BANGALORE INSTITUTE OF TECHNOLOGY

K.R.ROAD, V.V.PURAM Bangalore-560004

DEPARTMENT OF ECE

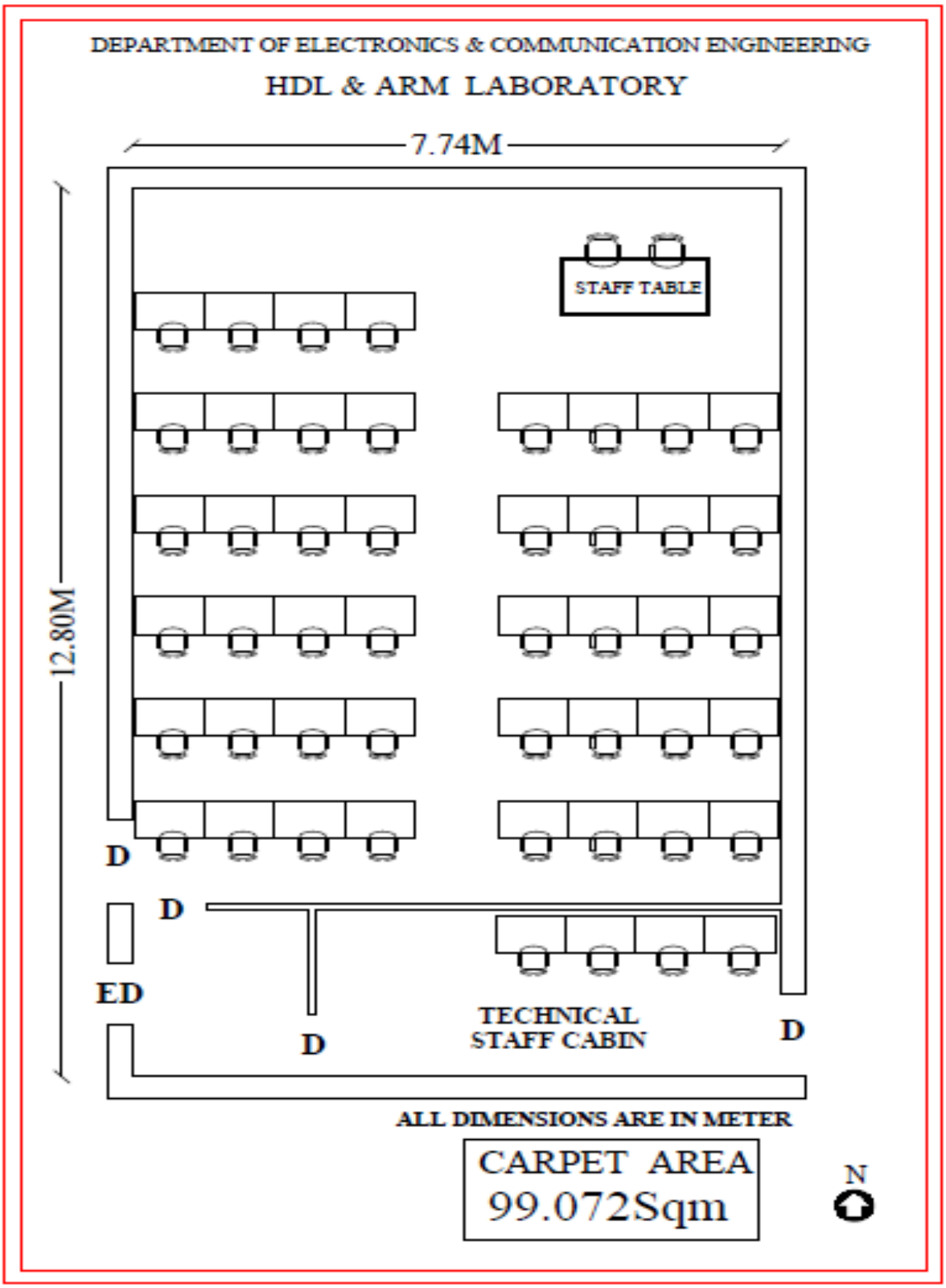
Name of the Laboratory	**:**	**HDL LAB**
Semester / Year	**:**	**V / 2020**
No. of Students/Batch	**:**	**20**
No. of Equipment's	**:**	**23 (SPARTAN-3 KIT)**
Major Equipment's	**:**	**SPARTAN-3 TRAINER KIT,** **Softwares:XILINX-ISE 9.2i &Modelsim6.3f,** **Digital Storage Oscilloscope,** **DELL Computers &** **HP Workstations.**
Area in square meters	**:**	**99.072 Sqm**
Location	**:**	**4th Floor**
Total Cost of Lab	**:**	**Course Handling**
	:	**Dr. VIJAYA PRAKASH**
	:	**Dr. KALPANA A.B**
		THEJASWINI.B.M,
Instructors	**:**	**VASUDEVA.B**
Mechanic	**:**	**SUHAS.R**
HOD	**:**	**Dr.K.V.Prasad**

DO'S AND DONT'S

Students should follow the below DO's & DONT's:

1. Follow the schedule time, late comers will not be permitted.
2. Sign the Log book available in the Lab.
3. Leave the Foot wears in the specified stand before entering Lab.
4. Keep belongings in the specified place.
5. Students are expected to come prepared for experiments &VIVA.
6. Show the completed Observation Book and submit Record to the Teacher before the Lab session begins.
7. Cycle of experiments should be followed.
8. Follow all the safety measures as suggested by the Teacher/Lab Instructor.
9. Observe the instructions given by the Teacher/ Lab Instructor and strictly follow accordingly.
10. Report to the Teacher/Lab Instructor immediately in case of any software/hardware/ Electrical failure during working. Never try to fix it manually/individually.
11. Computers and any other experimental Kits should be switched OFF and chairs should be repositioned before leaving the Lab.
12. Get the observations verified/signed by the teacher before leaving the Lab.
13. Maintain Discipline & tidiness inside the Lab. Attend the Lab in formal attire.
14. Usage of Mobile Phones, Pen Drive and Electronics Gadgets are restricted during the Lab session.

LAB LAYOUT



SYLLABUS

[As per Choice Based Credit System (CBCS) scheme]			
Laboratory Code	**18ECL58**	**CIE Marks**	**40**
Number of Lecture Hours/Week	**02 Hr Tutorial (Instructions)+ 02 Hours Laboratory**	**SEE Marks**	**60**
RBT Level	**L1, L2, L3**	**Exam Hours**	**03**
CREDITS – 02			

Course Objectives: This course will enable students to:

- Familiarize with the CAD tool to write HDL programs.
- Understand simulation and synthesis of digital design.
- Program FPGAs/CPLDs to synthesize the digital designs.
- Interface hardware to programmable ICs through I/O ports.
- Choose either Verilog or VHDL for a given Abstraction level.

Note: Programming can be done using any compiler. Download the programs on a FPGA/CPLD board and performance testing may be done using 32 channel pattern generator and logic analyzer apart from verification by simulation with tools such as Altera/Model sim or equivalent.

Laboratory Experiments

PART A :Programming

1. WriteVerilogprogramforthefollowingcombinationaldesignalongwithtest bench to verify the design:
   a. 2 to 4 decoder realization using NAND gates only (structural model)
   b. 8 to 3 encoder with priority and without priority (behavioural model)
   c. 8 to 1 multiplexer using case statement and if statements
   d. 4-bitbinarytograyconverterusing1-bitgraytobinaryconverter1-bit adder and subtractor

2. Model in Verilog for a full adder and add functionality to perform logical operations of XOR, XNOR, AND and OR gates. Write test bench with appropriate input patterns to verify the modeled behaviour.

3. Verilog32-bitALUshowninfigurebelowandverifythefunctionalityof ALUby selecting appropriate test patterns. The functionality of the ALU is presented in Table1.
   a. Write test bench to verify the functionality of the ALU considering allpossible input patterns
   b. The enable signal will set the output to required functions if enabled, if disabled all the outputs are set totri-state
   c. The acknowledge signal is set high after every operation iscompleted

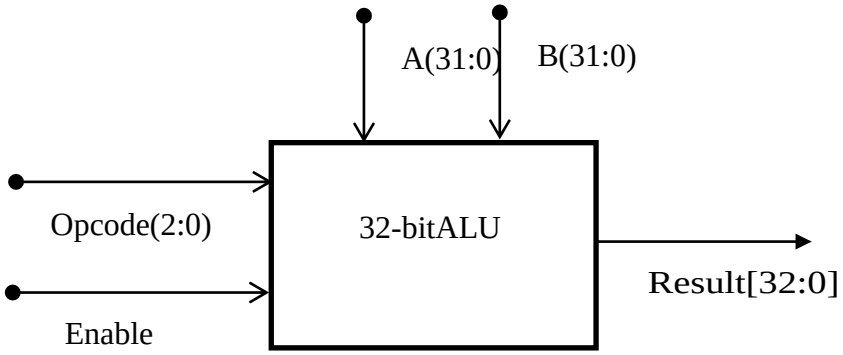


Figure1ALUtoplevelblockdiagram Table 1 ALU Functions

Opcode (2:0)	ALU Operation	Remarks	
000	A + B	Addition of two numbers	Both A and B are in two'scomplement format
001	A – B	Subtraction of two numbers	
010	A + 1	Increment Accumulator by 1	A is in two's complement format
011	A - 1	Decrement accumulator by 1	
100	A	True	Inputs can be in any format
101	A Complement	Complement	
110	A OR B	Logical OR	
111	A AND B	Logical AND	

4. Write Verilog code for SR, D and JK and verify the flip flop.

5. Write Verilog code for 4-bit BCD synchronous counter.

6. Write Verilog code for counter with given input clock and check whether itworks as clock divider performing division of clock by 2, 4, 8 and 16. Verify the functionality of the code.

PART-B:Interfacing and Debugging(ED Win XP, P-Spice, Multi-Sim, Proteus, Circuit Lab or any other equivalent tool can be used)

1. Write a Verilog code to design a clock divider circuit that generates 1/2, 1/3rdand 1/4thclock from a given input clock. Port the design to FPGA and validate the functionality through oscilloscope.

2. Interface a DC motor to FPGA and write Verilog code to change its speed and direction.

3. 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.

4. Interface a DAC to FPGA and write Verilog code to generate Sine wave of frequency FKHz(eg.200KHz)frequency.Modify the code to down sample the frequency to F/2 KHz. Display the Original and Down sampled signals by connecting them to an oscilloscope.

5. Write Verilog code using FSM to simulate elevator operation.

6. Write Verilog code to convert an analog input of a sensor to digital form and to display the same on a suitable display like set of simple LEDs, 7-segment display digits or LCD display.

Course Outcomes: At the end of this course, students should be able to:

- Write the Verilog/VHDL programs to simulate Combinational circuits in Dataflow, Behavioral and Gate level Abstractions.
- Describe sequential circuits like flip flop sand counters in Behavioral description and obtain simulation waveforms.
- Synthesize Combinational and Sequential circuits on programmable ICs and test the hardware.
- Interface the hardware to the programmable chips and obtain the required output.

Conduct of Practical Examination:

- All laboratory experiments are to be included for practical examination.
- Students are allowed to pick one experiment from the lot.
- Strictly follow the instructions as printed on the cover page of answer script for breakup of marks.
- Change of experiment is allowed only once and Marks allotted to the procedure part to be made zero.

CO-PO & JUSTIFICATION

308	18ECL58	HDL LAB

COURSE OUTCOMES

18ECL58	HDL LAB	C308.1	Choose either Verilog or VHDL for a given Abstraction level.
		C308.2	Familiarize with the CAD tool to write HDL programs to. Understand simulation and synthesis of digital design.
		C308.3	Program FPGAs/CPLDs to synthesize the digital designs.
		C308.4	Interface hardware to programmable ICs through I/O ports.

CO / PO MAPPING TABLE

			PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
18ECL58	HDL LAB	C308.1	3	2	2	1	3							1	2	3	
		C308.2	3	2	2	1	3							1	1	3	
		C308.3	3	2	2	1	3							1	1	3	
		C308.4	3	2	2	3	3							1	3	3	
AVERAGE			3	2	2	1.5	3							1	1.75	3	

JUSTIFICATION

			Justification	Coverage of PO's	Coverage of PSO's	Taxonomy Levels	Blooms Taxonomy keywords
18ECL58	HDL LAB	CO308.1	1.Describe the HDL code in different styles. 2.Design the combinational circuits. 3.Identify the code to design the digital system.	PO1-H PO2-M PO3-M P04-L PO5-H PO12-L	PSO1-M PSO2-H	L1 L2 L5	Describe Design Identity
		CO308.2	1.Design sync/asyncFlifflops and counters. 2.Analyze &evaluate the functionality of the design with timing analysis.	PO1-H PO2-M PO3-M PO4-L PO5-H PO12-L	PSO1-L PSO-H	L2 L4 L5	Distinguish Design Analyze
		CO308.3	1.Construct the digital system with combinational and sequential circuits. 2.Evaluate the performance using programmable IC's like FPGA and CPLD.	PO1-H PO2-M PO3-M PO4-L PO5-H PO12-L	PSO1-L PSO2-H	L4 L5	Analyze Evaluate design
		CO308.4	1.Build the hardware models for different applications like LCD, seven segment, motors, etc. 2.Evaluate the design by interfacing.	PO1-L PO2-M PO3-M PO4-H PO5-H PO12-L	PSO1-H PSO2-H	L5 L6	Build Evaluate Design

Course Handling Faculty **Course Co-ordinator** **Module Co-ordinator** **Program Co-ordinator**

DEPARTMENT OF ECE

QUESTION BANK

Subject Title: HDL LAB **Max. Exam Marks: 60**
Subject Code: 18ECL58 **Duration of Exam: 3 hrs**

1. a) Write VERILOG structural model to simulate and synthesize 2 to 4 decoder using NAND gates only.
   b) Write a VERILOG code to change the Direction of Stepper motor.

2. a) Write a VERILOG model for 32-bit ALU to simulate and synthesize using 4 bit I/P for the following functions and demonstrate the operation.
   i) A + B ii) A - B iii) A -1
   iv) A +1 v) A Complement vi) A TRUE
   vii) A * B viii) A OR B.
   b) Write VERILOG code to interface DAC and generate a Square Waveform.

3. a) Write a VERILOG code to simulate and synthesize 8:1 Multiplexer using Case statement.
   b) Write a VERILOG code to control the speed of DC motor and demonstrate the operation.

4. a) Write VERILOG behavioral model to simulate and synthesize 8 to 3 encoder without priority and demonstrate the operation.
   b) Write VERILOG code to interface DAC and generate a SINE Waveform.

5. a) Write VERILOG behavioral model to simulate and synthesize 8 to 3 encoder with priority and demonstrate the operation.
   b) Write a VERILOG code to change the Direction of Stepper motor.

6. a) Write a VERILOG code to simulate & synthesize function of Full adder gate add functionality for logical operations of XOR, XNOR, AND and OR gates.
   b) Write VERILOG code to interface DAC and generate a Triangle Waveform.

7. a) Write a VERILOG code to simulate and synthesize BCD Counter.
   b) Write a VERILOG code to change the Direction of Stepper motor.

8. a) Write VERILOG code to simulate and synthesize 4-bit binary to gray converter using 1-bit gray to binary converter, 1-bit adder and subtractor.
   b) Write a VERILOG code to control the speed of DC motor and demonstrate the operation.

9. a) Write a VERILOG code to simulate and synthesize D flip and demonstrate the operation.
   b) Write VERILOG code to interface DAC and generate a SINE Waveform.

10. a) Write a VERILOG code to simulate and synthesize SR flip-flop and demonstrate the operation.
    b) Write a VERILOG code to control the speed of DC motor and demonstrate the operation.

11. a) Write a VERILOG code to simulate and synthesize JK flip-flop and demonstrate the operation.
    b) Write VERILOG code to interface DAC and generate a Square Waveform.

12.a) Write Verilog code for counter with given input clock and check whether it works as clock divider performing division of clock by 2, 4, 8 and 16. Verify the functionality of the code.
   b) Write VERILOG code to interface DAC and generate a Triangle Waveform.

13.a) Write a VERILOG code to simulate and synthesize Binary Counter and demonstrate the operation.
   b) Write a VERILOG code to control the speed of DC motor and demonstrate the operation.

14.a) Write a VERILOG code to simulate and synthesize 8:1 Multiplexer using If else statement.
   b) Write a VERILOG code to change the Direction of Stepper motor.

LAB INDEX CARD

BANGALORE INSTITUTE OF TECHNOLOGY, BENGALURU-560004
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
MAY 2020-21

LABORATORY INDEX CARD

LAB: HDL **CODE:18ECL58**

STUDENT DETAILS	
NAME	
USN	
CLASS	
SECTION	
SEM	

COURES OUTCOMES		
Students will be able to:		POs / PSOs
CO 1	Choose either Verilog or VHDL for a given Abstraction level.	1,2,3,4,5,12 / 1,2
CO 2	Familiarize with the CAD tool to write HDL programs to understand simulation and synthesis of digital design.	1,2,3,4,5,12 / 1,2
CO 3	Program FPGAs/CPLDs to synthesize the digital designs	1,2,3,4,5,12 / 1,2
CO 4	Interface hardware to programmable ICs through I/O ports	1,2,3,4,5,12 / 1,2

SL NO.	DESCRIPTION	CO	PO / PSO	DESIGN/ CALCULATION (OBSERVATION BOOK)	VIVA	CONDUCTION/ EXECUTION	OBSERVATION/ RESULT/ EVALUATION (RECORD)	TOTAL	STAFF SIGNATURE
	MARKS			5	5	10	10	30	
	CYCLE I - COMBINATIONAL CIRCUITS								
1	2:4 DECODER USING ONLY NAND	1,2,3	1,2,3,4,5,12 1,2						
2	4 BIT BINARY TO GRAY CODE	1,2,3	1,2,3,4,5,12 1,2						
3	8:1 MUX	1,2,3	1,2,3,4,5,12 1,2						
4	8:3 ENCODER (Without priority)	1,2,3	1,2,3,4,5,12 1,2						
5	8:3 ENCODER (With priority)	1,2,3	1,2,3,4,5,12 1,2						
	CYCLE II - SEQUENTIAL CIRCUITS								
1	FULL ADDER GATE	1,2,3	1,2,3,4,5,12 1,2						
2	32 BIT ALU	1,2,3	1,2,3,4,5,12 1,2						
3	D FLIP FLOP	1,2,3	1,2,3,4,5,12 1,2						
4	SR FLIP FLOP	1,2,3	1,2,3,4,5,12 1,2						
5	JK FLIP FLOP	1,2,3	1,2,3,4,5,12 1,2						
	CYCLE III - COMBINATIONAL CIRCUITS								
1	BINARY COUNTERS WITH SYNCHRONOUS RESET	1,2,3	1,2,3,4,5,12 1,2						
2	BCD COUNTERS WITH SYNCHRONOUS RESET	1,2,3	1,2,3,4,5,12 1,2						
3	COUNTER WITH CLOCK DIVIDER	1,2,3	1,2,3,4,5,12 1,2						
	CYCLE IV - INTERFACING PROGRAMS								
1	SIGN WAVE GENERATION USING DAC	1,2,3,4	1,2,3,4,5,12 1,2						
2	TRAINGULAR WAVE GENERATION USING DAC	1,2,3,4	1,2,3,4,5,12 1,2						
3	SQUARE WAVE GENERATION USING DAC	1,2,3,4	1,2,3,4,5,12 1,2						
4	SPEED CONTROL OF A DC MOTOR	1,2,3,4	1,2,3,4,5,12 1,2						
5	DIRECTION CONTROL OF A STEPPER MOTOR	1,2,3,4	1,2,3,4,5,12 1,2						
						AVERAGE (Y)			

TEST			
	WRITEUP	10 M	
	CONDUCTION	10 M	
	VIVA		
	TOTAL(Z)	20 M	

FINAL MARKS	Y+Z	

COMMENTS IF ANY:

EXTRA PROGRAMs 1. LOGIC GATES
2. LCD DISPLAY

STUDENT SIGNATURE **STAFF SIGNATURE**

BANGALORE INSTITUTE OF TECHNOLOGY

K.R.ROAD, V.V.PURAM Bangalore-560004

I CYCLE

1. 2:4 DECODER USING ONLY NAND
2. 4 BIT BINARY TO GRAY CODE
3. 8:1 MUX
4. 8:3 ENCODER (Without priority)
5. 8:3 ENCODER (With priority)

II CYCLE

6. FULL ADDER GATE
7. 32 BIT ALU
8. D FLIP FLOP
9. SR FLIP FLOP
10. JK FLIP FLOP

III CYCLE

11. BINARY COUNTERS WITH SYNCHRONOUS RESET
12. BCD COUNTERS WITH SYNCHRONOUS RESET
13. COUNTER WITH CLOCK DIVIDER

IV CYCLE

14. SINE WAVE GENERATION USING DAC
15. TRAINGULAR WAVE GENERATION USING DAC
16. SQUARE WAVE GENERATION USING DAC
17. SPEED CONTROL OF A DC MOTOR
18. DIRECTION CONTROL OF A STEPPER MOTOR
19.

EXTRA PROGRAM

1. LOGIC GATES
2. LCD DISPLAY

INTRODUCTION TO VERILOG

Verilog is a hardware description language (HDL) used to model electronic systems. The language supports the design, verification, and implementation of analog, digital, and mixed - signal circuits at various levels of abstraction

The designers of Verilog wanted a language with syntax similar to the C programming language so that it would be familiar to engineers and readily accepted. The language is case- sensitive, has a preprocessor like C, and the major control flow keywords, such as "if" and "while", are similar. The formatting mechanism in the printing routines and language operators and their precedence are also similar

The language differs in some fundamental ways. Verilog uses Begin/End instead of curly braces to define a block of code. The concept of time, so important to a HDL won't be found in C The language differs from a conventional programming language in that the execution of statements is not strictly sequential. A Verilog design consists of a hierarchy of modules are defined with a set of input, output, and bidirectional ports. Internally, a module contains a list of wires and registers. Concurrent and sequential statements define the behavior of the module by defining the relationships between the ports, wires, and registers Sequential statements are placed inside a begin/end block and executed in sequential order within the block. But all concurrent statements and all begin/end blocks in the design are executed in parallel, qualifying Verilog as a Dataflow language. A module can also contain one or more instances of another module to define sub-behavior

A subset of statements in the language is synthesizable. If the modules in a design contains a netlist that describes the basic components and connections to be implemented in hardware only synthesizable statements, software can be used to transform or synthesize the design into the net list may then be transformed into, for example, a form describing the standard cells of an integrated circuit (e.g. ASIC) or a bit stream for a programmable logic device (e.g. FPGA).

INTRODUCTION TO FPGA(FIELD PROGRAMMABLE GATE ARRAY)

FPGA contains a two dimensional arrays of logic blocks and interconnections between logic blocks. Both the logic blocks and interconnects are programmable. Logic blocks are programmed to implement a desired function and the interconnects are programmed using the switch boxes to connect the logic blocks.

To implement a complex design (CPU for instance), the design is divided into small sub functions and each sub function is implemented using one logic block. All the sub functions implemented in logic blocks must be connected and this is done by programming the interconnects.

INTERNAL STRUCTURE OF AN FPGA

FPGAs, alternative to the custom ICs, can be used to implement an entire System On one Chip (SOC). The main advantage of FPGA is ability to reprogram. User can reprogram an FPGA to implement a design and this is done after the FPGA is manufactured. This brings the name “Field Programmable.”

Custom ICs are expensive and takes long time to design so they are useful when produced in bulk amounts. But FPGAs are easy to implement within a short time with the help of Computer Aided Designing (CAD) tools.

XILINX FPGA

Xilinx logic block consists of one Look Up Table (LUT) and one Flip-flop. An LUT is used to implement number of different functionality. The input lines to the logic block go into the LUT and enable it. The output of the LUT gives the result of the logic function that it implements and the output of logic block is registered or unregistered output from the LUT.

PROCEDURE TO CONDUCT EXPERIMENT:

The Procedure to be followed for Software and Hardware Programs are as follows:

Step 1: Go to Start Menu → All Programs Select → Xilinx ISE 9.1i and Project Navigator.

Step 2: Go to File Menu and select Close project to close previously opened project if any, and then Select New Project.

Step 3: Enter the Project name and location and Select the Top level module type as HDL.

Step 4: Select the Device family and Device name as Spartan3 and family xc3s400, pin density pq208, speed -5 for FPGA click next.

Step 5: Right click on the source file and select new source followed by Verilog module and Give the file name same as the name of the module.

Step 6: Define the ports used and their respective directions (I/P | O/P) in the next window that opens.

Step 7: Write the body of the program in Verilog code that opens and save the file after completion of editing.

Step 8: Go to the Process view window and right click on the Synthesize - XST and Select Run. Correct the errors if any.(Repeat 7 if errors are found)

Step 9: Select and Right click the source file and click on the New Source tab and then select the Test Bench Waveform and give the appropriate file name for the same.

Step 10: Make the alterations in the Clock information and initial length of the test bench if needed.

Step 11: Set or Reset the inputs as required and save the test bench waveform file.

Step 12: Go to Process view and under Xilinx ISE Simulator Right click on the Simulate Behavioral model to see the output for the input conditions.

Step 13: Make the appropriate connections between the PC and the FPGA kit for the observation of outputs in the FPGA kit and for other Hardware Programming.

Step 14: In order to download program on kit double click implement design then select Back Annotated pins under place & route.

Step 15: Go to Process view and under User Constraints, double click on the Edit Constraints (Text).

Step 16: Edit the user constraints file as required and save the same.

Step 17: Select the main source file and right click on the Implement design in the process view window and select run.

Step 18: Under the Generate Programming file tab, right click on the Configure device (Impact) and click on the Run option.

Step 19: Select the appropriate mode and make changes in the jumper settings of the FPGA Kit as required (by pass the memory device), select the appropriate.bit extension file in the pop up window.

Step 20: Right click on the Chip picture in the pop up window and right click & Select "Program" & give ok.

Step 21: Make appropriate connection and Check the output on FPGA kit.

CYCLE - I

1. Write Verilog program for 2 to 4 decoder and realize using NAND gates only (structural model). Write a test bench to verify the design:

//Main module

```
module dec(a,b,e,D);
input a,b;
output [3:0]D;
input e;
                wire ac,bc,d0,d1,d2,d3;
                nand2 n1(a,a,ac);
                nand2 n2(b,b,bc);
                nand3 n3(ac,bc,e,d0);
                nand3 n4(ac,b,e,d1);
                nand3 n5(a,bc,e,d2);
                nand3 n6(a,b,e,d3);
                nand2 n7(d0,d0,D[0]);
                nand2 n8(d1,d1,D[1]);
                nand2 n9(d2,d2,D[2]);
                nand2 n10(d3,d3,D[3]);
endmodule
```

//2-input nand gate

```
module nand2(a,b,c);
input a,b;
output c;
wire y;
and(y,a,b);
not(c,y);
endmodule
```

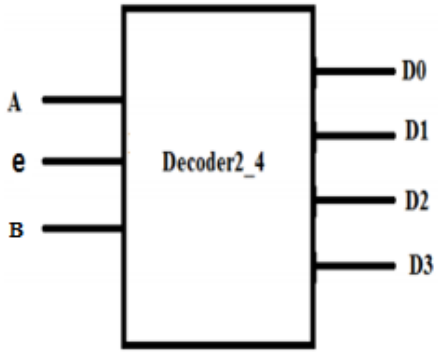
//3-input nand gate module

```
module nand3(a,b,c,d);
input a,b,c;
output d;
wire y;
and(y,a,b,c);
not(d,y);
endmodule
```

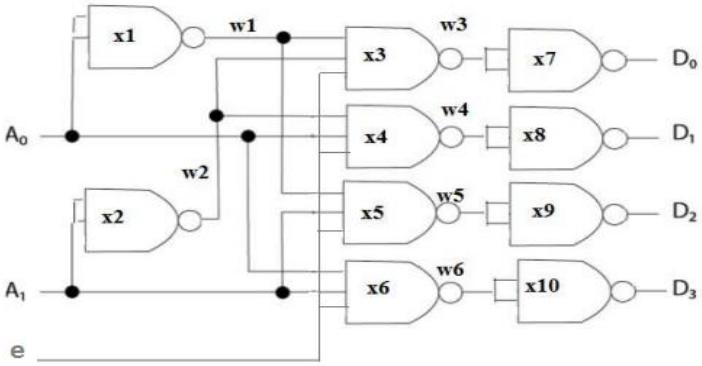
// Test bench module

```
module stimulus();
reg a,b,e;
wire [3:0]D;
dec dl(a,b,e,D);
initial
begin
 a=1'b0;b=1'b0;e=1'b0;#10 a=1'b0;b=1'b0;e=1'b1;
#10 a=1'b0;b=1'b1;e=1'b1;#10 a=1'b1;b=1'b0;e=1'b1;
#10 a=1'b1;b=1'b1;e=1'b1;#10 a=1'b1;b=1'b1;e=1'b0;
#10 $finish;
end
endmodule
```

Block Diagram:



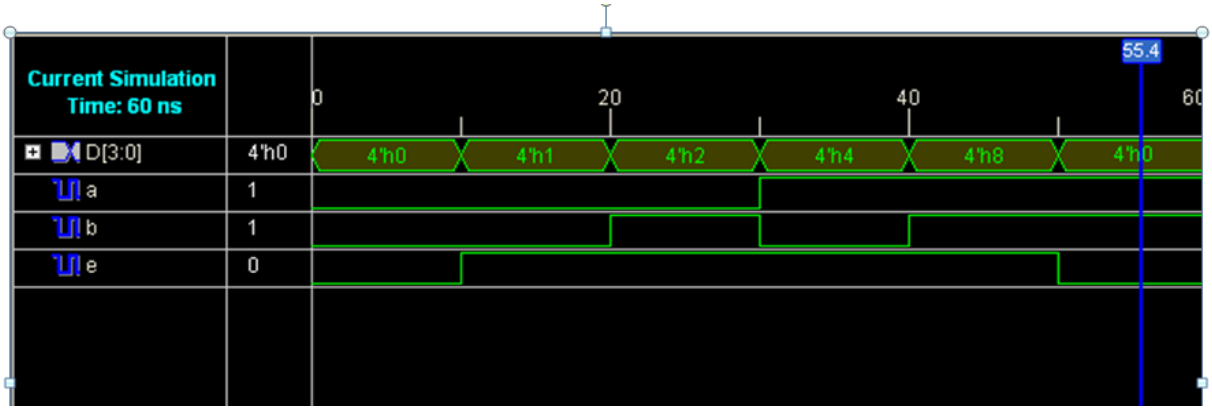
RTL Schematic:



Truth Table:

ENABLE	INPUT LINES		OUTPUT LINES			
e	A	B	D[3]	D[2]	D[1]	D[0]
0	0	0	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

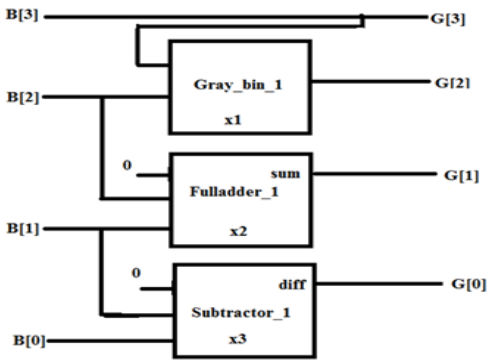
Waveform



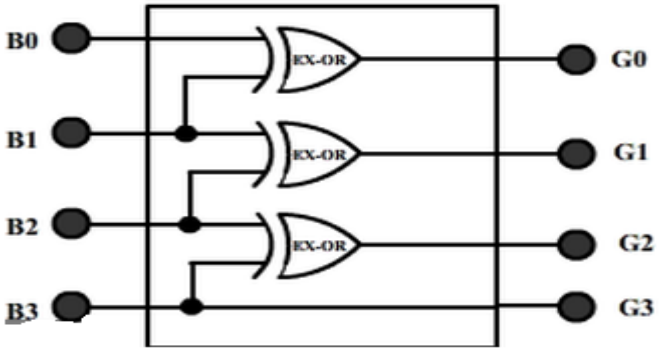
2. Write Verilog program for 4-bit binary to gray code converter using 1-bit gray to binary converter 1-bit adder and Subtractor. Write a test bench to verify the design:

```
//Main module
module BG(B,G);
output [3:0]G;
input [3:0]B;
assign G[3]=B[3];
Gray_bin_1 x1(G[2],G[3],B[2]);
Fulladder_1 x2(G[1], ,1'b0,B[2],B[1]);
Subtractor_1 x3(G[0], ,1'b0,B[1], B[0]);
endmodule
//Module for 1 bit G 2 B
module Gray_bin_1(b0,g1,g0);
output b0;
input g0,g1;
assign b0= g0 ^ g1;
endmodule
//Module for 1 bit Adder
module Fulladder_1 (sum,cout, a, b, c);
output sum, cout;
input a, b,c;
assign sum= a ^ b ^ c;
assigncout= (a & b) | (b & c) | (c & a);
endmodule
//Module for 1 bit Subtractor
module Subtractor_1 (diff,bout, a, b, c);
output diff, bout;
input a, b,c;
assign diff= a ^ b ^ c;
assign bout= ((~a) & b) | ((((~a) | b)) & c);
endmodule
//Test Bench module
module stimulus();
reg [3:0]B;
wire [3:0]G;
BG bg(B,G);
initial
begin
B=4'h0;
#10 B=4'h1; #10B=4'h3;
#10 B=4'h7; #10B=4'h9;
#10 B=4'hf;  #10B=4'ha;
#10 $finish;
end
endmodule
```

Block Diagram:



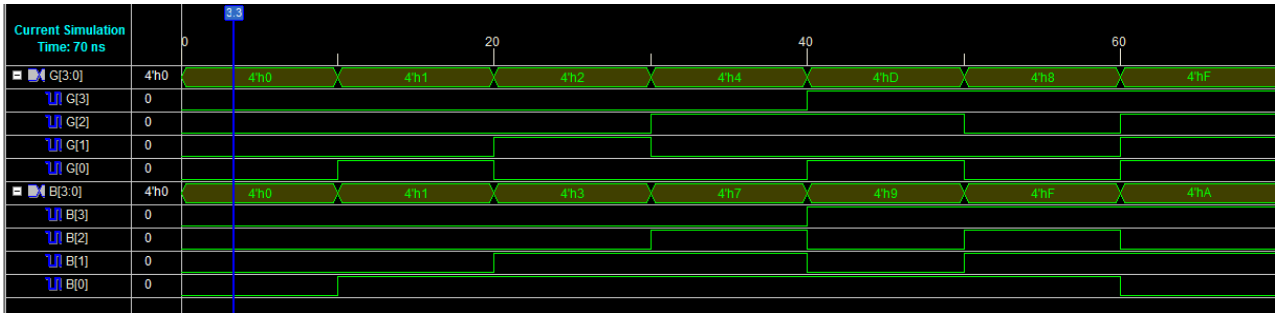
RTL Schematic:



Truth Table:

BINARY				GRAY			
B3	B2	B1	B0	G3	G2	G1	G0
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

Waveform



3. A) Write Verilog program for 8 to 1 multiplexer using case statement. Write a test bench to verify the design:

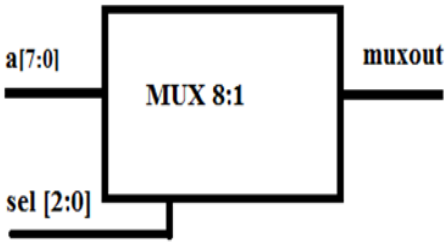
// Main module

```
module mux8_1(sel,a,muxout);
input [2:0] sel;
input [7:0] a;
output muxout;
reg muxout;

always @(sel,a)
begin

case(sel)

        3'd0: muxout = a[0];
        3'd1: muxout = a[1];
        3'd2: muxout = a[2];
        3'd3: muxout = a[3];
        3'd4: muxout = a[4];
        3'd5: muxout = a[5];
        3'd6: muxout = a[6];
        3'd7: muxout = a[7];
        default:;
endcase
end

endmodule
```

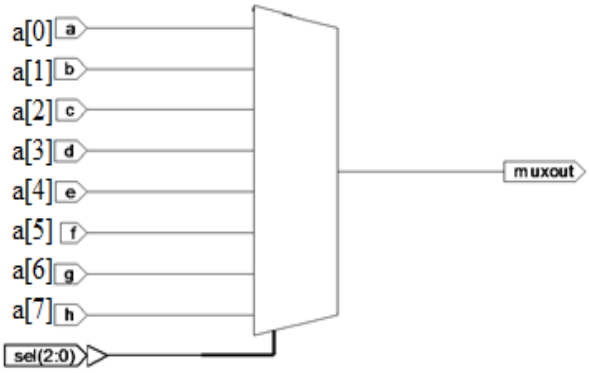
// Test bench module

```
module stimulus();
reg [2:0]sel;
reg [7:0]a;
wire muxout;
mux8_1 m1 (sel,a,muxout);
initial
begin
a=8'b00000000;
#10 a=8'b00000001;sel=3'b000;
#10 a=8'b00000010;sel=3'b001;
#10 a=8'b00000100;sel=3'b010;
#10 a=8'b00001000;sel=3'b011;
#10 a=8'b00010000;sel=3'b100;
#10 a=8'b00100000;sel=3'b101;
#10 a=8'b01000000;sel=3'b110;
#10 a=8'b10000000;sel=3'b111;
#10 $finish;
end
endmodule
```

Block Diagram:



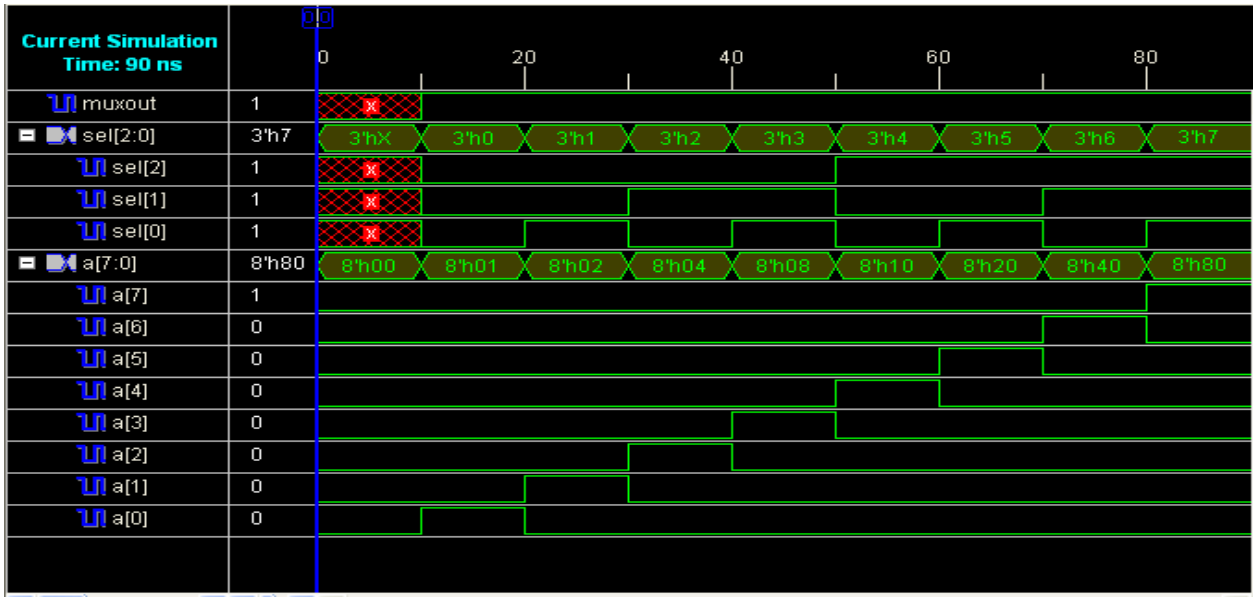
RTL Schematic:



Truth Table:

SELECT DATA INPUTS			OUTPUTS
Sel[2]	**Sel[1]**	**Sel[0]**	**muxout**
0	0	0	a [0]
0	0	1	a [1]
0	1	0	a [2]
0	1	1	a [3]
1	0	0	a [4]
1	0	1	a [5]
1	1	0	a [6]
1	1	1	a [7]

Waveform:



3. B) Write Verilog program for 8 to 1 multiplexer using if statement. Write a test bench to verify the design:

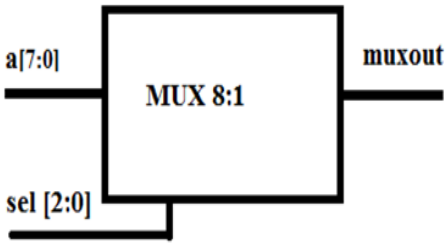
// Main module

```
module mux(a,s1,s2,s3,,y);
input [7:0] a;
input  s1,s2,s3;
output y;
reg y;
always @(a,s1,s2,s3)
begin
if ((s1 ==0)&&(s2==0)&&(s3==0))y = a[0];
else if ((s1 ==1)&&(s2==0)&&(s3==0))y = a[1];
else if ((s1 ==0)&&(s2==1)&&(s3==0))y = a[2];
else if ((s1 ==1)&&(s2==1)&&(s3==0))y = a[3];
else if ((s1 ==0)&&(s2==0)&&(s3==1))y = a[4];
else if ((s1 ==1)&&(s2==0)&&(s3==1))y = a[5];
else if ((s1 ==0)&&(s2==1)&&(s3==1))y = a[6];
else if ((s1 ==1)&&(s2==1)&&(s3==1))y = a[7];
else
begin
y = 1'bz;
end
end
endmodule
```

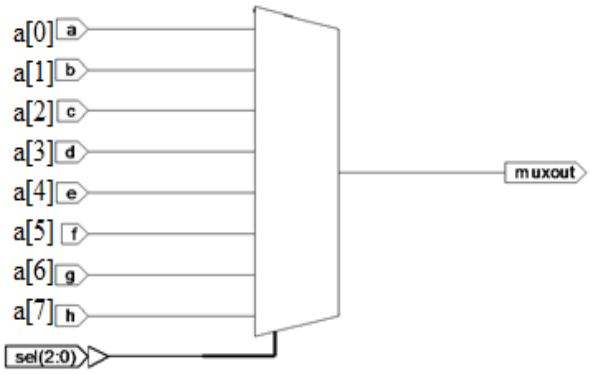
// Test bench module

```
module stimulus();
reg s1,s2,s3;
reg [7:0]a;
wire y;
mux  m1 (a,s1,s2,s3,y);
initial
begin
a=8'b00000000;
#10 a=8'b00000001; {s3,s2,s1}=3'b000;
                                    //#10 a=8'b00000000; {s3,s2,s1}=3'b000;
#10 a=8'b00000010; {s3,s2,s1}= 3'b001;
                                    //#10 a=8'b00000000; {s3,s2,s1}=3'b001;
#10 a=8'b00000100; {s3,s2,s1}=3'b010;
#10 a=8'b00001000; {s3,s2,s1}= 3'b011;
#10 a=8'b00010000; {s3,s2,s1}= 3'b100;
#10 a=8'b00100000; {s3,s2,s1}= 3'b101;
#10 a=8'b01000000; {s3,s2,s1}= 3'b110;
#10 a=8'b10000000; {s3,s2,s1}= 3'b111;
#10 $finish;
end
endmodule
```

Block Diagram:



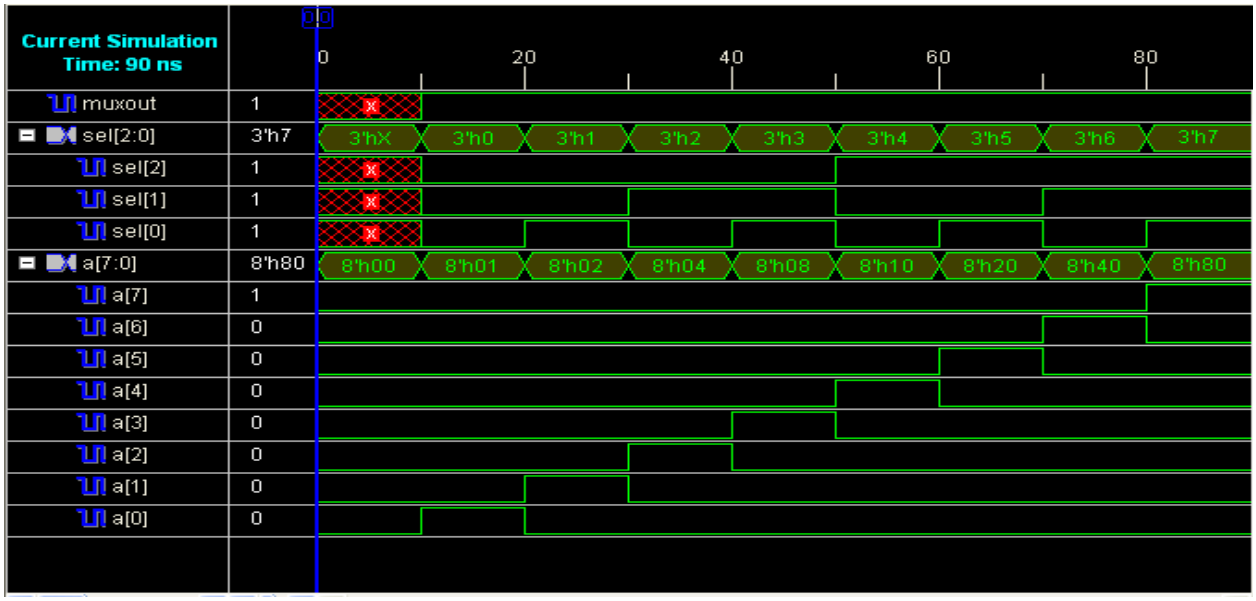
RTL Schematic:



Truth Table:

SELECT DATA INPUTS			OUTPUTS
sel_2	sel_1	sel_0	muxout
0	0	0	a [0]
0	0	1	a [1]
0	1	0	a [2]
0	1	1	a [3]
1	0	0	a [4]
1	0	1	a [5]
1	1	0	a [6]
1	1	1	a [7]

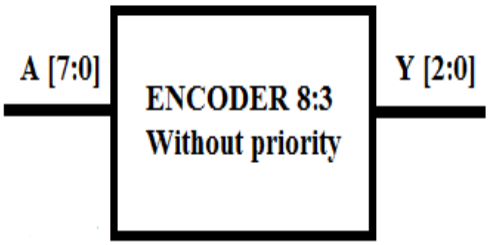
Waveform:



4. Write Verilog program 8 to 3 encoder without priority (behavioral model). Write a test bench to verify the design:

```
        // Main module
module encoder8_3(A, Y);
input [7:0] A;
output [2:0] Y;

reg [2:0] Y;

always @(A)
        begin
                case (A)
        8'b10000000:Y=3'd7;
        8'b01000000:Y=3'd6;
        8'b00100000:Y=3'd5;
        8'b00010000:Y=3'd4;
        8'b00001000:Y=3'd3;
        8'b00000100:Y=3'd2;
        8'b00000010:Y=3'd1;
         8'b00000001:Y=3'd0;
        default:Y=3'bXXX;
        endcase
        end
endmodule

        // Test bench module
module stimulus();

reg [7:0]A;
wire [2:0]Y;
encoder8_3 e1(A,Y);
initial
begin
   A=8'b00000000;
#10 A=8'b00000001;
#10 A=8'b00000010;
#10 A=8'b00000100;
#10 A=8'b00001000;
#10 A=8'b00010000;
#10 A=8'b00100000;
#10 A=8'b01000000;
#10 A=8'b10000000;
#10 $finish;
end
endmodule
```

Block Diagram:

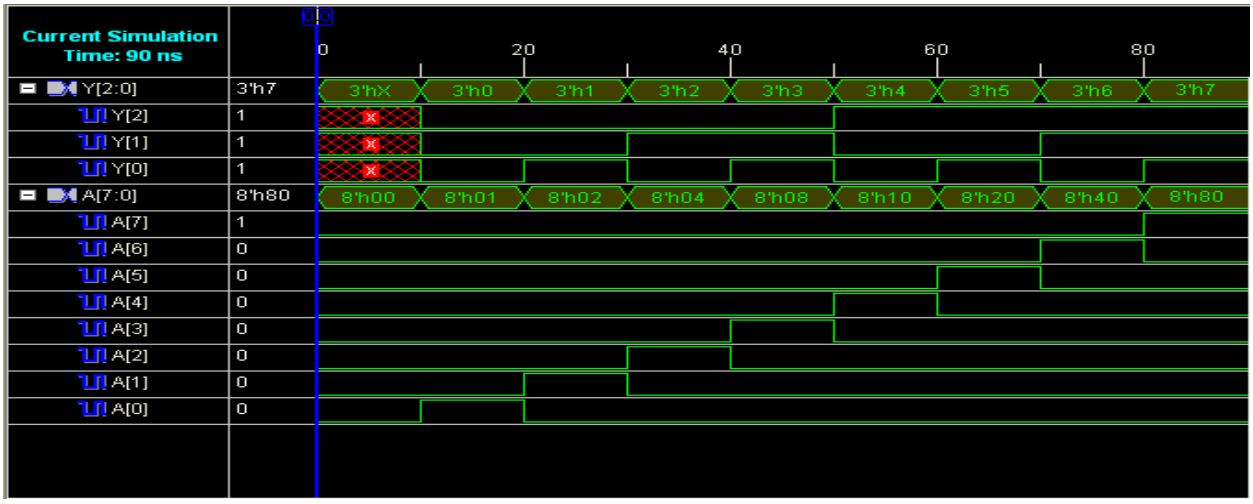


RTL Schematic:

Truth Table:

INPUTS								OUTPUTS		
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	0	X	X	X

Waveform:

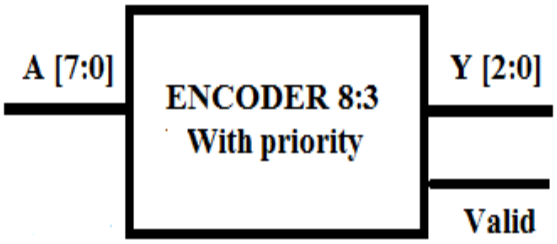


5. Write Verilog program 8 to 3 encoder with priority (behavioral model). Write a test bench to verify the design:

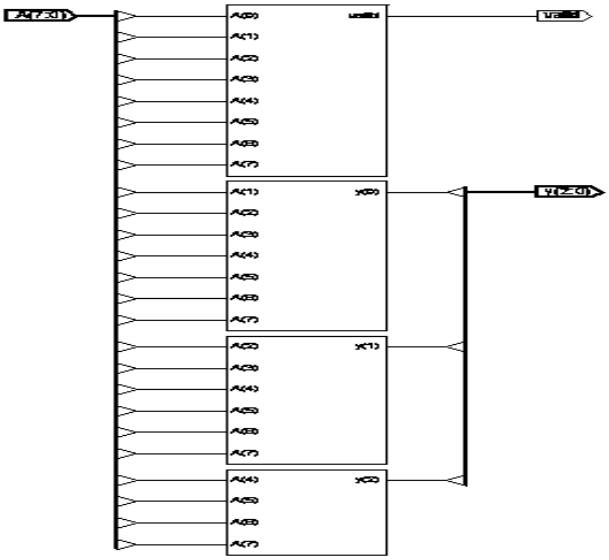
```
// Main module
module enco_p_8_3(A, valid, y);
        input [7:0] A;
        output valid;
        output [2:0]y;
        reg valid;
        reg [2:0]y;

        always @(A)
        begin
        valid = 1;

        if (A[7] ==1)y = 3'd7;
        else if (A[6] ==1)y = 3'd6;
        else if (A[5] ==1)y = 3'd5;
        else if (A[4] ==1)y = 3'd4;
        else if (A[3] ==1)y = 3'd3;
        else if (A[2] ==1)y = 3'd2;
        else if (A[1] ==1)y = 3'd1;
        else if (A[0] ==1)y = 3'd0;
else
        begin
                valid = 0;
                y = 3'bXXX;
        end
        end
endmodule
```

```
// Test bench module
module stimulus();
reg [7:0]A;
wire valid;
wire [2:0]Y;
enco_p_8_3 e1(A,valid,Y);
initial
begin
  A=8'b00000000;
#10 A=8'b00000001;
#10 A=8'b0000001X;
#10 A=8'b000001XX;
#10 A=8'b00001XXX;
#10 A=8'b0001XXXX;
#10 A=8'b001XXXXX;
#10 A=8'b01XXXXXX;
#10 A=8'b1XXXXXXX;
#10 A=8'b011XXXXX;

#10 $finish;
end
endmodule
```

Block Diagram:



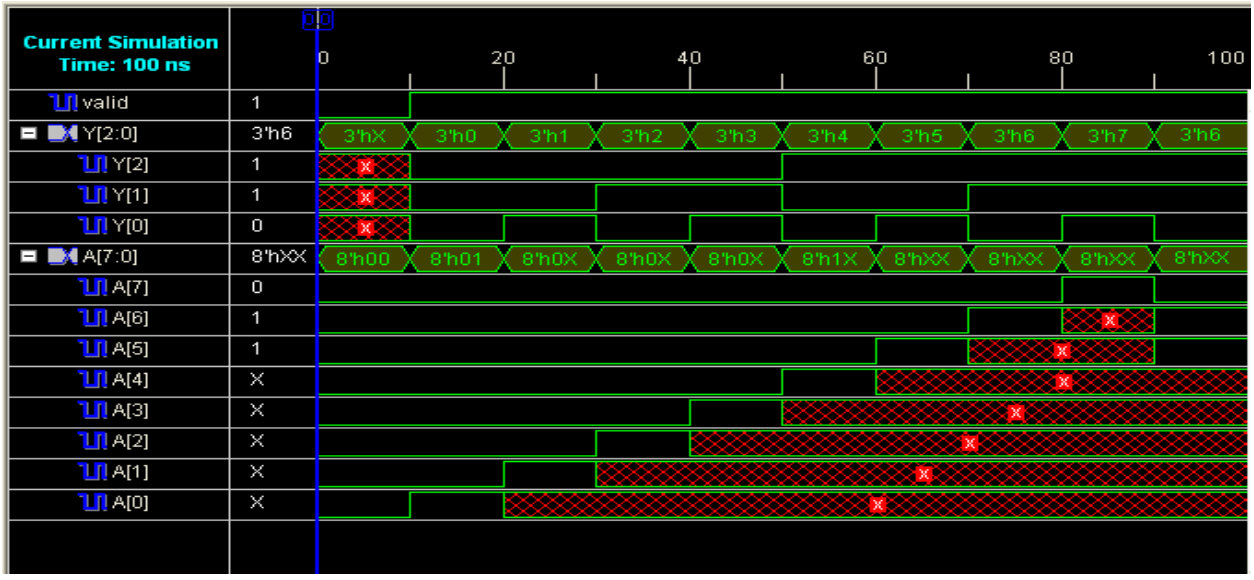
RTL Schematic:



Truth Table:

INPUTS								OUTPUTS			
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Y_2	Y_1	Y_0	Valid
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	1
0	0	0	0	0	0	0	0	X	X	X	0

Waveform:



CYCLE - II

6. Model in Verilog for a full adder and add functionality to perform logical operations of XOR, XNOR, AND and OR gates. Write test bench with appropriate input patterns to verify the modeled behaviour.

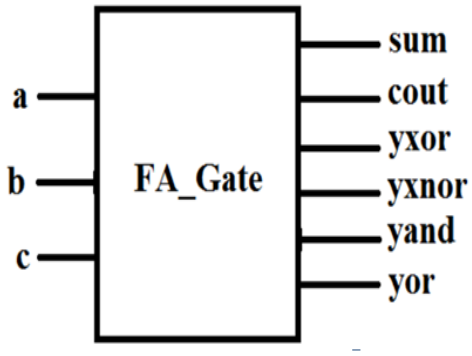
// Main module

```
module fulladd(sum,cout,yxor,yxnor,yand,yor,a,b,c);
output sum,cout,yxor,yxnor,yand,yor;
input a,b,c;
assign sum = a^b^c;
assign cout = (a&b)|(b&c)|(c&a);
assign yxor = a^b^c;
assign yxnor = ~(a^b^c);
assign yand = a&b&c;
assign yor = a|b|c;
endmodule
```

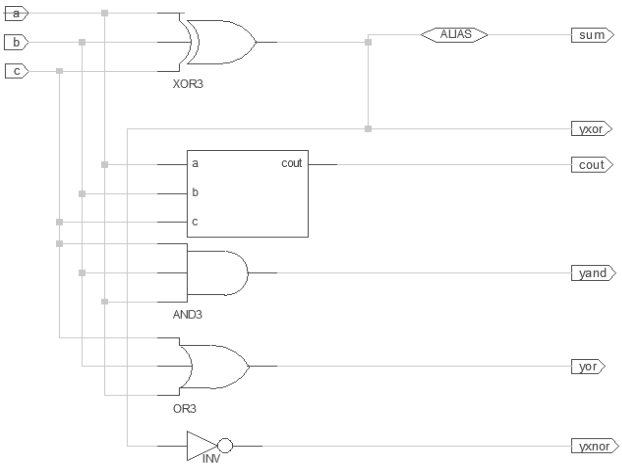
// Test bench module

```
module stimulus();
wire sum,cout,yxor,yxnor,yand,yor;
reg a,b,c;
fulladd g(sum,cout,yxor,yxnor,yand,yor,a,b,c);
task display;
begin
$display (" a= ",a," b= ",b,  "c=",c,"sum = " ,sum, "cout =", cout,
 " yxor= ",yxor," yxnor= ",yxnor,  "yand = ",yand, " yor = ",yor);
end
endtask
initial
begin
a=1'b0;b=1'b0;c=1'b0;   #5;display;
a=1'b0;b=1'b0;c=1'b1;   #5;display;
a=1'b1;b=1'b1;c=1'b0;   #5;display;
a=1'b1;b=1'b1;c=1'b1;   #5;display;
a=1'b1;b=1'b0;c=1'b0;   #5;display;
a=1'b1;b=1'b0;c=1'b1;   #5;display;
a=1'b1;b=1'b1;c=1'b0;   #5;display;
a=1'b1;b=1'b1;c=1'b1;   #5;display;
end
endmodule
```

Block Diagram:



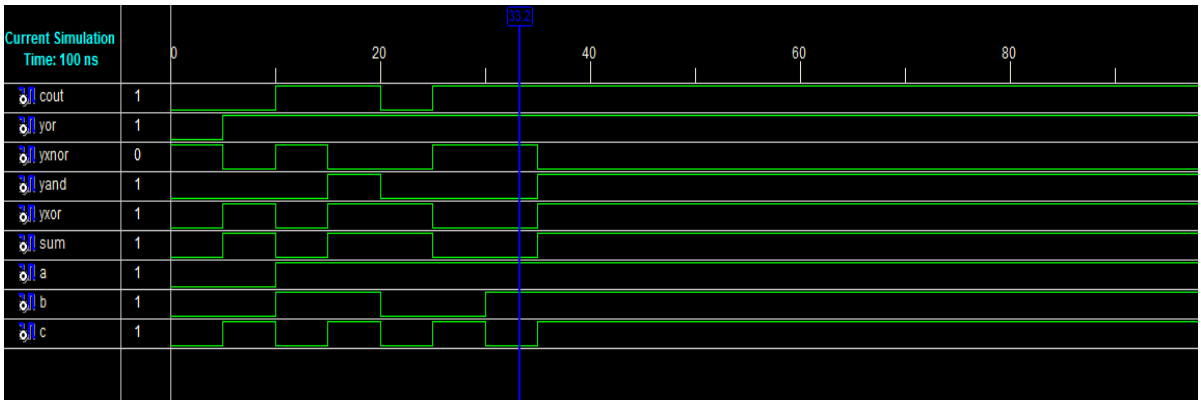
RTL Schematic:



Truth Table:

INPUTS				OUTPUTS				
A	B	C	ANDG	ORG	XORG	XNORG	SUM	COUT
0	0	0	0	0	0	1	0	0
0	0	1	0	1	1	0	1	0
`0	1	0	0	1	1	0	1	0
0	1	1	0	1	0	1	0	1
1	0	0	0	1	1	0	1	0
1	0	1	0	1	0	1	0	1
1	1	0	0	1	0	1	0	1
1	1	1	1	1	1	0	1	1

Waveform:



7. Verilog 32-bitALUshowninfigurebelowandverifythefunctionalityof ALUby selecting appropriate test patterns. The functionality of the ALU is presented in Table-1.

a. Write test bench to verify the functionality of the ALU considering all possible input patterns
b. The enable signal will set the output to required functions if enabled, if disabled all the outputs are set to tri-state
c. The acknowledge signal is set high after every operation is completed

Opcode (2:0)	ALU Operation	Remarks	
000	A + B	Addition of two numbers	Both A and B are in two's complement format
001	A – B	Subtraction of two numbers	
010	A + 1	Increment Accumulator by 1	A is in two's complement format
011	A - 1	Decrement accumulator by 1	
100	A	True	Inputs can be in any format
101	A Complement	Complement	
110	A OR B	Logical OR	
111	A AND B	Logical AND	

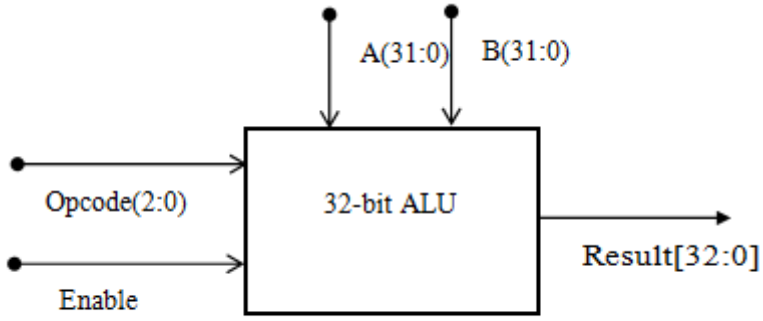


Figure 1 ALU top level block diagram

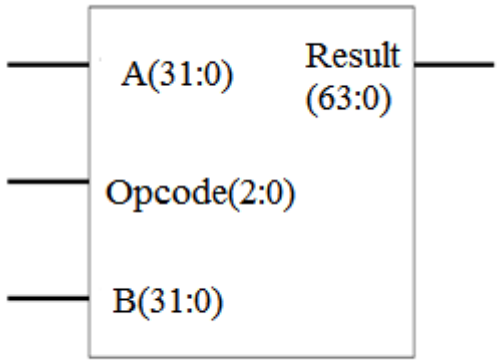
// Main module

```
module alu(Result, A, B, opcode, Enable);
output [63:0] Result;
input signed[31:0] A, B;
input [2:0] opcode;
input Enable;
reg [63:0] Result;
always@(opcode,A,B,Enable)
begin
if(Enable==0)
begin
Result=64'bx;
end

else
begin
case(opcode)

3'b000: begin Result=A+B; end
3'b001: begin Result=A-B; end
3'b010: begin Result=A+1; end
3'b011: begin Result=A-1; end
3'b100: begin
if(A)
 Result =1'b1;
else
Result =1'b0;
end
3'b101: begin Result=~A; end
3'b110: begin Result=A|B; end
3'b111: begin Result=A&B; end
endcase
end
end
endmodule
```

// Test bench module

```
module stimulus();
reg [2:0]opcode;
reg [31:0] a,b;
reg Enable;
wire [63:0]Result;
alu  m1 (Result, a, b, opcode, Enable);
initial
begin
a=16'h0000; b=16'h0000; Enable =1'b0;
#10 Enable=1'b1; a=32'hABCD1212; b=32'h71611111; opcode=3'b000;
#10 Enable=1'b1; a=32'hABCD1212; b=32'h71611111; opcode=3'b001;
#10 Enable=1'b1; a=32'hABCD1212; b=32'h71611111; opcode=3'b010;
#10 Enable=1'b1; a=32'hABCD1212; b=32'h71611111; opcode=3'b011;
#10 Enable=1'b1; a=32'hABCD1212; b=32'h71611111; opcode=3'b100;
#10 Enable=1'b1; a=32'hABCD1212; b=32'h71611111; opcode=3'b101;
#10 Enable=1'b1; a=32'hABCD1212; b=32'h71611111; opcode=3'b110;
#10 Enable=1'b1; a=32'hABCD1212; b=32'h71611111; opcode=3'b111;
#10 $finish;
end
endmodule
```

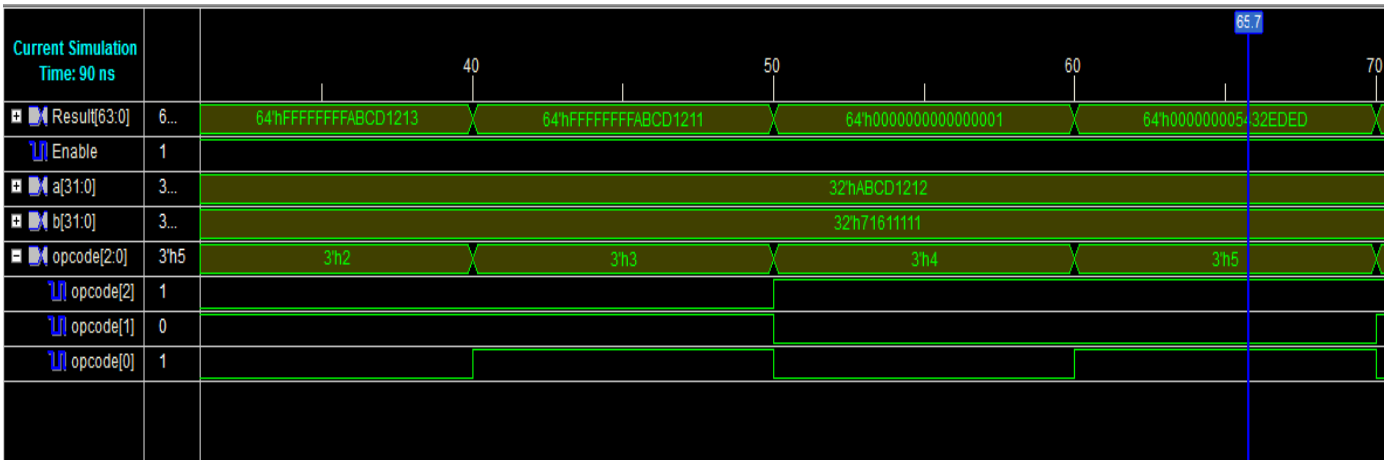
Block Diagram:



Truth Table:

Operation	Opcode	A (32' h)	B (32' h)	Result
a+b	000	ABCD1212	71611111	64'h 000000001D2E2323
a-b	001	ABCD1212	71611111	64'h FFFFFFFF3A6C0101
a +1	010	ABCD1212	71611111	64'h FFFFFFFFABCD1213
a -1	011	ABCD1212	71611111	64'h FFFFFFFFABCD1211
A (True)	100	ABCD1212	71611111	64'h 0000000000000001
A Complement	101	ABCD1212	71611111	64'h 000000005432EDED
A OR B	110	ABCD1212	71611111	64'h FFFFFFFFFBED1313
A AND B	111	ABCD1212	71611111	64'h 0000000021411010

Waveform:

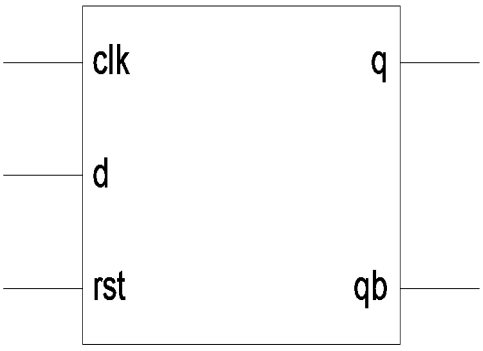


8. Write a VERILOG code to simulate and synthesize D flip-flop and Demonstrate the operation.

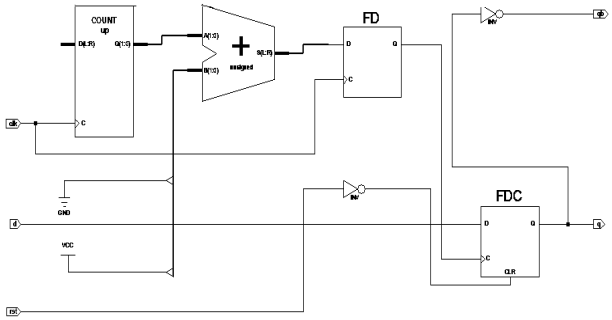
```
        // Main  module
module dff( d, clk,rst, q, q_bar);
input d, clk, rst;
output reg q, q_bar;
//*USE the following  block of code only during synthesis*
 reg[20:0]clk_signal;
 reg clk_1;
 initial
        clk_signal=21'd0;
        always@(posedge clk)
begin
clk_signal=clk_signal+1;
clk_1=clk_signal[16];
end

always @(posedge clk or negedge rst)    //*for synthesis use clk_1 instead of clk*
                begin
                if(!rst)
                begin
                        q <= 1'b0;
                        q_bar<=1'b1;
                end
                else
                begin
                        q <= d;
                        q_bar<= !d;
                end
        end
endmodule
        //Test bench module
module dff_test;
regd,rst,clk;
wire q,qbar;
        // Instantiate the D-flipflop
dff d1 (.d(d),.clk(clk),.rst(rst), .q(q), .q_bar(q_bar));
task display;
begin
        $display("time=%0d",$time,"ns"," n_rst=",rst," din=",d," q=",q," q_bar=",q_bar);
end
endtask
initial
begin
        d = 0;
        clk = 0;
end
          // Generating clock signal
always
#5 clk = ~clk;
        // Add stimulus here
initial
begin
        rst=1'b0;d=1'b1;#10;display;
        rst=1'b1;d=1'b1;#10;display;
                d=1'b0;#10;display;
                d=1'b1;#10;display;
```

```
        rst=1'b0;d=1'b1;#10; display;
        rst=1'b1;d=1'b1;#10;display;
end
endmodule
```

Block Diagram:



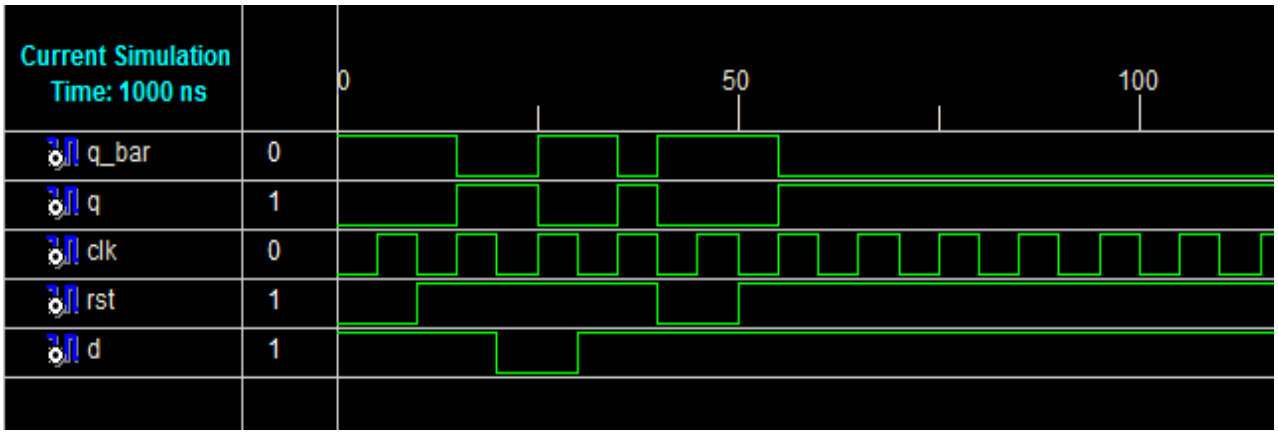
RTL Schematic:



Truth Table:

Inputs			Outputs	
CLK	RESET	D	Q	$\bar{Q}_b$
1	1	1	1	0
1	1	0	0	1
X	0	X	0	1

Waveform:

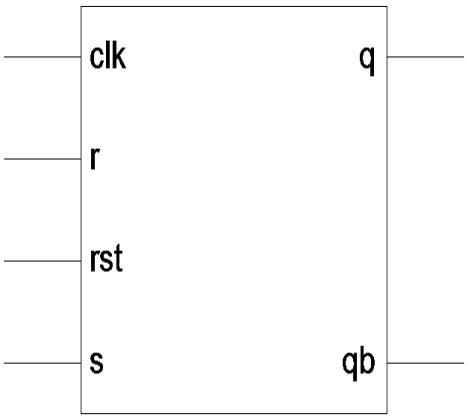


9. Write a VERILOG code to simulate and synthesize SR flip-flop and Demonstrate the operation.

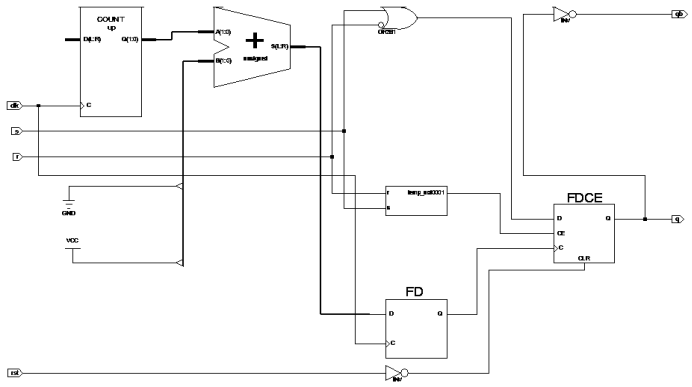
```
        //Main module
module srff(q,qbar,s,r,clk,rst);
output q,qbar;
input clk,rst,s,r;
reg q;
//*USE the following  block of code only during synthesis*
 reg[20:0]clk_signal;
 reg clk_1;
 initial
        clk_signal=21'd0;
        always@(posedge clk)
begin
clk_signal=clk_signal+1;
clk_1=clk_signal[16];
end

always @(posedge clk or negedge rst)    //*for synthesis use clk_1 instead of clk*
                begin
        if (!rst)
                q <= 1'b0;

                else if (s == 1'b0 && r == 1'b0)
                        q <= q;
                else if (s == 1'b0 && r == 1'b1)
                        q <= 1'b0;
                else if (s == 1'b1 && r == 1'b0)
                        q <= 1'b1;
                else if (s == 1'b1 && r == 1'b1)
                        q <= 1'bx;
        end
        assign qbar = ~q;
endmodule

         // Test bench module
module sr_ff_test;
        reg clk,rst,s,r;
        wire q,qbar;
        srff  sr1(q,qbar,s,r,clk,rst);
        task display;
        begin
        $display("time=%0d",$time,"ns","  rst=",rst," s=",s," r=",r," q=",q," qbar=",qbar);
        end
        endtask
        initial
clk = 1'b0;
always
 #5 clk = ~clk;
initial
begin
                rst=0;  s=1;r=0;        #10;display;
                rst=1;  s=1;r=0;        #10;display;
                        s=0;r=0;        #10;display;
                        s=0;r=1;        #10;display;
                        s=1;r=1;        #10;display;
        end
```

```
 initial
 #70 $finish;
endmodule
```

Block Diagram:



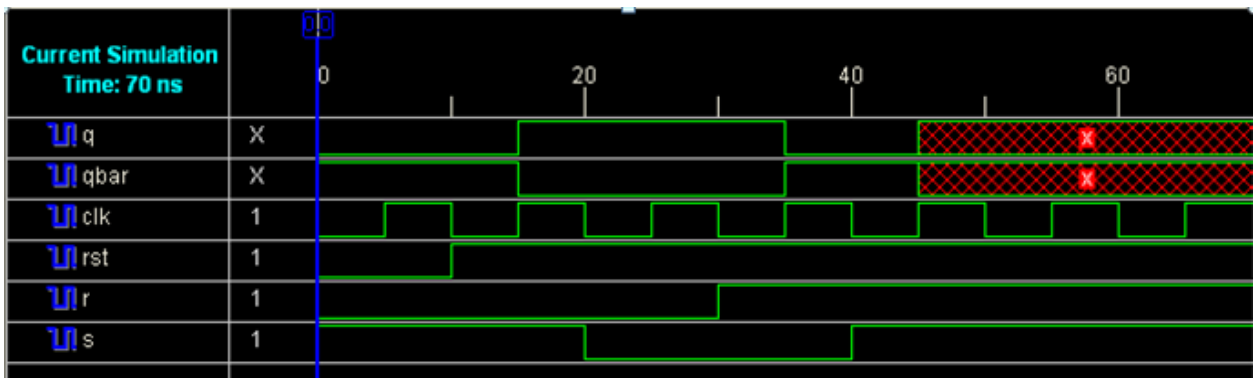
RTL Schematic:



Truth Table:

Inputs				Outputs	
CLK	**RESET**	**S**	**R**	**Q**	**$\bar{Q}_b$**
X	0	X	X	0	1
X	1	X	X	1	0
1	1	0	0	Q	Qb
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	X	X

Waveform:

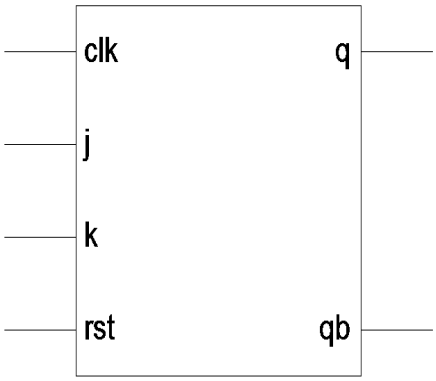


10. Write a VERILOG code to simulate and synthesize JK flip-flop and demonstrate the operation.

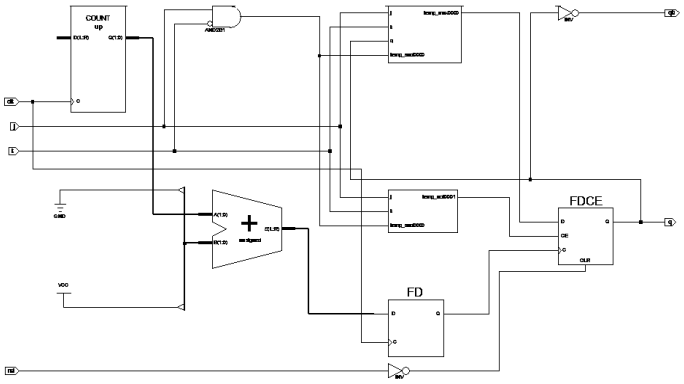
```
        // Main module
module jkff1(q,qbar,clk,rst,j,k);
input clk,rst,j,k;
output q,qbar;
reg q;
//*USE the following  block of code only during synthesis*
 /*reg[20:0]clk_signal;
 reg clk_1;
 initial
        clk_signal=21'd0;
        always@(posedge clk)
 begin
clk_signal=clk_signal+1;
clk_1=clk_signal[16];
 end  */
    always @(posedge clk or negedge rst)    //*for synthesis use clk_1 instead of clk*
        begin
        if (!rst)
        begin
                q <= 1'b0;
        end
        else  begin
                if (j == 1'b1 && k == 1'b0)
                        begin
                                q <=1'b1;
                        end
                else  if
                 (j == 1'b0 && k == 1'b0)
                        begin
                                q <=q;
                        end

                else if (j == 1'b0 && k == 1'b1)
                        begin
                                q <=1'b0 ;
                        end
              else if (j == 1'b1 && k == 1'b1)
                        begin
                                q <=~q;
                        end
        end      end
        assign qbar=~q;
endmodule
        // Test bench module
module jk_ff_test;
reg clk,rst,j,k;
wire q,qbar;
jkff1 inst(q,qbar,clk,rst,j,k);
task display;
begin
        $display("time=%0d",$time,"ns","  rst=",rst," j=",j," k=",k," q=",q," qbar=",qbar);
end
endtask
initial
clk = 1'b0;
```

```
always
#5 clk = ~clk;
initial
begin
        rst=0;          j=1;k=0;#10;    display;
        rst=1;          j=1;k=0;#10;    display;
                        j=0;k=0;#10;    display;
                        j=0;k=1;#10;    display;
                        j=1;k=1;#10;    display;
                        j=1;k=1;#10;    display;
                                #10;
            end
            initial
            #70 $finish;
           endmodule
```

Block Diagram:



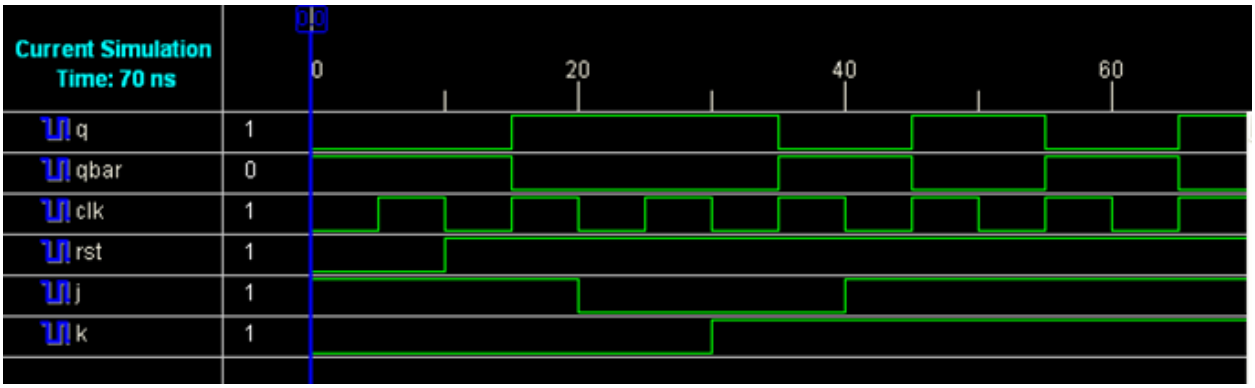
RTL Schematic:



Truth Table:

Inputs				Outputs	
CLK	**RESET**	**J**	**K**	**Q**	**States**
X	0	X	X	0	Reset
1	1	0	0	Q	NC
1	1	0	1	0	Reset
1	1	1	0	1	Set
1	1	1	1	Toggle	

Waveform:

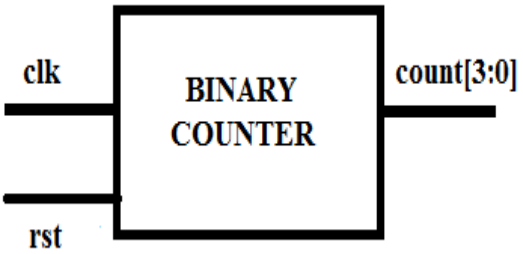


CYCLE - III

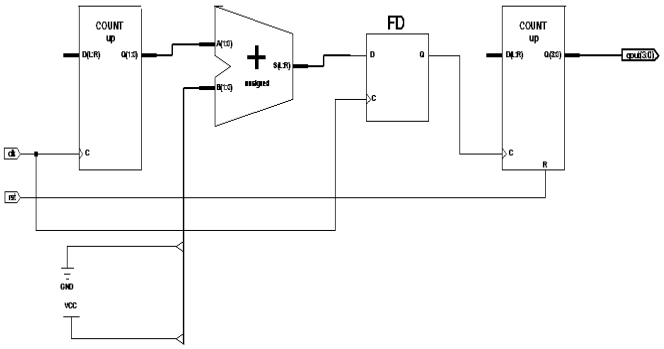
11. Write a VERILOG code to simulate and synthesize Binary Counter with Synchronous reset and demonstrate the operation.

```
        //Main module

module sync(count,rst,clk);
input wire rst, clk;
output reg [3:0] count;

//*USE the following  block of code only during synthesis*
reg[20:0]clk_signal;
reg clk_1;
initial
        clk_signal=21'd0;
        always@(posedge clk)
begin
clk_signal=clk_signal+1;
clk_1=clk_signal[16];
end

always @(posedge clk or negedge rst)    //*for synthesis use clk_1 instead of clk*

if (rst)
count<= 4'b0000;
else
count<= count + 4'b0001;

endmodule

        // Test bench module
module sycounter_t ;
wire [3:0] count;
reg rst,clk;
sync  m1 ( count, rst,clk);
initial
clk = 1'b0;

always
 #5 clk = ~clk;

initial
begin
#10; rst = 1'b1 ;
#10; rst = 1'b0 ;
#10; rst = 1'b1 ;
#10; rst = 1'b0 ;
#200 $finish;
end

initial
$monitor ($time,  "Output count = %d ",count );

endmodule
```

Block Diagram:



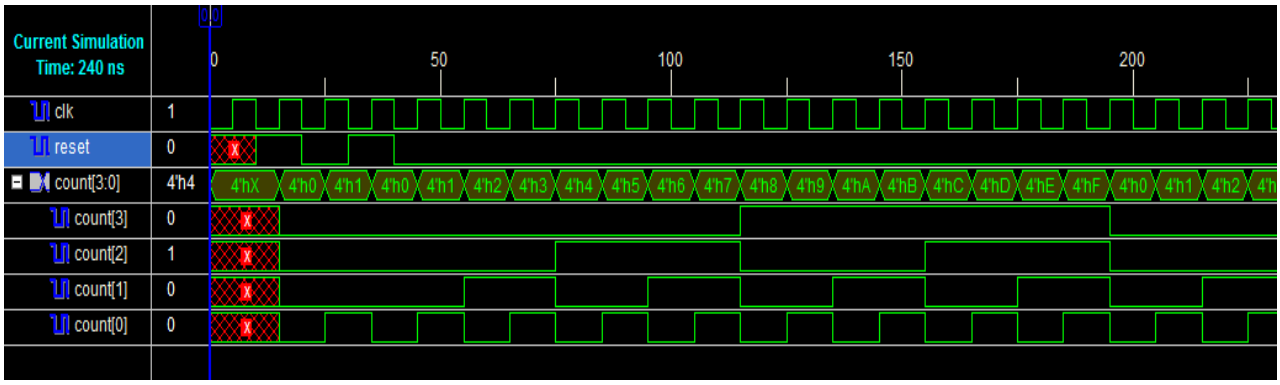
RTL Schematic:



Truth Table:

Clock Pulse	Reset	Count[3]	Count[2]	Count[1]	Count[0]
X	1	0	0	0	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	0	1	1
1	0	0	1	0	0
1	0	0	1	0	1
1	0	0	1	1	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	0	1
1	0	1	0	1	0
1	0	1	0	1	1
1	0	1	1	0	0
1	0	1	1	0	1
1	0	1	1	1	0
1	0	1	1	1	1

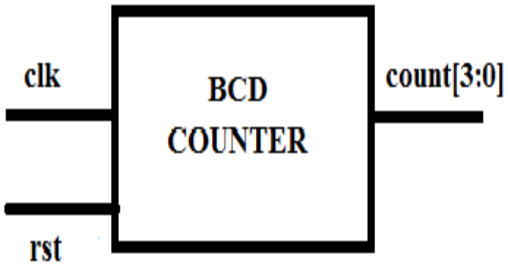
Waveform:



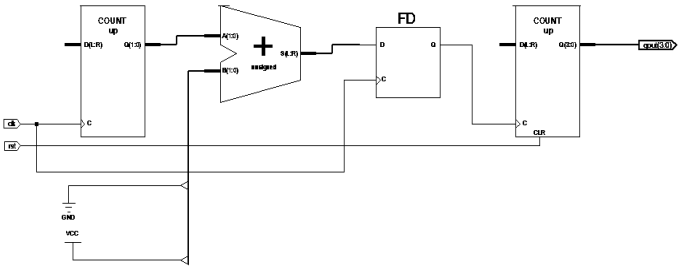
12. Write a VERILOG code to simulate and synthesize 4 Bit BCD Counter and demonstrate the operation.

```
            // Main module
module BCD(count, rst,clk);
input wire rst, clk;
output reg [3:0] count;

  //*USE the following  block of code only during synthesis*
   reg[20:0]clk_signal;
   reg clk_1;
   initial
            clk_signal=21'd0;
            always@(posedge clk)
  begin
  clk_signal=clk_signal+1;
  clk_1=clk_signal[16];
  end

always @(posedge clk or negedge rst)    //*for synthesis use clk_1 instead of clk*
begin
if ((rst ==1)|(count==4'b1001))
count =4'b0000;
else
count = count + 4'b0001;
end
endmodule

            // Test bench module
module BCD_t ;
wire [3:0] count;
reg rst,clk;
BCD  m1 (count, rst,clk);
initial
clk = 1'b0;
always
 #5 clk = ~clk;
initial
begin
#10; rst = 1'b1 ;
#10; rst = 1'b0 ;
#10; rst = 1'b1 ;
#10; rst = 1'b0 ;
#200 $finish;
end
initial
$monitor ($time,  "Output count = %d ",count );
endmodule
```

Block Diagram:



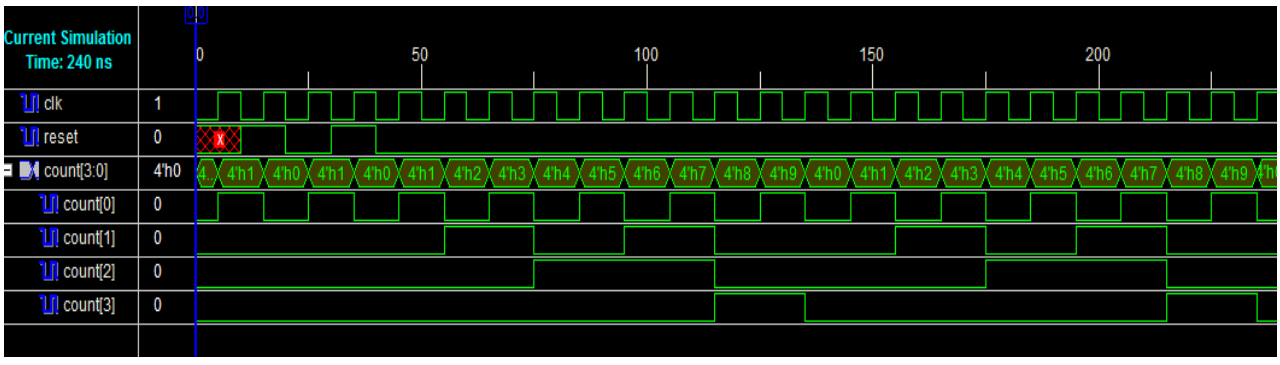
RTL Schematic:



Truth Table:

Clock Pulse	Reset	Count[3]	Count[2]	Count[1]	Count[0]
X	1	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	0	0	0	0

Waveform:

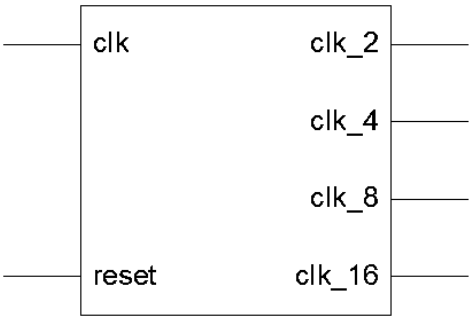


13. Write Verilog code for counter with given input clock and check whether itworks as clock divider performing division of clock by 2, 4, 8 and 16. Verify the functionality of the code.

```
        // Main module
module freq_div(clk_2,clk_4,clk_8,clk_16, rst,clk);
input wire rst, clk;
output reg clk_2,clk_4,clk_8,clk_16;
initial
count[3:0]=4'b0000;
    reg [3:0]count;
    //*USE the following  block of code only during synthesis*
     reg[20:0]clk_signal;
     reg clk_1;
     initial
          clk_signal=21'd0;
          always@(posedge clk)
     begin
    clk_signal=clk_signal+1;
    clk_1=clk_signal[16];
    end
        always @(posedge clk)    //*for synthesis use clk_1 instead of clk*
        begin
                    if (rst ==1)
                    begin count =4'b0000; end
                    else
          begin count = count + 4'b0001;end
         clk_2=count[0];
         clk_4=count[1];
         clk_8=count[2];
         clk_16=count[3];
         end
        endmodule

        // Test bench module
        module freq_div_test ;
        wire clk_2,clk_4,clk_8,clk_16;
        reg rst,clk;
        freq_div  m1 ( clk_2,clk_4,clk_8,clk_16, rst,clk);
        initial
        clk = 1'b0;
        always
         #5 clk = ~clk;

        initial
        begin
        #10; rst = 1'b1 ;
        #10; rst = 1'b0 ;
        #10; rst = 1'b1 ;
        #10; rst = 1'b0 ;
```

```
#200 $finish;
end
initial
$monitor ($time,"Output clk_2=%d " ,clk_2, "Output clk_4=%d " ,clk_4, "Output clk_8=%d"
 ,clk_8, "Output clk_16=%d" ,clk_16);
endmodule
```

Block Diagram:

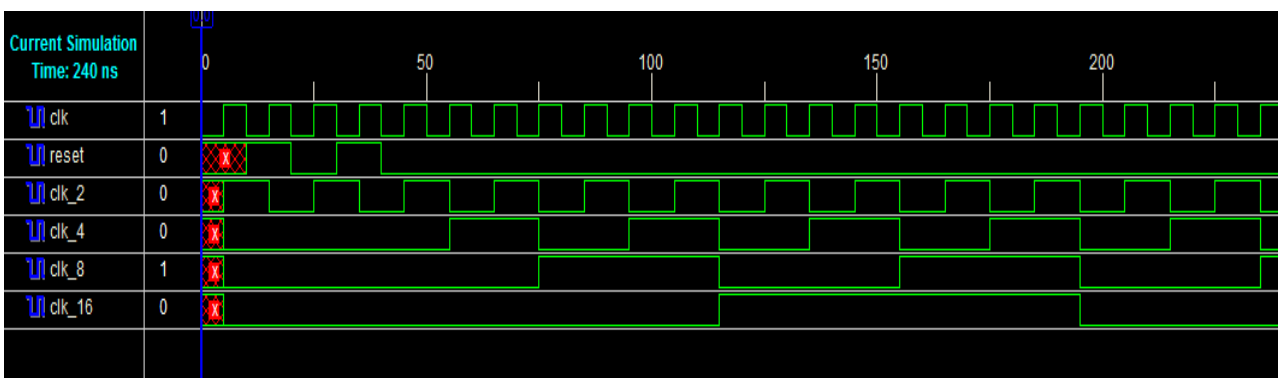


RTL Schematic:

Truth Table:

Clock Pulse	Reset	Clk_16	Clk_8	Clk_4	Clk_2
X	1	0	0	0	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	0	1	1
1	0	0	1	0	0
1	0	0	1	0	1
1	0	0	1	1	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	0	1
1	0	1	0	1	0
1	0	1	0	1	1
1	0	1	1	0	0
1	0	1	1	0	1
1	0	1	1	1	0
1	0	1	1	1	1

Waveform:



CYCLE –IV

14. Write a VERILOG code to interface DAC and generate a SINE Waveform.

// Main module

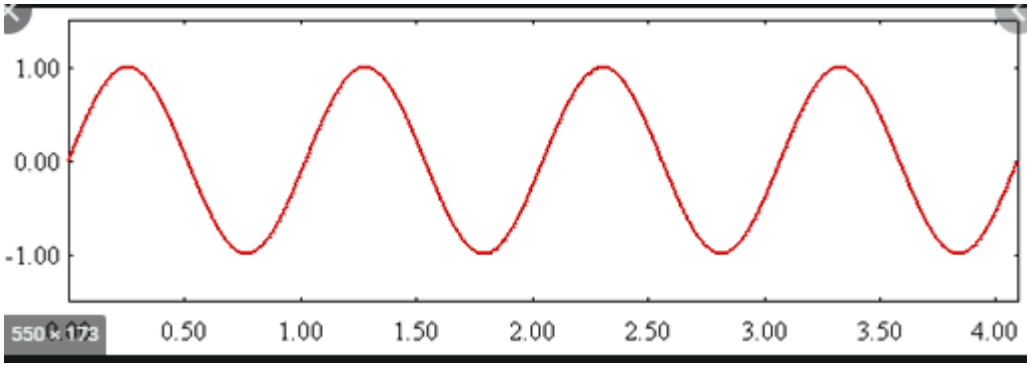
```
module sine_wave(Clk,data_out);
                        //declare input and output
input Clk;
output [7:0] data_out;
                        //declare the sine ROM - 30 registers each 8 bit wide.
reg [7:0] sine [0:63];
                        //Internal signals
integeri;
reg [7:0] data_out;
                        //Initialize the sine rom with samples.
initial begin
i = 0;
sine[0] = 135;//77;
sine[1] =150;//93 ;
sine[2] = 164;//108;
sine[3] = 177;//122;
sine[4] = 189;//135;
sine[5] = 200;//147;
sine[6] = 210;//158;
sine[7] = 219;//168;
sine[8] = 227;
sine[9] = 234;
sine[10] =240;
sine[11] =245;
sine[12] =249;
sine[13] =252;
sine[14] =254;
sine[15] =255;
sine[16] =255;
sine[17] =254;
sine[18] =252;
sine[19] =249;
sine[20] =245;
sine[21] =240;
sine[22] =234;
sine[23] = 227;
sine[24] = 219;
sine[25] =210;
sine[26] =200;
sine[27] =189;
sine[28] =177;
sine[29] =164;
sine[30] =150;
sine[31] =135;
sine[32]=119 ;
sine[33]=104 ;
```

```
sine[34] = 90;
sine[35] = 77;
sine[36] =65 ;
sine[37] = 54;
sine[38] = 44;
sine[39] = 35;
sine[40] = 27;
sine[41] = 20;
sine[42] =14;
sine[43] =9;
sine[44] =5;
sine[45] =2;
sine[46] =0;
sine[47] =0;
sine[48] =2;
sine[49] =5;
sine[50] =9;
sine[51] =14;
sine[52] =20;
sine[53] =27;
sine[54] =35;
sine[55] = 44;
sine[56] = 54;
sine[57] =65;
sine[58] =77;
sine[59] =90;
sine[60] =104;
sine[61] =119;
sine[62] =135;
sine[63] =150;
end
```

//At every positive edge of the clock, output a sine wave sample.

```
always@ (posedge(Clk))
begin
data_out = sine[i];
i = i+ 1;
if(i == 63)
i = 0;
end
endmodule
```

Waveform:



#PINLOCK_BEGIN

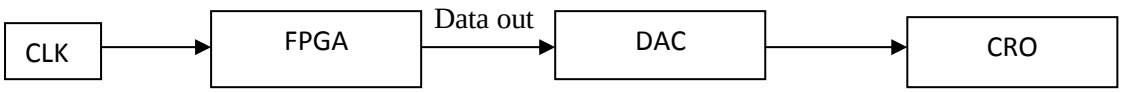
NET "clk" LOC = P2; → Connect to 100K Clock Pin

NET "op<0>" LOC = P4; -> To Pin 21 of DAC
NET "op<1>" LOC = P5; -> To Pin 22 of DAC
NET "op<2>" LOC = P7; -> To Pin 19 of DAC
NET "op<3>" LOC = P9; -> To Pin 20 of DAC
NET "op<4>" LOC = P10; ->To Pin 17 of DAC
NET "op<5>" LOC = P11; ->To Pin 18 of DAC
NET "op<6>" LOC = P12; ->To Pin 15 of DAC
NET "op<7>" LOC = P13; ->To Pin 16 of DAC

→ Connect Pin 26 of DAC to GND Pin

#PINLOCK_END

BLOCK DIAGRAM OF RAMP / TRIANGULAR / SQUARE WAVEFORM

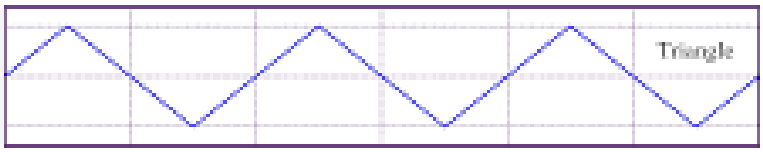


15. Write a VERILOG code to interface DAC and generate a Triangular Waveform.

//Main module

```
module tri_dac (clk,rst,op);

inputclk;
inputrst;
output   [7:0] op;

wire    [7:0] op;
reg     [7:0] q;
regud;

initial
begin
ud = 1'b 0;
end

always @(posedge clk or negedge rst)
begin
if (rst == 1'b 0)
       q = 8'b 0;
else
begin

if (ud == 1'b 0)
         q = q + 1;

else if (ud == 1'b 1 )

         q = q - 1;

end
end

assign op = q;

always @(posedge clk)
begin

if (q == 8'b 11111110)

ud<= 1'b 1;

else if (q == 8'b 00000001 )

ud<= 1'b 0;
end
endmodule
```

Waveform:



#PINLOCK_BEGIN

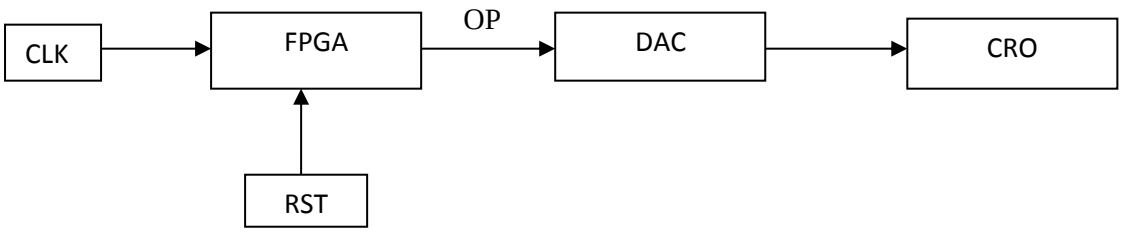
NET "clk" LOC = P2; → Connect to 100K Clock Pin

NET "rst" LOC = P3;

NET "op<0>" LOC = P4; -> To Pin 21 of DAC
NET "op<1>" LOC = P5; -> To Pin 22 of DAC
NET "op<2>" LOC = P7; -> To Pin 19 of DAC
NET "op<3>" LOC = P9; -> To Pin 20 of DAC
NET "op<4>" LOC = P10; ->To Pin 17 of DAC
NET "op<5>" LOC = P11; ->To Pin 18 of DAC
NET "op<6>" LOC = P12; ->To Pin 15 of DAC
NET "op<7>" LOC = P13; ->To Pin 16 of DAC
→ Connect Pin 26 of DAC to GND Pin

#PINLOCK_END

BLOCK DIAGRAM OF RAMP / TRIANGULAR / SQUARE WAVEFORM

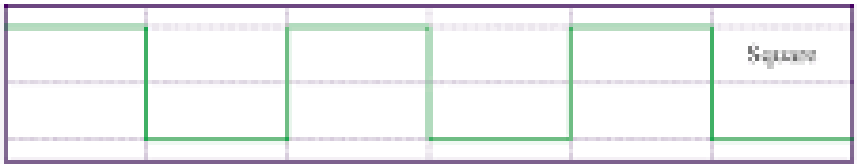


16..Write a VERILOG code to interface DAC and generate a Square Waveform.

//Main module

```
module squarewave(clk,rst,op);

input clk,rst;
output [7:0] op;

wire [7:0] op;
reg [7:0] temp;
reg [7:0] q;

always@(posedge clk)

begin

if(~rst)
q=8'b0;
else
q=q+1;
end

always@(q)
begin

if(q<=127)
temp=8'b00000000;
else
temp=8'b11111111;
end

assign op = temp;

endmodule
```

Waveform:



#PINLOCK_BEGIN

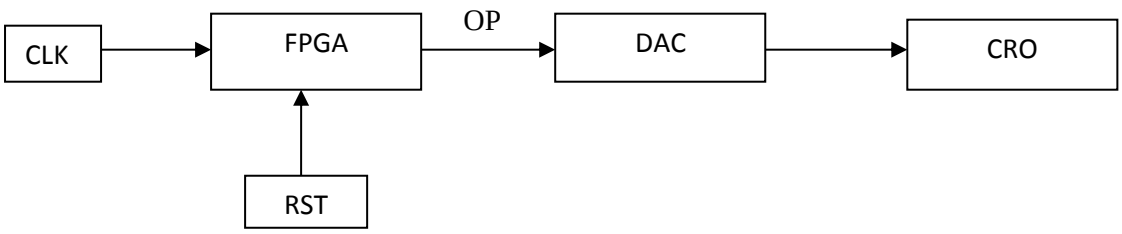
NET "clk" LOC = P2; → Connect to 100K Clock Pin

NET "rst" LOC = P3;

NET "op<0>" LOC = P4; -> To Pin 21 of DAC
NET "op<1>" LOC = P5; -> To Pin 22 of DAC
NET "op<2>" LOC = P7; -> To Pin 19 of DAC
NET "op<3>" LOC = P9; -> To Pin 20 of DAC
NET "op<4>" LOC = P10; ->To Pin 17 of DAC
NET "op<5>" LOC = P11; ->To Pin 18 of DAC
NET "op<6>" LOC = P12; ->To Pin 15 of DAC
NET "op<7>" LOC = P13; ->To Pin 16 of DAC
→ Connect Pin 26 of DAC to GND Pin

#PINLOCK_END

BLOCK DIAGRAM OF RAMP / TRIANGULAR / SQUARE WAVEFORM



17. Write a VERILOG code to control the speed of DC motor and demonstrate the operation.

Theory:

A DC motor is any of a class of rotary electrical machines that converts direct current electrical energy into mechanical energy. The most common types rely on the forces produced by magnetic fields. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current flow in part of the motor**.**

//Main module

```
module dcmotor (psw, pdcm, clk,dir_in,dir_out);
input   [2:0] psw;
output pdcm;
input clk;
input dir_in;
output dir_out;
reg dir_out;
reg pdcm;
reg     [11:0] sclkdiv;

//  count upto 3000 rpm
regvdcm;

always @(posedge clk)
begin
sclkdiv<= sclkdiv + 1;
end
        always @ (psw or sclkdiv)
        begin
        if(dir_in == 1'b0)
        dir_out = 1'b 0;
        else
        dir_out=1'b 1;
        end

always @(psw or sclkdiv)
begin

if (sclkdiv == 12'b 000000000000)
begin
                vdcm = 1'b 1;
end

//  1f4, 320, 44c, 578, 6a4, 7d0, 8fc, 9c4

if (psw == 3'b 000 &sclkdiv == 12'b 000111110100)
begin
        vdcm = 1'b 0;
end
else if (psw == 3'b 001 &sclkdiv == 12'b 001100100000 )
begin
vdcm = 1'b 0;
end
else if (psw == 3'b 010 &sclkdiv == 12'b 010001001100 )
begin
vdcm = 1'b 0;
end
else if (psw == 3'b 011 &sclkdiv == 12'b 010101111000 )
```

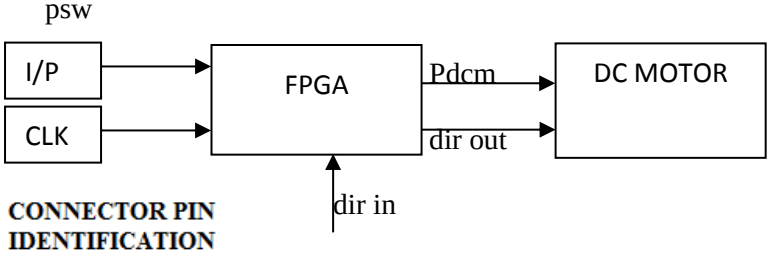
```
begin
vdcm = 1'b 0;
end
else if (psw == 3'b 100 &sclkdiv == 12'b 011010100100 )
begin
vdcm = 1'b 0;
end
else if (psw == 3'b 101 &sclkdiv == 12'b 011111010000 )
begin
vdcm = 1'b 0;
end
else if (psw == 3'b 110 &sclkdiv == 12'b 100011111100 )
begin
vdcm = 1'b 0;
end
else if (psw == 3'b 111 &sclkdiv == 12'b 100111000100 )
begin
vdcm = 1'b 0;
end
                    if (vdcm == 1'b 1)
                            begin
                                    pdcm<= 1'b 1;
                            end
          else
                            begin
                                    pdcm<= 1'b 0;
                            end
end
endmodule
```

#PINLOCK_BEGIN

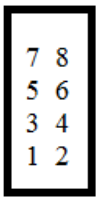
```
NET "clk" LOC = P2;          → Connect to 100K Clock Pin
NET "dir_in" LOC = P4;       → Direction control PIN
NET "dir_out" LOC = P5       → Connect to Pin No. 2 of DCMOTOR
NET "psw<0>" LOC = P10;
NET "psw<1>" LOC = P11;
NET "psw<2>" LOC = P12;
NET "pdcm" LOC = P9; -       → Connect to Pin No. 4 of DCMOTOR
```

#PINLOCK_END

BLOCK DIAGRAM OF DC MOTOR



CONNECTOR PIN IDENTIFICATION

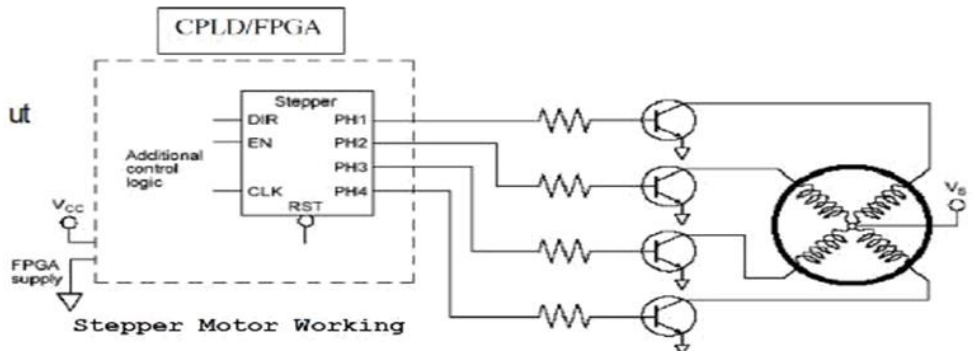


18. Write a VERILOG code to control the Direction and speed of Stepper motor and demonstrate it.

Theory:

A stepper motor is a digital motor. It can be driven by digital signal. The motor has two phase with center tap winding. The centre taps of these windings are connected to the 12V supply. Due to this motor can be excited by grounding four terminals of the two windings.

Motor can be rotated in steps by giving proper excitation to these windings. These excitation signals are buffered using transistor. The transistors are selected such that they can source the stored current for the windings.Motor is rotated by 1.8 degree per excitation. Speed can be changed by varying the clock.



//Main module

```
module stepm(clk,rst,dir,step_ctrl,dout);
input clk,rst,dir;
input [1:0] step_ctrl;
output [3:0] dout;

reg [3:0] dout;
reg [20:0] dclk;
reg div_clk;

always@(posedge clk)
        begin
                dclk=dclk+1;
        end

always@(posedge clk)
begin
        case(step_ctrl)
                2'b00:div_clk=dclk[20];
                2'b01:div_clk=dclk[18];
                2'b10:div_clk=dclk[16];
                2'b11:div_clk=dclk[14];
        endcase
end

always@(posedge div_clk)
begin

        if(!rst)
                dout=4'b1001;

        else

        case(dir)
```

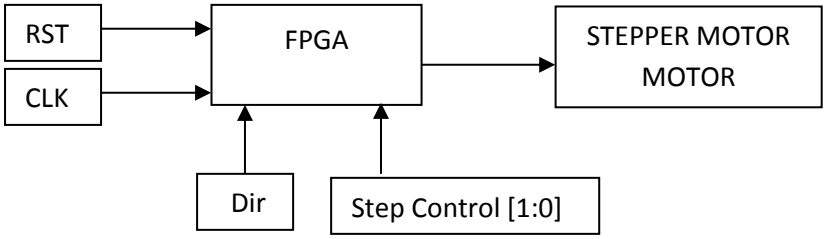
```
                1'b0:dout={dout[0],dout[3:1]};
                1'b1:dout={dout[2:0],dout[3]};
        endcase
end

endmodule
```

#PINLOCK_BEGIN

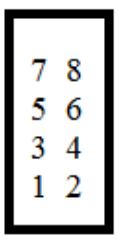
```
NET "clk" LOC = P2;                  → Connect to 100K Clock Pin
NET "rst" LOC = P3;
NET "dir" LOC = P4;
NET "step_ctrl<0>" LOC = P5;  }→ Speed Control PIN's
NET "step_ctrl<1>" LOC = P7;  }

NET "dout<0>" LOC = P31;      }
NET "dout<1>" LOC = P33;      }
NET "dout<2>" LOC = P34;      }→ Connect Pin 1, 2, 3 and 4 of Stepper Motor
NET "dout<3>" LOC = P35;      }
```

#PINLOCK_END

BLOCK DIAGRAM OF STEPPER MOTOR



CONNECTOR PIN IDENTIFICATION

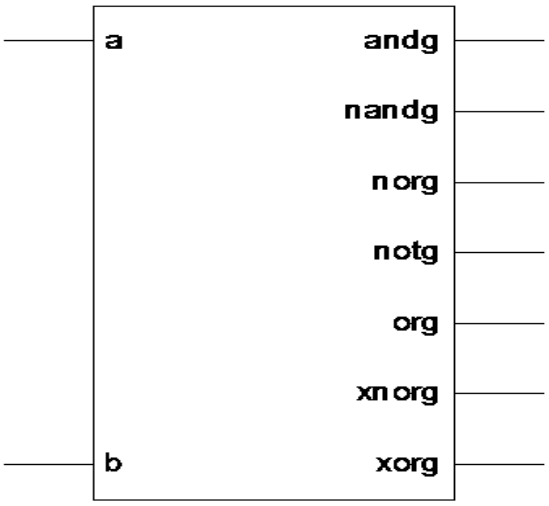


Extra Program

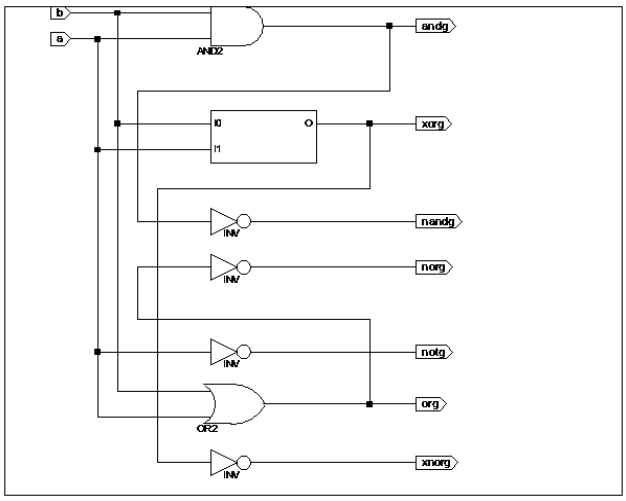
1. **Write a VERILOG code to simulate and synthesize Logic Gates and demonstrate the operation.**

```
module gates(a,b,andg,org,nandg,norg,xorg,xnorg);
output andg,org,nandg,norg,xorg,xnorg;
inputa,b;
assign andg =a&b;
assign org = a|b;
assign nandg =~(a&b);
assign norg =~(a|b);
assign xorg =(a&(~b))|((~a)&b);
assign xnorg =(a&b)|((~a)&(~b));
endmodule

module gates_test;
wire andg,org,nandg,norg,xorg,xnorg;
rega,b;
gates g(a,b,andg,org,nandg,norg,xorg,xnorg);
task display;
begin
$display (" a= ",a," b= ",b," and_out= ",andg," or_out= ",org," nand_out= ",nandg,"
nor_out= ",norg," xor_out= ",xorg,
" xnor_out= ",xnorg);
end
endtask
initial
begin
a=1'b0;b=1'b0;#5;display;
a=1'b0;b=1'b1;#5;display;
a=1'b1;b=1'b0;#5;display;
a=1'b1;b=1'b1;#5;display;
end
endmodule
```

Block Diagram:



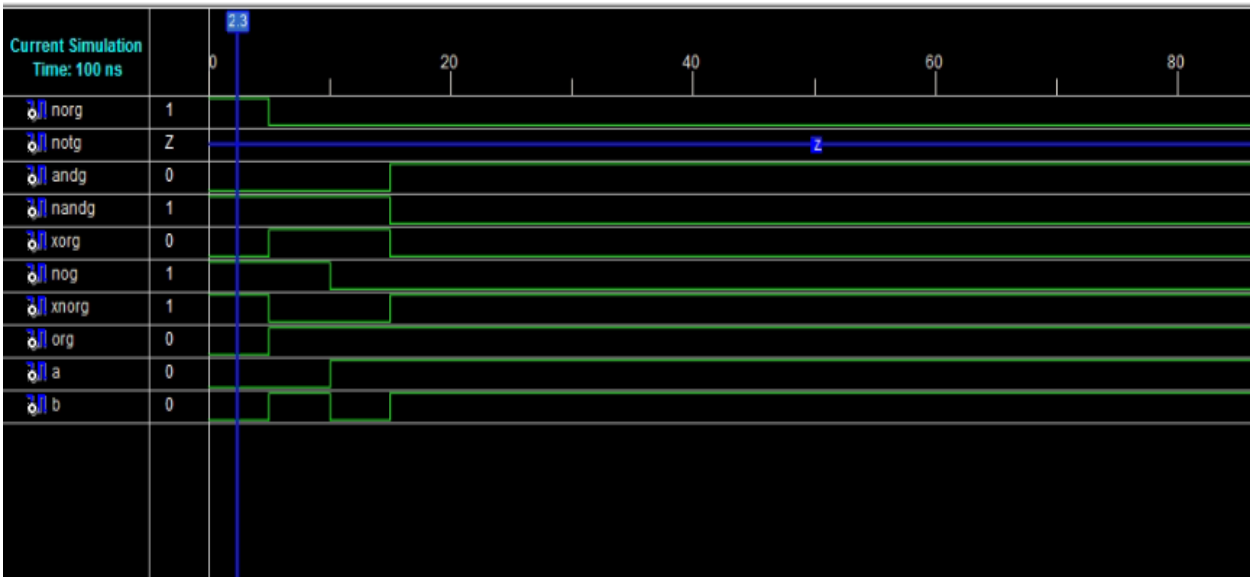
RTL Schematic:



Truth Table:

INPUTS			OUTPUTS					
A	B	ANDG	NOTG	ORG	NANDG	NORG	XORG	XNORG
0	0	0	1	0	1	1	0	1
0	1	0	1	1	1	0	1	0
1	0	0	0	1	1	0	1	0
1	1	1	0	1	0	0	0	1

Waveform:



2. Write a VERILOG code for LCD Display and demonstrate it.

```
module LCD1602(clk, rs, rw, en,dat,LCD_P,LCD_N);
input clk;
output [7:0] dat;
output  rs,rw,en,LCD_P,LCD_N;

reg e;
reg [7:0] dat;
regrs;
reg  [15:0] counter;
reg [4:0] current,next;
regclkr;
reg [31:0] cnt=0;

parameter  set0=4'h0;
parameter  set1=4'h1;
parameter  set2=4'h2;
parameter  set3=4'h3;
parameter  dat0=4'h4;
parameter  dat1=4'h5;
parameter  dat2=4'h6;
parameter  dat3=4'h7;
parameter  dat4=4'h8;
parameter  dat5=4'h9;

parameter  dat6=4'hA;
parameter  dat7=4'hB;
parameter  dat8=4'hC;
parameter  dat9=4'hD;
parameter  dat10=4'hE;
parameter  dat11=5'h10;
parameter  nul=4'hF;

assign LCD_N=0;
assign LCD_P=1;

always @(posedge clk)
begin
          counter<=counter+1;
if(counter==16'h000f)
clkr<=~clkr;
end
always @(posedge clkr)
begin
current<=next;
case(current)
   set0:   begin  rs<=0; dat<=8'h31; next<=set1; end
   set1:   begin  rs<=0; dat<=8'h0C; next<=set2; end
   set2:   begin  rs<=0; dat<=8'h6; next<=set3; end
   set3:   begin  rs<=0; dat<=8'h1; next<=dat0; end


   dat0:   begin  rs<=1; dat<="A"; next<=dat1; end
   dat1:   begin  rs<=1; dat<="D"; next<=dat2; end
   dat2:   begin  rs<=1; dat<="M"; next<=dat3; end
   dat3:   begin  rs<=1; dat<=" "; next<=dat4; end
   dat4:   begin  rs<=1; dat<="P"; next<=dat5; end
```

```
   dat5:   begin  rs<=1; dat<="V"; next<=dat6; end
   dat6:   begin  rs<=1; dat<="T"; next<=dat7; end
   dat7:   begin  rs<=1; dat<=" "; next<=dat8; end
   dat8:   begin  rs<=1; dat<="L"; next<=dat9; end
   dat9:   begin  rs<=1; dat<="T"; next<=dat10; end
   dat10:  begin  rs<=1; dat<="D"; next<=dat11; end
   dat11:  begin  rs<=1; dat<=" "; next<=nul; end

nul:   begin rs<=0;  dat<=8'h00;

if(cnt!=2'h2)
begin
                e<=0;next<=set0;cnt<=cnt+1;
end
else
begin next<=nul; e<=1;
end
end
default:   next<=set0;
endcase
end

assignen=clkr|e;
assignrw=0;
endmodule
```

#PINLOCK_BEGIN

NET "clk" LOC = P52; → Connect to 100K Clock Pin

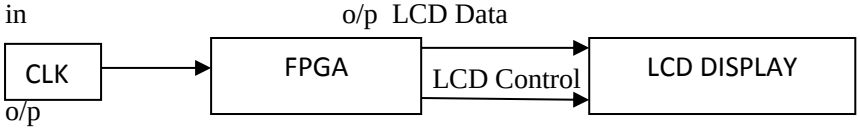
NET "en" LOC = P2; → Connect to LCD Control Pin - EN
NET "rs" LOC = P3; → Connect to LCD Control Pin - RS
NET "rw" LOC = P4; → Connect to LCD Control Pin - RW

NET "LCD_N" LOC = P7; - → Connect to GND
NET "LCD_P" LOC = P5; → Connect to +5V

NET "dat<0>" LOC = P31;
NET "dat<1>" LOC = P33;
NET "dat<2>" LOC = P34;
NET "dat<3>" LOC = P35;
NET "dat<4>" LOC = P36;
NET "dat<5>" LOC = P37;
NET "dat<6>" LOC = P39;
NET "dat<7>" LOC = P40;
→ Connect Header - H3 to LCD Data Pins

#PINLOCK_END

BLOCK DIAGRAM OF MOVING DISPLAY



EXPECTED VIVA QUESTION

1. Write a Verilog code to swap contents of two register with & without a temporary register?
2. Difference between task & function?
3. Difference between Inter statement &Intra statement delay ?
4. Difference between $Monitor & $Display
5. Tell me how blocking & non-blocking statements get executed ?
6. Variable & signal which will be updated first ?
7. What is sensitivity list ?
8. In a pure combinational circuit is it necessary to mention all the inputs is sensitivity disk ?if YES, why ?
9. Difference between Verilog &VHDL ?
10. Can you tell me some of system tasks & their purpose?
11. What is the difference between === & == ?
12. What is the difference between wire & register?
13. Data types, operators of Verilog & VHDL ?
14. What is the difference between simulation & synthesize ?
15. What is an event
16. What is the purpose of a for loop
17. Name any two programming tool of VHDL
18. What is the difference between concurrent & sequential statements?
19. What is an alias & write its syntax
20. What is the difference between array & record?
21. What are signal?
22. List out the four model for port in VERILOG
23. What do we need to generate hardware from VHDL model ?
24. Difference between FPGA &CPLD ?
25. Expansion of VHDL, FPGA, CPLD, ASIC
26. Number of pin in CPLD & FPGA
27. Device , family & speed of FPGA
28. Different styles of abstraction levels
29. Difference between always & initial
30. Difference between case, casex&casez
31. Difference between synchronous & asynchronous reset.