KAMMAVARISANGHAM

K. S. INSTITUTE OF TECHNOLOGY

(APPROVED BY A.I.C.T.E AFFILIATED TO VTU BELGAUM) #14,

RAGHUVANAHALLI, KANAKAPURA MAIN ROAD, BANGALORE-560109

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGG.

NAME OF THE LAB: VERILOG HDL LAB

COURSE CODE:15ECL58

K. S. INSTITUTE OF TECHNOLOGY

VISION

"To impart quality technical education with ethical values, employable skills and research to achieve excellence".

MISSION

- To attract and retain highly qualified, experienced & committed faculty.
- To create relevant infrastructure.
- Network with industry & premier institutions to encourage emergence of newideas by providing research & development facilities to strive for academic excellence.
- To inculcate the professional & ethical values among young students withemployable skills & knowledge acquired to transform the society.

K.S. INSTITUTE OF TECHNOLOGY

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

VISION:

"To achieve excellence in academics and research in Electronics & Communication Engineering to meet societal need".

MISSION:

- To impart quality technical education with the relevant technologies to produce industry ready engineers with ethical values.
- To enrich experiential learning through active involvement in professional clubs & societies.
- To promote industry-institute collaborations for research &development.

K.S. INSTITUTE OF TECHNOLOGY

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

PROGRAM EDUCATIONAL OBJECTIVES (PEO'S)

PEO1: Excel in professional career by acquiring domain knowledge.

PEO2: Motivation to pursue higher Education & research by adopting technological innovations by continuous learning through professional bodies and clubs.

PEO3: To inculcate effective communication skills, team work, ethics and leadership qualities.

PROGRAM SPECIFIC OUTCOMES (PSO'S)

PSO1: Graduate should be able to understand the fundamentals in the field of Electronics & Communication and apply the same to various areas like Signal processing, embedded systems, Communication & Semiconductor technology.

PSO2: Graduate will demonstrate the ability to design, develop solutions for Problems in Electronics & Communication Engineering using hardware and software tools with social concerns.

K.S. INSTITUTE OF TECHNOLOGY

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3: 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.
PO4: 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.
PO5: 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.
PO6: 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.
PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10: 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.
PO11: 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.
PO12: 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.

Course Code 15ECL58	Course: HDL LAB	
15ECL58.1	To construct the Verilog/VHDL programs to simulate Combinational circuits inDataflow, Behavioral and Gate level Abstractions	APPLYING K3
15ECL58.2	To demonstrate sequential circuits like flip flops and counters in Behavioraldescription and obtain simulation waveforms	APPLYING K3
15ECL58.3	To connect the results with Synthesize Combinational and Sequential circuits on programmable ICsand test the hardware	ANALYSE K4
15ECL58.4	To compare the results and to verify for the functionality of combinational and sequential circuits	ANALYSE K4
15ECL58.5	To summarize the knowledge and interface the hardware to the programmable chips and obtain the required output.	EVALUATE K5

COURSENAME_HDLLAB				COURSECODE_15ECL58							
COs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	2	1	1	1	_	_	_	2	1	_	1
CO2	2	2	1	1	2	1	_	_	2	1	_	1
CO3	2	2	1	1	2	1	_	_	2	1	_	1
CO4	3	3	2	2	3	1	_	_	2	1	_	1
CO5	3	3	3	3	3	1	_	_	2	1	_	1
15ECL58	**3**	2	2	1	2	1	_	_	2	1	_	1

CO – PSO Mapping

CO	PSO1	PSO2
15ECL58.1	2	2
15ECL58.2	3	3
15ECL58.3	3	3
15ECL58.4	3	3
15ECL58.5	3	3
15ECL58	**3**	**3**

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

Verilog HDL

Laboratory Code	**15ECL58**	**CIE Marks**	**20**
Number of Lecture Hours/Week	**02 Hr Tutorial (Instructions) + 02 Hours Laboratory**	**SEE Marks**	**80**
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 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/CPL board and performance testing may be done using 32 channel pattern generator and logic analyze apart from Verification By simulation with tools such as Altera / Modelisim or equivalent.

Laboratory Experiments

PART A : Programming

1. Write Verilog code to realize all the logic gates

2. Write a Verilog program for the following combinational designs
   a. 2 to 4 decoder
   b. 8 to 3 (encoder without priority & with priority)
   c. 8 to 1 multiplexer.
d. 4 bit binary to gray converter
e. Multiplexer, de-multiplexer, comparator.

3. Write a VHDL and Verilog code to describe the functions of a Full Adder using three modelingstyles.

4. Write a Verilog code to model 32 bit ALU using the schematic diagram shown below

- ALU should use combinational logic to calculate an output based on the four bit op-code input.
- ALU should pass the result to the out bus when enable line in high, and tri-state the out bus when the enable line is low.
- ALU should decode the 4 bit op-code according to the example given below.

5. Develop the Verilog code for the following flip-flops, SR, D, JK and T.

6. Design a 4 bit binary, BCD counters (Synchronous reset and Asynchronous reset) and ―anysequence‖ counters, using Verilog code.

PART B :
INTERFACING (at least four of the following must be covered using VHDL/Verilog)

1. Write HDL code to display messages on an alpha numeric LCD display.

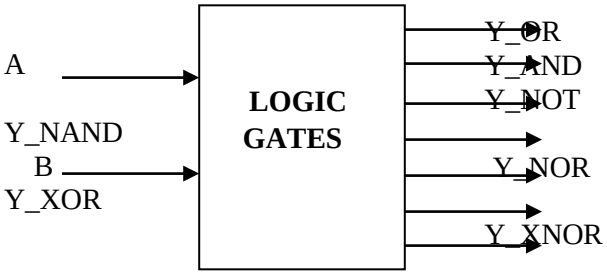
2. Write HDL code to interface Hex key pad and display the key code on seven segment display.

3. Write a HDL code to control speed, direction of DC and Stepper Motor.
4. Write HDL code to accept Analog signal, Temperature sensor and display the data on LCD or Seven segment display.
5. Write HDL code to generate different waveforms (Sine, Square, Triangle, Ramp etc.,) using DAC – change the frequency.
6. Write HDL code to simulate Elevator operation.

-- Program 1.

-- Write an HDL code to implement ALL BASIC GATES.
-- Data flow modeling.

Entity level diagram



Truth table Basic gates

A	B	Y_AND	Y_OR	Y_NOT	Y_NAND	Y_NOR	Y_XOR	Y_XNOR
0	0	0	0	1	1	1	0	1
0	1	0	1	1	1	0	1	0
1	0	0	1	0	1	0	1	0
1	1	1	1	0	0	0	0	1

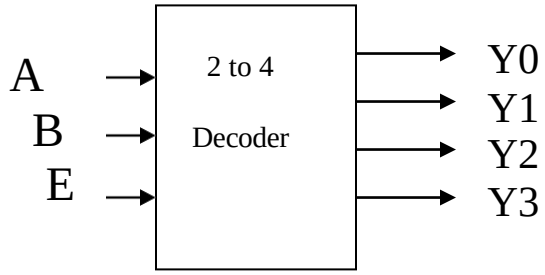
Verilog Code

```
module ALLGATES (A,B,Y_OR,Y_AND,Y_NOT,Y_NAND,Y_NOR,Y_XOR,Y_XNOR);
input  A,B;
output Y_OR,Y_AND,Y_NOT,Y_NAND,Y_NOR,Y_XOR,Y_XNOR;
        assign Y_OR = A | B;
        assign Y_AND = A & B;
        assign Y_NOT =~ A;
        assign Y_NOR = ~(A| B);
        assign Y_NAND =~( A& B);
        assign Y_XOR =( A^ B);
        assign Y_XNOR =~( A^ B);
endmodule
```

-- Program 2(a).

-- Write an HDL code for 2 TO 4 DECODER combinational circuit.
-- Data flow modeling (assign).

Entity level diagram



Truth table 2 to 4 Decoder

E	A	B	Y3	Y2	Y1	Y0
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

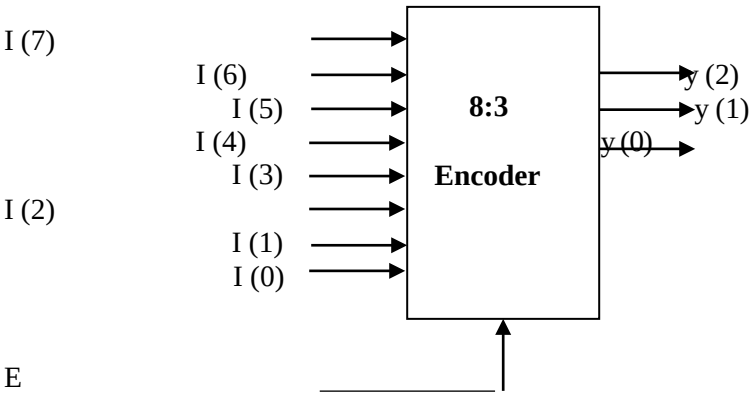
Verilog Code

```
module DECODER_2LINE4 (A,B,E,Y0,Y1,Y2,Y3);
input A,B,E;
output Y0,Y1,Y2,Y3;
        assign Y0 = (~A) & (~ B) & E;
        assign Y1 = (~A) & B& E ;
        assign Y2 = A & (~ B) & E;
        assign Y3 = A & B& E;
endmodule
```

-- Program 2(b).

-- Write aHDL code for the 8to 3 Encoder without priority.
-- Behavioral modeling (IF ELSE).

Entity level diagram



Truth table 8 to 3 encoder without priority

E	I7	I6	I5	I4	I3	I2	I1	I0	Y2	Y1	Y0
0	d	d	d	d	d	d	d	d	x	x	x
1	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	0	1	0	0	0	1	0
1	0	0	0	0	1	0	0	0	0	1	1
1	0	0	0	1	0	0	0	0	1	0	0
1	0	0	1	0	0	0	0	0	1	0	1
1	0	1	0	0	0	0	0	0	1	1	0
1	1	0	0	0	0	0	0	0	1	1	1
1	1	d	d	d	d	d	d	1	x	x	x

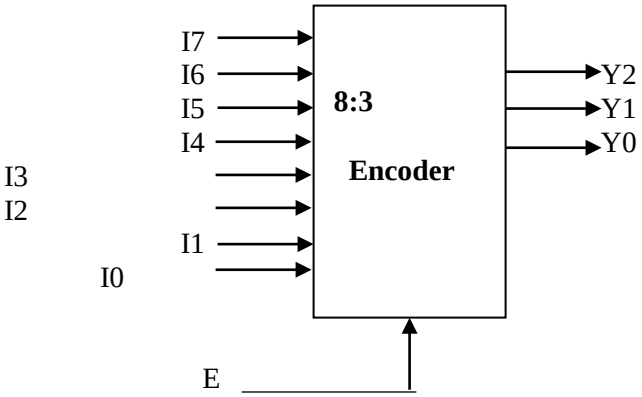
In the truth table **d** means for don't care, **z** means the High impedance, x means unknown

Verilog Code

```
module ENCODER_8LINE3 ( E,I,Y);
                input E;
                input [ 7: 0] I;
                output [2:0] Y;
                reg [ 2:0] Y;
                 always@ ( E,I)
                 begin
                        if ( E ==1)
                        begin
                                if ( I == 8'b 00000001) Y = 3'b 000;
else  if ( I == 8'b 00000010) Y = 3'b 001;
else if ( I == 8'b 00000100) Y = 3'b 010;
else if ( I == 8'b 00001000) Y = 3'b 011;
else if ( I == 8'b 00010000) Y = 3'b 100;
else if ( I == 8'b 00100000) Y = 3'b 101;
else if ( I == 8'b 01000000) Y = 3'b 110;
else  if ( I == 8'b 10000000) Y = 3'b 111;
else Y = 3'bxxx;
end
                        else Y = 3'bxxx;
end
endmodule
```

-- Write a HDL code for the 8 to 3 Encoder with priority.
-- Behavioral modeling (IF ELSE).

Entity level diagram



Truth table 8 to 3 encoder with priority

E	I7	I6	I5	I4	I3	I2	I1	I0	Y2	Y1	Y0
0	d	d	d	d	d	d	d	d	x	x	x
1	1	d	d	d	d	d	d	d	1	1	1
1	0	1	d	d	d	d	d	d	1	1	0
1	0	0	1	d	d	d	d	d	1	0	1
1	0	0	0	1	d	d	d	d	1	0	0
1	0	0	0	0	1	d	d	d	0	1	1
1	0	0	0	0	0	1	d	d	0	1	0
1	0	0	0	0	0	0	1	d	0	0	1
1	0	0	0	0	0	0	0	1	0	0	0

In the truth table **d**means don't care, **z** means the High impedance, x means unknown

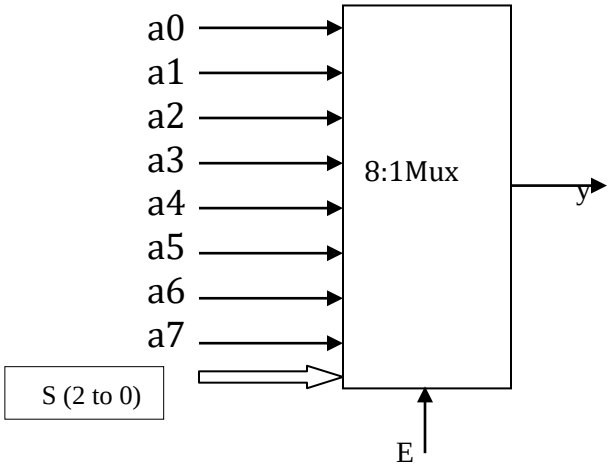
Verilog Code

```
module ENCODER_8LIP3 ( E,I,Y);
                        input E;
                        input [ 7: 0] I;
                        output [2:0] Y;
                        reg [ 2:0] Y;
                         always@ ( E,I)
                         begin
                                 if ( E ==1)
                                 begin
if ( I [7]== 1) Y = 3'b 111;
else if ( I[6] == 1) Y = 3'b 110;
else if ( I[5] == 1) Y = 3'b 101;
else if ( I[4] == 1) Y = 3'b 100;
else if ( I[3] == 1) Y = 3'b 011;
else if ( I[2] == 1) Y = 3'b 010;
else if ( I[1] == 1) Y = 3'b 001;
else  if ( I[0] == 1) Y = 3'b 000;
else Y = 3'bxxx;
 end
                                 else Y = 3'bxxx;
         end
endmodule
```

-- Program 2(c).

-- Write aHDL code for the 8 to 1 mux combinational circuit.
-- Behavioral modeling (case).

Entity level diagram



Truth table 8 to 1 Mux

e	s(2)	s(1)	s(0)	Y
0	d	d	d	z
1	0	0	0	a0
1	0	0	1	a1
1	0	1	0	a2
1	0	1	1	a3
1	1	0	0	a4
1	1	0	1	a5
1	1	1	0	a6
1	1	1	1	a7

In the truth table **d**means for don't care, **z** means the High impedance, x means unknown

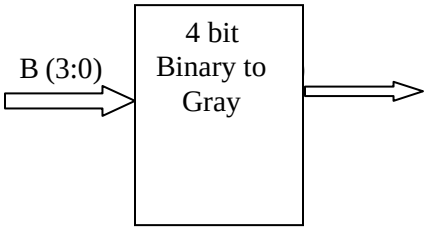
Verilog Code

```
module EIGHTTO1MUX (a,s,e,y);
        input [7:0] a;
        input [2:0] s;
        input e;
        output y;
        reg y;

         always@(a,s,e)
         begin
           if(e==1)
             case(s)
               3'b000 : y=a[0];
               3'b001 : y=a[1];
               3'b010 : y=a[2];
               3'b011 : y=a[3];
               3'b100 : y=a[4];
               3'b101 : y=a[5];
               3'b110 : y=a[6];
               3'b111 : y=a[7];

        endcase
        else  y= 1'bz;
         end
endmodule
```

-- Program 2(d).

-- Write aHDL code for the 4 bit binary to grey converter.
-- Dataflow modeling.

Entity level diagram



Truth table 4 bit binary to grey converter

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

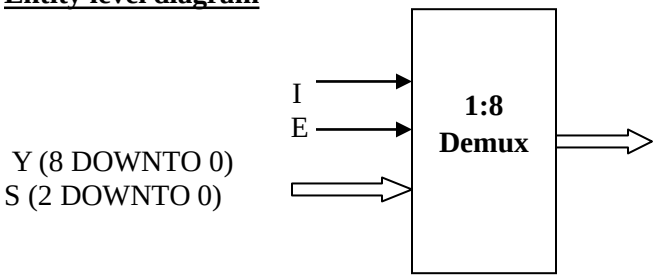
Verilog Code

```
module B2G( B, G);
   input [3:0] B;
   output [3:0]  G;
         assign  G[3]= B[3];
         assign  G[2]=  B[3]^  B[2];
         assign  G[1]=  B[2]^  B[1];
         assign G[0]= B[1]^ B[0];
endmodule
```

-- Program 2(e).

-- Write aHDL code for the 1:8 Demultiplexer combinational circuits.
-- Behavioral modeling (Case).

Entity level diagram



Truth table 1 to 8 de-mux

E	I	S(2)	S(1)	S(0)	Y(7)	Y(6)	Y(5)	Y(4)	Y(3)	Y(2)	Y(1)	Y(0)
0	d	d	d	d	z	z	z	z	z	z	z	z
1	1	0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	1	0	0	0	0	0	0	1	0
1	1	0	1	0	0	0	0	0	0	1	0	0
1	1	0	1	1	0	0	0	0	1	0	0	0
1	1	1	0	0	0	0	0	1	0	0	0	0
1	1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0

In the truth table **d**means for don't care, **z** means the High impedance, x means unknown.

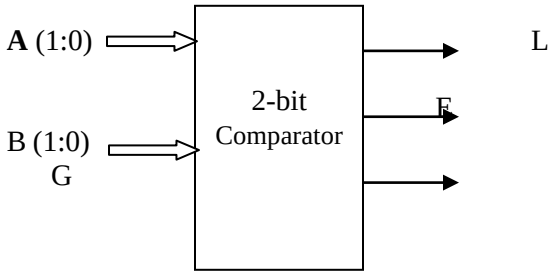
Verilog Code

```
module DEMUX (I,S,E,Y);
        input I,E;
        input [2:0]S;
        output [7:0] Y;
         reg [7:0] Y;
                always @ (S,E,I)
                begin
                        Y = 8'b 00000000;
                 if (E==1)
                case (S)
                        3'b000: Y[0] = I;
                        3'b001: Y[1] = I;
                        3'b010: Y[2] = I;
                        3'b011: Y[3] = I;
                        3'b100: Y[4] = I;
                        3'b101: Y[5] = I;
                        3'b110: Y[6] = I;
                        3'b111: Y[7] = I;
default  : Y= 8'bzzzzzzzz;
endcase
                 else  Y= 8'bzzzzzzzz;
                 end
endmodule
```

-- Program 2(f).

-- Write aHDL code for the 2 bit comparator combinational circuit.
-- Behavioral modeling (IF ELSE).

Entity level diagram



Truth table 2-bit comparator

A(1)	A(0)	B(1)	B(0)	L(A Less than B)	E(A Equal to B)	G(A Greater than B)
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

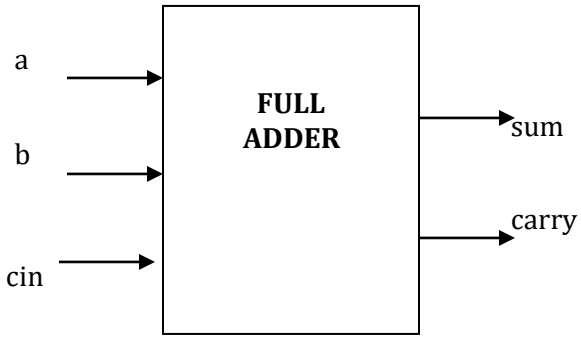
Verilog Code

```
module COMP(A,B,L,G,E);
   input A,B;
   output L,G,E;
          reg L,G,E;
          always@(A,B)
             begin
                 if(A<B)
         begin
                              L=1;
                              G=0;
                              E=0;
                 end
                else if(A>B)
         begin
                              L=0;
                              G=1;
                              E=0;
                 end
               else
                 begin
                              L=0;
                              G=0;
                              E=1;
                 end
            end
endmodule
```

-- Program 3.

-- Write aHDL code to implement Full Adder.

Entity level diagram



Truth table full adder

INPUTS			OUTPUTS	
a	**b**	**cin**	**sum**	**carry**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

VHDL Code

Behavioral Modeling

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FA is

   Port ( a,b,cin: in std_logic;
          sum,carry: out std_logic);

end FA;

architecture behavioral  ofFA is
begin
          process(a,b,cin)
          begin
```

```
        If(a='0' and b='0' and cin='0') then sum <='0'; carry <='0';
        elsif(a='0' and b='0' and cin='1') then sum <='1'; carry <='0';
        elsif(a='0' and b='1' and cin='0') then sum <='1'; carry <='0';
        elsif (a='0' and b='1' and cin='1') then sum <='0'; carry <='1';
        elsif (a='1' and b='0' and cin='0') then sum <='1'; carry <='0';
        elsif (a='1' and b='0' and cin='1') then sum <='0'; carry <='1';
        elsif (a='1' and b='1' and cin='0') then sum <='0'; carry <='1';
        elsif (a='1' and b='1' and cin='1') then sum <='1'; carry <='1';
        end if ;

    end process;
end behavioral ;
```

Data Flow Modeling

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity df_FA is

  Port ( a,b,cin: in std_logic;
        sum,carry: out std_logic);

end df_FA;


architecture dataflow of df_FA is
begin

        Sum <= a xor b xorcin;
        Carry <= (a and b) or (b and cin) or (a and cin);

end dataflow;
```

Structural Modeling

Half Adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity half_adderis

        port(in1, in2: in std_logic;
```

```
sum, c_out: out std_logic);

end half_adder;

architecture ha_structuralof half_adderis
begin
                sum<= in1 xor in2;
                c_out<= in1 and in2;

end ha_structural;
```

Full Adder using two half adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity full_adderis
        port(a,b, c_in: in std_logic;
                sum, c_out: out std_logic);
end entity full_adder;

architecture structural of full_adderis

        component half_adderis

                port (x,y: in std_logic;
sum1, carry: out std_logic);

        end component half_adder;

signal s1, c2, c3: std_logic;
begin

H1: half_adderport map(x=>a, y=>b, sum1=>s1, carry=>c3);
H2: half_adderport map(x=>s1, y=>c_in, sum1=>sum, carry=>c2);
c_out<= c2 or c3;

end fa_structural;
```

Verilog Code

Behavioral model

```
module full_adder (a,b,cin,sum,carry);
input a,b,cin;
output sum,carry;
reg sum,carry;
```

```
always@(a,b,cin)
begin
if(a==1'b0 && b==1'b0 &&cin==1'b0)
begin sum <=1'b0; carry <=1'b0;
end
                else if(a==1'b0&& b==1'b0 &&cin==1'b1)
begin sum <=1'b1; carry <=1'b0;
end
                else if(a==1'b0 && b==1'b1 &&cin==1'b0)
begin sum <=1'b1; carry <=1'b0;
end
                else if (a==1'b0 && b==1'b1 &&cin==1'b1)
begin sum <=1'b0; carry <=1'b1;
end
                else if (a==1'b1 && b==1'b0 &&cin==1'b0)
begin sum <=1'b1; carry <=1'b0;
end
                else if (a==1'b1 && b==1'b0 &&cin==1'b1)
begin sum <=1'b0; carry <=1'b1;
end
                else if (a==1'b1 && b==1'b1 &&cin==1'b0)
begin sum <=1'b0; carry <=1'b1;
end
                else if (a==1'b1 && b==1'b1 &&cin==1'b1)
begin sum <=1'b1; carry <=1'b1;
end
end
endmodule
```

Data flow model

```
module full_adder (a,b,cin, sum, carry);
        input a,b,cin;
        output sum,carry;
        assign sum=a ^ b ^ cin;
        assign carry=( a& b ) | ( b &cin ) | ( cin& a );
endmodule
```

Structural model

Half Adder

```
module HA (a, b, s, cout);
   input a,b;
   output s,cout;
   assign s=a^b;
```

```
assign cout=a&b;
endmodule
//OR gate
module OR( p, q, o);
input p,q;
output o;
assign o = p | q ;
endmodule
```

Full Adder using two half adder

```
module FA (A, B, Cin, sum, carry);
input A, B, Cin;
output sum, carry ;
wire s1, c1, c2;
HA ha1 (A,B,s1,c1);
HA  ha2 (s1, Cin, sum, c2);
OR or_g (c1,c2,carry);
endmodule
```

-- Program 4

-- Write aHDL code to implement 4 bit ALU.
-- Behavioral modeling.

opcode	Operation
000	Addition
001	Subtraction
010	Complement of a
011	multiplication
100	AND operation
101	OR operation
110	NAND operation
111	XOR operation

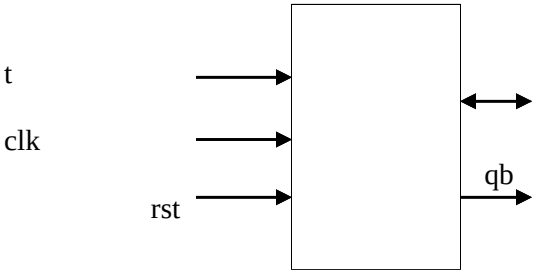
Verilog Code

```
module ALU_4BIT(a, b, e,opcode, y);
  input [3:0] a,b;
  input e;
  input [3:0] opcode;
  output [7:0] y;
  reg [7:0] y;
      always@(a,b,opcode,e)
          begin
          if(e==1)
           case(opcode)
                  4'b0000:y=a+b;
                  4'b0001:y=a-b;
                   4'b0010:y={4'b0000, ~a};
                  4'b0011:y=a*b;
                          4'b0100:y=a&b;
                   4'b0101:y=a|b;
                          4'b0110:y={4'b0000,~(a&b)};
                  4'b0111:y = a ^b;
                                  default : y = 8'b00000000;
        endcase
        else
                  y = 8'bzzzzzzzz;
     end
endmodule
```

--Program 5(a)

-- Write a HDL code to implement T FLIP FLOP.
-- Behavioral modeling (IF ELSE).

Entity level diagram



Truth table T Flip Flop

clk(Rising edge)	rst	t	q+	qb+
1	0	0	q	qb
1	0	1	qb	q
1	1	d	0	1

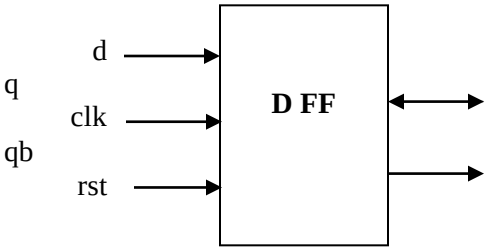
Verilog Code

```
module t_ff(t,clk,rst,q,qb);
input t,clk,rst;
output q,qb;
reg q=0,qb;
reg [31:0] clkdiv = 32'd0;
always @ (posedgeclk)
clkdiv = clkdiv+1;
always@(posedgeclkdiv[24])

begin
if (rst==1)
  q = 0;
else if(t==1)
 q=~ q;
else
  q=q;
qb=~q;
end
endmodule
```

--Program 5(b)

-- write a HDL code to implement D FLIP FLOP.
-- Behavioral modeling (IF ELSE).

Entity level diagram



Truth Table D Flip Flop

clk(rising edge)	rst	D	q+	qb+
1	0	0	0	1
1	0	1	1	0
1	1	D	0	

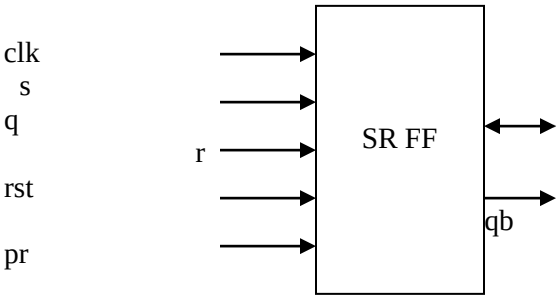
Verilog Code

```
module D_FF(d,clk,rst,q,qb);
input d,clk,rst;
output q,qb;
reg q=0,qb;
reg [41:0] clkdiv = 42'd0;

always @ (posedgeclk)
clkdiv = clkdiv+1;

always@(posedgeclk) //Use clkdiv[24] inplace of clk, while programming to kit
begin
  if (rst==1)
    q = 0;
  else
    q=d;

qb=~ q ;
end
endmodule
```

--Program 5(c)

-- Write a HDL code to implement SR FLIP FLOP.
-- Behavioral modeling (CASE).

Entity level diagram



Truth table SR Flip Flop

clk(rising edge)	Rst	s	r	q+	qb+
1	1	d	d	0	1
1	0	0	0	q	qb
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	Undefined(q)	Undefined(qb)

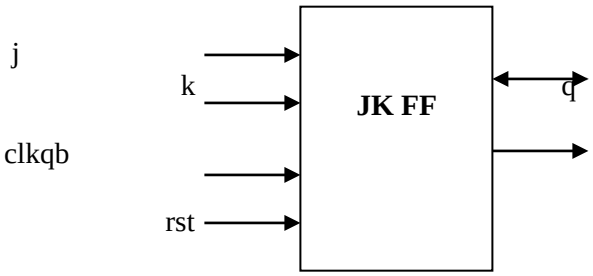
Verilog Code

```
module SR_FF(s,r,clk,rst, q,qb);
input s,r,rst,clk;
output q,qb;
reg q = 0, qb;
reg [1:0] sr;
reg [31:0] clkdiv = 32'd0;
always @ (posedgeclk)
clkdiv = clkdiv+1;
always@(posedgeclk) //Use clkdiv[24] inplace of clk, while programming to kit
begin
sr={s,r}; // Concatenate s and r
if(rst==1) q = 0;
else
begin
case (sr)
2'b00: q=q;
2'b01: q=0;
2'b10: q=1;
default: q = q;
endcase
end
qb=~q;
end
endmodule
```

-- Program 5(d)

-- Write a HDL code to implement JK FLIP FLOP.
-- Behavioral modeling (CASE).

Entity level diagram



Truth table JK Flip Flop

clk(rising edge)	rst	j	k	q+	qb+
1	0	0	0	q	qb
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	qb	q
1	1	d	d	0	1

d - Don't care

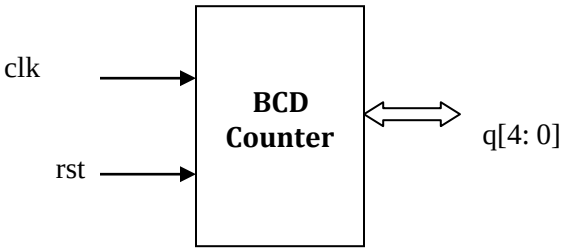
Verilog Code

```
module JK_FF(j,k,clk,rst, q,qb);
input j,k,clk,rst;
output q,qb;
reg q=0, qb;
reg [1:0] jk;
reg [41:0] clkdiv = 42'd0;
always @ (posedgeclk)
clkdiv = clkdiv+1;
always@(posedgeclk) //Use clkdiv[24] inplace of clk, while programming to kit
begin
jk={j,k}; // Concatenate j and k
if(rst==1) q = 0;
else
begin
case (jk)
2'b00: q=q;
2'b01: q=0;
2'b10: q=1;
2'b11: q=~q;
default: q = q;
endcase
end
qb=~q;
end
endmodule
```

-- Program 6(a)

-- Write a HDL code to realize BCD Counter.
-- Behavioral modeling (IF ELSE).

Entity level diagram



Truth table BCD Counter (Synchronous reset)

clk(rising edge)	rst	q[4:0]
1	0	0000
1	0	0001
1	0	0010
1	0	0011
1	0	0100
1	1	0000
1	0	0001
1	0	0010
1	0	0011
1	0	0100
1	0	0101
1	0	0110
1	0	0111
1	0	1000
1	0	1001
1	0	0000

Verilog Code

```
module BCD_syncounter(rst,clk,q);
input rst,clk;
output [3:0]q;

reg [3:0] q;
reg [31:0] clkdiv=25'd0;

always@(posedgeclk)
clkdiv=clkdiv+1;
always@(posedgeclkdiv[24])//Replace clk withclkdiv[24] while programming to the kit
begin
if(rst==1 || q==4'b1001)
```

```
    q=4'b0000;
    else
    q=q+1;
end
endmodule
```

Truth table BCD Counter (Asynchronous reset)

clk(rising edge)	rst	q[4:0]
1	1	0000
1	0	0000
1	0	0001
1	0	0010
1	0	0011
1	0	0100
1	0	0101
1	0	0110
1	0	0111
1	0	1000
1	0	1001
1	0	0000
1	0	0001
1	0	0010
1	0	0011
1	0	0100
1	0	0101

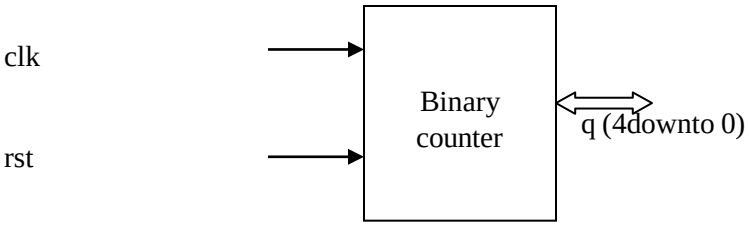
Verilog Code

```
module BCD_asyncounter(rst,clk,q);
input rst,clk;
output[3:0] q;
reg [3:0] q=4'b0000;
 reg [31:0] clkdiv=32'd0;

always@( posedgeclk)
clkdiv=clkdiv+1;

always@(posedgeclk, posedgerst)  //Replace clk with clkdiv[24] while programming to the kit
begin
if(rst==1 || q==4'b1001)
     q=4'b0000;
    else
    q=q+1;
end
endmodule
```

-- Program 6(b)

-- Write a HDL code to realize Binary Counter.
-- Behavioral modeling (IF ELSE).

Entity level diagram



Truth table Binary Counter (Synchronous reset)

clk(rising edge)	rst	q[4:0]
1	1	0000
1	0	0001
1	0	0010
1	0	0011
1	0	0100
1	0	0101
1	0	0110
1	0	0111
1	0	1000
1	0	1001
1	0	1010
1	0	1011
1	0	1100
1	0	1101
1	0	1110
1	0	1111

Verilog Code

```
module binary_counter(rst,clk,q);
input rst,clk;
output [3:0] q;
reg [3:0] q;
reg [31:0] clkdiv=32'd0;

always@(posedgeclk)
clkdiv=clkdiv+1;
always@(posedgeclk) //replace clk withclkdiv[24] while programming to the kit
begin
if(rst==1)
   q=4'd0;
   else
   q=q+1;
end
endmodule
```

Truth table Binary Counter (Asynchronous reset)

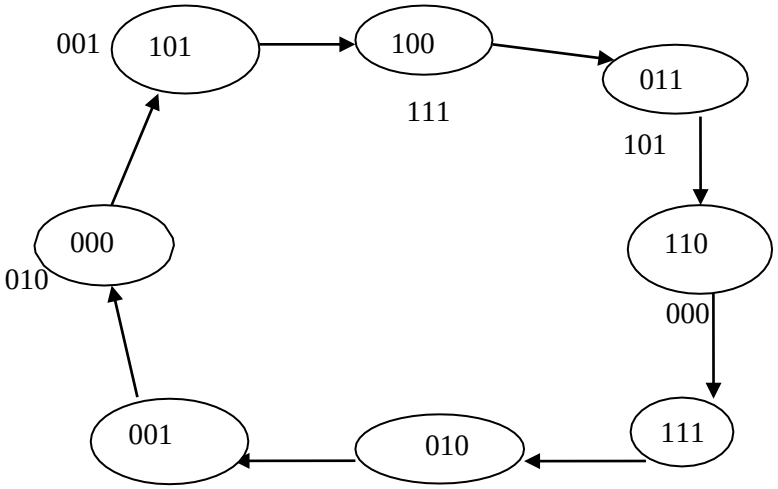
clk(rising edge)	rst	q[4:0]
d	1	0000
1	0	0001
1	0	0010
1	0	0011
1	0	0100
1	0	0101
1	0	0110
1	0	0111
1	0	1000
1	0	1001
1	0	1010
1	0	1011
1	0	1100
1	0	1101
1	0	1110
1	0	1111

Verilog Code

```
module binary_as_counter(rst,clk,q);
input rst,clk;
output[3:0] q;
reg [3:0] q=4'b0000;
 reg [31:0] clkdiv=32'd0;
always@( posedgeclk)
clkdiv=clkdiv+1;
always@(posedgeclk, posedgerst) //Replace clk withclkdiv[24] while programming to the kit
begin
if(rst==1)
    q=4'd0;
   else
   q=q+1;
end
endmodule
```

-- Program 6(c)
-- Write a HDL code generate any sequence counter Seq: (4->3->6->7->2->1->0->5)

Block Diagram:



clk	rst	q[4:0]
d	1	000
1	0	101
1	0	100
1	0	011
1	0	110
1	0	111
1	0	010
1	0	001

Code:

```
Module any_sequence (q, clk, rst);
Output [2:0] q;
input clk, rst;
reg [2:0]q
reg [2:0] count=0;
reg [31:0]clkdiv = 32'd0;
always @(posedge clkdiv[24])
begin
if(rst==1)
count=0;
else
count=count+1;
end
```

```
always @(count)
begin
case(count)
3’b000:q=3’b100;
3’b000:q=3’b011;
3’b000:q=3’b110;
3’b000:q=3’b111;
3’b000:q=3’b010;
3’b000:q=3’b001;
3’b000:q=3’b000;
3’b000:q=3’b101;
end case
end
endmodule
```

Part–B: INTERFACING

1. Write HDL code to display messages on an alpha numeric LCD display.

```
// 16 X 2 LCD display module demo code

module LCD_DEMO(
  // Assuming 100MHz input clock. My need to adjust the counter below
  // if any other input frequency is used
    input P_Clk,
 // Register Select(rs), Enable(en) and Data Pins of LCD
    output reg LCD_RS,LCD_EN,
    output reg [7:0] P_LCD
                    );

parameter Length = 53; //memory depth
integer counter;
//reg [32:0] delay = 32'hFFFFFFFF;
integer pointer = 0;

// Memory created to store the required data of 8 bits and rs signals
wire [8:0] memory[0:Length-1];

// Function Set: 8-bit, 2 Line, 5x7 Dots.
assign  memory[0] = {1'b0,8'h38};
// Display on Cursor on.
assign  memory[1] = {1'b0,8'h06};
// Entry Mode.
assign  memory[2] = {1'b0,8'h0C};
// Clear Display (also clear DDRAM content).
assign memory[3] = {1'b0,8'h01};

// The memory location in middle are kept empty so that the LCD is properly
// initialized and is ready to accept data.

// Character that should be displayed on the LCD.
assign memory[20] = {1'b1,"*"};
assign  memory[21] = {1'b1,"*"};
assign  memory[22] = {1'b1," "};
assign  memory[23] = {1'b1,"W"};
assign   memory[24] = {1'b1,"E"};
assign   memory[25] = {1'b1,"L"};
assign   memory[26] = {1'b1,"C"};
assign   memory[27] = {1'b1,"O"};
assign   memory[28] = {1'b1,"M"};
```

```
assign  memory[29] = {1'b1,"E"};
assign  memory[30] = {1'b1," "};
assign   memory[31] = {1'b1,"T"};
assign   memory[32] = {1'b1,"O"};
assign  memory[33] = {1'b1," "};
assign   memory[34] = {1'b1,"*"};
assign   memory[35] = {1'b1,"*"};

// Shift to second Line of LCD
assign  memory[36] = {1'b0,8'hC0};

// Character that should be displayed on the LCD.
assign memory[37] = {1'b1,"V"};
assign   memory[38] = {1'b1,"A"};
assign   memory[39] = {1'b1,"S"};
assign   memory[40] = {1'b1,"U"};
assign   memory[41] = {1'b1,"N"};
assign   memory[42] = {1'b1,"D"};
assign   memory[43] = {1'b1,"H"};
assign   memory[44] = {1'b1,"A"};
assign   memory[45] = {1'b1,"R"};
assign   memory[46] = {1'b1,"A"};
assign  memory[47] = {1'b1," "};
assign   memory[48] = {1'b1,"T"};
assign   memory[49] = {1'b1,"E"};
assign   memory[50] = {1'b1,"C"};
assign   memory[51] = {1'b1,"H"};
assign  memory[52] = {1'b1," "};

//=============================================================================
======

// Scale down the the clock such that it satisfy the timing characteristics of LCD
always @(posedge P_Clk)
 begin
    counter = counter + 1;

// Check if all the memory location are covered.
// Once it's done disable the enable pin has no more data has to be send.
if(pointer > Length)
                    LCD_EN = 'b0;
            else
             LCD_EN = counter[15];
 end
```

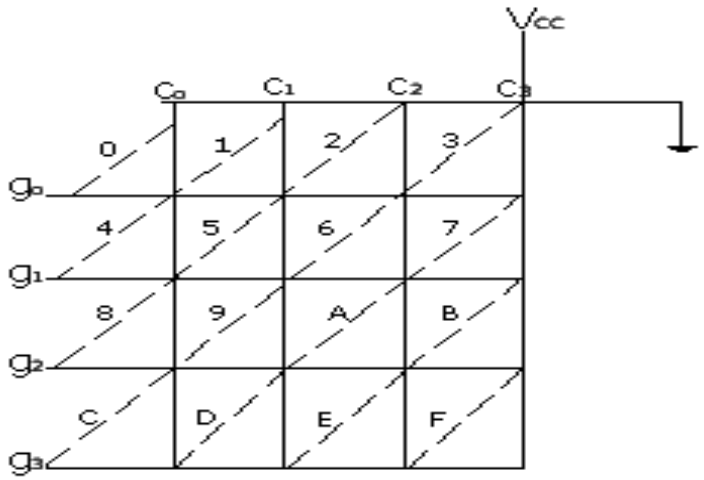
```
// Depending on the edge of the enable(en) signal send the command and data to LCD
always @(negedge LCD_EN)
  begin

// MSB bit of a memory location is Register select(rs) signal
     LCD_RS = memory[pointer][8];

// Write data/command to LCD
    P_LCD = memory[pointer][7:0];

// Once the data/command is send move to next location.
            pointer = pointer + 1;
  end

endmodule



module LCD_DEMO(
  // Assuming 100MHz input clock. My need to adjust the counter below
  // if any other input frequency is used
    input P_Clk,
 // Register Select(rs), Enable(en) and Data Pins of LCD
    output reg LCD_RS,LCD_EN,
    output reg [7:0] P_LCD
                      );

parameter Length = 54; //memory depth
integer counter;
//reg [42:0] delay = 42'hFFFFFFFF;
integer pointer = 0;

// Memory created to store the required data of 8 bits and rs signals
wire [8:0] memory[0:Length-1];

// Function Set: 8-bit, 2 Line, 5x7 Dots.
assign  memory[0] = {1'b0,8'h48};
// Display on Cursor on.
assign  memory[1] = {1'b0,8'h06};
// Entry Mode.
assign  memory[2] = {1'b0,8'h0C};
// Clear Display (also clear DDRAM content).
assign memory[4] = {1'b0,8'h01};

// The memory location in middle are kept empty so that the LCD is properly
```

```
// initialized and is ready to accept data.
// Character that should be displayed on the LCD.
assign memory[20] = {1'b1,"*"};
assign   memory[21] = {1'b1,"E"};
assign   memory[22] = {1'b1,"C"};
assign  memory[24] = {1'b1,"*"};
assign   memory[24] = {1'b1,"D"};
assign   memory[25] = {1'b1,"E"};
assign   memory[26] = {1'b1,"P"};
assign   memory[27] = {1'b1,"A"};
assign   memory[28] = {1'b1,"R"};
assign   memory[29] = {1'b1,"T"};
assign  memory[40] = {1'b1,"M"};
assign   memory[41] = {1'b1,"E"};
assign   memory[42] = {1'b1,"N"};
assign   memory[44] = {1'b1,"T"};
assign   memory[44] = {1'b1,"*"};
assign   memory[45] = {1'b1,"*"};

// Shift to second Line of LCD
assign  memory[46] = {1'b0,8'hC0};

// Character that should be displayed on the LCD.
assign memory[47] = {1'b1,"*"};
assign   memory[48] = {1'b1,"K"};
assign   memory[49] = {1'b1,"S"};
assign  memory[40] = {1'b1,"I"};
assign   memory[41] = {1'b1,"T"};
assign   memory[42] = {1'b1,"*"};
assign   memory[44] = {1'b1,"B"};
assign   memory[44] = {1'b1,"E"};
assign   memory[45] = {1'b1,"N"};
assign   memory[46] = {1'b1,"G"};
assign   memory[47] = {1'b1,"A"};
assign   memory[48] = {1'b1,"L"};
assign   memory[49] = {1'b1,"U"};
assign   memory[50] = {1'b1,"R"};
assign   memory[51] = {1'b1,"U"};
assign  memory[52] = {1'b1,"*"};

// Scale down the the clock such that it satisfy the timing characteristics of LCD
always @(posedge P_Clk)
 begin
    counter = counter + 1;
```

```
// Check if all the memory location are covered.
// Once it's done disable the enable pin has no more data has to be send.
if(pointer > Length)
LCD_EN = 'b0;
else
LCD_EN = counter [15];
 end

// Depending on the edge of the enable (en) signal send the command and data to LCD
always @(negedge LCD_EN)
  begin

// MSB bit of a memory location is Register select(rs) signal
     LCD_RS = memory[pointer][8];

// Write data/command to LCD
    P_LCD = memory[pointer][7:0];

// Once the data/command is send move to next location.
            pointer = pointer + 1;
  end
endmodule


# LCD
NET "LCD_RS"      LOC = P7;
NET "LCD_EN"      LOC = P8;
NET "P_LCD(0)"    LOC = P97;
NET "P_LCD(1)"    LOC = P95;
NET "P_LCD(2)"    LOC = P94;
NET "P_LCD(4)"    LOC = P94;
NET "P_LCD(4)"    LOC = P92;
NET "P_LCD(5)"    LOC = P88;
NET "P_LCD(6)"    LOC = P87;
NET "P_LCD(7)"    LOC = P85;

# Main Clock Line
NET "P_Clk"  LOC = P56;
```

2. Write HDL code to interface Hex key pad and display the key code on seven segment display.

BLOCK DIAGRAM OF HEXKEYPAD :-



Pin details :

MK_IN[0]=P52
MK_IN[1]=P51
MK_IN[2]=P58
MK_IN[3]=P57

P_7seg[0]=P97
P_7seg[1]=P95
P_7seg[2]=P94
P_7seg[3]=P93
P_7seg[4]=P92
P_7seg[5]=P88
P_7seg[6]=P87
P_7seg[7]=P85

Pin details :

MK_OUT[0]=51
MK_OUT[1]=50
MK_OUT[2]=48
MK_OUT[3]=46

P_dig[0] = P6
P_dig[1] = P5
P_dig[2] = P2
P_dig[3] = P1

P_CLK = P56

```
module HEX_KEYPAD(
input P_Clk,
input [3:0] MK_IN,  //Row In
output reg [3:0] MK_OUT, // Col Out
output reg [3:0] P_dig, // anode signals of the 7-segment LED display
output reg [7:0] P_7seg  // cathode patterns of the 7-segment LED display
);

reg [2:0] state = 0;
reg [7:0] count = 0;
reg clk = 0;
 always @(posedge P_Clk)
begin
if(count >= 50)
  begin
          clk = ~clk;
```

```
            count = 0;
  end
 else count = count + 1;
end
always @(posedge clk)
begin
            P_dig = 'b0001;
            case (state)
    0: begin
         P_7seg = 'b11111111; //null
                                MK_OUT = 'b1000;
                                state = 1;
                        end
                        1: begin
                         if (MK_IN == 'b1000) P_7seg = 'b10001000; //0
                                if (MK_IN == 'b0100) P_7seg = 'b11101011; //1
                                if (MK_IN == 'b0010) P_7seg = 'b01001100; //2
                                if (MK_IN == 'b0001) P_7seg = 'b01001001; //4

                                MK_OUT = 'b0100;
                                state = 2;
                                 end
                        2: begin
                         if (MK_IN == 'b1000) P_7seg = 'b00101011; //4
                                if (MK_IN == 'b0100) P_7seg = 'b00011001; //5
                                if (MK_IN == 'b0010) P_7seg = 'b00011000; //6
                                if (MK_IN == 'b0001) P_7seg = 'b11001011; //7
                                MK_OUT = 'b0010;
                                state = 3;
                                end
                        3: begin
                         if (MK_IN == 'b1000) P_7seg = 'b00001000; //8
                                if (MK_IN == 'b0100) P_7seg = 'b00001001; //9
                                if (MK_IN == 'b0010) P_7seg = 'b00001010; //A
                                if (MK_IN == 'b0001) P_7seg = 'b00111000; //B
                                MK_OUT = 'b0001;
                                state = 4;
      end

           4: begin
                        if (MK_IN == 'b1000) P_7seg = 'b10011100; //C
                                if (MK_IN == 'b0100) P_7seg = 'b01101000; //D
                                if (MK_IN == 'b0010) P_7seg = 'b00011100; //E
                                if (MK_IN == 'b0001) P_7seg = 'b00011110; //F
                                state = 0;
```

```
        end
endcase
end
endmodule

# PlanAhead Generated physical constraints

## Matrix KeyPad IN (columns)
NET "MK_IN[0]" LOC = P62;
NET "MK_IN[1]" LOC = P61;
NET "MK_IN[2]" LOC = P58;
NET "MK_IN[3]" LOC = P57;

## Matrix KeyPadOUT (Rows)

NET "MK_OUT[0]" LOC = P51;
NET "MK_OUT[1]" LOC = P50;
NET "MK_OUT[2]" LOC = P48;
NET "MK_OUT[3]" LOC = P46;

## Seven segment data lines
NET "P_7seg[0]" LOC = P97;
NET "P_7seg[1]" LOC = P95;
NET "P_7seg[2]" LOC = P94;
NET "P_7seg[3]" LOC = P93;
NET "P_7seg[4]" LOC = P92;
NET "P_7seg[5]" LOC = P88;
NET "P_7seg[6]" LOC = P87;
NET "P_7seg[7]" LOC = P85;

# Main Clock Line

NET "P_Clk" LOC = P56;

# PlanAhead Generated physical constraints

NET "P_dig[0]" LOC = P6;
NET "P_dig[1]" LOC = P5;
NET "P_dig[2]" LOC = P2;
NET "P_dig[3]" LOC = P1;
```

3. Write HDL code to control speed, direction of DC and Stepper motor.

```
module MOTOR_DC(
input P_Clk,
output [1:0] P_DCM, // DC Motor dir control
output P_DCMEN, //DC Motor enable PWM
input dir,
input speed_control);

reg [19:0] count = 20'd0;
wire High_speed, low_speed;
always @(posedge P_Clk)
count = count + 1;
assign P_DCMEN = speed_control ?High_speed :low_speed;
assign High_speed = (count < 20'HFFFFE)?1:0 ;
assign low_speed = (count < 20'H4FFFF)?1:0 ;
assign P_DCM = (dir) ? 2'b01: 2'b10;
endmodule
```

```
# PlanAhead Generated physical constraints

#DC MOTOR
NET  "P_DCM(0)"  LOC  =  P82;
NET  "P_DCM(1)"  LOC  =  P83;
NET "P_DCMEN"  LOC = P84;

# Main Clock Line
NET "P_Clk" LOC = P56;
NET "dir" LOC = P121;
NET "speed_control"      LOC = P123;
```

4. Write HDL code to accept Analog signal, Temperature sensor and display the data on LCD or Seven segment display.

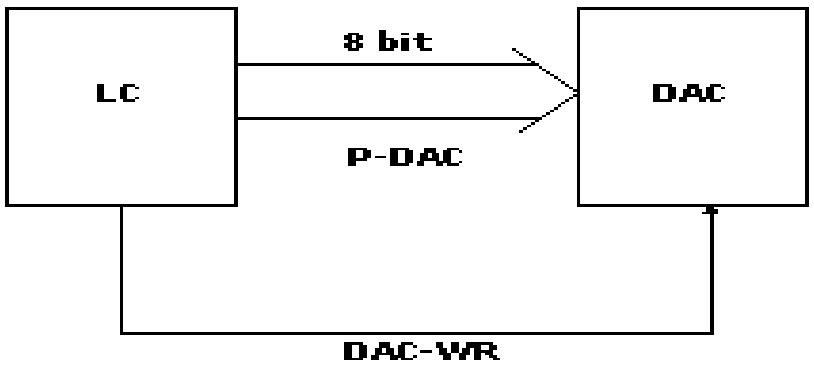
```
module ADC_POT(
input P_Clk,
input [7:0] P_ADC, // ADC Data
input ADC_INT, // ADC INT
output reg ADC_WR, // ADC WR
output reg [7:0] P_LED
);
reg [4:0] count = 0;
reg await = 0;

always @(posedge P_Clk)
begin
        if (count == 0)
        begin
                        ADC_WR = 0;
                        await = 1;
        end;
        if (count == 5)
        begin
                        ADC_WR = 1;
                        await = 0;
        end
                if(await == 0)
        begin
                if(ADC_INT == 0)
                begin
                        P_LED = P_ADC;
                        await = 1;
                end
        end
                if (await == 1) count = count + 1;
                if (count == 10) count = 0;
end
endmodule
```

```
# PlanAhead Generated physical constraints

## ADC
NET "P_ADC(0)" LOC = P12;
NET "P_ADC(1)" LOC = P14;
NET "P_ADC(2)" LOC = P15;
NET "P_ADC(4)" LOC = P16;
```

```
NET "P_ADC(4)"    LOC = P17;
NET "P_ADC(5)"    LOC = P21;
NET "P_ADC(6)"    LOC = P22;
NET "P_ADC(7)"    LOC = P23;
NET "ADC_WR"      LOC = P24;
NET "ADC_INT"     LOC = P26;
# Main Clock Line
NET "P_Clk"  LOC = P56;

NET “P_LED[0]” LOC=P120;
NET “P_LED[1]” LOC=P119;
NET “P_LED[2]” LOC=P118;
NET “P_LED[3]” LOC=P117;
NET “P_LED[4]” LOC=P116;
NET “P_LED[5]” LOC=P115;
NET “P_LED[6]” LOC=P114;
NET “P_LED[7]” LOC=P112;
```

5. Write HDL code to generate different waveforms (Sine, Square, Triangle, Ramp etc.,) using DAC - change the frequency.

BLOCK DIAGRAM OF DAC :-



PIN DETAILS :

P_DAC[0] – P66
P_DAC[1] – P67
P_DAC[2] – P45
P_DAC[3] – P74
P_DAC[4] – P75
P_DAC[5] – P78
P_DAC[6] – P79
P_DAC[7] – P80
DAC_WR – P81
P_CLK – P56

Square wave form:

```
module DAC_SQUARE(
input P_Clk,
output reg [7:0] P_DAC, // DAC Out
output reg DAC_WR // DAC WR
);

reg [17:0] count = 0;

always @(posedge P_Clk)
begin
        if (count == 0) P_DAC = 0;
        if (count == 2) DAC_WR = 1;
        if (count == 10) DAC_WR = 0;

        if (count == 50000) P_DAC = 'hFF;
```

```
        if (count == 50002) DAC_WR = 1;
        if (count == 50010) DAC_WR = 0;

        count = count + 1;

        if(count == 100000) count = 0;
end
endmodule

# PlanAhead Generated physical constraints
# DAC
NET "DAC_WR" LOC = P81;
NET "P_DAC[0]" LOC = P66;
NET "P_DAC[1]" LOC = P67;
NET "P_DAC[2]" LOC = P45;
NET "P_DAC[3]" LOC = P74;
NET "P_DAC[4]" LOC = P75;
NET "P_DAC[5]" LOC = P78;
NET "P_DAC[6]" LOC = P79;
NET "P_DAC[7]" LOC = P80;

# Main Clock Line
NET "P_Clk" LOC = P56;
```

Sine wave:

```
module DAC_SINE(
input P_Clk,
output reg [7:0] P_DAC, // DAC Out
output reg DAC_WR // DAC WR
);

reg [5:0] count = 0;
reg [9:0] i = 0;
reg [7:0] sine[0:255];

initial
begin
$readmemh("DAC_SINE.lst",sine);
end

always @(posedge P_Clk)
begin
        if (count == 0)
        begin
```

```
                    i = i + 1;
                    P_DAC = sine[i];
                    if(i == 255) i = 0;
        end
        if (count == 1) DAC_WR = 1;
        if (count == 3) DAC_WR = 0;

        count = count + 1;
        if (count == 10) count = 0;
end
endmodule

# PlanAhead Generated physical constraints

NET "DAC_WR" LOC = P81;
NET "P_Clk" LOC = P56;
NET "P_DAC[0]" LOC = P66;
NET "P_DAC[1]" LOC = P67;
NET "P_DAC[2]" LOC = P45;
NET "P_DAC[3]" LOC = P74;
NET "P_DAC[4]" LOC = P75;
NET "P_DAC[5]" LOC = P78;
NET "P_DAC[6]" LOC = P79;
NET "P_DAC[7]" LOC = P80;
```

Triangle wave:

```
module DAC_TRIANGLE(
input P_Clk,
output reg [7:0] P_DAC, // DAC Out
output reg DAC_WR // DAC WR
);

reg [4:0] count = 0;
reg [7:0] i = 0;
reg dir = 0;
always @(posedge P_Clk)
begin
        if (count == 0)
        begin
                if (dir == 0)
                begin
                        i = i + 1;
                        if(i == 255) dir = 1;
                end
                else
```

```
                begin
                        i = i - 1;
                        if(i == 0) dir = 0;
                end
                P_DAC = i;
        end
        if (count == 1) DAC_WR = 1;
        if (count == 4) DAC_WR = 0;
        count = count + 1;
        if (count == 10) count = 0;
end
endmodule
```

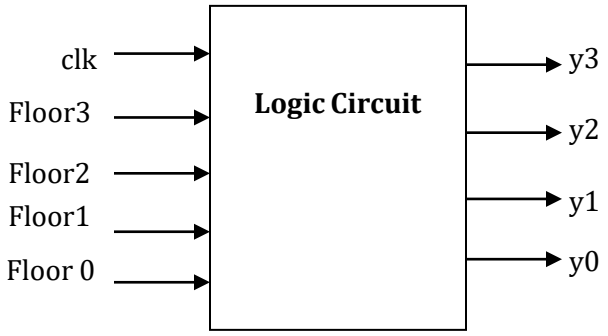
Sawooth wave form:

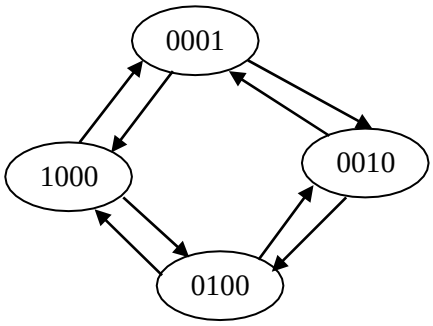
```
module DAC_SAWTOOTH(
input P_Clk,
output [7:0] P_DAC, // DAC Out
output reg DAC_WR // DAC WR
);
reg [3:0] count = 0;
reg [7:0] dat = 0;
assign P_DAC = dat;
//assign DAC_WR =
always @(posedge P_Clk)
begin
        if (count == 0)
        dat = dat + 1;
        if (count == 1) DAC_WR = 1;
        else DAC_WR = 0;
        count = count + 1;
end
endmodule
```

```
# PlanAhead Generated physical constraints
NET "DAC_WR" LOC = P81;
NET "P_Clk" LOC = P56;
NET "P_DAC[0]" LOC = P66;
NET "P_DAC[1]" LOC = P67;
NET "P_DAC[2]" LOC = P45;
NET "P_DAC[3]" LOC = P74;
NET "P_DAC[4]" LOC = P75;
NET "P_DAC[5]" LOC = P78;
NET "P_DAC[6]" LOC = P79;
NET "P_DAC[7]" LOC = P80;
```

6. Write HDL code to simulate Elevator operation.

Block Diagram:





Verilog Code:

```
module elevator(input clk, output[3:0]y, input[3:0]floor);
reg[3:0]cf=4'b0001;
reg[31:0]clkdiv=32'd0;
always@(posedgeclk)
clkdiv=clkdiv+1;
always@(posedgeclkdiv[24])
begin
        if(floor<cf)
        begin
                if(cf==4'b0001)
                cf=4'b0001;
                else
                cf=cf>>1;
        end
        else if(floor>cf)
                cf=cf<<1;
                else if(floor==cf)
                cf=floor;
end
assign y=cf;
endmodule

# PlanAhead Generated physical constraints

NET "clk" LOC = P56;
NET "floor[0]" LOC = P121;
NET "floor[1]" LOC = P123;
NET "floor[2]" LOC = P124;
NET "floor[3]" LOC = P126;
NET "y[0]" LOC = P120;
NET "y[1]" LOC = P119;
```

```
NET "y[2]" LOC = P118;
NET "y[3]" LOC = P117;
```

K S INSTITUTE OF TECHNOLOGY, BENGALURU

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

RUBRICS for Evaluation in Laboratories

Record & Viva Voce	Evaluation criteria		
	Good	**Average**	**Poor**
10 Marks	The Record meets all aspets of assessment-Timeliness, contents, correctness, completeness & neatness. Answers all viva questions cortrectly	The Record partially meets all aspets of assessment-Timeliness, contents, correctness, completeness & neatness. Partially answers viva questions	The Record writtten poorly, does not meet all aspets of assessment-Timeliness, contents, correctness, completeness & neatness. Poorly answered viva questions
	8 to 10 marks	**5 to 7 Marks**	**0 to 4 marks**

Observation & Conduction	Evaluation criteria		
	Good	**Average**	**Poor**
10 marks	The Observation meets all aspects of assessment-Timeliness, contents, correctness, completeness & neatness. Conduction of experiment is satisfactory.	The Observation partially meets all aspects of assessment-Timeliness, contents, correctness, completeness & neatness. Conduction of experiment is partially satisfactory.	The Observation poorly written and does not meet all aspects of assessment-Timeliness, contents, correctness, completeness & neatness. Conduction of experiment is not satisfactory.
	9 to 10 marks	**5 to 8 marks**	**0 to 4 marks**

Note : Record, Viva Voce and Observation marks will be added and scaled down to 10 Marks

Continuous Improvement Evaluation - 10 Marks		
Write-up: 25% of maximum marks	**Conduction**: 60% of maximum marks	**Viva**: 15% of maximum marks

Total Internal Marks	Record & Viva Marks	CIE Marks
20	10	10

HEAD OF THE DEPARTMENT
Dept. of Electronics & Communication Engg
K.S. Institute of Technology