EXPERIMENT - 1

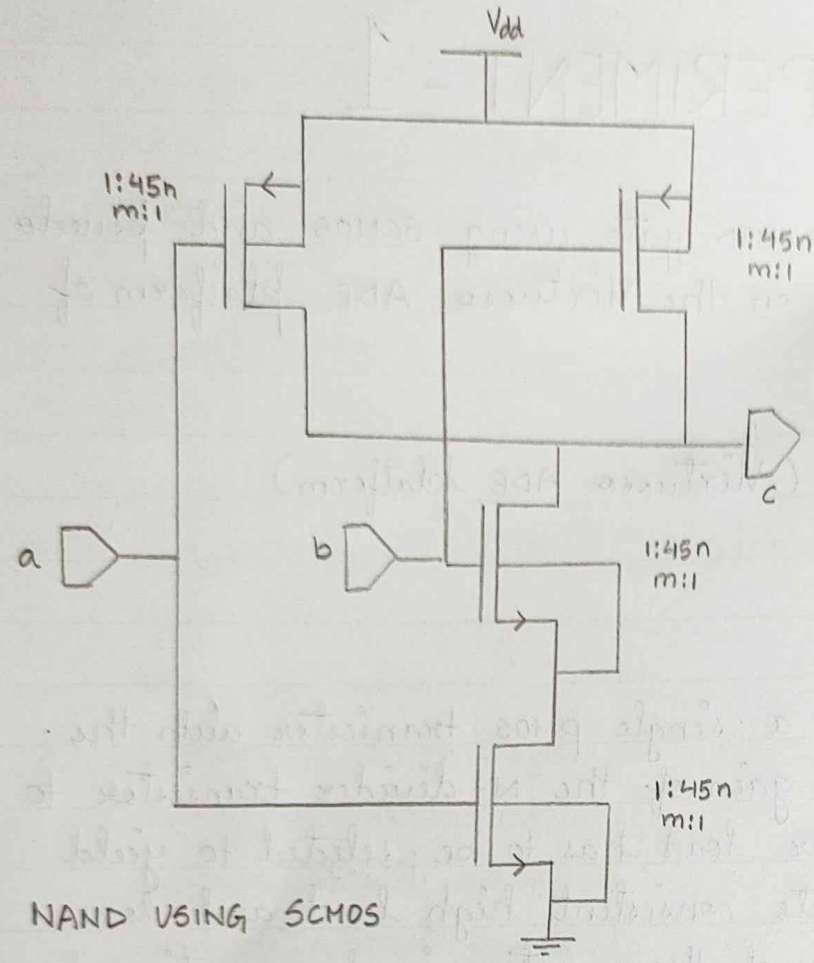
AIM - Design a 2 input NAND gate using SCMOS and pseudo nMOS logic family on the Virtuoso ADE platform of Cadence.

SOFTWARE USED - Cadence (Virtuoso ADE platform)

THEORY -

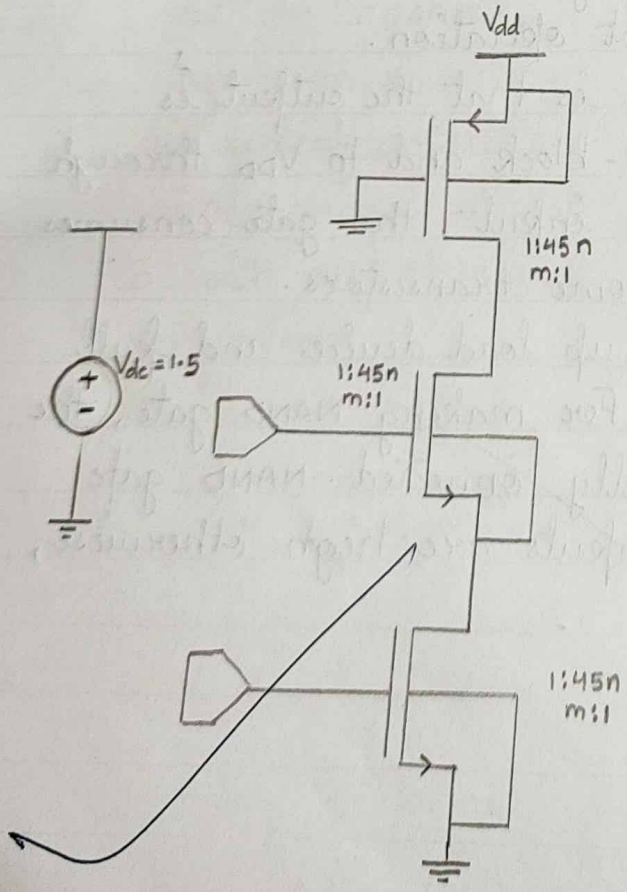
A pseudo nMOS logic is a single pMOS transistor with the gate connected to V_{SS}. The gain of the N-device transistor to the p-device transistor load has to be selected to yield sufficient gain to generate consistent high level and low level logic. The design of these gates involves rationed transistor sizes to ensure correct operation.
The principle of static CMOS logic is that the output is connected to ground through a n-block and to V_{DD} through p block, without charges of the input, the gate consumes only the leakage currents of some transistors.
pMOS & nMOS are used as pull up load device and pull down load device respectively. For making NAND gate, the two nMOS transistors are serially connected. NAND gate becomes low only when both inputs are high otherwise, it remains high.

For NAND gate $Y = \overline{A \cdot B}$



NAND USING SCMOS

STIMULI	a	b
voltage 1	1.2	1.2
voltage 2	0	0
period	100n	100n
delay time	20n	1n
rise time	1n	1n
fall time	1n	1n
pulse width	50n	50n
stop time	500n	500n

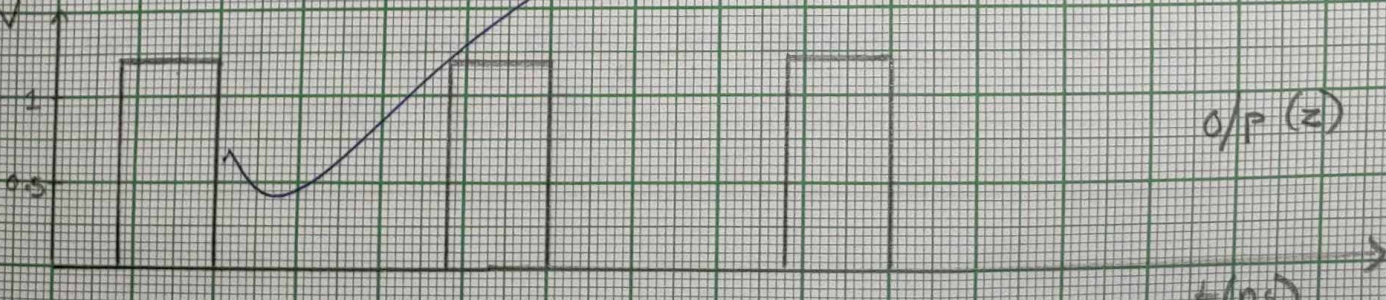


NAND USING PSUEDO nMOS

STIMULI	a	b
Voltage 1	1.2	1.2
Voltage 2	0	0
Period	100n	100n
Delay time	20n	1n
Rise time	1n	1n
Fall time	1n	1n
Pulse width	50n	50n
Stop width	500n	500n

Observation for Nand Gate

Observation for Pseudo Nand



PROCEDURE -

1. Start the Cadence software.
2. Create new Cell view in your pre-created library or first create a library, then create new cell view under this library.
3. Click on "add instance" tab and take all components from the analog library.
4. For selecting nMOS and pMOS, select NMOS4 & PMOS4 from analog library respectively.
5. Give input using pin.
6. After completing design, check and save the same and execute, using ADE tool according to the instruction given by instructor.

RESULT -

A 2 input NAND gate using SCMOS and pseudo nMOS logic family was successfully designed on Cadence and the obtained output was plotted in the graph as shown.

PRECAUTIONS -

1. Make sure all the connections are correct.
2. W/L ratio must be choosen carefully.
3. Do not forget to give delay in between inputs so as to obtain all possible cases of input.

11/02/19

EXPERIMENT - 2

AIM - Design a 4-bit full adder using Verilog and VHDL on Active HDL / Xilinx ISE platform.

SOFTWARE USED - Xilinx ISE 10.1

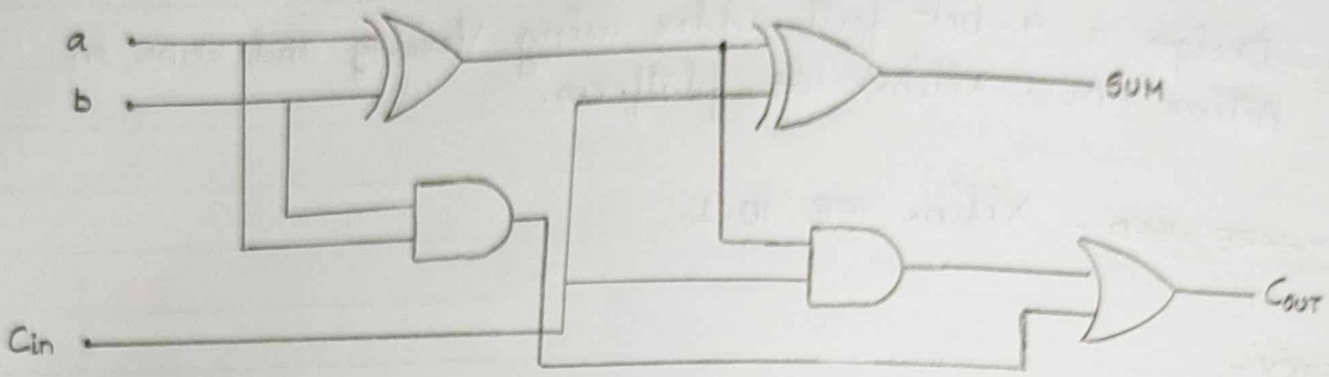
THEORY -

A full adder is a combinational circuit that performs the addition of three input bits. For this, the circuit needs three binary inputs and two binary outputs. A combinational circuit that performs an addition of two 4-bit binary numbers in parallel is called a 4-bit full adder. The circuit can be implemented using four 1 bit full adders. The bits of the input binary numbers designate the angend and addend bits whereas the output binary number is the 4 bit sum and 1 bit carry.

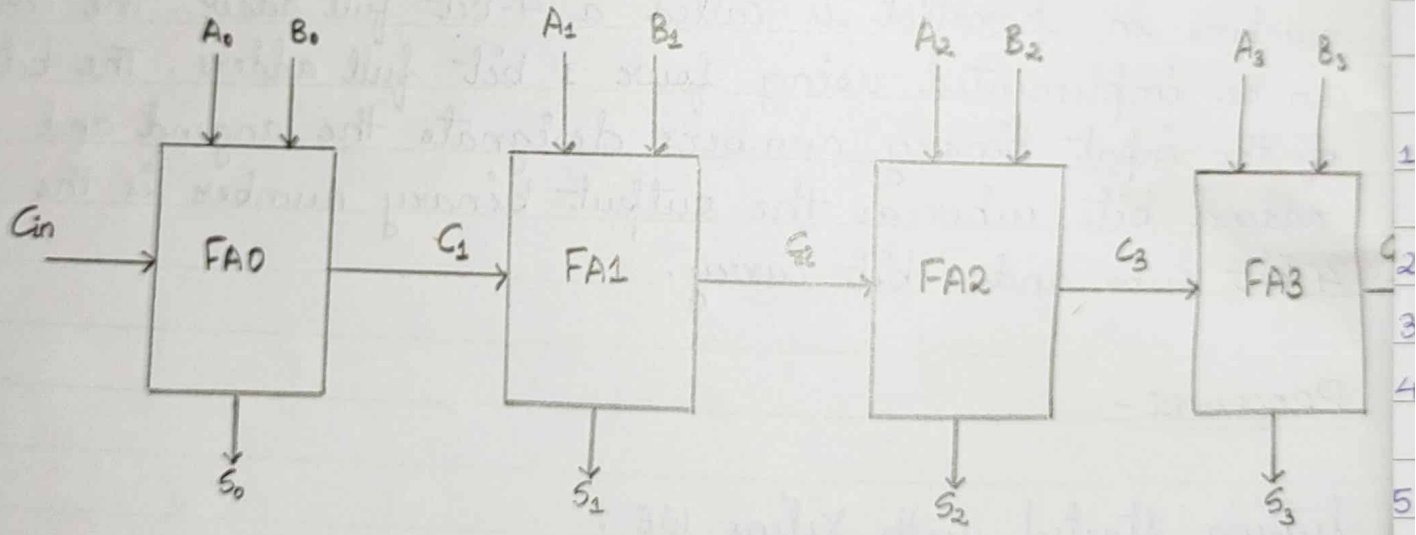
PROCEDURE -

Getting Started with Xilinx ISE :

1. Click on Xilinx ISE icon.
2. Go to the file menu. Click on close project.
3. Again go to the file menu and click on New project.
   a) New project wizard will come.
   b) Give the project name and then press Next to finish.
   c) Now, go to create new source.



GATE LEVEL DIAGRAM FOR FULL ADDER



BLOCK DIAGRAM : 4 BIT CARRY FULL ADDER

d) A source wizard will appear. Select VHDL module and give the file name in the blank, this name will be the entity name in the program.
e) Again press next and then define ports in the blank space
f) Select input and output port for the given port name.
g) Press next and then finish.

4. The VHDL file opens. Write commands after begin and before end.
5. Check that the tab at the left sources for: is there in Synthesis/ Implementation.
6. Click on Synthesize XST.
7. After successful completion click on view schematics.
8. Change the source for: to Behavioural.

Simulation:

1. Go to create new source in the left window. Click on file and select create source option.
2. Select Test Bench Waveform, give its file name.
3. Select the combinational clock option.
4. On the test bench waveform, click on the horizontal lines to change input of the input port and then save the file.
5. Now go to Xilinx ISE simulator and simulate the behavioural mode and finally check the output.

Codes : VHDL

Code for implementing one bit full adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity FA1bit is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           cin : in STD_LOGIC;
           sum : out STD_LOGIC;
           cout : out STD_LOGIC);
end FA1bit;

architecture Behavioral of FA1bit is
begin
sum <= a xor b xor cin;
cout <= (a and b) or (cin and (a xor b));
end behavioral;
```

Code for implementing 4 bit full adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity FA4bit is
    Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
           b : in STD_LOGIC_VECTOR (3 downto 0);
```

```
        cin : in STD-LOGIC;
        S : out STD-LOGIC-VECTOR (3 downto 0);
        cout : out STD-LOGIC);
end FA4bit;

architecture Structural of FA4bit is
signal C1, C2, C3 : STD-LOGIC;
component FA1bit is
    Port ( a : in STD-LOGIC;
           b : in STD-LOGIC;
           cin : in STD-LOGIC;
           sum : out STD-LOGIC;
           cout : out STD-LOGIC );
end component;
begin
FA0 : FA1bit PORT MAP (a(0), b(0), cin, S(0), C1);
FA1 : FA1bit PORT MAP (a(1), b(1), C1, S(1), C2);
FA2 : FA1bit PORT MAP (a(2), b(2), C2, S(2), C3);
FA3 : FA1bit PORT MAP (a(3), b(3), C3, S(3), cout);
end Structural;
```

Codes : Verilog

Code for implementing one bit full adder

```
module FA-1bit (
    input a,
    input b,
    input cin,
    output sum,
    output cout
);
```

```
assign sum = a^b^cin;
assign cout = (a&b)|(cin&(a^b));
endmodule
```

Code for implementing 4 bit full adder

```
module FA_4bit(
    input [3:0] a,
    input [3:0] b,
    output [3:0] s,
    input cin,
    output cout
);

wire c1, c2, c3;
FA_1bit FA0 (a[0], b[0], cin, s[0], c1);
FA_1bit FA1 (a[1], b[1], c1, s[1], c2);
FA_1bit FA2 (a[2], b[2], c2, s[2], c3);
FA_1bit FA3 (a[3], b[3], c3, s[3], cout);
endmodule
```

RESULT - A 4 bit full carry full adder is implemented using VHDL and Verilog on Xilinx ISE platform and the resulting waveform is obtained and is plotted on the graph.

PRECAUTION - Software should be used carefully.

11/02/19

EXPERIMENT - 3

AIM - To draw the layout of 2 input CMOS NAND gate and 2 input CMOS NOR gate on microwind 3.1 by the direct translation of their schematics

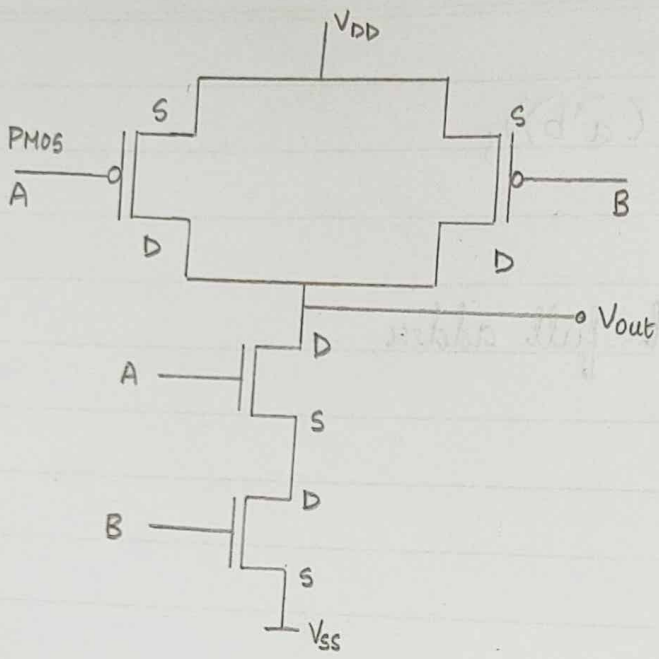
APPARATUS REQUIRED - Microwind 3.1

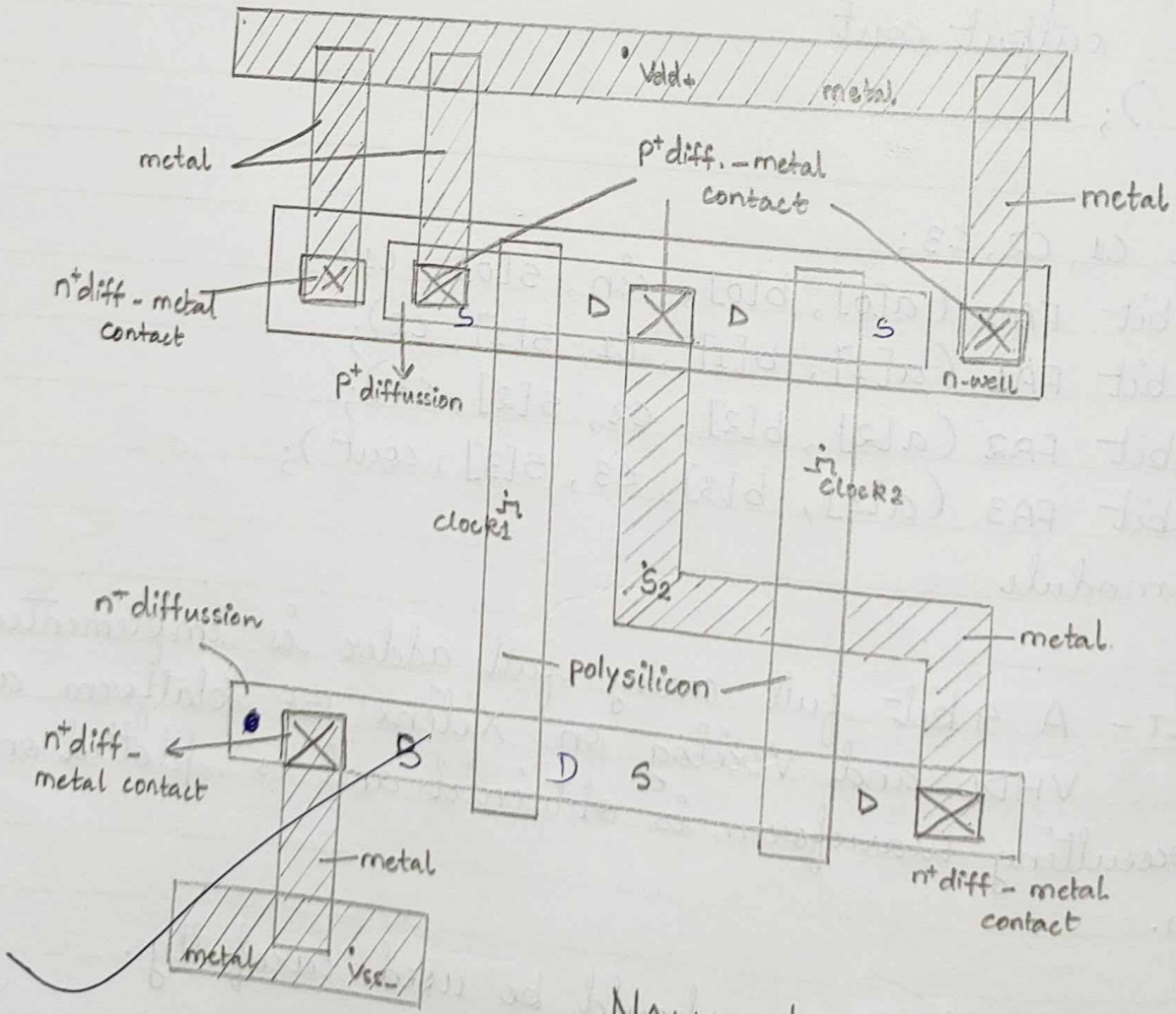
THEORY -

A two-input CMOS NAND gate consists of two p-type units in parallel and two n-type units in series. If all inputs are high, both p-channel transistors turn off and both n-channel transistors turn on. The output has a low impedance to ground and produces a low state. If any input is low, the n-channel transistor is turned off and p-channel transistor is turned on. The output is coupled to V_{DD} and goes to the high state.

A two input NOR gate consists of two n-type units in parallel and two p-type units in series. When all inputs are low, both p-channel units are on and both n-channel units are off. The output is coupled to V_{DD} and goes to the high state. If any input is high, the associated p-channel transistor is turned off and the associated n-channel transistor turns on. This connects the output to ground, causing a low-level output.

2 INPUT CMOS NAND GATE





NAND LAYOUT

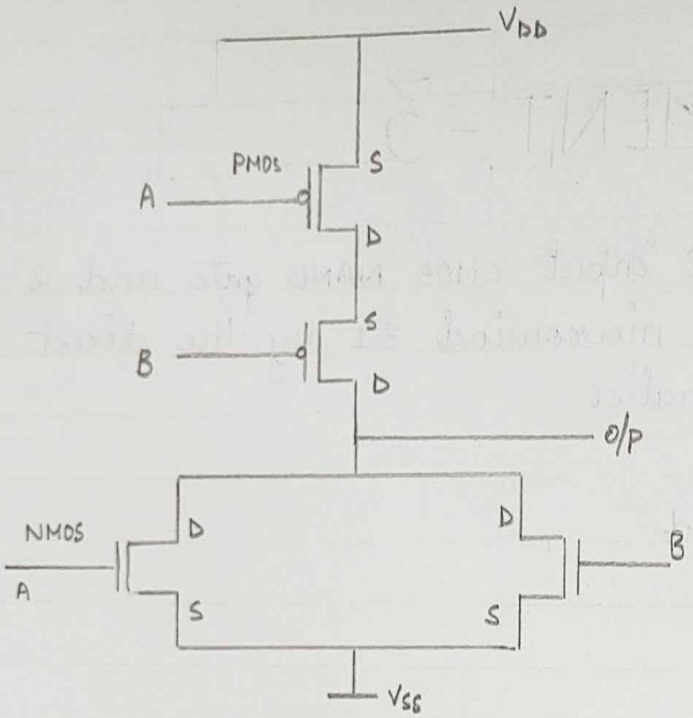
PROCEDURE –

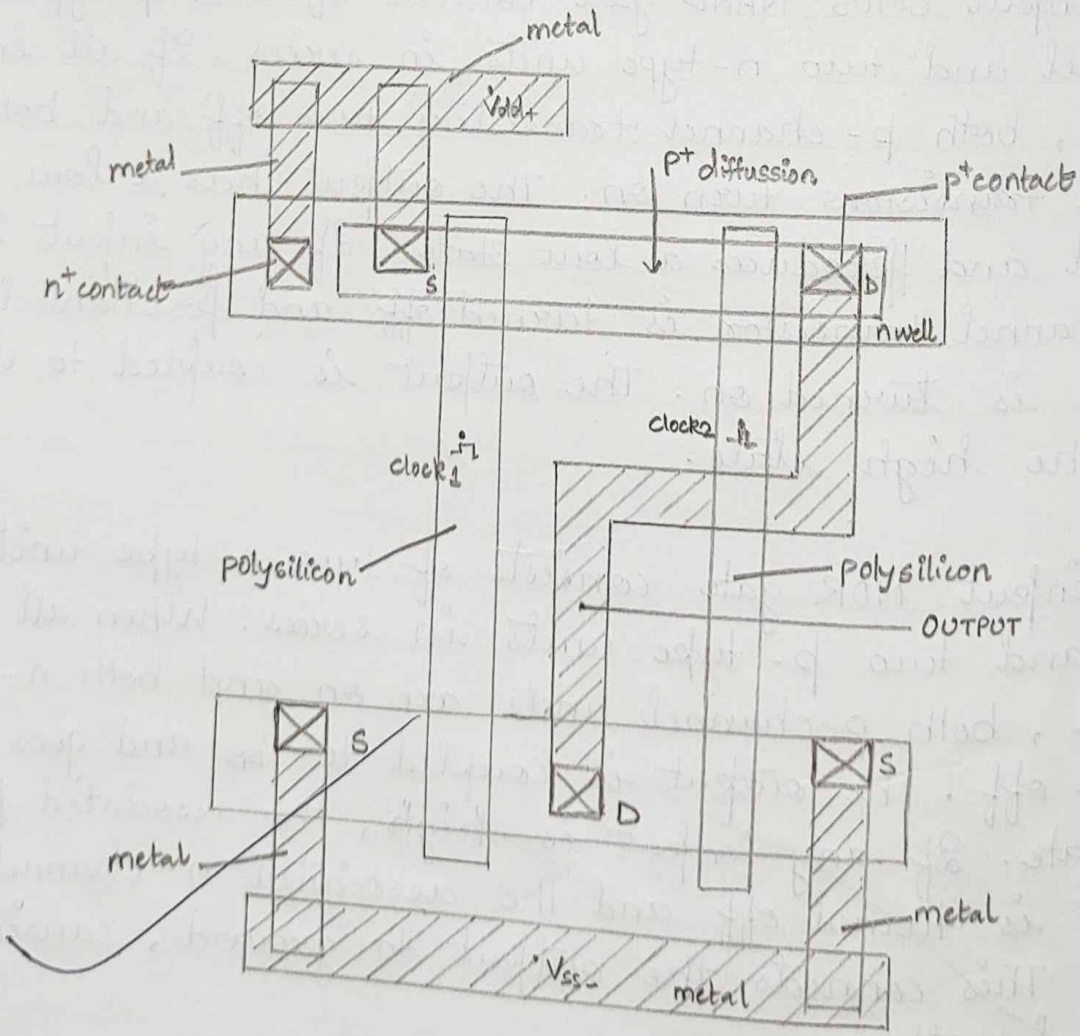
Click on microwind 3.1.
The microwind 3.1 will appear on the screen.
For both the 2-input CMOS NAND and NOR gate, we select n-well from the palette and make a rectangle shape on the p-substrate.
Select p^+ diffusion over the n-well.
Again select n^+ diffusion which is placed below the n well.
By selecting polysilicon, we make gate for both the p^+ and n^+ diffusion.
According to the CMOS NAND gate, we select metal-1 and make it connected to V_{DD} & Vss.
The polysilicons are connected with two clocks A & B respectively.
We now select the appropriate contact for the metal-1.
The output is also connected.
Repeat steps upto 6 for CMOS NOR gate.
Using DRC, we get errors. The errors are removed and the circuit is simulated.

ERRORS :

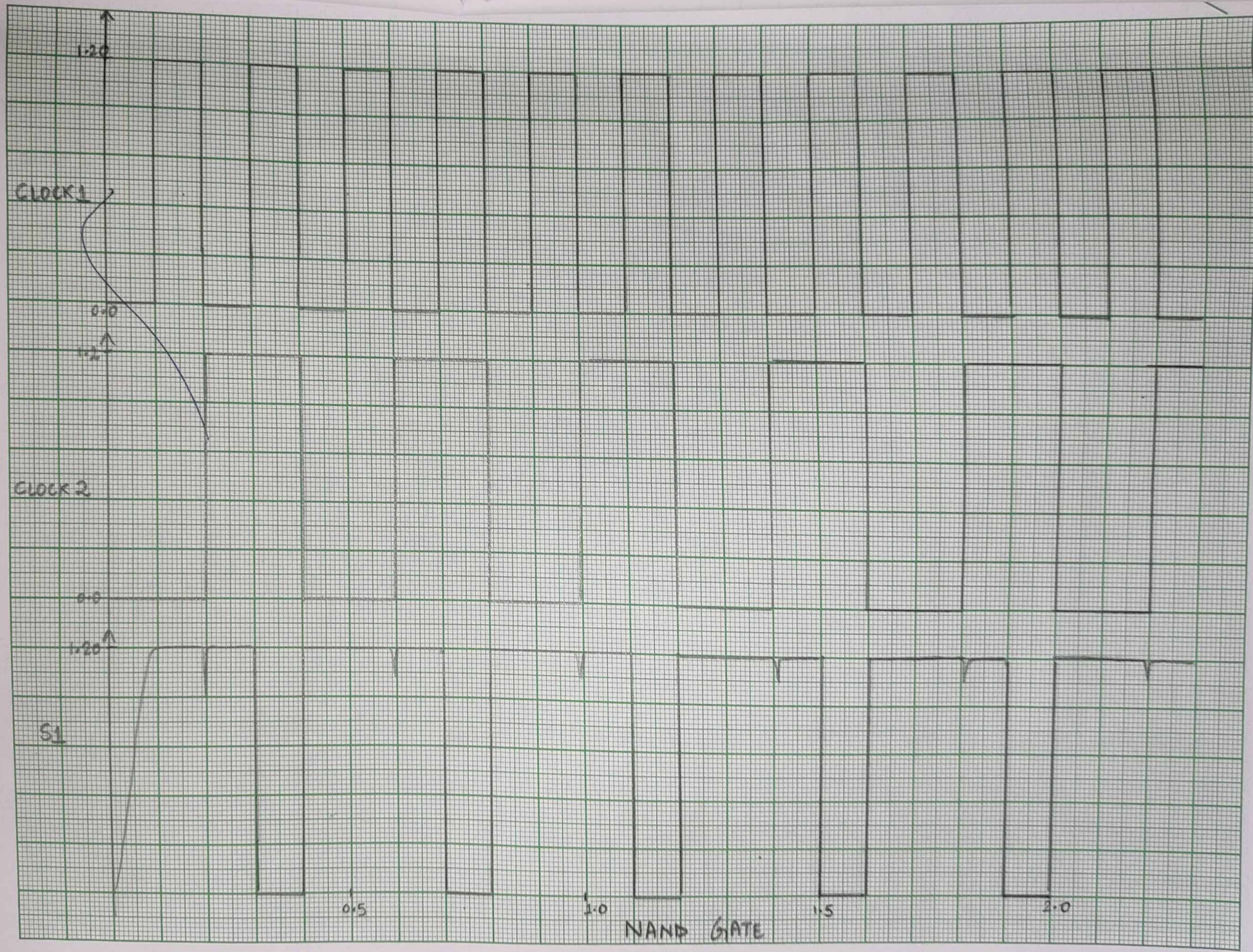
1. The n well width is less than 10 lambda (r101).
2. The spacing between diff p is less than 4 lambda (r201).
3. The diff p width is less than 4 lambda (r201).
4. The extra n well surrounding diff p is less than 6 lambda.
5. The extra poly surrounding diff p is less than 3 lambda (r307)
6. The metal width is less than 3 lamba.
7. The diff n width is less than 4 lambda (r201)
8. The spacing between metal is less than 4 lambda (r501).

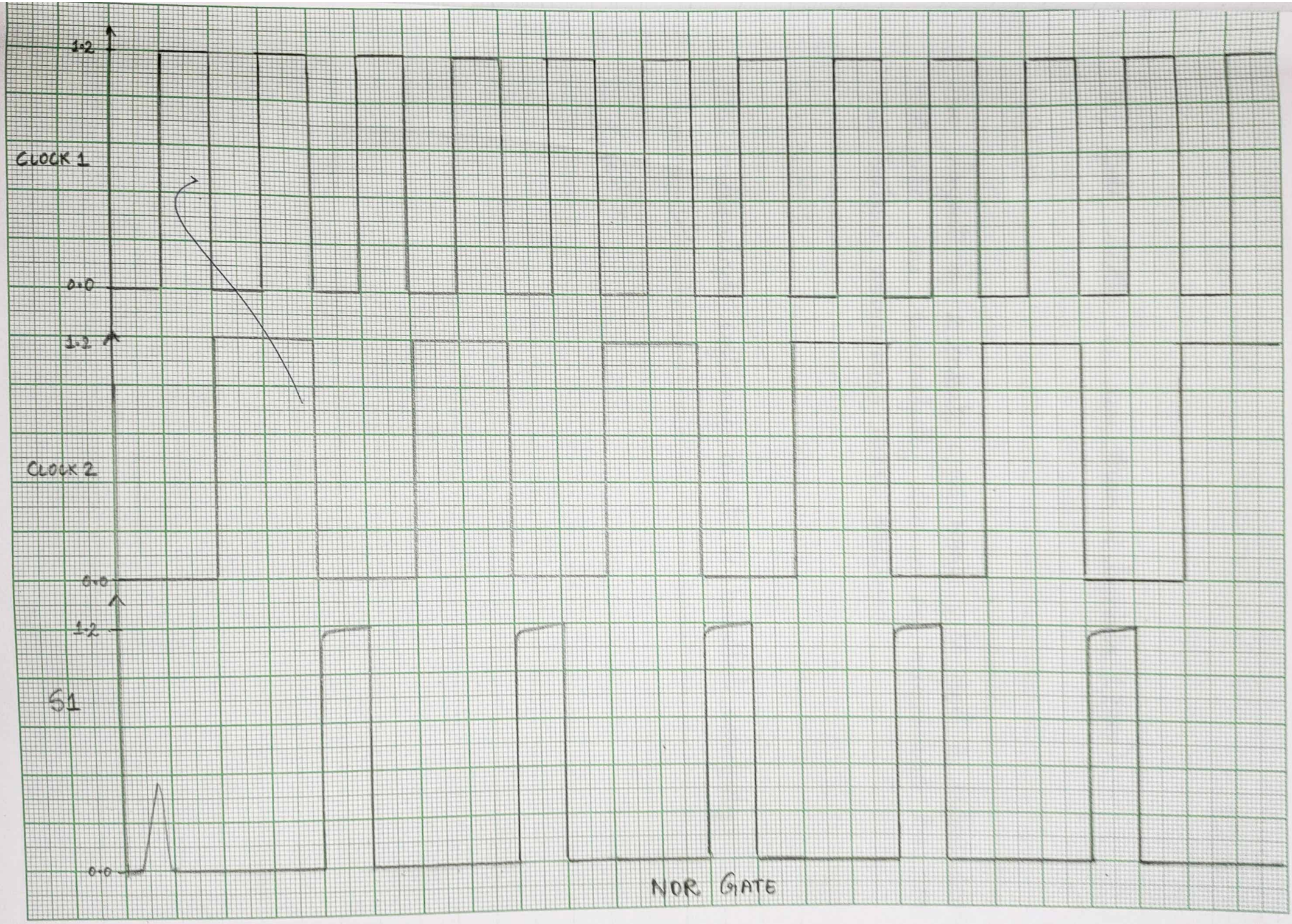
2 INPUT CMOS NOR GATE





NOR LAYOUT


1.20
CLOCK 1
0.0
1.2
CLOCK 2
0.0
1.20
S1
0.5
1.0
1.5
2.0
NAND GATE


1·2
CLOCK 1
0·0
1·2
CLOCK 2
0·0
1·2
S1
0·0
NOR GATE

9. The poly width is less than 2 lambda (γ301).

RESULT-

The layout of 2 input CMOS NAND gate and 2 input CMOS NOR gate was successfully drawn on microwind 3.1 and the output waveforms were observed.

PRECAUTIONS -

- The microwind software must be handled carefully.

Saun
18/03/19

CLASSMATE
Date 25/2/19
Page: 11.

EXPERIMENT - 4

AIM – Design a 2-input XOR using symbols of 2-input NAND gate on the Virtuoso ADE platform of Cadence.

SOFTWARE USED – Cadence (Virtuoso ADE platform)

THEORY –

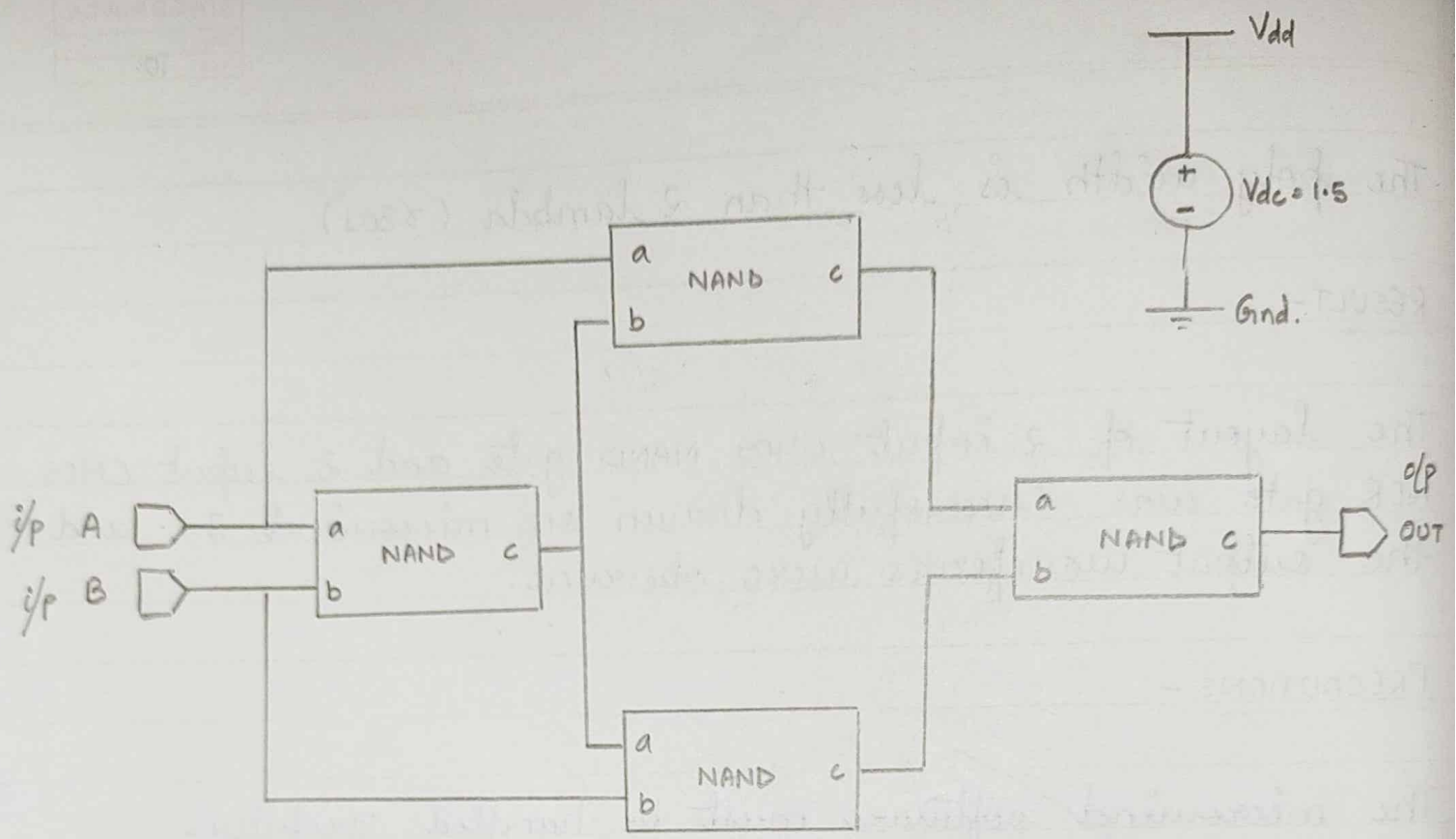
XOR gate is the heart of digital electronics. In digital applications, most important considerations are delay and power dissipation. There are different types of logic for performing some logic function.
According to the requirement, logic is selected. CMOS logic is one of them. Properly designed XOR gate can be very useful in today's high speed era. For two input XOR gate, output is related to the input according to the following equation:

$$Y = A\bar{B} + \bar{A}B$$

where, A and B are the two inputs and Y is the output.

PROCEDURE –

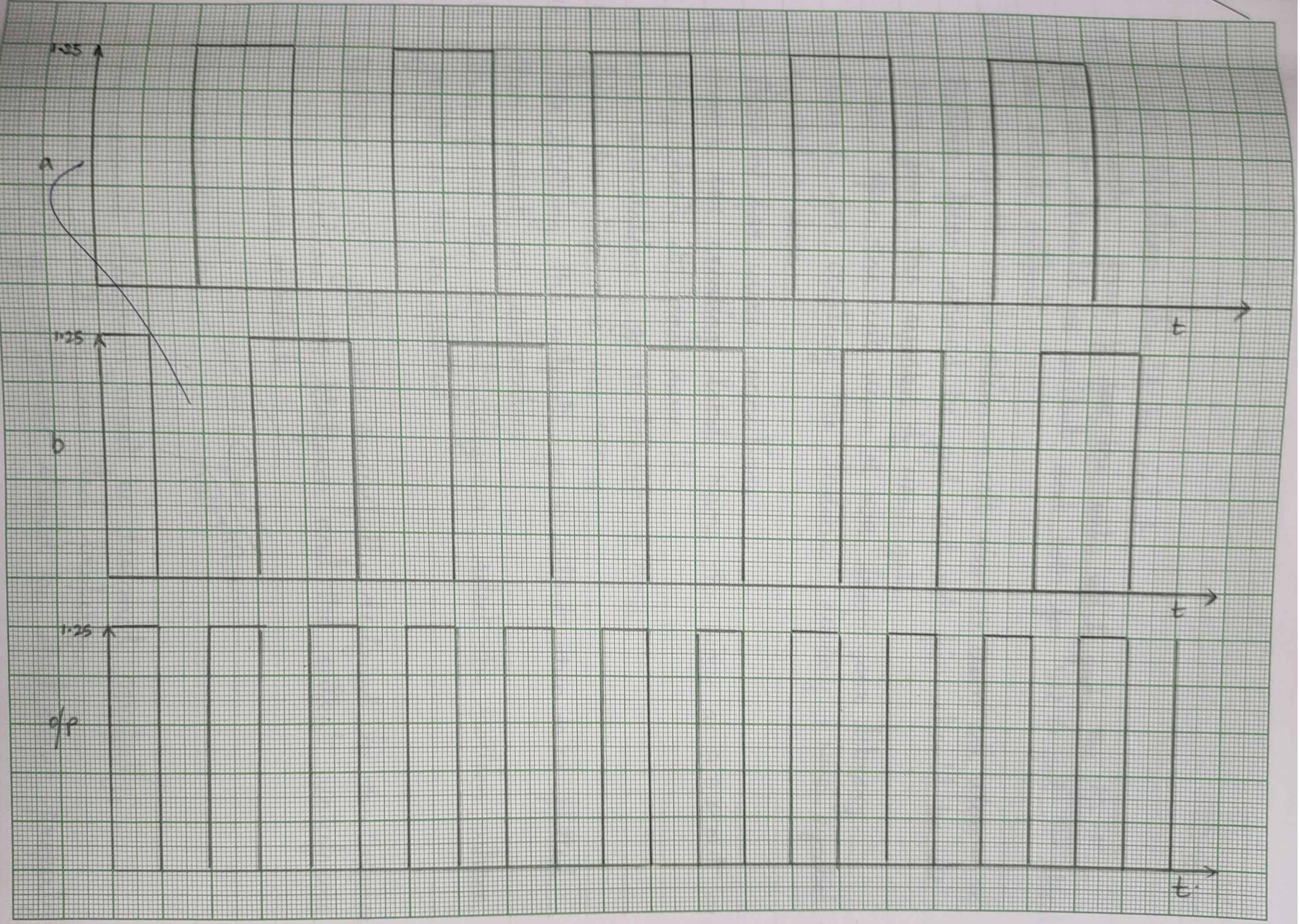
1. Start Cadence Virtuoso ADE.
2. Create new cell view in your pre-created library named as 'XOR' and 'NAND_symbol'.
3. Open the 'NAND_symbol' cell view from your library.
4. Copy the circuit diagram of NAND gate to NAND_symbol.
5. Press p for the pins. For input pins, name it A & B



CIRCUIT DIAGRAM FOR XOR GATE USING NAND GATE

STIMULI	a	b
Voltage 1	1.2	1.2
Voltage 2	0	0
Period	100n	100n
Rise time	1n	1n
Fall time	1n	1n
Pulse width	50n	50n
Delay time	1n	20n

Stop time 500n


1·25
a
t
1·25
b
t
1·25
o/p
t

and direction should be input.

7. Similarly, press p for the out, vdd & gnd. The direction of out should be output and both for Vdd & gnd it should be input output.
8. Now, go to create option → cell view → From cell view → OK. Bottom pins should be gnd and top pin should be Vdd.
9. This gives the symbol of NAND gate.
10. Now, open the cell view of XOR from your library.
11. Press I for creating the instance. Library → NAND-symbol → symbol → close.
12. Press W for wire connections.
13. Press L for creating label. Name the wires as A, B & out and then check it and save it.
14. For simulation, go to launch → ADEL
15. Give the stop time i.e, 500ns.
16. For plotting the output, go to output then to be plotted then select on schematics.
17. Then from the circuit diagram, select the portion whose graph is to be plotted.

RESULT - XOR gate was successfully designed in the Cadence Virtuoso ADE platform and the output waveforms were obtained.

18/03/19.

PRECAUTIONS -

1. The circuit diagram must be designed carefully.
2. The wires should be connected properly.
3. After completing, software should be shut down properly.

EXPERIMENT - 5

AIM - Design a 4x16 decoder by using 5 2x4 decoder using BDE on Active HDL / Xilinx ISE platform.

SOFTWARE USED - Xilinx ISE 10.1

THEORY -

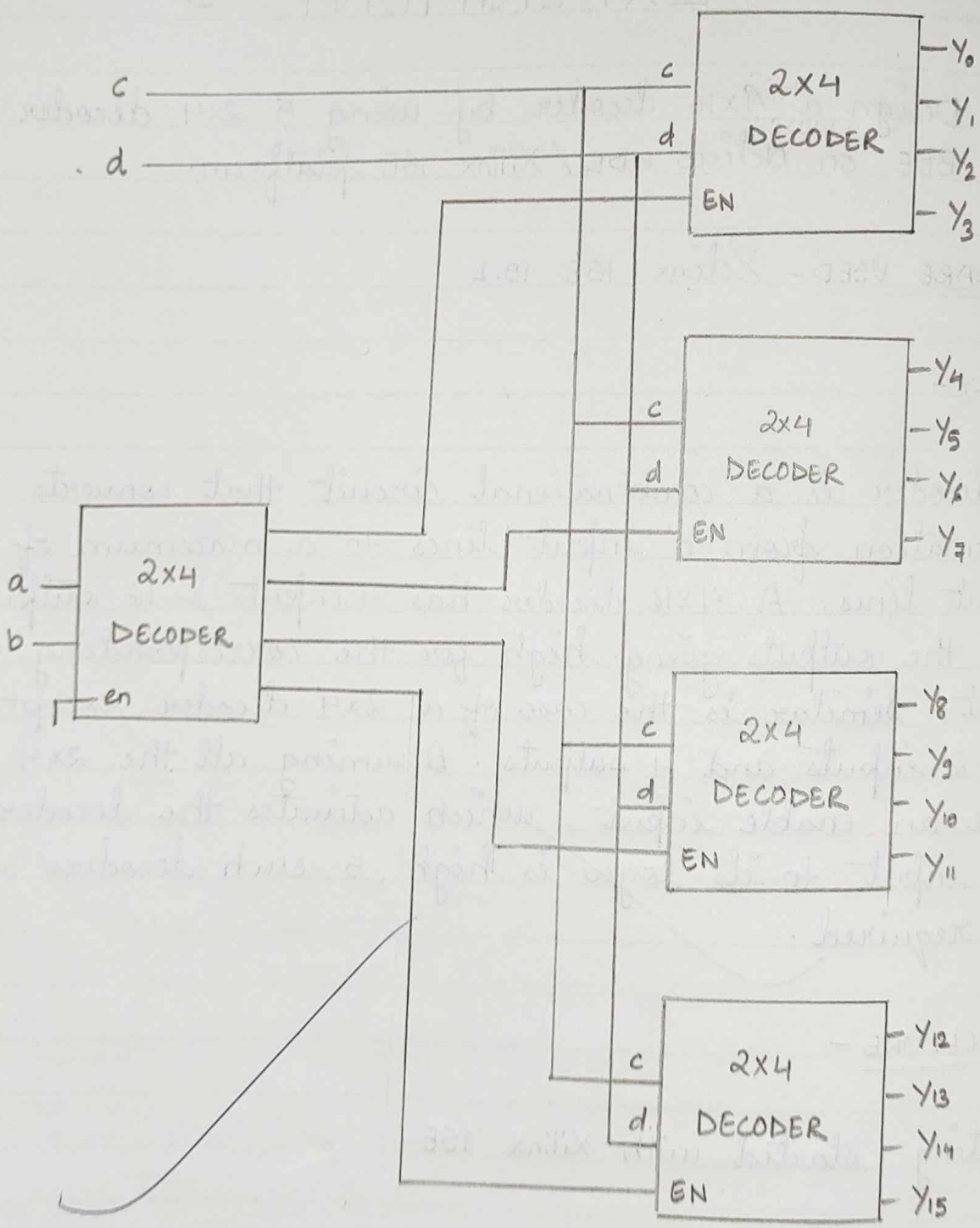
A decoder is a combinational circuit that converts binary information from 'n' input lines to a maximum of 2^n output lines. A 4x16 decoder has 4 inputs & 16 outputs, with the outputs going high for the corresponding 4-bit input. Similar is the case of a 2x4 decoder except for its 2 inputs and 4 outputs. Assuming all the 2x4 decoder have an enable input, which activates the decoder when the input to its logic is high, 5 such decoders would be required.

PROCEDURE -

Getting started with Xilinx ISE:

1. Click on Xilinx ISE icon.
2. Go to file menu. Click on close project.
3. Again go to file file menu & click on New project.

a) New project wizard will come.
b) Give the project name and press next to finish.
c) Now go to create new source.



BLOCK DIAGRAM OF 4X16 DECODER USING 2X4 DECODER

d) A source wizard will appear. Select VHDL / Verilog module and give the file name in the blank, this name will be the entity name in the program.
e) Again press next and define ports in the blank spaces.
f) Select input and output port for the given port name.
g) Press next and then finish.
4. The VHDL / Verilog file opens, write commands after begin and before end.
5. Check that the tab at left sources for: is there is in Synthesis / Implementation.
6. Click on Synthesize XST.
7. After successful completion, click on view Schematics.
8. Change the source for: to Behavioural.

Simulation:

1. Go to create new source in the left window.
2. Click on file and select create source option.
3. Select Test Bench Waveform, give its file name.
4. Select the combinational clock option.
5. On the test bench waveform, click on the horizontal lines to change input of the input port and save the file.
6. Now, go to Xilinx ISE simulation and simulate the behavioural mode & finally check the output.

Codes : VHDL

Code for 2x4 decoder using VHDL.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_AIRTH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity decoder24 is
        Port ( A: in STD_LOGIC;
               B: in STD_LOGIC;
               EN: in STD_LOGIC;
               X: out STD_LOGIC_VECTOR (3 downto 0));
end decoder24
architecture Behavioural of decoder24 is
begin
   X(0) <= (EN) and (not a) and (not B);
   X(1) <= (EN) and (not a) and (B);
   X(2) <= (EN) and (A) and (not B);
   X(3) <= (EN) and (A) and (B);
end Behavioural;
```

Code for 4x16 decoder using 2x4 decoder using VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity decoder416 is
    Port ( a: in STD_LOGIC;
           b: in STD_LOGIC;
           c: in STD_LOGIC;
           d: in STD_LOGIC;
          en: in STD_LOGIC;
           y: out STD_LOGIC_VECTOR (15 downto 0));
end decoder416;
architecture Structural of decoder416 is
component decoder24 is
        Port ( A: in STD_LOGIC;
               B: in STD_LOGIC;
              EN: in STD_LOGIC;
               X: out STD_LOGIC_VECTOR (3 downto 0));
end component;
Signal m: STD_LOGIC_VECTOR (3 downto 0);
begin
DE1 : decoder24 port map (a, b, en, m (3 downto 0));
DE2 : decoder24 port map ( c, d , m(0) , y (3 downto 0));
DE3 : decoder24 port map ( c, d, m(1) , y (7 downto 4));
DE4 : decoder24 port map (c, d, m(2), y (11 downto 8));
DE5 : decoder24 port map (c, d, m(3), y(15 downto 12));
end Structural.
```

Code : Verilog

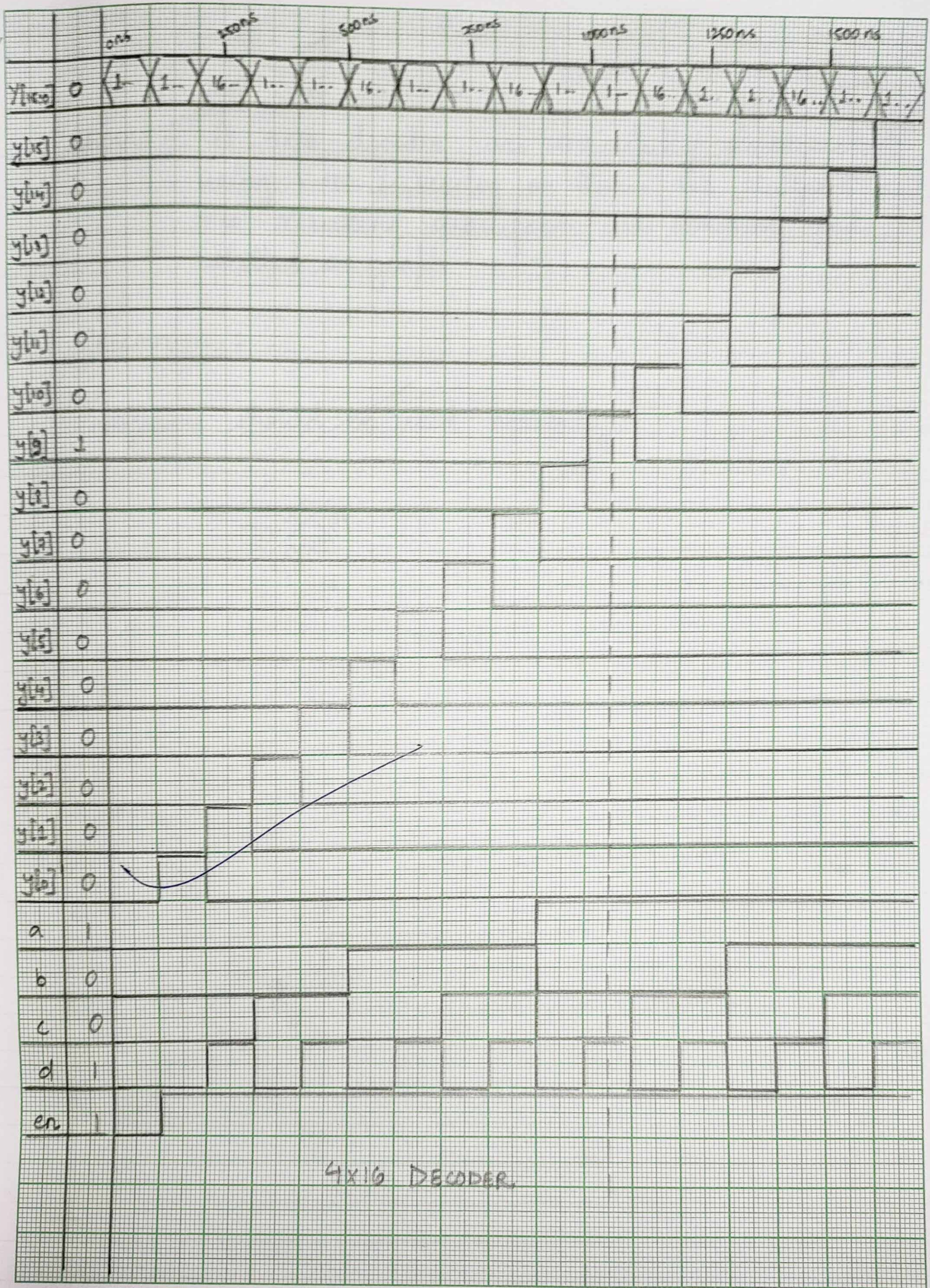
code for 2x4 decoder using Verilog

```
module decoder24 (
                  input A,
                  input B,
```

```
        input EN,
        output [3:0] X );

assign X[0] = (EN) & (!A) & (!B);
assign X[1] = (EN) & (!A) & B;
assign X[2] = (EN) & (A) & (!B);
assign X[3] = (EN) & (A) & (B);
end module
```

Code for 4x16 decoder using 2x4 decoder using Verilog

```
module decode416 (
        input a,
        input b,
        input c,
        input d,
        input en,
        output [15:0] y);

wire [3:0] m;
decoder24 DE1 (a, b, en, m[3:0]);
decoder24 DE2 ( c, d, m[0], y[3:0]);
decoder24 DE3 (c,d, m[1], y[7:4]);
decoder24 DE4 (c, d, m[2], y[11:8]);
decoder24 DE5 (c,d, m[3], y[15:12]);

end module
```



4X16 DECODER.

RESULT – A 4x16 decoder was successfully implemented using VHDL & Verilog on Xilinx ISE platform and the resulting waveform was obtained and plotted on graph.

PRECAUTIONS :

Software should be used carefully.

Sant
01/04/19

EXPERIMENT - 6

AIM - Design a Moore machine with the following state transition diagram using Verilog and VHDL on Active HDL / Xilinx ISE platform.

SOFTWARE USED - Xilinx ISE 10.1

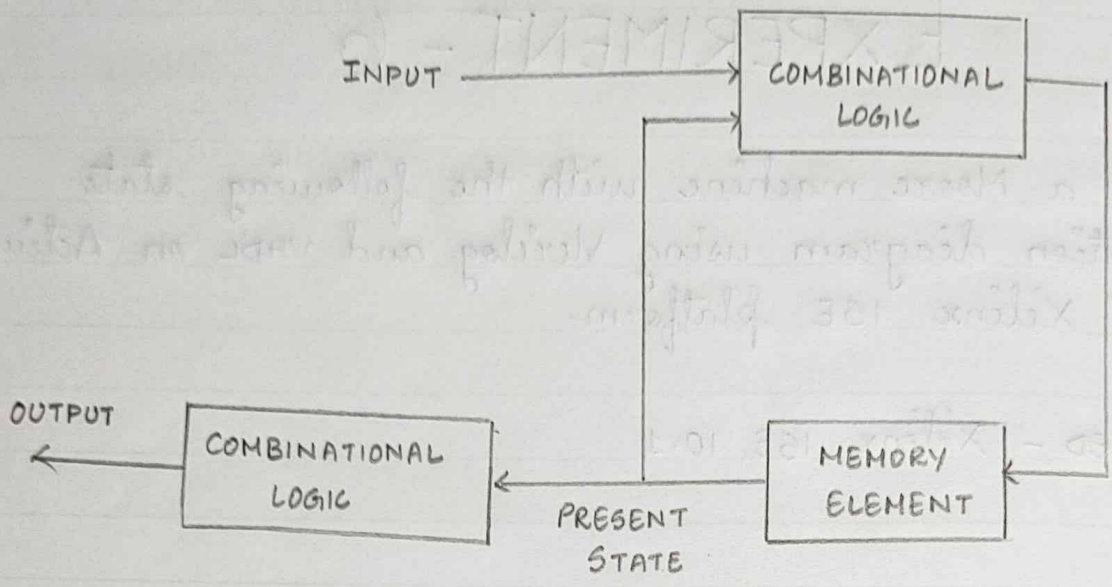
THEORY -

A Moore machine is a finite state machine whose output values are determined only by its current state. A combinational logic block maps the inputs and the current state into the necessary flip flops input to store the appropriate next state. However, the outputs are computed by a combinational logic block whose inputs are only the flip flops state outputs. In the Moore Machine, the outputs change synchronously with the state transition by the active clock edge.

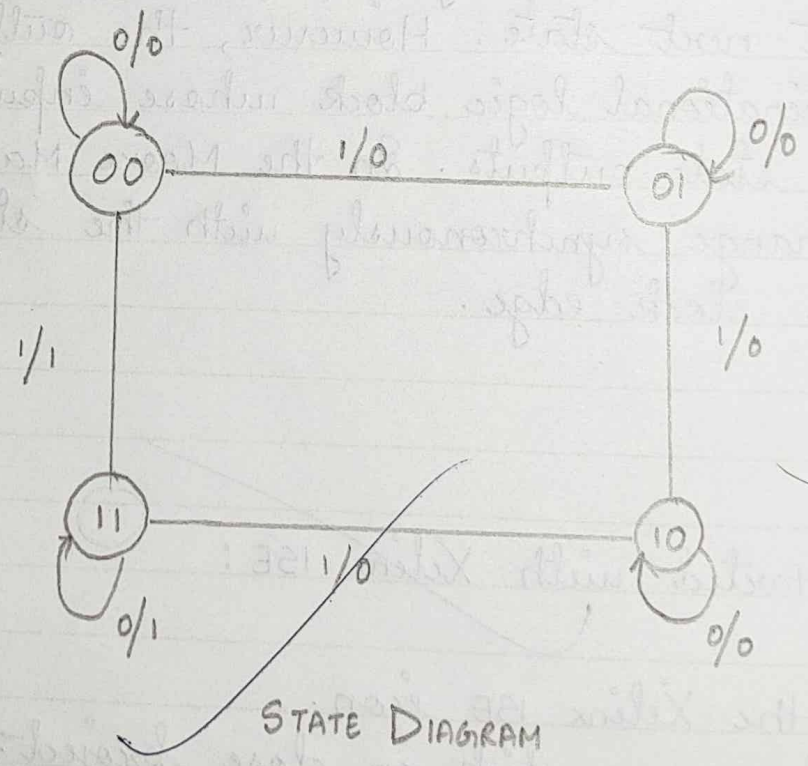
PROCEDURE -

Getting Started with Xilinx ISE:

1. Click on the Xilinx ISE icon.
2. Go to file menu, click on close project.
3. Again go to file and click on New project.
   a) New project wizard will come.
   b) Give project name



MOORE TYPE MACHINE BLOCK DIAGRAM



STATE DIAGRAM

c) Top level source type is selected
Select VHDL / Verilog - press next
d) Now press next and then press new source.
e) Select VHDL / Verilog module and give file name in the blank, this name will be the entity name in program.
f) Again press next and then define ports in the blank space.
g) Select input and output port for the given code. Press finish.

4. Write commands after begin and before end.
5. Check that the tab at left source for : is there in synthesis / Implementation.
6. Click on Synthesize XST.
7. After successful completion, click on view schematics.
8. Change the source for : to behavioural.

Simulation :

1. Go to create new source in the left window.
2. Click on file and select create source option. Select Test Bench Waveform. Give its file name.
3. Select the combinational clock option.
4. On the test bench waveform, click on the horizontal lines to change input of the input port and then save the file.
5. Now go to Xilinx ISE simulator and simulate the behavioural mode and finally check the output.

TABLE :

INPUT X	PRESENT STATE y	NEXT STATE	OUTPUT Z
0	0 0	0 0	0
1	0 0	0 1	0
0	0 1	0 1	0
1	0 1	1 0	0
0	1 0	1 0	0
1	1 0	1 1	0
0	1 1	1 1	1
1	1 1	0 0	1

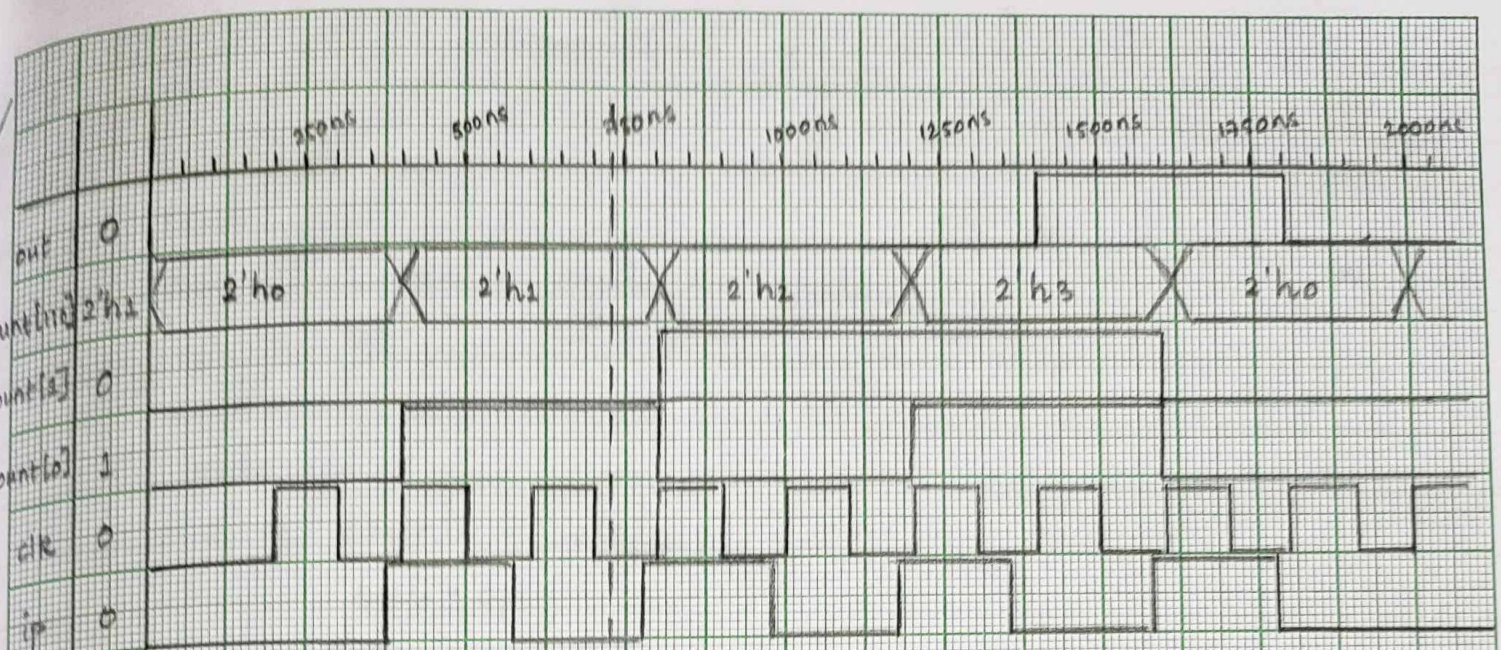
Codes : VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity moore_vhdl is
        Port ( clk : in STD_LOGIC;
               ip : in STD_LOGIC;
               output : out STD_LOGIC;
              count : out STD_LOGIC_VECTOR (1 downto 0));
end moore_vhdl;

architecture Behavioural of moore_vhdl is
signal STATE : STD_LOGIC_VECTOR (1 downto 0) := "00";
signal op : STD_LOGIC := '0';
begin
process ( clk, STATE, ip)
 begin
if ( clk = '1' and clk'event and ip = '0' ) then
 case STATE is
when "11" => STATE <= STATE ; op <= '1';
when others => STATE <= STATE ; op <= '0';
end case;
end if;
if ( clk = '1' and clk'event and ip = '1' ) then
case STATE is
when "11" => STATE <= STATE + 1; op <= '1';
when others => STATE <= STATE + 1 ; op <= '0';
 end case;
```

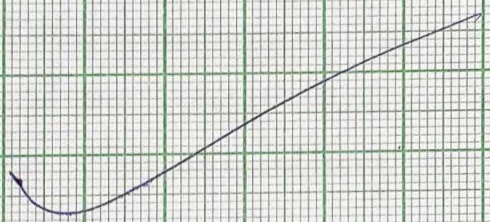
```
end if;
end process;
count <= STATE;
output <= op;
end Behavioural;
```

Code : Verilog

```
module moore_verilog (
        input clk;
        input ip;
        output out;
        output [1:0] count
      );
reg [1:0] state = 2'b00;
reg op = 1'b0;
always @ (posedge clk)
 if (ip)
 case (state)
2'b11 : begin state <= state + 2'b01;
 op <= 1'b1;
 end
default : begin state <= state + 2'b01;
 op <= 1'b0;
 end
 endcase
 else
 case (state)
2'b11 : begin state <= state;
op <= 1'b1;
```



MOORE MACHINE



```
end
default : begin state <= state ;
op <= 1'bo ;
end
endcase
assign count = state ;
assign out = op ;
endmodule
```

RESULT -

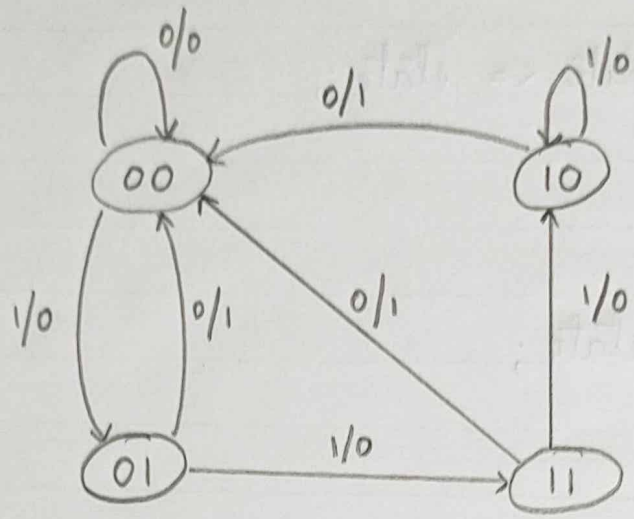
The Moore machine for the given state transition diagram is implemented using VHDL and Verilog on Xilinx ISE platform and the resulting waveform is obtained and is plotted on the graph.

PRECAUTIONS -

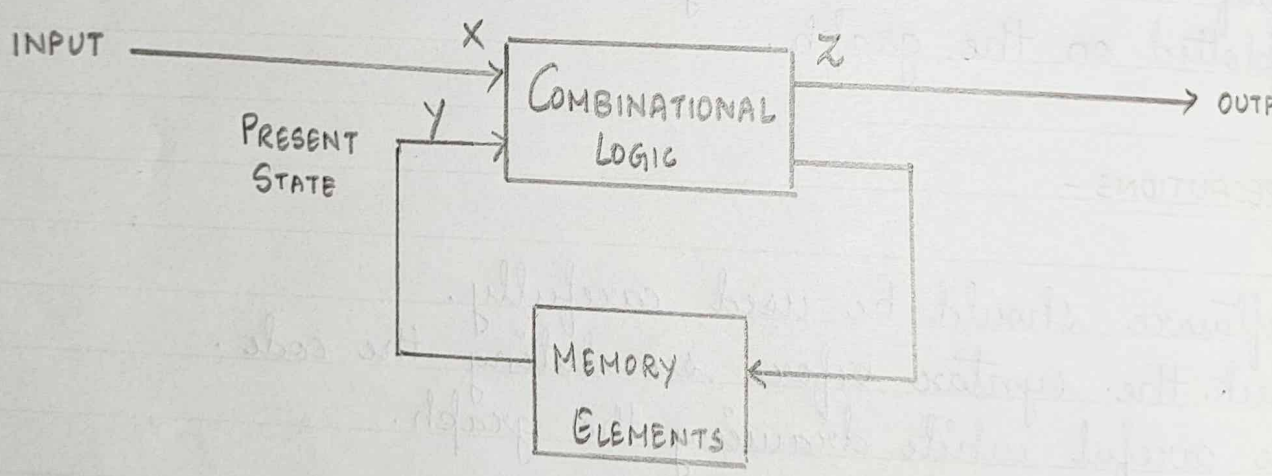
1. Software should be used carefully.
2. Check the syntax before simulating the code.
3. Be careful while drawing the graph.

Jain
1/04/19

State Diagram



Block Diagram



Mealy Type Machine

CLASSMATE
Date: 1/4/19
Page: 24.

EXPERIMENT - 7

AIM - Design a Mealy Machine with the following state transition diagram using Verilog and VHDL on Active HDL / Xilinx ISE platform.

SOFTWARE USED - Xilinx ISE 10.1

THEORY -

A Mealy Machine is a finite state machine whose output values are determined both by its current state and the current inputs. A mealy machine is a deterministic finite state transducer, for each state and input, at most one transition is possible. In this machine, the outputs may change asynchronously in response to any change in the input.

PROCEDURE -

Getting Started with Xilinx ISE -

1. Click on Xilinx ISE platform icon.
2. Go to file menu, click on close project.
3. Again go to file and click on New project.
   a) New project wizard will come.
   b) Give project name.
   c) Top level source type is selected.
      Select VHDL → press next.

Code for VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity mealy_vhdl is
    Port ( clk : in STD_LOGIC;
           ip : in STD_LOGIC;
           output : out STD_LOGIC;
           count : out STD_LOGIC_VECTOR (1 downto 0));
end mealy_vhdl;
architecture Behavioral of mealy_vhdl is
signal STATE : STD_LOGIC_VECTOR (1 downto 0) := "00";
signal op : STD_LOGIC := '0';
begin
process (clk, STATE, ip)
 begin
if ( clk = '1' and clk'event and ip = '0') then
case STATE is
when "00" => STATE <= "00"; op <= '0';
when others => STATE <= "00"; op <= '1';
end case;
end if;
if ( clk = '1' and clk'event and ip = '1' ) then
case STATE is
when "00" => STATE <= "01"; op <= '0';
when "01" => STATE <= "11"; op <= '0';
when others => STATE <= "10"; op <= '0';
end case;
```

```
end if;
end process ;
count <= STATE ;
output <= op ;
end Behavioral;
```

Code for Verilog

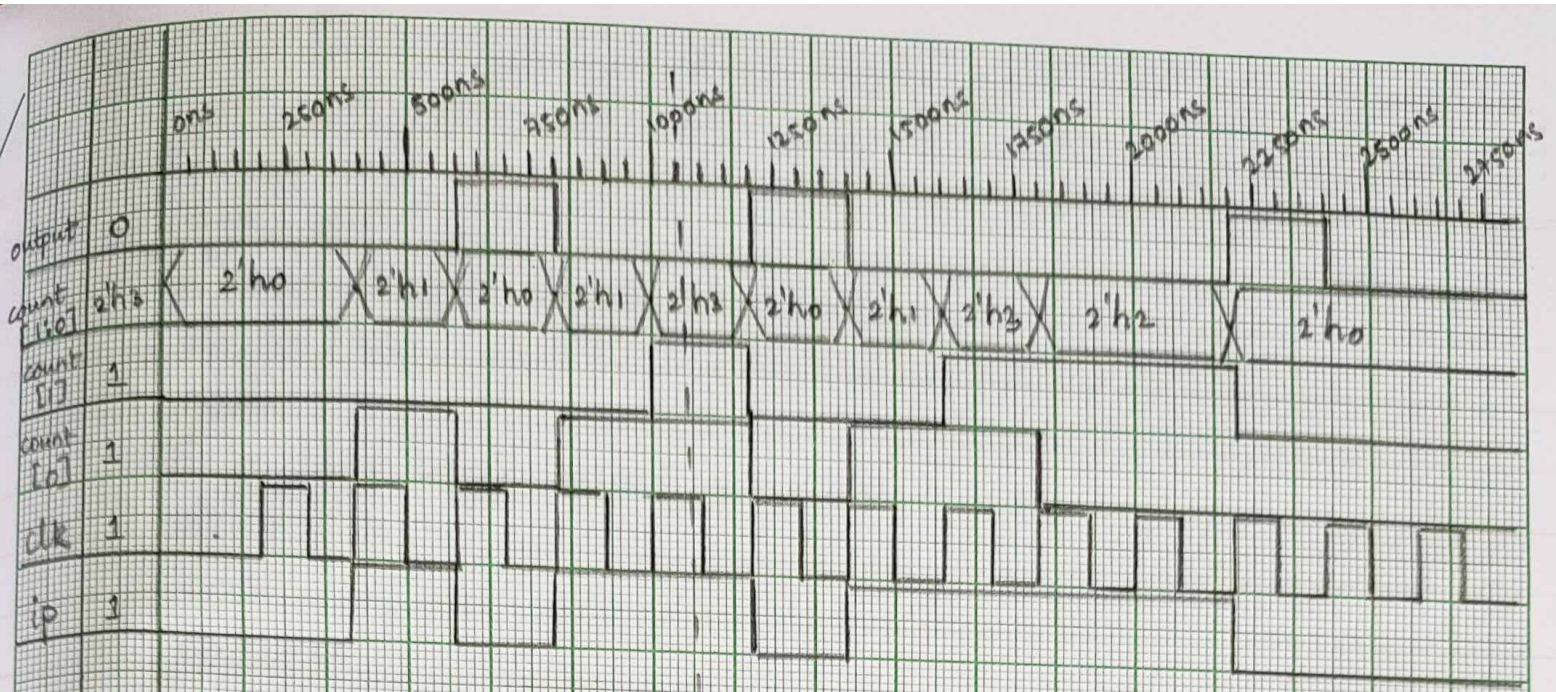
```
module mealyveri (
        input clk,
        input ip,
        output out,
        output [1:0] count
    );
reg [1:0] STATE = 2'b00;
reg op = 1'b0;
always @ (posedge clk)
if (ip)
case (STATE)
2'b00 : begin STATE <= 2'b01;
op <= 1'b0;
end
2'b01 : begin STATE <= 2'b11;
op <= 1'b0;
end
default : begin STATE <= 2'b10;
 op <= 1'b0;
end
endcase
else
```

d) press next, then press new source.
e) Select VHDL module and give file name.
g) Again press next and then define ports in the blank space.
h) Select input and output port for the given code.
i) Press finish.

4. The VHDL opens. Write commands after begin and before end.
5. Check that the tab at left source for: is there in synthesis / implementation.
6. Click on Synthesis XST.
7. After successful completion, click on view schematics.
8. Change the source for: to behavioural.

Simulation

1. Go to create new source in the left window. Click on file and select create source option.
2. Select test bench waveform. Give its file name.
3. Select the combenational clock option.
4. On the test bench waveform, click on the horizontal lines to change input of the input port and then save the file.
5. Now go to Xilinx ISE simulation and simulate the behavioural mode and check the output.



MEALY MACHINE

```
case (STATE)
2'b00 : begin STATE <= 2'b00;
op <= 1'b0;
end
default : begin STATE <= 2'b00;
op <= 1'b1;
end
endcase
assign count = STATE;
assign out = op;
endmodule
```

RESULT –

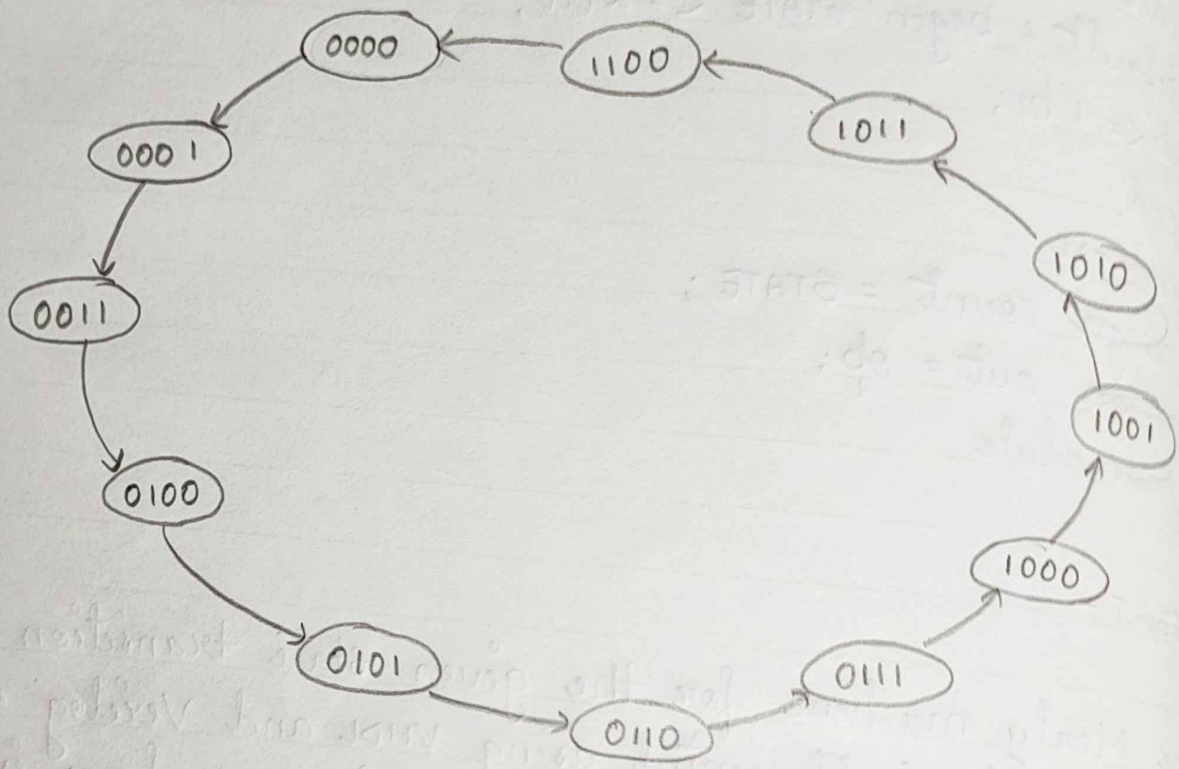
The Mealy machine for the given state transition diagram is implemented using VHDL and Verilog on Xilinx ISE platform and the resulting waveform is obtained and plotted on the graph.

PRECAUTIONS –

The software should be used carefully.

Saurav
15/04/19

Synchronous Counter States



CLASSMATE
Date: 1/4/19
Page: 29.

EXPERIMENT - 8

AIM - Design a synchronous counter with the given states on Xilinx ISE platform.

SOFTWARE USED - Xilinx ISE 10.1

THEORY -

In 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 (in parallel) at the same time giving a fixed time relationship. In other words, changes in the output occur in 'synchronization' with the clock signal.
The result of this synchronisation is that all the individual output bits changing state at exactly the same time in response to the common clock signal with no ripple effect and therefore, no propagation delay.

PROCEDURE -

Getting started with Xilinx ISE :

1. Click on the Xilinx ISE icon.
2. Go to the file menu. Click on close project.
3. Again go to file menu & click on new project.
   a) New project wizard will appear. Give the project

name and then press next to finish.
b) Now go to create new source
c) Select VHDL / Verilog module & give the file name.
d) Press next and then define ports in the space.
e) Select input / output port for the given port name.
f) Press next and then finish.

4. The VHDL / Verilog file opens, write commands after begin and before end.
5. Click on synthesize XST.
6. After successful completion, click on view schematics.
7. Change the source for - to Behavioral

Simulation

1. Go to create new source in the left window.
2. Click on file and select create source option.
3. Select test bench waveform, give the file name.
4. Select the single clock option.
5. Now go to Xilinx ISE simulator and simulate the behavioral mode and finally check the output.

CODE : VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity counter is
    Port ( clk : in STD_LOGIC;
           clr : in STD_LOGIC;
```

```
    count : out STD_LOGIC_VECTOR (3 downto 0));
end counter;
architecture Behavioral of counter is
Signal STATE : STD_LOGIC_VECTOR (3 downto 0) := "0000";
begin
process ( clk, clr, STATE)
 begin
if (clr = '1') then STATE <= "0000";
else if (clk = '1' and clk'event) then
case STATE is
when "1100" => STATE <= "0000";
when "0001" => STATE <= STATE + 2;
when others => STATE <= STATE + 1;
end case;
end if;
end if;
count <= STATE;
end process;
end Behavioral;
```

CODE : Verilog

```
module counter_veri (
        input clr,
        input clk,
        output [3:0] count
        );
reg [3:0] temp = 4'b0000;
always @ (posedge clk)
if (clr)
```

SYNCHRONOUS COUNTER

```
temp = 4'b0000;
else
case (temp)
4'b1100 : temp <= 4'b0000;
4'b0001 : temp <= temp + 4'b0010;
default : temp <= temp + 4'b0001;
endcase
assign count = temp;
endmodule
```

RESULT -

A synchronous counter is implemented using VHDL and Verilog on Xilinx ISE platform and the resulting waveform is obtained and plotted on the graph.

PRECAUTIONS -

The software should be used carefully.

15/04/13

EXPERIMENT - 9

AIM - Design a 1-bit Full Adder circuit using TG logic on the Virtuoso ADE platform of Cadence.

SOFTWARE USED - Cadence (Virtuoso ADE platform)

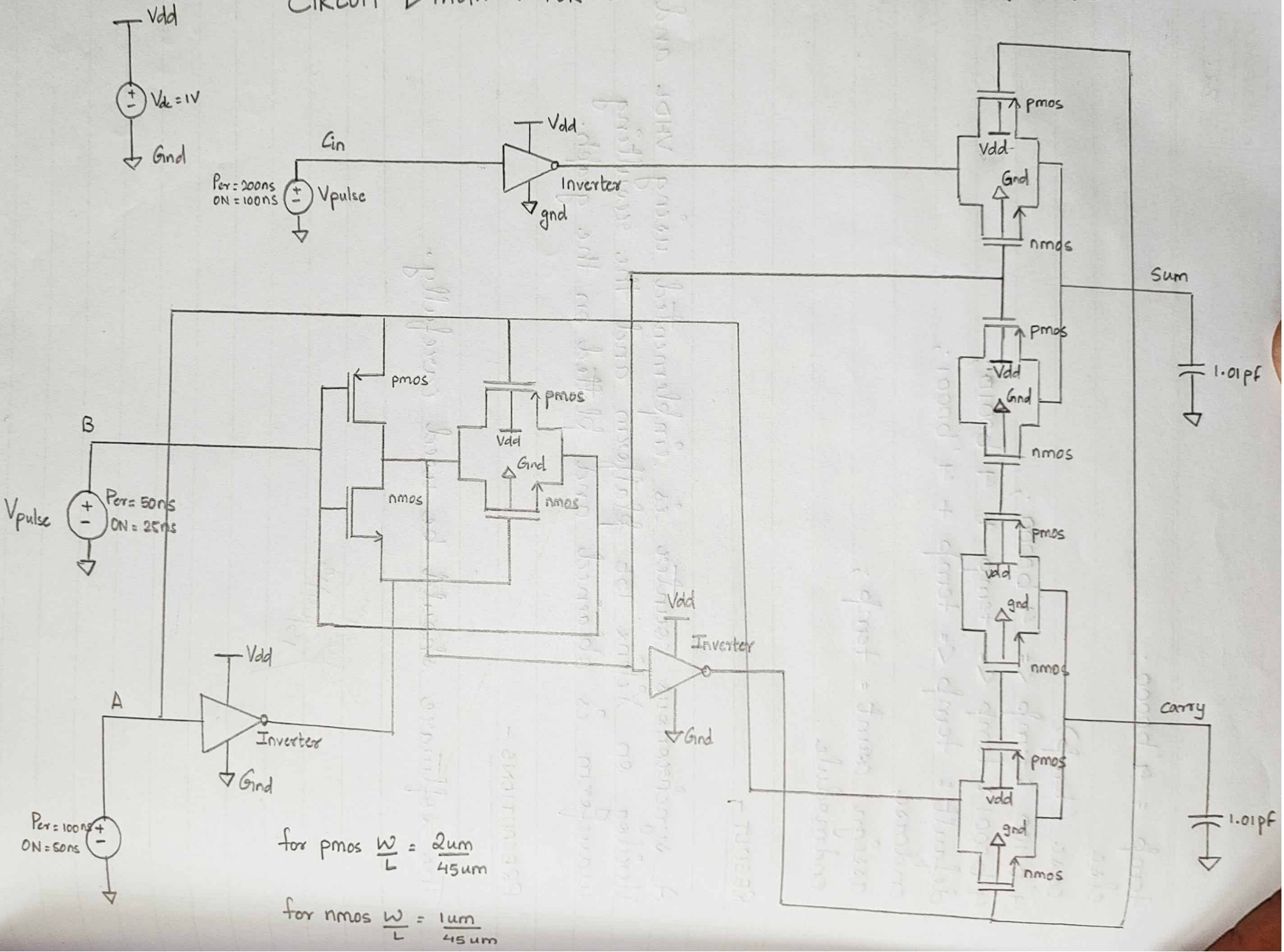
THEORY -

Transmission gate can be operated like a switch with low resistance and capacitance & the DC characteristics of this gate are independent of the input levels. It is designed by connecting both source to source and drain to drain terminals of NMOS and PMOS transistors respectively. As the NMOS transistor is passing strong '0' signal i.e, it discharges the output level to zero and PMOS transistor passes strong '1' signal towards the output i.e, it changes the output to logic high, the enable signal will turn on or off both transistor at a time. Sizing of transistor varies as the resistance and capacitance decrease and increase respectively with the gate W/L ratio is increased.

PROCEDURE -

1. Start Cadence
2. Create new cell view in your precreated library named 'Inverter_symbol' and 'Fulladder_using_TG'.
3. Open the full adder_using_TG cell view from your

Circuit Diagram For 1 Bit Full Adder Using TG Logic



OBSERVATION TABLE

PREVIOUS STATE					NEXT STATE				
A	B	Cin	S	Cout	A	B	Cin	S	Cout
1	1	1	1	1	1	0	1	0	1
1	0	1	0	1	0	1	1	0	1
0	1	1	0	1	0	0	1	1	0
0	0	1	1	0	1	1	0	0	1

SUM

CARRY

TRANSIENT TIME		DELAY TIME		DELAY TIME		TRANSIENT TIME	
TYPE	TIME	TYPE	TIME	TYPE	TIME	TYPE	TIME
Fall	1.75ns	Fall	1.099ns	-	-	-	-
-	-	-	-	-	-	-	-
Rise	2.916ns	Rise	951.1 ps	Fall	821.1ps	Fall	1.058ns
Fall	1.75ns	Fall	1.095 ns	Rise	779 ps	Rise	1.94ns

library.

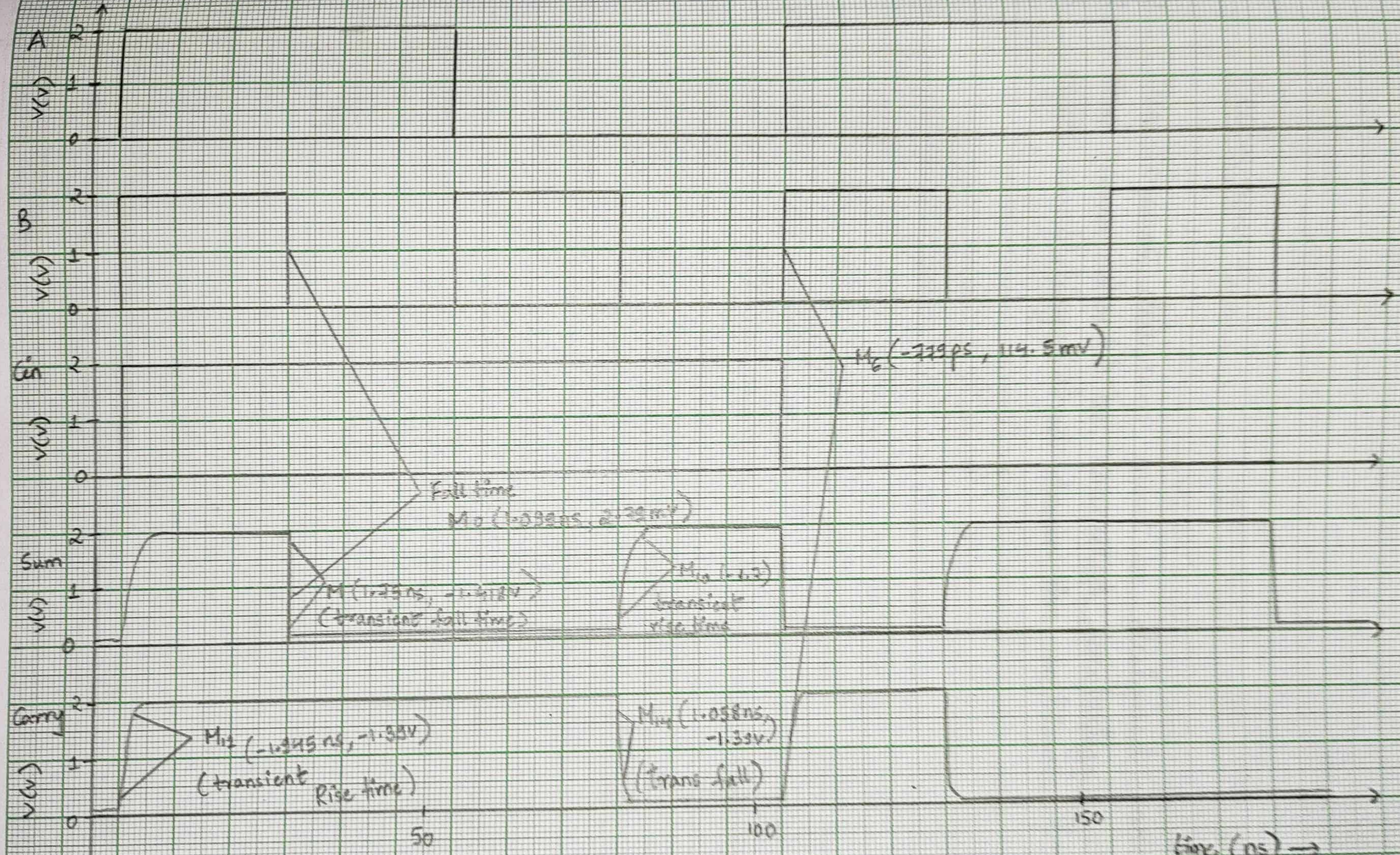
4. Click on add instance. Take all components from gpdk045 library i.e, nmos1v and pmos1v.
5. Press P for pins.
6. Now, go to create option → cell view → from cell view → OK.
7. This genes symbol of inverter.
8. Now, open cell view of Full adder_using_TG from your library.
9. Again, add instance, do the wire connections, label, then check and save it and then simulate it.
10. Give the stop time : i.e, 430ns.
11. Plot the graph.

SPECIFICATIONS - VPULSE

	Cin	A	B
Voltage 1	0V	0V	0V
Voltage 2	1.8V	1.8V	1.8V
Period	200ns	100ns	50ns
Delay time	3ns	3ns	3ns
Rise time	10ps	10ps	10ps
Fall time	10ps	10ps	10ps
Pulse width	100ns	50ns	25ns

RESULT -

The full adder 1 bit was designed using transmission gate in the Cadence Virtuoso ADE platform and output waveform were obtained as per the requirement.


A
B
Cin
Sum
Carry
V(v)
Fall time
M_6 (-779ps, 114.5mV)
(transient fall time)
transient rise time
M_{12} (-1.345 ns, -1.38V)
(transient Rise time)
(trans fall)
50
100
150
time (ns) →

PRECAUTIONS –

1. The circuit diagram must be designed carefully.
2. The wires should be connected properly.
3. After completing, shut down the software properly.

15/04/19

EXPERIMENT-10

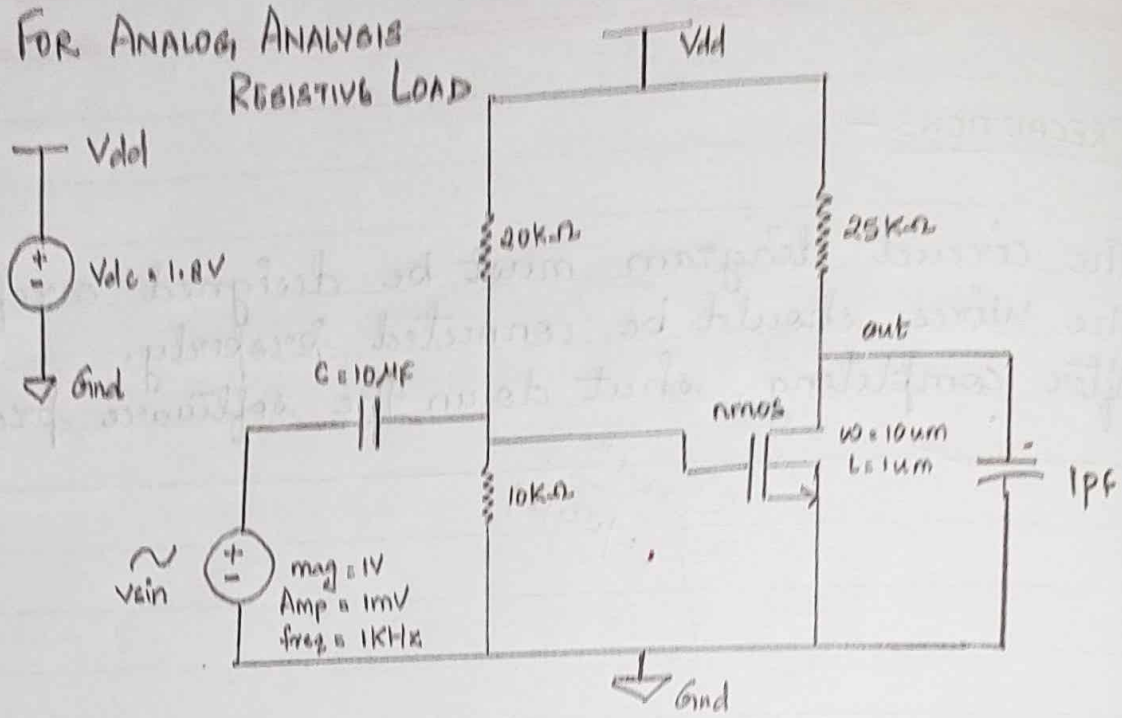
AIM - Design a common source (CS) amplifier using an nMOSFET (Q_1) with a small signal gain of atleast 3 wrt gnd. on the Virtuoso ADE platform of Cadence. Implement the load current with current source I_{DC} & pMOSFET current mirror, i/p device of which is Q_3 and output device of which is Q_2.

SOFTWARE USED - Cadence Virtuoso ADE platform

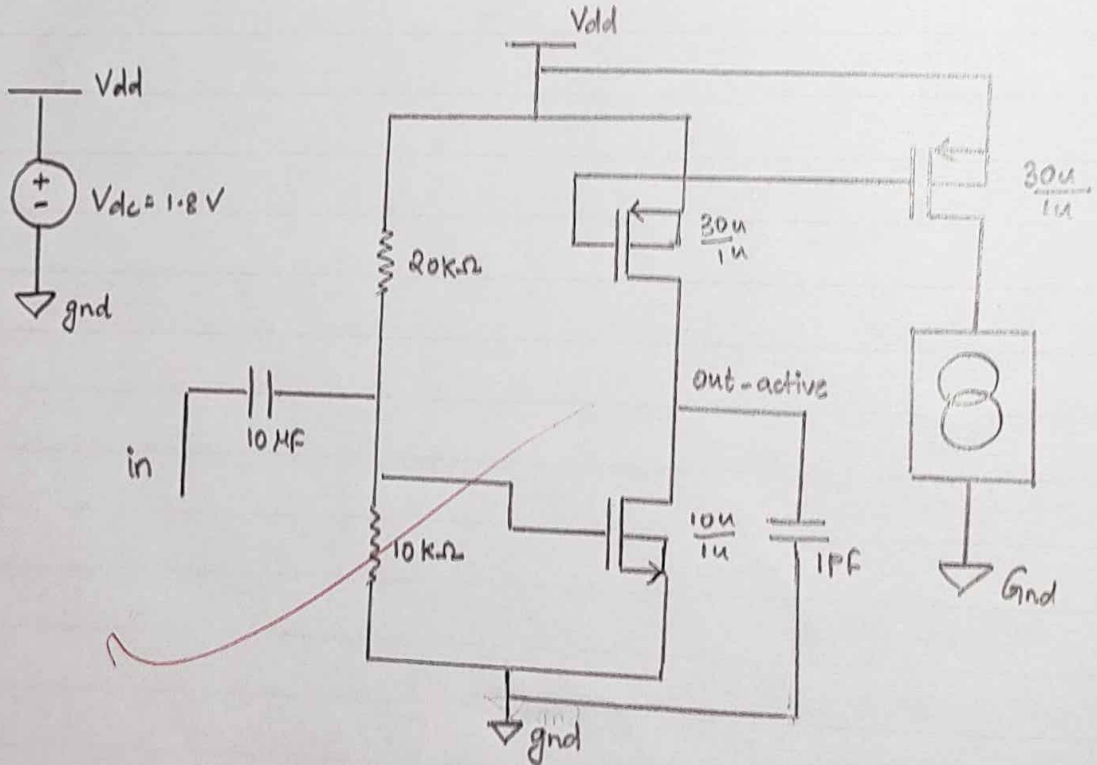
THEORY -

In electronics, a common source amplifier is one of three basic single-stage FET amplifier topologies, typically used as a voltage or transconductance amplifier. The easiest way to tell if an FET is common source, common drain or common gate is to examine where the signal enters and leaves. 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 o/p resistance typically is not high enough for a reasonable transconductance amplifier (ideally infinite) nor low enough for a decent voltage amplifier (ideally zero).

FOR ANALOG ANALYSIS
RESISTIVE LOAD



FOR ACTIVE LOAD



CIRCUIT DIAGRAM FOR CS AMPLIFIER

OBSERVATIONS

FOR RESISTIVE LOAD

$i = 10\mu$

1) DC bias

$R_D = 90\,K\Omega$

nmos

$W = 3.2\mu w$

$i_d = 10.3379\,\mu$

$V_{gs} = 600\,m$

$V_{ds} = 869.472\,m$

$g_m = 124.036\,\mu$

2) Gain = 20.61 dB

= 10.733 V

Bandwidth = 3.454 MHz

FOR ACTIVE LOAD

$i = 10\mu$

1) DC bias

pmos: $W_p = 8.2\mu w$

$i_d = -10.3252\mu$

$V_{gs} = -650\,nm$

$V_{ds} = -959.22\,n$

$g_m = 90.1887\,\mu$

nmos:

$W_n = 3.2\mu w$

$i_d = 10.3251$

$V_{gs} = 600\,m$

$V_{ds} = 840.78\,m$

$g_m = 123.918\,\mu$

2) Gain = 142.65

= 43.08 dB

Bandwidth = 127.4 kHz

3) Transient

Out: Peak to Peak = 494.1 Ms

In:

Max to min = 496.9 Ms

gds of pmos = 417.931 n

gds of nmos = 450.637 n

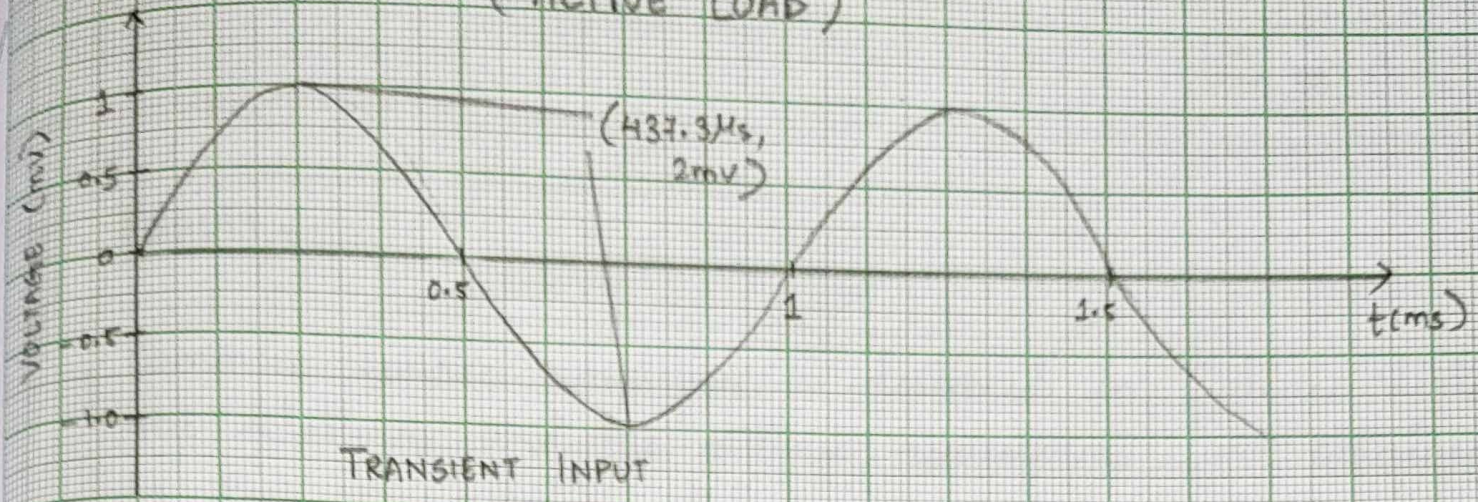
PROCEDURE –

1. Start Cadence Virtuoso ADE.
2. Create library and then cell view.
3. Create instance by selecting from the menu bar.
4. Select V_{DC}, V_{DD}, gnd, Vsin from Analog library.
5. Select NMOS from gpdk180 library.
6. Press W for wire connections.
7. Press L for the wire name.
8. Check and save the circuit diagram.
9. Click on launch → ADEL → choose analysis i.e, dc, ac, trans.
10. For analog analysis, after simulating, go to result then Annotate → DC operating point.
    start 10 stop 16 → OK.
11. Result → Direct plot → Main form ⇒ ac → voltage → dB20 → Add to o/p → Select out (from ckt) → OK.
    This is for ac analyses.
12. For trans analysis, stop time = 1m.
    (select appropriate a/c to i/p freq.)

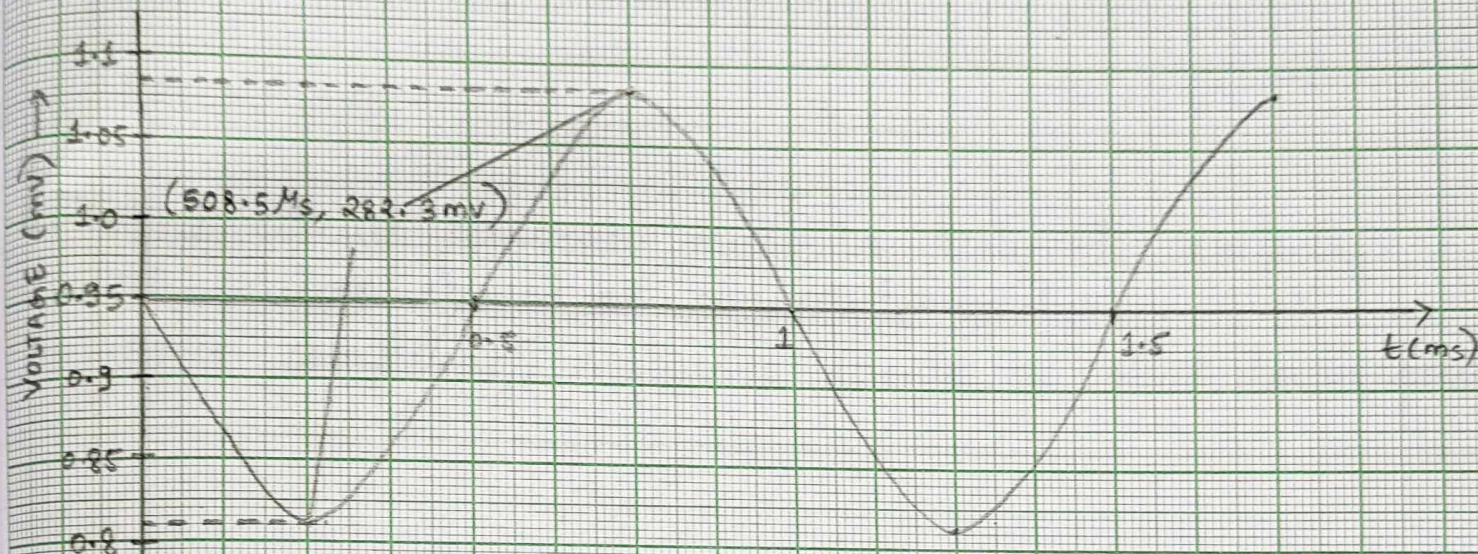
SPECIFICATIONS –

Vsin — AC magnitude 1V
amplitude 1mV
freq. 1KHz
Vsin is used instead of Vpulse.

(ACTIVE LOAD)

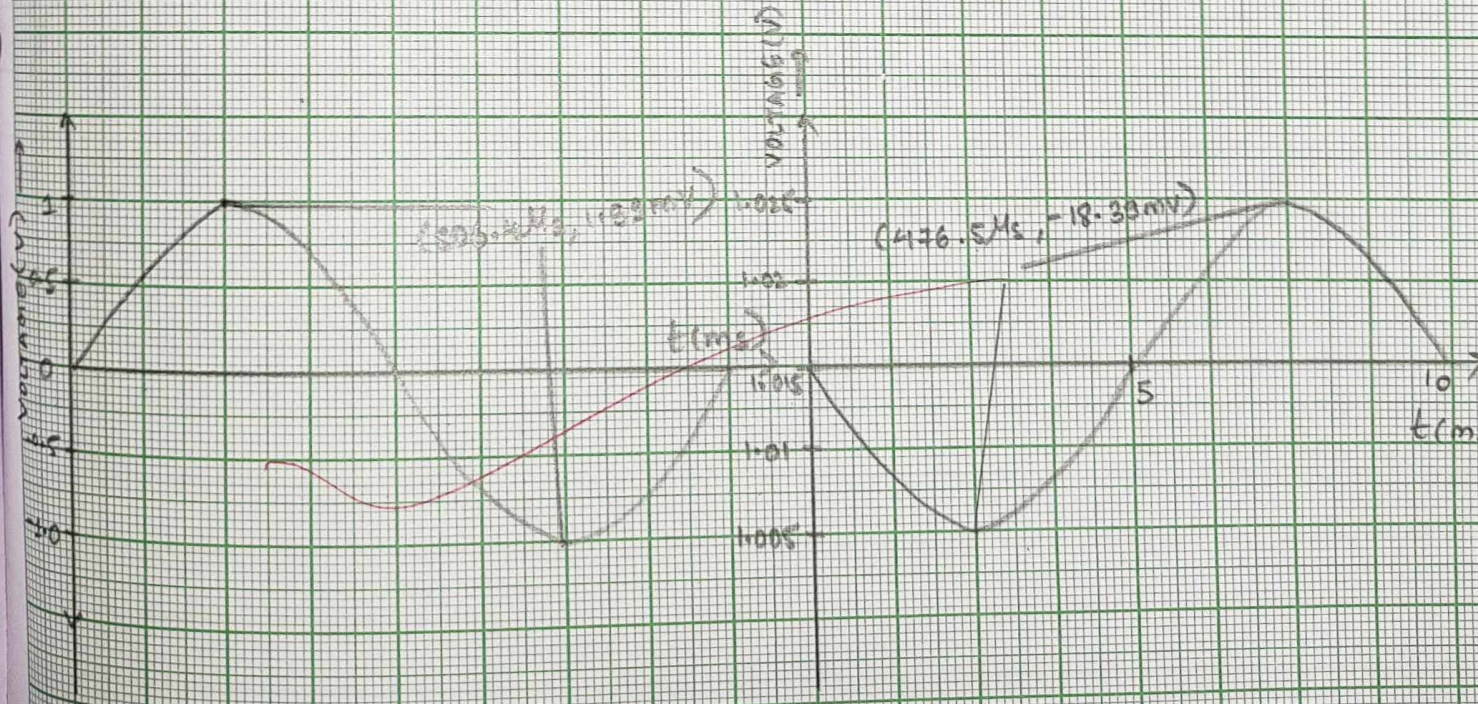


TRANSIENT INPUT



TRANSIENT OUTPUT

RESISTIVE LOAD



RESULT -

Common source amplifier is designed in Cadence Virtuoso ADE platform and the output waveforms were obtained as per the requirement.

PRECAUTIONS -

1. The circuit diagram must be designed carefully.
2. The wires should be connected properly.
3. After completing, software should be shut down properly.

Saurav
15/04/19

EXPERIMENT - 11

AIM - Design an inverting differential amplifier with a gain of atleast 3 with respect to ground on the Virtuoso ADE platform of Cadence.

SOFTWARE USED - Cadence Virtuoso ADE platform

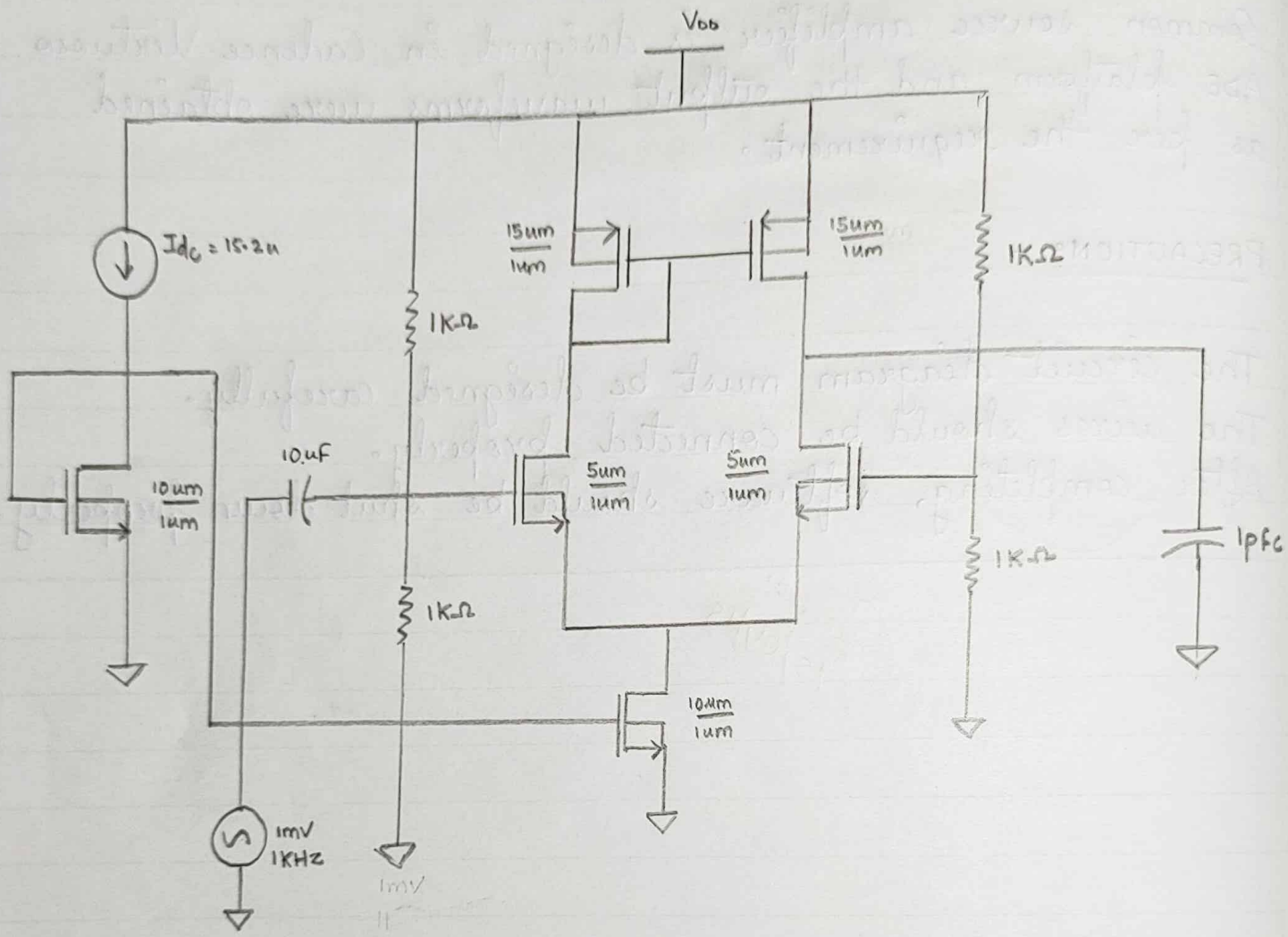
THEORY -

The important advantage of differential operation over single ended signaling is higher immunity to environmental noise. Since the common mode level of the two phases is disturbed but the differential output is not corrupted we say this arrangement "reject" common mode noise. Other advantages of differential circuits over single ended circuits include simpler biasing and higher linearity.

PROCEDURE -

1. Start cadence.
2. Create new cell view in ~~your~~ precreated library or first create a library with your name and attach that library to design ~~library~~, then create new cell view under this library.
3. Click on "add instance" take all components from analog and gpdk180 library.
4. Give input using pins.
5. After completing design, check and save the same and

Circuit Diagram For Inverting Differential Amplifier



OBSERVATIONS –

1. DC bias

   R_1 = 100kΩ

   R_0 = 80kΩ

   For biasing nmos in current mirror circuit

   R_1 = 100kΩ , R_2 = 50kΩ for bias circuit CMOS

2. Frequency gain

   gain = 35.5V

   = 39.32dB

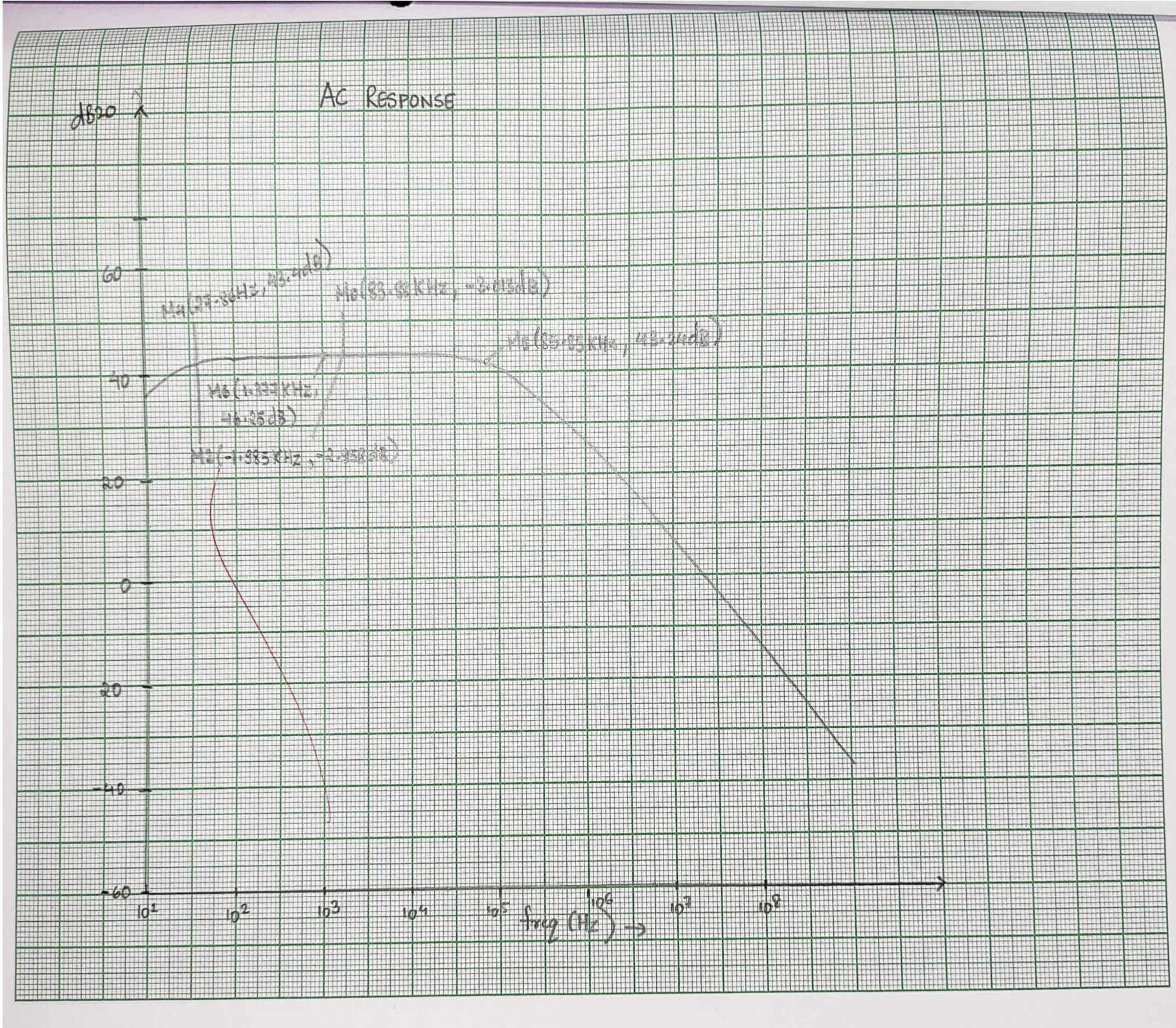
3. Bandwidth = 142.8 kHz

   Unity Bandwidth = 17.79 MHz

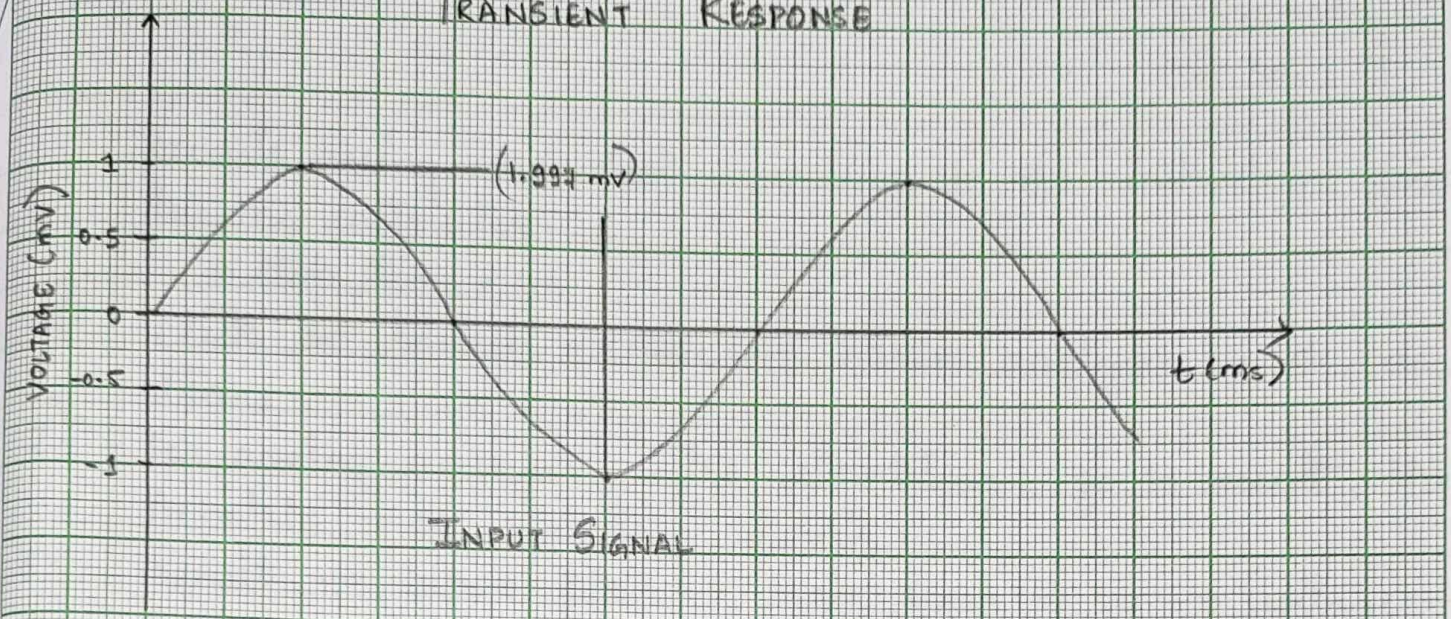
4. Transient

   out = Peak to peak = 264.3 mV , 517.2 μs

   In = max to min : 44.9 nA , 521.2 μs


AC RESPONSE
dB20
60
40
20
0
-20
-40
-60
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
Freq (Hz) →
H4(27.36KHz, 36.42dB)
H6(63.68KHz, -3.015dB)
H5(63.03KHz, 43.21dB)
H6(1.833KHz, -6.25dB)
H2(-1.985KHz, -2.953dB)

TRANSIENT RESPONSE



INPUT SIGNAL

OUTPUT SIGNAL

execute using ADE tool according to the instruction given by the instructor.

stop time = 1m

6. Click on launch → ADEL → choose analysis i.e dc, ac, trans
7. For analog analysis, after simulating, go to result, then Annotate → DC operating points.
8. Result → Direct plot → main form ⇒ ac → voltage → dB20 → Add to o/p → select out (from ckt) → OK
9. For trans analysis, select appropriate a/c to i/p freq.

RESULT –

Inverting differential amplifier was successfully designed using Cadence and the output waveforms were obtained as per the requirement.

PRECAUTIONS –

1. The circuit diagram must be designed carefully.
2. The wires should be connected properly.
3. After completing, software should be shut down properly.

EXPERIMENT - 12

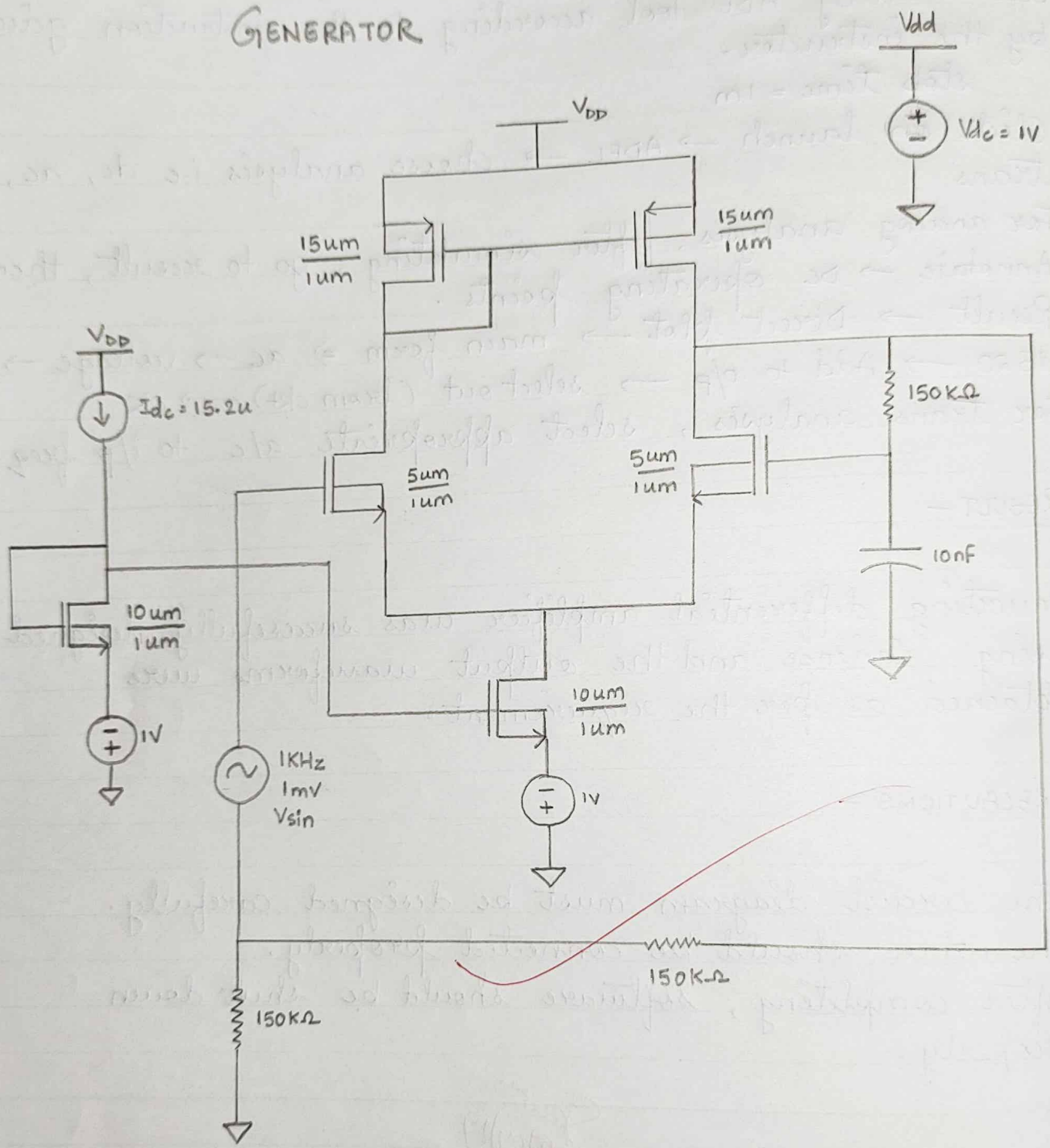
AIM - Design an Op Amp based square wave generator with a frequency of atleast 1 MHz on the Virtuoso ADE platform of Cadence.

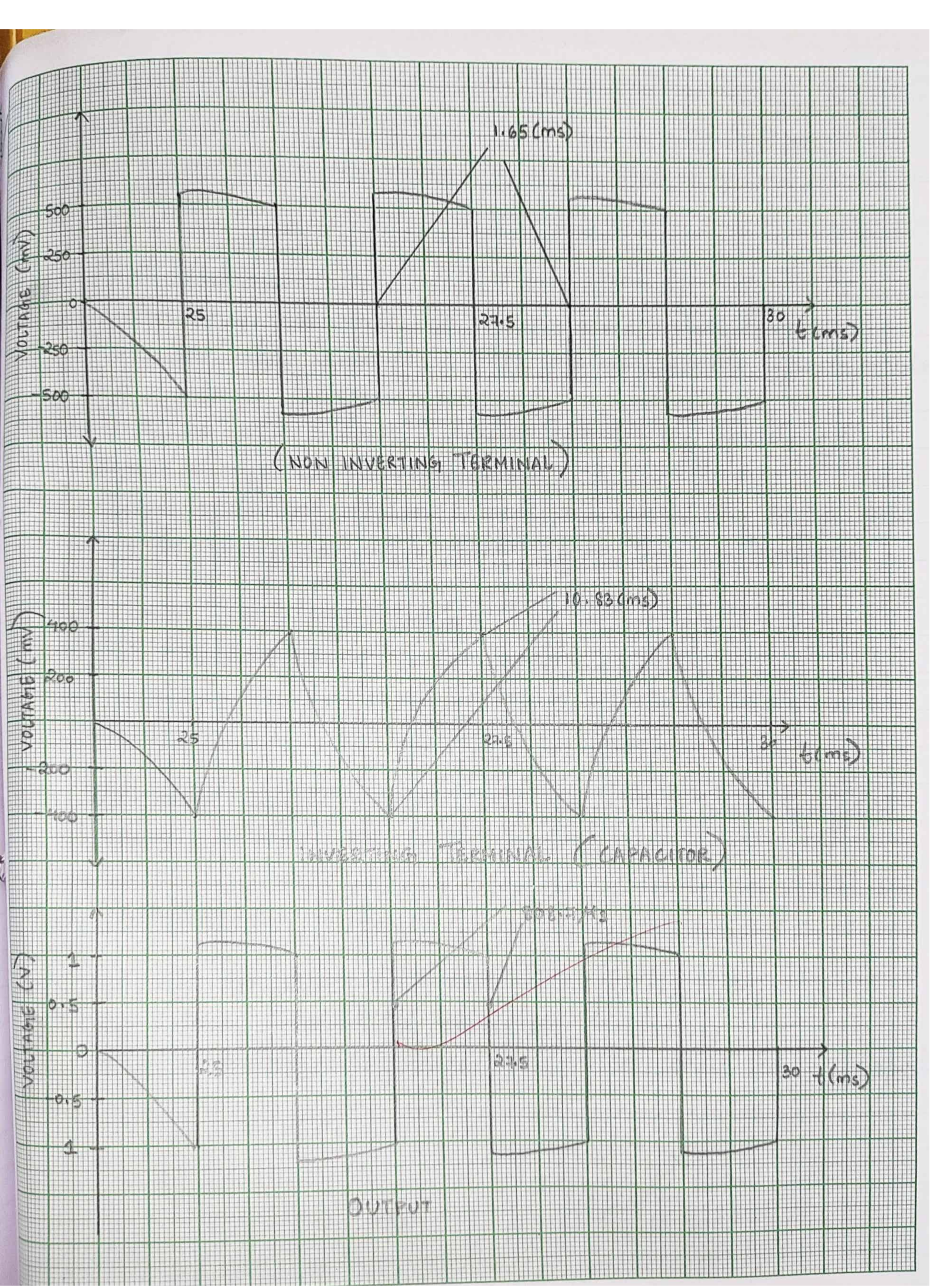
SOFTWARE USED - Cadence (Virtuoso ADE platform)

THEORY -

Square wave generator can be constructed using Schmitt Trigger inverter like TTL. It is the easy way to make a basic astable waveform generator. While producing clock or timing signals, this astable multivibrator produces a square wave generator waveform that switches between high and low. The op-amp is one of the basic building blocks of linear design. It consists of two input terminals, one of which inverts the phase of the signal, the other preserves the phase and an output terminal. It is one type of differential amplifier opamps. Opamps are among the most widely used electronic devices today, being used in a vast array of consumer, industrial and scientific devices. Op-amps may be packaged as components, or used as elements of more complex integrated circuits.

Circuit Diagram For OPAMP Based Square Wave Generator




1.65 (ms)
500
250
0
250
500
VOLTAGE (mV)
25
27.5
30
t (ms)
(NON INVERTING TERMINAL)
10.83 (ms)
400
200
200
400
VOLTAGE (mV)
25
27.5
30
t (ms)
INVERTING TERMINAL (CAPACITOR)
1
0.5
0
0.5
1
VOLTAGE (V)
25
27.5
30
t (ms)
OUTPUT

PROCEDURE –

1. Start Cadence.
2. Create new cell view in your precreated library.
3. Click on "add instance". Take all components from analog and gpdk180 libraries.
4. Click on "add instance". Give inputs by pressing L.
5. Press 'w' for wire connections.
6. After completing design, check and save the same and execute using ADE tool according to the instructions given by the instructor.
7. trans – stop time = 500m
8. Plot the waveform in the graph.

RESULT –

Op-amp based square wave generator was designed using Cadence tools.

PRECAUTIONS –

1. Circuit diagram must be designed carefully.
2. The wires should be connected properly.
3. After completing, software should be shut down properly.

Saurav
15/04/19