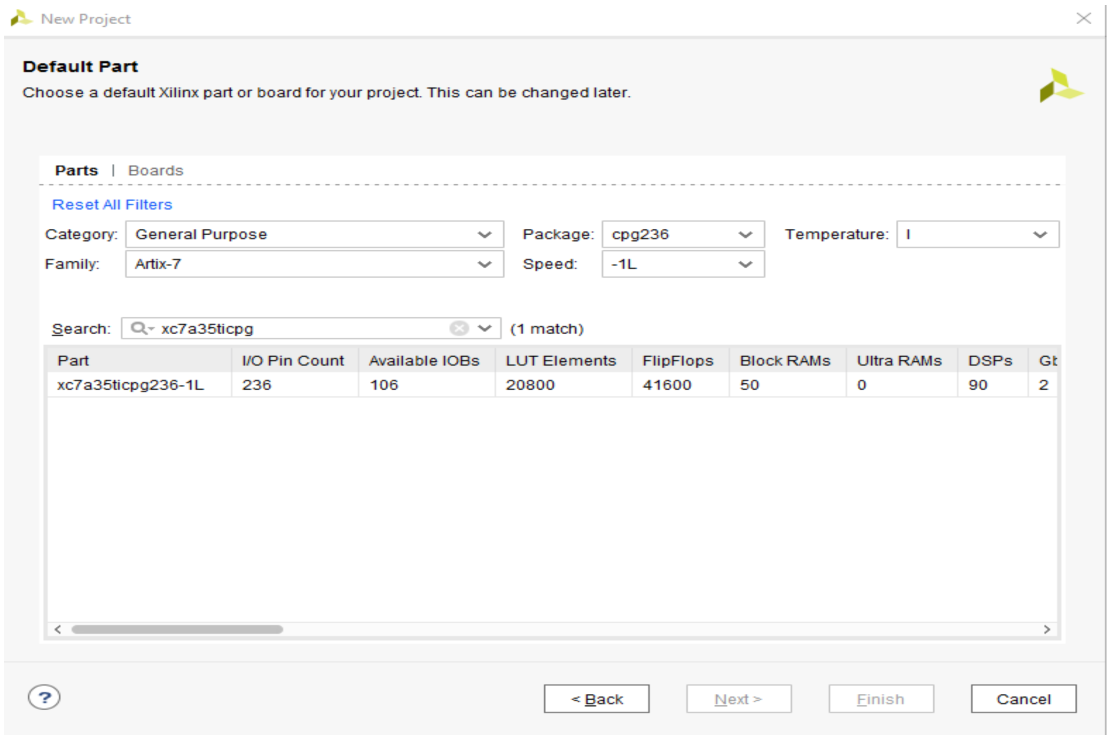
Specify or create constraint files for physical and timing constraints.

Use Add Files or Create File buttons below

- [ ] Copy constraints files into project

6. In the **Xilinx - Project Navigator** window, select the following
   - Category: “General Purpose”
   - Family: “Artix-7”
   - Package: “cpg236”
   - Speed: “-1”
   - Choose “xc7a35tcpg236-1” that corresponds to the board we are using.

Then Choose Finish.



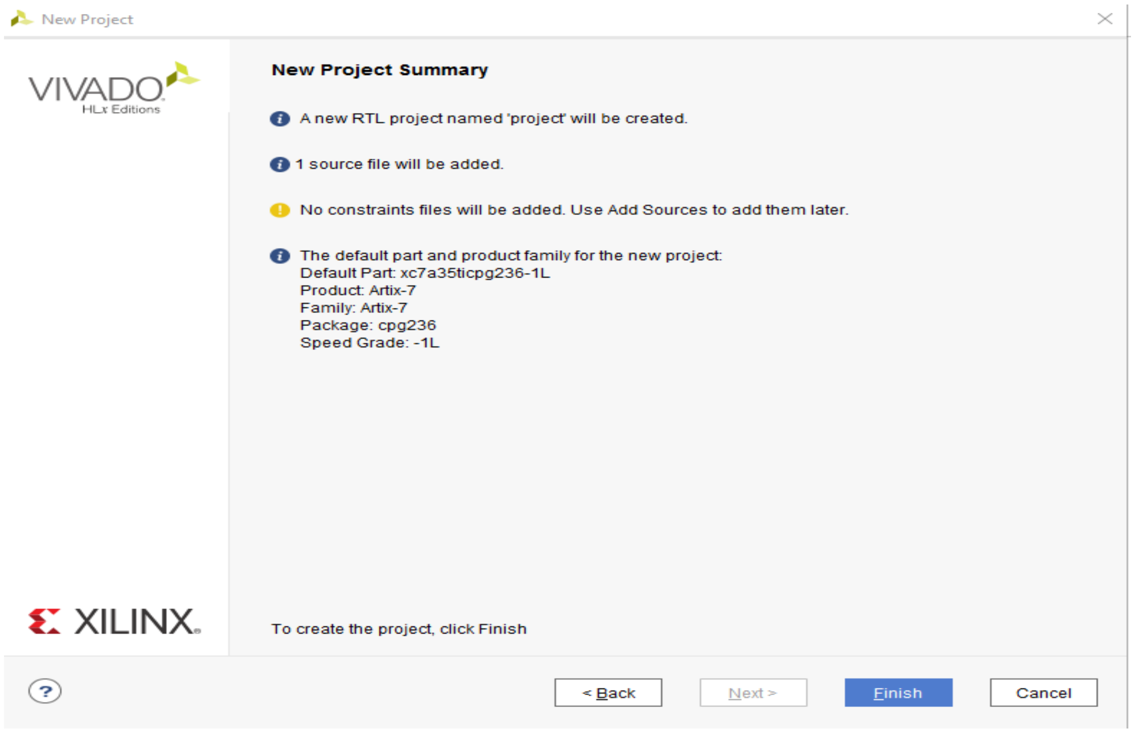
Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Category: General Purpose
Package: cpg236
Temperature: I
Family: Artix-7
Speed: -1L

Search: xc7a35ticpg (1 match)

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gt
xc7a35ticpg236-1L	236	106	20800	41600	50	0	90	2



New Project Summary

A new RTL project named 'project' will be created.

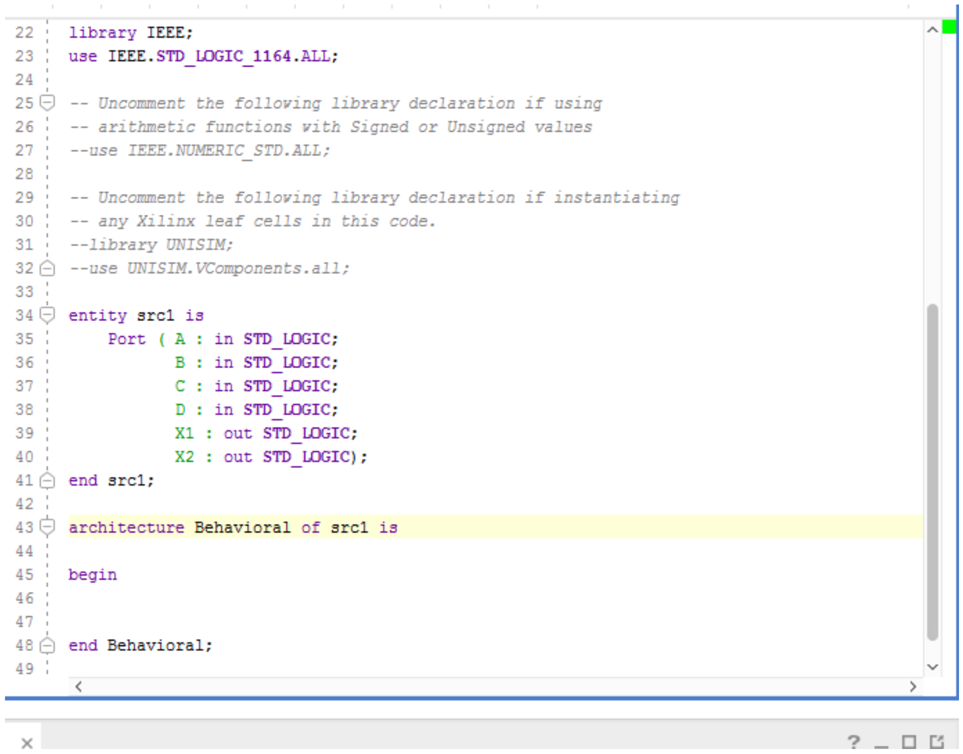
1 source file will be added.

No constraints files will be added. Use Add Sources to add them later.

The default part and product family for the new project:
Default Part: xc7a35ticpg236-1L
Product: Artix-7
Family: Artix-7
Package: cpg236
Speed Grade: -1L

To create the project, click Finish

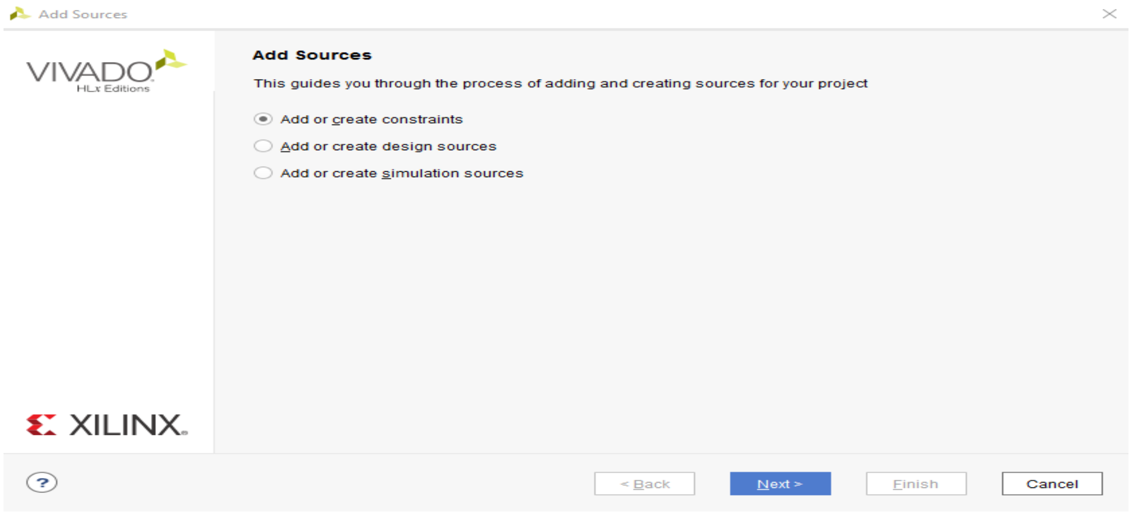
7. The Define Module Window that will appear, we will choose the input and output labels for the gates under investigation in this experiment. In this experiment, we are investigating Boolean Distributive Law and we use 4 inputs to get 2 outputs. Under “Port Name”, add “A”, “B”, “C”, and “D” as inputs and add “X1”, “X2” as outputs and select OK.



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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity src1 is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C : in STD_LOGIC;
           D : in STD_LOGIC;
           X1 : out STD_LOGIC;
           X2 : out STD_LOGIC);
end src1;

architecture Behavioral of src1 is

begin


end Behavioral;
```

X1 = (A+B)(C+D)
X2 = AC+AD+BC+BD

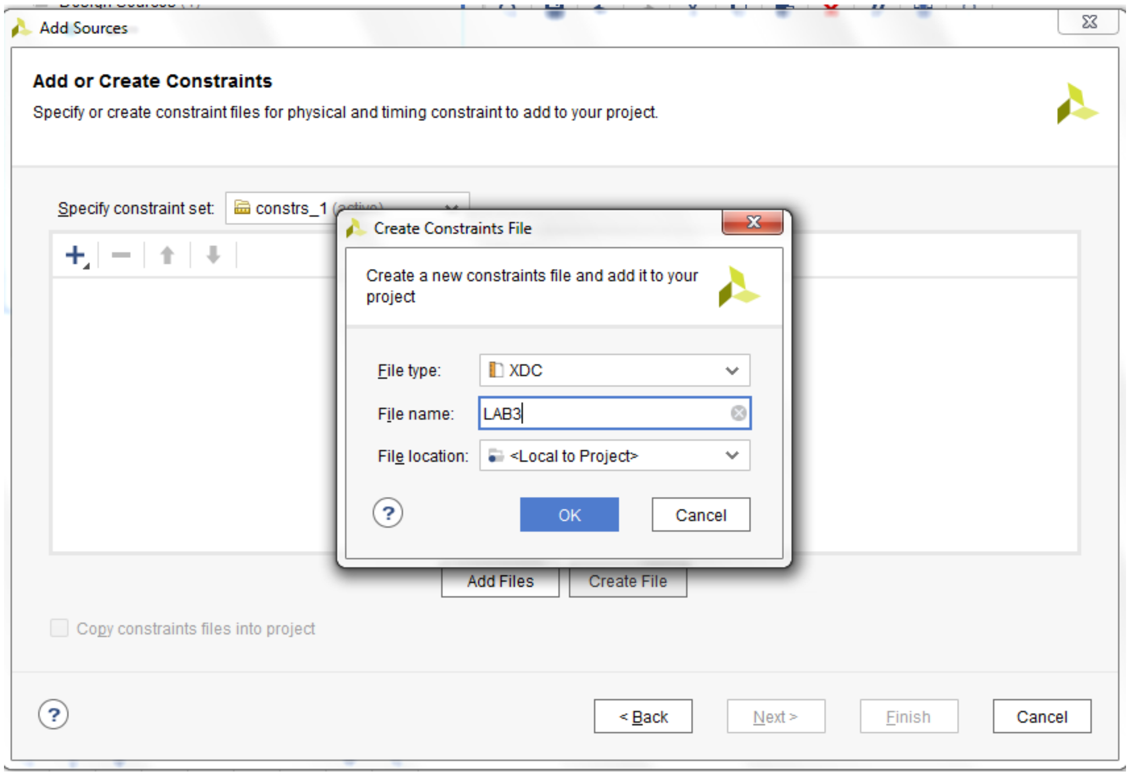
8. In the “source_1.vhd” created file, type the gates equivalent VHDL code for the X1 and X2 between the “begin” and “end Behavioral” as follows and then save the file.



Add Sources

This guides you through the process of adding and creating sources for your project

● Add or create constraints
○ Add or create design sources
○ Add or create simulation sources

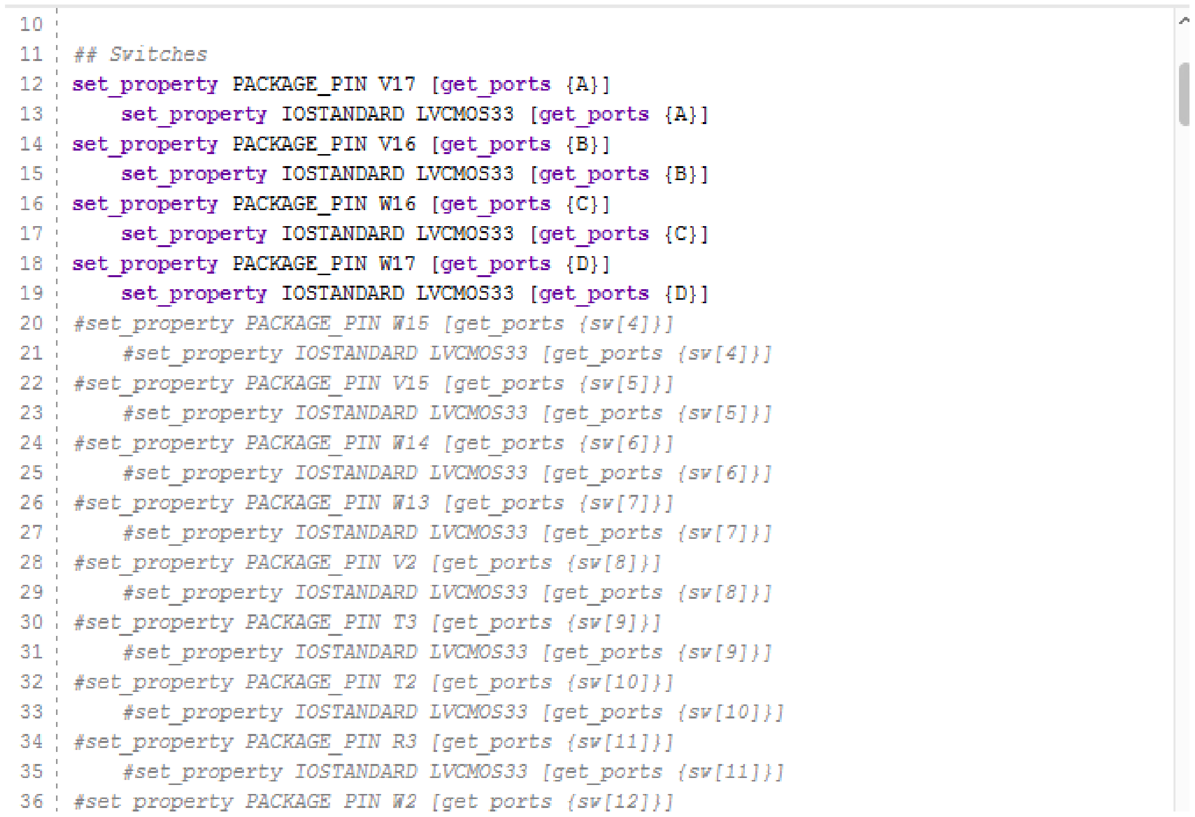

Add Sources
Add or Create Constraints
Specify or create constraint files for physical and timing constraint to add to your project.
Specify constraint set: constrs_1 (active)
Create Constraints File
Create a new constraints file and add it to your project
OK
Cancel
Add Files
Create File
Copy constraints files into project
< Back
Next >
Finish
Cancel

File type: XDC
File name: LAB3
File location: <Local to Project>

9. Next, we need to add to add a constraint file with the".xdc" extension, as following: Go to "Flow Navigator" and from "Project Manager" select "Add Sources" then "Add or create constraints". Next, choose "Create File" and enter the file name "lab_2" then "OK" followed by "Finish".

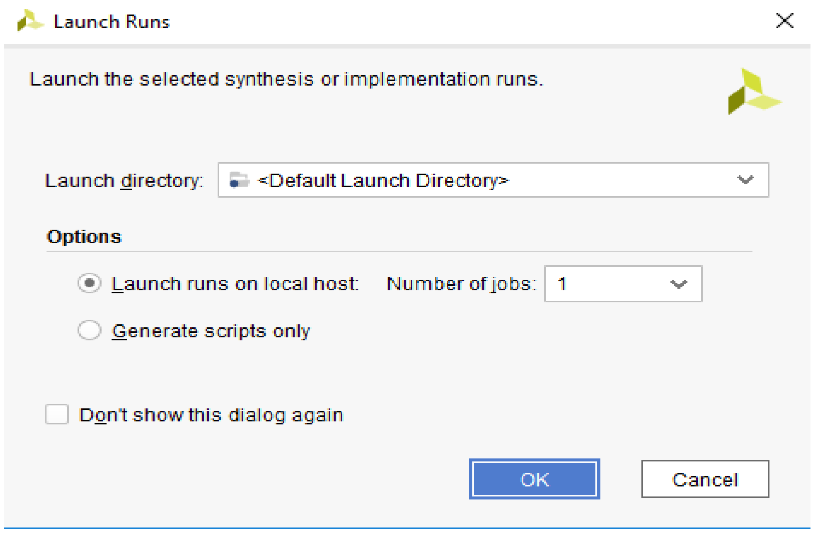
10. Then, we need to get a template xdc file that is going to be edited according to the different experiments. Google "basys 3 xdc file" and choose the "xilinix" link that appears (https://www.xilinx.com/support/documentation/university/Vivado-Teaching/HDL-Design/2015x/Basys3/Supporting%20Material/Basys3_Master.xdc). Copy the whole file and paste it into the "lab_2.xdc" that you have just created in the last step. Then uncomment and edit the input Switches and the output LEDs as in the next step.

11. Uncomment (by deleting the # sign) sw[0], sw[1], sw[3],….. led[0], led[1],… lines. Note that each of them has two successive lines (Uncomment both of them). Do the following replacements: sw[0] ➔ A, sw[1] ➔ B,……, led[0] ➔ X1, led[1] ➔ X2 then save the file.



```

## Switches
set_property PACKAGE_PIN V17 [get_ports {A}]
    set_property IOSTANDARD LVCMOS33 [get_ports {A}]
set_property PACKAGE_PIN V16 [get_ports {B}]
    set_property IOSTANDARD LVCMOS33 [get_ports {B}]
set_property PACKAGE_PIN W16 [get_ports {C}]
    set_property IOSTANDARD LVCMOS33 [get_ports {C}]
set_property PACKAGE_PIN W17 [get_ports {D}]
    set_property IOSTANDARD LVCMOS33 [get_ports {D}]
#set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
#set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
#set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
#set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
#set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
#set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]
#set_property PACKAGE_PIN T2 [get_ports {sw[10]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[10]}]
#set_property PACKAGE_PIN R3 [get_ports {sw[11]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[11]}]
#set_property PACKAGE_PIN W2 [get_ports {sw[12]}]
```

12. From the tool tab choose the play button ▶ and then "Run Implementation". Select "Number of jobs" =1 and then press OK.



Launch Runs

Launch the selected synthesis or implementation runs.

Launch directory: <Default Launch Directory>

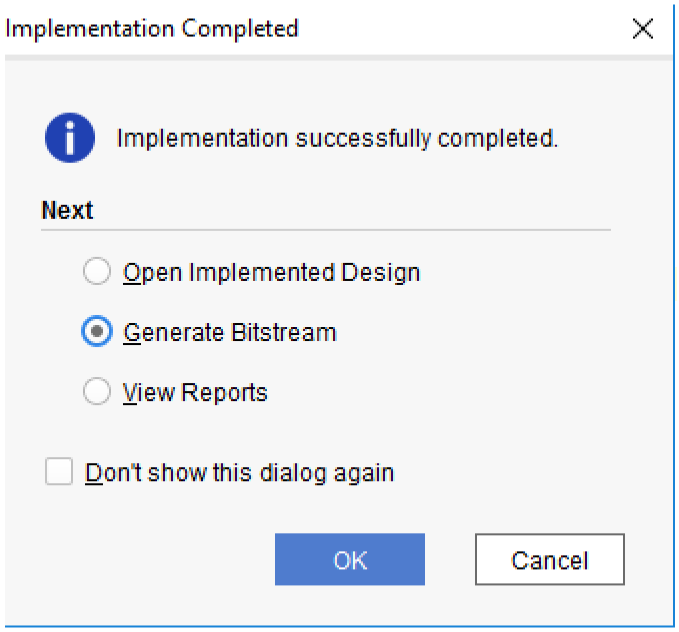
Options

- [✓ Launch runs on local host:] Number of jobs: 1
- Generate scripts only

☐ Don't show this dialog again

13. The implementation errors window will appear if any or the successfully completed window. From this window select "Generate Bitstream" and then OK.

This will make the software generate ".bin" file to be used in programing the hardware BAYAS 3.



Implementation Completed

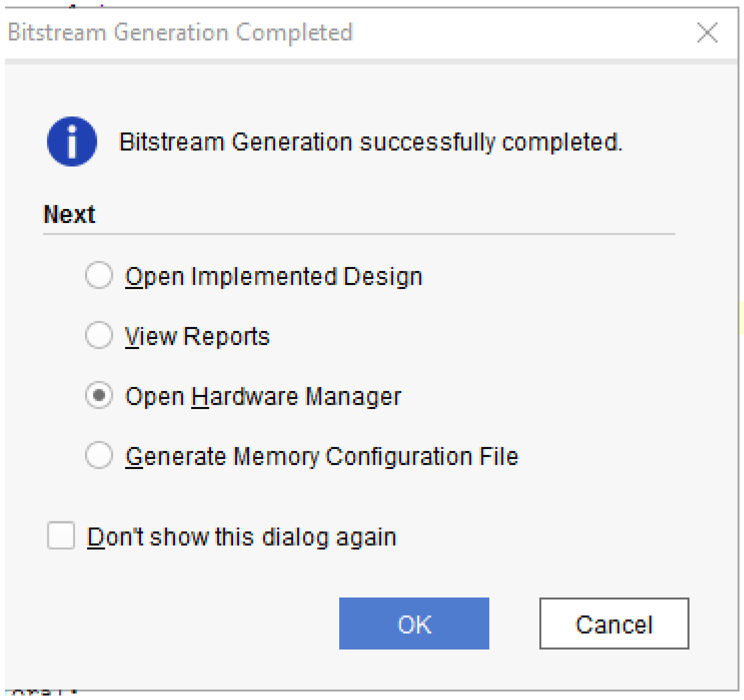
Implementation successfully completed.

Next

( ) Open Implemented Design
[✓ Generate Bitstream]
( ) View Reports

☐ Don't show this dialog again

14. The next window will appear in which choose "Open Hardware Manger", connect the Hardware Kit to the USB port and then press OK.



Bitstream Generation Completed

Bitstream Generation successfully completed.

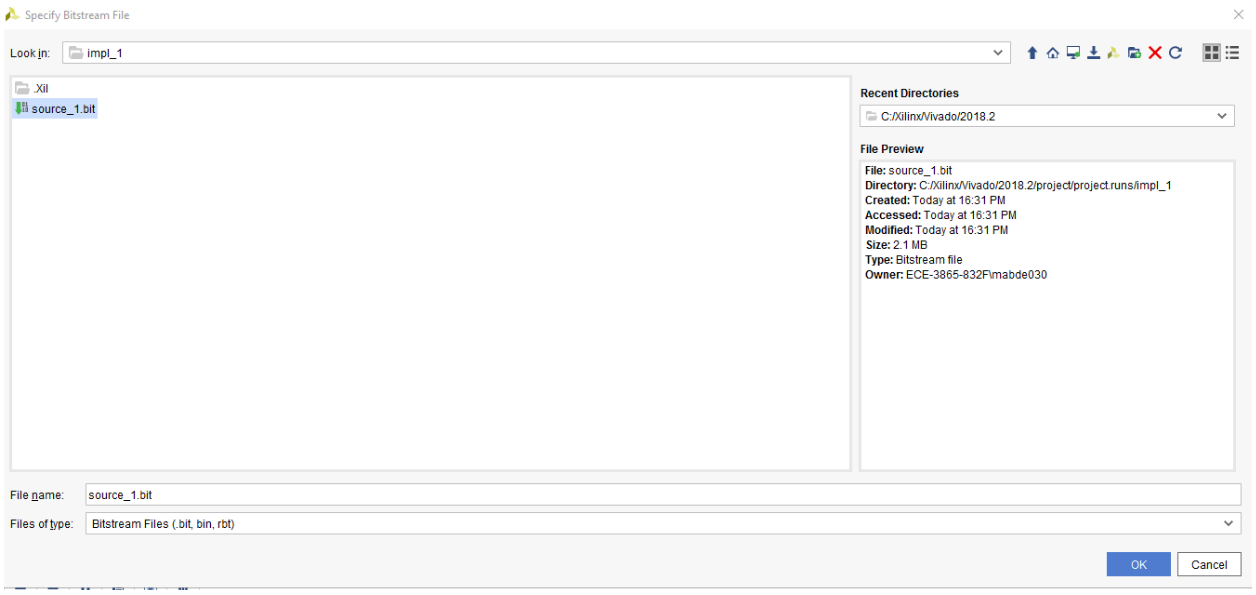
Next

( ) Open Implemented Design
( ) View Reports
[✓ Open Hardware Manager]
( ) Generate Memory Configuration File

☐ Don't show this dialog again

15. A green tab will appear in the top of the Vivado window, from which choose "open target" to program the hardware.

16. From the window appears, select the ".bin" file from the Project you create by browsing for the generated ".bit file" under the ".runs" folder and program the board then press OK.



File: source_1.bit
Directory: C:/Xilinx/Vivado/2018.2/project/project.runs/impl_1
Created: Today at 16:31 PM
Accessed: Today at 16:31 PM
Modified: Today at 16:31 PM
Size: 2.1 MB
Type: Bitstream file
Owner: ECE-3865-832F\mabde030

File name: source_1.bit
Files of type: Bitstream Files (.bit, bin, rbt)

17. Test the program on your board by going through all the input combinations and observing the two outputs. Fill the truth table.

Truth Table (1)

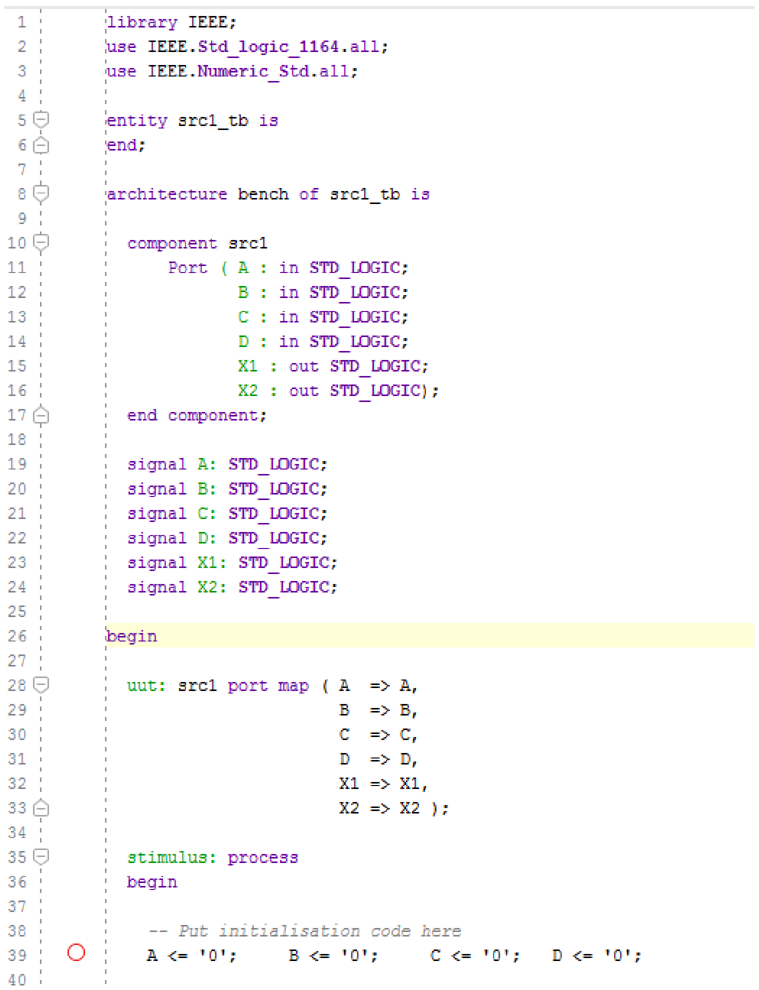
A	B	C	D	X1	X2
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		

Symbol

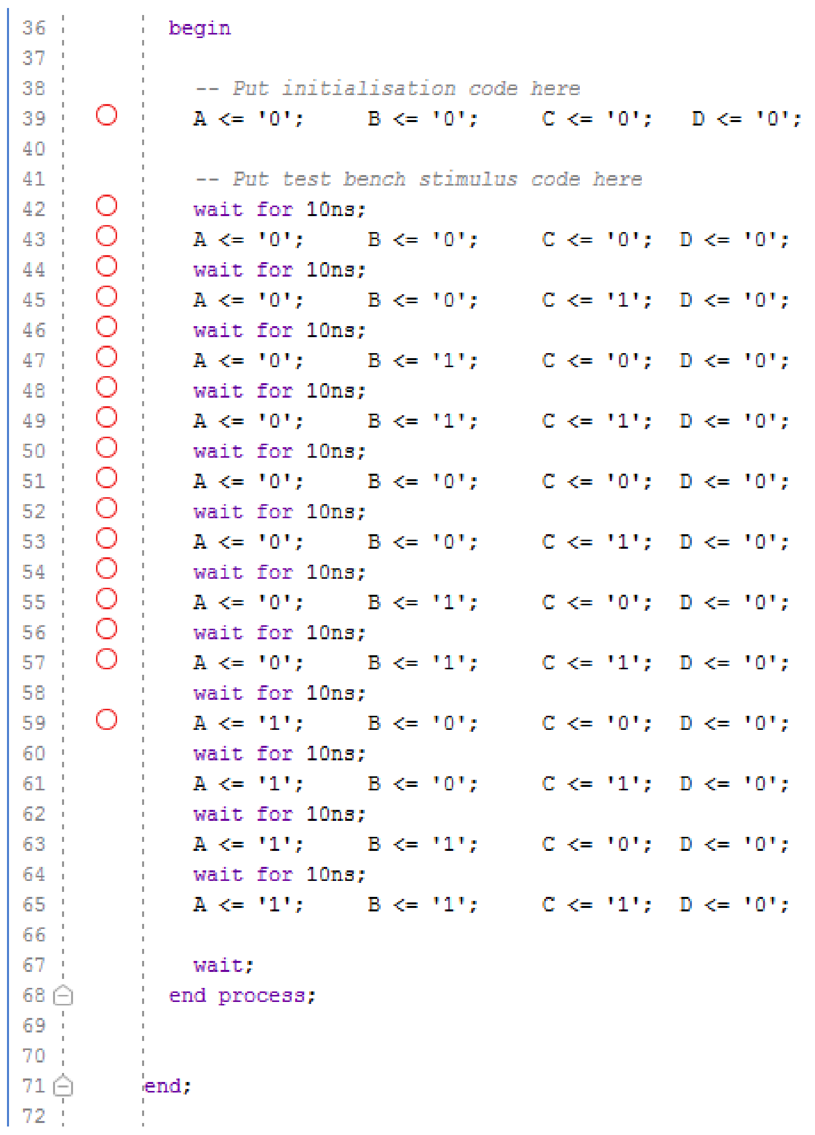
Boolean Equation

0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

18. Then you can use the simulation tools to verify the Boolean distributive law. For simulation, we need to create a simulation source file as following:
19. “Flow Navigator” ➔ “Project Manager” ➔ “Add Sources” ➔ “Add or create simulation sources” ➔ Name it “TB” (Test Bench) ➔ “VHDL” ➔ No need for switches and leds assignments as we will not be working on board. ➔ “OK”.
20. In the “initialization” section write “A <= '0'; B <= '0'; C <= '0'; D <='0';”. In the stimulus section, the following code (as in the figure) is written to simulate the whole truth table:



```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity src1_tb is
end;

architecture bench of src1_tb is

  component src1
      Port ( A : in STD_LOGIC;
             B : in STD_LOGIC;
             C : in STD_LOGIC;
             D : in STD_LOGIC;
             X1 : out STD_LOGIC;
             X2 : out STD_LOGIC);
  end component;

  signal A: STD_LOGIC;
  signal B: STD_LOGIC;
  signal C: STD_LOGIC;
  signal D: STD_LOGIC;
  signal X1: STD_LOGIC;
  signal X2: STD_LOGIC;

begin

  uut: src1 port map ( A  => A,
                       B  => B,
                       C  => C,
                       D  => D,
                       X1 => X1,
                       X2 => X2 );

  stimulus: process
  begin

    -- Put initialisation code here
    A <= '0';     B <= '0';     C <= '0';    D <= '0';

```



```
  begin

    -- Put initialisation code here
    A <= '0';     B <= '0';     C <= '0';    D <= '0';

    -- Put test bench stimulus code here
    wait for 10ns;
    A <= '0';     B <= '0';     C <= '0';  D <= '0';
    wait for 10ns;
    A <= '0';     B <= '0';     C <= '1';  D <= '0';
    wait for 10ns;
    A <= '0';     B <= '1';     C <= '0';  D <= '0';
    wait for 10ns;
    A <= '0';     B <= '1';     C <= '1';  D <= '0';
    wait for 10ns;
    A <= '0';     B <= '0';     C <= '0';  D <= '0';
    wait for 10ns;
    A <= '0';     B <= '0';     C <= '1';  D <= '0';
    wait for 10ns;
    A <= '0';     B <= '1';     C <= '0';  D <= '0';
    wait for 10ns;
    A <= '0';     B <= '1';     C <= '1';  D <= '0';
    wait for 10ns;
    A <= '1';     B <= '0';     C <= '0';  D <= '0';
    wait for 10ns;
    A <= '1';     B <= '0';     C <= '1';  D <= '0';
    wait for 10ns;
    A <= '1';     B <= '1';     C <= '0';  D <= '0';
    wait for 10ns;
    A <= '1';     B <= '1';     C <= '1';  D <= '0';

    wait;
  end process;


end;

```

Section II. Boolean Absorption Rule

The two shown below are called absorption rules.

$$A + \overline{A}B = A + B$$

$$\overline{A} + AB = \overline{A} + B$$

You are asked to prove them using either your target board or simulation tools.
TIPS: You can combine the Section II and Section III together, draw them in only one project.

1) Create a new project call BRules

2) Using the same process before to build the circuit to verify the two absorption equation. You can use two inputs A and B, four outputs X1, X2, Y1, Y2. If you show that the output waveforms X1and X2 are the same, and the Y1 and Y2 are the same, you have already verify the Boolean absorption rule.

$$X1 = A + \overline{A}B \qquad X2 = A + B$$

$$Y1 = \overline{A} + AB \qquad Y2 = \overline{A} + B$$

Input		Output			
A	B	X1	X2	Y1	Y2

Section III. DeMorgan's Theorem

The two shown below are called DeMorgan's Theorem

$$\overline{A + B} = \overline{A}\,\overline{B}$$

$$\overline{AB} = \overline{A} + \overline{B}$$

You are asked to prove them using either your target board or simulation tools.

1) Create a new project called Demorgan, build the circuit that enable you to prove the two equations. You will need two inputs A and B, two pairs of outputs.

$$X1 = \overline{A + B} \qquad X2 = \overline{A}\,\overline{B}$$

$$Y1 = \overline{AB} \qquad Y2 = \overline{A} + \overline{B}$$

2) Fill the form

Input		**Output**			
A	**B**	**X1**	**X2**	**Y1**	**Y2**

Checked by________________________________ Date ________________________

QUESTIONS:

1) Draw the logic diagrams and list the truth tables for all the Boolean algebra rules except the absorption rule.

2) Apply Boolean laws and rules and DeMorgan’s theorem to simplify the following Boolean equations. Draw the simplified logic diagrams.

a) $X=(A+B)\overline{ABC}+\overline{BC}$ b) $Y=(\overline{A+B})\overline{B}+B+\overline{AC}$