VLSI DESIGN LAB

R20 Regulation

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING,

Lakireddy Bali Reddy College of Engineering (AUTONOMOUS),

L.B.Reddy Nagar, MYLAVARAM – 521230.

VLSI DESIGN LAB

LIST OF EXPERIMENTS

PART-1: VLSI FRONT END DESIGN USING XILINX TOOL:

1. Implementation of Carry-Look-Ahead adder.
2. Implementation of 4 X 4 Array Multiplier.
3. Implementation of a 4-bit ALU.
4. Implementation of Zero /One Detector.
5. Implementation of flip flops: SR, D, JK, T.

PART-2: VLSI BACK END DESIGN USING CADENCE/MENTOR GRAPHICS TOOLS:

PART-2.1: Full Custom Design:

1. Design and analysis of NMOS Inverter.
2. Design and analysis of CMOS Inverter
3. Design and analysis of CMOS NOR gate.
4. Design and analysis of CMOS NAND gate.
5. Design and analysis of CMOS D- Flip Flop

PART-2.2: Semi Custom Design

1. Design and analysis of Full Adder
2. Design and analysis of Decoder
3. Design and analysis of 8- bit Binary Counter
4. Design and analysis of Shift Register
5. Design and analysis of Sequence Detector Note: Minimum of 3 experiments from part-1 and

Implementation of Carry-Look-Ahead adder

EXP NO -1

DATE:

AIM: To design and simulate carry-look- a head adder using Xilinx's VIVADO and its implementation on Zed board Evaluation and Development Kit

COMPONENTS & TOOLS REQUIRED:

Target devices: Xilinx Zynq-7000- Zed board Evaluation and Development Kit/Zybo board

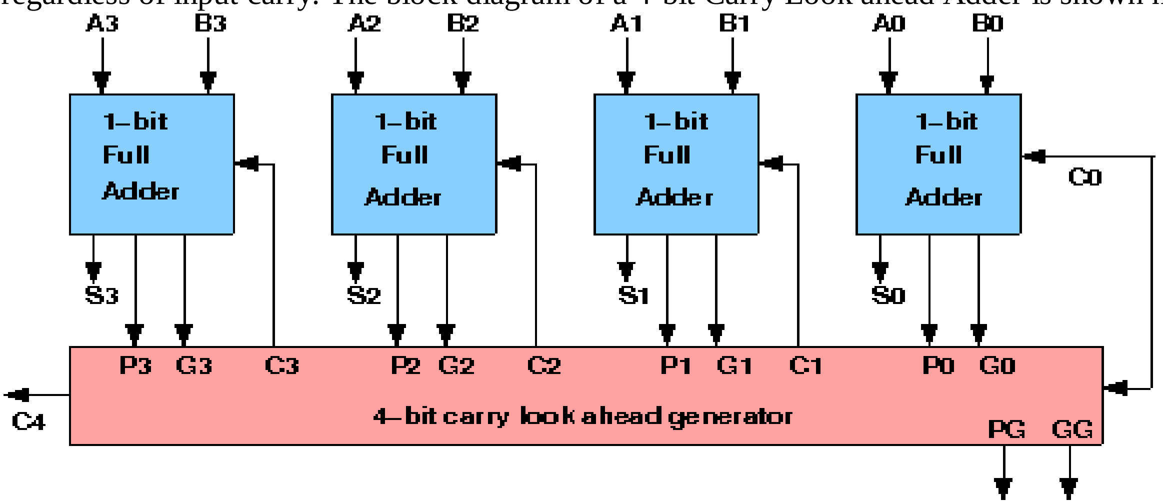
Tools: Xilinx VIVADO suite

Preferred language- Verilog

THEORY:

DESIGN OF CARRY LOOKAHEAD ADDERS :

To reduce the computation time, there are faster ways to add two binary numbers by using carry lookahead adders. They work by creating two signals P and G known to be Carry Propagator and Carry Generator. The carry propagator is propagated to the next level whereas the carry generator is used to generate the output carry, regardless of input carry. The block diagram of a 4-bit Carry Look ahead Adder is shown here below



The corresponding boolean expressions are given here to construct a carry lookahead adder. In the carry-look ahead circuit we need to generate the two signals carry propagator(P) and carry generator(G),

$\mathbf{P_i = A_i \oplus B_i}$ ….(1)

$\mathbf{G_i = A_i \cdot B_i}$ ….(2)

The output sum and carry can be expressed as

$\mathbf{Sum_i = P_i \oplus C_i}$ ….(3)

$\mathbf{C_{i+1} = G_i + (P_i \cdot C_i)}$ ….(4)

Using equation(4) the Boolean function for the carry output of each stage is obtained as

$\mathbf{C_1 = G_0 + P_0 \cdot C_0}$

$\mathbf{C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0}$

$\mathbf{C_3 = G_2 + P_2 \cdot C_2 = G_2\ P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0}$

$\mathbf{C_4 = G_3 + P_3 \cdot C_3 = G_3\ P_3 \cdot G_2\ P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0}$

The carry-look ahead 4-bit adder can also be used in a higher-level circuit by having each CLA logic circuit produce a propagate and generate signal to a higher-level CLA logic circuit.

VERILOG PROGRAM:

```
module Carry_look_ahead(
    input [3:0] a,b,
    output [4:0] s
    );
    wire [4:0] c;
    wire [3:0] g,p,sum;
    // Generate signals
    assign g[0] = a[0] & b[0],g[1] = a[1] & b[1],g[2] = a[2] & b[2],g[3] = a[3] & b[3];
    // Propagate signals
    assign p[0]=a[0]^b[0],p[1]=a[1]^b[1],p[2]=a[2]^b[2],p[3]=a[3]^b[3];
    // Create the carry terms
    assign c[0]=1'b 0;
     assign c[1]= g[0] |(p[0] &c[0]);
      assign c[2]=g[1] |(p[1] &c[1]);
       assign c[3]=g[2] |(p[2] &c[2]);
       assign c[4]=g[3] |(p[3] &c[3]);
full_adder fa1(a[0],b[0],c[0],sum[0]),
               fa2(a[1],b[1],c[1],sum[1]),
               fa3(a[2],b[2],c[2],sum[2]),
               fa4(a[3],b[3],c[3],sum[3]);
 assign s= {c[4],sum};
endmodule
// Full adder module
module full_adder(
    input a,
    input b,
    input cin,
    output y,
    output co
    );
    assign y = a^b^cin, co = a&b|b&cin|cin&a;
endmodule
```

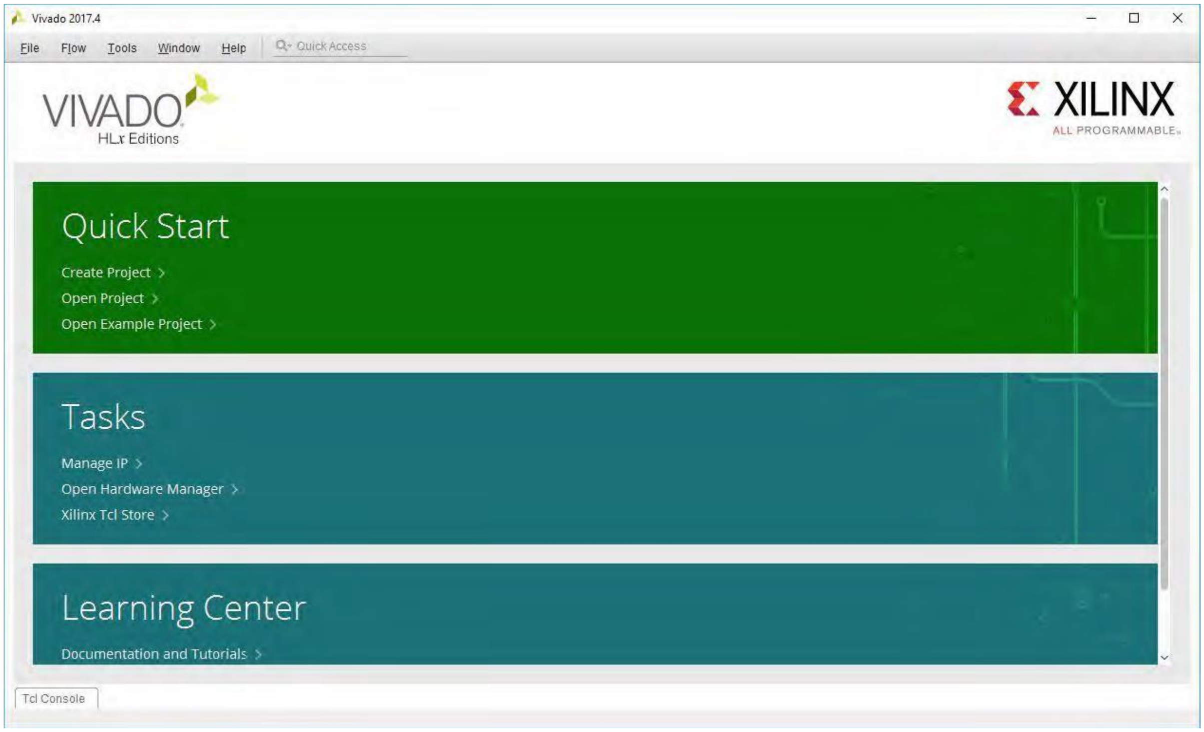
TEST BENCH PROGRAM:

```
module test_cla1( );
    reg [3:0] a,b;
    wire [4:0] s;
    Carry_look_ahead l1(a,b,s);
    initial
    begin
    a = 4'h 0;
    b= 4'h 0;
    end
    always
```

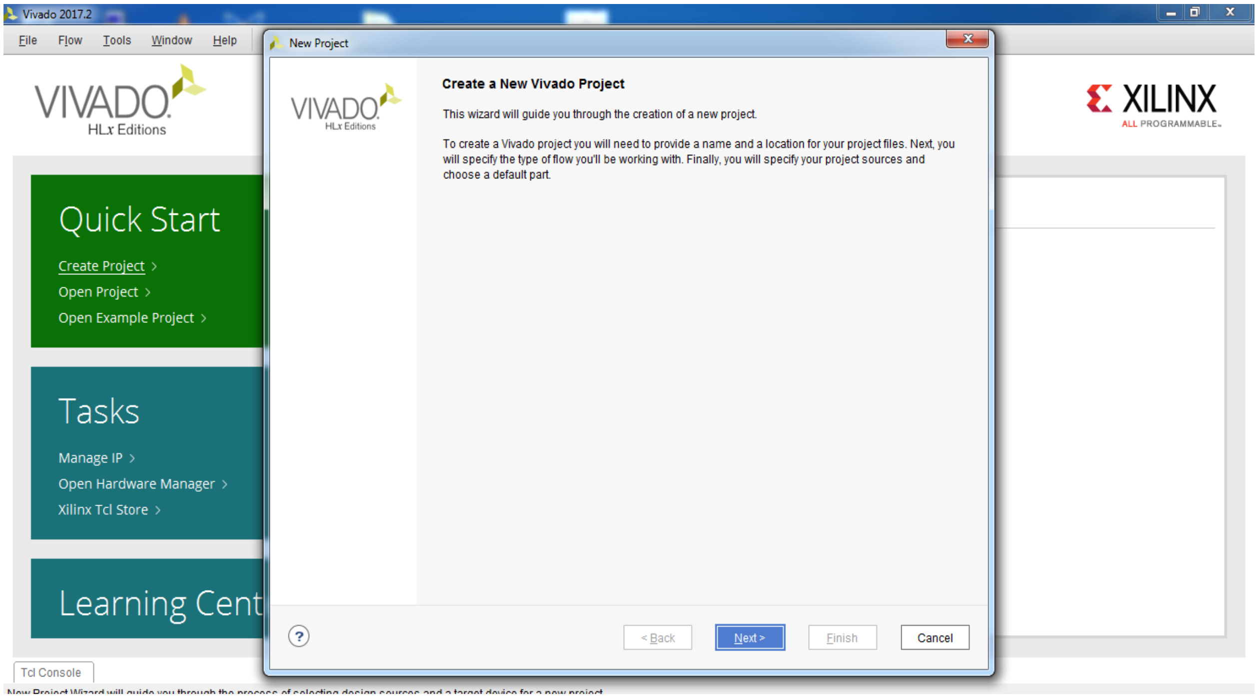
```
begin
#3 a=a+4'h 1;
#3 b = b + 4'h 1;
end
endmodule
```

PROCEDURE:

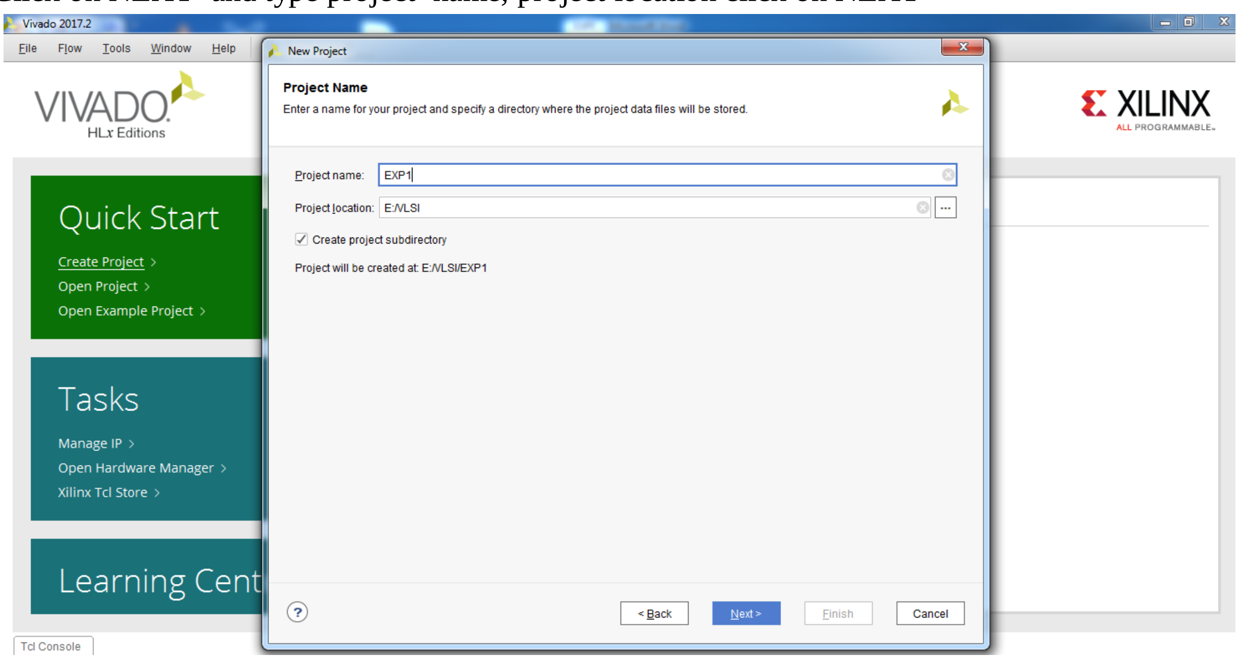
1. Double click on the Vivado2017.2 icon on your desktop to open up the welcome window of the development tool (as shown below). Three main sections can be observed in this window: “Quick Start”, “Tasks”, and “Learning Center”.



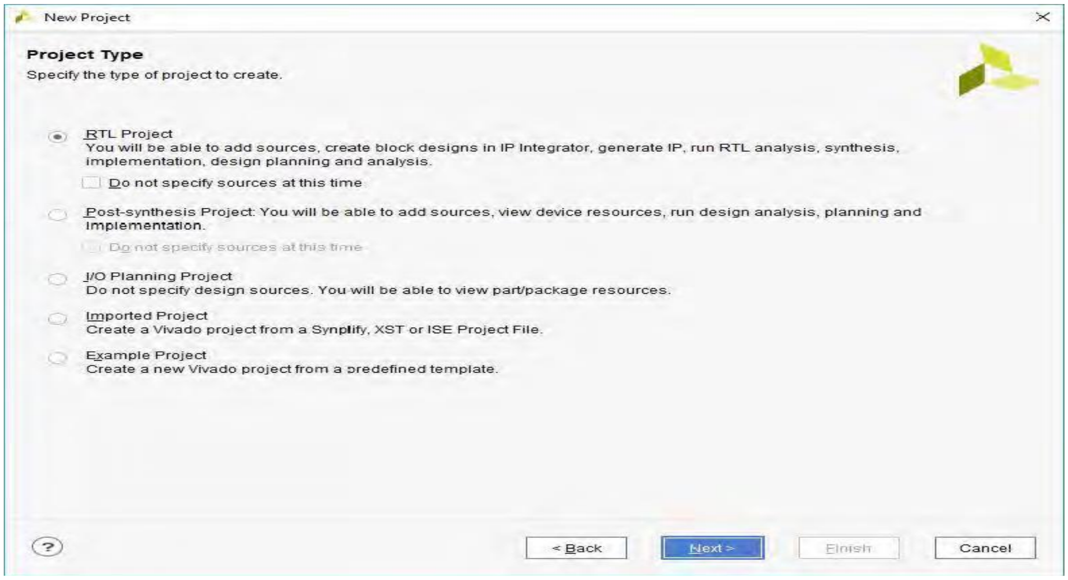
2. Now, click on “Create Project” to create a new project. You have to be careful about where to save your project file .



3. Click on NEXT and type project name, project location click on NEXT

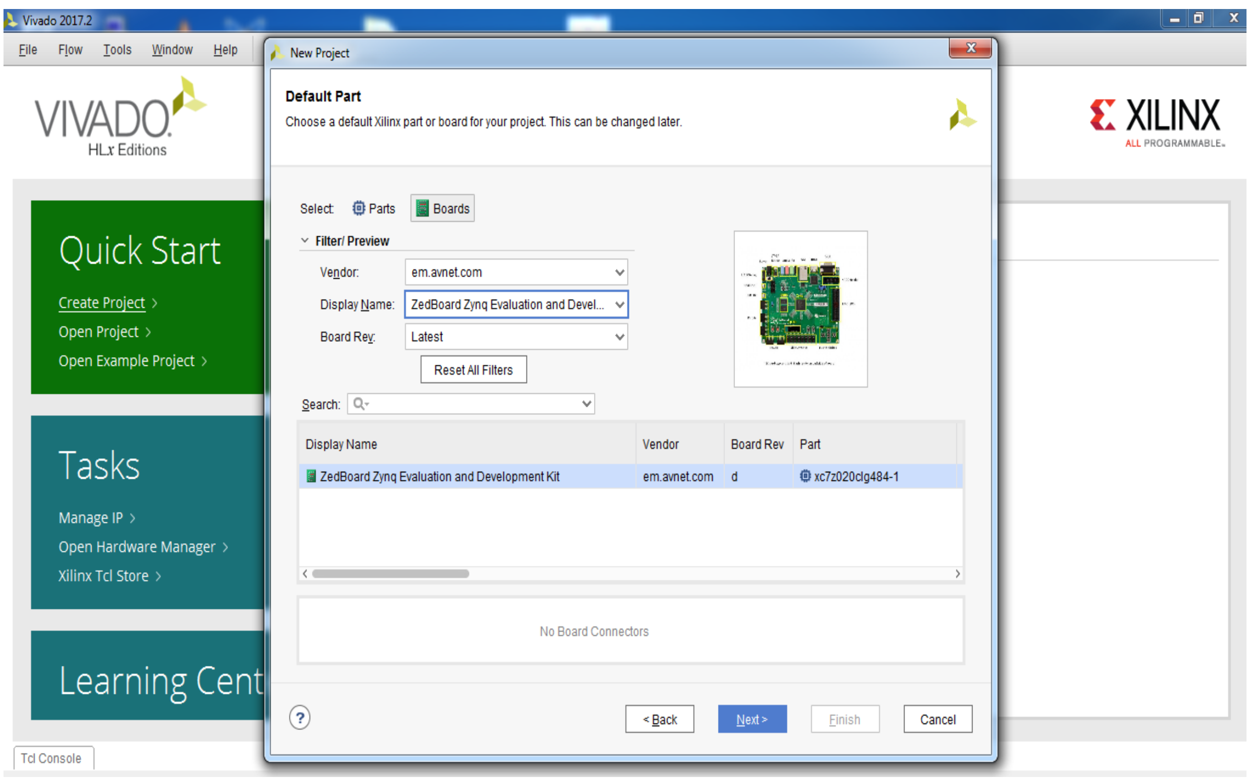


4. In the next window, choose "RTL Project" as the project type. (click on select button), click on NEXT

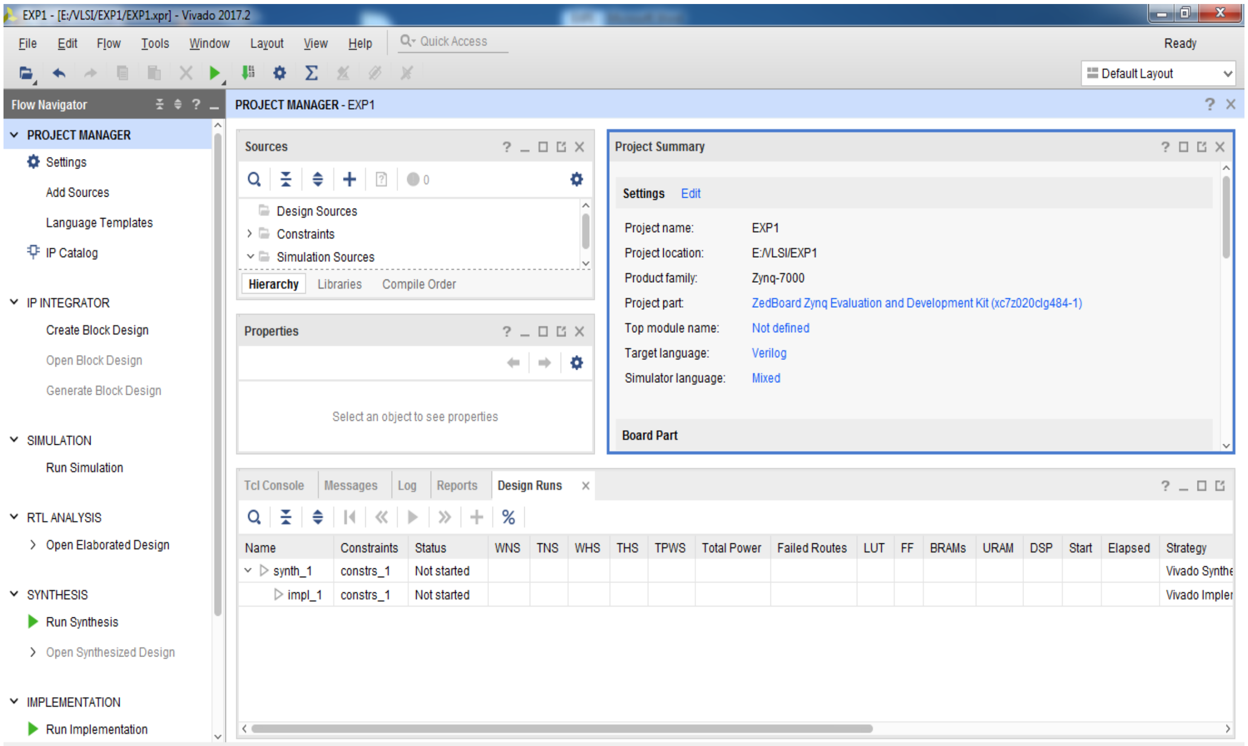


5. Click on Boards:
   i. vendor:em.avnet.com
   ii. Display Name: Zed board Evaluation and Development Kit.
   iii. Board Rev : Latest

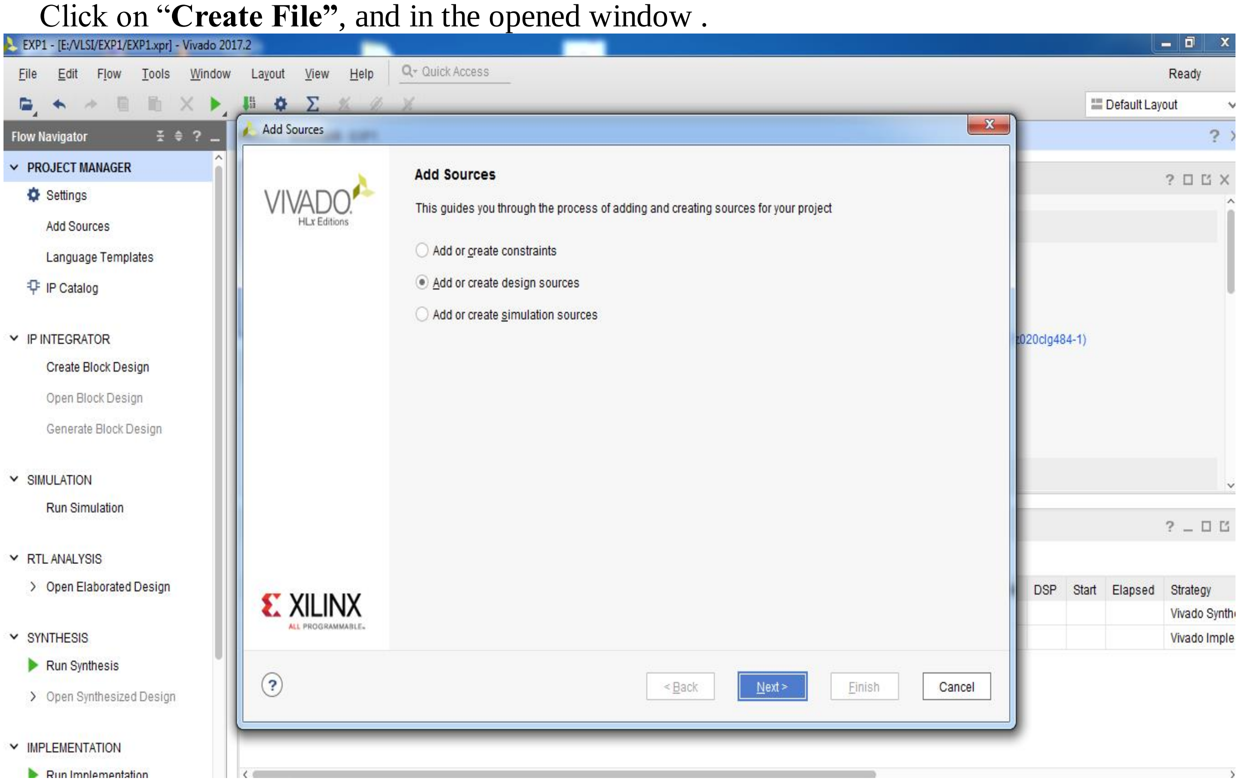
   click on NEXT,FINISH



6. Click on plus symbol(Add source)

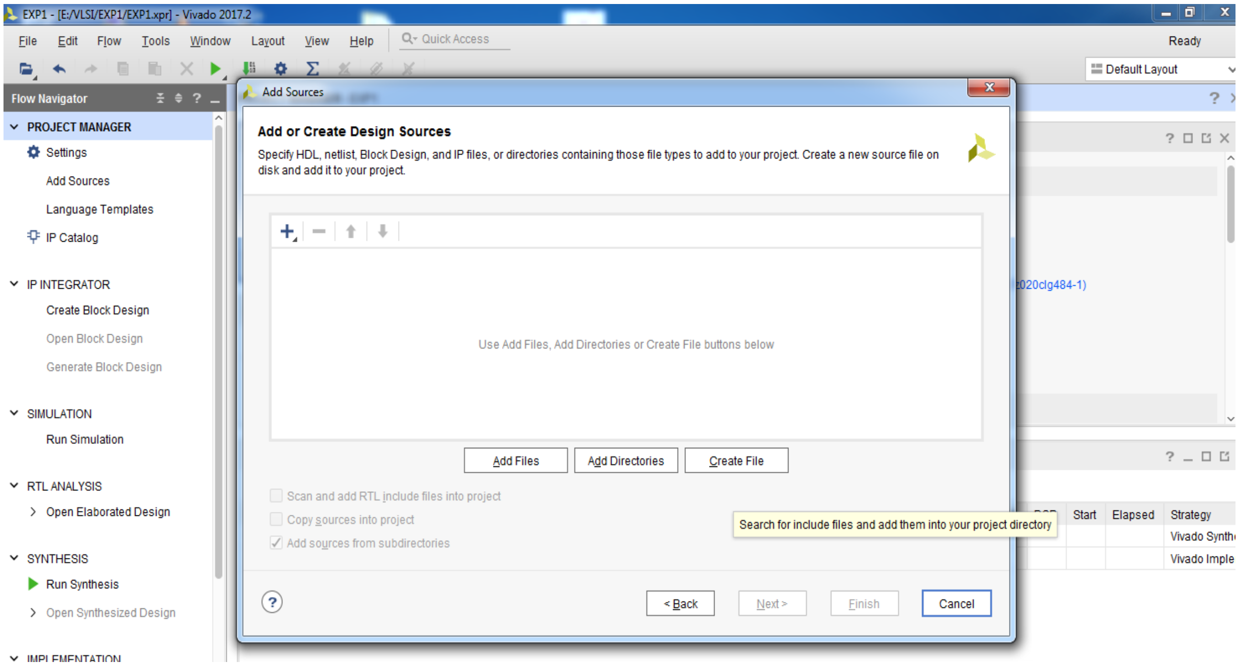


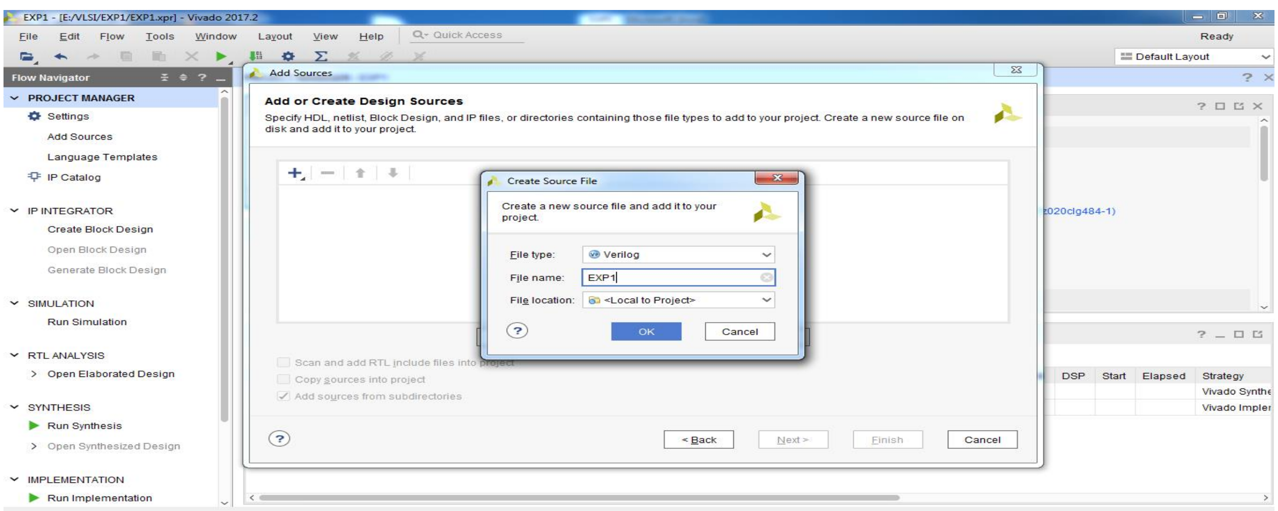
7. Click on **Add or create design source and NEXT** .In the opened window, you can create source file (Verilog/Verilog Header/SystemVeilog) for your new project or add sources from the existing projects. Click on "**Create File**", and in the opened window .



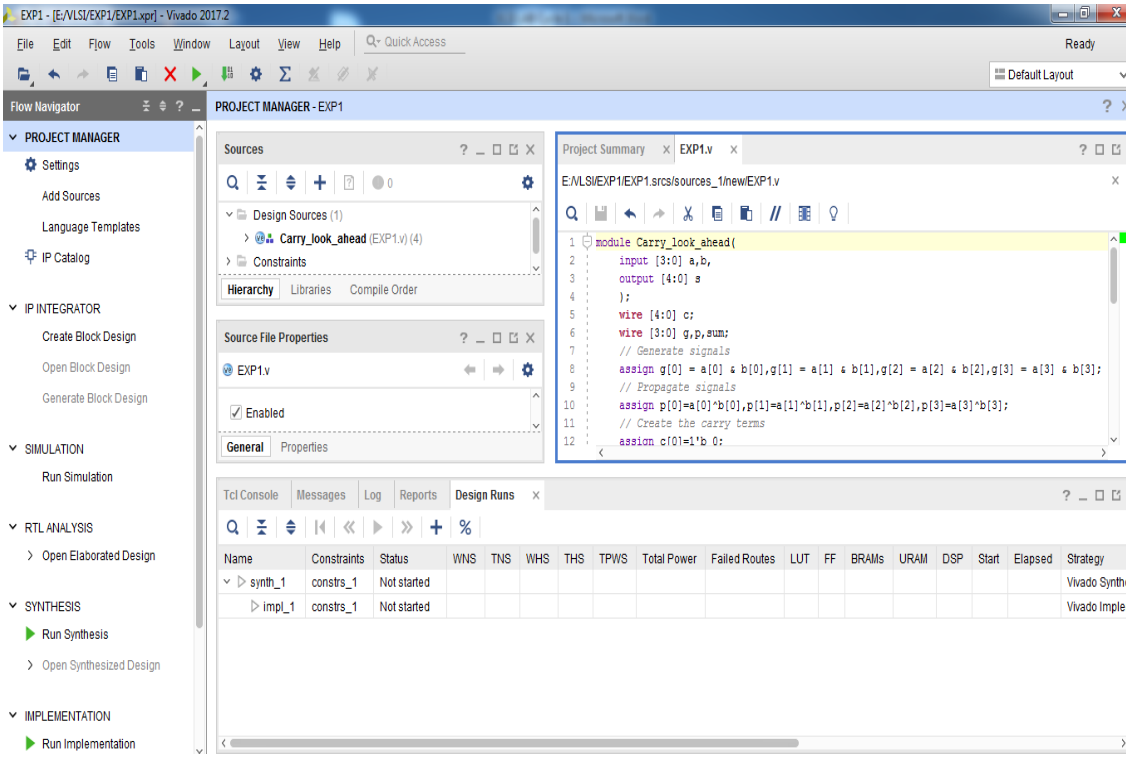
8. **Click on Create file and type the file name**

The opened window is the main environment for your project that is called "Project Manager". You can explore it by seeing the options of each category in the toolbar on top of the window. In the left side, you can see the "Settings", "Add Sources", ""Language Template", "IP Catalog", "IP Integrator", "Simulation", "RTL Analysis", "Synthesis", "Implementation", and "Program and Debug". Each of these serves a part of the digital design flow. In the middle, you can see the windows for “Sources”, “Properties”, “Project Summary”, and the reports and summaries for the execution of the project files.

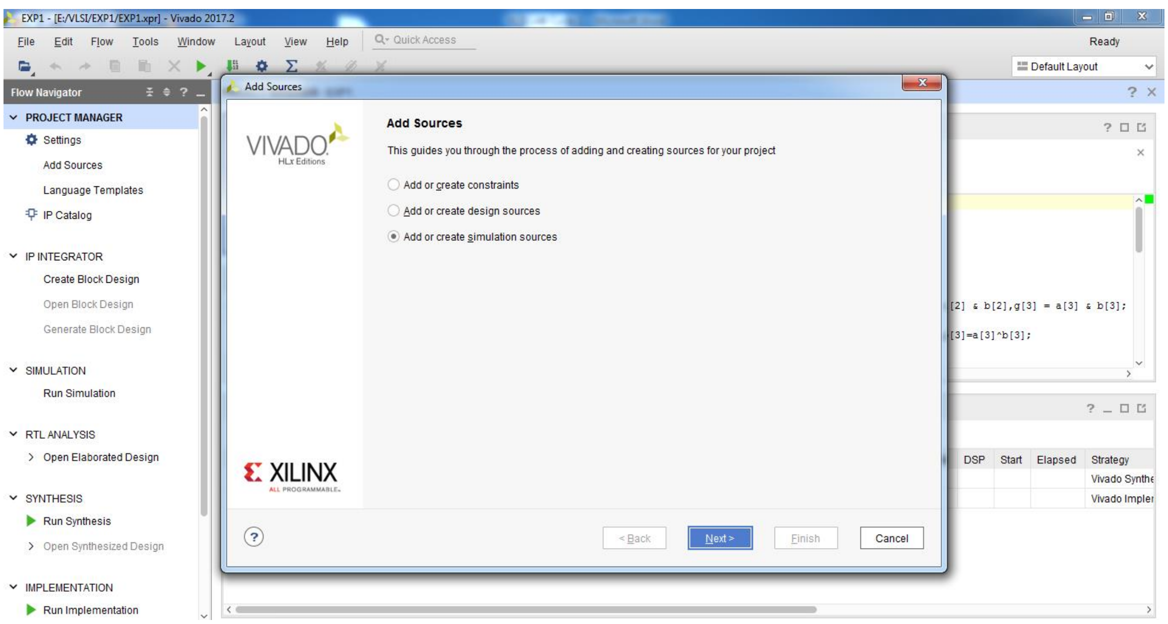




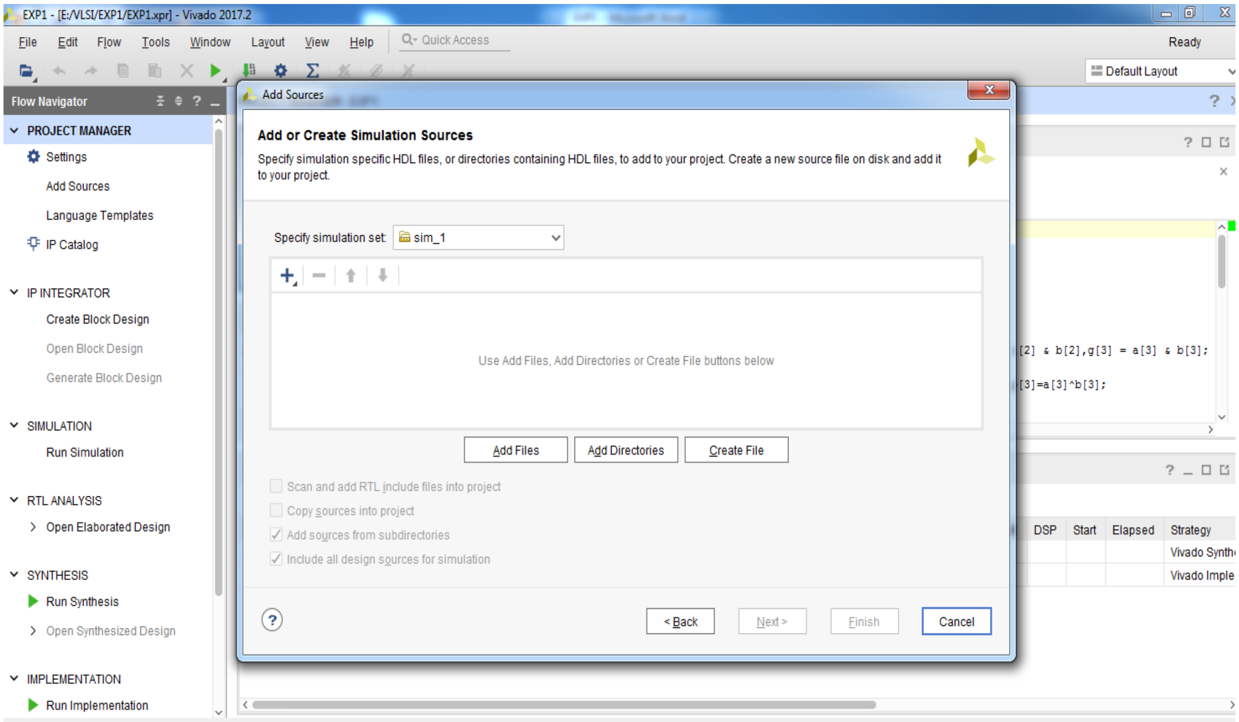
9. Type the program

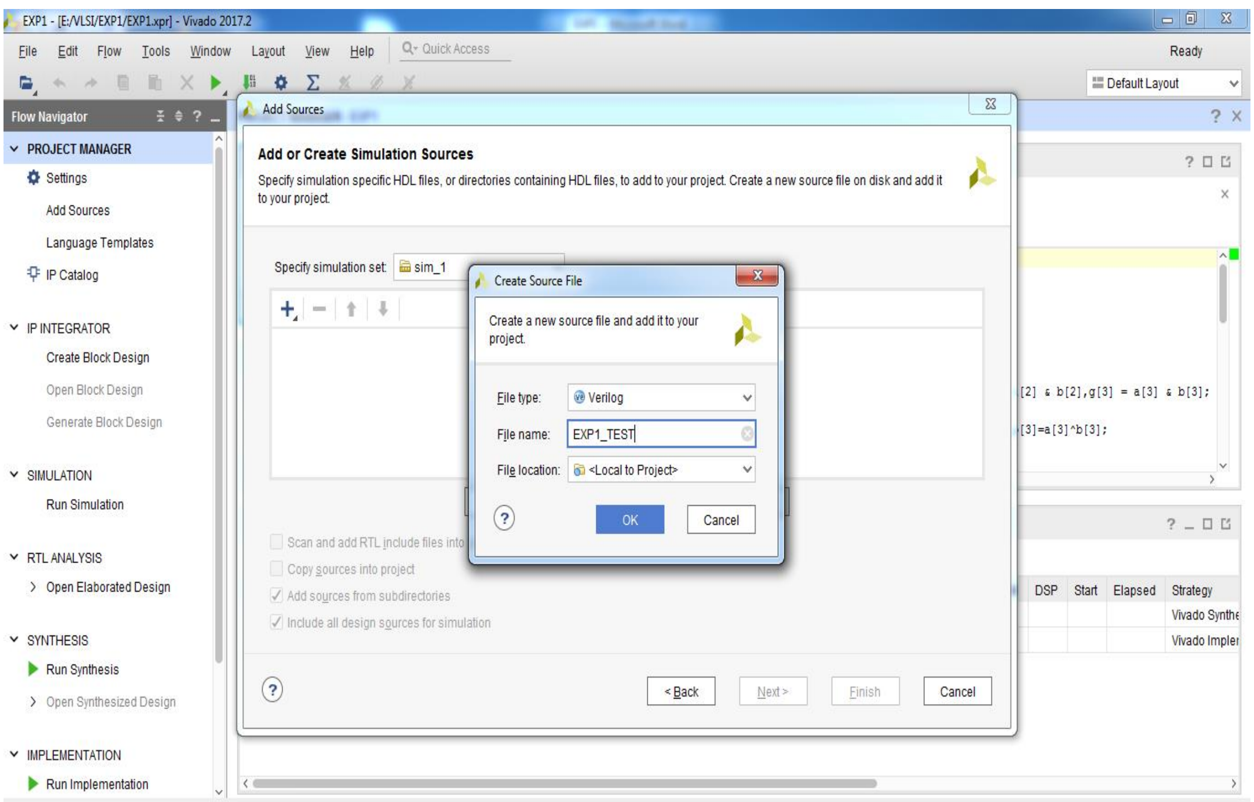


10. Click on plus button(Add sources) and select Add or create simulation sources and NEXT

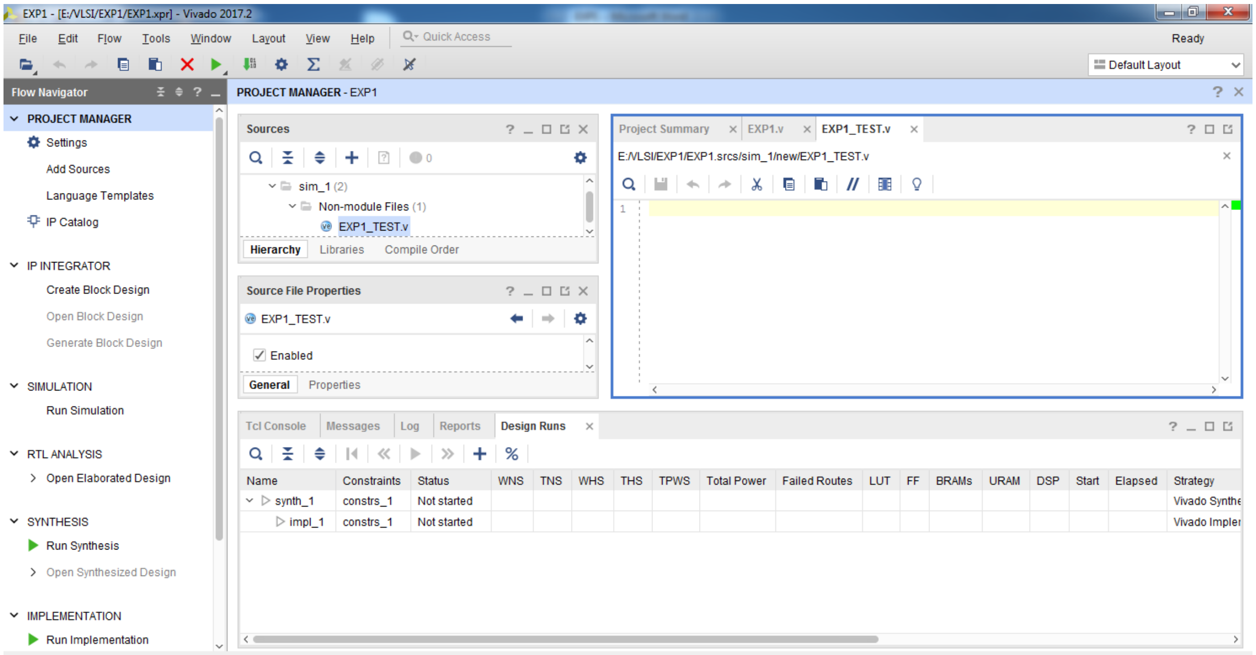


11. Click on the Create file and type the file name

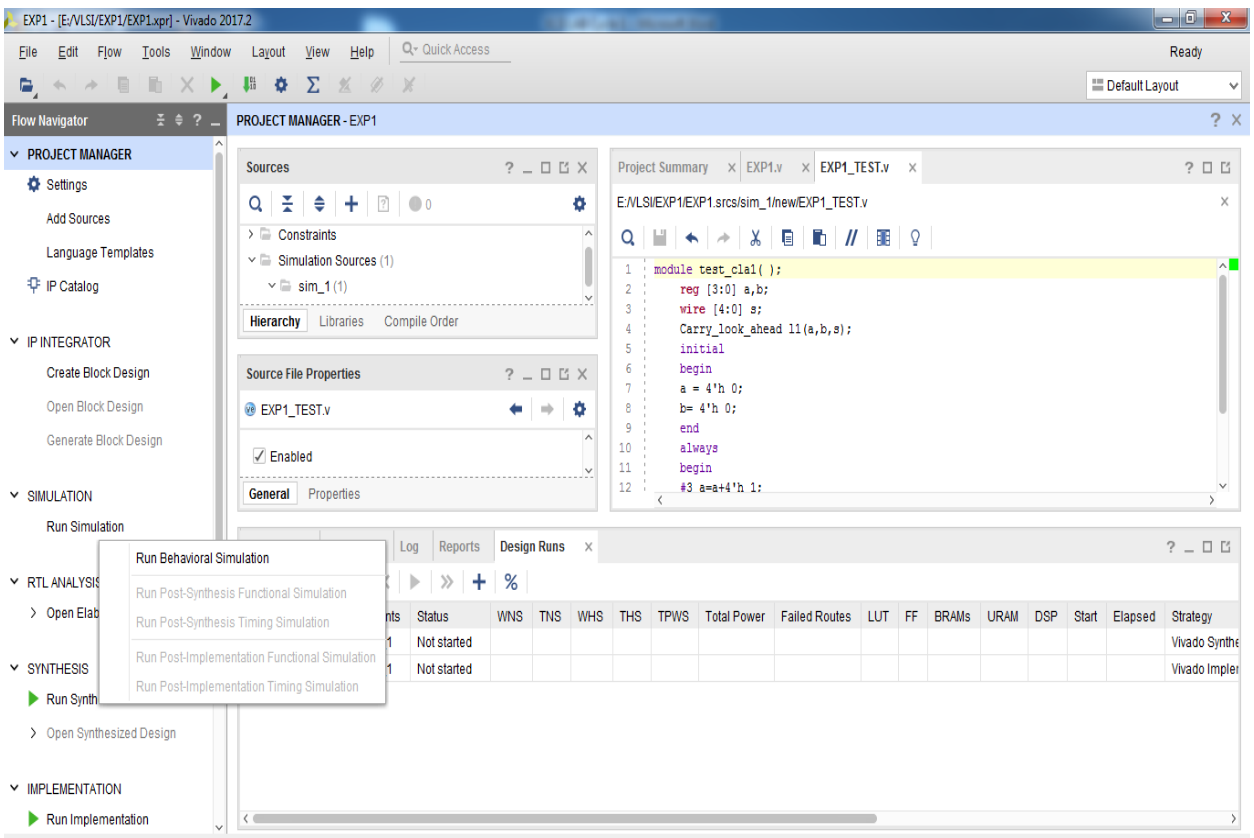




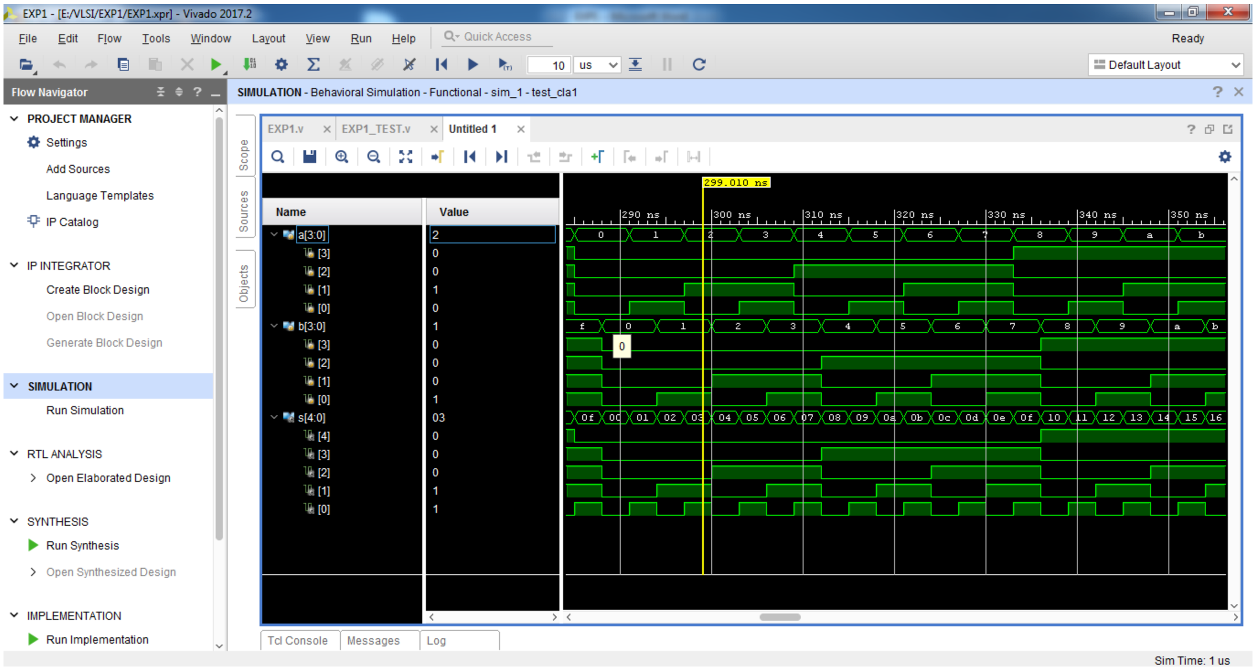
12. Type the test bench program and save



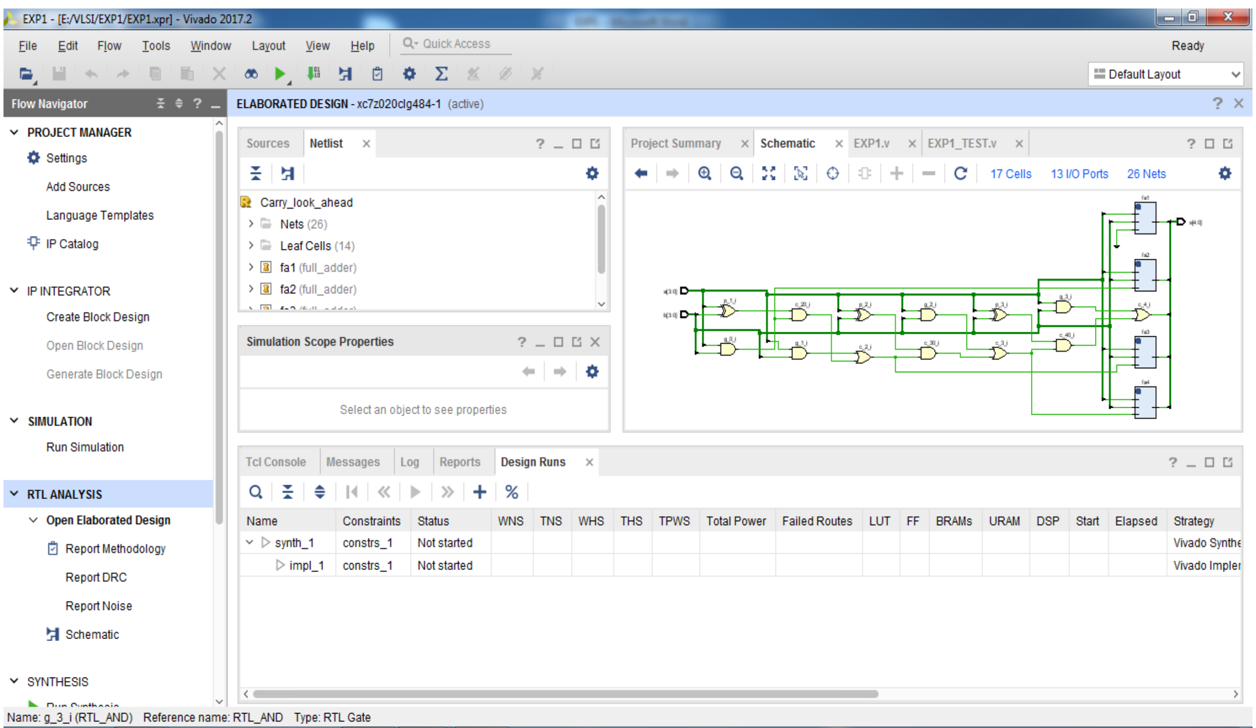
13. Run simulation -> click on Run behavioral simulation



14. **SIMULATION OUTPUT:**



15.Click on RTL Analysis -> open elaborate designs(click on I/O ports)

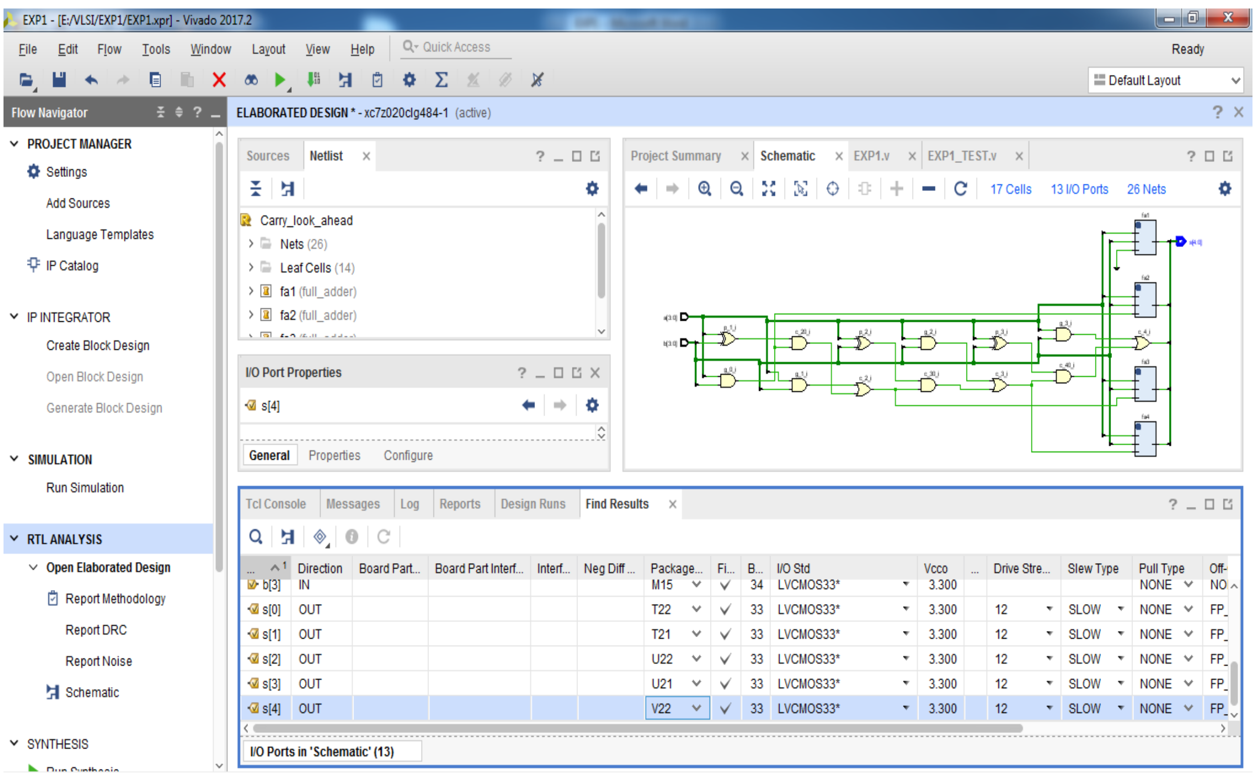


16. Assign port packages(assign pin number) and I/O std (select LVCMOS33)

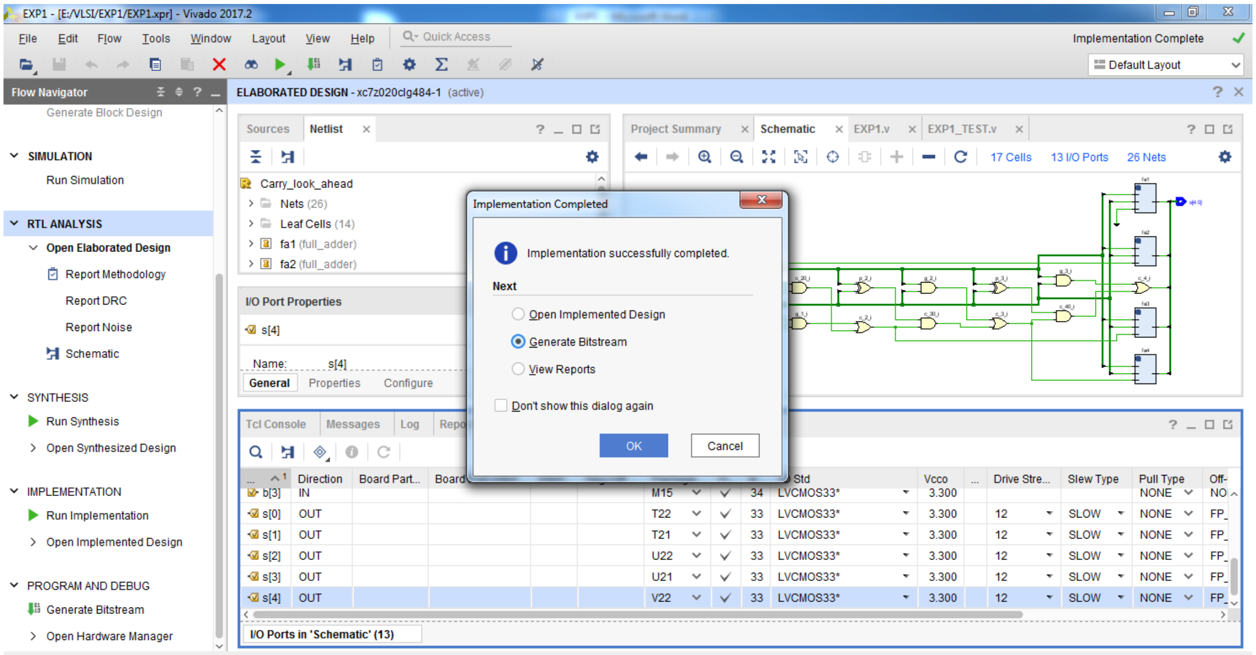
INPUT PIN NUMBERS	OUTPUT LED PIN NUMBERS

SW0	F22	LD0	T22
SW1	G22	LD1	T21
SW2	H22	LD2	U22
SW3	F21	LD3	U21
SW4	H19	LD4	V22
SW5	H18	LD5	W22
SW6	H17	LD6	U19
SW7	M15	LD7	U14

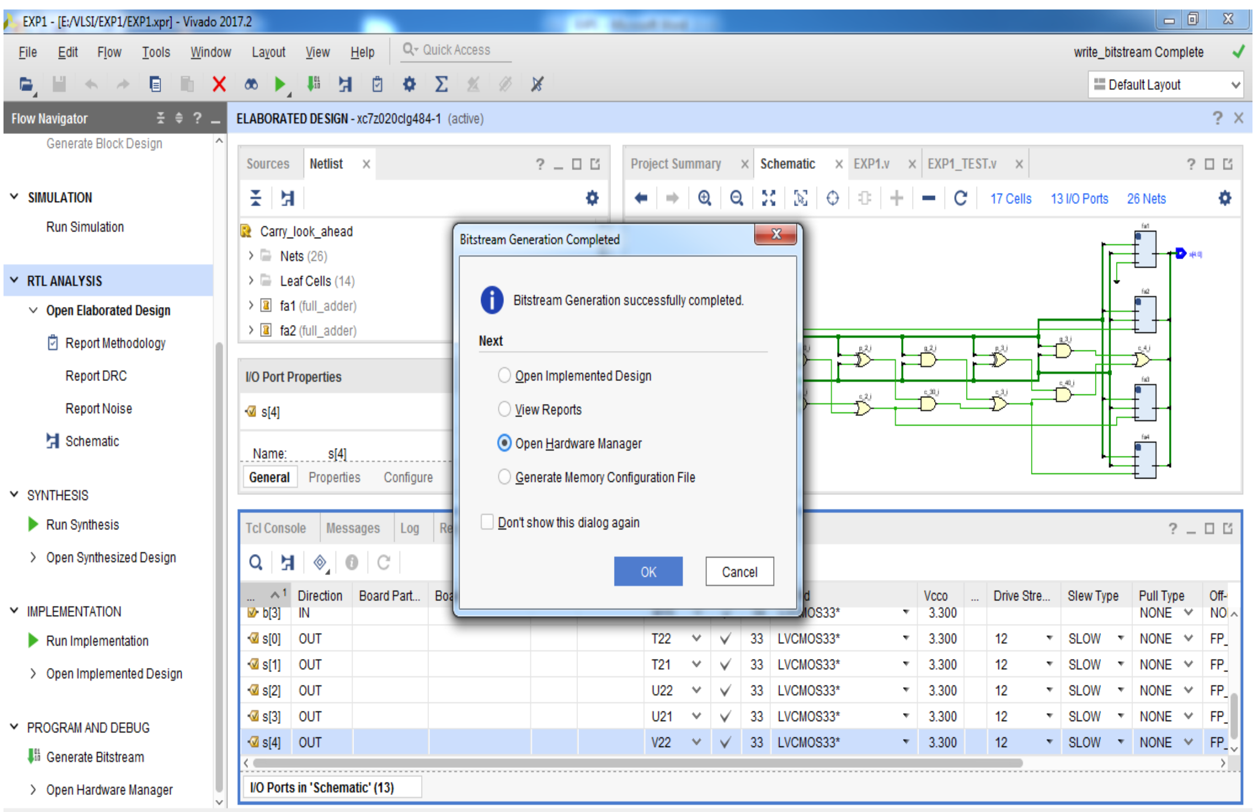
17. Save and type the XDC File name



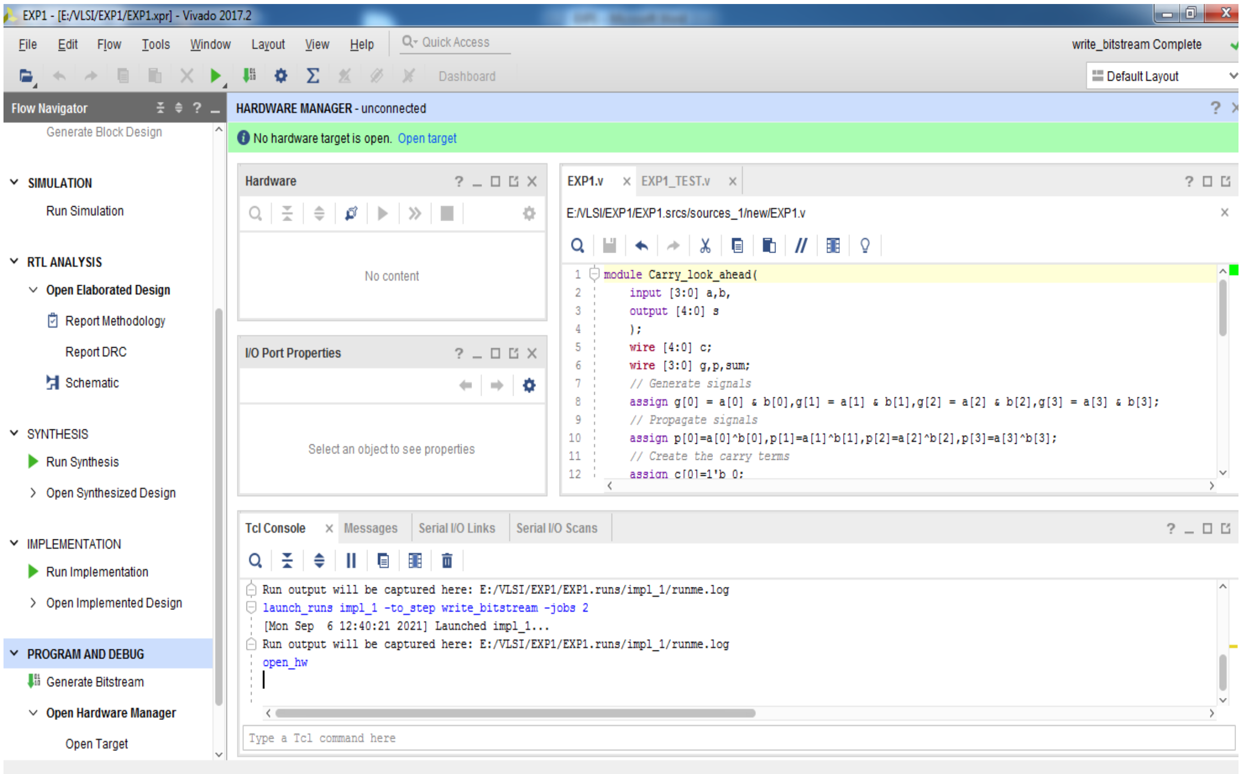
18. Run the Implementation and select generate bit stream click OK



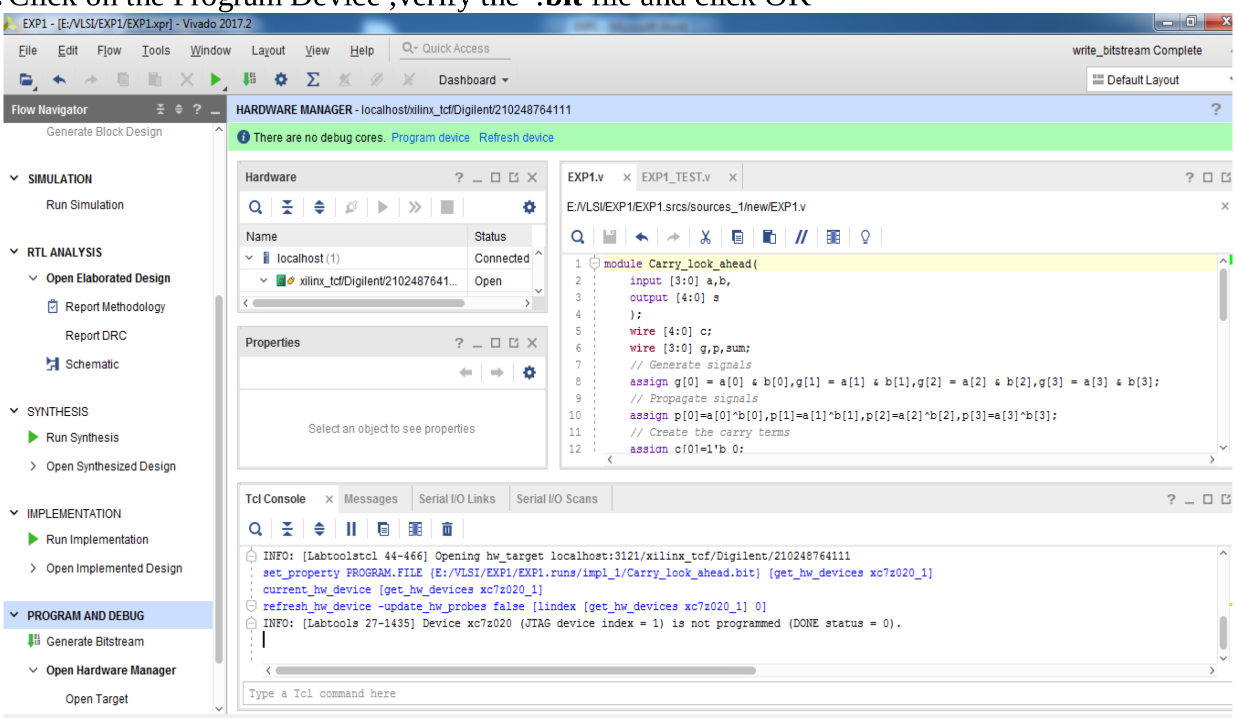
19.Select the Open Hardware Manager and click on OK

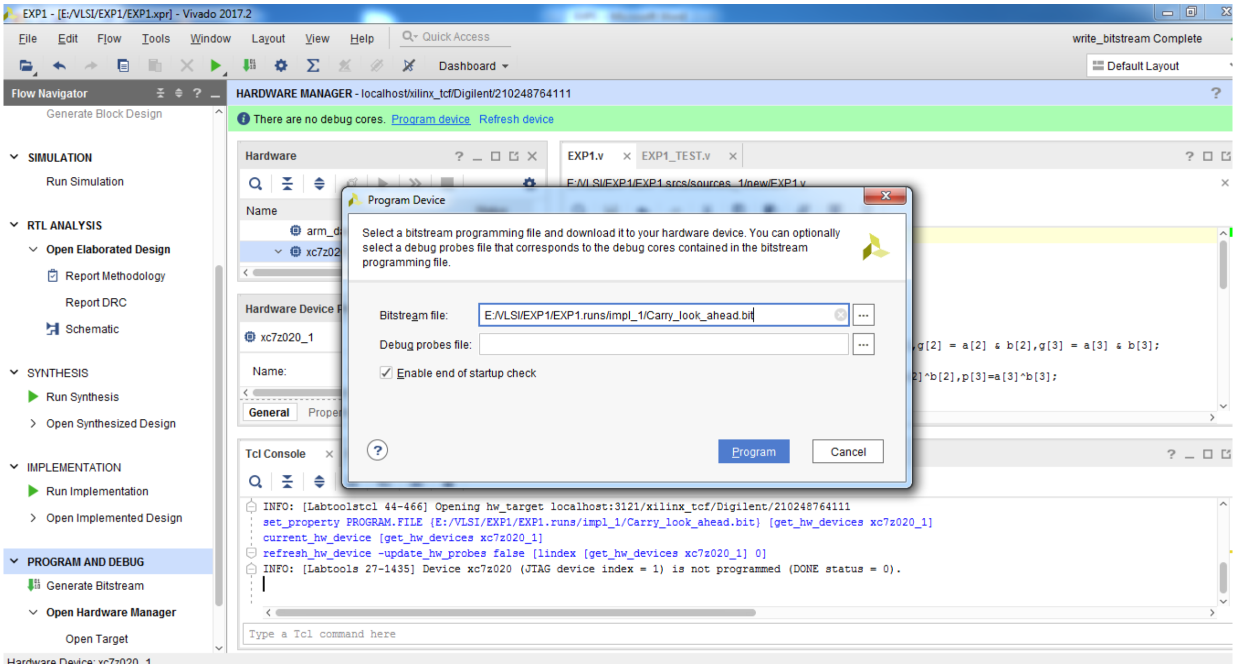


20.Connect the Hardware kit (Ex: ZedBoard) and Click on Open Target -> auto connect

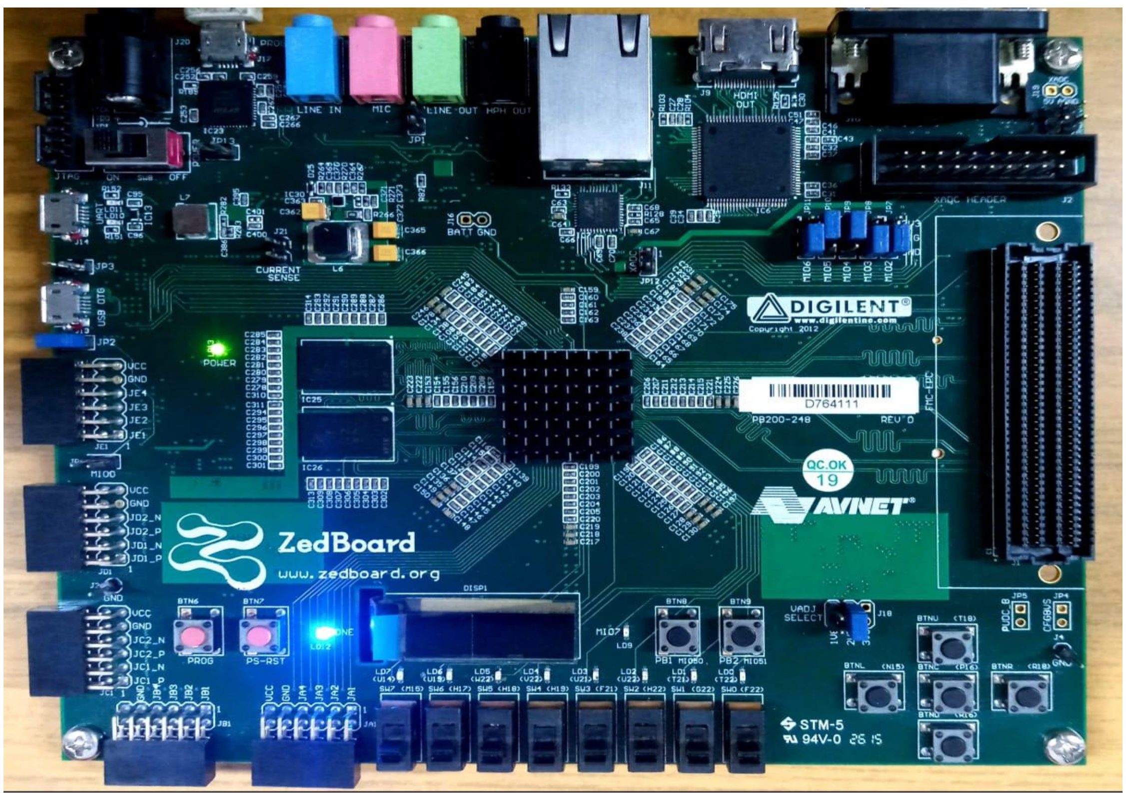


21. Click on the Program Device ,verify the **.bit** file and click OK





22.Verify the function table on Zedboard.



CONCLUSION: Hence a carry look-ahead adder is designed and implemented on Zed board using Xilinx Vivado2017.2

PRECAUTIONS:

1. Give connections carefully such as Zed Board ,JTAG Power cable, power supply etc.
2. Switch on the power supply.
3. Handle the Zed Board carefully.
4. Check pin configuration before configuring the Target Device.

VIVA Questions:

1. List the disadvantages of ripple adder.
2. Mention the advantages of carry look ahead adder.
3. Define the Pi,Gi signals in CLA?
4. Why carry look-ahead adder is faster?
5. List out carry expressions in CLA.

# IMPLEMENTATION OF 4X4 ARRAY MULTIPLIER	**EXP NO-2**
	DATE:

AIM: To design and simulate 4 X 4 Array Multiplier using Xilinx's VIVADO and its implementation on Zed board Evaluation and Development Kit

COMPONENTS & TOOLS REQUIRED:

Target devices: Xilinx Zynq-7000- Zed board Evaluation and Development Kit/Zybo board

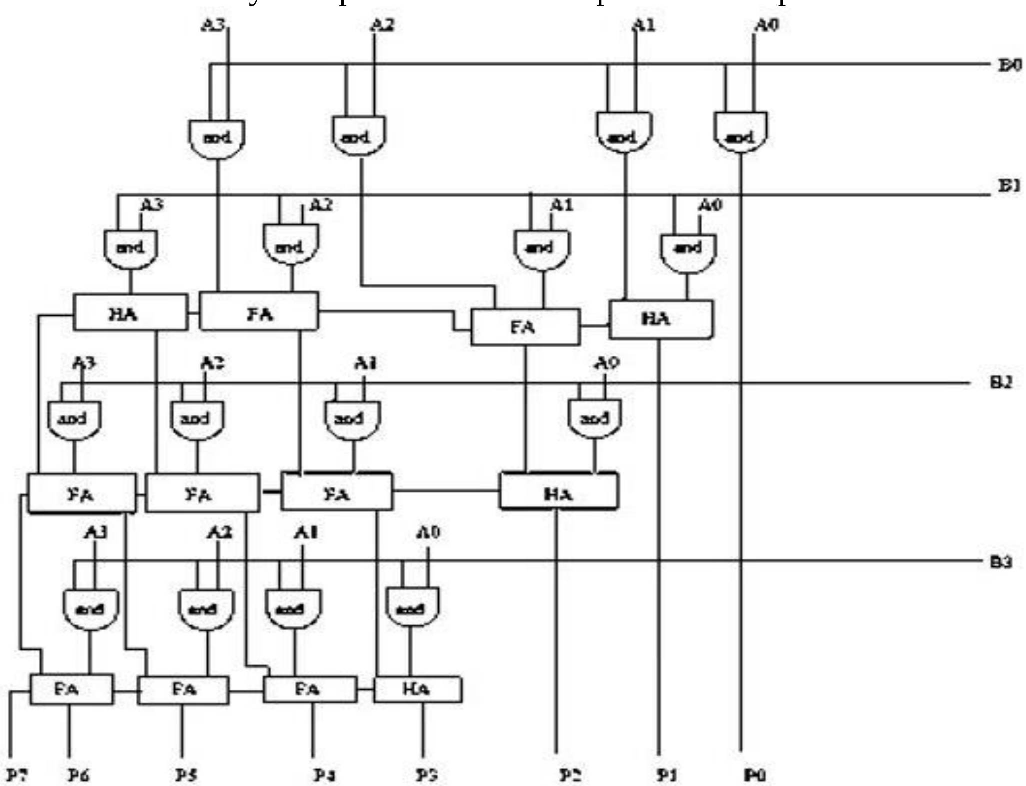
Tools: Xilinx VIVADO suite

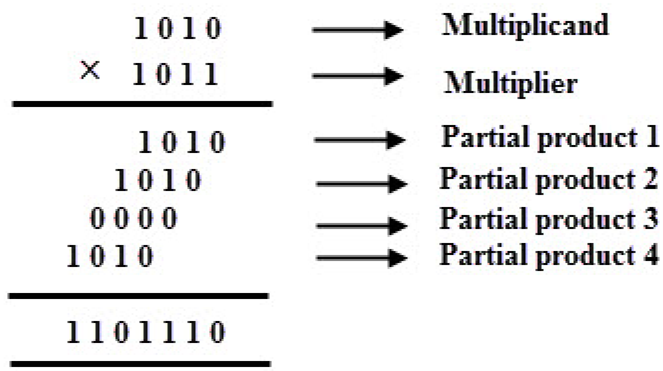
Preferred language- Verilog

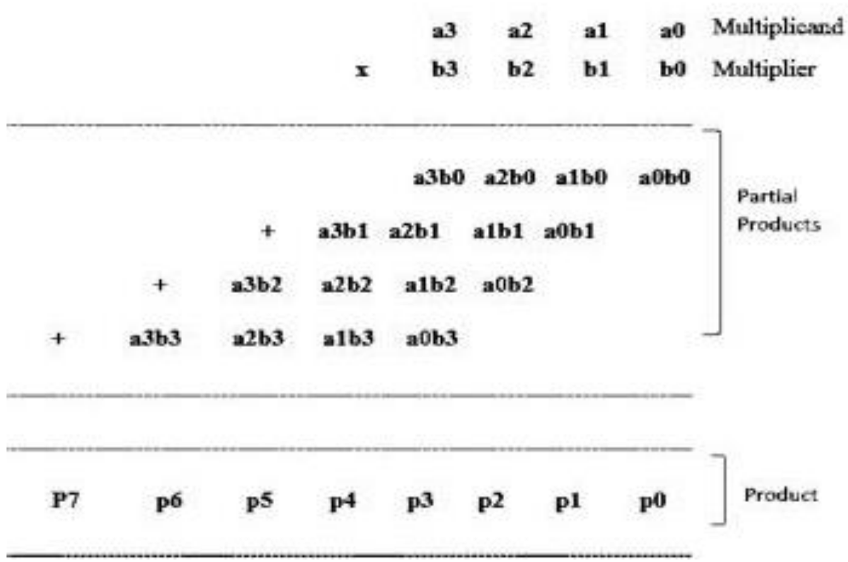
THEORY: The design structure of the array Multiplier is regular, it is based on the add shift algorithm principle.

Partial product = the multiplicand * multiplier bit

where AND gates are used for the product, the summation is done using Full Adders and Half Adders where the partial product is shifted according to their bit orders. In an n*n array multiplier, n*n AND gates compute the partial products and the addition of partial products can be performed by using n* (n – 2) Full adders and n Half adders. The 4×4 array multiplier shown has 8 inputs and 8 outputs







VERILOG CODE:

```
module array_mult_4x4(
   input [3:0] a,
   input [3:0] b,
   output [7:0] p
   );
   wire [15:0] pp; wire [9:0] psum;
   and g1(pp[0],a[0],b[0]),
   g2(pp[1],a[1],b[0]),
```

```
    g3(pp[2],a[2],b[0]),
    g4(pp[3],a[3],b[0]),
    g5(pp[4],a[0],b[1]),
    g6(pp[5],a[1],b[1]),
    g7(pp[6],a[2],b[1]),
    g8(pp[7],a[3],b[1]),
    g9(pp[8],a[0],b[2]),
    g10(pp[9],a[1],b[2]),
    g11(pp[10],a[2],b[2]),
    g12(pp[11],a[3],b[2]),
    g13(pp[12],a[0],b[3]),
    g14(pp[13],a[1],b[3]),
    g15(pp[14],a[2],b[3]),
    g16(pp[15],a[3],b[3]);
    adder_4bit  a1({1'b 0,pp[3:1]}, pp[7:4], psum[4:0]),
                a2(psum[4:1],pp[11:8],psum[9:5]),
                a3(psum[9:6],pp[15:12],p[7:3]);
          assign  p[2:0] = {psum[5],psum[0],pp[0]};
endmodule
```

4-BIT PARALLEL ADDER

```
module adder_4bit(
    input [3:0] x,y,
    output [4:0] s
    );
    wire [4:1] c; wire [3:0] sum; supply0 gnd;
    full_adder fa1(x[0],y[0],gnd, sum[0],c[1]),
               fa2(x[1],y[1],c[1],sum[1],c[2]),
               fa3(x[2],y[2],c[2],sum[2],c[3]),
               fa4(x[3],y[3],c[3],sum[3],c[4]);
           assign s = {c[4],sum};
endmodule
full-adder
module full_adder(
    input a,b,cin,
    output s,co
    );
    assign s=a^b^cin,co=(a&b)|(b&cin)|(cin&a);
endmodule
```

<u>**TEST BENCH FOR 4X4 ARRAY MULTIPLIER:**</u>

```
module tst_array(    );
    reg [3:0] a,b;
    wire [7:0] p;
    array_mult_4x4 l1(a,b,p);
    initial
    begin
    a = 4'h 0;
    b= 4'h 0;
    end
    always
```

```
begin
#3 a=a+4'h 1;
#3 b = b + 4'h 1;
end
endmodule
```

PROCEDURE:

1. Double click on the Vivado2017.2 icon on your desktop to open up the welcome window of the development tool (as shown below). Three main sections can be observed in this window: “Quick Start”, “Tasks”, and “Learning Center”.
2. . Now, click on “Create Project” to create a new project. You have to be careful about where to save your project file .
3. Click on NEXT and type project name, project location click on NEXT
4. In the next window, choose “RTL Project” as the project type. (click on select button), click on NEXT
5. Click on Boards:
   i. vendor:em.avnet.com
   ii. Display Name: Zed board Evaluation and Development Kit.
   iii. Board Rev : Latest, click on NEXT,FINISH

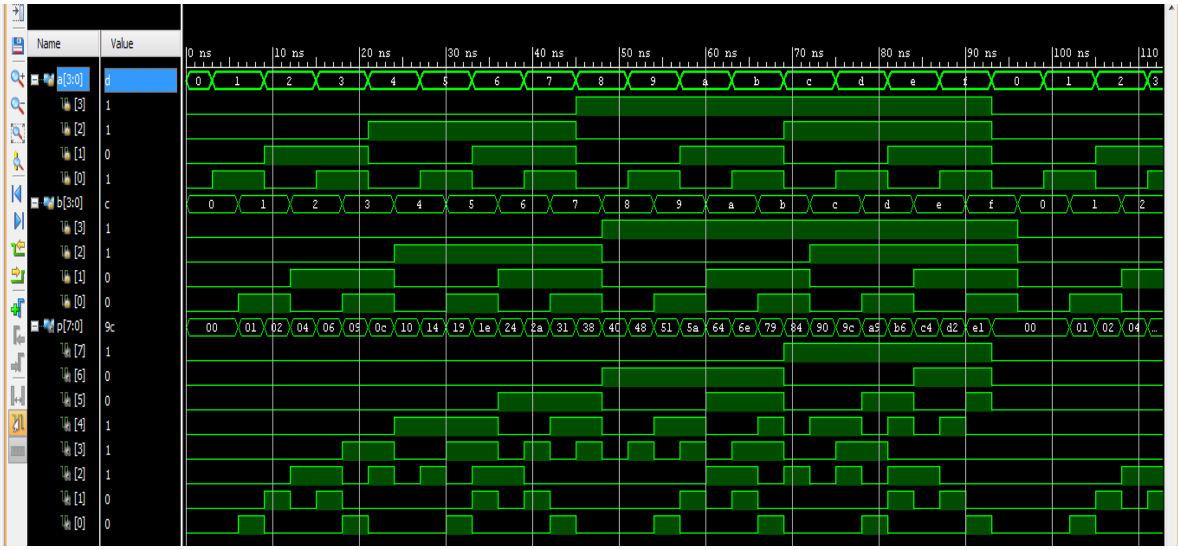
6. Click on plus symbol(Add source)

7. Click on **Add or create design source and NEXT** .In the opened window, you can create source file (Verilog/Verilog Header/SystemVeilog) for your new project or add sources from the existing projects. Click on “**Create File”**, and in the opened window .

8.Click on Create file and type the file name
The opened window is the main environment for your project that is called “Project Manager”. You can explore it by seeing the options of each category in the toolbar on top of the window. In the left side, you can see the “Settings”, “Add Sources”, “”Language Template”, “IP Catalog”, “IP Integrator”, “Simulation”, “RTL Analysis”, “Synthesis”, “Implementation”, and “Program and Debug”. Each of these serves a part of the digital design flow. In the middle, you can see the windows for “Sources”, “Properties”, “Project Summary”, and the reports and summaries for the execution of the project file

9. Type the program
10. Click on plus button(Add sources) and select Add or create simulation sources and NEXT
11. Click on the Create file and type the file name
12. Type the test bench program and save
13. Run simulation -> click on Run behavioral simulation

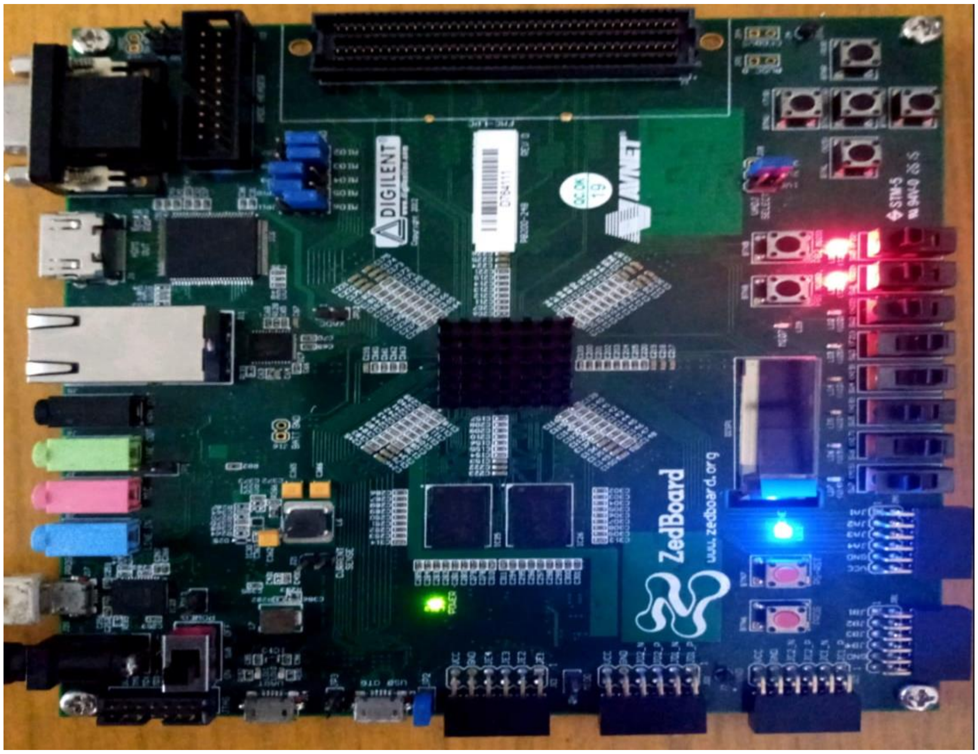
14.SIMULATION OUTPUT



15. Click on RTL Analysis -> open elaborate designs(click on I/O ports)
16. Assign port packages(assign pin number) and I/O std (select LVCMOS33)

INPUT PIN NUMBERS		OUTPUT LED PIN NUMBERS	
SW0	F22	LD0	T22
SW1	G22	LD1	T21
SW2	H22	LD2	U22
SW3	F21	LD3	U21
SW4	H19	LD4	V22
SW5	H18	LD5	W22
SW6	H17	LD6	U19
SW7	M15	LD7	U14

17. Save and type the XDC File name
18. Run the Implementation and select generate bit stream click OK
19.Select the Open Hardware Manager and click on OK
20. Connect the Hardware kit (Ex: ZedBoard) and Click on Open Target -> auto connect
21.Click on the Program Device ,verify the **.bit** file and click OK
22. Verify the function table on Zedboard



CONCLUSION: Hence a 4x4 array multiplier is designed and implemented on Zed board using Xilinx Vivado2017.2

PRECAUTIONS:

1. Give connections carefully such as Zed Board ,JTAG Power cable, power supply etc.
2. Switch on the power supply.
3. Handle the Zed Board carefully.
4. Check pin configuration before configuring the Target Device.

VIVA –QUESTIONS:

1. What is array multiplier?
2. What is parallel adder?
3. How many adders are required to implement 4x4 array multiplier?
4. What are the disadvantages of array multipliers?
5. Classify multipliers.

Implementation of a 4-Bit Arithmetic & Logic Unit

EXP NO -3

DATE:

AIM: To design and simulate of a 4-Bit Arithmetic & Logic using Xilinx's VIVADO and its implementation on Zed board Evaluation and Development Kit

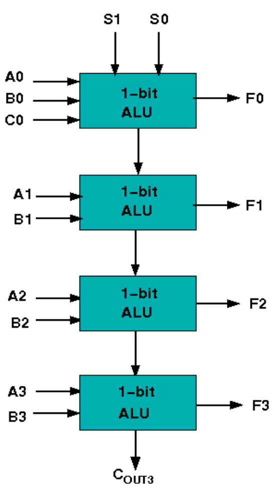
COMPONENTS & TOOLS REQUIRED:

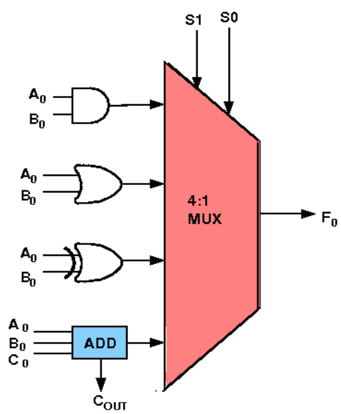
Target devices: Xilinx Zynq-7000- Zed board Evaluation and Development Kit/Zybo board

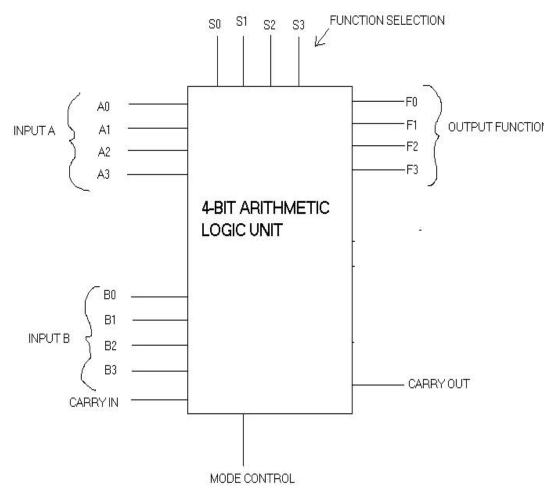
Tools: Xilinx VIVADO suite

Preferred language- Verilog

THEORY: ALU or Arithmetic Logical Unit is a digital circuit that performs arithmetic operations like addition, subtraction, division, multiplication and logical oparations like and, or, xor, nand, nor etc.







<u>VERILOG CODE :</u>

```
module alu(input [3:0] A,B,                          // ALU 8-bit Inputs
input [3:0] ALU_Sel,                                 // ALU Selection
output [4:0] ALU_Out,                                // ALU 8-bit Output
output  CarryOut);                                   // Carry Out Flag
reg [4:0] ALU_Result;
wire [5:0] tmp;
assign ALU_Out=ALU_Result;                           // ALU out
assign tmp= {1'b0,A} + {1'b0,B};
assign CarryOut=tmp[5];                              // Carryout flag
always @(*)
begin
case(ALU_Sel)
4'b0000:   ALU_Result=A+B ;          // Addition
 4'b0001:   ALU_Result=A-B ;         // Subtraction
```

```
4'b0010:  ALU_Result=A*B;              // Multiplication

4'b0011: ALU_Result=A/B;                    // Division

4'b0100: ALU_Result=A<<1;                  // Logical shift left

4'b0101: ALU_Result=A>>1;                  // Logical shift right

4'b0110: ALU_Result= {A[2:0],A[3]};      // Rotate left

 4'b0111: ALU_Result= {A[0],A[3:1]};      // Rotate right

4'b1000:  ALU_Result=A&B;                   //  Logical and

4'b1001: ALU_Result=A|B;                      //  Logical or

4'b1010:  ALU_Result=A^B;                   //  Logical xor

4'b1011: ALU_Result=~(A|B);                  //  Logical nor

4'b1100:  ALU_Result=~(A&B);               // Logical nand

4'b1101: ALU_Result=~(A^B);                  // Logical xnor

4'b1110:  ALU_Result= (A>B)?4'd1:4'd0 ;    // Greater comparison

4'b1111:   ALU_Result= (A==B)?4'd1:4'd0 ;  // Equal comparison

default:ALU_Result=A+B ;
endcase
end
endmodule
```

TEST BENCH FOR 4-BIT ALU:

```
module tst_alu();
reg [3:0] a,b;
reg [3:0] alu_sel;
wire [4:0] alu_out;
alu call(a,b,alu_sel,alu_out);
initial
begin
 a= 4'h 6; b=4'h 5; alu_sel = 4'h 0;
#320 a= 4'h A; b =4'h9;
end
always
#20 alu_sel = alu_sel + 4'h1;
endmodule
```

PROCEDURE:

1. Double click on the Vivado2017.2 icon on your desktop to open up the welcome window of the development tool (as shown below). Three main sections can be observed in this window: “Quick Start”, “Tasks”, and “Learning Center”.
2. . Now, click on “Create Project” to create a new project. You have to be careful about where to save your project file .
3. Click on NEXT and type project name, project location click on NEXT
4. In the next window, choose “RTL Project” as the project type. (click on select button), click on NEXT
5. Click on Boards:

   i. vendor:em.avnet.com

   ii. Display Name: Zed board Evaluation and Development Kit.

   iii. Board Rev : Latest, click on NEXT,FINISH

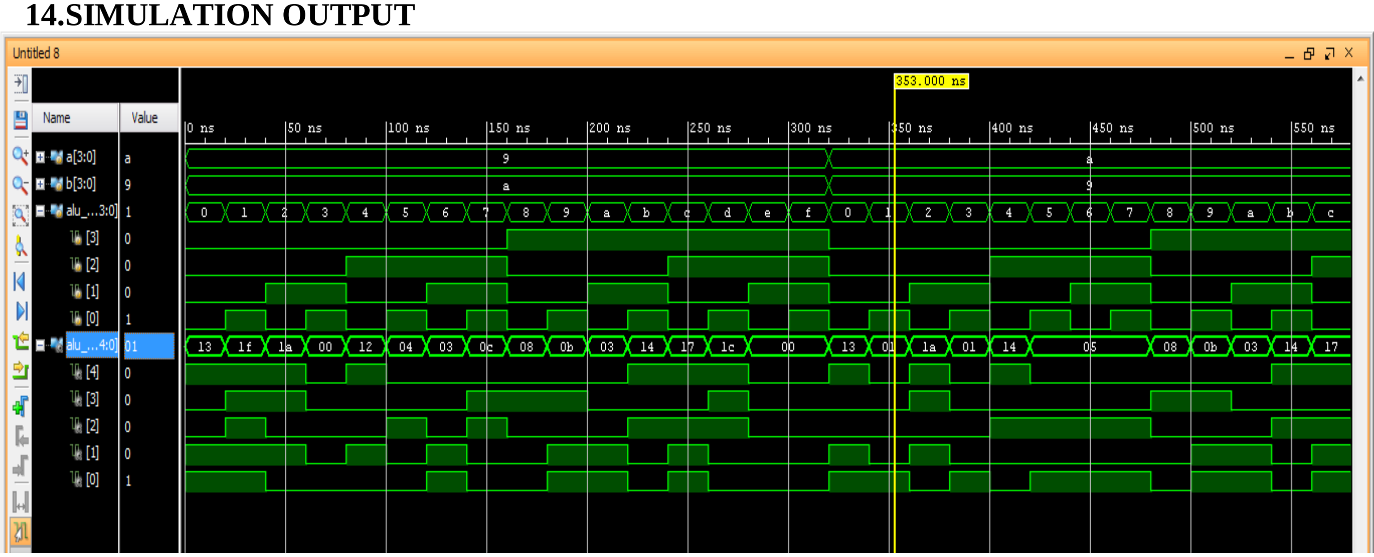
6. Click on plus symbol(Add source)

7. Click on **Add or create design source and NEXT** .In the opened window, you can create source file (Verilog/Verilog Header/SystemVeilog) for your new project or add sources from the existing projects. Click on “**Create File”**, and in the opened window .

8.Click on Create file and type the file name
The opened window is the main environment for your project that is called “Project Manager”. You can explore it by seeing the options of each category in the toolbar on top of the window. In the left side, you can see the “Settings”, “Add Sources”, “”Language Template”, “IP Catalog”, “IP Integrator”, “Simulation”, “RTL Analysis”, “Synthesis”, “Implementation”, and “Program and Debug”. Each of these serves a part of the digital design flow. In the middle, you can see the windows for “Sources”, “Properties”, “Project Summary”, and the reports and summaries for the execution of the project file

9. Type the program
10. Click on plus button(Add sources) and select Add or create simulation sources and NEXT
11. Click on the Create file and type the file name
12. Type the test bench program and save
13. Run simulation -> click on Run behavioral simulation

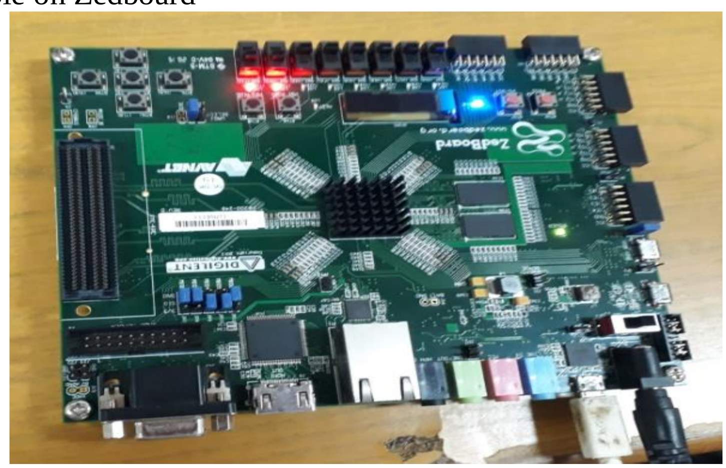
14.SIMULATION OUTPUT



15. Click on RTL Analysis -> open elaborate designs(click on I/O ports)
16. Assign port packages(assign pin number) and I/O std (select LVCMOS33)

INPUT PIN NUMBERS		OUTPUT LED PIN NUMBERS	
SW0	F22	LD0	T22
SW1	G22	LD1	T21
SW2	H22	LD2	U22
SW3	F21	LD3	U21
SW4	H19	LD4	V22
SW5	H18	LD5	W22
SW6	H17	LD6	U19
SW7	M15	LD7	U14
S0	T18		
S1	N15		
S2	P16		
S3	R18		

17. Save and type the XDC File name
18. Run the Implementation and select generate bit stream click OK
19.Select the Open Hardware Manager and click on OK
20. Connect the Hardware kit (Ex: ZedBoard) and Click on Open Target -> auto connect
21.Click on the Program Device ,verify the **.bit** file and click OK
22. Verify the function table on Zedboard



CONCLUSION: Hence a 4-bit ALU is designed and implemented on Zed board using Xilinx Vivado2017.2

PRECAUTIONS:
1. Give connections carefully such as Zed Board ,JTAG Power cable, power supply etc.
2. Switch on the power supply.
3. Handle the Zed Board carefully.
4. Check pin configuration before configuring the Target Device.

VIVA –QUESTIONS:
1. What does ALU means?
2. What operations does ALU perform?
3. Mention the role of control unit and ALU in computer system?
4. What is the difference between ALU and control unit?
5. What is the use of multiplexer in ALU design?

Implementation of Zero /one Detector

EXP NO-4

DATE:

AIM: To design and simulate Zero /one Detector using Xilinx's VIVADO and its implemention on Zed Board Evaluation and Development Kit

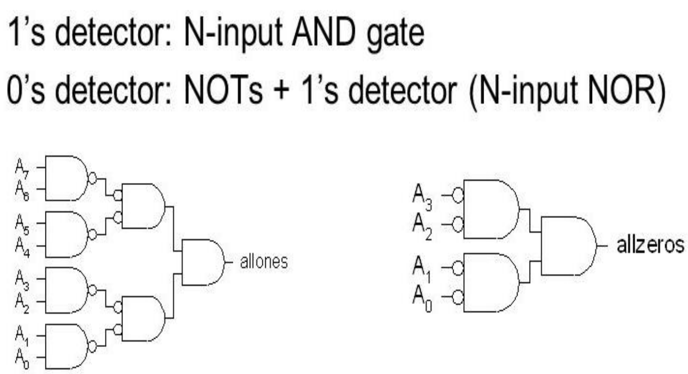
COMPONENTS & TOOLS REQUIRED:

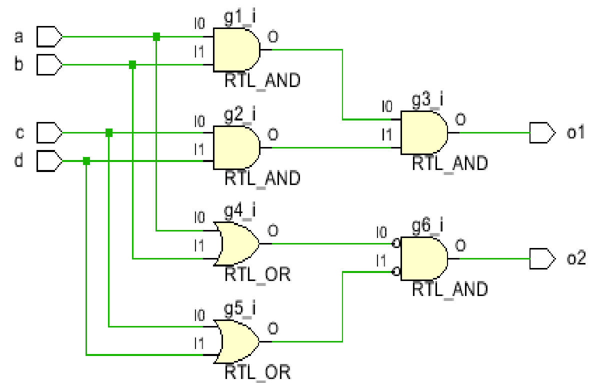
Target devices: Xilinx Zynq-7000- Zed board Evaluation and Development Kit/Zybo board

Tools: Xilinx VIVADO suite

Preferred language- Verilog

THEORY: Detecting all ones or all zeros on wide N-bit words requires large fan-in AND or NOR gates.





VERILOG CODE :

```
module zeroone (o1,o2,a,b,c,d);
input a,b,c,d;
output o1,o2;
wire w,x,y,z;
and  g1(w,a,b),g2(x,c,d),g3(o1,w,x);
nor g4(y,a,b),g5(z,c,d);
and g6(o2,y,z);
endmodule
```

TEST BENCH

```
module tb();
reg a,b,c,d;
wire o1,o2;
zeroone tb(o1,o2,a,b,c,d);
initial
begin
a=0;b=0;c=0;d=0;
#2 a=1;b=1;c=0;d=1;
#2 a=1;b=1;c=1;d=1;
```

```
end
initial #10 $stop;
initial $monitor($time,"o1=%b,o2=%b,a=%b,b=%b,c=%b,d=%b",o1,o2,a,b,c,d);
endmodule
```

PROCEDURE:

1. Double click on the Vivado2017.2 icon on your desktop to open up the welcome window of the development tool (as shown below). Three main sections can be observed in this window: “Quick Start”, “Tasks”, and “Learning Center”.
2. . Now, click on “Create Project” to create a new project. You have to be careful about where to save your project file .
3. Click on NEXT and type project name, project location click on NEXT
4. In the next window, choose “RTL Project” as the project type. (click on select button), click on NEXT
5. Click on Boards:
   i. vendor:em.avnet.com
   ii. Display Name: Zed board Evaluation and Development Kit.
   iii. Board Rev : Latest, click on NEXT,FINISH

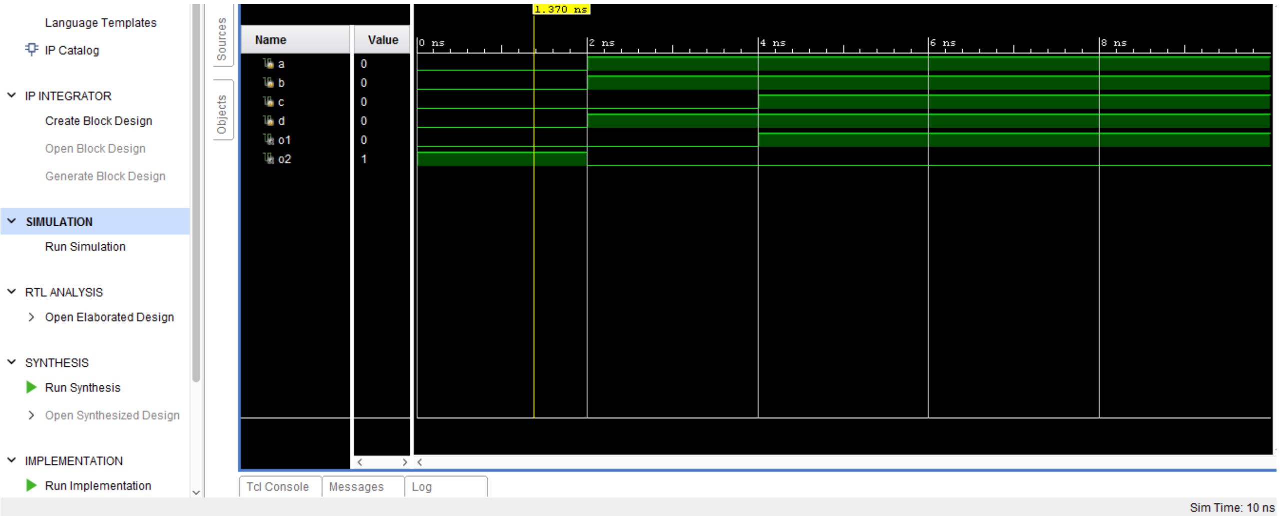
6. Click on plus symbol(Add source)

7. Click on **Add or create design source and NEXT** .In the opened window, you can create source file (Verilog/Verilog Header/SystemVeilog) for your new project or add sources from the existing projects. Click on “**Create File”**, and in the opened window .

8.Click on Create file and type the file name
The opened window is the main environment for your project that is called “Project Manager”. You can explore it by seeing the options of each category in the toolbar on top of the window. In the left side, you can see the “Settings”, “Add Sources”, “”Language Template”, “IP Catalog”, “IP Integrator”, “Simulation”, “RTL Analysis”, “Synthesis”, “Implementation”, and “Program and Debug”. Each of these serves a part of the digital design flow. In the middle, you can see the windows for “Sources”, “Properties”, “Project Summary”, and the reports and summaries for the execution of the project file

9. Type the program
10. Click on plus button(Add sources) and select Add or create simulation sources and NEXT
11. Click on the Create file and type the file name
12. Type the test bench program and save
13. Run simulation -> click on Run behavioral simulation

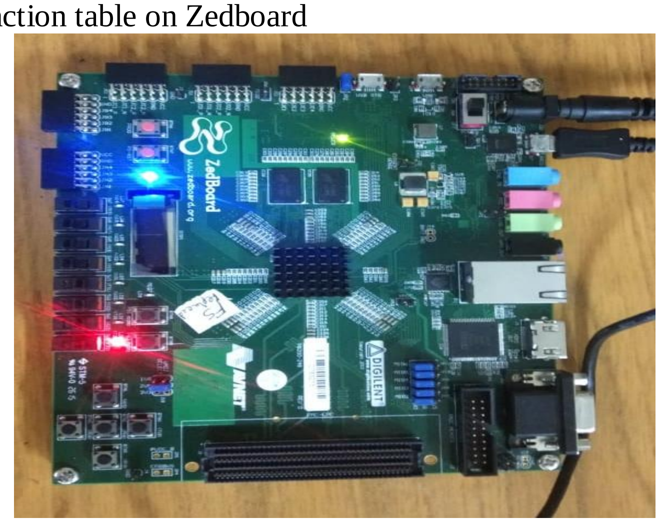
14. SIMULATION OUTPUT



15. Click on RTL Analysis -> open elaborate designs(click on I/O ports)
16. Assign port packages(assign pin number) and I/O std (select LVCMOS33)

INPUT PIN NUMBERS		OUTPUT LED PIN NUMBERS	
SW0	F22	LD0	T22
SW1	G22	LD1	T21
SW2	H22	LD2	U22
SW3	F21	LD3	U21
SW4	H19	LD4	V22
SW5	H18	LD5	W22
SW6	H17	LD6	U19
SW7	M15	LD7	U14

17. Save and type the XDC File name
18. Run the Implementation and select generate bit stream click OK
19.Select the Open Hardware Manager and click on OK
20. Connect the Hardware kit (Ex: ZedBoard) and Click on Open Target -> auto connect
21.Click on the Program Device ,verify the **.bit** file and click OK
22. Verify the function table on Zedboard



CONCLUSION: Hence a zero /one detector is designed and implemented on Zed board using Xilinx

Vivado2017.2

PRECAUTIONS:

1. Give connections carefully such as Zed Board , JTAG Power cable, power supply etc.
2. Switch on the power supply.
3. Handle the Zed Board carefully.
4. Check pin configuration before configuring the Target Device.

VIVA –QUESTIONS:

1. What does zero/one means?
2. What are the applications of zero/one detector?
3. Develop a sequence detector?

Implementation of Flip Flops: SR, JK,T,D

EXP NO-5

DATE:

AIM: To design and simulate SR, D, JK, T Flip- Flops using Xilinx's VIVADO and its implementation on Zed Board Evaluation and Development Kit

COMPONENTS & TOOLS REQUIRED:

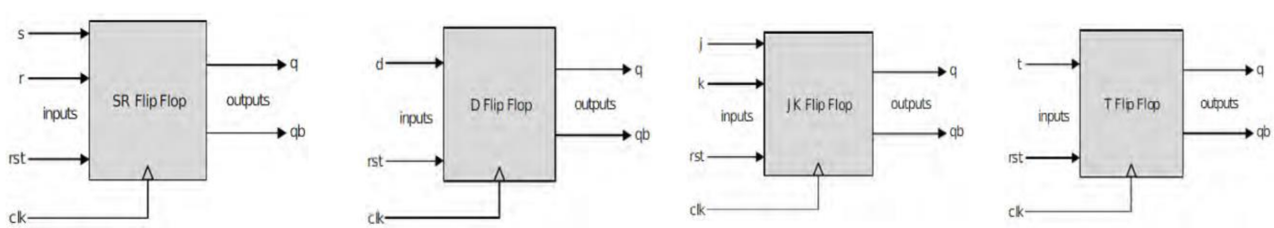
Target devices: Xilinx Zynq-7000- Zed board Evaluation and Development Kit/Zybo board

Tools: Xilinx VIVADO suite

Preferred language- Verilog

THEORY : A flip flop is an electronic circuit with two stable states that can be used to store binary data. It is the basic storage element in sequential logic. There are majorly 4 types of flip-flops, with the most common one being SR flip-flop. This simple flip-flop circuit has a set input (S) and a reset input (R). In this system, when Set "S" as active, the output "Q" would be high and "Q'" will be low when reset the input. Due to the undefined state in the SR flip-flop, another flip-flop is required in electronics. The JK flip-flop is an improvement on the SR flip-flop where the undefined state which occurs when S=R=1 is eliminated.. The JK Flip Flop when J=K=1,the output is complement of previous state.In D flip-flop the output (Q) is same as the input. A T flip-flop is like a JK flip-flop. This is basically a single input version of JK flip-flops. This modified form of JK flip-flop is obtained by connecting both inputs J and K together. It has only one input along with the clock input.Because of its ability to complement its state (i.e.) Toggle, hence the name Toggle flip-flop.

LOGIC SYMBOLS OF FLIP-FLOPS



TRUTH TABLE:

S-R Flip Flop

Inputs				Outputs		
rst	clk	s	r	q	qb	Action
1	↑	X	X	q	qb	No Change
0	↑	0	0	q	qb	No Change
0	↑	0	1	0	1	Reset
0	↑	1	0	1	0	Set
0	↑	1	1	-	-	Illegal

D-Flip Flop

Inputs			Outputs		
rst	clk	d	q	qb	Action
1	↑	X	q	qb	No Change
0	↑	0	0	1	Reset
0	↑	1	1	0	Set

J-K Flip Flop

Inputs				Outputs		
rst	clk	j	k	q	qb	Action
1	↑	X	X	q	qb	No Change
0	↑	0	0	q	qb	No Change
0	↑	0	1	0	1	Reset
0	↑	1	0	1	0	Set
0	↑	1	1	q'	q'	Toggle

T- Flip Flop

Inputs			Outputs		
rst	clk	t	q	qb	Action
1	↑	X	q	qb	No Change
0	↑	0	q	qb	No Change
0	↑	1	q'	q'	Toggle

VERILOG CODE:

SR FLIPFLOP:

```
module srff(
    input s,r,
    input clk,
    output reg q,nq
    );
    initial
        begin
            q=1'b0;
            nq=1'b1;
        end
        always @(posedge clk)
          begin
          case({s,r})
             {1'b0,1'b0}: begin q=q; nq=nq; end
             {1'b0,1'b1}: begin q=1'b0; nq=1'b1; end
             {1'b1,1'b0}: begin q=1'b1; nq=1'b0; end
             {1'b1,1'b1}: begin q=1'bx; q=1'bx; end
        endcase
        end
endmodule
```

TEST BENCH PROGRAM

```
module sr_ff_test;
reg s,r,clk;
wire q,nq;
sr_ff sr_ff_test(s,r,clk,q,nq);
initial
begin
forever
begin
clk=1;
#50 clk=0;
#50 clk=1;
end
end
initial
begin
     s=0;r=1;
#100 s=0;r=0;   #100 s=1;r=0;   #100 s=1;r=1;
end
initial
begin
$monitor($time,"s=%b,r=%b,clk=%b,q=%b,nq=%b",s,r,clk,q,nq);
end
endmodule
```

JK FLIPFLOP WITH CLOCK DIVISION:

```
module jk_ff(q,nq,j,k,clk);
output reg q,nq;
input j,k,clk;
reg clkd; reg [30:0] div;
always @ (posedge clk)
begin
div <= div+1'b1;
clkd <= div[26];
end
initial begin q=1'b0; nq=1'b1; end
always @ (posedge clkd)
  begin
          case({j,k})
                    {1'b0,1'b0}:begin q=q; nq = nq; end
                    {1'b0,1'b1}: begin q=1'b0; nq =1'b1; end
                    {1'b1,1'b0}:begin q=1'b1; nq =1'b0; end
                    {1'b1,1'b1}: begin q=~q; nq =~ nq; end
          endcase
         end
endmodule
```

TEST BENCH PROGRAM:

```
module tb_jk ();
   reg j,k,clk;

wire q,nq;
initial begin
  clk=0;
end;

   always #5 clk = ~clk;
   jk_ff swt ( q,nq,j,k,clk);
   initial
   begin
      j <= 0;
      k <= 0;
      #5 j <= 0;
         k <= 1;
      #15 j <= 1;
          k <= 0;
      #25 j <= 0;
          k <= 0;
     #35 j <= 1;
         k <= 1;
   end;
 endmodule
```

T- FLIPFLOP:

```
module tff ( q, clk,reset, t);
```

```
input clk,reset,t;
output reg q;
  always @ (posedge clk) begin
    if (reset)
      q <= 0;
    else
            if (t)
                        q <= ~q;
            else
                        q <= q;
  end
endmodule
```

TEST BENCH PROGRAM:

```
module tb_tff();
reg t;
reg clk;
reg reset;
wire q;

tff dut(q, clk,reset, t);

initial begin
  clk=0;
     forever #10 clk = ~clk;
end
initial begin
 reset=1;
  #100;
 reset=0;
 t <= 1;
 #100;
 t <= 0;
 #100;
 t <= 1;
end
endmodule
```

T- FLIPFLOP:

```
module tff ( q, clk,reset, t);
input clk,reset,t;
output reg q;
always @ (posedge clk) begin
if (reset)
q <= 0;
else
if (t)
q <= ~q;
else
```

```
q <= q;
end
endmodule
```

TEST BENCH PROGRAM:

```
module tb_tff();
reg t;
reg clk;
reg reset;
wire q;
tff dut(q, clk,reset, t);
initial begin
clk=0;
forever #10 clk = ~clk;
end
initial begin
reset=1;
#100;
reset=0;
t <= 1;
#100;
t <= 0;
#100;
t <= 1;
end
endmodule
```

D- FLIPFLOP:

```
module D_ff(
    input d,clk,async_reset,
    output reg q
    );
    always @(posedge clk or posedge async_reset)
    begin
     if(async_reset==1'b1)
      q <= 1'b0;
     else
      q <= d;
    end
endmodule
```

TEST BENCH PROGRAM:

```
module tb_DFF();
reg D; reg clk; reg reset;
wire Q;

D_ff dut(D,clk,reset,Q);

initial begin
  clk=0;
     forever #10 clk = ~clk;
```

```
end
initial begin
 reset=1;
 D <= 0;
 #100;   reset=0;  D <= 1;
 #100;  D <= 0;
 #100;   D <= 1;
end
endmodule
```

PROCEDURE:

1. Double click on the Vivado2017.2 icon on your desktop to open up the welcome window of the development tool (as shown below). Three main sections can be observed in this window: “Quick Start”, “Tasks”, and “Learning Center”.
2. . Now, click on “Create Project” to create a new project. You have to be careful about where to save your project file .
3. Click on NEXT and type project name, project location click on NEXT
4. In the next window, choose “RTL Project” as the project type. (click on select button), click on NEXT
5. Click on Boards:
   i. vendor:em.avnet.com
   ii. Display Name: Zed board Evaluation and Development Kit.
   iii. Board Rev : Latest, click on NEXT,FINISH

6. Click on plus symbol(Add source)

7. Click on **Add or create design source and NEXT** .In the opened window, you can create source file (Verilog/Verilog Header/SystemVeilog) for your new project or add sources from the existing projects. Click on “**Create File”**, and in the opened window .

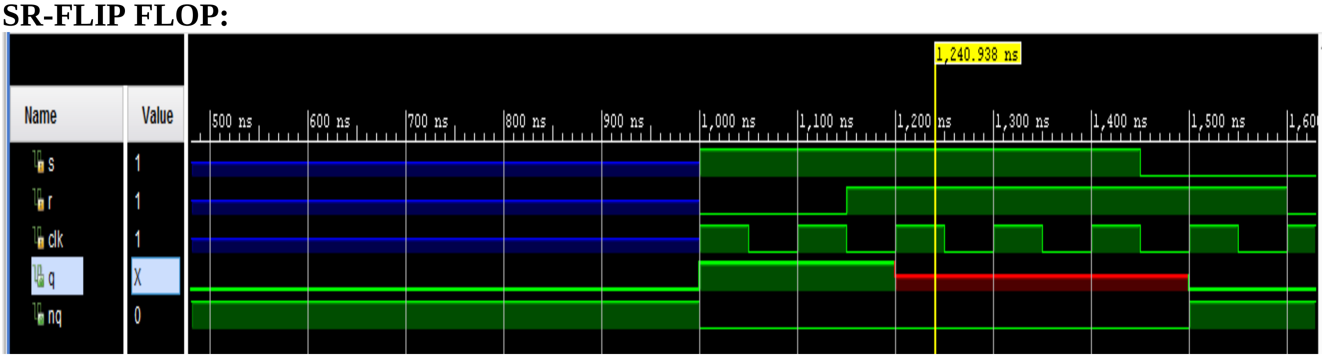
8.Click on Create file and type the file name

The opened window is the main environment for your project that is called “Project Manager”. You can explore it by seeing the options of each category in the toolbar on top of the window. In the left side, you can see the “Settings”, “Add Sources”, “”Language Template”, “IP Catalog”, “IP Integrator”, “Simulation”, “RTL Analysis”, “Synthesis”, “Implementation”, and “Program and Debug”. Each of these serves a part of the digital design flow. In the middle, you can see the windows for “Sources”, “Properties”, “Project Summary”, and the reports and summaries for the execution of the project file

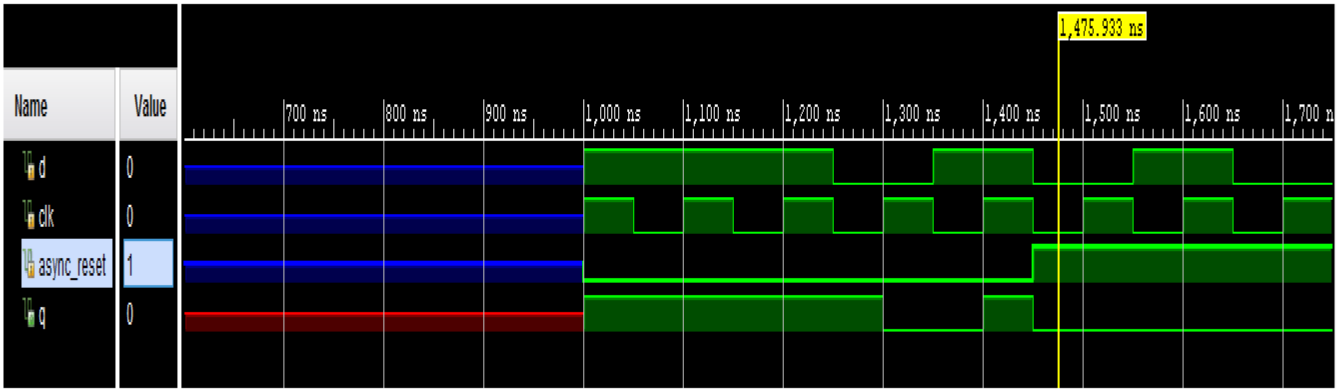
9. Type the program
10. Click on plus button(Add sources) and select Add or create simulation sources and NEXT
11. Click on the Create file and type the file name
12. Type the test bench program and save
13. Run simulation -> click on Run behavioral simulation

14.SIMULATION OUTPUT

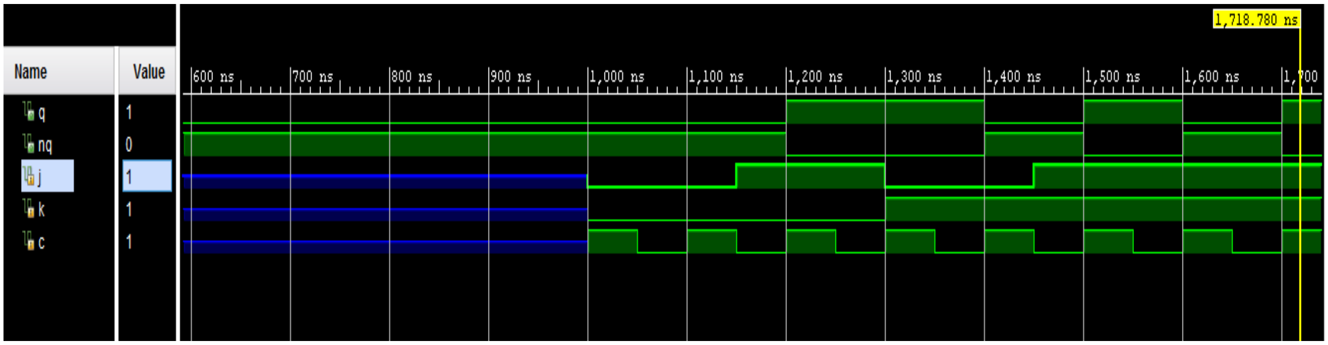
SR-FLIP FLOP:



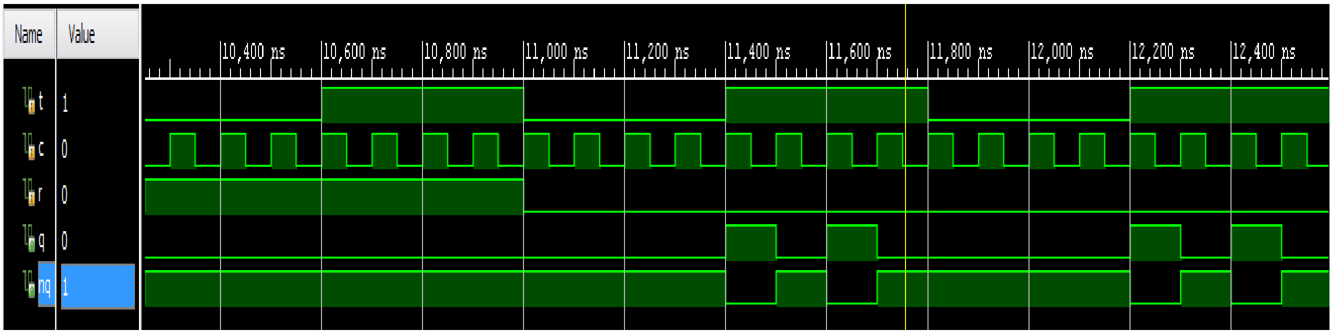
D-FLIP FLOP:



JK-FLIP FLOP:



T-FLIP FLOP:



15. Click on RTL Analysis -> open elaborate designs(click on I/O ports)
16. Assign port packages(assign pin number) and I/O std (select LVCMOS33)

INPUT PIN NUMBERS		OUTPUT LED PIN NUMBERS	
SW0	F22	LD0	T22
SW1	G22	LD1	T21
SW2	H22	LD2	U22
SW3	F21	LD3	U21
SW4	H19	LD4	V22
SW5	H18	LD5	W22
SW6	H17	LD6	U19
SW7	M15	LD7	U14

17. Save and type the XDC File name
18. Run the Implementation and select generate bit stream click OK
19. Select the Open Hardware Manager and click on OK
20. Connect the Hardware kit (Ex: ZedBoard) and Click on Open Target -> auto connect
21. Click on the Program Device ,verify the .**bit** file and click OK
22. Verify the function table on Zedboard

CONCLUSION: Hence all the flip-flops are designed and implemented on Zed board using Xilinx

Vivado2017.2

PRECAUTIONS:

1. Give connections carefully such as Zed Board , JTAG Power cable, power supply etc.
2. Switch on the power supply.
3. Handle the Zed Board carefully.
4. Check pin configuration before configuring the Target Device.

<u>VIVA-QUESTIONS</u>:

1. In a J-K flip-flop, if J=K the resulting flip-flop is?
2. The characteristic equation of J-K flip-flop is?
3. What does the direct line on the clock input of a J-K flip-flop mean?
4. List the differences between latch and flipflop
5. Define Sequential and Combinational circuts.
6. Define level triggering and edge triggering.

Ful custom IC design flow in Cadence:

Design Entry

Go to **cds_work** folder-> create folder->RightClick->open in Terminal->virtuoso

Opens **cds.log**

Go to tools ->**Library manager**

File->**New**->**Library**->library_name->**Attach to an existing library**->**gpdk90**

Select libraryname->**File**->**New**->**cellview**->cellename->opens **virtuoso schematic editor**

Enter the design:

I->Instance

P->Pin

W->Wire

F->Fit

Q->Properties

R->Rotate

To bring pmos and **nmos** instances

Press **I**->Browse for **gpdk90** library->select either **pmos1v** or **nmos1v**->enter->edit ->**width** value->OK->place component in proper position

To move objects

Select schematic object->Click and drag to move objects

For properties of objects

Select schematic object->press key **Q** keyboard for properties of objects

To zoom in and zoom out

Ctrl+Mouse scroll up and scroll down[keys **[**->zoomout and **]**->zoomin]

To bring pins

Press key **P**->Give pin name->select direction->input [or output]->OK->place pin in proper position

To provide wiring

Press key **W**->click at the point from which wiring to be started ,to be turned and click at the point at which wiring to be ended

[use key **ESC** ->to come out of wiring]

[use undo and redo options]

[select wiring and press key **delete** to delete wiring]

Check and save

Create Symbol

Go to **create**->**cell view** ->**From cell view**->Proper alignment for pins

Close symbol and schematic

Pre-Layout simulation:

Select libraryname in library manager->**File**->**New**->**cellview**->cellename for testbench ->opens **virtuoso schematic editor**

Enter the testbench setup:

I->Instance

P->Pin

W->Wire

F->Fit

Q->Properties

To bring DC power supply and vpulse

Press **I**->Browse for **analogLib** library->select **DC powersupply** ->give dc voltage value[1.8] ->place dc power supply in proper position

[Press **I**->Browse for **analogLib** library->select **vpulse**->

Voltage1 value[0]

Voltage2 value[1.8]

Period->[40n]

Delay->[1p]

Rise time->[1p]

Fall time->[1p][delay,rise and fall time as minimum as possible]

Pulse width[20n][for 50% duty cycle]

With above values place vpulse in proper position]

Check and save

Go to **Launch**->**ADE L**->Opens ADE L window

[Go to **Variables**->**Edit**->Variable name and value->**Add**->ok]

Go to **Analyses**->**Choose**->**tran**

Stop time

Accuracy

[Analyses->**dc**->select **dc operating point**

Design variable->**select design variable**

Sweep range-> start and stop values]

Go to **outputs**->**to be ploted**->**select on schematic**->select input and output wires in schematic

Run simulation in ADE L->observe waveforms

[measure required parameters]

Layout design

Open schematic of design

Go to **launch**->**Layout XL**->**ok**->**ok**-> opens virtuoso layout editor

In virtuoso layout editor

Go to **Connectivity**->**generate**->**All from the source**->opens dialog box->Give **separation**(0.12)->Tick **in boundary**->ok->layout objects appear in PR boundary

To move objects

Select layout object->Click and drag to move objects

For properties of objects

Select layout objects->press key **Q** keyboard for properties of objects

To Extend edge of PR boundary(stretch)

Press **S** key in keyboard->Select any edge of PR boundary->Move curser and click

To zoom in and zoom out

Mouse scroll up and scroll down[ctrl+Z and shift +Z]

To get ruler->Short cut key->**K**

To remove ruler->short cut->**shift+K**

To select all->**ctrl+A**

To deselect ->**ctrl+D**

To get substrate taps for transistor

Select on transistor->**RC**->**parameter**->**Bodytietype**->change as **integrated (for inverter)**[select either **left** or **right tap**]

[**Detached** (for NAND and NOR)[select **top tap** for pmos and **bottom tap** for nmos]]

[We can also draw taps as customized]

To draw routing

Select layer in layer pallette

Press **P** [or **ctrl+shift+W**][path] and click at start point,release the mouse,move in required direction and press **Enter** at end point
[use key **ESC** ->to come out of routing]

[use **undo** and **redo** options]

[select layer[path or route] and press key **delete** to delete wiring]

To align layers

Select layer->press key **A**->select one edge of the layer to be moved and click at the point[edge] to which the layer to be moved

[Complete the layout with above information]

Verification of layout

Go to **assura** tab in layout editor->**Technology**->Browse and select path as [*home/buet/cadence/gpdk90 v4.6/assura_tech.lib*]->ok

DRC

Go to **assura**->**runDRC**->Give runname ->select Technology

LVS

Go to **assura**->**runLVS**->Give runname ->select Technology

PEX or **RCX**

Go to **assura**->**runRCX**->opens dialog box

Setup tab->**outpt**->select as **extractedview**

Extraction tab->**Extraction Type**->**RC**

Refnode->VSS![or GND!]

Filtering tab->**Enter power nets**->VDD! [**Enter**] VSS![or GND!]

Enter ground nets->GND!->OK

Post-Layout simulation

Select *testbench* **cell** in **library manager**

Go to **File**->**New**->**cellview**->change type as **config**->OK->opens new configuration->change view as **schematic**->**use** template->change name as spectre->ok and ok

Go to **tree view**->unplus **I0**->Right click->**set instance view**-> **av_exctracted view**

Click **open**-opens testbench cell view

Go to **session** in testbench schematic cell view->**load state**->**run simulation**->observe post-layout simulation waveforms

[measure required parameters]

[Refer instructions given by demonstrator]

GDSII generation

Select CDS.log window

Go to **File**->**Export**->**stream**->opens dialog box

Give stream file name-> *stremfilename*.gds[select proper location]

Add Technology library

Browse for Library->cell->view->ok

Click **Translate**

[Stream file will be located in respective location]

[find stream file]

[open in terminal]

[vim *stremfilename*.gds]-> this linux command opens stream file

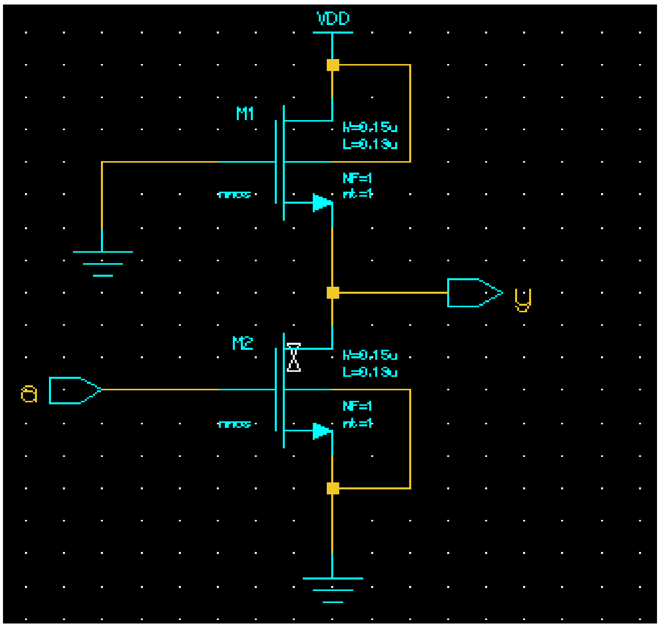
1. Design of NMOS Inverter

AIM: To design, simulate and verify the operation of NMOS inverter using Cadence tools at different VDDs, Widths of NMOS transistor by ensuring minimum Lengths and widths for its Power and Delay analysis.

Tools used:

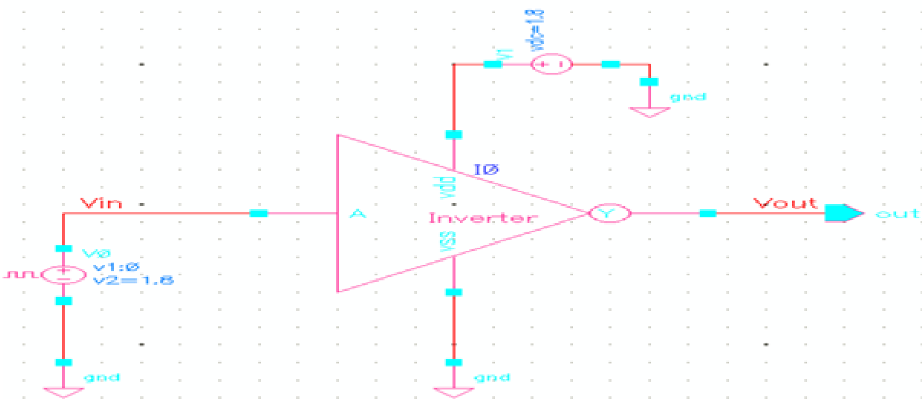
1. Virtuoso Tool for Schematic Designs
2. Spectre Tool for Simulation

Circuit:

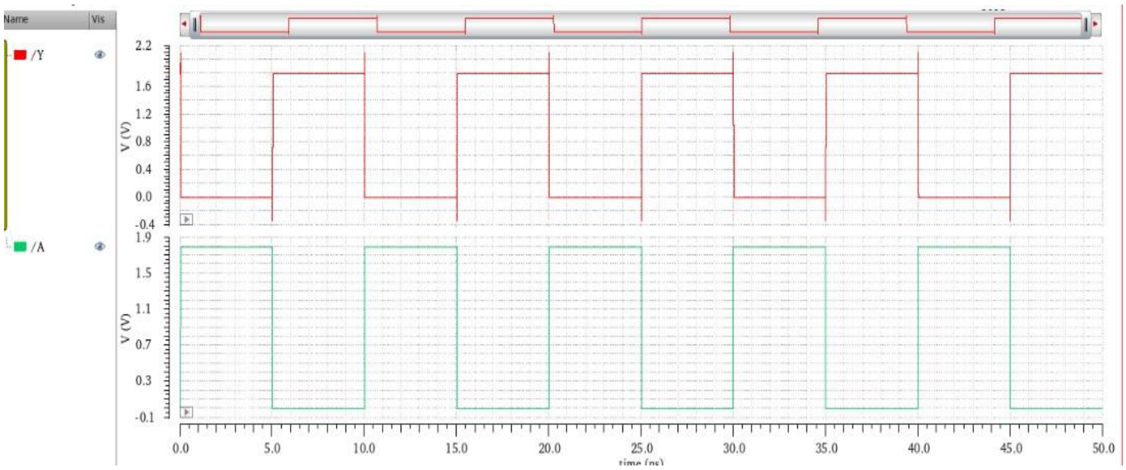


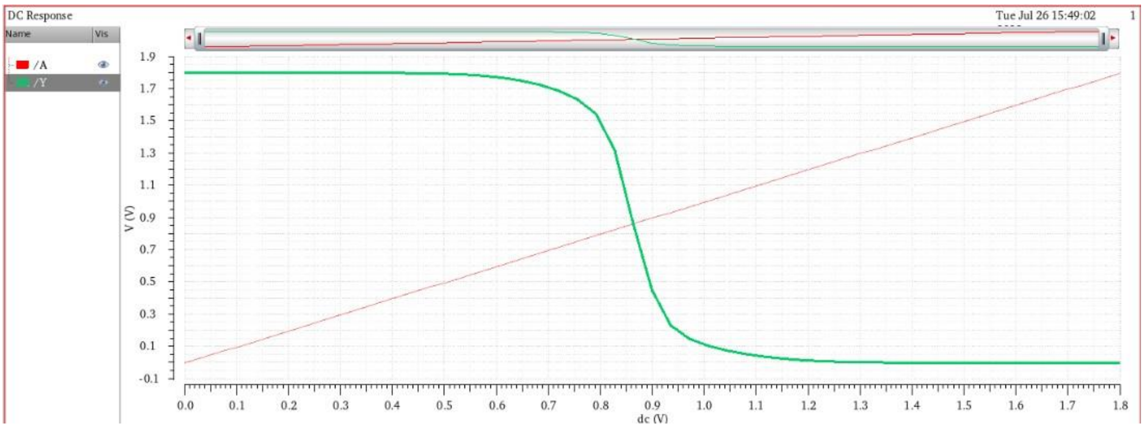
Procedure: Refer Annexure I

Testbench:



Simulation Results:





Transient and DC analysis

Result:

The design, simulation and verification of CMOS inverter using Cadence tools at different VDD, Widths of NMOS and PMOS transistors for its Power and Delay analysis was performed.

1) VDD:________, Width of NMOS :____________: Width of Power:__________:Delay:______________

2) VDD:________, Width of NMOS :____________: Width of Power:__________:Delay:______________

3) VDD:________, Width of NMOS :____________: Width of Power:__________:Delay:______________

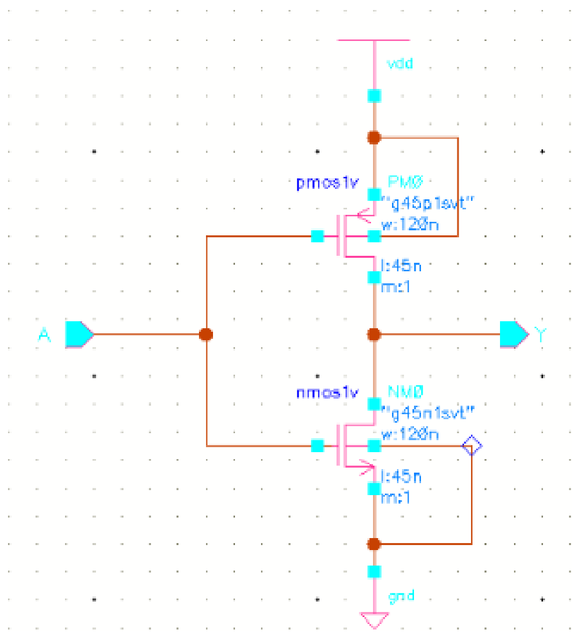
2. Design of CMOS Inverter

AIM: To design, simulate and verify the operation of CMOS inverter using Cadence tools at different VDDs, Widths of NMOS and PMOS transistors by ensuring minimum Lengths and widths for its Power and Delay analysis.

Tools used:

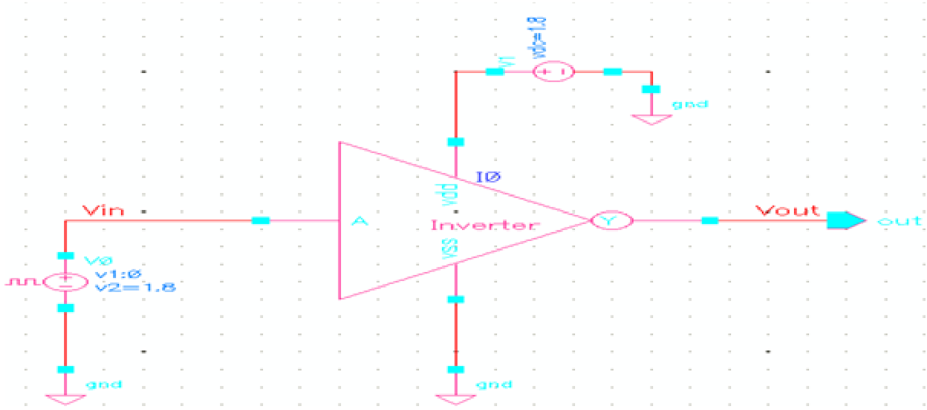
3. Virtuoso Tool for Schematic Designs
4. Spectre Tool for Simulation

Circuit:

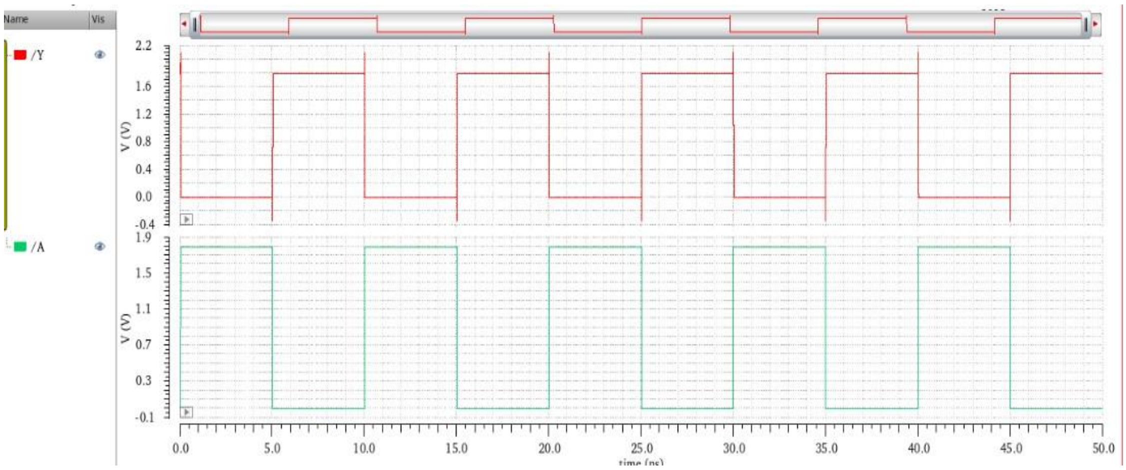


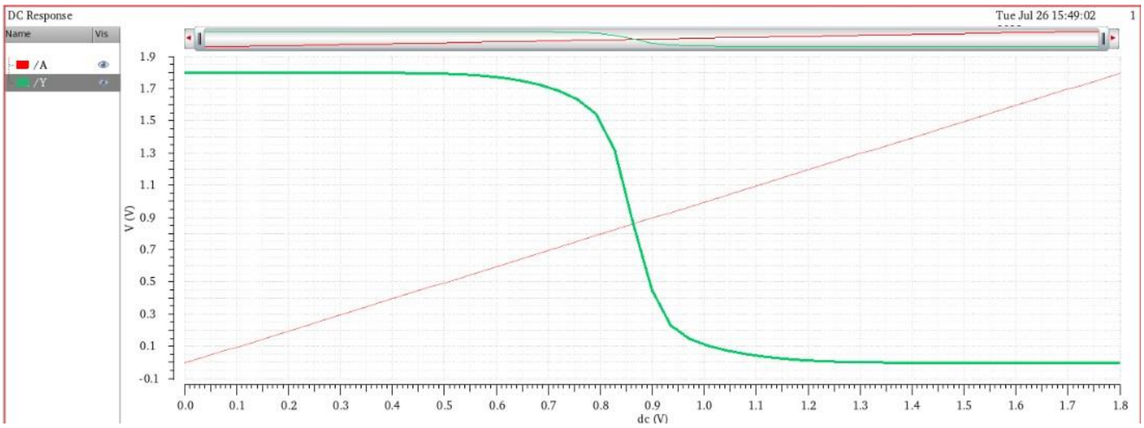
Procedure: Refer Annexure I

Testbench:



Simulation Results:





Transient and DC analysis

Result:

The design, simulation and verification of CMOS inverter using Cadence tools at different VDD, Widths of NMOS and PMOS transistors for its Power and Delay analysis was performed.

1) VDD:________, Width of NMOS :____________: Width of PMOS:____________:Power:___________:Delay:______________

2) VDD:________, Width of NMOS :____________: Width of PMOS:____________:Power:___________:Delay:______________

3) VDD:________, Width of NMOS :____________: Width of PMOS:____________:Power:___________:Delay:______________

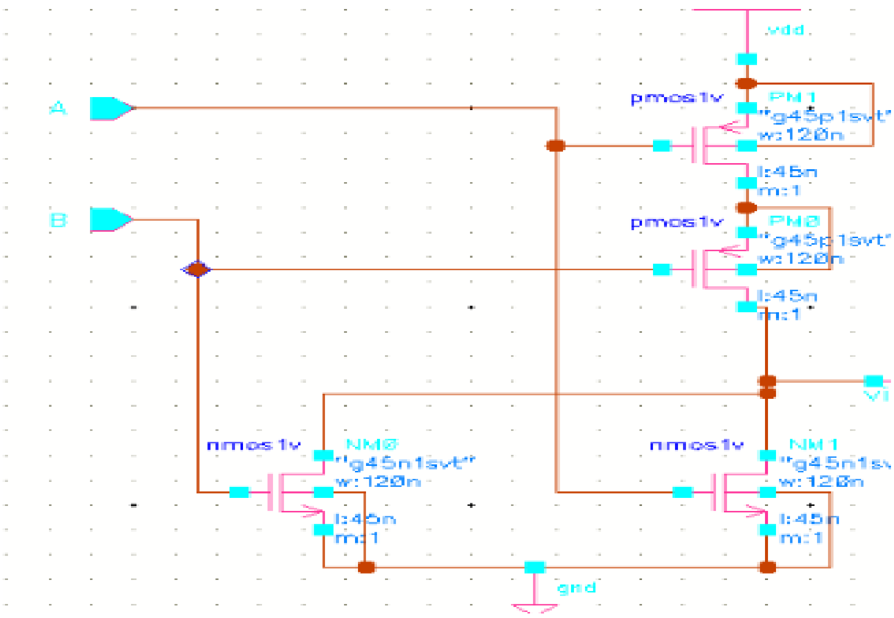
3. Design of 2-input CMOS NOR gate

AIM: To design, simulate and verify the operation of CMOS NOR using Cadence tools at different VDDs, Widths of NMOS and PMOS transistors by ensuring minimum Lengths and widths for its Power and Delay analysis.

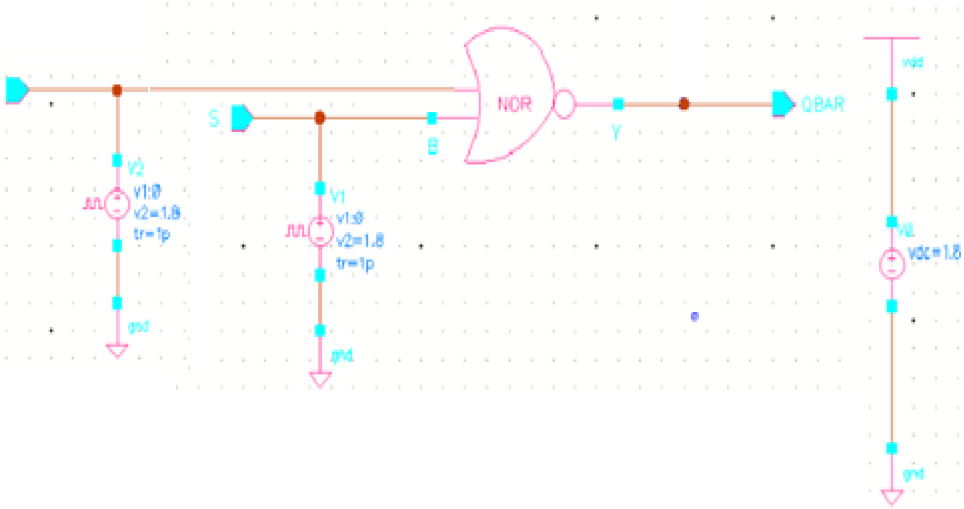
Tools used:

1. Virtuoso Tool for Schematic Designs
2. Spectre Tool for Simulation

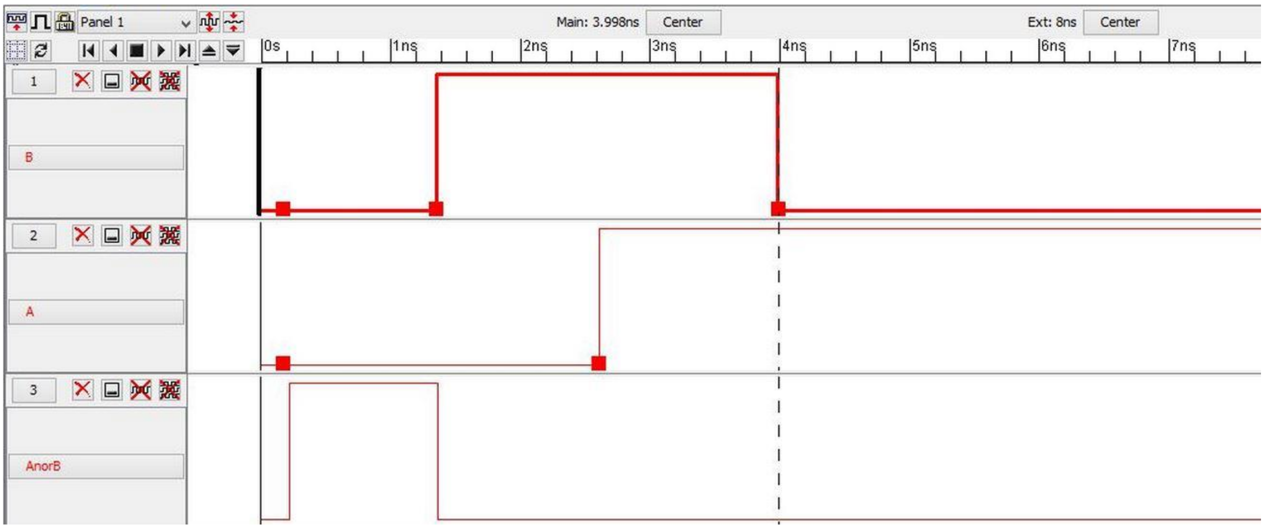
Circuit:



Testbench:



SimulationResults:



Result:

The design, simulation and verification of CMOS NOR Gate using Cadence tools at different VDD, Widths of NMOS and PMOS transistors for its Power and Delay analysis was performed.

1) VDD:________, Width of NMOS :____________: Width of PMOS:___________:Power:__________:Delay:______________

2) VDD:________, Width of NMOS :____________: Width of PMOS:___________:Power:__________:Delay:______________

3) VDD:________, Width of NMOS :____________: Width of PMOS:___________:Power:__________:Delay:______________

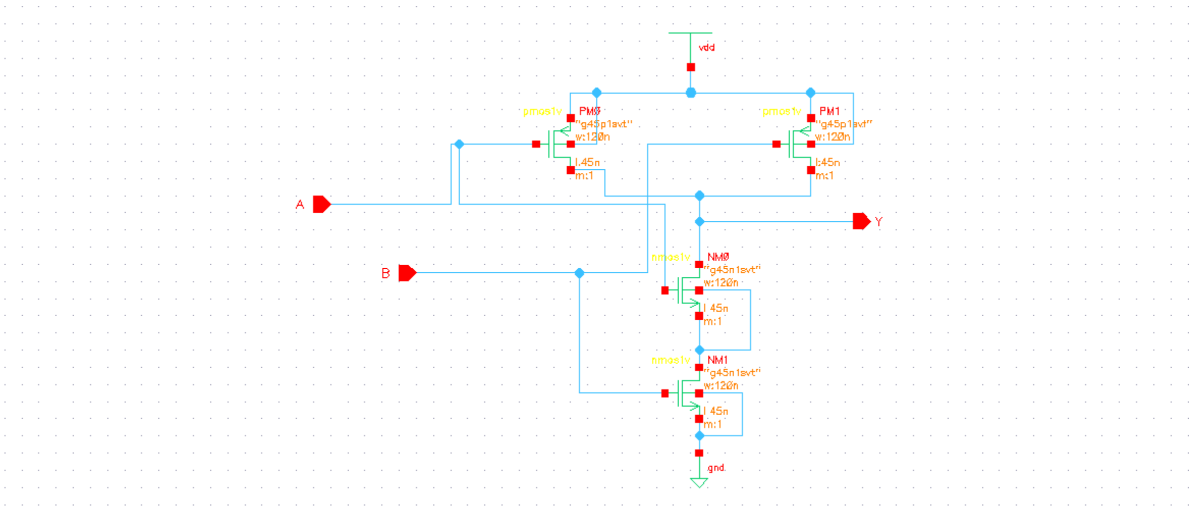
4.Design of 2-input NAND Gate Design

AIM: To design, simulate and verify the operation of CMOS NAND using Cadence tools at different VDDs, Widths of NMOS and PMOS transistors by ensuring minimum Lengths and widths for its Power and Delay analysis.

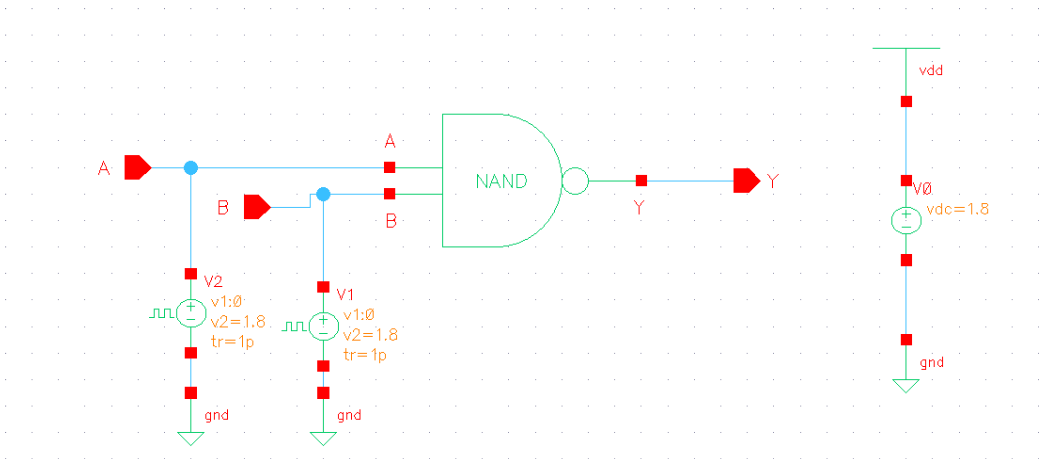
Tools used:

1. Virtuoso Tool for Schematic Designs
2. Spectre Tool for Simulation

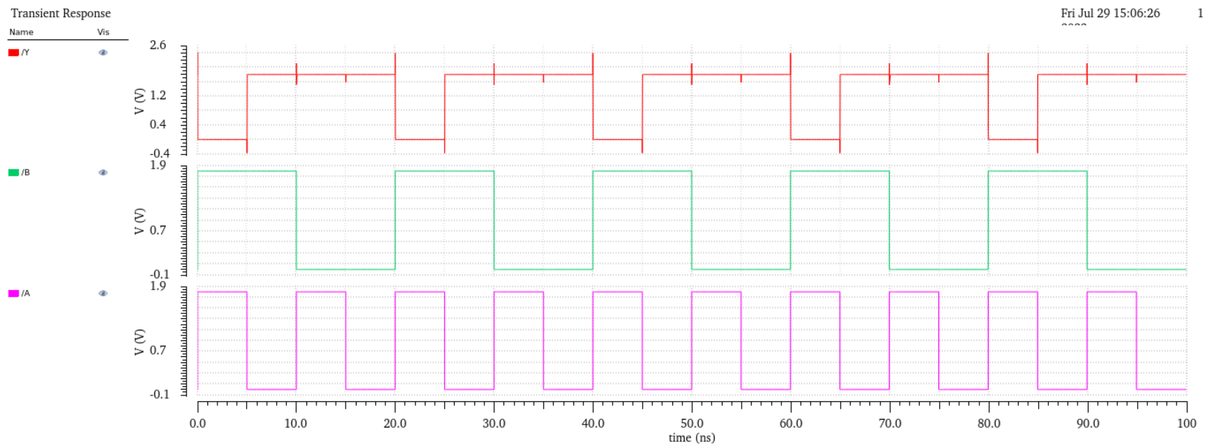
Circuit:

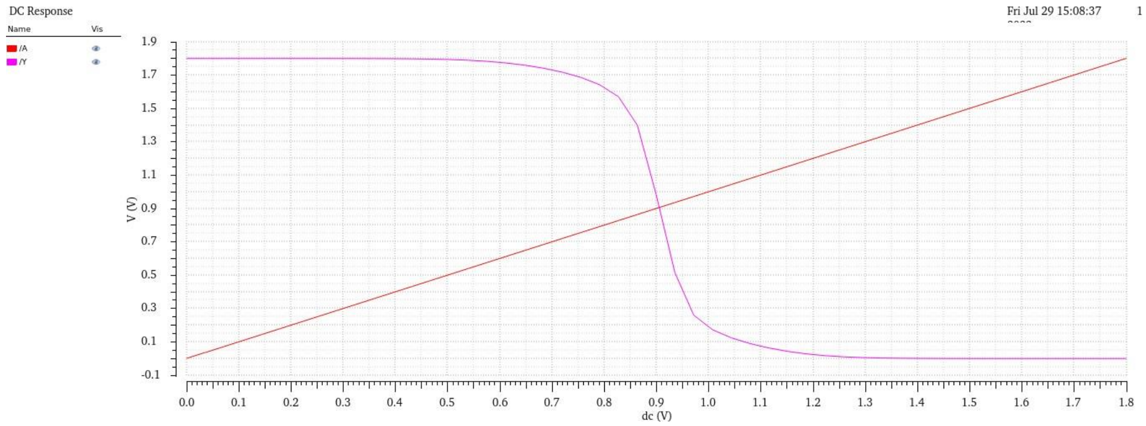


Testbench:



Simulation Results:





Transient and DC analysis

Result:

The design, simulation and verification of CMOS NAND Gate using Cadence tools at different VDD, Widths of NMOS and PMOS transistors for its Power and Delay analysis was performed.

1) VDD:________, Width of NMOS :____________: Width of PMOS:____________:Power:___________:Delay:______________

2) VDD:________, Width of NMOS :____________: Width of PMOS:____________:Power:___________:Delay:______________

3) VDD:________, Width of NMOS :____________: Width of PMOS:____________:Power:___________:Delay:______________

.

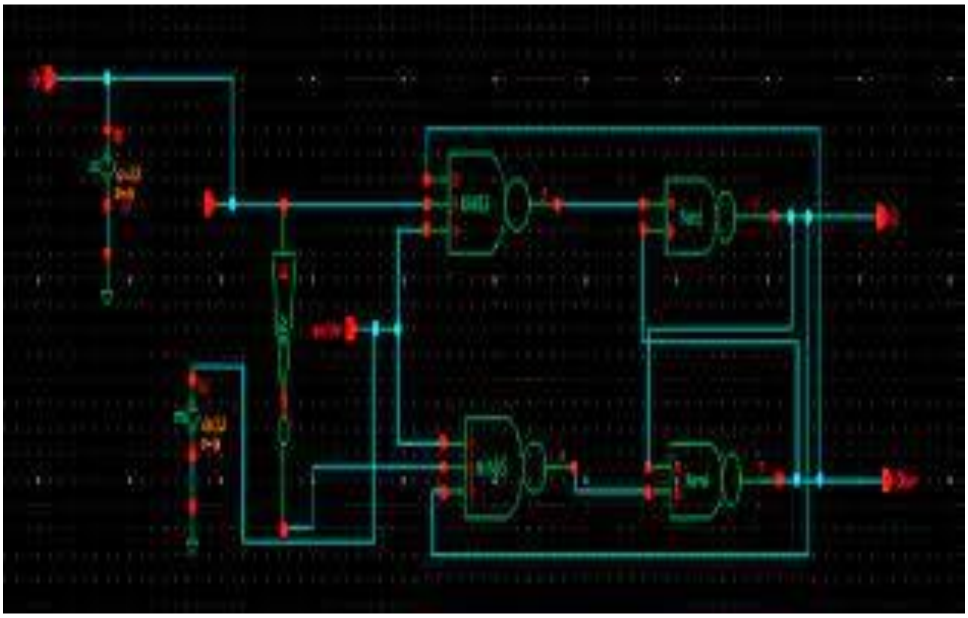
5.Design D- Flip Flop

AIM: To design, simulate and verify the operation of CMOS D-Flip flop using Cadence tools at different VDDs, Widths of NMOS and PMOS transistors by ensuring minimum Lengths and widths for its Power and Delay analysis.

Tools used:

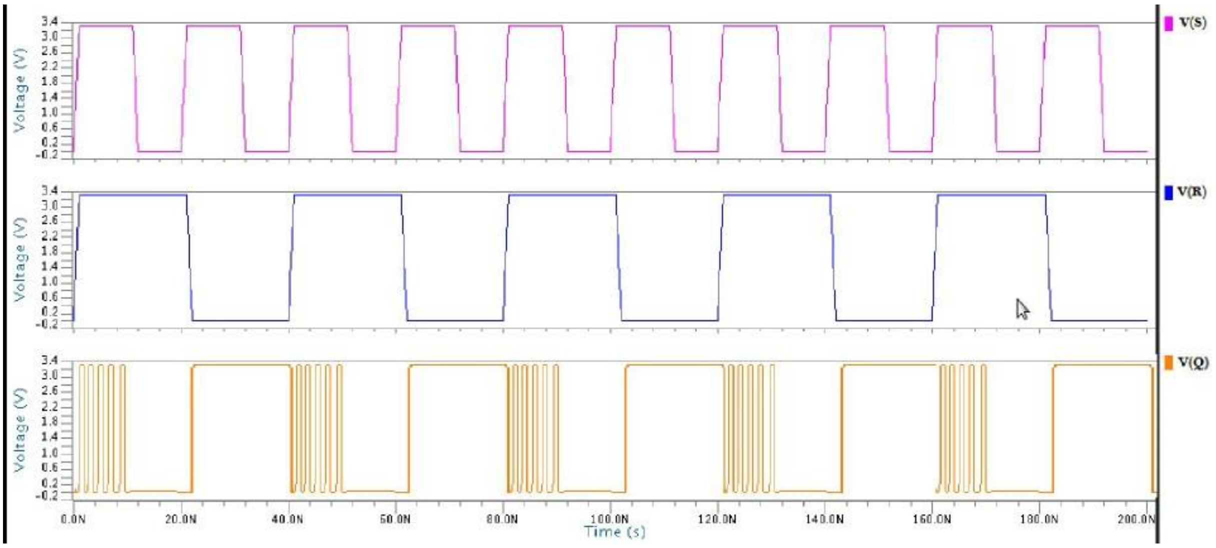
1. Virtuoso Tool for Schematic Designs
2. Spectre Tool for Simulation

Circuit:



Testbench:

Simulation Results:



Transient analysis

Result:

The design, simulation and verification of CMOS D-Flip flop using Cadence tools at different VDD, Widths of NMOS and PMOS transistors for its Power and Delay analysis was performed.

1) VDD:________, Width of NMOS :___________: Width of PMOS:___________:Power:__________:Delay:_____________

2) VDD:________, Width of NMOS :___________: Width of PMOS:___________:Power:__________:Delay:_____________

Tools used for ASIC Flow:

1. **INCISIVE** - Used for Functional Simulation
2. **GENUS** - Used for Synthesis and pre-Layout Timing Analysis
3. **INNOVUS** - Used for Physical Design

Getting Started :

1. Make sure the Licensing Server is switched ON and the client is connected to server."
2. Open the "**counter**" directory and make a right click to "**Open in Terminal**".
3. To open the tools to be used, type in the command "**csh**" (Press Enter) followed by "**source /home/install/cshrc**" <Or the path of tools whichever is applicable>.
4. A welcome string "**Welcome to Cadence Tool Suite**" appears indicating terminal ready to invoke Cadence Tools available for you.

Module 1: Creating an RTL Code

In order to create an RTL Code, you can open a text editor and type in your Verilog code or VHDL Code.

1. In the terminal, type in "**gedit <filename>.v [OR] <filename>.vhdl**". The file extensions depends on the type of RTL Code you write as shown.

2. Similarly, using same command, Test Bench also could be written as shown below.

```
`timescale 1ns/1ps
module counter(clk,m,rst,count);
input clk,m,rst;
output reg [7:0] count;
always@(posedge clk or negedge rst)
begin
if(!rst)
count=0;
else if(m)
count=count+1;
else
count=count-1;
end
endmodule
```

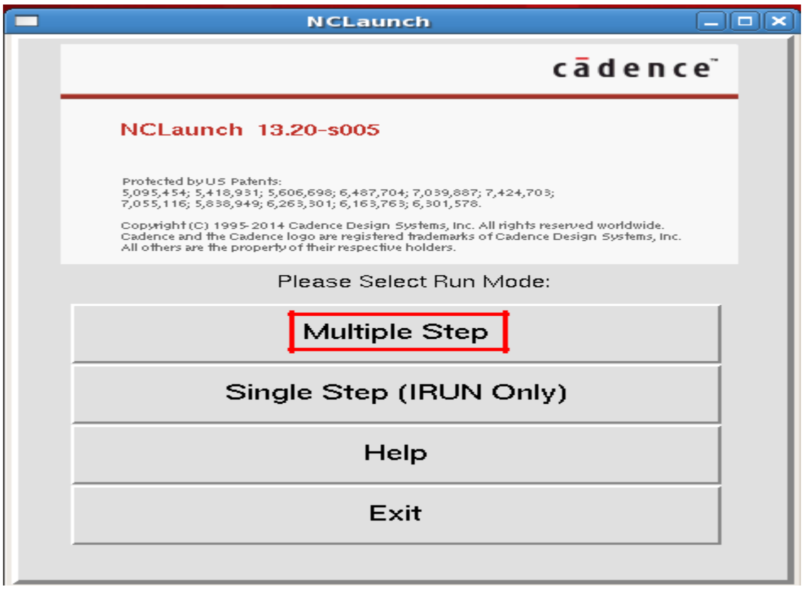
RTL Code for a 16-bit Synchronous Up-Down Counter

```
`timescale 1ns/1ps
module counter_test;
reg clk, rst,m;
wire [15:0] count;
initial
begin
clk=0;
rst=0;#25;
rst=1;
end
initial
begin
m=1;
#600 m=0;
rst=0;#25;
rst=1;
#500 m=0;
end
initial $sdf_annotate ("delays.sdf" , counter_test.counter1, ,"sdf.log");
initial $sdf_annotate ("counter.sdf" , counter_test.counter1, ,"sdf.log");

counter counter1(clk,m,rst, count);

always #5 clk=~clk;

initial $monitor("Time=%t rst=%b clk=%b count=%b", $time,rst,clk,count);

initial
#1400 $finish;

endmodule
```

Test Bench for the Up-Down Counter

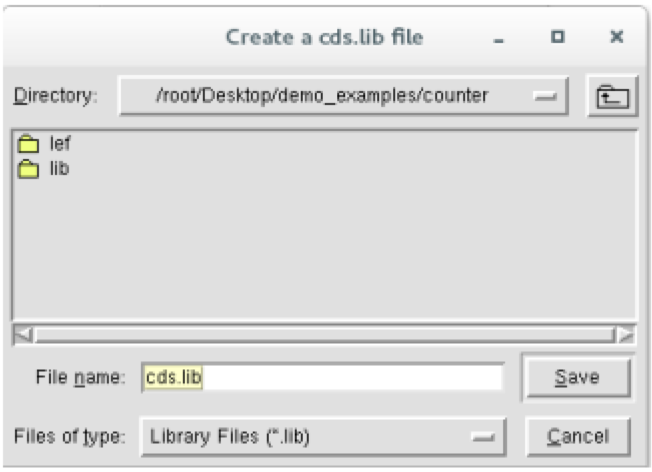
Module 2: Functional Simulation

1. To perform Functional Simulation, “**Incisive**” tool is to be used.
2. In your terminal, type the command “**nclaunch -new**” to open the tool.

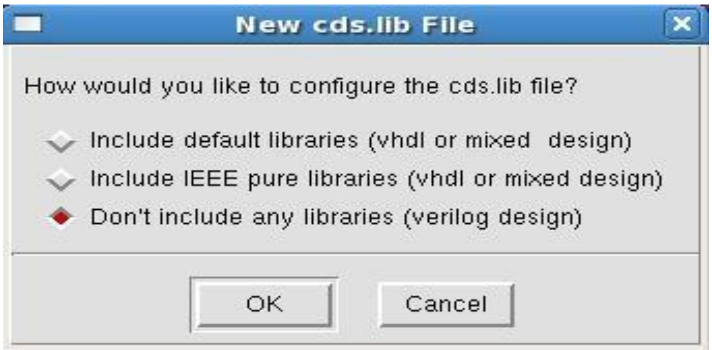


3. Select “**Multiple Step**”. And then select “**Create cds.lib**”

Note : The ‘-new’ switch is used only for the first time the design is being run. For the next time on wards, the command to be used could be ‘nclaunch’ only.

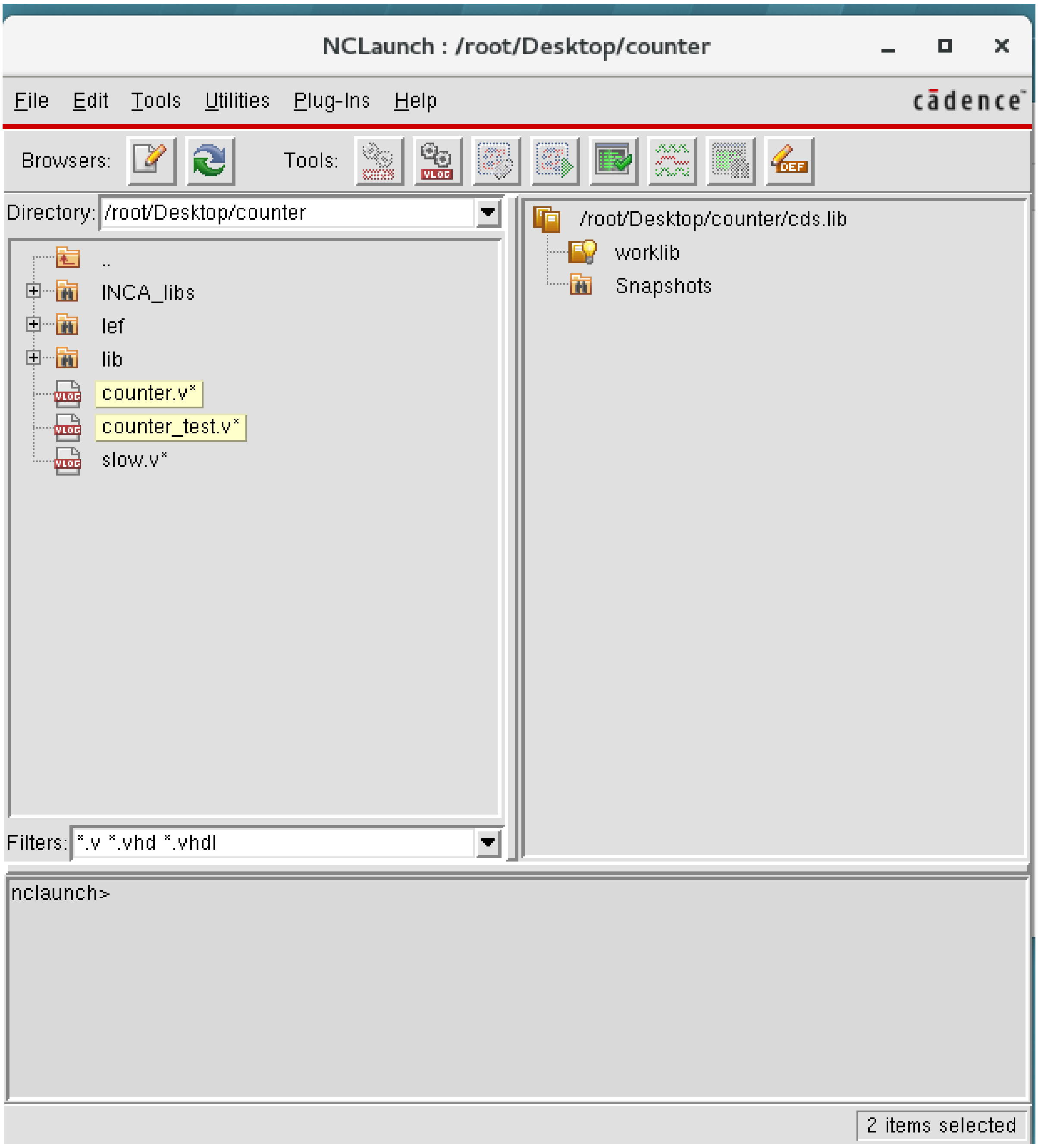


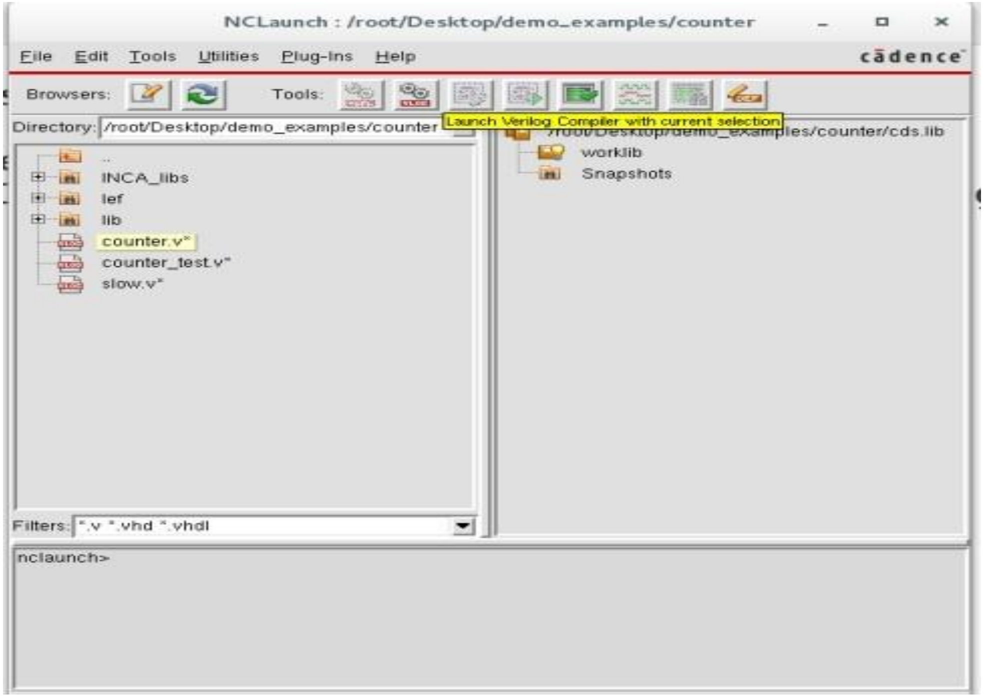
4. Save the cds.lib file. It is a tool file that holds the design location information for easy access by the tool.

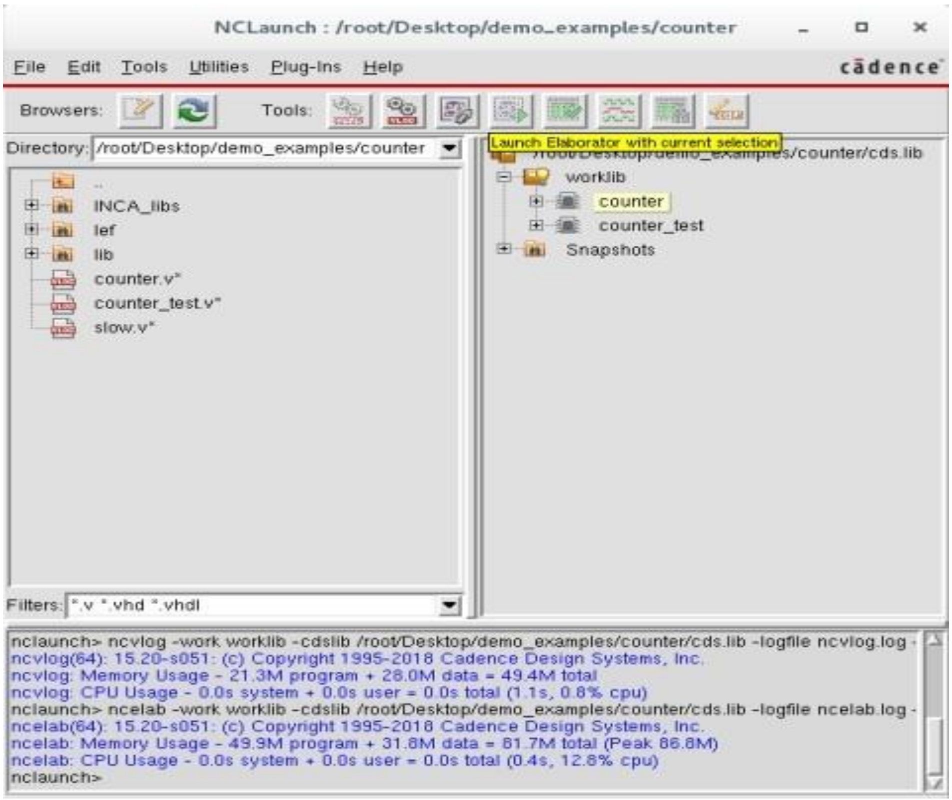


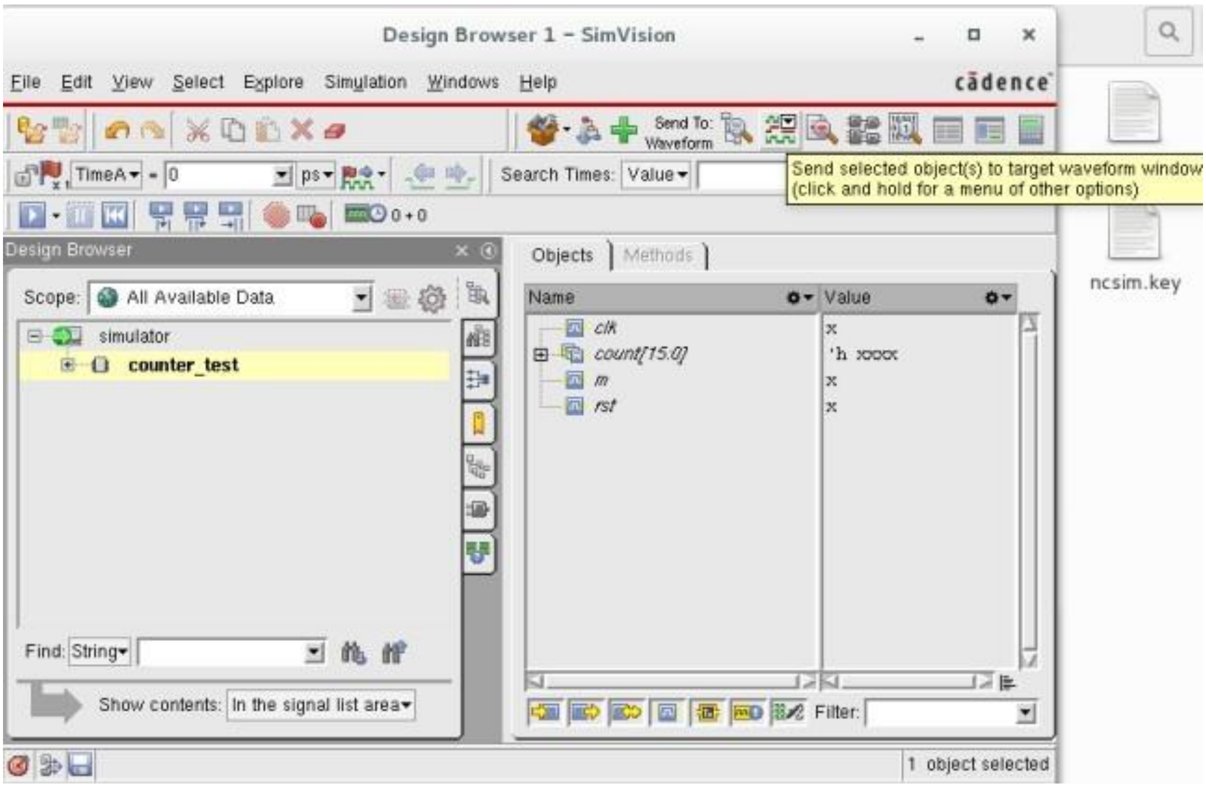
5. Based on the Libraries available and the type of RTL Code written, one of the three shown above is to be selected. Cadence tool suite provides default gpdk libraries. Here, counter RTL is of Verilog Format and hence third option is selected.
6. A new pop-up “nclaunch” opens which will contain all the .v and .vhdl files as per the cds.lib file created.
7. **Functional Simulation using Cadence runs in 3 stages:**
   ➔ **Compilation of Verilog/VHDL Code and/or Test Bench**
   ➔ **Elaboration of the Code & Test Bench Compiled**
   ➔ **Simulating the Test Bench or Top Module[in absence of Test Bench]**

8. A set of tools are shown in the nclaunch window which refer to **VHDL Compiler, Verilog Compiler, Elaborator, Simulator** corresponding from Left To Right.
9. Select the .v or .vhdl files to be compiled and launch Compiler. On successful completion of compilation, on the Right hand Side, the modules appear under “**Worklib**” .
10. Select the Module under Worklib and “**Launch Elaborator**” . On successful completion of Elaboration, “**Snapshots**” are generated.
11. Select the Test Bench under snapshots and “**Launch Simulator**” .
12. The above steps are depicted under following snapshosts.


NCLaunch : /root/Desktop/counter
File Edit Tools Utilities Plug-Ins Help
cadence
Browsers:
Tools:
Directory: /root/Desktop/counter
..
INCA_libs
lef
lib
counter.v*
counter_test.v*
slow.v*
Filters: *.v *.vhd *.vhdl
/root/Desktop/counter/cds.lib
worklib
Snapshots
nclaunch>
2 items selected


NCLaunch : /root/Desktop/demo_examples/counter
File Edit Tools Utilities Plug-Ins Help
cādence
Browsers:
Tools:
Directory: /root/Desktop/demo_examples/counter
Launch Verilog Compiler with current selection
worklib
Snapshots
INCA_libs
lef
lib
counter.v*
counter_test.v*
slow.v*
Filters: *.v *.vhd *.vhdl
nclaunch>

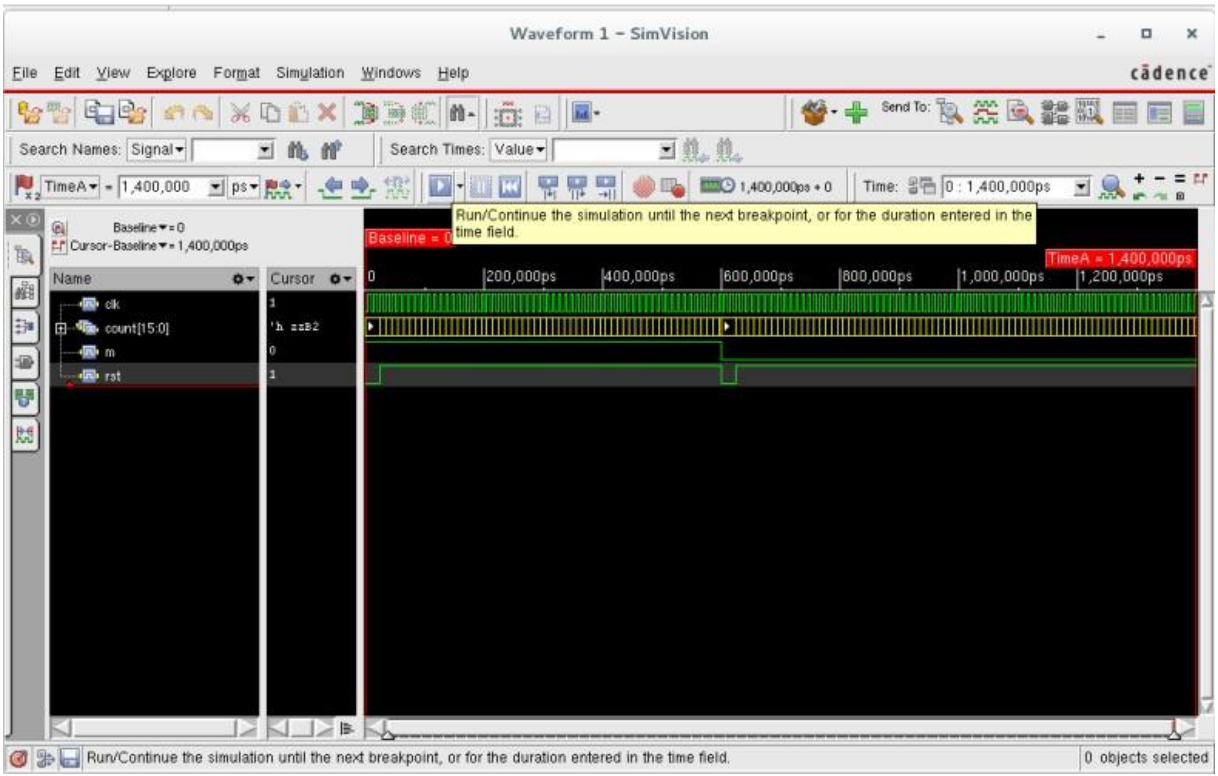

NCLaunch : /root/Desktop/demo_examples/counter
File Edit Tools Utilities Plug-Ins Help
cādence
Browsers:
Tools:
Directory: /root/Desktop/demo_examples/counter
Launch Elaborator with current selection
/counter/cds.lib
worklib
counter
counter_test
Snapshots
INCA_libs
lef
lib
counter.v*
counter_test.v*
slow.v*
Filters: *.v *.vhd *.vhdl
nclaunch> ncvlog -work worklib -cdslib /root/Desktop/demo_examples/counter/cds.lib -logfile ncvlog.log -
ncvlog(64): 15.20-s051: (c) Copyright 1995-2018 Cadence Design Systems, Inc.
ncvlog: Memory Usage - 21.3M program + 28.0M data = 49.4M total
ncvlog: CPU Usage - 0.0s system + 0.0s user = 0.0s total (1.1s, 0.8% cpu)
nclaunch> ncelab -work worklib -cdslib /root/Desktop/demo_examples/counter/cds.lib -logfile ncelab.log -
ncelab(64): 15.20-s051: (c) Copyright 1995-2018 Cadence Design Systems, Inc.
ncelab: Memory Usage - 49.9M program + 31.8M data = 81.7M total (Peak 86.8M)
ncelab: CPU Usage - 0.0s system + 0.0s user = 0.0s total (0.4s, 12.8% cpu)
nclaunch>



The Design Browser pops-up and The Test Bench Module name can be seen on the left and the Pin list on the right when selected. For Simulation, The number of Pins / Ports to be simulated can be selected.

Make a right click on the selected and Select “Send to Waveform Window”.

In the waveform window, we can see different ports in the design. Now click on the Run simulation key to start the simulation. Use the ‘pause’ key to interrupt or stop the simulation. Use different options like zoom in, zoom out etc to analyze the plot.



Module 3: Synthesis

Inputs for Synthesis :

1. **RTL Code (.v or .vhdl)**
2. **Chip Level SDC (System Design Constraints)**
3. **Liberty Files (.lib)**

Expected Outputs of Synthesis :

1. **Gate Level Netlist**
2. **Block Level Netlist**
3. **Timing, Area, Power reports**

Synthesis is a 3-stage process which converts Virtual RTL Logic into Physical Gates in order to give a Physical Shape to the design through Physical Design.

Synthesis runs in following stages :

* Translation - RTL Codes are compiled
* Elaboration / Mapping - Pieces of Logic are replaced with corresponding Gates from Libraries with same Functionality
* Optimization - Tool tries to reduce cell count without affecting the functionality

To run the synthesis, the following script can be used.

```
set_db lib_search_path ./lib/90
set_db library slow.lib
set_db hdl_search_path /
read_hdl counter.v
elaborate
read_sdc constraints_top.sdc
synthesize -to_mapped -effort medium
write_sdf -timescale ns -nonegchecks -recrem split -edges check_edge > delays.sdf

write_hdl > counter_netlist.v
write_sdc > counter_sdc.sdc


gui_show
report timing > counter_timing.rep
report power > counter_power.rep
report area > counter_cell.rep
report messages > counter_message.rep
```

Chip Level SDC is as follows :

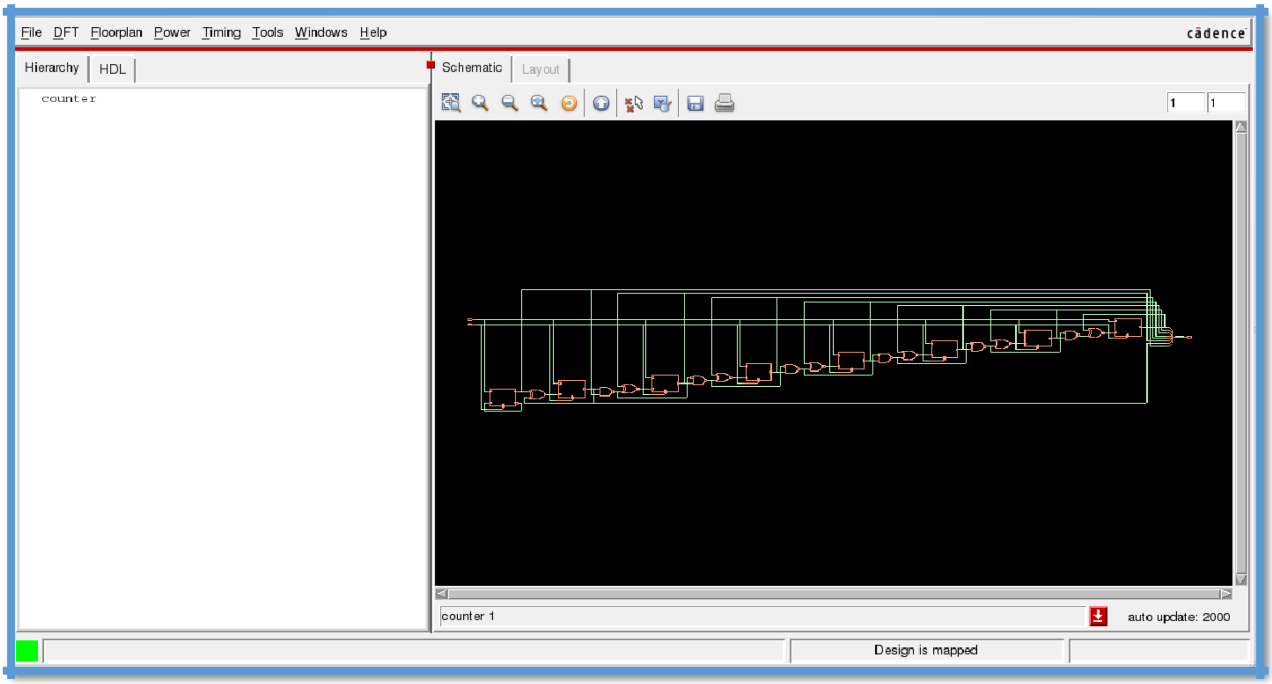
```
create_clock -name clk -period 2 -waveform {0 1} [get_ports "clk"]
set_clock_transition -rise 0.1 [get_clocks "clk"]
set_clock_transition -fall 0.1 [get_clocks "clk"]
set_clock_uncertainty 0.01 [get_ports "clk"]
set_input_delay -max 1.0 [get_ports "rst"] -clock [get_clocks "clk"]
set_output_delay -max 1.0 [get_ports "count"] -clock [get_clocks "clk"]
```

Close out all INCISIVE windows and in the same terminal type in the following command to run Synthesis.

genus -f rc_script.tcl

The tcl [Tool Command Language] script runs executing each command one after the other.

A window of Genus GUI pops – up with the top hier cell on the left top. Make a right click and select Schematic Viewer → In Main.



The Gate level Circuit that implements the RTL Logic can be seen and analysed.

As from the script, Block Level SDC, Gate Level Netlist, Timing, Power and Area reports are generated which are readable.

The Timing report gives the path with Worst timing.

The area report gives Cell count and Total area occupied by them.

The total power consumed by those cells are given in Power report.

Module 4 : Physical Design

Mandatory Inputs for PD :

1. **Gate Level Netlist [Output of Synthesis]**
2. **Block Level SDC [Output of Synthesis]**
3. **Liberty Files (.lib)**
4. **LEF Files (Layer Exchange Format)**

Expected Outputs from PD :

1. **GDS II File (Graphical Data Stream for Information Interchange – Feed In for Fabrication Unit).**

Close out all windows relating to Genus and in the terminal, type the command

innovus (Press Enter)

For Innovus tool, a GUI opens and also the terminal enters into innovus command prompt where in the tool commands can be entered.

Physical Design involves 5 stages as following :

After Importing Design,
* Floor Planning
* Power Planning
* Placement
* CTS (Clock Tree Synthesis)
* Routing

Module 4.1 : Importing Design

To Import Design, all the Mandatory Inputs are to be loaded and this can be done using script files named with .globals and .view/.tcl

```
###############################################################
#  Generated by:      Cadence Encounter 13.23-s047_1
#  OS:                Linux x86_64(Host ID cadence)
#  Generated on:      Tue May 24 02:16:38 2016
#  Design:
#  Command:           save_global Default.globals
###############################################################
#
# Version 1.1
#

set ::TimeLib::tsgMarkCellLatchConstructFlag 1
set conf_qxconf_file {NULL}
set conf_qxlib_file {NULL}
set defHierChar {/}
set init_design_settop 0
set init_gnd_net {VSS}
set init_lef_file {lef/gsclib090_translated.lef lef/gsclib090_translated_ref.lef}
set init_mmmc_file {Default.view}
set init_pwr_net {VDD}
set init_verilog {counter_netlist.v}
set lsgOCPGainMult 1.000000
set pegDefaultResScaleFactor 1.000000
set pegDetailResScaleFactor 1.000000
```

Globals File to import design using Mandatory Inputs

The Globals file reads in the LEF's and Gate Level Netlist and .view file implicitly.

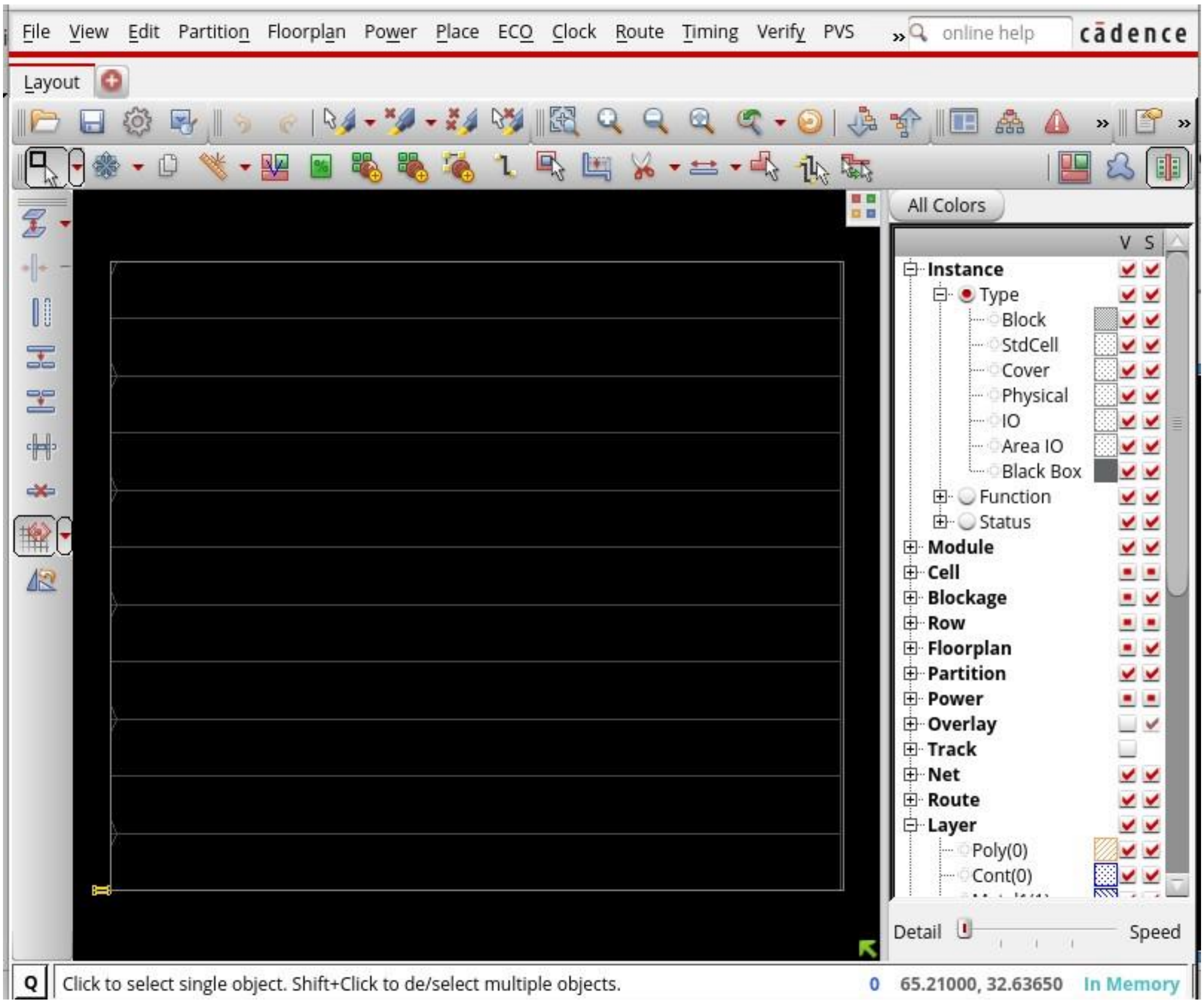
```
# Version:1.0 MMMC View Definition File
# Do Not Remove Above Line
create_library_set -name MAX_timing -timing {/root/Desktop/counter/lib/90/slow.lib}
create_library_set -name MIn_timing -timing {/root/Desktop/counter/lib/90/fast.lib}
create_constraint_mode -name Constraints -sdc_files {counter_sdc.sdc}
create_delay_corner -name Max_delay -library_set {MAX_timing}
create_delay_corner -name Min_delay -library_set {MIn_timing}
create_analysis_view -name Worst -constraint_mode {Constraints} -delay_corner {Max_delay}
create_analysis_view -name best -constraint_mode {Constraints} -delay_corner {Min_delay}
set_analysis_view -setup {Worst} -hold {best}
```

The .view file reads Liberty Files and Block Level SDC to create various PVT Corners for analysis.

In the terminal command prompt, type the commands as shown. The design is imported and “Core Area” is calculated by tool and shown on GUI.

```
File  Edit  View  Search  Terminal  Help
Options:
Date:           Tue May 14 12:13:39 2019
Host:           KrishnaCadence (x86_64 w/Linux 3.10.0-862.el7.x86_64) (2cores*4c
pus*Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz 3072KB)
OS:             Red Hat Enterprise Linux Server release 7.5 (Maipo)

License:
12:13:39 (cdslmd) OUT: "Innovus_Impl_System" root@KrishnaCadence
                invs    Innovus Implementation System   17.1    checkout succeed
ed
                8 CPU jobs allowed with the current license(s). Use setMultiCpuU
sage to set your required CPU count.
Create and set the environment variable TMPDIR to /tmp/innovus_temp_6984_Krishna
Cadence_root_ohoasS.

Change the soft stacksize limit to 0.2%RAM (31 mbytes). Set global soft_stack_si
ze_limit to change the value.

**INFO:  MMMC transition support version v31-84

[INFO] Loading PVS 16.12-s208 fill procedures
innovus 1> source Default.globals
1.000000
innovus 2> init_design
```

The Horizontal Lines on the GUI across the Core Area are alternative VDD and VSS tracks and Standard Cell Placement Rows.



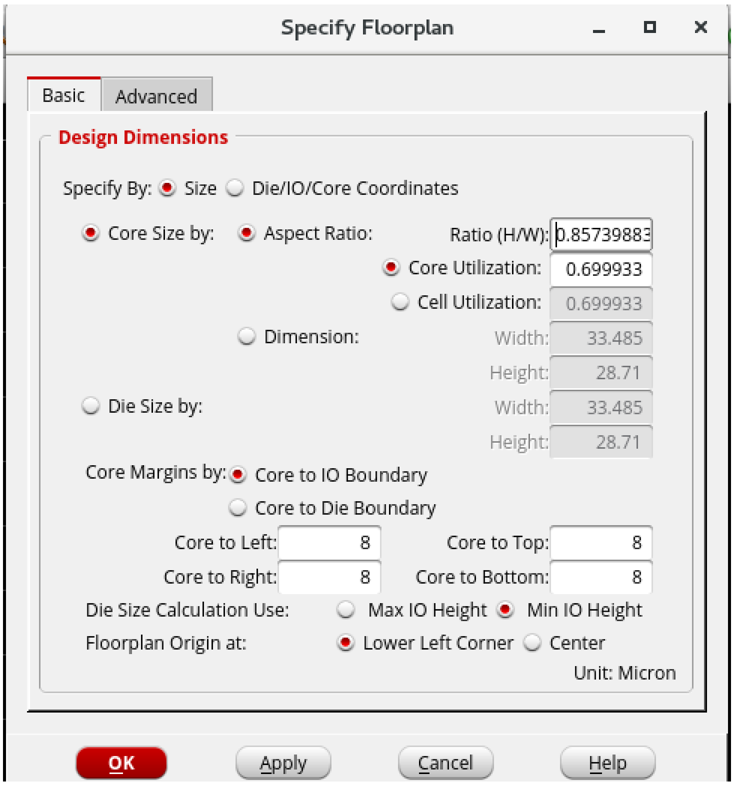
Module 4.2 : Floorplan

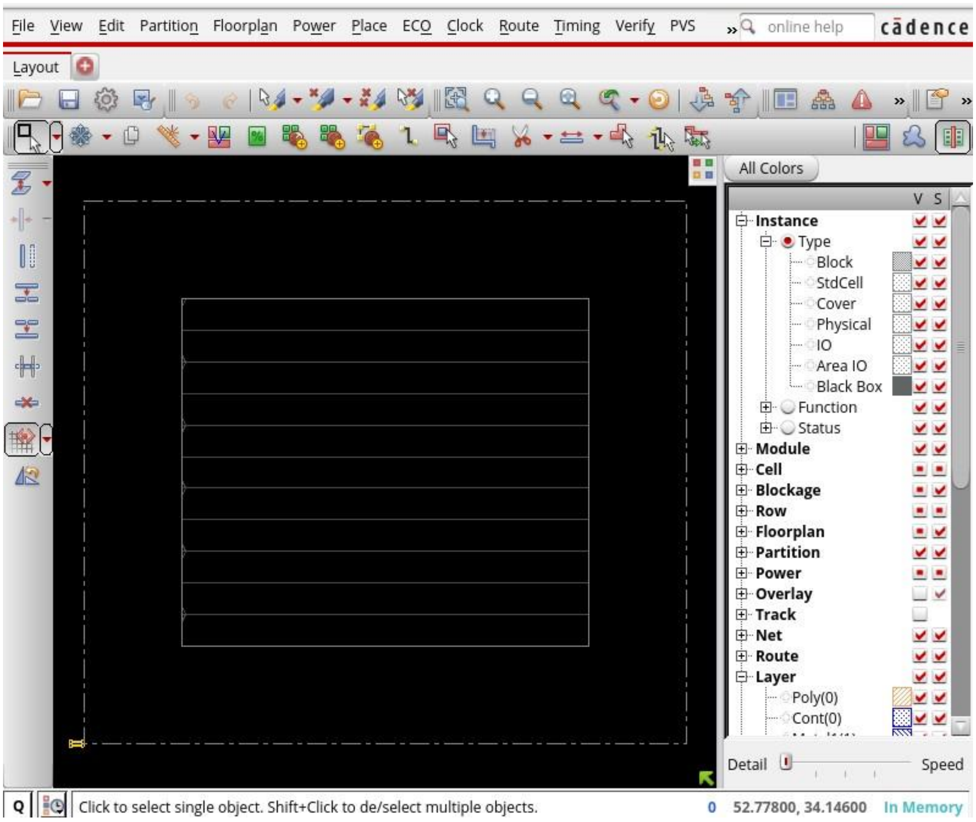
Steps under Floorplan :

1. **Aspect Ratio** [Ratio of Vertical Height to Horizontal Width of Core]
2. **Core Utilisation** [The total Core Area % to be used for Floor Planning]
3. **Channel Spacing between Core Boundary to IO Boundary**

Select **Floorplan → Specify Floorplan** to modify/add concerned values to the above Factors.

On adding/modifying the concerned values, the core area is also modified.





The Yellow patch on the Left Bottom are the group of "Unassigned pins" which are to be placed along the IO Boundary along with the Standard Cells [Gates].

Module 4.3 : Power Planning

Steps under Power Planning :

1. **Connect Global Net Connects**
2. **Adding Power Rings**
3. **Adding Power Strings**
4. **Special Route**

During the stage of Importing Design, under the Globals file, Two command lines state the names of Power and Ground Nets.

However, in order to Current flow through these Power nets, they are to converge at a point, preferably a common net connected to a Pin.

Under **Connect Global Net Connects**, we create two pins, one for VDD and one for VSS connecting them to corresponding Global Nets as mentioned in Globals file.

Select **Power → Connect Global Nets..** to create "Pin" and "Connect to Global Net" as shown and use "Add to list".

Click on "Apply" to direct the tool in enforcing the Pins and Net connects to Design and then Close the window.

In order to Tap in Power from a distant Power supply, Wider Nets and Parallel connections improve efficiency. Moreover, the cells that would be placed inside the core area are expected to have shorter Nets for lower resistance.

Hence Power Rings [Around Core Boundary] and Power Stripes [Across Core Boundary] are added which satisfies the above conditions.

Select **Power → Power Planning → Add Rings** to add Power rings 'around Core Boundary'.

* Select the Nets from Browse option OR Directly type in the Global Net Names separated by a space being Case and Spelling Senstive.

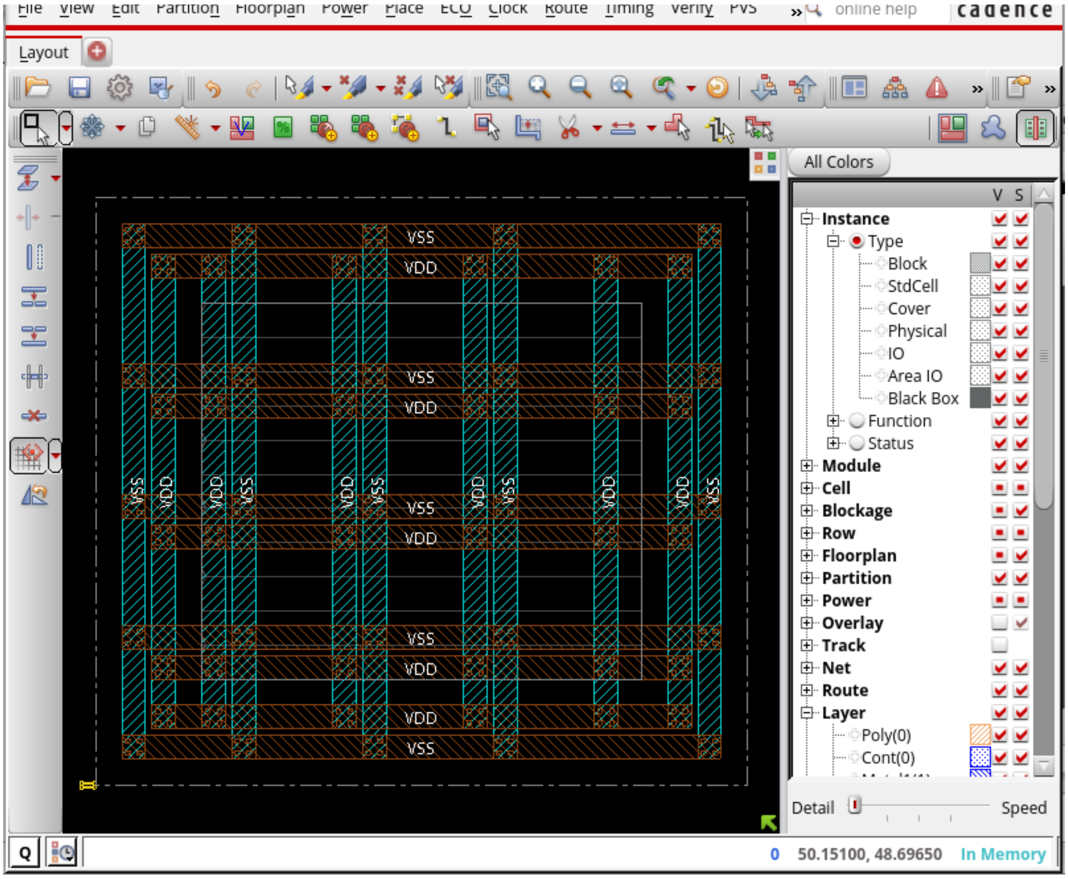
* Select the Highest Metals marked 'H' [Horizontal] for Top and Bottom and Metals marked 'V' [Vertical] for Right and Bottom. This is because Highest metals have Highest Widths and thus Lowest Resistance.

* Click on Update after the selection and "Set Offset : Center in Channel" in order to get the Minimum Width and Minimum Spacing of the corresponding Metals and then Click "OK".

* Similarly, Power Stripes are added using similar content to that of Power Rings.

Factors to be considered under Power Stripes :

* Nets
* Metal and It's Direction
* Width and Spacing [Updated]
* Set to Set Distance = (Minimum Width of Metal + Min. Spacing) x 2

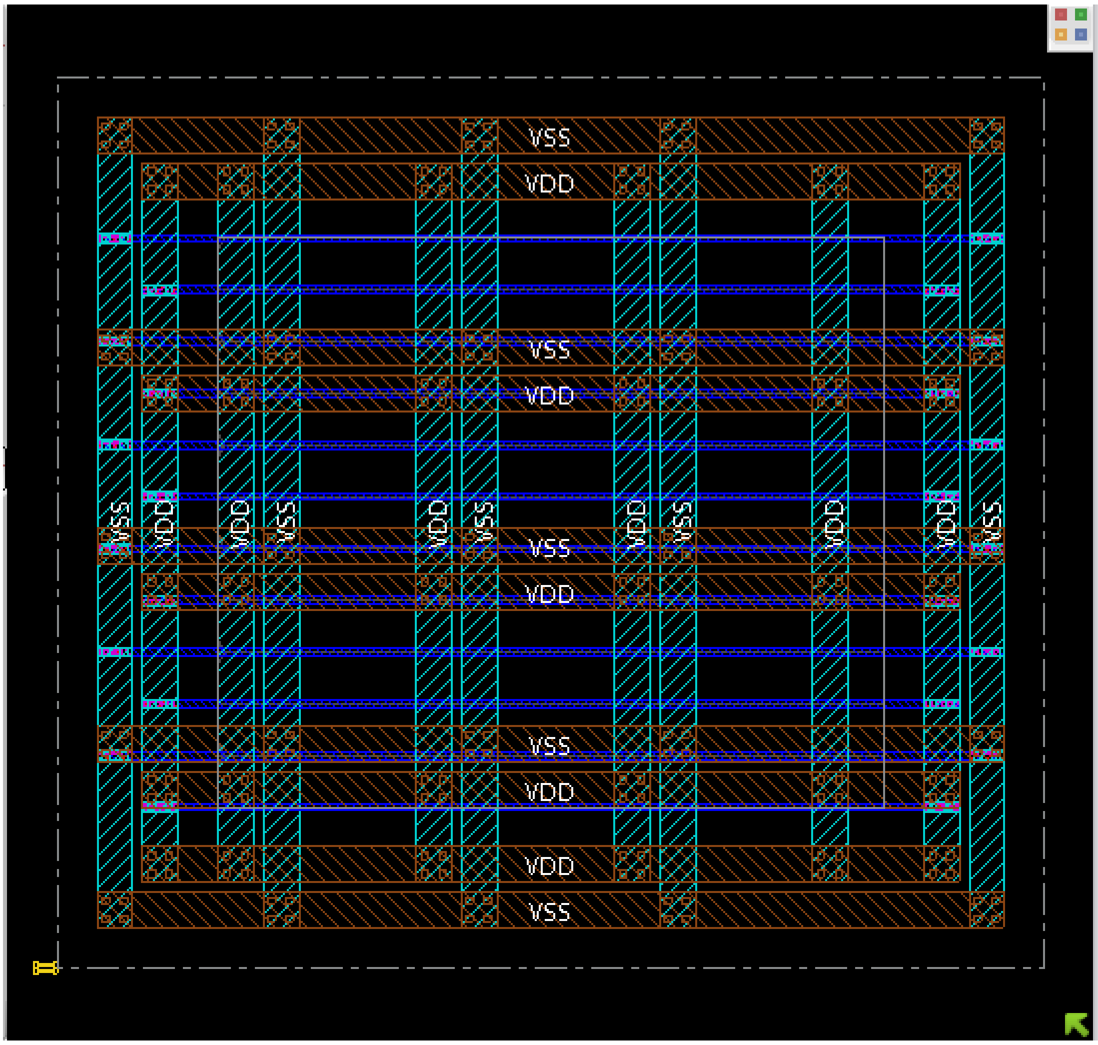


On adding Power Stripes, The Power mesh setup is complete as shown. However, There are no Vias that could connect Metal 9 or Metal 8 directly with Metal 1 [VDD or VSS of Standard Cells are generally made up of Metal 1].

The connection between the Highest and Lowest Metals is done through Stacking of Vias done using “Special Route”.

To perform Special Route, **Select Route → Special Route → Add Nets → OK.**

After the Special Route is complete, all the Standard Cell Rows turn to the Color coded for Metal 1 as shown below.

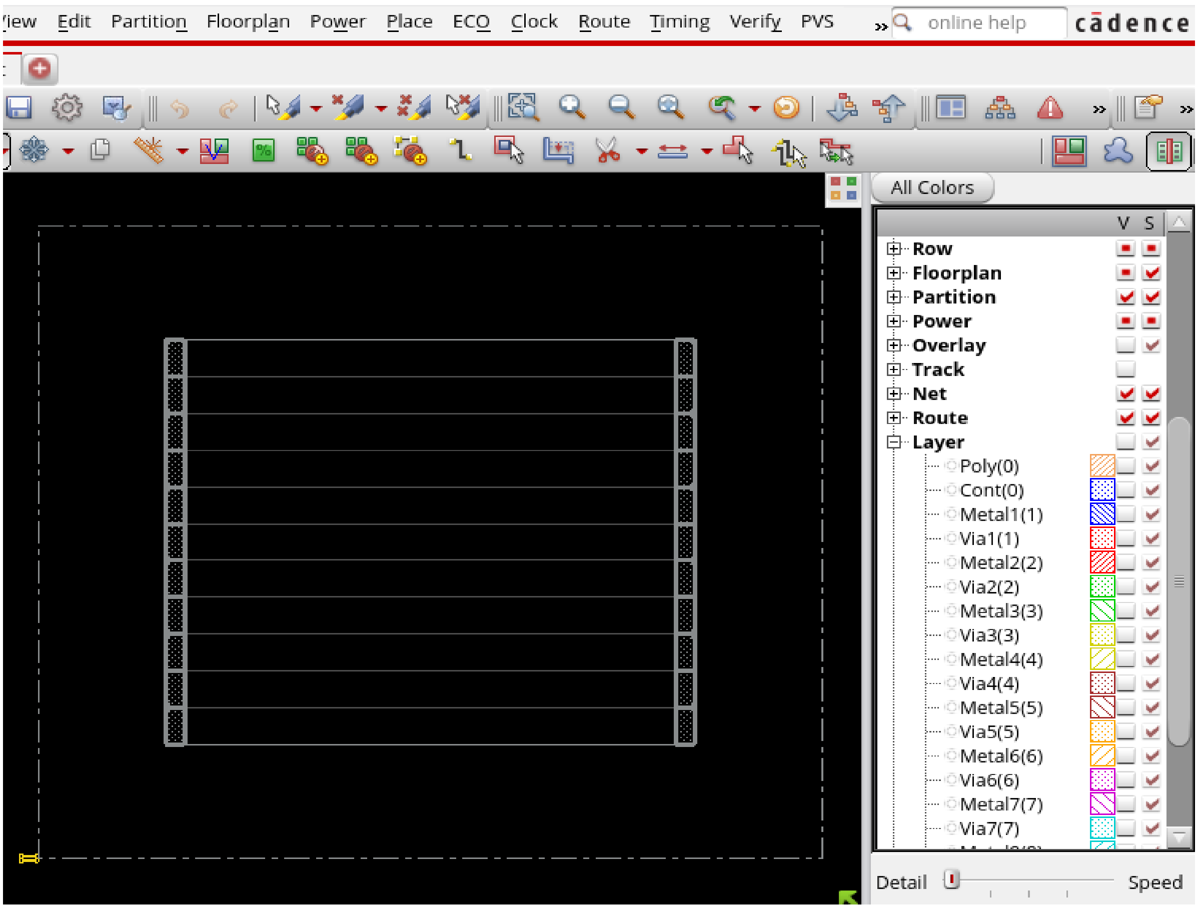


The complete Power Planning process makes sure Every Standard Cell receives enough power to operate smoothly.

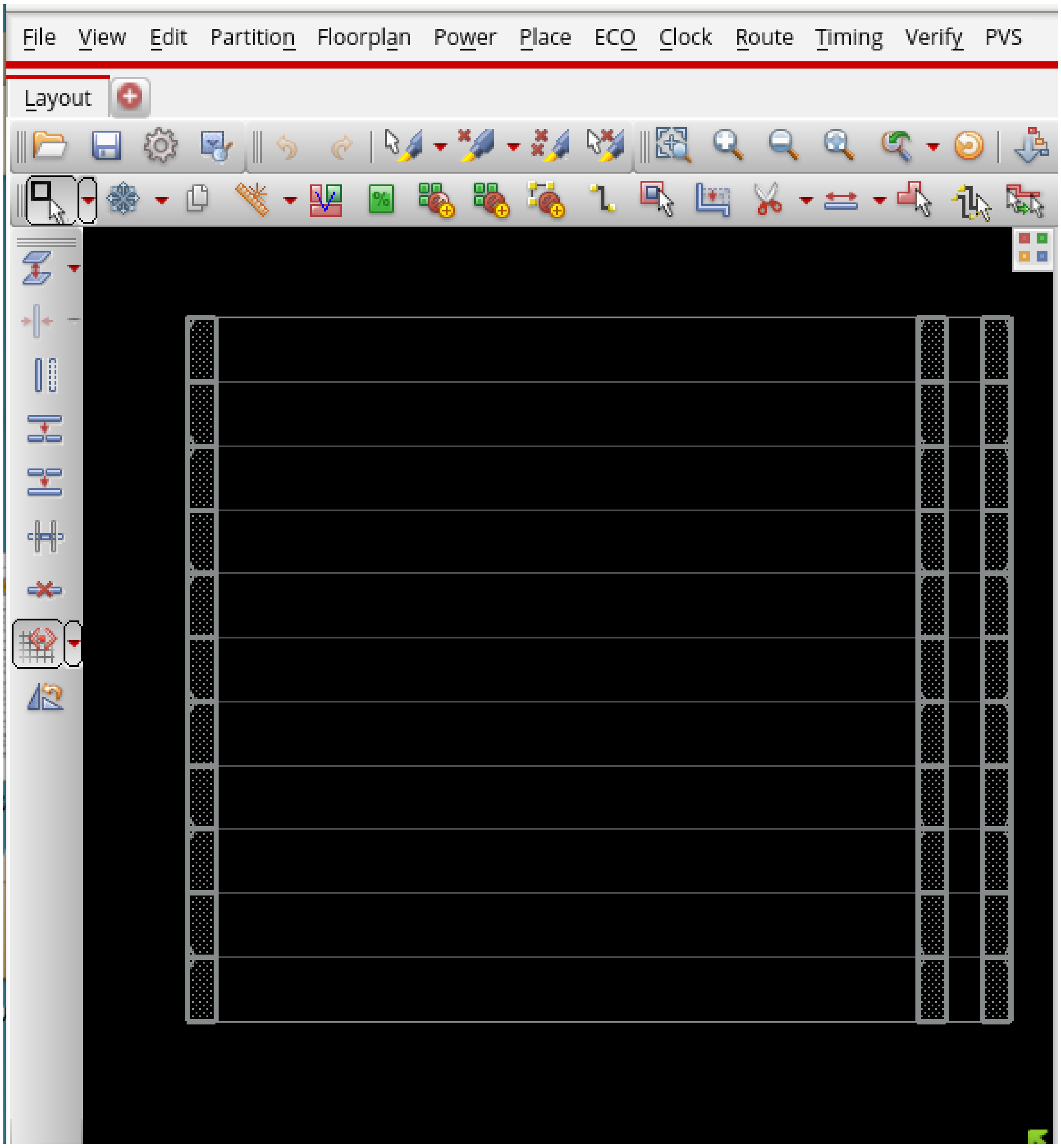
Module 4.4.1 : Pre - Placement

* After Power Planning, a few Physical Cells are added namely, **End Caps and Well Taps**.
* **End Caps :** They are Physical Cells which are added to the Left and Right Core Boundaries acting as blockages to avoid Standard Cells from moving out of boundary.
* **Well Taps :** They act like Shunt Resistance to avoid Latch Up effects.

1. To add End Caps, Select **Place → Physical Cell → Add End Caps** and "Select" the FILL's from the available list. Higher Fills have Higher Widths. As shown Below, The End Caps are added below your Power Mesh.

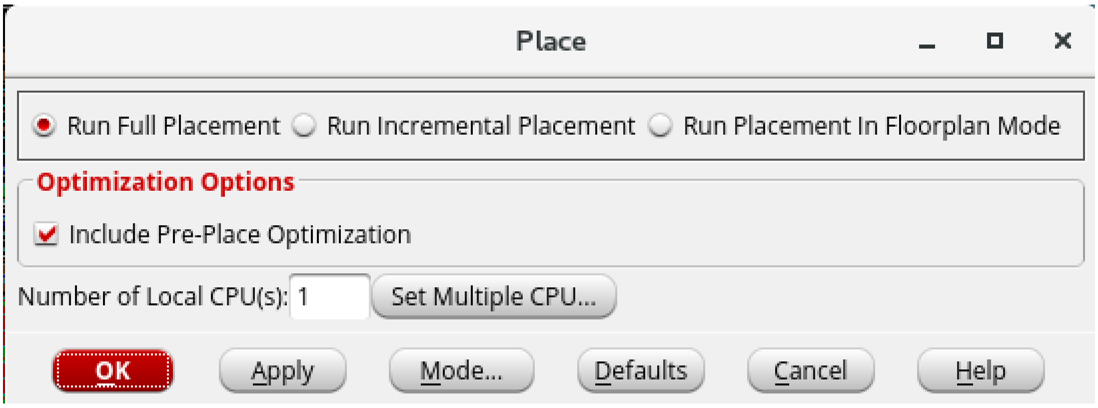


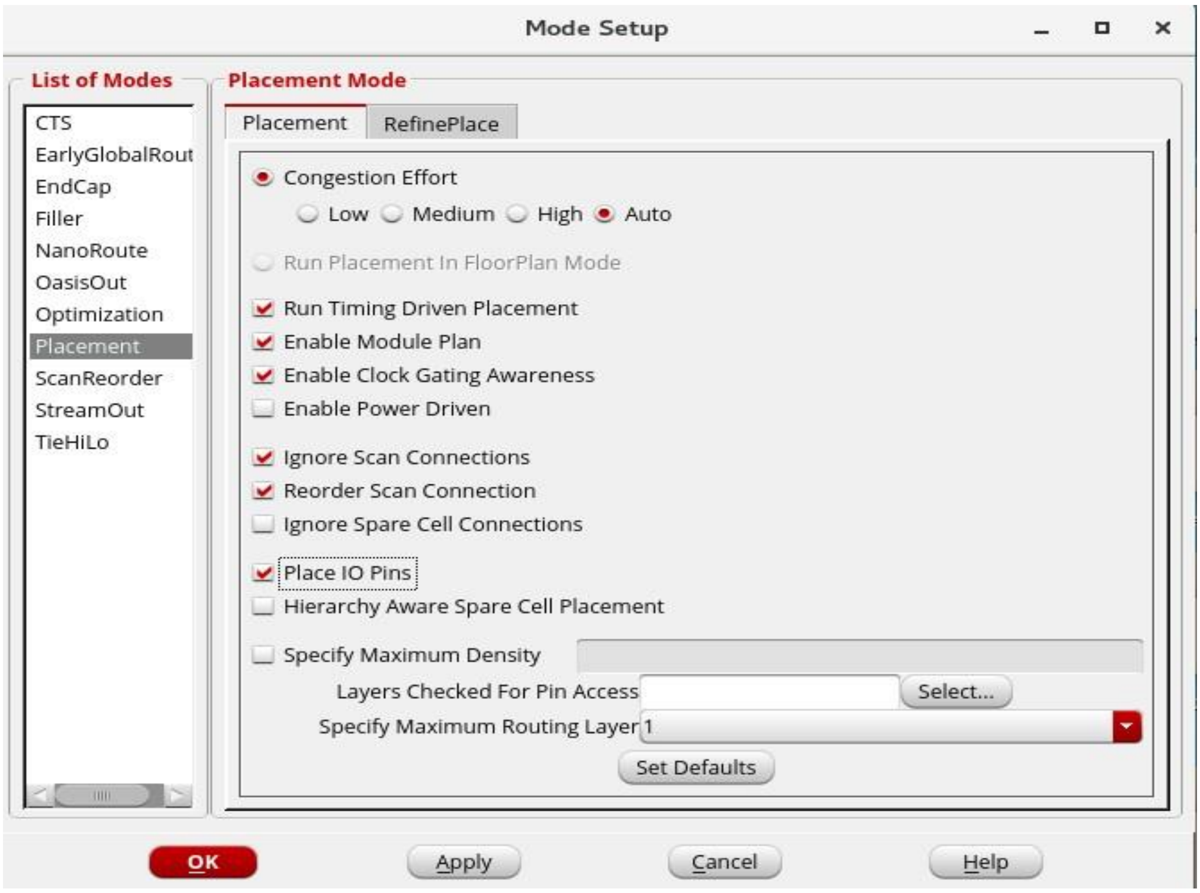
2. To add Well Taps, Select **Place → Physical Cell → Add Well Tap → Select → FillX** [X → Strength of Fill = 1,2,4 etc] **→ Distance Interval** [Could be given range of 30-45u] **→ OK**



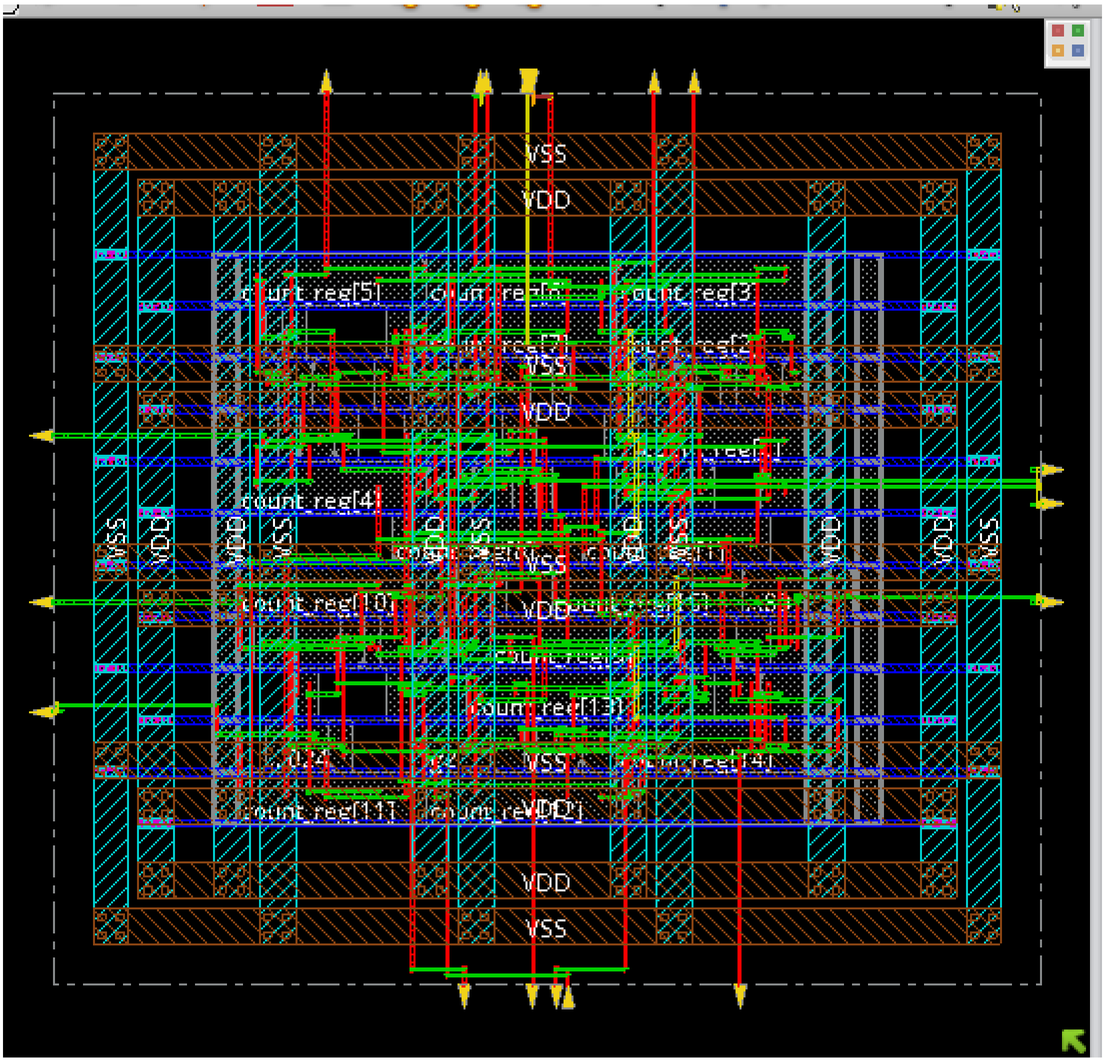
Module 4.5 : Placement

1. The Placement stage deals with Placing of Standard Cells as well as Pins.
2. Select **Place → Place Standard Cell → Run Full Placement → Mode → Enable 'Place I/O Pins' → OK → OK .**

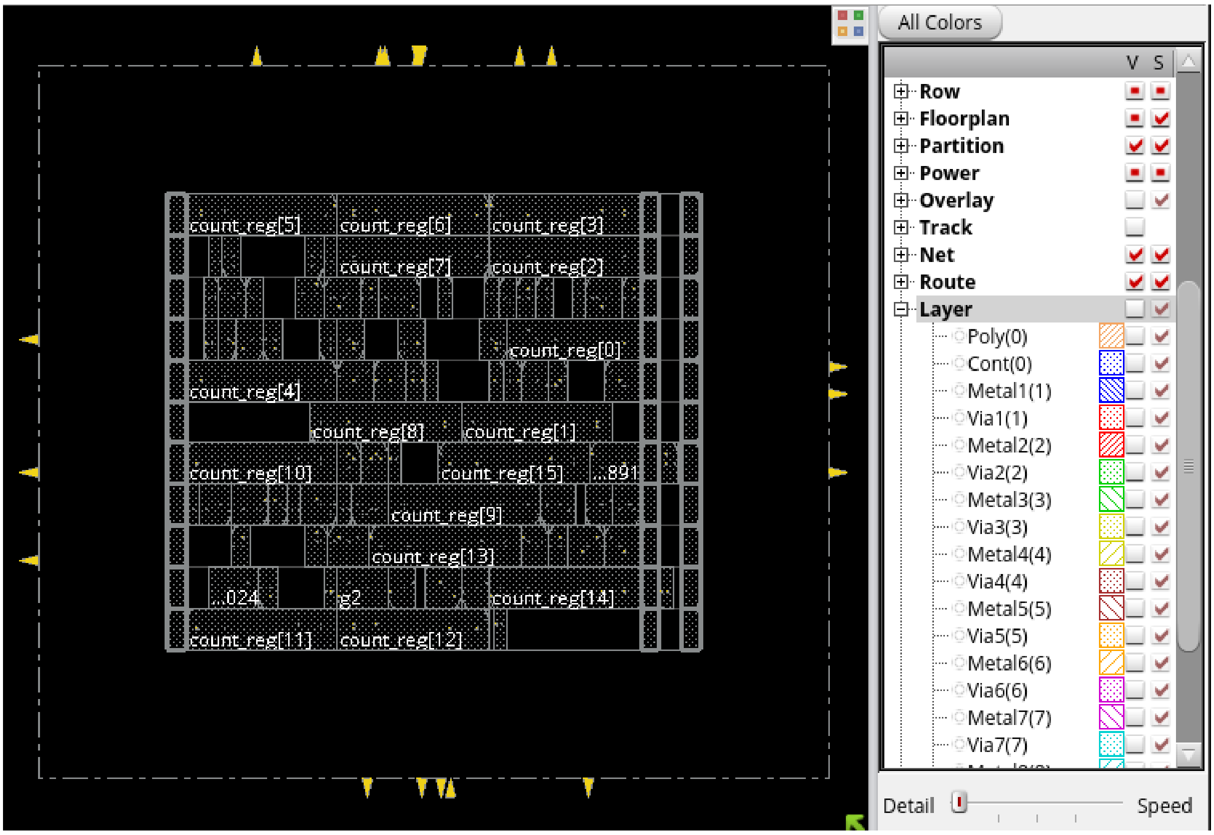




All the Standard Cells and Pins are placed as per the communication between them, i.e., Two communicating Cells are placed as close as possible so that shorter Net lengths can be used for connections as Shorter Net Lengths enable Better Timing Results.



Placed Design



Standard Cells Placed

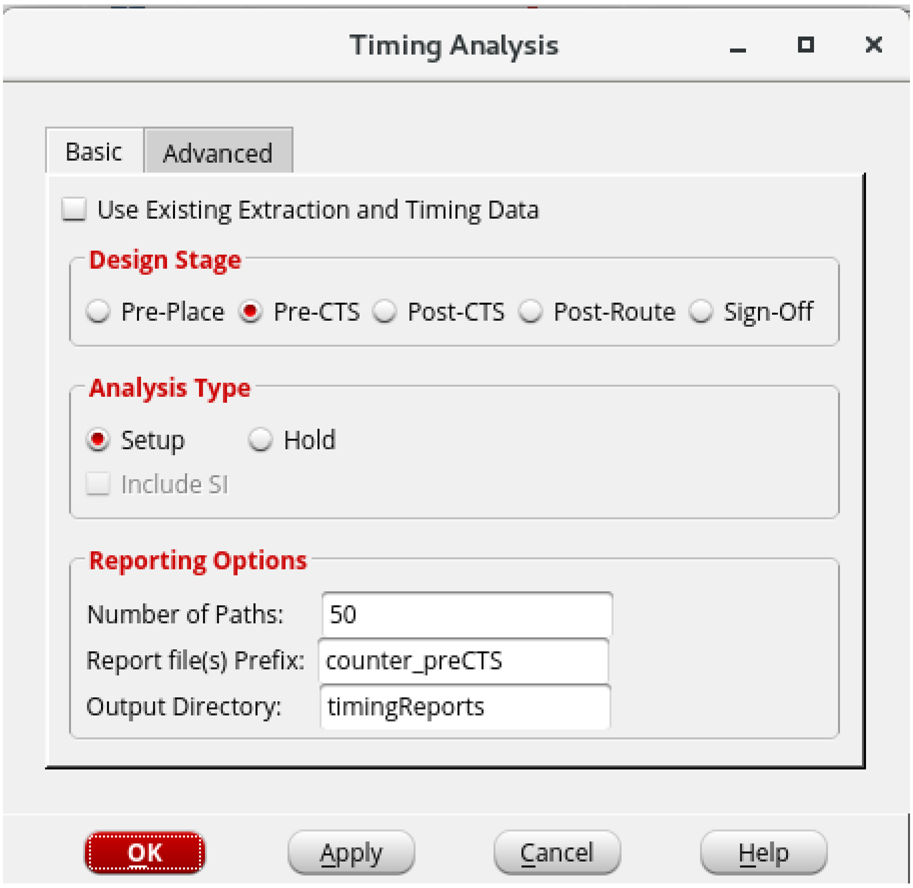
You can toggle the Layer Visibility from the list on the Right.

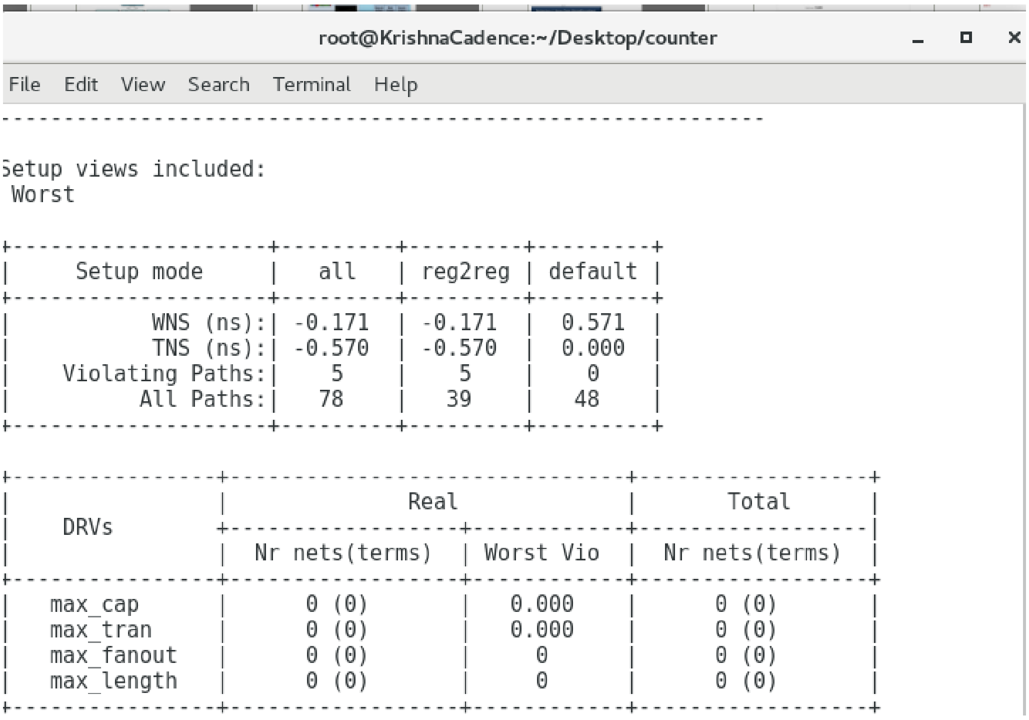
Report Generation and Optimization :

➔ Timing Report :

To generate Timing Report, **Timing → Report Timing → Design Stage – PreCTS ➔ Analysis Type – Setup → OK**

The Timing report Summary can be seen on the Terminal.






```
------------------------------------------------------------

Setup views included:
 Worst

+--------------------+----------+---------+---------+
|     Setup mode     |   all    | reg2reg | default |
+--------------------+----------+---------+---------+
|           WNS (ns):| -0.171   | -0.171  |  0.571  |
|           TNS (ns):| -0.570   | -0.570  |  0.000  |
|    Violating Paths:|    5     |    5    |    0    |
|          All Paths:|   78     |   39    |   48    |
+--------------------+----------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              Real             |       Total      |
|    DRVs        +------------------+------------+------------------|
|                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
|   max_cap      |      0 (0)       |   0.000    |      0 (0)       |
|   max_tran     |      0 (0)       |   0.000    |      0 (0)       |
|   max_fanout   |      0 (0)       |     0      |      0 (0)       |
|   max_length   |      0 (0)       |     0      |      0 (0)       |
+----------------+------------------+------------+------------------+
```

➔ **Area Report :**

To generate Area Report, Switch to the Terminal and type the command ,

report_area to see the Cell Count and Area Occupied.

```
innovus 3>
innovus 3> report_area
Depth  Name       #Inst  Area (um^2)
-----------------------------------
0      counter    84     672.8841
1
innovus 4> 
```

➔ **Power report :**

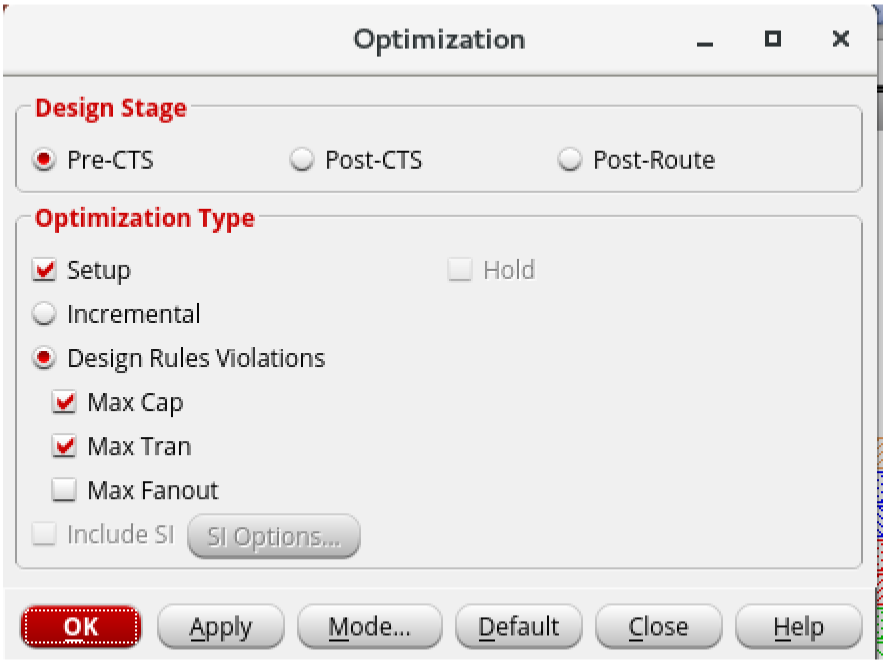
To generate Power Report, In the Terminal type the command

report_power to see the Power Consumption numbers.

```
*
*       report_power
*
-----------------------------------------------------------------------------------------

Total Power
-----------------------------------------------------------------------------------------
Total Internal Power:       0.19207683          90.2531%
Total Switching Power:      0.01693992           7.9597%
Total Leakage Power:        0.00380342           1.7872%
Total Power:                0.21282017
-----------------------------------------------------------------------------------------

Group                           Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
Sequential                        0.1793    0.007572    0.002415      0.1892       88.92
Macro                                  0           0           0           0           0
IO                                     0           0           0           0           0
Combinational                    0.01282    0.009368    0.001388     0.02358       11.08
Clock (Combinational)                  0           0           0           0           0
Clock (Sequential)                     0           0           0           0           0
-----------------------------------------------------------------------------------------
Total                             0.1921     0.01694    0.003803      0.2128         100
-----------------------------------------------------------------------------------------

Rail                  Voltage   Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
Default                   0.9     0.1921     0.01694    0.003803      0.2128         100
```

Design Optimization :

To optimize the Design, Select **ECO → Optimize Design → Design Stage [PreCTS] → Optimization Type – Setup → OK**




```
**optDesign ... cpu = 0:00:30, real = 0:00:30, mem = 1065.9M, totSessionCpu=0:15:31 **

------------------------------------------------------------
     optDesign Final Summary
------------------------------------------------------------

Setup views included:
 Worst

+--------------------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  0.004  |  0.004  |  0.576  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|   78    |   39    |   48    |
+--------------------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              Real             |       Total      |
|    DRVs        +------------------+------------+------------------|
|                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
|   max_cap      |     0 (0)        |   0.000    |     0 (0)        |
|   max_tran     |     0 (0)        |   0.000    |     0 (0)        |
|   max_fanout   |     0 (0)        |     0      |     0 (0)        |
|   max_length   |     0 (0)        |     0      |     0 (0)        |
+----------------+------------------+------------+------------------+

Density: 81.031%
Routing Overflow: 0.00% H and 0.00% V
------------------------------------------------------------
**optDesign ... cpu = 0:00:30, real = 0:00:30, mem = 1066.2M, totSessionCpu=0:15:32 **
*** Finished optDesign ***
innovus 5>
```

This step Optimizes your design in terms of Timing, Area and Power.

You can Generate Timing, Area, Power in similar way as above report Post – Optimization to compare the Reports.

Module 4.6 : Clock Tree Synthesis

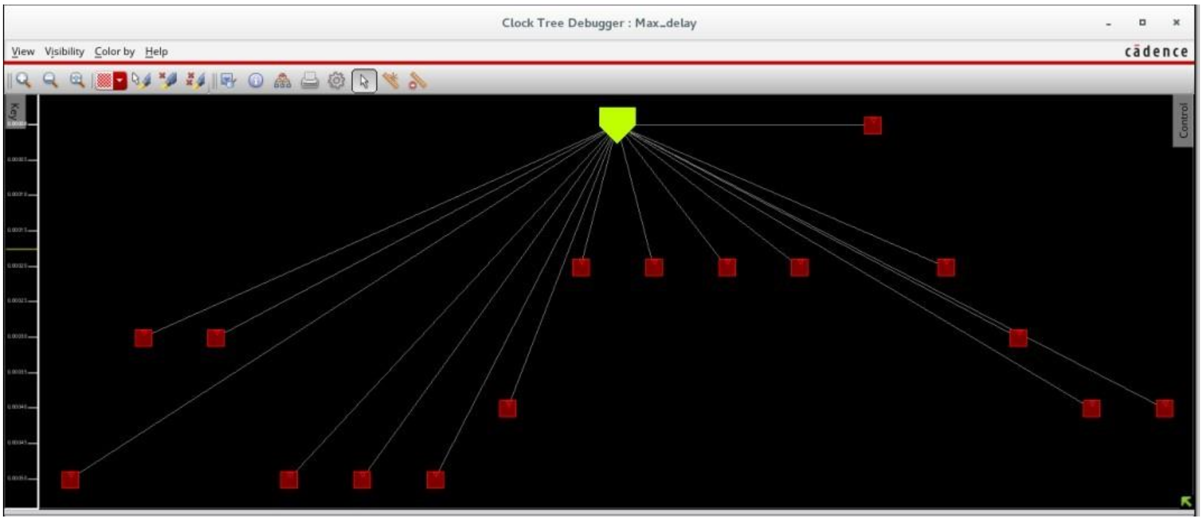
The CTS Stage is meant to build a Clock Distribution Network such that every Register (Flip Flop) acquires Clock at the same time (Atleast Approximately) to keep them in proper communication.

A Script can be used to Build the Clock Tree as follows :

```
add_ndr -width {Metal1 0.12 Metal2 0.14 Metal3 0.14 Metal4 0.14 Metal5 0.14 Metal6 0.14 Metal7 0.14 Metal8 0.14 Metal9 0.14 } -spacing {Metal1 0.12
Metal2 0.14 Metal3 0.14 Metal4 0.14 Metal5 0.14 Metal6 0.14 Metal7 0.14 Metal8 0.14 Metal9 0.14 } -name 2w2s
create_route_type -name clkroute -non_default_rule 2w2s -bottom_preferred_layer Metal5 -top_preferred_layer Metal6
set_ccopt_property route_type clkroute -net_type trunk
set_ccopt_property route_type clkroute -net_type leaf
set_ccopt_property buffer_cells {CLKBUFX8 CLKBUFX12}
set_ccopt_property inverter_cells {CLKINVX8 CLKINVX12}
set_ccopt_property clock_gating_cells TLATNTSCA*
create_ccopt_clock_tree_spec -file ccopt.spec
```

```
innovus 2> source ccopt.spec
Extracting original clock gating for clk...
  clock_tree clk contains 16 sinks and 0 clock gates.
  Extraction for clk complete.
Extracting original clock gating for clk done.
Checking clock tree convergence...
Checking clock tree convergence done.
```

Source the Script as shown in the above snapshot through the Terminal and then Select **Clock → CCOpt Clock Tree Debugger → OK** to build and view clock tree.



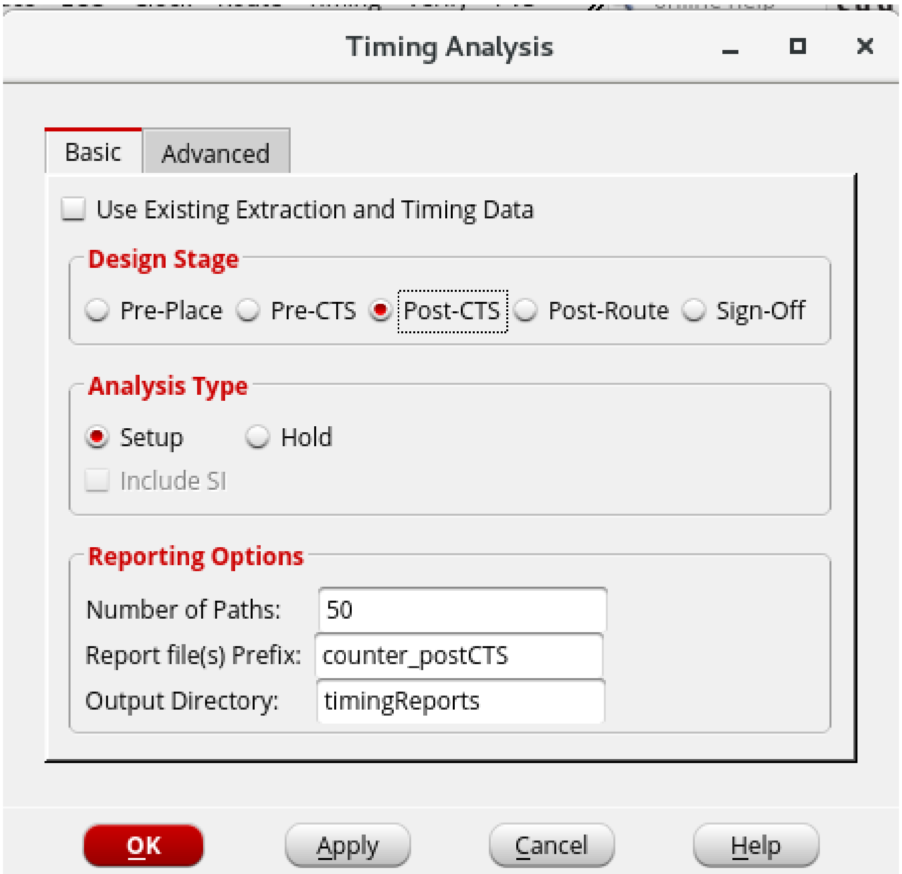
The Red Boxes are the Clock Pins of various Flip Flops in the Design while Yellow Pentagon on the top represents Clock Source.

The Clock Tree is built with Clock Buffers and Clock Inverters added to boost up the Clock Signal.

Report Generation and Design Optimization :

CTS Stage adds real clock into the Design and hence "Hold" Analysis also becomes prominent. Hence, **Optimizations can be done for both Setup & Hold, Timing Reports are to be Generated for Setup and Hold Individually**.

Setup Timing Analysis :

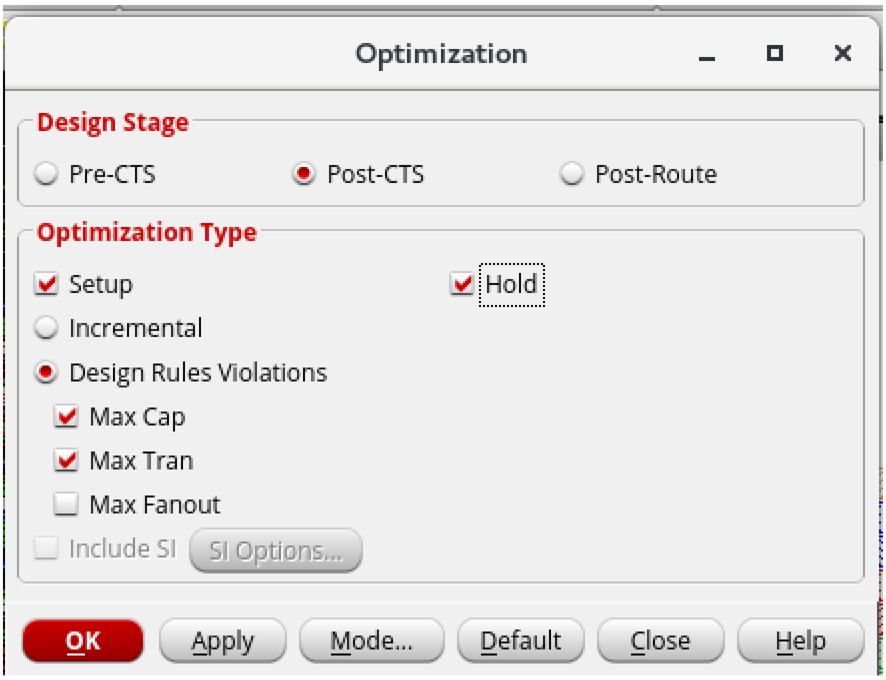


Hold Timing Analysis :

For Area and Power Report Generation,

report_area & report_power commands can be used.

Design Optimizations :

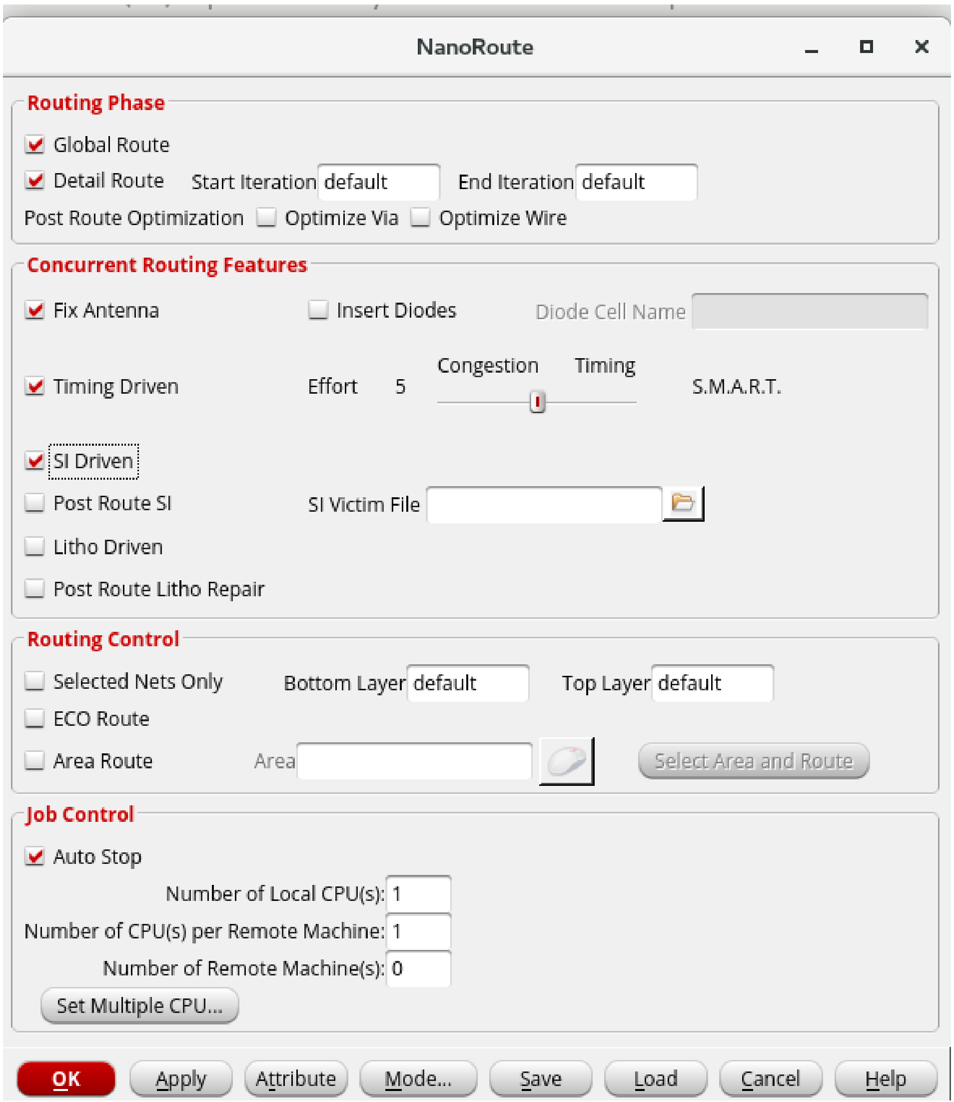


Module 4.7 : Routing

* All the net connections shown in the GUI till CTS are only based on the Logical connectivity.

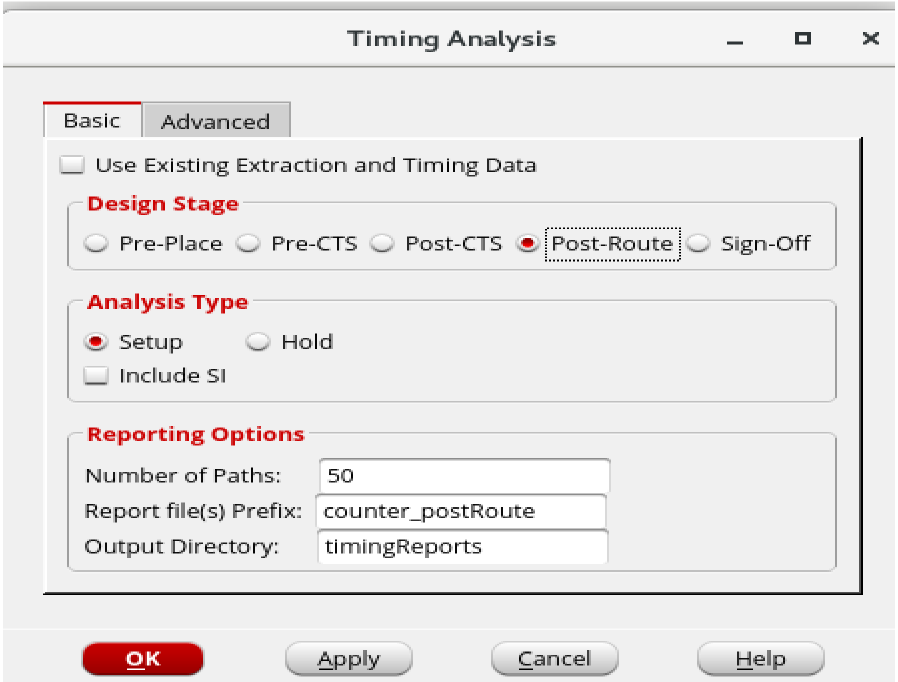
* These connections are to be replaced with real Metals avoiding Opens, Shorts, Signal Integrity [Cross Talks], Antenna Violations etc.

To run Routing, Select **Route → Nano Route → Route** and enable **Timing Driven** and **SI Driven** for Design Physical Efficiency and Reliability.

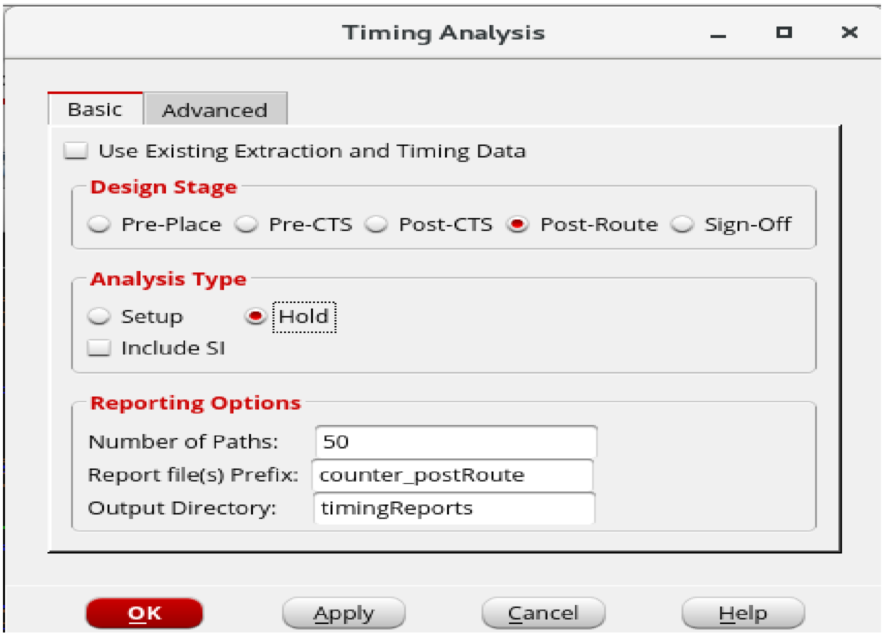


Report Generation and Design Optimization :

Setup Report :



Hold Report :



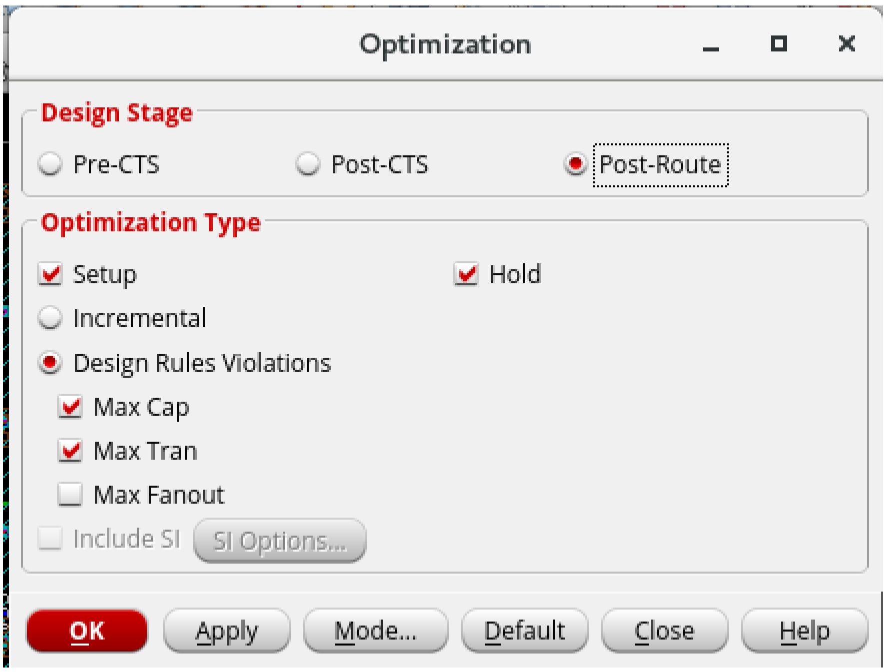
Area and Power Reports :

Use the commands **report_area** and **report_power** for Area and Power Reports respectively.

Design Optimization :

```
innovus 5>
innovus 5> setAnalysisMode -analysisType onChipVariation -cppr both
innovus 6>
```

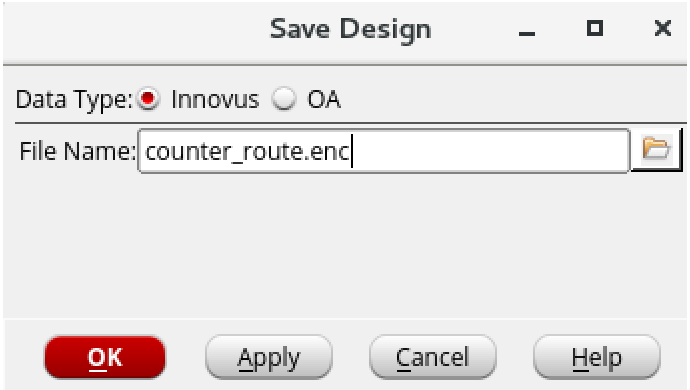
Enter the above shown command in the Terminal in order to run the Design Optimization first Post-Route.



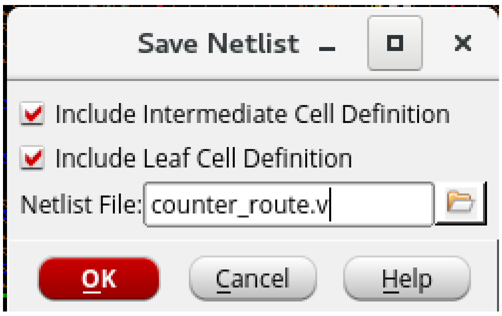
The Report generation is same as shown prior to Design Optimization.

Saving Database :

1. **Saving Design => File → Save Design → Data Type : Innovus → <DesignName>.enc → OK**



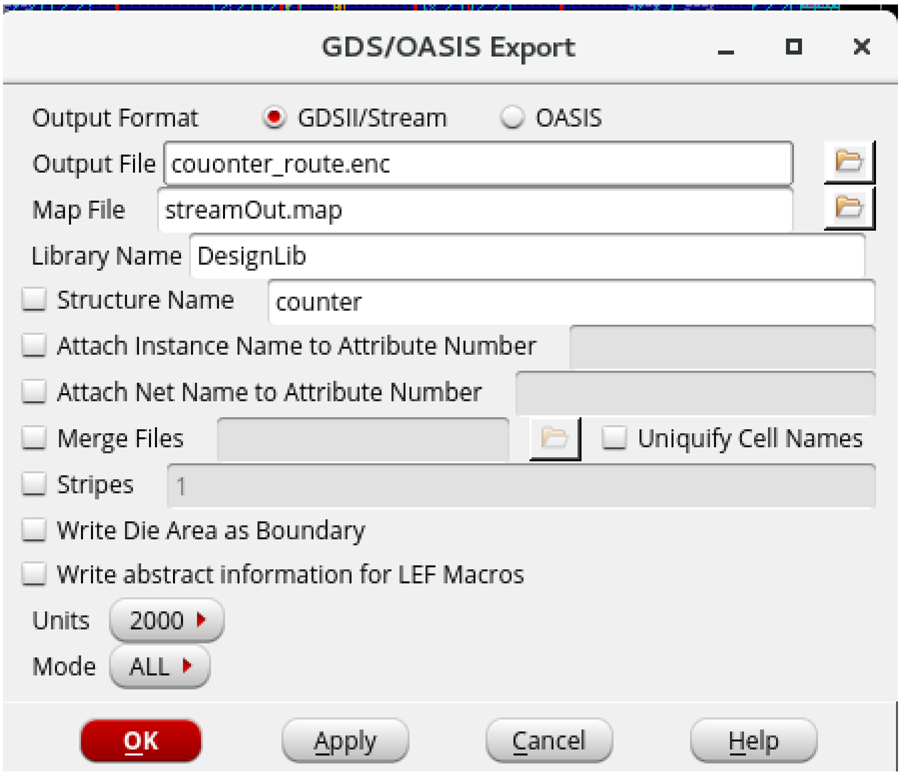
2. **Saving Netlist => File → Save → Netlist → <NetlistName>.v → OK**



It is recommended to save Netlist and Design at every stage.

To restore a Design Data Base, type **source <DesignName>.enc** in the terminal.

3. **Saving GDS => File → Save → GDS/OASIS → <FileName>.gds → OK**



The GDS File is a Binary Format File which is not Readable and is fed to the Fabrication unit with data of various layers used depicted in terms of Geometrical Shapes.

Design and analysis of Full Adder

verilog code:

```
module fa(input a,b,cin,outputs,cout);
assign s=a^b^cin;
assign cout=(a&b)|(b&cin)|(a&cin);
endmodule
```

Test Bench:

```
module fa_tb();
reg a,b,cin;
wire s,cout;
fa dut(a,b,cin,s,cout);
initial
begin
cin=0;b=0;a=0;
#50 cin=0;b=0;a=0;
#50 cin=0;b=0;a=1;
#50 cin=0;b=1;a=0;
#50 cin=0;b=1;a=1;
#50 cin=1;b=0;a=0;
#50 cin=1;b=0;a=1;
#50 cin=1;b=1;a=0;
#50 cin=1;b=1;a=1;
end
endmodule
```

Design and analysis of 3 to 8 Decoder
Verilog code

```
module dec328( input A,B,C,G1,G2A,G2B, output [7:0] Y );
 reg [7:0] Y;
 always@(A,B,C,G1,G2A,G2B)
    begin
    if({G1,G2A,G2B}==3'b100)
        begin
        case({A,B,C})
        3'b000: Y=8'b11111110;
        3'b001: Y=8'b11111101;
        3'b010: Y=8'b11111011;
        3'b011: Y=8'b11110111;
        3'b100: Y=8'b11101111;
        3'b101: Y=8'b11011111;
        3'b110: Y=8'b10111111;
        3'b111: Y=8'b01111111;
endcase
        end
        else
         Y=8'b11111111;
         end
endmodule
```

Test bench:

```
module dec328_tb();
reg A,B,C,G1,G2A,G2B;
wire [7:0]Y;
dec328 uut(A,B,C,G1,G2A,G2B,Y);
initial begin
G1=0;G2A=0;G2B=0;#100;
G1=1;G2A=0;G2B=0;
A=0;B=0;C=0;#100;
A=0;B=0;C=1;#100;
A=0;B=1;C=0;#100;
A=0;B=1;C=1;#100;
A=1;B=0;C=0;#100;
A=1;B=0;C=1;#100;
A=1;B=1;C=0;#100;
A=1;B=1;C=1;#100;
end
endmodule
```

Design and analysis of 8-bit counter

Verilog code:

```
module counter(input CLK,CLR,E,output reg [7:0] count);

   always@(posedge CLK)
   begin
if(CLR)
count<=0;
      else if(E)
            if(count==4'b1111)
count<=0;
            else
count<=count+1;
   end

endmodule
```

Test bench :

```
module counter_tb();

   reg CLK,CLR,E;
   wire [7:0]count;

   counter uut(CLK,CLR,E,count);

   initial
      CLK=0;
      always #10 CLK=~CLK;
   initial begin
      CLR=1;
      #100 CLR=0;
      #100 E=1;
   end

endmodule
```

Design and analysis of m-bit shift register:

Verilog code:

```
module shiftregnbit(CLK,W,RESET,Q);
parameter m=4;
input CLK,W,RESET;
output [1:m]Q;
reg [1:m]Q;
integer k;
always@(posedge CLK or negedge RESET)
if(!RESET)
Q<=0;
else
begin
for(k=m;k>1;k=k-1)
Q[k]<=Q[k-1];
Q[1]<=W;
end
endmodule
```

Test bench:

```
module shiftregnbit_tb();
reg CLK,RESET,W;
wire [3:0]Q;
shiftregnbituut(CLK,W,RESET,Q);

initial
CLK=0;
always #10 CLK=~CLK;
initial begin
RESET=0;
#100 RESET=1;W=0;
#100 W=1;
#100
#100 W=0;
#100;
end
endmodule
```