2. Write a Verilog code for D Flip-Flop with synchronous and asynchronous reset and verify its functionality using test bench. Synthesize the design, tabulate the area, power and timing report.

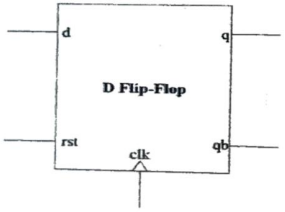
Tools required:

- *Functional Simulation: Incisive Simulator (ncvlog, ncelab, ncsim)*
- *Synthesis: Genus*

D Flip-Flop: *A D (or Delay) Flip Flop is a digital electronic circuit used to delay the change of state of its output signal (Q) until the next rising edge of a clock timing input signal occurs. The D Flip Flop acts as an electronic memory component since the output remains constant unless deliberately changed by altering the state of the D input followed by a rising clock signal.*

a) D Flip-Flop with synchronous reset:

D Flip-Flop with synchronous reset block diagram:



D Flip-Flop with synchronous reset truth table:

rst	clk	d	q	qb
1	↑	X	0	1
0	↑	0	0	1
0	↑	1	1	0
1	↑	X	0	1

Verilog Code for D Flip-Flop with synchronous reset:

```
module d_ff (q, qb, d, clk, rst);
output reg q;
output qb;
```



```
input d, clk, rst;

always @(posedge clk)

begin
if (rst)
q = 0;
else
q = d;
end

assign qb = ~q;

endmodule
```

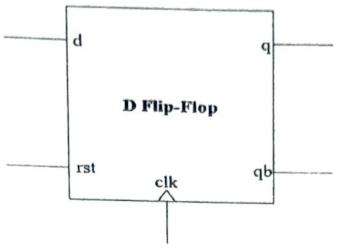
Testbench for D Flip-Flop with synchronous reset module:

```
module d_ff_test;
reg clk, rst, d;
wire q, qb;
d_ff d1(q, qb, d, clk, rst);

initial
begin
$monitor ("time = %0d", $time, "ns", "rst =", rst, "d =", d, "q =", q, "qb =", qb);
#40 $finish;
end

initial
clk = 0;
always
#5 clk = ~clk;
```

```
initial
begin
rst = 1; d = 0;
#10 rst = 0;
#10 d = 1;
#10 rst = 1;
end

endmodule
```

b) D Flip-Flop with asynchronous reset:

D Flip-Flop with asynchronous reset block diagram:



D Flip-Flop with synchronous reset truth table:

rst	clk	d	q	qb
0	X	X	0	1
1	↑	0	0	1
1	↑	1	1	0
0	X	X	0	1

Verilog Code for D Flip-Flop with asynchronous reset:

```
module dff (q, qb, d, clk, rst);
output reg q;
output qb;
input d, clk, rst;

always @(posedge clk, negedge rst)

begin
if (!rst)
q = 0;
else
q = d:
end

assign qb = ~q:

endmodule
```

Testbench for D Flip-Flop with asynchronous reset module:

```
module dff_test;
reg d, rst, clk;
wire q, qb;
dff d1 (q, qb, d, clk, rst);
initial
begin
$monitor ("time=%0d", $time, "ns", "rst =", rst, "d =", d, "q =", q, "qb =", qb);
#40 $finish;
end

initial
begin
d = 0;
clk = 0;
end

always
#5 clk = ~clk;

initial
begin
rst = 0;
#10 rst =1;
#10 d =1;
#10 rst = 0;
end

endmodule
```

4. Write a Verilog code for SR Flip-Flop with synchronous and asynchronous reset and verify its functionality using test bench. Synthesize the design, tabulate the area, power and timing report.

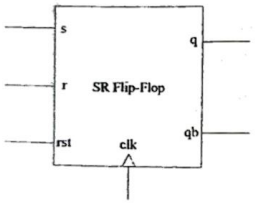
Tools required:

- *Functional Simulation: Incisive Simulator (ncvlog, ncelab, ncsim)*
- *Synthesis: Genus*

SR Flip-Flop: *The SR flip flop is a 1-bit memory bistable device having two inputs, i.e., SET and RESET. The SET input 's' set the device or produce the output 1, and the RESET input 'r' reset the device or produce the output 0. The SET and RESET inputs are labelled as s and r, respectively. The SR flip flop stands for "Set-Reset" flip flop. The reset input is used to get back the flip flop to its original state from the current state with an output 'q'. This output depends on the set and reset conditions, which is either at the logic level "0" or "1".*

a) SR Flip-Flop with synchronous reset:

SR Flip-Flop with synchronous reset block diagram:



SR Flip-Flop with synchronous reset truth table:

rst	clk	S	r	q	qb
1	↑	X	X	0	1
0	↑	0	0	q	qb
0	↑	0	1	0	1
0	↑	1	0	1	0
0	↑	1	1	X	X

Verilog Code for SR Flip-Flop with synchronous reset:

```
module sr_ff (q, qb, s, r, clk, rst);
output reg q;
```



```
output qb;
input s, r, clk, rst;

always @(posedge clk)

begin
if (rst)
q = 0;
else
if (s == 0 && r == 0)
q = q;
else
if (s == 0 && r == 1)
q = 0;
else
if (s == 1 && r == 0)
q = 1;
else
if (s == 1 && r == 1)
q = 1'bx;
end
assign qb = ~q;

endmodule
```

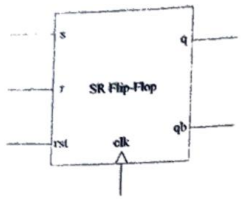
Testbench for SR Flip-Flop with synchronous reset module:

```
module sr_ff_test;
reg s, r, clk, rst;
wire q, qb;

sr_ff s1(q, qb, s, r, clk, rst);

initial
```

```
begin
$monitor ("time = %0d", $time, "ns", "s =", s, "r =", r, "rst =", rst, "q =", q, "qb =", qb);
#80 $finish;
end

initial
clk = 1'b0;

always
#5 clk = ~clk;

initial
begin
rst = 1; s = 1; r = 0;
#10; rst = 0;
#10; s = 0; r = 0;
#10; s = 0; r = 1;
#10; s = 1; r = 0;
#10; s = 1; r = 1;
#10; s = 1; r = 0;
#10; rst = 1;
end
endmodule
```

b) SR Flip-Flop with asynchronous reset:

SR Flip-Flop with asynchronous reset block diagram:



SR Flip-Flop with asynchronous reset truth table:

rst	clk	S	r	q	qb
0	X	X	X	0	1
1	↑	0	0	q	qb
1	↑	0	1	0	1
1	↑	1	0	1	0
1	↑	1	1	X	X

Verilog Code for SR Flip-Flop with asynchronous reset:

```
module srff (q, qb, s, r, clk, rst);
output reg q;
output qb;
input s, r, clk, rst;

always @(posedge clk, negedge rst)

begin
if (!rst)
q = 0;
else
if (s == 0 && r == 0)
q = q;
else
if (s == 0 && r == 1)
q = 0;
```



```
else
if (s == 1 && r == 0)
q = 1;
else
if (s == 1 && r == 1)
q = 1'bx;
end

assign qb = ~q;

endmodule
```

Testbench for SR Flip-Flop with asynchronous reset module:

```
module srff_test;
reg s, r, clk, rst;
wire q, qb;

srff s1(q, qb, s, r, clk, rst);

initial
begin
$monitor ("time = %0d", $time,"ns", "s =", s, "r =", r, "rst =", rst, "q =", q, "qb =", qb);
#80 $finish;
end

initial
clk = 1'b0;

always
#5 clk = ~clk;
initial
begin
rst = 0; s = 1; r = 0;
```

```
#10; rst = 1;
#10; s = 0; r = 0;
#10; s = 0; r = 1;
#10; s = 1; r = 0;
#10; s = 1; r = 1;
#10; s = 1; r = 0;
#10; rst = 0;
end

endmodule
```

6. Write a Verilog code for JK Flip-Flop with synchronous and asynchronous reset and verify its functionality using test bench. Synthesize the design, tabulate the area, power and timing report.

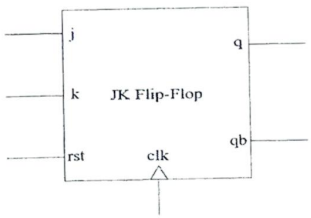
Tools required:

- *Functional Simulation: Incisive Simulator (ncvlog, ncelab, ncsim)*
- *Synthesis: Genus*

JK Flip-Flop: *The JK flip-flop is the most versatile of the basic flip flops. A JK flip-flop is used in clocked* sequential *logic circuits to store one bit of data. It is almost identical in function to an SR flip flop. The only difference is eliminating the undefined state where both S and R are 1. Due to this additional clocked input, a JK flip-flop has four possible input combinations, such as "logic 1", "logic 0", "no change" and "toggle".*

a) JK Flip-Flop with synchronous reset:

JK Flip-Flop with synchronous reset block diagram:



JK Flip-Flop with synchronous reset truth table:

rst	clk	J	k	q	qb
1	↑	X	X	0	1
0	↑	0	0	q	qb
0	↑	0	1	0	1
0	↑	1	0	1	0
0	↑	1	1	~q	~qb

Verilog Code for JK Flip-Flop with synchronous reset:

```
module jk_ff (q, qb, j, k, clk, rst);
output reg q;
output qb;
input j, k, clk, rst;
```

```
always @(posedge clk)

begin
if (rst)
q = 0;
else
if (j == 0 && k == 0)
q = q;
else
if (j == 0 && k == 1)
q = 0;
else
if (j == 1 && k == 0)
q = 1;
else
if (j == 1 && k == 1)
q = ~q;
end

assign qb = ~q;

endmodule
```

Testbench for JK Flip-Flop with synchronous reset module:

```
module jk_ff_test;
reg j, k, clk, rst;
wire q, qb;

jk_ff jl(q, qb, j, k, clk, rst);

initial
begin
$monitor ("time = %0d", $time, "ns", "j =", j, "k =", k, "rst =", rst, "q =", q, "qb =", qb);
```



```
#80 $finish;
end

initial
clk = 1'b0;

always
#5 clk = ~clk;

initial
begin
rst = 1; j = 1; k = 0;
#10; rst = 0;
#10; j = 0; k = 0;
#10; j = 0; k = 1;
#10; j = 1; k = 0;
#10; j = 1; k = 1;
#10; j = 1; k = 0;
#10; rst = 1;
end
endmodule
```

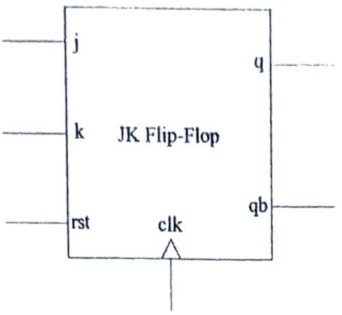
Result:

Non-GUI Output:

time	J	K	rst	q	qb
0	1	0	1	x	x
5	1	0	1	0	1
10	1	0	0	0	1
15	1	0	0	1	0
20	0	0	0	1	0
25	0	0	0	1	0
30	0	1	0	1	0
35	1	1	0	1	0

b) JK Flip-Flop with asynchronous reset:

JK Flip-Flop with asynchronous reset block diagram:



JK Flip-Flop with asynchronous reset truth table:

rst	clk	J	k	q	qb
0	X	X	X	0	1
1	↑	0	0	q	qb
1	↑	0	1	0	1
1	↑	1	0	1	0
1	↑	1	1	~q	~qb

Verilog Code for JK Flip-Flop with asynchronous reset:

```
module jkff (q, qb, j, k, clk, rst);
output reg q;
output qb;
input j, k, clk, rst;

always @(posedge clk, negedge rst)

begin
if (!rst)
q = 0;
else
if (j == 0 && k == 0)
q = q;
else
if (j == 0 && k == 1)
q = 0;
```

```
else
if (j == 1 && k == 0)
q = 1;
else
if (j == 1 && k == 1)
q = ~q;
end

assign qb = ~q;

endmodule
```

Testbench for JK Flip-Flop with asynchronous reset module:

```
module jkff_test;
reg j, k, clk, rst;
wire q, qb;

jkff j1(q, qb, j, k, clk, rst);

initial
begin
$monitor ("time = %0d", $time, "ns", "j =", j, "k =", k, "rst =", rst, "q =", q, "qb =", qb);
#80 $finish;
end

initial
clk = 1'b0;

always
#5 clk = ~clk;
initial
begin
rst = 0; j = 1; k = 0;
```



```
#10; rst = 1;
#10; j = 0; k = 0;
#10; j = 0; k = 1;
#10; j = 1; k = 0;
#10; j = 1; k = 1;
#10; j = 1; k = 0;
#10; rst = 0;

end

endmodule
```

Result:

Non-GUI Output:

time	j	k	rst	q	qb
0	1	0	0	0	1
10	1	0	1	1	0
20	0	0	1	1	0
30	0	1	1	0	1
40	1	0	1	1	0
50	1	1	1	0	1
55	1	1	1	1	0
60	1	0	1	1	0
70	1	0	0	0	1

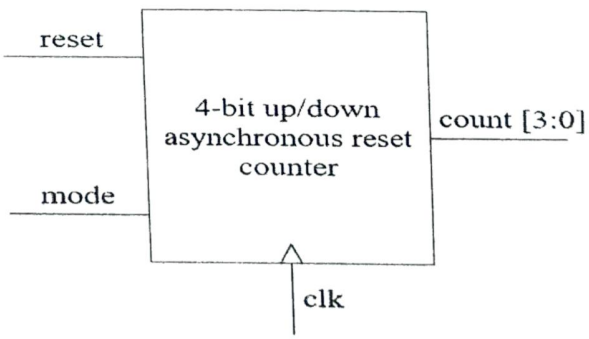
7. **Write verilog code for 4-bit up/down asynchronous reset counter and carry out the following:**
   a. **Verify the functionality using testbench**
   b. **Synthesize the design by setting area and timing constraints. Obtain the gate level netlist, find the critical path and maximum frequency of operation. Record the area requirement in terms of number of cells required and properties of each cell in terms of driving strength, power and area requirements.**
   c. **Perform the above for 32-bit up/down counter and identify the critical path, delay of the critical path, and maximum frequency of operation, total number of cells required and total area.**

Tools required:

- *Functional Simulation: Incisive Simulator (ncvlog, ncelab, ncsim)*
- *Synthesis: Genus*

4-bit up/down: *An up/down counter is a digital counter which can be set to count either from 0 to maximum value or maximum value to 0. The direction of the count (mode) is selected using a single bit input. The up/down counter has 3 inputs - clk, reset and a up or down mode input. The output is count which is 4 bit in size. When Up mode is selected, counter counts from 0 to 15 and then again from 0 to 15. When Down mode is selected, counter counts from 15 to 0 and then again from 15 to 0. Changing mode doesn't reset the Count value to zero. You have to apply high value to reset, to reset the counter output.*

4-bit up/down asynchronous reset counter block diagram:



4-bit up/down asynchronous reset counter truth table:

clk	reset	mode	Count
X	1	X	0
↑	0	0	0
↑	0	0	1
↑	0	0	2
↑	0	0	3
↑	0	0	4
↑	0	1	3
↑	0	1	2
↑	0	1	1
↑	0	1	0
↑	0	1	F
↑	0	1	E
↑	0	1	D
↑	1	X	0

a) 4-bit up/down asynchronous reset counter

Verilog code for 4-bit up/down asynchronous reset counter:

```
module cnt_updown (count, mode, clk, reset);
output reg [3:0] count;
input mode;
input clk, reset;

always @ (posedge clk, posedge reset)

if (reset)
count = 4'b0;
else
if (mode)
count = count + 1;
else
count = count - 1;

endmodule
```


Testbench for 4-bit up/down asynchronous reset counter:

```
module tb_cnt_updown;
reg mode;
reg clk, reset;
wire [3:0] count;

cnt_updown M1(count, mode, clk, reset);

initial
begin
$monitor ("time = %0d", $time, "ns", "reset = 0x%0h", reset, " mode = 0x%0h", mode, " count
= 0x%0h", count);
#320 $finish;
end

always
#5 clk = ~clk;

initial
begin
mode = 1; clk=0; reset=1;
#10; reset = 0;
#100; mode = 0;
#50; reset = 1;
#30; reset = 0;
#100; mode = 1;
#10; reset = 1;
end

endmodule
```

Creating an SDC File:

- In terminal type "gedit counter_top.sdc" to create an SDC file.

```
create_clock -name clk -period 2 -waveform {0 1} [get_ports "clk"]

set_input_delay -max 0.8 -clock clk [all_inputs]
set_output_delay -max 0.8 -clock clk [all_outputs]

set_input_transition 0.2 [all_inputs]
set_max_capacitance 30 [get_ports]

set_clock_transition -rise 0.1 [get_clocks "clk"]
set_clock_transition -fall 0.1 [get_clocks "clk"]

set_clock_uncertainty 0.01 [get_ports "clk"]

set_input_transition 0.12 [all_inputs]
set_load 0.15 [all_outputs]
set_max_fanout 30.00 [current_design]
```

Performing Synthesize:

- The following are commands to perform synthesize (4-bit and 32-bit up/down asynchronous reset counter)

```
genus -gui
read_lib /home/install/cad/slow.lib
read_hdl cnt_updown.v
elaborate
read_sdc counter_top.sdc

set_db syn_generic_effort medium
set_db syn_map_effort medium
set_db syn_opt_effort medium
```



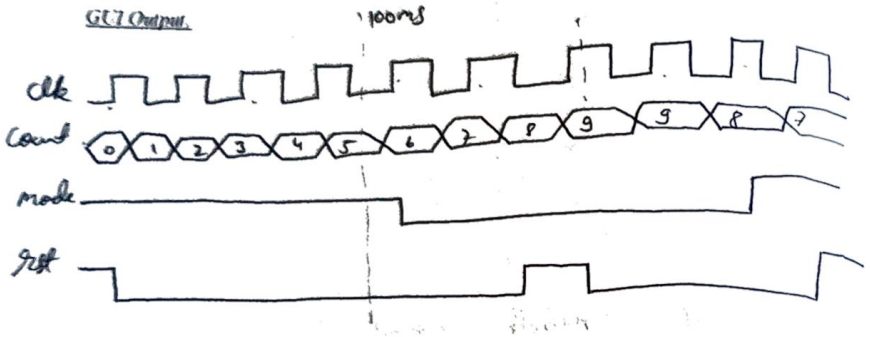
```
syn_generic
syn_map
syn_opt

write_hdl > cnt_updown_netlist.v
write_sdc > cnt_updown.sdc

report_power
report_gates
report_timing
report_area
report_qor -levels_of_logic -power -exclude_constant_nets
```

Result:

Non-GUI Output:

time	rst	mode	count
0	1	1	0
10	0	1	0
25	0	1	1
35	0	1	2
45	0	1	3
55	0	1	4
65	0	1	5
75	0	1	6
85	0	1	7
95	0	1	8
105	0	1	9
110	0	0	a
115	0	0	a

GUI Output.



Area Report

Cnt-updown	21	582.120	0	582.120

Power Report

memory	0	0	0	0
Reg	1.8232e-08	1.7483e-04	2.3433e-05	1.3828e-04
latch	0.0	0	0	
logic	5.1915e-09	1.20965e-05	5.5275 1e-05	
Clock	0	0	0	
Pad	0	0	0	
Pm	0	0	0	
bbom	0	0	0	

Gates Report:

Sequential	4	372.557	64.0
inverter	7	46.570	8.0
logic	10	162.994	28.0
Physical_cell	0	0	0

Timing Report:

Timing Flag	Arc	Edge	cell	Fanout	load	Trans	Delay
count_reg[3]/ck	-	-	R	arr	4	-	100	-
count_reg[3]/QN	-	CK→QN	F	DFFRXL	2	12.1	208	759
g315/y	-	A→Y	R	CLKINVX3	4	160.4	665	486
count[3]	cc	-	R	port	-	-	-	0

Required Time = 1200 ps

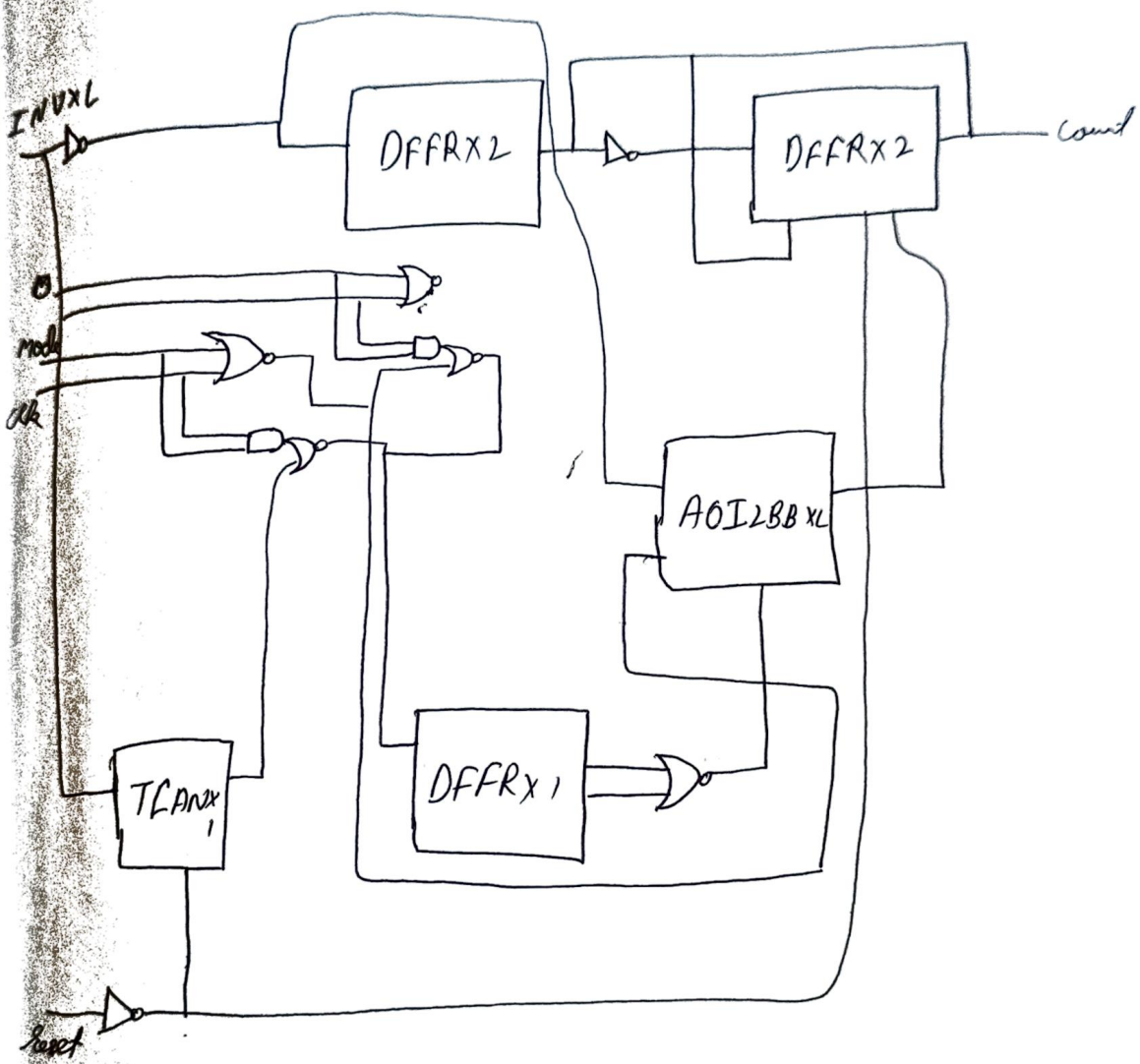
Arrival Time = 1185 ps

Slack = Required Time – Arrival Time = 15 ps

Critical Path Delay = Clock Period – Slack = 1.985 ns

Maximum Frequency of operation = $\frac{1}{\text{Arrival Time}}$ = 853.88 MHz

Schematic:


INVXL
DFFRX2
DFFRX2
count
0
mode
clk
AOI2BB XL
TLATNX 1
DFFRX 1
reset

b) 32-bit up/down asynchronous reset counter

Verilog code for 32-bit up/down asynchronous reset counter:

```
module cnt_updown (count, mode, clk, reset);
output reg [31:0] count;
input mode;
input clk, reset;

always @ (posedge clk, posedge reset)

if (reset)
count = 32'b0;
else
if (mode)
count = count + 1;
else
count = count - 1;

endmodule
```

Testbench for 32-bit up/down asynchronous reset counter:

```
module tb_cnt_updown;
reg mode;
reg clk, reset;
wire [31:0] count;

cnt_updown M1(count, mode, clk, reset);

initial
begin
$monitor ("time = %0d", $time, "ns", "reset = 0x%0h", reset, " mode = 0x%0h", mode, " count
= 0x%0h", count);
#900 $finish;
```



```
end

always
#5 clk = ~clk;

initial
begin
mode = 1; clk=0; reset=1;
#10; reset = 0;
#300; mode = 0;
#150; reset = 1;
#90; reset = 0;
#300; mode = 1;
#10; reset = 1;
end

endmodule
```

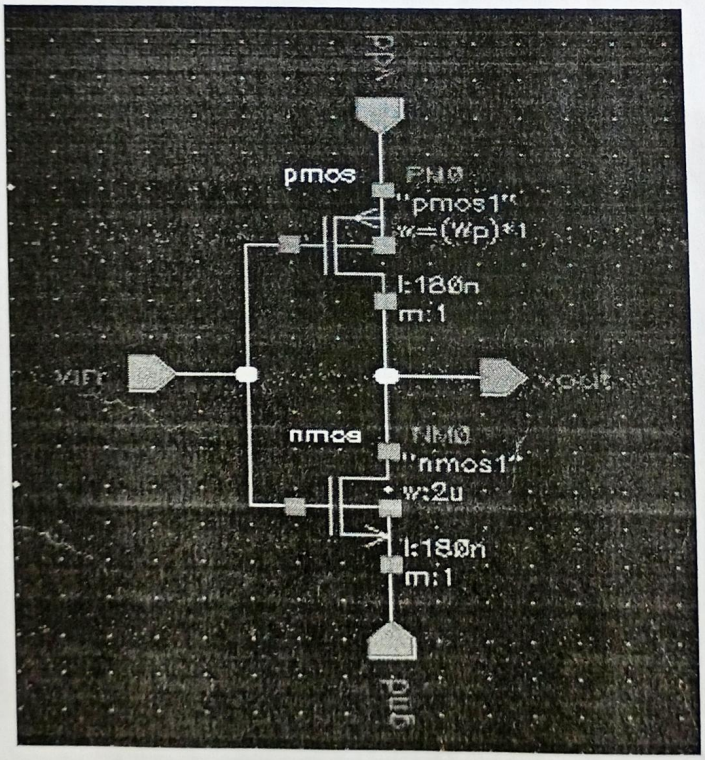
Follow the same steps for synthesize and observe the results

1. a) Capture the schematic of CMOS inverter with load capacitance of 0.1pF and set the widths of inverter with Wn = Wp, Wn = 2Wp, Wn = Wp/2 and length at the selected technology. Carry out the following:

   i) Set the input signal to a pulse with rise time, fall time of 1ns and pulse width of 10ns and time period of 20ns and plot the input voltage and output voltage of designed inverter?

   ii) From the simulation result compute t_{pHL}, t_{pLH} and t_d for all three geometrical settings of width?

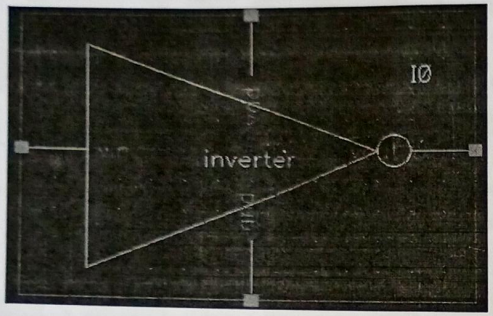
   iii) Tabulate the results of delay and find the best geometry for minimum delay for CMOS inverter



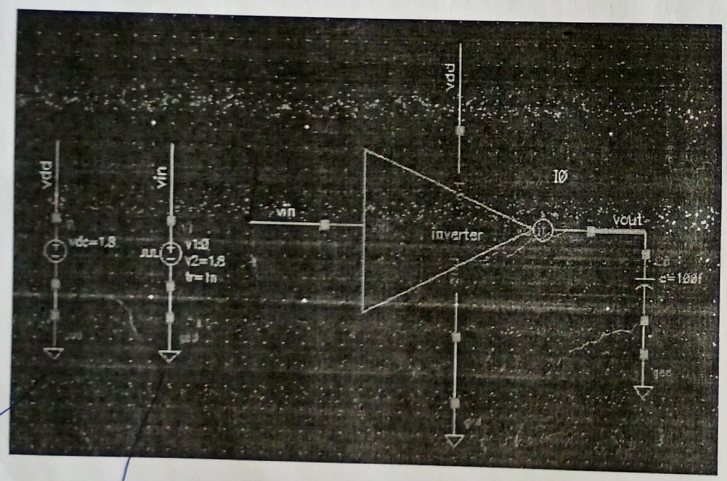
CMOS Inverter schematic

Table of components for building the schematic:

Library Name	Cell Name	Properties
gpdk180	pmos	W = Wp, L = 180n
gpdk180	nmos	W = 2u, L = 180n



CMOS Inverter symbol



CMOS Inverter test schematic

Table of components for building the test schematic:

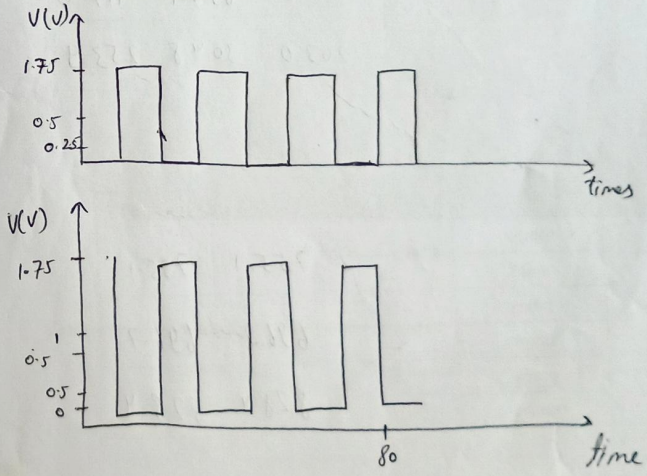
Library Name	Cell Name	Properties
analogLib	Vpulse	V1 = 0, V2 = 1.8, Period = 20n, Rise time = 1n, Fall time = 1n, Pulse width = 10n
analogLib	Vdc	Vdc = 1.8
analogLib	gnd	
analogLib	cap	0.1pF

Table of values to setup for different analysis:

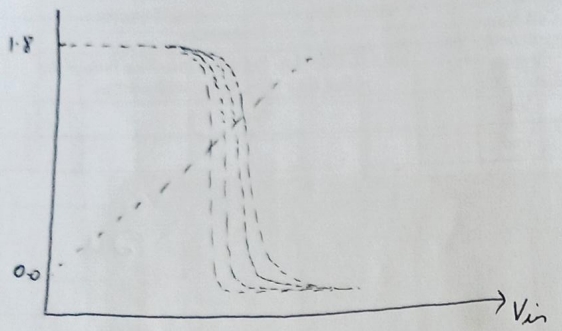
Analysis Name	Settings	Properties
Transient	trans	Stop time = 100n, moderate
DC	DC Analysis	Save DC Operating point
	Sweep Variable Component Parameter	Component Name = Select input signal component (Vpulse) Parameter Name = dc
	Sweep Range Start – Stop	Start = 0, Stop 1.8

Analog Simulation with spectre for inverter:

a) Transient Response



b) DC Response



Tabulated Values of Delays and DC Operating Points :

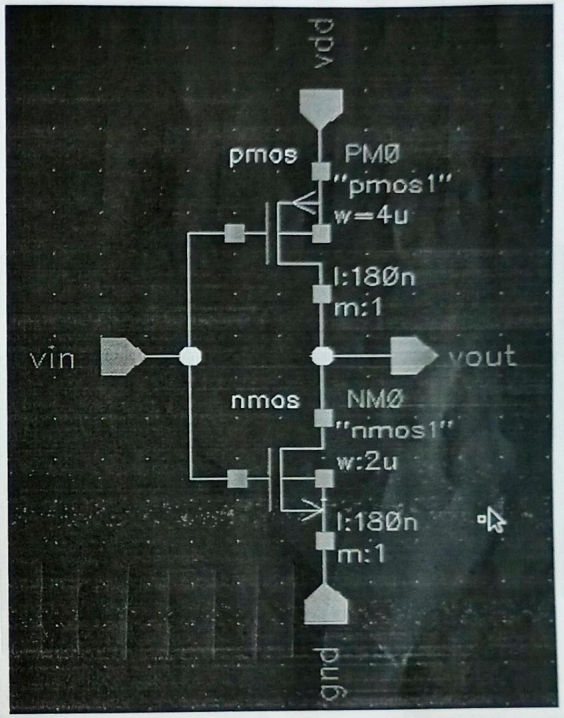
Values of t_{phl}, t_{plh} and t_{pd} for different geometries

Width setting	MOSFET	Width	t_{phl} (ps)	t_{plh} (ps)	t_{pd} (ps)
Wp = Wn	pmos	2u	187.5	445.5	316.5
	nmos	2u			
Wp = Wn / 2	pmos	1u	177.8	657.1	417.45
	nmos	2u			
Wp = 2 Wn	pmos	4u	203.0	304.8	253.9
	nmos	2u			

DC operating point values for different geometries

Width setting	MOSFET	Width	Vin (mV)	Vout (mV)
Wp = Wn	pmos	2u	755.1	755.1
	nmos	2u		
Wp = Wn / 2	pmos	1u	696.7	696.7
	nmos	2u		
Wp = 2 Wn	pmos	4u	828.4	828.4
	nmos	2u		

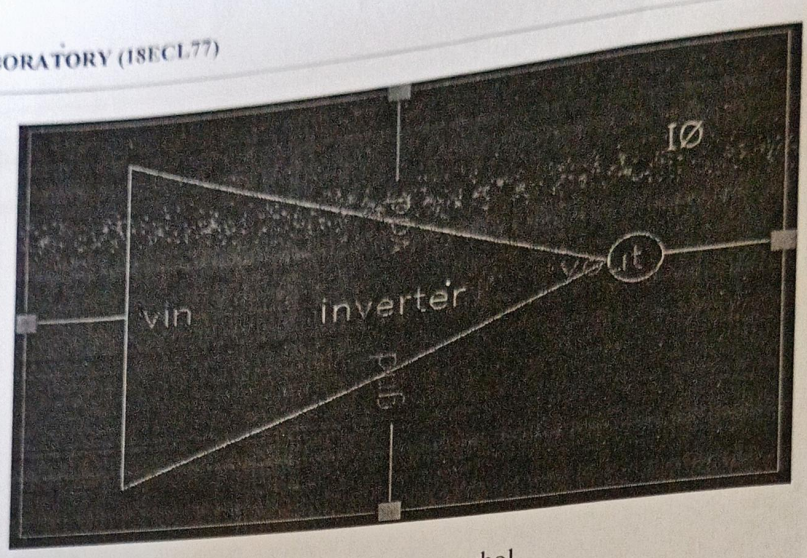
1. **b) Draw layout of inverter with Wp/Wn = 40/20, use optimum layout methods. Verify for DRS and LVS, extract parasitic and perform post layout simulations, compare the results of with pre-layout simulations. Record the observations.**



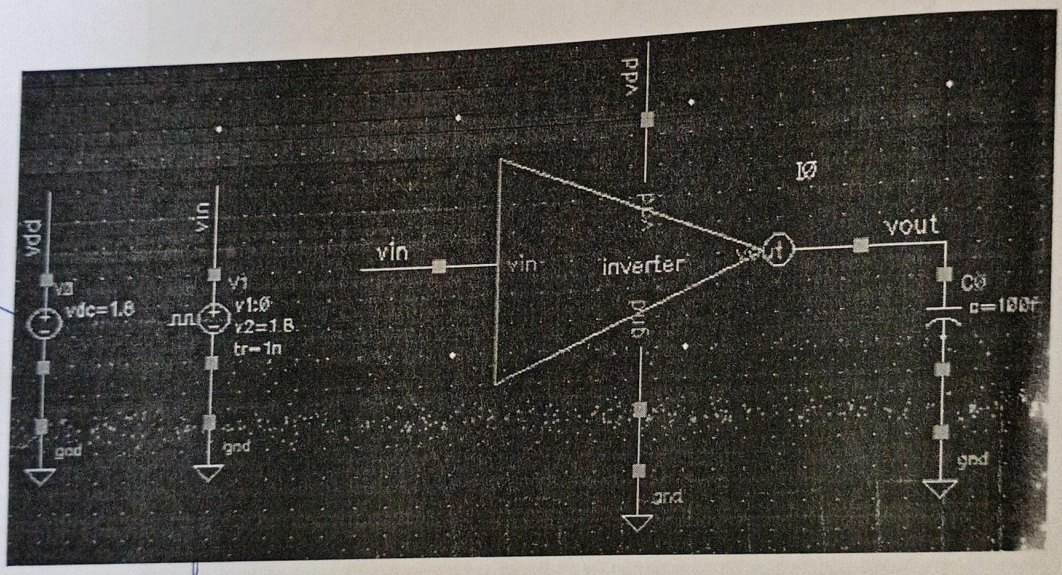
CMOS Inverter schematic

Table of components for building the schematic:

Library Name	Cell Name	Properties
gpdk180	pmos	W = 4u, L = 180n
gpdk180	nmos	W = 2u, L = 180n



CMOS Inverter symbol



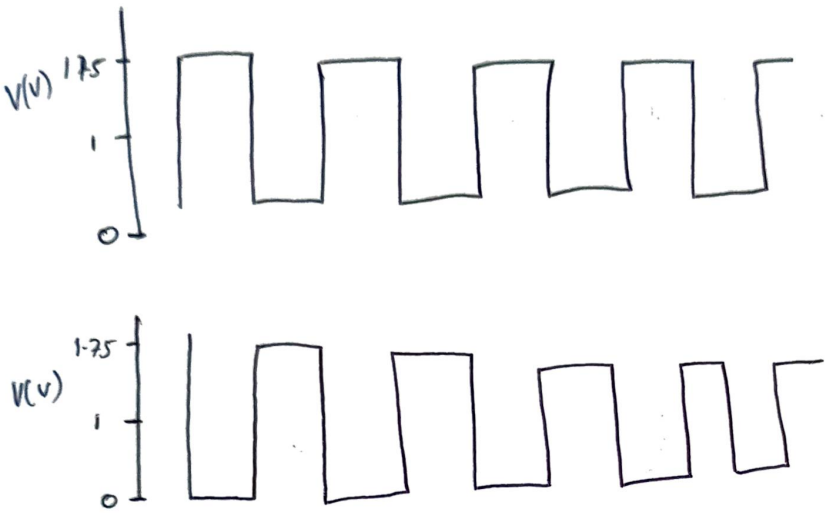
CMOS Inverter test schematic

Table of components for building the test schematic:

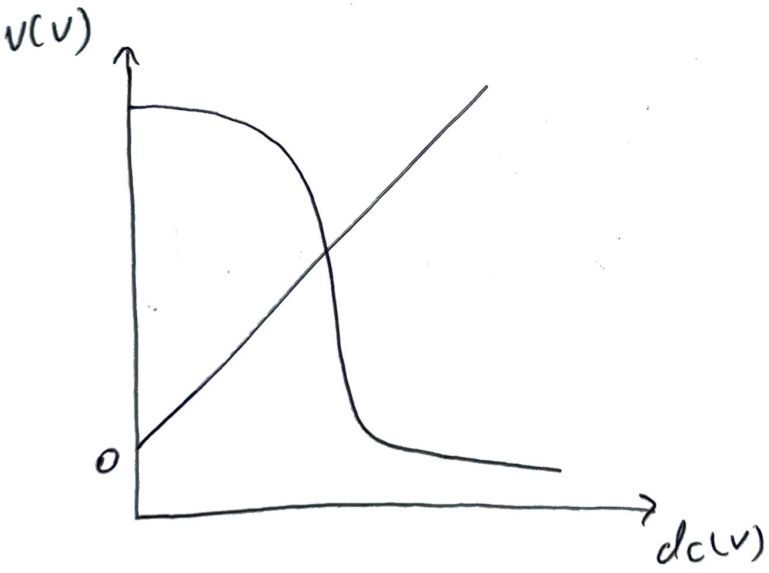
Library Name	Cell Name	Properties
analogLib	Vpulse	V1 = 0, V2 = 1.8, Period = 20n, Rise time = 1n, Fall time = 1n, Pulse width = 10n
analogLib	Vdc	Vdc = 1.8
analogLib	gnd	
analogLib	cap	0.1pF

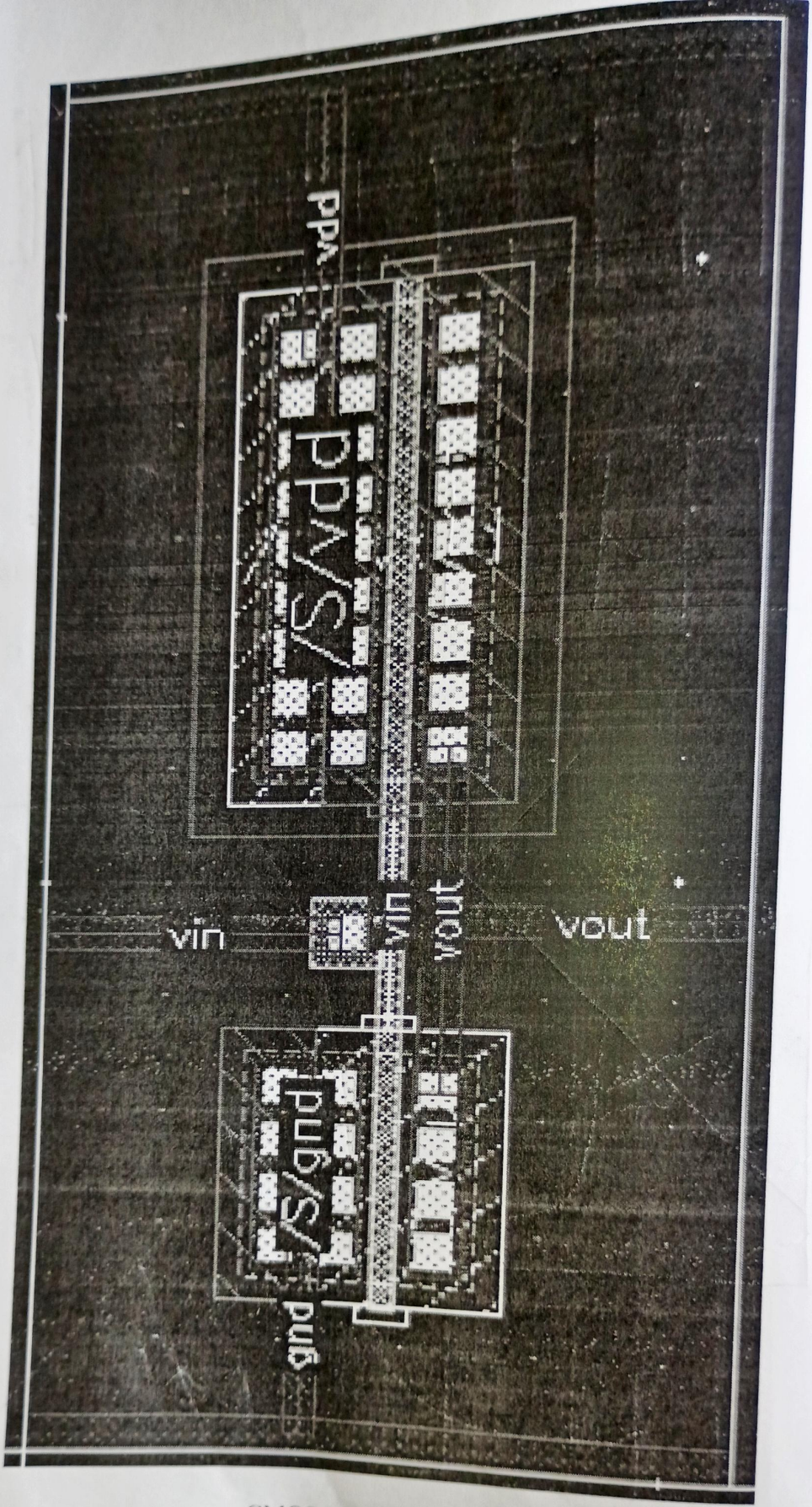
Analog Simulation with spectre for inverter test schematic:

a) Transient Response



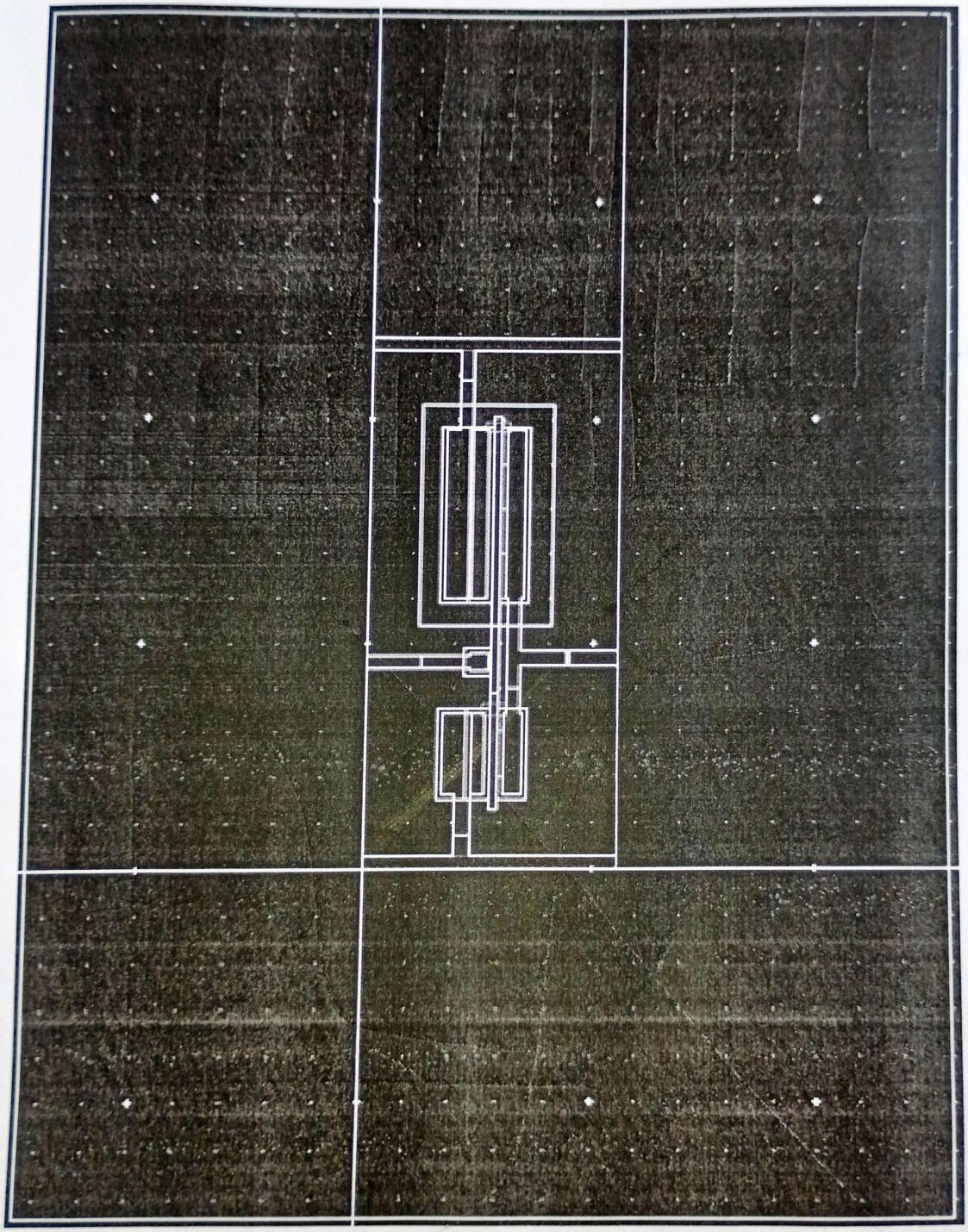
b) DC Response





CMOS Inverter Layout

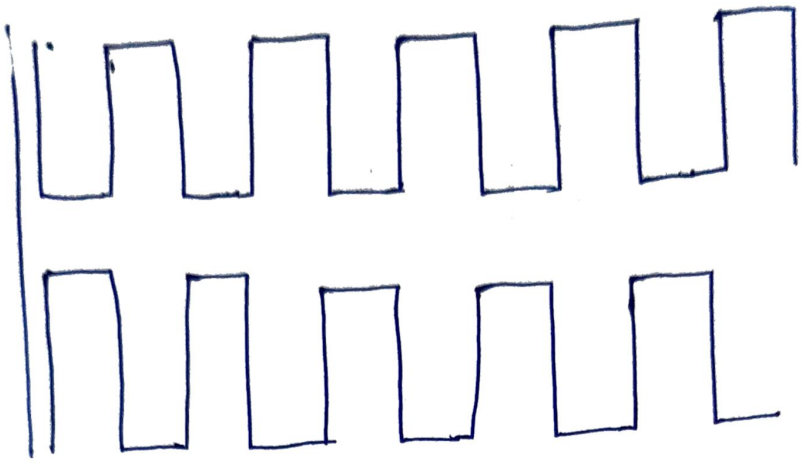
CMOS Inverter av_extracted view:



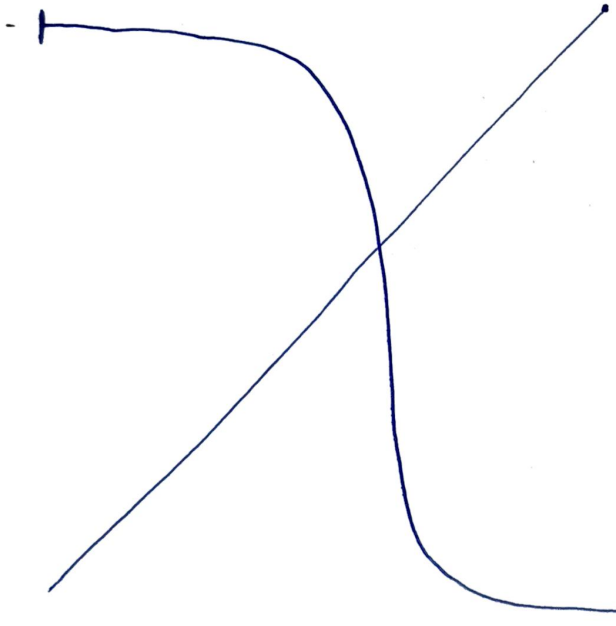
CMOS Inverter av_extracted view

Analog Simulation with spectre for inverter layout:

a) Transient Response



b) DC Response



Tabulated Values of Delay:

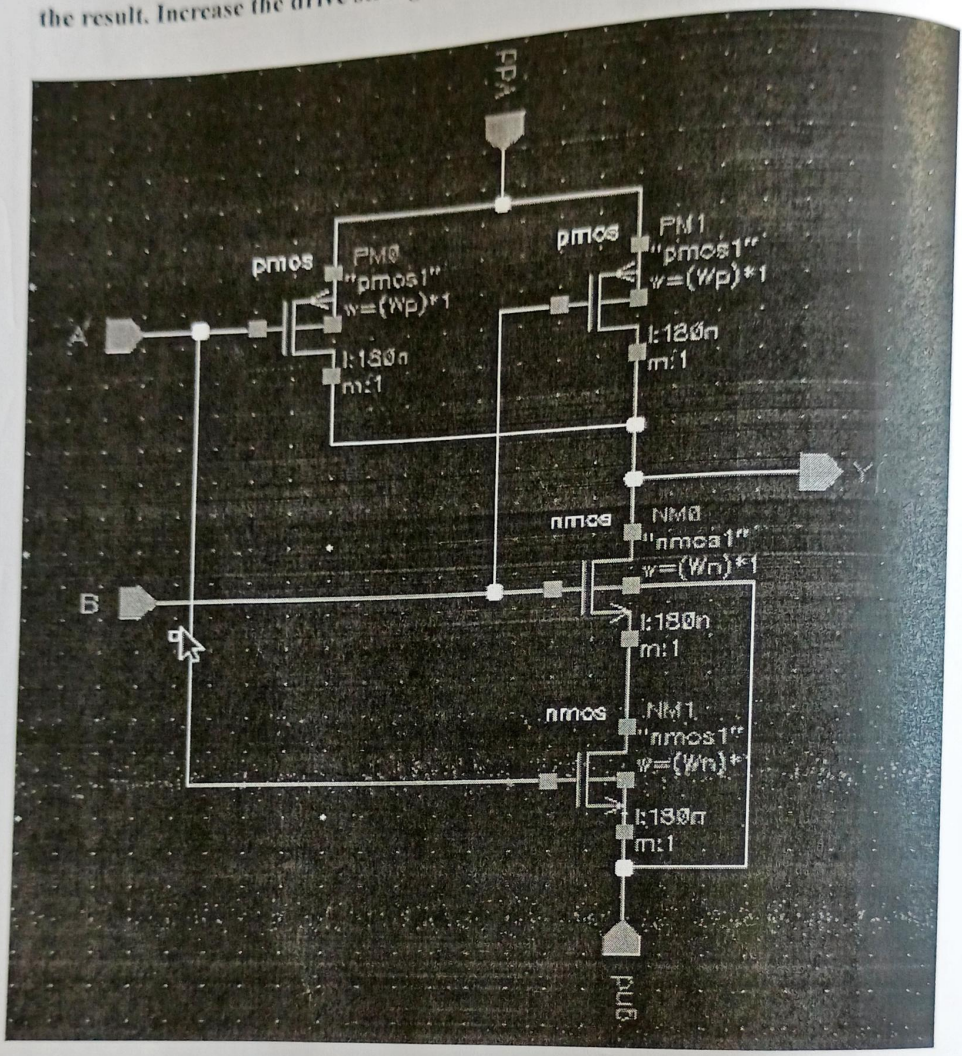
Values of t_{phl}, t_{plh} and t_{pd} for Wp/Wn = 40/20

	t_{phl} (ps)	t_{plh} (ps)	t_{pd} (ps)
CMOS Inverter Test Schematic	203.7	303.2	253.45
CMOS Inverter Layout	206.7	305.9	256.3

DC operating point values for Wp/Wn = 40/20

	Vin (mV)	Vout (mV)
CMOS Inverter Test Schematic	828.4	828.4
CMOS Inverter Layout	828.7	828.7

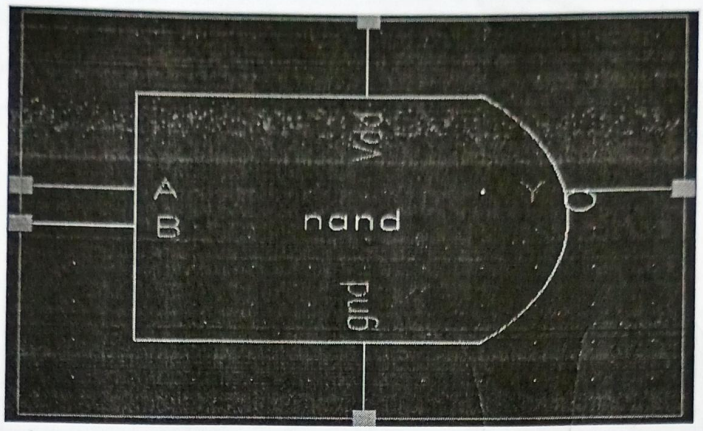
2. a) Capture the schematic of 2-input CMOS NAND gate having similar delay as that of CMOS inverter computed in experiment 1. Verify the functionality of NAND gate and also find out the delay to for all four possible combinations of input vectors. Tabulate the result. Increase the drive strength to 2X and 4X and tabulate the result.



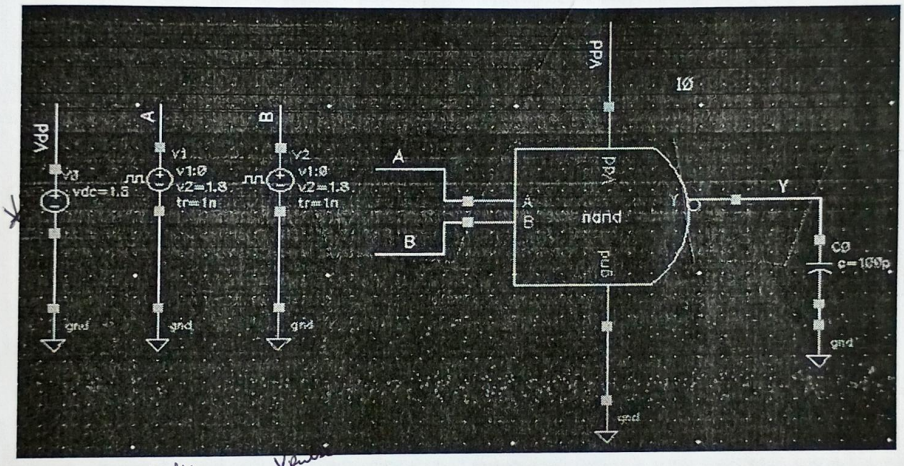
Two Input CMOS NAND Gate schematic

Table of components for building the schematic:

Library Name	Cell Name	Properties
gpdk180	pmos	W = Wp, L = 180n
gpdk180	nmos	W = Wn, L = 180n



Two Input CMOS NAND Gate symbol



Two Input CMOS NAND Gate test schematic

Table of components for building the test schematic:

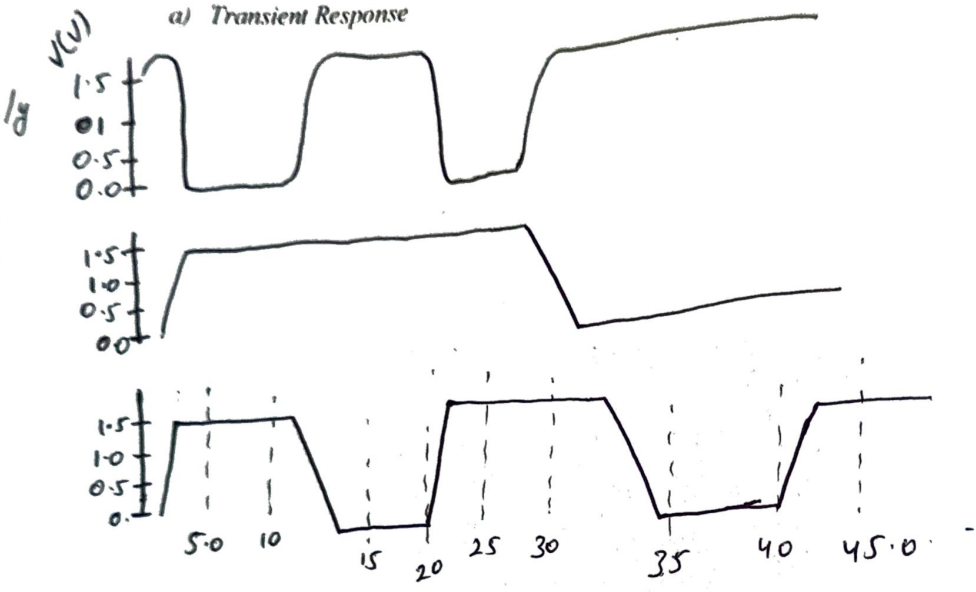
Library Name	Cell Name	Properties
analogLib	Vpulse	V1 = 0, V2 = 1.8, Period = 20n, Rise time = 1n, Fall time = 1n, Pulse width = 10n
analogLib	Vpulse	V1 = 0, V2 = 1.8, Period = 50n, Rise time = 1n, Fall time = 1n, Pulse width = 25n
analogLib	Vdc	Vdc = 1.8
analogLib	gnd	
analogLib	cap	~~100pF~~ 0.1 PF

Table of values to setup for different analysis:

Analysis Name	Settings	Properties
Transient	trans	Stop time = 50n, moderate

Analog Simulation with spectre for Two Input CMOS NAND Gate:

a) Transient Response



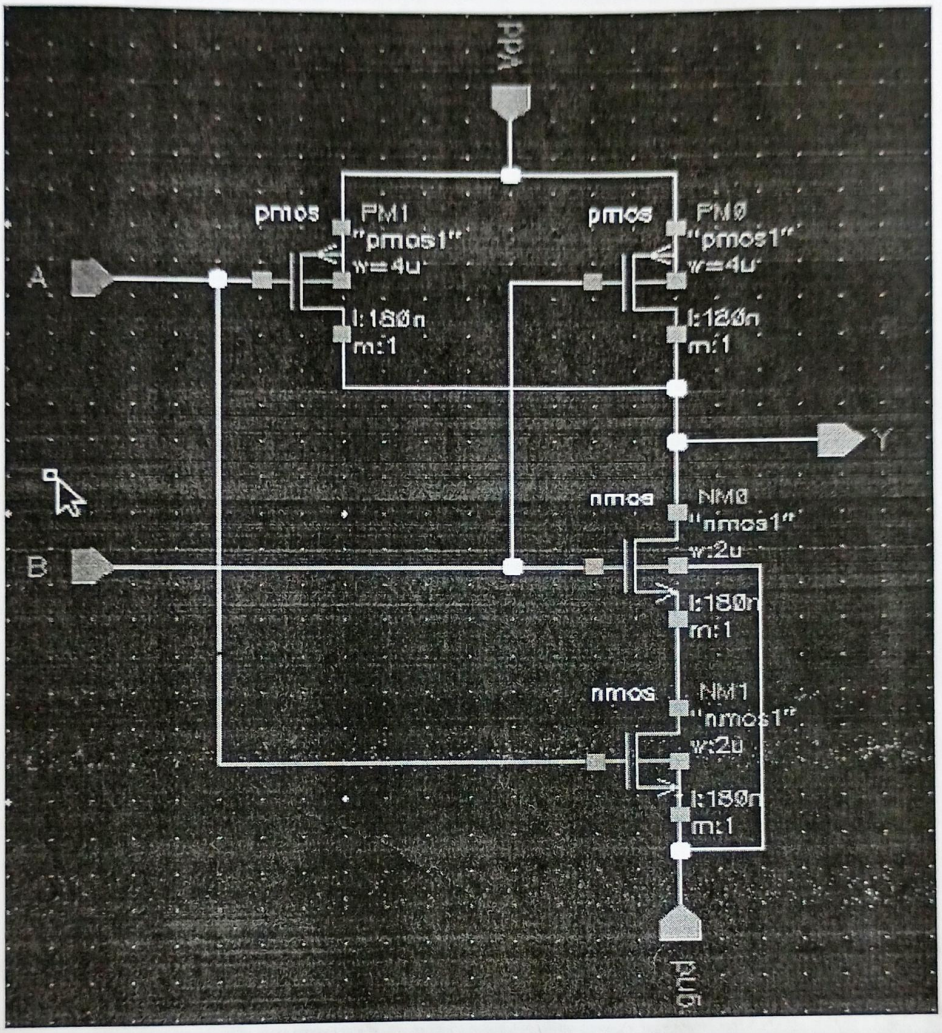
Tabulated Values of Delay:

Values of t_{phl}, t_{plh} and t_{pd} for different geometries

MOSFET	Width	t_{phl} (ps)	t_{plh} (ps)	t_{pd} (ps)
pmos	4u	337.1E-12	320.6E-12	328.85
nmos	2u			
pmos	8u	255.2E-12	241.1E-12	337.25
nmos	4u			
pmos	16u	210.2E-12	167.7E-12	
nmos	8u			

22/11/23

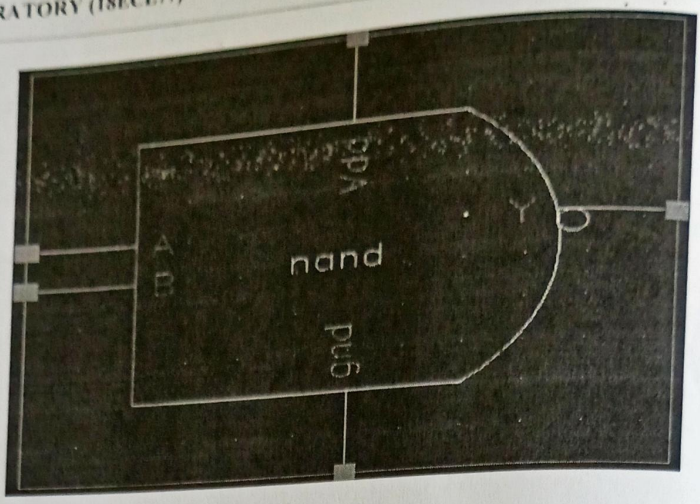
2. **b) Draw layout of NAND gate with Wp/Wn = 40/20, use optimum layout methods. Verify for DRS and LVS, extract parasitic and perform post layout simulations, compare the results of with pre-layout simulations. Record the observations.**



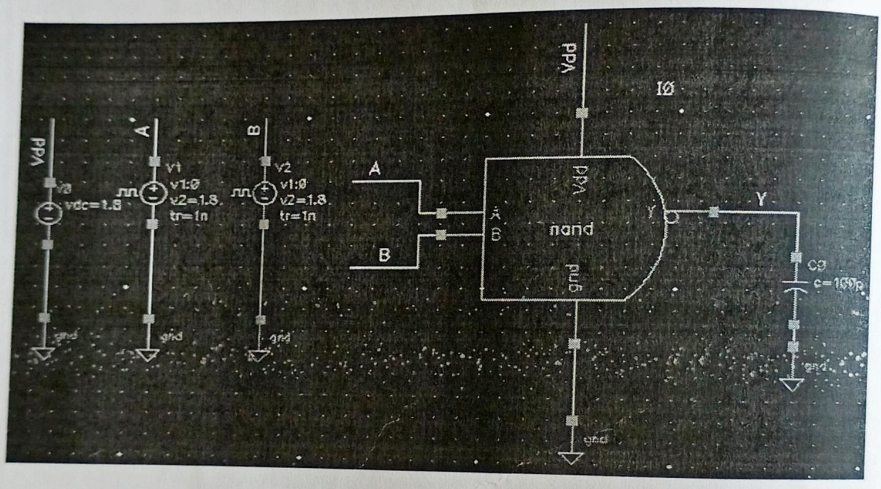
Two Input CMOS NAND schematic

Table of components for building the schematic:

Library Name	Cell Name	Properties
gpdk180	pmos	W = 4u, L = 180n
gpdk180	nmos	W = 2u, L = 180n



Two Input CMOS NAND Gate symbol



Two Input CMOS NAND Gate test schematic

Table of components for building the test schematic:

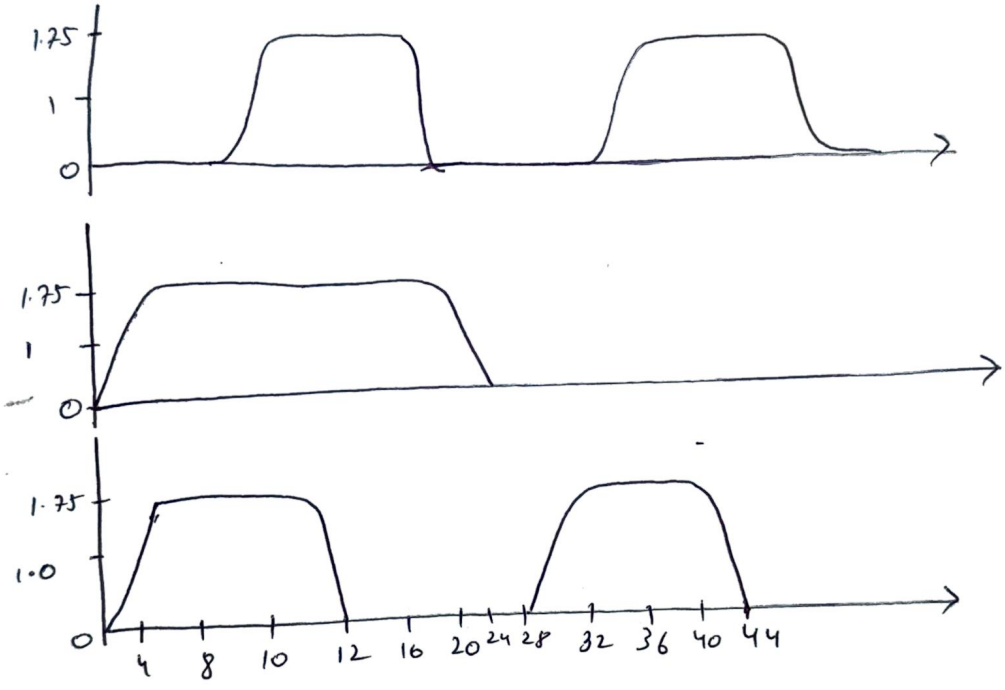
Library Name	Cell Name	Properties
analogLib	Vpulse	V1 = 0, V2 = 1.8, Period = 20n, Rise time = 1n, Fall time = 1n, Pulse width = 10n
analogLib	Vpulse	V1 = 0, V2 = 1.8, Period = 50n, Rise time = 1n, Fall time = 1n, Pulse width = 25n
analogLib	Vdc	Vdc = 1.8
analogLib	gnd	
analogLib	cap	100p F

Table of values to setup for Wp/Wn = 40/20:

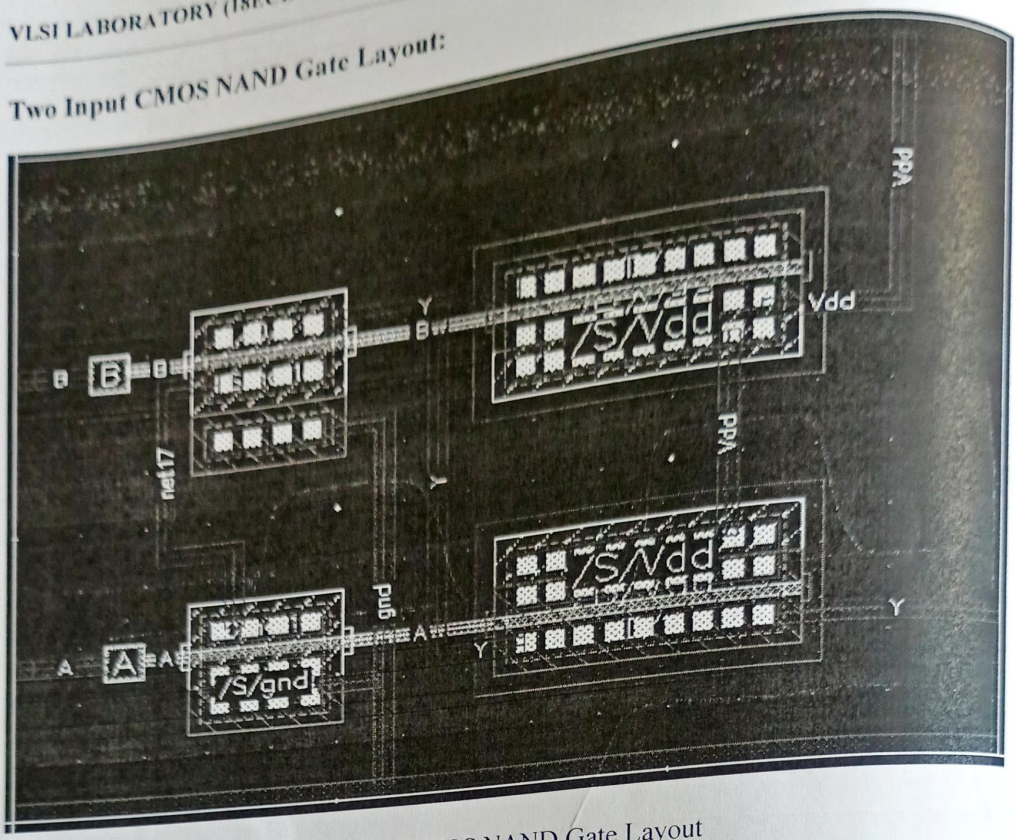
Analysis Name	Settings	Properties
Transient	trans	Stop time = 50n, moderate

Analog Simulation with spectre for Two Input CMOS NAND Gate test schematic:

a) Transient Response

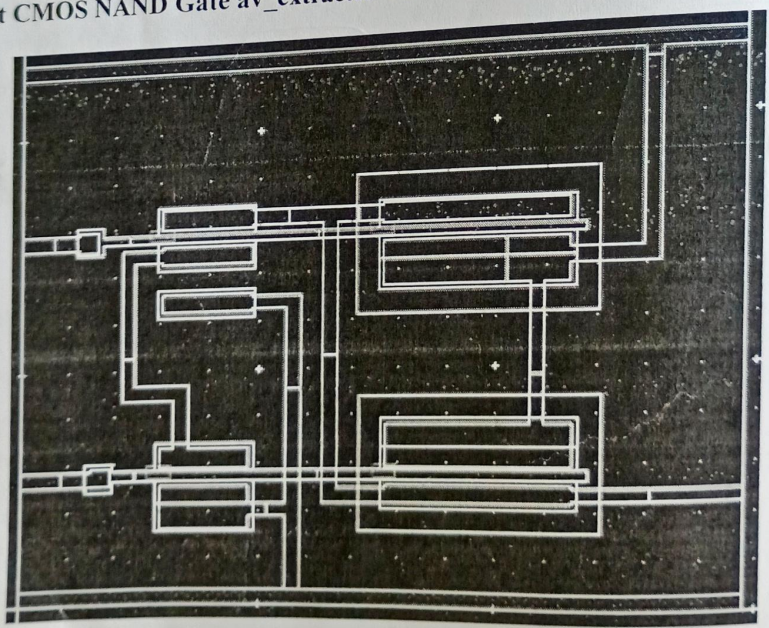


Two Input CMOS NAND Gate Layout:



Two Input CMOS NAND Gate Layout

Two Input CMOS NAND Gate av_extracted view:



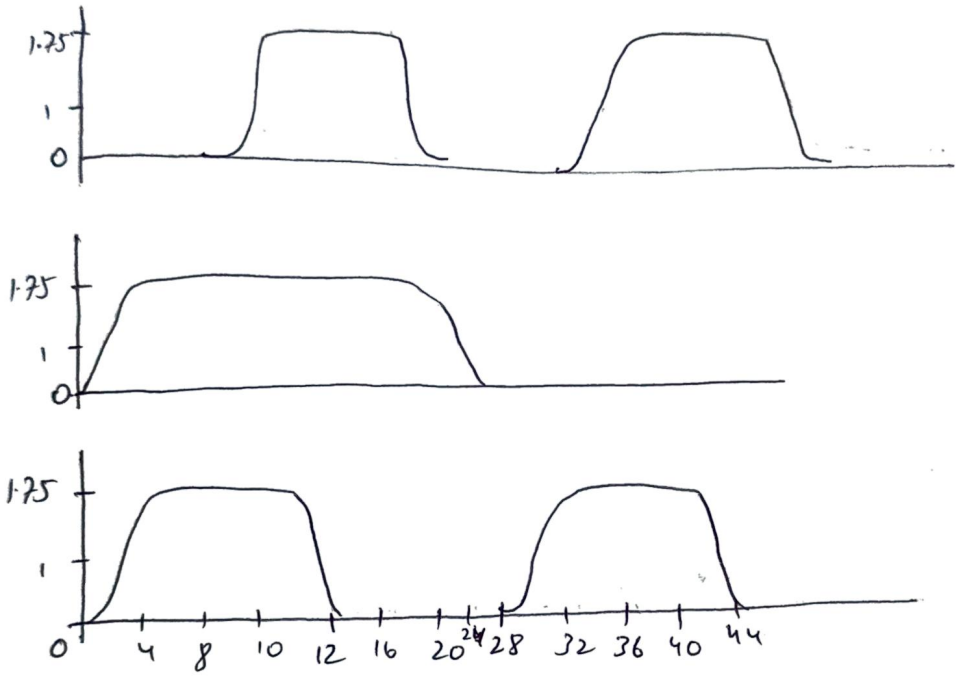
Two Input CMOS NAND Gate av_extracted view

Table of values to setup for Wp/Wn = 40/20:

Analysis Name	Settings	Properties
Transient	trans	Stop time = 50n, moderate

Analog Simulation with spectre for Two Input CMOS NAND Gate Layout:

a) Transient Response



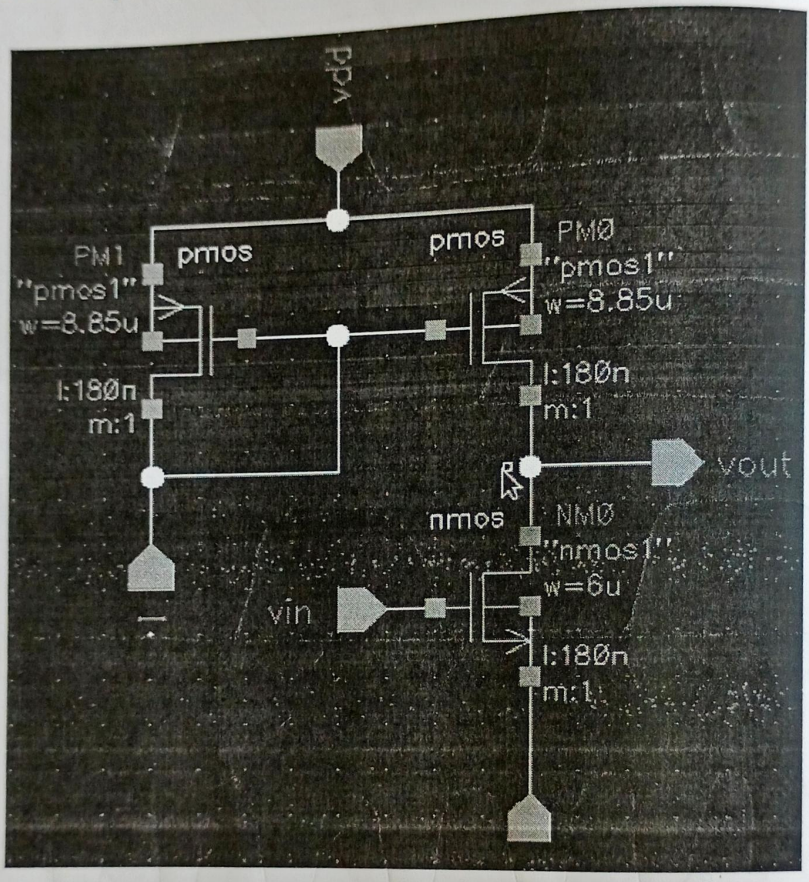
Tabulated Values of Delay:

Values of t_{phl}, t_{plh} and t_{pd} for different geometries

	t_{phl} (ps)	t_{plh} (ps)	t_{pd} (ps)
Two Input CMOS NAND Gate Test Schematic	337.1	320.6	328.85
Two Input CMOS NAND Gate Layout	345.9	329.6	337.75

3. **a) Capture the schematic of Common Source Amplifier with PMOS Current Mirror load and find its transient response and AC response? Measure the Unity Gain Bandwidth (UGB), amplification factor by varying transistor geometries, study the impact of variation in width to UGB.**

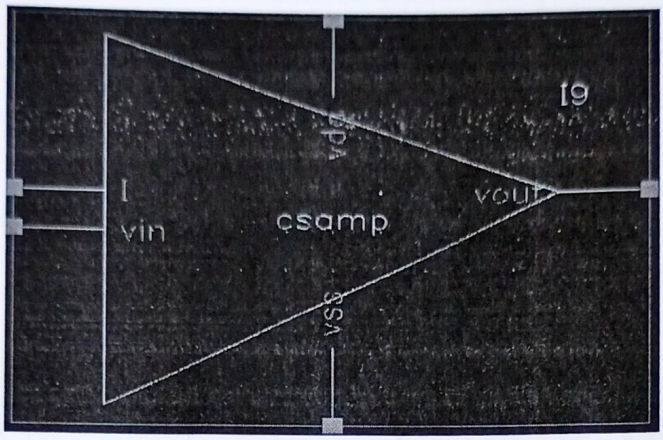
b) Draw layout of common source amplifier, use optimum layout methods. Verify for DRS and LVS, extract parasitic and perform post layout simulations, compare the results of with pre-layout simulations. Record the observations.



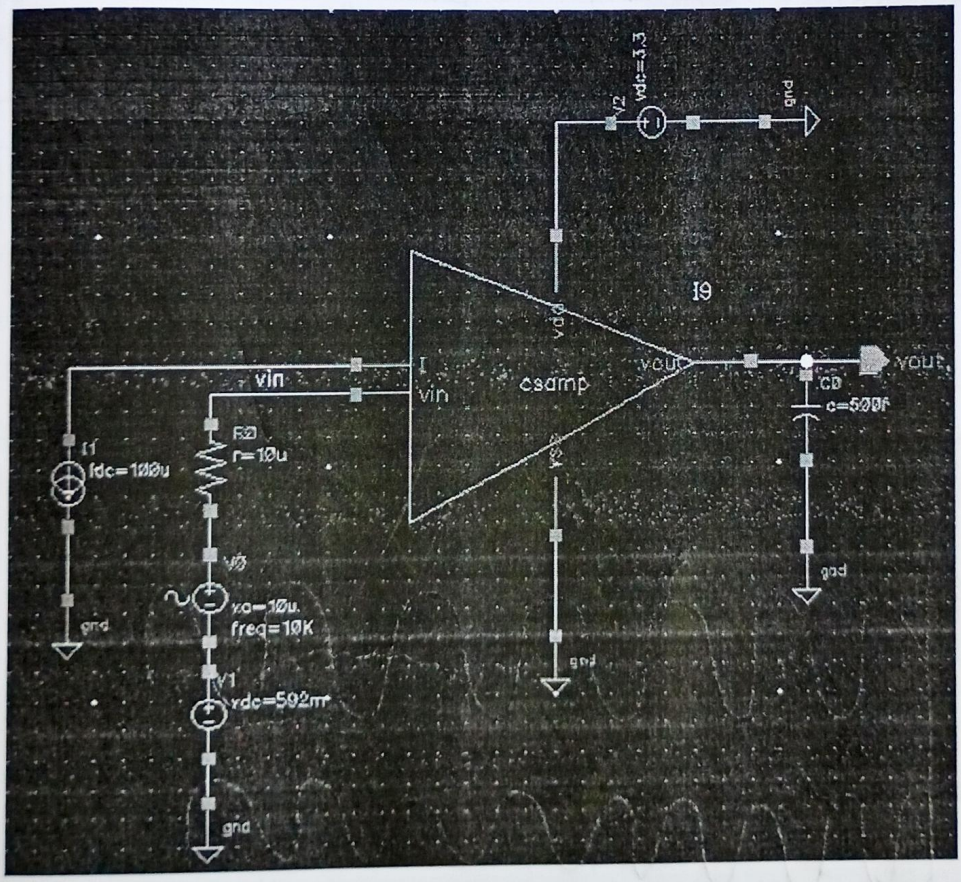
Common Source Amplifier schematic

Table of components for building the schematic:

Library Name	Cell Name	Properties
gpdk180	pmos	W = 8.85u, L = 180n
gpdk180	nmos	W = 6u, L = 180n



Common Source Amplifier symbol



Common Source Amplifier test schematic

Table of components for building the test schematic:

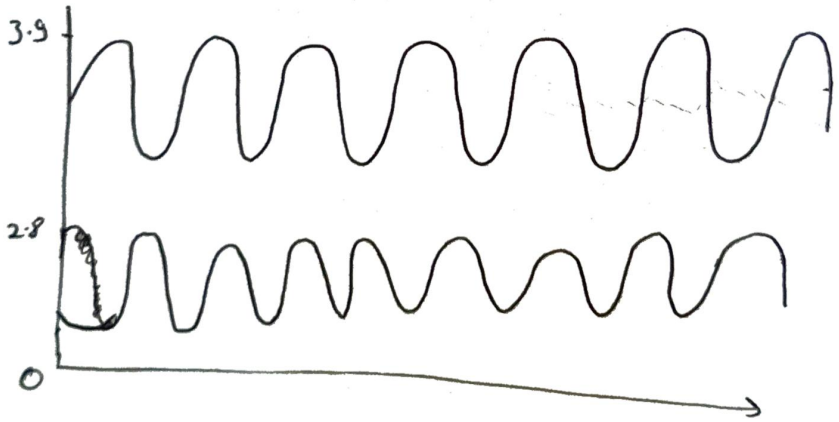
Library Name	Cell Name	Properties
analogLib	Vdc	DC Voltage = 3.3 V (V_{dd})
analogLib	Vsin	AC Magnitude = 1 V, Amplitude = 10m V, Frequency = 1K Hz
analogLib	Vdc	DC Voltage = 592m V
analogLib	res	Resistance = 10u Ohms
analogLib	idc	DC Current = 100u A
analogLib	cap	500f F

Table of values to setup for different analysis:

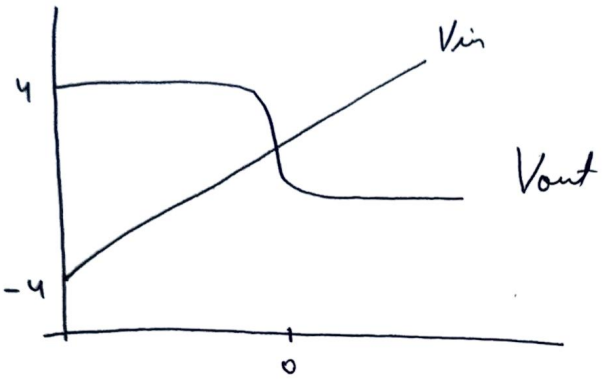
Analysis Name	Settings	Properties
Transient	trans	Stop time = 5m, moderate
DC	DC Analysis	Save DC Operating point
	Sweep Variable Component Parameter	Component Name = Select input signal component (Vpulse) Parameter Name = dc
	Sweep Range Start – Stop	Sweep Type = Linear Start = -5, Stop = 5, Step size = 10m V
AC	Sweep Range Start – Stop	Sweep Type = Logarithm, Start = 10, Stop = 10G, Points Per Decade = 10

Analog Simulation with spectre for Common Source Amplifier:

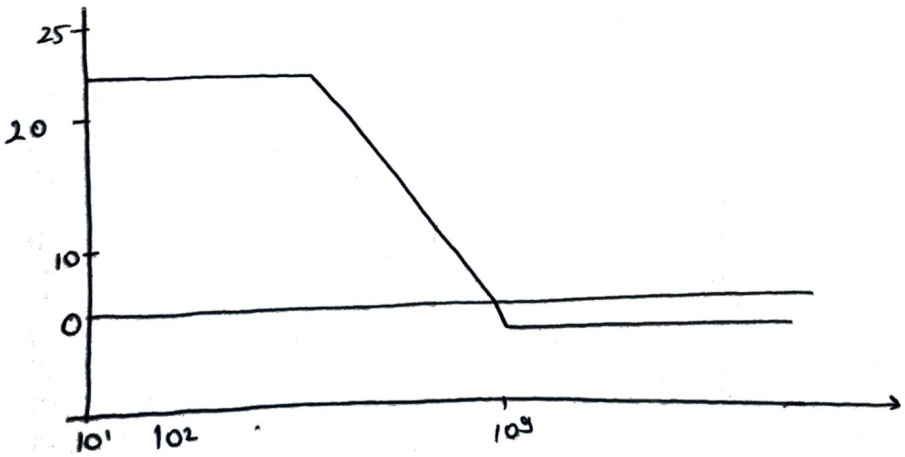
a) Transient Response



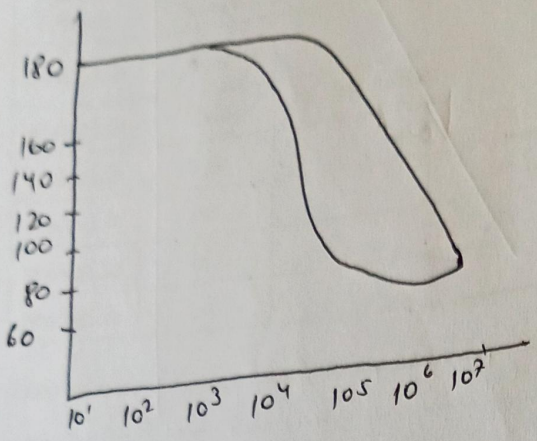
b) DC Response



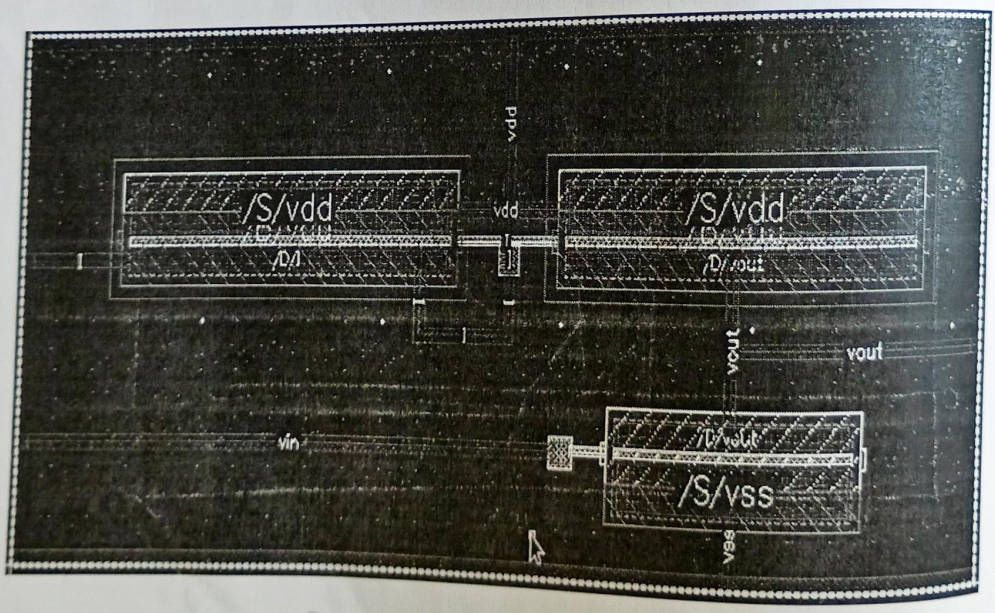
c) AC Response



d) AC Magnitude and Phase Response



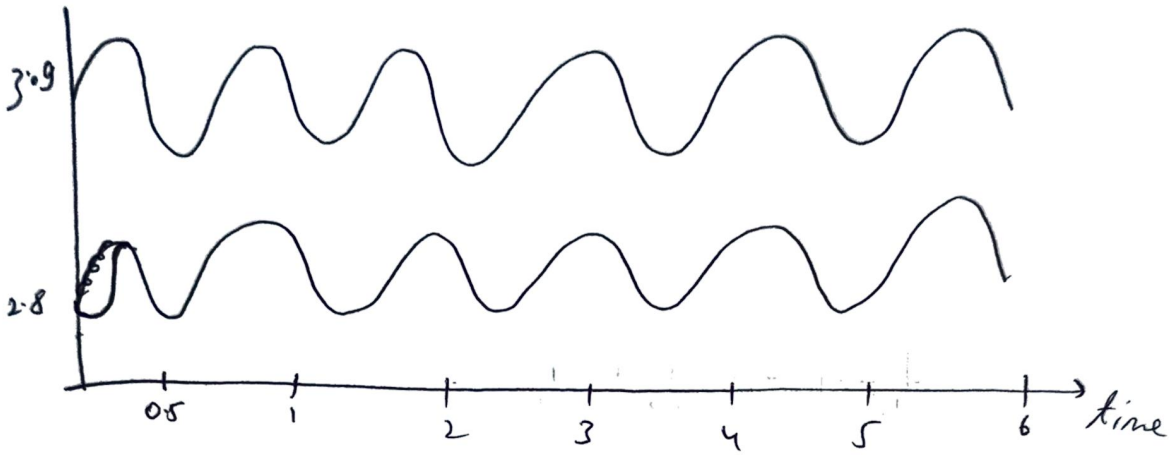
Common Source Amplifier Layout:



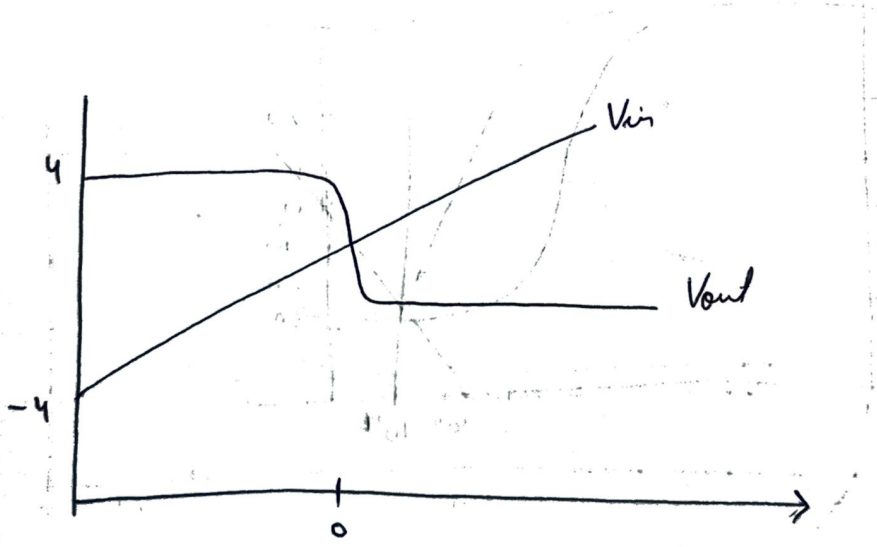
Common Source Amplifier Layout

Analog Simulation with spectre for Common Source Amplifier:

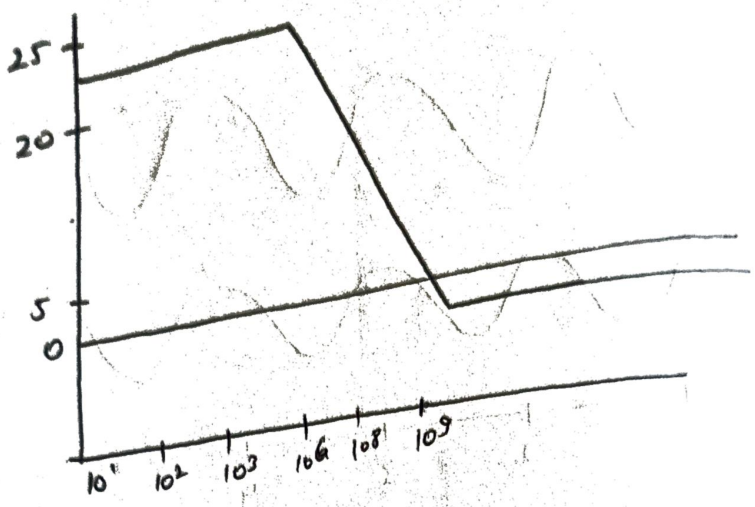
a) Transient Response



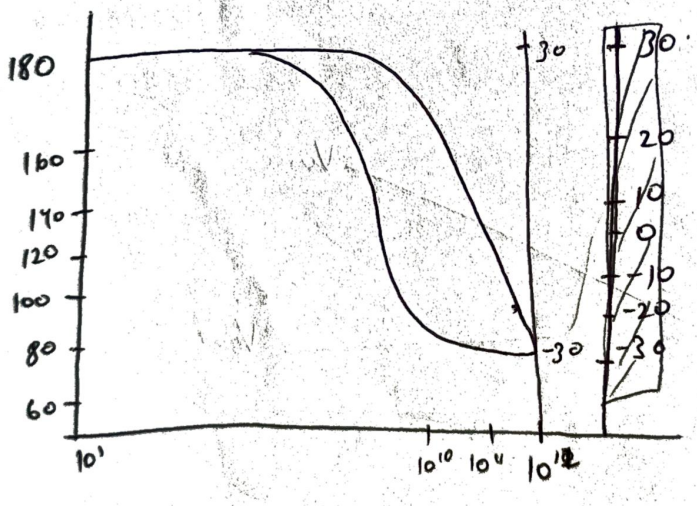
b) DC Response



c) *AC Response*



d) *AC Magnitude and Phase Response*



Results:

Common Source Amplifier		
	Gain	**UGB**
Schematic	23.685	401.102
Layout	23.7195	345.704.

4. a) Capture schematic of two-stage operational amplifier and measure the following:

i) UGB

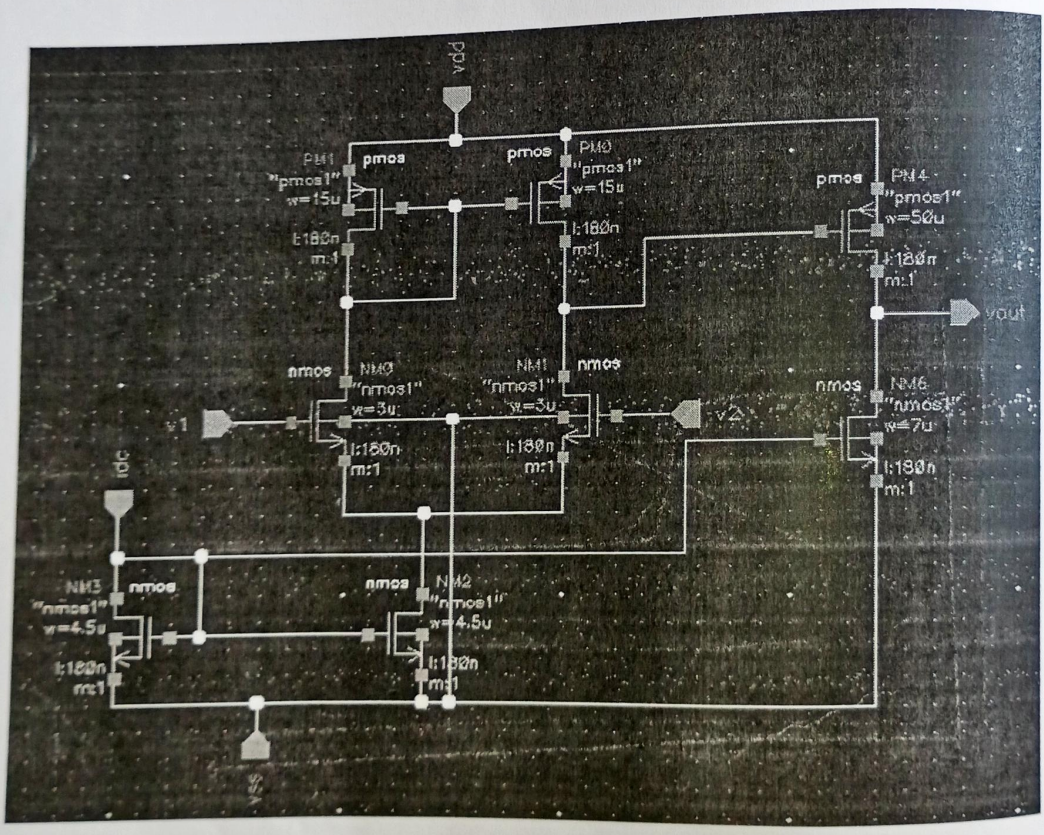
ii) dB bandwidth

iii) Gain margin and phase margin with and without coupling capacitance

iv) Use the op-amp in the inverting and non-inverting configuration and verify its functionality.

v) Study the UGB, 3dB bandwidth, gain and power requirement in op-amp by varying the stage wise transistor geometries and record the observation.

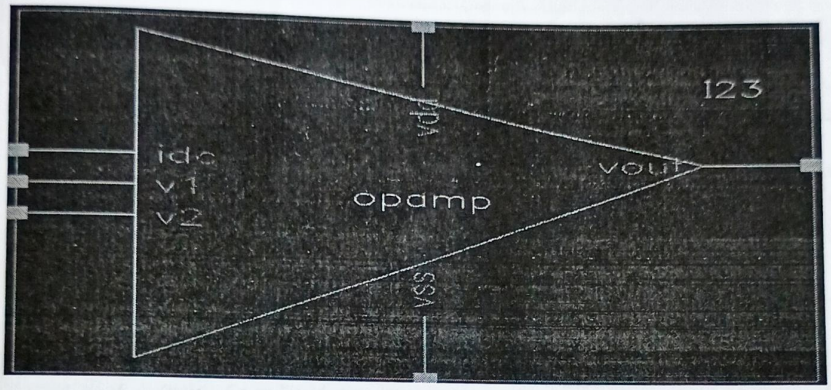
b) Draw layout of two-stage operational amplifier with minimum transistor width set to 300 (in 180/90/45 nm technology), choose appropriate transistor geometries as per the results obtained in 4. a. Use optimum layout methods. Verify for DRS and LVS, extract parasitic and perform post layout simulations, compare the results of with pre-layout simulations. Record the observations.



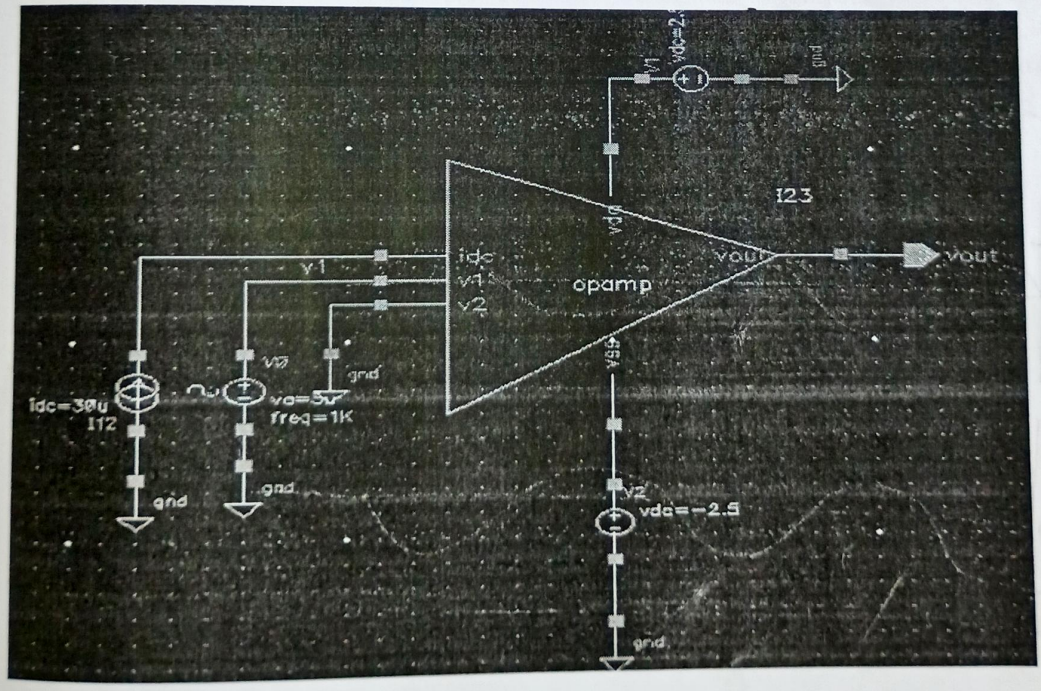
Operational Amplifier schematic

Table of components for building the schematic:

Library Name	Cell Name	Properties
gpdk180	pmos	W = 15u, L = 180n W = 50u, L = 180n
gpdk180	nmos	W = 3u, L = 180n W = 4.5u, L = 180n W = 7u, L = 180n



Operational Amplifier symbol



Operational Amplifier test schematic

Table of components for building the test schematic:

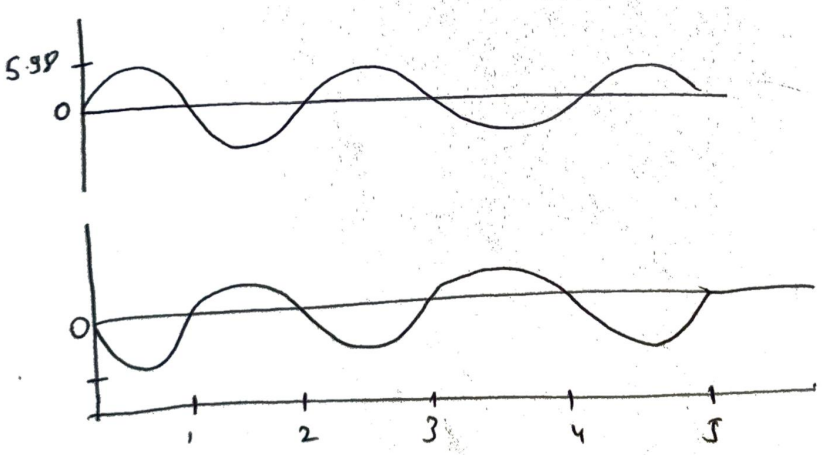
Library Name	Cell Name	Properties
analogLib	Vdc	DC Voltage = 2.5 V (V_{dd}) DC Voltage = -2.5 V (V_{ss})
analogLib	Vsin	AC Magnitude = 1 V, DC Voltage = 0 V, Offset Voltage = 0 V Amplitude = 5u V, Frequency = 1K Hz
analogLib	idc	DC Current = 30u A

Table of values to setup for different analysis:

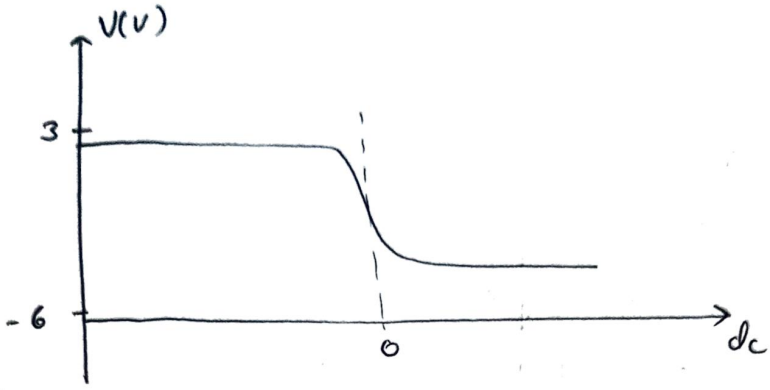
Analysis Name	Settings	Properties
Transient	trans	Stop time = 5m, moderate
DC	DC Analysis	Save DC Operating point
	Sweep Variable Component Parameter	Component Name = Select input signal component (Vpulse) Parameter Name = dc
	Sweep Range Start – Stop	Start = -5, Stop = 5
AC	Sweep Range Start – Stop	Sweep Type = Automatic, Start = 100, Stop = 10G,

Analog Simulation with spectre for Operational Amplifier:

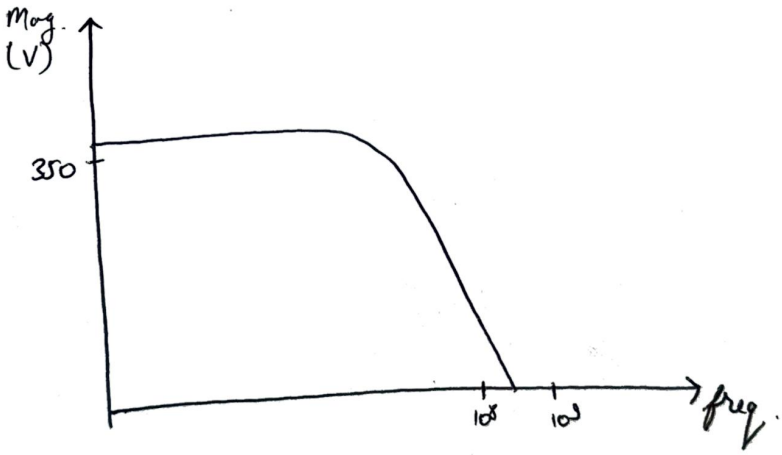
a) Transient Response



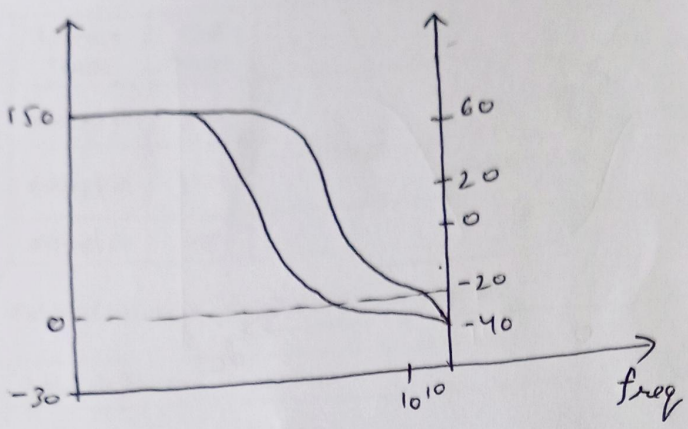
b) DC Response



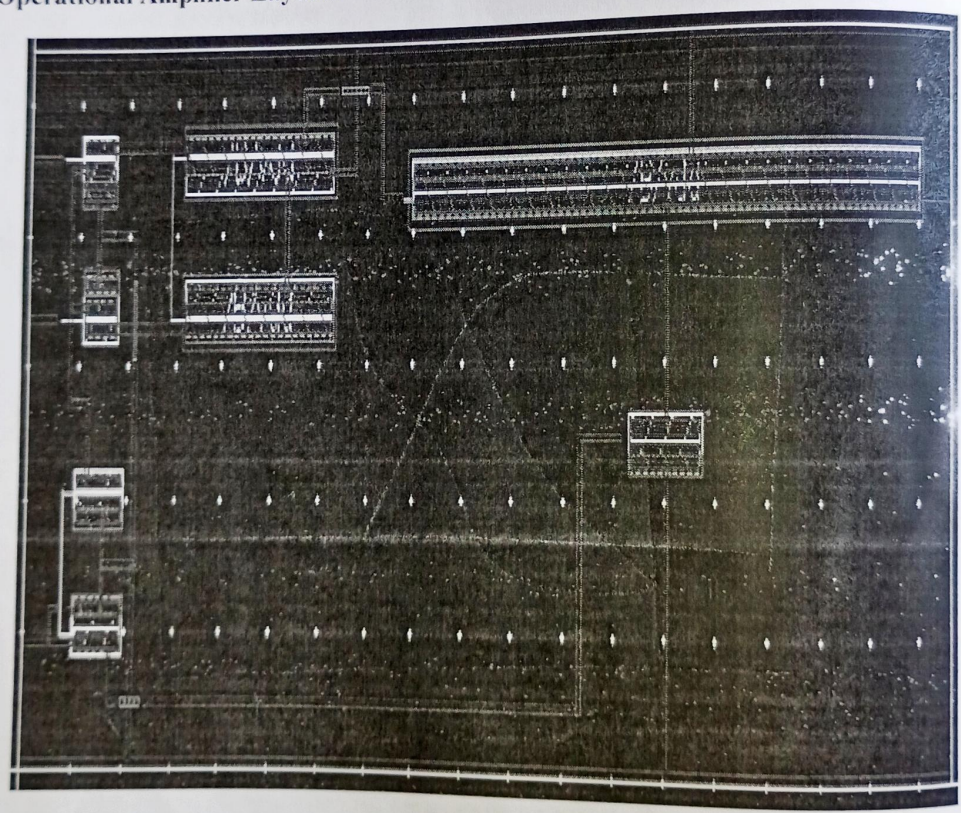
c) AC Response



d) AC Magnitude and Phase Response



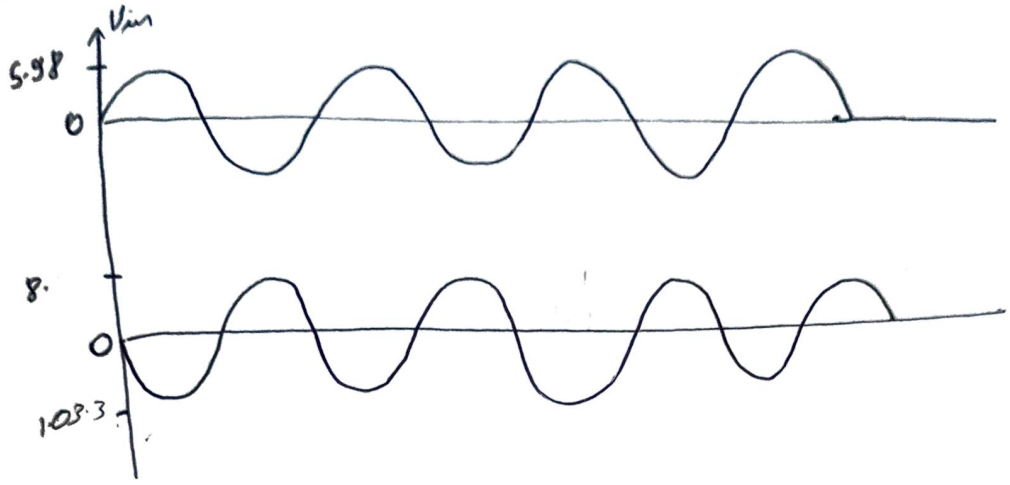
Operational Amplifier Layout:



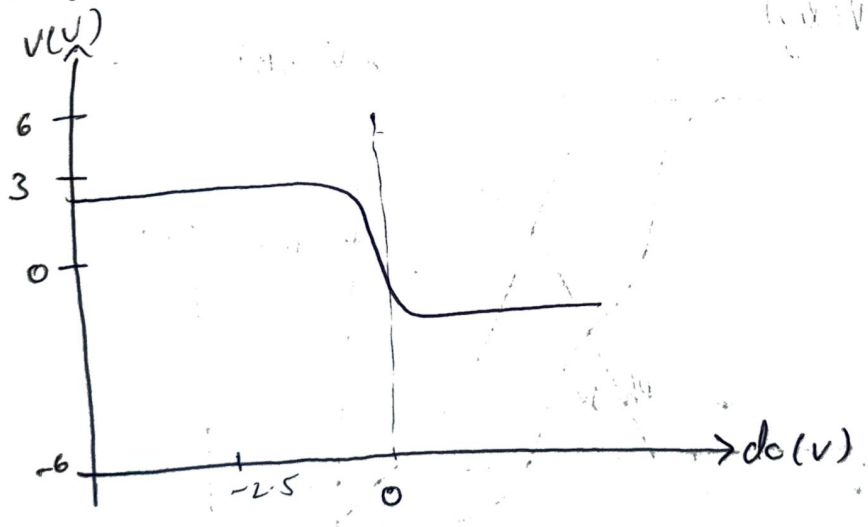
Operational Amplifier Layout

Analog Simulation with spectre for Operational Amplifier:

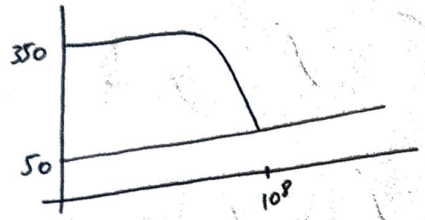
a) Transient Response



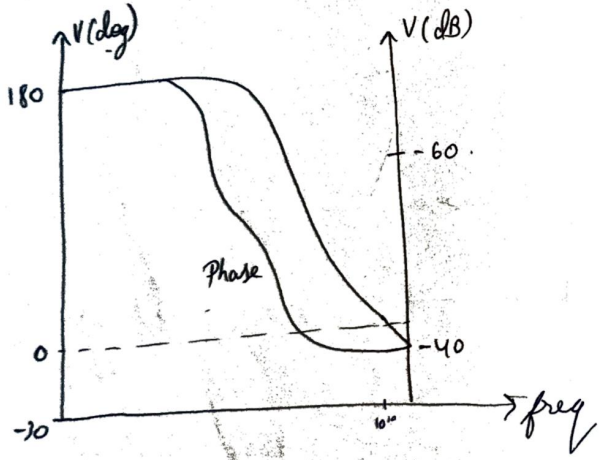
b) DC Response



c) AC Response



d) AC Magnitude and Phase Response



Results:

Two-stage Operational Amplifier		
	Gain	UGB
Schematic	365.6 V	1.16859 GHz - 167.17 mdB
Layout	364.6 V	1.16859 GHz

1. **Write verilog code for 32-bit ALU supporting four logical and four arithmetic operations, use case statement and if statement for ALU behavioral modeling.**
   a. **Perform functional verification using test bench.**
   b. **Synthesize the design targeting suitable library by setting area and timing constraints.**
   c. **For various constraints set, tabulate the area, power and delay for the synthesized netlist.**
   d. **Identify the critical path and set the constraints to obtain optimum gate level netlist with suitable constraints.**

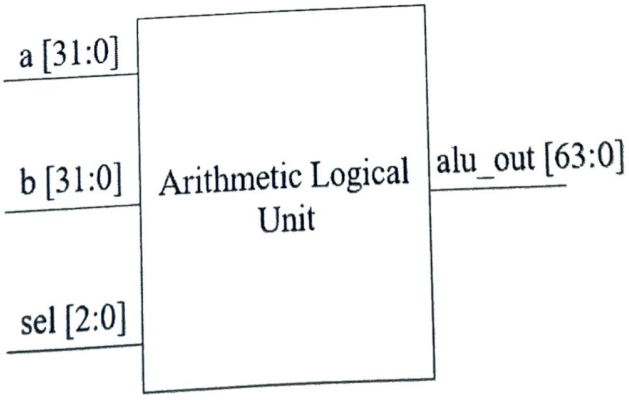
   Compare the synthesize results of ALU modeled using if and case statements.

Tools required:

- *Functional Simulation: Incisive Simulator (ncvlog, ncelab, ncsim)*
- *Synthesis: Genus*

Arithmetic Logical Unit (ALU): *ALU is the fundamental building block of the processor, which is responsible for carrying out the arithmetic and logical functions. ALU comprises of combinational logic that implements arithmetic operations such as addition, subtraction, multiplication, division etc.., and logical operations such as AND, OR, NAND, NOR etc.., The ALU reads two input operands a and b. The operation to perform on these input operands is selected using control input sel. The ALU performs the selected operation on the input operands a and b and produces the output alu_out.*

Arithmetic Logical Unit (ALU) block diagram:



Arithmetic Logical Unit (ALU) truth table:

a	b	sel	operation	alu_out
32'h FEDCBA98	32'h 89ABCDEF	0	Addition	64'h 00000001_88888887
		1	Subtraction	64'h 00000000_7530EC A9
		2	Multiplication	64'h 890F2A50_AD05EBE8
		3	Division	64'h 00000000_00000001
		4	AND	64'h 00000000_88888888
		5	OR	64'h 00000000_FFFFFFFF
		6	XOR	64'h 00000000_77777777
		7	XNOR	64'h FFFFFFFF_88888888

a) 32-bit ALU using case statement:

Verilog code for 32-bit ALU using case statement:

```
module alu (a, b, sel, alu_out);
input [31:0] a, b;
input [2:0] sel;
output reg [63:0] alu_out;
always @ (*)
begin
case (sel)
3'b000: alu_out = a + b;
3'b001: alu_out = a - b;
3'b010: alu_out = a * b;
3'b011: alu_out = a / b;
3'b100: alu_out = a & b;
3'b101: alu_out = a | b;
3'b110: alu_out = a ^ b;
3'b111: alu_out = ~(a ^ b);
default::
endcase
end
endmodule
```

Testbench for 32-bit ALU using case statement:

```
module alu_test;
reg [31:0] a, b;
reg [2:0] sel;
wire [63:0] alu_out;

alu a1 (a, b, sel, alu_out);

initial
begin
a = 32'hFEDCBA98;
b = 32'h89ABCDEF;
sel = 3'b000;
$monitor ("a = 0x%0h  b = 0x%0h  sel = 0x%0h  alu_out = 0x%0h", a, b, sel, alu_out);
#80; $finish;
end

always
#10 sel = sel + 3'b001;

endmodule
```

Creating an SDC File:

➢ In terminal type "gedit alu_top.sdc" to create an SDC file.

```
set_input_delay -max 0.2 [get_ports "a"]
set_input_delay -max 0.25 [get_ports "b"]
set_output_delay -max 0.2 [get_ports "alu_out"]
```

"Follow the same steps for synthesize"

Result:

Non-GUI output:

a = 0xfedcba98 b = 0x89abcdef sel = 0x0 alu_out = 0x88888887

a = " b = " sel = 0x1 alu_out = 0x3530ECA9

a = " b = " sel = 0x2 alu_out = 0x89af2a50

a = " b = " sel = 0x3 alu_out = 0x1a05ebc8

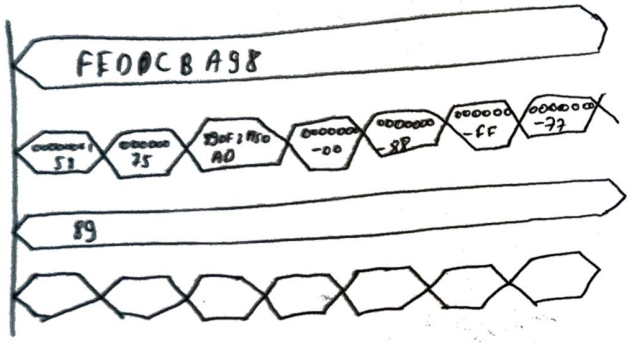
a = " b = " sel = 0x4 alu_out = 0x88888888

a = " b = " sel = 0x5 alu_out = 0xffffffff

a = " b = " sel = 0x6 alu_out = 0x77777777

a = " b = " sel = 0x7 alu_out = 0xffffffff

GUI Output:



Area report:

alu	4733	102100	0	102100.5	none

Power report:

memory	0	0	0	0	0
register	0	0	0	0	0
latch	0	0	0	0	0
logic	3.503e-06	9.850e-03	2.735e-03	1.259e-02	10x
bbox	0	0	0	0	0
clock	0	0	0	0	0
pad	0	0	0	0	0
pin	0	0	0	0	0

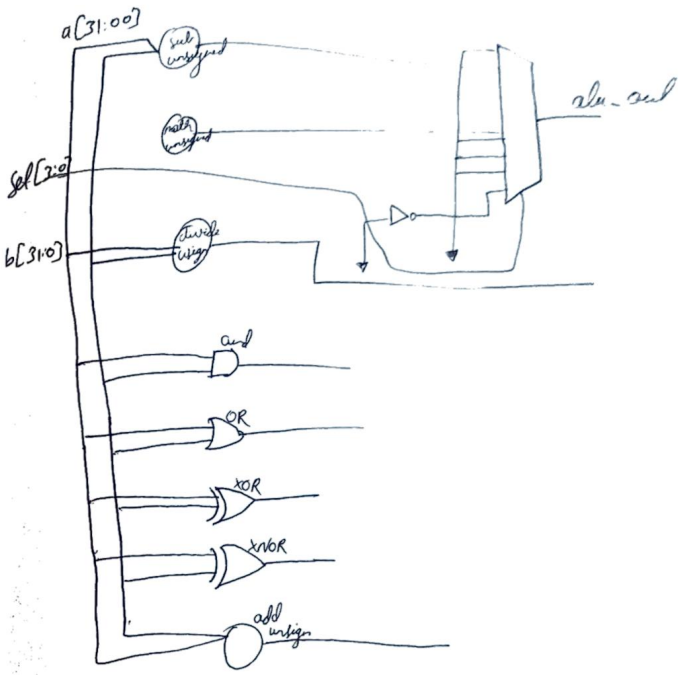
Gates Report:

Inverter	260	1729.728	1.7
logic	4473	100310.794	98.3
physical cell	0	0	0.0

Timing report:

	Reg	Arc	Edge	cell	Fanout	Load	Trans	delay	arrival
alu_out[0]	-	-	F	port	-	-	-	0	1752

Schematic:



b) 32-bit ALU using if statement:

Verilog code for 32-bit ALU using if statement:

```
module alu (a, b, sel, alu_out);
input [31:0] a, b;
input [2:0] sel;
output reg [63:0] alu_out;
always @ (*)
begin
if (sel == 3'b000)
alu_out = a + b;
else if (sel == 3'b001)
alu_out = a – b;
else if (sel == 3'b010)
alu_out = a * b;
else if (sel == 3'b011)
alu_out = a / b;
else if (sel == 3'b100)
alu_out = a & b;
else if (sel == 3'b101)
alu_out = a | b;
else if (sel == 3'b110)
alu_out = a ^ b;
else
alu_out = ~(a ^ b);
end
endmodule
```

Testbench for 32-bit ALU using if statement:

```
module alu_test;
reg [31:0] a, b;
reg [2:0] sel;
wire [63:0] alu_out;
```



```
alu a1 (a, b, sel, alu_out);

initial
begin
a = 32'hFEDCBA98;
b = 32'h89ABCDEF;
sel = 3'b000;
$monitor ("a = 0x%0h  b = 0x%0h  sel = 0x%0h  alu_out = 0x%0h", a, b, sel, alu_out);
#80; $finish;
end

always
#10 sel = sel + 3'b001;

endmodule
```

Result:

Non-GUI output:

a = 0xfedcba98 b = 0x89abcdef sel = 0x0 alu_out = 0x188887

a = " b = " sel = 0x1 alu_out = 0x25dec09

a = " b = " sel = 0x2 alu_out = 0x890f2a50

a = " b = " sel = 0x3 alu_out = 0x105e6e8

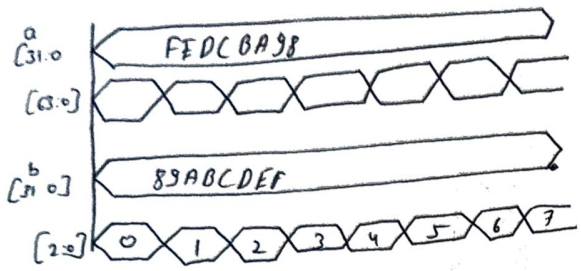
a = " b = " sel = 0x4 alu_out = 0x8883888

a = " b = " sel = 0x5 alu_out = 0xffffffff

a = " b = " sel = 0x6 alu_out = 0x7777777

a = " b = " sel = 0x7 alu_out = 0xffffffff

GUI Output:



Area report:

alu	4737	102093.87	0	102093.870	(none)

Power report:

memory	0	0	0	0	0
reg	0	0	0	0	0
latch	0	0	0	0	0
logic	3.50687e-06	9.839e-03	2.73572e-03	1.257e-02	100
bbox	0	0	0	0	0
clock	0	0	0	0	0
pad	0	0	0	0	0
pm	0	0	0	0	0

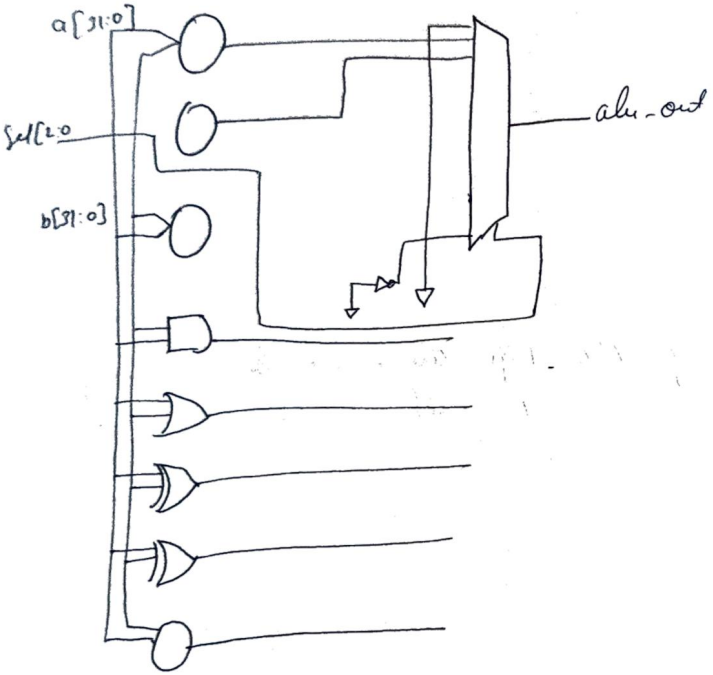
Gates Report:

Inverter	249	1656.547	16
logic	4488	100437.32	98.4
physical cells	0	0.0	0

Timing report:

Timing	Flag	Arc	Edge	all fanout	load	Trans	Delays	Arrival
alu-out[0]	-	-	f	port	-	-	-	0	175742

Schematic:



Compare the results of 32-bit ALU case statement and 32-bit ALU using if statement

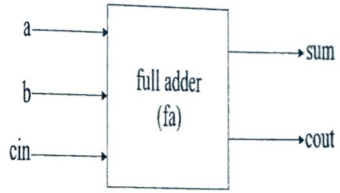
2. Write verilog code for 4-bit adder and verify its functionality using test bench. Synthesize the design by setting proper constraints and obtain netlist. From the report generated identify the critical path, and maximum delay, total number of cells, power requirement and total area required. Change the constraints and obtain optimum synthesis results.

Tools required:

- *Functional Simulation: Incisive Simulator (ncvlog, ncelab, ncsim)*
- *Synthesis: Genus*

Full-Adder: Full Adder is the adder which adds three inputs and produces two outputs. The first two inputs are a and b and the third input is an input carry as cin. The output carry is designated as cout and the normal output is sum.

Full-Adder block diagram:

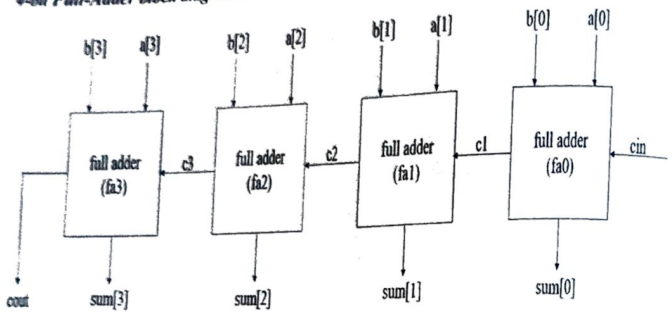


Full-Adder truth table:

Inputs			Outputs	
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

4-bit Full Adder: Binary adders are implemented to add two binary numbers. So in order to add two 4-bit binary numbers, we will need to use 4 full-adders. The 4 full-adders are connected in cascade form. In this implementation, cout of each full-adder is connected to next cin.

4-bit Full-Adder block diagram:



4-bit Full-Adder truth table:

Inputs			Outputs	
a	**b**	**cin**	**sum**	**cout**
4'b0001	4'b1010	0	4'b1011	0
4'b1100	4'b1101		4'b1001	1
4'b0101	4'b1011		4'b0000	1
4'b1111	4'b1111		4'b1110	1
4'b0001	4'b1010	1	4'b1100	0
4'b1100	4'b1101		4'b1010	1
4'b0101	4'b1011		4'b0001	1
4'b1111	4'b1111		4'b1111	1

Verilog code for 1-bit full-adder:

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

endmodule
```

Verilog code for 4-bit full-adder:

```
module four_bit_adder (a, b, cin, sum, cout);
input [3:0] a, b;
input cin;
output [3:0] sum;
output cout;

wire c1, c2, c3;

full_adder fa0 (a[0], b[0], cin, sum[0], c1);
full_adder fa1 (a[1], b[1], c1, sum[1], c2);
full_adder fa2 (a[2], b[2], c2, sum[2], c3);
full_adder fa3 (a[3], b[3], c3, sum[3], cout);

endmodule
```

Test bench for 4-bit full-adder:

```
module test_adder;
reg [3:0] a, b;
reg cin;
wire [3:0] sum;
wire cout;

four_bit_adder f1 (a, b, cin, sum, cout);

initial
begin
```

```
$monitor ("time = %0d", $time, "ns", "a = %0b", a, "b = %0b", b, "cin = %0b", cin, "sum =
%0b", sum, "cout = %0b", cout);
#50 $finish;
end

initial
begin
a = 4'b0011; b = 4'b0011; cin = 1'b0;
#10; a = 4'b1011; b = 4'b1000; cin = 1'b1;
#10; a = 4'b1111; b = 4'b1100; cin = 1'b1;

end

endmodule
```

Creating an SDC File:

➢ In terminal type "gedit adder_top.sdc" to create an SDC file.

```
set_input_delay -max 0.5 [all_inputs]
set_output_delay -max 0.15 [all_outputs]

set_input_transition 0.1 [all_inputs]
set_max_capacitance 2 [get_ports]

set_load 0.10 [all_outputs]
set_max_fanout 20.00 [current_design]
```

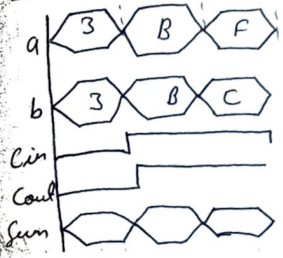
***Follow the same steps for synthesize"**

Result:

Non-GUI output:

time	a	b	Cin	Sum	Cout
0	11	0011	0	110	0
10	1011	1000	1	100	1
20	1111	1100	1	1100	1

GUI Output: 10ms 20ms 30ms



Area report:

fa0	full_adder	1	69.854	0	259.459
	full_adder	2	49.896	0	69.854
fa1	full_adder	1	69.854	0	49.896
fa2	full_adder	1	69.854	0	69.54
fa3	full_adder	1	69.854	0	69.54

Power report:

logic	1.36×10^{-8}	8.37×10^{-6}	2.35×10^{-5}	3.19×10^{-5}	100
subtotal	1.36×10^{-8}	8.37×10^{-6}	2.35×10^{-5}	3.19×10^{-5}	100

Gates Report

ADDFXL	3	29.593	tsmc18
OAI2BB2XL	1	23.255	tsmc18
XNOR2XL	1	26.611	tsmc18

Timing report:

Timing Point	Flag	Arc	Edge	Cell	Fanout	Load	Trans	Delay	Arrival
Cin	-	-	F	arrival	1	6.8	100	6	500
fa0/g72-2398/Co	-	B→Co	F	ADDFX2	3	11.9	162	540	1030
fa1/g43-5107/y	-	Bo→y	R	OA32BB2	1	100.0	2721	1590	2630
Sum[1]	-	-	R	port	-	-	-	0	2630

Schematic: