**List of Experiments**

**Part A: To write VHDL code, simulate with test bench, synthesis, implement on PLD (Any 5 to be performed)**

1. 4 bit ALU for Add, Subtract, AND, NAND, OR, XOR & XNOR.

2. Universal shift register with mode selection input for SISO, SIPO, PISO, & PIPO.

3. Mod - N Counter

4. FIFO memory

5. LCD Interface

6. Keypad interface

**Part B: To prepare CMOS layout in selected technology, simulate with & without capacitive load, comment on rise & fall times. (Any 3 to be performed)**

1. Inverter, NAND, NOR gates

2. Half Adder & Full Adder

3.2:1 Mux using logic gates & transmission gates

4.One bit SRAM Cell

**Practical No. 1**

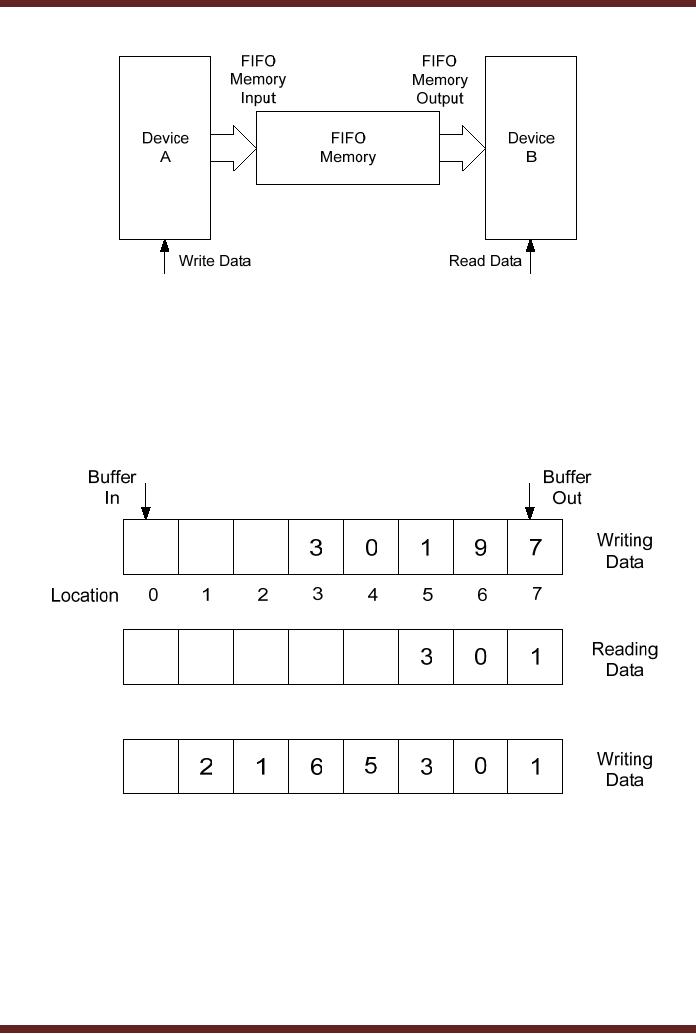
**16x8 FIFO MEMORY**

**Aim:**

To write a VHDL code for a 16x8FIFO Memory and implement it on FPGA kit.

**Theory:**

Digital systems receive data or transfer data to devices that are operating at different data rates. A Computer (microprocessor), for example, receives data from the Keyboard as a user types in the information. The keyboard is a very slow device which generates data at a rate of few bytes per second. The microprocessor on the other hand is very fast and can processes information at very high data rates. Devices that operate at different data rates cannot be connected to each other directly through their data lines because the devices that operate at very high data rates are slowed down to the data rate of the slow device.



 FIFO Memory Connecting Two Communicating Devices

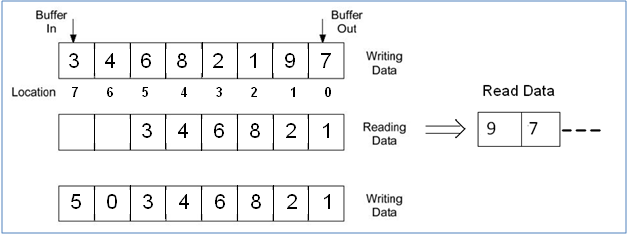
Each item in the memory is stored in a queue data structure. The first data which is added to the queue will be the first data to be removed. Processing continues to proceed sequentially in this same order. FIFO is generally implemented as a circular queue, and thus has a Read pointer and Write Pointer. An Enable input is also used. The enable pin is made ‘1’ while writing data to the FIFO Memory system. There are namely two types of FIFO Memory systems:

* Synchronous FIFO that uses the same clock for reading and writing.
* Asynchronous FIFO that uses separate clocks for reading and writing.

When data is being written to the memory, the Write Pointer is used. It starts from 0 and ends at 15, being a 16 location stack. Each location on the stack stores an 8 bit data. Most importantly writing or reading is done only when the Enable has been made ‘1’!

Data can be read from the memory at any point of time and the Read Pointer is used for this effect. The Read Pointer is always kept updated with the latest value of the number of location into which data has been written into.

Data is read only as long as the Read Pointer is less than the Write Pointer. If this is not followed then data not even valid or present in the memory would be read leading to compromised and erroneous data being used or displayed.



**VHDL PROGRAM**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity myfifo\_1 is

Port ( datain : in STD\_LOGIC\_VECTOR (7 downto 0);

en,clk,rst : in STD\_LOGIC;

W : in STD\_LOGIC;

DATAOUT : OUT STD\_LOGIC\_VECTOR (7 downto 0);

RED : OUT STD\_LOGIC);

end myfifo\_1;

architecture Behavioral of myfifo\_1 is

SIGNAL WPTR, RPTR :STD\_LOGIC\_VECTOR(3 DOWNTO 0);

Type fifo is array(15 downto 00) of STD\_LOGIC\_VECTOR(07 downto 00);

signal mem:fifo;

signal clk\_div: std\_logic\_vector(21 downto 0);

begin

process(clk,rst)

begin

if rst ='1' then

clk\_div<=(others=>'0');

elsif (clk'event and clk='1') then

clk\_div<=clk\_div+1;

end if;

end process;

Process(clk\_DIV(21),rst) ----0 for simulation 21 for implementation

begin

if rst ='1' then

wptr<= "0000";

rptr<="0000";

red <= '0';

elsif(clk\_DIV(21)'event and clk\_DIV(21)='1')then ---0 for simulation and 21 for implementation

if (en ='1')then

if(w='1')then

if (wptr< "1111")then

mem(conv\_integer(wptr)) <= datain;

wptr<= wptr +1;

else

red <= '1';

end if;

else

if(rptr<wptr)then

red <= '0';

dataout<= mem(conv\_integer(rptr));

rptr<= rptr+1;

else

red <= '1';

dataout<= "00000000";

end if;

end if;

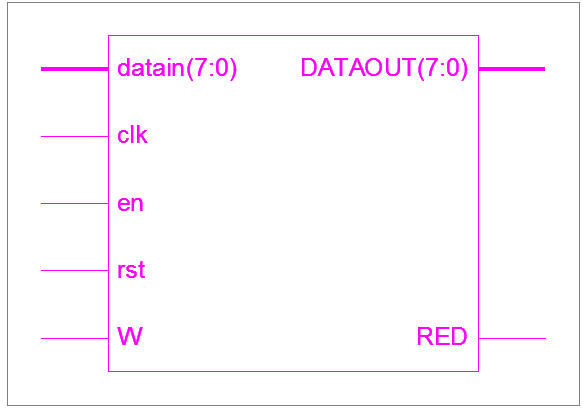
end if;

end if;

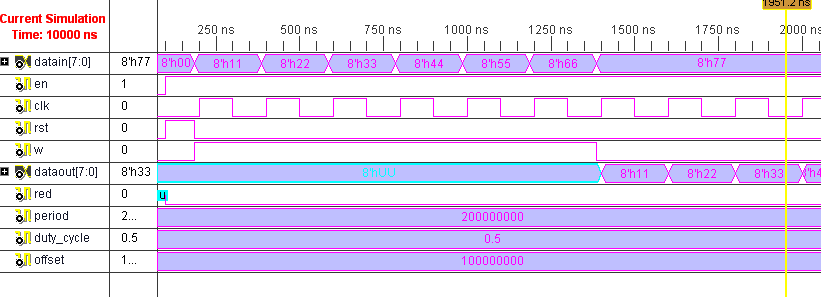
end process;

end Behavioral;

**RTL SCHEMATIC**



**SIMULATION RESULT**



**UCF FILE**

NET DATAIN(0) LOC=P191;

NET DATAIN(1) LOC=P190;

NET DATAIN(2) LOC=P196;

NET DATAIN(3) LOC=P194;

NET DATAIN(4) LOC=P198;

NET DATAIN(5) LOC=P197;

NET DATAIN(6) LOC=P200;

NET DATAIN(7) LOC=P199;

NET DATAOUT(0) LOC=P165;

NET DATAOUT(1) LOC=P166;0

NET DATAOUT(2) LOC=P161;

NET DATAOUT(3) LOC=P162;

NET DATAOUT(4) LOC=P155;

NET DATAOUT(5) LOC=P156;

NET DATAOUT(6) LOC=P152;

NET DATAOUT(7) LOC=P154;

NET EN LOC=P5;

NET CLK LOC=P76;

NET RST LOC=P108;

NET W LOC=P7;

NET RED LOC=P149;

**Practical No. 2**

**4 Bit Universal Register**

**Aim:**

To write VHDL code for 4-bit universal shift register in four modes of operation[SISO, SIPO, PISO, PIPO] using Mode Select.

**Theory:**

The Shift Register is a type of sequential logic circuit that can be used for the storage and transfer of data in the form of binary numbers. This sequential device loads the data present on its inputs and then moves or shifts it to its output once every clock cycle.

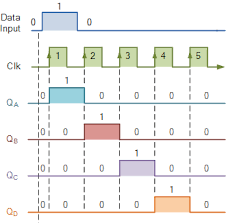
The 4-bit Universal Shift Register consists of 4 single bit D Flip-Flops connected together in a serial type daisy-chain arrangement so that the output from one data latch becomes the input of the next latch and so on.

|  |  |
| --- | --- |
| **MODE SELECT** | **OPERATION** |
| **00** | **SISO Right Shift** |
| **01** | **SIPO** |
| **10** | **PIPO** |
| **11** | **SISO Left Shift** |

The table of the 2-bit mode selects and the corresponding operation performed at the mode select value is:

Basic Data Movement Through A Shift Register

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clk pulse No | QA | QB | QC | QD |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 |
| 5 | 0 | 0 | 0 | 0 |

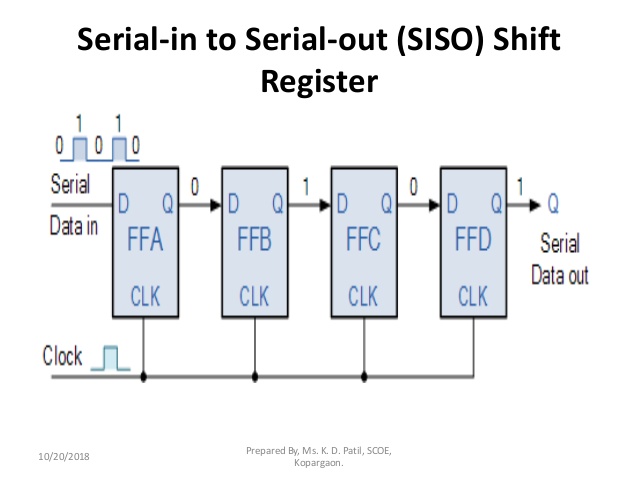


All in all, there are 4 modes of operations, which are:

1.**Serial In Serial Out (SISO) Mode:**

There is only one output, data leaves the shift register one bit at a time in a serial pattern, hence the name Serial In Serial Out Register or SISO. It is one of the simplest of all the four configurations as it has only three connections, serial input(SI) which determines what enters the left hand flip-flop, the serial output (SO) which is taken from the output of right hand flip-flop and the sequencing clock signal (clk).

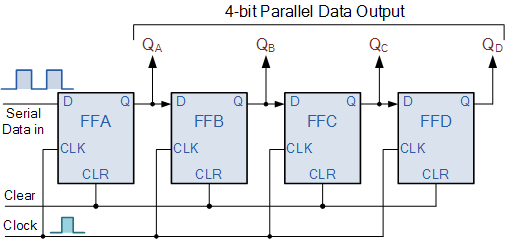
We have a mode control option wherein we can select if we want a right shift or left shift operation. The logic circuit diagram on the next page shows a generalized SISO shift register.



You may think what’s the point of a SISO shift register if the output data is same as that of input data. Well this type of shift register also acts as a temporary storage device or it can act as a time delay device for the data.

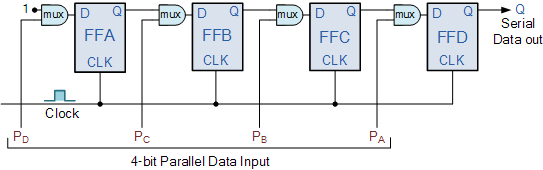
**2.Serial In Parallel Out (SIPO)Mode:**

In this mode, there is one input where data comes in serially and being a 4 -it data, there are 4 output ports, one each from each D flip flop. Hence it is identified as Serial In Parallel Out. The circuit below shows the 4 bit SIPO mode.



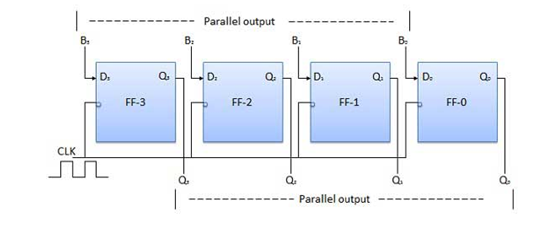
**3. Parallel In Serial Out (PISO) Mode:**

This mode is characterized by a parallel port input for each of the 4 flip flops and one final output port at the end of the last flip flop. It is shown in the diagram below. PISO can operate in two modes namely Load or Shift mode. But for our sake, due to coding limitations, PISO operation will not be performed. However, for knowledge sakes, PISO is done using IC74194 Bidirectional Universal Shift Register.



**4. Parallel Input Parallel Output(PIPO) Mode:**

This mode is also one of the simplest modes of operation. It has 4 bit parallel input as well as output for each flip flop and takes the least clock cycles. As soon as a clock edge is applied, the input binary bits will be loaded into the flip flop simultaneously. The loaded bits will appear simultaneously to the output side. Only clock pulse is essential to load all the bits. The circuit is shown below:



**VHDL PROGRAM**

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;

entity reg is

Port ( din : in STD\_LOGIC\_VECTOR (03 downto 0);

dout :inout STD\_LOGIC\_VECTOR (3 downto 0);

mode1 : in STD\_LOGIC\_VECTOR (1 downto 0);

clk : in STD\_LOGIC;

rst : in STD\_LOGIC);

end reg;

architecture Behavioral of reg is

signal MSBIN,LSBIN :STD\_LOGIC;

signal TEMP: STD\_LOGIC\_VECTOR(3 DOWNTO 0);

SIGNAL CLK\_DIV : STD\_LOGIC\_VECTOR(21 DOWNTO 0);

begin

MSBIN <= DIN(3);

LSBIN <= DIN(0);

PROCESS(CLK,RST)

BEGIN

IF (RST= '1')THEN

CLK\_DIV <= (OTHERS => '0');

ELSIF(CLK'EVENT AND CLK ='1')THEN

CLK\_DIV <= CLK\_DIV + '1';

END IF;

END PROCESS;

PROCESS(CLK\_DIV(21),RST)

BEGIN

IF(RST ='1')THEN

DOUT <= "0000";

temp <= "0000";

ELSIF(CLK\_DIV(21)'EVENT AND CLK\_DIV(21) ='1')THEN

CASE MODE1 IS

WHEN "00" =>

DOUT <= MSBIN &DOUT(3 DOWNTO 1); --SISO(rt shift)

WHEN "01" =>

DOUT <= DIN; --PIPO

WHEN "10" =>

temp <= MSBIN &temp(3 DOWNTO 1); --SIPO

dout<=temp;

WHEN "11" =>

DOUT <= DOUT(2 DOWNTO 0) & LSBIN; --SISO(left shift)

WHEN OTHERS =>

DOUT <= "0000";

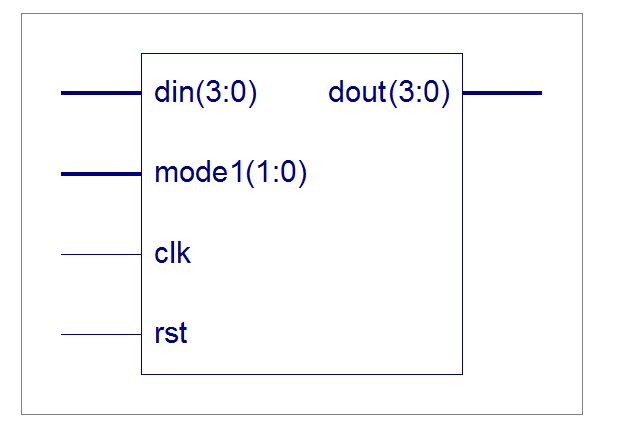
END CASE;

END IF;

END PROCESS;

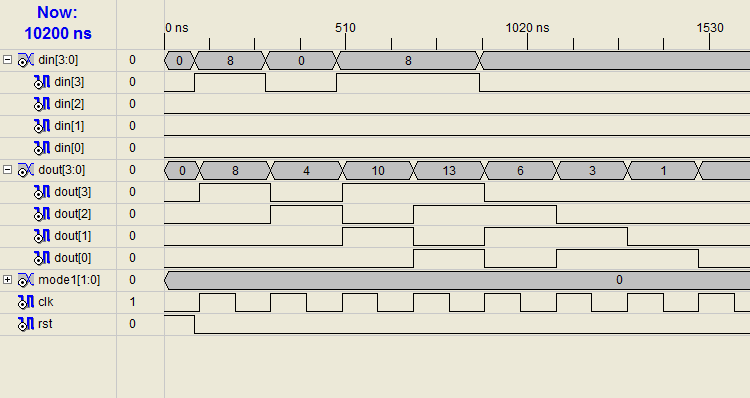
END BEHAVIORAL;

**RTL SCHEMATIC**

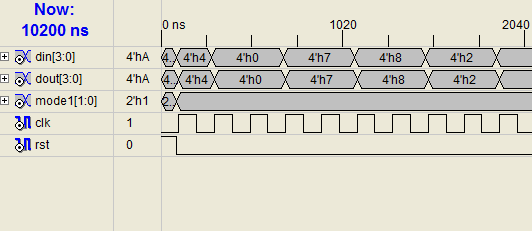


**SIMULATION RESULT**

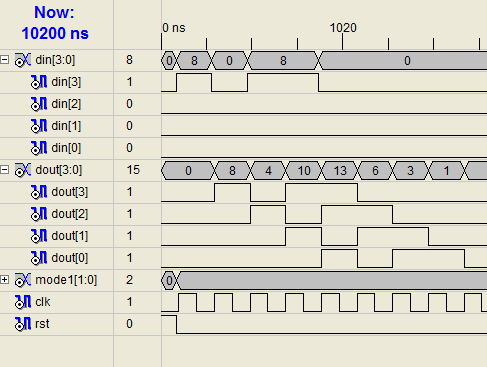
SISO (RIGHT)



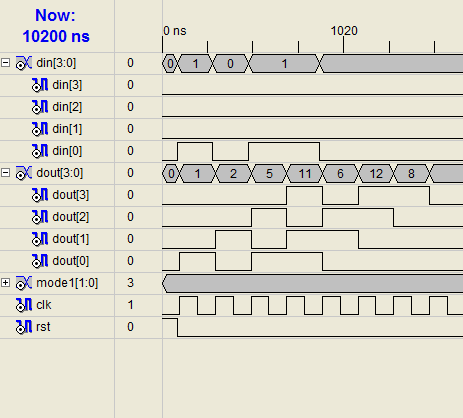
PIPO



SIPO



SISO (LEFT)



**UCF FILE**

NET DIN(3) LOC=P204;

NET DIN(2) LOC=P203;

NET DIN(1) LOC=P2;

NET DIN(0) LOC=P205;

NET DOUT(3) LOC=P143;

NET DOUT(2) LOC=P141;

NET DOUT(1) LOC=P146;

NET DOUT(0) LOC=P144;

NET MODE1(1) LOC=P190;

NET MODE1(0) LOC=P191;

NET CLK LOC=P76;

NET RST LOC=P108;

**Practical No. 3**

**4-Bit Arithmetic Logic Unit (ALU)**

**Aim:**

To write a VHDL code for 4 bit ALU performing the following operations +,-,AND, NAND, XOR, XNOR, OR and ALU Pass.

**Theory:**

The arithmetic logic unit (ALU) is a digital electronic circuit that performs arithmetic and bitwise logical operations on integer binary numbers. It is a fundamental building block of the central processing unit (CPU) found in many computers. This is in contrast to a floating-point unit (FPU), which is a digital circuit that operates on floating point numbers with the aid of one or more internal ALUs. Powerful and complex ALUs are often used in modem, high performance CPUs, FPUs and graphics processing units (GPUS). A single CPU, FPU or GPU may contain multiple ALUs.

The inputs to an ALU are the data to be operated on (called operands) and a code indicating the operation to be performed; the output of ALUis the result of the performed operation. In many designs, the ALU also exchanges additional information with a status register, which relates to the result of the current or previous operations.

The inputs used in this practical are:

* **A,B** : Operands 1 and 2 respectively. Each 4 it.

|  |  |
| --- | --- |
| MODE | OPERATION |
| 000 | ADDITION |
| 001 | SUBTRACTION |
| 010 | AND |
| 011 | NAND |
| 100 | XOR |
| 101 | XNOR |
| 110 | OR |
| 111 | ALU Pass |

* **Cin** : Carry input prior to the operation, if any.
* **Mode** : Allows us to choose from any of the operations mentioned above. The table for mode select is shown to the right:

The ALU has outputs:

* **Result:**4-bit output.
* **Cout** : Carry Output
* **Flag** : has a 4 bit structure depicted below

|  |  |  |  |
| --- | --- | --- | --- |
| S  (Sign) | Z  (Zero) | C  (Carry) | \_ |

**Arithmetic Operations:**

The set of arithmetic operations are addition and subtraction. This would be carried out on 4 bit logic with carry and borrow if needed.

**Logical Operations:**

The respective truth tables of allthe operations performed in the practical are:

AND GATE NAND GATE

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

EX-OR GATE EX-NOR GATE

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

OR GATE

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

**Flag/ Status Bits:**

Our ALU has 4 output status bits including sign, carry/borrow and zero result. It is usually the case that all operations can set the zero result status bit, but only arithmetic operations can set the overflow, negative result and carry/borrow bits.

**VHDL PROGRAM**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity ALU4BIT is

Port ( A : in STD\_LOGIC\_VECTOR (3 downto 0);

B : in STD\_LOGIC\_VECTOR (3 downto 0);

MODE1 : in STD\_LOGIC\_VECTOR (2 downto 0);

CIN : in STD\_LOGIC;

RESULT : out STD\_LOGIC\_VECTOR (3 downto 0);

COUT : out STD\_LOGIC;

FLAG : out STD\_LOGIC\_VECTOR (3 downto 0));

end ALU4BIT;

architecture Behavioral of ALU4BIT is

begin

PROCESS(A,B,CIN,MODE1)

VARIABLE TEMP:STD\_LOGIC\_VECTOR(4 DOWNTO 0);

BEGIN

CASE MODE1 IS

WHEN "000" => --ADDITION

TEMP := ('0' & A ) + B + CIN;

COUT <=TEMP(4);

FLAG(3)<= TEMP(3);

FLAG(1) <= TEMP(4);

IF (TEMP(3 DOWNTO 0)="0000")THEN

FLAG(2) <= '1';

else

flag(2)<=’0’;

END IF;

RESULT <= TEMP(3 DOWNTO 0);

WHEN "001" => --SUBTRACTION

TEMP := ('0' & A ) - B - CIN;

COUT <=TEMP(4);

FLAG(3)<= TEMP(3);

FLAG(1) <= TEMP(4);

IF (TEMP(3 DOWNTO 0)="0000")THEN

FLAG(2) <= '1';

else

flag(2)<=’0’;

END IF;

RESULT <= TEMP(3 DOWNTO 0);

WHEN "010" => --AND

TEMP(3 DOWNTO 0) := A AND B;

FLAG(3)<= TEMP(3);

IF (TEMP(3 DOWNTO 0)="0000")THEN

FLAG(2) <= '1';

else

flag(2)<=’0’;

END IF;

RESULT <= TEMP(3 DOWNTO 0);

WHEN "011" => --NAND

TEMP(3 DOWNTO 0) := A NAND B;

FLAG(3)<= TEMP(3);

IF (TEMP(3 DOWNTO 0)="0000")THEN

FLAG(2) <= '1';

else

flag(2)<=’0’;

END IF;

RESULT <= TEMP(3 DOWNTO 0);

WHEN "100" => --XOR

TEMP(3 DOWNTO 0) := A XOR B;

FLAG(3)<= TEMP(3);

IF (TEMP(3 DOWNTO 0)="0000")THEN

FLAG(2) <= '1';

else

flag(2)<=’0’;

END IF;

RESULT <= TEMP(3 DOWNTO 0);

WHEN "101" => --XNOR

TEMP(3 DOWNTO 0) := A XNOR B;

FLAG(3)<= TEMP(3);

IF (TEMP(3 DOWNTO 0)="0000")THEN

FLAG(2) <= '1';

else

flag(2)<=’0’;

END IF;

RESULT <= TEMP(3 DOWNTO 0);

WHEN "110" => --OR

TEMP(3 DOWNTO 0) := A OR B;

FLAG(3)<= TEMP(3);

IF (TEMP(3 DOWNTO 0)="0000")THEN

FLAG(2) <= '1';

else

flag(2)<=’0’;

END IF;

RESULT <= TEMP(3 DOWNTO 0);

WHEN "111" => --ALU PASS FOR A

RESULT <= A;

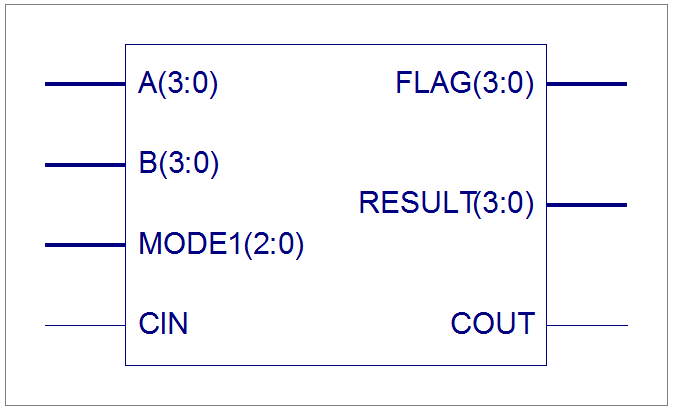
WHEN OTHERS => NULL ;

END CASE;

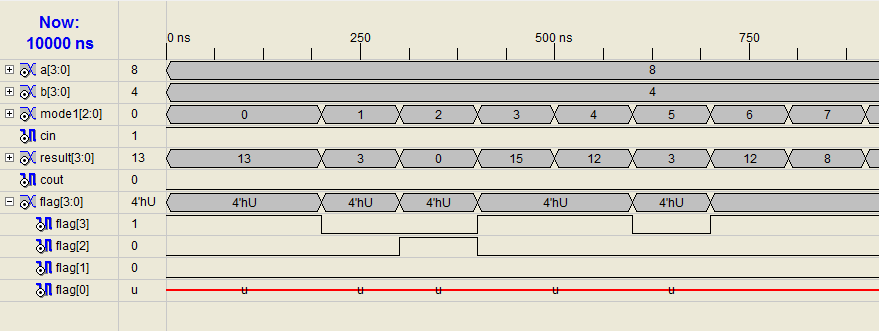
END PROCESS;

END BEHAVIORAL;

**RTL SCHEMATIC**



**SIMULATION RESULT**



**UCF FILE**

NET A(0) LOC=P141;

NET A(1) LOC=P140;

NET A(2) LOC=P139;

NET A(3) LOC=P138;

NET B(0) LOC=P137;

NET B(1) LOC=P135;

NET B(2) LOC=P133;

NET B(3) LOC=P132;

NET MODE1(0) LOC=P131;

NET MODE1(1) LOC=P130;

NET MODE1(2) LOC=P128;

NET CIN LOC=P122;

NET COUT LOC=P162;

NET FLAG(0) LOC=P152;

NET FLAG(1) LOC=P167;

NET FLAG(2) LOC=P150;

NET FLAG(3) LOC=P166;

NET RESULT(0) LOC=P161;

NET RESULT(1) LOC=P172;

NET RESULT(2) LOC=P156;

NET RESULT(3) LOC=P171;

**Practical No. 4**

**KEYPAD INTERFACING**

**Aim:**

To write a VHDL code for to interface 4x4 matrix keypad with FPGA and display result on seven segment display.

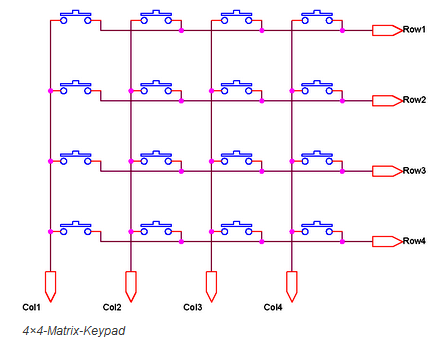
**Theory:**

**Matrix Keypad**

Matrix Keypads are commonly used in calculators, telephones etc where a number of input switches are required. We know that matrix keypad is made by arranging push button switches in row and columns. In the straight forward way to connect a 4×4 keypad (16 switches) to FPGA we need 16 inputs pins.

4×4-Matrix-Keypad

The status of each keys can be determined by a process called Scanning. For the sake of explanation lets assume that all the column pins (Col1 – Col4) are connected to the OUTPUT pins and all the row pins are connected to the INPUT pins of the FPGA. In the normal case all the rows pins are pulled up (HIGH state) by internal or external pull up resistors. Now we can read the status of each switch through scanning.

1. A logic LOW is given to column0 and others HIGH
2. Now each Row is scanned. If any switch is pressed corresponding row will pulled down (logic LOW) and we can detect the pressed key.
3. This process is repeated for all columns.
4. 

**VHDL PROGRAM**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.STD\_LOGIC\_ARITH.ALL;

USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;

-------------------------------------------------------------------------------

ENTITY KEYPAD\_INTERFACE IS

PORT (

CLK : IN STD\_LOGIC;

RESET : IN STD\_LOGIC;

ROWS : IN STD\_LOGIC\_VECTOR(3 DOWNTO 0);

COLUMNS: OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

KEY\_PRESS : OUT STD\_LOGIC;

SEG\_EN : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

SEG\_DISP : OUT STD\_LOGIC\_VECTOR(7 DOWNTO 0)

);

END KEYPAD\_INTERFACE;

--------------------------------------------------------------------------

ARCHITECTURE KEYPAD\_ARCH OF KEYPAD\_INTERFACE IS

----------------------------------------------------------------------

SIGNAL SCAN\_CLK : STD\_LOGIC; --SCAN CLOCK

SIGNAL CLK\_DIV : STD\_LOGIC\_VECTOR(14 DOWNTO 0);

SIGNAL COL\_SIGNAL : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

SIGNAL KEY\_IN : STD\_LOGIC;

SIGNAL KEY\_CODE\_S : STD\_LOGIC\_VECTOR(7 DOWNTO 0);

BEGIN

SEG\_EN <= "0001" ;

KEY\_PRESS <= KEY\_IN;

COLUMNS <= COL\_SIGNAL;

SCAN\_CLK <= CLK\_DIV(14);

-- CLOCK DIVIDER

PROCESS(CLK,RESET)

BEGIN

IF RESET = '1' THEN

CLK\_DIV <= (OTHERS=>'0');

ELSIF CLK'EVENT AND CLK='1' THEN

CLK\_DIV <= CLK\_DIV +1;

END IF ;

END PROCESS;

-- SCAN PATTERN GENERATOR

PROCESS(SCAN\_CLK,RESET)

BEGIN

IF(RESET = '1')THEN

COL\_SIGNAL <= "1110";

ELSIF(SCAN\_CLK'EVENT AND SCAN\_CLK = '0')THEN

COL\_SIGNAL <= COL\_SIGNAL(0) & COL\_SIGNAL(3 DOWNTO 1);

END IF;

END PROCESS;

--------------------------------------------------------------------------

-- PROCESS FOR KEY PRESS DETECTION

KEY\_IN <= '1' WHEN ROWS(0)='0' OR ROWS(1)='0' OR ROWS(2)='0' OR ROWS(3)='0' ELSE

'0';

-------------------------------------------------------------------------------

-- FOR LATCHING THE VALID KEY PRESSED

PROCESS(KEY\_IN, ROWS, RESET )

BEGIN

IF RESET = '1' THEN

KEY\_CODE\_S <= (OTHERS=>'0');

ELSIF( KEY\_IN = '1'AND KEY\_IN'EVENT )THEN

KEY\_CODE\_S <= COL\_SIGNAL & ROWS;

END IF;

END PROCESS;

-------------------------------------------------------------------------------

-- CHANGING THE KEY VALUES IN 7-SEGMENT CODING.

WITH KEY\_CODE\_S SELECT

-- ABCDEFG

SEG\_DISP <= "00000011" WHEN "11101110", -- 0

"10011111" WHEN "11101101", -- 1

"00100101" WHEN "11101011", -- 2

"00001101" WHEN "11100111", -- 3

"10011001" WHEN "11011110", -- 4

"01001001" WHEN "11011101", -- 5

"01000001" WHEN "11011011", -- 6

"00011111" WHEN "11010111", -- 7

"00000001" WHEN "10111110", -- 8

"00001001" WHEN "10111101", -- 9

"00000101" WHEN "10111011", -- A

"11000001" WHEN "10110111", -- B

"01100011" WHEN "01111110", -- C

"10000101" WHEN "01111101", -- D

"01100001" WHEN "01111011", -- E

"01110001" WHEN "01110111", -- F

"00000000" WHEN OTHERS;

END KEYPAD\_ARCH;

--------------------------------------------------------------------------

**UCF FILE**

NET "Clk" LOC = P183;

NET "Reset" LOC = P58;

NET "Key\_Press" LOC = P34;

NET "Columns[0]" LOC = P153;

NET "Columns[1]" LOC = P152;

NET "Columns[2]" LOC = P151;

NET "Columns[3]" LOC = P150;

NET "Rows[0]" LOC = P147;

NET "Rows[1]" LOC = P146;

NET "Rows[2]" LOC = P145;

NET "Rows[3]" LOC = P144;

#NET "Seg\_Disp[0]" LOC = P5;

NET "Seg\_Disp[1]" LOC = P177;

NET "Seg\_Disp[2]" LOC = P187;

NET "Seg\_Disp[3]" LOC = P180;

NET "Seg\_Disp[4]" LOC = P171;

NET "Seg\_Disp[5]" LOC = P178;

NET "Seg\_Disp[6]" LOC = P181;

NET "Seg\_Disp[7]" LOC = P186;

NET "Seg\_En[0]" LOC = P185;

NET "Seg\_En[1]" LOC = P179;

NET "Seg\_En[2]" LOC = P172;

NET "Seg\_En[3]" LOC = P167;

#NET "Seg\_En[4]" LOC = P9;

#NET "Seg\_En[5]" LOC = P12;

# PlanAhead Generated IO Constraints

NET "Rows[0]" PULLUP;

NET "Rows[1]" PULLUP;

NET "Rows[2]" PULLUP;

NET "Rows[3]" PULLUP;

**Practical – 05**

**Class: - B.E.E&TC Subject: - VLSI Design and Technology**

**Aim: -** To implement an inverter (not gate), NAND, NOR gate & 1 Bit Half Adder using CMOS technology

**Theory :-**

**A] CMOS Inverter:**

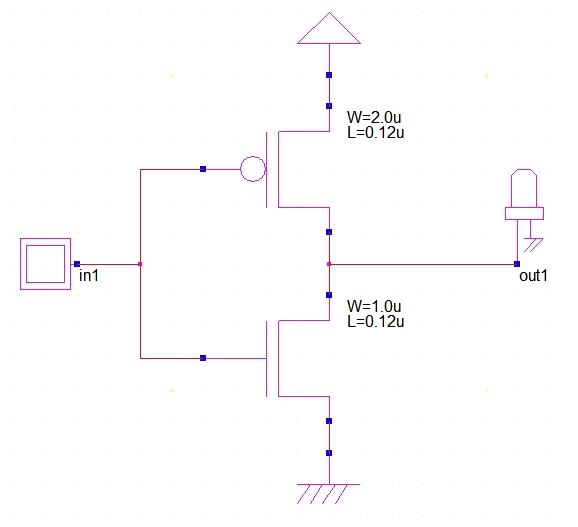
A logic inverter can be constructed by using one P-MOS and N-MOS inverter when there is ‘0’ at input , then output is ‘1’ and vice versa. In general a fully complementary always ha san n-Switch in pull down at output.

Truth table of Inverter:-

|  |  |
| --- | --- |
| INPUT | OUTPUT |
| 0 | 1 |
| 1 | 0 |

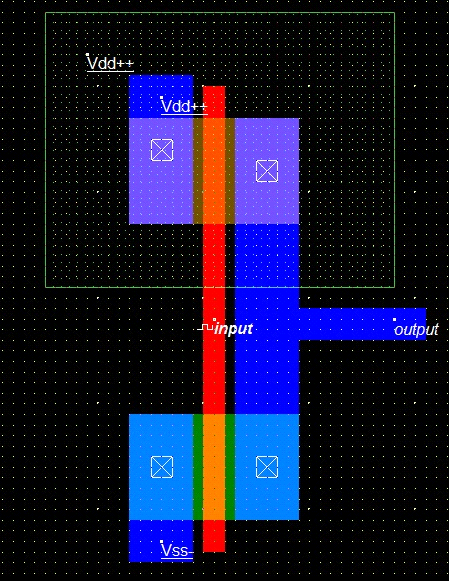
This is truth table of logic inverter where when there is a ‘0’ at the input the output is ‘1’ and when there is ‘1’ at the input one output will zero.

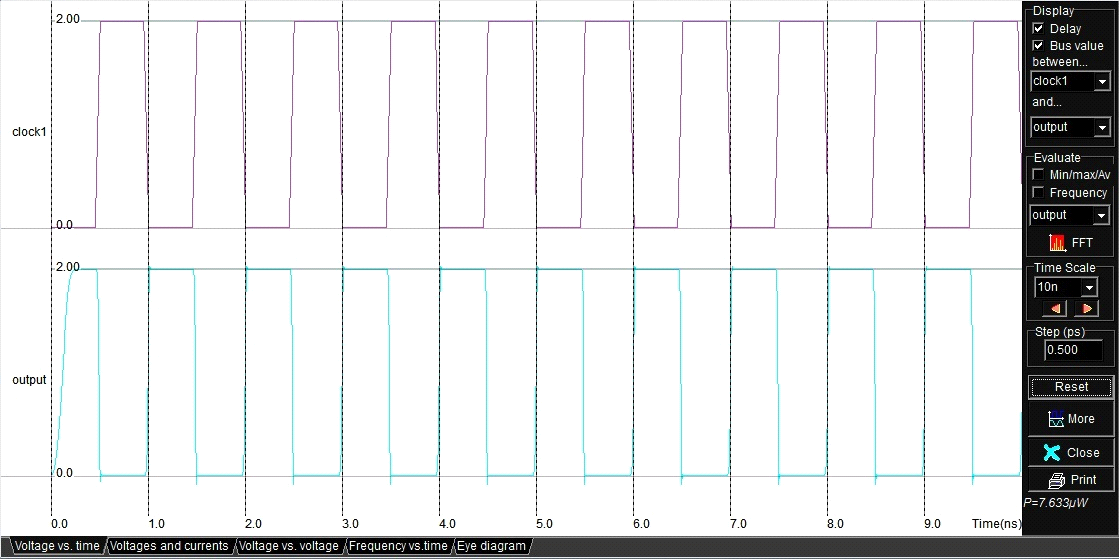
**Schematic of CMOS Inverter**



**Part A: Wp=Wn=1 µm , Ln=Lp=0.2 µm**

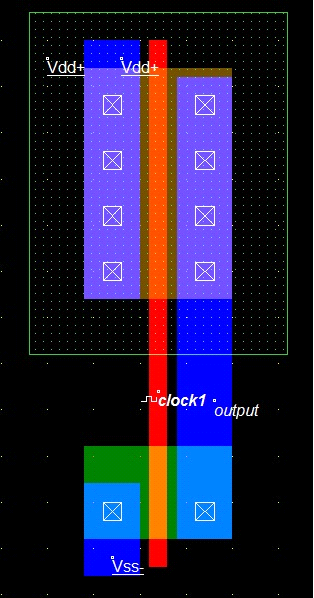
**Design Layout & Timing Diagram of CMOS Inverter**

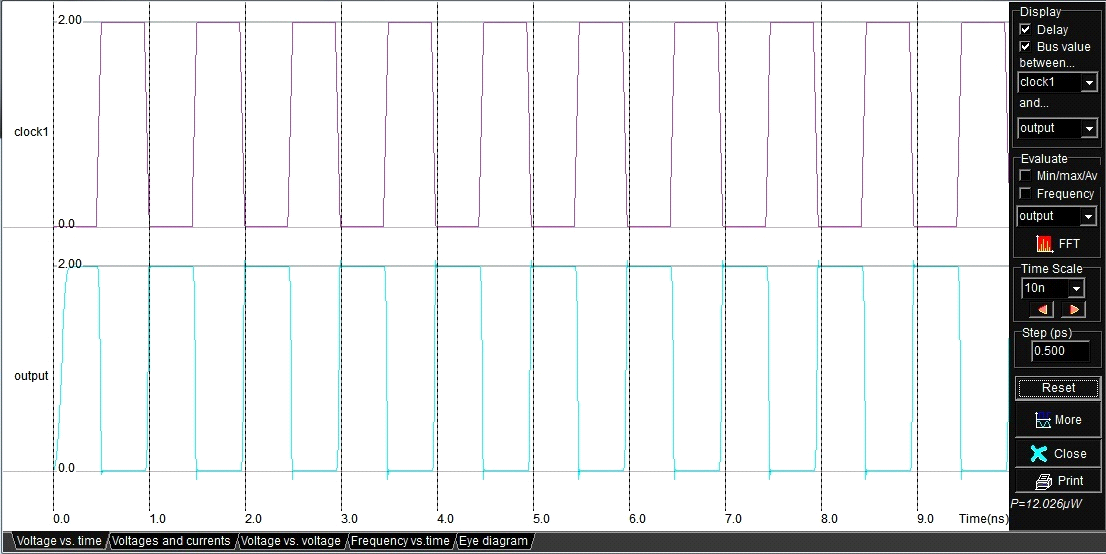




**Part B: Wp=2.5Wn (Wn=1 µm) , Ln=Lp=0.2 µm**

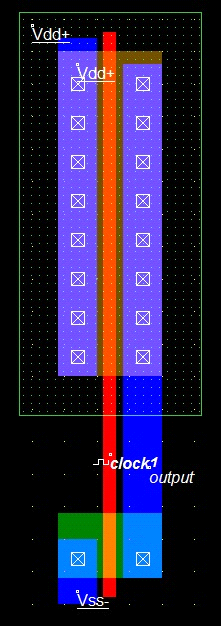
**Design Layout & Timing Diagram of CMOS Inverter**

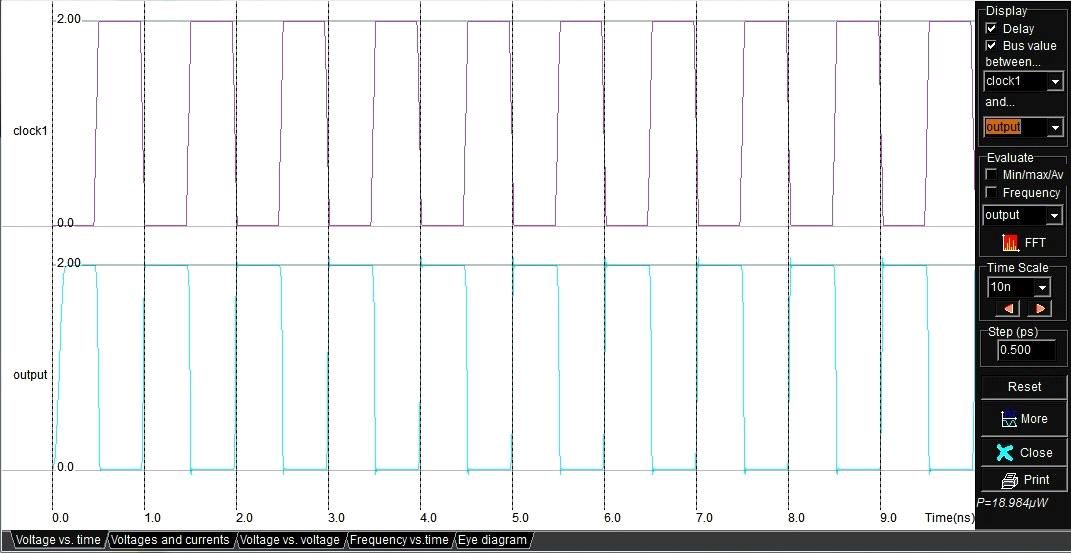




**Part C: Wp=5Wn (Wn=1 µm) , Ln=Lp=0.2 µm**

**Design Layout & Timing Diagram of CMOS Inverter**





**Observation Table:**

* **Effect of Change in Width on different parameters ( Without Capacitor)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr No** | **Width** | **Power Dissipation** | **Rise Time Tr** | **Fall Time Tf** |
| **1** |  |  |  |  |
| **2** |  |  |  |  |
| **3** |  |  |  |  |

* **Effect of Change in Width on different parameters ( With Capacitor=0.01pF)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr No** | **Width** | **Power Dissipation** | **Rise Time Tr** | **Fall Time Tf** |
| **1** |  |  |  |  |
| **2** |  |  |  |  |
| **3** |  |  |  |  |

**Conclusion:** -

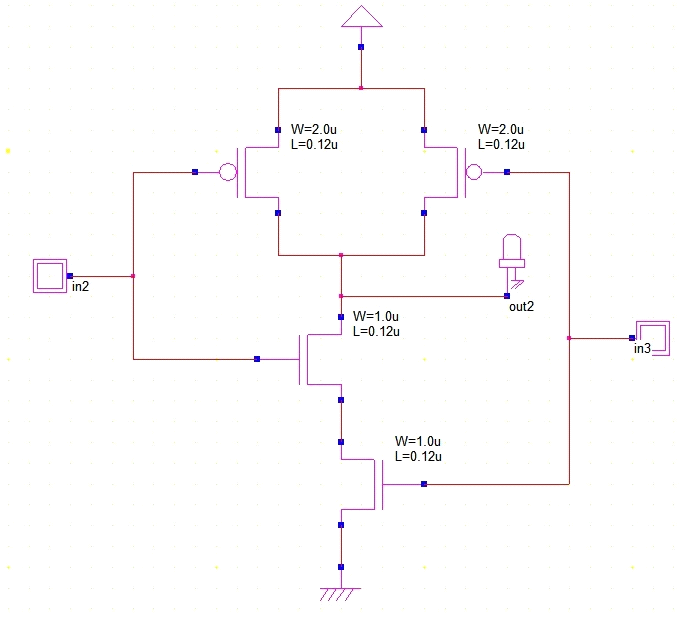
**B] CMOS NAND Gate**

**Theory :-** NAND Gate is formed by connecting an inverter i.e. NOT gate at the output of an AND gate

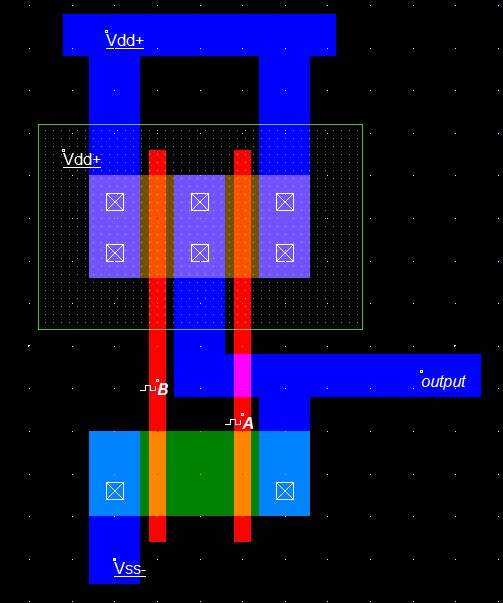
Truth Table:-

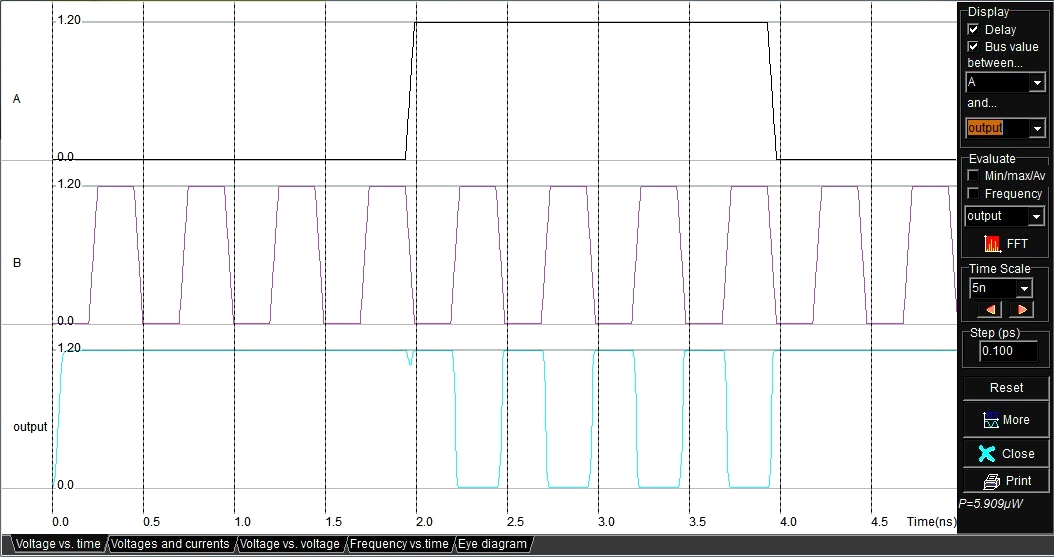
|  |  |  |
| --- | --- | --- |
| Input | | Outputs |
| A | B | C |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Schematic of CMOS NAND Gate:**



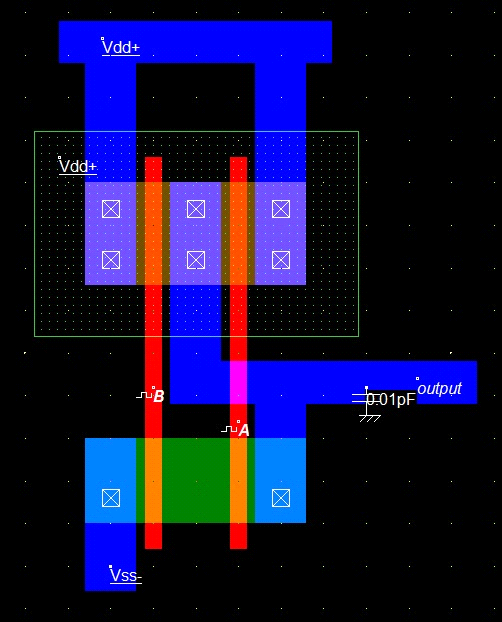
**Part A: Design Layout & Timing Diagram of CMOS NAND Gate without capacitor**

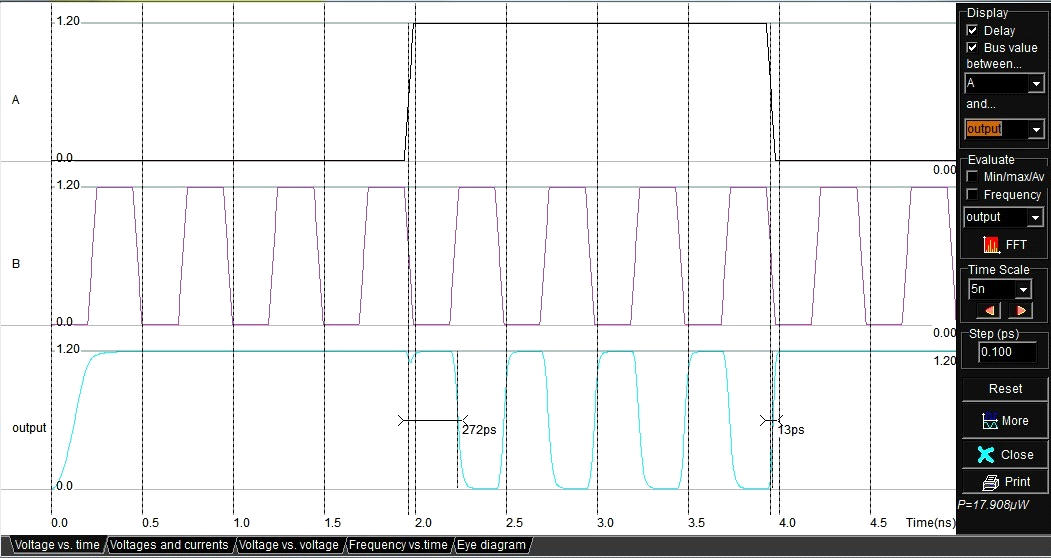




**Part B: Design Layout & Timing Diagram of CMOS NAND Gate**

**with capacitor**





**Observation Table:**

* **Effect of Capacitor on Power Dissipation**

|  |  |  |
| --- | --- | --- |
| **Sr No** | **Capacitor** | **Power Dissipation** |
| **1** | **Without Capacitor** |  |
| **2** | **With Capacitor (0.01pF)** |  |

**Conclusion:** -

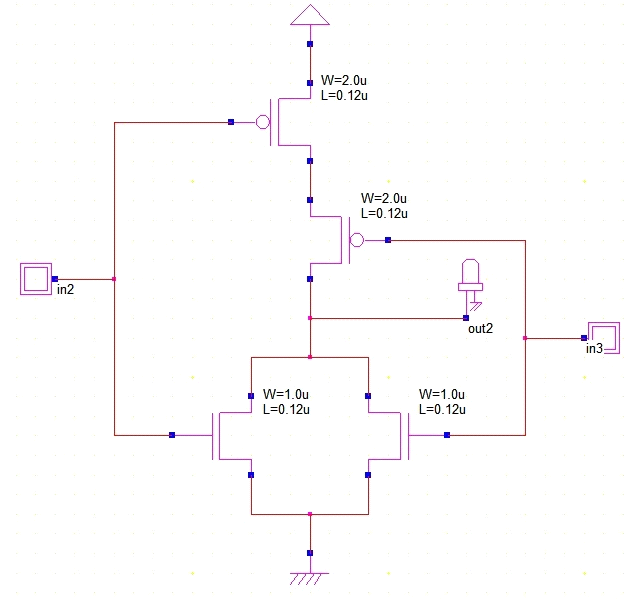
**C] CMOS NOR Gate:**

**Theory :-** A 2 input NOR gate can be constructed using NMOS & PMOS .

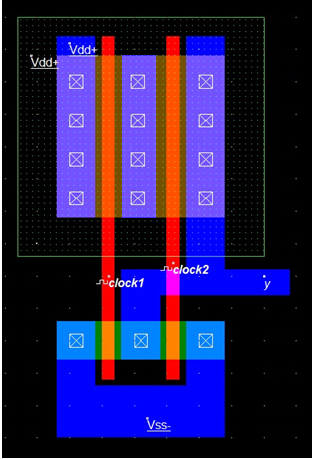
Truth Table:-

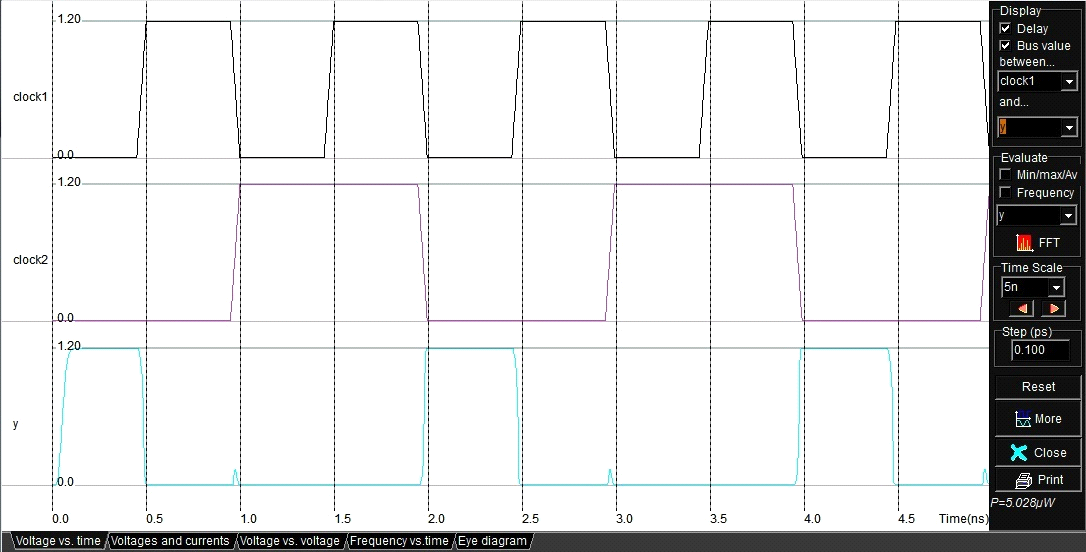
|  |  |  |
| --- | --- | --- |
| Input | | Outputs |
| A | B | C |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

**Schematic of CMOS NOR Gate:**



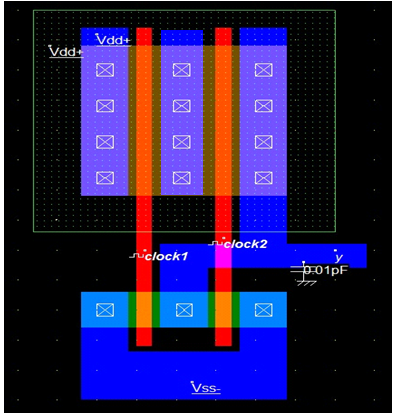
**Part A: Design Layout & Timing Diagram of CMOS NOR Gate without capacitor**

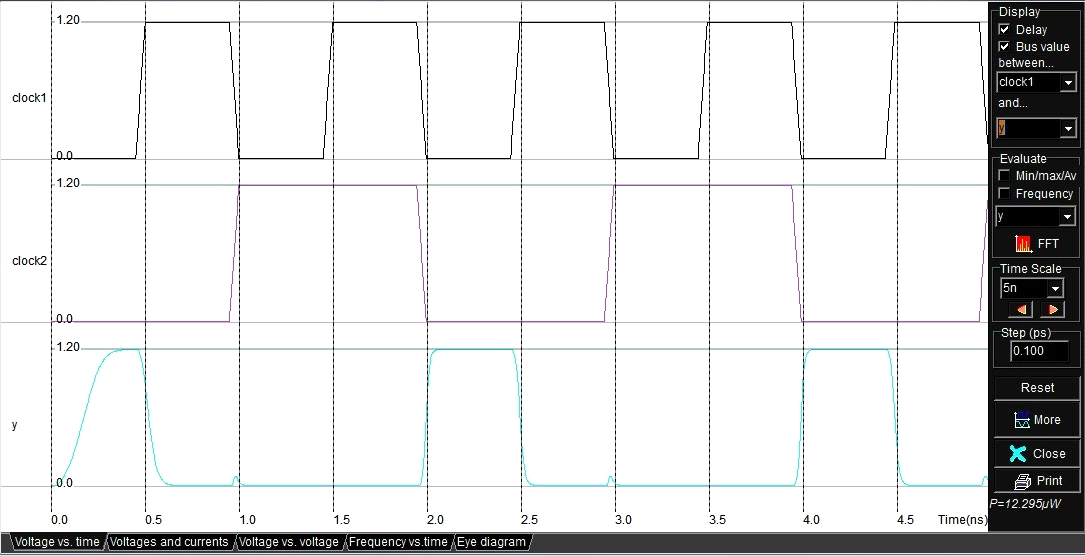




**Part B: Design Layout & Timing Diagram of CMOS NOR Gate**

**with capacitor**





**Observation Table:**

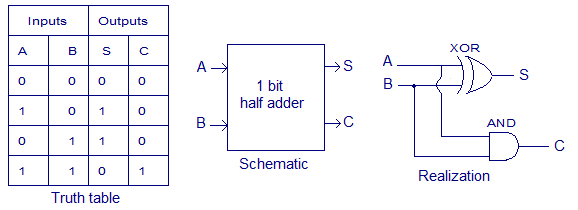
* **Effect of Capacitor on Power Dissipation**

|  |  |  |
| --- | --- | --- |
| **Sr No** | **Capacitor** | **Power Dissipation** |
| **1** | **Without Capacitor** |  |
| **2** | **With Capacitor (0.01pF)** |  |

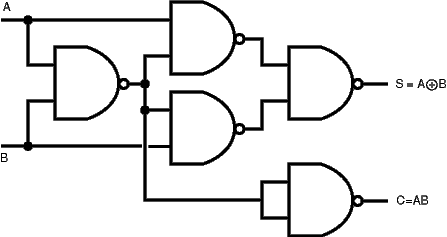
**Conclusion:** -

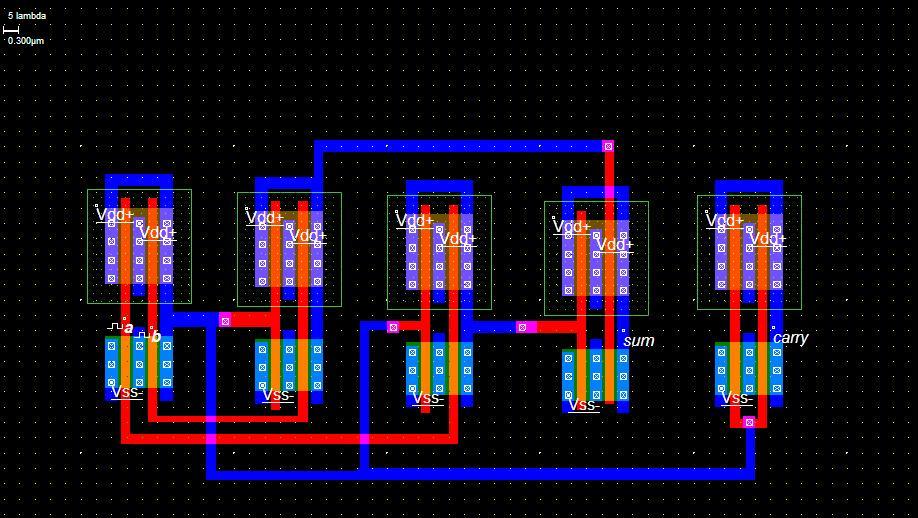
**D] CMOS HALF ADDER:**

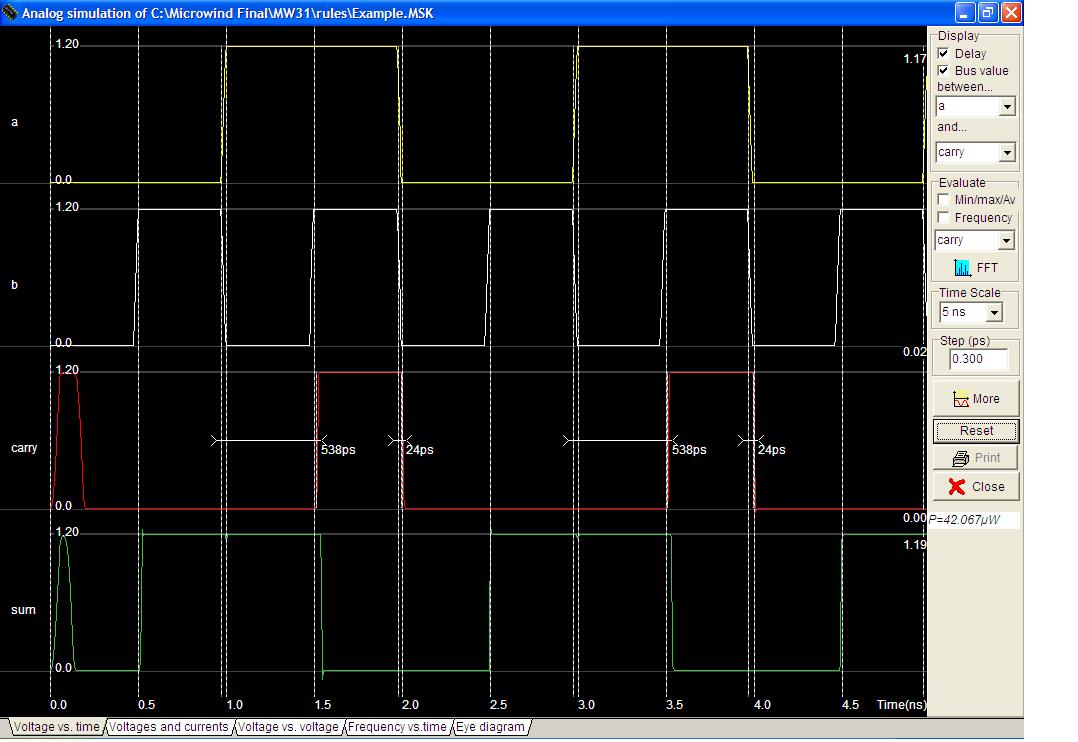
Theory: Half Adder if formed with the help of five CMOS NAND Gates.



**Schematic :**







**Practical - 06**

**Class: - B.E.E&TC Subject: - VLSI Design and Technology**

**Aim: -** To design and implement CMOS 2:1 multiplexer using Transmission gate & also comment on no. of pass transistors required to implement the same.

**Theory: -**

Transmission Gates

* Pass transistors produce degraded outputs
* *Transmission gates* pass both 0 and 1 well



**Multiplexers Using Transmission Gate**

* 2:1 *multiplexer* chooses between two inputs

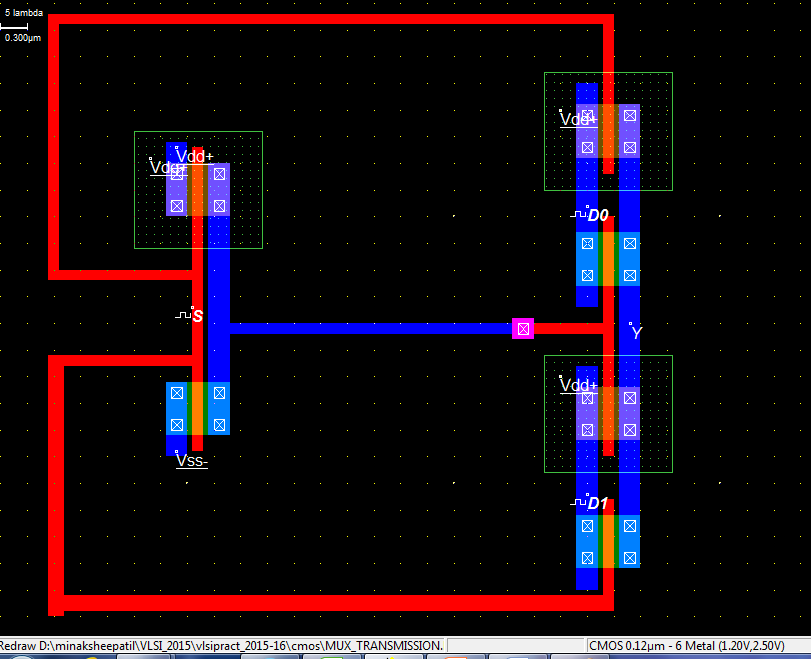
|  |  |  |  |
| --- | --- | --- | --- |
| S | D1 | D0 | Y |
| 0 | X | 0 | 0 |
| 0 | X | 1 | 1 |
| 1 | 0 | X | 0 |
| 1 | 1 | X | 1 |

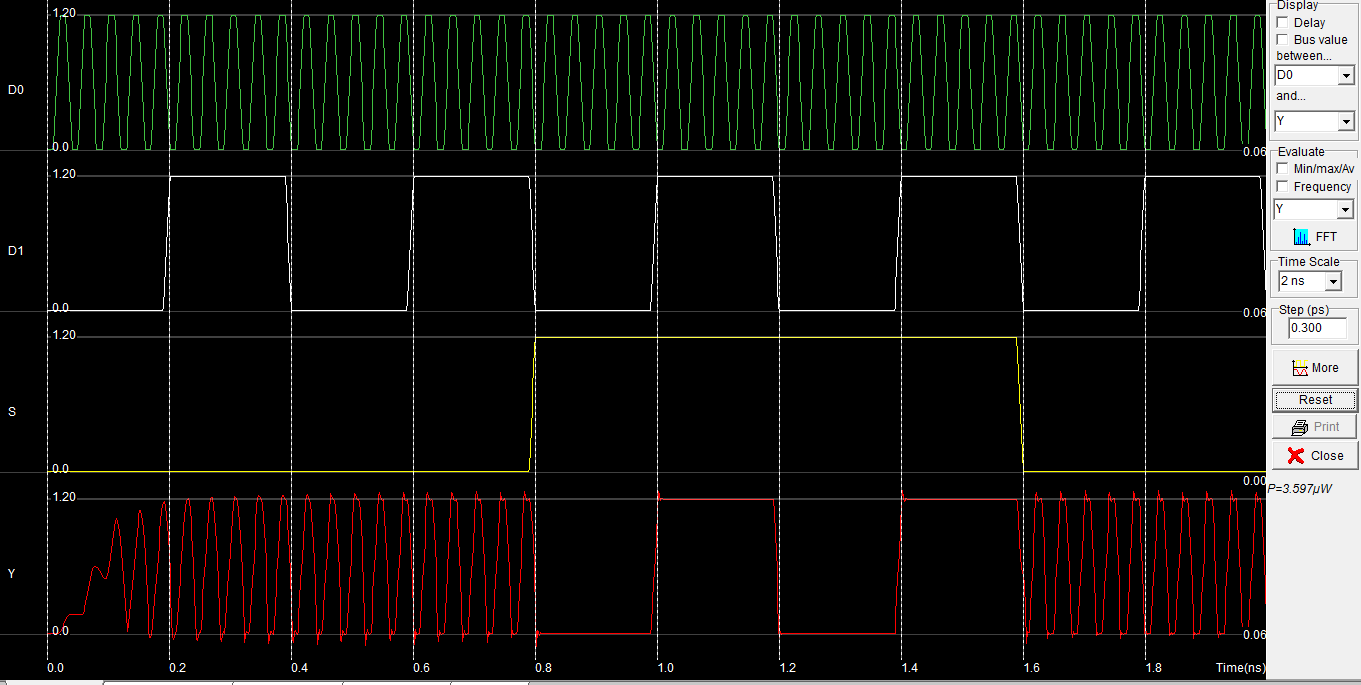
**Gate-Level Mux Design:**



**Transmission Gate Mux:**







**Practical - 07**

**Class: - B.E.E&TC Subject: - VLSI Design and Technology**

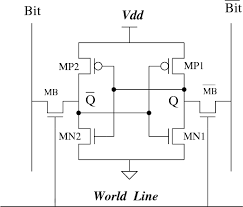
**Aim:** Design of CMOS single bit SRAM cell layout and verify the functionality by simulation

**Procedure:**

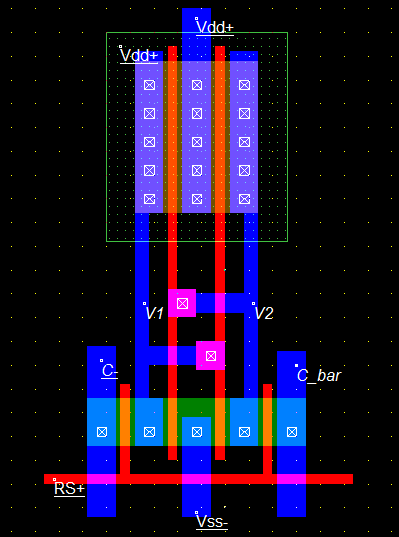
Prepare the CMOS layout of single bit SRAM cell. Do the functional simulation and verify the results.

Comment on the effect of no load and capacitive load on rise time and fall time.

**Circuit Diagram:**

****

**Layout Diagram:**

****

**Simulation Results:**

**Conclusion:**