Task Assigned Date: May 14, 2024,

Task Finished Date: May 14, 2024

Step by Step Document on Converting 4-bit array multiplier into 5-bit array multiplier in Verilog HDL.

This new task is to convert a 4-bit array Multiplier into **5-bit array Multiplier** using **Verilog HDL.**

The Code was already provided for the Half Adder, Full Adder and 4-bit array Multiplier.

1. Understanding Theory:

So, lets understand how to design a 2-bit Array Multiplier in HDL at first.

1.1. 2-bit Multiplier:

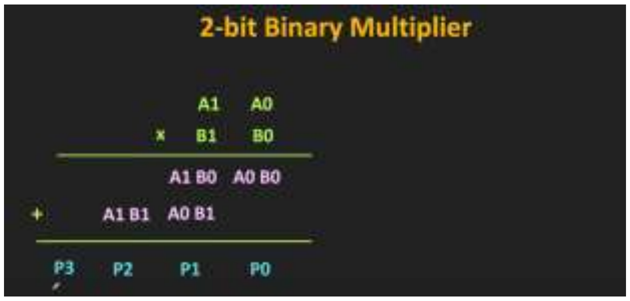


Figure 1: Two Bit Binary Multiplication

From Figure 1, we can see that the binary multiplication of 2-digits are as follows:

We can see that both Multiplicands are multiplied with each other then added throughout a column.

We can deduce that First, bits are multiplied to each other and then added together to form an output bit. So, for multiplication of the bit which are either ‘0’ or ‘1’. We can use AND gate for purpose. Then For adding up all the values together we can use the Adders Circuit. For inputs who do not require Carry Input we can use Half Adders. Then for those who require Carry we can use Full Adder. Let’s take schematic example of the 2-bit adder.

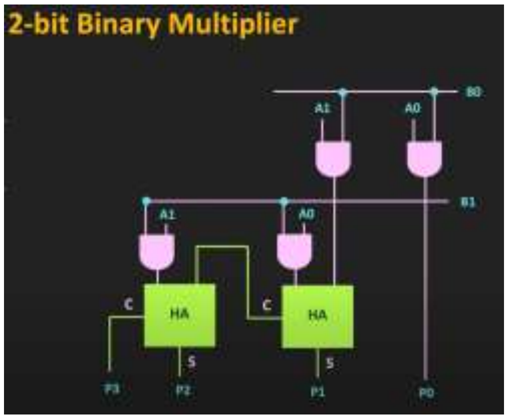


Figure 2: Logic Diagram of the 2-bit Multiplier

Here, in Figure 2, we can see that we only require two half adders as we only required Adders with 2-inputs and no carry is needed to carry into the adders. We multiply the Bits with AND gate.

Suppose if A0 = ‘0’ and B0 = ‘1’. Then it will give ‘0’ because it is multiplication. But if A0 = ‘1’ and B0 = ‘1’ only then it will pass a value.

Now, lets look into Half Adder and Full Adder Circuits.

1.1.1. Half Adder:

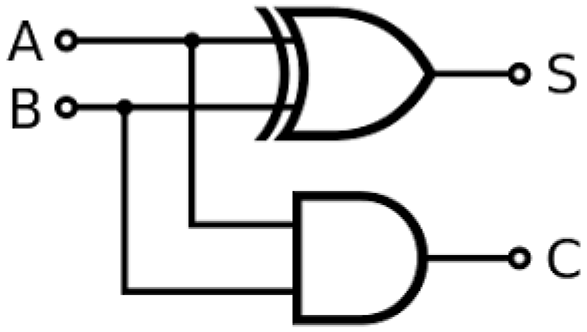
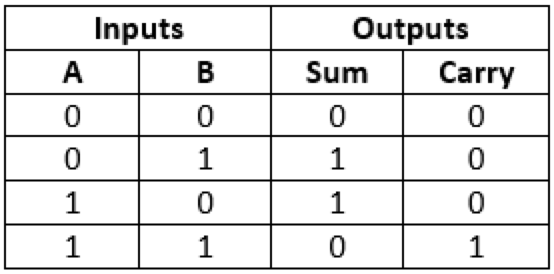


Figure 3: Half Adder Circuit

Half Adder Acts as the following truth table.



Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 4: Half Adder Truth Table

It gives output of sum if anyone of the inputs are **HIGH**. When Both Inputs are **HIGH**. Then, it only gives out the carry bit.

By simplifying it with Karnaugh Maps. The Equations for the Half Adder are as follows:

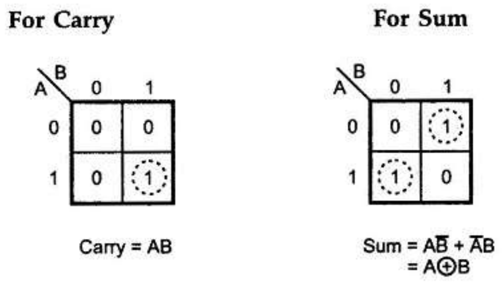


Figure 5: Equations of the Half Adder

Now, lets move on the Full Adder Circuit.

1.1.2. Full Adder Circuit:

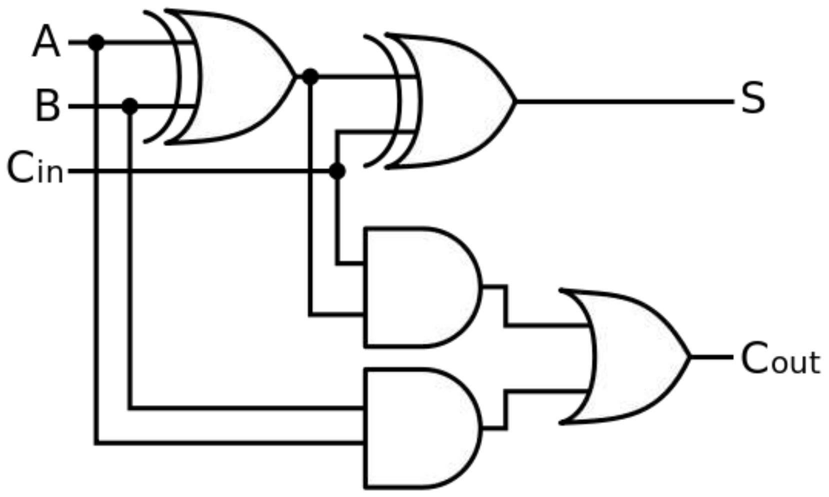
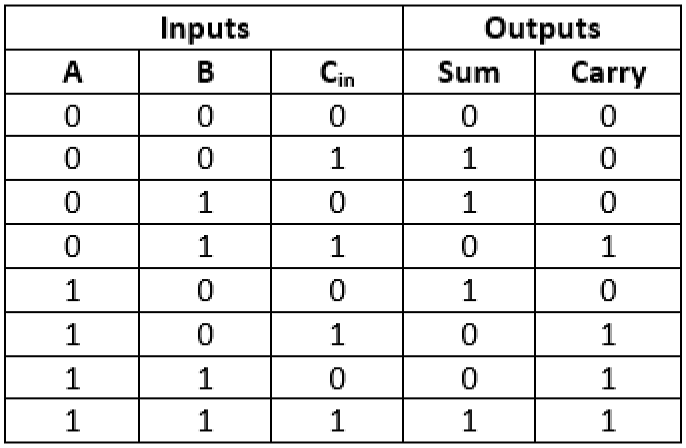


Figure 6: Full Adder Circuit

The Full Adder Circuits work with three incoming inputs. The only difference from the half adder circuit is that it also carries the Carry In bit or C_{in} bit. Now, it can provide different kinds of output depending on the inputs. The truth Table of the Full Adder Circuit is shown in Figure 7. The Simplified version is also available later.



Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 7: Truth Table of the Full Adder Circuit

We can simplify this table with Karnaugh maps and can make it into equations. The Simplified version is as shown in Figure 8.

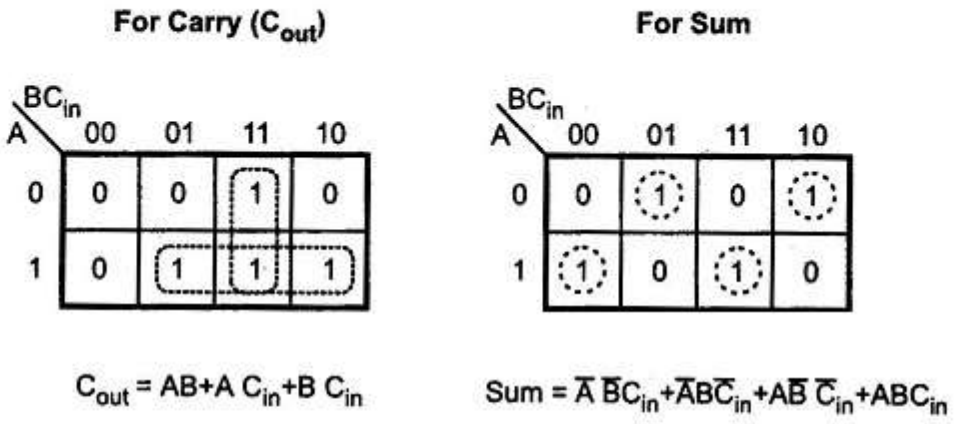

For Carry (C_{out})
For Sum
BC_{in}
A
00
01
11
10
$C_{out} = AB+A\,C_{in}+B\,C_{in}$
$Sum = \overline{A}\,\overline{B}C_{in}+\overline{A}B\overline{C}_{in}+A\overline{B}\,\overline{C}_{in}+ABC_{in}$

Figure 8: Equations of Full Adder

Now, that we have seen the Full Adder and Half Adder Circuit. We can now join them together to form a N-Multiplier Circuit.

1.2. Three-Bit Multiplier:

In three bit multiplier we can see that three inputs are used together so we need to join a full Adder in the circuit.

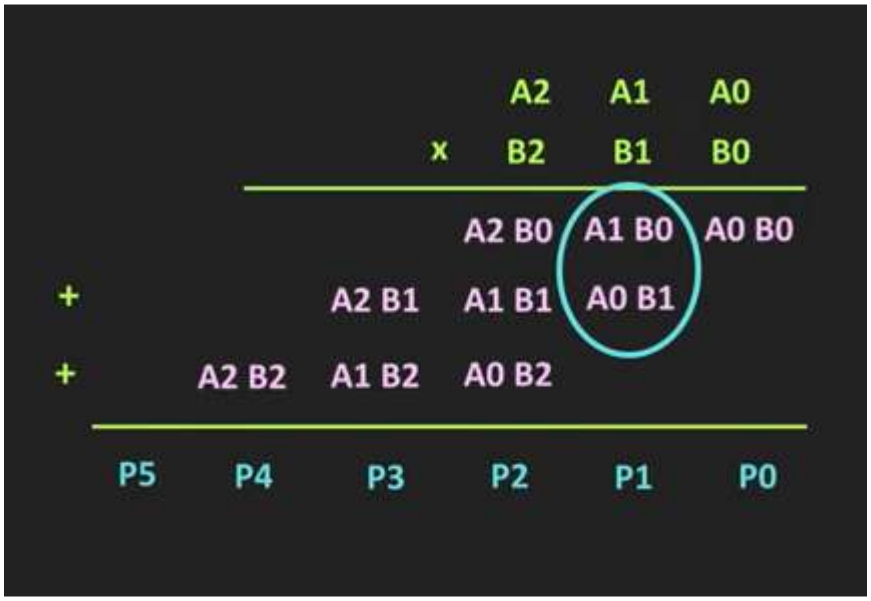


Figure 9: Three Bit Array Multiplication

For three-bit multiplication. We need to multiply Each Multiplicand of A to each Multiplicands of B. So, the circuit for the three-bit Circuit is as follows:

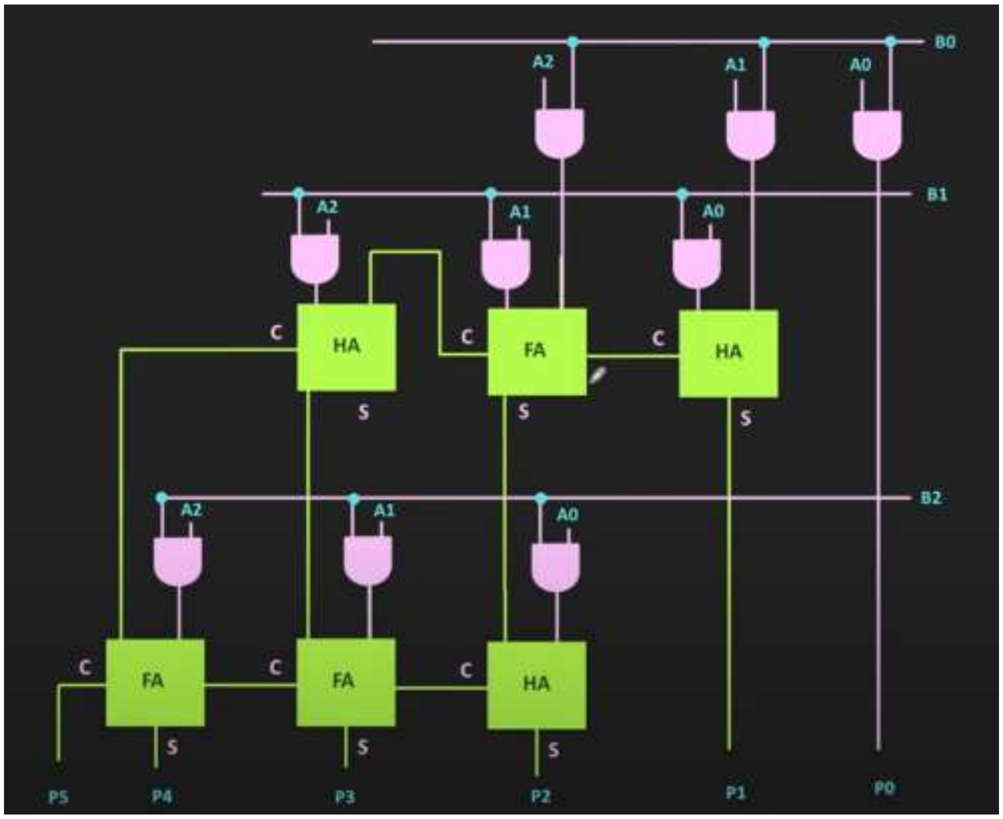
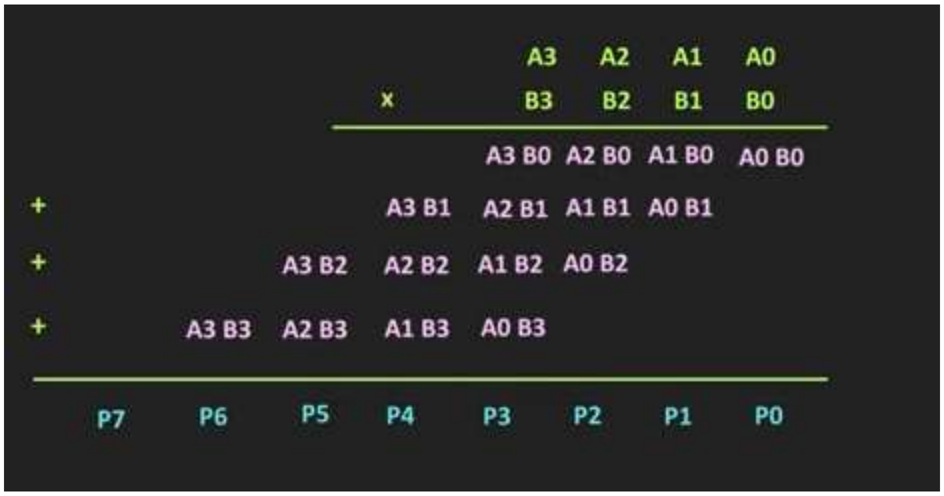


Figure 10: Three-Bit Multiplier Logic Diagram

Now, Mix of Full Adders and Half Adders are combined together in order to form the 3-bit multiplier circuit. Similarly, for 4-bit Multiplier we can do the same thing.

1.3. 4-bit Multiplier:

Four Bit Multiplier logic is as follows:



					A3	A2	A1	A0
				x	B3	B2	B1	B0
					A3 B0	A2 B0	A1 B0	A0 B0
+				A3 B1	A2 B1	A1 B1	A0 B1	
+			A3 B2	A2 B2	A1 B2	A0 B2		
+		A3 B3	A2 B3	A1 B3	A0 B3			
	P7	P6	P5	P4	P3	P2	P1	P0

Figure 11: 4-bit Multiplier Multiplication Table

The Logic Diagram of the circuit is as follows:

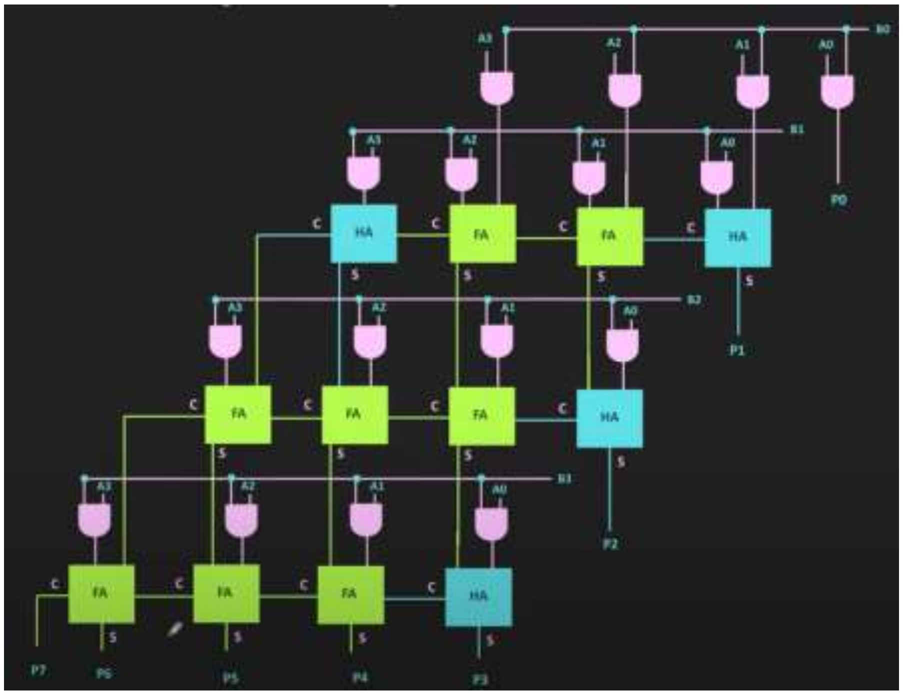


Figure 12: 4-bit Multiplier Logic Diagram

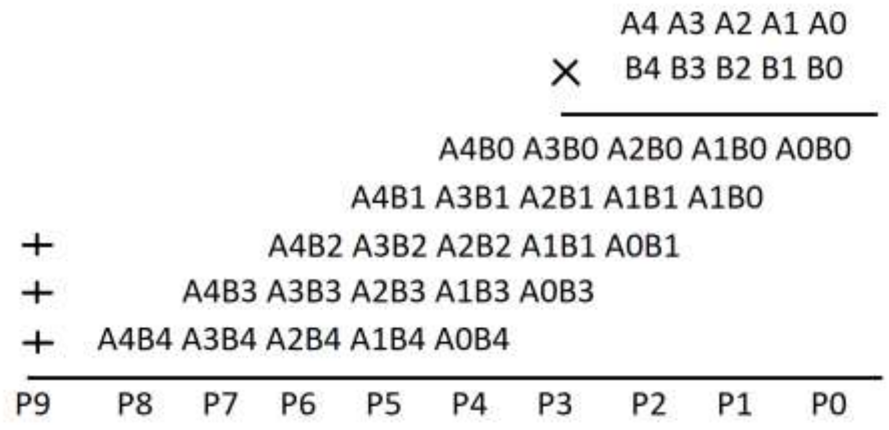
Now, that the multiplication table is drawn. We can draw logic diagram for the 4-bit multiplier.

Now, lets come over to our task. We need to convert this 4-bit logic Diagram into 5-bit Multiplier Circuit.

Lets revise things again. For 5-bit Multiplication. We have multiplicand bits on each Side. Let's name then X and Y bit for a bit of change.

The Multiplication Table is going to be as follows

1.4. Five-Bit Multiplier:



```
                                          A4 A3 A2 A1 A0
                                       ×  B4 B3 B2 B1 B0
                                       _________________
                              A4B0 A3B0 A2B0 A1B0 A0B0
                         A4B1 A3B1 A2B1 A1B1 A1B0
+                   A4B2 A3B2 A2B2 A1B1 A0B1
+              A4B3 A3B3 A2B3 A1B3 A0B3
+         A4B4 A3B4 A2B4 A1B4 A0B4
___________________________________________________
P9    P8   P7   P6   P5   P4   P3   P2   P1   P0
```

Figure 13: 5-bit multiplication table

So, the circuit diagram is going to be as follows:

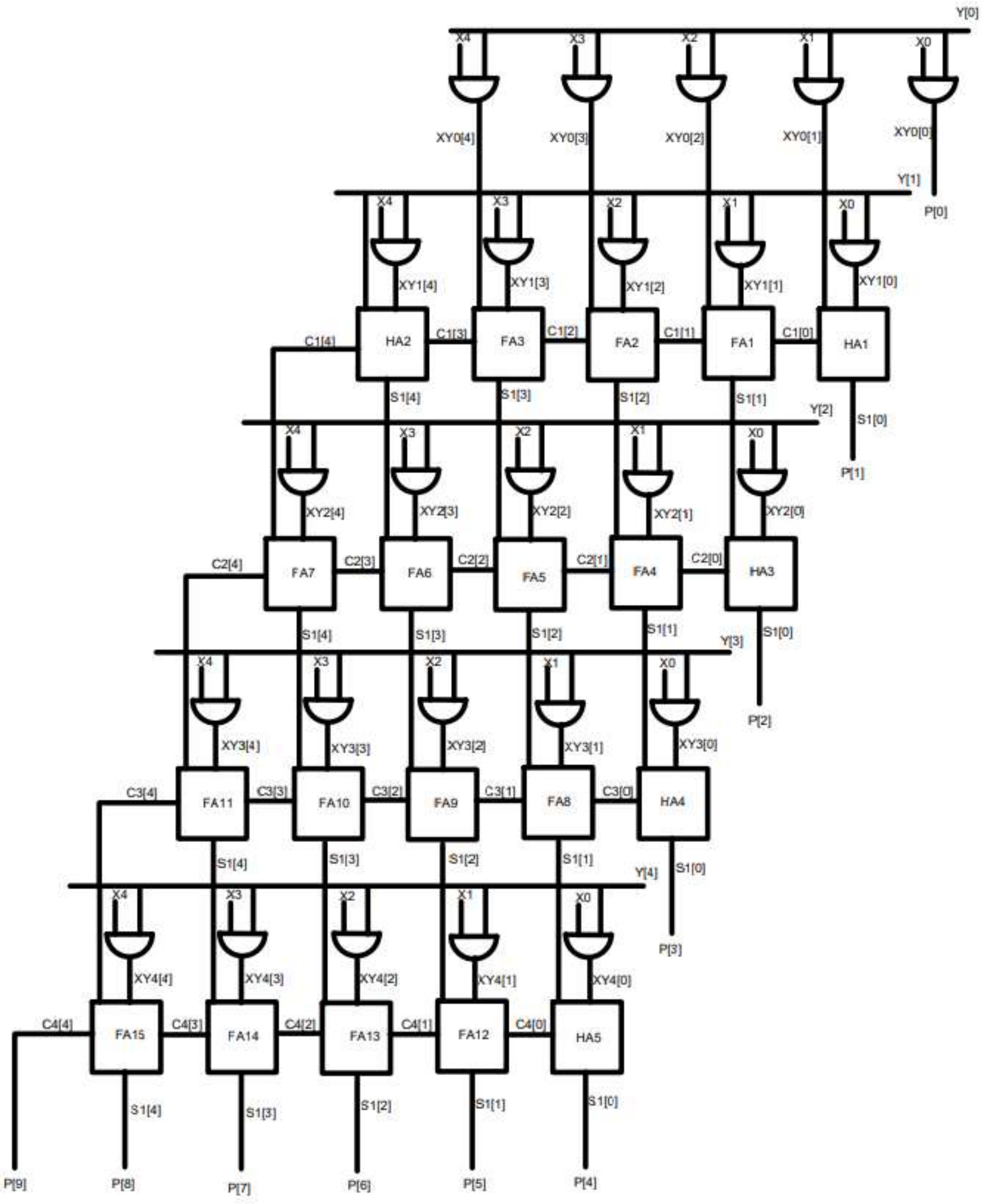


Figure 14: 5-bit multiplier logic diagram

2. Writing RTL Code:

Now, we need to write RTL/HDL Code for this Logic Diagram.

For that, we need to follow these steps below:

2.1. Step 1: Open Xilinx Vivado for Simulation Purposes.

Click on the Vivado Logo to open vivado.

2.2. Step 2: Creating a Project in Vivado

1. When Vivado is Opened. The Default Screen should look like this.

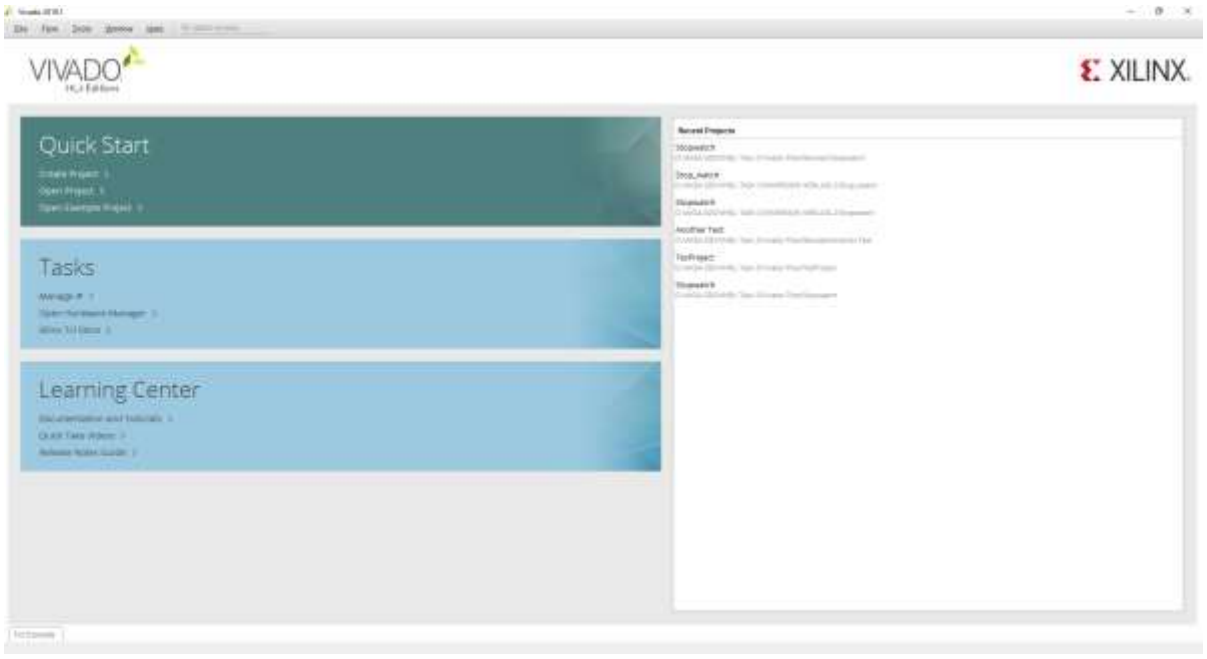
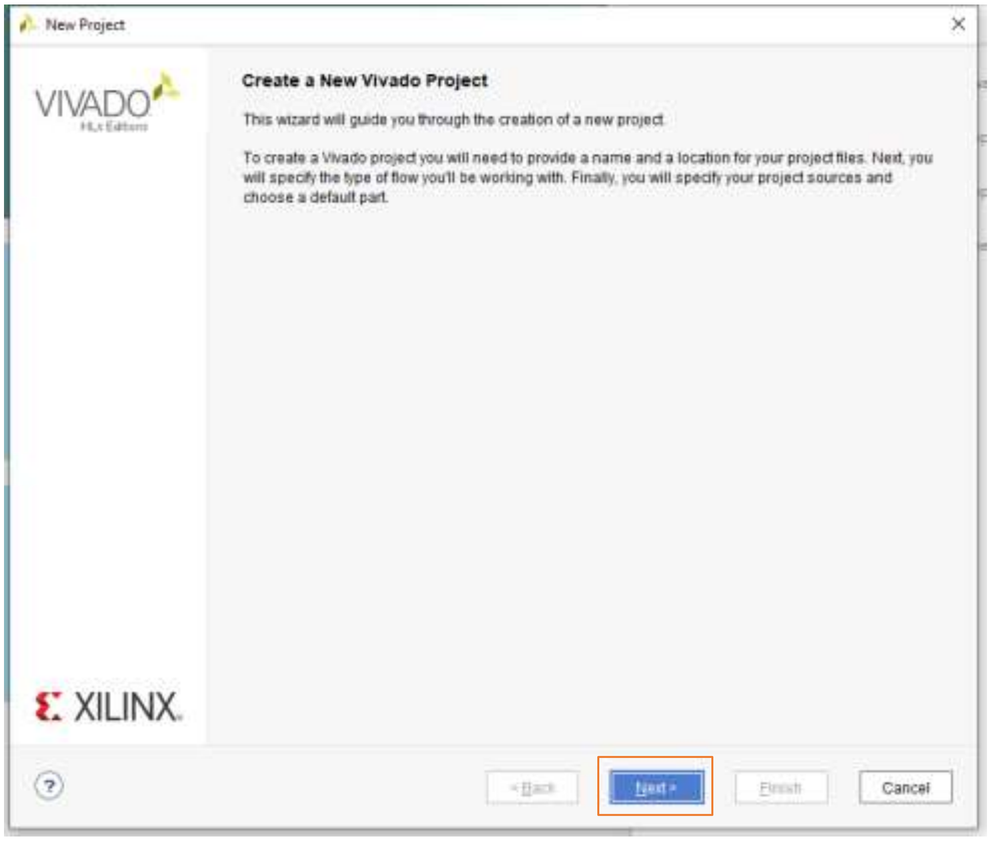


Figure 15: Vivado Default Screen

2. To Create a project in vivado click on Quick Start → Create Project as shown in the Figure 4.
3. A New Project Window will appear in the screen as shown in Figure 5.



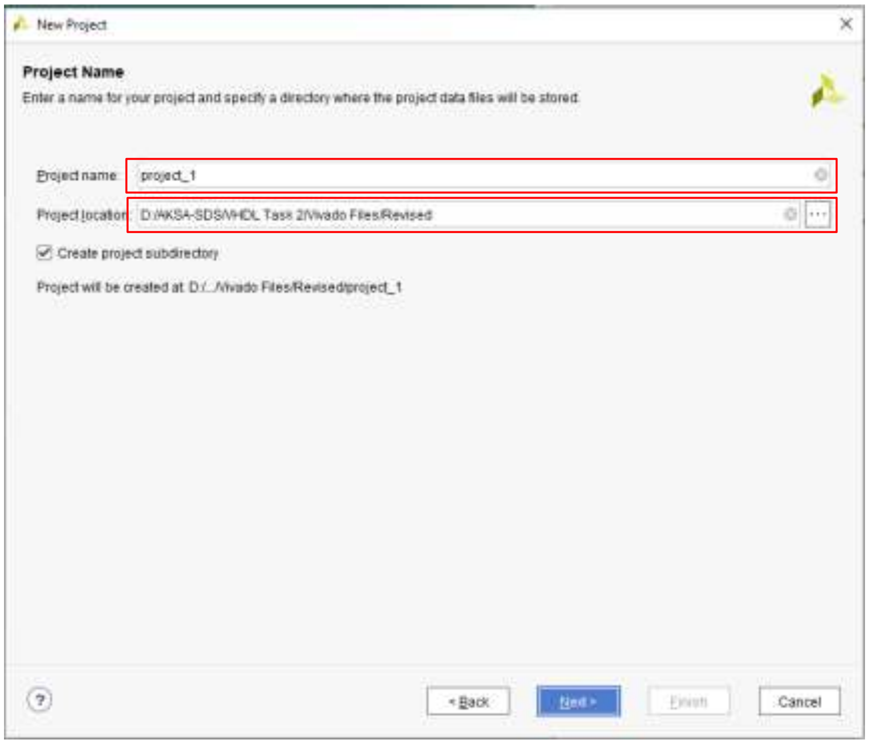
Create a New Vivado Project

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.

Figure 16: New Project Window in Vivado

4. Click on Next in the New Project Screen as highlighted in Figure 5.


New Project

Project Name

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

Project name: project_1

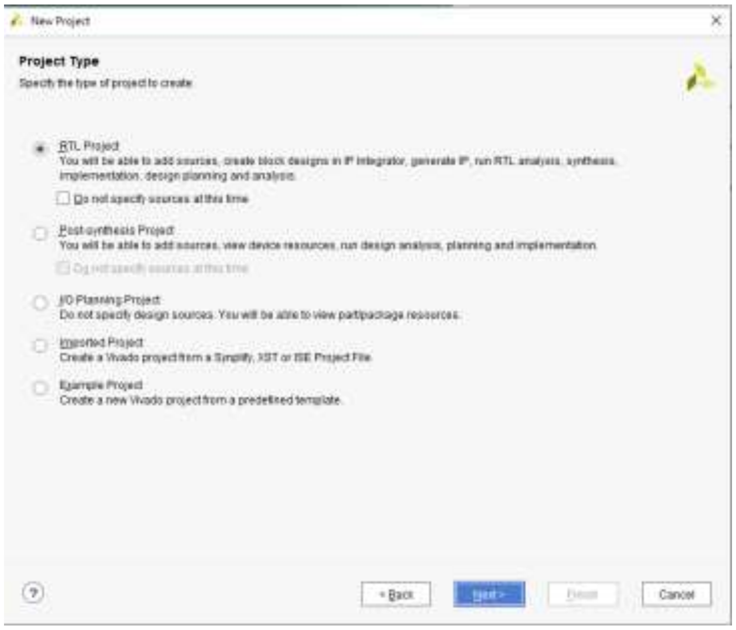
Project location: D:/AKSA-SDS/VHDL Task 2/Vivado Files/Revised

☑ Create project subdirectory

Project will be created at: D:/.../Vivado Files/Revised/project_1

Figure 17: Project Name and Destination Window

5. Now, Figure 6 will appear which will ask for project name and project location of the project.
6. Just Name the project whatever you want in this case. I want to name it FiveBitMultiplier because the task given to me is to create a FiveBitMultiplier. And you can choose the location of the project as well.
7. Click on Next and the window on Figure 7 will appear. Choose RTL Project and Click Next as well.


New 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.

Figure 18: Project Type Window

8. Click on **Next** → **Next** → **Next** → **Finish** to Create the Project.

2.3. Step 3: Adding Source and Testbench File in Vivado

1. The Default Project screen after creating a project in vivado will be as follows:

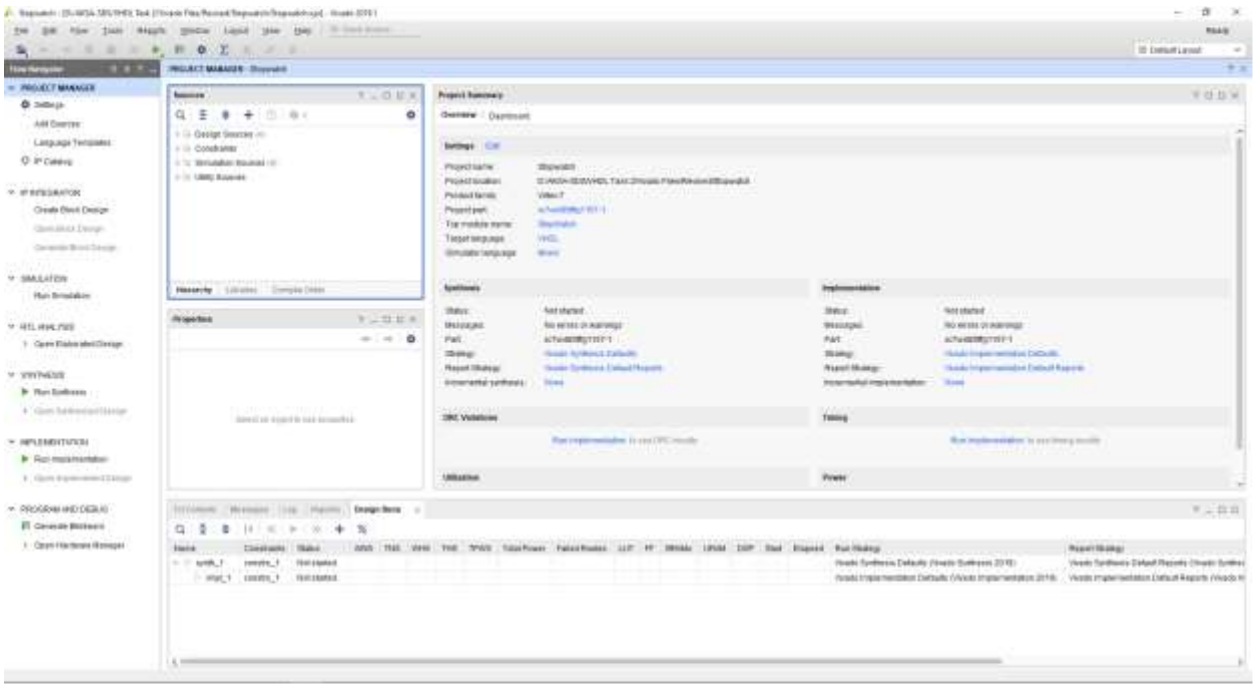


Figure 19: Default Project Screen in Vivado

2. Now to add sources in Vivado Click on Add Sources on Right Top Side of the vivado Screen as shown in the Figure 20.

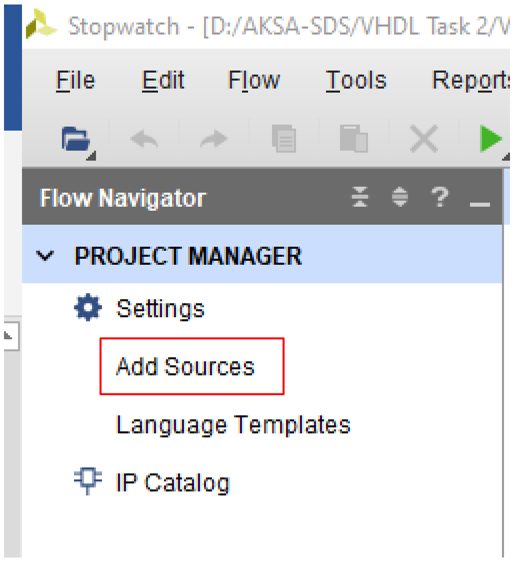
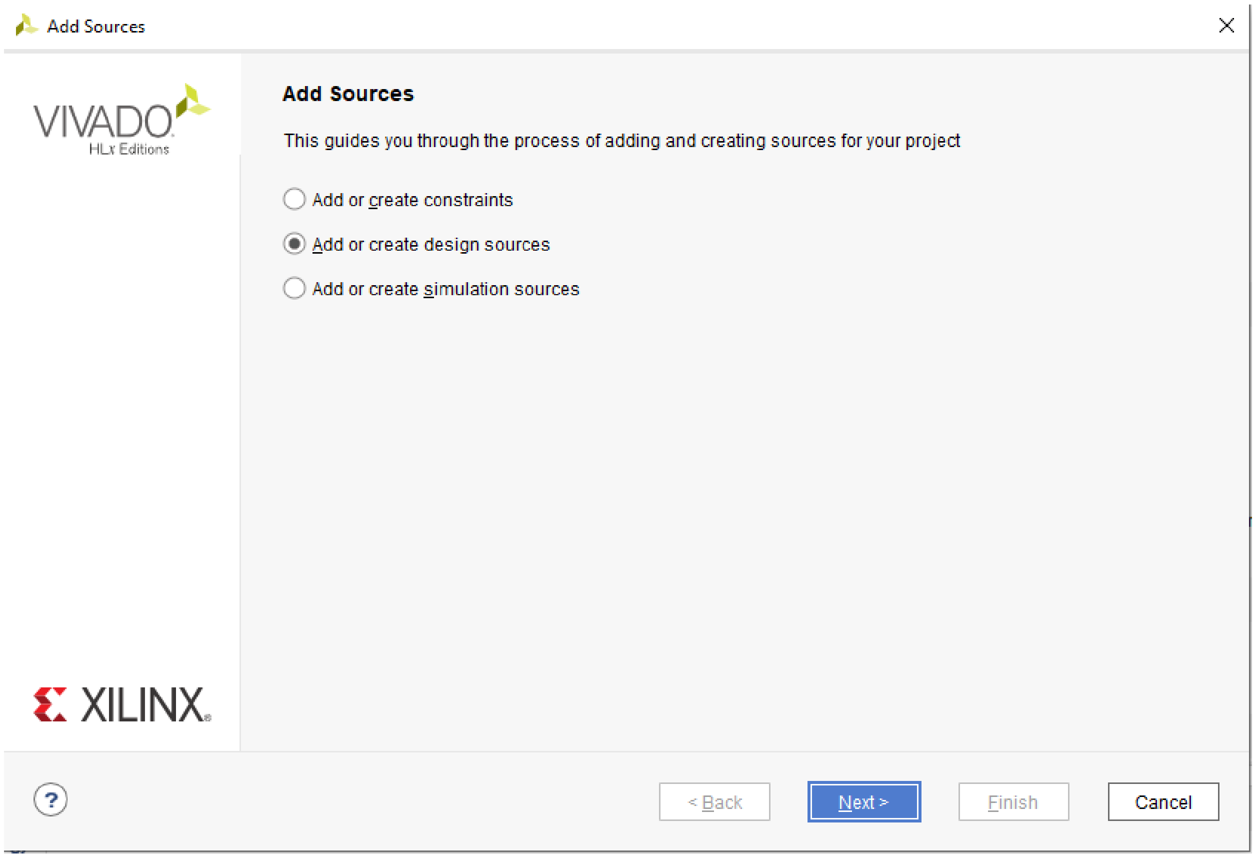


Figure 20: Add source button in vivado

3. When Clicked on Add souces button a following screen will appear.



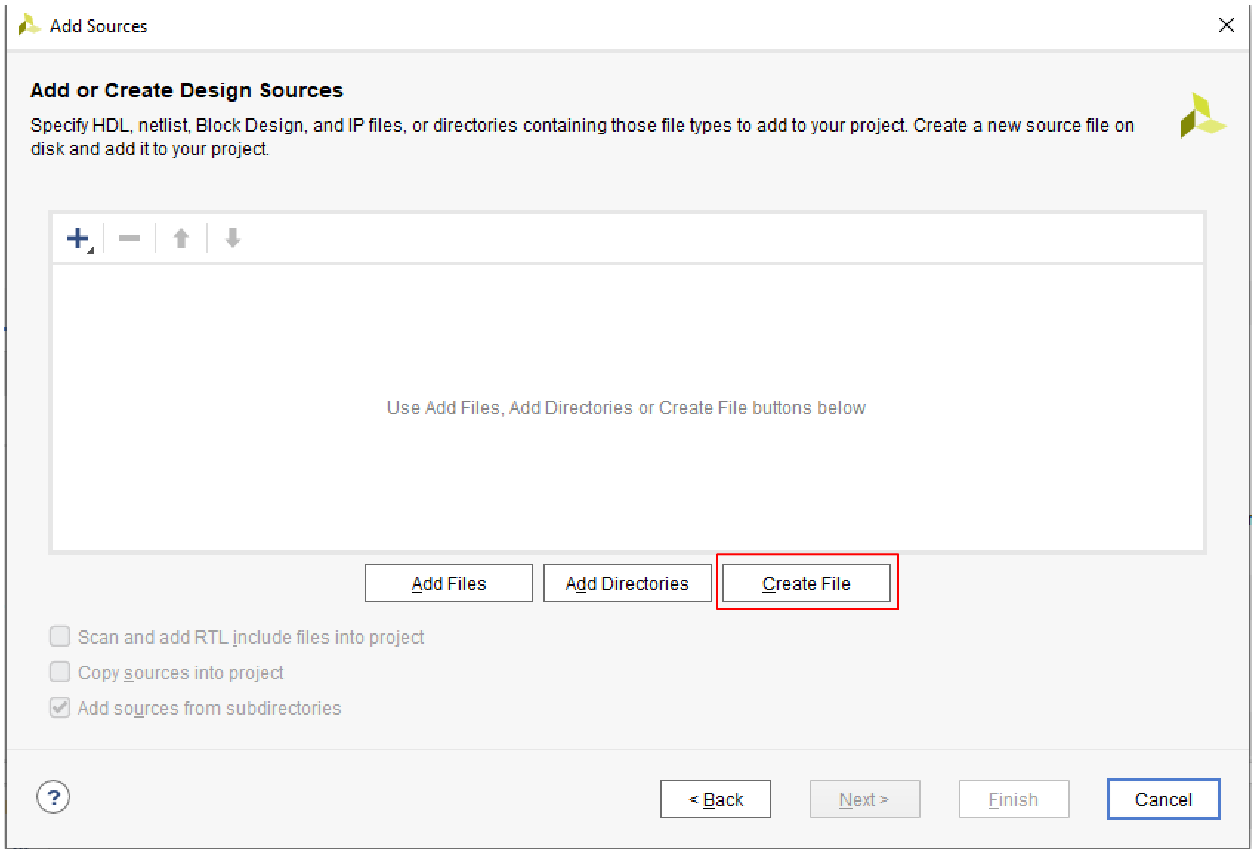
Add Sources

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

Figure 21: Add sources screen in Vivado

4. Now, chose whether to add design or simulation sources in vivado.
5. For, now we will choose design sources to create the design in vivado


Add Sources

Add or Create Design Sources

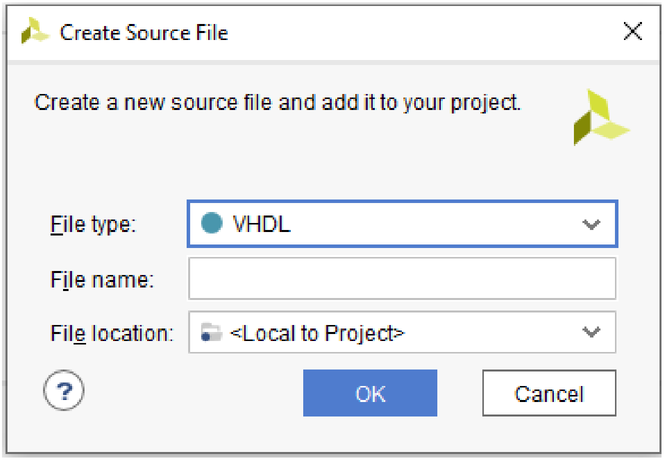
Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

Use Add Files, Add Directories or Create File buttons below

Add Files
Add Directories
Create File
Scan and add RTL include files into project
Copy sources into project
Add sources from subdirectories
< Back
Next >
Finish
Cancel

Figure 22: Creating Source File in Vivado

6. Click on Create File in Figure 22. The Following Screen in Figure 23 will appear.


Create Source File

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

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

Figure 23: Create Source File Window

7. Select File Type and File name and click on next to create the source file.

3. Code Section:

Now, we need to add components to instantiate which are Half Adder and Full Adder as discussed in the beginning of the report.

Let’s show the code of the modules of Half Adder and Full Adder.

3.1. Half Adder:

Code:

```
`timescale 1ns / 1ps

module HalfAdder(

   input x,

   input y,

   output Cout,

            output sum

   );

           assign {Cout, sum} = x + y;

endmodule
```

We are getting sum and Cout as the output as required.

Now, onto the Full Adder.

3.2. Full Adder:

Code:

```
`timescale 1ns / 1ps

module FullAdder(

   input x,

   input y,

   input Cin,

   output Cout,

            output sum

   );

           assign {Cout, sum} = x + y + Cin;

endmodule
```

we are getting the same output but this time Cin is added as an additional input.

Now, lets write the Code for the top module of the five-bit array multiplier.

3.3. Five-bit array multiplier:

Code:

```
`timescale 1ns / 1ps
module arrayMultiplier
(
   input  [4:0] X, // Multiplicand
   input  [4:0] Y, // Multiplier
   output [9:0] P  // Product
   );
          // Declaring wires for the Carry bit of the adders
          // For Row 1
          wire[4:0] C1;
          // For Row 2
          wire[4:0] C2;
          // For Row 3
          wire[4:0] C3;
          // For Row 4
          wire[4:0] C4;

          //Declaring wires for the sum out bit of the adders.
          // For Row 1
          wire[4:0] S1;
          // For Row 2
          wire[4:0] S2;
          // For Row 3
          wire[4:0] S3;
          // For Row 4
          wire[4:0] S4;

          // Declaring variables for output of the AND Gates for each row respectively.
          wire[4:0] XY0;
          wire[4:0] XY1;
          wire[4:0] XY2;
          wire[4:0] XY3;
          wire[4:0] XY4;

          // Taking AND gate for multiplication of each multiplicand of X with each multiplicand of Y
          assign XY0[0] = X[0] & Y[0];
          assign XY0[1] = X[1] & Y[0];
          assign XY0[2] = X[2] & Y[0];
          assign XY0[3] = X[3] & Y[0];
          assign XY0[4] = X[4] & Y[0];

          assign XY1[0] = X[0] & Y[1];
          assign XY1[1] = X[1] & Y[1];
          assign XY1[2] = X[2] & Y[1];
          assign XY1[3] = X[3] & Y[1];
          assign XY1[4] = X[4] & Y[1];

          assign XY2[0] = X[0] & Y[2];
          assign XY2[1] = X[1] & Y[2];
          assign XY2[2] = X[2] & Y[2];
          assign XY2[3] = X[3] & Y[2];
          assign XY2[4] = X[4] & Y[2];

          assign XY3[0] = X[0] & Y[3];
          assign XY3[1] = X[1] & Y[3];
          assign XY3[2] = X[2] & Y[3];
          assign XY3[3] = X[3] & Y[3];
          assign XY3[4] = X[4] & Y[3];

          assign XY4[0] = X[0] & Y[4];
          assign XY4[1] = X[1] & Y[4];
          assign XY4[2] = X[2] & Y[4];
          assign XY4[3] = X[3] & Y[4];
          assign XY4[4] = X[4] & Y[4];

          //Row 1
          // Instantiation of Half and Full Adders.
          HalfAdder HA1 (XY0[1],  XY1[0],  C1[0], S1[0]);
          FullAdder FA1 (XY0[2],  XY1[1],  C1[0], C1[1],  S1[1]);
          FullAdder FA2 (XY0[3],  XY1[2],  C1[1], C1[2],  S1[2]);
          FullAdder FA3 (XY0[4],  XY1[3],  C1[2], C1[3],  S1[3]);
          HalfAdder HA2 (XY1[4],  C1[3],   C1[4], S1[4]);

          // Row 2

          HalfAdder HA3 (S1[1],  XY2[0],  C2[0], S2[0]);
           FullAdder FA4 (S1[2],  XY2[1],  C2[0], C2[1], S2[1]);
          FullAdder FA5 (S1[3],  XY2[2],  C2[1], C2[2], S2[2]);
          FullAdder FA6 (S1[4],  XY2[3],  C2[2], C2[3], S2[3]);
          FullAdder FA7 (C1[4],  XY2[4],  C2[3], C2[4], S2[4]);

          // Row 3

          HalfAdder HA4  (S2[1],   XY3[0],  C3[0], S3[0]);
          FullAdder FA8  (S2[2],  XY3[1],  C3[0], C3[1], S3[1]);
          FullAdder FA9  (S2[3],  XY3[2],  C3[1], C3[2], S3[2]);
          FullAdder FA10 (S2[4],  XY3[3],  C3[2], C3[3], S3[3]);
          FullAdder FA11 (C2[4],  XY3[4],  C3[3], C3[4], S3[4]);

          // Row 3
```

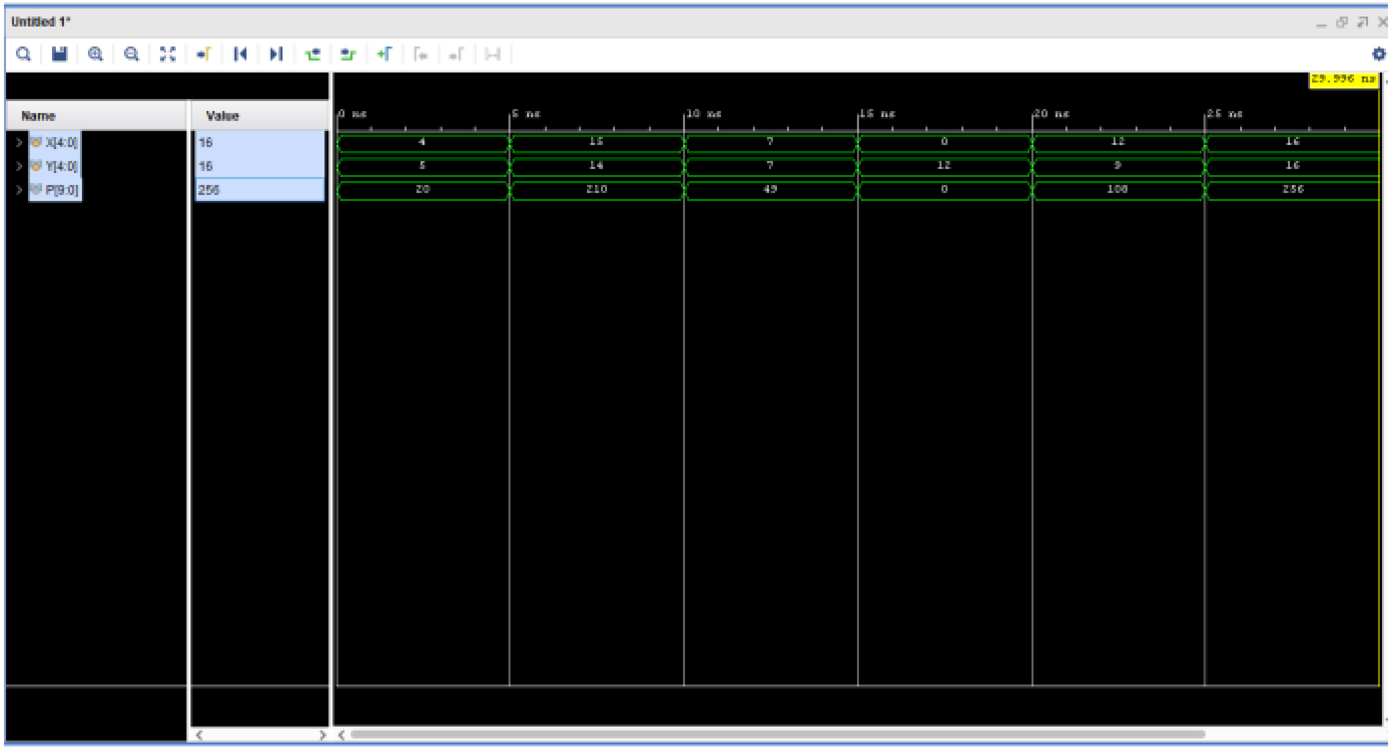
```
        HalfAdder HA5  (S3[1],   XY4[0],  C4[0], S4[0]);
         FullAdder FA12 (S3[2],  XY4[1],  C4[0], C4[1], S4[1]);
        FullAdder FA13 (S3[3],  XY4[2],  C4[1], C4[2], S4[2]);
        FullAdder FA14 (S3[4],  XY4[3],  C4[2], C4[3], S4[3]);
        FullAdder FA15 (C3[4],  XY4[4],  C4[3], C4[4], S4[4]);

        // Assigning Output Product Bits to send the multiplied output.
        assign P[0] = XY0[0];

        assign P[1] = S1[0];
        assign P[2] = S2[0];
        assign P[3] = S3[0];
        assign P[4] = S4[0];
        assign P[5] = S4[1];
        assign P[6] = S4[2];
        assign P[7] = S4[3];
        assign P[8] = S4[4];
        assign P[9] = C4[4];

endmodule
```

3.4. Testbench result of the Array Multiplier Module:

Testbench Code:

```
`timescale 1ns / 1ps
module testMult();
  reg [4:0] X, Y;
  wire [9:0] P;
        arrayMultiplier UUT(.X(X), .Y(Y), .P(P));
        initial begin
                X = 5'h4;
                Y = 5'h5; #5
                X = 5'hF;
                Y = 5'hE; 5
                X = 5'h7;
                Y = 5'h7;
                #5
                X = 5'h0;
                Y = 5'hC;
                #5
                X = 5'hC;
                Y = 5'h9;
                #5
                X = 5'd16;
                Y = 5'd16;
                #5
                $stop();
        end
endmodule
```

3.5. Testbench Waveform:



Name	Value	0 ns	5 ns	10 ns	15 ns	20 ns	25 ns
X[4:0]	16	4	15	7	0	12	16
Y[4:0]	16	5	14	7	12	9	16
P[9:0]	256	20	210	49	0	108	256

Figure 24: Testbench Result of the 5-bit multiplier.

The Following Figure 24 shows the result of the 5-bit multiplier.

The result is what we expected and it is running as intended. There is no issue with the multiplier.

This concludes the conversion of 4-bit multiplier into 5-bit multiplier.