EXPERIMENT NO. 01

im: To Synthesis, Simulate and Implement 4-bit ALU using behavioral style.

bjective: 1.To understand ALU addition, Subtraction, Multiplication, Division etc. depends on different select lines

2. To understand simulation on Hardware.

ardware Platform:

Device Family	Spartan 3E
Device	XC3S400
Package	PQ208

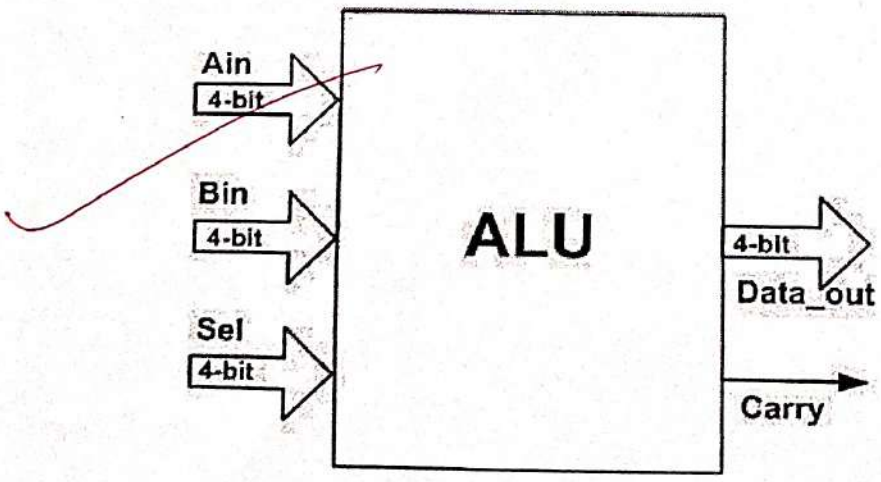
tware Platform:

Top Level Source Type	HDL
Synthesis Tool	Xilinx 9.2i
Simulator	ISE Simulator

ieory:

ock Diagram:

An arithmetic logic unit (ALU) is a logic circuit that performs various Boolean and arithmetic On ample of an ALU is the chip called the 74381. Table 6.1 specifies the functionality of this chip. It has 2 four : data inputs, A and B, a three-bit select input, S, and a four-bit output, F. As the table shows, F is defined by rious arithmetic or Boolean operations on the inputs A and B. In this table + means arithmetic addition, and - eans arithmetic subtraction. To avoid confusion, the table uses the words XOR, OR, and AND for the Boolean erations. Each Boolean operation is done in a bitwise fashion. For example, F = A AND B produces the four- result f0 = a0b0, f1 = a1b1, f2 = a2b2, and f3 = a3b3.



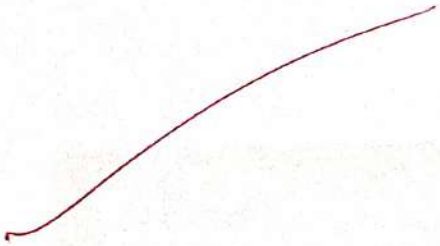
BE 140- Piyush Sawkar

4-bit ALU :

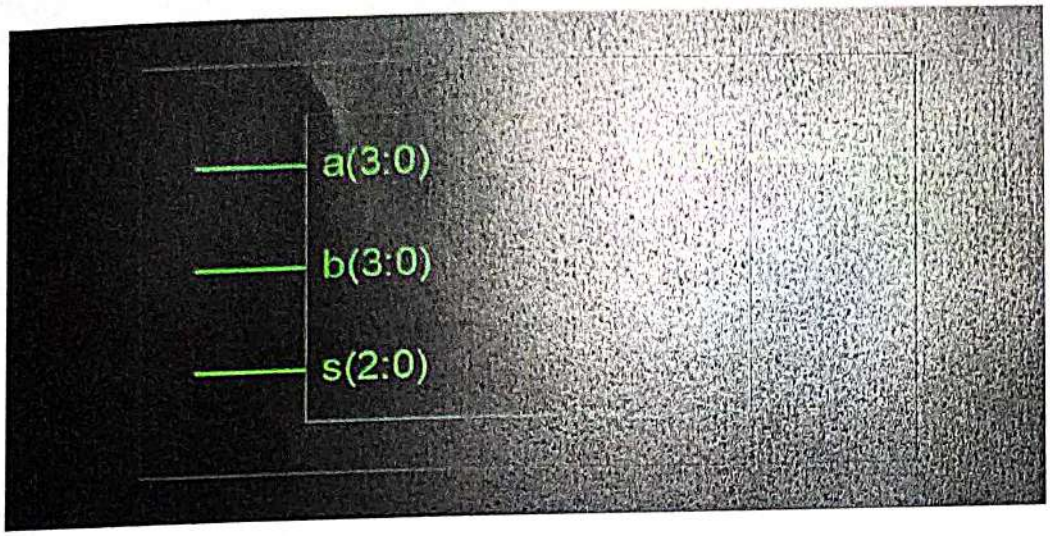
VHDL Code :

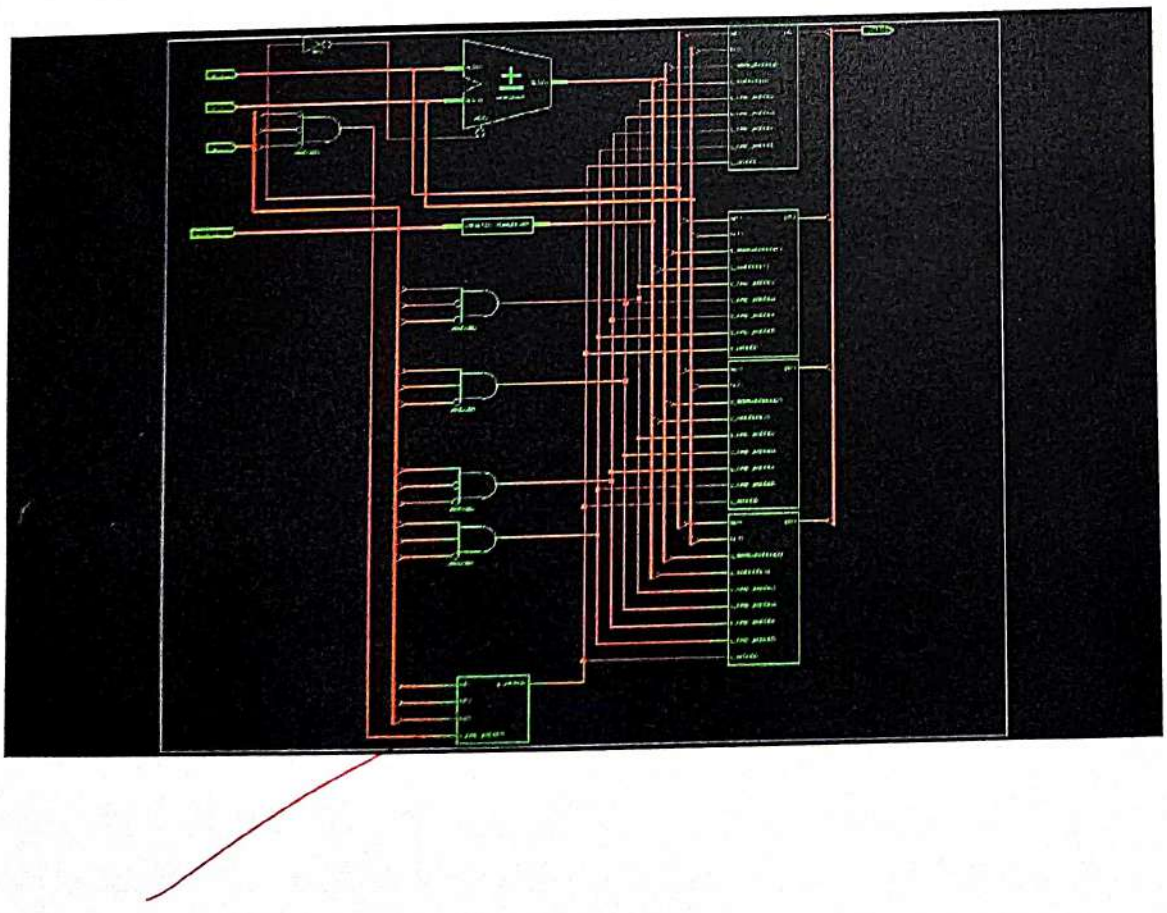
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity alu is
   Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
          b : in  STD_LOGIC_VECTOR (3 downto 0);
          s : in  STD_LOGIC_VECTOR (2 downto 0);
          y : out  STD_LOGIC_VECTOR (3 downto 0));
end alu;

architecture Behavioral of alu is

begin
process(a,b,s)
begin
case s is
when "000"=>y<=a+b;
when "001"=>y<=a-b;
```

```
when "010"=>y<=a and b;
when "011"=>y<=a or b;
when "100"=>y<=not b;
when "101"=>y<=a xor b;

when others=>y<="0000";
end case;
end process;
end Behavioral;
```



BE 140- Piyush Sawkar





BE 140- Piyush Sawkar

TestBench code :

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY alu1247_vhd IS
END alu1247_vhd;

ARCHITECTURE behavior OF alu1247_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT alu1447
    PORT(
        a : IN std_logic_vector(3 downto 0);
        b : IN std_logic_vector(3 downto 0);
        s : IN std_logic_vector(2 downto 0);
        y : OUT std_logic_vector(3 downto 0)
        );
    END COMPONENT;

    --Inputs
    SIGNAL a : std_logic_vector(3 downto 0) := (others=>'0');
    SIGNAL b : std_logic_vector(3 downto 0) := (others=>'0');
    SIGNAL s : std_logic_vector(2 downto 0) := (others=>'0');

    --Outputs
    SIGNAL y : std_logic_vector(3 downto 0);
```

BE 140- Piyush Sawkar

```
BEGIN

        -- Instantiate the Unit Under Test (UUT)
        uut: alu1447 PORT MAP(
                a => a,
                b => b,
                s => s,
                y => y
        );

        tb : PROCESS
        BEGIN
a<="1100"; b<="0000";
        s<="000";
        wait for 100 ns;
s<="001";
        wait for 100 ns;
        s<="010";
        wait for 100 ns;
                -- Wait 100 ns for global reset to finish
                --wait for 100 ns;

                -- Place stimulus here

                wait; -- will wait forever
        END PROCESS;
END;
```

BE 140- Piyush Sawkar

Function Table:

SEL	Operation Performed
000	No operation
001	Addition (Ain + Bin)
010	Subtraction (Ain – Bin)
011	AND (Ain AND Bin)
100	NAND (Ain NAND Bin)
101	OR (Ain OR Bin)
110	NOR (Ain NOR Bin)
111	XOR (Ain XOR Bin)

Table 6.1

Pin Assignments:

Sr. No.	Signal Name	Pin Designation	Direction	FPGA Pin No.
1	dataout[0]	TL1	Output	P34
2	dataout[0]	TL2	Output	P35
3	dataout[0]	TL3	Output	P36
4	dataout[0]	TL4	Output	P39
5	Cy_bw	TL5	Output	P40
6	Ain[0]	TL9	Input	P47
7	Ain[1]	TL10	Input	P48
8	Ain[2]	TL11	Input	P50
9	Ain[3]	TL12	Input	P60
10	Bin[0]	TL17	Input	P68
11	Bin[1]	TL18	Input	P69
12	Bin[2]	TL19	Input	P74
13	Bin[3]	TL20	Input	P75
14	sel[0]	TL25	Input	P83
15	sel[1]	TL26	Input	P89
16	sel[2]	TL27	Input	P90
17	sel[3]	TL28	Input	P93

Table 6.2

Conclusion: A 4 bit ALU was synthesized and implemented using a behavioural design approach, demonstrating correct functionality & efficient performance on selected hardware platform.

Questions:

1) **What are different modeling styles of VHDL?**
2) **Name few EDA tools for VLSI Design.**
3) **What are different types of HDL?**

Q.1) What are different modelling styles of VHDL ?

→. VHDL supports several modelling styles to describe digital system. These modelling styles are used to capture the behaviour and structure of electronic circuit & system. The main modelling styles are:

1) Behavioural Modelling Style:
It defines the behavior of the system using high level language code. Behavioral consist of one or more process statement.

2) Dataflow modelling Style:
It defines the system in terms of how data flow through the system. Dataflow description consist of one or more concurrent system assignment assignment statements.

3) Structural Modelling Style:
TH structural style of modelling an entity is described as a set of interconnected components.

Q.2) Name few EDA tools for VLSI Design.

→. Electronic Design tutomation (EDA) is a generic name for all methods for entering & performing simulation, synthesis and implementation of electronic circuit diagram.

There are many tools used some famous one are:

i) Aldec Active-HDL
ii) Cadence Incisive
iii) Mentor-Graphics Modelsim
iv) Mentor-Graphics Questa Advanced Simulator.
v) Synopsys VCS-MX.
vi) OrCAD
vii) Xilinx Vivado Design suite.

Q.3) What are different types of HDL.?

→ HDL or Hardware Description language is a specialized computer language used for design, modelling, & simulation of digital & analog circuits. There are several types but most commonly used are:

1) VHDL (VHSIC Hardware Description language)
· VHDL is one of most widely used HDLs. It is known for its versatility & is commonly used for both digital & mixed-signal design.

2) Verilog:
Verilog is another popular HDL used for digital circuits design. It is often favored for its concise syntax & widely used in industry for ASIC (Application-Specific Integrated Circuit).

EXPERIMENT NO.2

Aim: To Synthesis, Simulate and Implement 4-bit universal shift register.

Objective: 1. To understand shifting of a data.

2. To understand simulation on Hardware.

Hardware Platform:

Device Family	Spartan 3E
Device	XC3S400
Package	PQ208

Software Platform:

Top Level Source Type	HDL
Synthesis Tool	Xilinx 9.2i
Simulator	ISE Simulator

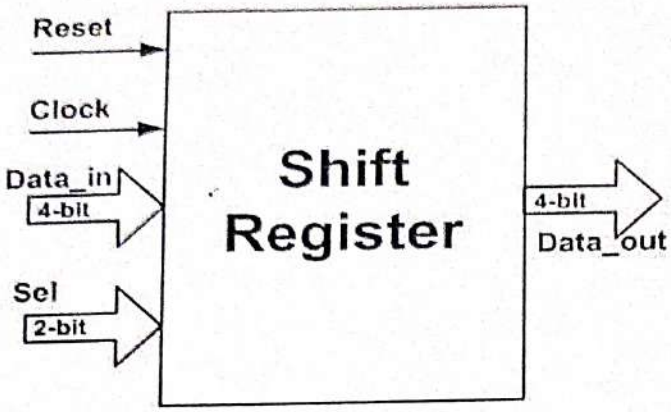
Theory:

This bidirectional shift register is designed to incorporate virtually all of the features a system designe may want in a shift register; they feature parallel inputs, parallel outputs, right-shift and left-shift serial inputs operating-mode-control inputs, and a direct overriding clear line. The register has four distinct modes o operation, namely: Parallel (broadside) load Shift right (in the direction Q A toward Q D) Shift left (in th direction Q D toward Q A) Inhibit clock (do nothing) Synchronous parallel loading is accomplished b applying the four bits of data and taking both mode control inputs, S0 and S1, HIGH. The data is loaded into th associated flip-flops and appear at the outputs after the positive transition of the clock input. During loading serial data flow is inhibited. Shift right is accomplished synchronously with the rising edge of the clock puls when S0 is HIGH and S1 is LOW. Serial data for this mode is entered at the shift-right data input. When S0 i LOW and S1 is HIGH, data shifts left synchronously and new data is entered at the shift-left serial input Clocking of the flip-flop is inhibited when both mode control inputs are LOW.

Features

- Parallel inputs and outputs
- Four operating modes: Synchronous parallel load Right shift Left shift Do nothing
- Positive edge-triggered clocking
- Direct overriding clear

Block Diagram:



Inputs									Outputs			
Clear	Mode		Clock	Serial		Parallel				Q_A	Q_B	Q_C	Q_D
	S1	S0		Left	Right	A	B	C	D				
L	X	X	X	X	X	X	X	X	X	L	L	L	L
H	X	X	L	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
H	H	H	↑	X	X	a	b	c	d	a	b	c	d
H	L	H	↑	X	H	X	X	X	X	H	Q_{An}	Q_{Bn}	Q_{Cn}
H	L	H	↑	X	L	X	X	X	X	L	Q_{An}	Q_{Bn}	Q_{Cn}
H	H	L	↑	H	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	H
H	H	L	↑	L	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	L
H	L	L	X	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}

Truth Table

Pin Assignments:

Sr. No.	Signal Name	Pin Designation	Direction	FPGA Pin No.
1	clk	CLK_12MHz	Input	P183
2	reset	Reset	Input	P58
3	dataout[0]	TL1	Output	P34
4	dataout[0]	TL2	Output	P35
5	dataout[0]	TL3	Output	P36
6	dataout[0]	TL4	Output	P39
7	datain[0]	TL9	Input	P47
8	datain[1]	TL10	Input	P48
9	datain[2]	TL11	Input	P50
10	datain[3]	TL12	Input	P60
11	SL_in	TL17	Input	P68
12	SR_in	TL18	Input	P69
13	sel[0]	TL25	Input	P83
14	sel[1]	TL26	Input	P89

Conclusion: A 4 bit universal shift register was successfully completed, providing versatile data manipulation capabilities. The device met specified requirement & demonstrate chooser hardware platform.

Questions: 1) What are different types of shift register?

2) What are applications of shift register?

3) What is clock skew? What are different types of clock skew?

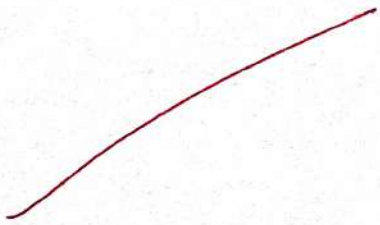
SHIFT REGISTER :

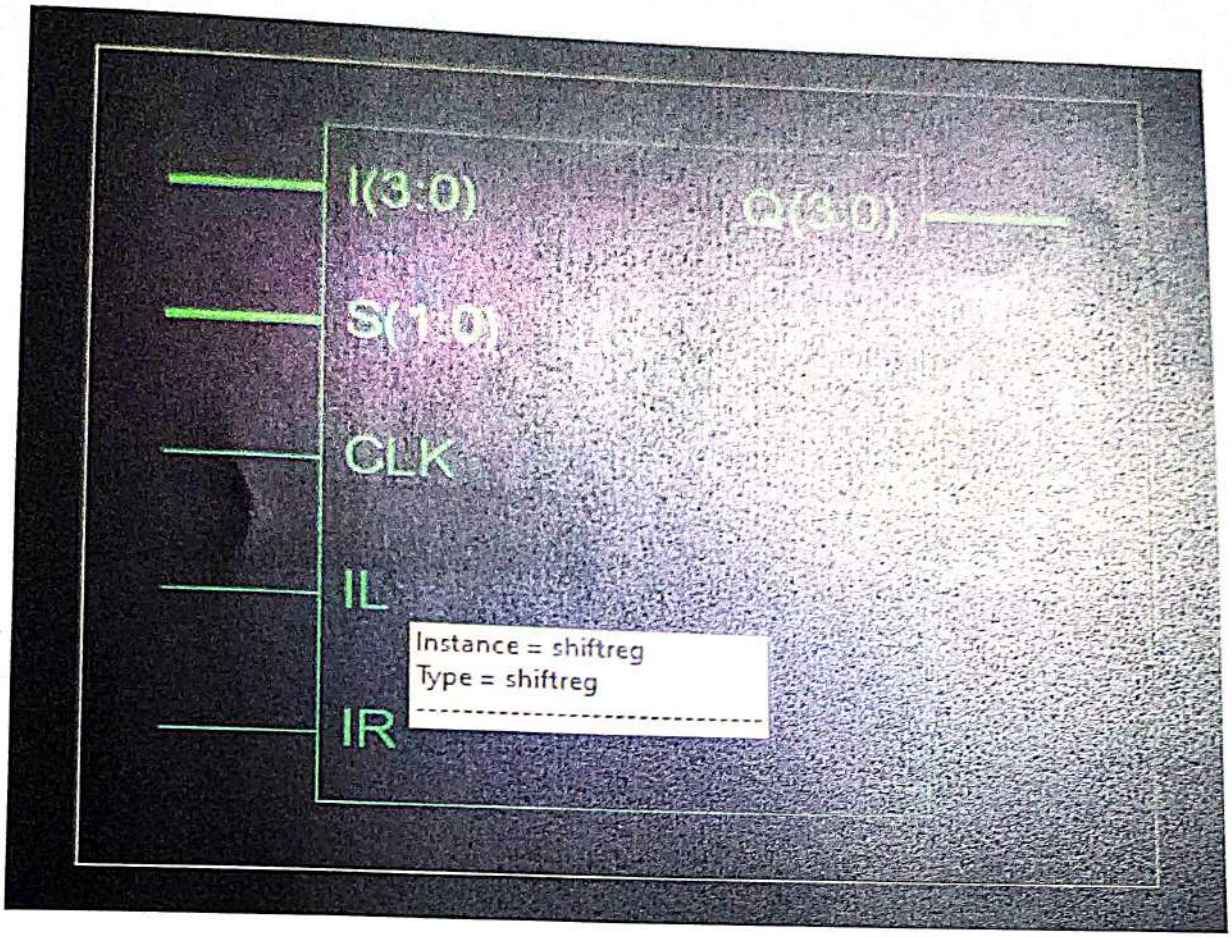
VHDL Code -

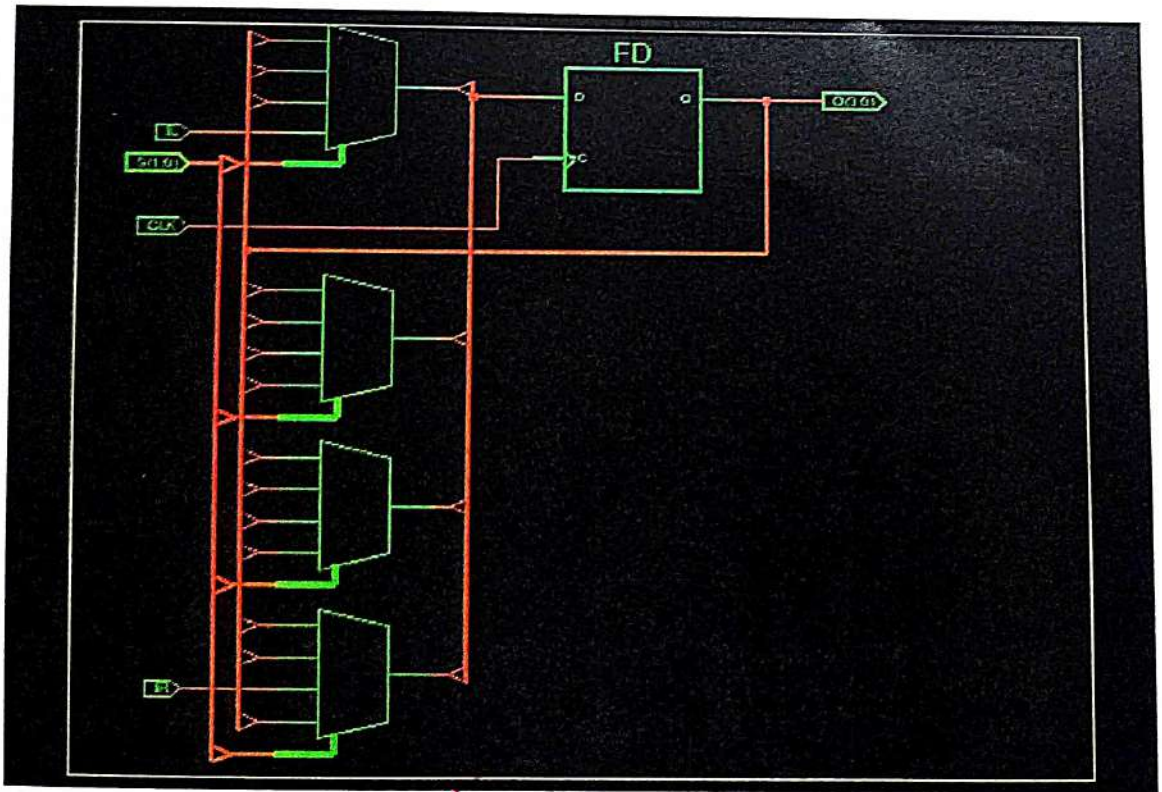
```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    13:25:11 10/27/2023
-- Design Name:
-- Module Name:    shiftreg - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity shiftreg is
    Port ( S : in   STD_LOGIC_VECTOR (1 downto 0);
           CLK : in   STD_LOGIC;
           IL : in   STD_LOGIC;
           IR : in   STD_LOGIC;
           Q : out   STD_LOGIC_VECTOR (3 downto 0);
           I : in   STD_LOGIC_VECTOR (3 downto 0));
end shiftreg;

architecture Behavioral of shiftreg is
signal qtemp:std_logic_vector(3 downto 0);

begin
process(CLK)
begin
if rising_edge(CLK)then
case S is
when "00"=>
```

```
Q<=qtemp;

when "01"=>
Q(3 downto 0)<=I(3 downto 0);

when "10"=>
qtemp(3 downto 1)<=qtemp(2 downto 0);
qtemp(0)<=IR;

when "11"=>
qtemp(2 downto 0)<=qtemp(3 downto 1);
qtemp(3)<=IL;

when others =>NULL;

end case;

end if;
Q<=qtemp;
end process;

end Behavioral;
```




I(3:0)
S(1:0)
CLK
IL
IR
Instance = shiftreg
Type = shiftreg


FD
D
Q
C
IL
CLK
IR

Testbench -

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   13:45:59 10/27/2023
-- Design Name:   shiftreg
-- Module Name:   C:/shiftreg/shifter_vhd.vhd
-- Project Name:  shiftreg
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: shiftreg
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
```

```
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY shifter_vhd_vhd IS
END shifter_vhd_vhd;

ARCHITECTURE behavior OF shifter_vhd_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT shiftreg
    PORT(
         S : IN std_logic_vector(1 downto 0);
         CLK : IN std_logic;
         IL : IN std_logic;
         IR : IN std_logic;
         I : IN std_logic_vector(3 downto 0);
         Q : OUT std_logic_vector(3 downto 0)
        );
  END COMPONENT;

  --Inputs
  SIGNAL CLK :  std_logic := '1';
  SIGNAL IL :  std_logic := '0';
```

```
        SIGNAL IR :   std_logic := '0';
        SIGNAL S :    std_logic_vector(1 downto 0) := (others=>'0');
        SIGNAL I :    std_logic_vector(3 downto 0) := (others=>'0');

        --Outputs
        SIGNAL Q :    std_logic_vector(3 downto 0);

BEGIN

        -- Instantiate the Unit Under Test (UUT)
        uut: shiftreg PORT MAP(
                S => S,
                CLK => CLK,
                IL => IL,
                IR => IR,
                Q => Q,
                I => I
        );

        tb : PROCESS
        BEGIN

                -- Wait 100 ns for global reset to finish
                --wait for 100 ns;

        s<="00";
        I<="1101";
        wait for 100 ns;
```

```
        s<="01";
        I<="1101";
        wait for 100 ns;

        s<="10";
        I<="1101";
        wait for 100 ns;

        s<="11";
        I<="1101";
        wait for 100 ns;

            -- Place stimulus here

            --wait; -- will wait forever
        END PROCESS;

END;
```

Q.1) What are different types of shift register ?

→ 1) There are basically four types of register

1) Serial in serial out shift register.
The shift register which allow serial input one bit after the other through a single data & procedures or serial output is known as serial in serial out & shift register.

2) Serial in ~~seria~~ parallel out shift register.
The shift register which declares serial input one bit after the other through a single data & produces serial output.

3) Serial in parallel register.
The shift registers which allows serial input one bit after the another through a single data & procedures a parallel out is known.

4) Parallel in serial out shift register.
The shift register which allows parallel input data is given seperately to each flipflop & in a simultaneous manner & produces a serial out shift register

Q2. What are application of shift register?

→ 1) They are used for data temporary storage.

2) They are also used for data transfer & data manipulation

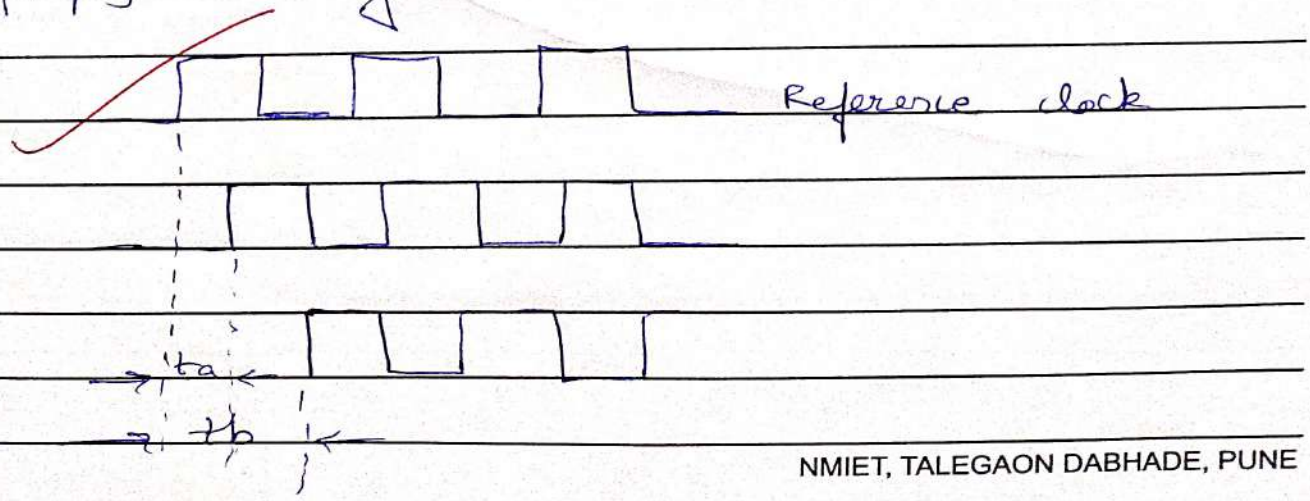
3) The serial in & serial out & parallel in parallel out shift register are used to produce time delay to Digital circuits.

4) A parallel in serial out shift register is used to convert parallel data into serial data.

Q3. What is clock skew? what are different types of skew.

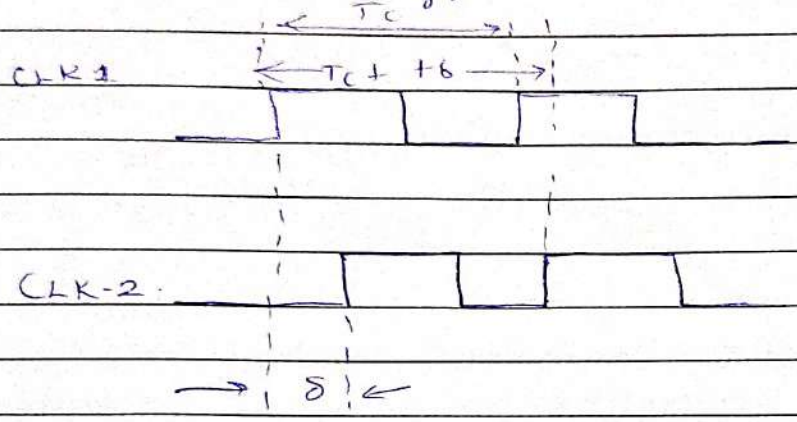
Clock skew is major timing issues in synchronous circuit design.

Clock skew is a phenenon in synchronous digital circuit system in which the same sourced clock signal arrives at different component at different times due to gate is more advanced semiconductor technology wire signal propagation delay



Types of clock skew

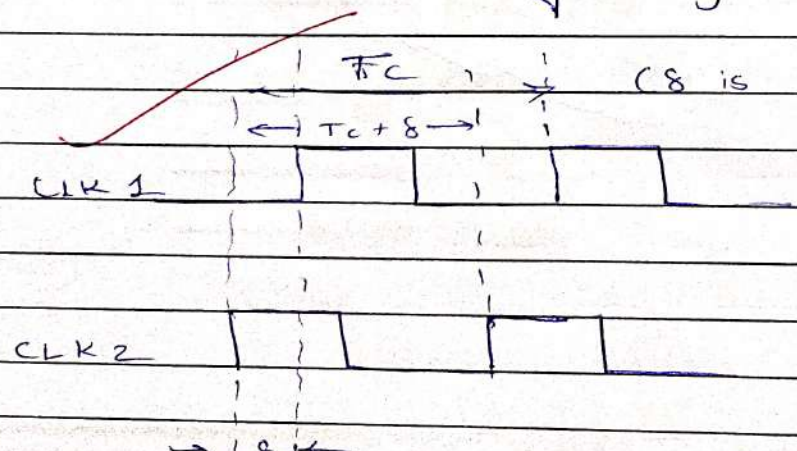
Positive clock skew: When the clock edge is delayed by a positive δ at the other node than it is referred as positive clock skew.



Positive clock skew

Negative clock skew

The transmitting register gets the clock tick later than the receiving register.



Negative clock skew

EXPERIMENT NO.3

Aim: To Synthesis, Simulate and Implement Standard FIFO Memory in FPGA.

Objective: 1. To understand FIFO function and reading/ writing of data.

2. To understand simulation on Hardware.

Hardware Platform:

Device Family	Spartan 3E
Device	XC3S250
Package	PQ208

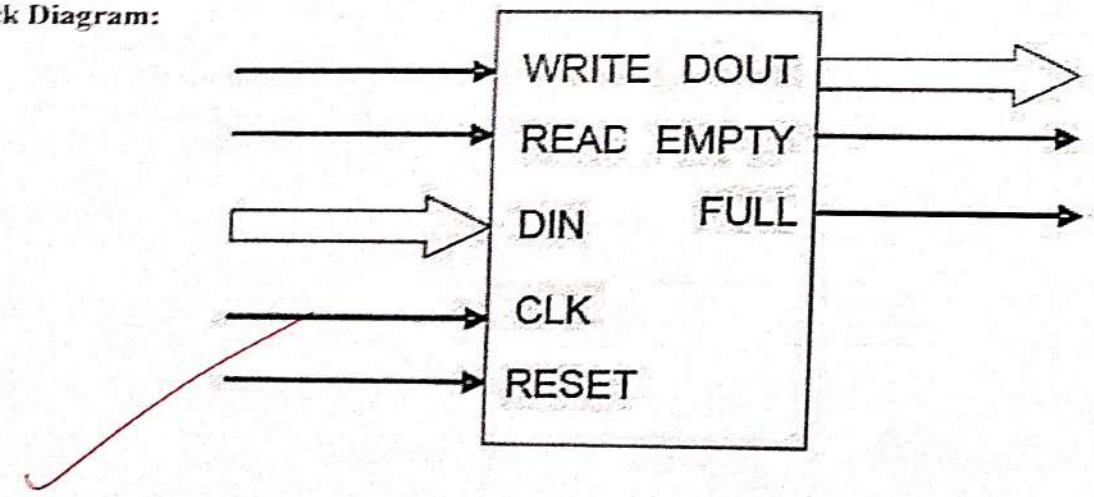
Software Platform:

Top Level Source Type	HDL
Synthesis Tool	Xilinx 9.2i
Simulator	ISE Simulator

Theory:

The FIFO provides first-in-first-out storage functionality to any block or interface that requires it. All I/O is synchronous with clock (including reset). Simultaneous read and write operations may occur. Individual read request (FIFO_read) and write request (FIFO_write) lines and data in and out (datain and dataout) buses are included to allow separate devices to request data write and read access. Two status signals indicate whether or not a read or write may be conducted. FIFO_empty indicates that the FIFO is currently empty: a read request will be ignored if this is true. FIFO_full indicates that the FIFO is *almost* full; a write request will be ignored while this signal is true.

Block Diagram:



Pin Assignments:

Sr. No.	Signal Name	Pin Designation	Direction	FPGA Pin No.
1	CLK	Clock_12Mhz	Input	P183
2	RST	Reset	Input	P58
3	WriteEN	Key0	Input	P162
4	ReadEN	Key1	Input	P142
5	Empty	TL1	Output	P34
6	Full	TL2	Output	P35
7	DataIn(0)	TI17	Input	P68
8	DataIn(1)	TI18	Input	P69
9	DataIn(2)	TI19	Input	P74
10	DataIn(3)	TI20	Input	P75
11	DataOut(0)	TL9	Output	P47
12	DataOut(1)	TL10	Output	P48
13	DataOut(2)	TL11	Output	P50
14	DataOut(3)	TL12	Output	P60

Conclusion: The FIFO memory in an FGPA were carried out efficiently, offering a reliable & efficient data buffering solution. The design met requirement & demonstrated robust performance.

Questions:

1) **What is FIFO Memory?**
2) **What is metastability? What are causes of metastability?**
3) **What is set up and hold time?**
4) **What is clock jitter?**

FIFO MEMORY :

VHDL Code -

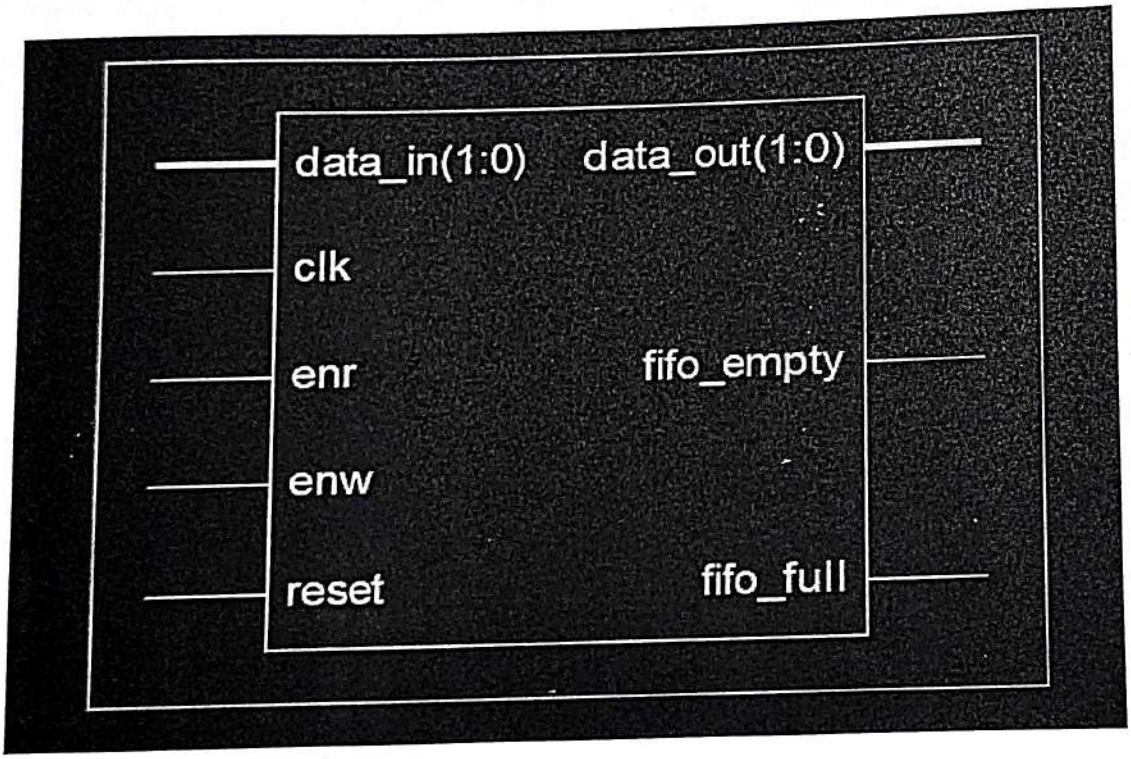
```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:      10:50:22 11/03/2023
-- Design Name:
-- Module Name:      fifo - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

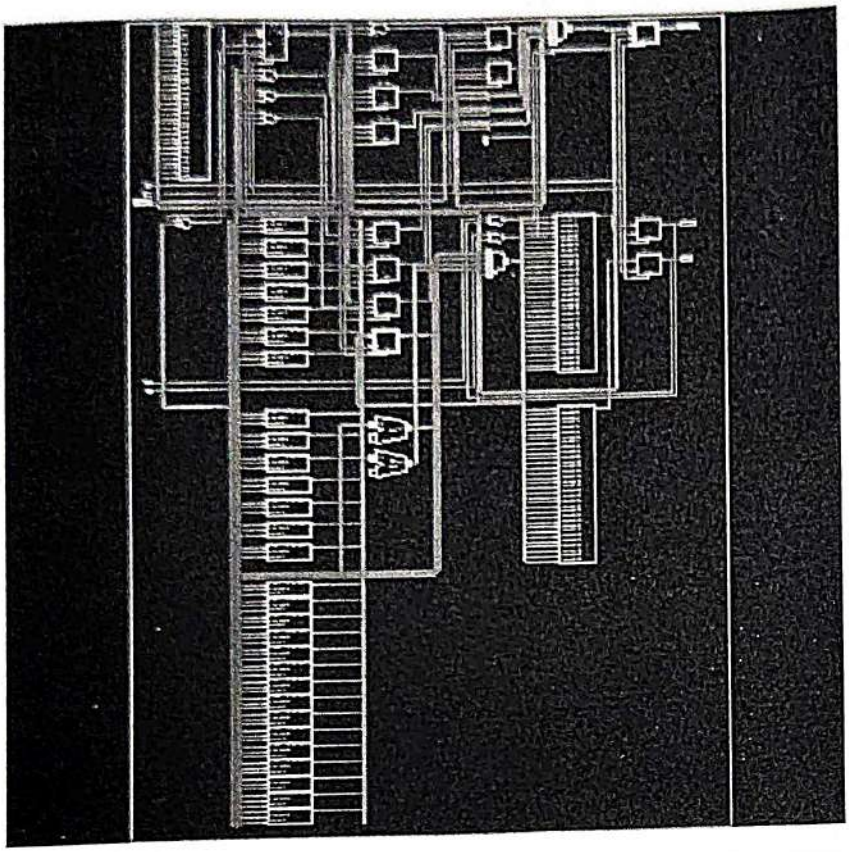
```
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity fifo is
Generic (
constant DATA_WIDTH : positive := 4;
constant FIFO_DEPTH : positive := 4);

    port ( clk : in   STD_LOGIC;
           reset : in   STD_LOGIC;
           enr : in   STD_LOGIC;
           enw : in   STD_LOGIC;
           data_in : in   STD_LOGIC_VECTOR (1 downto 0);
           data_out : out   STD_LOGIC_VECTOR (1 downto 0);
           fifo_empty : out   STD_LOGIC;
           fifo_full : out   STD_LOGIC);
end fifo;

architecture Behavioral of fifo is
```

```
type memory_type is array(0 to FIFO_DEPTH-1) of std_logic_vector(DATA_WIDTH-1 downto 0);
signal memory : memory_type :=(others=>(others =>'0'));
signal readptr,writeptr : integer :=0;
signal empty ,full: std_logic :='0';

begin

fifo_empty <= empty;
fifo_full<=full;

process(clk,reset)
variable num_elem : integer :=0;

begin
if(reset = '1') then

memory <=(others =>(others =>'0'));
data_out<=(others =>'0');
empty <='0';
full<='0';
readptr<=0;
writeptr<=0;
num_elem := 0;
elsif(clk'event and clk ='1') then
if(enr ='1' and empty ='0')then
```

```
data_out<=memory(readptr);
num_elem:=num_elem-1;
end if;

if (enw ='1' and full ='0') then
memory(writeptr)<=data_in;
writeptr<=writeptr +1;
num_elem :=num_elem+1;
end if;

if(readptr = FIFO_DEPTH) then
readptr <= 0;
end if;

if(writeptr = FIFO_DEPTH) then
writeptr <= 0;
end if;

if(num_elem = 0) then
empty <='1';
else
empty <='0';
end if;
if(num_elem = FIFO_DEPTH ) then
full <='1';
```

```
else
full <='0';
end if;
end if;
end process;
end Behavioral;
```





FIFO Testbench -

-- Company:

-- Engineer:

--

-- Create Date: 12:11:25 11/03/2023

-- Design Name: fifo

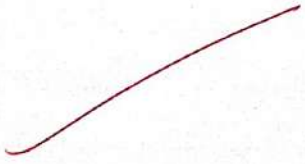
-- Module Name: C:/fifo/fifo_tb.vhd

-- Project Name: fifo

```
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: fifo
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;


ENTITY fifo_tb_vhd IS
```

```
END fifo_tb_vhd;

ARCHITECTURE behavior OF fifo_tb_vhd IS

 --Inputs and outputs
 constant DATA_WIDTH : positive := 4;
 constant FIFO_DEPTH : positive := 4 ;
 signal Clk,reset,enr,enw,empty,full : std_logic :=  '0';
 signal data_in,data_out : std_logic_vector(DATA_WIDTH - 1 downto 0) :=
(others => '0');
 --temporary signals
signal i : integer := 0;
 -- Clock period definitions
constant Clk_period : time := 10 ns;

BEGIN

      UUT : entity work.fifo GENERIC MAP
      (
      DATA_WIDTH => DATA_WIDTH,
      FIFO_DEPTH => FIFO_DEPTH)
      PORT MAP (
            clk => clk,
            reset => reset,
            enr => enr,
```

```
enw <= '1'; enr<= '0'; --write 4 values to fifo.

for i in 1 to 4 loop

Data_In <= conv_std_logic_vector(i,DATA_WIDTH);
wait for clk_period;
end loop;

enw <= '0'; enr <= '1'; --read 4 values from fifo.

wait for clk_period*4;
enw <= '0'; enr <= '0';
wait for clk_period*10; --wait for some clock cycles.

                wait; -- will wait forever
        END PROCESS;

END;
```



~~LCD~~

~~VHDL Code~~ -

Q1. What is FIFO Memory ?

→ The FIFO provides first in first out storage functionality to any block or interface that requires it. All I/O is synchronous with clock simultaneous links read and write operations may occur.
Individual read req (FIFO read), and write req. (FIFO) write line & data in out buses are included to allow seperate devices to request data write & read access.

Q2 What is metastability ? What are causes of metastability ?

→ - Metastability in digital system occurs when two asynchronous signal combine in such a way that their resulting output goes to an intermediate state.
- A common example is the case of data violation the setup & hold specification of latch or a flip flop.

Causes of metastability.

- When the input signal is asynchronous signal.
- When the clock skew is more.
- When interfacing two domain operating at two freequency.
- When combinational delay in such way that it changes flip-flop up in required window seat up+hold time window.

Q3. What is setup & hold time.?

→ Set up:

The setup time is the amount of time required for input to a flipflop to stable before a clock edge.

Hold time:

Hold time is minimum amount of time required for input to a flipflop to stable after a clock edge.

Q4.) What is clock jitter?

→ Clock jitter is a timing variation of set of signal edges from their values jitters in clock signal are typically caused by noise or other disturbance in the system.

Contributing factors including thermal noise, power supply variations, loading conditions, device no supply interference coupled from nearby circuits.

EXPERIMENT NO.4

Aim: To Synthesis, Simulate and Implement LCD Interface using behavioral style.

Objective: 1.To understand how data is display on LCD and different command of LCD display.

2. To understand simulation on Hardware.

Hardware Platform:

Device Family	Spartan 3E
Device	XC3S250
Package	PQ208

Software Platform:

Top Level Source Type	HDL
Synthesis Tool	Xilinx 9.2i
Simulator	ISE Simulator

Theory:

We use LCD display for the messages for more interactive way to operate the system or displaying error messages etc. 16×2 Liquid Crystal Display which will display the 32 characters at a time in two rows (16 characters in one row). Each character in the display of size 5 × 7 pixel matrixes, Although this matrix differs for different 16×2 LCD modules if you take JHD162A this matrix goes to 5×8. This matrix will not be same for all the 16×2 LCD modules. There are 16 pins in the LCD module, the pin configuration is given below

PIN NO	NAME	FUNCTION
1	VSS	Ground pin
2	VCC	Power supply pin of 5V
3	VEE	Used for adjusting the contrast commonly attached to the potentiometer.
4	RS	RS is the register select pin used to write display data to the LCD (characters), this pin has to be high when writing the data to the LCD. During the initializing sequence and other commands this pin should low.
5	R/W	Reading and writing data to the LCD for reading the data R/W pin should be high (R/W=1) to write the data to LCD R/W pin should be low (R/W=0)
6	E	Enable pin is for starting or enabling the module. A high to low pulse of about 450ns pulse is given to this pin.
7-14	DB0-DB7	DB0-DB7 Data pins for giving data(normal data like numbers characters or command data) which is meant to be displayed
15	LED+	Back light of the LCD which should be connected to Vcc

PIN NO	NAME	FUNCTION
16	LED-	Back light of LCD which should be connected to ground.

So by reading the above table you can get a brief idea how to display a character. For displaying a character you should enable the enable pin (pin 6) by giving a pulse of 450ns, after enabling the pin6 you should select the register select pin (pin4) in write mode. To select the register select pin in write mode you have to make this pin high (RS=1), after selecting the register select you have to configure the R/W to write mode that is R/W should be low (R/W=0).

Follow these simple steps for displaying a character or data

- E=1; enable pin should be high
- RS=1; Register select should be high
- R/W=0; Read/Write pin should be low.

To send a command to the LCD just follows these steps:

- E=1; enable pin should be high
- RS=0; Register select should be low
- R/W=1; Read/Write pin should be high.

Commands: There are some preset commands which will do a specific task in the LCD. These commands are very important for displaying data in LCD. The list of commands given below

Command	Function
0F	For switching on LCD, blinking the cursor.
1	Clearing the screen
2	Return home.
4	Decrement cursor
6	Increment cursor
E	Display on and also cursor on
80	Force cursor to beginning of the first line
C0	Force cursor to beginning of second line
38	Use two lines and 5x7 matrix
83	Cursor line 1 position 3
3C	Activate second line
0C3	Jump to second line position 3
0C1	Jump to second line position1

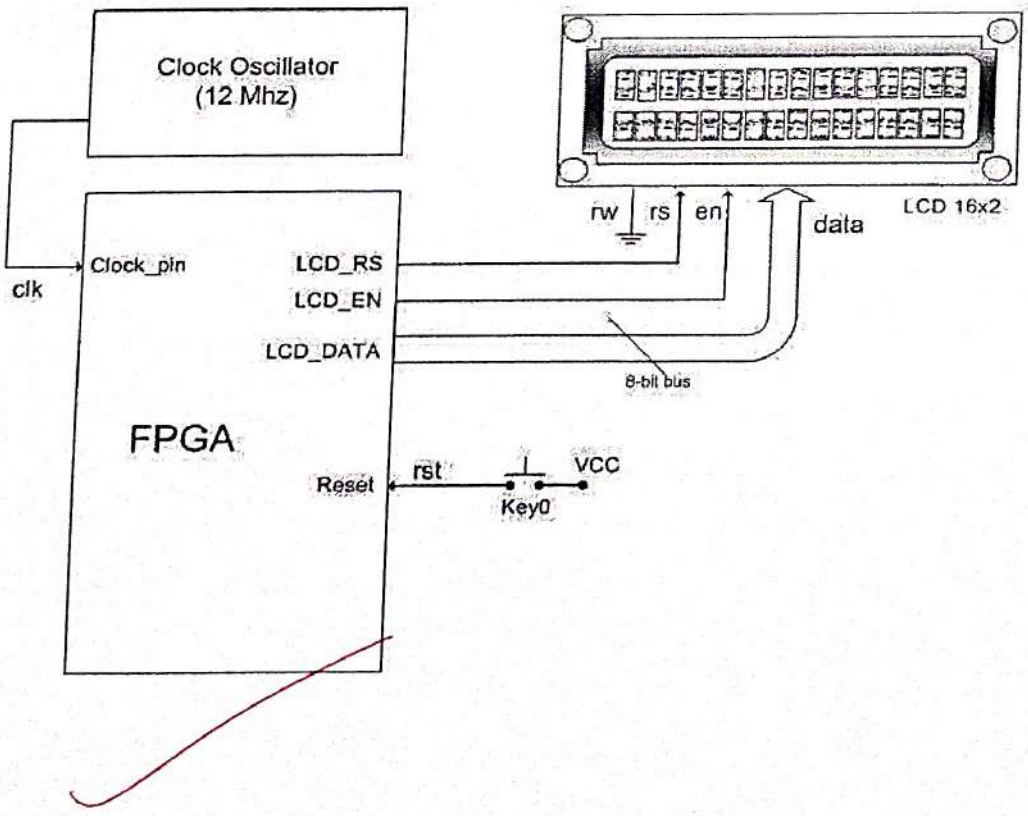
Initializing LCD: To initialize LCD to the 8051 the following instruction and commands are to be embed in to the functions

- 0x38 is used for 8-bit data initialization.
- 0xFH for making LCD on and initializing the cursor.
- 0X6H for incrementing the cursor which will help to display another character in the LCD
- 0x1H for clearing the LCD.

Sending Data to the LCD:

- E=1: enable pin should be high
- RS=1: Register select should be high for writing the data
- Placing the data on the data registers
- R/W=0: Read/Write pin should be low for writing the data.

Block Diagram:



Pin Assignments:

Sr. No.	Signal Name	Pin Designation	Direction	FPGA Pin No.
1	clk_12Mhz	Clock_12Mhz	Input	P183
2	reset	RESET	Input	P58
3	lcd_rs	RS Signal	Output	P203
4	lcd_en	Enable Signal	Output	P202
5	lcd_data[0]	Data0 Signal	Output	P200
6	lcd_data[1]	Data1 Signal	Output	P199
7	lcd_data[2]	Data2 Signal	Output	P197
8	lcd_data[3]	Data3 Signal	Output	P196
9	lcd_data[4]	Data4 Signal	Output	P193
10	lcd_data[5]	Data5 Signal	Output	P192
11	lcd_data[6]	Data6 Signal	Output	P190
12	lcd_data[7]	Data7 Signal	Output	P189

Conclusion: In the pratical we succesfully achieved, enabling seamless communication between FGPA & LCD display.

BVachhad

uestions: 1) What is difference between FPGA and CPLD?

2) What are different programming techniques of FPGA?

3) Draw ASIC design flow?

4) What are different types of ASIC?

LCD

VHDL code:-

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    11:38:38 11/03/2023
-- Design Name:
-- Module Name:    lcd - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
```

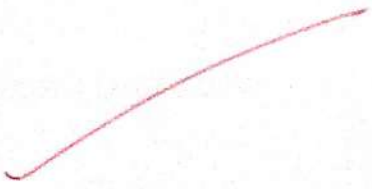
```
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity lcd is
    Port ( rst : in   STD_LOGIC;
           clk_12Mhz : in   STD_LOGIC;
           lcd_rs : out   STD_LOGIC;
           lcd_en : out   STD_LOGIC;
           lcd_data : out   STD_LOGIC_VECTOR (7 downto 0));
end lcd;

architecture Behavioral of lcd is
signal div : std_logic_vector(20 downto 0); --- delay timer 1
signal clk_fsm,lcd_rs_s: std_logic;
-- LCD controller FSM states
type state is (reset,func,mode,cur,clear,d0,d1,d2,d3,d4,hold,start);
signal ps,nx : state;
signal dataout_s : std_logic_vector(7 downto 0);

begin

process(rst,clk_12Mhz)

begin
```

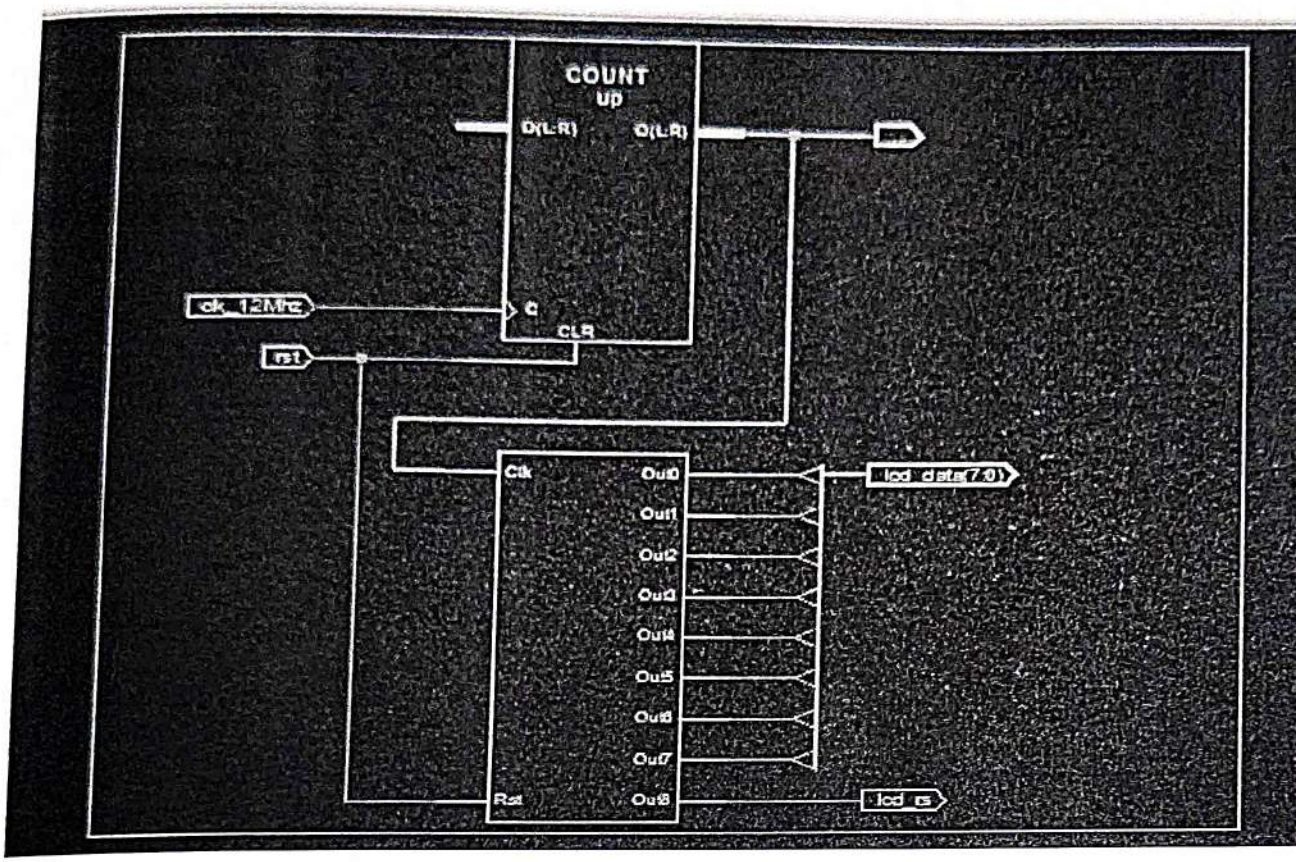
```
if(rst =  '1')then
div <= (others=>'0');
elsif( clk_12Mhz' event and clk_12Mhz ='1')then
div <= div + 1;
end if;
end process;

clk_fsm <= div(15);
----- Presetn state Register ----------
process(rst,clk_fsm)

begin

if(rst =  '1')then
ps <= reset;
elsif(clk_fsm' event and clk_fsm ='1')then
ps <= nx;
end if;
end process;
----- state and output decoding process
process(ps)

begin
case(ps) is
when reset =>
```

```
nx <= func;
lcd_rs_s <=    '0';
dataout_s <= "00111000"; -- 38h

when func =>

nx <= mode;
lcd_rs_s <= '0';
dataout_s <= "00111000"; -- 38h

when mode =>

nx <= cur;
lcd_rs_s <= '0';
dataout_s <= "00000110"; -- 06h

when cur =>

nx <= clear;
lcd_rs_s <= '0';
dataout_s <= "00001100";

when clear=>
```

```
nx <= start;
lcd_rs_s <= '0';
dataout_s <= "00000001"; -- 01h

when start=>

nx <= d0;
lcd_rs_s <= '0';
dataout_s <="11000100"; -- c4

when d0 =>

lcd_rs_s <= '1';
dataout_s <= "01000001"; -- A

nx <= d1;

when d1 =>

lcd_rs_s <= '1';
dataout_s <= "01000010"; -- B
nx <= d2;

when d2 =>
```

```
nx <= reset;
lcd_rs_s <= '0';
dataout_s<="00000001"; -- CLEAR

end case;
end process;
lcd_en <= clk_fsm;
lcd_rs <= lcd_rs_s;
lcd_data <= dataout_s;

end Behavioral;
```





Testbench Code -

-- Company:

-- Engineer:

--

-- Create Date: 12:32:34 11/03/2023

-- Design Name: lcd

-- Module Name: C:/lcd/lcd_tb.vhd

-- Project Name: lcd

-- Target Device:

-- Tool versions:

-- Description:

```
--
-- VHDL Test Bench Created by ISE for module: lcd
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY lcd_tb_vhd IS
END lcd_tb_vhd;

ARCHITECTURE behavior OF lcd_tb_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT lcd
```

```
    PORT(
            rst : IN std_logic;
            clk_12Mhz : IN std_logic;
            lcd_rs : OUT std_logic;
            lcd_en : OUT std_logic;
            lcd_data : OUT std_logic_vector(7 downto 0)
            );
    END COMPONENT;

    --Inputs
    SIGNAL rst :   std_logic := '1';
    SIGNAL clk_12Mhz :   std_logic := '1';

    --Outputs
    SIGNAL lcd_rs :   std_logic;
    SIGNAL lcd_en :   std_logic;
    SIGNAL lcd_data :   std_logic_vector(7 downto 0);
    constant clk_12Mhz_period : time :=10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: lcd PORT MAP(
            rst => rst,
            clk_12Mhz => clk_12Mhz,
            lcd_rs => lcd_rs,
            lcd_en => lcd_en,
            lcd_data => lcd_data
```

```
        );

        clk_12Mhz_process : PROCESS
        BEGIN

clk_12Mhz <='0';
wait for clk_12Mhz_period/2;
clk_12Mhz <='1';
wait for clk_12Mhz_period/2;
end process ;

stim_proc:process
begin
rst <='1';
wait for 20 ns;

rst <='0';

                wait; -- will wait forever
        END PROCESS;

END;
```

Q1. What is difference between FPGA & CPLD.

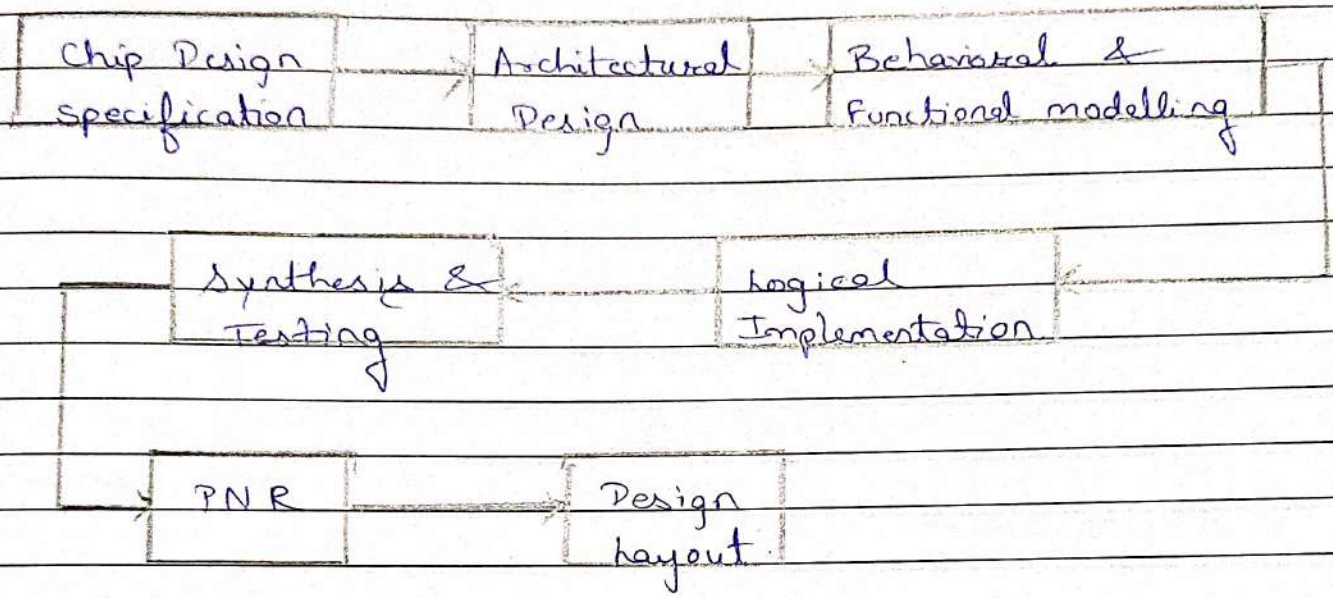
	CPLD	FGPA
1)	CPLD stands for complex programming logic design	FGPA stands for field programmable Gate Array.
2)	It has coarse grain architecture.	It has fine grain architecture.
3)	It is non-volatile.	It is volatile.
4)	It consumes less power as compared to FGPA.	It consumes more power
5)	CPLD's are better for simpler applications.	FGPA's are more suitable for complex applications.
6)	CPLD's are much cheaper than FGPA.	FGPA are more expensive than CPLD.

Q2. What are different programming techniques of FGPA.

→ There are main 3 types of programming for FGPA's are,

i) Static RAM programming
ii) Anti-fuse programming
iii) EPROM / EEROM programming

3. Draw ASIC design flow ?



4. what are different types of ASIC ?

Types of ASIC:

ASIC

i) Programmable
   a) FPGA's
   b) PLD's

ii) Semi custom
   a) Gate based Array
   b) structured
   c) Channel less

B) Std cell based

iii) Full custom.

EXPERIMENT NO.5

Title: To study and implement CMOS Inverter gate simulation and layout using Microwind Software.

Aim:

A. To prepare the layout for CMOS inverter for the different W/L ratios and Comment on the effect of change in width on the power dissipation with capacitor and without capacitor.

B. To prepare the layout for NAND and NOR gate comment on power dissipation with capacitor and without capacitor.

Objective: 1. To understand working of CMOS inverter and different regions of operations of Inverter.

2. To understand working of NAND and NOR gate.

3. To understand operation of half adder and full adder.

Specifications:

Technology: 0.12um

Supply voltage: 1.2 V

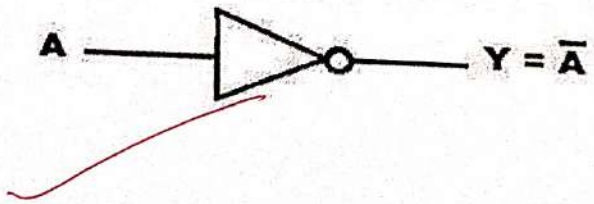
Software Platform: Microwind 3.5

Theory:

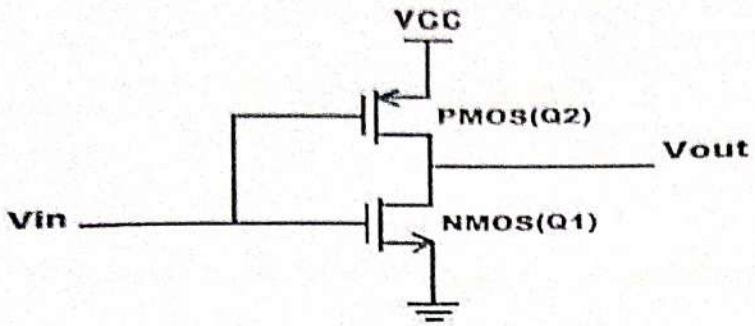
A] CMOS Inverter

CMOS Inverter/NOT gate is considered to be the heart of VLSI circuits, based on the understanding of NOT gate we can extend it easily to NAND and NOR gates which are the basic building blocks of more complex circuits e.g. multipliers and microprocessors, a NOT gate can be implemented using two FETs i.e. a pFET and an nFET both connected in series, in which Vdd is supplied to pFET and nFET is grounded, input Vin is applied to the gate terminals of both and the output is obtained at node Vout.

Design/ Diagram/Circuit:



Logic Gate



Truth Table:

V_{in}	Q1	Q2	V_{out}
0(L)	OFF	ON	5(H)
5(H)	ON	OFF	0(L)

MOS & PMOS transistor are used together in a complementary way to CMOS logic. The power supply oltage V_{DD}. typically may be in the range 1.6V & is after set at 5V for compatibility with older TTL logic ccording to functional behavior of the CMOS inverter circuit can be characterized by just two cases.

1. V_{IN} in 0V, the bottom n-channel transistor Q1 is OFF since its V_{gs} is 0 but top p-channel transistor Q2 is ON, since its V_{gs} is a large negative value (-5V).Therefore, Q2 presents only a small resistance between the power supply terminals (V_{DD} +5V) and the output terminals (V_{out}), and the output voltage is 5V.
2. V_{IN} is 5V, Then Q1 is ON since its V_{gs} is a large positive value (+5V), but Q2 is OFF since its V_{gs} is 0 Thus, Q1 represents a small resistance between the output terminal and ground and the output voltage is 0V

The mobility of the holes is less than that of electrons, and in CMOS inverter pFET is responsible for the conduction of current leading a logic "1" at the output, while nFET is responsible for the conduction of current leading a logic "0" at the output. This means that the gate delay form low to high will be greater than the gate delay for low to high voltage.

B] NAND and NOR Gate:

The circuit diagram of NAND and NOR gate is shown in fig. Now consider the worst case for the NMOS (pull down) section i.e. when both NMOS transistor are ON, R_n is given by,

$$R_n = (1/w)\, r_{eq} * (1/\beta_n)$$

As two NMOS are in series, r_{eq} is given by,

$$(W/L)_{eq} = (l_n/w_n) + (l_n / w_n) = (2 l_n / w_n)$$

Now for PMOS,

As two PMOS are in parallel. So for the worst case, we need to consider only one PMOS. Thus,

$$R_p = (l_p/w_p * \beta_p)$$

Now consider the total path for the current i.e. path with the maximum value of resistance R. Fo the worst case we need some values of R_n & R_p.

$$R_n = R_p$$

$$(2 l_n / w_n) = (1/ \beta_n) = (l_p/w_p * \beta_p)$$

Assume, $l_n = l_p = 2$, $\beta_n = 2.5\, \beta_p$.

Therefore, $(w_p / w_n) = (\beta_n /2\, \beta_p) = (2.5\, \beta_p /2\, \beta_p)$

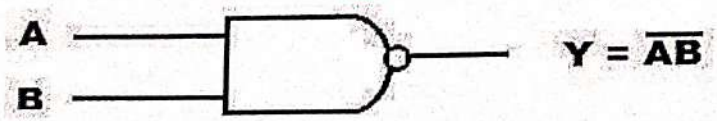
$$w_p = 1.25\, w_n$$

Width of PMOS transistor = 1.25 time more than NMOS transistor if length are kept same.

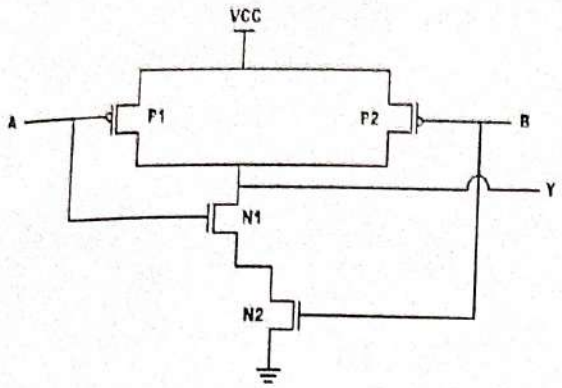
A) NAND Gate:

NAND gate can be implemented using four FETS i.e. two PMOS and two NMOS as the inputs o the gates two. PMOS are connected in parallel while NMOS are connected in series; V_{dd} is supplied to th parallel combination of PMOS while the series combination of NMOS is grounded. Inputs a & b ar applied to the gate terminals of all FETs, and the output f is obtained from the common junction of thes series and parallel combinations as illustrated in NAND circuit under the heading of Desig Diagram/Circuit.

Design Diagram/Circuit:



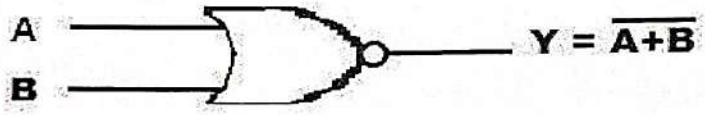
Logic Gate



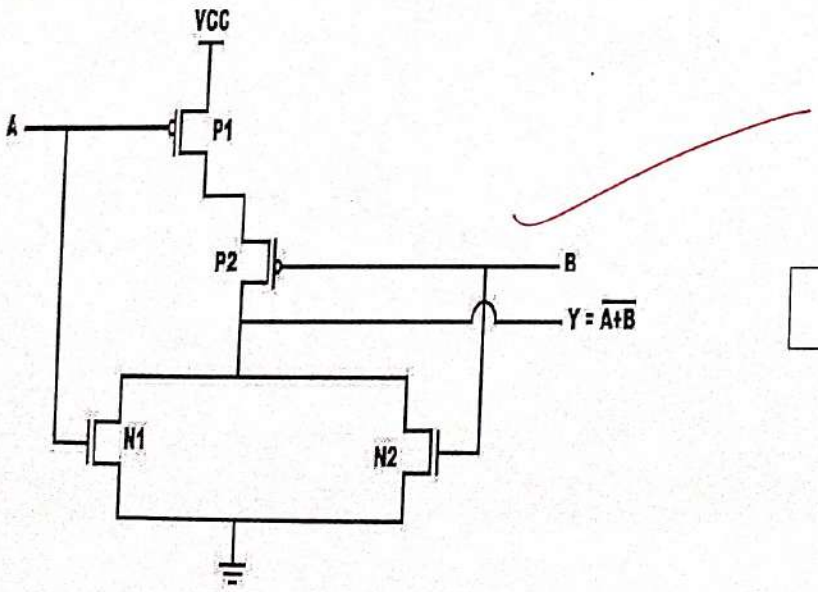
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

B) NOR Gate

A NOR gate can be implemented using four FETS i.e. two PMOS and two NMOS as the inputs of the gate is two. PMOS are connected in series while NMOS are connected in parallel; V_{dd} is supplied to the series combination of PMOS while the parallel combination of NMOS is grounded. Inputs a & b are applied to the gate terminals of all FETs, and the output f is obtained from the common junction of these parallel and series combinations as illustrated in NOR circuit under the heading of Design Diagram/Circuit.



DesignDiagram/Circuit:

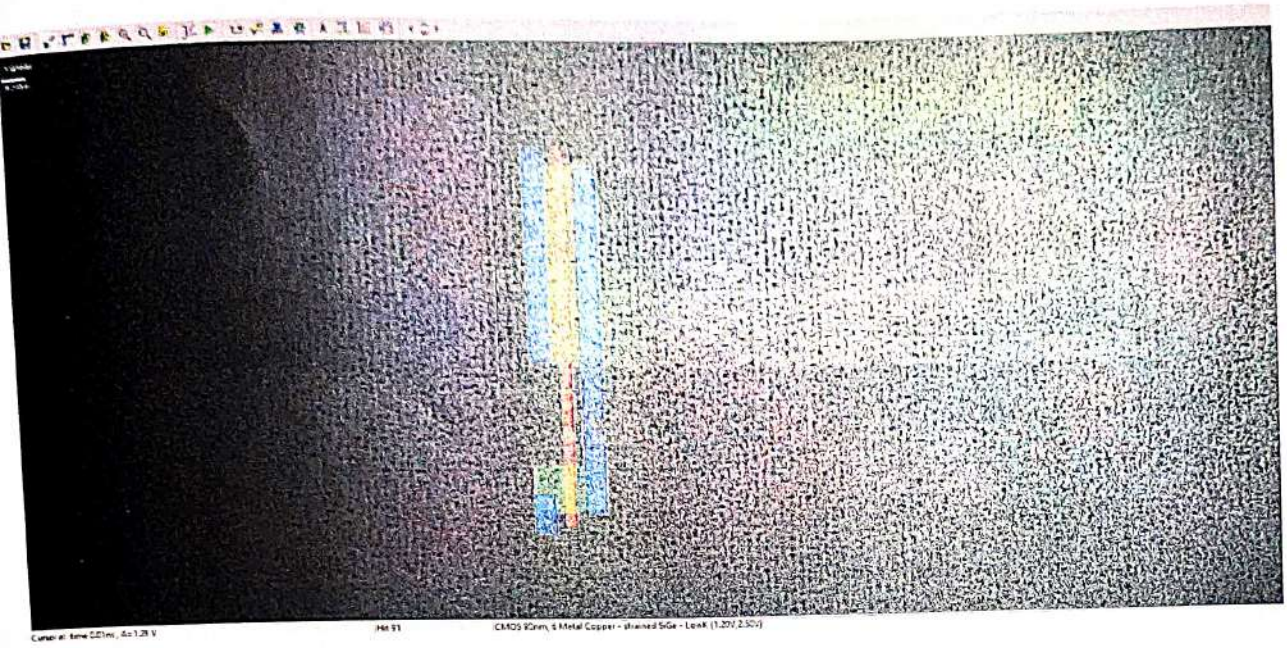


CMOS NOR

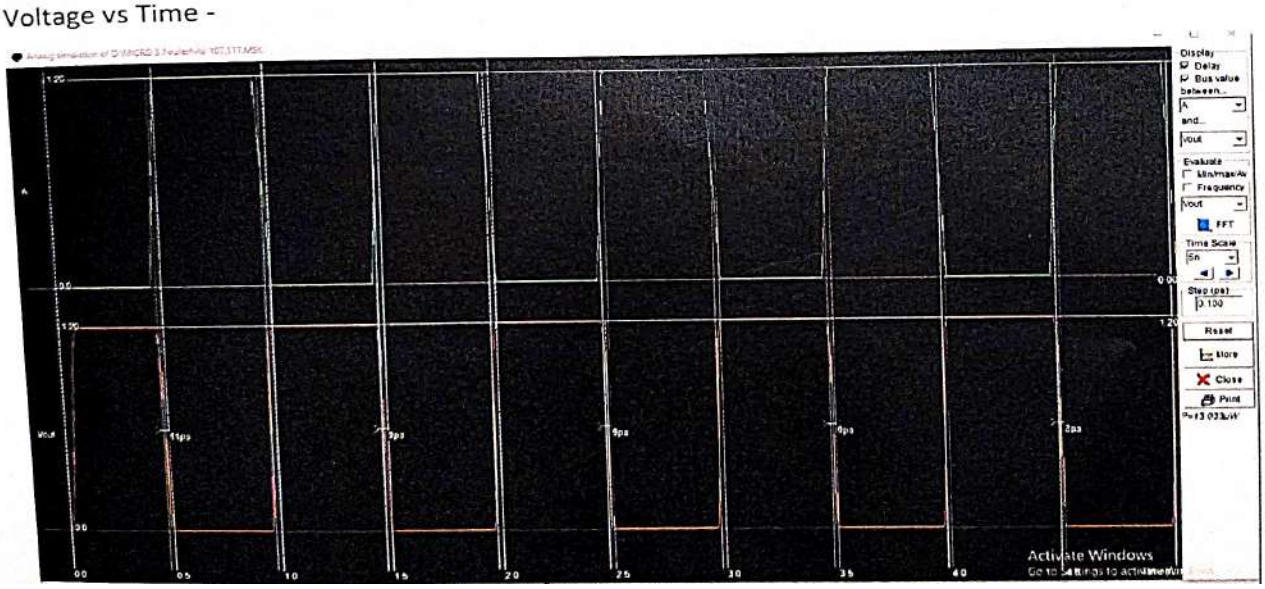
E 140- Piyush Sawkar

CMOS Inverter Gate :

A) CMOS Inverter :

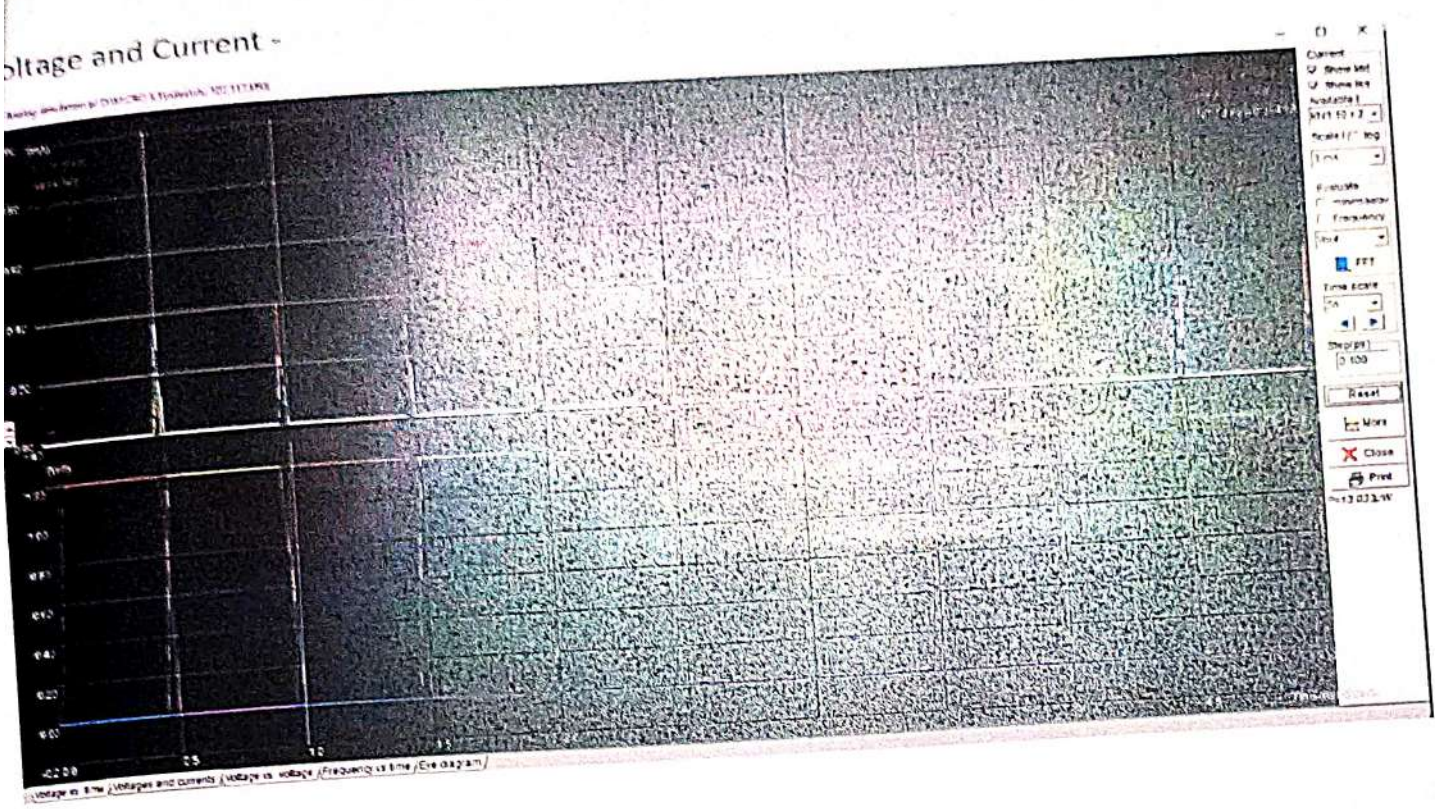


Voltage vs Time -

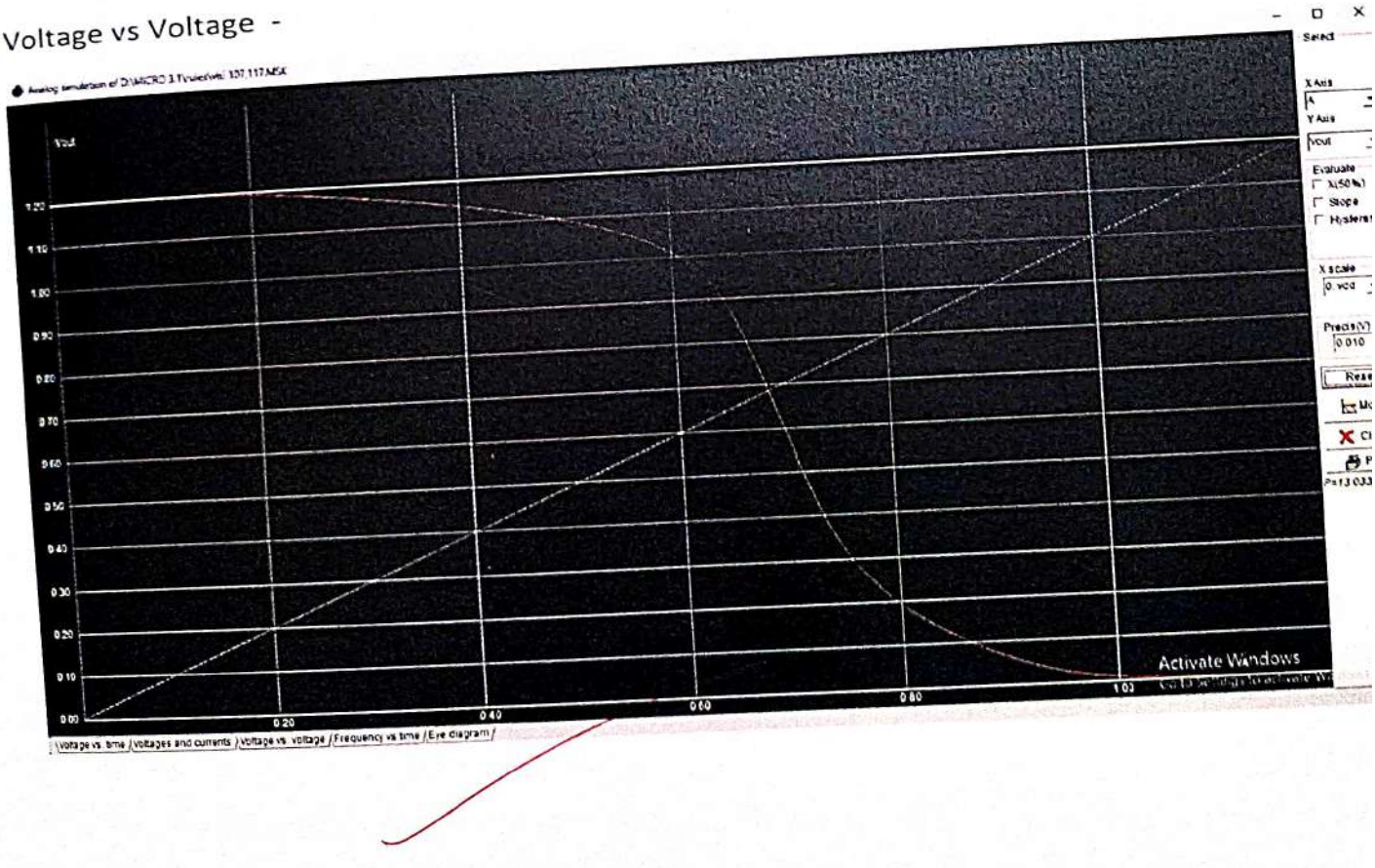


140- Piyush Sawkar

oltage and Current -

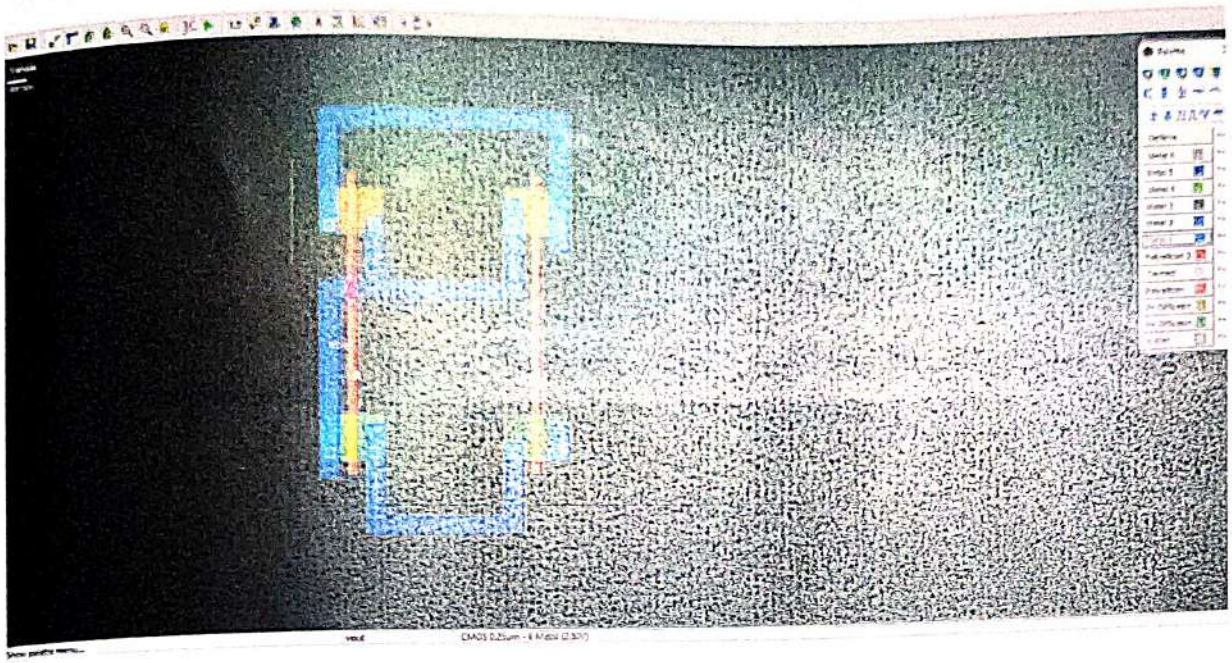


Voltage vs Voltage -

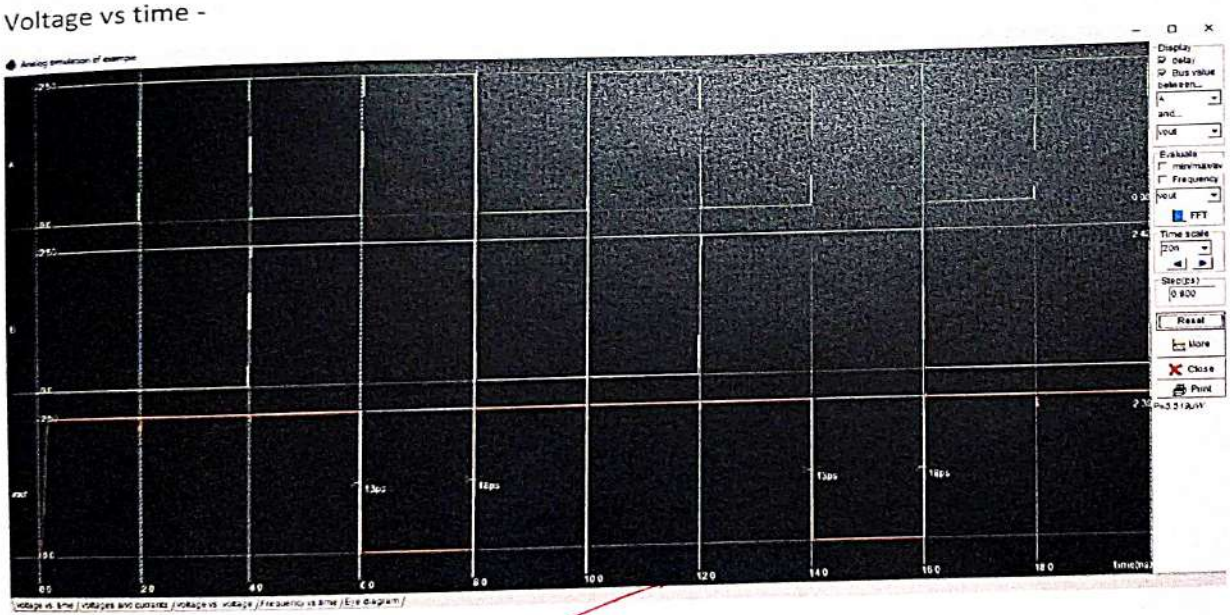


BE 140- Piyush Sawkar

B) CMOS NAND Gate :

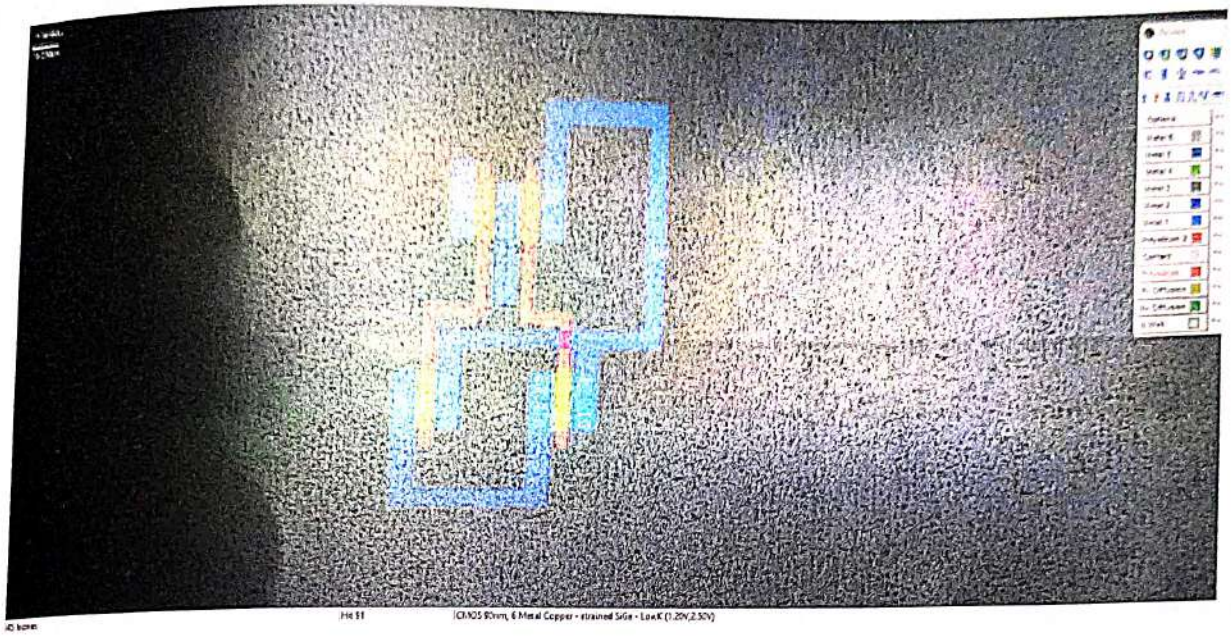


Voltage vs time -

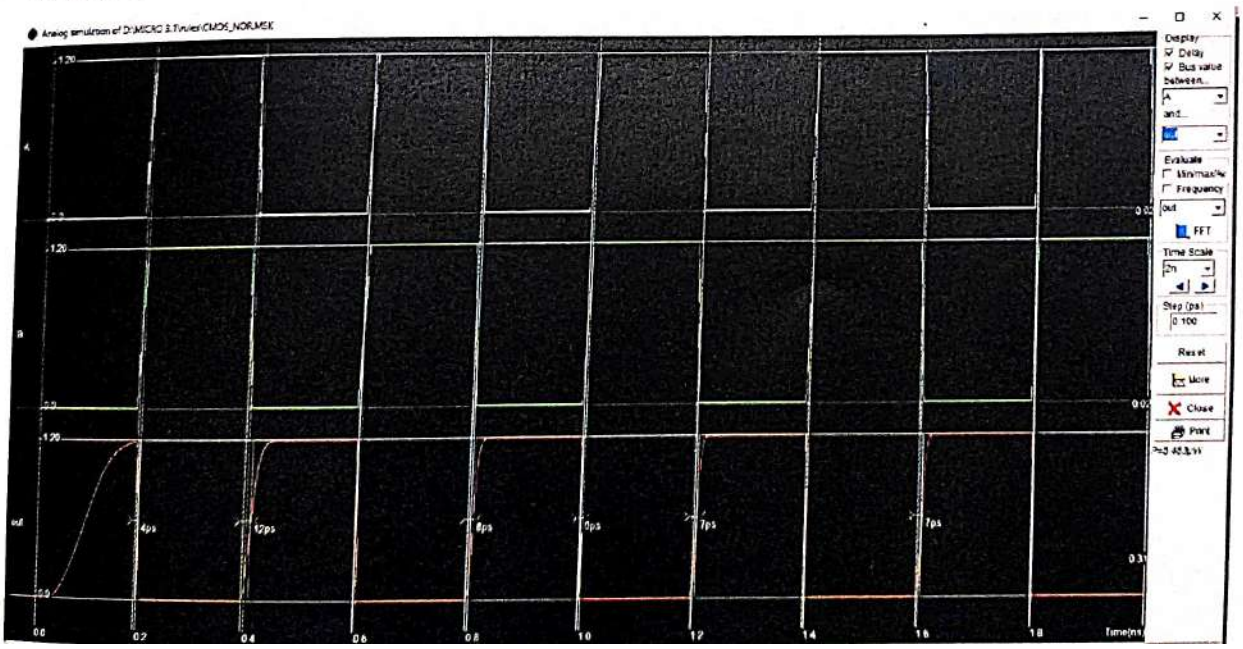


BE 140- Piyush Sawkar

C) CMOS NOR Gate :



Voltage vs time –



Q.1) What are difference between pull up & pull down network ?

→ 1) Pull-Up Network :

a) A pull-up network is used to ensure that a node remains at a logic High (1) state when it is not actively driven by any signal. It consist of component that help to source current to the node.

b) Pull-up network are often used when the default state of a signal should be High, and it should only go low when an active signal pulls it down.

2) Pull-down Network.

a) A pull-down network is used to ensure that a node remains at a logic low (0) state when it is not actively driven by any signal. It consist of component that help to sink current from the node.

b) Pull Down network are often used when default state of a signal should be low, and it should only go High when active signal pulls it up.

Q.2) What is De Morgan's theorem ?

→ De Morgan's Theorem is fundamental concept in

digital & VLSI design. It provides set of two important mathematical principles that are used to manipulate & simplify logical expressions. These principle that are used to named after British mathematician and logician Augustus De Morgan.
De Morgan's Theorem states the following two principles:

1) De Morgan's First Theorem: (The Complement of a Product)
. the complement of product of two or more variable is equal to sum of their complements.

$\sim(A * B * C) = \sim A + \sim B + \sim C$.

2) De Morgan's Second Theorem: (The Complement of Sum).
. The complement of sum of two or more variable is equal to product of their complements.

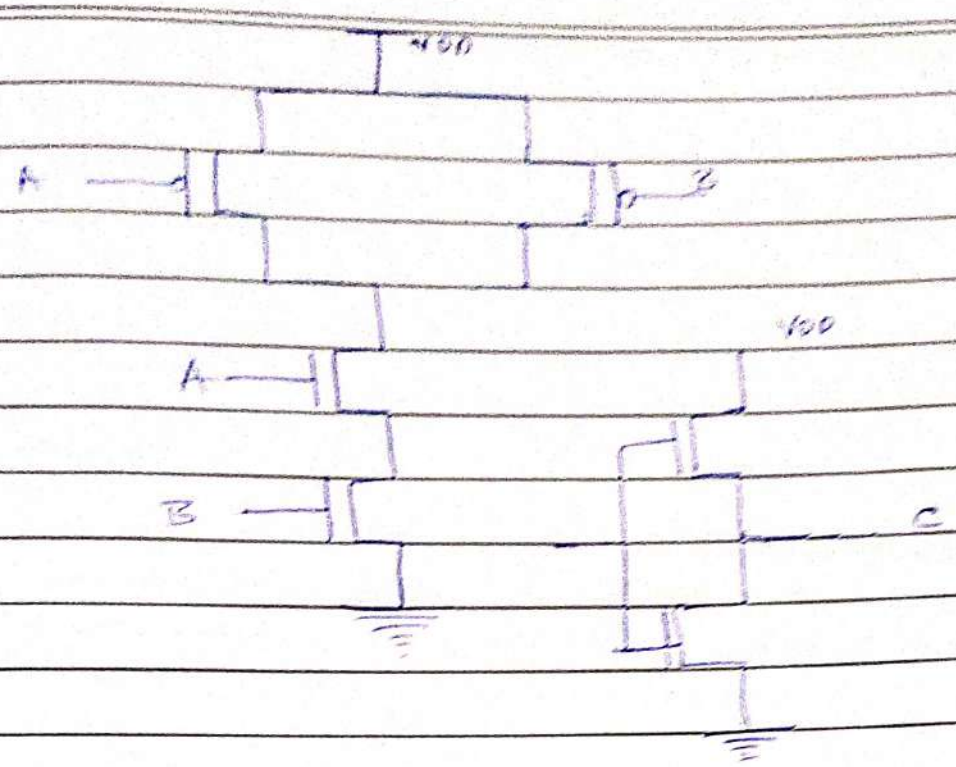
$\sim(A + B + C) = \sim A * \sim B * \sim C$.

$\overline{AB} = \overline{A} + \overline{B}$

Q.3) Implement AND, OR gate using CMOS inverter.

Truth Table

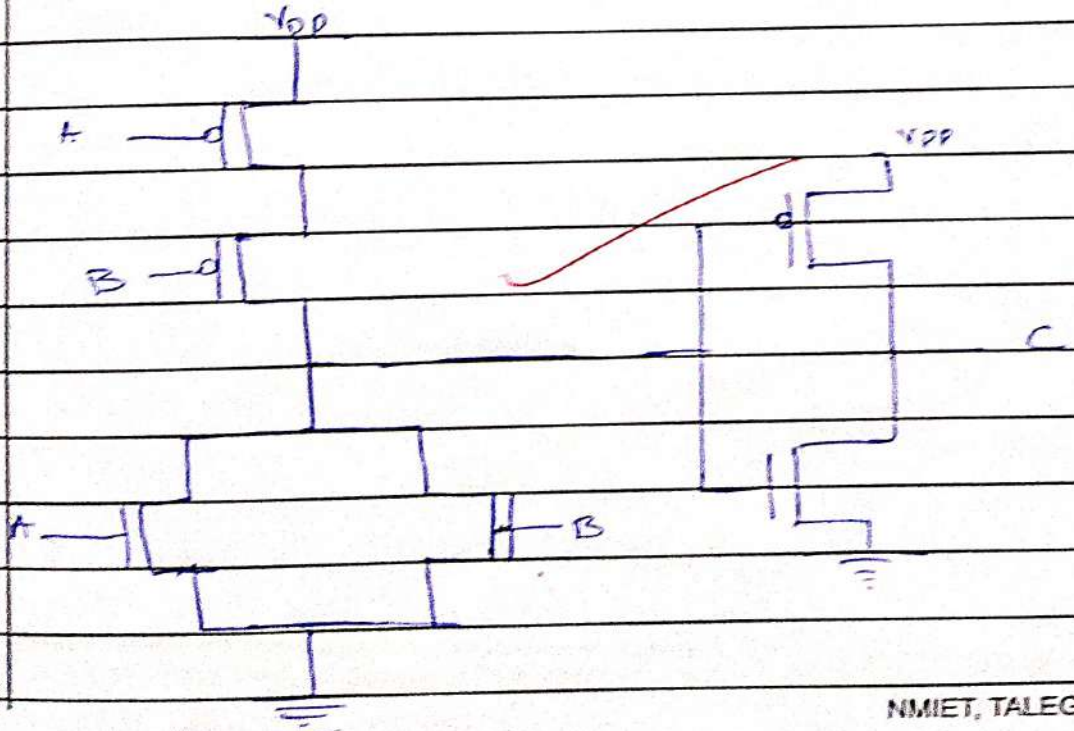
VA	VB	Vout NAND	V-out = Vout NAND
low	low	High	low
low	high	High	low
high	low	High	low
high	high	low	high.



OR GATE

Y = A+B (OR)

PDN → Y = A+B , PUN → Dual of PDN = A·B



EXPERIMENT NO.6

Aim:

To study and implement Half adder and Full adder simulation and layout using Microwind Software.

Objectives: To prepare the layout for Half adder

Software Platform: Microwind 3.5

Theory:

Half Adder

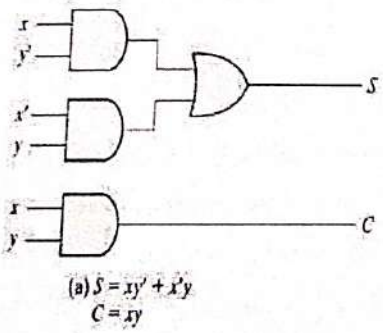
From the verbal explanation of a half adder, we find that this circuit needs two binary inputs and two binary outputs. The input variables designate the augend and addend bits; the output variables produce the sum and carry. We assign symbols x and y to the two inputs and S (for sum) and C (for carry) to the outputs. The truth table for the half adder is listed in Table 4.3 . The C output is 1 only when both inputs are 1.

The S output represents the least significant bit of the sum.

The simplified Boolean functions for the two outputs can be obtained directly from the truth table. The simplified sum-of-products expressions are

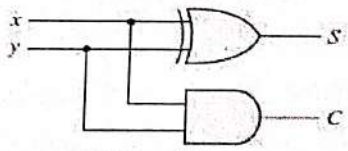
$$S = x'y + xy' \quad C = xy$$

The logic diagram of the half adder implemented in sum of products is shown in Fig (a). It can be also implemented with an exclusive-OR and an AND gate as shown in Fig.(b).



(a) $S = xy' + x'y$
$C = xy$

Fig. (a)



(b) $S = x \oplus y$
$C = xy$

Fig. (b)

Truth Table

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full Adder

Full adder is also a combinational logic circuit that can add two binary digits (bits) and a carry bit, and produce a sum bit and a carry bit as output.

In other words, a combinational circuit which is designed to add three binary digits and produces two outputs (sum and carry) is known as a full adder. Thus, a full adder circuit adds three binary digits, where two are the inputs and one is the carry forwarded from the previous addition. The block diagram and circuit diagram of the full adder are shown in Figure1 below

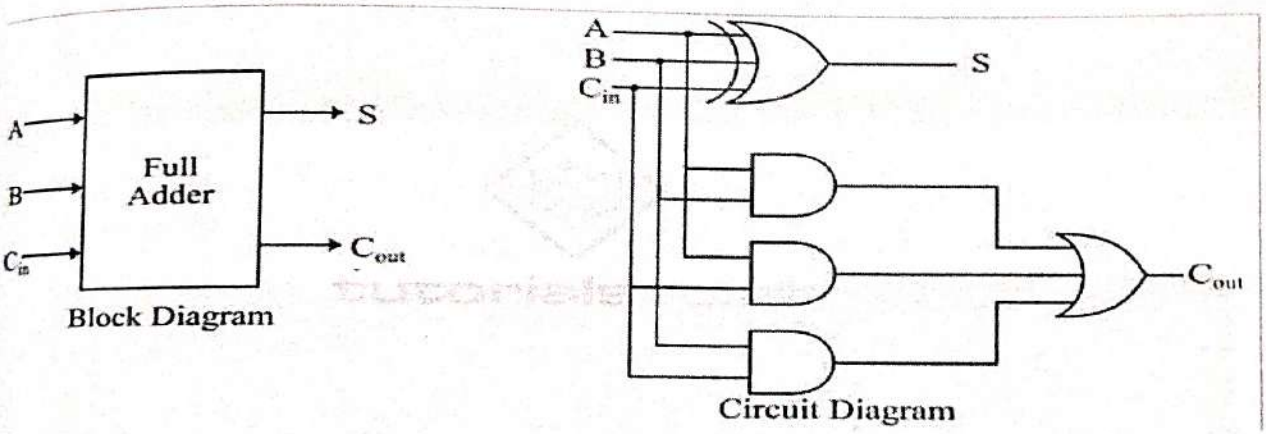


Fig 1. Full Adder

It is clear that the logic circuit of a full adder consists of one XOR gate, three AND gates and one OR gate, which are connected together as shown in Figure-1. Here, A and B are the input bits, Cin is the carry from previous addition, S is the sum bit, and Cout is the output carry bit.

The output equations of the full adder are,

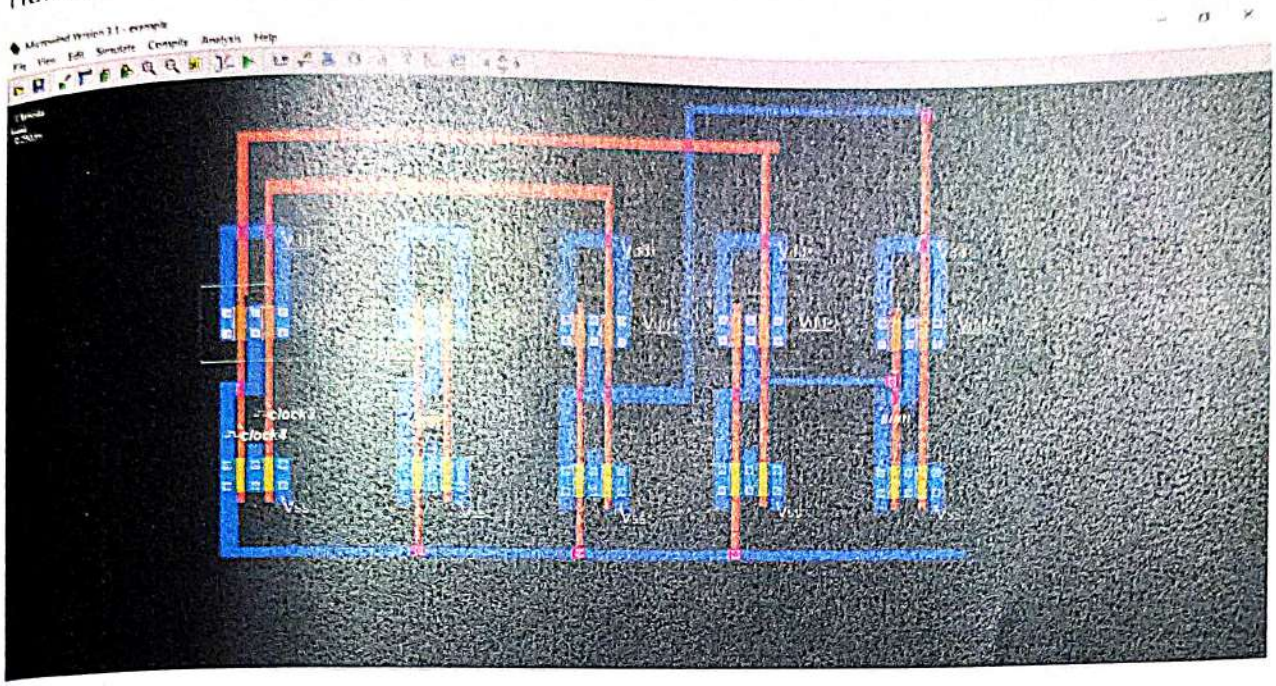
$$\text{Sum, } S = A \oplus B \oplus C_{in}$$

$$\text{Carry, } C_{out} = Ab + AC_{in} + BC_{in}$$

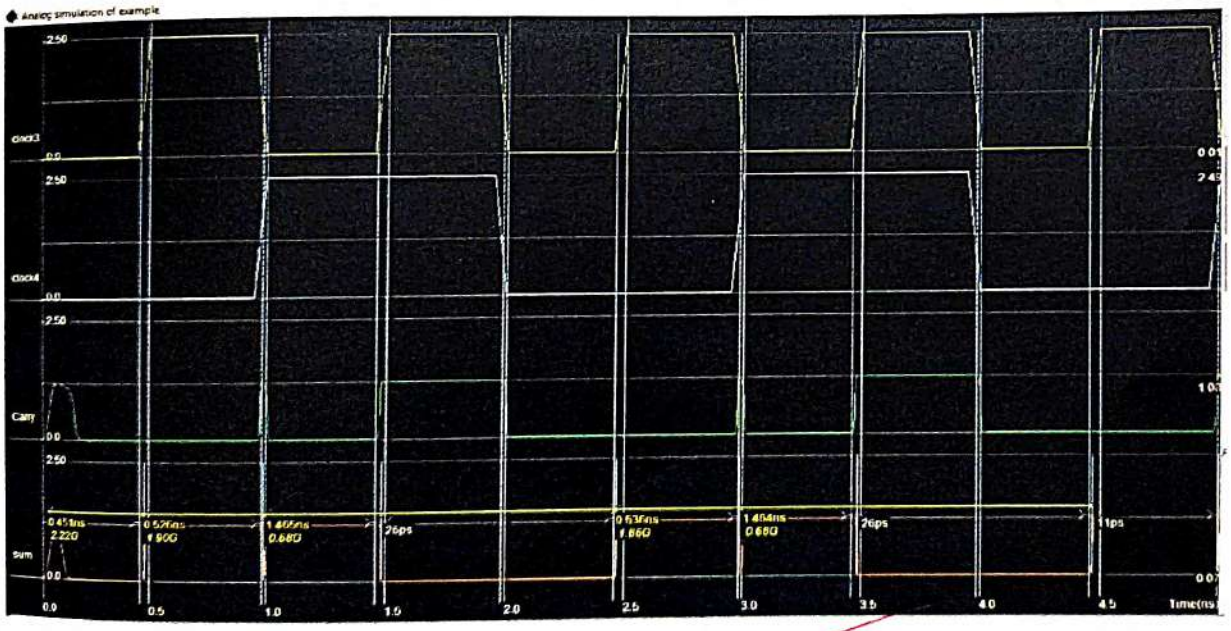
BE 140- Piyush Sawkar

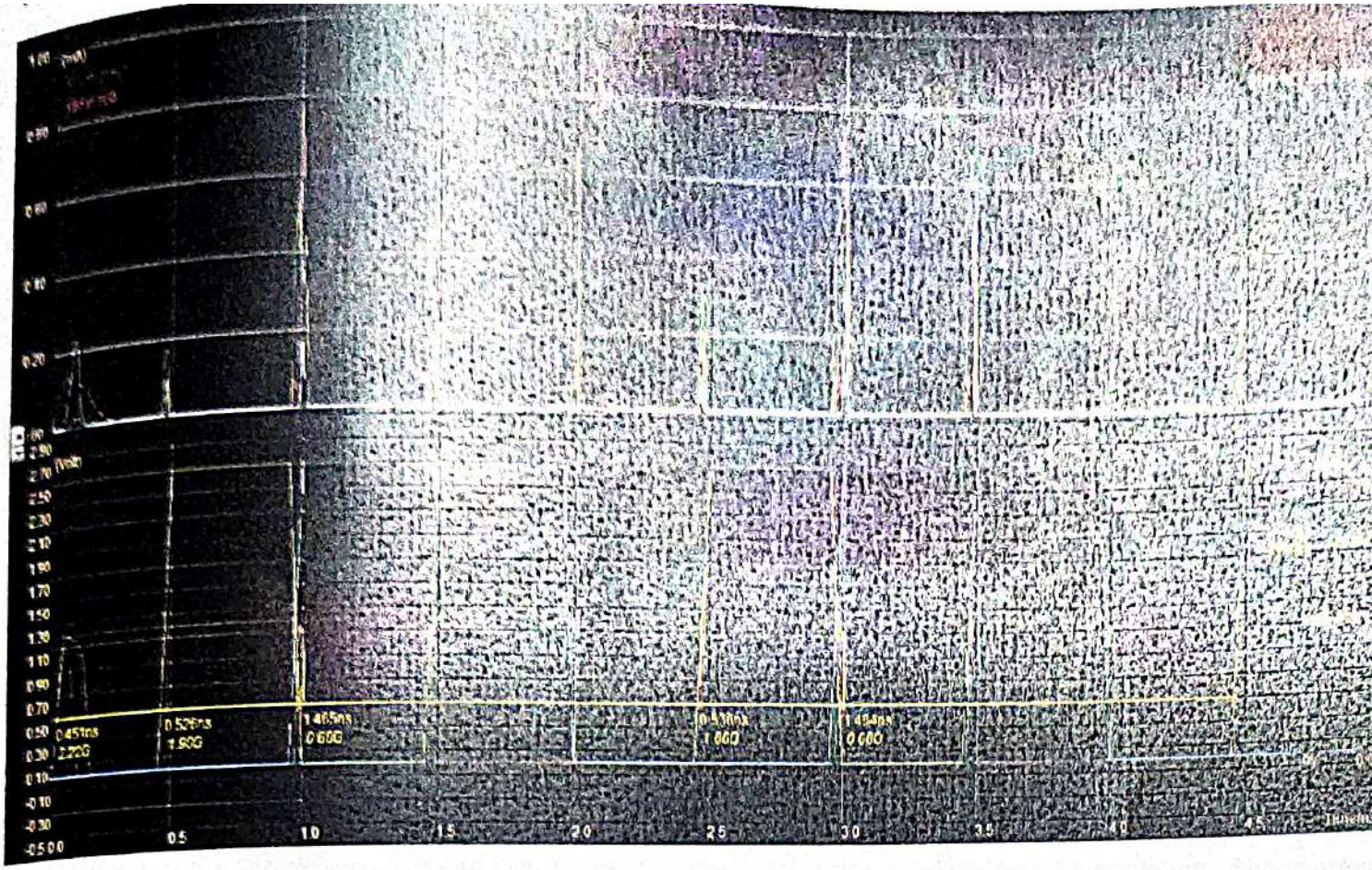
Half Adder and Full Adder :

Half Adder :

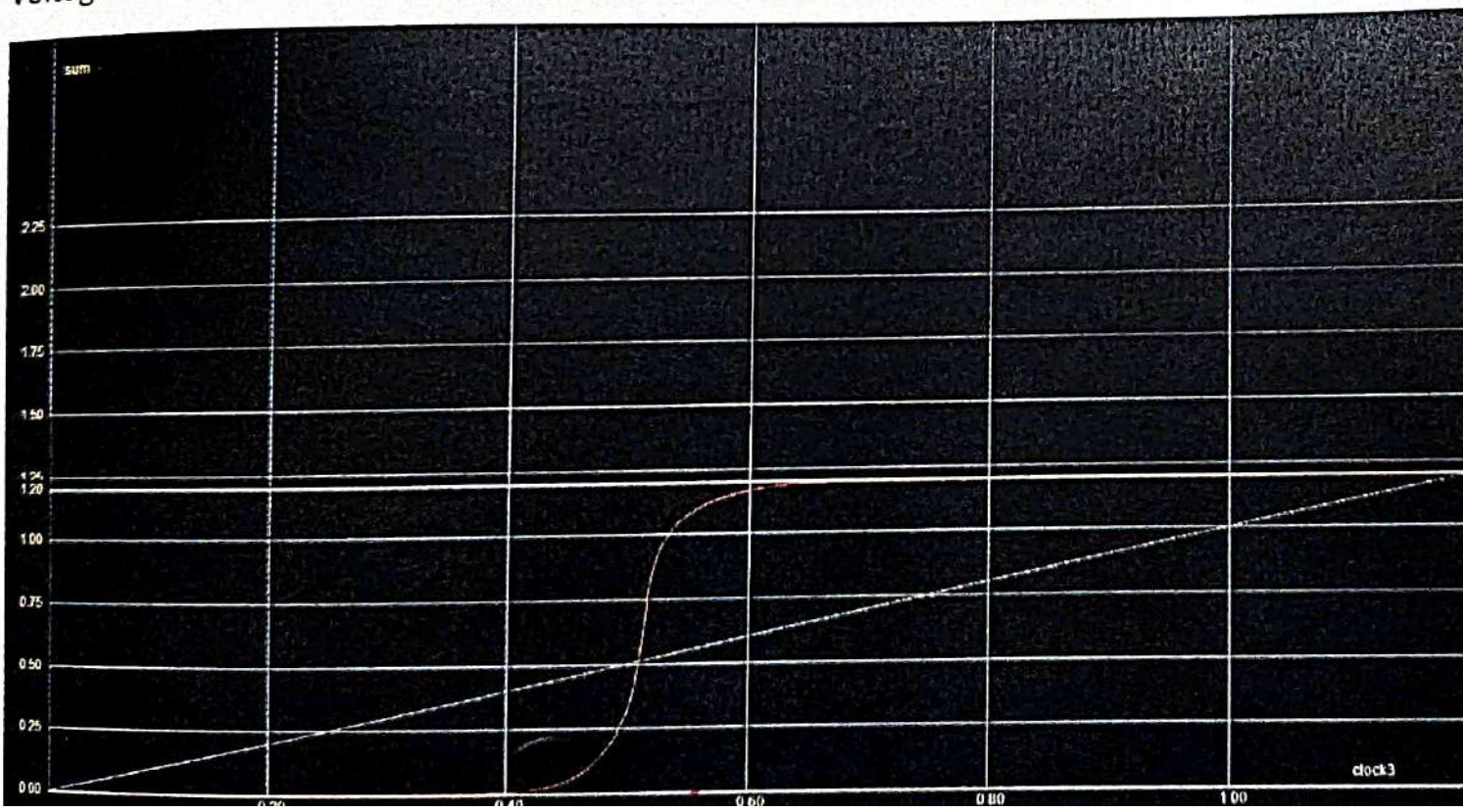


Voltage vs Time -





Voltage vs Voltage -



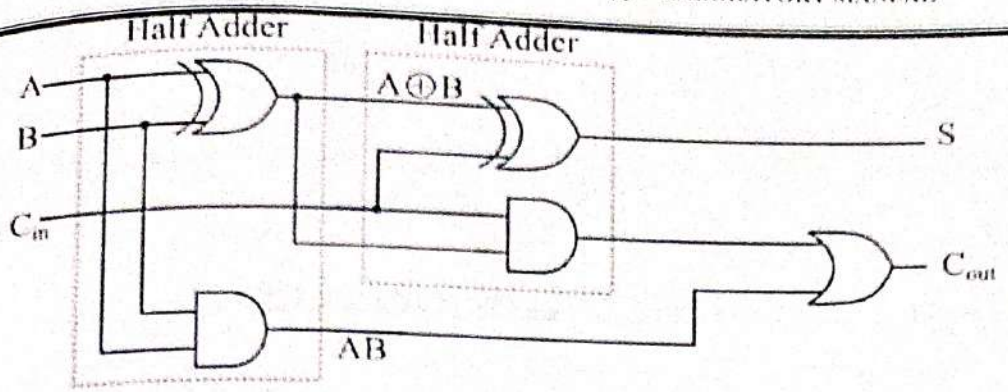


Fig 2. Full adder using Half Adder

Procedure:

Steps for design and simulation:-

1. Click on MOS generator from PALLETE window and generate PMOS and NMOS device.
2. Connect gate terminals of both NMOS devices using polysilicon substrate.
3. Connect source of PMOS and drain of NMOS using metal.
4. Apply VDD to drain of PMOS and VSS to source of NMOS.
5. Connect contacts for inputs and outputs.
6. Simulate the design.

Conclusion: We implemented half & full adder the half adder can only add two operands, whereas the full adder can add 3 operands also seen that the half adder has a simple hardware architecture, while full adder has a complex hardware architecture.

EXPERIMENT NO.7

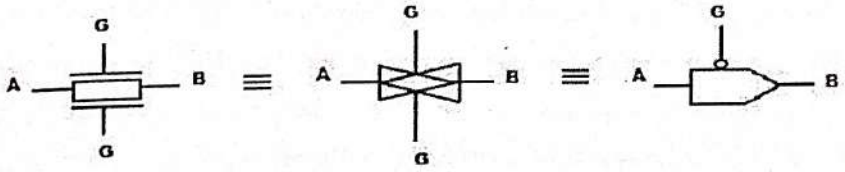
Aim:
Prepare the layout for 2:1 multiplexer using trans-mission gates and do the functional simulation and verify the results.

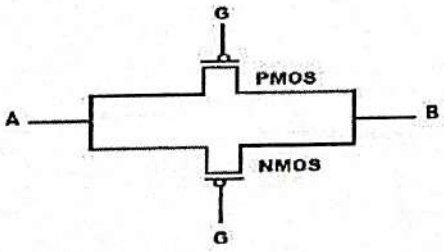
Objectives: 1. To understand strong '0' and weak '1' and strong '1' and weak '0' and Pass transistor

Specifications:
Technology: 0.12um
Supply voltage: 1.2 V

Software Platform: Microwind 3.5

Theory:





G=Control Signal

Representation of Transmission Gate

Truth Table:

INPUTS		SELECT LINES		OUTPUT
D1	D0	S	S	Y
X	0	0	1	0
X	1	0	1	1
0	X	1	0	0
1	X	1	0	1

Multiplexer:

1. MUX is a digital circuit which selects one of them data is inputs and route, it to the output.

2. Transmission gate: - The strength of a signal is measured by how closely it approximates ideal voltage sources. A strong signal can source of 1's or 0's are power supplies or rails (VDD & GND).

3. An NMOS transistor passes a strong '0' since, it's almost perfect switch when passing a zero. But, it's imperfect at passing a '1'. A PMOS transistor has exactly opposite behavior.

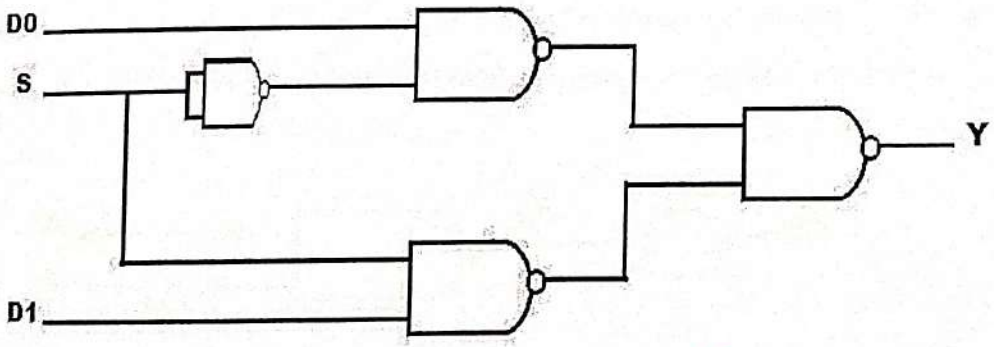
4. A PMOS or NMOS transistor alone is called as pass transistor. A parallel combination of NMOS and PMOS I s called to G in which 0's and 1's are both passed in an acceptable form.

5. The central signal (G) is applied to NMOS transistor and its complement (GB) is applied to PMOS transistor.

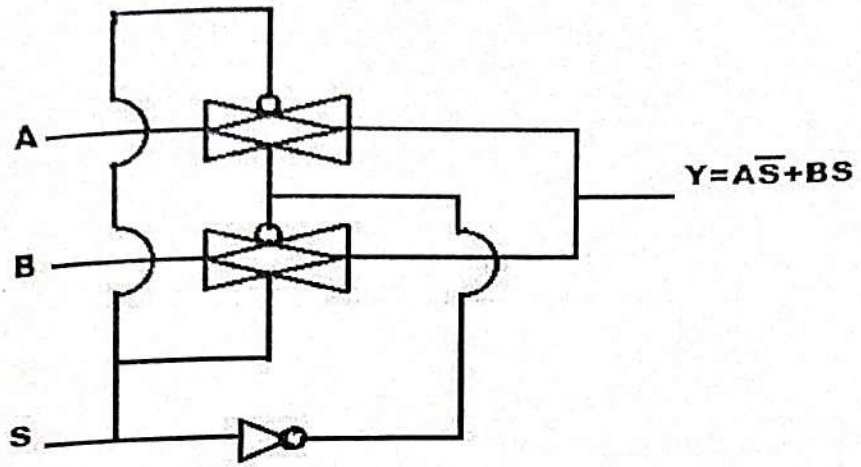
6. Transmission gate is used as bidirectional switch controlled by control signal (G).

7. When control signal is high (GB=low),both PMOS and NMOS transistor are ON and provides a vice versa. CMOS transmission gate are compact circuits which requires minimum transistors.

8. Transmission gate are used to implement OR operation, XOR operation 2 inputs MUX.



2:1 MUX USING NAND GATE



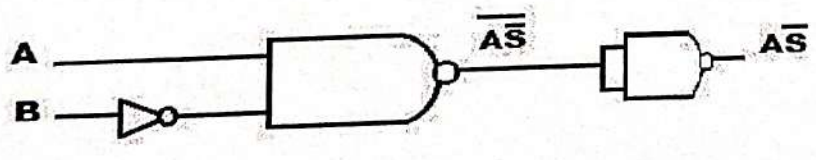
2:1 MUX USING TRANSMISSION GATE

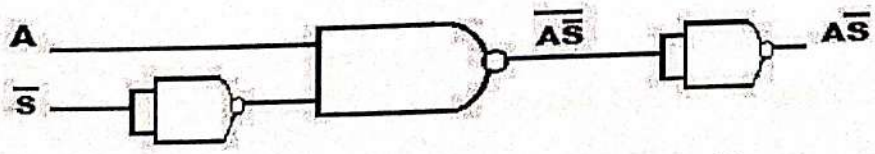
STEPS FOR DESIGN:-

1. MUX are important components in CMOS̄ memory elements and data manipulation structures. A MUX selects one of the inputs to the output line depending on select line.
2. In 2:1 MUX, Let D0 and D1 be inputs and S,S be select lines.
3. The input D0 is selected when select S=1 and S=0 and input D1 is selected when select S=0 and S=1.

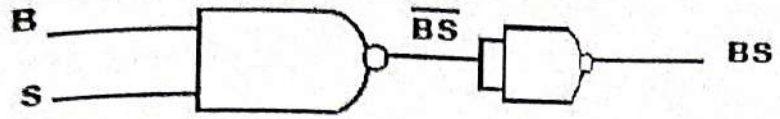
Implementation of 2:1 MUX using conventional method (using universal NAND gate)

1. AS̄



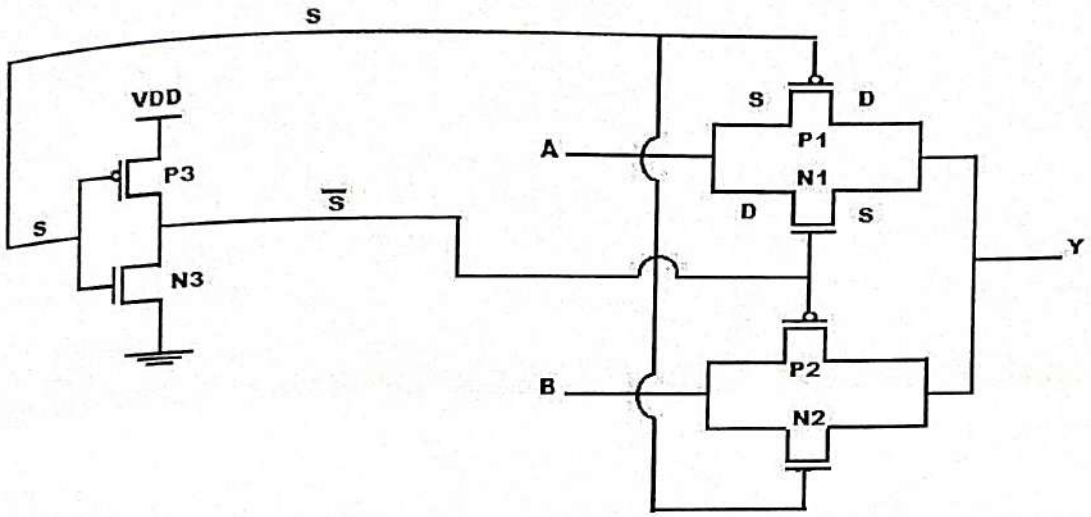


The equation obtained for 2:1 MUX using universal NAND gate is Y=AS̄+BS.

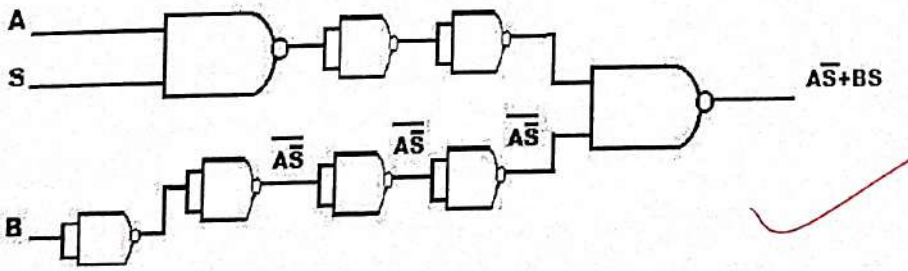


Now we will implement OR using NAND gate for $AS+\overline{BS}$.

Equation $Y=AS+BS$.



2:1 MUX USING TRANSMISSION GATE



For implementing 1 NAND gate, we require 2 PMOS and 2 NMOS.

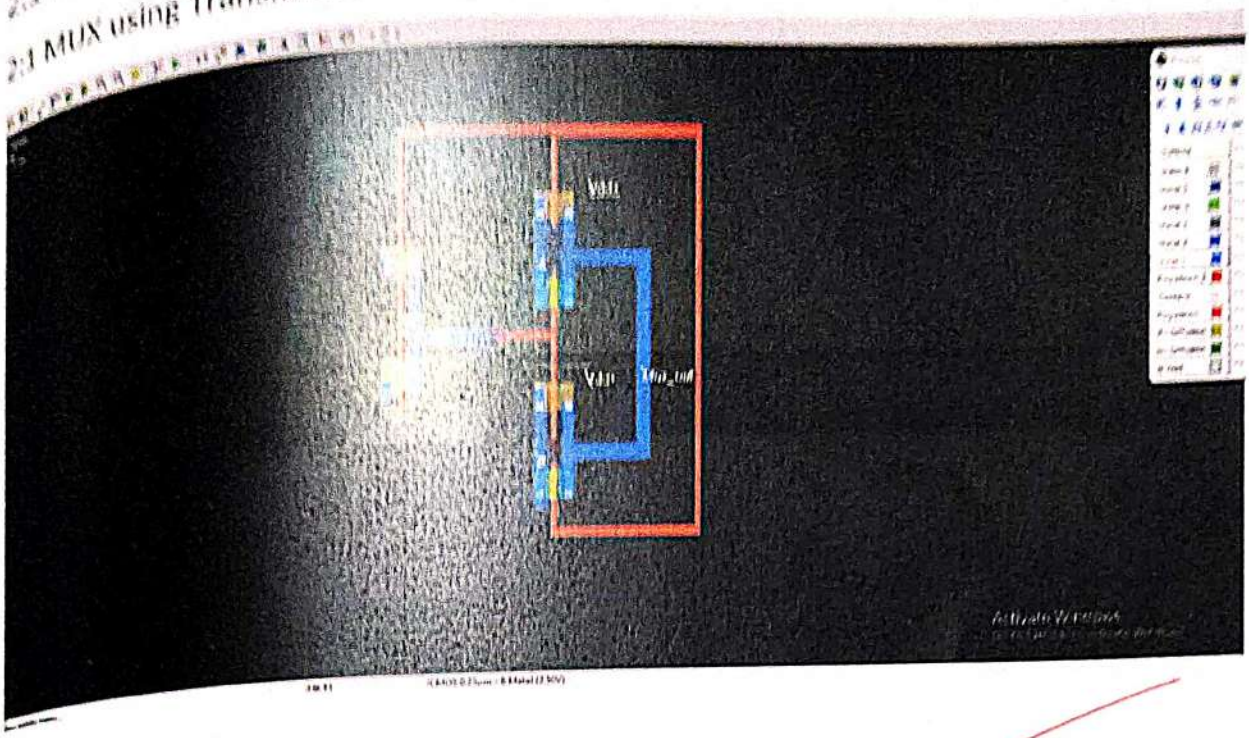
Therefore, implementing total 8 NAND gates we require 16 PMOS and 16 NMOS.

A 2:1 MUX can be formed by using, connecting 2 transmission gate together as shown in the figure

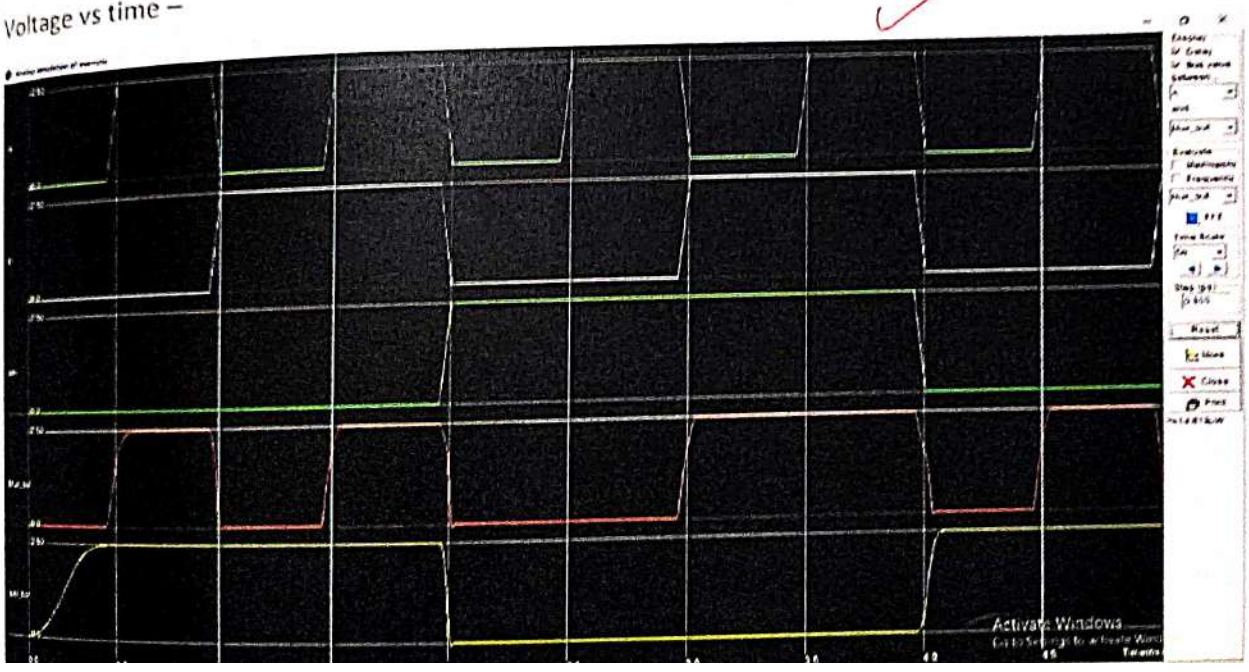
BT 140- Piyush Sawkar

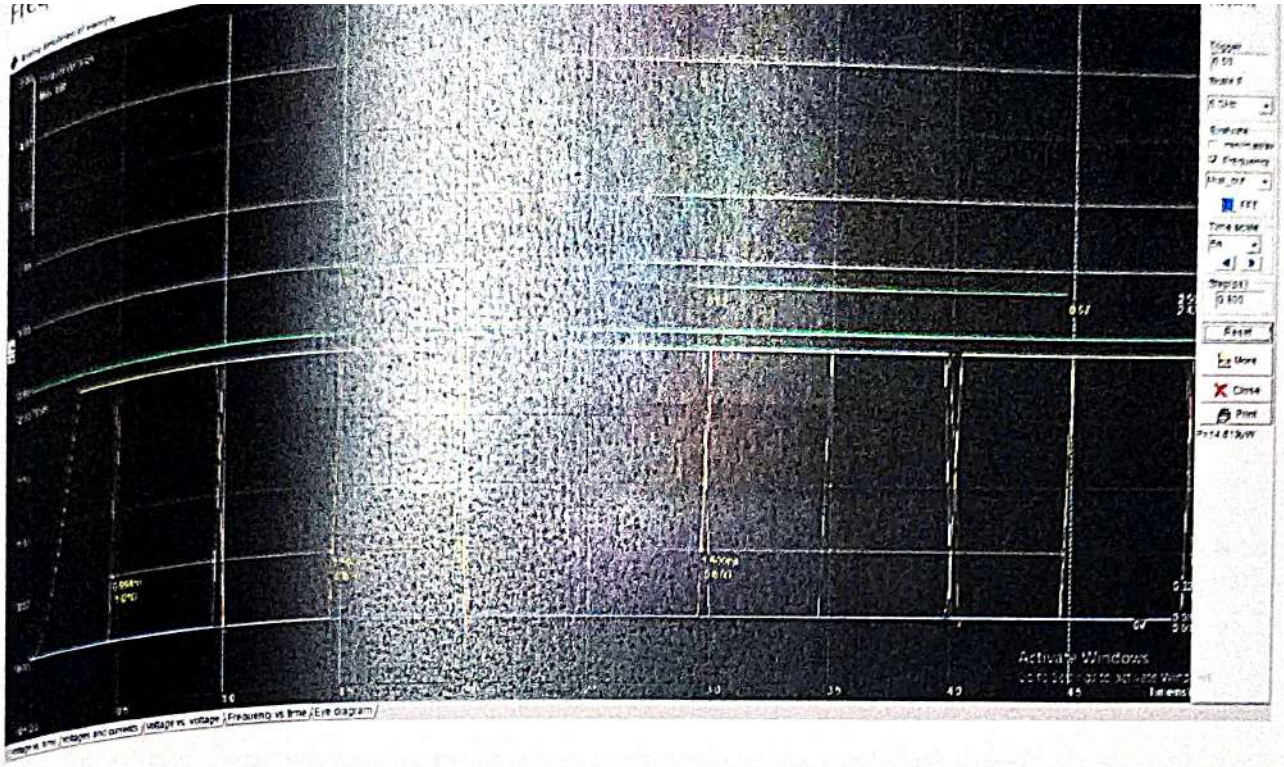
2:1 MUX using Transmission Gate and Logic gate :

2:1 MUX using Transmission Gate :

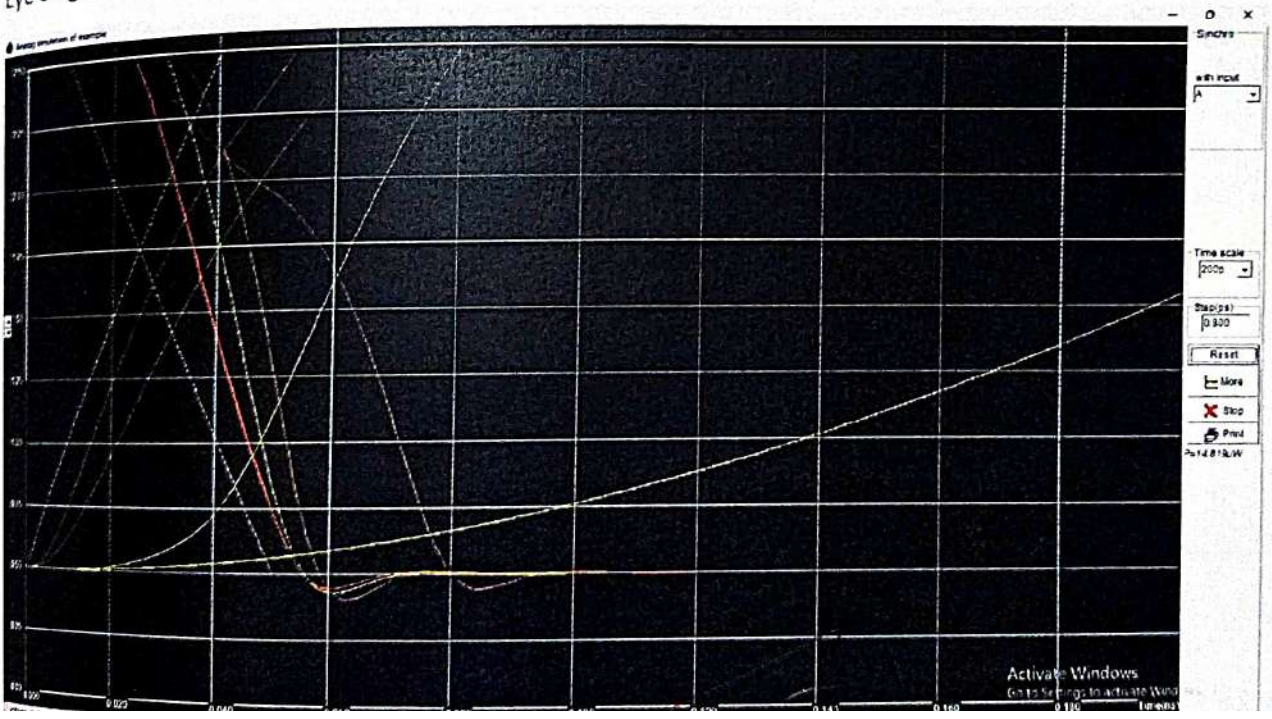


Voltage vs time –



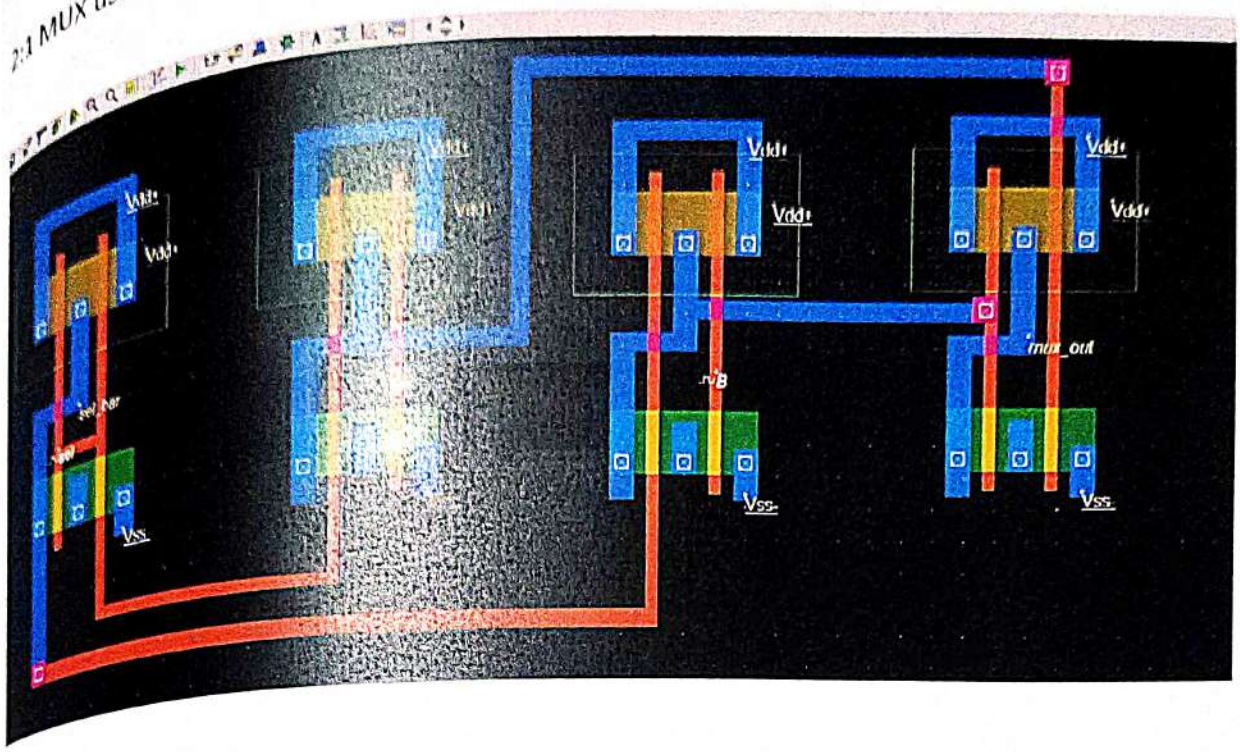

Activate Windows
Reset
More
Close
Print

Eye diagram –

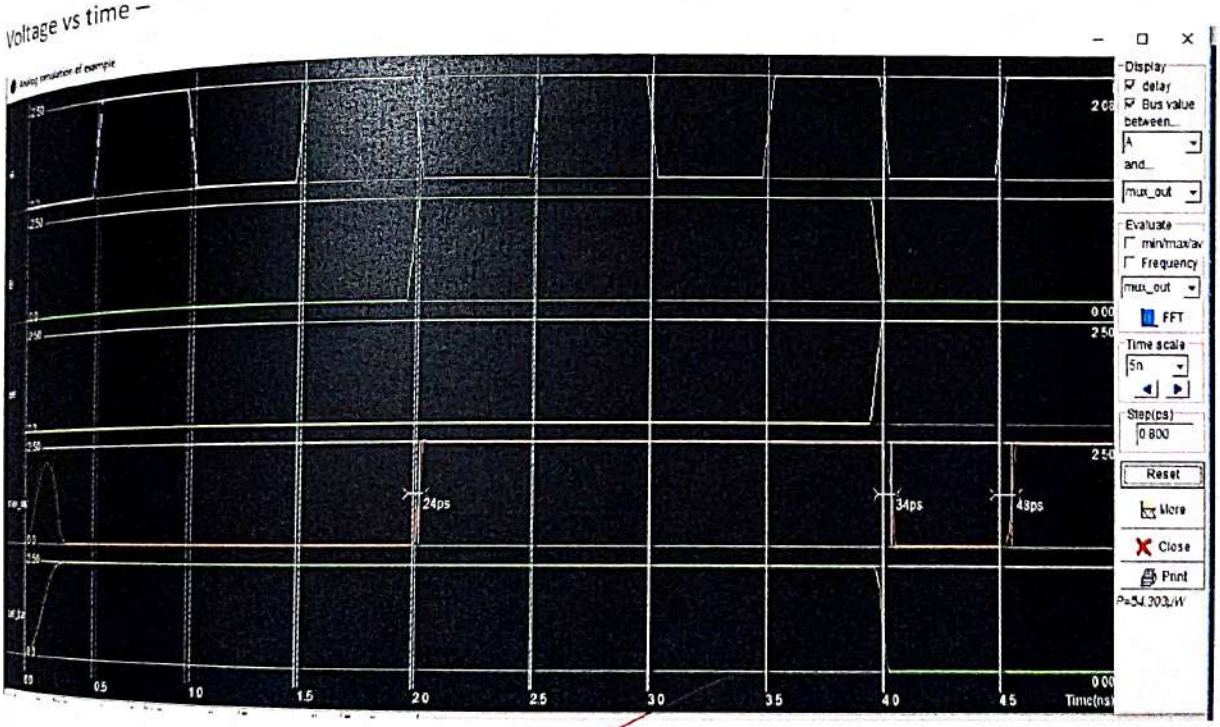

Sinchro
Time scale
Reset
More
Stop
Print
Activate Windows

BE 140- Piyush Sawkar

2:1 MUX using Logical gate (NAND) :

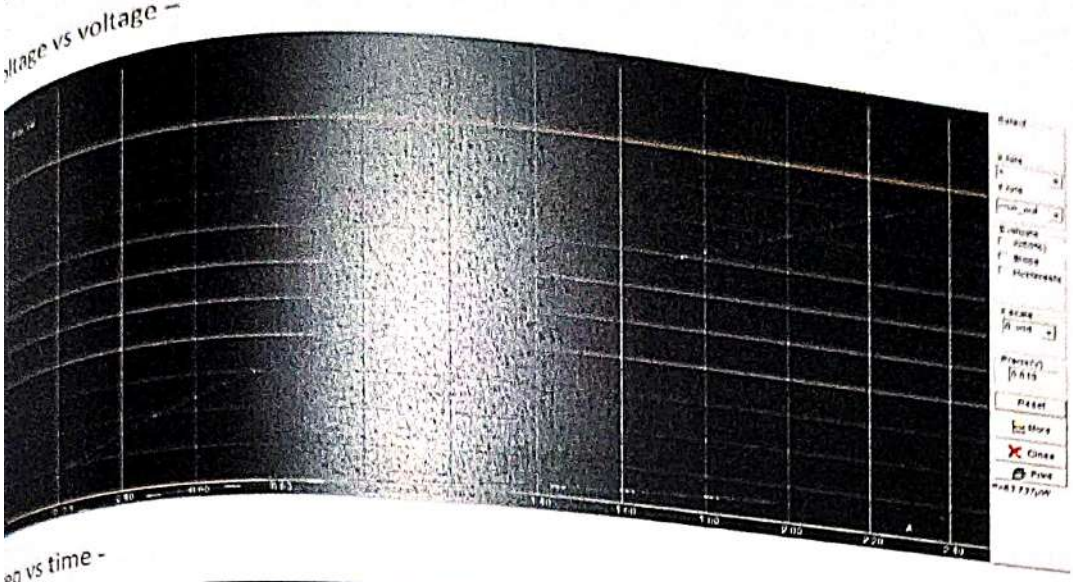


Voltage vs time –

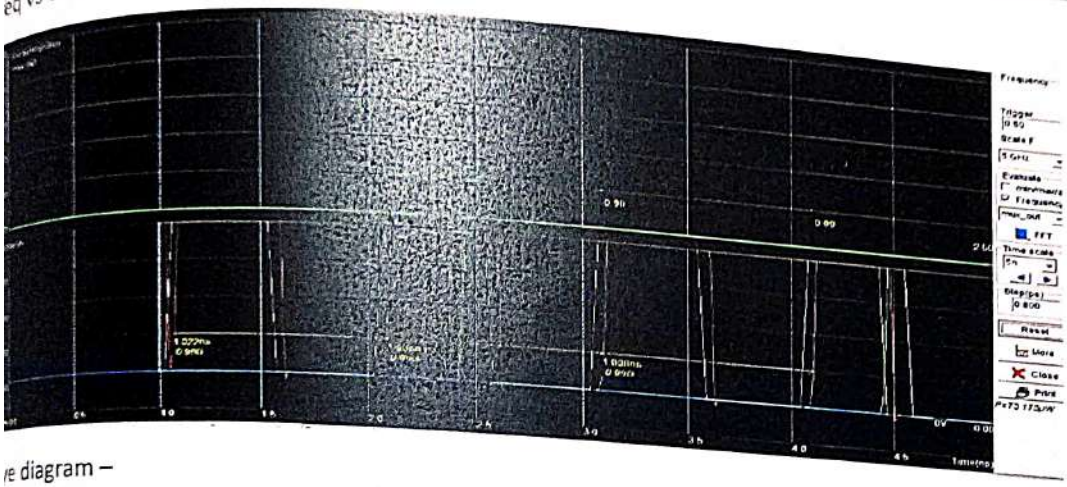


140- Piyush Sawkar

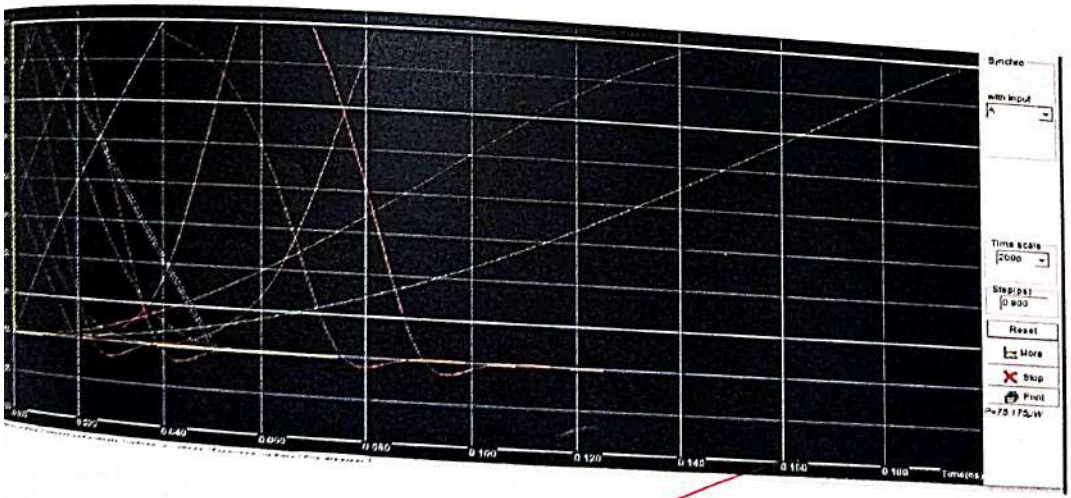
oltage vs voltage –



eq vs time -



e diagram –



Comparisons between

Sr.no.	Parameter		
1	Implementation		
2	No. of PMOS and NMOS required		
3	Total gates		
4	Time consumption, Cost, Speed		
5	Direction of flow	Bidirectional	

cedure:

s for design and simulation:-

on MOS generator from PALETTE window and generate PMOS and NMOS device.

nect gate terminals of both NMOS devices using polysilicon substrate.

nect source of PMOS and drain of NMOS using metal.

y VDD to drain of PMOS and VSS to source of NMOS.

nnect contacts for inputs and outputs.

mulate the design.

Conclusion: The layout design for 2:1 multiplexer using transmission gates was created efficiently & functional simulation confirmed its correct operation. Application of transmission gates in digital design & interconnection has multiple use functionality.

Questions:

1) What are advantages of transmission gate?

2) What are applications of transmission gate?

3) Draw 4:1 multiplexer using transmission gate?

3/11/23

What are advantages of transmission gate ?

1) Transmission gate are commonly used in VLSI circuit for various reason. Some advantages are :

a) Low Propagation Delay : Transmission gates have minimal delay, making suitable for high speed.

b) Low Power : They typically consume less static power consumption compared to other logic gates, as they do not have a direct path from power supply.

c) Bidirectional : Transmission gates can pass signal in both operation. directions, which is useful for bidirectional data transfer.

d) Space Efficiency : They are often more-area-efficient than their equivalent CMOS counterpart.

e) Improved Signal Integrity & Robustness.

Q.2) What are applications of transmission gate ?

→ i) Multiplexers : Transmission gates are often used as the pass gates in multiplexer circuit.

ii) Adder & Arthemetic Logic : They can be used in adder & ALU circuits to perform bitwise addition & logical operations efficiently.

Flip-Flops & Latches: Transmission gates are employed in employed in flip-flops & latches to enable or disable the clock

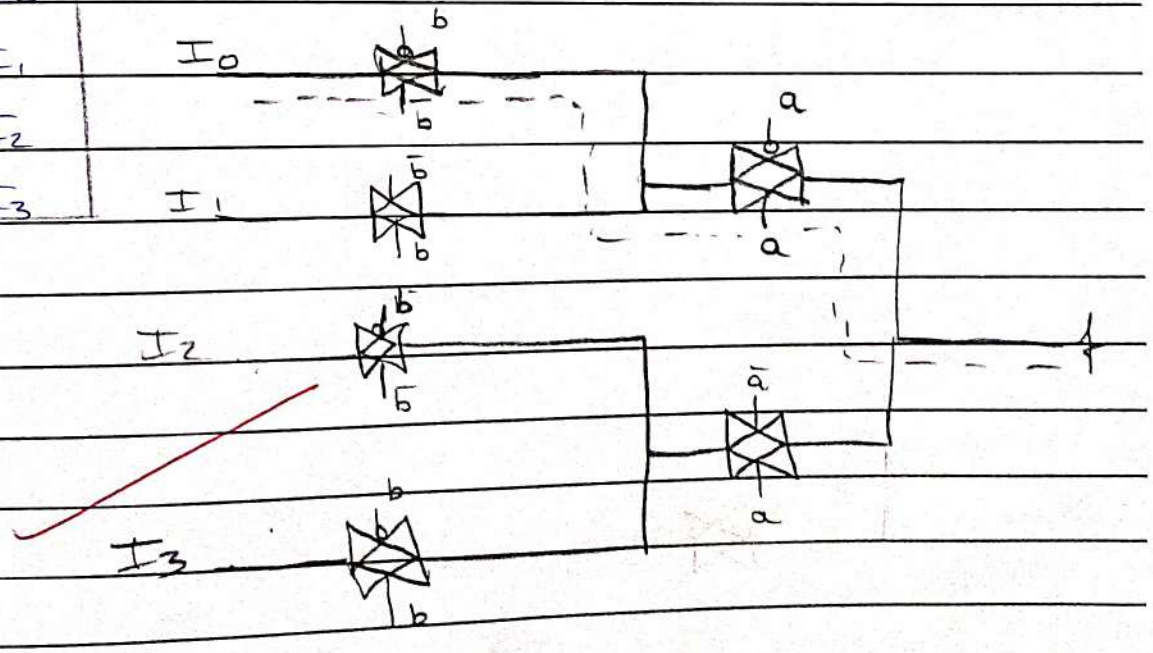
Data transmission: Transmission gates are used for bidirectional data transmission in various communication

Signal Routing: In complex VLSI layouts, transmission gates can be used for signal routing & switching purposes.

Draw 4:1 multiplexer using transmission gate?

a	b	f
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$f = \bar{a}\bar{b}I_0 + \bar{a}bI_1 + a\bar{b}I_2 + abI_3$



In would appear at the output if $\bar{a}=\bar{b}=1$ i.e. $a=b=0$