VSM's

Somashekhar R Kothiwale Institute of Technology, Nipani-591237

Department of Electronics & Communication Engg

VLSI LAB

15ECL77

BY

Prof. chetan alatagi

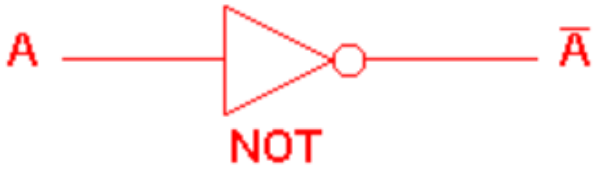
EXPT.NO.1

TITLE : Write Verilog Code for the inverter circuit and their Test Bench for verification, observe the waveform and synthesize the code with technological library with given Constraints. Do the initial timing verification with gate level simulation.

TOOL REQUIRED: XILINX 10.2

THEORY : The NOT gate or an inverter is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. This is also shown as A', or A with a bar over the top, as shown at the outputs. The diagrams below show two ways that the NAND logic gate can be configured to produce a NOT gate. It can also be done using NOR logic gates in the same way.

Symbol



Truth Table

NOT Gate	
INPUT	OUTPUT
A	A’
0	1
1	0

Verilog Code For Inverter:

```
module inv(a,b);
input a;
output b;
assign b = ~(a);
endmodule
```

Test Bench:

```
module inv_test;
        reg a;
        wire b;
        inv uut (
                .a(a),
                .b(b)
        );
        initial begin
        a = 0; #100;
        a = 1; #100;
        a = 0; #100;
```

```
        a = 1; #100;
        end
endmodule
```

RESULT: Verilog code for the inverter circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.

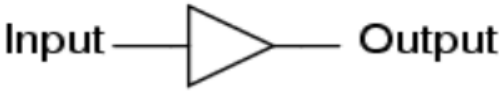
EXPT.NO.2

TITLE: Write Verilog Code for the buffer circuit and their Test Bench for verification, observe the waveform and synthesize the code with technological library with given Constraints. Do the initial timing verification with gate level simulation.

TOOL REQUIRED: XILINX 10.2

THEORY: A special logic gate called a buffer is manufactured to perform the same function as two inverters. Its symbol is simply a triangle, with no inverting "bubble" on the output terminal. Buffer gates merely serve the purpose of signal amplification: taking a "weak" signal source that isn't capable of sourcing or sinking much current, and boosting the current capacity of the signal so as to be able to drive a load.

Symbol



Truth Table

BUFFER	
INPUT	**OUTPUT**
0	**0**
1	**1**

Verilog Code For Buffer

```
module buffer (a,b);
input a;
output b;
assign b = (a);
endmodule
```

Test Bench:

```
module buffer_test;
                reg a;
                wire b;
                buffer uut (
                .a(a),
                .b(b)
                );

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

RESULT: Verilog code for the buffer circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.

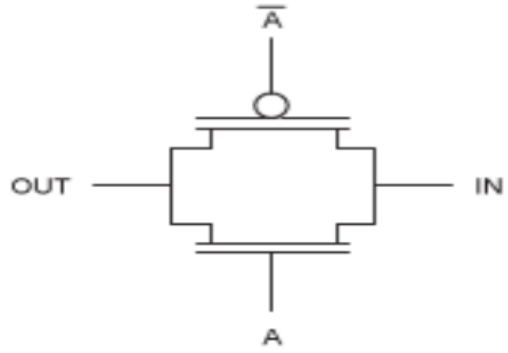
EXPT.NO.3

TITLE : Write Verilog Code for the transmission circuit and their Test Bench for verification, observe the waveform and synthesize the code with technological library with given Constraints. Do the initial timing verification with gate level simulation.

TOOL REQUIRED: XILINX 10.2

THEORY: A transmission gate, or analog switch, is defined as an electronic element that will selectively block or pass a signal level from the input to the output. Basic Operation This solid-state switch is comprised of a pMOS transistor and nMOS transistor. The control gates are biased in a complementary manner so that both transistors are either on or off. When the voltage on node A is a Logic 1, the complementary Logic 0 is applied to node active-low A, allowing both transistors to conduct and pass the signal at IN to OUT. When the voltage on node active-low A is a Logic 0, the complementary Logic 1 is applied to node A, turning both transistors off and forcing a high-impedance condition on both the IN and OUT nodes. This high-impedance condition represents the third "state" (high, low, or high-Z) that the channel may reflect downstream. The schematic diagram (Figure 1) includes the arbitrary labels for IN and OUT, as the circuit will operate in an identical manner if those labels were reversed. This design provides true bidirectional connectivity without degradation of the input signal. Figure 1. Schematic representation of a transmission gate.

Symbol



Truth Table

A	IN	OUT
H	H	H
H	L	L
L	X(don't care)	Z(high impedance)

3. Verilog Code For Transmission

```
module trans(a,b,an);
input a,en;
output b;
```

```
reg b;
always @(a,en)
begin
if(en == a'b1)
b=a;
else
b=1'bz;
end
endmodule
```

Test Bench:

```
module trans_test;
        reg a;
        reg en;
        wire b;
    trans uut (
        .a(a),
        .b(b),
        .en(en)
        );

    initial begin

        en = 1;  a=1  ;     #100;
        en = 1;  a=0  ;     #100;
        en = 0;  a=1  ;     #100;
        en = 0;  a=1  ;     #100;
    end
endmodule
```

RESULT: Verilog code for the transmission gate circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified

EXPT.NO.4

TITLE : Write Verilog Code for the Basic/Universal gates and their Test Bench for verification, observe the waveform and synthesize the code with technological library with given Constraints. Do the initial timing verification with gate level simulation.

TOOL REQUIRED: XILINX 10.2

THEORY: Digital systems are said to be constructed by using logic gates. These gates are the AND, OR, NOT, NAND, NOR, EXOR and EXNOR gates. The basic operations are described below with the aid of truth tables.

Symbol

Truth Table

INPUT		OUTPUT
A	**B**	**AB**
0	0	0
0	1	0
1	0	0
1	1	1

The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high. A dot (.) is used to show the AND operation i.e. A.B. Bear in mind that this dot is sometimes omitted i.e. AB

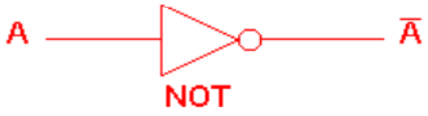
Symbol

Truth Table

INPUT		OUTPUT
A	**B**	**AB**
0	0	0
0	1	1
1	0	1
1	1	1

The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high. A plus (+) is used to show the OR operation.

Symbol



Truth Table

INPUT	OUTPUT
A	**A'**
0	1
1	0

The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. This is also shown as A', or A with a bar over the top, as shown at the outputs. The diagrams below show two ways that the NAND logic gate can be configured to produce a NOT gate. It can also be done using NOR logic gates in the same way.

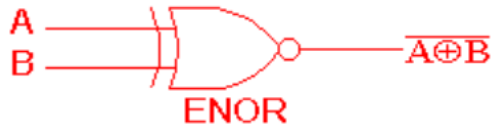
Symbol

Truth Table

INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

The 'Exclusive-OR' gate is a circuit which will give a high output if either, but not both, of its two inputs are high. An encircled plus sign ( ) is used to show the EOR operation.

Symbol



Truth Table

INPUT		OUTPUT
A	B	A XOR B
0	0	1
0	1	0
1	0	0
1	1	1

The 'Exclusive-NOR' gate circuit does the opposite to the EOR gate. It will give a low output if either, but not both, of its two inputs are high. The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion.

Symbol

Truth Table

INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	1

This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if any of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

Symbol

Truth Table

INPUT		OUTPUT
A	**B**	**A XOR B**
0	0	1
0	1	0
1	0	0
1	1	0

This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if any of the inputs are high. The symbol is an OR gate with a small circle on the output. The small circle represents inversion.

Verilog Code For Universal Gates

```
module uni(a,b,z1,z2,z3,z4,z5,z6);
input a,b;
output z1,z2,z3,z4,z5,z6;
assign z1 = a&b;
assign z2 = a | b;
assign z3 = ~(a&b);
assign z4 = ~(a | b);
assign z5 = a ^ b;
assign z6 = ~(a ^ b);
endmodule
```

Test Bench:

```
module uni_test;
        reg a; reg b;
        wire z1;        wire z2;        wire z3;
        wire z4;        wire z5;        wire z6;
        uni uut (
                .a(a), .b(b),
                .z1(z1), .z2(z2),.z3(z3),
                .z4(z4), .z5(z5),.z6(z6)
                );

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

```
endmodule
```

RESULT: Verilog code for the basic/universalgates circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.

EXPT.NO.5 a

TITLE : Write Verilog Code for the SR Flip Flop and their Test Bench for verification, observe the waveform and synthesize the code with technological library with given Constraints. Do the initial timing verification with gate level simulation.

TOOL REQUIRED: XILINX 10.2

THEORY: It can be seen that when both inputs S = "1" and R = "1" the outputs Q and Q can be at either logic level "1" or "0", depending upon the state of the inputs S or R BEFORE this input condition existed. Therefore the condition of S = R = "1" does not change the state of the outputs Q and Q. VLSI Lab However, the input state of S = "0" and R = "0" is an undesirable or invalid condition and must be avoided. The condition of S = R = "0" causes both outputs Q and Q to be HIGH together at logic level "1" when we would normally want Q to be the inverse of Q. The result is that the flip-flop looses control of Q and Q, and if the two inputs are now switched "HIGH" again after this condition to logic "1", the flip-flop becomes unstable and switches to an unknown data state based upon the unbalance as shown in the following switching diagram.

Symbol Circuit

Truth Table

Input		Output		Description
S	R	Q	Q	
0	0	0	0	Memory no change
0	0	0	1	
0	1	1	0	Reset
0	1	0	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	Z	Invalid Condition
1	1	1	Z	

Verilog Code For SR Flip-Flop

```
module srff( sr, reset, clk,q, qbar );
input [0:1]sr ;
input reset, clk ;
```

```
output q, qbar ;
reg q, qbar ;
always @( sr , reset, posedge clk )
begin
if ( reset == 0 )
begin
q = 1'b0;
qbar=1'b1;
end
else
begin
case(sr)
2'b00:begin q=q;qbar=qbar;end
2'b01:begin q=1'b0;qbar=1'b1;end
2'b10:begin q=1'b1;qbar=1'b0;end
2'b11:begin q=1'bz;qbar=1'bz;end
endcase
end
end
endmodule
```

Test Bench:

```
module srff_test;
reg [0:1] sr;reg reset;  reg clk;
wire q; wire qbar;
srff uut (.sr(sr), .reset(reset),  .clk(clk),
                    .q(q),   .qbar(qbar)     );
          initial
          clk =1'b0;
          always #10 clk=~clk;
          initial begin
          reset=0;          #100;
          reset=1;
          sr = 00;#100;
          sr = 01;#100;
          sr = 10;#100;
          sr = 11;#100;
 end
endmodule
```

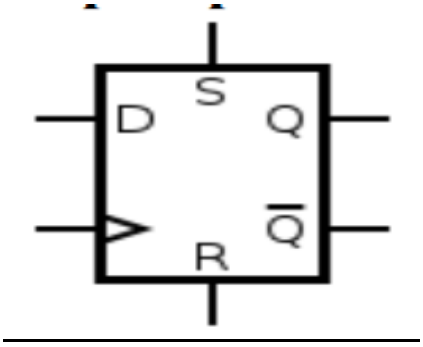
RESULT: Verilog code for the SR Flip Flop circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified

EXPT.NO.5 b

TITLE : Write Verilog Code for the D Flip Flop and their Test Bench for verification, observe the waveform and synthesize the code with technological library with given Constraints. Do the initial timing verification with gate level simulation.

TOOL REQUIRED: XILINX 10.2

THEORY: In electronics, a flip-flop or latch is a circuit that has two stable states and can be used to store state information. A flip-flop is a bistable multivibrator. The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs. It is the basic storage element in sequential logic. Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems. Flip-flops and latches are used as data storage elements. A flip-flop stores a single bit (binary digit) of data; one of its two states represents a "one" and the other represents a "zero". Such data storage can be used for storage of state, and such a circuit is described as sequential logic. D flip-flop The D flip-flop is widely used. It is also known as a "data" or "delay" flip-flop. The D flip-flop captures the value of the D-input at a definite portion of the clock cycle (such as the rising edge of the clock). That captured value becomes the Q output. At other times, the output Q does not change. The D flip-flop can be viewed as a memory cell, a zero-order hold, or a delay line. D flip-flop symbol



Verilog Code For D Flip-Flop

```
module dff( d, reset, clk,q, qbar );
input d ;
input reset, clk ;
output q, qbar ;
reg q, qbar ;
```

```
always @( reset, posedge clk )
begin
if ( reset == 0 )
begin
q = 1’b0;
qbar=1’b1;
end
else
begin
q = 1’b0;
qbar = 1’b1;
end
end
endmodule
```

Test Bench:

```
module dff_test;
        reg d;  reg reset;      reg clk;
        wire q; wire qbar;
        dff uut (.d(d), .reset(reset),.clk(clk),
                .q(q), .qbar(qbar)      );

initial
        clk =1'b0;
        always #10 clk=~clk;

        initial begin
        reset=0;        #100;
        reset=1;
        d=1'b0;         #100;
        d=1'b1;         #100;
        end
endmodule
```

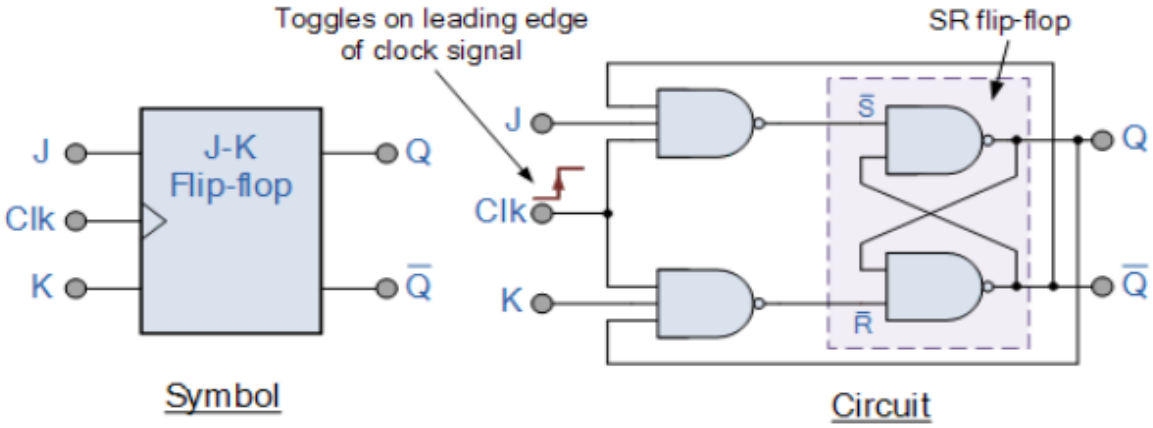
RESULT: Verilog code for the D Flip Flop circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.

EXPT.NO.5 c

TITLE : Write Verilog Code for the SR Flip Flop and their Test Bench for **verification**, observe the waveform and **synthesize** the code with technological library with given Constraints. Do the initial timing verification with gate level simulation.

TOOL REQUIRED: XILINX 10.2

THEORY: Then the JK flip-flop is basically an SR flip flop with feedback which enables only one of its two input terminals, either SET or RESET to be active at any one time thereby eliminating the invalid condition seen previously in the SR flip flop circuit. Also when both the J and the K inputs are at logic level “1” at the same time, and the clock input is pulsed either “HIGH”, the circuit will “toggle” from its SET state to a RESET state, or visa-versa. This results in the JK flip flop acting more like a T-type toggle flip-flop when both terminals are “HIGH”. Although this circuit is an improvement on the clocked SR flip-flop it still suffers from timing problems called “race” if the output Q changes state before the timing pulse of the clock input has time to go “OFF”. To avoid this the timing pulse period (T) must be kept as short as possible (high frequency). As this is sometimes not possible with modern TTL IC’s the much improved MasterSlave JK Flip-flop was developed.



Truth Table:

Input		Output		Description
J	K	Q	Q	
0	0	0	0	Memory no change
0	0	0	1	
0	1	1	0	Reset
0	1	0	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Toggle
1	1	1	0	

Verilog Code For JK Flip-Flop

```
module jkff( jk, reset, clk,q, qbar );
input [0:1]jk ;
input reset, clk ;
output q, qbar ;
reg q, qbar ;
always @( jk , reset, posedge clk )
begin
if ( reset == 0 )
begin
q = 1'b0;
qbar=1'b1;
end
else
begin
case(jk)
2'b00:begin q=q;qbar=qbar;end
2'b01:begin q=1'b0;qbar=1'b1;end
2'b10:begin q=1'b1;qbar=1'b0;end
2'b11:begin q=~q;qbar=~qbar;end
endcase
end
end
endmodule
```

Test Bench :

```
module jkff_test;
        reg [0:1] jk;reg reset;reg clk;
        wire q;wire qbar;
        jkff uut (.jk(jk),.reset(reset),.clk(clk),
                .q(q), .qbar(qbar));
        initial
        clk =1'b0;
        always #10 clk=~clk;
        initial begin
        reset=0;        #100;
        reset=1;
        jk=00;  #100;
        jk=01;  #100;
        jk=10;  #100;
        jk=11;  #100;
        end
endmodule
```

RESULT: Verilog code for the JK Flip Flop circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.

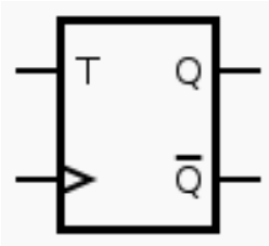
EXPT.NO.5 d

TITLE : Write Verilog Code for the SR Flip Flop and their Test Bench for **verification**, observe the waveform and **synthesize** the code with technological library with given Constraints. Do the initial timing verification with gate level simulation.

TOOL REQUIRED: XILINX 10.2

THEORY: When T is held high, the toggle flip-flop divides the clock frequency by two; that is, if clock frequency is 4 MHz, the output frequency obtained from the flip-flop will be 2 MHz. This "divide by" feature has application in various types of digital counters. A T flip-flop can also be built using a JK flip-flop (J & K pins are connected together and act as T) or a D flip-flop (T input XOR Qprevious drives the D input).

Symbol



Truth Table

Input	Output		Description
T	Q	Q	
0	0	0	Memory no change
0	1	1	
1	0	1	Toggle
1	1	0	

Verilog Code For T Flip-Flop

```
module tff( t, reset, clk,q, qbar );
input t ;
input reset, clk ;
output q, qbar ;
reg q, qbar ;
always @( t , reset, posedge clk )
begin
if ( reset == 0 )
begin
q = 1'b0;
qbar=1'b1;
end
else
begin
case(t)
1'b0: begin q=q; qbar=qbar; end
1'b1: begin q=qbar; qbar=~qbar; end
endcase
end
```

```
end
endmodule
```

Test Bench:

```
module tff_test;
        reg t;reg reset;reg clk;
        wire q;wire qbar;
        tff uut (.t(t),.reset(reset),.clk(clk),
                .q(q),.qbar(qbar));
        initial
        clk =1'b0;
        always #10 clk=~clk;

        initial begin
        reset=0;        #100;
        reset=1;
        t=1'b0; #100;
        t=1'b1; #100;
        end
endmodule
```

RESULT: Verilog code for the T Flip Flop circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.

EXPT.NO.6 a

TITLE : Write Verilog Code for the Serial adder circuit and their Test Bench for verification, observe the waveform and synthesize the code with technological library with given Constraints. Do the initial timing verification with gate level simulation.

TOOL REQUIRED: XILINX 10.2

THEORY: The serial binary adder or bit-serial adder is a digital circuit that performs binary addition bit by bit. The serial full adder has three single-bit inputs for the numbers to be added and the carry in. There are two single-bit outputs for the sum and carry out. The carry-in signal is the previously calculated carry-out signal. The addition is performed by adding each bit, lowest to highest, one per clock cycle. Serial binary addition is done by a flip-flop and a full adder. The flip-flop takes the carry-out signal on each clock cycle and provides its value as the carry-in signal on the next clock cycle. After all of the bits of the input operands have arrived, all of the bits of the sum have come out of the sum output.

Verilog Code For Serial Adder

```
module serial_adder(a,b,sum,cout);
input [3:0] a,b;
output cout;
output [3:0] sum;
reg [3:0] sum;
reg cout;
reg [4:0] carry;
integer i;
always @(a,b)
begin
carry[0]=0;
for(i=0;i<=3;i=i+1)
begin
sum[i] = a[i] ^ b[i] ^ carry[i];
carry [i+1] = ( a[i] & b[i] ) | (b[i] & carry[i]) | (carry[i] & a[i] ) ;
end
cout = carry[4];
end
endmodule
```

Test Bench:

```
module serial_adder_test;
        reg [3:0] a;reg [3:0] b;
```

```
wire [3:0] sum;wire cout;
serial_adder uut (.a(a),.b(b),
        .sum(sum),.cout(cout) );
initial begin
a = 0101;b = 1101; #100;
a = 0111;b = 1001; #100;
a = 1011;b = 0001; #100;
a = 0001;b = 1100; #100;
end

endmodule
```

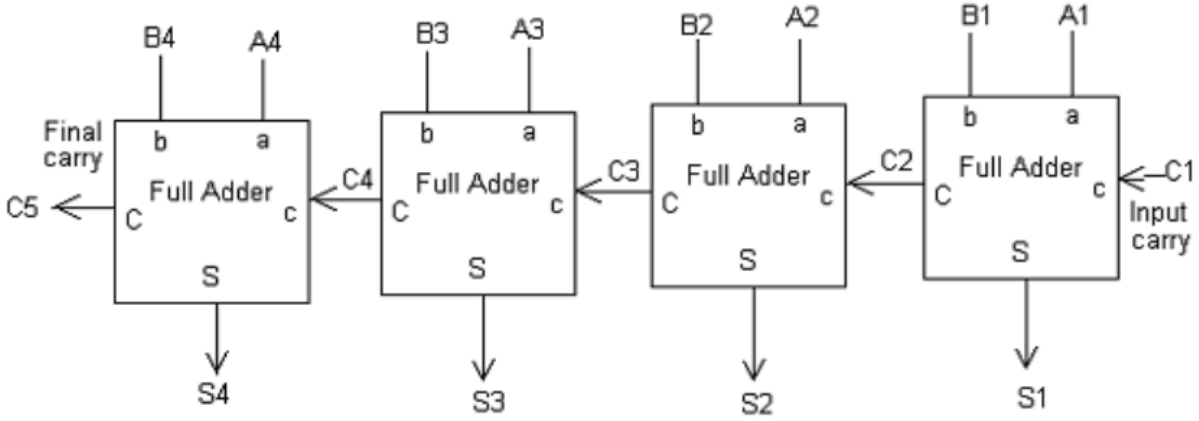
RESULT: Verilog code for the serial adder circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.

EXPT.NO.6 b

TITLE : Write Verilog Code for the parallel adder circuit and their Test Bench for verification, observe the waveform and synthesize the code with technological library with given Constraints. Do the initial timing verification with gate level simulation.

TOOL REQUIRED: XILINX 10.2

THEORY: Addition is a fundamental operation for any digital system, digital signal processing or control system. A fast and accurate operation of a digital system is greatly influenced by the performance of the resident adders. Adders are also very important component in digital systems because of their extensive use in other basic digital operations such as subtraction, multiplication and division Parallel adder is a combinatorial circuit (not clocked, does not have any memory and feedback) adding every bit position of the operands in the same time. Thus it is requiring number of bit-adders(full adders + 1 half adder) equal to the number of bits to be added. The Parallel adder is constructed by cascading full adders (FA) blocks in series. One full adder is responsible for the addition of two binary digits at any stage of the ripple carry. The carryout of one stage is fed directly to the carry-in of the next stage.



6. b) Verilog Code For Parallel Adder

```
module parallel ( a, b, cin, sum, cout );
input [3:0] a, b ;
input cin ;
output [3:0] sum;
output cout;
wire [3:0] c;
fa stage0 ( a[0], b[0], cin, sum[0], c[0] ) ;
fa stage1 ( a[1], b[1], c[0], sum[1], c[1] ) ;
```

```
fa stage2 ( a[2], b[2], c[1], sum[2], c[2] ) ;
fa stage3 ( a[3], b[3], c[2], sum[3], c[3] ) ;
assign cout = c[3] ;
endmodule
```

Component Code

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

Test Bench:

```
module parallel_test;
        reg [3:0] a;reg [3:0] b;reg cin;
        wire [3:0] sum;wire cout;
        parallel uut (
                .a(a),.b(b),.cin(cin),
                .sum(sum),.cout(cout)
                        );
        initial begin
        a = 0101;b = 1101; cin=0;#100;
        a = 0111;b = 1001; cin=0;#100;
        a = 1011;b = 0001; cin=0;#100;
        a = 0001;b = 1100; cin=0;#100;
        end
endmodule
```

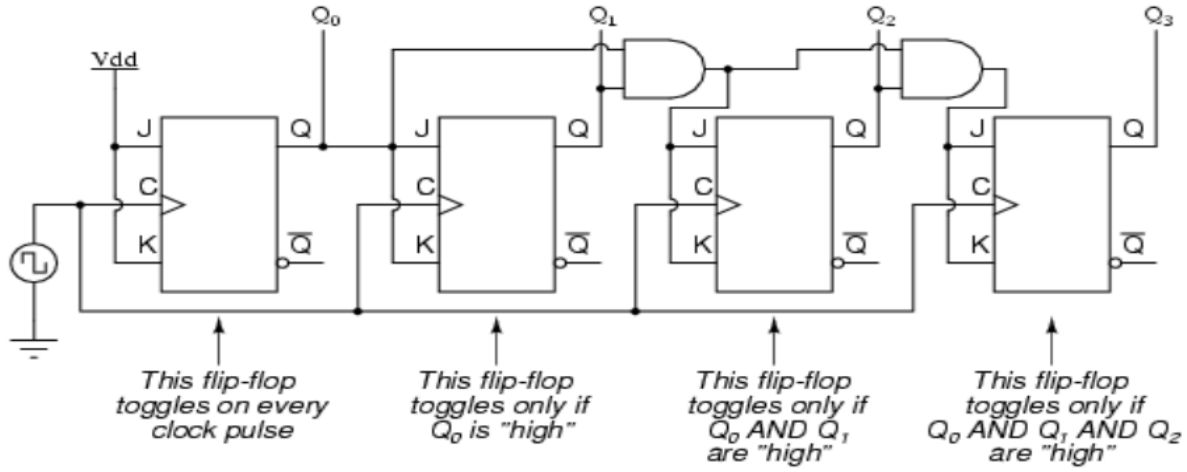
RESULT: Verilog code for the parallel adder circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.

EXPT.NO.7 a

TITLE : Write Verilog Code for the synchronous counter circuit and their Test Bench for verification, observe the waveform and synthesize the code with technological library with given Constraints. Do the initial timing verification with gate level simulation.

TOOL REQUIRED: XILINX 10.2

THEORY: A synchronous counter, in contrast to an asynchronous counter, is one whose output bits change state simultaneously, with no ripple. The only way we can build such a counter circuit from J-K flip-flops is to connect all the clock inputs together, so that each and every flip-flop receives the exact same clock pulse at the exact same time. The figure shows a four-bit synchronous “up” counter. Each of the higher-order flip-flops are made ready to toggle (both J and K inputs “high”) if the Q outputs of all previous flip-flops are “high.” Otherwise, the J and K inputs for that flip-flop will both be “low,” placing it into the “latch” mode where it will maintain its present output state at the next clock pulse. Since the first (LSB) flip-flop needs to toggle at every clock pulse, its J and K inputs are connected to Vcc or Vdd, where they will be “high” all the time. The next flip-flop need only “recognize” that the first flip-flop’s Q output is high to be made ready to toggle, so no AND gate is needed. However, the remaining flip-flops should be made ready to toggle only when all lower-order output bits are “high,” thus the need for AND gates.



I) Verilog Code For Synchronous Down Counter

```
module sync ( clk, q, qbar ) ;
input clk ;
output [3:0] q ;
output [3:0] qbar ;
```

```
wire [1:0] s;
supply1 vdd ;
jkff c0 ( vdd, vdd, clk , q[0], qbar [0] ) ;
jkff c1 (qbar [0], qbar [0], clk , q[1], qbar [1] ) ;
assign s[0]= qbar[0] & qbar[1];
jkff c2 (s[0],s[0], clk ,q[2], qbar [2]  ) ;
assign s[1]= qbar[0] & qbar[1] & qbar[2] ;
jkff c3 (s[1], s[1] , clk , q[3], qbar [3] ) ;
endmodule
```

Component Code

```
  module jkff ( j, k ,clk, qx, qxbar ) ;
input clk , j , k ;
output qx, qxbar ;
reg qx, qxbar ;
initial
begin
qx = 1'b0 ;
qxbar = 1'b1 ;
end
always @ ( posedge clk )
begin
qx = ( j & ( ~ qx ) ) | ( ( ~ k ) & qx ) ;
qxbar = ~ qx ;
end
endmodule
```

Test Bench :

```
module sync_test;
        reg clk;
        wire [3:0] q;wire [3:0] qbar;
        async uut (.clk(clk),
                .q(q),.qbar(qbar) );

        initial
        clk =1'b0;
        always #10 clk=~clk;
        endmodule
```

II) Verilog Code For Synchronous Down Counter

```
module sync ( clk, q, qbar ) ;
input clk ;
output [3:0] q ;
output [3:0] qbar ;
```

```
wire [1:0] s;
supply1 vdd ;
jkff c0 ( vdd, vdd, clk , q[0], qbar [0] ) ;
jkff c1 (q [0], q [0], clk , q[1], qbar [1] ) ;
assign s[0]= q[0] & q[1];
jkff c2 (s[0],s[0], clk ,q[2], qbar [2]  ) ;
assign s[1]= q[0] & q[1] & q[2] ;
jkff c3 (s[1], s[1] , clk , q[3], qbar [3] ) ;
endmodule
```

Component Code

```
  module jkff ( j, k ,clk, qx, qxbar ) ;
input clk , j , k ;
output qx, qxbar ;
reg qx, qxbar ;
initial
begin
qx = 1'b0 ;
qxbar = 1'b1 ;
end
always @ ( posedge clk )
begin
qx = ( j & ( ~ qx ) ) | ( ( ~ k ) & qx ) ;
qxbar = ~ qx ;
end
endmodule
```

Test Bench :

```
module sync_test;
        reg clk;
        wire [3:0] q;wire [3:0] qbar;
        async uut (.clk(clk),
                .q(q),.qbar(qbar));
        initial
        clk =1'b0;
        always #10 clk=~clk;
        endmodule
```

RESULT: Verilog code for the synchronous counter circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.

EXPT.NO.7 b

TITLE : Write Verilog Code for the asynchronous counter circuit and their Test Bench for verification, observe the waveform and synthesize the code with technological library with given Constraints. Do the initial timing verification with gate level simulation.

TOOL REQUIRED: XILINX 10.2

THEORY: Asynchronous counters are those whose output is free from the clock signal. Because the flip flops in asynchronous counters are supplied with different clock signals, there may be delay in producing output. The required number of logic gates to design asynchronous counters is very less. So they are simple in design. Another name for Asynchronous counters is "Ripple counters". The number of flip flops used in a ripple counter is depends up on the number of states of counter (ex: Mod 4, Mod 2 etc). The number of output states of counter is called "Modulus" or "MOD" of the counter. The maximum number of states that a counter can have is 2n where n represents the number of flip flops used in counter. For example, if we have 2 flip flops, the maximum number of outputs of the counter is 4 i.e. 22. So it is called as "MOD-4 counter" or "Modulus 4 counter".

I) Verilog Code For Asynchronous Down Counter

```
module async ( clk, q, qbar ) ;
input clk ;
output [3:0] q ;
output [3:0] qbar ;
supply1 vdd ;
jkff c0 ( vdd, vdd, clk , q[0], qbar [0] ) ;
jkff c1 (vdd, vdd, q[0] , q[1], qbar [1] ) ;
jkff c2 (vdd, vdd, q[1], q[2], qbar [2] ) ;
jkff c3 (vdd, vdd, q[2], q[3], qbar [3] ) ;
endmodule
```

Component Code

```
  module jkff ( j, k ,clk, qx, qxbar ) ;
input clk , j , k ;
output qx, qxbar ;
reg qx, qxbar ;
initial
begin
qx = 1'b0 ;
qxbar = 1'b1 ;
end
always @ ( posedge clk )
begin
qx = ( j & ( ~ qx ) ) | ( ( ~ k ) & qx ) ;
```

```
qxbar = ~ qx ;
end
endmodule
```

Test Bench :

```
module sync_test;
        reg clk;
        wire [3:0] q;wire [3:0] qbar;
        async uut (
                .clk(clk),
                .q(q),.qbar(qbar)
        );

        initial
        clk =1'b0;
        always #10 clk=~clk;
        endmodule
```

II) Verilog Code For Asynchronous Up Counter

```
module async ( clk, q, qbar ) ;
input clk ;
output [3:0] q ;
output [3:0] qbar ;
supply1 vdd ;
jkff c0 ( vdd, vdd, clk , q[0], qbar [0] ) ;
jkff c1 (vdd, vdd, qbar[0] , q[1], qbar [1] ) ;
jkff c2 (vdd, vdd, qbar[1], q[2], qbar [2] ) ;
jkff c3 (vdd, vdd, qbar[2], q[3], qbar [3] ) ;
endmodule
```

Component Code

```
  module jkff ( j, k ,clk, qx, qxbar ) ;
input clk , j , k ;
output qx, qxbar ;
reg qx, qxbar ;
initial
begin
qx = 1’b0 ;
qxbar = 1’b1 ;
end
always @ ( posedge clk )
begin
qx = ( j & ( ~ qx ) ) | ( ( ~ k ) & qx ) ;
qxbar = ~ qx ;
end
```

```
endmodule
```

Test Bench :

```
module sync_test;
        reg clk;
        wire [3:0] q;wire [3:0] qbar;
        async uut (
                .clk(clk),
                .q(q),.qbar(qbar)
        );

        initial
        clk =1'b0;
        always #10 clk=~clk;
        endmodule
```

RESULT: Verilog code for the asynchronous counter circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified

PART - B
ANALOG DESIGN

Steps of Execution for schematic :

1. Start RedHat OS
2. Click on **vsmitvlsi**
3. Password: **vlsilab**
4. Select network connection: **system eth0**
5. Right click and select **Open in Terminal**
6. Write **cd** enter, **source ams.cshrc** enter, **dmgr_ic** enter
7. Click on **File** --- **New** --- **Project** ----
8. Project path: /home/vsmitvlsi/**inverter**
9. Library: select: **PDK**---**generic13** ----- enter **OK**
10. Click on **add standard library** --- click --- **OK**
11. Right click on project--- new --- library ---- library name: **inv --- OK**
12. Right click on library inv --- new --- **Schematic ----**
13. Cell name **--- inv ---** press OK
14. After blank schematic window opens, **Add instance**
15. **Generic13/symbols – pmos,nmos,resistance,capacitance**
16. **Genericlib --- vdd, vss, ground etc**
17. **Source --- dc, ac, pulse voltage, current sources etc**
18. Draw Schematic Diagram with appropriate ports
19. Click on **save & check**
20. **Add --- generic symbol --- Choose shape**
21. **Customize pinlist --- press OK**
22. **Save & check the generated symbol**
23. In the project navigator **--- right click --- schematic (for test circuit)**
24. Select the schematic instance
25. Draw the test circuit --- **Save**
26. Enter Simulation mode
27. New design configuration
28. Name: **eldo ---** Press OK
29. Choose **Setup Environment**
30. **Viewer ---** select **----Satrt EZwave Automatically --- Press OK**
31. **Setup simulation --- Analysis --- Transient** (give appropriate values)
32. **Setup simulation --- Analysis --- DC** (give appropriate values)
33. **Setup simulation --- Analysis --- AC** (give appropriate values)
34. **Setup simulation --- libraries – Typical**
35. **Setup simulation --- Includes --- unselect include_all**
36. **Setup simulation – forces --- from schematic --- press ctrl & choose input output**
37. **Setup simulation – output --- (Choose inputs and outputs) Add**
38. **Save --- Run**

Steps of Execution for Layout:

1. In the project navigator **--- Select Schematic & in Project Hierarchy – right click --- new --- layout**
2. **Choose OK to Launch Editor**
3. **Setup – toolbar – SDL toolbar**
4. **Pick and place component from SDL tollbar**
5. **Pick and place ports from SDL toolbar**
6. **Setup – windows – object editor – add device -- $gb_p – psub, nwell for nmos , pmos**
7. **IRoute for connectivity using appropriate layers , add --- text on ports – M1 – OK**
8. **Save the layout**
9. **Tools -- Calibre DRC – Run DRC**
10. **Tools – Calibre LVS – Run LVS**
11. **Tick Mark and Smileys indicates NO LVS Errors**
12. **Tools – Calibre PEX – input – Browse " *.src.net " –input – Netlist – enable export from schematic – Run PEX**

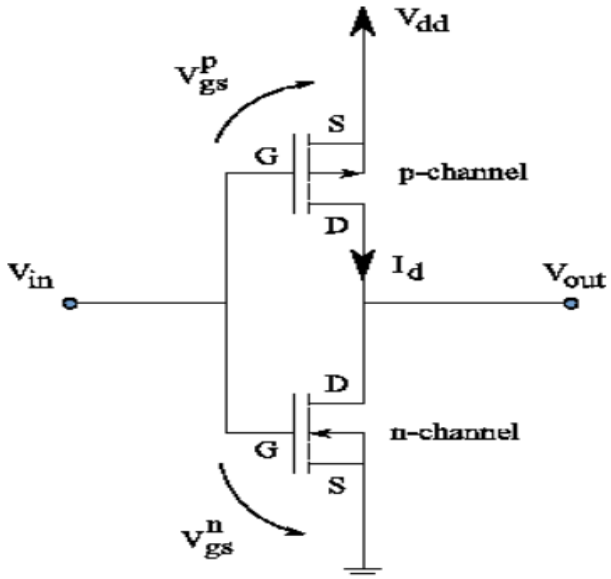
EXPT.NO.1

TITLE : To simulate the schematic of the CMOS inverter, and then to perform the physical verification for the layout of the same.

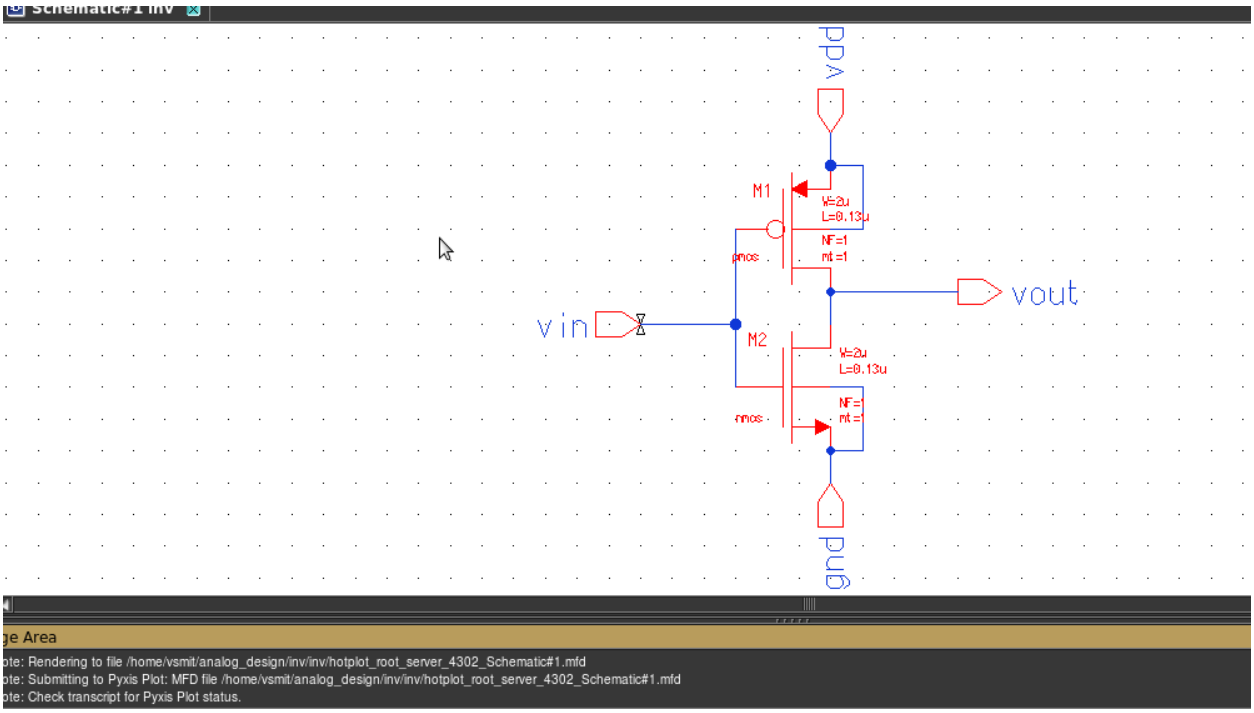
TOOL REQUIRED: Mentorgraphics

THEORY: The inverter is universally accepted as the most basic logic gate doing a Boolean operation on a single input variable. Fig.1 depicts the symbol, truth table and a general structure of a CMOS inverter. As shown, the simple structure consists of a combination of an pMOS transistor at the top and a nMOS transistor at the bottom.

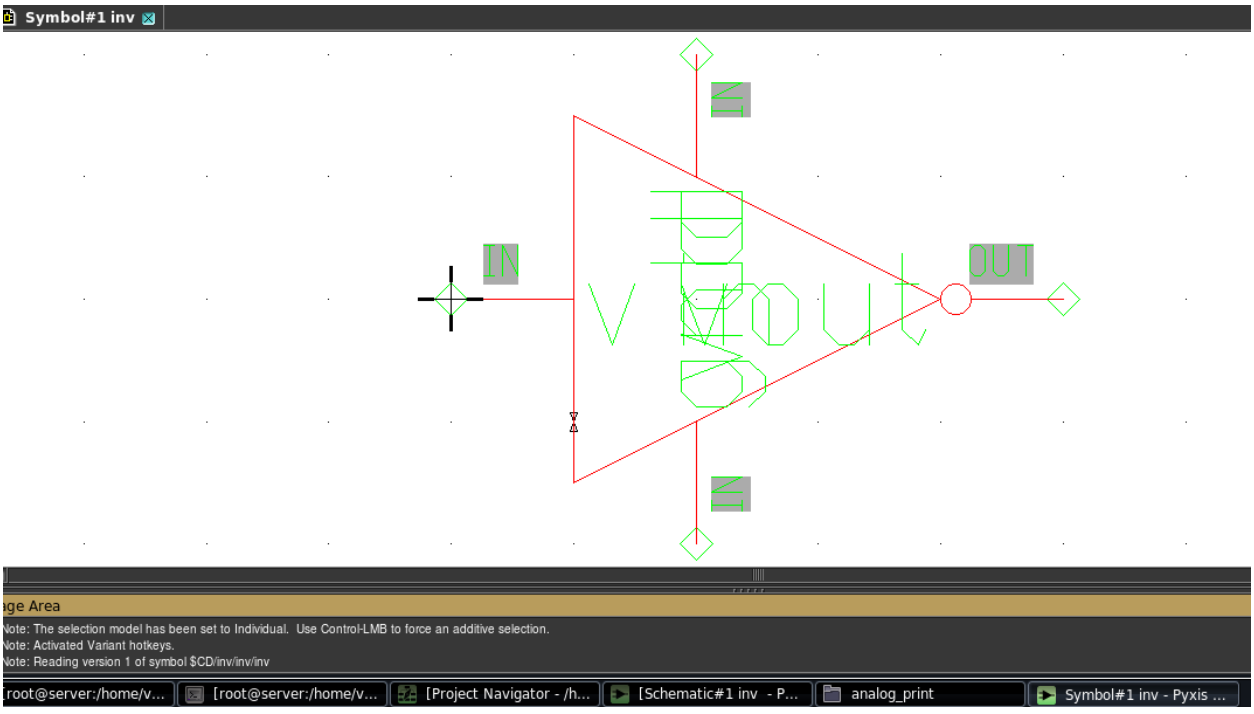
CMOS is also sometimes referred to as complementary-symmetry metal–oxide–semiconductor. The words "complementary-symmetry" refer to the fact that the typical digital design style with CMOS uses complementary and symmetrical pairs of p-type and n-type metal oxide semiconductor field effect transistors (MOSFETs) for logic functions. Two important characteristics of CMOS devices are high noise immunity and low static power consumption. Significant power is only drawn while the transistors in the CMOS device are switching between on and off states. Consequently, CMOS devices do not produce as much waste heat as other forms of logic, for example transistortransistor logic (TTL) or NMOS logic, which uses all n-channel devices without p-channel devices.



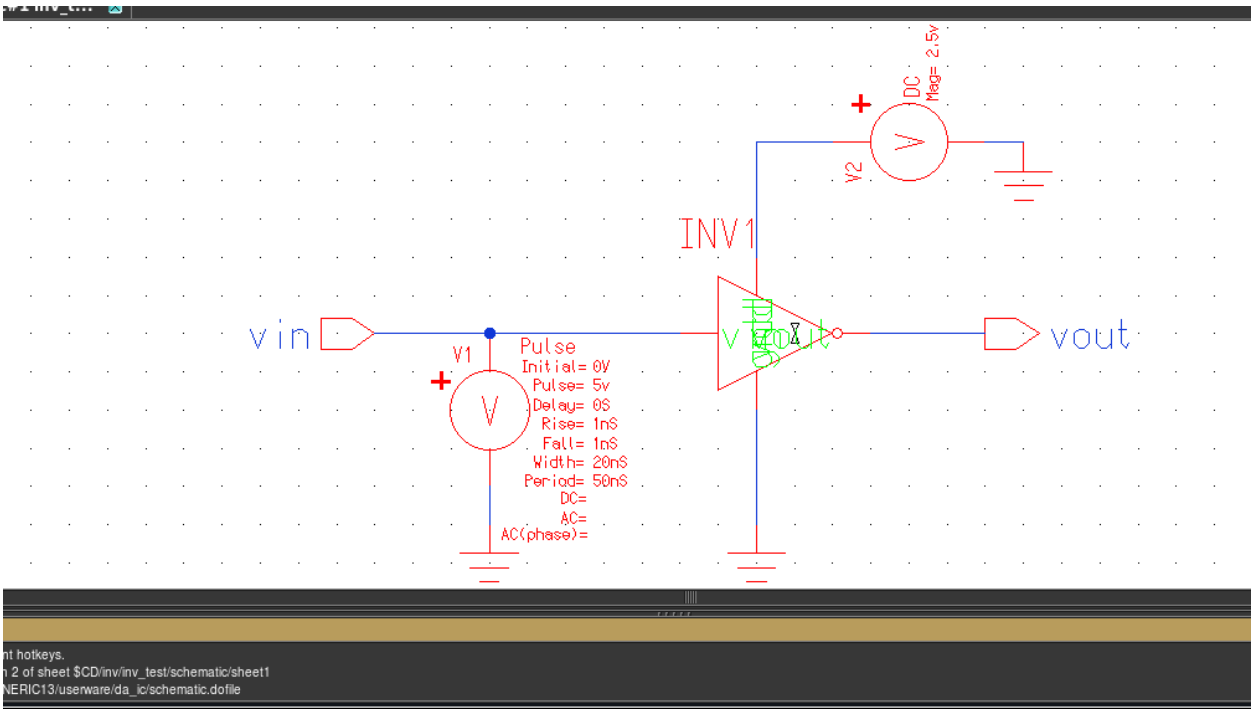
Inverter Schematic



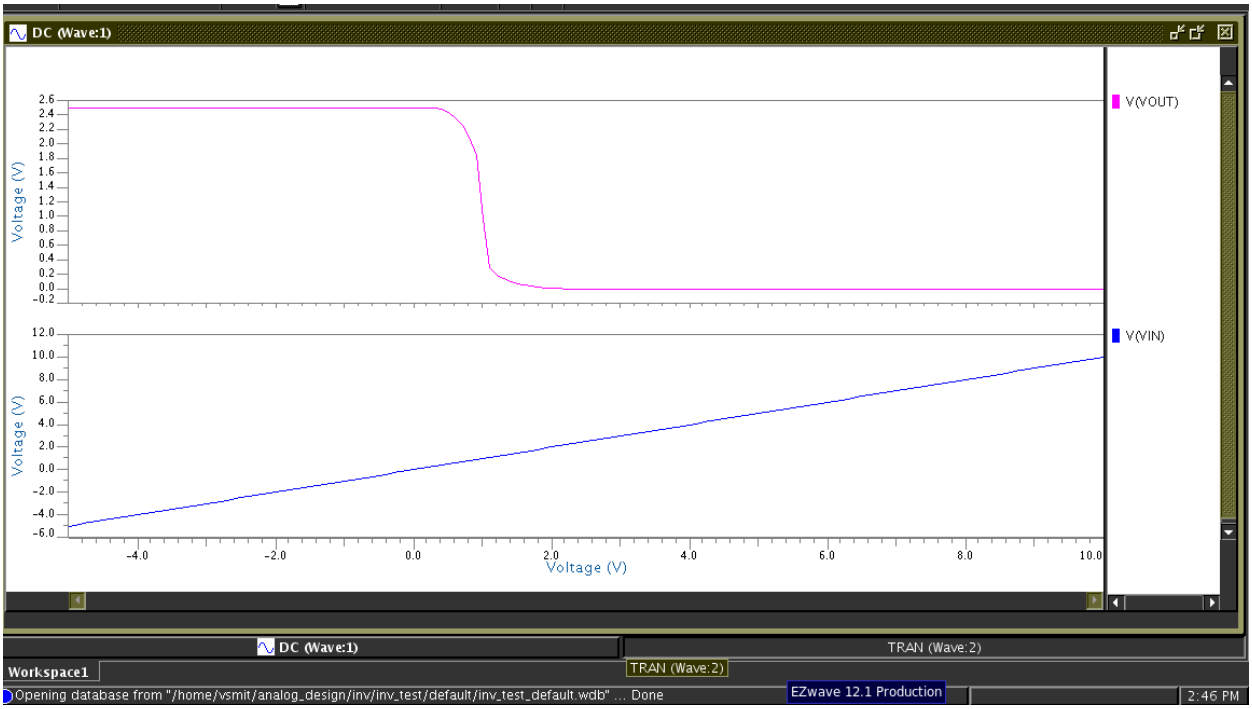
Inverter Symbol



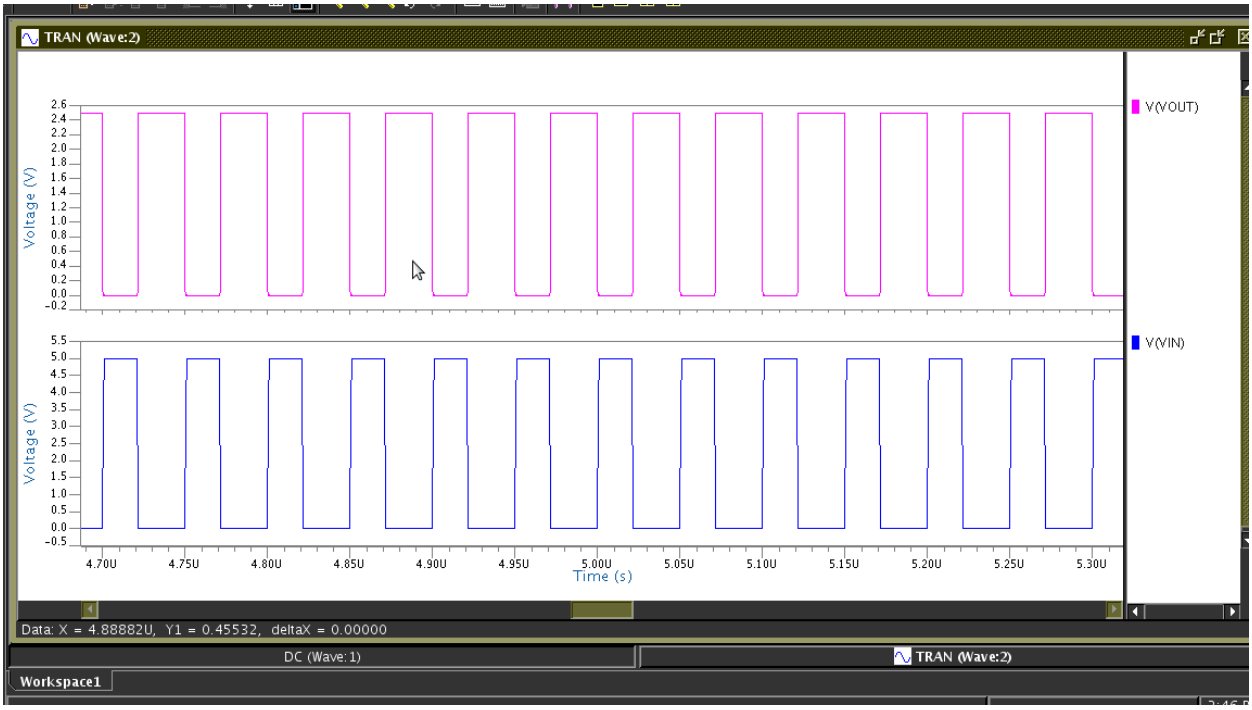
Inverter Test



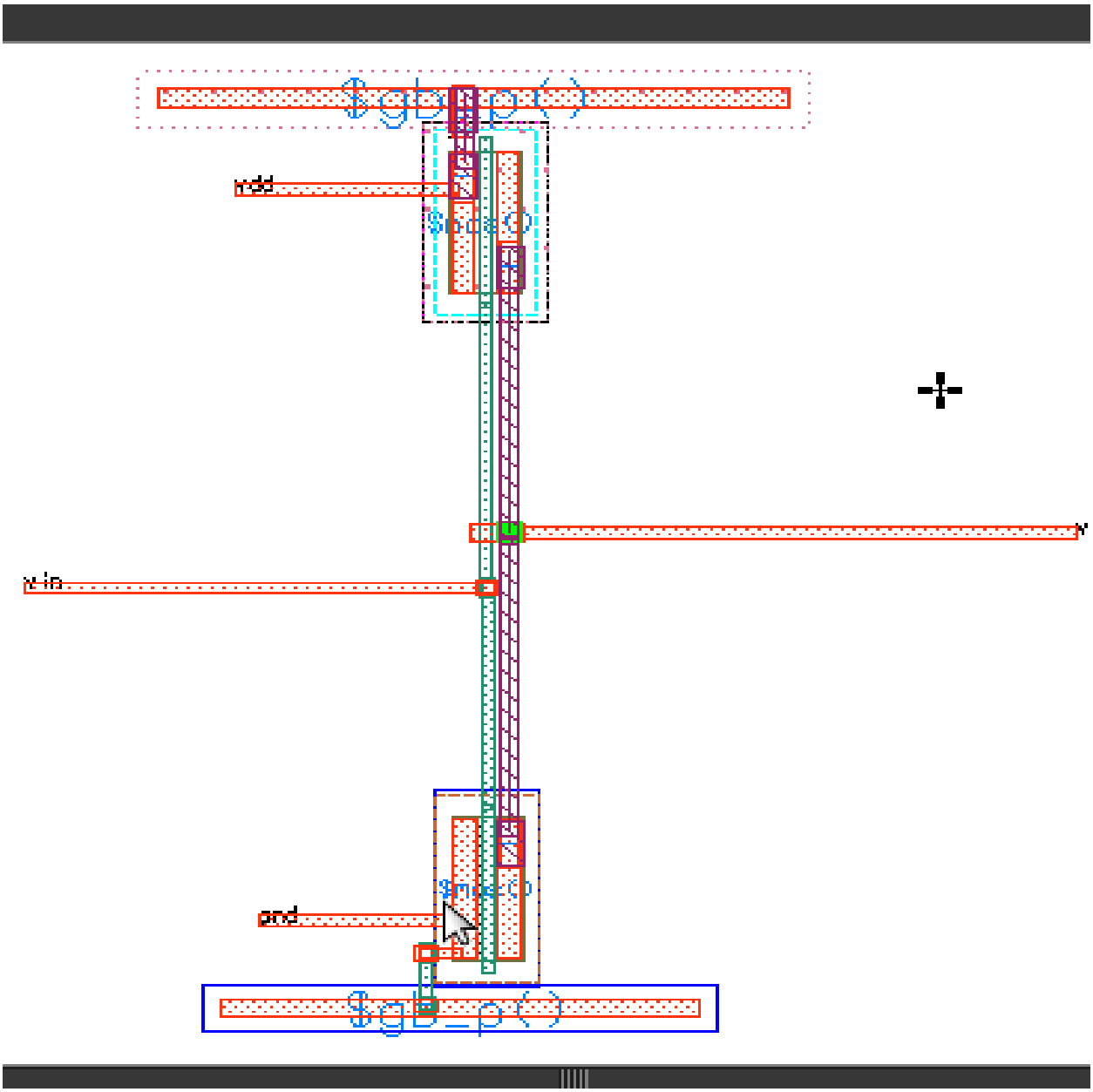
Inverter DCAnalysis



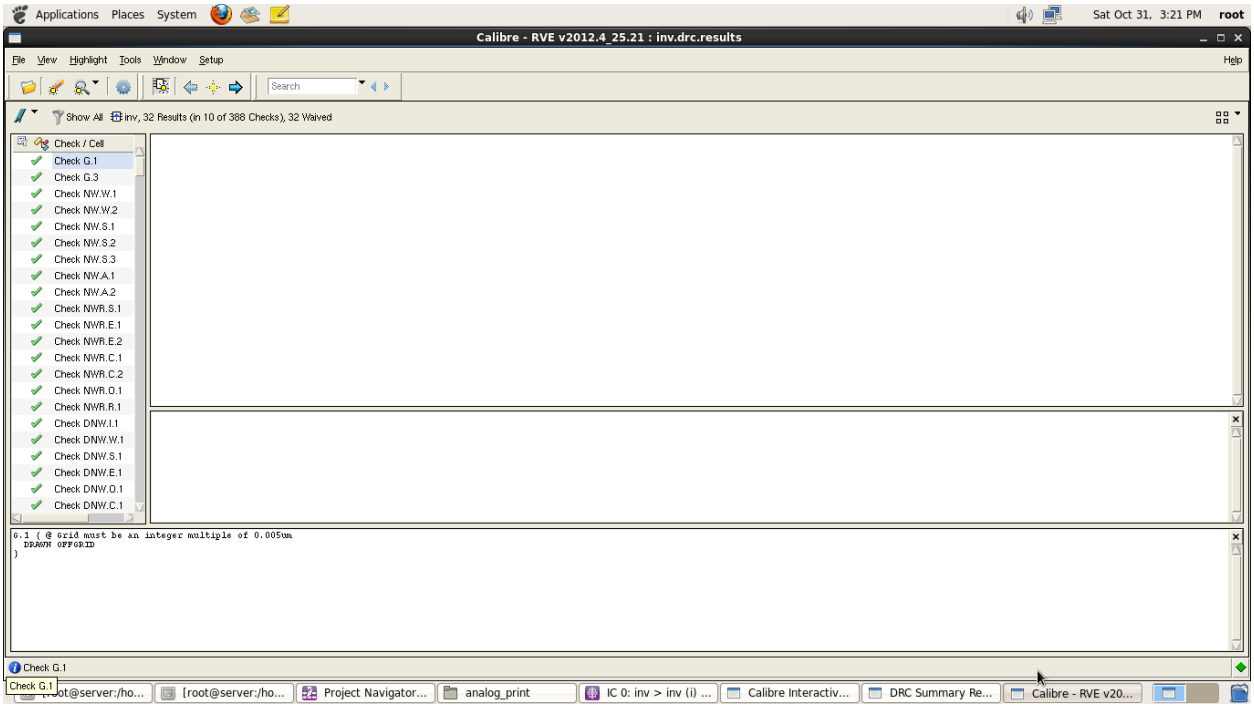
Inverter Transient Analysis



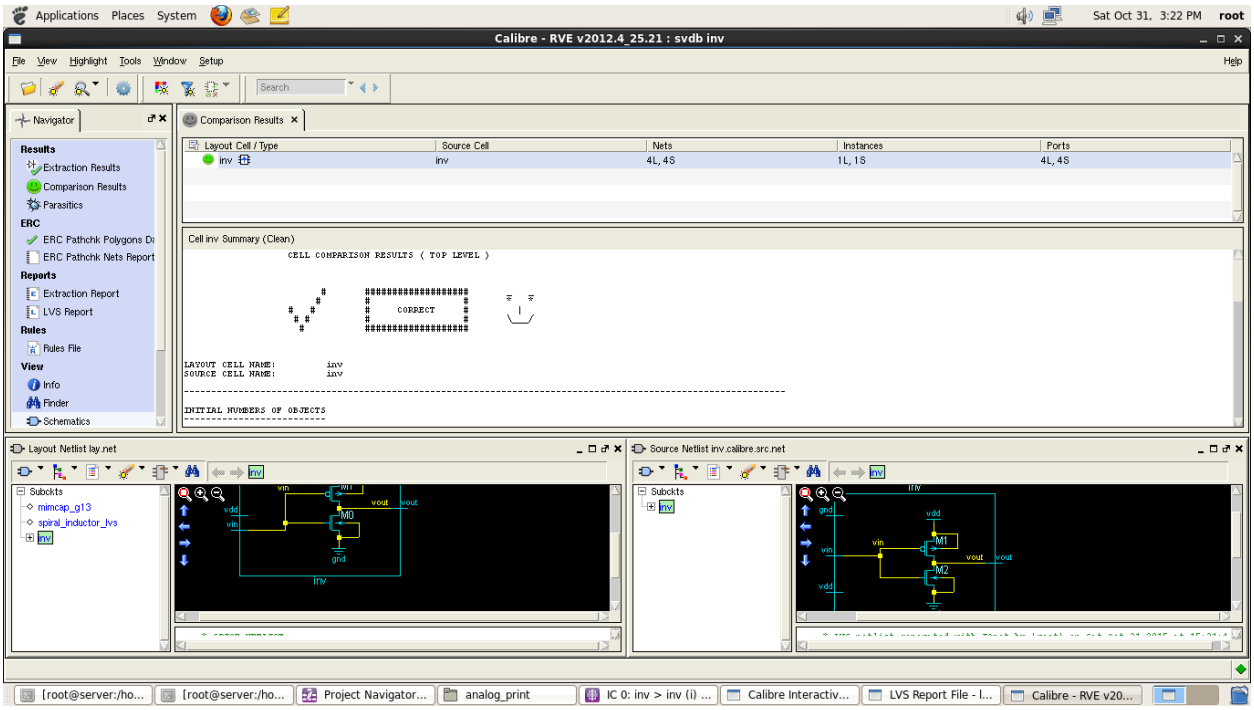
Inverter Layout



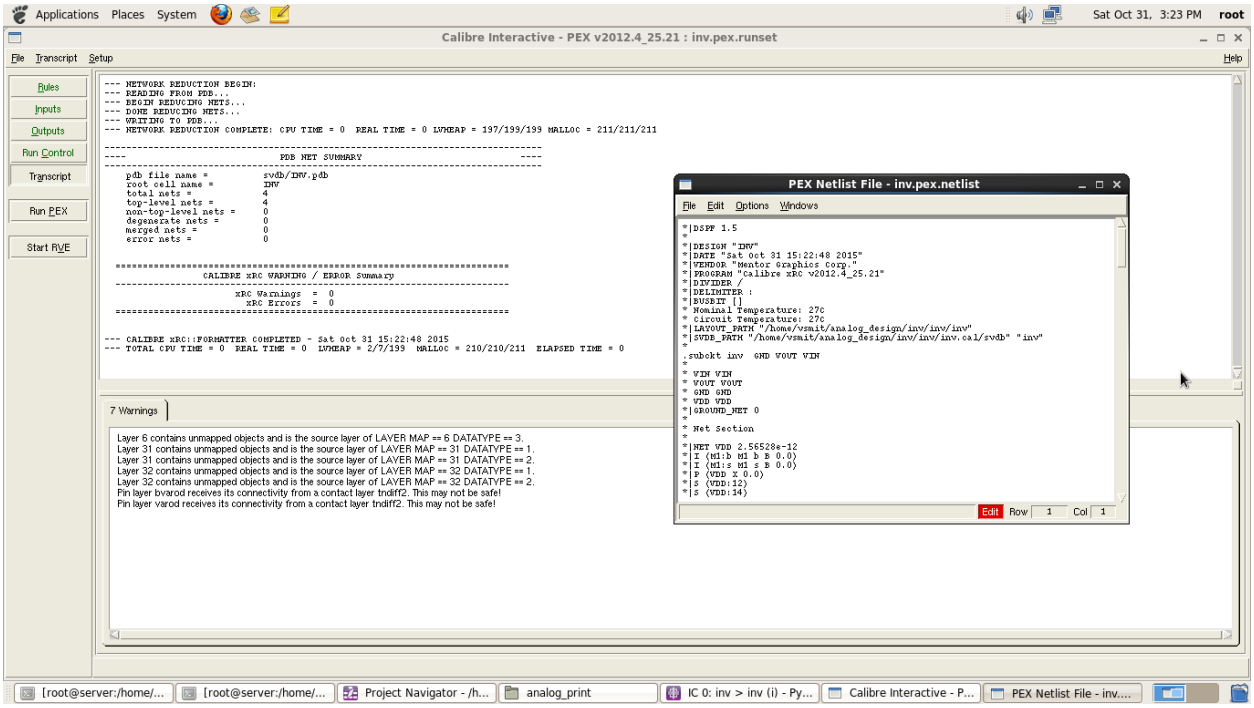
Inverter DRC



Inverter LVS



Inverter PEX

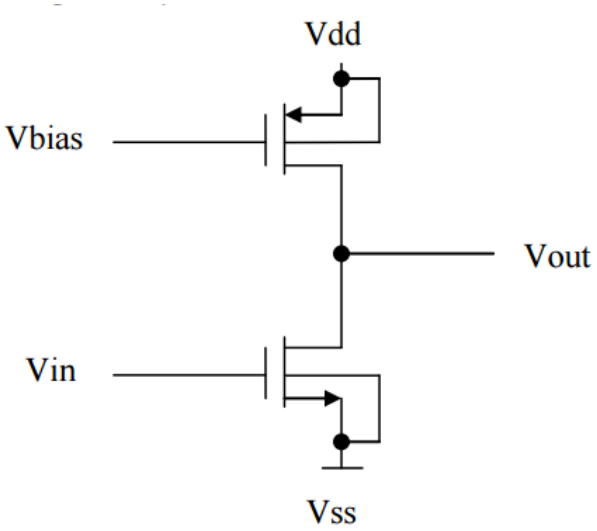

Applications Places System
Sat Oct 31, 3:23 PM root
Calibre Interactive - PEX v2012.4_25.21 : inv.pex.runset
File Transcript Setup
Help
Rules
Inputs
Outputs
Run Control
Transcript
Run PEX
Start RVE
--- NETWORK REDUCTION BEGIN:
--- READING FROM PDB...
--- BEGIN REDUCING NETS...
--- DONE REDUCING NETS...
--- WRITING TO PDB...
--- NETWORK REDUCTION COMPLETE: CPU TIME = 0 REAL TIME = 0 LVMEAP = 197/199/199 MALLOC = 211/211/211
PDB NET SUMMARY
pdb file name = svdb/INV.pdb
root cell name = INV
total nets = 4
top-level nets = 4
non-top-level nets = 0
degenerate nets = 0
merged nets = 0
error nets = 0
CALIBRE xRC WARNING / ERROR Summary
xRC Warnings = 0
xRC Errors = 0
--- CALIBRE xRC::FORMATTER COMPLETED - Sat Oct 31 15:22:48 2015
--- TOTAL CPU TIME = 0 REAL TIME = 0 LVMEAP = 2/7/199 MALLOC = 210/210/211 ELAPSED TIME = 0
PEX Netlist File - inv.pex.netlist
File Edit Options Windows
*|DSPF 1.5
*
*|DESIGN "INV"
*|DATE "Sat Oct 31 15:22:48 2015"
*|VENDOR "Mentor Graphics Corp."
*|PROGRAM "Calibre xRC v2012.4_25.21"
*|DIVIDER /
*|DELIMITER :
*|BUSBIT []
* Nominal Temperature: 27C
* Circuit Temperature: 27C
*|LAYOUT_PATH "/home/vsmit/analog_design/inv/inv/inv"
*|SVDB_PATH "/home/vsmit/analog_design/inv/inv/inv.cal/svdb" "inv"
*
.subckt inv GND VOUT VIN
*
* VIN VIN
* VOUT VOUT
* GND GND
* VDD VDD
*|GROUND_NET 0
*
* Net Section
*
*|NET VDD 2.56528e-12
*|I (M1:b M1 b B 0.0)
*|I (M1:s M1 s B 0.0)
*|P (VDD X 0.0)
*|S (VDD:12)
*|S (VDD:14)
Edit Row 1 Col 1
7 Warnings
Layer 6 contains unmapped objects and is the source layer of LAYER MAP == 6 DATATYPE == 3.
Layer 31 contains unmapped objects and is the source layer of LAYER MAP == 31 DATATYPE == 1.
Layer 31 contains unmapped objects and is the source layer of LAYER MAP == 31 DATATYPE == 2.
Layer 32 contains unmapped objects and is the source layer of LAYER MAP == 32 DATATYPE == 1.
Layer 32 contains unmapped objects and is the source layer of LAYER MAP == 32 DATATYPE == 2.
Pin layer bvarod receives its connectivity from a contact layer tndiff2. This may not be safe!
Pin layer varod receives its connectivity from a contact layer tndiff2. This may not be safe!
[root@server:/home/...
[root@server:/home/...
Project Navigator - /h...
analog_print
IC 0: inv > inv (i) - Py...
Calibre Interactive - P...
PEX Netlist File - inv....

EXPT.NO. 2 a

TITLE : To simulate the schematic of the CMOS common aource amplifier, and then to perform the physical verification for the layout of the same.

TOOL REQUIRED: Mentorgraphics

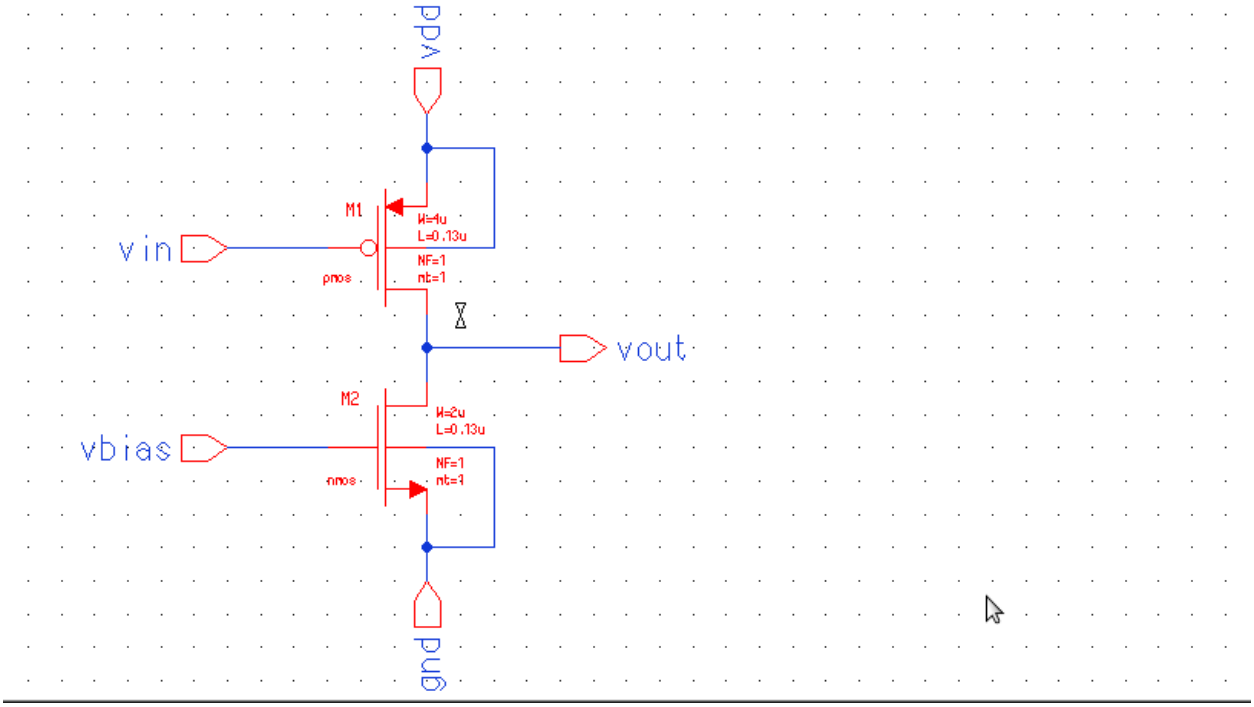
THEORY: In electronics, a common-source amplifier is one of three basic single-stage field-effect transistor (FET) amplifier topologies, typically used as a voltage or transconductance amplifier. The easiest way to tell if a FET is common source, common drain, orcommon gate is to examine where the signal enters and leaves. The remaining terminal is what is known as "common". In this example, the signal enters the gate, and exits the drain. The only terminal remaining is the source. This is a common-source FET circuit. The analogous bipolar junction transistor circuit is the common-emitter amplifier. The common-source (CS) amplifier may be viewed as a transconductance amplifier or as a voltage amplifier. (See classification of amplifiers). As a transconductance amplifier, the input voltage is seen as modulating the current going to the load. As a voltage amplifier, input voltage modulates the amount of current flowing through the FET, changing the voltage across the output resistance according to Ohm's law. However, the FET device's output resistance typically is not high enough for a reasonable transconductance amplifier (ideally infinite), nor low enough for a decent voltage amplifier (ideally zero). Another major drawback is the amplifier's limited high-frequency response.



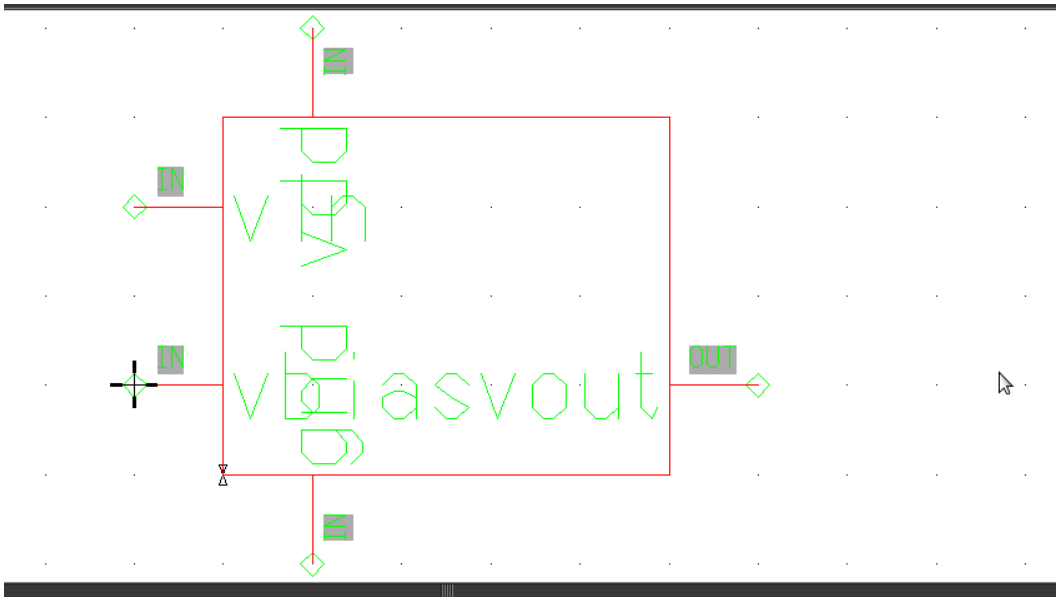
Common drain amplifier is a source follower or buffer amplifier circuit using a MOSFET. The output is simply equal to the input minus about 2.2V. The advantage of this circuit is that the MOSFET can provide current and power gain; the MOSFET draws no current from the input. It provides low output impedance to any circuit using the output of the follower, meaning that the output will not drop under load. Its output impedance is not as low as that of an emitter follower

using a bipolar transistor (as you can verify by connecting a resistor from the output to -15V), but it has the advantage that the input impedance is infinite.The MOSFET is in saturation, so the current across it is determined by the gatesource voltage. Since a current source keeps the current constant, the gate-source voltage is also constant.

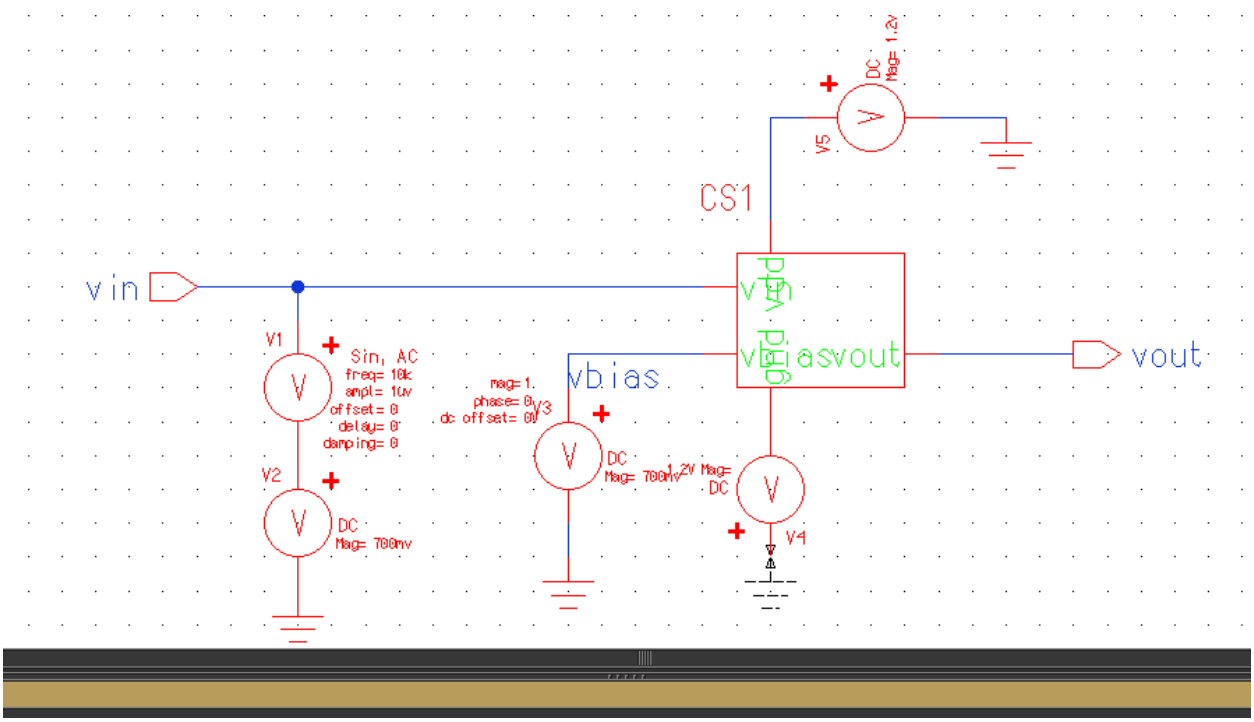
Common Source Schematic



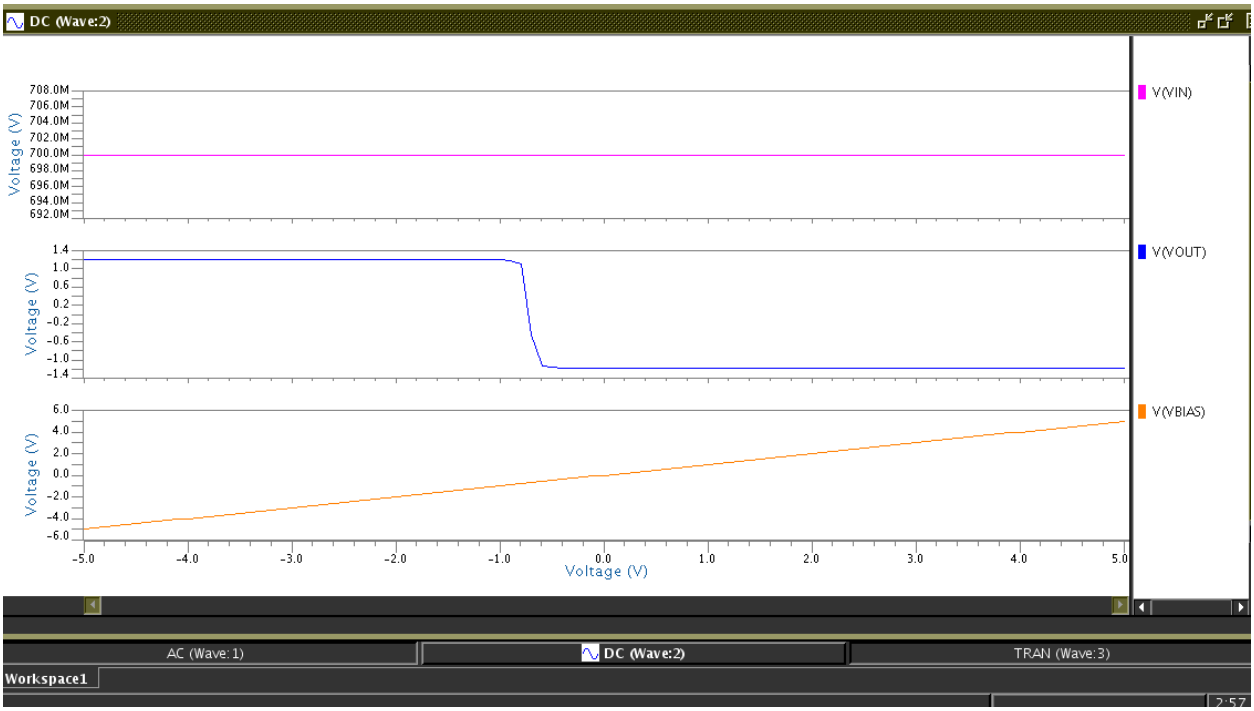
Common Source Symbol



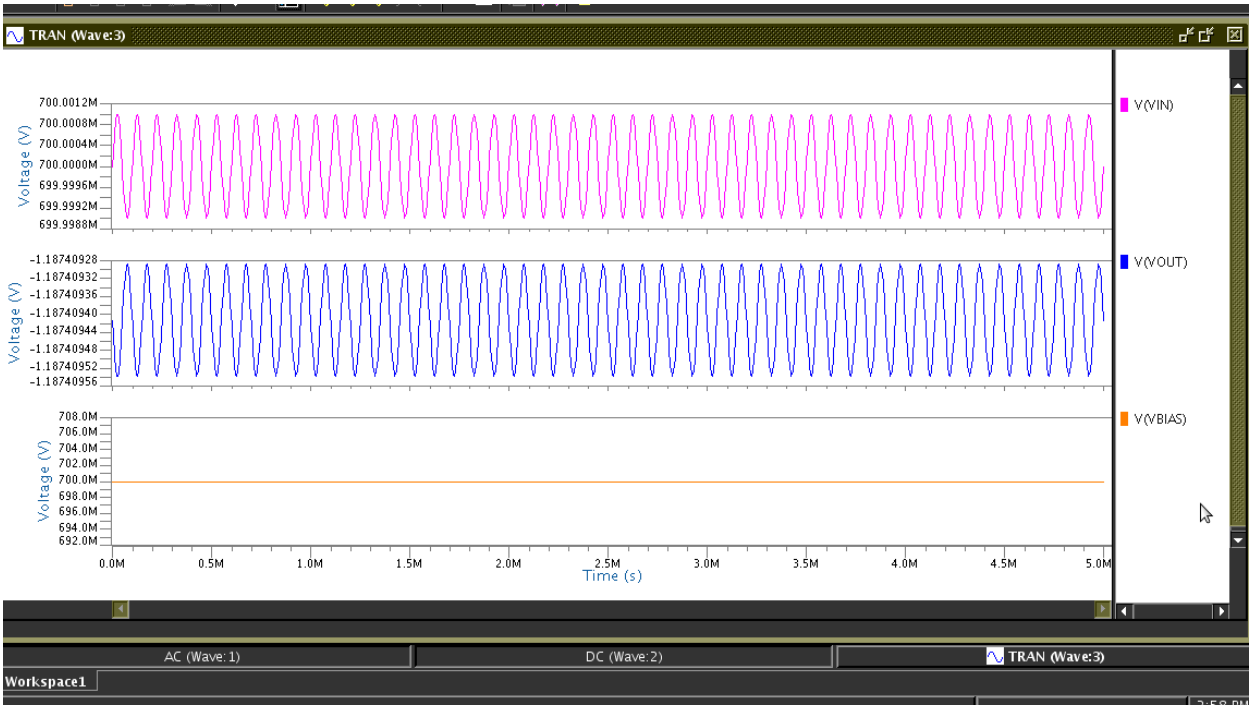
Common Source Test



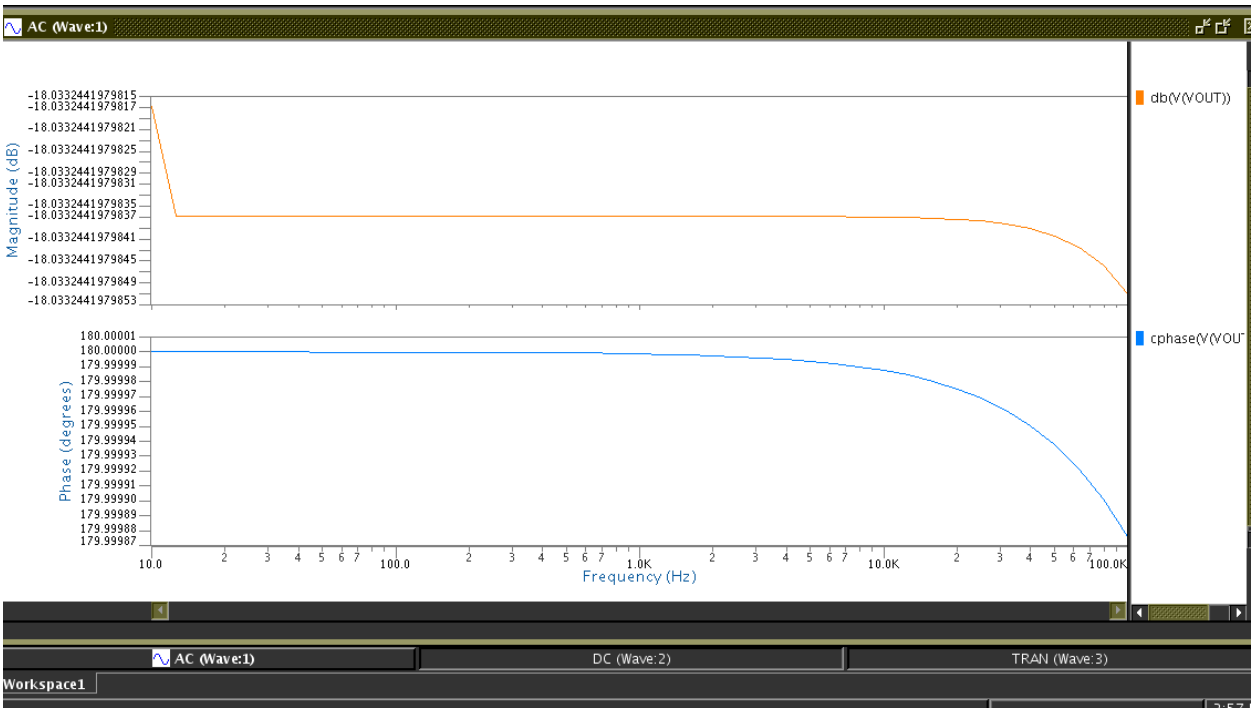
Common Source DC Analysis



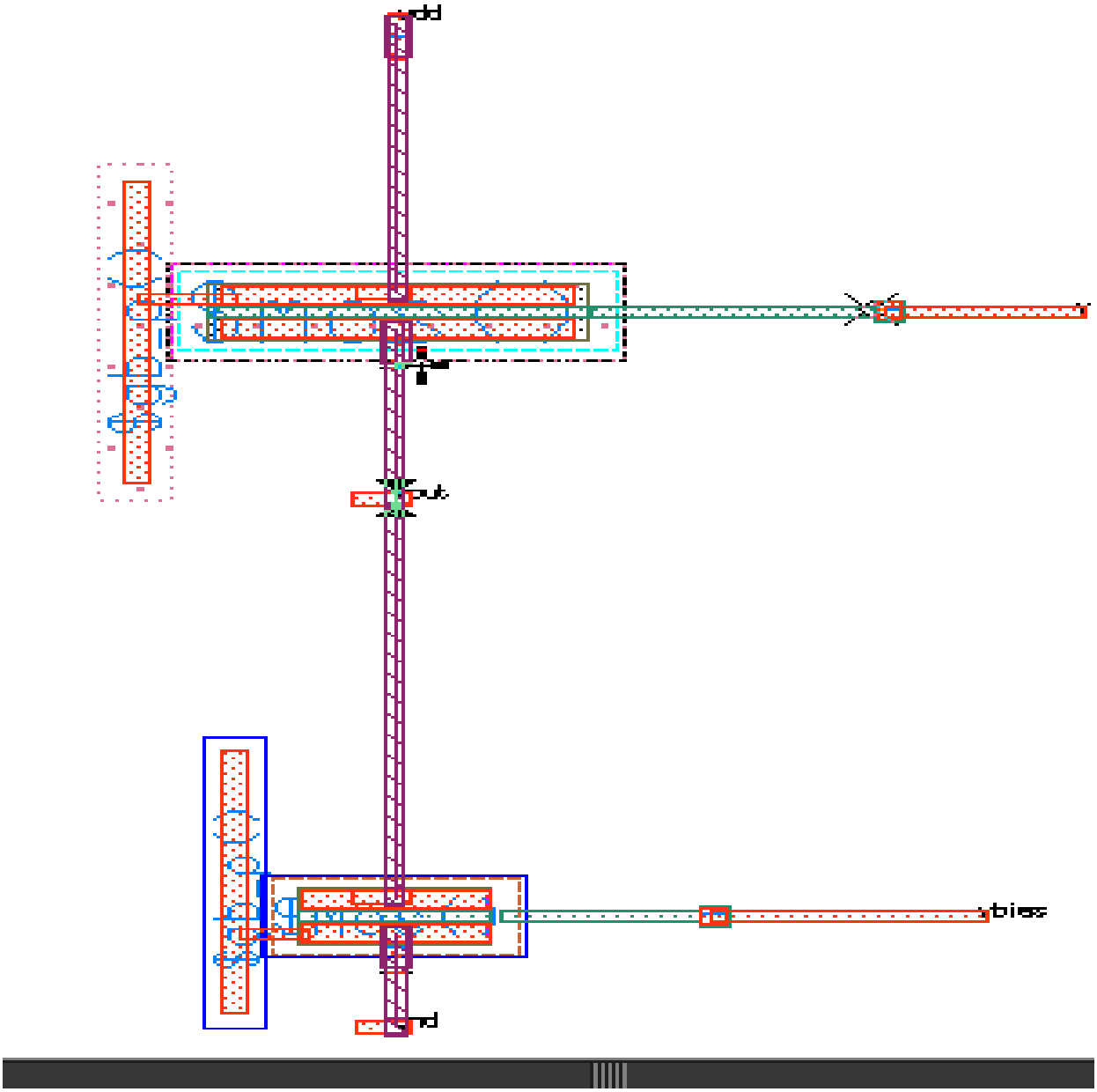
Common Source Transient Analysis



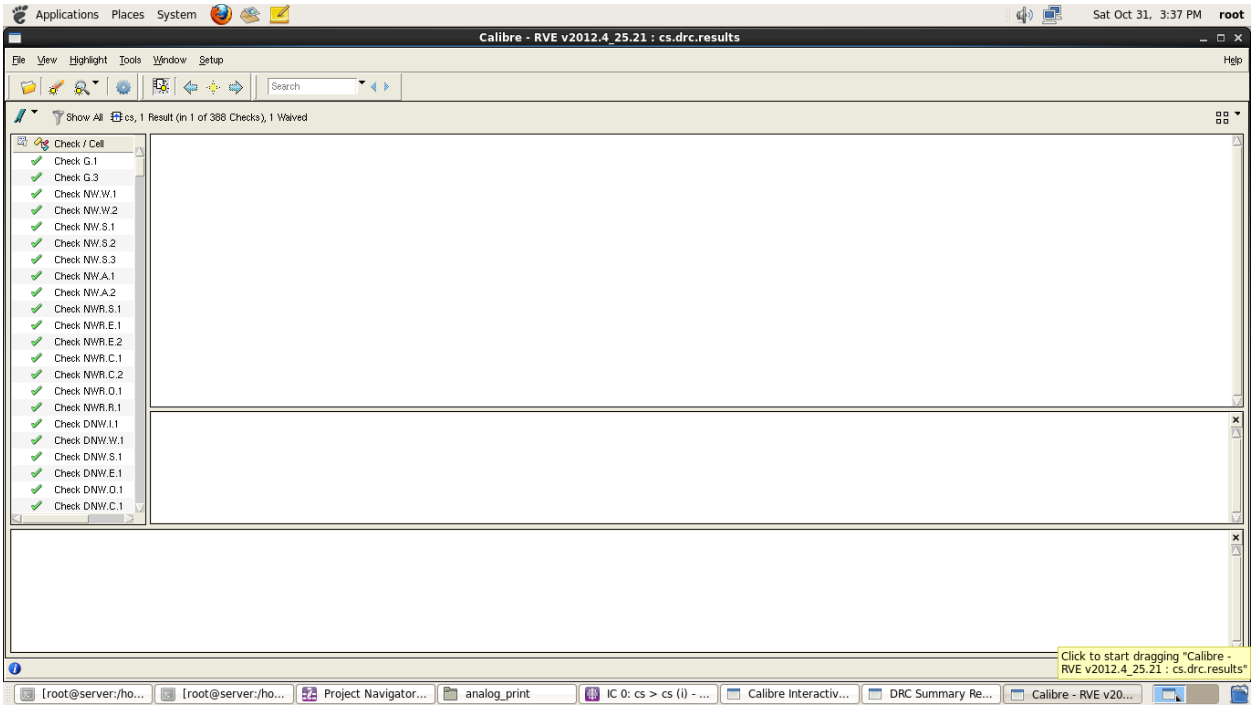
Common Source AC Analysis



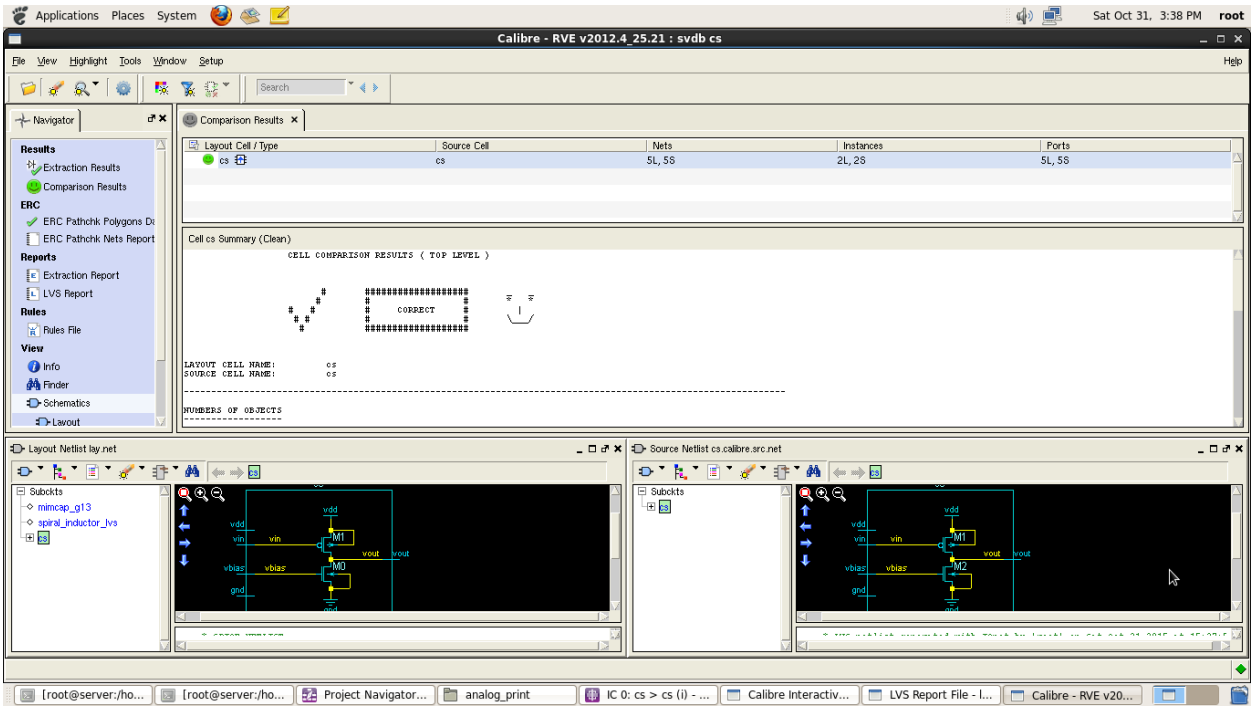
Common Source Layout



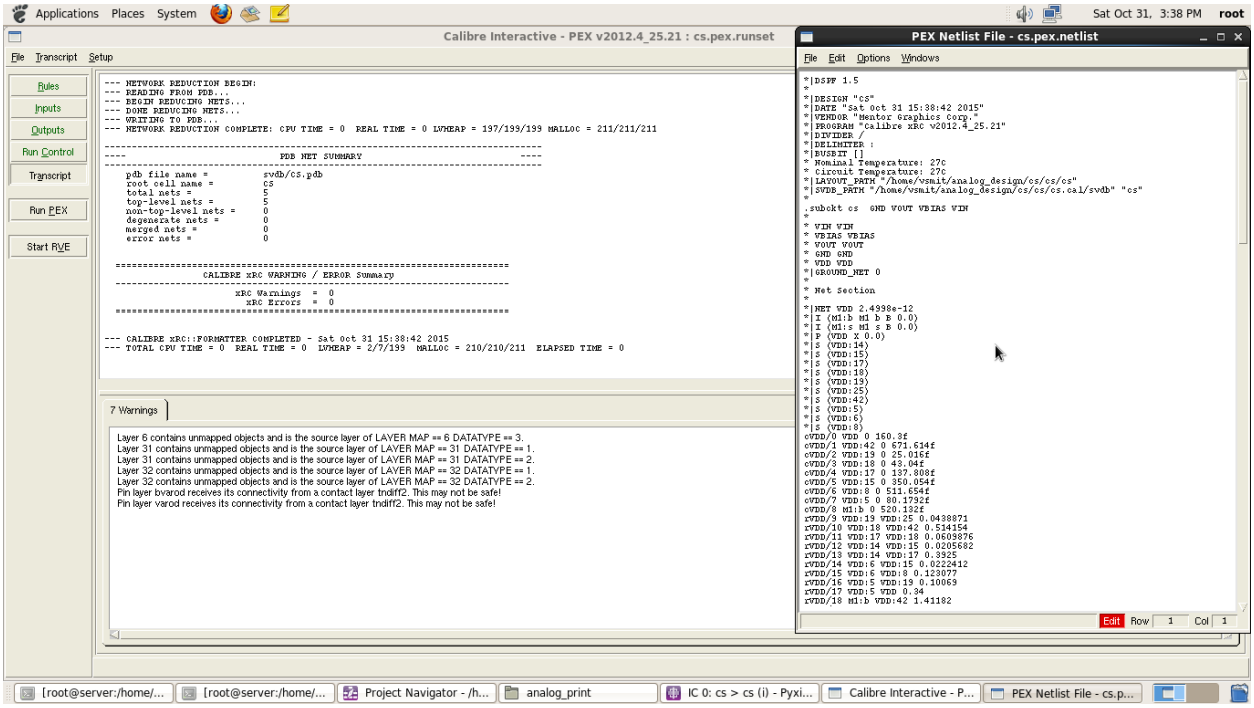
Common Source DRC



Common Source LVS



Common Source PEX



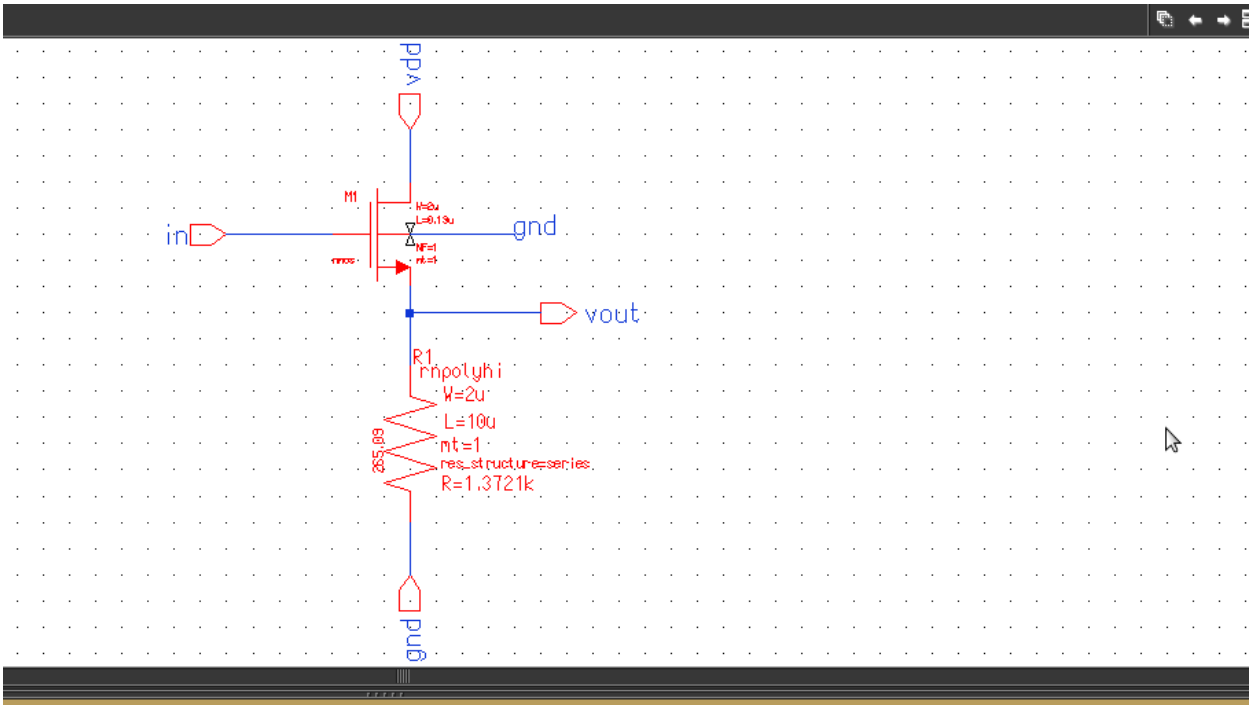
EXPT.NO.2 b

TITLE : To simulate the schematic of the CMOS common drain amplifier, and then to perform the physical verification for the layout of the same.

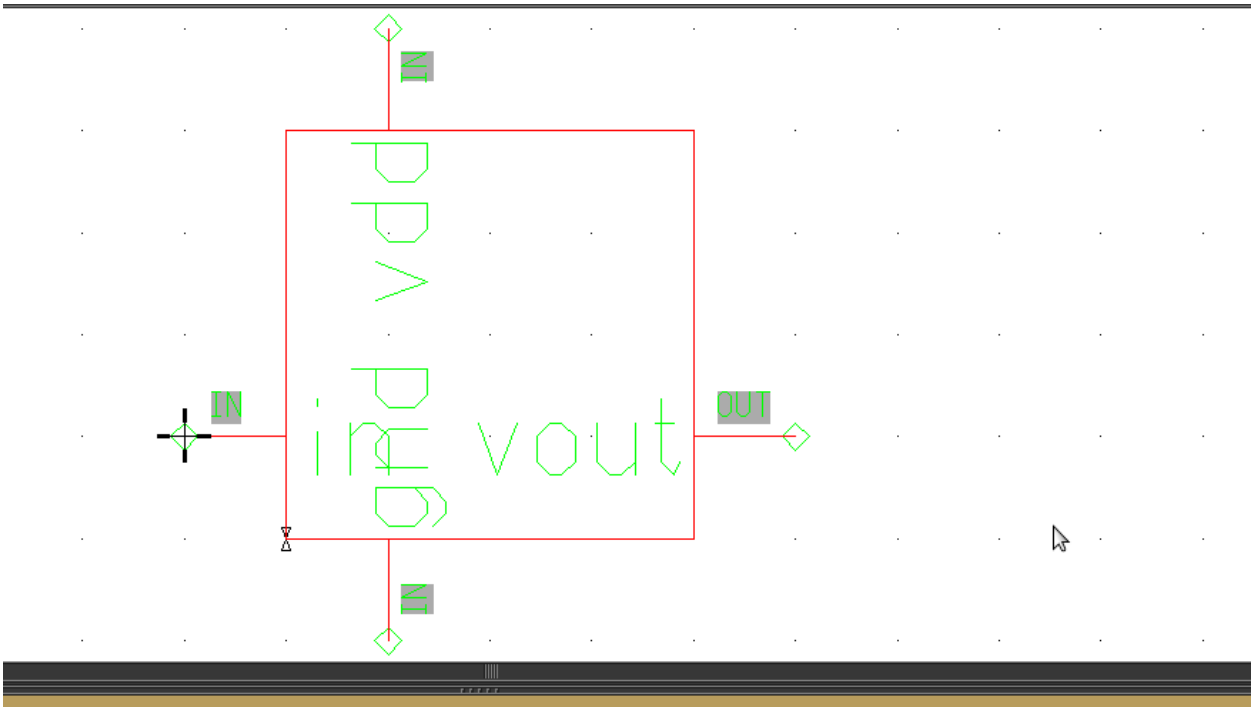
TOOL REQUIRED: Mentorgraphics

THEORY: Common drain amplifier is a source follower or buffer amplifier circuit using a MOSFET. The output is simply equal to the input minus about 2.2V. The advantage of this circuit is that the MOSFET can provide current and power gain; the MOSFET draws no current from the input. It provides low output impedance to any circuit using the output of the follower, meaning that the output will not drop under load. Its output impedance is not as low as that of an emitter follower using a bipolar transistor (as you can verify by connecting a resistor from the output to -15V), but it has the advantage that the input impedance is infinite.The MOSFET is in saturation, so the current across it is determined by the gatesource voltage. Since a current source keeps the current constant, the gate-source voltage is also constant.

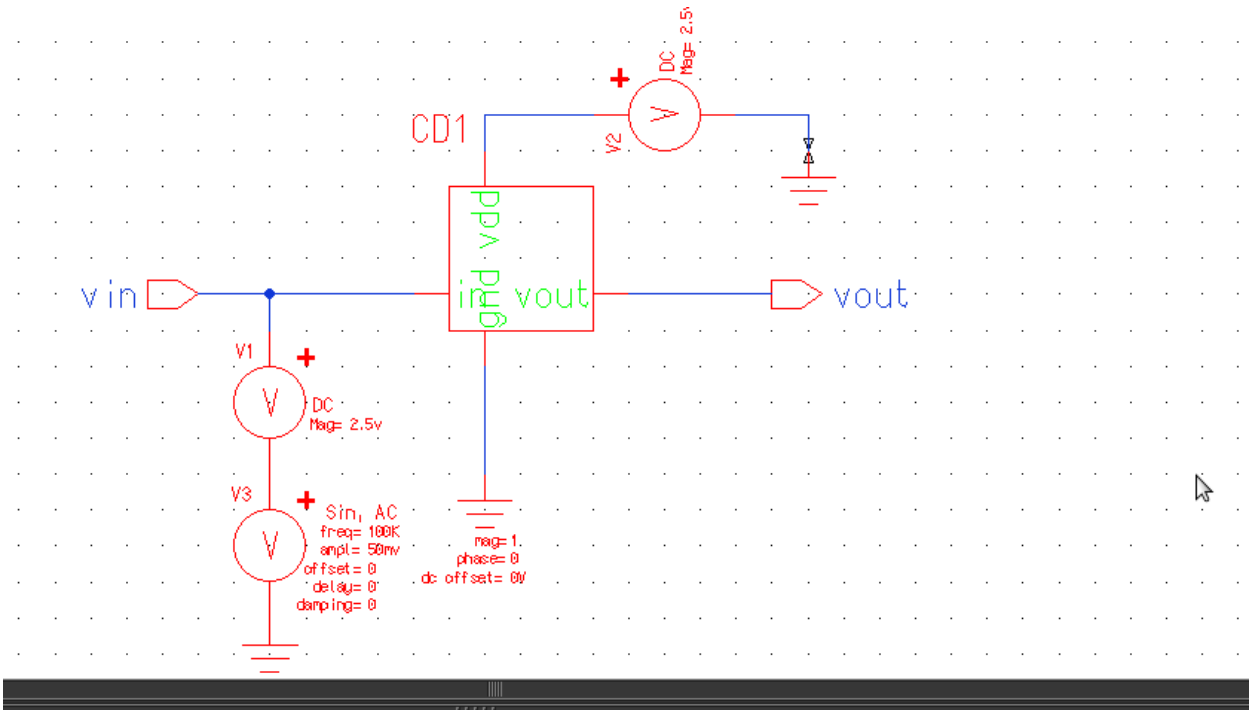
Common Drain Schematic



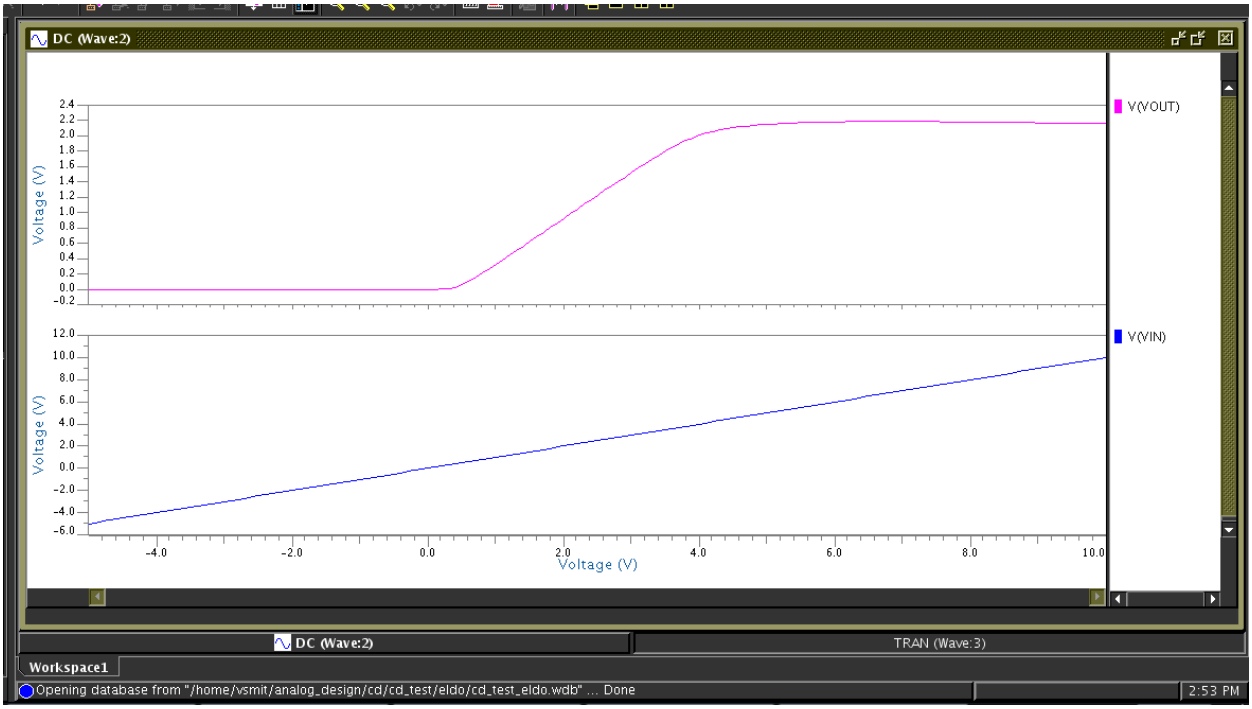
Common Drain Symbol



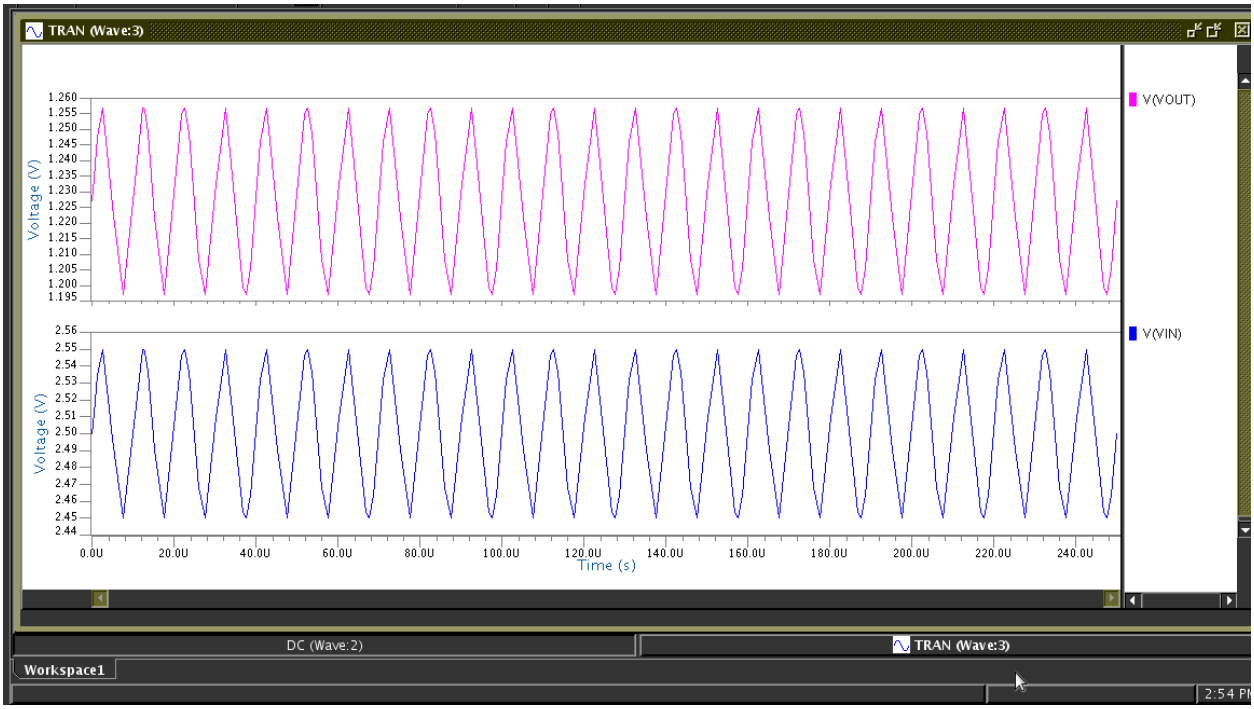
Common Drain Test



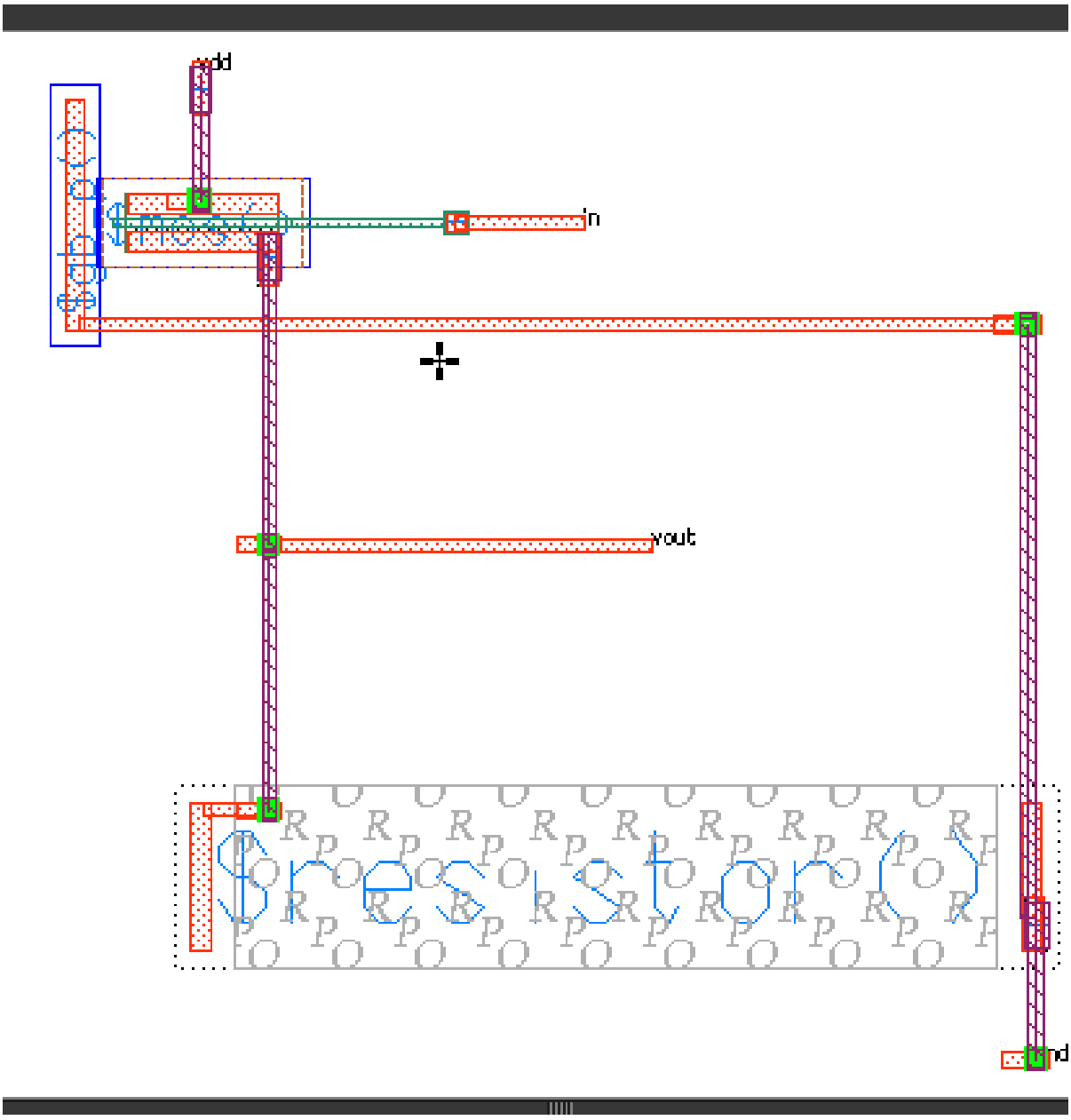
Common Drain DC Analysis



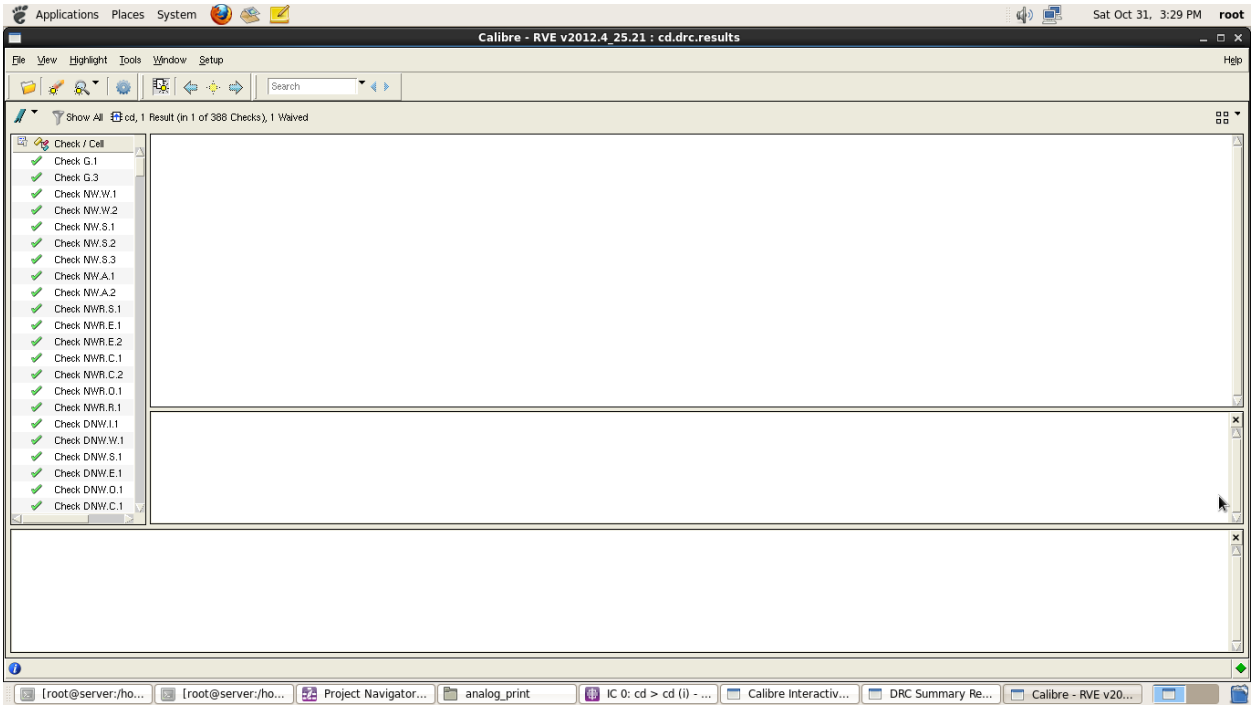
Common Drain Transient Analysis



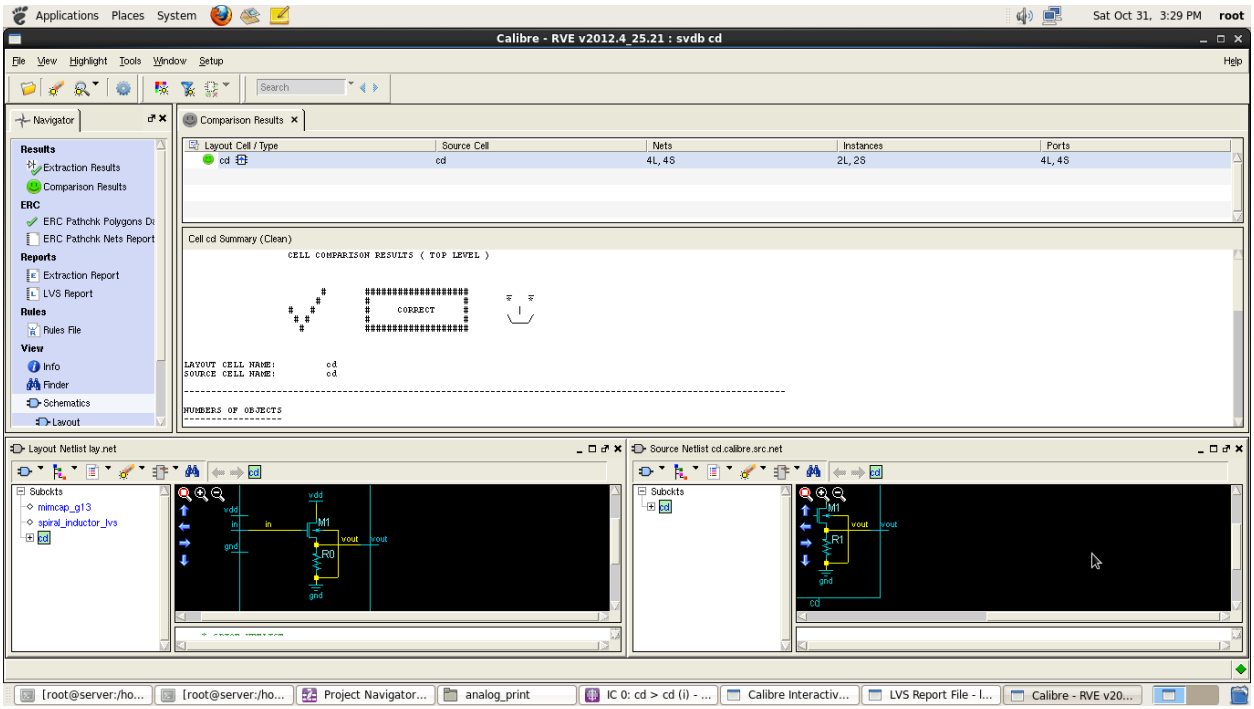
Common Drain Layout



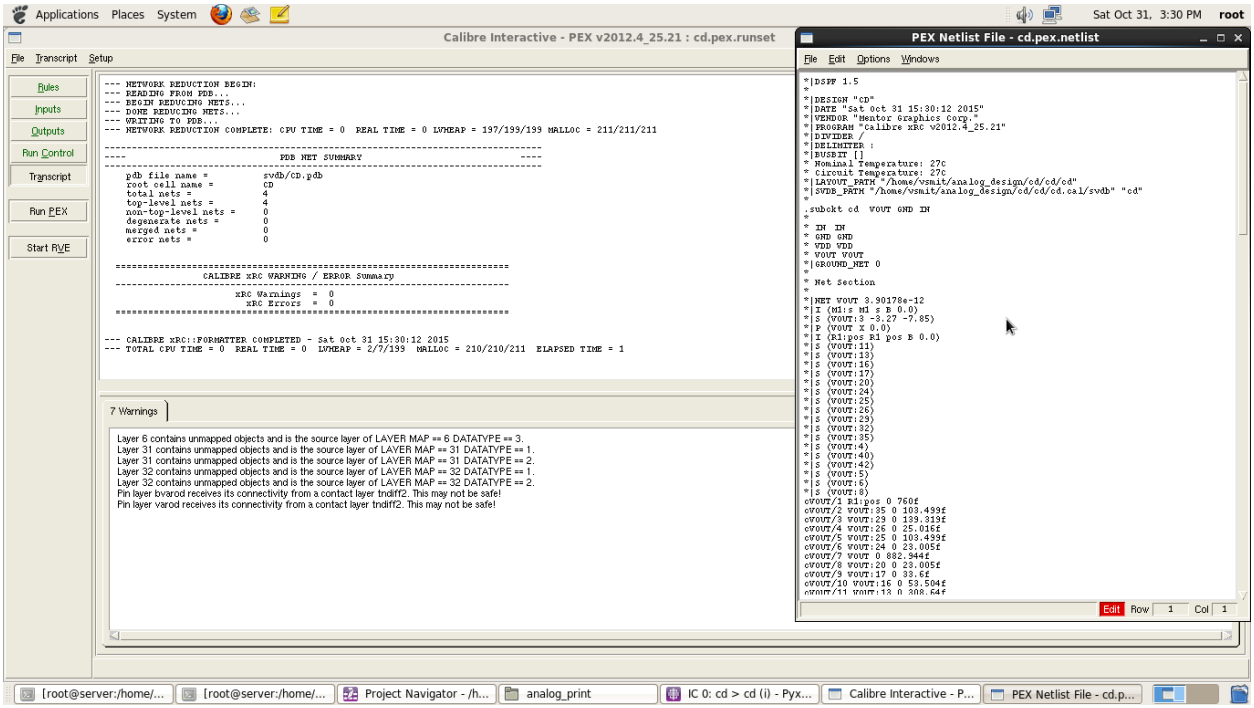
Common Drain DRC



Common Drain LVS



Common Drain PEX



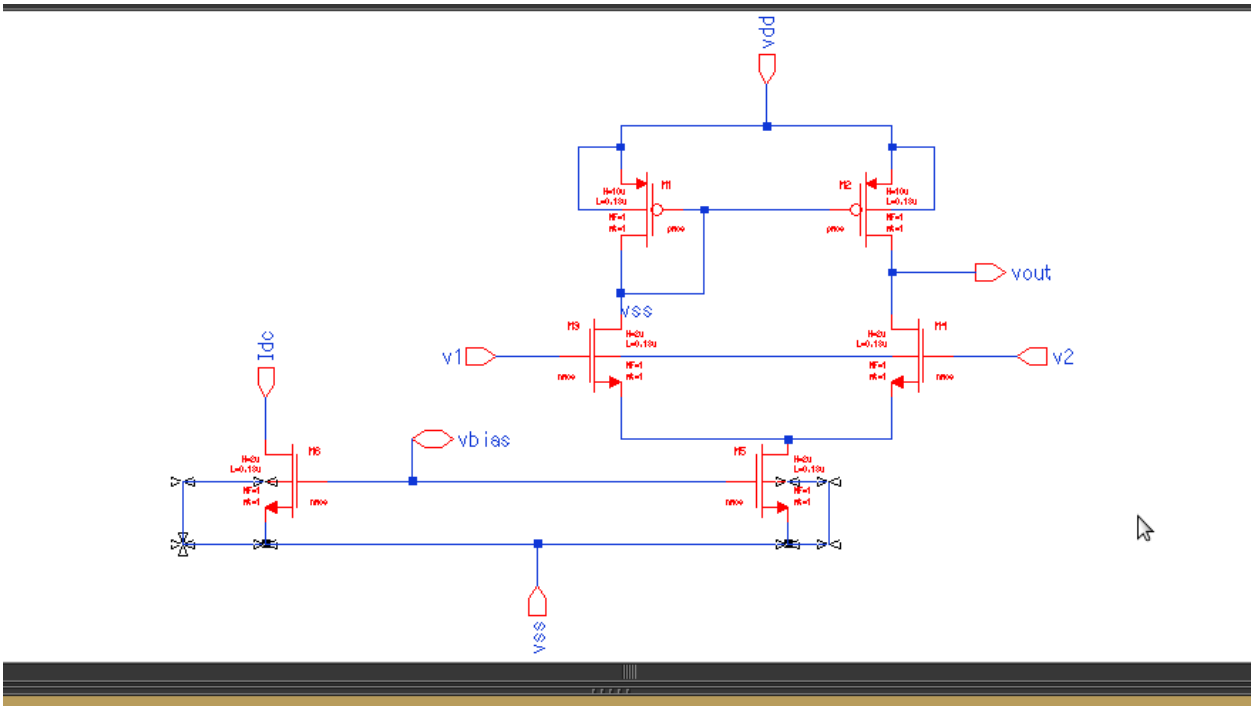
EXPT.NO.2 c

TITLE : To simulate the schematic of the CMOS differential amplifier, and then to perform the physical verification for the layout of the same.

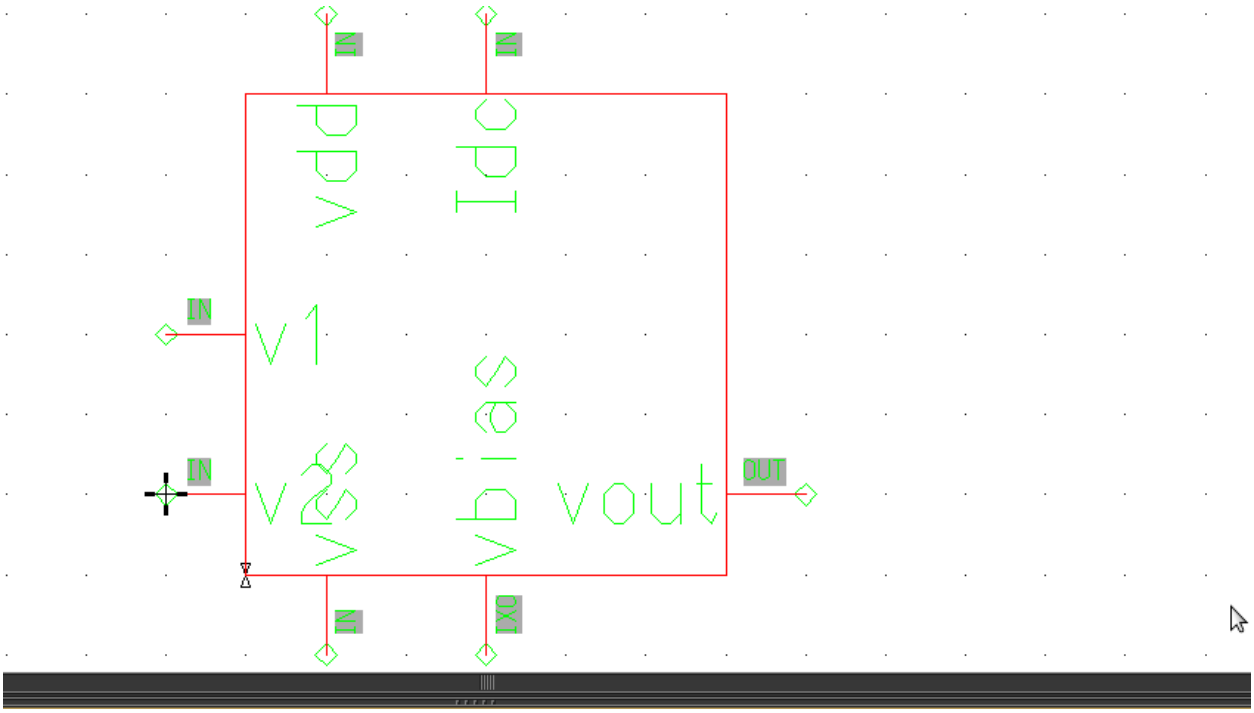
TOOL REQUIRED: Mentorgraphics

THEORY: The differential amplifier is probably the most widely used circuit building block in analog integrated circuits, principally op amps. We had a brief glimpse at one back in Chapter 3 section 3.4.3 when we were discussing input bias current. The differential amplifier can be implemented with BJTs or MOSFETs. A differential amplifier multiplies the voltage difference between two inputs (Vin+ - Vin-) by some constant factor Ad, the differential gain. It may have either one output or a pair of outputs where the signal of interest is the voltage difference between the two outputs. A differential amplifier also tends to reject the part of the input signals that are common to both inputs (Vin+ + Vin-)/2 . This is referred to as the common mode signal.

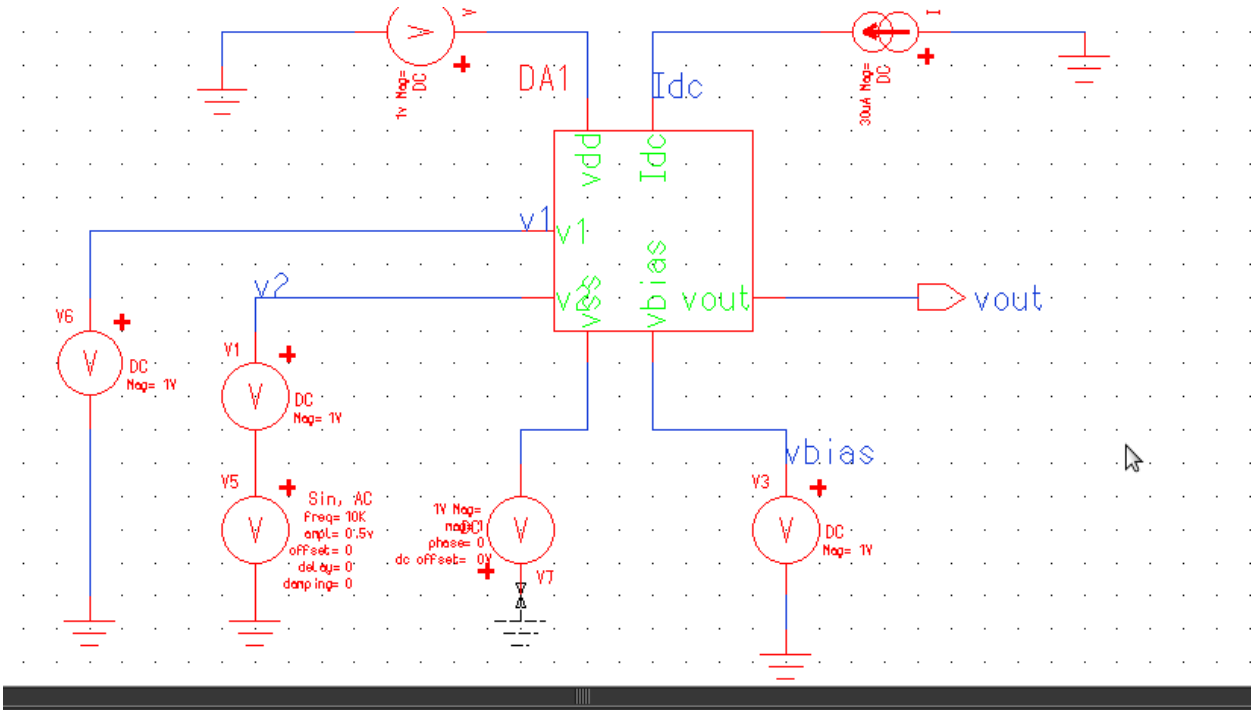
Differential Amplifier Schematic



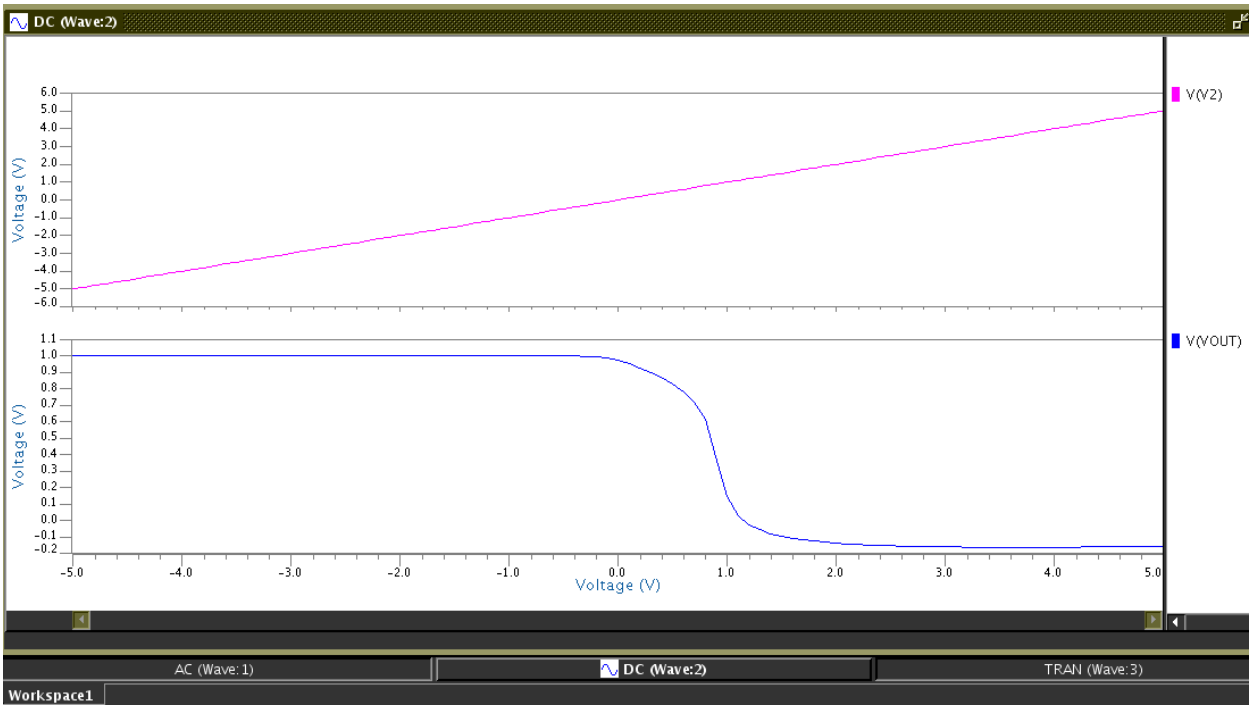
Differential Amplifier Symbol



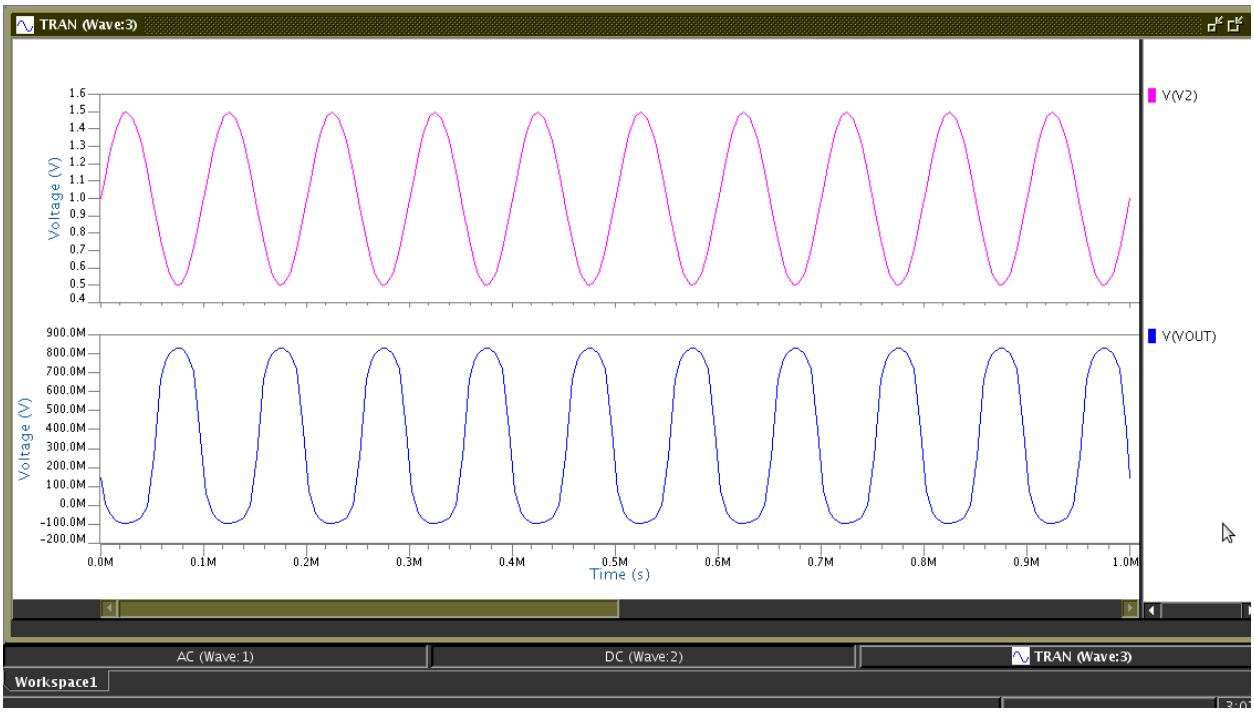
Differential Amplifier Test



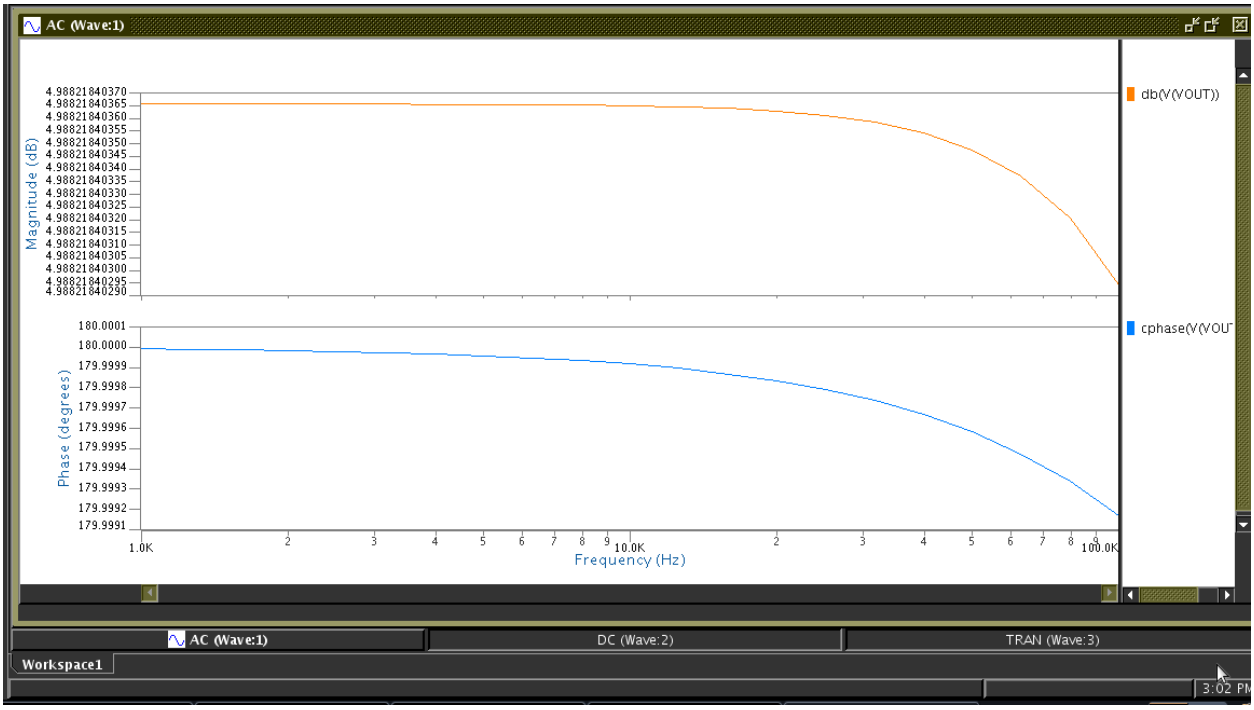
Differential Amplifier DC Analysis



Differential Amplifier Transient Analysis



Differential Amplifier AC Analysis



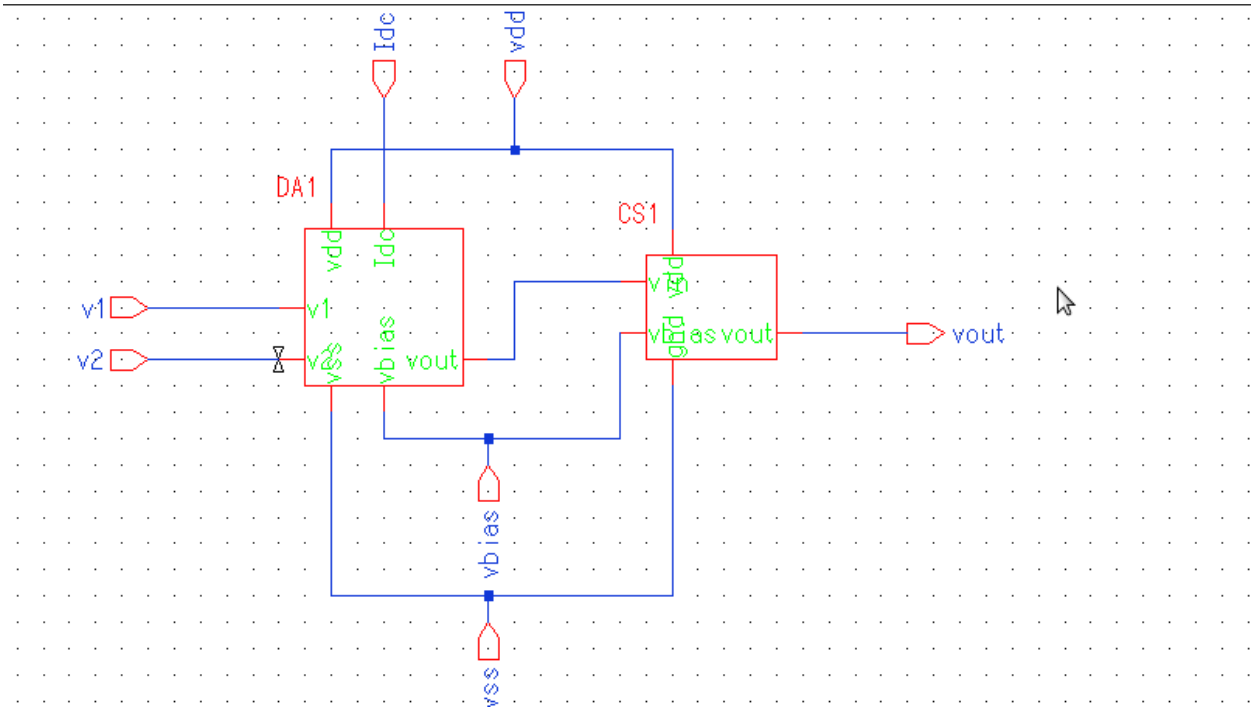
EXPT.NO.3

TITLE : To simulate the schematic of the CMOS op-amp and then to perform the physical verification for the layout of the same.

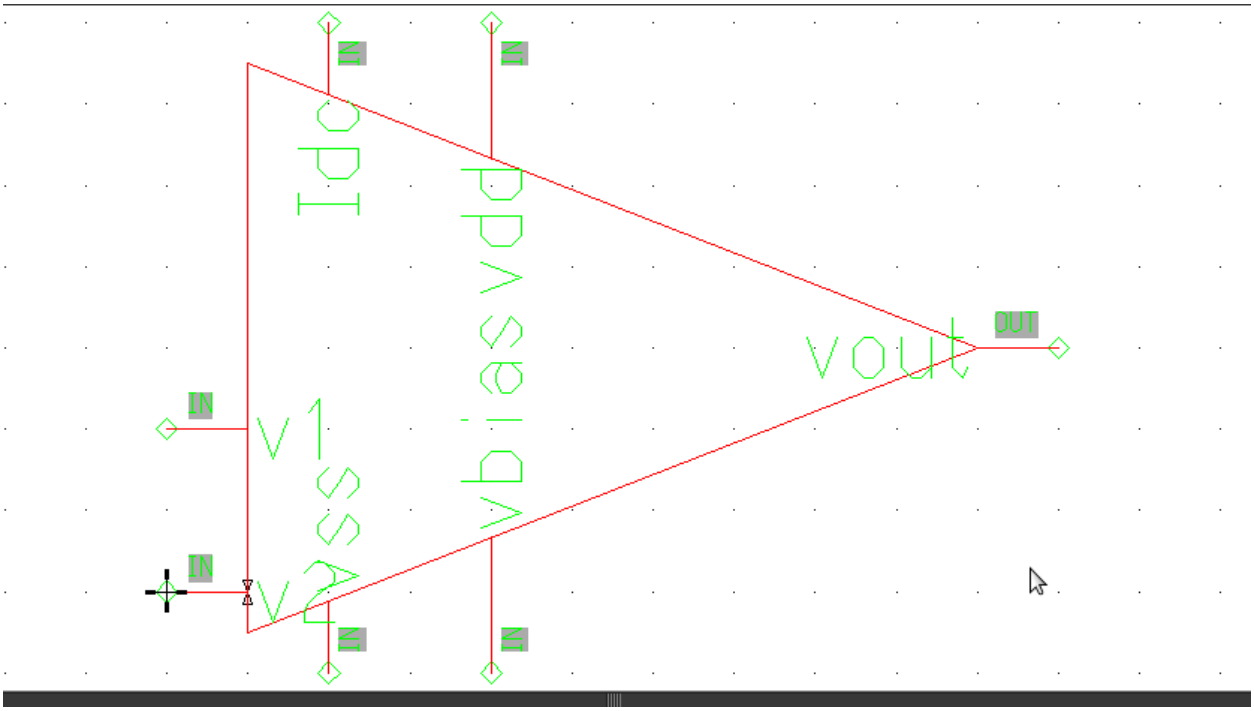
TOOL REQUIRED: Mentorgraphics

THEORY: An operational amplifier (often op-amp or opamp) is a DC-coupled high-gain electronic voltage amplifier with a differential inputand, usually, a single-ended output.[1] In this configuration, an op-amp produces an output potential (relative to circuit ground) that is typically hundreds of thousands of times larger than the potential difference between its input terminals. Operational amplifiers had their origins in analog computers, where they were used to perform mathematical operations in many linear, non-linear and frequency-dependent circuits. The popularity of the op-amp as a building block in analog circuits is due to its versatility. Due tonegative feedback, the characteristics of an op-amp circuit, its gain, input and output impedance, bandwidth etc. are determined by external components and have little dependence on temperature coefficients or manufacturing variations in the op-amp itself. Op-amps are among the most widely used electronic devices today, being used in a vast array of consumer, industrial, and scientific devices. Many standard IC op-amps cost only a few cents in moderate production volume; however some integrated or hybrid operational amplifiers with special performance specifications may cost over $100 US in small quantities. Op-amps may be packaged as components, or used as elements of more complex integrated circuits. The op-amp is one type of differential amplifier. The amplifier's differential inputs consist of a non-inverting input (+) with voltage V+ and an inverting input (–) with voltage V−; ideally the op-amp amplifies only the difference in voltage between the two, which is called the differential input voltage.

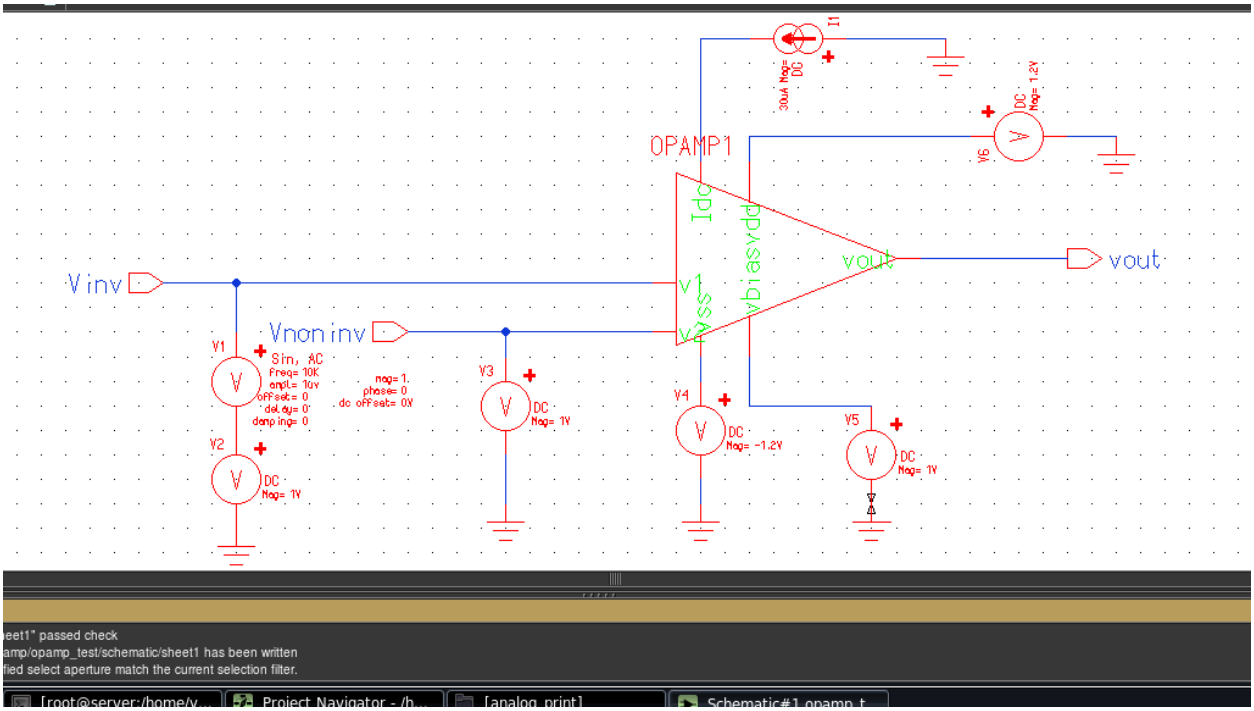
OPAMP Schematic



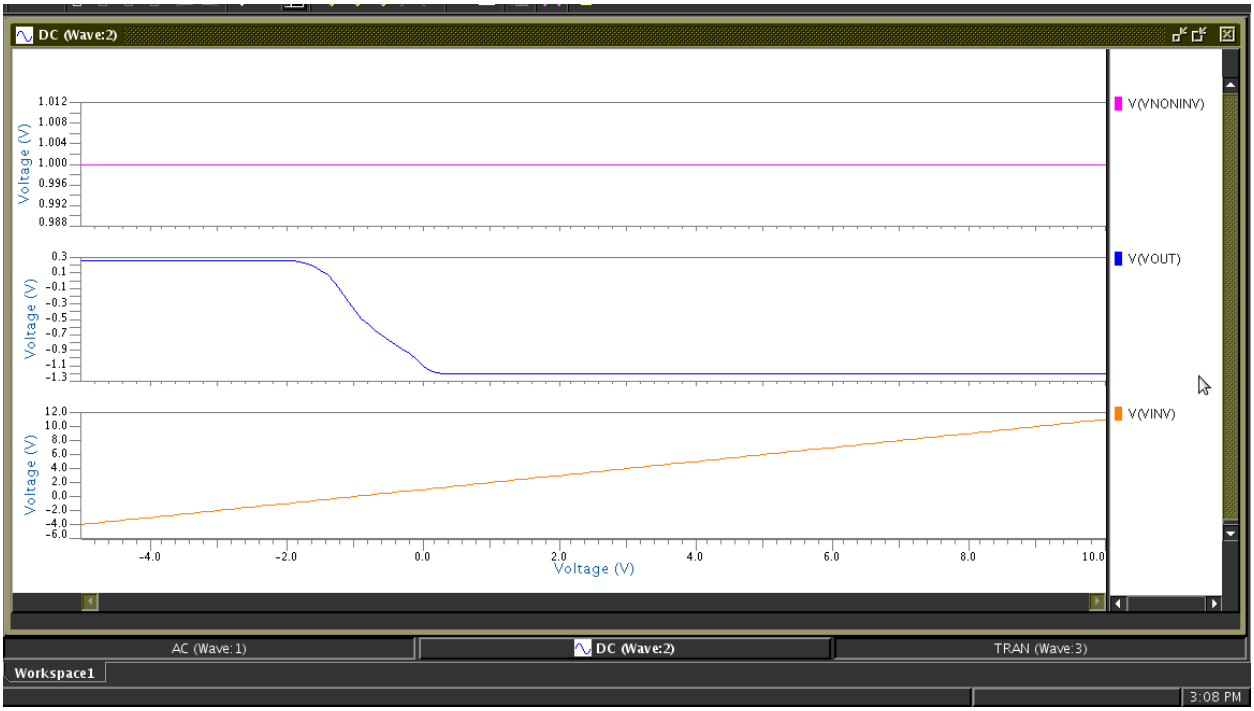
OPAMP Symbol



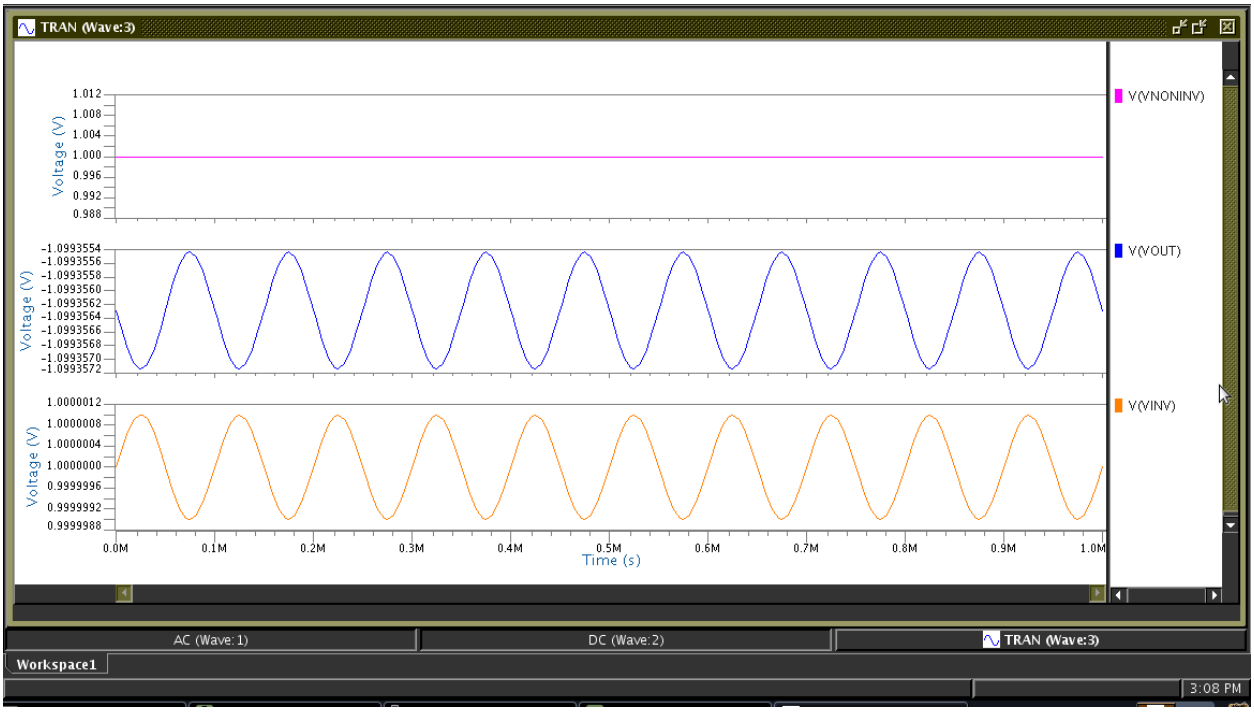
OPAMP Test



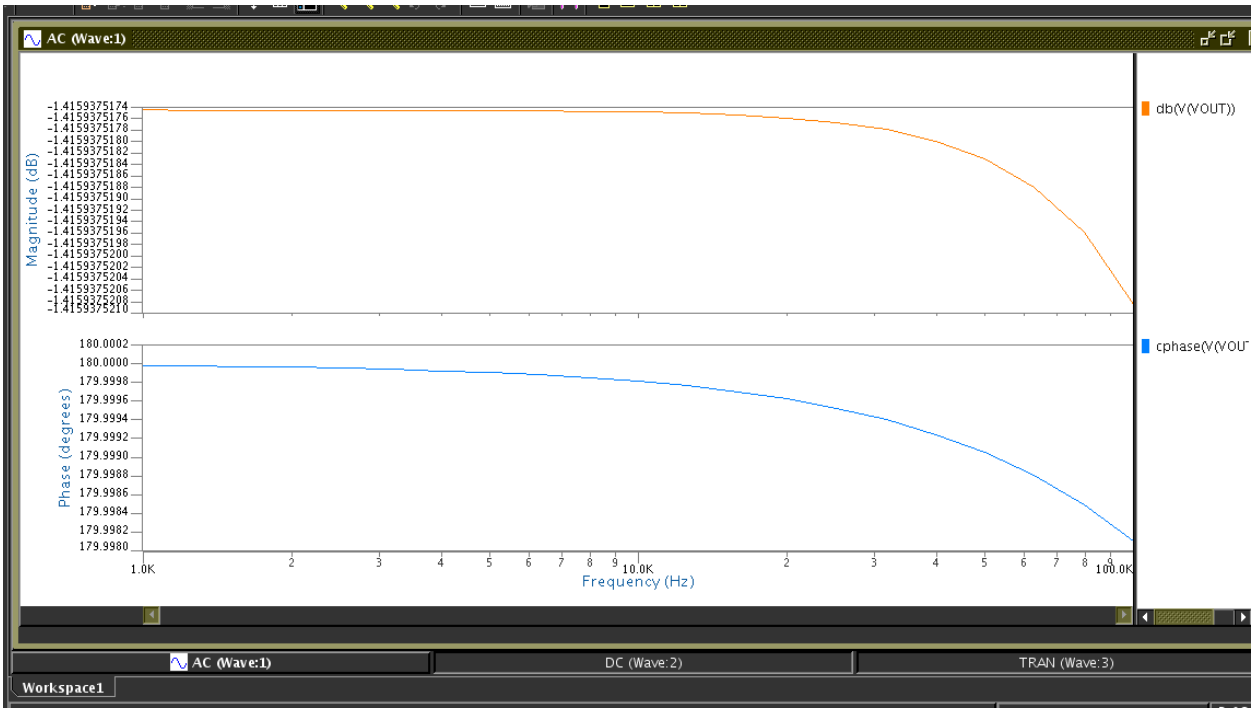
OPAMP DC Analysis



OPAMP Transient Analysis



OPAMP AC Analysis



EXPT.NO.3

TITLE : To simulate the schematic of the CMOS R - 2R ladder and then to perform the physical verification for the layout of the same.

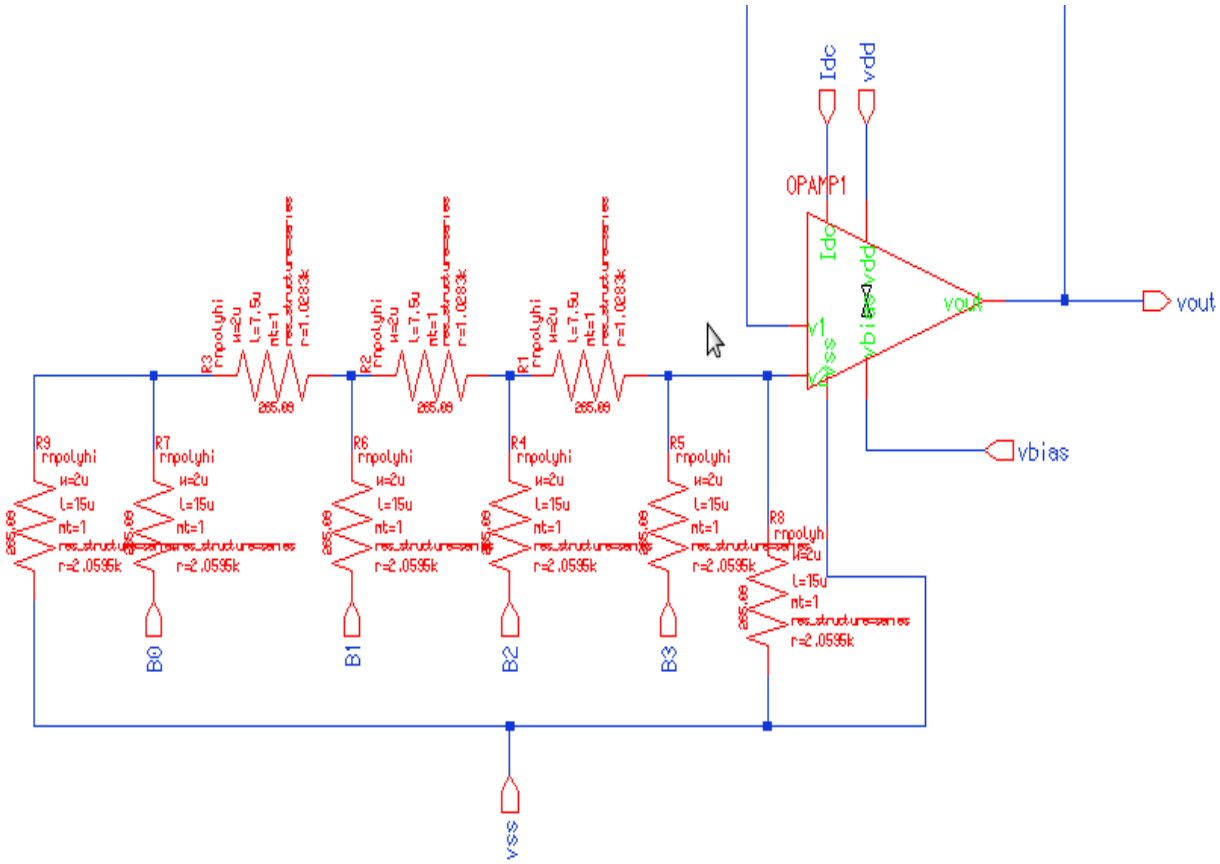
TOOL REQUIRED: Mentorgraphics

THEORY: The R-2R resistor ladder network directly converts a parallel digital symbol/word into an analog voltage. Each digital input (b0, b1, etc.) adds its own weighted contribution to the analog output. This network has some unique and interesting properties.

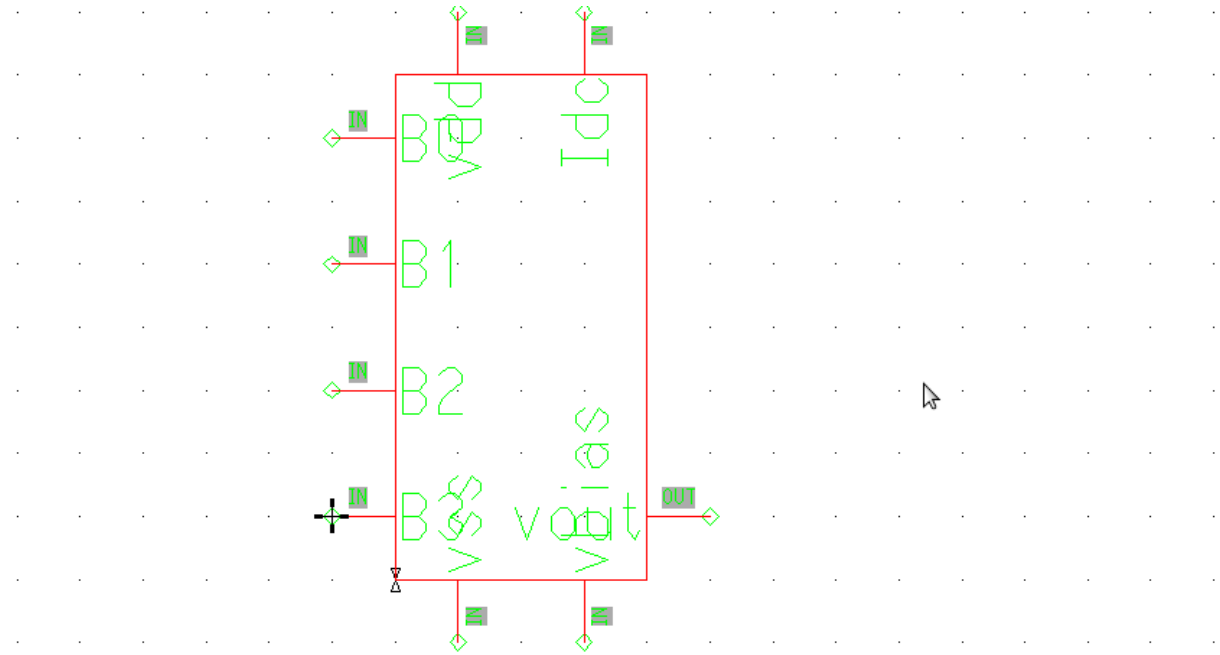
- Easily scalable to any desired number of bits
- Uses only two values of resistors which make for easy and accurate fabrication and integration
- Output impedance is equal to R, regardless of the number of bits, simplifying filtering and further analog signal processing circuit design

Analyzing the R-2R network brings back memories of the seemingly infinite variety of networks that you're asked to solve during your undergraduate electrical engineering studies. The reality though, is that the analysis of this network and how it works is quite simple. By methodical application of Thevenin Equivalent circuits and Superposition, we can easily show how the R-2R circuit works.

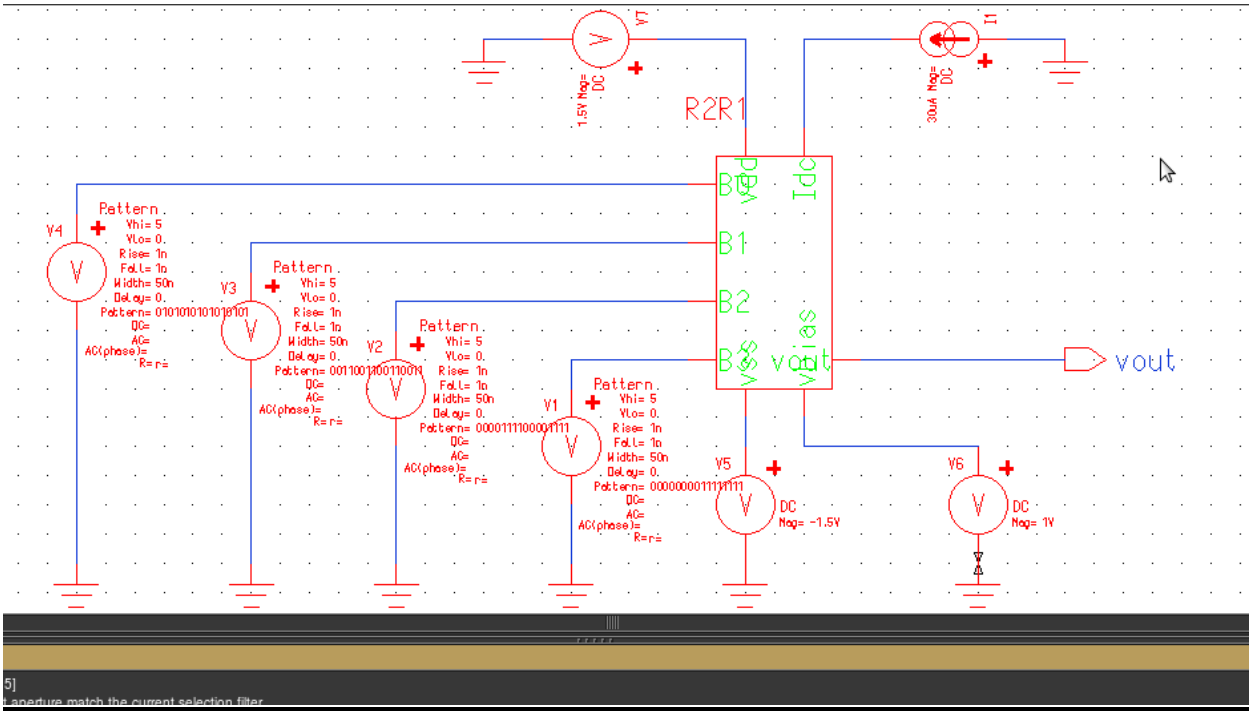
R2R Schematic



R2R Symbol



R2R Test



R2R Transient Analysis