HALDIA INSTITUTE OF TECHNOLOGY

Department of Electronics & Communication Engineering

VLSI DESIGN LAB MANUAL, EC-691

SYLLABUS

1. Design and Simulation of Half and Full adders,
2. Design and Simulation of Half and Full subtractor,
3. Design of MUX/DeMUX circuits.
4. Design of Serial Binary Adder and Ripple Carry Adder.
5. Design of 4-bit binary BCD counters (synchronous/ asynchronous reset).
6. Design of a N- bit shift register of Serial- in Serial –out, Serial in parallel out, Parallel in Serial out and Parallel in Parallel Out.
7. Design of Sequence Detector (Finite State Machine- Mealy and Moore Machines).
8. Design of 4- Bit Multiplier and 4-bit Divider.
9. Design of ALU to Perform – ADD, SUB, AND, OR, 1's compliment, 2's Compliment.
10. CMOS Inverter using SPICE simulation.
11. Design of amplifier using SPICE simulation.

Course outcomes (COs)

CO1:.Knowledge on modelling and synthesis of digital system design using HDL programming languages.

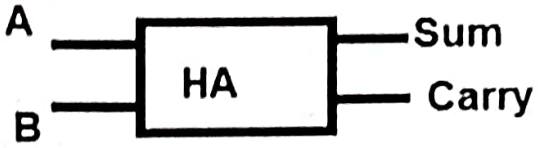
CO2:.Ability to design using FPGA/CPLD devices.

CO3: An exposure to critical path time calculations and RTL modules.

1. Design and Simulation of Half and Full adders.

Half adder:

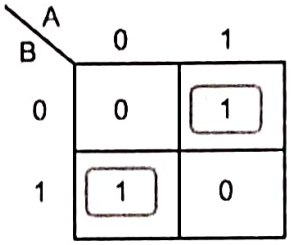
Half adder is also called as simple Binary Adder. It has two inputs A & B and outputs Sum & Carry. Sum is the XOR operation of inputs A and B. Carry is the AND operation of inputs A and B.



Truth Table:

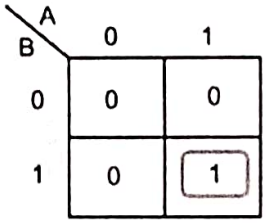
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Simplification using K-Map:



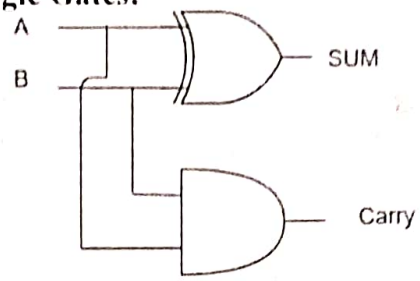
$S = A\,B' + A'\,B$

$S = A \; XOR \; B$



$Carry = A.B$

Implementation Using Logic Gates:



Verilog Code of Half Adder:

A: Dataflow Representation

```
module half_adder_dataflow (
input a, // Input 'a'
input b, // Input 'b'
output s, // Output 's' (Sum)
output c // Output 'c' (Carry)
);
assign s = a ^ b; // Dataflow expression for sum
assign c = a & b; // Dataflow expression for carry
endmodule
```

B: Structural Method

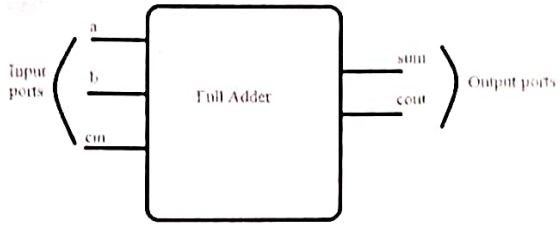
```
module half_adder_structural (
input a,   // Input 'a'
input b,   // Input 'b'
output s,  // Output 's' (Sum)
output c   // Output 'c' (Carry)
);
xor gate_xor (s, a, b); // XOR gate for sum
and gate_and (c, a, b); // AND gate for carry
endmodule
```

C: Behavioral Representation

```
module half_adder_behavioral (
input a, // Input 'a'
input b, // Input 'b'
output s, // Output 's' (Sum)
output c // Output 'c' (Carry)
);
// Combinational logic equations for sum and carry
always @(*) begin
s = a ^ b; // XOR operation for sum
c = a & b; // AND operation for carry
end

endmodule
```

Full Adder:

A full adder is a digital circuit in Verilog HDL that adds three binary numbers. It has two inputs for the numbers to be added, A and B, and one Carry-In input, Cin. The outputs are Sum, S, and Carry-Out, Cout.



Truth Table:

a	b	cin	sum	cout
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

Verilog Code for Full Adder:

A: Dataflow Representation

```
module full_adder_d (
   input a,b,cin,
```

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

B: Structural Representation

Verilog HDL code for Full Adder

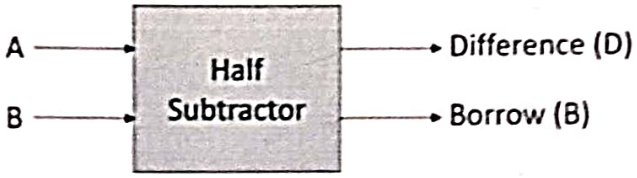
```
module full_add(a,b,cin,sum,cout);
  input a,b,cin;
  output sum,cout;
  wire x,y,z;
 // instantiate building blocks of full adder
  half_add h1(.a(a),.b(b),.s(x),.c(y));
  half_add h2(.a(x),.b(cin),.s(sum),.c(z));
  or o1(cout,y,z);
endmodule : full_add

// code your half adder design
module half_add(a,b,s,c);
  input a,b;
  output s,c;

// gate level design of half adder
  xor x1(s,a,b);
  and a1(c,a,b);
endmodule :half_add
```

2. Design and Simulation of Half and Full subtractor.

Half subtractor:

The half subtractor works opposite to the half adder as it substracts two single bits and results in a difference bit and borrow bit as an output.

Block Diagram:



Truth Table:

A	B	Difference (D)	Borrow (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Output:
D = A ^ B
B = A' · B

As half subtractor considers only two bits so along with the subtraction of two single bits, it cannot accommodate an extra borrow bit from the previously generated result. Hence, it is called a half-subtractor.

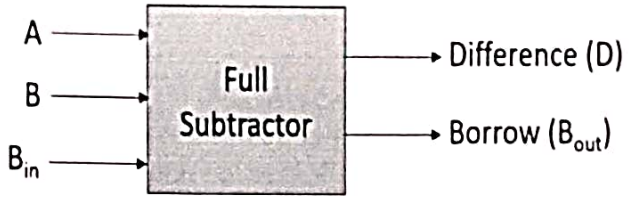
Half Subtractor using Verilog Code:

```
module half_subtractor(input a, b, output D, B);
  assign D = a ^ b;
  assign B = ~a & b;
endmodule
```

Full subtractor:

A full subtractor is designed to accommodate the extra borrow bit from the previous stage. Thus it has three single-bit inputs and produces two single-bit outputs.

Block Diagram:



Truth Table:

A	B	Bin	Difference (D)	Borrow (Bout)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Output: D = A ^ B ^ Bin

Bout = A' · B + (A ^ B)' · Bin

Full Subtractor using Verilog Code:

```
module full_subtractor(input a, b, Bin, output D, Bout);

 assign D = a ^ b ^ Bin;

 assign Bout = (~a & b) | (~(a ^ b) & Bin);

endmodule
```

3. Design of MUX/DeMUX circuits.

A multiplexer is a combinational type of digital circuit that is used to transfer one of the available input lines to a single output and, which input has to be transferred to the output will be decided by the state(logic 0 or logic 1) of the select line signal. 2:1 Multiplexer has two inputs, one select line (to select one of the two inputs), and a single output.

Truth Table:

select	out
0	in1
1	in2

2:1 MUX:

A. Verilog HDL code of 2:1 MUX:

```
module mux2_1(in1, in2, select, out);
input in1, in2, select;
output out;
assign out = select ? in2: in1;
endmodule
```

B. Verilog HDL code of 2:1 MUX:

```
module mux2_1(in1, in2, select, out);
input in1, in2, select;
output out;
assign out= (in1 & (~select))| (in2 & (select));
endmodule
```

4:1 MUX:

A. *Verilog HDL code of* 4:1 MUX (Dataflow):

```
module mux4x1(
   input s0,
   input s1,
   input a,
   input b,
   input c,
   input d,
   output y );
  assign y = (a & (~s1) & (~s0)) |(b & (~s1) & s0) |(c & s1 & (~s0)) |(d & s1 & s0);
endmodule
```

B: Verilog code of 4:1 MUX (Structural Modeling):

```
module mux4x1_gate_level(
   input a,
   input b,
   input c,
   input d,
   input s0,
   input s1,
   output y);
   wire n1, n2, n3, n4, n5, n6;
   not (n1, s1);
   not (n2, s0);
   and (n3, a, n1, n2);
   and (n4, b, n1, s0);
   and (n5, c, s1, n2);
   and (n6, d, s1, s0);
   or (y, n3, n4, n5, n6);
endmodule
```

C: 4-to-1 multiplexer by using gate level structural description. The module will take as arguments the following:

i. 4 input bits,
ii. 2 selection bits and
iii. 1 output bit.

Realize the operation of 4-to-1 multiplexer by instantiating required number of different logical gates and passing appropriate input and output parameters to each such logic gate.

Sample Test Cases

	Input	Output
Test Case 1	0	Pass: for s = 00 and a = 0001 out = 1
Test Case 2	1	Pass: for s = 00 and a = 0010 out = 0
Test Case 3	2	Pass: for s = 01 and a = 0010 out = 1
Test Case 4	3	Pass: for s = 10 and a = 0100 out = 1

```
//Write the verilog modules for 4-to-1 multiplexer here and
//keep the module name of 4-to-1 multiplexer "mux4x1"
module mux4x1(i, s, y );
input[3:0] i;
input[1:0] s;
output    y;
wire t1,t2,t3,t4,s0b,s1b;
not  n1( s0b, s[0] );
not n2(s1b, s[1]);
and  a1(t1, s0b, s1b,i[0]);
and a2(t2, s[0], s1b,i[1]);
and a3(t3, s0b, s[1],i[2] );
```

```
and a4(t4, s[0], s[1],i[3]);
or o1( y,t1,t2,t3,t4 );
endmodule
```

8-bit Multiplexer:

A: Four 8-bit signals A, B, C and D are to be multiplexed on an 8-bit bus called OUTBUS under the control of a 2-bit select control SEL. Write a behavioral specification to do this in Verilog.

Sample Test Cases

	Input	Output
Test Case 1	0	Pass: for A = 0, B = 15, C = 240, D = 255 and SEL = 00, OUTBUS = 0
Test Case 2	1	Pass: for A = 0, B = 15, C = 240, D = 255 and SEL = 11, OUTBUS = 255
Test Case 3	2	Pass: for A = 0, B = 15, C = 240, D = 255 and SEL = 10, OUTBUS = 240
Test Case 4	3	Pass: for A = 0, B = 15, C = 240, D = 255 and SEL = 01, OUTBUS = 15

```
//write the verilog modules to implement 8-bit multiplexer
module bus_multiplex (OUTBUS, A, B, C, D, SEL);
        input [7:0] A, B, C, D;
        input [1:0] SEL;
        output [7:0] OUTBUS;
assign OUTBUS =(SEL==0)? A : (SEL==1)? B: (SEL==2)? C: D;
endmodule
```

B: 8-bit Multiplexer_I" by writing a 4-to-1 multiplexer module, and instantiating it as many time as required to implement the required functionality.

Sample Test Cases

	Input	Output
Test Case 1	0	Pass: for A = 0, B = 15, C = 240, D = 255 and SEL = 10, OUTBUS = 240
Test Case 2	1	Pass: for A = 0, B = 15, C = 240, D = 255 and SEL = 11, OUTBUS = 255
Test Case 3	2	Pass: for A = 0, B = 15, C = 240, D = 255 and SEL = 00, OUTBUS = 0
Test Case 4	3	Pass: for A = 0, B = 15, C = 240, D = 255 and SEL = 01, OUTBUS = 15

```
//write the verilog modules to implement 8-bit multiplexer
module bus_multiplex (OUTBUS, A, B, C, D, SEL);
        input [7:0] A, B, C, D;
        input [1:0] SEL;
        output [7:0] OUTBUS;
wire [7:0] t;
wire [7:0] OUTBUS;
mux4 m0 (t[0], A[0], B[0], C[0], D[0], SEL);
mux4 m1 (t[1], A[1], B[1], C[1], D[1], SEL);
mux4 m2 (t[2], A[2], B[2], C[2], D[2], SEL);
mux4 m3 (t[3], A[3], B[3], C[3], D[3], SEL);
```

```
mux4 m4 (t[4], A[4], B[4], C[4], D[4], SEL);
mux4 m5 (t[5], A[5], B[5], C[5], D[5], SEL);
mux4 m6 (t[6], A[6], B[6], C[6], D[6], SEL);
mux4 m7 (t[7], A[7], B[7], C[7], D[7], SEL);
assign OUTBUS={t[7],t[6],t[5],t[4],t[3],t[2],t[1],t[0]};
endmodule

module mux4(out, a, b, c, d, sel);
          input a, b, c, d;
          input [1:0] sel;
           output out;
wire t1,t2,t3,t4,s0b,s1b;
 not  n1( s0b, sel[0] );
 not n2(s1b, sel[1]);
 and  a1(t1, s0b, s1b,a);
 and a2(t2, sel[0], s1b,b);
 and a3(t3, s0b, sel[1],c );
 and a4(t4, sel[0], sel[1],d);
 or o1( out,t1,t2,t3,t4);
endmodule
```

1:2 Demux:

A: Verilog Code of 1:2 Demux:

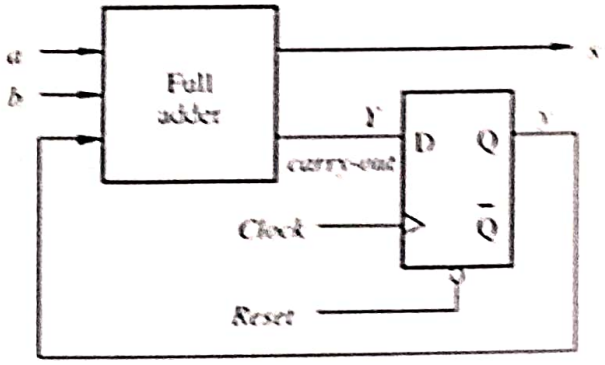
```
module demux_2_1(
  input sel,
  input i,
  output y0, y1);
 assign {y0,y1} = sel? {1'b0,i}: {i,1'b0};
 endmodule
```

B: 1:4 Demux Verilog Code

```
module demux_1_4(
 input [1:0] sel,
 input  i,
 output reg y0,y1,y2,y3);
 always @(*) begin
  case(sel)
   2'h0: {y0,y1,y2,y3} = {i,3'b0};
   2'h1: {y0,y1,y2,y3} = {1'b0,i,2'b0};
   2'h2: {y0,y1,y2,y3} = {2'b0,i,1'b0};
   2'h3: {y0,y1,y2,y3} = {3'b0,i};
   default: $display("Invalid sel input");
  endcase
 end
endmodule
```

4. Design of Serial Binary Adder and Ripple Carry Adder

A. Serial Adder:




```
//serial adder for N bits. Note that we dont have to mention N here.
module serial_adder
   (  input clk,reset,  //clock and reset
 input a,b,cin,  //note that cin is used for only first iteration.
output reg s,cout  //s comes out at every clock cycle and cout is valid only for last clock
cycle.
      );
reg c, flag;
always@(posedge clk or posedge reset)
begin
   if(reset == 1) begin //active high reset
      s = 0;
      cout = c;
      flag = 0;
   end else begin
      if(flag == 0) begin
         c = cin;  //on first iteration after reset, assign cin to c.
         flag = 1;  //then make flag 1, so that this if statement isnt executed any more.
      end
      cout = 0;
      s = a ^ b ^ c;  //SUM
      c = (a & b) | (c & b) | (a & c);  //CARRY
   end
end
endmodule
```

//Main module

```
module serial_adder(data_a, data_b, clk, reset, out, cout);
```

```
 input [3:0] data_a, data_b;
 input clk, reset;
 output cout;
 output [3:0] out;
 reg [3:0] out;
 reg [2:0] count;
 reg enable, cout;
 wire wire_a, wire_b, cout_temp, cin, sum;
 piso piso_a(clk, enable, reset, data_a, wire_a);
 piso piso_b(clk, enable, reset, data_b, wire_b);
 full_adder adder(wire_a, wire_b, cin, sum, cout_temp);
 d_flipflop dff(cout_temp, clk, enable, reset, cin);
 always @ (posedge clk or posedge reset) begin
  if (reset) begin
   enable = 1; count = 3'b000; out = 4'b0000;
  end
  else begin
   if (count > 3'b100)
    enable = 0;
   else begin
    if (enable) begin
     cout = cout_temp;
     count = count + 1;
     out = out >> 1;
     out[3] = sum;
    end
   end
  end
 end
endmodule
```

// PISO

```
module piso(clk, enable, rst, data, out);
 input enable, clk, rst;
 input [3:0] data;
 output out;
 reg out;
 reg [3:0] memory;
 always @ (posedge clk, posedge rst) begin
  if (rst == 1'b1) begin
   out <= 1'b0;
   memory <= data;
  end
  else begin
   if (enable) begin
```

```
      out = memory[0];
      memory = memory >> 1'b1;
    end
   end
  end
endmodule
```

//Full adder

```
module full_adder(a, b, cin, sum, cout);
  input a, b, cin;
  output sum, cout;
  assign {cout, sum} = a + b + cin;
endmodule
```

//D FF

```
module d_flipflop(d, clk, enable, reset, out);
  input d, clk, enable, reset;
  output out;
  reg out;
  always @ (posedge clk or posedge reset) begin
    if (reset)
      out = 0;
    else
      if (enable)
        out = d;
  end
endmodule
```

B: 4-bit Ripple Carry Adder

//main module 1

```
module ripple_carry_adder(in0, in1, out, cout);
                input [3:0] in0;
                input [3:0] in1;
                output [3:0] out;
                output cout;

                wire c1, c2, c3;
                full_adder fa0(in0[0], in1[0], 0, out[0], c1);
                full_adder fa1(in0[1], in1[1], c1, out[1], c2);
                full_adder fa2(in0[2], in1[2], c2, out[2], c3);
                full_adder fa3(in0[3], in1[3], c3, out[3], cout);
        endmodule
```

//Full adder

```
module full_adder(in0, in1, cin, out, cout);
```

```
        input in0, in1, cin;
        output out, cout;
        assign out = in0 ^ in1 ^ cin;
        assign cout = ((in0 ^ in1) & cin) | (in0 & in1);
endmodule
```

A: Write a verilog module to implement a 4-bit ripple carry adder. The module will take the following arguments:

i. Two 4-bit inputs A and B,
ii. 1-bit carry input Cin,
iii. 4-bit output Sum,
iv. 1-bit carry output Cout

Implement the above module by instantiating four 1-bit full adder (A 1-bit full adder takes as arguments two input bits to add, a carry input bit, one bit sum and one bit output carry).

Sample Test Cases

	Input	Output
Test Case 1	0	PASS: 0000 + 0000 + 0 = 00000
Test Case 2	1	PASS: 0111 + 1100 + 1 = 10100
Test Case 3	2	PASS: 0011 + 1000 + 0 = 01011
Test Case 4	3	PASS: 1111 + 0001 + 1 = 10001

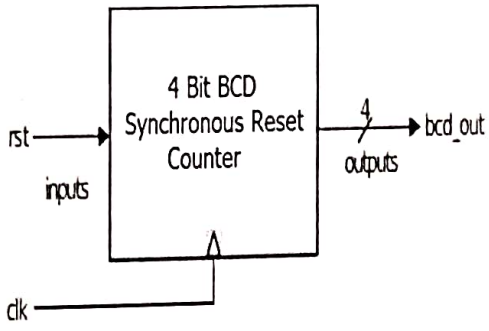
```
//write the verilog modules to implement 4-bit ripple carry adder and
//keep the module name of 4-bit adder "readder_4"
module readder_4 (A, B, Cin, Sum, Cout);
  input [3:0]A, B;
  input Cin;
  output [3:0]Sum;
  output Cout;
  wire c0, c1, c2;
  fulladder_1 M0 (A[0], B[0], Cin, Sum[0], c0);
  fulladder_1 M1 (A[1], B[1], c0, Sum[1], c1);
  fulladder_1 M2 (A[2], B[2], c1, Sum[2], c2);
  fulladder_1 M3 (A[3], B[3], c2, Sum[3], Cout);
  endmodule
module fulladder_1 (a, b, cin, sum, cout);
   input a, b, cin;
  output sum, cout;
   assign sum = a ^ b ^cin;
  assign cout = (a & b) | (b & cin) | (cin & a);
endmodule
```

5. Design of 4-bit binary BCD counters (synchronous/ asynchronous reset).

A: 4 Bit BCD counter with Synchronous Reset Counter :

The block diagram and truth table of 4 Bit BCD Synchronous Reset Counter Verilog Code is also mentioned.

Block Diagram:



Truth Table:

Clock	rst	bcd_out(3)	bcd_out(2)	bcd_out(1)	bcd_out(0)
↑	0	0	0	0	0
↑	1	0	0	0	0
↑	1	0	0	0	1
↑	1	0	0	1	0
↑	1	0	0	1	1
↑	1	0	1	0	0
↑	1	0	1	0	1
↑	1	0	1	1	0
↑	1	0	1	1	1
↑	1	1	0	0	0
↑	1	1	0	0	1
↑	1	0	0	0	0

Verilog Code of 4 Bit BCD Synchronous Reset Counter : (wrong)

```
module bin_sync( clk, rst, bcd_out);
input clk, rst;
output [3:0] bcd_out;
reg [3:0] bcd_out;
initial
begin
bcd_out=4'd0;
end
always @ (posedge clk)
begin
div = div+1'b1;
clkdiv = div[22];
end
always @ (posedge clkdiv)
begin
if (rst)
bcd_out=4'd0;
else if(bcd_out<4'd9)
bin_out=bin_out+4'd1;
else
bin_out=4'd0;
end
endmodule
```

6. Design of a N- bit shift register of Serial- in Serial –out, Serial in parallel out, Parallel in Serial out and Parallel in Parallel Out.

A: // Serial-in Serial-out (SISO) Shift Register

```
module siso_shift_reg #(parameter N = 8) (
   input clk,
   input data_in,
   output reg data_out );
   reg [N-1:0] shift_reg;
   always @(posedge clk) begin
      shift_reg <= {shift_reg[N-2:0], data_in};
      data_out <= shift_reg[N-1];
   end
endmodule
```

B: // Serial-in Parallel-out (SIPO) Shift Register

```
module sipo_shift_reg #(parameter N = 8) (
   input clk,
   input data_in,
   output reg [N-1:0] data_out);
   reg [N-1:0] shift_reg;
   always @(posedge clk) begin
      shift_reg <= {shift_reg[N-2:0], data_in};
      data_out <= shift_reg;
   end
endmodule
```

C: // Parallel-in Serial-out (PISO) Shift Register

```
module piso_shift_reg #(parameter N = 8) (
   input clk,
   input [N-1:0] data_in,
   output reg data_out);
   reg [N-1:0] shift_reg;
   always @(posedge clk) begin
      shift_reg <= {shift_reg[N-2:0], data_in[N-1]};
      data_out <= shift_reg[0];
   end
endmodule
```

D: // Parallel-in Parallel-out (PIPO) Shift Register

```
module pipo_shift_reg #(parameter N = 8) (
   input clk,
   input [N-1:0] data_in,
```

```
    output reg [N-1:0] data_out);
    always @(posedge clk) begin
      data_out <= data_in;
    end
endmodule
```

E: Universal Shift Register

Implement an 8-bit universal shift register with parallel load facility. Data load and shifting will occur in synchronism with the positive edge of "clk". If "ld" = 1, the input data "din" will be loaded into the register. If "ld" is 0 and "mode" is 0, 1-bit right shift will happen. If "ld" is 0 and "mode" is 1, 1-bit left shift will happen. "sin" is the serial data in, through which external data will be fed during shift operations.

Sample Test Cases

	Input	Output
Test Case 1	0	Pass: for clk=0, mode=x, ld=1, sin=x, din=255 and dout=255
Test Case 2	2	Pass: for clk=0, mode=0, ld=0, sin=0, din=192 and dout= 96
Test Case 3	3	Pass: for clk=0, mode=1, ld=0, sin=0, din=192 and dout=128
Test Case 4	1	Pass: for clk=0, mode=x, ld=1, sin=x, din= 15 and dout= 15
Test Case 5	4	Pass: for clk=0, mode=0, ld=0, sin=1, din=192 and dout=224
Test Case 6	5	Pass: for clk=0, mode=1, ld=0, sin=1, din=192 and dout=129

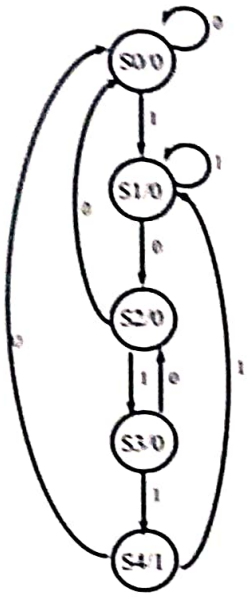
//Write the verilog modules to implement an 8-bit universal shift register

```
module shiftreg (sin, din, dout, ld, mode, clk);
  input [7:0] din;
  input sin, ld, mode, clk;
  output [7:0] dout;
  reg [7:0] dout;
  always @ (posedge clk)
    begin
      if (ld)
        dout <= din;
      else
        begin
          if(mode)
            dout <= {din[6:0],sin};
          else
            dout <= {sin,din[7:1]};
        end
    end
endmodule
```

7. Design of Sequence Detector (Finite State Machine- Mealy and Moore Machines).

Sequence Detector 1011 (Moore Machine + Mealy Machine Overlapping)

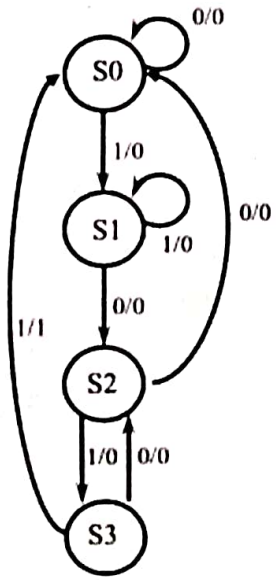
A: Moore Machine (Non-Overlapping):




```
module sd1011_moore(input bit clk,
                    input logic reset,
                    input logic din,
                    output logic dout);
  typedef enum logic [2:0] {S0, S1, S2, S3, S4} state_t;
  state_t state;
  always @(posedge clk or posedge reset) begin
    if(reset) begin
      dout <= 1'b0;
      state <= S0;
    end
    else begin
      case(state)
        S0: begin
          dout <=1'b0;
          if(din)
            state <= S1;
        end
        S1: begin
          dout <= 1'b0;
          if(~din)
            state <= S2;
```

```
      end
      S2: begin
        dout <= 1'b0;
        if(din)
          state <= S3;
        else
          state <= S0;
      end
      S3: begin
        dout <= 1'b0;
        if(din)
          state <= S4;
        else
          state <= S2;
      end
      S4: begin
        dout <= 1'b1;
        if(din)
          state <= S1;
        else
          state <= S0;
      end
    endcase
  end
 end
endmodule
```

B: Mealy Machine (Non-Overlapping):



```
module sd1011_mealy(input bit clk,
                    input logic reset,
                    input logic din,
                    output logic dout);
 typedef enum logic [1:0] {S0, S1, S2, S3} state_t;
 state_t state;
 always @(posedge clk or posedge reset) begin
  if(reset) begin
   dout <= 1'b0;
   state <= S0;
  end
  else begin
   case(state)
    S0: begin
     if(din) begin
      state <= S1;
      dout <=1'b0;
     end
     else
      dout <=1'b0;
    end
    S1: begin
     if(~din) begin
      state <= S2;
      dout <=1'b0;
     end
     else begin
      dout <=1'b0;
     end
    end
    S2: begin
     if(~din) begin
      state <= S0;
      dout <=1'b0;
     end
     else begin
      state <= S3;
      dout <=1'b0;
     end
    end
    S3: begin
     if(din) begin
      state <= S0;
      dout <=1'b1;
     end
```

```
        else begin
          state <= S2;
          dout <=1'b0;
        end
      end
    endcase
  end
end
endmodule
```

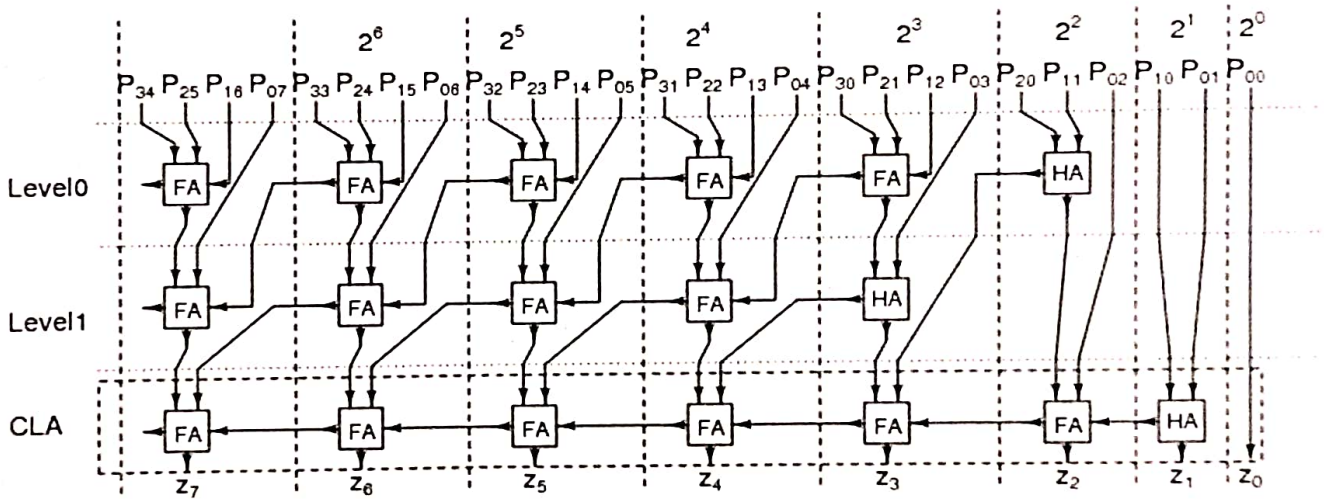
8. Design of 4- Bit Multiplier and 4-bit Divider.

A: 4- Bit Multiplier:

```
module Multiplier_4bit(
   input [3:0] A,      // 4-bit input A
   input [3:0] B,      // 4-bit input B
   output reg [7:0] P   // 8-bit output P (result of multiplication)
);
always @ (A or B) begin
   P = A * B;          // Multiplication operation, result stored in P
end
endmodule
```

Multiplier 4-bit with verilog using just full adders (wrong):

$$
\begin{array}{cccccccc}
 & & & x_3 & x_2 & x_1 & x_0 & \\
 & & & y_3 & y_2 & y_1 & y_0 & \\
\hline
P_{07} & P_{06} & P_{05} & P_{04} & P_{03} & P_{02} & P_{01} & P_{00} \\
P_{16} & P_{15} & P_{14} & P_{13} & P_{12} & P_{11} & P_{10} & \\
P_{25} & P_{24} & P_{23} & P_{22} & P_{21} & P_{20} & & \\
P_{34} & P_{33} & P_{32} & P_{31} & P_{30} & & & \\
\hline
z_7 & z_6 & z_5 & z_4 & z_3 & z_2 & z_1 & z_0
\end{array}
$$




```
module FA(input a,input b,input cin,output s,output cout);
wire z1,z2,z3;
xor(z1,a,b);
xor(s,z1,cin);
and(z2,z1,cin);
and(z3,a,b);
or(cout,z2,z3);
endmodule
```

```
module multiplier(m0,m1,m2,m3,p);
input[7:0] m0,m1,m2,m3;
output[7:0]p;
wire[11:1]w; // wire
wire[6:1]o; // outputs for the FA
// first part of the diagram
FA stage1(p[1],w[1],m0[1],m1[1]);
FA stage2(o[1],w[2],m0[2],m1[2],m2[2]);
FA stage3(o[2],w[3],m1[3],m2[3],m3[3]);
FA stage4(o[3],w[4],m0[4],m1[4],m2[4]);
// second part of the diagram
FA stage5(p[2],o[1],w[1],w[5]);
FA stage6(o[4],o[2],w[2],w[6],m0[3]);
FA stage7(o[5],o[3],w[3],w[6],w[7]);
FA stage8(o[6],m0[5],m1[5],w[4],w[8]);
// third part of the diagram
FA stage9(p[3],o[4],w[5],w[9]);
FA stage10(p[4],o[5],w[9],w[10]);
FA stage11(p[5],o[6],w[7],w[10],w[11]);
FA stage12(p[6],p[7],w[8],w[11],m0[6]);
endmodule
```

B: 4-Bit Division using verilog:

```
module div( input clk,
input [7:0] A,
input [3:0] B,
output reg [3:0] Q,
output reg [3:0] R);
reg [3:0] Q_pre;
reg [3:0] R_pre;
always @*
begin
Q_pre = A / B;
R_pre = A - Q * B;
end
always @(posedge clk)
begin
Q <= Q_pre;
R <= R_pre;
end
endmodule
```

9. Design of ALU to Perform – ADD, SUB, AND, OR, 1's compliment, 2's Compliment.

Verilog Code of ALU:

```
module alu(
        input [7:0] A,B,  // ALU 8-bit Inputs
        input [3:0] ALU_Sel,// ALU Selection
        output [7:0] ALU_Out, // ALU 8-bit Output
        output CarryOut // Carry Out Flag
    );
    reg [7:0] ALU_Result;
    wire [8:0] tmp;
    assign ALU_Out = ALU_Result; // ALU out
    assign tmp = {1'b0,A} + {1'b0,B};
    assign CarryOut = tmp[8]; // Carryout flag
    always @(*)
    begin
      case(ALU_Sel)
      4'b0000: // Addition
        ALU_Result = A + B ;
      4'b0001: // Subtraction
        ALU_Result = A - B ;
      4'b0010: // Multiplication
        ALU_Result = A * B;
      4'b0011: // Division
        ALU_Result = A/B;
      4'b0100: // Logical shift left
        ALU_Result = A<<1;
      4'b0101: // Logical shift right
        ALU_Result = A>>1;
      4'b0110: // Rotate left
        ALU_Result = {A[6:0],A[7]};
      4'b0111: // Rotate right
        ALU_Result = {A[0],A[7:1]};
      4'b1000: //  Logical and
        ALU_Result = A & B;
      4'b1001: //  Logical or
        ALU_Result = A | B;
      4'b1010: //  Logical xor
```

```
        ALU_Result = A ^ B;
       4'b1011: //  Logical nor
        ALU_Result = ~(A | B);
       4'b1100: // Logical nand
        ALU_Result = ~(A & B);
       4'b1101: // Logical xnor
        ALU_Result = ~(A ^ B);
       4'b1110: // Greater comparison
        ALU_Result = (A>B)?8'd1:8'd0 ;
       4'b1111: // Equal comparison
         ALU_Result = (A==B)?8'd1:8'd0 ;
       default: ALU_Result = A + B ;
      endcase
  end
endmodule
```

OTHER CIRCUITS:

A: JK Flip-Flop Implement a JK flip-flop with asynchronous set and reset.
Sample Test Cases

	Input	Output
Test Case 1	0	pass: for Q(t) = 0 and [J, K] = [1, 0], Q(t+1) = 1
Test Case 2	2	pass: for Q(t) = 1 and [J, K] = [1, 1], Q(t+1) = 0
Test Case 3	1	pass: for Q(t) = 1 and [J, K] = [0, 1], Q(t+1) = 0
Test Case 4	3	pass: for Q(t) = 0 and [J, K] = [1, 1], Q(t+1) = 1

//Write the verilog modules to implement a JK flip-flop

```
module ffjk(J, K, set, reset, Clk, Q);
   input J, K, Clk, reset, set;
   output Q;
reg Q,q1;
always @ (posedge Clk or posedge set or posedge reset)
  begin
  if (reset)
  begin
  Q <= 0;
  q1 <= 1;
 end
 else if (set)
 begin
   Q <= 1;
   q1 <= 0;
  end
```

```
 else case({J,K})
                {1'b0,1'b0}: begin Q<=Q;    q1<=q1; end
                {1'b0,1'b1}: begin Q<=1'b0; q1<=1'b1; end
                {1'b1,1'b0}: begin Q<=1'b1; q1<=1'b0; end
                {1'b1,1'b1}: begin Q<=~Q;   q1<=~q1; end
        endcase
  end
endmodule
```

B: User Defined Primitive

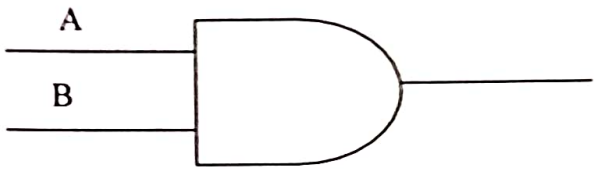
```
//Write the user-defined primitive for F = A.B + B.C
 primitive myfunc (F, A, B, C, D);
        input A, B, C, D;
        output F;
table
//A     B       C       D       F
0       0       0       0:      0;
0       0       0       1:      0;
0       0       1       0:      0;
0       0       1       1:      0;
0       1       0       0:      0;
0       1       0       1:      0;
0       1       1       0:      1;
0       1       1       1:      1;
1       0       0       0:      0;
1       0       0       1:      0;
1       0       1       0:      0;
1       0       1       1:      0;
1       1       0       0:      1;
1       1       0       1:      1;
1       1       1       0:      1;
1       1       1       1:      1;
endtable
endprimitive
```

A: T-SPICE PROGRAM

10. CMOS Inverter using SPICE simulation.

Prob1: CMOS Logic Gate Implementation Using T-Spice (model file- TSMC018.MD)

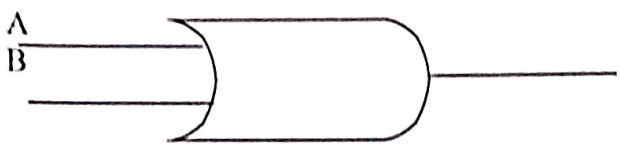
i. AND GATE



SPICE CODE

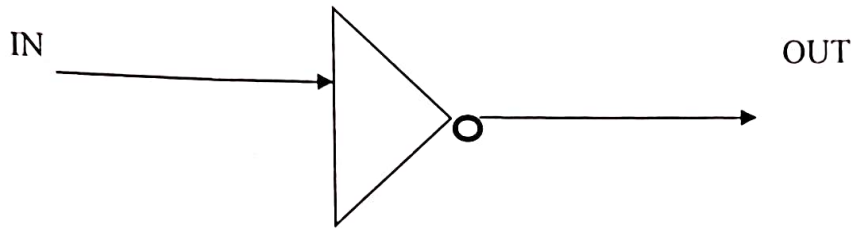
```
.GLOBAL VDD
.include "D:\MTechFinalProject\TSPICE\TSMC018.MD"

.SUBCKT AND A B OUT
MP1 N A VDD VDD PMOS W=5U L=0.18U
MP2 N B VDD VDD PMOS W=5U L=0.18U
MP3 OUT N VDD VDD PMOS W=5U L=0.18U
MN1 N A  1   0 NMOS W=2U L=0.18U
MN2 1 B  0   0 NMOS W=2U L=0.18U
MN3 OUT N 0 0 NMOS W=2U L=0.18U
.ENDS
X1 A B OUT AND
CL OUT 0 0.1PF
VIN1 A 0 PULSE 0 1.8 0N 1N 1N 10N 20N
VIN2 B 0 PULSE 0 1.8 0N 1N 1N 10N 20N
VDD VDD 0 1.8
.TRAN 1P 40N
.PLOT TRAN A B OUT
.measure tran tplhA trig v(A) val=0.5 rise=1 targ V(OUT) val=0.5 rise=1
.measure tran tphlA trig v(A) val=0.5 fall=1 targ V(OUT) val=0.5 fall=1
.measure tran tplhB trig v(B) val=0.5 rise=1 targ V(OUT) val=0.5 rise=1
.measure tran tphlB trig v(B) val=0.5 fall=1 targ V(OUT) val=0.5 fall=1
.measure tran totaldelayA param='(tplhA+tphlA)/2'
.measure tran totaldelayB param='(tplhB+tphlB)/2'
.power VDD 0 40N
.END
```

ii. ORGATE




```
OR GATE
.GLOBAL VDD
.include "D:\MTechFinalProject\TSPICE\TSMC018.MD"
.SUBCKT OR A B OUT
MP1 1 A VDD VDD PMOS W=5U L=0.18U
MP2 N B 1 1 PMOS W=5U L=0.18U
MP3 OUT N VDD VDD PMOS W=5U L=0.18U
MN1 N A 0 0 NMOS W=2U L=0.18U
MN2 N B 0 0 NMOS W=2U L=0.18U
MN3 OUT N 0 0 NMOS W=2U L=0.18U
.ENDS
X1 A B OUT OR
CL OUT 0 0.1PF
VIN1 A 0 PULSE 0 0 0N 1N 1N 10N 20N
VIN2 B 0 PULSE 0 0 5N 1N 1N 10N 20N
VDD VDD 0 1.8
.TRAN 1P 40N
.PLOT TRAN A B OUT
.measure tran tplhA trig v(A) val=0.5 rise=1 targ V(OUT) val=0.5 rise=1
.measure tran tphlA trig v(A) val=0.5 fall=1 targ V(OUT) val=0.5 fall=1
.measure tran totaldelayA param='(tplhA+tphlA)/2'
.measure tran tplhB trig v(B) val=0.5 rise=1 targ V(OUT) val=0.5 rise=1
.measure tran tphlB trig v(B) val=0.5 fall=1 targ V(OUT) val=0.5 fall=1
.measure tran totaldelayB param='(tplhB+tphlB)/2'
 .power VDD 0 40N
.END
```

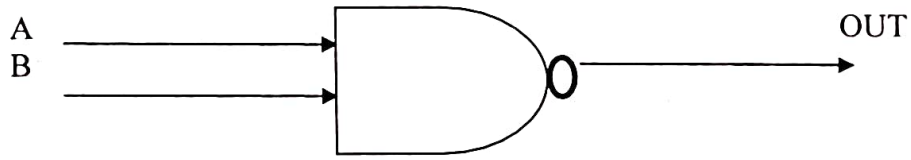
iii. NOT GATE



CMOS inverter:

```
.global vdd
.include "C:\Users\Sandip\Desktop\TSPICE\TSMC018.MD"
.subckt inv11 in out
M1 OUT IN VDD VDD PMOS W=5U L=0.18U
M2 OUT IN 0 0     NMOS W=2U L=0.18U
.ends
X1 IN OUT INV11
CL OUT 0 1PF
VIN IN 0 PULSE 0 1.8 0N 1N 1N 10N 20N
VDD VDD 0 1.8
.TRAN 1P 40N
.PLOT TRAN IN OUT
.END
```

iv. NAND GATE



SPICE CODE OF NAND GATE

```
.GLOBAL VDD
.include "D:\MTechFinalProject\TSPICE\TSMC018.MD"
.SUBCKT NAND A B OUT
MP1 OUT A VDD VDD PMOS W=5U L=0.18U
MP2 OUT B VDD VDD PMOS W=5U L=0.18U
MN1 OUT A  1   0 NMOS W=5U L=0.18U
MN2  1  B  0   0 NMOS W=5U L=0.18U
.ENDS
X1 A B OUT NAND
CL OUT 0 0.1PF
```

```
VIN1 A 0 PULSE 0 1.8 0N 1N 1N 10N 20N
VIN2 B 0 PULSE 0 1.8 0N 1N 1N 10N 20N
VDD VDD 0 1.8
.TRAN 1P 40N
.PLOT TRAN A B OUT
.measure tran tplh trig v(in) val=0.5 rise=1 targ V(out) val=0.5 rise=1
.measure tran tphl trig v(in) val=0.5 fall=1 targ V(out) val=0.5 fall=1
.measure tran totaldelay param='(tplh+tphl)/2'
.power VDD 0 40N
.END
```

v. NOR GATE

```
.GLOBAL VDD
.include "D:\MTechFinalProject\TSPICE\TSMC018.MD"
.SUBCKT OR A B OUT
MP1 NOD A VDD VDD PMOS W=5U L=0.18U
MP2 OUT B NOD NOD PMOS W=5U L=0.18U
MN1 OUT A 0 0 NMOS W=2U L=0.18U
MN2 OUT B 0 0 NMOS W=2U L=0.18U
.ENDS
X1 A B OUT OR
CL OUT 0 0.1PF
VIN1 A 0 PULSE 0 1.8 0N 1N 1N 10N 20N
VIN2 B 0 PULSE 0 1.8 5N 1N 1N 10N 20N
VDD VDD 0 1.8
.TRAN 1P 40N
.PLOT TRAN A B OUT
.power VDD 0 40N
.END
```

vi. XOR GATE

```
.GLOBAL VDD
.include "D:\MTechFinalProject\TSPICE\TSMC018.MD"
.SUBCKT XOR A B OUT
MPINV1 N1 A VDD VDD PMOS W=5U L=0.18U
MNINV1 N1 A 0 0 NMOS W=2U L=0.18U
MPINV2 N2 B VDD VDD PMOS W=5U L=0.18U
MNINV2 N2 B 0 0 NMOS W=2U L=0.18U
MPINV3 S1 Y1 VDD VDD PMOS W=5U L=0.18U
MNINV3 S1 Y1 0 0 NMOS W=2U L=0.18U
MPINV4 S2 Y2 VDD VDD PMOS W=5U L=0.18U
MNINV4 S2 Y2 0 0 NMOS W=2U L=0.18U
MPINV5 OUT T VDD VDD PMOS W=5U L=0.18U
MNINV5 OUT T 0 0 NMOS W=2U L=0.18U
MP1AND1 Y1 N1 VDD VDD PMOS W=5U L=0.18U
MP2AND1 Y1 B  VDD VDD PMOS W=5U L=0.18U
MN1AND1 Y1 N1 P P NMOS W=2U L=0.18U
MN2AND1 P B 0 0 NMOS W=2U L=0.18U
MP1AND2 Y2 A VDD VDD PMOS W=5U L=0.18U
MP2AND2 Y2 N2  VDD VDD PMOS W=5U L=0.18U
MN1AND2 Y2 A Q Q NMOS W=2U L=0.18U
MN2AND2 Q N2 0 0 NMOS W=2U L=0.18U
MPOR1 X S1 VDD VDD PMOS W=5U L=0.18U
MPOR2 T S2 X X PMOS W=5U L=0.18U
MNOR1 T S1   0 0 NMOS W=2U L=0.18U
MNOR2 T S2   0 0 NMOS W=2U L=0.18U
.ENDS
X1 A B OUT XOR
CL OUT 0 0.1PF
VIN1 A 0 PULSE 0 1.8 0N 1N 1N 10N 20N
VIN2 B 0 PULSE 0 1.8 5N 1N 1N 10N 20N
VDD VDD 0 1.8
.TRAN 1P 40N
.PLOT TRAN A B OUT
.END
```

vii. XNOR GATE

```
.GLOBAL VDD
.include "D:\MTechFinalProject\TSPICE\TSMC018.MD"
.SUBCKT XNOR A B OUT
MPINV1 N1 A VDD VDD PMOS W=5U L=0.18U
MNINV1 N1 A 0 0 NMOS W=2U L=0.18U
MPINV2 N2 B VDD VDD PMOS W=5U L=0.18U
MNINV2 N2 B 0 0 NMOS W=2U L=0.18U
```

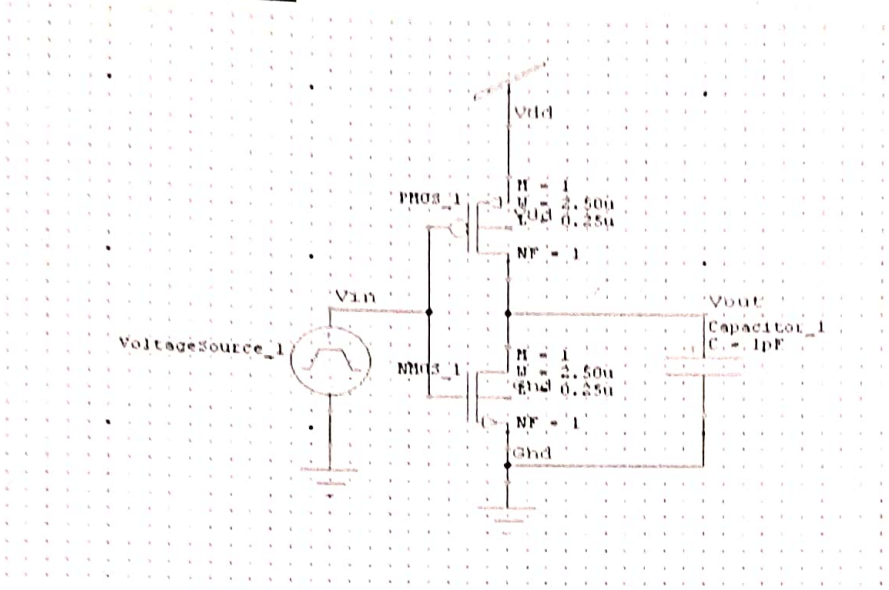
```
MPINV3 S1 Y1 VDD VDD PMOS W=5U L=0.18U
MNINV3 S1 Y1 0 0 NMOS W=2U L=0.18U
MPINV4 S2 Y2 VDD VDD PMOS W=5U L=0.18U
MNINV4 S2 Y2 0 0 NMOS W=2U L=0.18U
MPINV5 OUT T VDD VDD PMOS W=5U L=0.18U
MNINV5 OUT T 0 0 NMOS W=2U L=0.18U
MP1AND1 Y1 N1 VDD VDD PMOS W=5U L=0.18U
MP2AND1 Y1 N2  VDD VDD PMOS W=5U L=0.18U
MN1AND1 Y1 N1 P P NMOS W=2U L=0.18U
MN2AND1 P N2 0 0 NMOS W=2U L=0.18U
MP1AND2 Y2 A VDD VDD PMOS W=5U L=0.18U
MP2AND2 Y2 B  VDD VDD PMOS W=5U L=0.18U
MN1AND2 Y2 A Q Q NMOS W=2U L=0.18U
MN2AND2 Q B 0 0 NMOS W=2U L=0.18U
MPOR1 X S1 VDD VDD PMOS W=5U L=0.18U
MPOR2 T S2 X X PMOS W=5U L=0.18U
MNOR1 T S1   0 0 NMOS W=2U L=0.18U
MNOR2 T S2   0 0 NMOS W=2U L=0.18U
.ENDS
X1 A B OUT XNOR
CL OUT 0 0.1PF
VIN1 A 0 PULSE 0 1.8 0N 1N 1N 10N 20N
VIN2 B 0 PULSE 0 1.8 5N 1N 1N 10N 20N
VDD VDD 0 1.8
.TRAN 1P 40N
.PLOT TRAN A B OUT
.END
```

Prob2: Write a T-spice program of CMOS Inverter for transient response and calculate the propagation delay and power dissipation?

Program:

```
CCapacitor_1 b Gnd  1p
MNMOS_1 b a Gnd Gnd Nh W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MPMOS_1 b a vdd vdd Ph W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
VVoltageSource_1 a Gnd  PULSE(0 5 0 5n 5n 95n 200n)
vdd vdd gnd dc 5v
.include "C:\Documents and Settings\user\Desktop\dual.md"
.tran 2ns 400ns
.print tran v(a) v(b)
.measure tran tplh trig v(a) val=.5 rise=1 targ v(b) val=.5 rise=1
.measure tran tphl trig v(a) val=.5 fall=1 targ v(b) val=.5 fall=1
.measure tran tp param='tphl+tplh'
.power Vdd 0n 400n
.end
```

Schematic Diagram:



Results:

Power Results

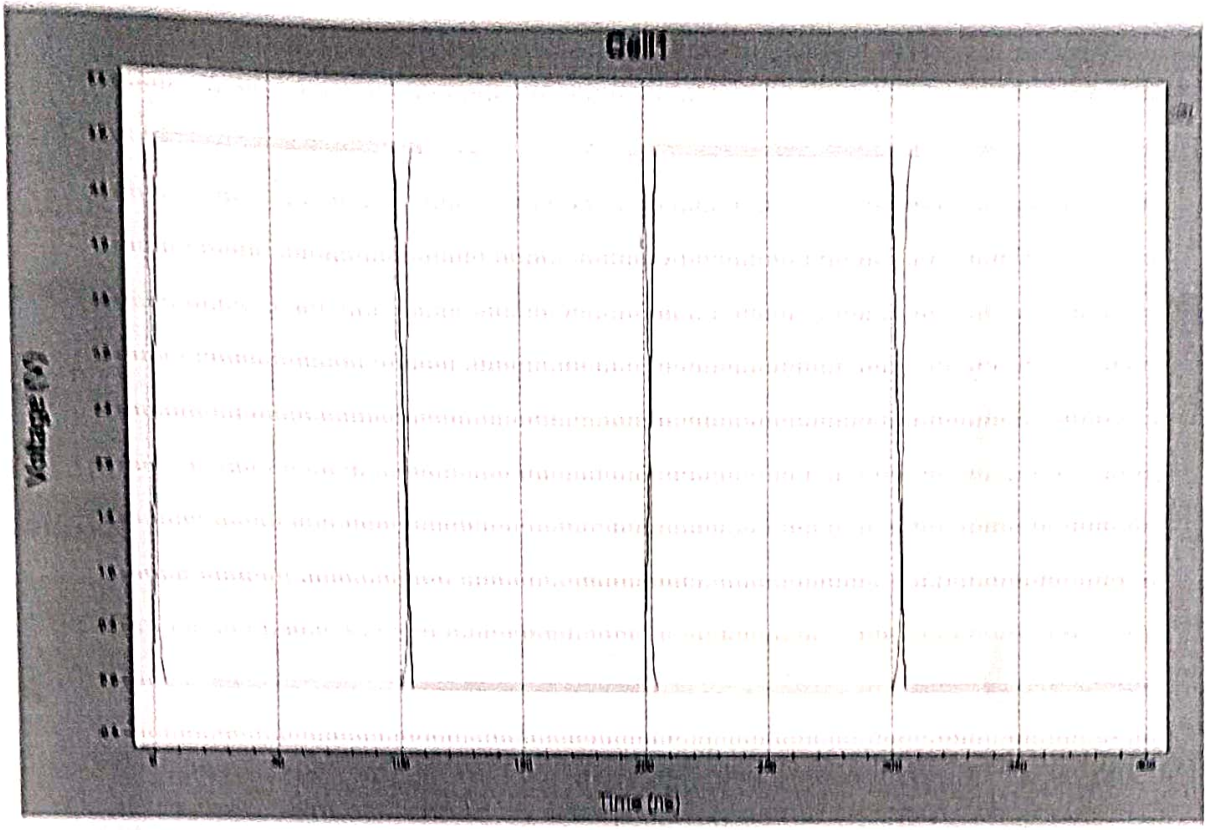
Vdd from time 0 to 4e-007
Average power consumed -> 4.450431e-004 watts
Max power 1.701567e-002 at time 3.04516e-007
Min power 1.528037e-007 at time 1.35e-008

DELAY MEASUREMENT RESULTS

tplh = 1.0233e-007
  Trigger = 5.0000e-010
  Target = 1.0283e-007
tphl = -1.0147e-007
  Trigger = 1.0450e-007
  Target = 3.0332e-009

tp = 8.6017e-010

Waveform:



Prob3: Write a T-spice program of CMOS Inverter for DC response and from graph calculate the threshold voltage?

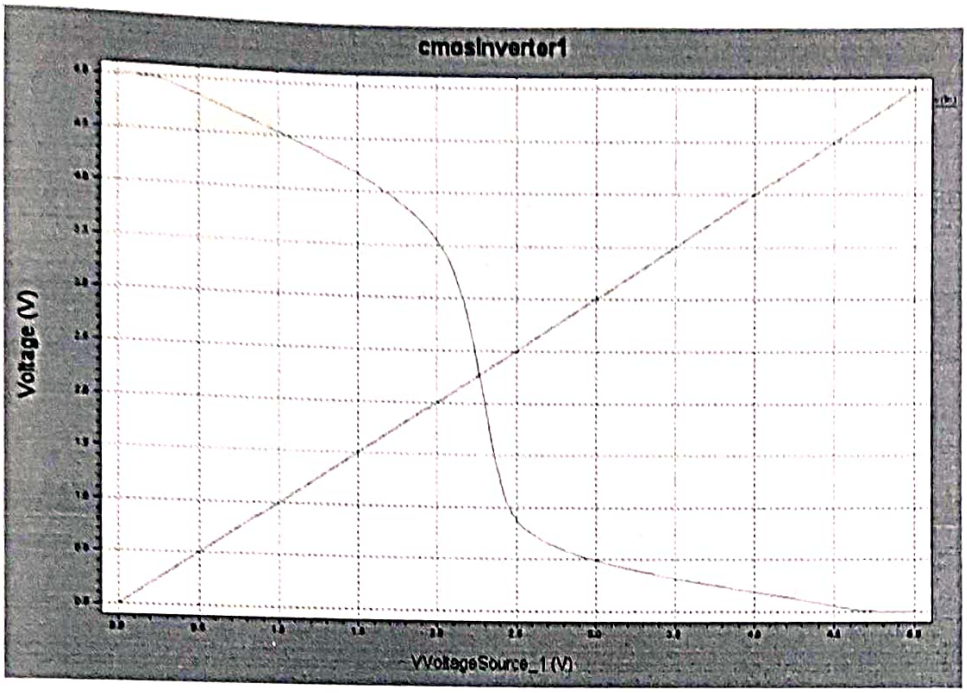
Program:

```
CCapacitor_1 Out Gnd  1p
MNMOS_1 Out In Gnd Gnd NH W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p
PD=6.8u
MPMOS_1 Out In Vdd Vdd PH W=5u L=150n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
VVoltageSource_1 In Gnd  PULSE(0 5 0 5n 5n 95n 200n)
Vdd Vdd Gnd Dc 5v
.dc VVoltageSource_1 0v 5v .05v
.print dc v(In) v(Out)
.include "C:\Documents and Settings\admin\Desktop\dual.md"
.end
```

Results:

Threshold Voltage=2.27v

Waveform:



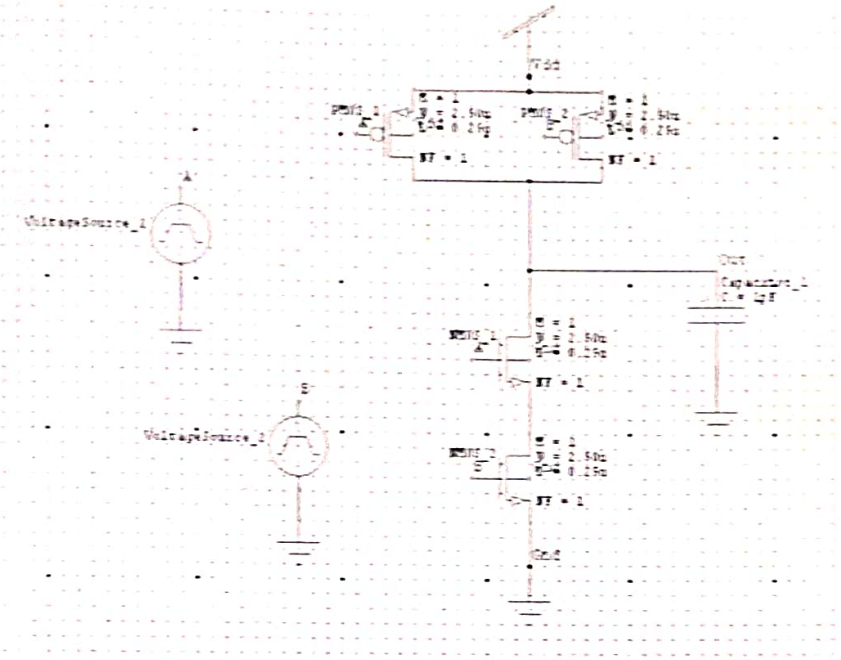
Prob 4: Write a T-spice program of CMOS NAND2 and calculate the propagation delay and power dissipation?

Program:

```
CCapacitor_1 Out Gnd  1p
MNMOS_1 Out A N_6 Gnd Nh W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p
PD=6.8u
MNMOS_2 N_6 B Gnd Gnd Nh W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p
PD=6.8u
MPMOS_1 Out A Vdd Vdd Ph W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MPMOS_2 Out B Vdd Vdd Ph W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
VVoltageSource_1 A Gnd  PULSE(0 5 0 5n 5n 95n 200n)
VVoltageSource_2 B Gnd  PULSE(0 5 0 5n 5n 95n 200n)
Vdd Vdd Gnd DC 5v
.include "C:\Documents and Settings\admin\Desktop\dual.md"
.tran 5n 250n
.print tran v(A) V(B) v(Out)
.measure tran tphl trig V(A) val=0.5 rise=1 targ V(Out) val=0.5 fall=1
.measure tran tplh trig V(A) val=0.5 fall=1 targ V(Out) val=0.5 rise=1
.measure tran tp param='(tphl+tplh)/2'
```

.power Vdd 0n 250n
.end

Schematic Diagram:



Results:

Power Results

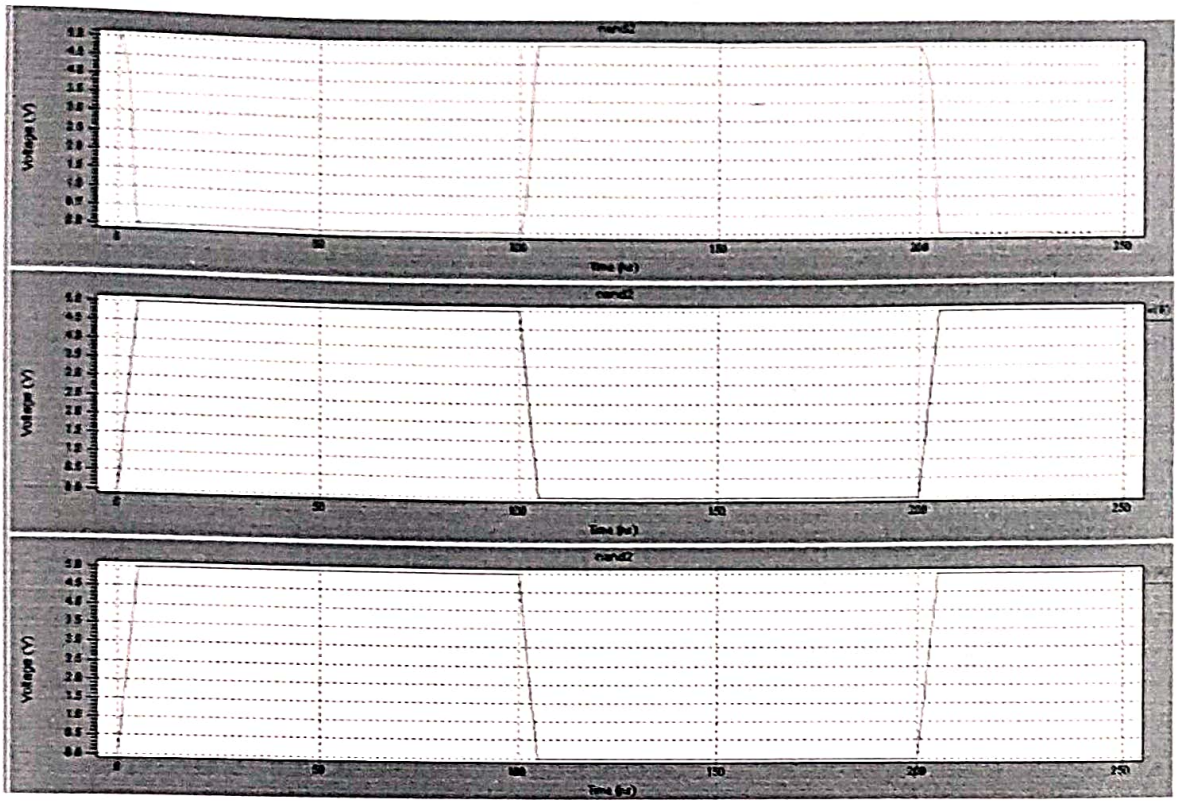
Vdd from time 0 to 2.5e-007
Average power consumed -> 5.598962e-004 watts
Max power 2.449521e-002 at time 1.03286e-007
Min power 1.676166e-009 at time 0

DELAY MEASUREMENT RESULTS

tphl = 3.7678e-009
  Trigger = 5.0000e-010
  Target = 4.2678e-009

tplh = -3.1709e-009
  Trigger = 1.0450e-007
  Target = 1.0133e-007

tp = 2.9844e-010



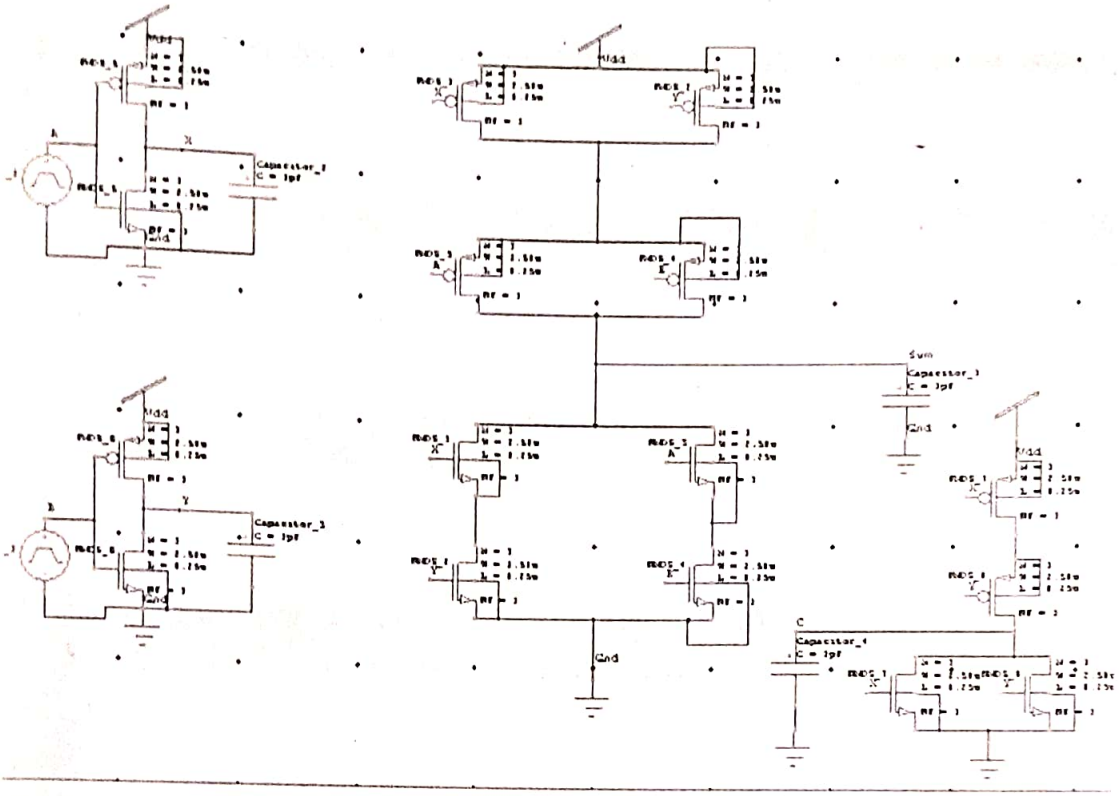
Prob5: Write a T-spice program of CMOS HALF ADDER for transient response and calculate the propagation delay and power dissipation?

Program:

```
CCapacitor_1 Sum Gnd  10f
CCapacitor_2 X Gnd  100f
CCapacitor_3 Y Gnd  100f
CCapacitor_4 C Gnd  10f
MNMOS_3 Sum A N_3 N_3 Nh W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p
PD=6.8u
MNMOS_4 N_3 B Gnd Gnd Nh W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p
PD=6.8u
MNMOS_5 X A Gnd Gnd Nh W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MNMOS_6 Y B Gnd Gnd Nh W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MNMOS_7 C X Gnd Gnd Nh W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MNMOS_8 C Y Gnd Gnd Nh W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
```

```
MNMOS_1 Sum X N_2 N_2 Nh W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p
PD=6.8u
MNMOS_2 N_2 Y Gnd Gnd Nh W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p
PD=6.8u
MPMOS_1 N_1 X Vdd Vdd Ph W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p
PD=6.8u
MPMOS_2 N_1 Y Vdd Vdd Ph W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p
PD=6.8u
MPMOS_3 Sum A N_1 N_1 Ph W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p
PD=6.8u
MPMOS_4 Sum B N_1 N_1 Ph W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p
PD=6.8u
MPMOS_5 X A Vdd Vdd Ph W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MPMOS_6 Y B Vdd Vdd Ph W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
MPMOS_7 N_4 X Vdd Vdd Ph W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p
PD=6.8u
MPMOS_8 C Y N_4 N_4 Ph W=2.5u L=150n AS=2.25p PS=6.8u AD=2.25p PD=6.8u
VVoltageSource_1 B Gnd  PULSE(0 5 0 5n 5n 95n 200n)
VVoltageSource_2 A Gnd  PULSE(0 5 0 5n 5n 95n 200n)
Vdd Vdd Gnd DC 5v
.include "C:\Documents and Settings\admin\Desktop\dual.md"
.tran 5n 250n
.print tran v(A) V(B) v(Sum) V(C)
.measure tran tphl trig V(A) val=0.5 rise=1 targ V(Sum) val=0.5 fall=1
.measure tran tplh trig V(A) val=0.5 fall=1 targ V(Sum) val=0.5 rise=1
.measure tran tp param='(tphl+tplh)/2'
.power Vdd 0n 250n
.end
```

Schematic Diagram:



RESULTS:

Power Results

Vdd from time 0 to 2.5e-007
Average power consumed -> 1.515183e-003 watts
Max power 5.825236e-002 at time 1.03624e-007
Min power 1.755376e-006 at time 1.21186e-007

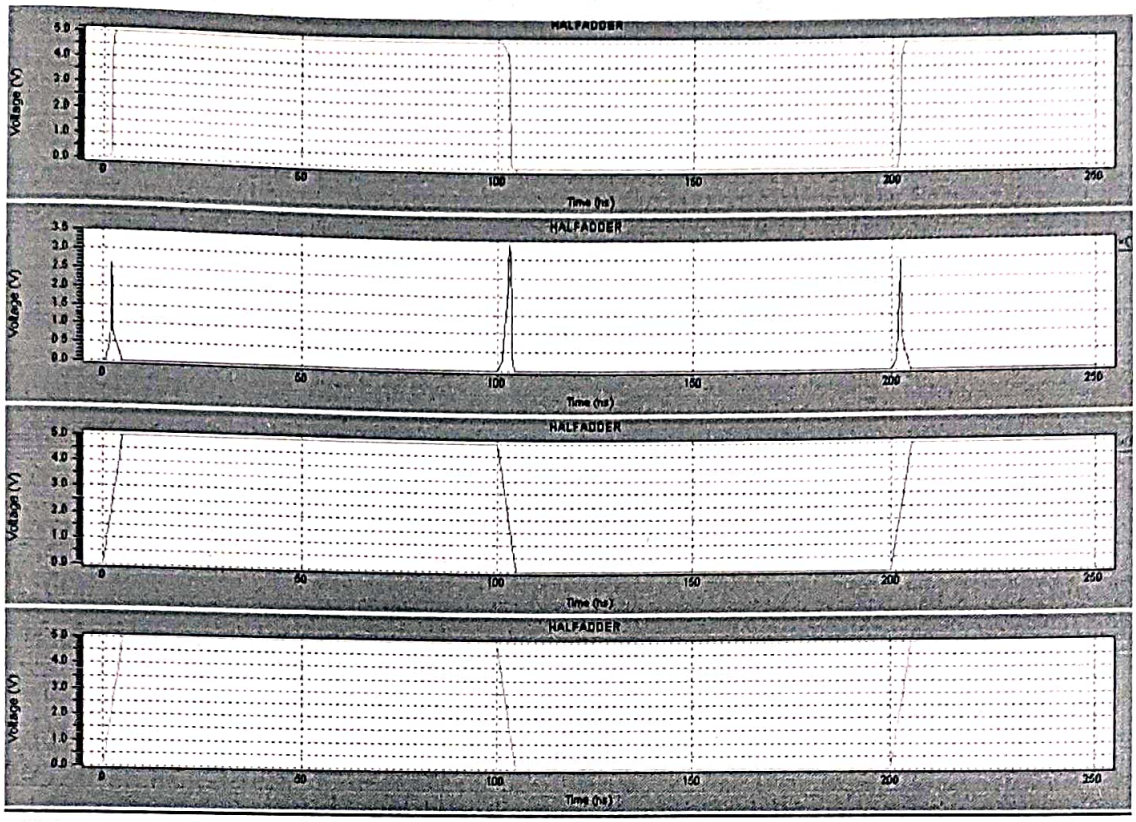
DELAY MEASUREMENT RESULTS

tphl = 2.7434e-009
  Trigger = 5.0000e-010
  Target = 3.2434e-009

tplh = -1.0276e-007
  Trigger = 1.0450e-007
  Target = 1.7373e-009

tp = -5.0010e-008

Waveform:



Prob 6: Write a T-spice program of CMOS 1 bit Full Adder and calculate the propagation delay and power dissipation?

Program:

```
.GLOBAL VDD
*.include "D:\MTechFinalProject\TSPICE\TSMC018.MD"

.SUBCKT XOR A B OUT
MPINV1 N1 A VDD VDD PH W=5U L=.15u
MNINV1 N1 A 0 0 NH W=2U L=.15u

MPINV2 N2 B VDD VDD PH W=5U L=.15u
MNINV2 N2 B 0 0 NH W=2U L=.15u

MPINV3 S1 Y1 VDD VDD PH W=5U L=.15u
MNINV3 S1 Y1 0 0 NH W=2U L=.15u

MPINV4 S2 Y2 VDD VDD PH W=5U L=.15u
MNINV4 S2 Y2 0 0 NH W=2U L=.15u

MPINV5 OUT T VDD VDD PH W=5U L=.15u
MNINV5 OUT T 0 0 NH W=2U L=.15u

MP1AND1 Y1 N1 VDD VDD PH W=5U L=.15u
MP2AND1 Y1 B  VDD VDD PH W=5U L=.15u
MN1AND1 Y1 N1 P P NH W=2U L=.15u
MN2AND1 P B 0 0 NH W=2U L=.15u

MP1AND2 Y2 A VDD VDD PH W=5U L=.15u
MP2AND2 Y2 N2  VDD VDD PH W=5U L=.15u
MN1AND2 Y2 A Q Q NH W=2U L=.15u
MN2AND2 Q N2 0 0 NH W=2U L=.15u


MPOR1 X S1 VDD VDD PH W=5U L=.15u
MPOR2 T S2 X X PH W=5U L=.15u
MNOR1 T S1   0 0 NH W=2U L=.15u
MNOR2 T S2   0 0 NH W=2U L=.15u
.ENDS

.SUBCKT AND A B OUT
MP1 N A VDD VDD PH W=5U L=.15u
MP2 N B VDD VDD PH W=5U L=.15u
MP3 OUT N VDD VDD PH W=5U L=.15u
```

```
MN1 N A  1   0 NH W=2U L=.15u
MN2 1 B  0   0 NH W=2U L=.15u
MN3 OUT N 0 0 NH W=2U L=.15u
.ENDS

.SUBCKT OR A B OUT
MP1 1 A VDD VDD PH W=5U L=.15u
MP2 N B 1 1 PH W=5U L=.15u
MP3 OUT N VDD VDD PH W=5U L=.15u
MN1 N A 0 0 NH W=2U L=.15u
MN2 N B 0 0 NH W=2U L=.15u
MN3 OUT N 0 0 NH W=2U L=.15u
.ENDS

X1 A B OUT1 XOR
X2 OUT1 CIN SUM XOR
X3 CIN OUT1 OUT2 AND
X4 A B OUT3 AND
X5 OUT2 OUT3 COUT OR


CL1 COUT 0 0.1PF
CL2 SUM 0 0.1PF

VIN1 A 0 PULSE 0 1.8 0N 1N 1N 10N 20N
VIN2 B 0 PULSE 0 0 0N 1N 1N 10N 20N
VIN3 CIN 0 PULSE 0 0 0N 1N 1N 10N 20N


VDD VDD 0 1.8
.TRAN 1P 40N
.PLOT TRAN A B CIN COUT SUM
.include "C:\Documents and Settings\admin\Desktop\dual.md"
.measure tran tphl trig V(A) val=0.5 rise=1 targ V(SUM) val=0.5 fall=1
.measure tran tplh trig V(A) val=0.5 fall=1 targ V(SUM) val=0.5 rise=1
.measure tran tp param='(tphl+tplh)/2'
.power Vdd 0n 250n
.END
```

RESULTS:

Power Results

Vdd from time 0 to 2.5e-007
Average power consumed -> 2.385742e-005 watts
Max power 8.324213e-003 at time 1.15503e-008
Min power 1.186880e-007 at time 3.09982e-008

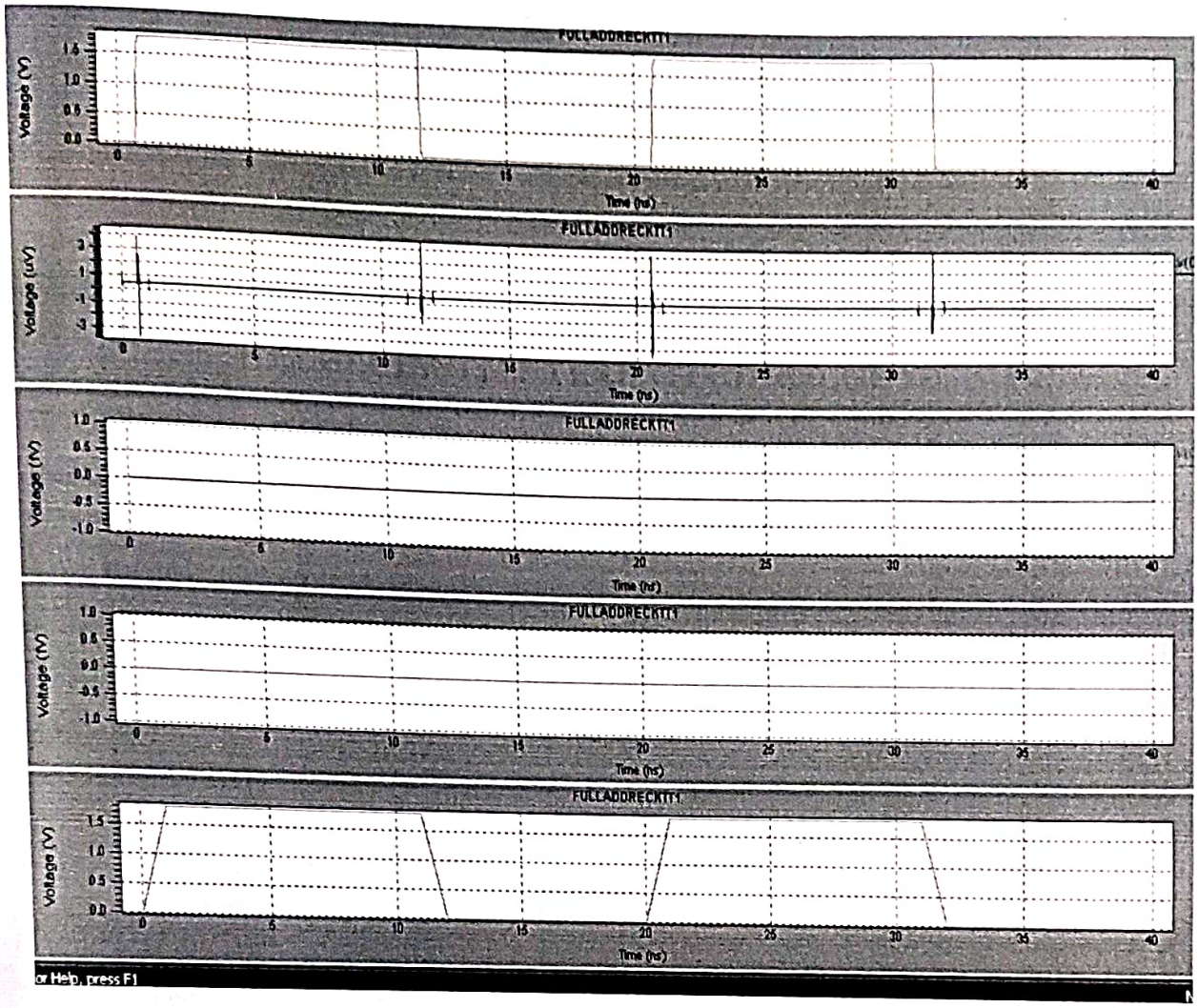
DEALY MEASUREMENT RESULTS

tphl = 1.1361e-008
Trigger = 2.7778e-010
Target = 1.1639e-008

tplh = -1.1059e-008
Trigger = 1.1722e-008
Target = 6.6341e-010

tp = 1.5122e-010

Waveform:


FULLADDRECKT1
Voltage (V)
Time (ns)
FULLADDRECKT1
Voltage (uV)
Time (ns)
FULLADDRECKT1
Voltage (V)
Time (ns)
FULLADDRECKT1
Voltage (V)
Time (ns)
FULLADDRECKT1
Voltage (V)
Time (ns)
or Help, press F1