Vivekananda College of Engineering & Technology
[A Unit of Vivekananda Vidyavardhaka Sangha Puttur ®]
Affiliated to Visvesvaraya Technological University
Approved by AICTE New Delhi & Recognised by Govt of Karnataka

TCP02
Rev 1.4
EC
16/08/24

COURSE PLAN

A. COURSE OVERVIEW

Degree:	B.E.	Programme:	EC
Academic Year:	2024-25	Semester:	III
Course Title:	Digital System Design Using Verilog	Course Code:	BEC302
L-T-P-S:	3-0-2-0	Duration of SEE	3 Hrs
Total Contact Hours:	40 hours theory + 8-10 lab slots	SEE Marks:	50*
CIE Marks:	50	IA Test	25
Credits:	4	Lab Components	25
Lesson Plan Author:	Mr. Shreyas H	Sign	Date 16/08/24
Checked By:	Mr. Akshay S P	Sign Asp	Date 16/08/24

*The SEE will be conducted for 100 marks and proportionally reduced to 50 marks.

B. PREREQUISITES

- The fundamentals of Introduction to Electronics Engineering (BESCK104C-204C).
- The basic knowledge of Introduction to C Programming (BESCK104E-204E).
- The fundamentals of Introduction to Electrical Engineering (BESCK104B-204B) .

C. COURSE DESCRIPTION

i) Course Outcomes

At the end of the course, the student will be able to;

1. Simplify Boolean functions using K-map and Quine-McCluskey minimization technique.
2. Analyze and design for combinational logic circuits.
3. Analyze the concepts of Flip Flops (SR, D, T and JK) and to design the synchronous sequential circuits using Flip Flops.
4. Model Combinational circuits (adders, subtractors, multiplexers) and sequential circuits using Verilog descriptions.

ii) Relevance of the Course

- VLSI Design and Testing
- VLSI Laboratory
- Advanced VLSI
- Advanced Design Tools for VLSI

iii) Applications areas

- Verilog hardware description language has become an industry standard as a result of extensive use in the design of integrated circuit chips and digital systems.
- Supports mixed-level design representations comprising switches, gates, RTL, and higher levels of abstractions of digital circuits.
- Design of FPGA and ASIC.
- Information (computers), Telecommunications, Control systems etc.

	Asp	
Prepared by: Shreyas H	Checked by: Akshay S P	HOD

COURSE PLAN

D1. ARTICULATION MATRIX, CO v/s PO

Mapping of CO to PO												
	POs											
COs	1	2	3	4	5	6	7	8	9	10	11	12
1. Simplify Boolean functions using K-map and Quine-McCluskey minimization technique.	3	3	2	-	-	-	-	-	-	-	1	3
2. Analyze and design for combinational logic circuits.	3	3	2	1	3	-	-	-	2	-	1	3
3. Analyze the concepts of Flip Flops (SR, D, T and JK) and to design the synchronous sequential circuits using Flip Flops.	3	3	2	-	3	1	-	-	2	-	2	3
4. Model Combinational circuits (adders, subtractors, multiplexers) and sequential circuits using Verilog descriptions.	3	3	2	1	3	1	-	-	2	-	2	3

D2. ARTICULATION MATRIX, CO v/s PSO

Mapping of CO to PSO		
	PSOs	
COs	1	2
1. Simplify Boolean functions using K-map and Quine-McCluskey minimization technique.	3	-
2. Analyze and design for combinational logic circuits.	3	-
3. Analyze the concepts of Flip Flops (SR, D, T and JK) and to design the synchronous sequential circuits using Flip Flops.	3	1
4. Model Combinational circuits (adders, subtractors, multiplexers) and sequential circuits using Verilog descriptions.	3	1

E. MODULE PLANS

MODULE – I

Title:	Principles of Combinational Logic	Appr. Time:	8 Hrs
MO:			RBT
At the end of the Module, the student will be able to:			
1. Define Combinational logic and canonical forms.			L1
2. Describe combinational logic by Boolean equations, truth tables or logic diagrams.			L2
3. Construct the truth table for various verbal problem statement.			L3
4. Develop the karnaugh maps for reducing Boolean equations.			L3
5. Minimize the logic circuits by applying various reduvtion techniques like K-map and Quine-McCluskey.			L3
Lesson Schedule:			

Lecture No.	Portion to be covered	CO

1	Definition of combinational logic, canonical forms,	CO1
2	Generation of switching equations from truth tables,	CO1
3	Karnaugh maps-3,4,5 variables,	CO1,2
4	Karnaugh maps-3,4,5 variables,	CO1,2
5	Karnaugh maps-3,4,5 variables,	CO1,2
6	Incompletely specified functions (Don‘t care terms).	CO1
7	Simplifying Max term equations,	CO1,2
8	Quine-McClusky techniques – 3 & 4 variables	CO1,2

Application Areas:

- Karnaugh maps are used to simplify real-world logic requirements so that they can be implemented using a minimum number of physical logic gates.
- The Karnaugh map reduces the need for extensive calculations by taking advantage of humans pattern-recognition capability.
- Q-M method are used to simplify real-world logic requirements so that they can be implemented using a minimum number of physical logic gates.
- Q-M method reduces the need for extensive calculations by taking advantage of humans pattern-recognition capability.

Review Questions / Questions Appeared in the Previous Years (CO):

1 Define the following: i)TVUM table, ii)Combinational Circuit, iii)Canonical SOP, iv)Canonical POS. (CO1)

2 Define K-map, incompletely specified function, essential prime implicants and grey code. (CO1)

3 Write the truth table and design a circuit to generate output using K-map for the problem statement given:output of a combinational circuit having 4 inputs and an output becomes logical '1' when two or more inputs go to logical level '1'. (CO2)

4 Simplify the following logical expressions using Karnaugh maps, $Y=\bar{A}B+AB+AB$(CO2)

5 Express the following function in standard SOP form, F1=AB+CD+ABC(CO1)

6 Simplify using Kamaugh map. Write the Boolean equation and realize using NAND gates. D = f(w,x,y,z) = $\sum(0,2,4,6,8)+\sum d(10,11,12,13,14,15)$ (CO2)

7 Use the K-map method to simplify the following function (CO2)
i. f1(A,B,C,D,E)=Σm(8,9,10,11,13,15,16,18,21,24,25,26,27,30,31)
ii. f2(A,B,C,D)=Σm(1,3,5,8,9,11,15)+d(2,13)

8 Two motors M2 and M1 are controlled by three sensors S3, S2 and S1. One motor M2 is to run any time all three sensors are on. The other motor is to run whenever sensors S2 or S1 but not both are on and S3 is off. For all sensor combinations where M1 is on, M2 is to be off except when all the three sensors are off and then both motors must remain off. Construct the truth table and write the Boolean output equation.(CO2)

9 Use the Quine-McCluskey method to simplify the following function (CO2)
i. f1(A,B,C,D,E)=Σm(8,9,10,11,13,15,16,18,21,24,25,26,27,30,31)
ii. f2(A,B,C,D)=Σm(1,3,5,8,9,11,15)+d(2,13)

10 Define prime implicant and essential prime implicant. Find prime implicant and essential prime implicants for the following function using Quine-McClusky method:
f(a, b, c, d) = Σm(0, 2, 3, 6, 7, 8, 10 , 12, 13) (CO1)

11 Design a combinational circuit to output the 2's complement of a 4 bit binary number.(CO2)

12 Identify all prime implications and essential prime implications of following function using k-map.
f(a, b, c, d) = Σm(6, 7, 9, 10, 13) +dc(1, 4, 5, 11, 15). Draw the diagram using NAND gates. (CO2)

13	Explain briefly K-map, incompletely specified functions, essential prime implicants and Grey code. (CO1)

MODULE – II

Title:	Logic Design with MSI Components and Programmable Logic Devices	Appr. Time:	8 Hrs

MO:	RBT
At the end of the Module, the student will be able to:	
1 Understand the concepts of adders, subtractors, Multiplier and dividers	L2
2 Understand the concepts of Decoder, Encoder, Multiplexer and Demultiplexer	L2
3 Understand the importance of look ahead carry adder,binary comparators etc.	L2
4 Understand Programmable Logic Devices.	L2
5 Construct various combinational circuits.	L3

Lesson Schedule:

Lecture No.	Portion to be covered	CO
1	Decoders,	CO1,2
2	Encoders,	CO1,2
3	Digital multiplexers	CO1,2
4	Adders and subtractors,	CO1,2
5	Look ahead carry,	CO1,2
6	Binary comparators.	CO1,2
7	Programmable Logic Devices,	CO1,2
8	Complex PLD.	CO1,2

Application Areas:

- MUX is used as a single pole multiposition switch.
- A desktop PC uses a parallel adder, because it wants to be able to do as much arithmetic as possible in a given amount of time
- Encoders are used to convert a certain type of characters into other type of signals.
- For example encoders are used to convert wav file into mp3 format and also they are used in modem(ADC). Because modem receives analog inputs from transmission line but computers only understand digital binary inputs.
- Application of decoders are code converters, implementation of combinational circuits, address decoding, BCD to 7-segment decoder.

Review Questions / Questions Appeared in the Previous Years (CO):

1	Define encode, decoder, priority encoder and multiplexer. (CO1)
2	Write block diagram representation of a full adder using 3:8 decoders. (CO1)
3	Design full adder using i)8:1 MUX and ii)4:1 MUX (CO2)
4	Explain full adder using 3:8 decoder. (CO1)
5	Design 2 bit comparator and briefly explain. (CO2)
6	Draw the logic diagram of 2 to 4 decoder with active low enable and active high data outputs. Construct a truth table and identify the data inputs, enable inputs and the outputs. Describe the circuit function. Draw the logic symbol.(CO2)
7	Explain the terms(CO1) i) Ripple- carry propagation ii) Propagation delay iii) Look- ahead carry
8	Design a circuit that realizes the following two functions using a decoder and logic gates:

	F1(A, B) = $\Sigma m(0, 3)$ and F2(A, B) = $\Sigma m(1, 2)$(CO2)
9	Define encoder. Design decimal-to-BCD encoder?(CO2)
10	Write a short note on Priority encoder.(CO1)
11	Implement the following using 3 to 8 decoder with active low enable and active high outputs. i) f_1 (a, b, c, d) = $\Sigma m(0, 1, 5, 6, 7, 9, 10, 15)$ ii) f_2(a, b, c) = $\Pi(1, 3, 6, 7)$ (CO2)
12	Explain 4-bit carry look-ahead adder with necessary diagrams and relevant expressions. (CO1)
13	Design 4 line to 2 line priority encoder which gives MSB the highest priority and LSB least priority. (CO2)

MODULE – III

Title:	Flip-Flops and its Applications	Appr. Time:	8 Hrs

MO: At the end of the Module, the student will be able to:	RBT
1 Introduce the basics of digital circuits using building blocks such as gates and Flip Flops.	L1
2 Distinguish between combinational and sequential circuits, between synchronous and asynchronous sequential circuits	L2
3 Describe the flip flops with their characteristic equation.	L2
4 Understand the applications of sequential circuits	L2
5 Differentiate between counters and registers	L2

Lesson Schedule:

Lecture No.	Portion to be covered	CO
1	Basic Bistable elements,	CO3
2	Latches,	CO3
3	The master-slave flipflops : SR flip-flops, JK flip-flops,	CO3
4	D Flipflops, T flipflops	CO3
5	Characteristic equations,	CO3
6	Registers	CO3
7	Binary ripple counters,	CO3
8	Synchronous binary counters.	CO3

Application Areas:

•Flip flops can be used extensively to transfer the data.

• Flip flops are used for digital data storage and hence can be used in computers in the form of memory elements.

• Another major application of flip flops is a digital counter. It is used to count pulses or events and it can be made by connecting a series of flip flops. The major application of a shift register is parallel to serial data conversion. Shift

•registers are also used as keyboard encoders. Sequential circuits are used to sequence data for microprocessors.

• Counters are one of the many applications of sequential logic that has a widespread use from

•simple digital alarm clocks to computer memory pointers.

Review Questions / Questions Appeared in the Previous Years (CO):

1	Define bistable element, latch, flip flop and function table.(CO3)
2	Sketch timing diagrams for JK flip flop and D flip flop. (CO3)
3	Explain M/S JK flip-flop with the help of circuit diagram and waveforms. (CO3)
4	Find characteristic equations for T and SR flip-flops with the help of function tables. ((CO3)

	(CO3)
5	List the functions of asynchronous inputs(CO3)
6	Explain the different types of flip-flops along with their truth table. Also explain the race-around condition in a flip-flop.(CO3)
7	Explain the concept of johnson counter with neat circuit diagram.(CO3)
8	State various applications of shift register.(CO3)
9	With the help of a diagram, explain the following with respect to shift register:(CO3) I) Ring counter and twisted ring counter using JK flip flop
10	List the basic types of shift registers in terms of data movement.(CO3)
11	Explain clocked SR flip-flop using NAND gates with necessary truth table and waveform.(CO3)
12	Explain with neat diagram and truth table, a four bit SIPO shift register to store binary number 1011. (CO3)
13	What is race around condition? Explain JK master slave flip flop with a diagram, function table and timing diagram. (CO3)

MODULE – IV

Title:	Introduction to Verilog, Verilog Data flow description	Appr. Time:	8 Hrs
MO:			RBT
At the end of the Module, the student will be able to:			
1.Understand instantiation of gates, gate symbols, and truth tables for and/or and buf/not type gates.			L2
2. Construct a Verilog description from the logic diagram of the circuit.			L3
3. Describe data flow description in verilog HDL.			L2
4. Describe the continuous assignment (assign) statement, restrictions on the assign statement and the implicit continuous assignment statement.			L2
5. Define expressions, operators, and operands			L1

Lesson Schedule:

Lecture No.	Portion to be covered	CO
1	Introduction to Verilog	CO4
2	Structure of Verilog module	CO4
3	Operators	CO4
4	Operators	CO4
5	Data Types	CO4
6	Styles of Description	CO4
7	Highlights of Data flow description	CO4
8	Structure of Data flow description	CO4

Application Areas:

- Design of logic gates and adder and subtractor circuits.
- Design of multiplexers.
- Design of flip-flops.
- Design of counters

Review Questions / Questions Appeared in the Previous Years (CO):

1	Write a verilog code to implement all the logic gates. (CO4)
2	Write a verilog code and test bench program for 4:1 multiplexer. (CO4)
3	Write a verilog description for 4 bit ripple carry adder with stimulus and also show the output wave form. (CO4)

4	Explain the concept of gate delay. (CO4)
5	Explain the continuous assignment statement in data flow modeling. (CO4)
6	Explain the expressions, operators and operands of verilog. (CO4)
7	Using a data flow description write a program for full adder. (CO4)
8	Write a verilog code for edge triggered D – flipflop using data flow modeling. (CO4)

MODULE – V

Title:	Verilog Behavioral description, Verilog Structural description	Appr. Time:	8 Hrs

MO: At the end of the Module, the student will be able to:	RBT
1.Explain the significance of structured procedures always and initial in behavioral modeling.	L2
2. Understand delay-based timing control mechanism in behavioral modeling. Use regular delays, intra-assignment delays, and zero delays	L2
3. Use level-sensitive timing control mechanism in behavioral modeling.	L2
4. Explain conditional statements using if and else and describe multiway branching, using case, casex, and casez statements	L2
5. Explain the significance of structured procedures always and initial in structural modeling.	L2

Lesson Schedule:

Lecture No.	Portion to be covered	CO
1	Verilog Behavioral description:	CO4
2	Structure, Variable Assignment Statement,	CO4
3	Sequential Statements, Loop Statements	CO4
4	Verilog Behavioral Description of Multiplexers	CO4
5	Verilog Structural description:	CO4
6	Highlights of Structural description,	CO4
7	Organization of structural description,	CO4
8	Structural description of ripple carry adder.	CO4

Application Areas:

- Expresses a digital circuit in terms of the algorithm it implements
- Delay based timing control
- Event based timing control and level sensitive timing control

Review Questions / Questions Appeared in the Previous Years (CO):

1	Explain the significance of structured procedures always and initial in behavioral modeling. (CO4)
2	Define blocking and non blocking procedural assignment. (CO4)
3	Explain the delay based timing control mechanism in behavioral modeling. (CO4)
4	Define sequential and parallel blocks. (CO4)
5	Write a verilog code for 4:1 multiplexer using case statements. (CO4)
6	Explain the concept of loop with an example. (CO4)
7	List out the differences between task and function(CO4)
8	Explain task declaration and invocation (CO4)
9	Explain function declaration and invocation (CO4)
10	Explain automatic task and function along with example(CO4)

COURSE PLAN

F. LABORATORY CONTENTS:

Expt No.	Title of the Experiments	RBT	CO
1	To simplify the given Boolean expressions and realize using Verilog program.	L4	CO1,2,4
2	To realize Adder/Subtractor (Full/half) circuits using Verilog data flow description.	L4	CO1,2,4
3	To realize 4-bit ALU using Verilog program.	L4	CO1,2,4
4	To realize the following Code converters using Verilog Behavioral description a) Gray to binary and vice versa b) Binary to excess3 and vice versa	L4	CO1,2,4
5	To realize using Verilog Behavioral description: 8:1 mux, 8:3 encoder, Priority encoder	L4	CO1,2,4
6	To realize using Verilog Behavioral description: 1:8 Demux, 3:8 decoder, 2-bit Comparator	L4	CO1,2,4
7	To realize using Verilog Behavioral description: Flip-flops: a) JK type b) SR type c) T type and d) D type		CO1,2,3,4
8	To realize Counters - up/down (BCD and binary) using Verilog Behavioral description.	L4	CO1,2,3,4
9	Verilog Program to interface a Stepper motor to the FPGA/CPLD and rotate the motor in the specified direction (by N steps).	L4	CO1,2,3,4
10	Verilog programs to interface Switches and LEDs to the FPGA/CPLD and demonstrate its working.	L4	CO1,2,3,4

EXPERIMENTS

1. EXPERIMENT NO:1

2. TITLE: TO SIMPLIFY THE GIVEN BOOLEAN EXPRESSIONS AND REALIZE USING VERILOG PROGRAM.

3. LEARNING OBJECTIVES:

- To understand the concept of Boolean Expressions.
- Apply Demorgan's Theorem to complex Boolean expressions.
- To simplify the given expression using K-Map.
- To simulate,synthesize and implement Boolean Expressions using HDL on FPGA.

4. AIM:

- TO SIMPLIFY THE GIVEN BOOLEAN EXPRESSIONS AND REALIZE USING VERILOG PROGRAM.

5. MATERIAL REQUIRED:

- FPGA Board, FRC's jumper, Power Supply

6. THEORY:

- The output of any logical circuit can be expressed in terms of the equation called as the Boolean expression. There are several basic terms to denote the Boolean expression. Minterm is the special case product term that contain all of the input variable, that make the Boolean expression. Maxterm is the special case sum term that contains all the input variables that make the Boolean expression. Karnaugh map is used to eliminate the redundancies. It is the matrix of squares where each square represents a minterm or the maxterm of the Boolean expression.

7. FORMULA:

- Y=BC+BD

- Y=B(C+D)

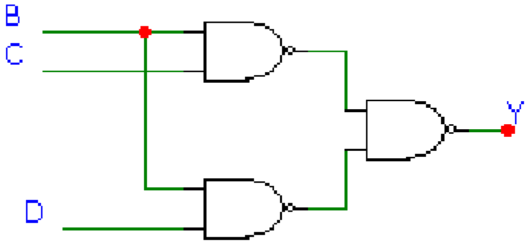
8. PROGRAMME:

VERILOG CODE	TEST BENCH

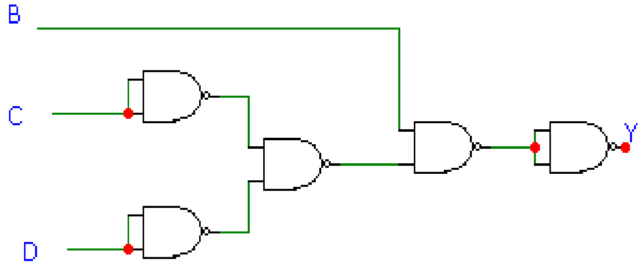
module be1(Y, B, C, D);
output Y;
 input B, C, D;
assign Y = (B & C) | (B & D);
endmodule
	module be1_tb;
	wire Y;
	reg B,C,D;
	be1 uut (.B(B),.C(C),.D(D),.Y(Y));
	 initial begin
	B=0;C=0;D=0;#100;
	 B=0;C=0;D=1;#100;
	B=0;C=1;D=0;#100;
	B=0;C=1;D=1;#100;
	B=1;C=0;D=0;#100;
	B=1;C=0;D=1;#100;
	B=1;C=1;D=0;#100;
	B=1;C=1;D=1;#100;
	end
	endmodule

module be2(Y, B, C, D);
output Y;
 input B, C, D;
assign Y = B & (C | D);
endmodule
	module be2_tb;
	wire Y;
	reg B,C,D;
	be2 uut (.B(B),.C(C),.D(D),.Y(Y));
	 initial begin
	B=0;C=0;D=0;#100;
	 B=0;C=0;D=1;#100;
	B=0;C=1;D=0;#100;
	B=0;C=1;D=1;#100;
	B=1;C=0;D=0;#100;
	B=1;C=0;D=1;#100;
	B=1;C=1;D=0;#100;
	B=1;C=1;D=1;#100;
	end
	endmodule

9. BLOCK DIAGRAM:
a) Y=BC+BD
Using NAND gates:



b) Y=B(C+D)
Using NAND gates:



10. TRUTH TABLE:

For Demorgan's Theorem:

1. $\overline{(A+B)}=\overline{(A)}\,\overline{(B)}$

A	B	$\overline{A+B}$	$\bar{A}\bar{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

2. $\overline{(AB)}=\overline{(A)}+\overline{(B)}$

A	B	$\overline{AB}$	$\bar{A}+\bar{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

For SOP

$Y=\Sigma m(5,6,7,13,14,15)$

AB CD	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

- Reduced expression Y= BC + BD
- Truth table:

B	C	D	BC	BD	BC+BD
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1

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

For POS:

$Y = \Pi M(0,1,2,3,4,8,9,10,11,12)$

CD \ AB	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

- Reduced Expression: Y=B(C+D)
- Truth table:

B	C	D	C+D	Y=B(C+D)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

11. RESULTS & CONCLUSIONS:

- Boolean expression circuit is realized using HDL codes.

12. LEARNING OUTCOMES :

- Boolean expression circuit is realized and simulated using Verilog code.
- Boolean expression is synthesized and implemented on FPGA.

13. APPLICATION AREAS:

- Demorgan's theorem is used to design many digital circuit.
- Karnaugh maps are used to simplify real world logic requirements, so that they can be implemented using a minimum number of physical logic gates.
- Karnaugh maps are used to facilitate the simplification of Boolean algebra functions.

14. REMARKS:

-
-

-

1. EXPERIMENT NO: 2

2. TITLE: TO REALIZE ADDER/SUBTRACTOR (FULL/HALF) CIRCUITS USING VERILOG

DATA FLOW DESCRIPTION.

3. LEARNING OBJECTIVES:

- To realize the adder circuits using Verilog.
- To realize the subtractor circuit using verilog.

4. AIM:

- TO REALIZE ADDER/SUBTRACTOR (FULL/HALF) CIRCUITS USING VERILOG DATA FLOW DESCRIPTION.

5. MATERIAL REQUIRED:

- FPGA Board, FRC's jumper, Power Supply

6. THEORY:

- Half-Adder: A combinational logic circuit that performs the addition of two data bits, A and B, is called a half-adder. Addition will result in two output bits; one of which is the sum bit, S, and the other is the carry bit, C. The Boolean functions describing the half-adder are:

$$S = (A \oplus B) \quad , \qquad C = A.B$$

- Full-Adder: The half-adder does not take the carry bit from its previous stage into account. This carry bit from its previous stage is called carry-in bit. A combinational logic circuit that adds two data bits, A and B, and a carry-in bit (Cin), is called a full-adder. The Boolean functions describing the full-adder are:

$$S = (A \oplus B) \oplus Cin \quad , \qquad C = AB + Cin \quad (A \oplus B)$$

- Half Subtractor: Subtracting a single-bit binary value B from another A (i.e. A-B) produces a difference bit D and a borrow out bit Bo. This operation is called half subtraction and the circuit to realize it is called a half subtractor.
- Full Subtractor: Subtracting two single-bit binary values, B, C from a single-bit value A produces a difference bit D and a borrow out Bo bit. This is called full subtractor.

7. FORMULA:

- **Half Adder:**

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

C=AB

Half Subtractor:

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

$B_0 = \overline{A} \quad B$

- **Full Adder:**

$$S = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC$$

C_{OUT}=AB+BC+AC

Full Subtractor:

$$D = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC$$

$$B0 = \overline{A}\,B + BC + \overline{A}C$$

8. PROGRAMME:

VERILOG CODE	TEST BENCH

module half_adder (A , B , S , C);
input A , B;
output S , C;
xor (S , A , B);
and (C , A , B);
endmodule
	module tb_half_adder();
	reg a , b ;
	wire s , c;
	half_adder uut(.a(a),.b(b),.s(s),.c(c));
	initial
	begin

	a = 1'b0;
	b = 1'b0;
	#100
	a = 1'b1;
	b = 1'b0;
	#100
	a = 1'b0;
	b = 1'b1;
	#100
	a = 1'b1;
	b = 1'b1;
	end
	
	endmodule

Module Half_Subtractor_2(output D,B, input X,Y);
assign D = X ^ Y;
assign B = ~X & Y;
endmodule
	module Half_Subtractor_2_tb;
	wire D, B;
	reg X, Y;
	Half_Subtractor_2 uut (.D(D), .B(B), .X(X), .Y(Y));
	initial begin
	X = 0; Y = 0;
	#1 X = 0; Y = 1;
	#1 X = 1; Y = 0;
	#1 X = 1; Y = 1;
	X = 1'b0;
	Y = 1'b0;
	#100;
	X = 1'b1;
	Y = 1'b0;
	#100;
	X = 1'b0;

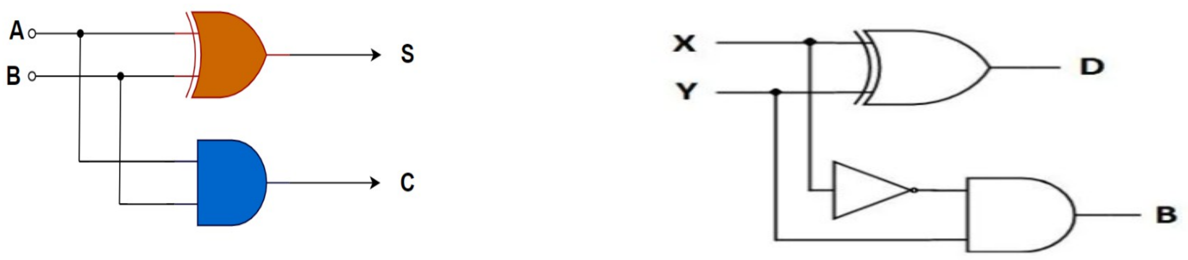
	Y = 1'b1;
	#100;
	X = 1'b1;
	Y = 1'b1;
	end
	endmodule

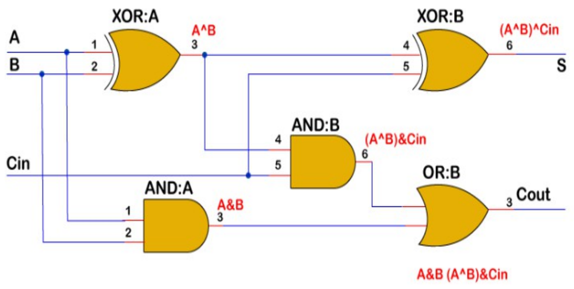
module fulladder (input [3:0] a,
input [3:0] b,
input c_in,
output c_out,
output [3:0] sum);
assign {c_out, sum} = a + b + c_in;
endmodule
	module tb_fulladd;
	reg [3:0] a;
	reg [3:0] b;
	reg c_in;
	wire [3:0] sum;
	integer i;
	fulladd fa0 (.a (a),
	.b (b),
	.c_in (c_in),
	.c_out (c_out),
	.sum (sum));
	initial begin
	#100 a=4'd0;
	b=4'd0;
	c_in=4'd0;
	#100 a=4'd0;
	b=4'd0;
	c_in=4'd1;
	#100 a=4'd0;
	b=4'd1;
	c_in=4'd0;
	#10 a=4'd0;
	b=4'd1;
	c_in=4'd1;
	#10 a=4'd1;

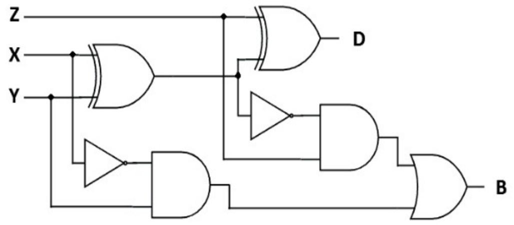
	b=4'd0;
	c_in=4'd0;
	 #10 a=4'd1;
	b=4'd0;
	c_in=4'd1;
	      end
	endmodule

module Full_Subtractor_3(output D, B, input X, Y, Z);
assign D = X ^ Y ^ Z;
assign B = ~X & (Y^Z) | Y & Z;
endmodule
	include "Full_Subtractor_3.v"
	module Full_Subtractor_3_tb;
	wire D, B;
	reg X, Y, Z;
	Full_Subtractor_3 Instance0 (D, B, X, Y, Z);
	initial begin
	    X = 0; Y = 0; Z = 0;
	#1 X = 0; Y = 0; Z = 1;
	#1 X = 0; Y = 1; Z = 0;
	#1 X = 0; Y = 1; Z = 1;
	#1 X = 1; Y = 0; Z = 0;
	#1 X = 1; Y = 0; Z = 1;
	#1 X = 1; Y = 1; Z = 0;
	#1 X = 1; Y = 1; Z = 1;
	end
	endmodule

9. BLOCK DIAGRAM:







10. TRUTH TABLE:

- **Half adder:**

Input		*Output*	
A	*B*	*S*	*C*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- **Full Adder:**

Input			*Output*	
A	B	C	S	C
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

- **Half Subtractor:**

Input		*Output*	
A	B	D	B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

- **Full subtractor:**

Input			*Output*	
A	B	C	D	B_o
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

11. RESULTS & CONCLUSIONS:

- The truth table of half/full adder and half/full subtractor circuits is verified using verilog.

12. LEARNING OUTCOMES :

- The half/full adder and half/full subtractor are realized using logic gates and outputs are verified.

13. APPLICATION AREAS:

- The ALU (arithmetic logic circuitry) of a computer uses half adder to compute the binary addition operation on two bits.
- Half adder is used to make full adder as a full adder requires 3 inputs, the third input being an input carry i.e. we will be able to cascade the carry bit from one adder to the other.
- Ripple carry adder is possible to create a logical circuit using multiple full adders to add N-bit numbers. Each full adder inputs a C(in), which is the C(out) of the previous adder. This kind of adder is called **Ripple Carry Adder**, since each carry bit "ripples" to the next full adder. Note that the first full adder (and only the first) may be replaced by a half adder.
- Full adder reduces circuit complexity.
- The optical signal processing is one of the application of the half subtractor circuit. Binary numbers is another application.

14. REMARKS:

- -
- -

-

1. EXPERIMENT NO: 3

2. TITLE: TO REALIZE 4-BIT ALU USING VERILOG PROGRAM.

3. LEARNING OBJECTIVES:

- To understand the functionalities of an ALU
- To simulate,synthesize and implement 4 bit ALU using HDL on FPGA.

4. AIM:

To design a model for 4 bit ALU, that uses combinational logic to calculate an output, based on the four bit op-code input, that passes the result to the out bus when enable line in high, and tri-state the out bus when the enable line is low and decodes the 4 bit op-code according to the given below.

- A + B
- A – B
- A+1
- A-1
- A
- A Complement
- A AND B
- A OR B

5. MATERIAL / EQUIPMENT REQUIRED:

- FPGA Board, FRC's jumper, Power Supply

6. THEORY / HYPOTHESIS:

- The arithmetic logic unit (ALU) is a digital circuit that calculates an arithmetic operation (like an addition, subtraction, etc.) and logic operations (like XOR, AND, NOT etc.,) between two numbers. The ALU is a fundamental building block of the central processing unit of a computer.

7. FORMULA / CALCULATIONS:

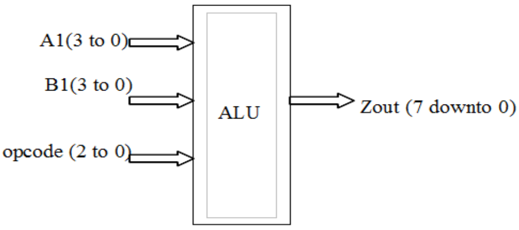
- A + B
- A – B
- A+1
- A-1
- A
- A Complement
- A AND B
- A OR B

8. PROGRAMME:

<table>
<tr><th>VERILOG CODE</th><th>TEST BENCH</th></tr>
<tr><td>module ALU(result,a,b,opcode,en,ack);</td><td>module test;</td></tr>
<tr><td>output [4:0]result;</td><td>reg [3:0] a,b;</td></tr>
<tr><td>input [3:0]a,b;</td><td>reg[2:0] opcode;</td></tr>
<tr><td>input [2:0]opcode;</td><td>reg en;</td></tr>
<tr><td>input en;</td><td>wire [4:0]result;</td></tr>
<tr><td>reg[4:0] result;</td><td>ALU (result,a,b,opcode,en);</td></tr>
<tr><td>always@(en,opcode,a,b)</td><td>initial</td></tr>
<tr><td>begin</td><td>a=4'b0000;b=4'b1010; en=1'b0;</td></tr>
<tr><td>if(en==0)</td><td>opcode=3'b000;</td></tr>
<tr><td>result=4'bzzzz;</td><td>#10 en=1'b1;</td></tr>
<tr><td>else</td><td>#10 opcode=3'b001;</td></tr>
<tr><td>case(opcode)</td><td>#10 opcode=3'b010;</td></tr>
<tr><td>3'b000:result=a+b;</td><td>#10 opcode=3'b011;</td></tr>
<tr><td>3'b001:result=a-b;</td><td>#10 opcode=3'b100;</td></tr>
<tr><td>3'b010:result=a+1;</td><td>#10 opcode=3'b101;</td></tr>
<tr><td>3'b011:result=a-1;</td><td>#10 opcode=3'b110;</td></tr>
<tr><td>3'b100:result=a;</td><td>#10 opcode=3'b111;</td></tr>
<tr><td>3'b101:result=~a;</td><td>#10 $finish;</td></tr>
<tr><td>3'b110:result=a|b;</td><td>end</td></tr>
<tr><td>3'b111:result=a&b;</td><td>endmodule</td></tr>
<tr><td>endcase</td><td></td></tr>
</table>

end
endmodule	

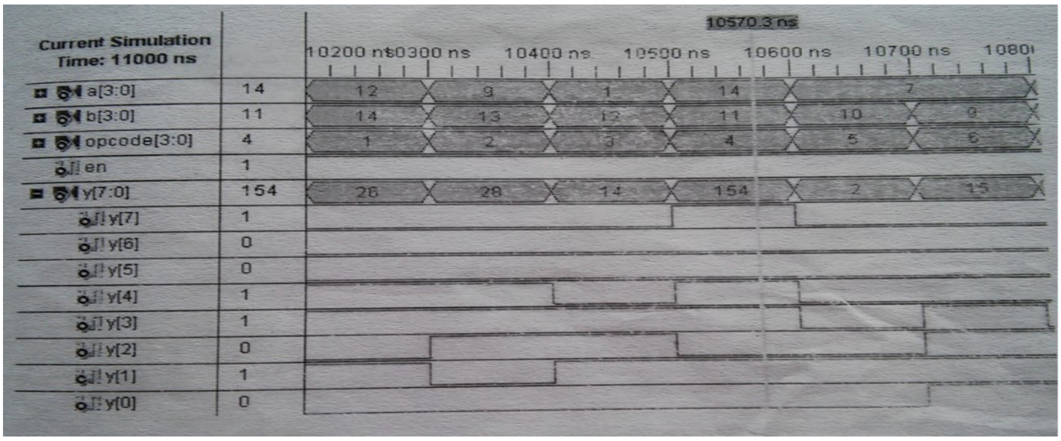
9. BLOCK / CIRCUIT / MODEL DIAGRAM / REACTION EQUATION:



10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:

Operation	Opcode
a+b	000
a-b	001
a+1	010
a-1	011
a	100
~a	101
a and b	110
a or b	111

11. OUTPUTS:



12. RESULTS & CONCLUSIONS:

- 4 bit ALU model is realized using HDL codes.

13. LEARNING OUTCOMES :

- 4 bit ALU model is realized and simulated using Verilog codes.
- 4 bit ALU model is synthesized and implemented on FPGA.

14. APPLICATION AREAS:

- Design of CPU's
- Computing systems

15. REMARKS:

- -
- -

-

1. EXPERIMENT NO:4

2. TITLE: TO REALIZE THE FOLLOWING CODE CONVERTERS USING VERILOG BEHAVIORAL DESCRIPTION A) GRAY TO BINARY AND VICE VERSA B) BINARY TO EXCESS3 AND VICE VERSA

3. LEARNING OBJECTIVES:

- To understand the technique of binary to gray code conversion and binary to excess3 and vice versa.
- To simulate and synthesize and implemen 4 bit binary to gray code conversion and binary to excess3 using HDL on FPGA.

4. AIM:

- TO REALIZE THE FOLLOWING CODE CONVERTERS USING VERILOG BEHAVIORAL DESCRIPTION A) GRAY TO BINARY AND VICE VERSA B) BINARY TO EXCESS3 AND VICE VERSA

5. MATERIAL REQUIRED:

- FPGA Board, FRC's jumper, Power Supply

6. THEORY:

- A Gray code represents each number in the sequence of integers as a binary string of length N in an order such that adjacent integers have Gray code representations that differ in only one bit position. The advantage of the gray code is that only one bit will change as it proceeds from one number to the next. To obtain gray code, one can start with any bit combination by changing only one bit from 0 to 1 or 1 to 0 in any desired random fashion, as long as two numbers do not have identical code assignments.
- There are the following steps to convert the binary number into Excess-3 code: Convert the binary number into decimal. Add 3 in each digit of the decimal number. Find the binary code of each digit of the newly generated decimal number.

7. FORMULA:

a) 3-bit Binary to gray conversion:

$$G2 = B2$$

$$G1 = B2 \oplus B1 = B2\overline{B1} + \overline{B2}B1$$

$$G0 = B1 \oplus B0 = B1\overline{B0} + \overline{B1}B0$$

b)3-bit gray to binary conversion:

$$B2 = G2$$

$$B1 = B2 \oplus G1 = B2\overline{G1} + \overline{B2}G1$$

$$B0 = B1 \oplus G0 = B1\overline{G0} + \overline{B1}G0$$

a) 4-bit BCD to Excess-3 Converter:

$$E0 = \overline{A}$$
$$E1 = BA + \overline{B}\,\overline{A}$$
$$E2 = C\overline{B}\,\overline{A} + \overline{C}A + \overline{C}B$$
$$E3 = D + CA + CB$$

b)4-bit Excess-3 to BCD Converter:

$$A = \overline{E0}$$
$$B = \overline{E1}E0 + E1\overline{E0}$$
$$C = E2E1E0 + \overline{E2}\,\overline{E1} + \overline{E2}\,\overline{E0}$$
$$D = E3E2 + E3E1E0$$

8. PROGRAMME:

VERILOG CODE

```
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
```

```
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
```

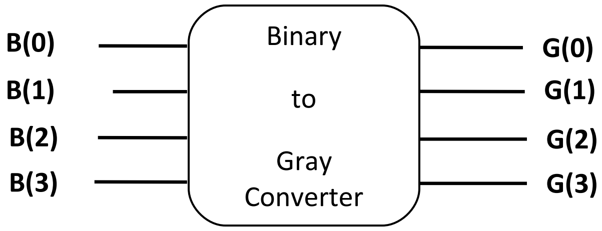
Test bench

```
module tb();
  reg [3:0] bin;
   wire [3:0] G,bin_out;
  bin2gray uut1(bin,G);
  gray2bin uut2(G,bin_out);
  always
  begin
    bin <= 0; #10;
      bin <= 1;   #10;
      bin <= 2;   #10;
      bin <= 3;   #10;
      bin <= 4;   #10;
      bin <= 5;   #10;
      bin <= 6;   #10;
      bin <= 7;   #10;
      bin <= 8;   #10;
      bin <= 9;   #10;
      bin <= 10;  #10;
      bin <= 11;  #10;
      bin <= 12;  #10;
      bin <= 13;  #10;
      bin <= 14;  #10;
      bin <= 15;  #10;
      #100;
    $stop;
  end
endmodule
```

```
module bcd_ex3_Dataflow(
   input a,
    input b,
    input c,
    input d,
    output w,
    output x,
    output y,
    output z
    );

assign w = (a | (b & c) | (b & d));
assign x = (((~b) & c) | ((~b) & d) | (b & (~c) &
(~d)));
assign y = ((c & d) | ((~c) & (~d)));
assign z = ~d;

endmodule
```

```
module ex3_to_bcd(
    input w,
    input x,
    input y,
    input z,
    output a,
    output b,
    output c,
    output d
    );
assign a = ((w & x) | (w & y & z));
assign b = (((~x) & (~y)) | ((~x) & (~z)) | (x & y
& z));
assign c = (((~y) & z) | (y & (~z)));
assign d = ~z;

endmodule
```

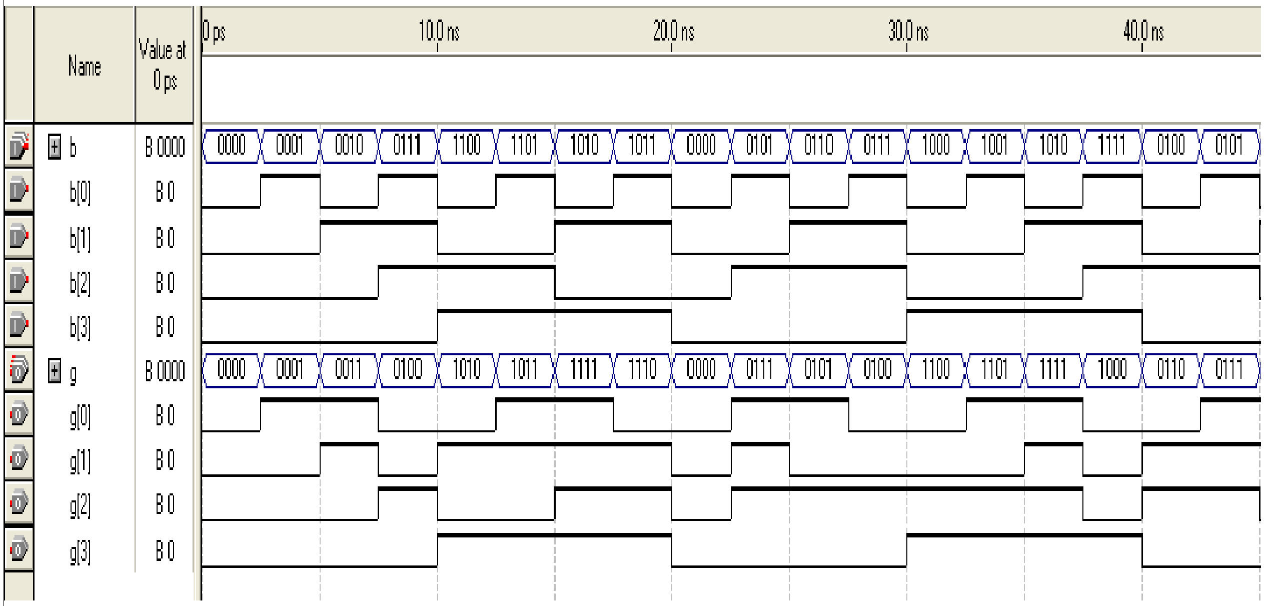
9. BLOCK / CIRCUIT / MODEL DIAGRAM / REACTION EQUATION:



10. TRUTH TABLE:

Binary Codes				Gray Codes			
B(3)	B(2)	B(1)	B(0)	G(3)	G(2)	G(1)	G(0)
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

11. OUTPUTS:



12. RESULTS & CONCLUSIONS:

- 4 bit binary to gray conversion and vice versa is realized using HDL codes.
- 4 bit binary to excess3 conversion and vice versa is realized using HDL codes.

13. LEARNING OUTCOMES :

- 4 bit binary to gray conversion and excess 3 conversion is realized and simulated using Verilog code.
- 4 bit binary to gray conversion and excess 3 conversion is synthesized and implemented on FPGA.

14. APPLICATION AREAS:

- Telegraphy.
- Position transducers.
- Digital communications.

15. REMARKS:

- -
- -

-

1. EXPERIMENT NO:5a

2. TITLE: TO REALIZE USING VERILOG BEHAVIORAL DESCRIPTION: 8:1 MUX

3. LEARNING OBJECTIVES:

- To understand the concept of multiplexer.
- To simulate,synthesize and implement 8:1 multiplexer using HDL on FPGA.

4. AIM:

- To realize and simulate 8:1 multiplexer using VERILOG.

5. MATERIAL / EQUIPMENT REQUIRED:

- FPGA Board, FRC's jumper, Power Supply

6. THEORY:

- A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. A multiplexer is also called as data selector, since it selects one of many inputs and steers the binary information to the output line. The selection of particular input line is controlled by a set of selection lines. Normally there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

7. PROGRAMME:

VERILOG CODE

i) Using case statement

```
module mux81(y,i,e,s);
input e;
input [7:0]i;
input [2:0]s;
output y;
reg y;
always @(e,s,i)
begin
if (e==0)
y=1'bz;
```

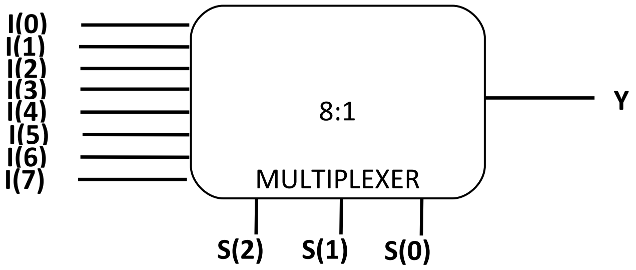
TEST BENCH

```
module testmux;
        reg [7:0] i;
        reg e;
        reg [2:0] s;
        wire y;
mux81 uut (
                .y(y),
                .i(i),
                .e(e),
                .s(s)
        );
```

```
else if(e==1)
begin
case (s)
3'd0:y=i[0];
3'd1:y=i[1];
3'd2:y=i[2];
3'd3:y=i[3];
3'd4:y=i[4];
3'd5:y=i[5];
3'd6:y=i[6];
3'd7:y=i[7];
endcase
end
end
endmodule
```

```
initial begin
            i = 0;
            e = 0;
            s = 0;

#10;
i = 8'b01010101;e = 1;
s = 3'b001; #10;
s = 3'b010; #10;
s = 3'b011; #10;
s = 3'b100; #10;
s = 3'b101; #10;
s = 3'b110; #10;
s = 3'b111; #10;
end
endmodule
```

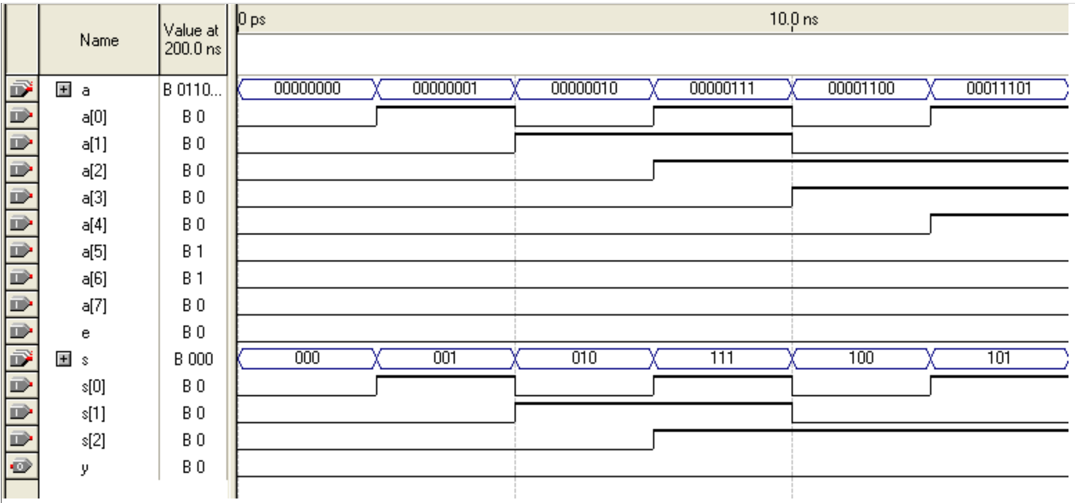
8. BLOCK DIAGRAM:



9. TRUTH TABLE:

Inputs				Outputs
en	s(2)	s(1)	s(0)	Y
0	x	x	x	0
1	0	0	0	i(0)
1	0	0	1	i(1)
1	0	1	0	i(2)
1	0	1	1	i(3)
1	1	0	0	i(4)
1	1	0	1	i(5)
1	1	1	0	i(6)
1	1	1	1	i(7)

10. OUTPUTS:



11. RESULTS & CONCLUSIONS:

- 8:1 multiplexer is realized using HDL codes.

12. LEARNING OUTCOMES :

- 8:1 multiplexer is realized and simulated using Verilog code.
- 8:1 multiplexer is synthesized and implemented on FPGA.

13. APPLICATION AREAS:

- Digital communication.
- Video processing.
- Digital broadcasting.

14. REMARKS:

- -
- -

-

1. EXPERIMENT NO: 5b

2. TITLE: TO REALIZE USING VERILOG BEHAVIORAL DESCRIPTION: 8:3 ENCODER, PRIORITY ENCODER

3. LEARNING OBJECTIVES:

- To learn about the encoder circuits.
- To simulate,synthesize and implement 8:3 encoder using HDL on FPGA.

4. AIM:

- To realize, simulate and synthesis 8 to 3 encoder (without priority and with priority) using Verilog.

5. MATERIAL REQUIRED:

- FPGA Board, FRC's jumper, Power Supply

6. THEORY:

- An encoder is a digital function that produces a reverse operation from that of decoder. An encoder has 2^n (or less) input lines and n output lines. The encoder assumes that only one input line can be equal to 1 at any time. Otherwise the circuit has no meaning. If the encoder has 8 inputs and could have $2^8 = 256$ possible input combinations.
- Priority Encoder: These encoders establish an input priority to ensure that only the highest priority line is encoded. For example 8-to-3-line priority encoder the inputs are I(0) ,I(1) ,

…….I(7). If the priority is given to an input with higher subscript number over one with a lower subscript number, then if both I(2) and I(5) are logic-1 simultaneously, the output will be 101 because I(5) has higher priority over I(2).

7. PROCEDURE / PROGRAMME / ACTIVITY:

8:3 encoder without priority

VERILOG CODE

```
module encoder(x,y,e);
input e;
input [7:0]i;
output [2:0]y;
reg [2:0]y;
always @(e,i)
begin
if (e==1'b0)
y=3'b000;
else
begin
case (x)
8'b00000001:y=3'b000;
8'b00000010:y=3'b001;
8'b00000100:y=3'b010;
8'b00001000:y=3'b011;
8'b00010000:y=3'b100;
8'b00100000:y=3'b101;
8'b01000000:y=3'b110;
8'b10000000:y=3'b111;
default:y=3'bzzz;
endcase
end
end endmodule
```

Test Bench

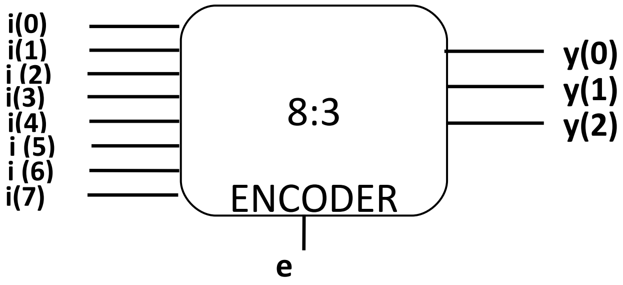
```
module encoder_tb;

reg [0:7] i;
wire [2:0]y;
encoder uut (.i(i),.y(y));
initial begin
#0 e=0;
#100 x=8'b00000001;e=1;
#100 x=8'b00000011;
#100 x=8'b00000110;
#100 x=8'b00001101;
#100 x=8'b00010101;
#100 x=8'b00101011;
#100 x=8'b01011011;
#100 x=8'b10000110;
#10$stop;
end
endmodule
```

8:3 Encoder with Priority

```
module pe8_3(x,y,e);
input e;
input [7:0]x;
output [2:0]y;
reg [2:0]y;

always @(e,x)
begin
if (e==0)
y=3'bZZZ;
else
begin
casex(x)
8'b1XXXXXXX : y = 3'b111;
8'b01XXXXXX : y = 3'b110;
8'b001XXXXX : y = 3'b101;
8'b0001XXXX : y = 3'b100;
```

```
8'b00001XXX : y = 3'b011;
8'b000001XX : y = 3'b010;
8'b0000001X : y = 3'b001;
8'b00000001 : y = 3'b000;
default:y=3'bzzz;
endcase
end
end
endmodule
```

8. BLOCK DIAGRAM:



9. TRUTH TABLE:

8:3 Encoder without Priority.

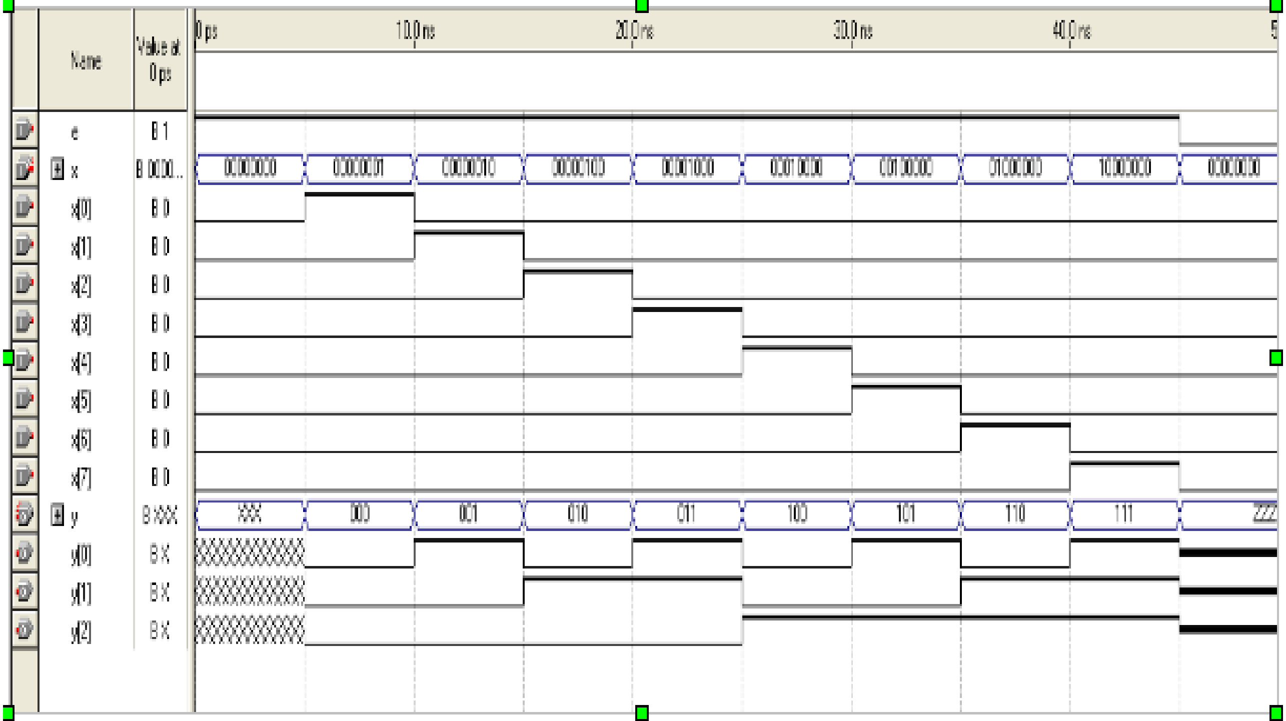
e	Inputs								Outputs		
	i(7)	i(6)	i(5)	i(4)	i(3)	i(2)	i(1)	i(0)	Y(2)	Y(1)	Y(0)
0	X	X	X	X	X	X	X	X	0	0	0
1	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	0	1	0	0	0	1	0
1	0	0	0	0	1	0	0	0	0	1	1
1	0	0	0	1	0	0	0	0	1	0	0
1	0	0	1	0	0	0	0	0	1	0	1
1	0	1	0	0	0	0	0	0	1	1	0
1	1	0	0	0	0	0	0	0	1	1	1

8:3 Encoder with Priority.

e	Inputs								Outputs		
	i(7)	i(6)	i(5)	i(4)	i(3)	i(2)	i(1)	i(0)	Y(2)	Y(1)	Y(0)
0	X	X	X	X	X	X	X	X	0	0	0
1	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	X	0	0	1
1	0	0	0	0	0	1	X	X	0	1	0
1	0	0	0	0	1	X	X	X	0	1	1
1	0	0	0	1	X	X	X	X	1	0	0
1	0	0	1	X	X	X	X	X	1	0	1
1	0	1	X	X	X	X	X	X	1	1	0
1	1	X	X	X	X	X	X	X	1	1	1

10.OUTPUTS:

8:3 Encoder with Priority.



11. RESULTS & CONCLUSIONS:

- 8:3 encoder circuit (With and without priority) is realized using HDL codes.

12. LEARNING OUTCOMES :

- 8:3 encoder circuit (With and without priority) realized and simulated using verilog code.
- 8:3 encoder circuit (With and without priority) synthesized and implemented on FPGA.

13. APPLICATION AREAS:

- Compression (Audio/Video/Images).
- Communication systems.
- Electronic circuits.

14. REMARKS:

- -
- -

-

1. EXPERIMENT NO: 6

2. TITLE: TO REALIZE USING VERILOG BEHAVIORAL DESCRIPTION: 1:8 DEMUX, 3:8 DECODER, 2-BIT COMPARATOR

3. LEARNING OBJECTIVES:

- To learn about the 1:8 demux, 3:8 decoder, 2 bit comparator circuits.
- To simulate,synthesize and implement 1:8 demux, 3:8 decoder, 2 bit comparator using HDL on FPGA.

4. AIM:

- TO REALIZE USING VERILOG 1:8 DEMUX, 3:8 DECODER, 2-BIT COMPARATOR

5. MATERIAL REQUIRED:

- FPGA Board, FRC's jumper, Power Supply

6. THEORY:

- A 1 to 8 demultiplexer consists of one input line, 8 output lines and 3 select lines. Let the input be D, S1 and S2 are two select lines and eight outputs from Y0 to Y7. It is also called as 3 to 8 demux because of the 3 selection lines. Below is the block diagram of 1 to 8 demux.
- The 3 to 8 line decoder is also known as Binary to Octal Decoder. In a 3 to 8 line decoder, there is a total of eight outputs, i.e., Y0, Y1, Y2, Y3, Y4, Y5, Y6, and Y7 and three outputs, i.e., A0, A1, and A2. This circuit has an enable input 'E'. Just like 2 to 4 line decoder, when enable 'E' is set to 1, one of these four outputs will be 1.
- A comparator used to compare two binary numbers each of two bits is called a 2-bit Magnitude comparator. It consists of four inputs and three outputs to generate less than, equal to, and greater than between two binary numbers.

7. PROGRAMME:

VERILOG CODE

```
module
Demultiplexer(in,s0,s1,s2,d0,d1,d2,d3,d4,d5,d6,
d7);
input in,s0,s1,s2;
output d0,d1,d2,d3,d4,d5,d6,d7;
assign d0=(in & ~s2 & ~s1 &~s0),
d1=(in & ~s2 & ~s1 &s0),
d2=(in & ~s2 & s1 &~s0),
d3=(in & ~s2 & s1 &s0),
d4=(in & s2 & ~s1 &~s0),
d5=(in & s2 & ~s1 &s0),
d6=(in & s2 & s1 &~s0),
d7=(in & s2 & s1 &s0);
endmodule
```

TEST BENCH

```
module TestModule;
reg in;
reg s0;
reg s1;
reg s2;
wire d0;
wire d1;
wire d2;
wire d3;
wire d4;
wire d5;
wire d6;
wire d7;
Demultiplexer uut (
.in(in),
.s0(s0),
.s1(s1),
.s2(s2),
.d0(d0),
.d1(d1),
.d2(d2),
.d3(d3),
.d4(d4),
.d5(d5),
.d6(d6),
.d7(d7)
);
initial begin
in = 0;
s0 = 0;
s1 = 0;
s2 = 0;
#100;
```

```
in = 1;
s0 = 0;
s1 = 1;
s2 = 0;
#100;
end
endmodule
```

```
module decoder3_to_8(in,out, en);
input [2:0]  in;
input en;
output [7:0] out;
  reg [7:0] out;

 always @( in or en)
           begin
      if (en)
        begin
          out=8'd0;
          case (in)
              3'b000: out[0]=1'b1;
              3'b001: out[1]=1'b1;
              3'b010: out[2]=1'b1;
              3'b011: out[3]=1'b1;
              3'b100: out[4]=1'b1;
              3'b101: out[5]=1'b1;
              3'b110: out[6]=1'b1;
              3'b111: out[7]=1'b1;
              default: out=8'd0;
          endcase
      end
else
out=8'd0;
end
endmodule
```

```
module decoder_tb;
wire [7:0] out;
reg en;
reg [2:0] in;
integer i;

decoder3_to_8 dut(in,out,en);

initial begin
   for ( i=0; i<16; i=i+1)
        begin
            {en,in}  = i;
             #1;
        end
end
endmodule
```

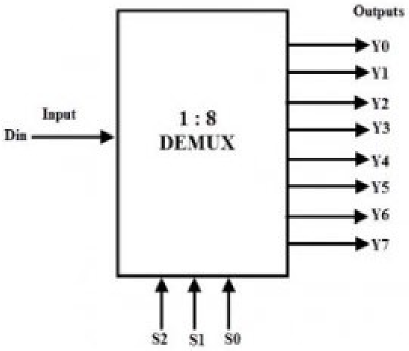
```
// Verilog code for 2-bit comparator
 module comparator(input [1:0] A,B, output
A_less_B, A_equal_B, A_greater_B);
 wire tmp1,tmp2,tmp3,tmp4,tmp5, tmp6, tmp7,
tmp8;
 // A = B output
 xnor u1(tmp1,A[1],B[1]);
 xnor u2(tmp2,A[0],B[0]);
 and u3(A_equal_B,tmp1,tmp2);
 // A less than B output
 assign tmp3 = (~A[0])& (~A[1])& B[0];
 assign tmp4 = (~A[1])& B[1];
 assign tmp5 = (~A[0])& B[1]& B[0];
 assign A_less_B = tmp3 | tmp4 | tmp5;
```

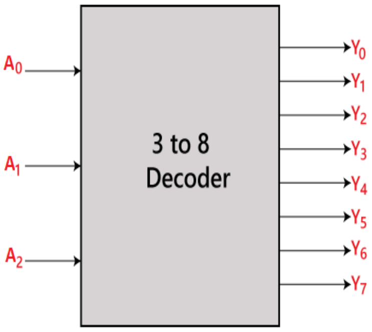
```
module tb_comparator;
 reg [1:0] A, B;
 wire A_less_B, A_equal_B, A_greater_B;
 integer i;
 comparator uut(A,B,A_less_B, A_equal_B,
A_greater_B);
 initial begin
      for (i=0;i<4;i=i+1)
      begin
           A = i;
           B = i + 1;
           #20;
      end
```

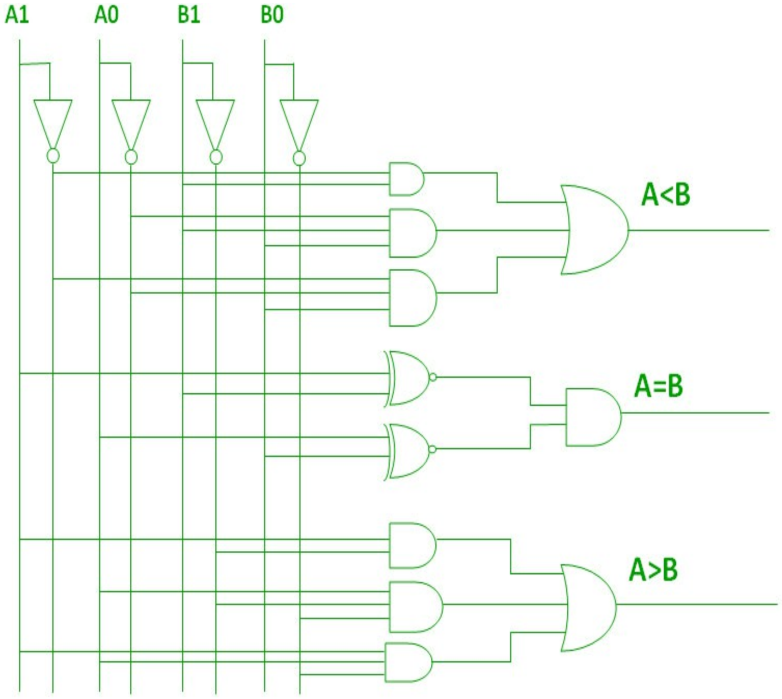
```
// A greater than B output
assign tmp6 = (~B[0])& (~B[1])& A[0];
assign tmp7 = (~B[1])& A[1];
assign tmp8 = (~B[0])& A[1]& A[0];
assign A_greater_B = tmp6 | tmp7 | tmp8;
endmodule
```

```
    for (i=0;i<4;i=i+1)
    begin
        A = i;
        B = i;
        #20;
    end
    for (i=0;i<4;i=i+1)
    begin
        A = i+1;
        B = i;
        #20;
    end
end
endmodule
```

8. BLOCK DIAGRAM:

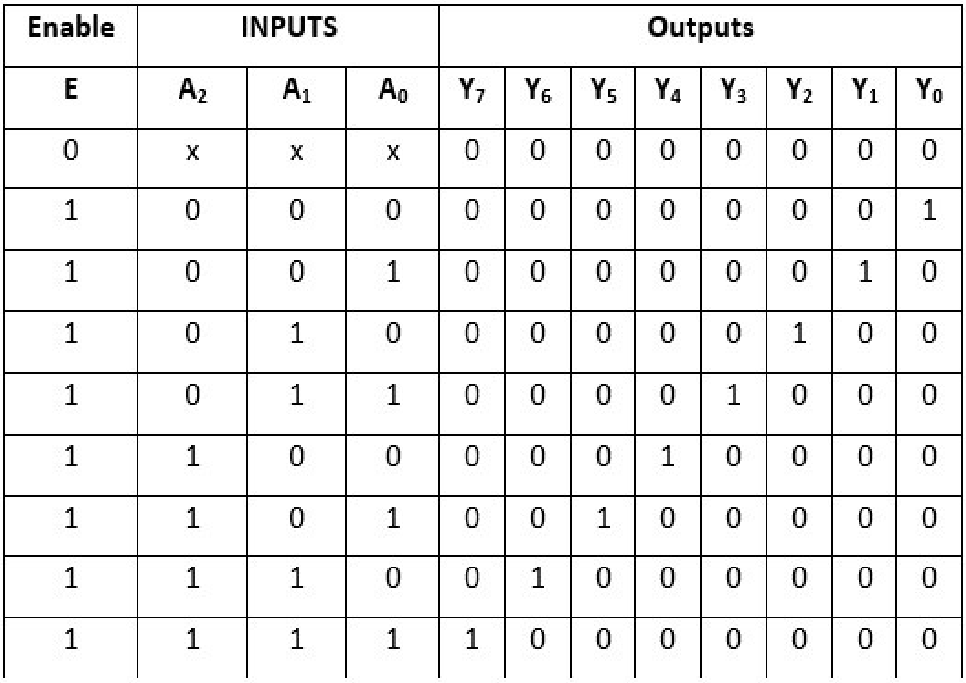




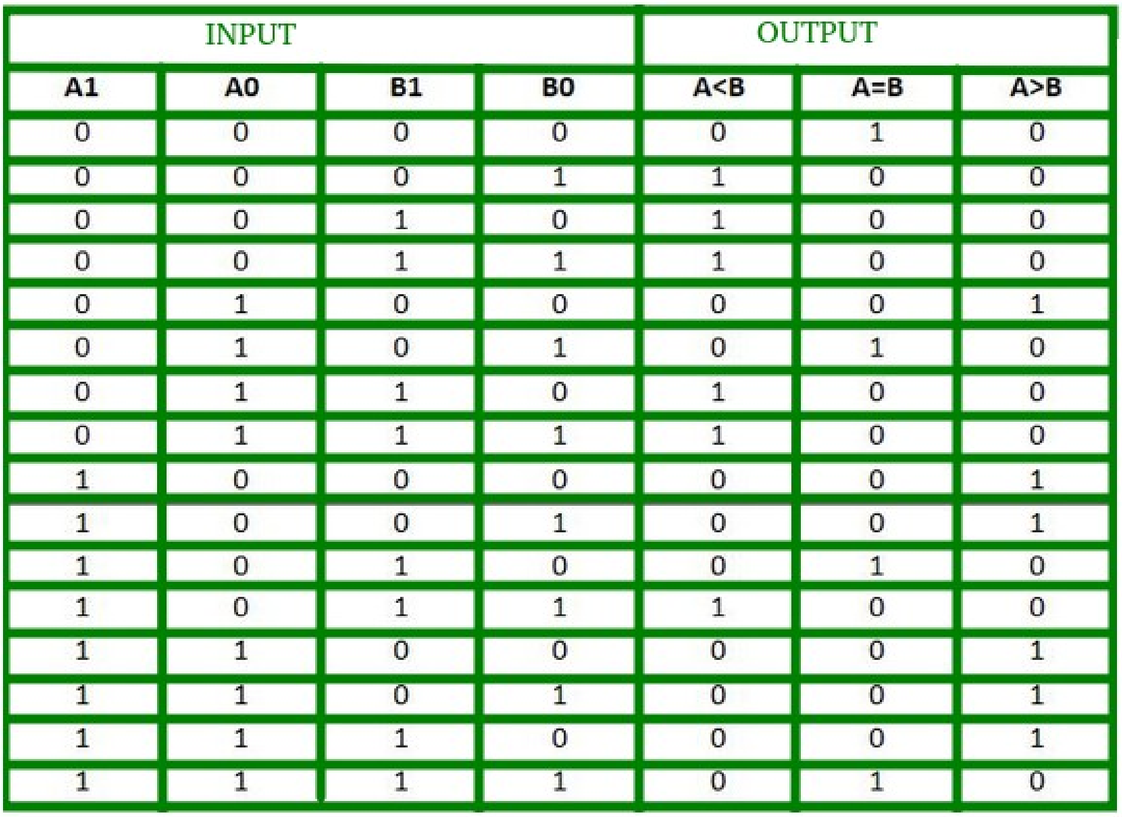


9. TRUTH TABLE:

- 1:8 DEMUX
- 3:8 DECODER
- 2 bit comparator:



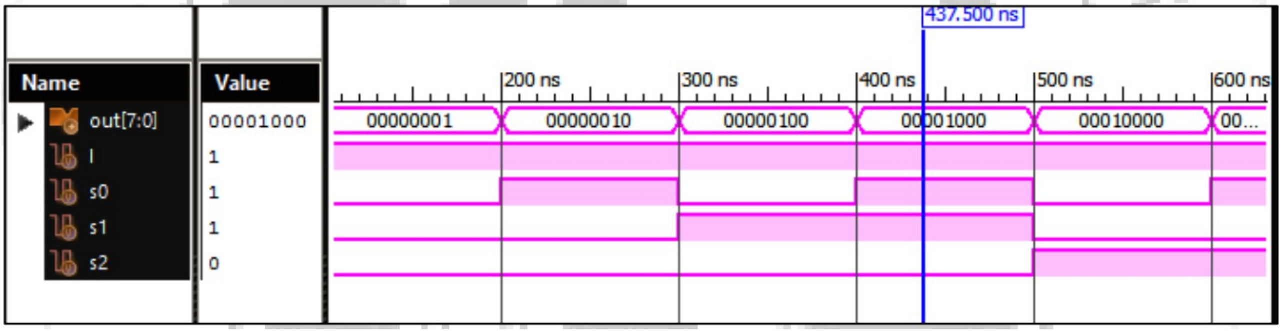
Enable	INPUTS			Outputs							
E	A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0



INPUT				OUTPUT		
A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

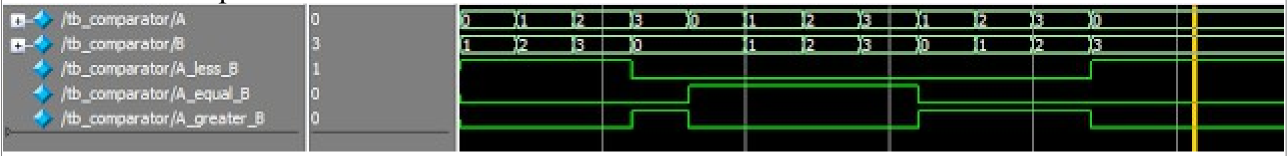
10. OUTPUTS:

- 1:8 DEMUX



Name	Value
out[7:0]	00001000
I	1
s0	1
s1	1
s2	0

- 2 bit comparator



/tb_comparator/A 0
/tb_comparator/B 3
/tb_comparator/A_less_B 1
/tb_comparator/A_equal_B 0
/tb_comparator/A_greater_B 0

11. RESULTS & CONCLUSIONS:

- 1:8 demux, 3:8 decoder and 2 bit comparator are realized using HDL codes.

12. LEARNING OUTCOMES :

- 1:8 demux, 3:8 decoder and 2 bit comparator are realized and simulated using verilog code.
- 1:8 demux, 3:8 decoder and 2 bit comparator are synthesized and implemented on FPGA.

13. APPLICATION AREAS:

- Compression (Audio/Video/Images).
- Electronic circuits.
- Digital communication.
- Video processing.
- Digital broadcasting.

14. REMARKS:

- -
- -

-

1. EXPERIMENT NO: 7

2. TITLE: TO REALIZE USING VERILOG BEHAVIORAL DESCRIPTION: FLIP-FLOPS: A) JK TYPE B) SR TYPE C) T TYPE AND D) D TYPE

3. LEARNING OBJECTIVES:

- To understand the functionality of SR, JK, D and T-flip flop
- To simulate, synthesize and implement SR, JK, D and T-flip flop using HDL on FPGA.

4. AIM:

- To realize using Verilog Behavioral description: Flip-flops: a) JK type b) SR type c) T type and d) D type

5. MATERIAL REQUIRED:

- FPGA Board, FRC's jumper and Power Supply

6. THEORY:

- The basic flip-flop is an asynchronous sequential circuit. The RS flip-flop consists of a basic NOR flip-flop and two AND gates. The outputs of the two NAND gates remain at ‘0’ as long as the clock pulse is ‘0’, regardless of the S and R input values. When the clock pulse goes to ‘1’, the information from the S and R inputs is allowed to reach the basic flip-flop. The set state is reached with with S = 1 and R = 0 and Clock pulse= 1. To change it to clear state the inputs must be S = 0, R = 1 and Clock pulse= 1. When both S = R = 1, the occurrence of the clock pulse causes both outputs to go to 0.
- A JK flip-flop is a refinement of the RS flip-flop in that the indeterminate state of the RS flip-flop is defined in the JK flip-flop. Inputs J and K behave like inputs S and R to set and clear the flip-flop. The JK flip-flop behaves like an RS flip-flop except when both J and K are equal to 1. When both J and K are 1, the clock pulse is transmitted through one AND gate only-the one whose Q=1, the output of upper AND gate becomes 1, upon application of clock pulse, and the flip-flop is cleared. If Q’ is 1, the output of lower AND gate becomes a 1, and the flip-flop is set.
- A flip-flop is a binary cell capable of storing one bit of information. A flip-flop circuit has two outputs, one for the normal value and one for the compliment of bit stored in it. A flip-flop circuit can maintain a binary state indefinitely until directed by an input signal to switch

states. The major difference between various flip-flops is in number of inputs they possess and in the manner in which the inputs affect the binary state. The D flip-flop receives the designation from its ability to transfer data into a flip-flop. It's basically an RS flip-flop with an inverter in the R input. This type of flip-flop sometimes called as D- latch.

- The T flip-flop is also called toggle flip-flop. It is a change of the JK flip-flop. The T flip flop is received by relating both inputs of a JK flip-flop. The T flip-flop is received by relating the inputs 'J' and 'K'. When T = 0, both AND gates are disabled. Therefore, there is no change in the output. When T= 1, the output toggles.

7. PROGRAMME:

VERILOG CODE	TESTBENCH CODE

module sr_ff(sr,clk,q,qb);
input [1:0]sr;
input clk;
output q,qb;
reg q,qb;
always @(posedge(clk))
begin
case(sr)
2'b00:q=q;
2'b01:q=1'b0;
2'b10:q=1'b1;
2'b11:q=1'bx;
endcase
qb= ~q;
end
endmodule
	module srfftest;
	reg [1:0] sr;
	reg clk;
	wire q;
	wire qb;
	sr_ff uut (.sr(sr), .clk(clk), .q(q), .qb(qb));
	initial
	clk=0;
	always #50 clk=~clk;
	initial
	begin
	sr=01;#100;
	sr=10;#100;
	sr=00;#100;
	sr=11;#100;
	#100 $stop;
	end
	endmodule

module jk_ff(jk,clk,q,qb);
input [1:0]jk;
input clk;
output q,qb;
reg q,qb;
always @(posedge(clk))
begin
case(jk)
2'b00:q=q;
2'b01:q=1'b0;
2'b10:q=1'b1;
2'b11:q=~q;
endcase
qb= ~q;
end
endmodule
	module jkfftest;
	reg [1:0] jk;
	reg clk;
	wire q;
	wire qb;
	jk_ff uut (.jk(jk), .clk(clk), .q(q), .qb(qb));
	initial clk=0;
	always #50 clk=~clk;
	initial begin
	jk=00;#100;
	jk=01;#100;
	jk=10;#100;
	jk=11;#100;
	#100 $stop;
	end
	endmodule

module d_ff(d,clk,q,qb);
input d,clk;
output q,qb;
reg q,qb;
	module dfftest;
	reg d;
	reg clk;
	wire q;

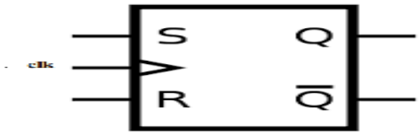
```
always@(posedge (clk))
begin
if (d==1'b1)
q=d;
else if(d==1'b0)
q=d;
qb=~q;
end
endmodule
```

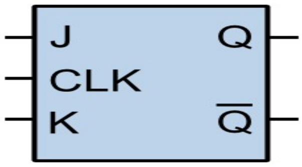
```
wire qb;
d_ff uut (.d(d), .clk(clk), .q(q), .qb(qb));
 initial
clk = 0;
always #50 clk=~clk;
 initial begin
 d=0;    #100;
 d=1; #100;
d=0; #100;
# 100 $stop;
end
endmodule
```

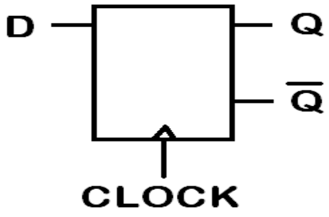
```
module t_ff ( t, clk, q, q_bar);
input t, clk;
output q, q_bar;
reg q, q_bar;
always @(posedge clk)
begin
q = ~t;
q_bar = ~q;
end
endmodule
```

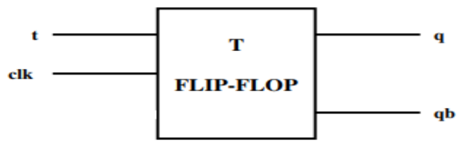
```
module tflipflop;
reg t,clk;
wire q, q_bar;
t_ff uut ( .t(t), .clk(clk), .q(q), .q_bar(q_bar) );
initial
clk = 0;
always #50 clk=~clk;
initial begin
t = 0;clk = 0; #100;
t = 0;clk = 1; #100;
t = 1;clk = 1; #100
t = 1;clk = 0; #100;
end
end module
```

8. BLOCK DIAGRAM:









9. TRUTH TABLE:

D-FLIP FLOP:

clk	D	q	q_bar
↑	0	0	1
↑	1	1	0
0 or 1	x	q	q_bar

SR -FLIP FLOP:

clk	s	r	q	qb
0 or 1	x	x	q	qb
↑	0	0	q	qb
↑	0	1	0	1
↑	1	0	1	0
↑	1	1	z	z

JK-FLIP FLOP:

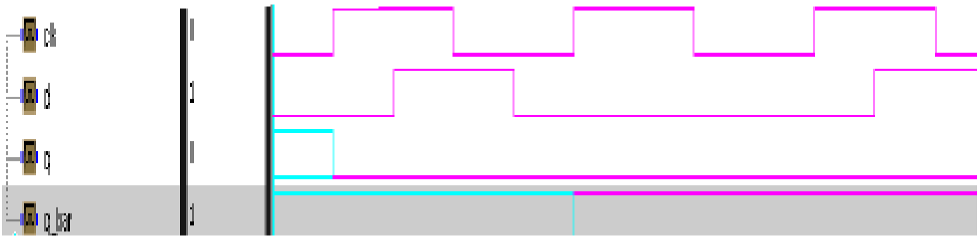
clk	j	k	q	qb
0 or 1	x	x	q	qb
↑	0	0	q	qb
↑	0	1	0	1
↑	1	0	1	0
↑	1	1	qb	q

T-FLIP FLOP:

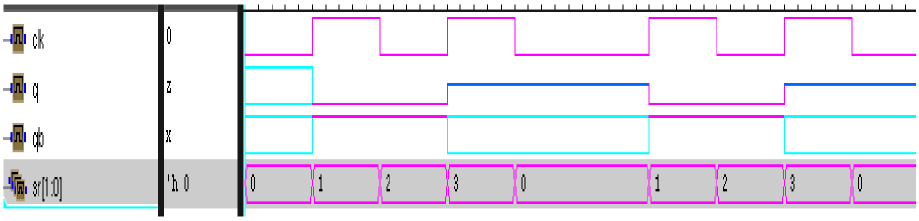
clk	T	q	q_bar
↑	0	q	q_bar
↑	1	$\bar{q}$	
0 or 1	x	q	q_bar

10. OUTPUTS:

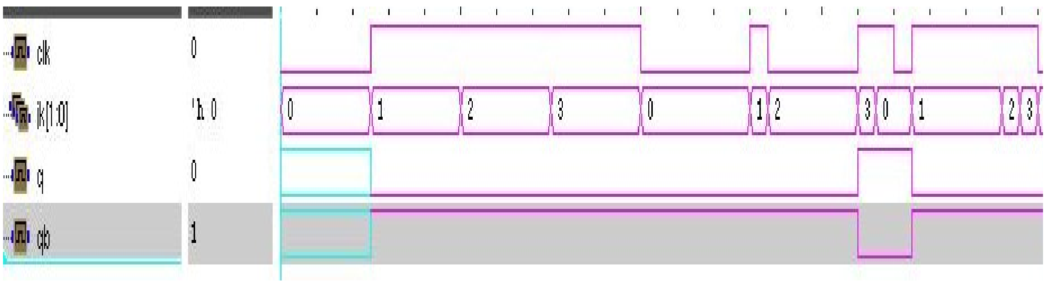
D-FLIP FLOP:



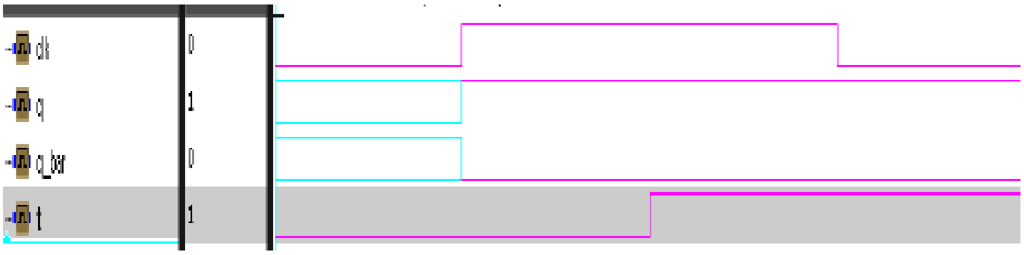
SR-FLIP FLOP:



JK FLIP-FLOP:



T FLIP-FLOP:



11. RESULTS & CONCLUSIONS:

- The waveforms for D,SR,JK and T flip flops are generated and truth tables are verified

13. LEARNING OUTCOMES :

- Verilog code for D,SR,JK and T Flip-flops are written and outputs are verifiied

14. APPLICATION AREAS:

- Computers
- Communication Systems
- Counter Design
- Registers

15. REMARKS:

- -
- -

-

1. EXPERIMENT NO: 8

2. TITLE: TO REALIZE COUNTERS - UP/DOWN (BCD AND BINARY) USING VERILOG BEHAVIORAL DESCRIPTION.

3. LEARNING OBJECTIVES:

- Write a verilog code for synchronous/Asynchronous counters and their test bench for verification .Observe the waveform.

4. AIM:

- To Compile and simulate the Verilog Code for synchronous/Asynchronous counters . Verfiy the waveform and synthesize the code with the technological library.

5. MATERIAL REQUIRED:

- FPGA Board, FRC's jumper, Power Supply

6. THEORY :

SYNCHRONOUS COUNTERS:

In synchronous counters, the clock inputs of all the flip-flops are connected together and are triggered by the input pulses. Thus, all the flip-flops state changes simultaneously (in parallel).Synchronous Counters uses edge-triggered flip-flops that change states on either the “positive-edge” (rising edge) or the “negative-edge” (falling edge) of the clock pulse on the control input resulting in one single count when the clock input changes state. A synchronous counter however, has an internal clock, and the external event is used to produce a pulse which is synchronised with this internal clock .

ASYNCHRONOUS COUNTERS:

In asynchronous counters, the clock inputs of all the flip-flops are not connected together and are triggered by the input pulses. Asynchronous reset counter depends on the reset. When reset is high irrespective value of the clock, output goes to zero. On every clock pulse the resulting outputs count upwards from 0 (0000) to 15 (1111). Therefore, this type of counter is also known as a 4-bit Asynchronous Up Counter .

Asynchronous Counters use edge-triggered flip-flops that change states on either the “positive-edge” (rising edge) or the “negative-edge” (falling edge) of the clock pulse on the control input resulting in one single count when the clock input changes state.

•

7. PROGRAM:

VERILOG CODE	TESTBENCH CODE

module synbcdup(Q,clk,rst);
input clk ,rst;
output [3:0] Q;
reg [3:0] Q;
initial begin
Q=4'b0000;
end
always@(posedge clk)
begin
if(rst==1)
Q=4'd0;
else if (rst==0)
Q=Q+1;
end
endmodule
	module syncnttest;
	reg clk, rst;
	wire[3:0] Q;
	synbcdup uut (.Q(Q),.clk(clk),.rst(rst));
	initial begin
	clk=0;
	forever#50
	clk=~clk;
	end
	initial begin
	rst=0;#200;
	rst=1; #100;
	rst=0;#1500;
	rst=1;
	end
	endmodule

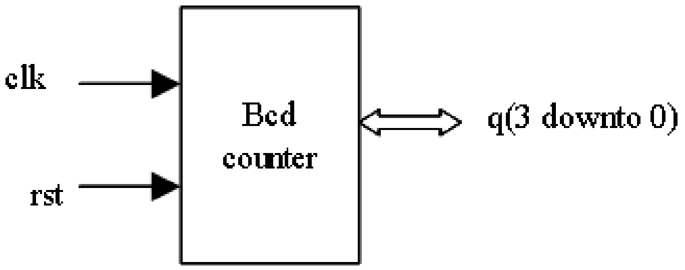
module syncntdwn(Q,clk,rst);
input clk, rst;
output [3:0] Q;
reg [3:0] Q;
initial begin
Q=4'd15;
end
always@(posedge clk)
begin
if(rst==1)
Q=4'd0;
else if(rst==0)
Q=Q - 1;
end
endmodule
	module syndwntest;
	reg clk, rst;
	wire[3:0] Q;
	syncntdwn uut (.Q(Q),.clk(clk),.rst(rst));
	initial begin
	clk=0;
	forever#50
	clk=~clk;
	end
	initial begin
	rst=0;#200;
	rst=1;#100;
	rst=0;#1500;
	rst=1;
	end
	endmodule

module asynbinup(Q,clk,rst);
input clk, rst;
output [3:0] Q;
reg [3:0] Q;
initial begin
Q=4'd0;
end
always@(posedge clk or posedge rst)
begin
if(rst)
	module asyncnttest;
	reg clk, rst;
	wire[3:0] Q;
	asynbinup uut (.Q(Q),.clk(clk),.rst(rst));
	initial begin
	clk=0;
	forever#50
	clk=~clk;
	end
	initial begin

Q=4'd0;
else
Q=Q+1;
end
endmodule
	rst=0;#200;
	rst=1; #100;
	rst=0;#1500;
	rst=1;
	end
	endmodule

module asyncntdwn(Q,clk,rst);
input clk rst;
output [3:0] Q;
reg [3:0] Q;
initial begin
Q=4'd0;
end
always@(posedge clk or posedge rst)
begin
if(rst)
Q=4'd0;
else
Q=Q - 1;
end
endmodule
	module asyndwntest;
	reg clk, rst;
	wire[3:0] Q;
	asyncntdwn uut (.Q(Q),.clk(clk),.rst(rst));
	initial begin
	clk=0;
	forever#50
	clk=~clk;
	end
	initial begin
	rst=0;#200;
	rst=1;#100;
	rst=0;#1500;
	rst=1;
	end
	endmodule

8. BLOCK DIAGRAM:



9. TRUTH TABLE:

SYNCHRONOUS RESET BINARY UP COUNTER:

Clk	Rst	Q
↑	0	0000
↑	0	0001
↑	0	0010
↑	0	0011
↑	0	0100
↑	0	0101
↑	0	0110
↑	0	0111
↑	0	1000
↑	0	1001
↑	0	1010
↑	0	1011
↑	0	1100
↑	0	1101
↑	0	1110
↑	0	1111
X	1	0000

SYNCHRONOUS RESET BINARY DOWN COUNTER:

Clk	Rst	Q
↑	0	1111
↑	0	1110
↑	0	1101
↑	0	1100
↑	0	1011
↑	0	1010
↑	0	1001
↑	0	1000
↑	0	0111
↑	0	0110
↑	0	0101

↑	0	0100
↑	0	0011
↑	0	0010
↑	0	0001
↑	0	0000
X	1	0000

ASYNCHRONOUS RESET BINARY UP COUNTER

Clk	Rst	Q
↑	0	0000
↑	0	0001
↑	0	0010
↑	0	0011
↑	0	0100
↑	0	0101
↑	0	0110
↑	0	0111
↑	0	1000
↑	0	1001
↑	0	1010
↑	0	1011
↑	0	1100
↑	0	1101
↑	0	1110
↑	0	1111
X	1	0000

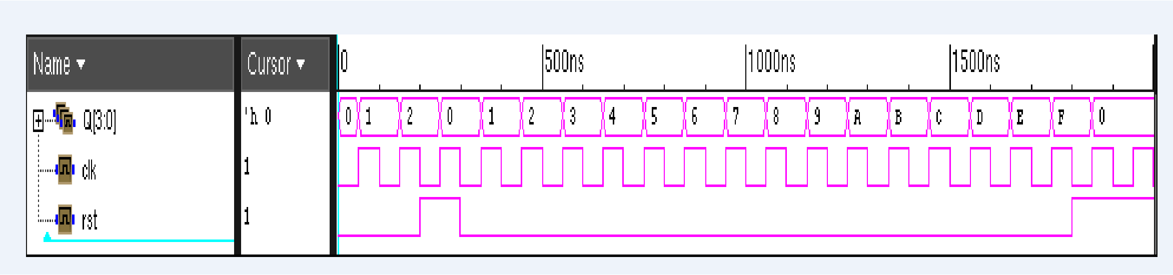
ASYNCHRONOUS RESET BINARY DOWN COUNTER:

Clk	Rst	Q
↑	0	1111
↑	0	1110
↑	0	1101
↑	0	1100
↑	0	1011
↑	0	1010

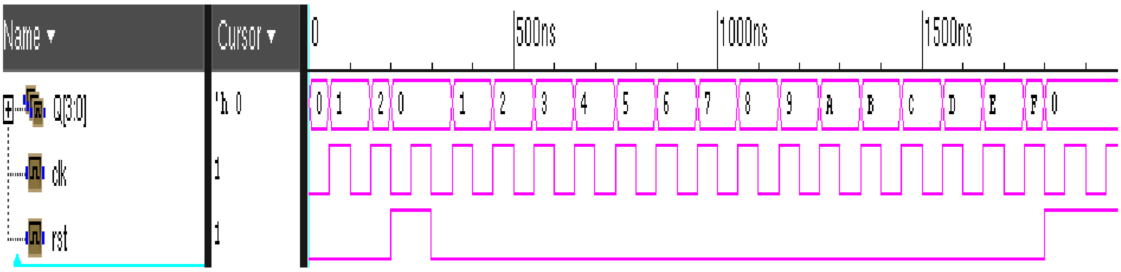
↑	0	1001
↑	0	1000
↑	0	0111
↑	0	0110
↑	0	0101
↑	0	0100
↑	0	0011
↑	0	0010
↑	0	0001
↑	0	0000
X	1	0000

10. OUTPUTS:

SYNCHRONOUS RESET BINARY UP COUNTER:



ASYNCHRONOUS RESET BINARY UP COUNTER:



11. RESULTS & CONCLUSIONS:

- The waveforms for synchronous and asynchronous counters are generated and the truth tables are verified

12. LEARNING OUTCOMES :

- Verilog code for synchronous and asynchronous counters are written and their outputs are verifiied

13. APPLICATION AREAS:

- Digital clocks/Alarms
- Microwave ovens to set time
- Frequency divider circuits

14. REMARKS:

- -
- -

-

1. EXPERIMENT NO: 9

2. TITLE: VERILOG PROGRAM TO INTERFACE A STEPPER MOTOR TO THE FPGA/CPLD AND ROTATE THE MOTOR IN THE SPECIFIED DIRECTION (BY N STEPS).

3. LEARNING OBJECTIVES:

- To understand the concept of Stepper motor.
- To realize and interface Stepper motor with FPGA.

4. AIM:

- Interface a Stepper motor to FPGA and write Verilog code to control the Stepper motor rotation which in turn may control a Robotic Arm. External switches to be used for different controls like rotate the Stepper motor

(i) +N steps if Switch no.1 of a Dip switch is closed

(ii) +N/2 steps if Switch no. 2 of a Dip switch is closed

(iii) –N steps if Switch no. 3 of a Dip switch is closed etc.

5. MATERIAL REQUIRED:

- FPGA Board, FRC's Jumper, Power supply

6. THEORY:

- **Stepper motor**

The basic principle of stepper motor or any other motor is electromagnetic induction which means while apply a current to the circuit, the EMF will be induced due to change of magnetic field of the circuit.

- Working of stepper motor

The rotor shaft of the stepper motor is surrounded by the electromagnetic stator. The rotor and stator have poles which would be teethed or depends upon the motor type. When apply the electrical pulse to the stator which gets energized and produced EMF. The EMF cuts the conductor of the rotor. Now the torque will be introduced. This torque (force) is used to rotate the rotor shaft which means the rotor align itself along with stator.

A full rotation of stepper is divided into number of steps which would be around 200 steps. The stepper receives only one pulse at the time for one step. Normally the rotating angle of stepper motor typically is 1.8 degrees.

The stepper motor operates at various modes such as full step, half step.

- Full step

The full step of stepper motor is 200 steps per revolution. So the angle of rotation is 1.8 degrees which is calculating from given formula.

Step angle = 360 degrees / no of steps.

Step angle = 360 / 200 = 1.8 degrees.

The motor is used to covert mechanical energy into electrical energy. Stepper motor is not like as any other motor which means is not rotating continuously. It is rotating step by step according to given electrical pulse. The FPGA cannot drive stepper motor directly. Because of the I/O capability of FPGA is 3.3v. So for the driver circuit are required able to drive the stepper motor. A full rotation divides into a number of equal steps. So this motor called as also stepper motor.

7. PROCEDURE / PROGRAMME:

STEPPER MOTOR

PROCEDURE:

- Make the connection between FRC9 of the FPGA board to the stepper motor connector of the VTU card2.
- Make the connection between FRC1 of the FPGA board to the Dip switch connector of the VTU card2.
- Connect the downloading cable and power supply to the FPGA board.
- Then open the Xilinx iMPACT software (refer ISE flow) select the slave serial mode and select the respetive BIT file and click program.
- Make the reset switch on (active low).
- Press the HEX keys and analyze the data.

VERILOG CODE:

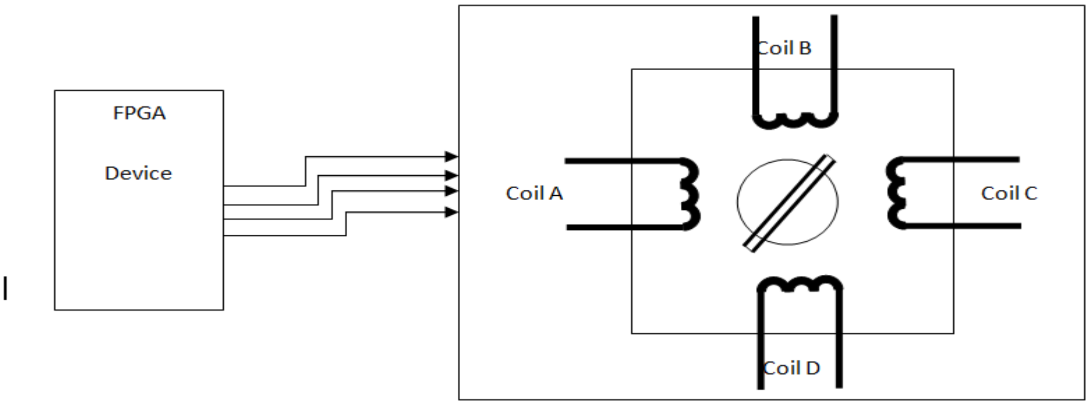
```
module stepper(clk,reset, dipsw1, dipsw2, dipsw3, dout);
input clk, reset,  dipsw1,  dipsw2,  dipsw3;
output reg[3:0]dout;
reg [0:35]divclk;
wire intclk;
reg [3:0]register;
integer n=200;
integer i=0;
always@(posedge clk)
begin
divclk=divclk+1;
end
assign intclk=divclk[16];
always@(posedge intclk)
begin
if(reset)
begin
register<=4'b1001;
i=0;
end
else
begin
if(dipsw1)
begin
if(i<n)
begin
register<={register[2:0],register[3]};
dout=register;
i=i+1;
end
```

```
end
else if(dipsw2)
begin
if(i<n/2)
begin
register<={register[2:0],register[3]};
dout=register;
i=i+1;
end
end
else if(dipsw3)
begin
if(i<n)
begin
register<={register[0],register[3:1]};
dout=register;
i=i+1;
end
end
end
end
endmodule
```

UCF file(User constraint File)

STEPPER MOTOR (KIT CONECTION)	
FRC1	DIP SWITCH
FRC7	STEPPER MOTOR KIT
PIN CONNECTION	
CLK	P52
RESET	P74
dipsw1	P76
dipsw2	P77
dipsw3	P79
dout[0]	P5
dout[1]	P4
dout[2]	P2
dout[3]	P141

8. MODEL DIAGRAM :



9. RESULTS & CONCLUSIONS:

- Stepper motor experiments is realized using verilog and interfaced with FPGA.

10. LEARNING OUTCOMES :

- Stepper motor is realized using HDL.
- Stepper motor is interfaced with FPGA.

11. APPLICATION AREAS:

- Floppy disk drivers.
- Computer printers, Image scanners.
- Cranes, elevators, etc.

12. REMARKS:

- -
- -

-

1. EXPERIMENT NO: 10

2. TITLE: VERILOG PROGRAMS TO INTERFACE SWITCHES AND LEDS TO THE FPGA/CPLD AND DEMONSTRATE ITS WORKING.

3. LEARNING OBJECTIVES:

- To understand the concept of 7 seven segment LED.
- To realize how seven segment LED display can be interfaced and integrated with FPGA.

4. AIM:

- To display the message on seven segment LED display by accepting HEX key pad input data.

5. MATERIAL REQUIRED:

- FPGA Board, FRC's Jumper, Power supply, HEX key pad.

6. THEORY:

- 7-Segment display can display the digits 0-9 and the hex extension (A-F).
- Seven Segment display are of 2 types:
  a. Common Anode type – ‘0’ indicates segment ON ,‘1’ indicates segment OFF
  b. Common Anode type – ‘1’ indicates segment ON, ‘0’ indicates segment OFF

7. PROCEDURE / PROGRAMME:

- Make the connection between FRC5 of the FPGA board to the seven segment connector of the VTU card1.

- Make the connection between FRC4 of the FPGA board to the key board connector of the VTU card1.
- Make the connection between FRC6 of the FPGA board to the Dip switch connector of the VTU card1.
- Connect the downloading cable and power supply to the FPGA board.
- Then open the Xilinx iMPACT software (refer ISE flow) select the slave serial mode and select the respetive BIT file and click program.
- Press the HEX keys and analyze the data.

Verilog Program:

```
module key(clk, read, scan, dispn, disp); //SCAN= ROW & READ=COLUMN
input clk;
input [3:0]read;
output reg [3:0]scan=4'b0000;
output [3:0]dispn;
output reg [6:0]disp;
reg [1:0]count=2'b00;
always@(posedge clk)
begin
count<=count+1'b1;
end a
ssign dispn=4'b0111;
always@(count)
begin
case(count)
2'b00:scan=4'b1000; //Row3
2'b01:scan=4'b0100; //Row2
2'b10:scan=4'b0010; //Row1
2'b11:scan=4'b0001; //Row0
default:scan=4'b0000;
endcase
end
always@(scan,read)
begin
case(scan)
4'b0001: case(read)
4'b0001:disp=7'b1111110; // 0
4'b0010:disp=7'b0110011; // 4
4'b0100:disp=7'b1111111; //8
4'b1000:disp=7'b1001110; //c
default:disp=7'b0000000;
endcase
4'b0010: case(read)
4'b0001:disp=7'b0110000;    //1
4'b0010:disp=7'b1011011;    //5
4'b0100:disp=7'b1111011;   //9
4'b1000:disp=7'b0111101; // D
default:disp=7'b0000000;
```

```
endcase 4'b0100:
case(read)
4'b0001:disp=7'b1101101; // 2
4'b0010:disp=7'b1011111;//6
4'b0100:disp=7'b1110111;//A
4'b1000:disp=7'b1001111; //E
default:disp=7'b0000000; endcase
4'b1000: case(read)
4'b0001:disp=7'b1111001;//3
4'b0010:disp=7'b1110000;//7
4'b0100:disp=7'b0011111;//B
4'b1000:disp=7'b1000111; //F
default:disp=7'b0000000;
endcase
default:disp=7'b0000000;
endcase
end
endmodule
```

8. Truth Table:

Segment Values							Segment output
g	f	e	d	c	b	a	
0	1	1	1	1	1	1	0
0	0	0	0	1	1	0	1
1	0	1	1	0	1	1	2
1	0	0	1	1	1	1	3
1	1	0	0	1	1	0	4
1	1	0	1	1	0	1	5
1	1	1	1	1	0	1	6
0	0	0	0	1	1	1	7
1	1	1	1	1	1	1	8
1	1	0	1	1	1	1	9
1	1	1	0	1	1	1	A
1	1	1	1	1	0	0	b
0	1	1	1	0	0	1	C
1	0	1	1	1	1	0	d
1	1	1	1	0	1	1	E
1	1	1	0	0	0	1	F

9. RESULTS & CONCLUSIONS:

- A message displayed on seven segment LED display by accepting HEX key pad input data.

10. LEARNING OUTCOMES :

- 7 seven segment LED display experiment is realized using verilog.
- Seven segment LED display is interfaced and integrated with FPGA.

11. APPLICATION AREAS:

- Digital clocks.
- Electronic meters.
- Basic calculators.

12. REMARKS:

- -
- -

H. INTERNAL ASSESSMENT TEST MODEL QUESTION PAPER

Dept: ECE	Sem / Div: III / A	Course: Digital System Design Using Verilog	Course Code: BEC302
Date: DD/MM/YY	Time: 90 Min.	Max Marks: 50	Elective: N
Note: Answer any 2 FULL questions.			

QN		Questions	Marks	RBT	CO
PART A					
1	a	Design a logic circuit that has 4 inputs, the output will be high when the majoriety of the inputs are high. Use K-map to simplify.	9	L3	CO1
	b	Define combinational circuit and place the following equation in to the proper canonical form. G = f(w, x, y, z) = w'x + yz' T = f(a, b, c) = (a + b') (b' + c)	8	L2	CO1
	c	Design 2-bit magnitude comparator and implement using basic gates	8	L3	CO2
OR					
2	a	Simplify the following using Quine-McClusky Minimization Technique D = f(a, b, c, d) = ∑(0, 1, 2, 3, 6, 7, 8, 9, 14, 15)	9	L3	CO1
	b	Define the following 1) Literal 2) Sum of Products 3) Product of sums 4) Minterm 5) Maxterm. 6) Canonical Sum of Products 7) Canonical Product of Sums	8	L2	CO1
	c	Simplify using K-map B = f(w, x, y, z) = ∑(1, 2, 3, 4, 9) + ∑d(10, 11, 12, 13, 14, 15)	8	L3	CO1
PART B					
3	a	Simplify using K-map y = f(a, b, c, d) = Σ(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)	8	L3	CO1
	b	Identify the Prime implicants and Essential Prime implicants of the boolean function using K-map f(a, b, c) = Σ(0, 2, 3, 7) and get the simplified expression.	8	L3	CO1,2
	c	Draw the keypad interfacing diagram to a digital system using 10 line decimal to BCD encoder and explain.	9	L2	CO2
OR					

4	a	Implement the following multiple output function using 74LS138 and external gates.
		F1(A, B, C) = ∑m(1, 4, 5, 7)
		F2(A, B, C) = πM(2, 3, 6, 7)
			8	L3	CO2

	b	Define decoder. Implement the function F = ∑m(1, 2, 3, 7) using 3:8 decoder with active high outputs.	8	L3	CO1,2
	c	Implement full adder using two half adder and an OR gate.	9	L3	CO2

COURSE PLAN

G. CONTINUOUS INTERNAL EVALUATION

Evaluation	Weightage in Marks
IA Test – 1	15
IA Test – 2	15
Other Marks	10
Lab Component Marks	25
Final CIE	15+10+25 = 50