HDL LABORATORY

18ECL58

V SEMESTER

CBCS 2018 scheme

DEPARTMENT OF

ELECTRONICS & COMMUNICATION ENGINEERING

Prepared By

Prof. Naazneen M G
Assistant Professor
Department of ECE

Prof. Shaik Imam
Assistant Professor
Department of ECE

HKBK COLLEGE OF ENGINEERING

#22/1,Opp. Manyata Tech Park, Nagavara,

Bengaluru, Karnataka 560045

B. E. (EC / TC)
Choice Based Credit System (CBCS) and Outcome Based Education (OBE)
SEMESTER – V

HDL LABORATORY

Laboratory Code	**18ECL58**	**CIE Marks**	**40**
Number of Lecture Hours/Week	**02Hr Tutorial (Instructions)+ 02 Hours Laboratory**	**SEE Marks**	**60**
RBT Level	**L1, L2, L3**	**Exam Hours**	**03**

CREDITS – 02

Course Learning Objectives: This course will enable students to:

- Familiarize with the CAD tool to write Deprograms.
- 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/Modelsim or equivalent.

Laboratory Experiments

PART A : Programming

1. Write Verilog program for the following combinational design along with test 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 (behavioral model)
   c. 8 to 1 multiplexer using case statement and if statements
   d. 4-bit binary to gray converter using 1-bit gray to binary converter 1-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 behavior.

3. Verilog 32-bit ALU shown in figure below and verify the functionality of ALU by 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 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

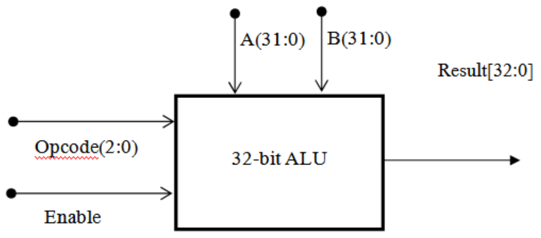


Figure 1 ALU top level block diagram
Table 1 ALU Functions

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	Logic OR	
111	A AND B	Logic 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 it works as clock divider performing division of clock by 2, 4, 8 and 16. Verify the functionality of the code.

PART-B : Interfacing and Debugging (EDWinXP, PSpice, MultiSim, Proteus, CircuitLab or any other equivalent tool can be used)

1. Write a Verilog code to design a clock divider circuit that generates 1/2, 1/3rd and 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 F KHz (eg. 200 KHz) 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 flops and 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.

B.E: Electronics & Communication Engineering

Program Outcomes (POs)

At the end of the B.E program, students are expected to have developed the following outcomes.

1. **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern Tool Usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The Engineer and Society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and Sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and Teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project Management and Finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one‘s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

B) PROGRAM SPECIFIC OUTCOMES (PSOs)

1. **Professional Skills:** An ability to understand the basic concepts in Electronics & Communication Engineering and to apply them to various areas, like Electronics, Communications, Signal processing, VLSI, Embedded systems etc., in the design and implementation of complex systems.

2. **Problem-Solving Skills:** An ability to solve complex Electronics and communication Engineering problems, using latest hardware and software tools, along with analytical skills to arrive cost effective and appropriate solutions.

3. **Entrepreneur:** An ability to become an entrepreneur or to contribute to industrial services and / or Govt. Organizations in the field of Electronics and Communication Engg.

4. **Multidisciplinary Programming**: An ability to work on multidisciplinary teams with efficiency indifferent Programming techniques.

Procedure for Xilinx ISE Design

1. Click on the ISE Design suit icon on Desktop.
2. On the toolbar select file menu and select the new project.
3. Create the folder in the working directory.
4. Enter the project name (should be same as the entity name)
5. Click next.
6. Select the family as SPARTAN6, package TQG144, speed -3
   Available deviceXC6SLX4.
7. Click next and finish.
8. On the Hierarchy select the XC6SLX4 , right click and select New Source.
9. Select Verilog Module and give a file name and write the code.
10. Click Next and type the code.
11. Select Implementation in view and click on synthesize-XST.
12. Correct bugs if any.

Procedure for simulation

1. On the Hierarchy select the XC6SLX4, right click and select New Source.
2. Select Verilog Test Fixture and give a file name and write the testbench.
3. Click Next and type the test benchcode.
4. Select Simulation in view and click on Behavioral Check Syntax.
5. Correct bugs if any in the test code.
6. Click on Simulate Behavioral Model and observe the Waveform.

Procedure for downloading program on to Spartan6 FPGA kit

1. Select Implementation in view, In User Constraints, click on I/O Pin Planning (Plan Ahead-Post Synthesis).
2. Do the Pin assignment.
3. Click on Configure Target Device.
4. Double Click on Boundary Scan.
5. Right Click and select Initialize Chain.
6. Select Yes and then No for other device.
7. Cancel the .mgc file and select the .bitfile.
8. Right Click on FPGA Device and Program.
9. Observe the output on Spartan6 FPGA Device.

EXPERIMENT: 1 2:4 DECODER

Aim: Write Verilog program for 2 to 4 decoder realization using NAND gates only (structural model) along with test bench to verify the design:

THEORY:

A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. A binary code of n bits is capable of representing up to 2^n distinct elements of the coded information.

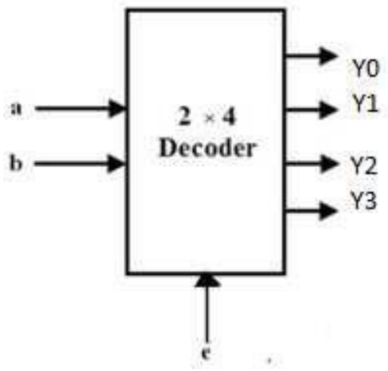
A 2-to-4-line decoder, the two inputs are decoded into four outputs, each output representing one of the min-terms of the 2-input variables. A 2-to-4 decoder can be used for decoding any 2 –bit code to provide 4-outputs, one for each element of the code.

2 to 4 decoder: A decoder is a digital logic circuit that converts n-bits binary input code in to M output lines. OR It is a logic circuit that decodes from binary to octal, decimal, Hexadecimal or any other code such as 7-segment etc.

Truth Table:

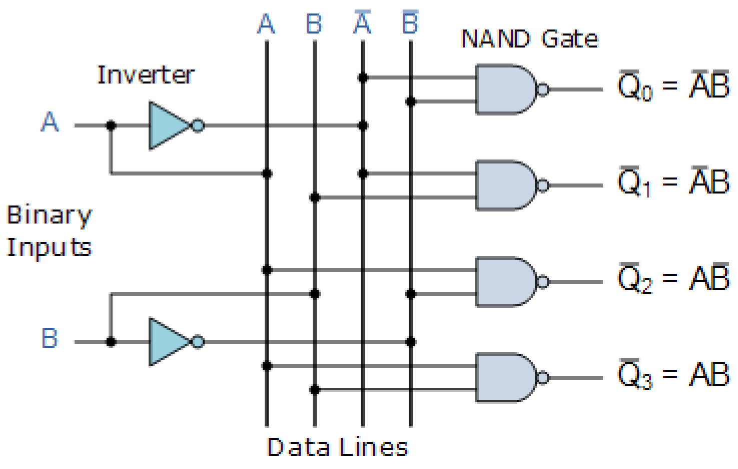
INPUTS		OUTPUTS			
Data Select I/P's					
a	b	Y0	Y1	Y2	Y3
X	X	1	1	1	1
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Block Diagram:



Verilog HDL code for 2:4 decoder

```
module dec24(a,b,y);
input a,b;
output [3:0] y;
wire na,nb;
nand  g1 (na , a);
nand  g2 (nb , b);
nand g3 (y[0], na, nb);
nand  g4 (y[1], na, b);
nand  g5 (y[2], a, nb);
```

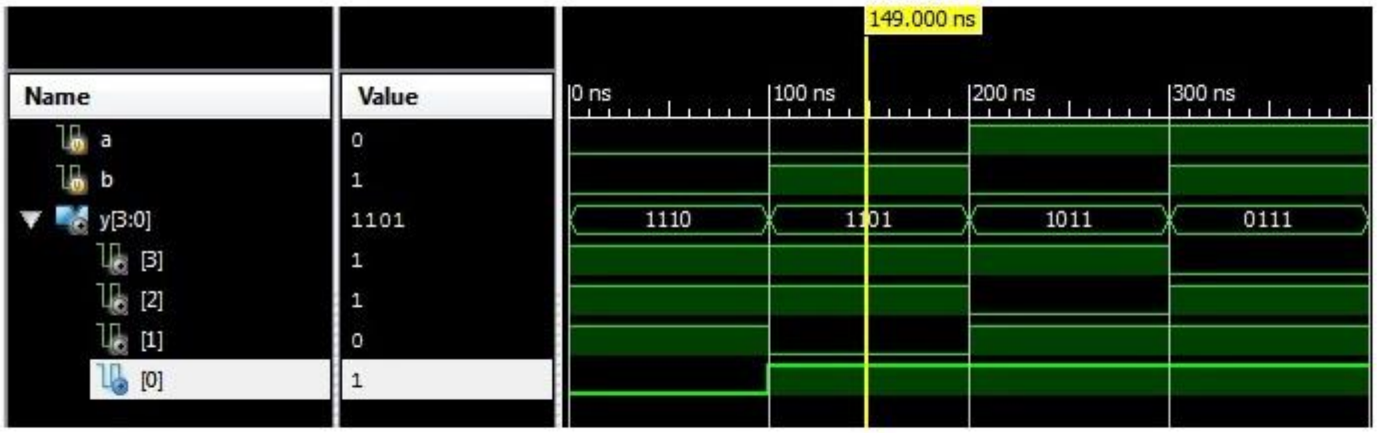


A
B
$\overline{A}$
$\overline{B}$
NAND Gate
Inverter
A
Binary Inputs
B
$\overline{Q_0} = \overline{A}\,\overline{B}$
$\overline{Q_1} = \overline{A}B$
$\overline{Q_2} = A\overline{B}$
$\overline{Q_3} = AB$
Data Lines

```
nand  g6 (y[3],a, b);
endmodule
```

Test Bench for decoder

```
module dec24_t;
reg a,b;
wire [3:0] y;

module dec24 uut (.a(a),.b(b),.y(y));

initial
begin
a=0; b=0; #100;
a=0; b=1; #100;
a=1; b=0; #100;
a=1; b=1; #100;
end
endmodule
```



RESULT: The 2 to 4 decoder design is realized and simulated using HDL codes successfully.

APPLICATIONS:

Decoding is necessary in applications such as data multiplexing, 7 segment display and memory address decoding, wireless control systems.

EXPERIMENT: 2 8 to 3 ENCODER

AIM: Write Verilog program for 8 to 3 encoder with priority and without priority (behavioral model) along with test bench to verify the design:

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 output lines generate the binary code for 2^n input variables. 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.

Encoder: The term 'encode' specifies the conversion of information (number or character) into a coded form. An encoder is a combinational logic circuit that converts information such as a decimal number or an alphabetic character, into some coded form. An encoder accepts an active level on one of its inputs representing a digit, such as decimal or octal digit, and converts it to a coded output, such as binary or BCD.

Without Priority Encoder:

These encoders establish an input priority to ensure that only the set input line is encoded. For example 8-to-3-line priority encoder the inputs are I(7) ,I(6),…….I(0).If the input set is I(4) as logic-1 and all other inputs as logic-0, the output will be 100 because I(4) is the only input set and the other are no other choices.

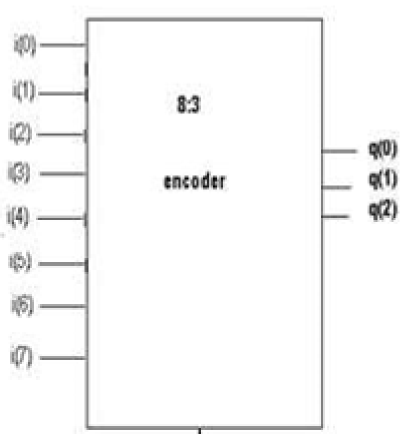
Priority Encoder:

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(7) ,I(6),…….I(0).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).

Truth Table for without priority Encoder

Inputs								Output		
I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Q_2	Q_1	Q_0
X	X	X	X	X	X	X	X	0	0	0
0	0	0	0	0	0	0	0	0	0	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

Fig: Block Diagram

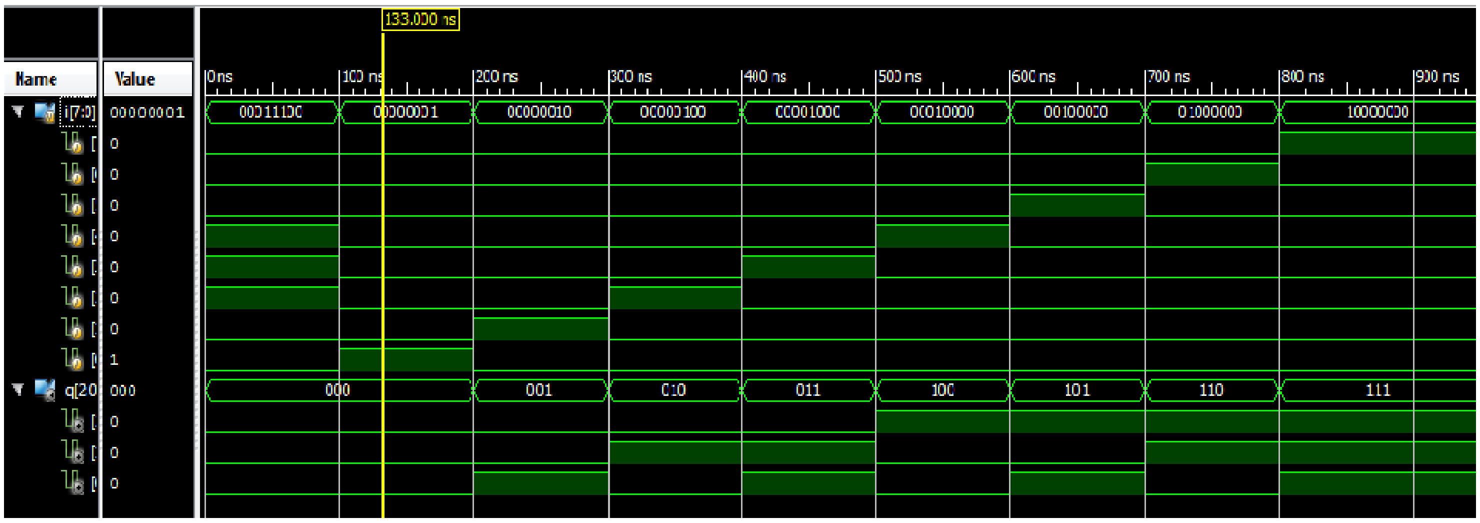


2a.Verilog Code to Implement Encoder Without Priority.

```
module enc83wop (i,q);

input [7:0] i;
output reg  [2:0] q;
always@ (i)
 begin

  case (i)
     8’b00000001: q=3’b000;
     8’b00000010: q=3’b001;
     8’b00000100: q=3’b010;
     8’b00001000: q=3’b011;
     8’b00010000: q=3’b100;
     8’b00100000: q=3’b101;
     8’b01000000: q=3’b110;
     8’b10000000: q=3’b111;
     default:q=3’b000;
 endcase
 end
endmodule
```

Test Bench for Without Priority Encoder

```
module enc83wop_t;
reg [7:0] i;
wire [2:0] q;
enc83wop uut ( .i(i), .q(q) );

initial
begin
i = 8'b00000001; #100;
i = 8'b00000010; #100;
i = 8'b00000100; #100;
i = 8'b00001000; #100;
i = 8'b00010000; #100;
i = 8'b00100000; #100;
i = 8'b01000000; #100;
i = 8'b10000000; #100;
end
endmodule
```



Truth Table for with Priority Encoder

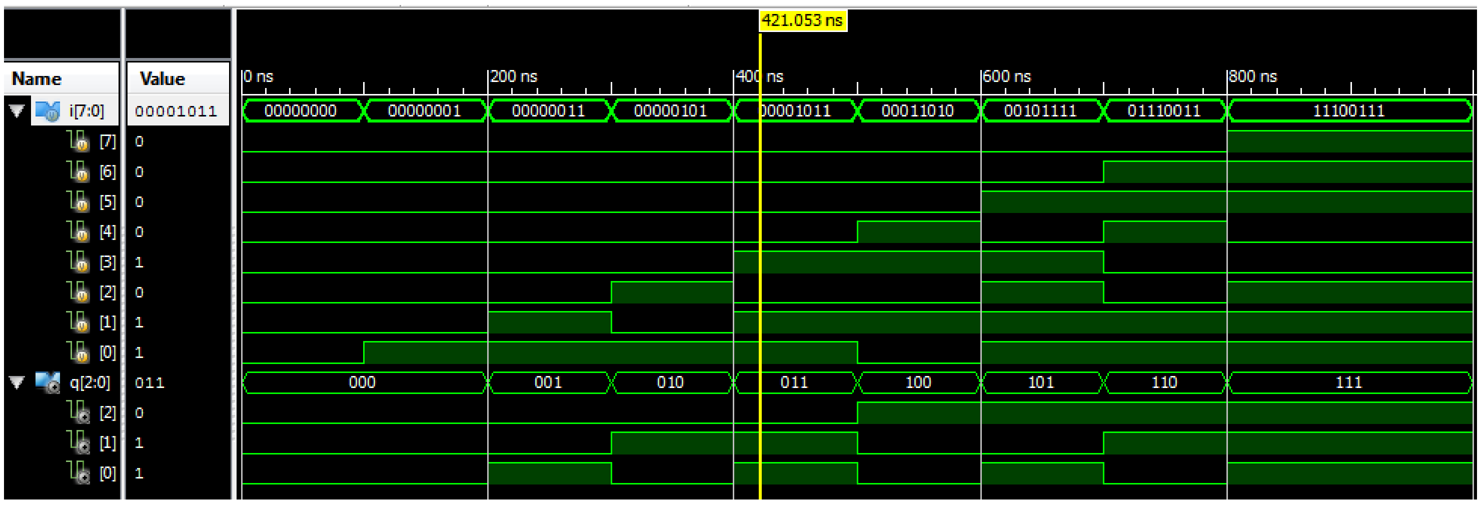
Inputs								Outputs		
I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	0	z	z	z
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

2b. Verilog Code to Implement Encoder with Priority.

```
module enc83wp(i,q);
input [7:0] i;
output reg  [2:0] q;
always@ (i)
begin
    casex (i)
        8’ b00000001:    q=3’b000;
        8’ b0000001x:    q=3’b001;
        8’ b000001xx:    q=3’b010;
        8’ b00001xxx:    q=3’b011;
        8’  b0001xxxx:  q=3’b100;
        8’  b001xxxxx:  q=3’b101;
        8’  b01xxxxxx:  q=3’b110;
        8’ b1xxxxxxx:   q=3’b111;
        default:            q=3’bxxx;
    endcase
end
endmodule
```

Test Bench for high Priority Encoder

```
module enc83wp _t;
        reg [7:0] i;
        wire [2:0] q;
        enc83wp uut ( .i(i), .q(q) );
        initial
         begin
                i = 8'b00000000;  #100;
                i = 8'b00000001;  #100;
                i = 8'b00000011;  #100;
                i = 8'b00000101;  #100;
                i = 8'b00001011;  #100;
                i = 8'b00011010;  #100;
                i = 8'b00101111;  #100;
                i = 8'b01110011;  #100;
                i = 8'b11100111;  #100;
                end
                endmodule
```



RESULT : The 8 to 3 encoder design is realized and simulated using verilog code successfully.

APPLICATIONS :

Encoding is used in most wireless control systems to prevent interference. It is useful in web processes, handling and inspection systems that use conveyors and simple speed or position control in high vibration environments.

EXPERIMENT: 3 8:1 MULTIPLEXER

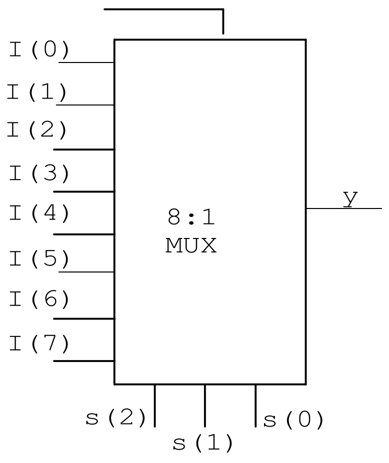
Aim: Write Verilog program for 8 to 1 multiplexer using case statement and if statements along with test Bench to verify the design:

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.

Multiplexer is simply a data selector. It has multiple inputs and one output. Any one of the input line is transferred to output depending on the control signal. This type of operation is usually referred as multiplexing .In 8:1 multiplexer, there are 8 inputs. Any of these inputs are transferring to output, which depends on the control signal. For 8 inputs we need, 3 bit wide control signal. If control signal is "000",then the first input is transferring to output line. If control signal is "111", then the last input is transferring to output. Similarly for all values of control signals**.**

TRUTH TABLE:

Inputs				output
e	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]

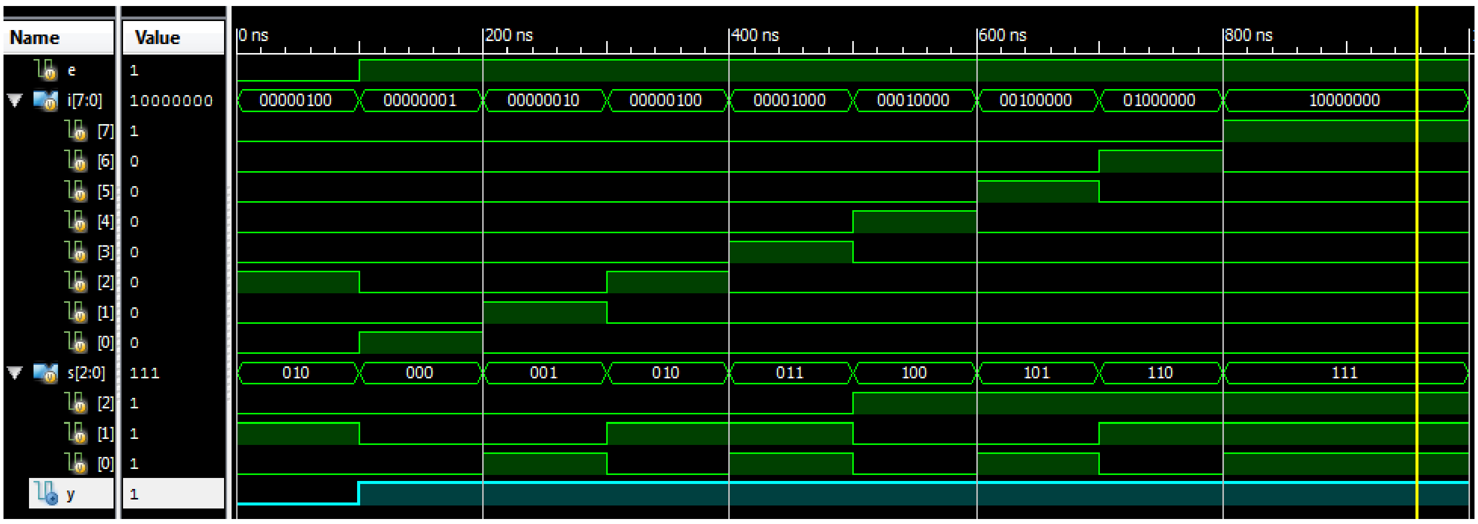


Verilog HDL code for 8:1 Mux

```
module mux8 ((i,s,e,y);
input [7:0]i;
input[2:0]s;
input e;
output reg y;

always@ (i, e)
begin
if(e ==1)

            case(s)

            3' b000: y=i [0];
            3' b001: y=i [1];
            3' b010: y=i [2];
            3' b011: y= i [3];
            3' b100: y= i [4];
            3' b101: y= i [5];
            3' b110: y= i [6];
            3' b111: y= i [7];
            endcase
 else
y=0;
end
endmodule
```

Test Code for 8:1 Mux

```
module mux8_t;
reg [7:0] i;
reg[2:0] s;
reg e;
wirey;
mux8 uut ((.i(i),.s(s),.e(e),.y(y));
initial
begin

e=0;  s=3'b010;  i=8'b00000100;  #100;
e=1;  s=3'b000;  i=8'b00000001;  #100;
e=1;  s=3'b001;  i=8'b00000010;  #100;
e=1;  s=3'b010;  i=8'b00000100;  #100;
e=1;  s=3'b011;  i=8'b00001000;  #100;
e=1;  s=3'b100;  i=8'b00010000;  #100;
e=1;  s=3'b101;  i=8'b00100000;  #100;
e=1;  s=3'b110;  i=8'b01000000;  #100;
e=1; s=3'b111;   i=8'b10000000; #100;
end
endmodule
```



RESULT : The 8 to 1 multiplexer design is realized and simulated using HDL codes successfully.

EXPERIMENT: 4 4-BIT BINARY TO GRAY CODECONVERTION

AIM: To write a Verilog HDL code for a 4 bit Binary to Gray code converter and download the programs on a Spartan 6 FPGA board.

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.

The Gray code is a binary numeral system where two successive values differ in only one bit. The gray code is sometimes referred to as reflected binary. It is a non-weighted code; therefore, it is not a suitable for arithmetic operations. It is a cyclic code because successive code words in this code differ in one bit position only i.e. it is a unit distance code

TRUTH TABLE:

Binary inputs				Gray code o /p’s			
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

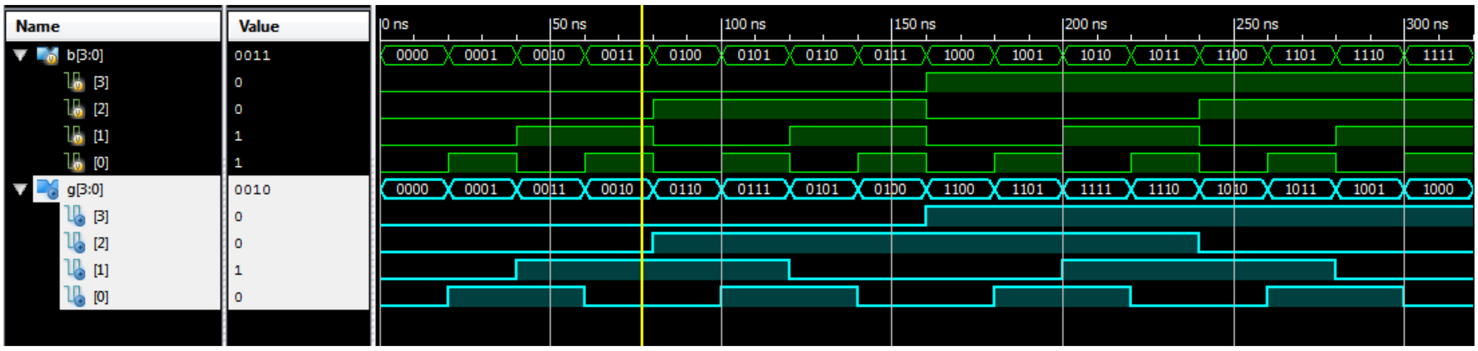
SIMPLIFIED EXPRESSIONS: $G_0 = B_0 \oplus B_1$, $G1 = B_1 \oplus B_2$, $G2 = B_2 \oplus B_3$, $G_3 = B_3$

Verilog HDL code for conversion of 4 bit Binary to Gray

```
module bingry (b,g);
input [3:0] b;
output [3:0] g;
assigng[0]=b[0]^b[1];
assigng[1]=b[1]^b[2];
assigng[2]=b[2]^b[3];
assign g[3]=b[3];
endmodule
```

Test Bench

```
    module bingry_t(b,g);
    reg [3:0] b;
    wire [3:0] g;
    bingry uut (.b(b), .g(g) );
    initial
        begin
        b=4’b0000;  #20;
        b=4’b0001;  #20;
        b=4’b0010;  #20;
        b=4’b0011;  #20;
        b=4’b0100;  #20;
        b=4’b0101;  #20;
        b=4’b0110;  #20;
        b=4’b0111;  #20;
        b=4’b1000;  #20;
        b=4’b1001;  #20;
        b=4’b1010;  #20;
        b=4’b1011;  #20;
        b=4’b1100;  #20;
        b=4’b1101;  #20;
        b=4’b1110;  #20;
        b=4’b1111;  #20;
        end
endmodule
```



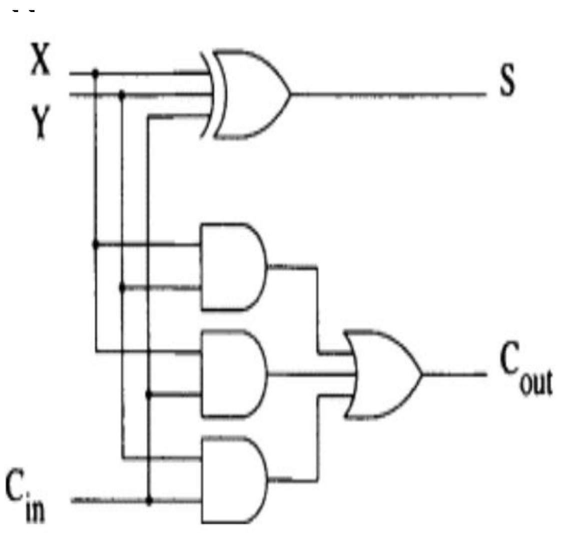
RESULT: The 4 bit binary to gray converter design have been realized and simulated using HDL codes**.**

APPLICATIONS: Gray codes are widely used to facilitate error correction in digital communications such as digital terrestrial television and some cable TV systems.

EXPERIMENT: 5 **Realization of logic gates using Full Adder**

5. 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.

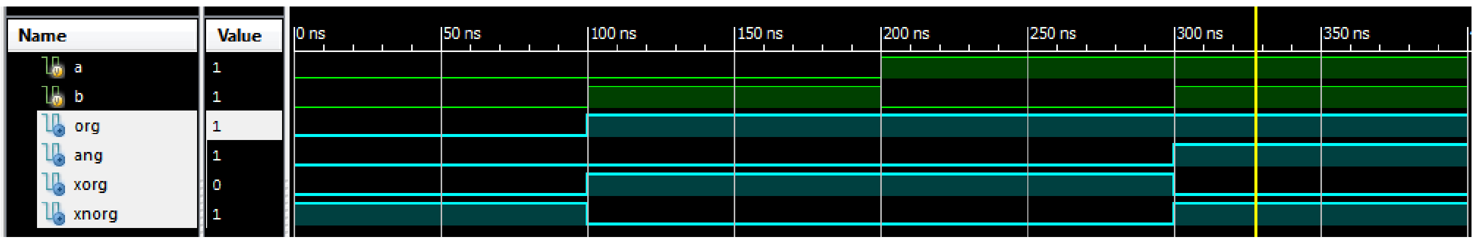
Verilog HDL code for realization of logic gates using ful



```
module gates_adder(a,b,xorg,xnorg,ang,org);
input a,b;
output reg xorg,xnorg,ang,org;
always@(a,b)
begin
fulladder(xorg,ang,a,b,0);
fulladder(xnorg,org,a,b,1);
end

task fulladder;
output sum,cout;
input a,b,c;
begin
sum=a^b^c;
cout=a&b|b&c|c&a;
end
endtask
endmodule
```

Test Bench

```
module gates_adder_t;
reg a,b;
wire xorg,xnorg,ang,org;
gates_adder uut(.a(a),.b(b),.xorg(xorg),.xnorg(xnorg),.ang(ang),.org(org));
initial
begin
a=0;b=0;#100;
a=0;b=1;#100;
a=1;b=0;#100;
a=1;b=1;#100;
end
endmodule
```



RESULT: The full adder functionality as logic gates is realized and simulated using HDL codes successfully

APPLICATIONS: Adders and Subtractor can be used in op amp circuits, that is as comparators or differentiators. Arithmetic operations are extensively used in many VLSI applications such as signal processing, and digital communication. Adders are basically used in calculators. They are used in all processors – microprocessors and microcontrollers and also DSP processors.

EXPERIMENT 6: 32-BIT ALU

Verilog 32-bit ALU shown in figure below and verify the functionality of ALU by 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 all possible 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 is completed

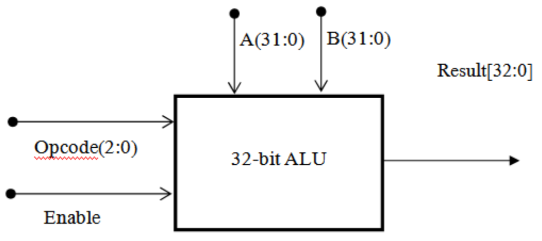


Figure 1 ALU top level block diagram
Table 1 ALU Functions

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	Logic OR	
111	A AND B	Logic AND	

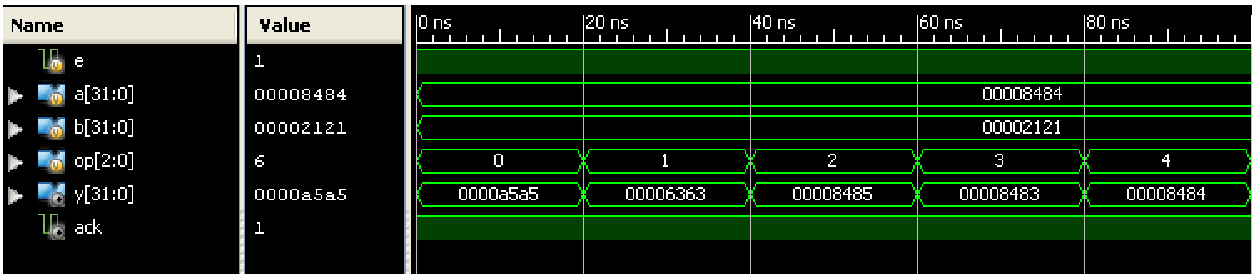
Verilog HDL code for 32 bit ALU

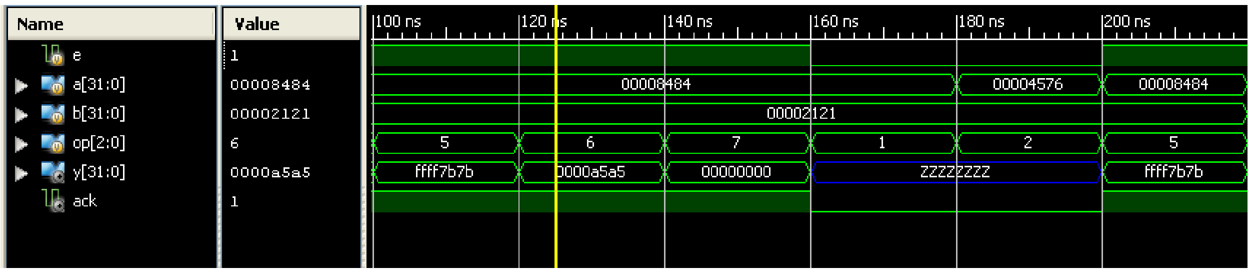
```
module alu32(a,b,op,e,y,ack);
input [31:0] a,b;
input [2:0] op;
input e;
output reg [31:0] y;
output  reg ack;
always @(a,b,op,e)
begin
   ack=0;
   if(e==1)
         begin
         case (op)
             4'd0:  y=a+b;
             4'd1:   y=a-b;
             4'd2:y=a+1;
             4'd3: y=a-1;
             4'd4: y=a;
             4'd5: y=~a;
             4'd6: y=a | b;
             4'd7: y=a & b;
          endcase
          ack=1;
```

```
            end
         else
            y=32'bZ;
     end
  endmodule
```

Test Bench ALU32

```
module alu32_t;
        reg [31:0]a;
        reg [31:0]b;
        reg [3:0] op;
        reg e;

        wire [31:0] y;
        wire ack;
              ALU32 uut ( .a(a), .b(b), .op(op), .e(e), .y(y),.ack(ack) );
               initial
              begin
                a = 32'h8484; b = 32'h2121; op = 4'd0;  e = 1;  #20;
                a = 32'h8484; b = 32'h2121; op = 4'd1;  e = 1;  #20;
                a = 32'h8484; b = 32'h2121; op = 4'd2;  e = 1;  #20;
                a = 32'h8484; b = 32'h2121; op = 4'd3;  e = 1;  #20;
                a = 32'h8484; b = 32'h2121; op = 4'd4;  e = 1;  #20;
                a = 32'h8484; b = 32'h2121; op = 4'd5;  e = 1;  #20;
                a = 32'h8484; b = 32'h2121; op = 4'd6;  e = 1;  #20;
                a = 32'h8484; b = 32'h2121; op = 4'd7;  e = 1;  #20;
                a = 32'h8484; b = 32'h2121; op = 4'd1;  e = 0;  #20;
                a = 32'h8484; b = 32'h1212; op = 4'd1;  e = 0;  #20;
                a = 32'h8484; b = 32'h1111; op = 4'd1;  e = 1;  #20;
                end
                endmodule
```





RESULT: 32-bit ALU design is realized and simulated using HDL codes successfully

EXPERIMENT 7: FLIP FLOPS

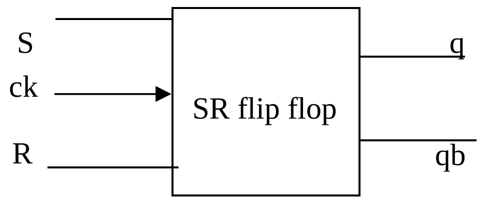
Write Verilog code for SR, D and JK and verify the flip flop.

Flip-flop: Flip-flop is a sequential logic circuit, which is 'One '-bit memory element. OR It is a basic memory element in digital systems (same as the bi-stable multivibrator) It has two stable state logic '1' and logic '0'.

D Flip-Flop: In D-Flip-flop the transfer of data from the input to the Output is delayed and hence the name delay D-Flip-flop. The D-Type Flip-flop is either used as a delay device or as a latch to store '1' bit of binary information. D input transferred to Q output when clock asserted

SR Flip-flop truth table:

clk	S	R	Qn+1
↑	0	0	q
↑	0	1	0
↑	1	0	1
↑	1	1	Z
0/1	X	X	q

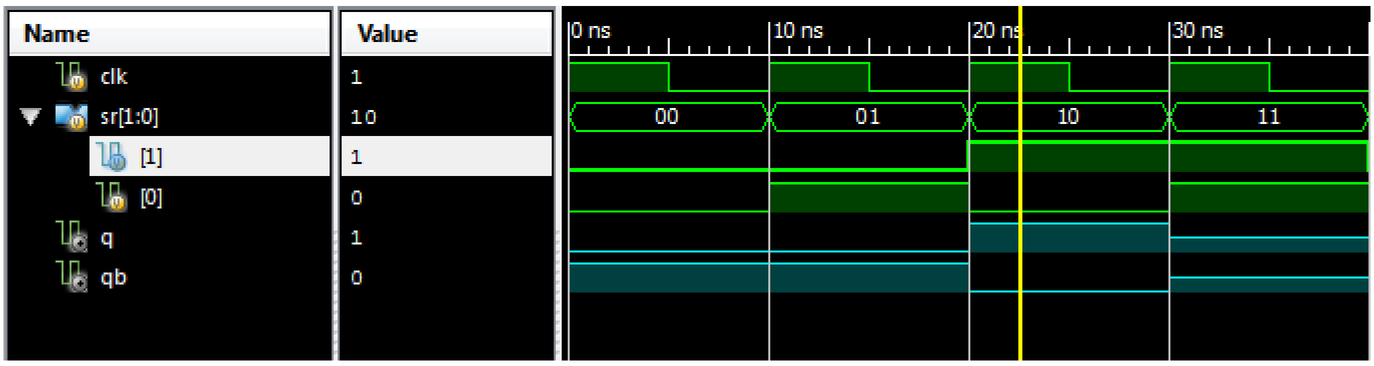


```
module srff(clk,sr,q,qb);
input clk;
input[1:0] sr;
output reg q=0,qb=1;

always @(posedge clk)
begin
  case(sr)
     2'b00:q=q;
     2'b01:q=0;
     2'b10:q=1;
     2'b11:q=1'bz;
  default :q=0;
  endcase
qb=~q;
end
endmodule
```

Test Bench SR Flip Flop

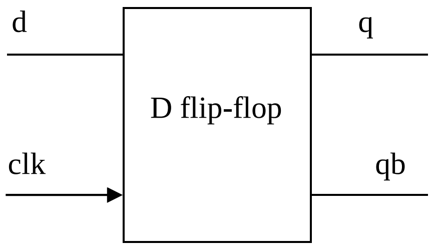
```
modulesrff_t;

            regclk;
            reg [1:0] sr;
```

```
wire q, qb;

srff  uut (.clk(clk), .sr(sr),.q(q), .qb(qb) );

initial
clk = 1;
always  #5      clk=~clk;

initial
sr[0] = 0;
always #10 sr[0]=~sr[0];

initial
sr[1] = 0;
always #20 sr[1]=~sr[1];
endmodule
```



D flip flop truth table

Clk	D	Q n+1
$\uparrow$	0	0
$\uparrow$	1	1
0/1	X	q



When clk=1 Q=d and otherwise
Q retains its previous state

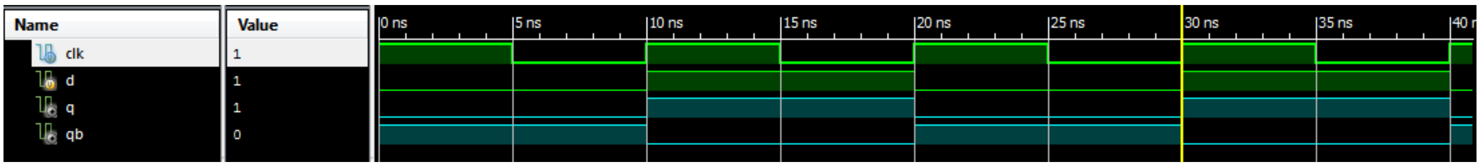
b.Verilog HDL code for dff

```
module dff(clk,d,q,qb);
input clk,d;
output reg q=0,qb=1;
```

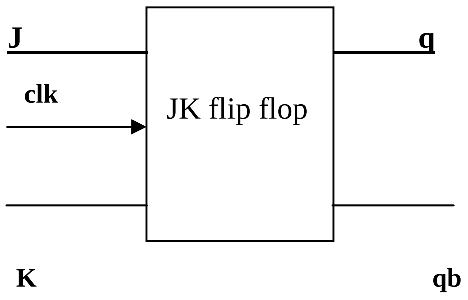
```
always @(posedge clk)
begin
  case(d)
        0:q=0;
        1:q=1;
  default: q=0;
  endcase
qb=~q;
end
endmodule
```

Test Bench D Flip Flop

```
moduledff_t;

        regclk , d;

        wire q , wireqb;

        dff  uut ( .d(d), .clk(clk), .q(q), .qb(qb) );
        initial
        clk = 1;
        always    #5    clk=~clk;

        initial
        d = 0;
        always  #10     d=~d;

    endmodule
```



<u>J K Flip-flop truth table</u>



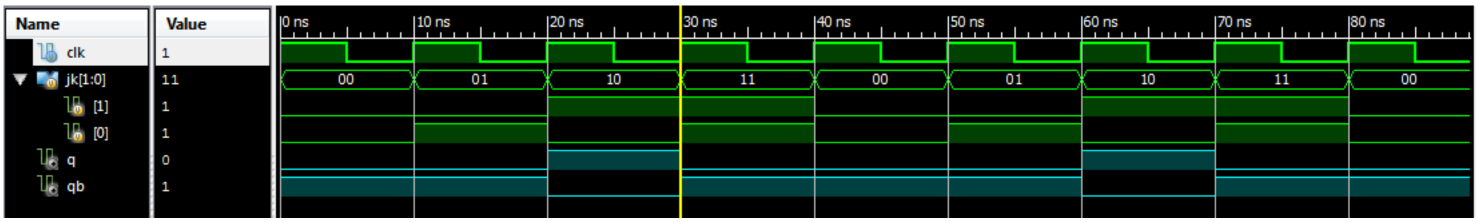
clk	J	K	Qn+1
↑	0	0	q
↑	0	1	0
↑	1	0	1
↑	1	1	q
0/1	X	X	q

Verilog HDL code for JK-ff

```
module jkff ( clk,jk,q,qb);
input clk;
input[1:0] jk;
output regq=0,qb=1;

always @(posedge clk)
 begin
      case(jk)
       2'b00: q=q;
      2'b01: q=0;
      2'b10: q=1;
      2'b11: q=~q;
      default :q=0;
      endcase
    qb=~q;
    end
endmodule
```

Test Bench JK Flip Flop

```
module jkff_t;

            reg clk;
            reg [1:0] jk;

            wire q, qb;

            jkff uut (.clk(clk), .jk(jk),.q(q), .qb(qb) );

             initial
            clk =1;
            always  #5        clk=~clk;

            initial
            jk[0] = 0;
            always#10       jk[0]=~jk[0];

            jk[1] = 0;
            always  #20     jk[1]=~jk[1];

endmodule
```



RESULT: SRFF, DFF and JKFF design is realized and simulated using HDL codes successfully

EXPERIMENT 9: 4-BIT BCD SYNCHRONOUS COUNTER

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

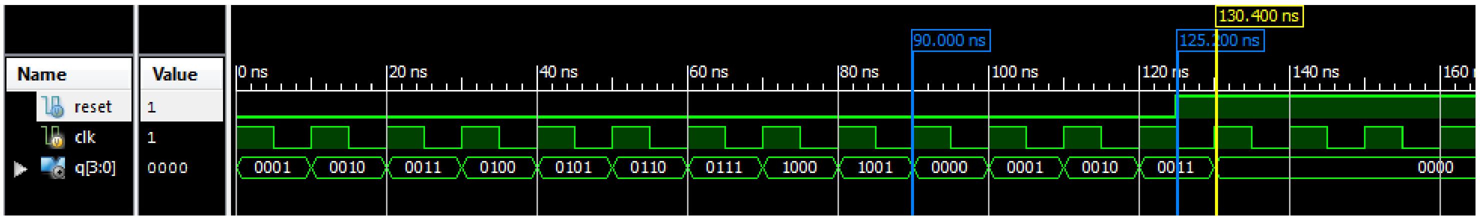
COUNTER: Counter is a digital circuit that can counts the member of pulse for building the counters, Flip-flop are used. Relation between number of Flip-flop used and number of state of counter is (Regular/binary counter

reset	**Count clk pulse**	**OUT PUT COUNT SEQUENCE**			
		Q_3	**Q_2**	**Q_1**	**Q_0**
	initial	0	0	0	0
0	↑	0	0	0	1
0	↑	0	0	1	0
0	↑	0	0	1	1
0	↑	0	1	0	0
0	↑	0	1	0	1
0	↑	0	1	1	0
0	↑	0	1	1	1
0	↑	1	0	0	0
0	↑	1	0	0	1
0	↑	0	0	0	0
0	↑	0	0	0	1
0	↑	0	0	1	0
1	↑	0	0	0	0

```
module bcdsyncnt (clk,q, reset);
input clk , reset;
output reg [3:0] q=4’b0000;

always @(posedge clk)
begin
  if (reset==1) q=4'b0000;
  else if (q==4'b1001) q=4'b0000;
  else
  q=q+1;
end
endmodule
```

TEST BNCH FOR BCD COUNTERS

```
module bcdsyncnt_t ;
regclk , reset;

wire [3:0]q;

bcdsyncnt uut (.clk(clk), .reset(reset),.q(q)  );

initial
clk=1;
always    #5        clk=~clk;

initial
reset =0;
always  #100       reset=~reset;
endmodule
```



RESULT: 4-bit BCD Synchronous Counter design is realized and simulated using HDL codes successfully

EXPERIMENT 10: COUNTER AS CLOCK DIVIDER CIRCUIT

10.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.

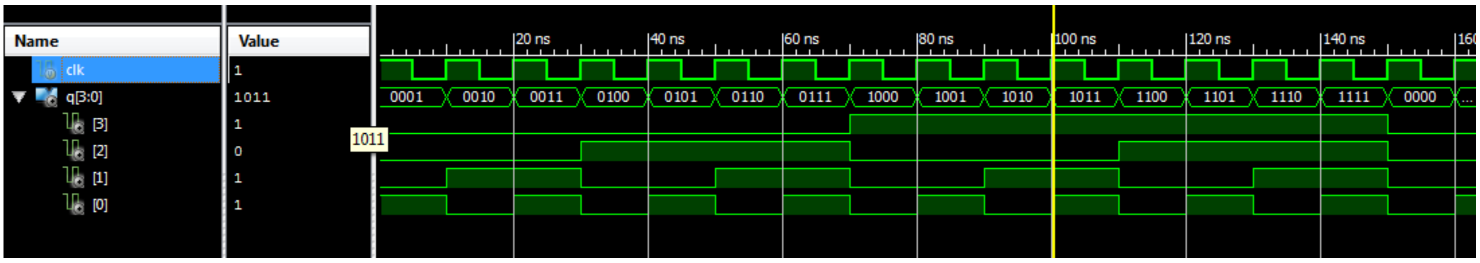
Count clk pulse	**OUT PUT COUNT SEQUENCE**			
	Q_3	**Q_2**	**Q_1**	**Q_0**
initial	0	0	0	0
↑	0	0	0	1
↑	0	0	1	0
↑	0	0	1	1
↑	0	1	0	0
↑	0	1	0	1
↑	0	1	1	0
↑	0	1	1	1
↑	1	0	0	0
↑	1	0	0	1
↑	1	0	1	0
↑	1	0	1	1
↑	1	1	0	0
↑	1	1	0	1
↑	1	1	1	0
↑	1	1	1	1
↑	0	0	0	0

```
module bincnt (clk,q, reset);
input clk , reset;
output reg [3:0] q=4’b0000;


always @(posedge clk)
begin
  if(q=4’b1111)
  q=4’b0000;
  else
  q=q+1;
end
endmodule
```

TEST BENCH FOR BINARY COUNTER

```
module bincnt_t ;
```

```
reg clk;
wire [3:0]q;

bincnt  uut  (.clk(clk), .q(q));
initial
clk=1;
always #5 clk=~clk;

endmodule
```



Input Clock Signal Time Period =
Input Clock Signal Frequency   =

Clock Signal	**Clock Period**	**Clock Frequency**
Q3		
Q2		
Q1		
Q0		

RESULT: Counter as clock divider circuit design is realized and simulated using HDL codes successfully

PART-B

INTERFACING PROGRAMS

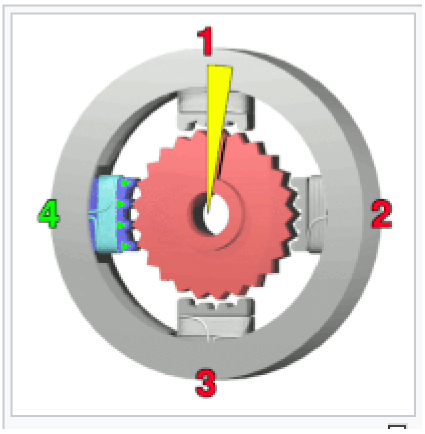
EXPERIMENT1: STEPPERMOTOR

AIM: Write Verilog HDL code to control speed, direction of Stepper motor

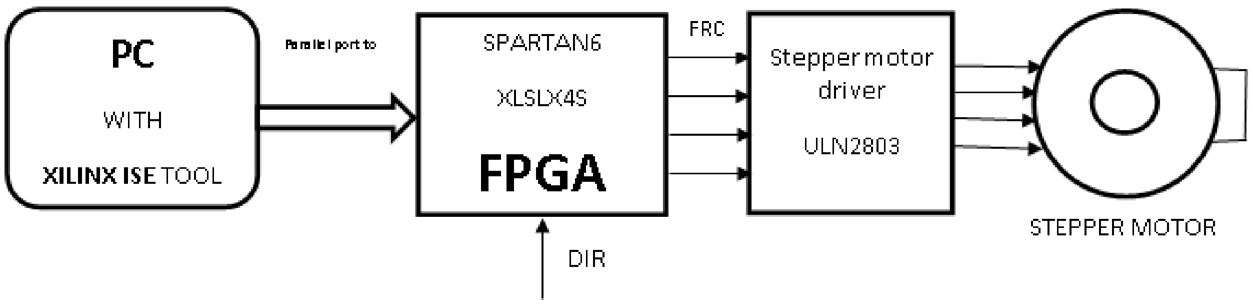
Hardware and software Requirements:
Hardware: Spartan 6 FPGA Board, Stepper motor driver, Stepper Motor, RPS, FRC, PPI to JTAG interface, PC
Software: Xilinx ISE 14.7

Theory:

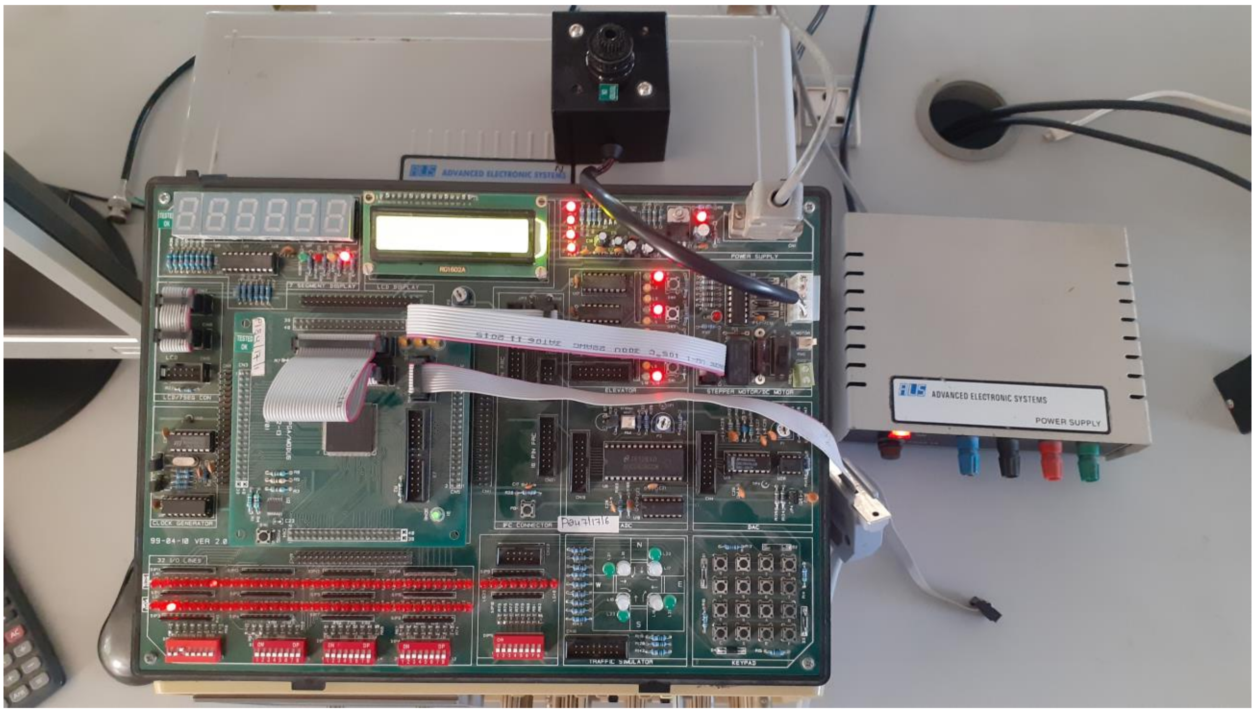
A stepper motor is a digital motor. It can be driven by digital signal. The motor has two phase with center tap winding. The centre taps of these windings are connected to the 12V supply. Due to this motor can be excited by grounding four terminals of the two windings.
Motor can be rotated in steps by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



Interfacing Block Diagram:



```
module stmot(clk,sw,dout);
input clk;
input [2:0] sw;
output [3:0] dout;

reg [3:0] temp=4’b0111;

reg [21:0] cnt = 22’d0;
reg tclk;

always @(posedge clk)
begin
cnt=cnt+1;
tclk=cnt[21];
end

always (posedge tclk)
integer i=0,j=0,k=0;
begin
if (sw==3’b100 && i<N)
  begin
  temp = { temp[0] ,temp[3:1]};              --clockwise N rotation
  i=i+1;
  end
if (sw==3’b010 && j<N/2)
  begin
   temp = { temp[0] ,temp[3:1]};          --clockwise N/2 rotation
   j=j+1;
   end
if (sw==3’b001&& k<N)
  begin
   temp = { temp[2:0] ,temp[3]};            --anticlockwise  N rotation
   k=k+1;
   end
end
assign dout = temp;

endmodule
```



PIN DETAILS STEPPER MOTOR

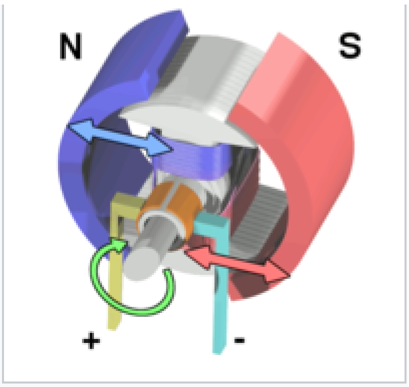
```
NET"clk"           LOC = "p51" | IOSTANDARD = LVTTL ;
NET "sw<0>"        LOC = "p22" | IOSTANDARD = LVTTL ;
NET"sw<1>"         LOC = "p24" | IOSTANDARD = LVTTL ;
NET"sw<2>"         LOC = "p27" | IOSTANDARD = LVTTL ;
NET "dout<0>"  LOC = "p114" | IOSTANDARD = LVTTL ;
NET "dout<1>"  LOC = "p117" | IOSTANDARD = LVTTL ;
NET "dout<2>"  LOC = "p118" | IOSTANDARD = LVTTL ;
NET "dout<3>"  LOC = "p119" | IOSTANDARD = LVTTL;
```

EXPERIMENT2: **DC MOTOR**

AIM: Write Verilog HDL code to control speed, direction of DC motor

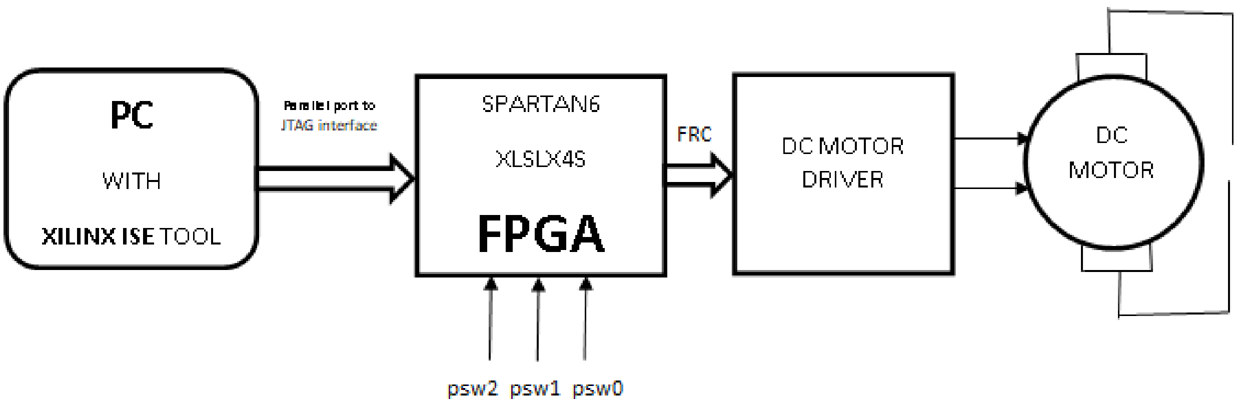
Hardware and software Requirements:
Hardware: Spartan 6 FPGA Board, DC motor driver, DC Motor, RPS, FRC, PPI to JTAG interface, PC
Software: Xilinx ISE14.7

Theory:
A DC motor is a rotary electrical machine 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.



Workings of a brushed electric motor with a two-pole rotor (armature) and permanent magnet stator. "N" and "S" designate polarities on the inside axis faces of the magnets; the outside faces have opposite polarities. The + and - signs show where the DC current is applied to the commutator which supplies current to the armature coils

Interfacing Block Diagram:



```
module dcmot(sw,pdm,clk);
input clk;
input [2:0] sw;
output reg pdcm;

reg[11:0] clkdiv=12'h0;
reg [7:0] cnt=8'd0;
reg tclk;


always@(posedge clk)
begin
cnt=cnt+1;
tclk=cnt[7];
end

always@(posedge clk)
begin
    clkdiv=clkdiv+1;
  if (clkdiv==12'hBB8)
    clkdiv=12'h000;
 end

always@(posedge tclk)
begin
if(clkdiv==12'h0) pdcm=1;
else if (sw==3'h0 && clkdiv==12'h1f4) pdcm=0;
else if (sw==3'h1 && clkdiv==12'h320) pdcm=0;
else if (sw==3'h2 && clkdiv==12'h44c) pdcm=0;
else if (sw==3'h3 && clkdiv==12'h578) pdcm=0;
else if (sw==3'h4 && clkdiv==12'h6a4) pdcm=0;
else if (sw==3'h5 && clkdiv==12'h7d0) pdcm=0;
else if (sw==3'h6 && clkdiv==12'h8fc) pdcm=0;
else if (sw==3'h7 && clkdiv==12'h9c4) pdcm=0;
end
endmodule
```

PIN DETAILS - DC MOTOR

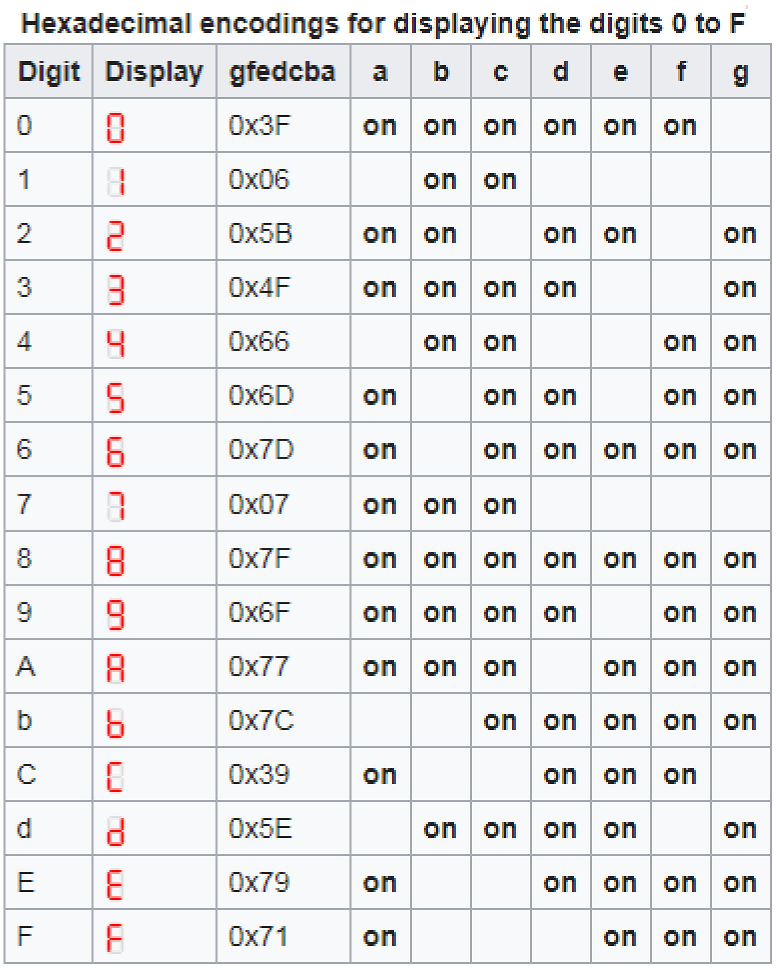
```
#PACE: Start of PACE I/O Pin Assignments
NET  "clk"     LOC  =  "p51"     | IOSTANDARD  =  LVTTL  ;
NET  "pdcm"  LOC  =  "p142"  |  IOSTANDARD  =  LVTTL  ;
NET  "psw<0>"  LOC  =  "p27"  |  IOSTANDARD  =  LVTTL  ;
NET  "psw<1>"  LOC  =  "p24"  |  IOSTANDARD  =  LVTTL  ;
NET "psw<2>" LOC = "p22" | IOSTANDARD = LVTTL;
```

EXPERIMENT3: SEVEN SEGMENT DISPLAY

AIM: Write Verilog HDL code to display given Hex digit on any selected Seven Segment Display
.
Hardware and software Requirements:
Hardware: Spartan 6 FPGA Board, multiplexed Seven segment displays, RPS, FRC, PPI to JTAG interface, PC
Software: Xilinx ISE14.7

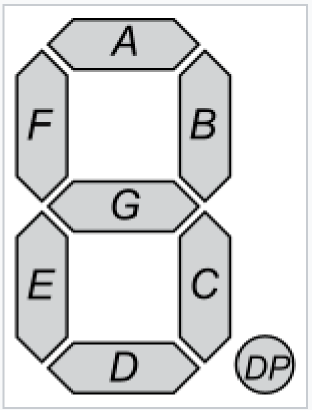
Theory:

A seven-segment display is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays.
Seven-segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information



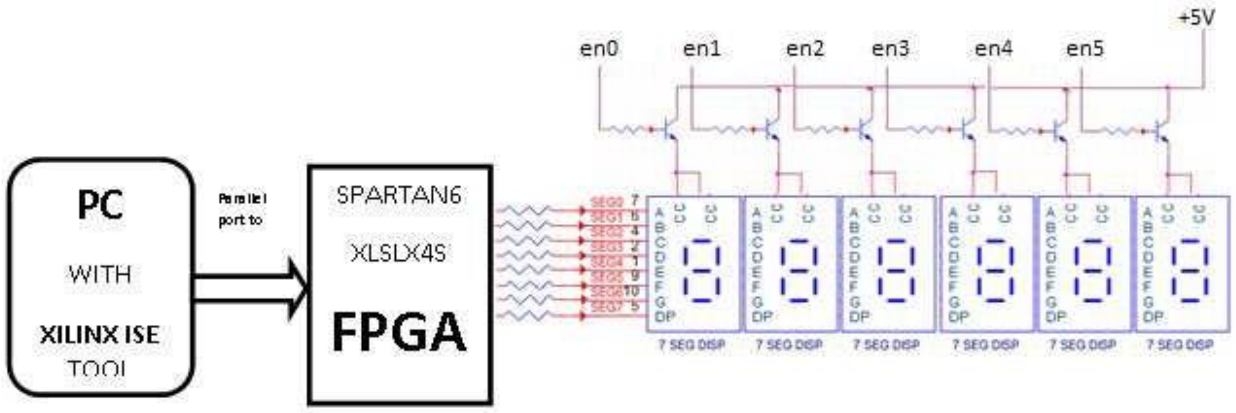
Hexadecimal encodings for displaying the digits 0 to F

Digit	Display	gfedcba	a	b	c	d	e	f	g
0	0	0x3F	on	on	on	on	on	on	
1	1	0x06		on	on				
2	2	0x5B	on	on		on	on		on
3	3	0x4F	on	on	on	on			on
4	4	0x66		on	on			on	on
5	5	0x6D	on		on	on		on	on
6	6	0x7D	on		on	on	on	on	on
7	7	0x07	on	on	on				
8	8	0x7F	on	on	on	on	on	on	on
9	9	0x6F	on	on	on	on		on	on
A	A	0x77	on	on	on		on	on	on
b	b	0x7C			on	on	on	on	on
C	C	0x39	on			on	on	on	
d	d	0x5E		on	on	on	on		on
E	E	0x79	on			on	on	on	on
F	F	0x71	on				on	on	on

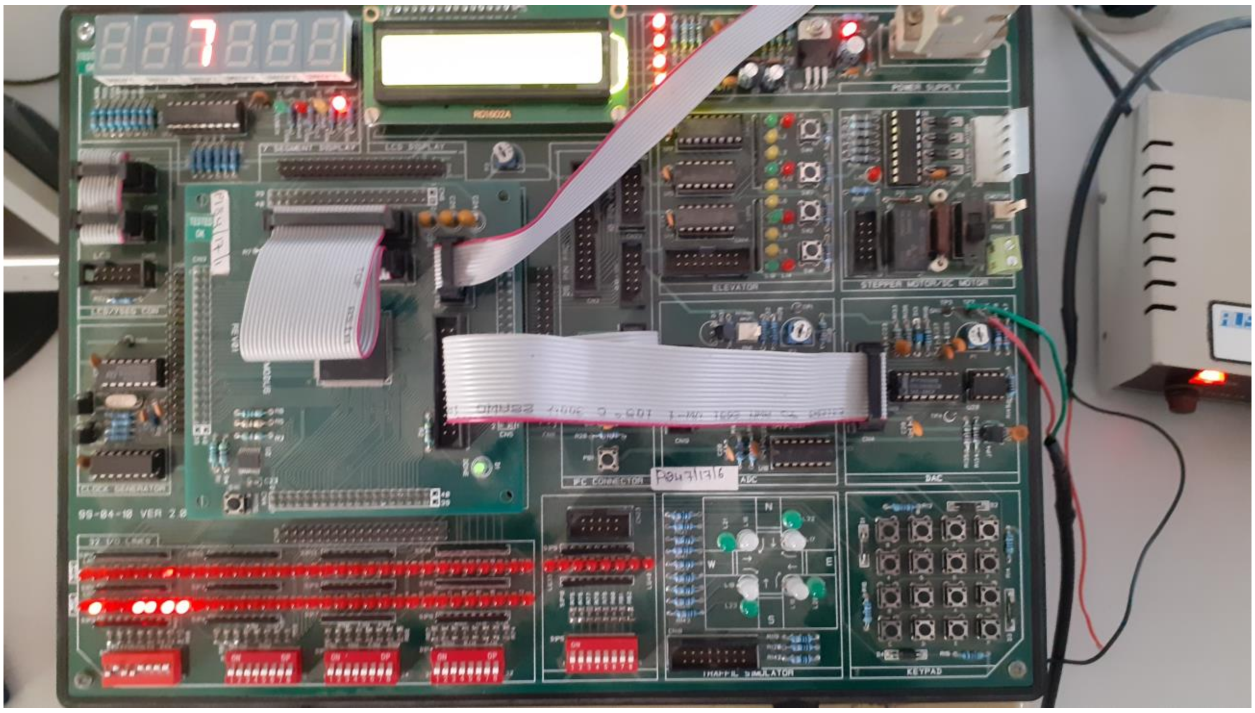


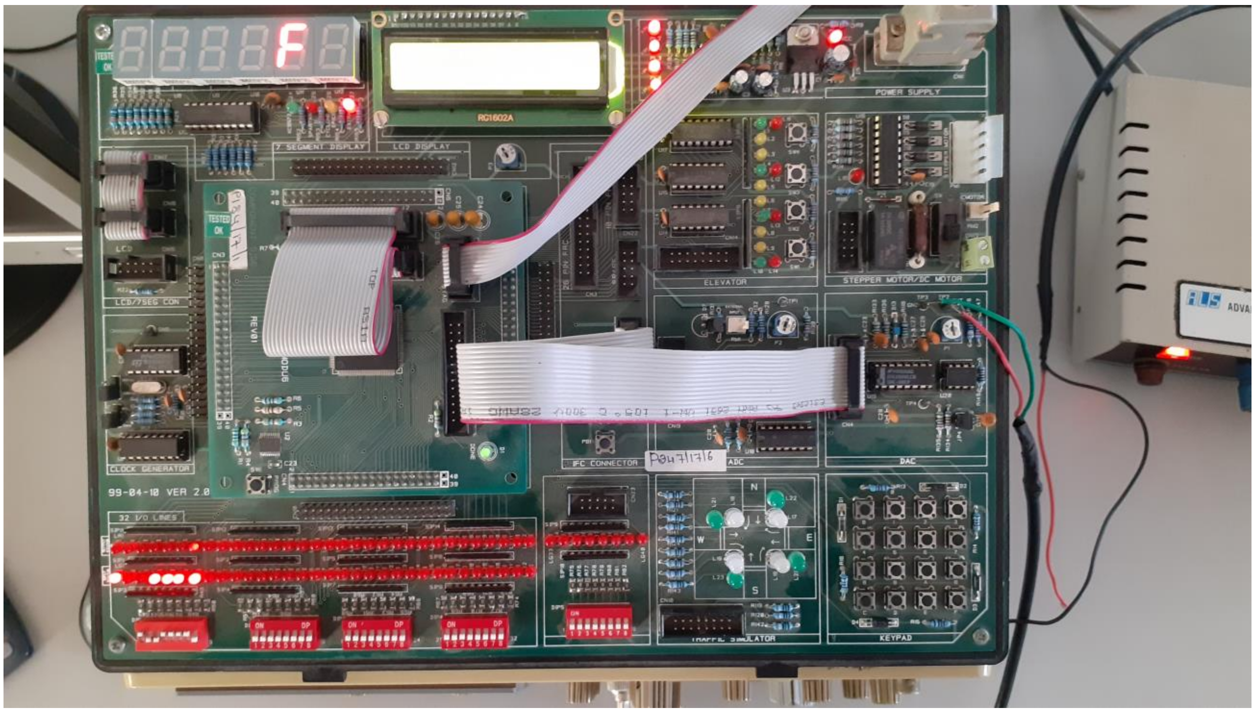
The individual segments of a seven-segment display

Interfacing Block Diagram:



```
module segled(en,sel,val,disp);
    input [2:0] sel,
    input [3:0]  val
    output reg [5:0] en;
    output reg [7:0]disp,

     always @(sel)
      begin
        case (sel)
            3'b000:en=6'b000001;
            3'b001:en=6'b000010;
            3'b010:en=6'b000100;
            3'b011:en=6'b001000;
            3'b100:en=6'b010000;
            3'b101:en=6'b100000;
            default: en=6'b000000;
         endcase

        case (val)
            4'h0:disp=8'h3F;
            4'h1:disp=8'h06;
            4'h2:disp=8'h5B;
            4'h3:disp=8'h4F;
            4'h4:disp=8'h66;
            4'h5:disp=8'h6D;
            4'h6:disp=8'h7D;
            4'h7:disp=8'h07;
            4'h8:disp=8'h7F;
            4'h9:disp=8'h6F;
            4'hA:disp=8'h77;
            4'hB:disp=8'h7C;
            4'hC:disp=8'h39;
            4'hD:disp=8'h5E;
            4'hE:disp=8'h79;
            4'hF:disp=8'h71;
        endcase
end
endmodule
```





PIN DETAILS - SEVEN SEGMENT DISPLAY

```
#PACE: Start of PACE I/O Pin Assignments
NET "sel<0>" LOC = "p27" | IOSTANDARD = LVTTL ;
NET "sel<1>" LOC = "p24" | IOSTANDARD = LVTTL ;
NET "sel<2>"  LOC = "p22" | IOSTANDARD = LVTTL ;

NET "val<0>" LOC = "p40" | IOSTANDARD = LVTTL ;
NET "val<1>" LOC = "p35" | IOSTANDARD = LVTTL ;
NET "val<2>" LOC = "p33" | IOSTANDARD = LVTTL ;
NET "val<3>"  LOC = "p30" | IOSTANDARD = LVTTL ;
NET "disp<0>" LOC = "p9" | IOSTANDARD = LVTTL | SLEW = SLOW ;
NET "disp<1>" LOC = "p10" | IOSTANDARD = LVTTL | SLEW = SLOW ;
NET "disp<2>" LOC = "p11" | IOSTANDARD = LVTTL | SLEW = SLOW ;
NET "disp<3>" LOC = "p12" | IOSTANDARD = LVTTL | SLEW = SLOW ;
NET "disp<4>" LOC = "p14" | IOSTANDARD = LVTTL | SLEW = SLOW ;
NET "disp<5>" LOC = "p15" | IOSTANDARD = LVTTL | SLEW = FAST ;
NET "disp<6>" LOC = "p16" | IOSTANDARD = LVTTL | SLEW = FAST ;
NET "disp<7>" LOC = "p17" | IOSTANDARD = LVTTL | SLEW = SLOW ;

NET "en<0>" LOC = "p8" | IOSTANDARD = LVTTL | SLEW = SLOW ;
NET "en<1>" LOC = "p7" | IOSTANDARD = LVTTL | SLEW = SLOW ;
NET "en<2>" LOC = "p6" | IOSTANDARD = LVTTL | SLEW = SLOW ;
NET "en<3>" LOC = "p5" | IOSTANDARD = LVTTL | SLEW = SLOW ;
NET "en<4>" LOC = "p2" | IOSTANDARD = LVTTL | SLEW = SLOW ;
NET  "en<5>" LOC = "p1"    | IOSTANDARD = LVTTL   | SLEW = SLOW ;
```

EXPERIMENT6: DAC

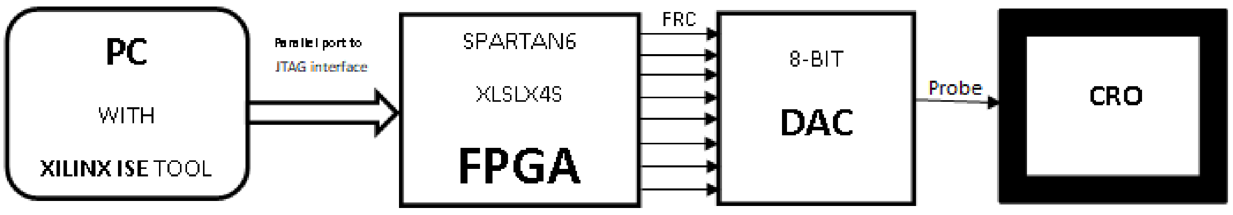
AIM: Write Verilog HDL code to generate different waveforms (Square, Triangle, sinusoidal) of suitable frequency and amplitude using 8- bit DAC
Hardware and software Requirements:
Hardware: Spartan 6 FPGA Board, 8 –bit DAC Interface, CRO, Probes, RPS, FRC, PPI to JTAG interface, PC
Software: Xilinx ISE14.7

Theory:

In electronics, a digital-to-analog converter (DAC, D/A, D2A, or D-to-A) is a system that converts a digital signal into an analog signal.

DACs are commonly used in music players to convert digital data streams into analog audio signals. They are also used in televisions and mobile phones to convert digital video data into analog video signals. These two applications use DACs at opposite ends of the frequency/resolution trade-off. The audio DAC is a low-frequency, high-resolution type while the video DAC is a high-frequency low- to medium-resolution type.

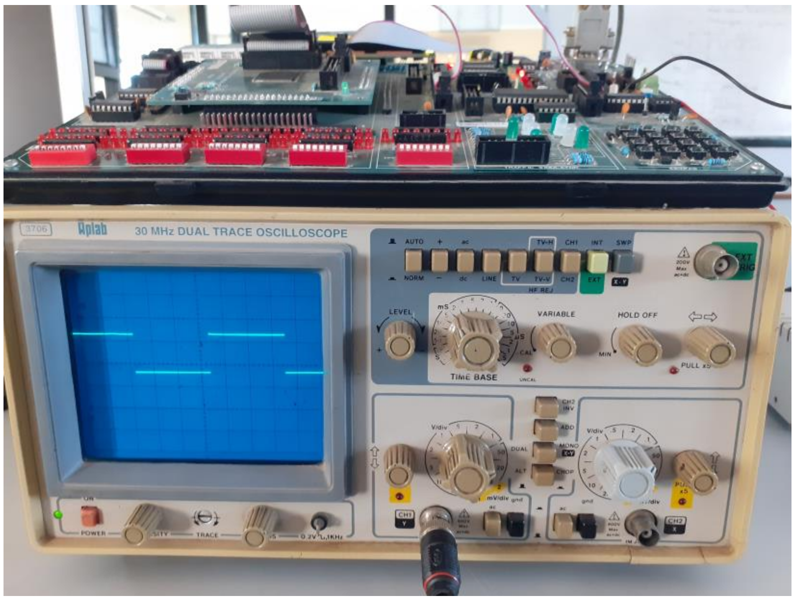
Interfacing Block Diagram:



1) SQUARE WAVEGENERATION

```
module sqrwv (clk,dout);
input clk;
output reg[7:0] dout=8’d0;

reg [22:0] cnt=23'd0;
reg tclk;

always@(posedge clk)
begin
  cnt=cnt+1;
tclk=cnt [22];
end
```

```
always@(posedge tclk)

   begin
      dout =~ dout;
  end
endmodule
```



2) TRIANGULAR WAVEGENERATION

```
module trgwv(clk,dout);
input clk;
output reg[7:0] dout;

reg [5:0] cnt=8'd0;
reg tclk;

reg [7:0] step=8'h00;

always@(posedge clk)
begin
cnt=cnt+1;
tclk=cnt [5];
end

always@(posedge tclk)
begin
      if(step<=8'h7f)
          dout=dout+1;
      else
          dout=dout-1;
```

```
  step=step+1;
end

endmodule
```

SINUSOIDAL WAVE GENERATION

```
module sinewv(clk,dout);
input clk;
output reg[7:0] dout;

reg [5:0] cnt=8'd0;
reg tclk;
reg [7:0] val [0:35]={128,150,172,192,210,226,238,248,254,255,254
248,238,226,210,192,172,150,128,,106,84,64,46,30,17,8,2,0,2,8,17,30,46,64,84,106}
integer i=0;
always@(posedge clk)
begin
cnt=cnt+1;
tclk=cnt [5];
end

always@(posedge tclk)
begin
dout=val[i];
i=i+1;
if(i==36)  i=0;
end
endmodule
```

PIN DETAILS –SQUARE, TRIANGULAR AND SINUSOIDAL WAVE

```
#PACE: Start of PACE I/O Pin Assignments

NET "clk" LOC = "p51”;

NET "dout<0>" LOC = "p131" | IOSTANDARD = LVTTL ;
NET "dout<1>" LOC = "p127" | IOSTANDARD = LVTTL ;
NET "dout<2>" LOC = "p133" | IOSTANDARD = LVTTL ;
NET "dout<3>" LOC = "p132" | IOSTANDARD = LVTTL ;
NET "dout<4>" LOC = "p137" | IOSTANDARD = LVTTL ;
NET "dout<5>" LOC = "p134" | IOSTANDARD = LVTTL ;
NET "dout<6>" LOC = "p139" | IOSTANDARD = LVTTL ;
NET "dout<7>" LOC = "p138" | IOSTANDARD = LVTTL ;
```

EXPERIMENT 6: CLOCK DIVIDER CIRCUIT DESIGN

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

```
module clkdiv (clk,f1,f2,f3,f4);
input clk;
output f2,f3,f4;
reg Q1=1,Q2=0,Q3=0,Q4=0;
always@(posedge clk)
reg Q1=1,Q2=0;
begin
dff d1(clk,Q2,Q1);
dff d2(clk,Q1,Q2);
f2=Q1;
end

always@(posedge clk)
reg clkb,Qd;
reg Q1=1,Q2=0,Q3=0;

begin
clkb=~clk;
dff d3(clk,Q3,Q1);
dff d4(clk,Q1,Q2);
dff d5(clk,Q2,Q3);
dff  d6(clkb,Q1,Qd);
or  g2(f3,Q1,Qd);    --f3=Q1| Qd;
end

always@(posedge clk)
reg Q1=1,Q2=0,Q3=0,Q4=0;
begin
dff d1(clk,Q4,Q1);
dff d2(clk,Q1,Q2);
dff d1(clk,Q2,Q3);
dff d1(clk,Q3,Q4);
or  g1(f4,Q1,Q2);   --f4=Q1| Q2;
end

task  dff ;
input clk,d;
output q;

begin
q=d;
end
endtask
endmodule
```

VIVA QUESTIONS

1) What is agate?
Gate is digital system that performs a specific logic. It comprises of transistors.

2) What is an IC?
It is a Integrated circuit where on a single silicon number of components (transistors, resistors, capacitors etc) are integrated.

3) Which are the basic gates and the universal gates?
Basic gates : and AND, OR, NOT, **Universal gates**: NAND, NOR

4. What is decoder?
It is a circuit which is having inputs and 2^n outputs.

5. Give a Real time application of decoder?
It is used in a CISC processor to decode the complex instructions.

6. What is a MUX?
It is a circuit which is having multiple inputs and a single output based on the select line

7. Give a Real time application of a MUX?
It is used in a transport layer of TCP/IP protocol suit, remote controller, satellite comm.. etc.

8.What is a Gray Code.

The **reflected binary code**, also known as **Gray code** after Frank Gray, is a binary numeral system where two successive values differ in only one bit. It is a non-weighted code. The reflected binary code was originally designed to prevent spurious output from electromechanical switches. Today, Gray codes are widely used to facilitate error correction in digital communications such as digital terrestrial television and some cable TV systems.

9. Give the application of Gray codes.

Gray codes are used in position encoders (linear encoders and rotary encoders), in preference to straightforward binary encoding. This avoids the possibility that, when several bits change in the binary representation of an angle, a misread will result from some of the bits changing before others.

10. Give a Real time application of a MUX?
It is used in a transport layer of TCP/IP protocol suit, remote controller, satellite comm.. etc

11. What is a DEMUX?
It is a circuit which is havingsingle input and multiple outputs based on the select line

12. Give a Real time application of a DEMUX?
It is used in a transport layer of TCP/IP protocol suit, remote controller, satellite comm... etc.

13. Which are the 2 kinds of Comparators?
Equality and Inequality Comparators.

14. Give an application of a comparator.
Parallel adder can be used as a Subtractor by giving one of its input in 2'complement form[X+(-Y)] =[X-Y].

15. What is a FULLADDER?
It is a circuit which is having 3 single bit inputs (A,B, Cin) and two single bit outputs(SUM,COUT)

16. Give a Real time application of a Full adder?
It is used in realizing a parallel adder, carry look ahead adder, ripple adder, serial adder, FSM.

17. What is an ALU?
In computing, an arithmetic and logic unit (ALU) is a digital circuit that performs arithmetic and logical operations.

18.. Give the uses of ALU.
The ALU is a fundamental building block of the central processing unit of a computer, and even the simplest microprocessors contain one for purposes such as maintaining timers. The processors found inside modern CPUs and graphics processing units (GPUs) accommodate very powerful and very complex ALUs; a single component may contain a number of ALUs.

19. Define a flip-flop.
In, a **flip-flop (has clock i/p)** or **latch (does not have clock i/p)** is that which has two stable states and can be used to store state 1 bit information(either 0 or 1),clock is a controlling input.

20. Give the difference between flip-flop and a Latch.
A flip-flop (has clock i/p) or Latch(does not have clock i/p) .

21. Expand the following.
JK (J-JACKK-KILBY),
T-TOGGLE (a change from 0 to1 or 1 to 0), D-Data.

22. List the applications offlip-flops.
Flip flops are generally used in Counters, memories, registers etc..

23. What is a Master Slave JK Flip-flop?
Master slave JKFF is used to overcome race around condition where the output keeps toggling between 1 and 0.

24. What is a Counter?
It is a circuit which counts the number of clock cycles.

25. Give a Real time application of counter?

It is used in traffic signals, digital watches, stop watch…etc

26. What is a Synchronous counter?
It is a counter having a common clock given to all the flip-flops.

27. What is a asynchronous reset counter?
It is a counter having different clock, wherein the output of one flip-flop is given as the clock input to the next flip-flop. It is also called a ripple counter.

28. What is a synchronous reset counter?

In this the output clears to zero only for the next rising edge of clock provided reset is also zero.

29. What is an asynchronous reset counter?
In this the output clears to zero as soon as the reset goes to zero irrespective of the clock pulse.

30. What is difference between MOD-8 and Decade counter?
MOD-8 : counter which counts from0-7.
Decade counter which counts from 0-9.