SYLLABUS

1. Implementation of Logic Gates –Data flow model and Behavioral model
2. Combinational logic circuits –Adders and Subtractor
3. Code converters-Binary to Gray and Gray to Binary
4. 3 to 8 Decoder –74138
5. 4 Bit Comparator –7485
6. 8 x 1 Multiplexer –74151 and 2X4 Demultiplexer –74155
7. 16 x 1 Multiplexer –74150 and 4X16 Demultiplexer –74154
8. Sequential circuits -Flip-Flops
9. Decade counter –7490.
10. Synchronous & Asynchronous Counters
11. Shift registers –7495.
12. Universal shift registers –74194/195.
13. RAM (16 x 4) –74189 (Read and Write operations).
14. Stack and Queue Implementation using RAM.

EX.NO:1 **DATE:**

Design Logic Gates Using Verilog HDL in Xilinx Platform

Aim:

To develop the source code for logic gates by using VERILOG and obtain the simulation.

Software Required:

- Xilinx – Project Navigator
- ModelSim Simulator

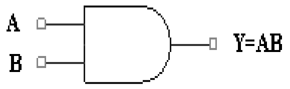
Procedure:

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the module by using source code.
Step3: Write the source code in VERILOG.
Step4: Check the syntax and debug the errors if found, obtain the synthesis is report.
Step5: Verify the output by simulating the source code.

Logic Diagram:

AND Gate:

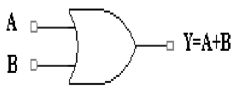
Logic Diagram: Truth Table:



A	B	Y=AB
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate:

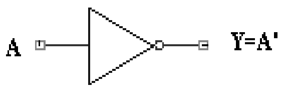
Logic diagram Truth Table



A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate:

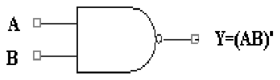
Logic Diagram: Truth Table:



A	Y=A’
0	1
1	0

NAND Gate:

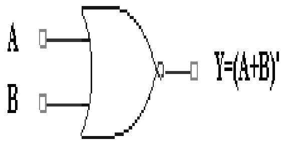
Logic Diagram Truth



A	B	Y=(ab)’
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate:

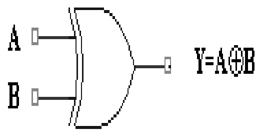
Logic Diagram: Truth Table:



A	B	Y=(A+B)’
0	0	1
0	1	0
1	0	0
1	1	0

XOR Gate:

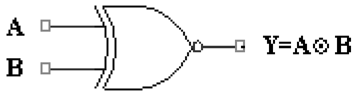
Logic Diagram Truth Table



A	B	Y=A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate:

Logic diagram: Truth table:



A	B	Y=A⊙B
0	0	1
0	1	0
1	0	0
1	1	1

Verilog Code:

AND Gate: **Simulation Output:**

```
module andgate(a,b, c);
input a,b;
output c;
assign c=a & b;
endmodule
```

OR Gate: **Simulation Output:**

```
module orgate(a,b, c);
input a,b;
output c;
assign c= a |b;
endmodule
```

NOT Gate: **Simulation Output:**

```
module notgate(a, abar);
input a;
output abar;
assign abar = ~a;
endmodule
```

NAND Gate: **Simulation output:**

```
module nandgate(a,b, c);
input a,b;
output c;
assign c = ~(a & b);
endmodule
```

NOR Gate: **Simulation Output:**

```
module norgate(a,b, c);
input a,b;
output c;
assign c = ~(a | b);
endmodule
```

XOR Gate: **Simulation Output:**

```
module exorgate(a,b, c);
input a,b;
output c;
assign c = a ^ b;
endmodule
```

XNOR Gate: **Simulation Output:**

```
module exnorgate(a,b, c);
input a,b;
output c;
assign c = ~(a ^ b);
endmodule
```

Result:

Thus the gates (AND, OR, NOT, NAND, NOR, XOR and XNOR gates) are simulated verified with VERILOG program.

EX.NO:2 **DATE:**

BASIC EXPERIMENTS

Aim:

To Simulate and synthesis Adders and subtractors using Verilog HDL

Software tools required:

Synthesis tool: Xilinx ISE 8.2
Simulation tool: Project Navigator

a) Half Adder and Half subtractors.

ALGORITM:

Step1: Define the specifications and initialize the design.

Step2: Declare the name of the entity and architecture by using VERILOG source code.

Step3: Write the source code in VERILOG.

Step4: Check the syntax and debug the errors if found, obtain the synthesis report.

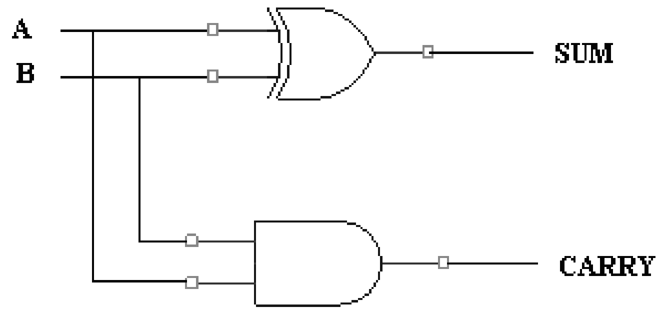
Step5: Verify the output by simulating the source code.

Step6: Write all possible combinations of input using the test bench.

Step7: Obtain the place and route report.

HALF ADDER:

LOGIC DIAGRAM:



TRUTH TABLE:

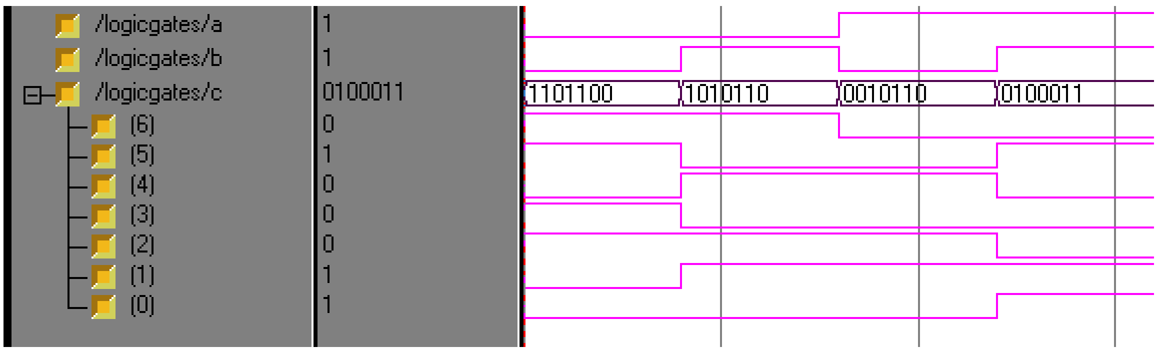
A	**B**	**SUM (s)**	**CARRY (ca)**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

VERILOG CODING:

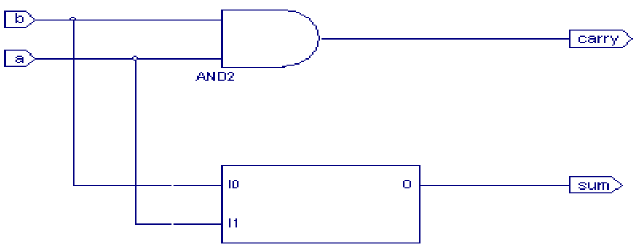
Dataflow Modeling:

```
module ha_dataflow(a, b, s, ca);
   input a;
   input b;
   output s;
   output ca;
          assign#2 s=a^b;
          assign#2 ca=a&b;
endmodule
```

Simulation output:

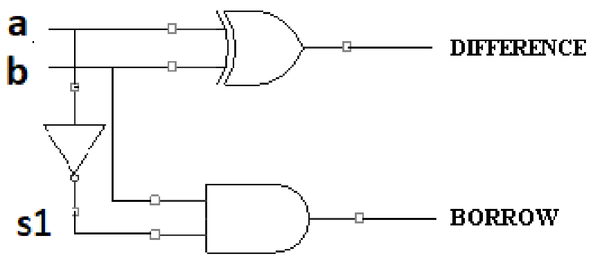


Synthesis RTL Schematic:



HALF SUBSTRACTOR:

LOGIC DIAGRAM:



TRUTH TABLE:

A	B	DIFFERENCE (diff)	BORROW (bor)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

VERILOG CODING:

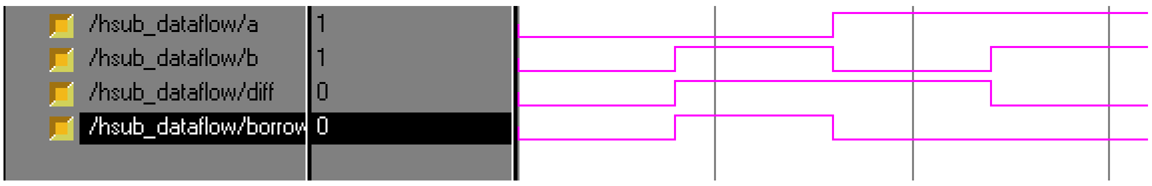
Dataflow Modeling:

```
module hs_dataflow(a, b, dif, bor);
   input a;
   input b;
   output dif;
   output bor;
          wire s1;
          assign#3 abar=~a;
          assign#3 dif=a^b;
          assign#3 bor=b&s1;
endmodule
```

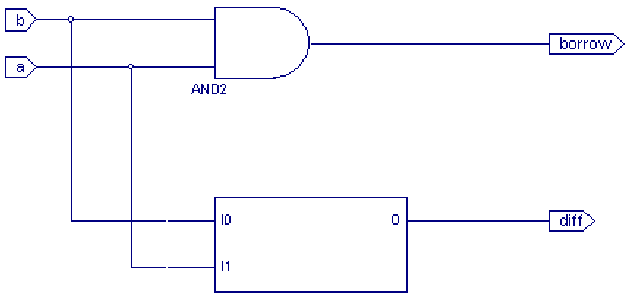
HALF SUBTRACTOR:

Synthesis RTL Schematic:

Simulation output:



Synthesis RTL Schematic:



b) **Full Adder and Full Subtractors.**

Aim:

To Simulate and synthesis Full Adders and Full subtractors using Verilog HDL

Software tools required:

Synthesis tool: Xilinx ISE 8.2
Simulation tool: Project Navigator

ALGORITM:

Step1: Define the specifications and initialize the design.

Step2: Declare the name of the entity and architecture by using Verilog source code.

Step3: Write the source code in VERILOG.

Step4: Check the syntax and debug the errors if found, obtain the synthesis report.

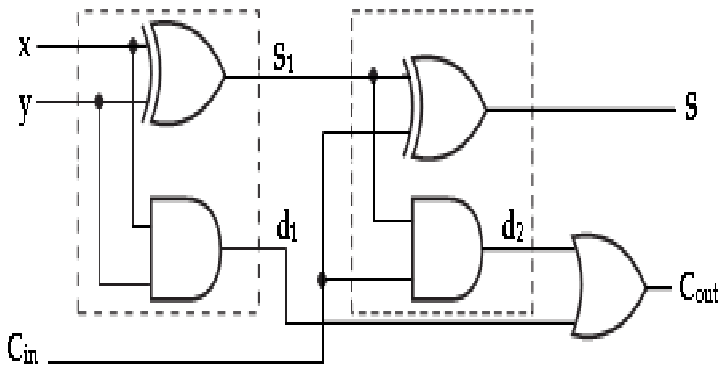
Step5: Verify the output by simulating the source code.

Step6: Write all possible combinations of input using the test bench.

Step7: Obtain the place and route report.

FULL ADDER:

LOGICDIAGRAM:



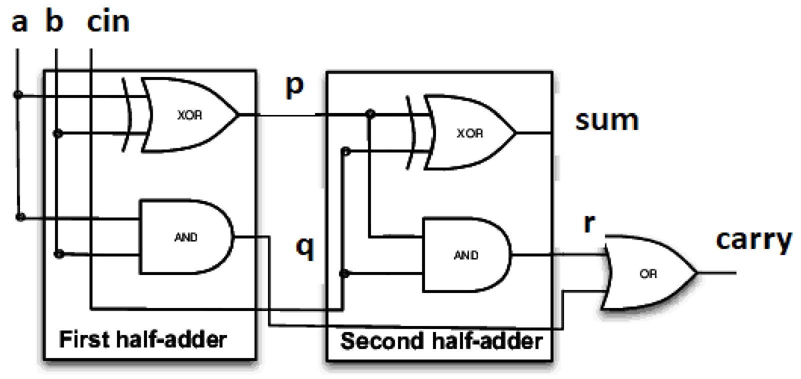
TRUTH TABLE:

A	B	C	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

Dataflow Modeling:

```
module fulsubdataflow(a, b, cin, sum, carry);
    input a;
    input b;
    input cin;
    output sum;
    output carry;
             wire s1,d1,d2;
             assign s1= a^b;
             assign diff=s1^cin;
             assign d1= a and b;
              assign d2= s1 and cin;
              assign carry=(d1 | d2);
 endmodule
```

Structural Modeling Full Adder using Half Adder:

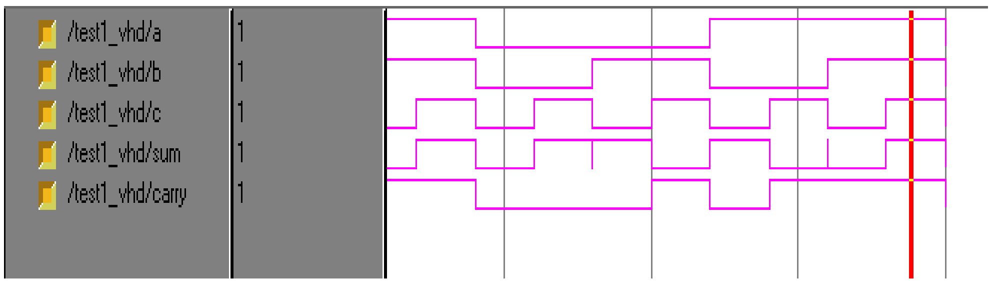


```
module fa_2ha(a, b, cin, sum, carry);
    input a;
    input b;
    input cin;
    output sum;
    output carry;
    wire p,q,r;
             ha_dataflow
             h1(a,b,p,q),
             h2(p,cin,sum,r);
             or o1(carry,q,r);
endmodule
```

Dataflow Modeling

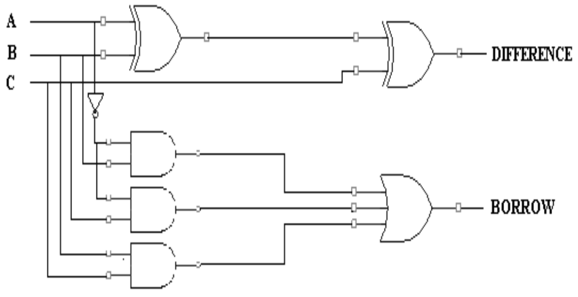
```
module ha_dataflow(a, b, sum, carry);
   input a;
   input b;
   output sum;
   output carry;
          assign#3 sum=a^b;
          assign#3 carry=a&b;
endmodule
```

Simulation output:



FULL SUBTRACTOR:

LOGIC DIAGRAM:



TRUTH TABLE:

A	B	C	DIFFERENCE	BORROW
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Dataflow Modeling:

```
module fulsubdataflow(a, b, c, diff, borrow);
   input a;
   input b;
   input c;
   output diff;
   output borrow;
          wire s1,s2,s3,s4,s5;
          assign s1= a^b;
          assign diff=s1^cin;
            assign s2= ~a;
            assign s3= s2 and b;
            assign s4= b and c;
            assign s5=s2 and c;
          assign borrow=(s3 | (s4) s5;
 endmodule
```

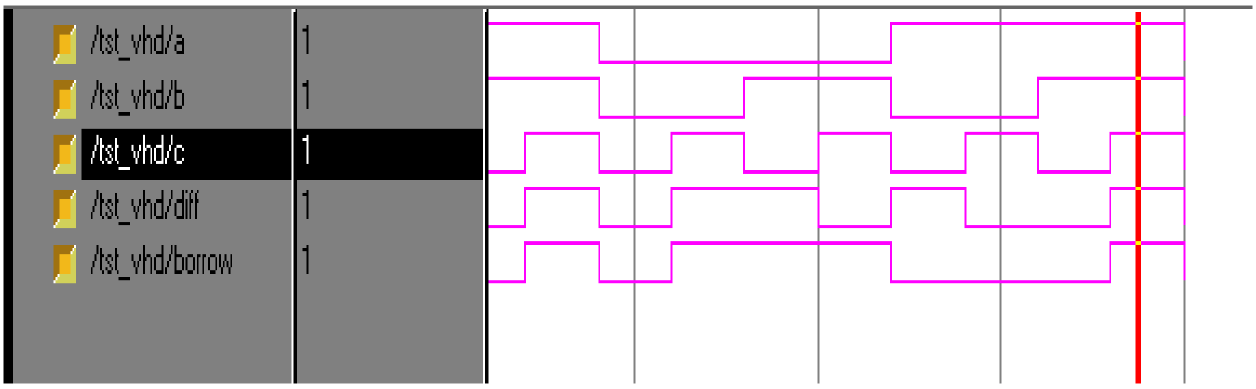
Structural Modeling:

```
module fs_2hs(a, b, c, diff, borrow);
   input a;
   input b;
   input c;
   output diff;
   output borrow;
   wire p,q,r;
           hs_dataflow
           h1(a,b,p,q),
           h2(p,c,diff,r);
           or o1(borrow,q,r);
endmodule
```

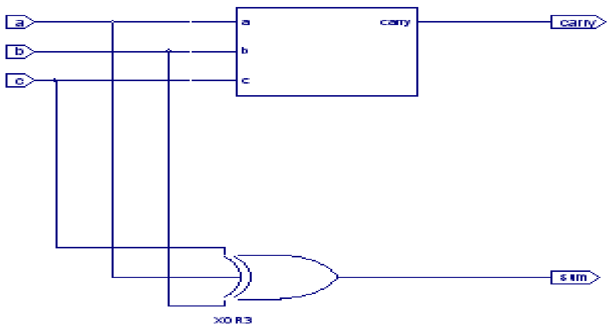
Dataflow Modeling

```
module hs_dataflow(a, b, dif, bor);
   input a;
   input b;
   output dif;
   output bor;
           wire abar;
           assign#3 abar=~a;
           assign#3 dif=a^b;
           assign#3 bor=b&abar;
endmodule
```

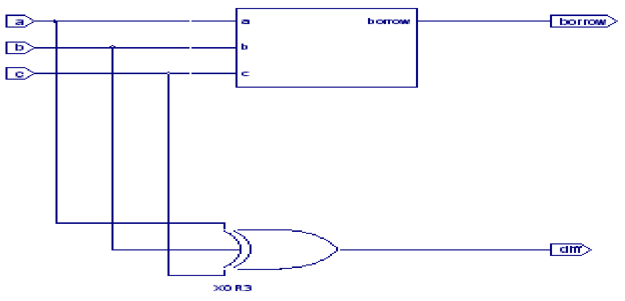
Simulation output:



FULL ADDER:
Synthesis RTL Schematic:



<u>FULL SUBTRACTOR:</u>
<u>Synthesis RTL Schematic:</u>



Result:

Thus the Adders and subtractors are simulated verified with VERILOG program.

EX.NO:3 **DATE:**

CODE CONVERTERS-BINARY TO GRAY AND GRAY TO BINARY

Aim:

To Simulate and synthesis Code converters- Binary to Gray and Gray to Binary using Verilog HDL

Software tools required:

Synthesis tool: Xilinx ISE 8.2
Simulation tool: Project Navigator

a) Binary to Gray and Gray to Binary

ALGORITM:

Step1: Define the specifications and initialize the design.

Step2: Declare the name of the entity and architecture by using VERILOG source code.

Step3: Write the source code in VERILOG.

Step4: Check the syntax and debug the errors if found, obtain the synthesis report.

Step5: Verify the output by simulating the source code.

Step6: Write all possible combinations of input using the test bench.

Step7: Obtain the place and route report.

Verilog Code for Binary to Gray code conversion:

```
module bin2gray
        (input [3:0] bin, //binary input
         output [3:0] G //gray code output);
//xor gates.
assign G[3] = bin[3];
assign G[2] = bin[3] ^ bin[2];
assign G[1] = bin[2] ^ bin[1];
assign G[0] = bin[1] ^ bin[0];
endmodule
```

Verilog Code for Gray code to Binary conversion:

```
module gray2bin
        (input [3:0] G, //gray code output
         output [3:0] bin   //binary input        );
assign bin[3] = G[3];
assign bin[2] = G[3] ^ G[2];
assign bin[1] = G[3] ^ G[2] ^ G[1];
assign bin[0] = G[3] ^ G[2] ^ G[1] ^ G[0];
endmodule
```

Result:

Thus the Binary to Gray and Gray to Binary are simulated verified with VERILOG program.

EX.NO:4 **DATE:**

3 TO 8 DECODER & ENCODER

Aim:

To develop the source code for Decoder & Encoder by using VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

Software tools required:

Synthesis tool: Xilinx ISE 8.2
Simulation tool: Project Navigator

DECODER & ENCODER:

Algoritm:

Step1: Define the specifications and initialize the design.

Step2: Declare the name of the entity and architecture by using VHDL source code.

Step3: Write the source code in VERILOG.

Step4: Check the syntax and debug the errors if found, obtain the synthesis report.

Step5: Verify the output by simulating the source code.

Step6: Write all possible combinations of input using the test bench.

Step7: Obtain the place and route report.

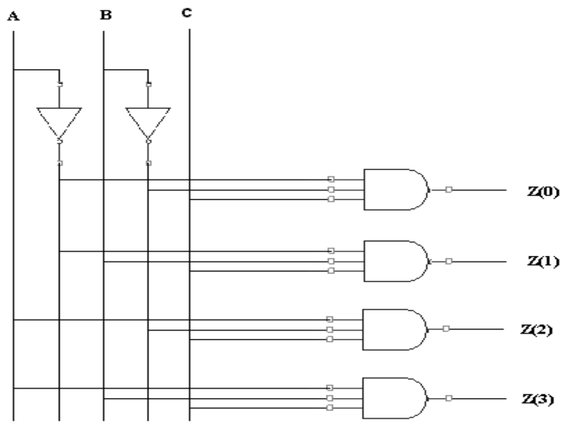
PROGRAM:

```
module Decoder(a,b,c,d0,d1,d2,d3,d4,d5,d6,d7);
input a,b,c;
output d0,d1,d2,d3,d4,d5,d6,d7;
assign d0=(~a&~b&~c),
d1=(~a&~b&c),
d2=(~a&b&~c),
d3=(~a&b&c),
d4=(a&~b&~c),
d5=(a&~b&c),
d6=(a&b&~c),
d7=(a&b&c);
endmodule
```

DECODERS:

Logic Diagram: Truth Table

A	**B**	**C**	**Z(0)**	**Z(1)**	**Z(2)**	**Z(3)**
0	0	1	1	0	0	0
0	1	1	0	1	0	0



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

<u>ENCODER:</u>

```
module encoder(D, x,y,z);
input [7:0] D;
output x,y,z;
assign x=D[4]|D[5]|D[6]|D[7];
assign y=D[2]|D[3]|D[6]|D[7];
assign z=D[1]|D[3]|D[5]|D[7];
endmodule
```

<u>DECODER:</u>

Behavioral-module

```
module decoder(A,B,C,Z);
input A,B,C;
output [3:0] Z;
reg [3:0] Z;
reg xbar,ybar,zbar;
always @ (A or B ) begin
Abar=~A;
Bbar=~B;
Z[0]=Abar&Bbar&C;
Z[1]=Abar&B&C;
Z[2]=A&Bbar&C;
Z[3]=A&B&C;
end
endmodule
```

Result:

Thus the Decoder & Encoder are simulated and verified with VERILOG program.

EX.NO:5 **DATE:**

4 BIT COMPARATOR

To develop the source code for 4 Bit Comparator by using VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

Software tools required:

Synthesis tool: Xilinx ISE 8.2
Simulation tool: Project Navigator

4 BIT COMPARATOR:

Algoritm:

Step1: Define the specifications and initialize the design.

Step2: Declare the name of the entity and architecture by using VHDL source code.

Step3: Write the source code in VERILOG.

Step4: Check the syntax and debug the errors if found, obtain the synthesis report.

Step5: Verify the output by simulating the source code.

Step6: Write all possible combinations of input using the test bench.

Step7: Obtain the place and route report.

4 bit Comparator:

```
//declare the Verilog module - The inputs and output signals.
module comparator(
    Data_in_A,   //input A
    Data_in_B,   //input B
    less,        //high when A is less than B
    equal,        //high when A is equal to B
     greater          //high when A is greater than B
    );

    //what are the input ports.
    input [3:0] Data_in_A;
    input [3:0] Data_in_B;
    //What are the output ports.
    output less;
     output equal;
     output greater;
     //Internal variables
     reg less;
     reg equal;
     reg greater;

    //When the inputs and A or B are changed execute this block
    always @(Data_in_A or Data_in_B)
     begin
        if(Data_in_A > Data_in_B)   begin  //check if A is bigger than B.
            less = 0;
            equal = 0;
            greater = 1;    end
        else if(Data_in_A == Data_in_B) begin //Check if A is equal to B
            less = 0;
            equal = 1;
            greater = 0;    end
        else    begin //Otherwise - check for A less than B.
```

```
            less = 1;
            equal = 0;
            greater =0;
        end
    end
endmodule
```

Result:

Thus the 4 Bit Comparator is simulated and verified with VERILOG program.

EX.NO:6 **DATE:**

8 X 1 MULTIPLEXER & 1X4 DEMULTIPLEXER

Aim:

To develop the source code for Multiplexer and Demultiplexer by using VERILOG and obtain the simulation, synthesis, place and route and implement into FPGA.

Software tools required:

Synthesis tool: Xilinx ISE 8.2
Simulation tool: Project Navigator

MULTIPLEXER AND DEMULTIPLEXER:

Algoritm:

Step1: Define the specifications and initialize the design.

Step2: Declare the name of the entity and architecture by using VHDL source code.

Step3: Write the source code in VERILOG.

Step4: Check the syntax and debug the errors if found, obtain the synthesis report.

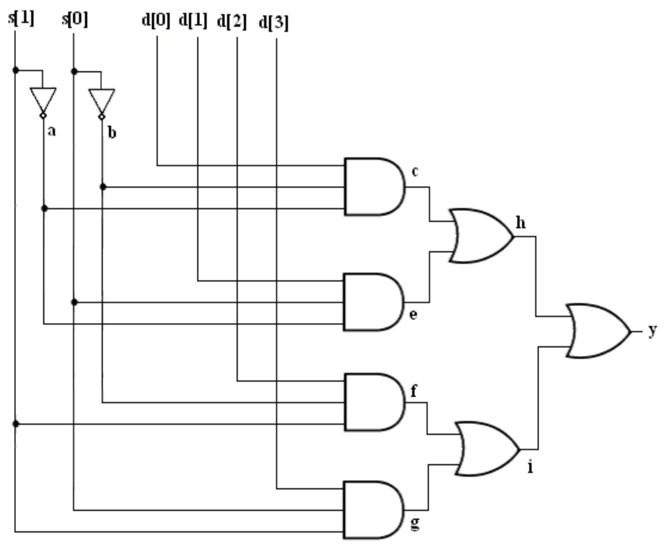
Step5: Verify the output by simulating the source code.

Step6: Write all possible combinations of input using the test bench.

Step7: Obtain the place and route report.

MULUTIPLEXER:

Logic diagram:



Truth table:

INPUT		OUTPUT
s[1]	**s[0]**	**y**
0	0	D[0]
0	1	D[1]
1	0	D[2]
1	1	D[3]

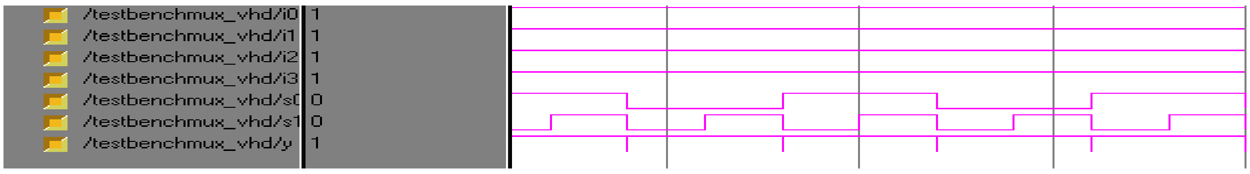
VERILOG SOURCE CODE:

Dataflow Modeling:

```
module  muxdataflow(s, i, y);
   input [1:0]s;
   input [3:0]i;
   output y;
         wire f1,f2,f3,f4,f5,f6;
         assign f1=~s[1];
         assign f2=~s[0];
```

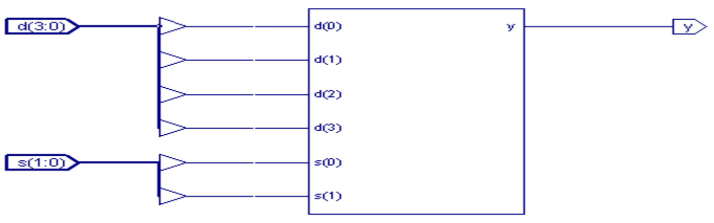
```
            assign f3=i[0]&f1&f2;
            assign f4=i[1]&f1&s[0];
            assign f5=i[2]&s[1]&s[0];
            assign f6=i[3]&s[1]&s[0];
            assign y=f3|f4|f5|f6;
endmodule

module Mulitplexer(d0,d1,d2,d3,d4,d5,d6,d7,sel,out);
input d0,d1,d2,d3,d4,d5,d6,d7;
input [2:0] sel;
output reg out;
always@(sel)
begin
case(sel)
3'b000:out=d0;
3'b001:out=d1;
3'b010:out=d2;
3'b011:out=d3;
3'b100:out=d4;
3'b101:out=d5;
3'b110:out=d6;
3'b111:out=d7;
endcase
end
endmodule
```

Simulation output:



```
/testbenchmux_vhd/i0 1
/testbenchmux_vhd/i1 1
/testbenchmux_vhd/i2 1
/testbenchmux_vhd/i3 1
/testbenchmux_vhd/s0 0
/testbenchmux_vhd/s1 0
/testbenchmux_vhd/y  1
```

Synthesis RTL Schematic:

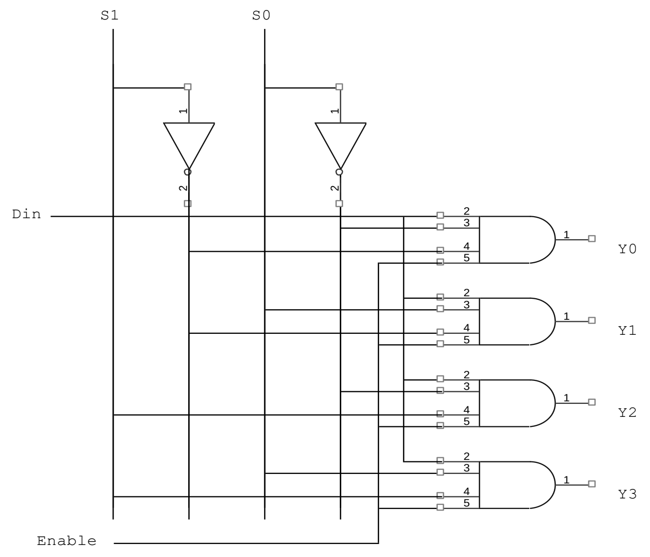


DEMULTIPLEXER:

LOGIC DIAGRAM:

Truth table:

INPUT			OUTPUT			
D	S0	S1	Y0	Y1	Y2	Y3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

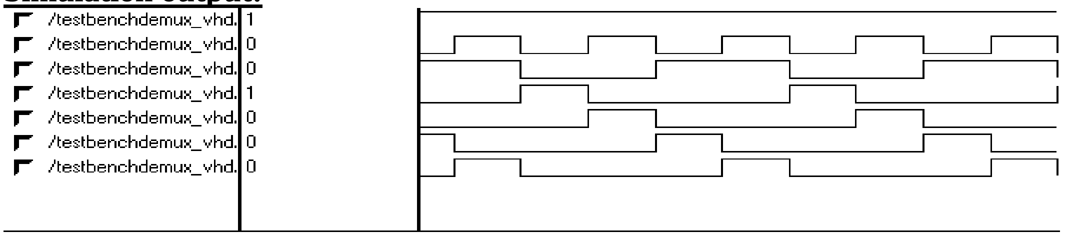


VERILOG SOURCE CODE:

Dataflow Modeling:

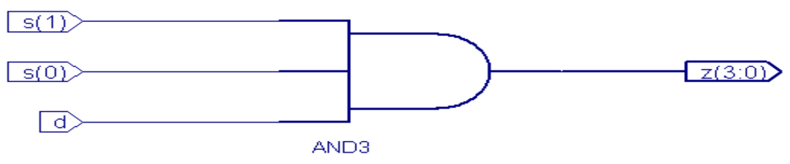
```
module demuxdataflow((din, s0, s1, d);
    input din;
    input s0;
     input s1;
    output [3:0]d;
    wire f1,f2;
     assign f1=~s1;
     assign f2=~s0;
     assign d[0]=din&f1&f2;
     assign d[1]=din&f1&s0;
     assign d[2]=din&s1&f2;
     assign d[3]=din&s1&s0;
endmodule
```

Simulation output:



```
/testbenchdemux_vhd. 1
/testbenchdemux_vhd. 0
/testbenchdemux_vhd. 0
/testbenchdemux_vhd. 1
/testbenchdemux_vhd. 0
/testbenchdemux_vhd. 0
/testbenchdemux_vhd. 0
```

Synthesis RTL Schematic:



RESULT:

Thus the output's of 8 X 1 MULTIPLEXER & 1X4 DEMULTIPLEXER are verified by synthesizing and simulating the VERILOG code.

EXP NO: 07 **DATE:**

DESIGN OF FILP-FLOPS USING VERILOG HDL IN XILINX PLATFORM

Aim:

To develop the source code for flip-flops by using VERILOG and obtain the simulation.

Software Required:

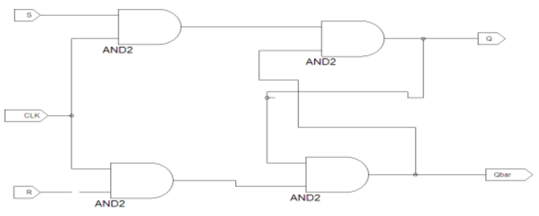
- Xilinx – Project Navigator
- ModelSim Simulator

Procedure:

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the module by using VERILOG source code.
Step3: Write the source code in VERILOG.
Step4: Check the syntax and debug the errors if found, obtain the synthesis is report.
Step5: Verify the output by simulating the source code.

SR Flip Flop:

Logic Diagram:



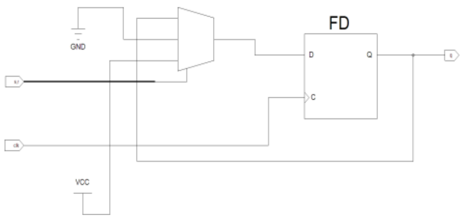
Truth Table:

Q(t)	S	R	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	x

SR Flip Flop - Program:

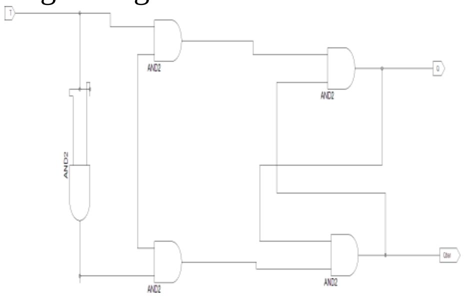
```
module srff_clk(s, r, clk, q, qn);
input s, r, clk, q;
output qn;
reg qn;
always @(posedge clk,s,r,q)begin
qn=(s|((~r)&q));
end
endmodule
```

Simulation Output: **Schematic Diagram:**



D Flip Flop:

Logic Diagram:



Truth Table:

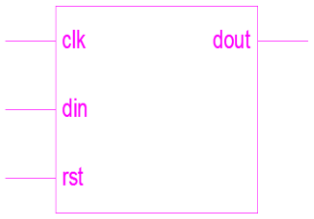
Q(t)	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

Program:

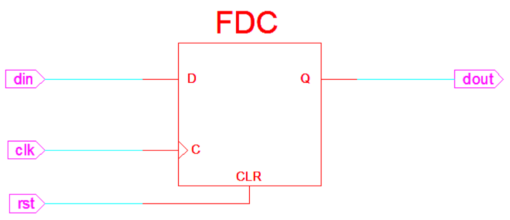
```
module DFF(d, clk, q);
input d;
input clk;
output q;
reg q;
always @(posedge clk)begin
q=d;
end
endmodule
```

Simulation Output:

RTL Schematic:

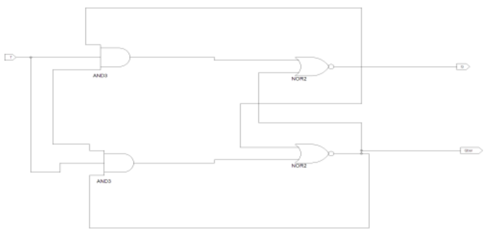


Schematic Diagram:



T- Flip Flop:

Logic Diagram:



Truth Table:

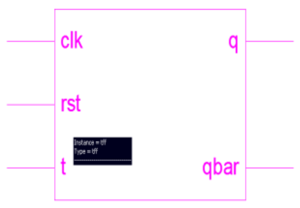
Q(t)	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

Program:

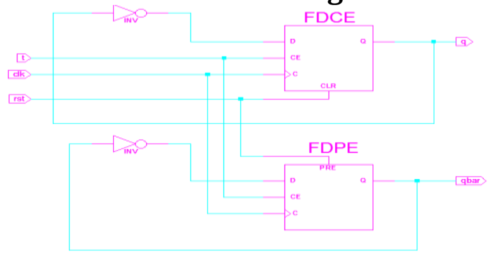
```
module TFF(t, clk, q,qout);
input t,q;
input clk;
output qout;
reg qout;
always @(posedge clk)begin
qout=t^q;
end
endmodule
```

Simulation Output:

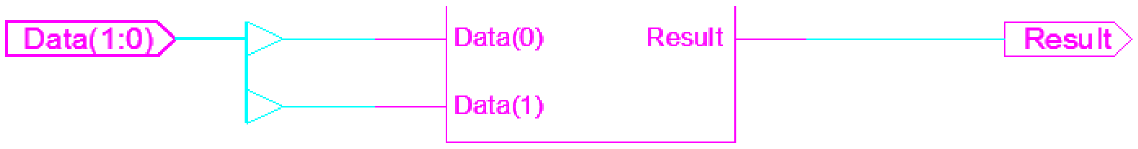
RTL Schematic:



Schematic Diagram:

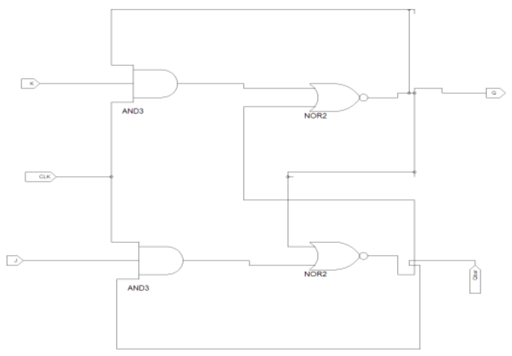


Schematic Diagram:



<u>JK Flip Flop:</u>

Logic Diagram:



Truth Table:

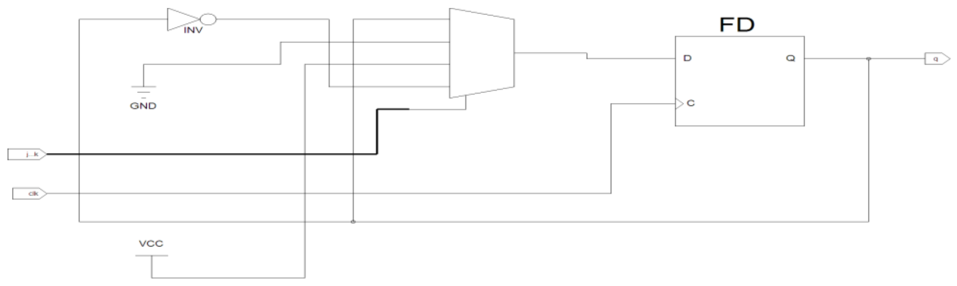
Q(t)	**J**	**K**	**Q(t+1)**
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Program:

```
module JKFF(J,K, clk, q,qout);
input J,K,q;
input clk;
output qout;
reg qout;
always @(posedge J,K,q, clk)begin
qout=(J&(~q))|(q&(~K));
end
endmodule
```

Simulation Output:

Schematic Diagram:



Result: Thus, the flip flops (SRFF,DFF,TFF.JKFF) are simulated and synthesized with verilog program.

EXP NO: 08 **DATE:**

DESIGN OF COUNTER USING VERILOG HDL IN XILINX PLATFORM

Aim:

To develop the source code for counter by using VERILOG and obtain the simulation.

Software Required:

- Xilinx – Project Navigator
- ModelSim Simulator

Procedure:

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the module by using VERILOG source code.
Step3: Write the source code in VERILOG.
Step4: Check the syntax and debug the errors if found, obtain the synthesis is report.
Step5: Verify the output by simulating the source code.

Verilog Code:

I. Up_Counter:

```
module upcounter    (out, enable, clk, reset);
output [7:0] out;
input enable, clk, reset;
reg [7:0] out;
always @(posedge clk)
if (reset) begin
out <= 8'b0 ;
end else if (enable) begin
out <= out + 1;
end
endmodule
```

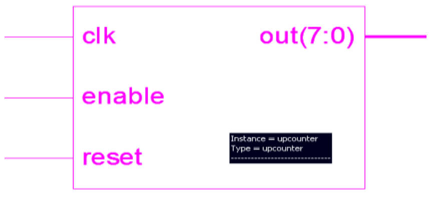
II. Up_Down_Counter:

Program:

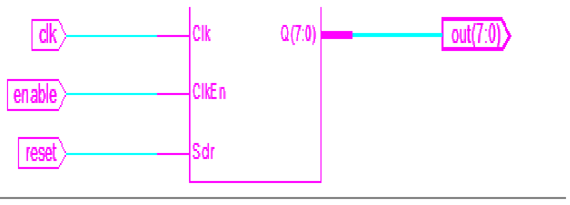
```
module up_down_counter    (out , up_down
,clk ,reset );
output [7:0] out; input up_down, clk, reset;
reg [7:0] out;
always @(posedge clk)
if (reset) begin
out <= 8'b0 ;
end else if (up_down) begin
out <= out + 1;
end else begin
out <= out - 1;
end
endmodule
```

Simulation Output:

RTL Schematic:



Schematic Diagram:



Result:

Thus the various counters are simulated and synthesized with verilog program.

Exp No: 09 **Date:**

4 BIT UP COUNTER WITH ASYNCHRONOUS

AIM:

To develop the source code for 4 BIT UP COUNTER with Asynchronous generator by using verilog and obtain the simulation.

Software Required:

- Xilinx – Project Navigator
- ModelSim Simulator

Procedure:

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the module by using verilog source code.
Step3: Write the source code in verilog.
Step4: Check the syntax and debug the errors if found, obtain the synthesis is report.
Step5: Verify the output by simulating the source code.

Asynchronous Counter:

4-bit Unsigned Up Counter with Asynchronous

```
module counter (C, CLR, Q);
input C, CLR;
output [3:0] Q;
reg [3:0] tmp;
   always @(posedge C or posedge CLR)
    begin
      if (CLR)
        tmp = 4'b0000;
      else
        tmp = tmp + 1'b1;
      end
  assign Q = tmp;
endmodule
```

4-bit Unsigned Down Counter with Synchronous:

```
module counter (C, S, Q);
input C, S;
output [3:0] Q;
reg [3:0] tmp;
  always @(posedge C)
    begin
      if (S)
        tmp = 4'b1111;
      else
        tmp = tmp - 1'b1;
    end
  assign Q = tmp;
endmodule
```

Result:

Thus the 4 bit and 8 bit PRBS generator is simulated and synthesized with verilog program.

Exp No: 10 **Date:**

SHIFT REGISTER

Aim:

To develop the source code for Shift Register by using verilog and obtain the simulation.

Software Required:

- Xilinx – Project Navigator
- ModelSim Simulator

Procedure:

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the module by using verilog source code.
Step3: Write the source code in verilog.
Step4: Check the syntax and debug the errors if found, obtain the synthesis is report.
Step5: Verify the output by simulating the source code.

SHIFT REGISTER:

```
 module shift_register(s1,d,clk,s0,q);
  parameter n=3;
  input  s1,clk;
  input [n:0] d;
  output s0;
  output [n:0] q;
  genvar i;
  assign d[3]=s1;
  generate
  for(i=0; i<=n; i=i+1)
     dff U1(.d(d[i]),.q(q[i]),.clk(clk));
  endgenerate

  assign q[3]=d[2];
  assign q[2]=d[1];
  assign q[1]=d[0];
  assign q[0]=s0;
  endmodule
```

Result:

Thus the Shift Register is simulated and synthesized with verilog program.

Exp No:11 **Date:**

DESIGN OF ACCUMULATOR USING VERILOG HDL IN XILINX PLATFORM

Aim:

To develop the source code for accumulator by using verilog and obtain the simulation.

Software Required:

- Xilinx – Project Navigator
- ModelSim Simulator

Procedure:

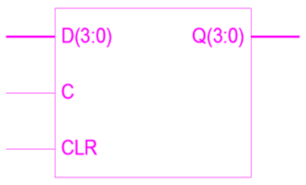
Step1: Define the specifications and initialize the design.
Step2: Declare the name of the module by using verilog source code.
Step3: Write the source code in verilog.
Step4: Check the syntax and debug the errors if found, obtain the synthesis is report.
Step5: Verify the output by simulating the source code.

Program:

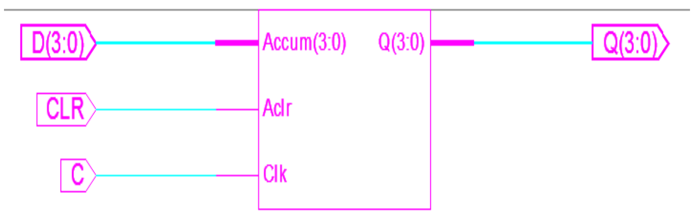
```
module accum (C, CLR, D, Q);
input C, CLR;
input  [3:0] D;
output [3:0] Q;
reg    [3:0] tmp;
always @(posedge C or posedge CLR)
   begin
     if (CLR)
       tmp = 4'b0000;
     else
       tmp = tmp + D;
   end
  assign Q = tmp;
endmodule
```

Simulation Output:

RTL Schematic:



Schematic Diagram:



Result:

Thus the accumulator is simulated and synthesized with verilog program.

EXP NO: 12 **DATE:**

DESIGN OF ARITHMETIC LOGIC UNIT USING VERILOG HDL IN XILINX PLATFORM

Aim:

To develop the source code for arithmetic logic unit by using verilog and obtain the simulation.

Software Required:

- Xilinx – Project Navigator
- ModelSim Simulator

Procedure:

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the module by using verilog source code.
Step3: Write the source code in verilog.
Step4: Check the syntax and debug the errors if found, obtain the synthesis is report.
Step5: Verify the output by simulating the source code.

Program: **Simulation Output:**

```
module alu(out, a,b, opcode);
output [7:0] out;
input [3:0] a,b;
input [1:0] opcode;
reg [7:0]out;
parameter
ADD=2'b00,
SUB=2'b01,
MUL=2'b10,
DIV=2'b11;
always @(a,b,opcode)
case(opcode)
ADD:out=a+b;
SUB:out=a-b;
MUL:out=a*b;
DIV:out=a/b;
endcase
endmodule
```

Result:

Thus the arithmetic logic unit is simulated and synthesized with verilog program.

Exp No: 13
Date:

DESIGN OF PRBS GENERATOR USING VERILOG HDL IN XILINX PLATFORM

Aim:

To develop the source code for PRBS generator by using verilog and obtain the simulation.

Software Required:

- Xilinx – Project Navigator
- ModelSim Simulator

Procedure:

Step1: Define the specifications and initialize the design.
Step2: Declare the name of the module by using verilog source code.
Step3: Write the source code in verilog.
Step4: Check the syntax and debug the errors if found, obtain the synthesis is report.
Step5: Verify the output by simulating the source code.

4-Bit PRBS Generator:

Program:

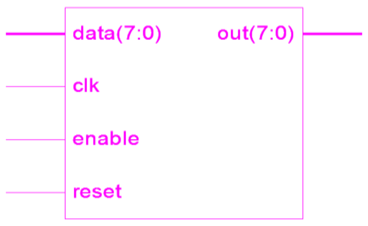
```
module prbs(rand,clk,rst);
input rst,clk;
output [3:0]rand;
wire [3:0]rand;
reg [3:0]temp;
always @(posedge clk or posedge rst)
begin
if(rst)
begin
temp<=4'b1111;
end else
begin
temp<={temp[0]^temp[1],temp[3],temp[2],te
mp[1]};
end end
assign rand=temp;
endmodule
```

8-Bit PRBS Generator:

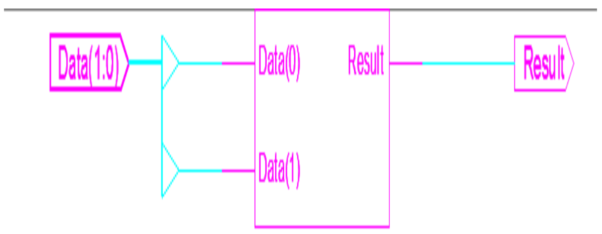
Linear feedback shift register

```
module lfsr  (data, out, enable , clk , reset);
output [7:0]
input [7:0] data;
input enable, clk, reset;
reg [7:0] out;
wire        linear_feedback;
assign linear_feedback = !(out[7] ^ out[3]);
always @(posedge clk)
if (reset) begin
out <= 8'b0 ;
end else if (enable) begin
out <= {out[6],out[5],
out[4],out[3],
out[2],out[1],
out[0], linear_feedback};
end
endmodule
```

RTL Schematic:



Schematic Diagram:



Result:

Thus the 4 bit and 8 bit PRBS generator is simulated and synthesized with verilog program.

Exp No: 14
Date:

STUDY OF PLACE AND ROUTE AND BACK ANNOTATION FOR FPGA

Aim:

To study Place and Route and Back annotation for FPGA using Xilinx Project Navigator software.

Software Required:

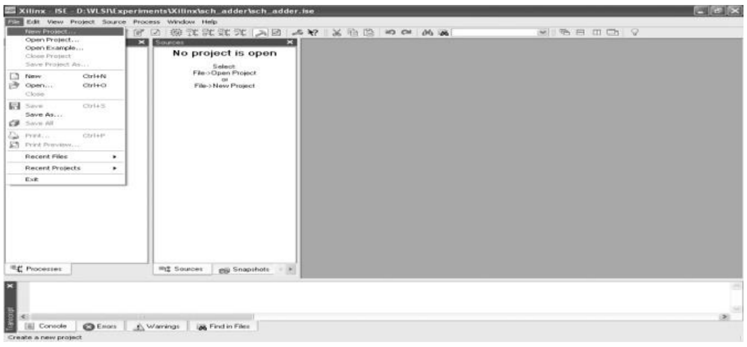
- Xilinx – Project Navigator.
- ModelSim Simulator.

Theory:

Before timing simulation can occur, the physical design information must be translated and distributed back to the logical design. For FPGAs, this back-annotation process is done with a program called NetGen. For CPLDs, back-annotation is performed with the TSim Timing Simulator. NetGen distributes information about delays; setup and hold times, clock to out, and pulse widths found in the physical NCD design file back to the logical NGD file and generate a Verilog or VHDL netlist for use with supported timing simulation, equivalence checking, and static timing analysis tools.

Procedure:

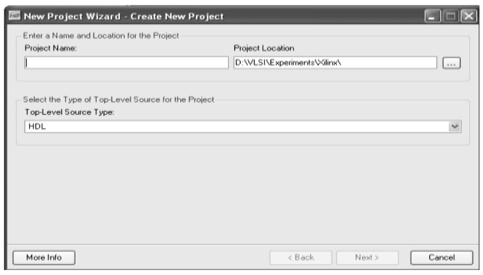
- Open Xilinx -> Project Navigator.
- Select File -> NEW PROJECT.



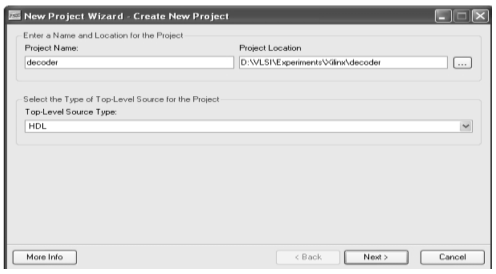
No project is open
Select
File->Open Project
or
File->New Project

Create a new project

- In the <u>New Project Wizard</u>, do the following:
  - In the Project Name field, enter a name for the project as decoder.
  - In the Project Location field, enter the directory name or browse to the directory.
  - In the Top-Level Module Type drop-down list, select HDL and click Next.


New Project Wizard - Create New Project
Enter a Name and Location for the Project
Select the Type of Top-Level Source for the Project
More Info
< Back
Next >
Cancel

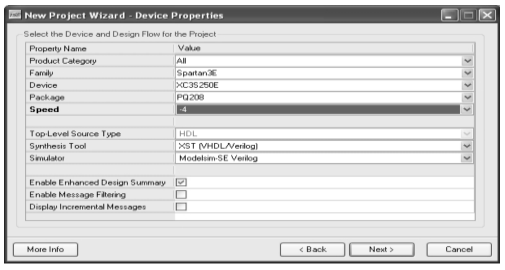
Project Name:
Project Location: D:\VLSI\Experiments\Xilinx\
Top-Level Source Type: HDL


New Project Wizard - Create New Project
Enter a Name and Location for the Project
Select the Type of Top-Level Source for the Project
More Info
< Back
Next >
Cancel

Project Name: decoder
Project Location: D:\VLSI\Experiments\Xilinx\decoder
Top-Level Source Type: HDL

- In the <u>Device Properties page</u> of the New Project Wizard, set the following options.
  - Product Category – All
  - Family – Spartan3E
  - Device – XC3S500E
  - Package – FG320

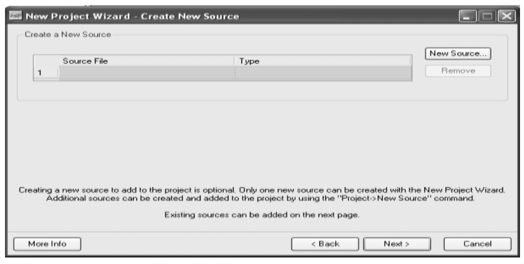
- Speed Grade – 4
- Verify that Enable Enhanced Design Summary is selected.



New Project Wizard - Device Properties

Select the Device and Design Flow for the Project

Property Name	Value
Product Category	All
Family	Spartan3E
Device	XC3S250E
Package	PQ208
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Modelsim-SE Verilog
Enable Enhanced Design Summary	✓
Enable Message Filtering	☐
Display Incremental Messages	☐



New Project Wizard - Create New Source

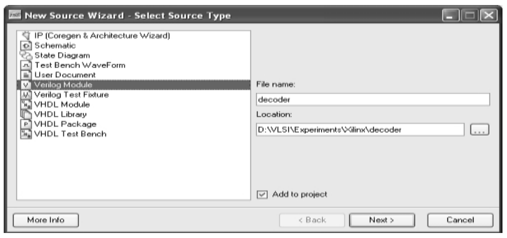
Create a New Source

	Source File	Type
1		

Creating a new source to add to the project is optional. Only one new source can be created with the New Project Wizard. Additional sources can be created and added to the project by using the "Project->New Source" command.

Existing sources can be added on the next page.

- If you are creating an HDL or schematic project, click Next, and optionally, create a new source file for your project.
- Click New Source. Select Verilog Module in New Source Window and enter the File Name as decoder and click Next.
- Again click Next and click Finish.
- Click Next.

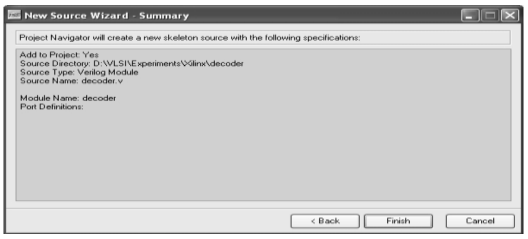


New Source Wizard - Select Source Type

File name: decoder

Location: D:\VLSI\Experiments\Xilinx\decoder

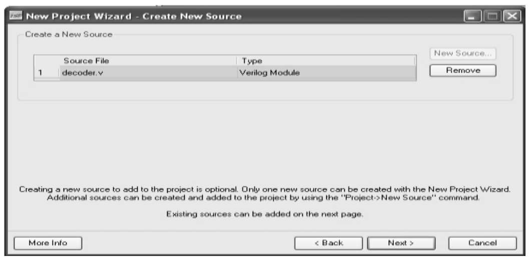
✓ Add to project



New Source Wizard - Summary

Project Navigator will create a new skeleton source with the following specifications:

```
Add to Project: Yes
Source Directory: D:\VLSI\Experiments\Xilinx\decoder
Source Type: Verilog Module
Source Name: decoder.v

Module Name: decoder
Port Definitions:
```



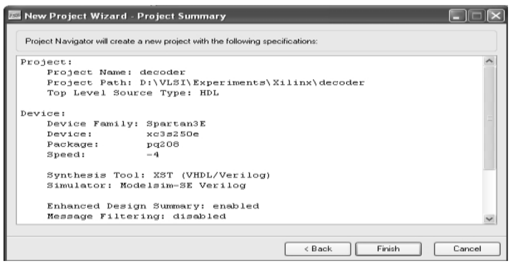
New Project Wizard - Create New Source

Create a New Source

	Source File	Type
1	decoder.v	Verilog Module

Creating a new source to add to the project is optional. Only one new source can be created with the New Project Wizard. Additional sources can be created and added to the project by using the "Project->New Source" command.

Existing sources can be added on the next page.

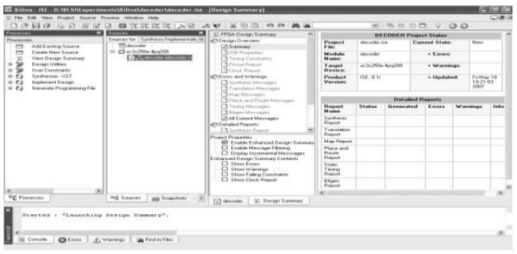


New Project Wizard - Project Summary

Project Navigator will create a new project with the following specifications:

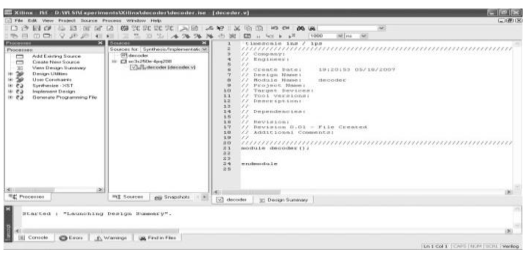
```
Project:
    Project Name: decoder
    Project Path: D:\VLSI\Experiments\Xilinx\decoder
    Top Level Source Type: HDL

Device:
    Device Family: Spartan3E
    Device:        xc3s250e
    Package:       pq208
    Speed:         -4

    Synthesis Tool: XST (VHDL/Verilog)
    Simulator: Modelsim-SE Verilog

    Enhanced Design Summary: enabled
    Message Filtering: disabled
```

- Click Next to display the Information page of the New Project Wizard and finally click finish to create the project.
- Double click the file name, which is inside the Sources in Project window and write the program for 3:8 Decoder using Verilog.



Xilinx - ISE - D:\VLSI\Experiments\Xilinx\decoder\decoder.ise - [Design Summary]

```
Started : "Launching Design Summary".
```



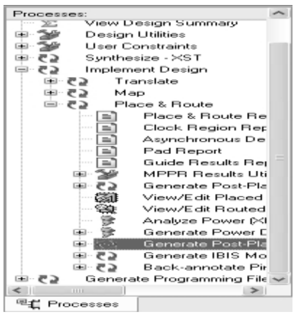
Xilinx - ISE - D:\VLSI\Experiments\Xilinx\decoder\decoder.ise - [decoder.v]

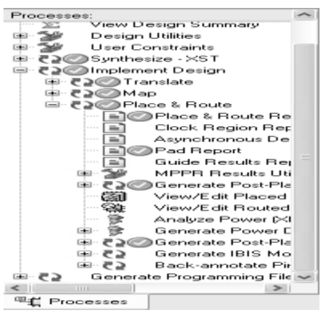
```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    19:20:53 05/18/2007
// Design Name:
// Module Name:    decoder
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module decoder();


endmodule
```

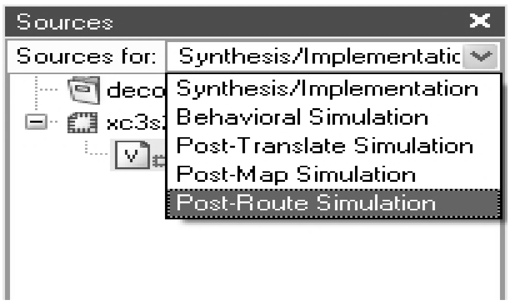
```
Started : "Launching Design Summary".
```

- Highlight the *.v file in the Sources in Project window.
- Go to Processes for Source window.
- Double click Generate Post–Place & Route Simulation Model, which is inside Place and Route.
- After all the processes before Place and Route is over, the window looks as shown below.

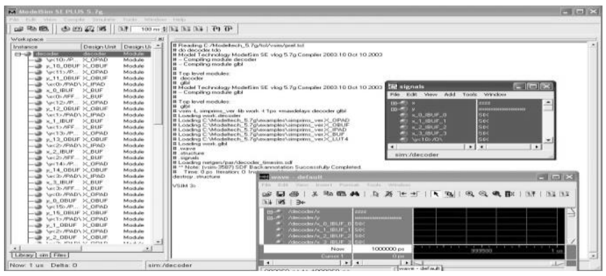
- Select Post-Route Simulation in Sources window.

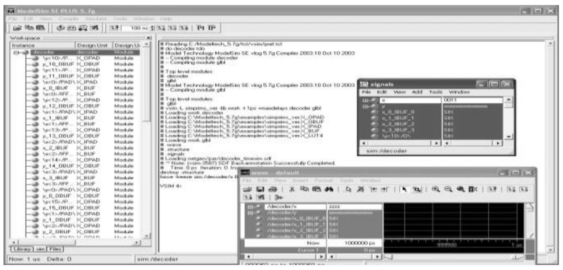




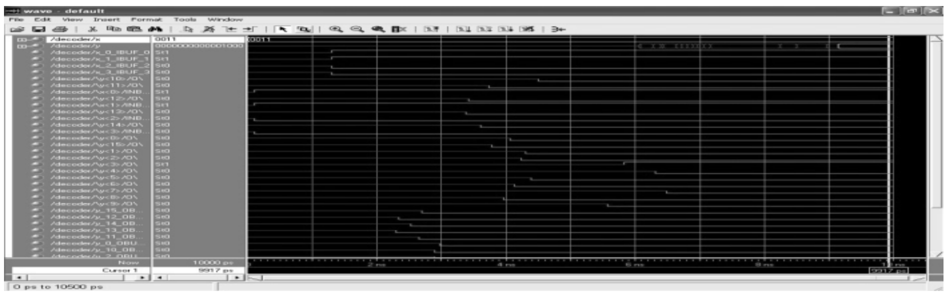


- After you select the file name, Run Simulate Post-Place and Route Model in Processes Window.
- After you run the previous process, ModelSim Window is opened.
- Now Force signal “x” to “001” and click the Run button in Waveform Window
- Now you can see the output waveform including the gate delays and net delays





- Thus by doing Post-Place and Route Simulation, we get the Back-Annotated waveform.



Program of any Combinational Circuits:

Result:

Thus we have studied the Place and Route and Back annotation for FPGA software.

Exp No: 15 **Date:**

STUDY OF SYNTHESIS FOR XILINX TOOLS

Aim:

To study the various synthesis tools used in the Xilinx - Project Navigator software using Full adder logic circuit as an example.

Software Required:

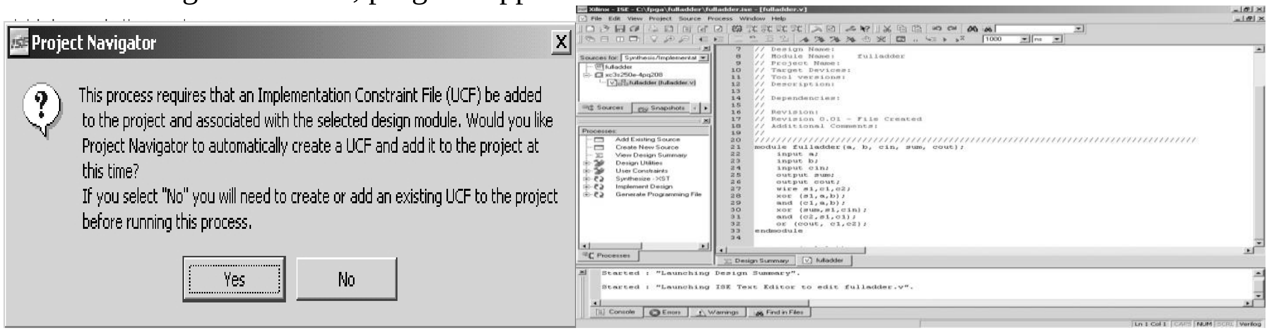
- Xilinx – Project Navigator.

Theory:

At the time when the development of VHDL was initiated the major concern was to have a standardized and unique method for documentation of complex digital circuits which would also allow simulating the circuit descriptions. Based on these objectives, VHDL provided semantic elements mainly for simulation purposes.. In the context of software tools and VHDL, synthesis is an automatic method of converting a higher level abstraction, such as a behavioral description, to a lower level abstraction, for example, a gate-level net list.

Procedure:

- Now start the Project Navigator. The previous project appears. Select the Verilog module, in the right hand side, program appears.



Project Navigator

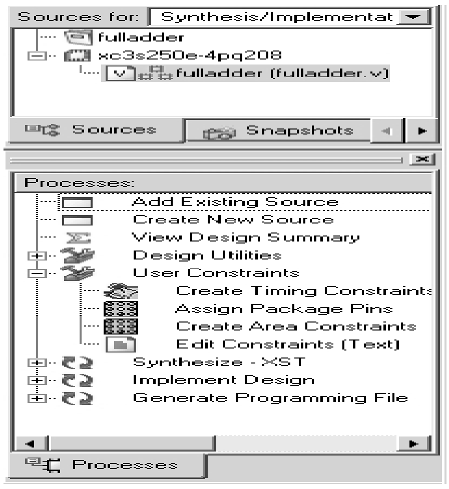
This process requires that an Implementation Constraint File (UCF) be added to the project and associated with the selected design module. Would you like Project Navigator to automatically create a UCF and add it to the project at this time?
If you select "No" you will need to create or add an existing UCF to the project before running this process.

Yes No

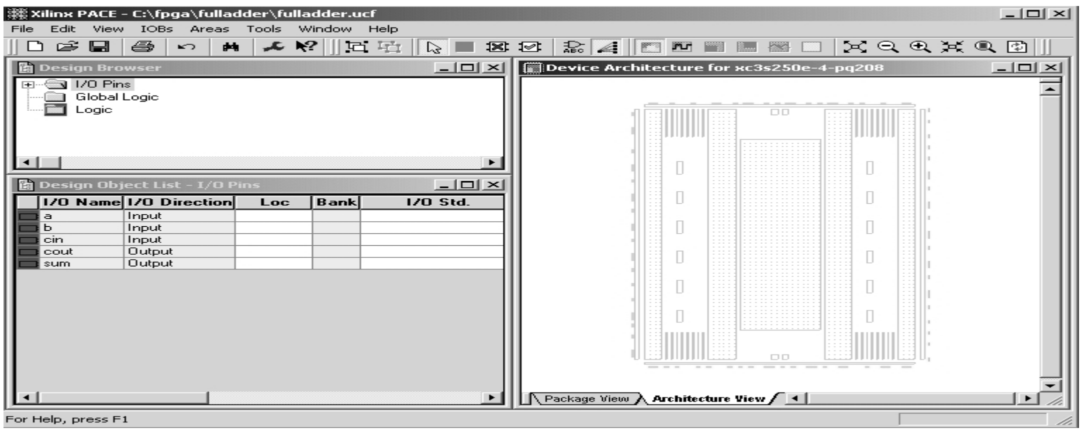
```
// Design Name:
// Module Name:    fulladder
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module fulladder(a, b, cin, sum, cout);
    input a;
    input b;
    input cin;
    output sum;
    output cout;
    wire s1,c1,c2;
    xor (s1,a,b);
    and (c1,a,b);
    xor (sum,s1,cin);
    and (c2,s1,c1);
    or (cout, c1,c2);
endmodule
```

```
Started : "Launching Design Summary".
Started : "Launching ISE Text Editor to edit fulladder.v".
```

- In Sources window, select Synthesis/Implementation. In process, select + in User Constraints.

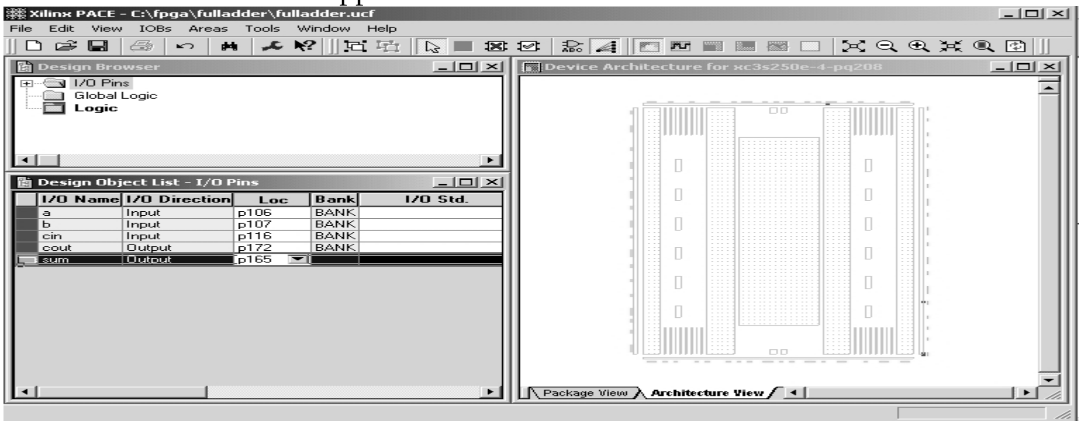


- Select Assign Package Pins. It gives message that User Constraint File will be created. Click Yes.
- Xilinx PACE Software window appears with I/O names. Select Loc.

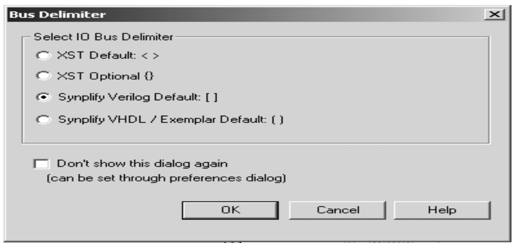

Xilinx PACE - C:\fpga\fulladder\fulladder.ucf
File Edit View IOBs Areas Tools Window Help
Design Browser
I/O Pins
Global Logic
Logic
Design Object List - I/O Pins
Device Architecture for xc3s250e-4-pq208
Package View
Architecture View
For Help, press F1

I/O Name	I/O Direction	Loc	Bank	I/O Std.
a	Input			
b	Input			
cin	Input			
cout	Output			
sum	Output			

- Give Pin numbers say for signal a, l13. In our FPGA Kit, l13 is connected to DIP switch.
- Select input pins from switches and output pins from LED bar graph and then save.
- The Bus Delimiter screen appears. Check OK.

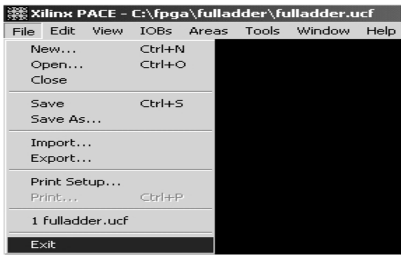

Xilinx PACE - C:\fpga\fulladder\fulladder.ucf
File Edit View IOBs Areas Tools Window Help
Design Browser
I/O Pins
Global Logic
Logic
Design Object List - I/O Pins
Device Architecture for xc3s250e-4-pq208
Package View
Architecture View

I/O Name	I/O Direction	Loc	Bank	I/O Std.
a	Input	p106	BANK	
b	Input	p107	BANK	
cin	Input	p116	BANK	
cout	Output	p172	BANK	
sum	Output	p165		

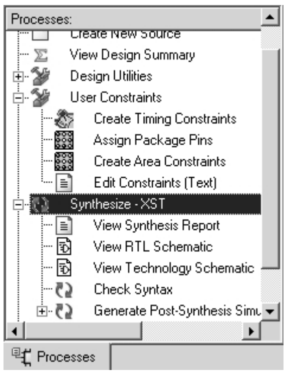

Bus Delimiter

Select IO Bus Delimiter
- XST Default: < >
- XST Optional {}
- [✓ Synplify Verilog Default: [ ]]
- Synplify VHDL / Exemplar Default: ( )

Don't show this dialog again
(can be set through preferences dialog)

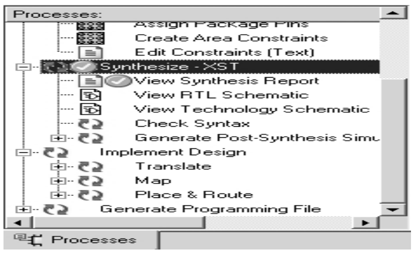


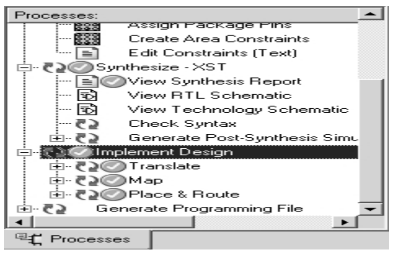
- Then exit from the Xilinx PACE window.
- In the main window, fulladder.ucf file is added.

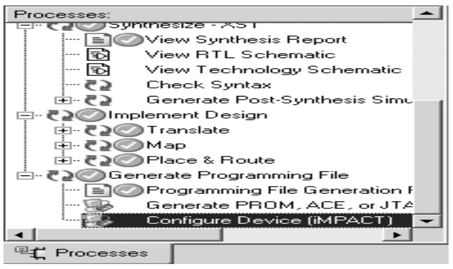


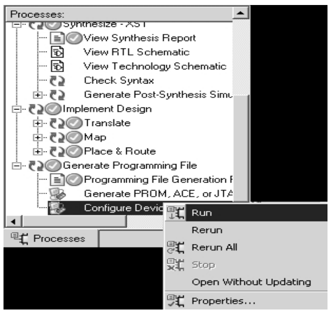
- Click the verilog module. In Processes window, Synthesis, Implement design and Generate Programming file appears.
- Select + in Synthesis.

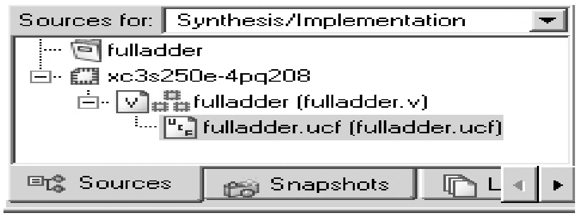
- Select Synthesis and right click and Run.
- After completing the synthesis, a tick mark appears
- Then select + in Implement design and select it. Right click and Run.
- It will run the following : Translate, Map, Place and Route functions
- Then select + in Generate Programming Files and generate it by right clicking and select Run.



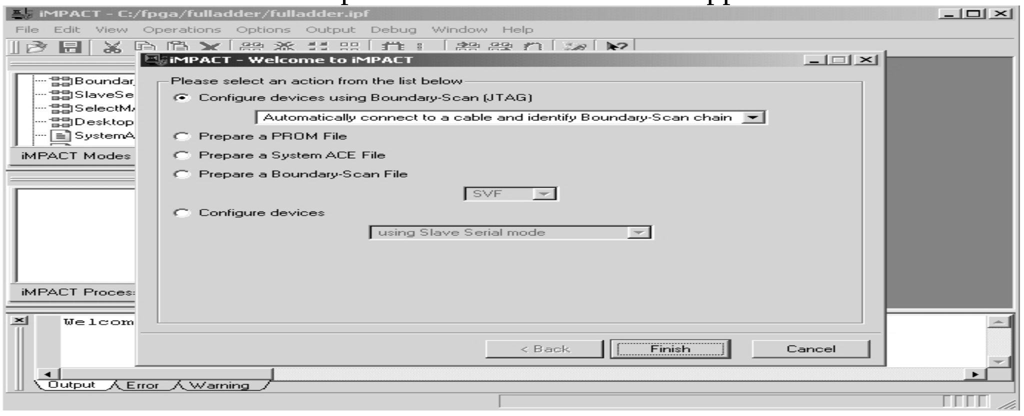



Processes:
Synthesize - XST
View Synthesis Report
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Simu
Implement Design
Translate
Map
Place & Route
Generate Programming File
Programming File Generation R
Generate PROM, ACE, or JTA
Configure Device (iMPACT)
Processes

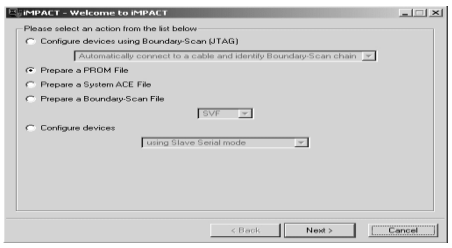



Sources for: Synthesis/Implementation
fulladder
xc3s250e-4pq208
fulladder (fulladder.v)
fulladder.ucf (fulladder.ucf)
Sources
Snapshots
L

- After generating the Programming file, select Configure Device (IMPACT) and right click and Run.
- Xilinx IMPACT software is opened and welcome screen appears.


iMPACT - C:/fpga/fulladder/fulladder.ipf
File Edit View Operations Options Output Debug Window Help
Boundar
SlaveSe
SelectM
Desktop
SystemA
iMPACT Modes
iMPACT Proces
Welcom
Output
Error
Warning
iMPACT - Welcome to iMPACT
Please select an action from the list below
Configure devices using Boundary-Scan (JTAG)
Automatically connect to a cable and identify Boundary-Scan chain
Prepare a PROM File
Prepare a System ACE File
Prepare a Boundary-Scan File
SVF
Configure devices
using Slave Serial mode
< Back
Finish
Cancel

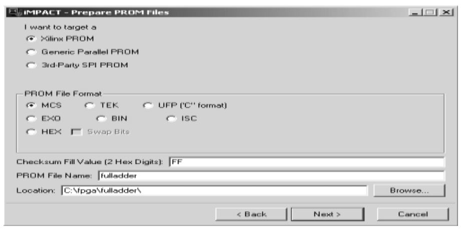
- Select Prepare a PROM File and then next.



iMPACT - Welcome to iMPACT

Please select an action from the list below

- ( ) Configure devices using Boundary-Scan (JTAG) — Automatically connect to a cable and identify Boundary-Scan chain
- (•) Prepare a PROM File
- ( ) Prepare a System ACE File
- ( ) Prepare a Boundary-Scan File — SVF
- ( ) Configure devices — using Slave Serial mode



iMPACT - Prepare PROM Files

I want to target a

- (•) Xilinx PROM
- ( ) Generic Parallel PROM
- ( ) 3rd-Party SPI PROM

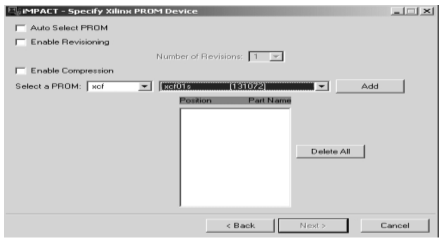
PROM File Format: (•) MCS ( ) TEK ( ) UFP ('C'' format) ( ) EXO ( ) BIN ( ) ISC ( ) HEX ☐ Swap Bits

Checksum Fill Value (2 Hex Digits): FF

PROM File Name: fulladder

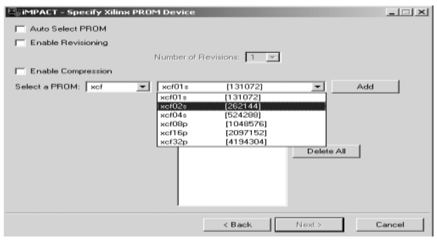
Location: C:\fpga\fulladder\

- In this next screen, select Xilinx PROM and PROM file format MCS. In our FPGA Kit, Xilinx PROM is used to store the program for FPGA and the PROM file format is MCS.
- In the next screen, the particular Xilinx PROM is selected
- In our Kit, xcf04s Platform Flash PROM is present and this device is chosen.
- Then click Add button is pressed to select this PROM.
- In the next screen, the file generation summary is given.
-



iMPACT - Specify Xilinx PROM Device

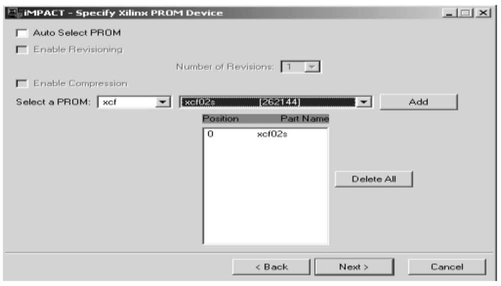
☐ Auto Select PROM
☐ Enable Revisioning
Number of Revisions: 1
☐ Enable Compression
Select a PROM: xcf xcf01s [131072]



iMPACT - Specify Xilinx PROM Device

☐ Auto Select PROM
☐ Enable Revisioning
Number of Revisions: 1
☐ Enable Compression
Select a PROM: xcf xcf01s [131072]

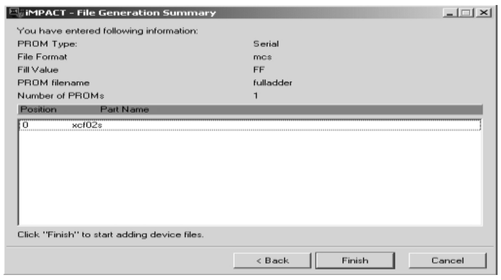
xcf01s	[131072]
xcf02s	[262144]
xcf04s	[524288]
xcf08p	[1048576]
xcf16p	[2097152]
xcf32p	[4194304]



iMPACT - Specify Xilinx PROM Device

☐ Auto Select PROM
☐ Enable Revisioning
Number of Revisions: 1
☐ Enable Compression
Select a PROM: xcf xcf02s [262144]

Position	Part Name
0	xcf02s



iMPACT - File Generation Summary

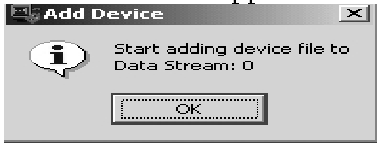
You have entered following information:

PROM Type:	Serial
File Format	mcs
Fill Value	FF
PROM filename	fulladder
Number of PROMs	1

Position	Part Name
0	xcf02s

Click "Finish" to start adding device files.

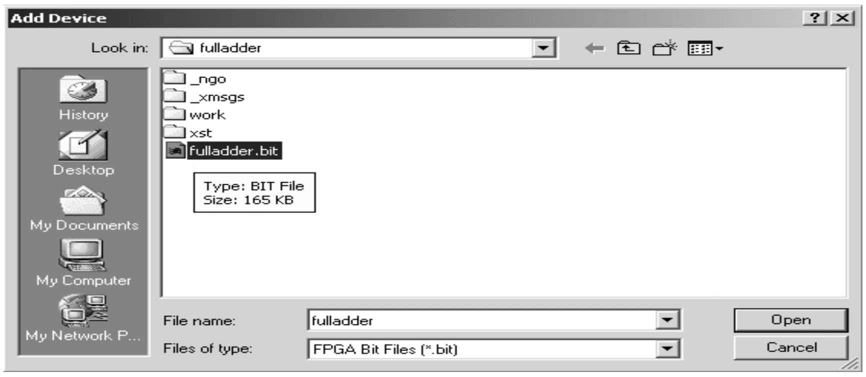
- In the next screen, diagram of Xilinx PROM appears. In this screen, click OK.



Add Device

Start adding device file to Data Stream: 0

- The generated bit file is used to generate the PROM file in the MCS format. The next screen displays the bit file.



Add Device

Look in: fulladder

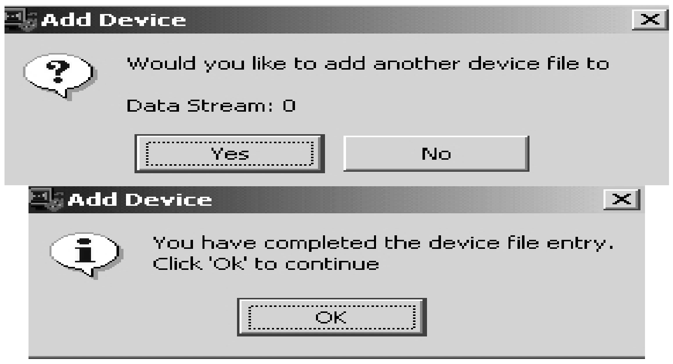
_ngo
_xmsgs
work
xst
fulladder.bit

Type: BIT File
Size: 165 KB

File name: fulladder

Files of type: FPGA Bit Files (*.bit)

- When it asks for adding of additional device, click NO.
- Click OK in the next window when it asks for continuation.


Add Device

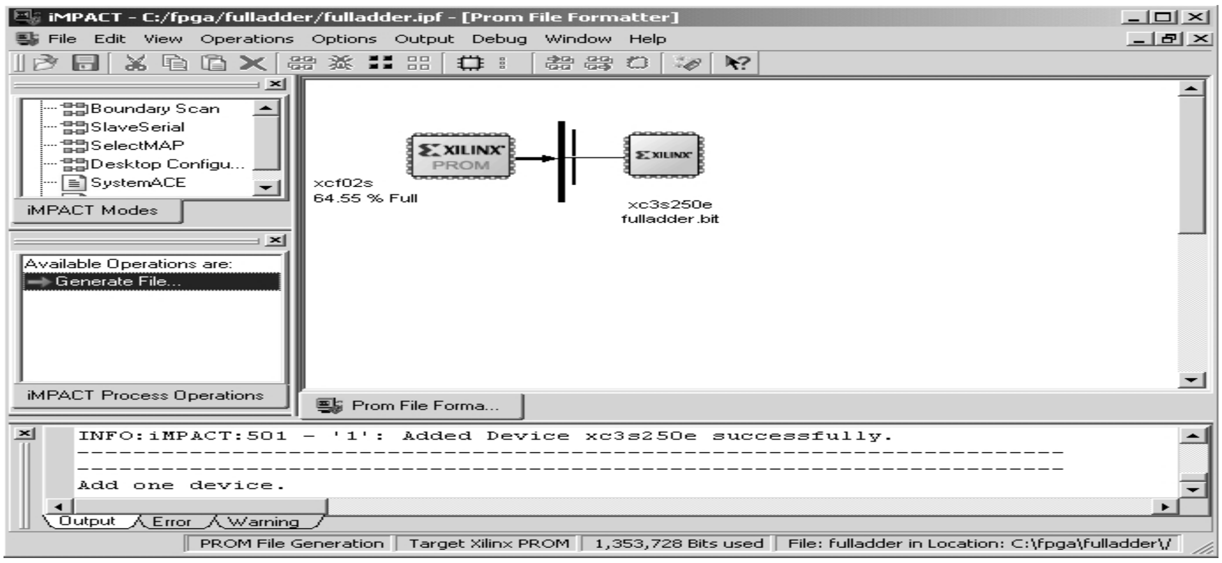
Would you like to add another device file to

Data Stream: 0

Add Device

You have completed the device file entry.
Click 'Ok' to continue

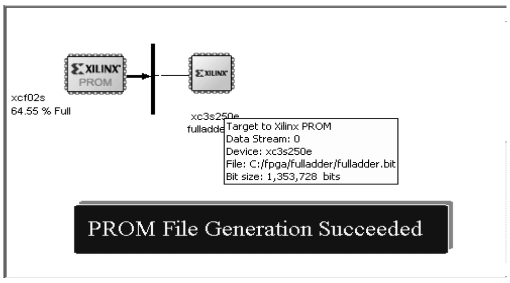
- Click Generate file in the available operations and the PROM file is generated and it is indicated in the screen.
-


iMPACT - C:/fpga/fulladder/fulladder.ipf - [Prom File Formatter]

```
INFO:iMPACT:501 - '1': Added Device xc3s250e successfully.
----------------------------------------------------------------------
----------------------------------------------------------------------
Add one device.
```

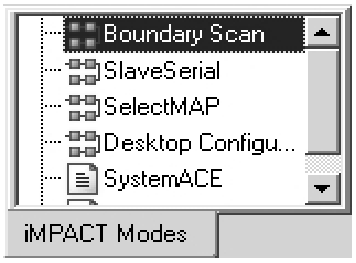
PROM File Generation | Target Xilinx PROM | 1,353,728 Bits used | File: fulladder in Location: C:\fpga\fulladder\/

- Switch on the Kit which is connected to the Parallel port of the system. Double click Boundary Scan in the iMPACT modes. During Boundary scan, the kit should be ON.

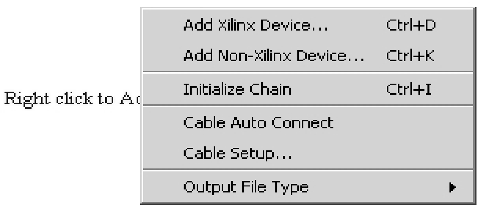


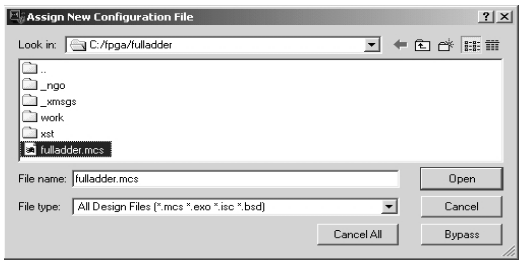
Target to Xilinx PROM
Data Stream: 0
Device: xc3s250e
File: C:/fpga/fulladder/fulladder.bit
Bit size: 1,353,728 bits

PROM File Generation Succeeded



- In the right hand side, plain window appears. Right click in this window and select Initialize chain.




Assign New Configuration File

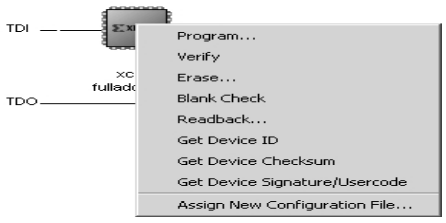
Look in: C:/fpga/fulladder

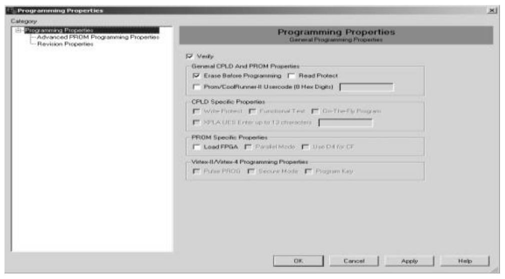
..
_ngo
_xmsgs
work
xst
fulladder.mcs

File name: fulladder.mcs

File type: All Design Files (*.mcs *.exo *.isc *.bsd)

- Next, a window appears with generated PROM file. Select the MCS format file and click Open.





[✓ Verify]

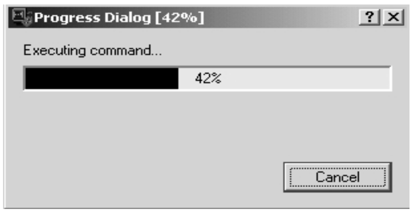
General CPLD And PROM Properties: [✓ Erase Before Programming] Read Protect; Prom/CoolRunner-II Usercode (8 Hex Digits)

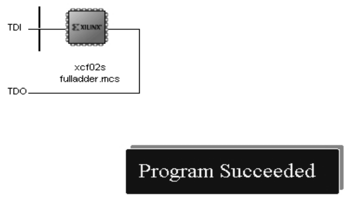
CPLD Specific Properties: Write Protect; Functional Test; On-The-Fly Program; XPLA UES Enter up to 12 characters

PROM Specific Properties: Load FPGA; Parallel Mode; Use D4 for CF

Virtex-II/Virtex-4 Programming Properties: Pulse PROG; Secure Mode; Program Key

- Now, that file has been taken for programming. Keep the cursor on the device and right click.





- Select the Program .Programming Properties screen appears. Select verify and erase before programming.
- Once you click OK, the program is downloaded to the PROM.
- Finally, the "Program succeeded" screen appears.
- In the kit, press the Prog key. Now, the program in the PROM is loaded to the FPGA and the FPGA acts according to the circuit designed.

Result:

Thus the study about various synthesis tools used in the Xilinx - Project Navigator software is studied.