





## **DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

# **17ECCC90-FPGA SYSTEM DESIGN LAB**

Program/ Branch	: B. E. / ECE
Year / Semester	: III/ VI
Academic Year	: 2020 – 2021 (Even Semester)
Regulation	: R 2017

Sund

HOD/ ECE

17ECCC90		) F	FPGA SYSTEM DESIGN LAB				Cate	egory	Ι		Т	Р	Cre	edit	
								С	C	(	)	0	4		2
PREA solution cores	<b>PREAMBLE</b> - This lab-oriented course will focus on the design of large-scale system-on-a-chip (SOC) solutions within field-programmable gate arrays (FPGAs). Modern FPGA densities and commercially available cores enable a single developer to design highly complex systems within a single FPGA.														
PREF	REQU	ISITE	- Nil												
COU	RSE O	BJEC	TIVES	5											
1	To de softw	esign ai vare.	nd simu	ılate ba	sic log	ic circui	its, co	mbinat	ional a	nd sequ	ential lo	ogic circ	uits usir	ng HDL	
2	To in	npleme	nt the o	lesigne	d logic	circuits	in FF	PGA de	evice.						
3	To ve	erify th	e input	and ou	tput of	designe	ed logi	ic circu	iits						
COU	RSE O	UTCO	OMES												
On the	e succe	essful c	omplet	ion of t	the cour	rse, stud	lents v	will be	able to	)					
CO1.	Desig	n and s	imulat	ion of c	ligital l	ogic cire	cuits							Apply	
CO2.	Desig	n and i	mplem	ent the	combin	national	logic	circuit	s in FP	GA dev	vice			Evalua	ate
CO3.	Desigr	n and ir	npleme	ent seve	eral Seq	uential	circui	ts in F	PGA de	evice				Evalua	ate
CO4.	Devel	op con	nplex lo	ogic cir	cuits									Evalua	ate
MAP	PING	WITH	I PRO	GRAM	ME O	UTCON	MES	AND F	PROG	RAMM	E SPEC	CIFIC (	OUTCO	MES	
СО	PO	РО	РО	РО	РО	PO	PO	PO	PO	PO1	PO1	PO1	PSO	PSO	PSO
S	1	2	3	4	5	6	7	8	9	0	1	2	1	2	3
CO 1	М	М	М	М	L	-	М	-	-	-	-	М	М	S	L
CO 2	S	М	S	L	М	-	-	-	-	-	-	М	М	L	-
CO 3	S	S	S	S	L	-	М	-	-	-	L	М	S	S	L
CO 4	S	М	L	L	L	-	-	-	-	-	-	М	М	L	-
S- Str	S- Strong; M-Medium; L-Low														

#### 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.

COURSE DESIGNERS							
S.N	Name of the	Designatio	Department	Mail ID			
0	Faculty	n					
1	Dr. I. V. Hama	Professor	ECE	hamall Qavit as in			
1	Dr.L.K.Heina	& Head	ECE	nemaik@avit.ac.in			
2 Dr T Shaala		Associate	ECE	shaale Oumkuss adu in			
2	DI.I.Sheela	Professor	ECE	sheeta@viikvec.edu.iii			
2	Mr.C.Arunkumar	Assistant	ECE	arunkumarmadhuvappan@vmkvec.edu.in			
3	Madhuvappan	Professor	ECE				
		Assistant	ECE				
4	Mr.S.Selvam	Professor		selvam@avit.ac.in			
		(Gr-II)					

## **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: Table Truth Table:

**OR Gate:** Logic diagram

Y=A+B

Аг

B D

Truth

A B

0 0

0 1

1

0

1 1

Truth

Y=A+B

0

1

1

1

A	o	V-AD
B	0	I=AB

Α	B	Y=AB
0	0	0
0	1	0
1	0	0
1	1	1

NOT Gate: Logic Diagram: Table

Truth Table:



Α	Y=A'
0	1
1	0



**NAND Gate:** 

Logic Diagram

A	В	Y=(ab)'
0	0	1
0	1	1
1	0	1
1	1	0

**Truth Table** 

Α	В	Y=A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

## Truth table:

Α	В	$Y = A \otimes B$
0	0	1
0	1	0
1	0	0
1	1	1



-⊡ Y=A⊗B

λo-

Logic Diagram:

**NOR Gate:** 



**XNOR Gate:** 

A 🛛

B

Logic diagram:

Α	B	Y=(A+B)'
0	0	1
0	1	0
1	0	0
1	1	0

**XOR Gate:** Logic Diagram



Verilog Code:	Simulation Quantum
AND Gale: modulo andgoto(a b. a);	Simulation Output:
input a b:	
output c:	
assign $c \rightarrow k$ h:	
andmodule	
OP Cate	Simulation Output:
module orgate(a b, c):	Sinulation Output.
input a b:	
output c:	
assign $c = a$ b:	
assign $c = a   b$ ,	
NOT Cata	Simulation Output
module notgate(a_abar);	Simulation Output.
input a:	
output abar:	
assign abar = aa	
assign abar $- a$ ,	
NAND Cate:	Simulation output:
module pandasta(a b. c):	Simulation output.
input a b:	
output c:	
$c_{assign} c_{assign} = c_{ass}(a_{ass} b_{ass})$	
assign $C = \sim (a \otimes b),$	
NOP Cata	Simulation Output
module porgate(a b. c):	Simulation Output.
input a b:	
input a,0,	
output $c$ ,	
assign $C = \sim (a \mid b)$ ,	
VOD Coto:	Simulation Output
AUR Gale:	Simulation Output:
input a h	
input a,o,	
output $c$ ,	
assign $c = a + b$ ,	
VNOP Cata	Simulation Output
ANOR Gale:	Simulation Output:
inoute exilorgate(a,0, c),	
mput a,0,	
$\begin{array}{l} \text{output } c, \\ outpu$	
assign $c = \sim (a \land D);$	
enamodule	

## **Result:**

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

## EX.NO:2

#### **BASIC EXPERIMENTS**

#### DATE:

## 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:

Α	В	SUM	CARRY
		<b>(s)</b>	(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:



## **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

## TRUTH TABLE:

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

## 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:

А	В	С	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 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 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

<u>Dataflow Modeling</u>
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

<u>Simulation output:</u>
```



## **FULL SUBTRACTOR:**



## TRUTH TABLE:

Α	B	С	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= $s1^c$ in; assign  $s2= \sim a$ ; assign s3=s2 and b; assign s4=b and c; assign s5=s2 and c; assign borrow=( $s3 \mid (s4) s5$ ; 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:**



## <u>FULL ADDER:</u> <u>Synthesis RTL Schematic:</u>



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



**Result:** 

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

## EX.NO:3 DATE: <u>CODE CONVERTERS-BINARY TO GRAY AND GRAY TO BINARY</u> <u>Aim:</u>

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:

Verilog Code for Gray code to Binary conversion:

#### **Result:**

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

## EX.NO:4

### **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

Α	B	С	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

## **ENCODER:**

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

## **DECODER:**

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&Bc; end endmodule

## **Result:**

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

## EX.NO:5

#### **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 <u>4 BIT COMPARATOR:</u> 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
    req less;
    reg equal;
   req 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

## 8 X 1 MULTIPLEXER & 1X4 DEMULTIPLEXER

#### DATE:

#### 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 <u>MULTIPLEXER AND DEMULTIPLEXER:</u> <u>Algoritm:</u>

**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:



INI	PUT	OUTPUT
s[1]	s[0]	У
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:
```



### **Synthesis RTL Schematic:**



DEMULTIPLEXER: LOGIC DIAGRAM:

#### **Truth table:**

INPUT				OUT	PUT	
D	S0	<b>S</b> 1	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&s0; endmodule

#### Simulation output:



## 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: <u>DESIGN OF FILP-FLOPS USING VERILOG HDL IN XILINX</u> 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.

#### <u>SR Flip Flop:</u> Logic Diagram:



## **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:



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



**Schematic Diagram:** 

## **D Flip Flop:**



Truth Table:

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

## **Simulation Output:**

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



## **RTL Schematic:**



<u>T- Flip Flop:</u> Logic Diagram:



**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

## Schematic Diagram:



Truth Table:

Q(t)	Т	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

**Simulation Output:** 



Data(1)





## **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
Schematic Diagram:



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

## **Simulation Output:**



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

### **EXP NO: 08**

## 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

**DATE:** 

## **Simulation Output:**



## **Result:**

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

## **Exp No: 09**

#### Date:

## AIM:

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

**4 BIT UP COUNTER WITH ASYNCHRONOUS** 

## **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.

#### SHIFT REGISTER

Date:

#### 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</pre>
```

```
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.

#### 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.

## Simulation Output:

```
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
```

## **RTL Schematic:**

#### **Schematic Diagram:**



### **Result:**

Thus the accumulator is simulated and synthesized with verilog program.

#### EXP NO: 12 DATE: <u>DESIGN OF ARITHMETIC LOGIC UNIT USING VERILOG HDL IN XILINX</u> 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; linear\_feedback; wire assign linear feedback = !(out[7] ^ out[3]); always @(posedge clk) if (reset) begin out <= 8'b0; end else if (enable) begin  $out \le {out[6], out[5], }$ 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.

1	Edit View Proje	ct. Sou	rce	Proces	is Window I	terip								 1.777	and the second second
	Reve Project Open Project Open Example Close Project Save Project As			×	No pro	ojectis op	ND A	¥? ×	10.05	10.01	<i>0</i> % a6	(e)   1	680		
	New O Open O Close	url+N url+O			File	->Open Project 01 ->New Project									
61 67	Save As Save As	011+5													
20	Pyrd Cl Print Preview	578+8°													
	Recent Files		<u> </u>												
	Recent Projects		-												
-	Processes				mt Sources	ejų Snapsh	ote = k								
-															

- 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 Projec	t		🖾 New Project Wizard - Create New Project		
Enter a Name and Location for the Project Project Name:	Project Location D:\VLSI\Experiments\Vilinx\		Enter a Name and Location for the Project Project Name: decoder	Project Location D:WLSI\ExperimentsWilms\decoder	
Select the Type of Top Level Source for the Project Top Level Source Type HDL			Select the Type of Top-Level Source for the Project Top-Level Source Type HDL		<b>&gt;</b>
More Info	< Back Next >	Cancel	More Info	< Back Next >	Cancel

- 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 📰		New Project V	Vizard - Create New Sour	ce.	
-Select the Device and Design Flow fo	r the Project		Constant Many Con			
Property Name	Value		Create a New Soc	ace		
Product Category	All	~	0.1			New Source
Family	Spartan3E	~	Source	ne .	Type	Bernove
Device	XC35250E	~	1			- Tomoro
Package	PQ208	~				
Speed	-4	~				
Top-Level Source Type	HDL	~ I				
Synthesis Tool	XST (VHDL/Verilog)	~				
Simulator	Modelsim-SE Verilog	~				
Enable Enhanced Design Summary						
Enable Message Filtering			Creating a new sou	rce to add to the project is option	al. Only one new source can be create	d with the New Project Wizard.
Display Incremental Messages			Additional so	surces can be created and added	d to the project by using the "Project->N	lew Source'' command.
				Existing sources	s can be added on the next page.	
More Info	<back next=""> Can</back>	oel lec	More Info		< Back	Next > Cancel

- 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		🖾 New Source Wizard - Summary
El Conspire & Ackfletone Wicknell     Schwarze     S	in/\decoder	Project Navigator will create a new skeleton source with the following specification: Add to Project Yes Source Type Vetely Addate Source Type Vetely Addate Source Name: decoder v Module Nete Geoder Pot Definition:
Add to project      More Info     < C Back     [	Next > Cancel	<back carcel<="" fixish="" td=""></back>
岡 New Project Wizard - Create New Source - Create a New Source		New Project Wizard - Project Summary  Project Navigstor will create a new project with the following specifications:
Source File Type 1 decoder v Verilog Module	New Source Remove	Projecti Project Mawe: decoder Project Pathi D.\ULDIYExperiments\Xilinx\decoder Top Level Source Type: HDL
Creating a new source to add to the project is optional. Only one new source can be Additional sources can be created and added to the project by using the "Proj	sreated with the New Project Wizard. st⇒New Source* command.	Device: Device: Family: Spartan3E Device: xc3250c Package: pq200 Spend: -4 Synthesis Tool: XST (VHDL/Verilog) Simulator: Nodelsim=SE Verilog Enhanced Perion Summary: enabled
Existing sources can be added on the next page		Ressage Filtering: disabled
More Into	Next > Cancel	< Back Finish Cancel

- 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.

		and the second s		000	DDER Pres	ect Status		
Add Destrop Seasce	Sources for Synthesis./hiplenerdate.st	Design Oversiery 22 Summary	Project File	decoder ise	0	eront State:	New	
Conste Neer Source     Vere Design Guarman     Design Uniter     User Constants     User Constants	······································	California Constantia	Module	decoder ic3/250e-4pg200		• Enter:		
		Contract Report	Target			+ Warnin	p41	
Spretweisten - 200 T     Spretweisten - 200 T     Spretweisten Deutspre     Generature Programmeng Fale		Christian and Warrings	Presbuct Version	158.01		+ Updated		µ 10 03
		Diffuse and Distances	In the local division of the local divisiono	Charles and	10000			
		Children Hennigen	Report	Sitafaa	Generated	Errore	Warnings	Inte
		All Current Mennages	Sardfunds Datud					
		El Sertion Paper M	Taxabaham Report					
		Propert Progenters D Enable Enhanced Design Summary	Map Report					
		Enable Message Filtering     Display Incremental Messages     Enbarriered Design Turnings Contents	Place and Flouis Flouis					
		Shew Errori Shew Warnings	State Taning					
		C Show Clock Paget	BApen					
			<.				-	1.00
Processes	IR\$ Sources (B) Snapshots - P	Fill decoder   30 Decky Summer						

Construction     C	Concernet Systematic Applications (A) Concernet (C)	Control of the second sec
e a	ME Seature gip Snapshots ( )	Al III Alexandro I an Alexandro III Alexandro IIII Alexandro III Alexandro IIII Alexandro III Alexandro III Alexandro III Alexan

- 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

M Alexandration SE PLUS S. 7g			100 Cont 100 Cont
Pile E.S. view Comple Drudate Fields bilitation	10031	M Madarino Ni Pillo 3. 7g	
ar Ro Kits   do an 42 (HS   (b.F) - 100 m 5 (b	ha na ha i te to te	a no ma an	
	<pre>start = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1</pre>	Note         Note <th< th=""><th></th></th<>	

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

	e de	rault																	_
Ed	A Mary	/ Inser	s Porr	net 1	Toola	Wandow	Ψ.												
		36 B	in 1995. 1	an 1	. Dr . J	26 T.e.	± [ ] [	× 100	10,0		DC   [3.3	D.1 D.3	: DN34 D1	K   ∋⊷					
10.00	Zdecos	here/in		1 001 1			dalah bi												6 H
		Ser / M																1	
			IBUP_1																-
				510															
				12.00															
				500															
								_											п
				510									_		-	-	 _		
																			-1
																			-
																			-
																			-
																			- 10
				10100															-
																			-
																	-		
																			- 1
				1000															-
				<b>H</b>															- 1
																			-
																			- 11
																			- 11
																			-1
																			-
	Adverter	Sec. 24 13																	-
																	_		- 11
																			-1
																			- 1
		Section 2.		500															
							and the second second			-							 Same and the second second		48
		E14	rang 1	_	Size.	81 Z ma												99	66
							10.1												-

**Program of any Combinational Circuits:** 

#### **Result:**

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

## Exp No: 15

## 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.

		and know - for - estimptin (remitteden geminten)	(Territoria)	- INCLUSI
	1 10 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Y File Edit View Project Source Process	Window Help	
		TOSHG TOBROS	29 TE AL DE LA 19 18 19 19 19 19 19 19 19 19 19 19 19 19 19	
Inte Drojo	ct Navigator X	100000V00000	二월 22 / 4 / 4 / 4 / 4 / 2 / 2 / 4 / 4 / 4	
see riuje		15.	7 // Design Namei	
		Sources for: Synthesis/Implemental *	8 // Rodule Name: Tulladder	
		- @ fuladder	9 // Project Name:	
		(ii)- [ii] xc3x250e-4pq208	1 // Tool versions:	
	meters and a set of the set of th	·····[v]@fs/uladder(tuladder.v)	2 // Description:	
( <b>a</b> )	This process requires that an Implementation Constraint File (LICE) be added	1	3 // Dependencies:	
<b>Y</b> 1	This process requires that an impomentation constraint his (our yes added	and designed in the second sec	5 //	
	and the second second second to the second	-to sources else shappings i 1	6 // Revision:	
1 V J	to the project and associated with the selected design module. Would you like	(.8J 1	7 // Pevision 0.01 - File Created	
N		Processes: 1	G // Additional Commence:	
-	Design Manufactor to automatically sease a LICE and add it to the project at	- C Add Existing Source 2	o ////////////////////////////////////	
	Project Navigator to automatically create a OCF and add it to the project at	Create New Source 2	1 module fulladder (a, b, cin, sum, cout);	
		- X View Design Summary	a input bi	
	this time?	an Se User Constraints 2	4 input cini	
	uns une:	DO-CO Synthesize - XST	5 output sum	
	and the first of the second se	Implement Design	vire al. cl. c2)	
	It you select "No" you will need to create or add an existing LICE to the project	⊕-€⊉ Generale Programming File	8 xor (s1,a,b);	
	If you beloce the you minimous to create or add an existing out to the project		9 and (c1, a, b) /	
	h of our monotone that a monotone		and (d2, d1, c1);	
	Defore running this process.	3	a or (cout, cl.cl);	
		2	3 endmodule	
		4 1 1	•	-
		St Decement		
			Design Summay: 🔽 Auladder	
		B Started : "Launching Desi	Ign Summary".	-
	I: YES II INO I	I I and a second s		
	1	Started   "Launching ISE	Text Editor to edit fulladder.v".	
		Til Console CEnors / Warning	n an Fridan Elles	
			ILN I COLI CAVE HUM	ECRE Vertoo

• 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.

ﷺ Xilinx PACE - C:\fpga\fulladder\fulladder.ucf	
File Edit View IOBs Areas Tools Window Help	
🗅 😂 😓 🛤 🖊 😢    🗄 🗟 🔜 😂	2 2 4
Design Browser	Device Architecture for xc3s250e-4-pq208
Global Logic	
Logic	
L/O Name L/O Direction Loc Bank L/O Std.	
a Input	
cin Input	
sum Output	
For Help, press F1	

- Give Pin numbers say for signal a, 113. In our FPGA Kit, 113 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	
🗅 😂 🔜 🗠 🗰 🦊 🛠 🛛 🗓 🔁 🔂 🔜 🗃	፼ \$ 4   ■ ■ ■ ■ × ٩ € × ٩ ₽
Design Browser      Design Browser      Design Object List - 1/0 Pins      Design Object List - 1/0 Pins      Design Object List - 1/0 Pins      Design Object List - 1/0 Pins	Device Architecture for xc3s250e-4-pq208
a Input p106 BANK b Input p106 BANK cin Input p116 BANK cout Dutput p118 BANK cout Dutput p172 BANK cout Dutput p172 BANK	Package View





- 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. Assign Fackage Fins Create Area Constraints Edit Constraints (Text) View RTL Schematic View RTL Schematic Check Syntax Generate Post-Synthesis Simu Check Syntax Generate Post-Synthesis Simu Place & Route Processes Assign Fackage Fins Assign Fackage Fins Create Area Constraints Edit Constraints (Trext) Call Constraints (Trext) Call Synthesis Factor Wiew Synthesis Report Wiew RTL Schematic Check Syntax Call Constraints (Trext) C 22 t Processes [ C Processes ses: IS ynthesize - AS T ces View BTL Schematic View Technology Sche View Technology Schematic
     Check Syntax
     Generate Post-Synthesis Simu
     Implement Design
     Office & Route
     Generate Programming File
     Generate Programming File
     Generate PROM, ACE, or JTA
     Generate PROM, ACE, or JTA View Technology Schematic Check Syntax C Processes View Synthesis Report View Synthesis Report View RTL Schematic Check Syntax Generate Post-Swetcesses: View Technology Schematic View Technology Schematic Check Syntax Sources for: Synthesis/Implementation -… 河 fulladder nerate PROM. ACE. or JTA Configure Devid Ė.- Va#afulladder (fulladder.v) Reru Pro ିଁដ Rerun All ucf (fulladder.ucf) السلم Open Without Updating Properties.. ⊡t‡ Sources 😁 Snapshots
- After generating the Programming file, select Configure Device (IMPACT) and right click and Run.

- 0 ×

• Xilinx IMPACT software is opened and welcome screen appears.

File Edit View O	perations Options Output Debug Window Help	
1 🖻 🖬  🕹 🖻		1
	IMPACT - Welcome to IMPACT	
	Please select an action from the list below	
	<ul> <li>Configure devices using Boundary-Scan (JTAG)</li> </ul>	
	Automatically connect to a cable and identify Boundary-Scan chain 💌	
📄 SystemA	C Prepare a PROM File	
iMPACT Modes	C Prepare a System ACE File	
	Prepare a Boundary-Scan File	
	SVF -	
	using Slave Serial mode	
I IMPACT Proces:		
Melcom		<u>^</u>
		-
	AWarning	<u> </u>

• Select Prepare a PROM File and then next.

IMPACT - Welcome to IMPACT	
Please select an action from the list below	
Configure devices user Boundary Scan (Idaa) Foregare a PROM File Propers a PROM File Propers a Boundary-Scan File Propers a Boundary-Scan File Configure devices Configure devices Userg Stave Stetial mode	
< Baok Next >	Cancel

- 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	UiMPACT - Specify Xilinx PROM Device	
Auto Select PROM	Auto Select PROM	
Enable Revisioning	Enable Revisioning	
Number of Flevisions: 1 💌	Number of Revisions: 1 💌	
Enable Compression	Enable Compression	
Select a PROM: xcf 💌 KSUIP KEID/A	Select a PROM: xcf 💌 xcf01s (131072)	Add
Position Part Name	xcf01s [131072]	
	xcf04s [524299]	
	xcf08p [1048576] xcf15p [2097152]	
	xcf32p [4194304]	- 1
Delete Al	Delete All	
< Back Next > Cancel	< Back Next >	Cancel
MPACT - Specify Xilins PROM Device	MPACT - File Generation Summary	
C Auto Select PROM		
	Tou have entered following information:	
Enable Revisioning	Tou have entered following information: PRDM Type: Serial	
Enable Revisioning	Tou have entered following information: PROM Type: Serial File Format mcs	
Enable Revisioning     Number of Revisions:	Tou have entered following information: FRIOM Type: Serial File Format mcs Fill Value FF	
Enable Revisioning     Number of Revision: 1     Enable Compression	You have entered following information: Serial PRION Type: Serial File Format mcs Fill Value FF PRION filename fulladder	
Enable Revisioning     Number of Revisions     Enable Compression     Select a PROM: work      Model     Model	Tou have entered following information PRIOM Type: Serial File Format mcs Fill Value FF PRIOM filename fulladder Number of PRIOMs 1	
Enable Revisions     Number of Revisions     Enable Compression     Select a PROM: vol      Totolog     Pointing     Pointing     Pointing	Tou have entered rollowing information: PFIOM Type: File Format mos Fill Value FF PFIOM filename fulladder Number of PFIOMs 1 Position Part Name	
Enable Revisioning Number of Revision: Select a PRIOM: wof Park Name Position Park Name Operation Park Name Operation Park Name Operation Data Data Data Data Data Data Data Dat	You have entered rollowing information         Serial           FPICID Type:         Serial           File Format         mcs           File Yable         FF           PPICIM Iterative         Iterative           PRIDM Iterative         Number of PPIOMs           Position         Pat Name           [0         x0/2x	
Enable Revisions     Number of Revisions     Enable Compression     Select a PROM: vol      Contion     Part Name     0	You have entered rollowing information:     Serial       PRION Type:     Serial       File Format     mos       File Value     FF       PRION flemame     fulladder       Number of PRIOM ≠     1       Position     Patt Name       0     xcf02 ≠	
Enable Revisions     Number of Revisions     Enable Compression     Select a PROM: xef     Position     Part Name     0     xef02a	PFICUT Type:     Serial       PFICUT Type:     mcs       File Format     mcs       File Tormat     File       PFICUM Interance     Killadder       Number of PFICUs     1       Position     Pat Name       0     xer02s	
Enable Revisioning     Number of Revision:     Enable Comparison     Select a PRIOM: Mol     Control     Cont	PRION Type:     Serial       PRION Type:     Serial       File Format     mcs       File Value     FF       PRION filename     Kulladder       Number of PRIONs     1       Position     Part Name       0     xcl/02s	
	PFICUT Type:     Serial       PFICUT Type:     mcs       File Format     mcs       File Tormat     File       PFICUT Memory     File       PFICUT Memory     File       POinter of PFIDMs     1       Position     Part Name       0     xef02e	
Enable Revisions     Number of Revisions     Enable Compression     Select a PRIOM: vol      Control     Contro     Control     Control     Control     Contr	PRION Type:     Serial       PRION Type:     Serial       File Format     mcs       File Value     FF       PRIOM fleename     fulladder       Number of PRIONs     1       Position     Part Name       0     xclr02s	
Enable Revision:     Number of Revision:     Enable Compression:     Select a PROM: Med     Position     Robition     Robition     Revision	You have entered following information     Serial       FPIO Type:     Serial       File Format     mcs       File Tomat     File       PTION filename     Kulladder       Number of PPIOMs     1       Position     Pat Name       0     xcl/02e	;
	PRION Type:     Serial       PRION Type:     Serial       File Format     mcs       File Value     FF       PRION filename     fulladder       Number of PRIONs     1       Position     Part Name       0     xci02s	
	Vol have entered following information  PF(UI Type: Serial  Pfe Format mics  Pfi Value PF  PF(D)h famme Nated of PF0Ms 1  Pasterior Part Name  O sc(02s  Cick: "Finish" to start adding device lifes.	
	Vol have entered relationing information: PFIOM Type: Serial File Format mics File Value FF PFIOM Iterame Mulladder Number of PFIOM 1  Position Pat Name    Click: "Finish" to start adding device files.	
	You have entered following information:     FFIGURA       FFIGURA     mcs       FIGURA     FFIGURA       FIGURA     FFIGURA       PROM figura     FFIGURA       Number of FRIDMs     1       Position     Part Name       0     scil/02s       Click: "Finish" to start adding device files.	uth Cancel

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



• 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	-	] + • • •	]-
History Desktop My Documents My Computer	ngo xmsgs work xst fulladder.bit Size: 165 KB	8		
	File name:	fulladder	-	Open
My Network P	Files of type:	FPGA Bit Files (*.bit)	-	Cancel

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



• 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]	
I File Edit View Operations Options Output Debug Window Help	_ 8 ×
沙    × 10 13 × (器 ※ # #   # # #   # # # 0   ≫   ?	
Image: Start	
Available Operations are:	J
MPACT Process Operations Frie Forma	
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\full	adder\/ 🥢

• 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.



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

Right click to A(	Add Yiliov Device	Christo
		cultur
	Add Non-Xilinx Device	Ctri+K
	Initialize Chain	Ctrl+I
	Cable Auto Connect	
	Cable Setup	
	Output File Type	•

Assign New Configuration File	? ×
Look in: C:/fpga/fulladder	1 🕹 🔠 🏢
work	
🖸 fulladder.mcs	
File name:  fulladder.mcs	Open
File type: All Design Files (*.mcs *.exo *.isc *.bsd)	Cancel
Cancel All	Bypass

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

		Advanced PROM Programming Properties     Revision Properties	Programming Properties Gameral Programming Properties
DIEN	Program Verify Erase Blank Check Readback Get Device ID Get Device Checksum		P Vete           Orman LG 20 Ard PROF Negatives           P Lease Mains Programming
-	Assign New Configuration File		DR. Cancel Apply

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

Progress Dialog [42%]	
42%	xcr02a fullaader.mcs
Cancel	Program Succeeded

- 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.