DEPARTMENT

OF

ELECTRONICS & COMMUNICATION ENGINEERING HDL LABORATORY MANUAL

V Semester (19EC5DLHDL)

Autonomous Course 2021

HOD : Dr. Manjunath T C

In -Charge: Dr. Rajagopal A.

Name of the Student	**:**	
Semester /Section	**:**	
USN	**:**	
Batch	**:**	

Dayananda Sagar College of Engineering Shavige Malleshwara Hills, Kumaraswamy Layout,

Banashankari, Bangalore-560078, Karnataka

Tel : +91 80 26662226 26661104 Extn : 2731 Fax : +90 80 2666 0789 Web - http://www.dayanandasagar.edu Email : hod-ece@dayanandasagar.edu (An Autonomous Institute Affiliated to VTU, Approved by AICTE & ISO 9001:2008 Certified) (Accredited by NBA, National Assessment & Accreditation Council (NAAC) with 'A' grade)

HDL LAB

1. Write HDL code to realize all the logic gates.
2. Write a HDL code to describe the functions of a Full Adder using three modelling styles and full adder using Half adder.
3. Write HDL code for 4-bit ripple carry adder using full adders.
4. Write HDL code for Adder/Subtractor circuit
5. Write a HDL 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. De-multiplexer
    e. 4 bit Binary to Gray converter.
    f. 4 bit Gray to Binary converter
    g. Comparator
6. Write HDL code for BCD Adder
7. Develop the HDL code for the following flip-flops
    a. D Flip-Flop
    b. JK Flip-Flop
    c. T Flip-Flop
8. Write a model for 8bit ALU. ALU should use combinational logic to calculate an output based on the 3-bit op-code input. ALU should decode the 3-bit op-code according to the given in example below. Opcode (2:0)

    OPCODE ALU OPERATION
    1. A + B
    2. A – B
    3. A Complement
    4. A * B
    5. A AND B
    6. A OR B
    7. A NAND B
    8. A XOR B
9. Design 4 bit Binary counters (Synchronous reset and Asynchronous reset) and any sequence Counters.
10. Design 4 bit BCD counters(Synchronous reset and Asynchronous reset)
11. Design Mealy and Moore state machine for a given sequence.
12. Stepper motor
13. VGA Display
14. RAM

TOOLS used: Modelsim/Quartus Prime Standard edition 20.1/De-Sim/Iverilog/Xilinx ISE.

Experiment No: 1 Date: _____

LOGIC GATES

Aim: Write a HDL code to realize all LOGIC GATES.

Verilog

```
module logicgates(a,b,y);
input a,b;
output[6:0] y;
assign y[0] = ~a;
assign y[1] = a & b;
assign y[2] = a|b;
assign y[3] = ~(a&b);
assign y[4] = ~(a|b);
assign y[5] = a^b;
assign y[6] = ~(a^b);
endmodule
```

Test Bench

Results

Experiment No: 2 Date: _____

FULL ADDER

Aim: Write a HDL code FULL ADDER in all styles.

1. DATA FLOW

2. BEHAVIORAL

3. STRUCTURAL

1. DATA FLOW

Verilog

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

2. BEHAVIORAL

Verilog

```
 module fulladder_bh( a, b, cin, sum, cout);
 input a, b, cin;
 output sum,cout;
   reg sum,cout;
  always @(a,b,cin)
  {cout,sum}=a+b+cin;
endmodule
```

3. STRUCTURAL

Verilog

```
module fulladder_str (a, b, cin, sum, cout); input a, b, cin;
output sum, cout; wire s1,s2,s3;
xor x1(sum,a,b,cin);
```

and a1(s1,a,b);

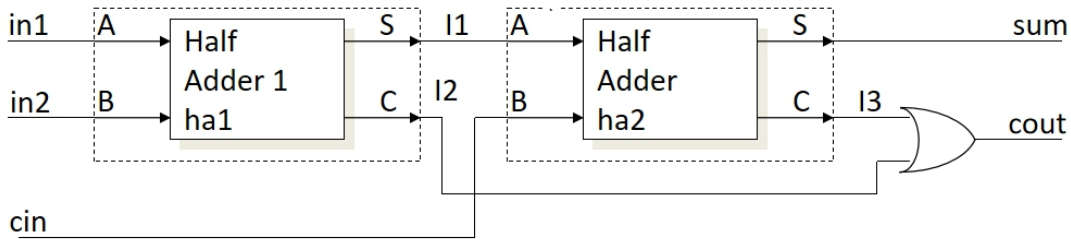
and a2(s2,a,cin);

and a3(s3,b,cin);

or o1(cout,s1,s2,s3);

endmodule

Aim: Full adder using Halfadder



module half_adder(S, C, A, B);

output S, C;

input A, B;

assign S = A ^ B;

assign C = A & B;

endmodule

module full_adder(sum, cout, in1, in2, cin);

output sum, cout;

input in1, in2, cin;

wire I1, I2, I3;

half_adder ha1(I1, I2, in1, in2);

half_adder ha2(sum, I3, I1, cin);

assign cout = I2 || I3;

endmodule

Test Bench

Results

Experiment No: 3 Date: _____

RIPPLE CARRY ADDER

Aim: Write a HDL code for Ripple Carry Adder

Verilog

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

module ripple_add(a,b,cin,s,carry,);
output [3:0]s;
output carry;
input [3:0]a,b;
input cin;
wire c1,c2,c3;
fulladder f1(a[0],b[0],cin,s[0],c1);
fulladder f2(a[1],b[1],c1,s[1],c2);
fulladder f3(a[2],b[2],c2,s[2],c3);
fulladder f4(a[3],b[3],c3,s[3],carry);
endmodule
```

Test Bench:

Results

Experiment No: 4 Date: _____

ADDER SUBTRACTOR

Aim: Write a HDL code for Ripple Carry Adder Subtractor

Verilog

```
module adder_subtractor(S, C, V, A, B, Op);
  output [3:0] S;   // The 4-bit sum/difference.
  output       C;   // The 1-bit carry/borrow status.
  output       V;   // The 1-bit overflow status.
  input [3:0]  A;   // The 4-bit augend/minuend.
  input [3:0]  B;   // The 4-bit addend/subtrahend.
  input        Op;  // The operation: 0 => Add, 1=>Subtract.

  wire  C0; // The carry out bit of fa0, the carry in bit of fa1.
  wire  C1; // The carry out bit of fa1, the carry in bit of fa2.
  wire  C2; // The carry out bit of fa2, the carry in bit of fa3.
  wire  C3; // The carry out bit of fa2, used to generate final carry/borrrow.

  wire  B0; // The xor'd result of B[0] and Op
  wire  B1; // The xor'd result of B[1] and Op
  wire  B2; // The xor'd result of B[2] and Op
  wire  B3; // The xor'd result of B[3] and Op
   // Looking at the truth table for xor we see that
  // B xor 0 = B, and
  // B xor 1 = not(B).
  // So, if Op==1 means we are subtracting, then
  // adding A and B xor Op alog with setting the first
  // carry bit to Op, will give us a result of
  // A+B when Op==0, and A+not(B)+1 when Op==1.
```

```
  // Note that not(B)+1 is the 2's complement of B, so
  // this gives us subtraction.
  xor(B0, B[0], Op);
  xor(B1, B[1], Op);
  xor(B2, B[2], Op);
  xor(B3, B[3], Op);
  xor(C, C3, Op);     // Carry = C3 for addition, Carry = not(C3) for subtraction.
  xor(V, C3, C2);     // If the two most significant carry output bits differ, then we have an overflow.

  full_adder fa0(S[0], C0, A[0], B0, Op);    // Least significant bit.
  full_adder fa1(S[1], C1, A[1], B1, C0);
  full_adder fa2(S[2], C2, A[2], B2, C1);
  full_adder fa3(S[3], C3, A[3], B3, C2);    // Most significant bit.
endmodule // ripple_carry_adder_subtractor

module full_adder (sum, cout,a, b, cin, );
input a, b, cin;
output sum, cout; wire s1,s2,s3;
xor x1(sum,a,b,cin);
and a1(s1,a,b);
and a2(s2,a,cin);
and a3(s3,b,cin);
 or o1(cout,s1,s2,s3);
endmodule
```

Test Bench:

Results

Experiment No:5a Date: _____

DECODER (2:4)

Aim: Write a HDL code to design 2:4 DECODER

Verilog

```
module decoder(I,D);
input [1:0] I;
output [3:0] D;
reg [3:0] D;
always @(I)
begin
if (I==2'B00) D=4'B0001;
else if (I==2'B01) D=4'B0010;
else if (I==2'B10) D=4'B0100;
else if (I==2'B11) D=4'B1000;
else D=4'BZZZZ;
end
endmodule
```

Test Bench

Results

Experiment No:5b Date: _____

ENCODER (8:3)

Aim: Write a HDL code to design 8:3 ENCODER with and without priority.

Without priority Verilog

```
module encoder_wop (I,D);
input [7:0] I;
output [2:0] D;
reg [2:0] D;
always @(I)
begin
case (I)
8'd1 : D=3'd0;
8'd2 : D=3'd1;
8'd4 : D=3'd2;
8'd8 : D=3'd3;
8'd16 : D=3'd4;
8'd32 : D=3'd5;
8'd64 : D=3'd6;
8'd128 : D=3'd7;
default: D=3'BZZZ;
endcase
end
endmodule
```

Test Bench

With Priority Verilog

```
module encoder_wp (I,D);
input [7:0] I;
output [2:0] D;
reg [2:0] D;
always @ (I)
begin
if (I[7]==1'B1) D=3'B111;
else if (I[6]==1'B1) D=3'B110;
else if (I[5]==1'B1) D=3'B101;
else if (I[4]==1'B1) D=3'B100;
else if (I[3]==1'B1) D=3'B011;
else if (I[2]==1'B1) D=3'B010;
else if (I[1]==1'B1) D=3'B001;
else if (I[0]==1'B1) D=3'B000;
else D=3'BZZZ;
end
endmodule
```

Test Bench

Results

Experiment No: 5c Date: _____

MULTIPLEXER (8:1)

Aim: Write a HDL code to design 8:1 MULTEPLEXER

Verilog

```
module mux8_1 (I, S, D);
input [7:0] I;
input [2:0] S;
output D;
reg D;
always @ (I, S)
begin
 case(S)
 3’d0: D=I[0];
 3’d1: D=I[1];
 3’d2: D=I[2];
 3’d3: D=I[3];
 3’d4: D=I[4];
 3’d5: D=I[5];
 3’d6: D=I[6];
 3’d7: D=I[7];
default: D=1'BZ;
endcase
end
endmodule
```

Test Bench

Results

Experiment No: 5d Date: _____

DEMULTIPLEXER

Aim: Write a HDL code to design 1:8 DEMULTEPLEXER

Verilog

```
module demux1_8(I,S,D);
input I;
input [2:0] S;
output [7:0]D;
reg [7:0]D;
always @(I,S)
begin
D=8’d0;
 CASE (S)
 3'd0 : D[0]=I;
 3'd1 : D[1]=I;
 3'd2 : D[2]=I;
 3'd3 : D[3]=I;
 3'd4 : D[4]=I;
 3'd5 : D[5]=I;
 3'd6 : D[6]=I;
 3'd7 : D[7]=I;
default: D=8'B00000000;
endcase
end
endmodule
```

Test Bench

Results

Experiment No: 5e Date: _____

BINARY TO GRAY

Aim: Write a HDL code to convert 4-bit BINARY TO GRAY

Verilog

```
module bin_to_gray(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
```

Test Bench

Results

Experiment No: 5f Date: _____

GRAY TO BINARY

```
module gray_to_bin
      (input [3:0] G, //gray code output
       output [3:0] bin   //binary input );
 assign bin[3] = G[3];
assign bin[2] = G[3] ^ G[2];
assign bin[1] = G[3] ^ G[2] ^ G[1];
assign bin[0] = G[3] ^ G[2] ^ G[1] ^ G[0];
endmodule
```

Test Bench

Results

Experiment No: 5g Date: _____

COMPARATOR

Aim: Write a HDL code to design 4-bit comparator.

Verilog

```
module comparator (a, b, alb, aeb, agb)
input[3:0] a, b;
output alb, aeb, agb;
reg alb, aeb, agb;
always@(a,b)
begin
if(a<b)
begin
alb=1'b1; aeb=1'b0; agb=1'b0;
end
else if (a>b)
begin
alb=1'b0; aeb=1'b0; agb=1'b1;
end
else
begin
alb=1'b0; aeb=1'b1; agb=1'b0;
end
end
endmodule
```

Test Bench

Results

Experiment No: 6 Date: _____

BCD ADDER

Verilog

```
//module declaration with inputs and outputs
module bcd_adder(a,b,carry_in,sum,carry);
   input [3:0] a,b;
   input carry_in;
   output [3:0] sum;
   output carry;
   //Internal variables
   reg [4:0] sum_temp;
   reg [3:0] sum;
   reg carry;
  always @(a,b,carry_in)
   begin
      sum_temp = a+b+carry_in; //add all the inputs
      if(sum_temp > 9)    begin
         sum_temp = sum_temp+6; //add 6, if result is more than 9.
         carry = 1;  //set the carry output
         sum = sum_temp[3:0];    end
      else
          begin
         carry = 0;
         sum = sum_temp[3:0];
      end
   end
 endmodule
```

Test Bench

Results