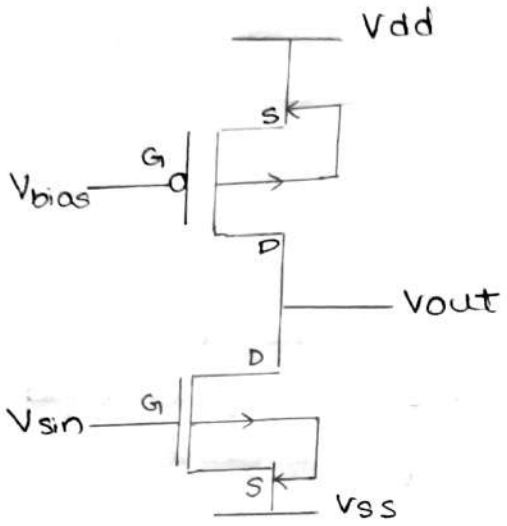
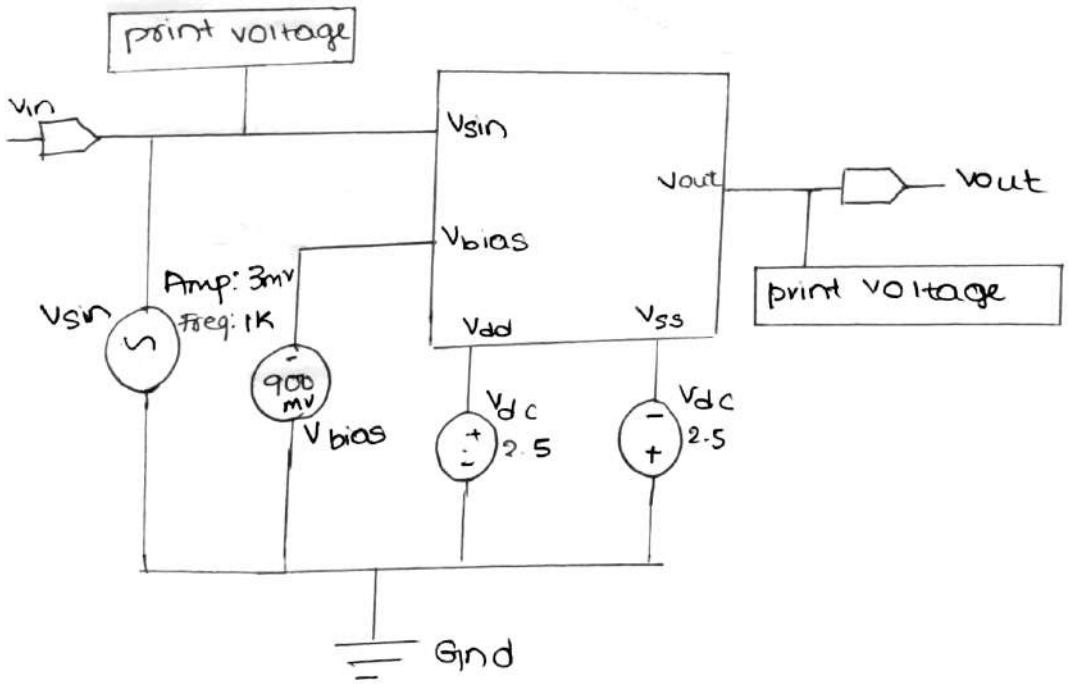
Common source



Test Circuit:



PAGE NO. EXP NO.

Experiment-NO.3

COMMON SOURCE AND COMMON DRAIN

Aim: To draw the schematic of common source and common drain and verify using transient analysis and DC analysis

Software required: Mentor Graphics, tanner EDA tool

theory:

a) common source:

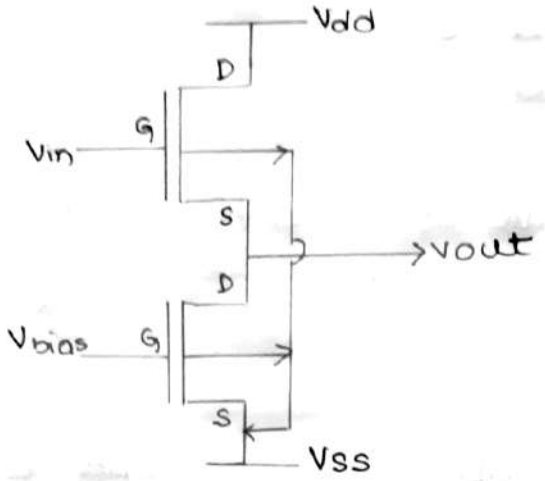
A common source amplifier is one of three basic single-stage field-effect transistor [FET] amplifier topologies, typically used as a voltage (or) transconductance amplifier.

The signal enters through gate, and output is taken from drain. The only terminal remaining is the source hence it is common source FET circuit. Common source is also known as single stage common emitter amplifier Output as phase shift, Output has 180° phase shift of input

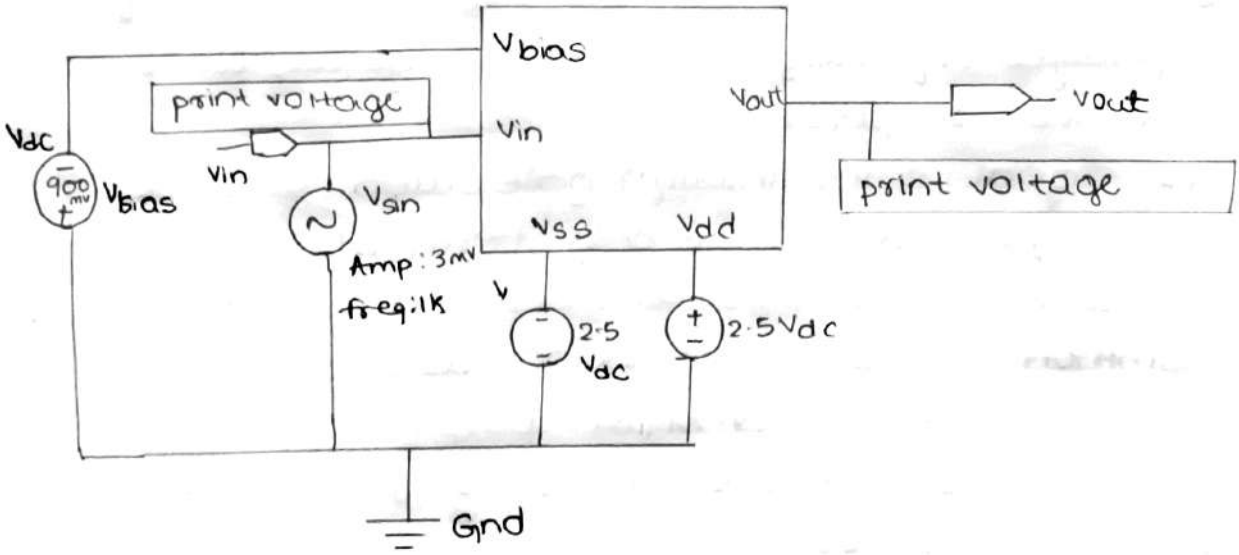
Specification: common source

Transient Analysis	DC Analysis
Stop time: 5ms	
print start time: 0ms	Sweep type: linear step
Max step time: 1uS	start time: -2.5
print step time: 1uS	stop time: 2.5

Common drain:



Test Circuit:



DATE PAGE NO. EXP. NO.

b) Common Drain:

The common drain is a unity gain amplifier. It is also known as source follower. In common drain output follows the input. Application of common drain is, it acts as a buffer.

The signal at the gate is sensed and drives the load at the source which allows the source potential to follow the gate voltage.

The source follower thus worked as a buffer stage.

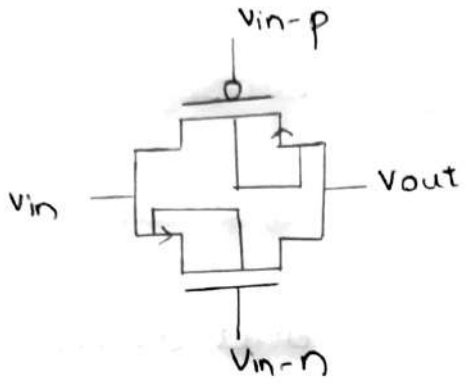
Specification: Common drain.

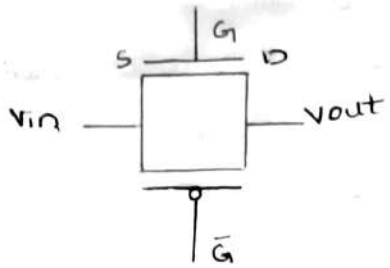
Transient Analysis	DC Analysis
Stop time : 5ms	
Print Start time: 0ms	Sweep type: linear step
Max step time : 1us	start time : -2.5
Print step time: 1us	stop time : 2.5

Result:

To draw the schematic of common source and common drain and verify using transient analysis and DC analysis is done and output waveform is verified.

Transmission gate

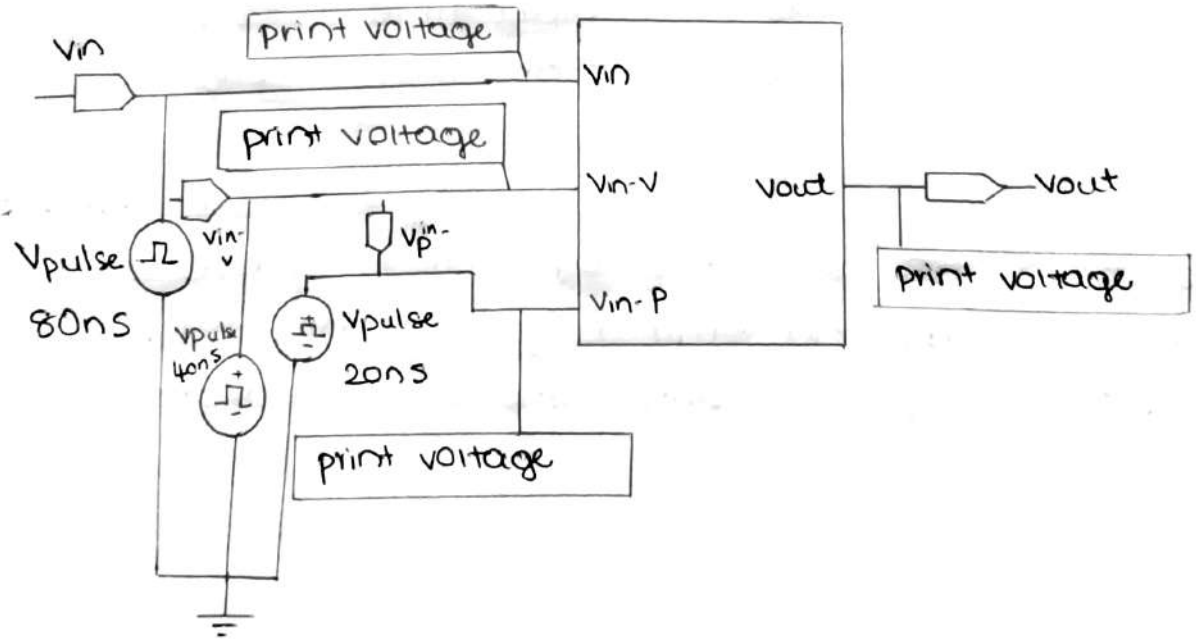




Truth table

A	$\bar{A}$	Vout
1	0	Vin
0	Z	Z

Test Circuit for transmission gate



DATE PAGE NO. EXP NO.

Experiment No. 4
Transmission Gate

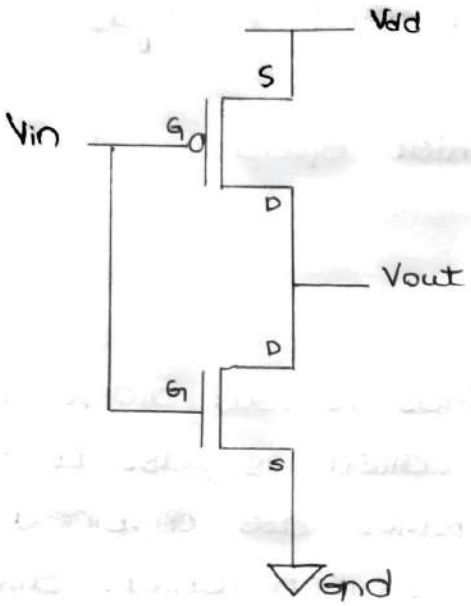
Aim: Draw the schematic of transmission gate and verify using transient Analysis

Software required: Mentor Graphics, tanner EDA tool

theory:
When an NMOS or PMOS is used alone as an imperfect switch, it is called as pass transistors. But when a PMOS and NMOS are combined parallel it becomes a perfect switch which allows both '0's and '1's in any acceptable fashion. This combination is called a transmission gate for the gate to operate, both the control input and it's complement are required and this is called double rail logic.

Result:
To draw the schematic of transmission gate and verify using transient analysis is done and the output wave form is verified

CMOS Inverter



Vin	Vout
0	1
1	0

DATE ..... PAGE NO. ..... EXP NO. .....

Experiment NO: 5

Aim: Write switch level verilog code and verify using test bench

(i) CMOS INVERTER

(ii) 2 input NAND Gate

(iii) CMOS 2 input NOR gate

(iv) CMOS 2 input XOR gate.

Software required: Questa simulator

(i) CMOS INVERTER:

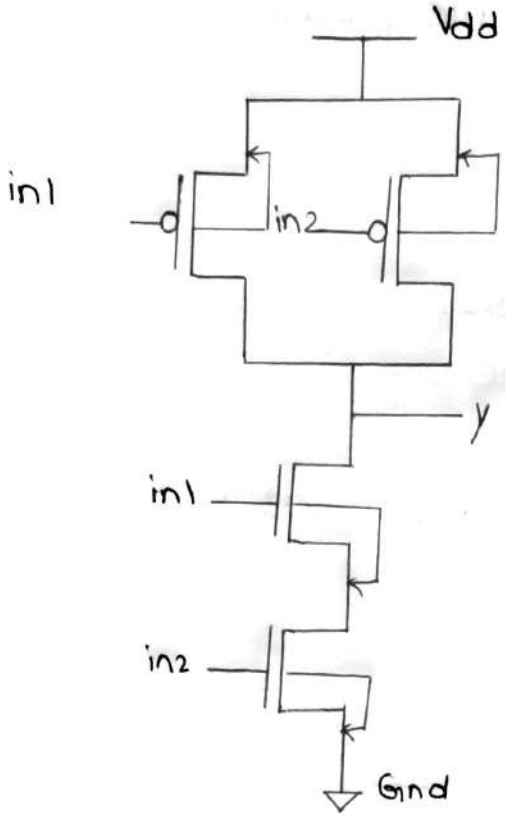
program : Inverter.v

```
Module inverter( Vout, Vin);
input Vin;
Output Vout;
Supply1 pwr;
Supply0 gnd;
pmos (Vout, pwr, Vin);
nmos (Vout, pwrgnd, Vin);
end module
```

TestBench module : Inverter_test.v

```
`timescale 1ns/1ns
module inv_test;
wire Vout;
reg Vin;
inverter i1 (Vout,Vin);
initial
  begin
```

NAND Gate



in1	in2	Out
0	0	1
0	1	1
1	0	1
1	1	0

```
    Vin = 1'b0; #10;
    Vin = 1'b0; #10;
    Vin = 1'bX; #10;
    Vin = 1'bz; #10;
  end
endmodule
```

(ii) CMOS two input NAND Gate:

program: Nandg.v

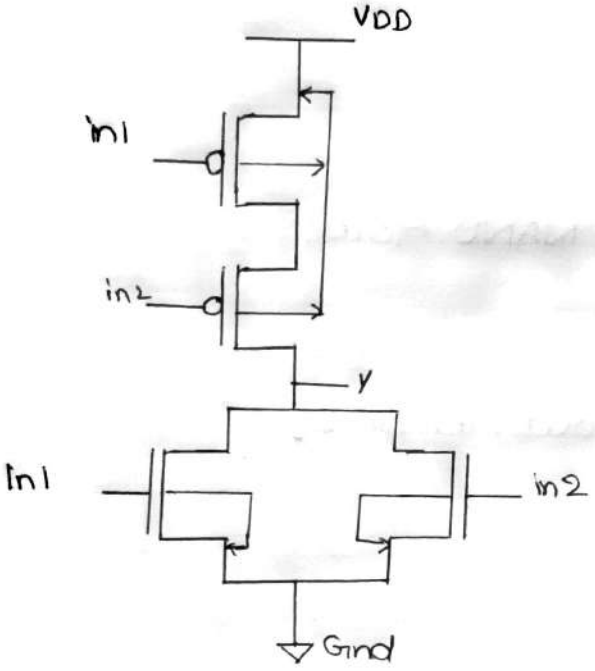
```
`timescale 1ns/1ns
module nandgate (out, in1, in2);
output out;
input in1, in2;
Supply1 pwr;
Supply0 gnd;
wire contact;
pmos (out, pwr, in1);
pmos (out, pwr, in2);
nmos (out, contact, in1);
nmos (contact, gnd, in2);
endmodule
```

Test Bench Module: Nand_test.v

```
`timescale 1ns/1ns
module nand_test;
wire out;
reg in1, in2;
`use nandgate n1 (out, in1, in2);
initial
  begin
    in1 = 1'b0; in2 = 1'b0; #10;
```

NHC

NOR gate.



in1	in2	Out
0	0	1
0	1	0
1	0	0
1	1	0

```
in1=1'b0 ; in2 =1'b1; #10;
in1=1'b1 ; in2=1'b0; #10;
in1=1'b1; in2=1'b1, #10;
end
endmodule
```

(iii) CMOS two NOR gate:

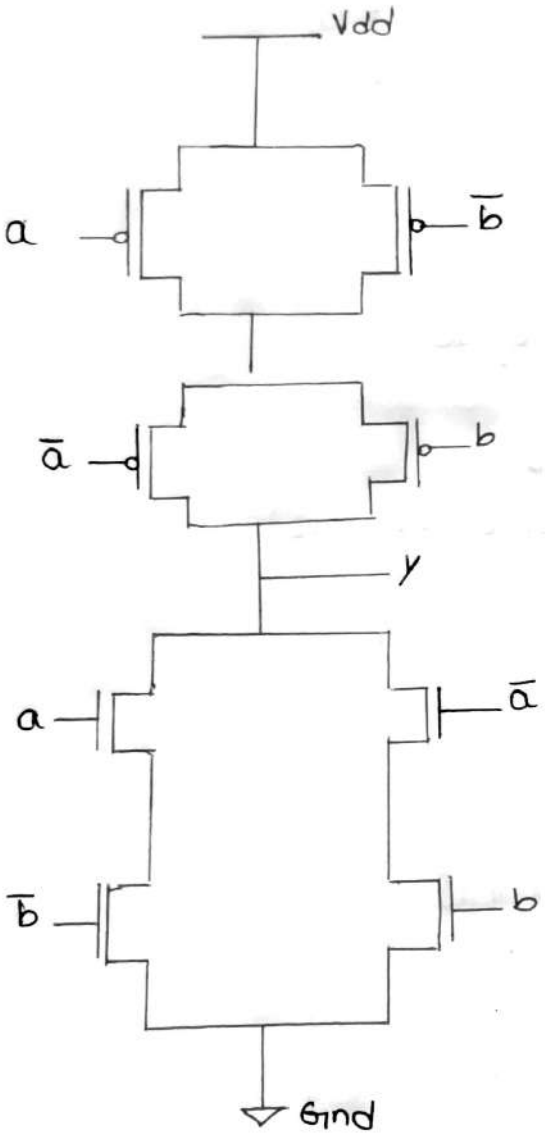
program: Norg.v

```
"timescale 1ns/1ns
module norgate (out, in1, in2);
output out;
input in1, in2;
Supply1 pwr;
Supply0 gnd;
wire contact;
pmos (contact, pwr, in1);
pmos (out, contact, in2);
nmos (out, gnd, in1);
nmos (out, gnd, in2);
endmodule
```

Test bench module: Nor_test.v

```
"timescale 1ns /1ns
module nor_test;
wire out;
reg in1, in2;
norgate n1 (out, in1, in2);
initial
  begin
    in1 =1'b0; in2= 1'b0; #10
    in1 =1'b0; in2 = 1'b1; #10;
```

XOR gate



a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

DATE PAGE NO. EXP NO.

in1 = 1'b1; in2 = 1'b0, #10;
in1 = 1'b1; in2 = 1'b1; #10;
end
endmodule

(iv) EXOR Gate.

program: Exorg.v

```
"timescale 1ns / 1ns
module xorgate (out, a, b);
output out;
input a, b;
supply1 pwr;
supply0 gnd;
wire w1, w2, w3, w4;
pmos p1(w2, pwr, ~b);
pmos p2(out, w2, a);
pmos p3(w2, pwr, ~b);
pmos p4(out, w2, a);
nmos n1(out, w4, b);
nmos n2(out, w3, ~b);
nmos n3(w4, gnd, a);
nmos n4(w3, gnd, ~a);
endmodule
```

Test bench module: Exorg-test.v

```
"timescale 1ns/1ns
module xor_test;
wire out;
reg a, b;
xorgate(out, a, b);
initial
```

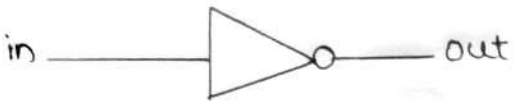
DATE ..... PAGE NO..... EXP. NO.....

```
begin
    a=1'b0; b=1'b0; #10;
    a=1'b0; b=1'b1; #10;
    a=1'b1; b=1'b0; #10;
    a=1'b1; b=1'b1; #10;
 end
endmodule
```

Result:

The switch level verilog code was written and verified using test bench for CMOS inverter, CMOS two input nand gate, nor gate, xor gate.

Inverter.



in	out
0	1
1	0

DATE ..... PAGE NO..... EXP. NO.....

Exp No. 6

Aim: Write the verilog code for the following circuits using Gate level

(i) CMOS inverter

(ii) two-input CMOS NAND gate

(iii) two-input CMOS NOR gate

Software required: Questa Simulator

(i) CMOS Inverter:

program:

```
module inverter (in, out);
input in;
Output out;
not n1(out,in);
end module
```

Test Bench:

```
`timescale 1ns/1ns
module inv_test;
wire out;
reg in;
inverter i1(in, out);
initial
begin
in = 1'b0; #10;
in = 1'b1; #10;
end
endmodule
```

NOR gate

Vin1	Vin2	Vout
0	0	0
0	1	0
1	0	0
1	1	1

DATE : PAGE NO. EXP NO.

(iii) 2- Input NOR gate

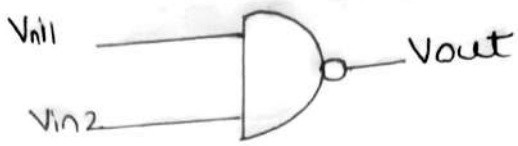
program:

```
module norgate (vin1, vin2, vout);
input vin1, vin2;
Output vout;
NOR n3(vout, vin1, vin2);
endmodule
```

Testbench:

```
`timescale 1ns/1ns;
module NOR_test;
wire vout;
reg vin1, vin2;
NOR gate i3(vin1, vin2, vout);
initial
begin
vin1 = 1'b0 ; vin2 = 1'b0 ; #10;
vin1 = 1'b0 ; vin2 = 1'b1 ; #10;
vin1 = 1'b1 ; vin2 = 1'b0 ; #10;
vin1 = 1'b1 ; vin2 = 1'b1 ; #10;
end
endmodule
```

NAND Gate



Vin1	Vin2	Vout
0	0	1
0	1	1
1	0	1
1	1	0

DATE ..... PAGE NO. ..... EXP NO. .....

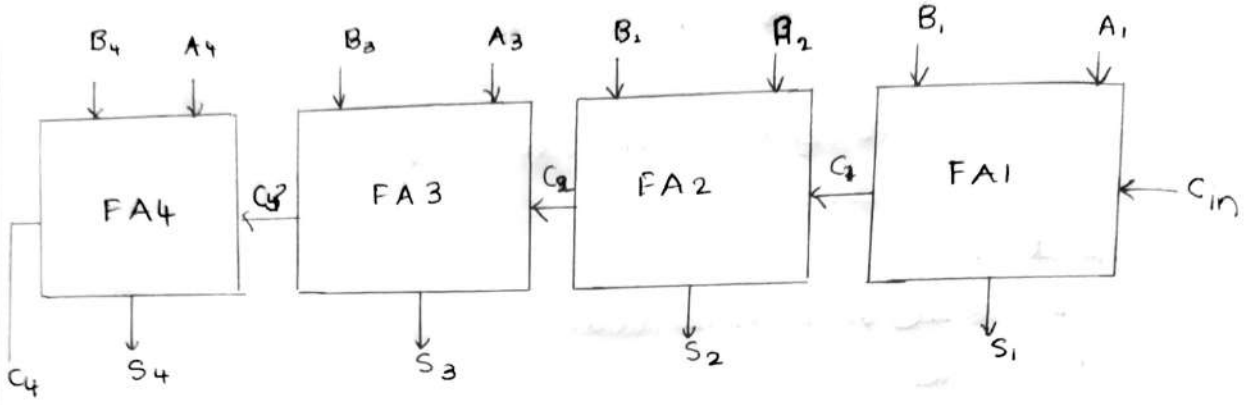
(iii) 2 - Input NAND gate

program:

```
module NAND gate (Vin1, Vin2, Vout);
input Vin1, Vin2;
Output Vout;
NAND N2(Vin1, Vin2, Vout);
endmodule

Testbench;
`timescale 1ns/1ns;
module NAND_test;
wire Vout;
reg Vin1, Vin2;
NAND gate i2 (Vin1, Vin2, Vout);
initial
begin
Vin1 = 1'b0; Vin2 = 1'b0; #10;
Vin1 = 1'b0; Vin2 = 1'b1; #10;
Vin1 = 1'b1; Vin2 = 1'b0; #10;
Vin1 = 1'b1; Vin2 = 1'b1; #10;
end
endmodule
```

Result: The gate level verilog code was written and verified using test bench for CMOS inverter, CMOS NAND gate and NOR gate.


B4
A4
B3
A3
B2
B1
B1
A1
FA4
FA3
FA2
FA1
Cin
C4
S4
S3
S2
S1

DATE PAGE NO. EXP NO.

Exp NO: 07

Aim: For the following circuits write the verilog code. verify the test bench and synthesis with the given constraits.

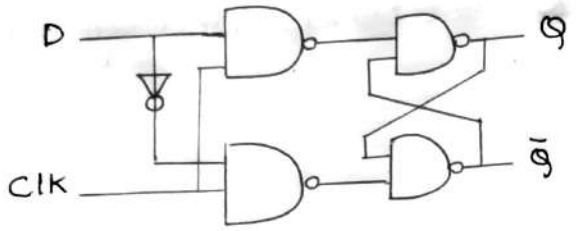
(i) 4 bit parallel adder
(ii) D- flipflop
(iii) T- Flipflop
(iv) 4-Synchronous counter

Software required: Questa simulator.

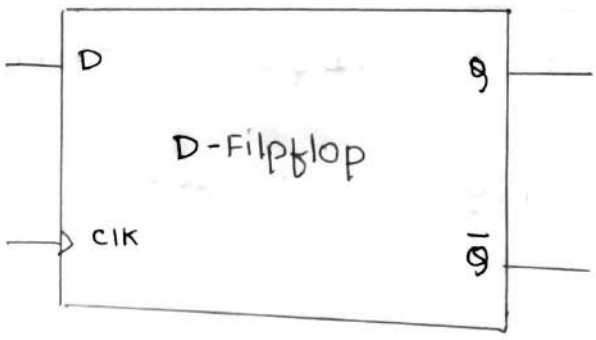
program:

```
module fulladder (a, b, cin, s, cout);
input a, b, cin;
output s, cout;
assign s = a^b^cin;
assign cout = (a & b) | (b & cin) | (a & cin);
endmodule
module padder (x, y, Cin, sum, Cout);
input [3:0] x, y;
input Cin;
Output [3:0] sum;
Output Cout;
wire C1, C2, C3;
fulladder stage0 (x[0], y[0], Cin, sum[0], C1);
fulladder stage1 (x[1], y[1], C1, sum[1], C2);
fulladder stage2 (x[2], y[2], C2, sum[2], C3);
full adder stage3 (x[3], y[3], C3, sum[3], Cout);
endmodule
```

D- Flipflop



D	Q	$\bar{Q}$	state.
0	0	1	Reset
1	1	0	set



DATE : PAGE NO. EXP NO.

Test bench:

```
module padder_test;
reg [3:0] x,y;
reg cin;
wire [3:0]sum;
wire cout;
padder a1(x,y,cin,sum,cout);
initial
begin
x = 4'b0000; y=4'b0000; cin=1'b0;
#20 x=4'b1111; y=4'b1010;
#40 x=4'b0100; y=4'b0110;
#50 $finish
```

(ii) D-Flipflop and T-Flipflop

The D-flipflop is widely used. It is known as a 'data' or 'delay' flipflop. The D flip-flop captures the value of the D-input as a define portion of the clock. The captured value becomes the Q output. At other times, at output Q does not change. The D-flipflop can be viewed as a memory cell, a zero order hold, or a delay time

In T-flipflop, if the T input is high, the T-flipflop changes states ("toggles") whenever the clock input is stored. if the T-input is low, the flipflop holds the previous value.

DATE ..... PAGE NO. ..... EXP. NO. .....

D Flip-flop.

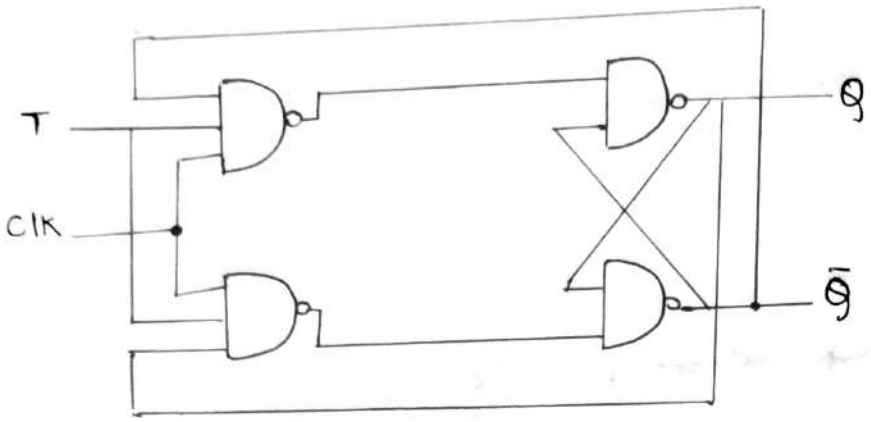
program:

```
module dff(q,qbar,d,clk,rst);
output q,qbar;
input clk,d,rst;
reg tq;
always @(posedge clk or posedge rst)
begin
if(rst)
 tq<=1'b0;
else
tq<=d;
end
assign q=tq;
assign qbar=~tq;
endmodule
```

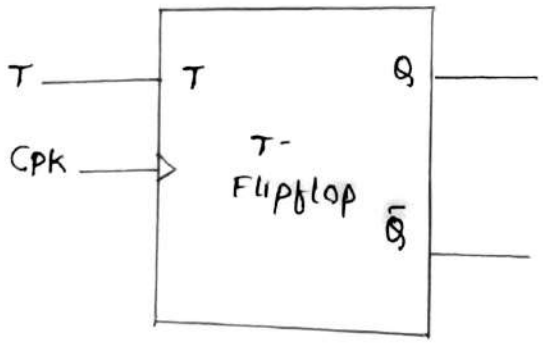
Test bench:

```
module dff_test;
reg clk,d,rst;
wire q,qbar;
dff d1(q,qbar,d,clk,rst);
initial
clk=1'b0
always
#10 clk=~clk;
initial
begin
```

T- Flipflop



T	Q	Q'	state
0	Q	Q'	Nochange
1	Q'	Q	toggle's



```
rst = 1'b1;
d = 1'b0;
#15 rst = 1'b0;
#25 d = 1'b1;
#20 d = 1'b0;
#20 d = 1'b1;
end
initial
# 110 $finish
endmodule
```

T-Flipflop.

program:

```
module tff(q,qbar,t,clk,rst);
output q,qbar;
input clk,t,rst;
reg tq;
always @(posedge clk or posedge rst)
begin
if(rst)
 tq <= 1'b0;
else
begin
 if(t)
  tq <= ~tq;
end
end
assign q = tq;
```

DATE ..... PAGE NO. ..... EXP. NO. .....

```
assign qbar = ~tq;
endmodule
Test Bench:

module T_ff_test;
reg clk, t, rst;
wire q, qbar;
tff t1(q,qbar,t,clk,rst);
initial
clk = 1'b0;
always
#10 clk = ~clk;
initial
begin
rst = 1'b1; t=1'b0;
#15 rst = 1'b0;
#25 t = 1'b1;
#300t = 1'b0;
#  200 t = 1'b1;
#  200 rst = 1'b1;
end initial
#800 $finish
endmodule
```

rst	clk	output
0	1	0000
1	0	0000
0	1	0001
0	1	0010
0	1	0011
0	1	0100
0	1	0101
0	1	0110
0	1	0111

rst	clk	output
0	1	1000
0	1	1001
0	1	1010
0	1	1011
0	1	1100
0	1	1101
0	1	1110
0	1	1111
0	1	0000

DATE ..... PAGE NO. ..... EXP. NO. .....

(iii) 4-bit synchronous counter.

Synchronous counter, the external clock signal is connected to the clock input of every individual flip-flop within the counter so that all of the flip-flops are clocked together simultaneously at the same time giving a fixed time relationship In other words, changes in the output occur in "Synchronisation" with the clock signal. A 4bit synchronous counter counts sequentially on every clock pulse the resulting outputs count upwards from 0 (0000) to 15 (1111)

program:

```
module s_counter (clk, reset, count);
input wire clk, reset;
out reg [3:0] count;
always @ (posedge clk)
if (reset)
count <= 4'b0000;
else
 count <= count + 4'b0001;
endmodule
```

test bench:

```
module s_counter_test;
reg clk, reset;
wire [3:0] count;
initial
```

```
clk = 1'b0;
always
#5 clk = ~clk;
S_counter m1(clk, reset, count);
initial
begin
 reset = 1'b0;
 #15 reset = 1'b1;
#30 reset = 1'b0;
#220 reset = 1'b1;
# 15 $finish
end
endmodule
```

Result: The verilog code written and verified for 4-bit parallel adder, T-Flipflop, D-Flipflop and 4-bit synchronous counter.