INDEX

Part One

Design & Simulation Problem Solving

Experiment No: 01

Experiment Name: Introduction to DSCH3 & Microwind software.

Objectives: To know about the simulation software like DSCH3 & Microwind software and also know about the working process and symbols used for these softwires.

Theory:

DSCH3 Software: The DSCH3 program is a logic editor and simulator. DSCH3 is used to validate the architecture of the logic circuit before the microelectronics design is started. DSCH3 provides a user-friendly environment for hierarchical logic design, and fast simulation with delay analysis, which allows the design and validation of complex logic structures. Some techniques for low power design are described in the manual. DSCH3 also features the symbols, models and assembly support for 8051 and 18f64. DSCH3 also includes an interface to SPICE. Following figure shows the DSCH3 user interface. Here is a simple DSCH3 software user interface.

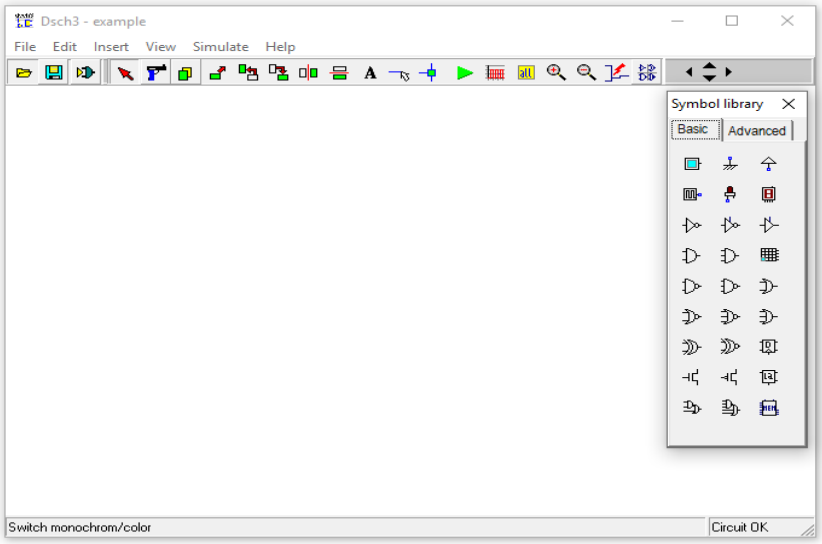
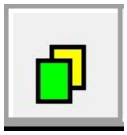


Figure 1DSCH3 user Interface

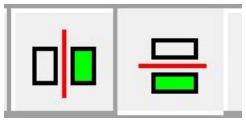
Some simple symbols and their functions given below:

Copy: To copy circuit



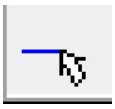
Cut: To Cut or erase any circuit

Flip: To flip anything

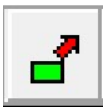


Layout: to move layout

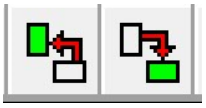
Line: For drawing any line



Move: To move anything



Rotate: To rotate the circuit

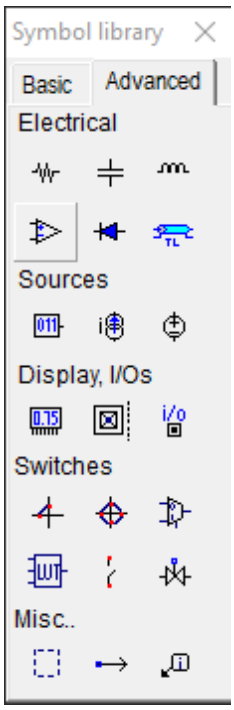


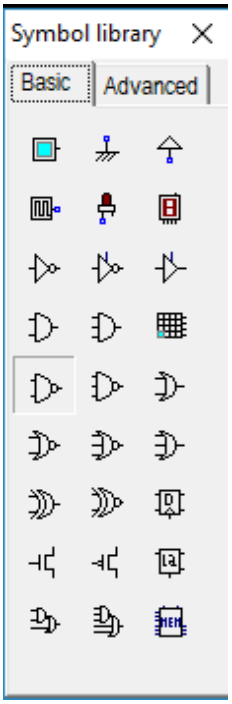
Run: To run the circuit

Save: Saving files

Select: Select any object

Symbol Library: Most important segment of DSCH3 software. Using this software anyone can draw the logic circuits.this segment is divided with the two types basic and advanced logic circuits





Text & Zoom: these two symbols are used for writing text and zooming objects.

Microwind Software: The Microwind program allows to design and simulate an integrated circuit at physical description level. The package contains a library of common logic and analog ICs to view and simulate. Microwind includes all the commands for a mask editor as well as original tools never gathered before in a single module. We gain access to circuit simulation by pressing one single key. The electric extraction of circuit is automatically performed and the analog simulator produces voltage and current curves immediately.

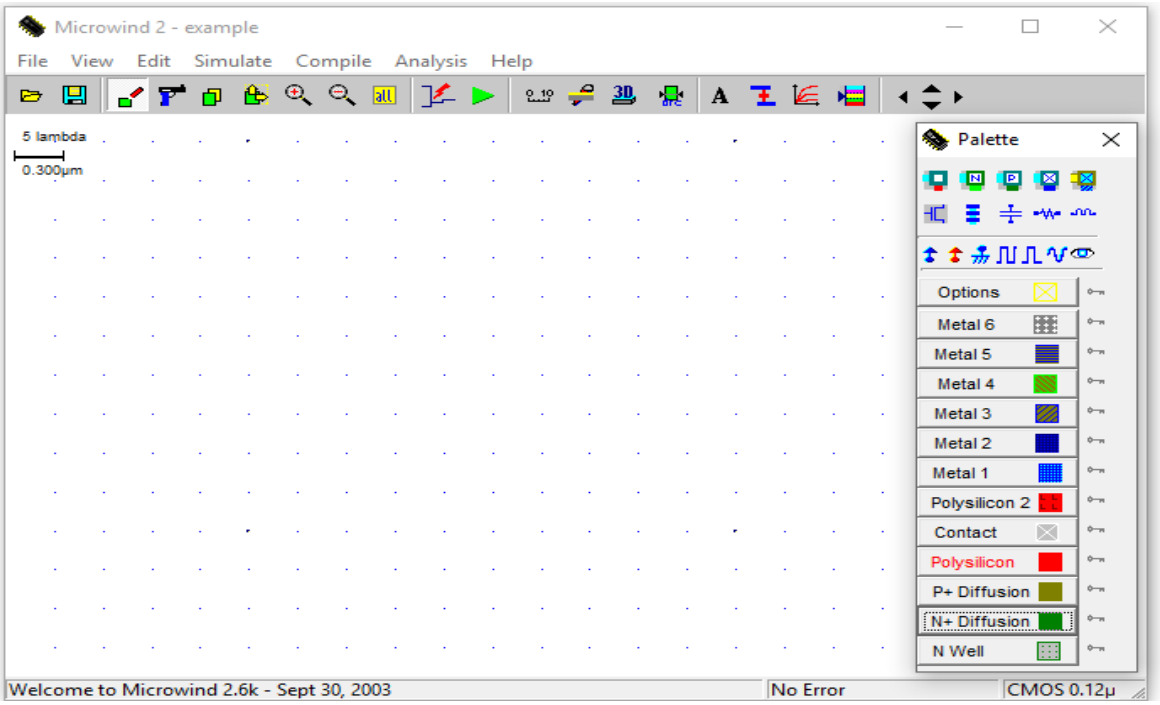


Figure 2: Microwind user Interface

Pallate: Where different types of layout library and active layes present there.

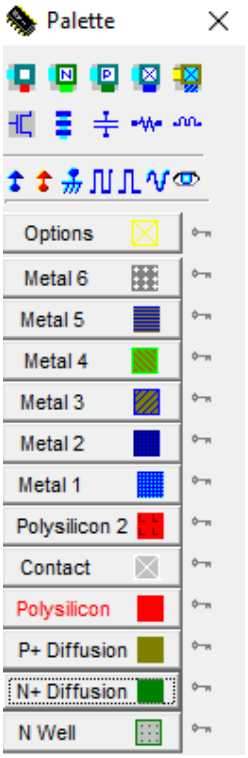


Figure 3: Pallate

Some important symbols and their functions are given below:

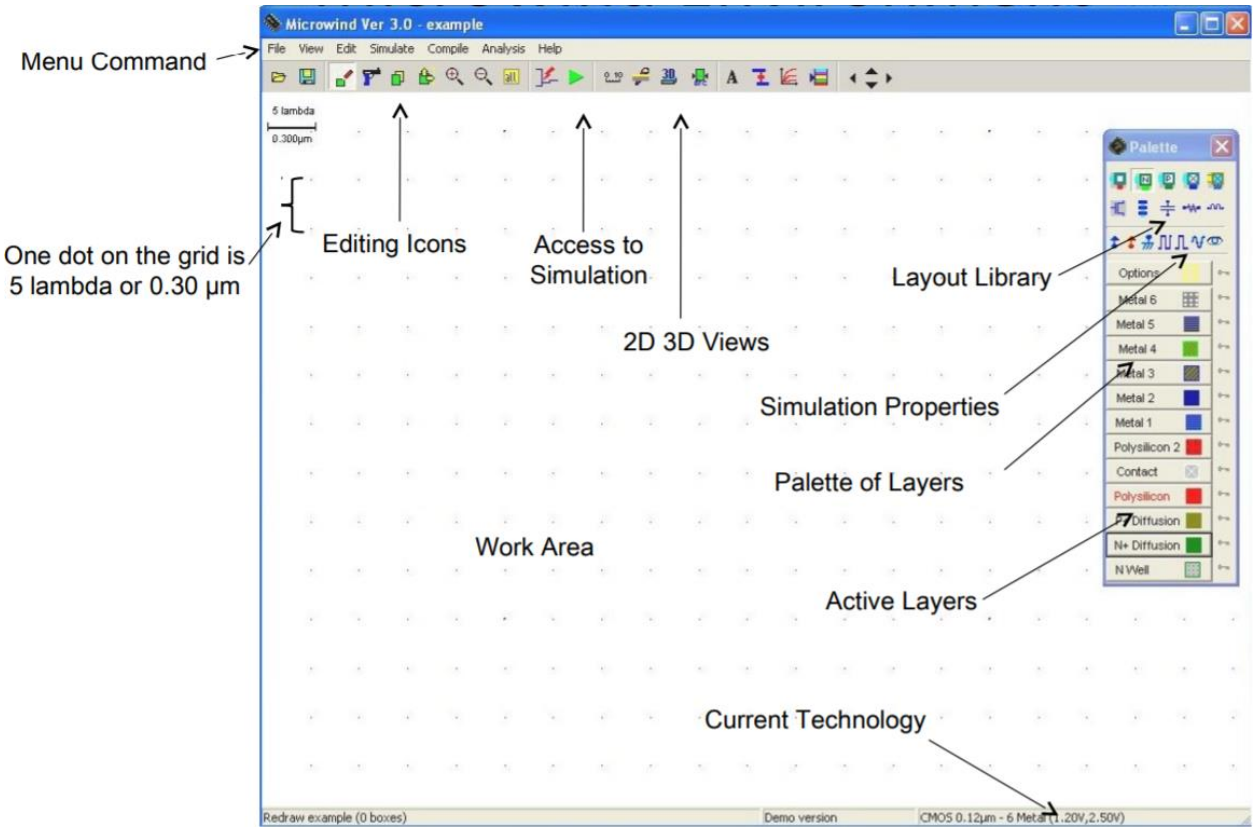


Figure 4: Some symbolic function

Result & Discussion: From this experiment we could know about the simulation software like DSCH3 & Microwind software and also know about the working process and symbols used for these softwires.

Experiment No: 02

Experiment Name: Design a CMOS NOT, NAND, NOR gate using simulator DSCH and show its layout using Microwind.

Objectives: To implement and design a CMOS NOT, NAND, NOR gate using simulator DSCH and show its layout using Microwind.

Theory: The The structure of a CMOS logic gate is based on complementary networks of n-channel and p-channel MOS circuits. Recall that the pMOS switch is good at passing logic signal '1', while nMOS switches are good at passing logic signal '0'.

NOT gate: The CMOS NOT design is detailed in the following figure. Here one p-channel MOS and one n-channel MOS transistors are used as switches. The channel width for pMOS devices is set to twice the channel width for nMOS devices.

When the input signal is logic 0, the nMOS is switched off while the PMOS passes VDD through the output, which turns to 1. When the input signal is logic 1.the pMOS is switched off while the nMOS passes VSS to the output, which goes back to 0. In that simulation, the MOS is considered as a simple switch. The n channel MOS symbol is a device that allows the current to flow between the source and the drain when the gate voltage is "1".

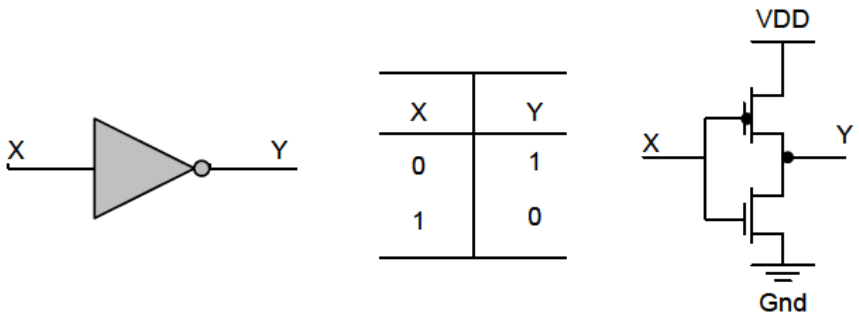


Figure 5:Symbol, Truth Table and CMOS NOT gate

Schematic diagram of the CMOS NOT gate:

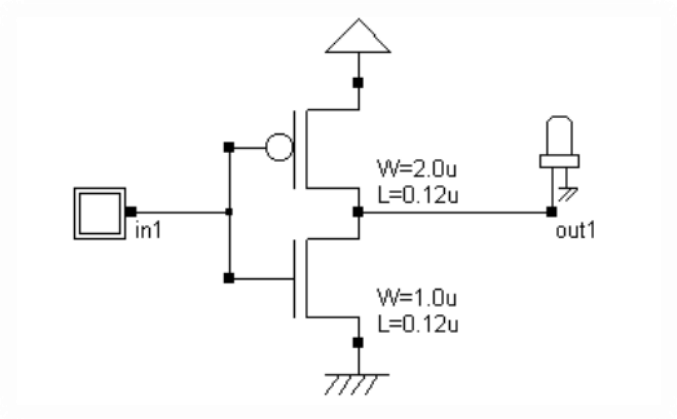


Figure 6: CMOS NOT schematic in DSCH

NAND gate: A NAND gate can be implemented using four FETS i.e. two pFETs and two nFETs as the inputs of the gate is two. pFETs are connected in parallel while nFETs are connected in series, Vdd is supplied to the parallel combination of pFETs while the series combination of nFETs 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 series and parallel combinations as illustrated in NAND circuit.

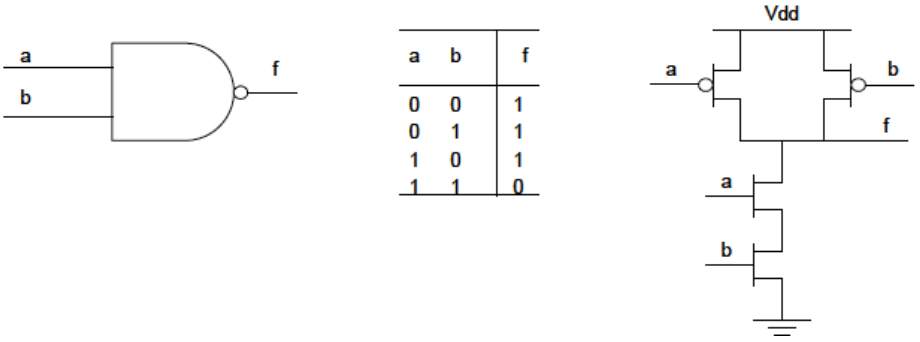


Figure 7: Symbol, Truth Table and CMOS Circuit of NAND gate

Schematic diagram of the CMOS NAND gate:

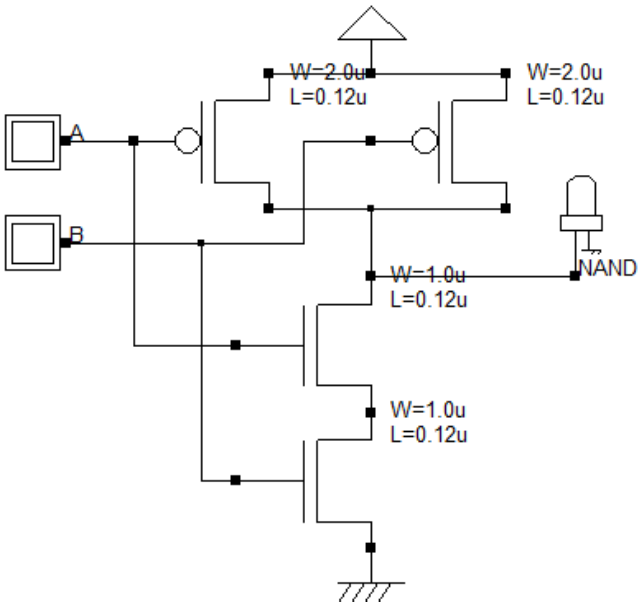


Figure 8: CMOS NAND schematic in DSCH

NOR gate: The two-input NOR gate shown on the left is built from four transistors. The parallel connection of the two n-channel transistors between GND and the gate-output ensures that the gate-output is driven low (logical 0) when either gate input A or B is high (logical 1). The complementary series-connection of the two transistors between VCC and gate-output means that the gate-output is driven high (logical 1) when both gate inputs are low (logical 0).

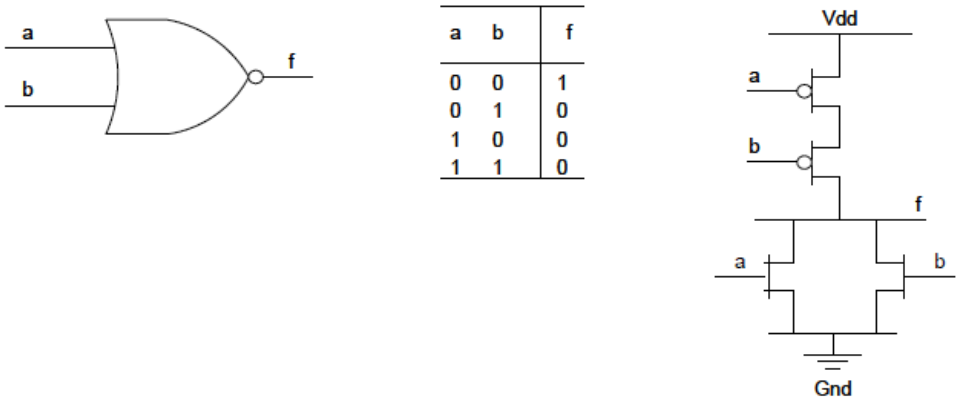


Figure 9: Symbol, Truth Table and CMOS Circuit of NOR gate

Schematic diagram of the CMOS NOR gate:

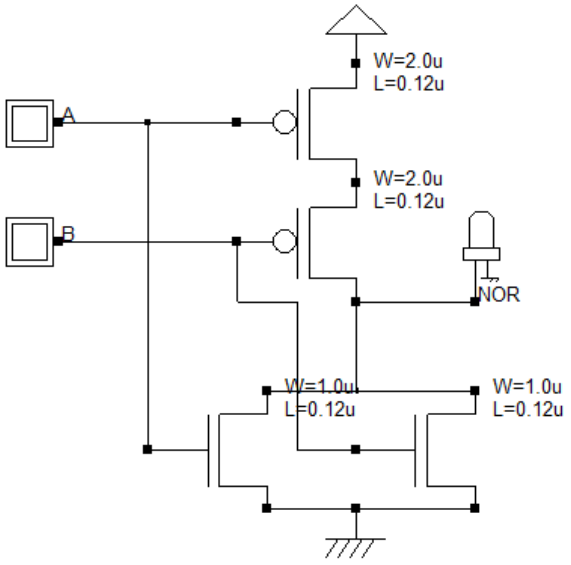


Figure 10: CMOS NOR schematic in DSCH

Working procedure:

- At first, we open Dsch3.exe.
- Then we select nNMOS transistor from *Symbol Library* of the main screen.
- Selecting the nNMOS transistor from *Symbol* to the main screen.
- Similarly selecting supply and ground symbols from *Symbol Library* to the main screen.
- Connecting all symbols as shown in the figure, we can use *Add a line* command to connect different nodes of these symbols
- Adding a Button Symbol to the input and Light symbol to the output of the circuit from Symbols library.
- This completes schematic entry.

Simulation process on DSCH3:

The logic simulation of the NOT gate:

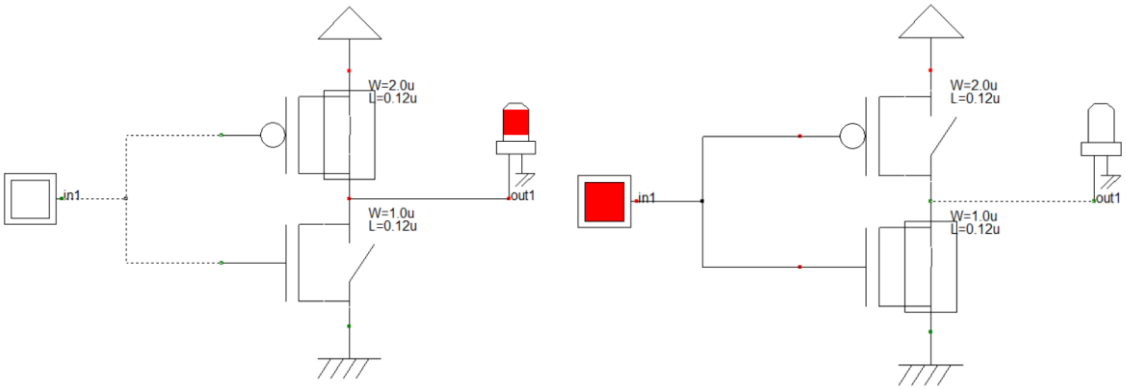


Figure 11: logic simulation of CMOS NOT in DSCH

The logic simulation of the NAND gate:

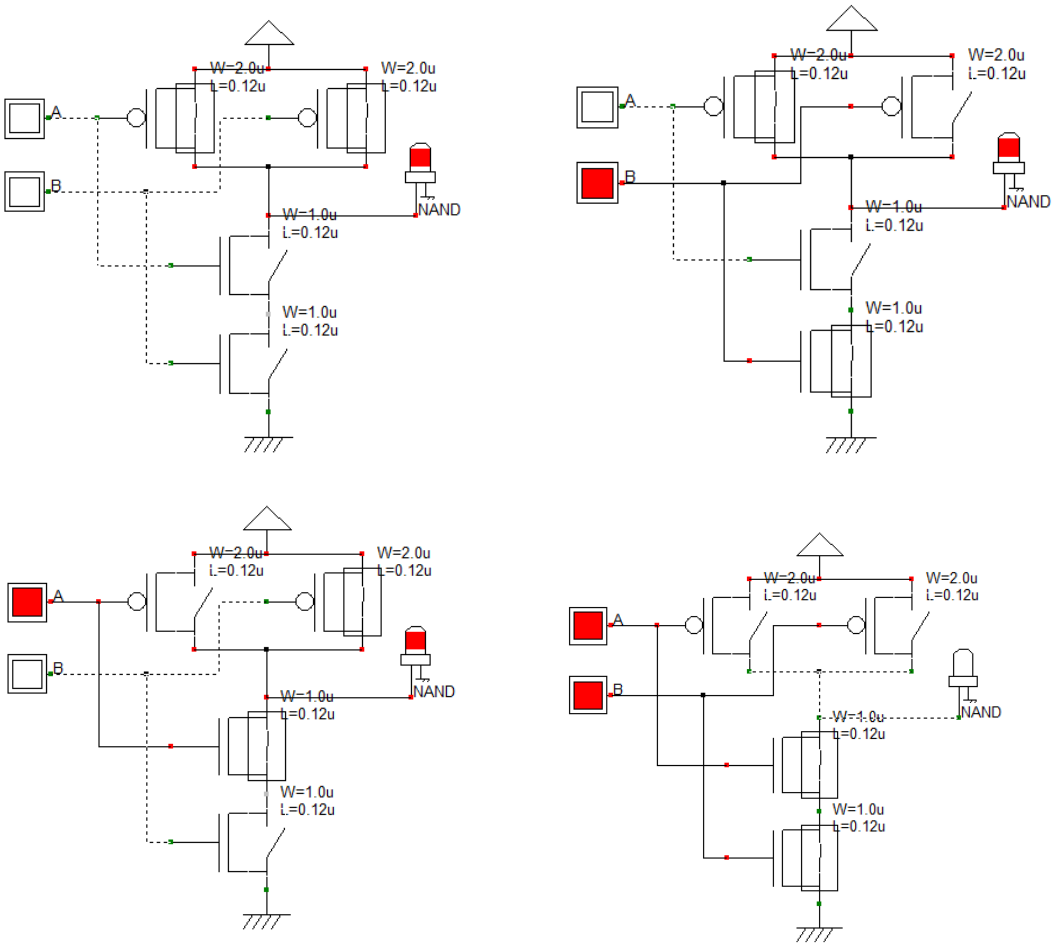


Figure 12: logic simulation of NAND gate in DSCH

The logic simulation of the NOR gate:

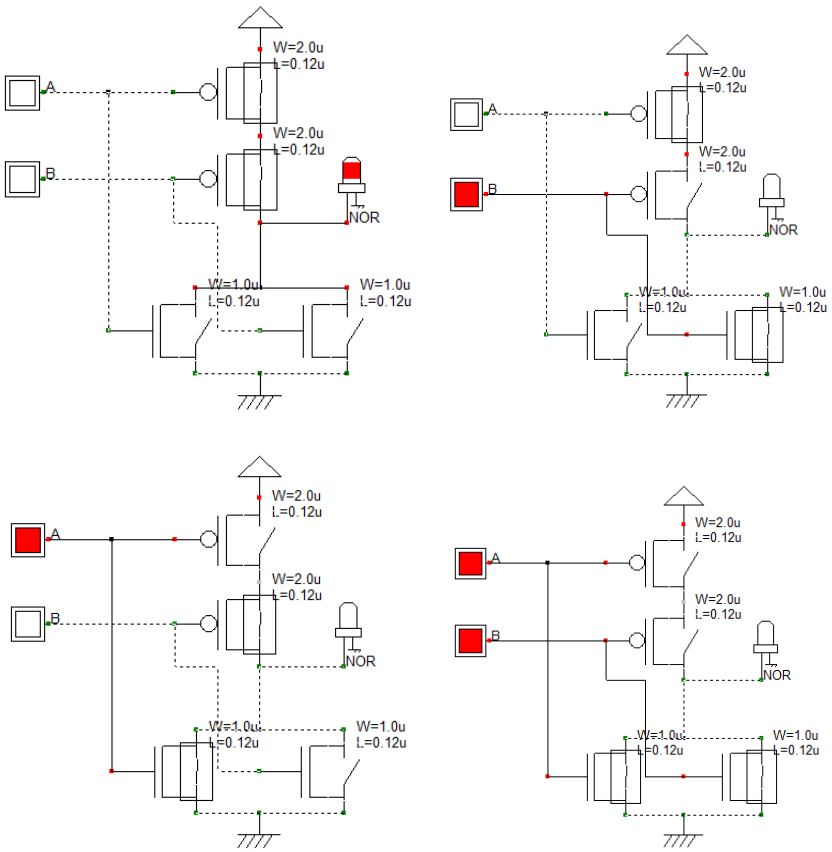


Figure 13: logic simulation of NOR gate in DSCH

Layout Stick diagram using Microwind:

The layout of CMOS NOT looks like follows:

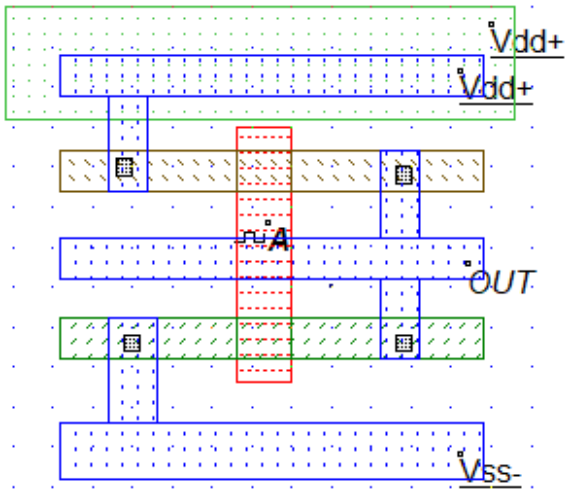


Figure 14: Layout of CMOS NOT

The layout of CMOS NAND looks like follows:

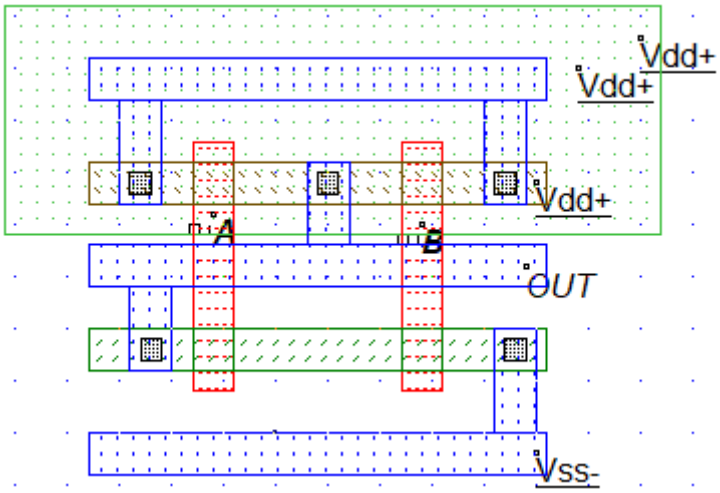


Figure 15: Layout of CMOS NAND

The layout of CMOS NOR looks like follows:

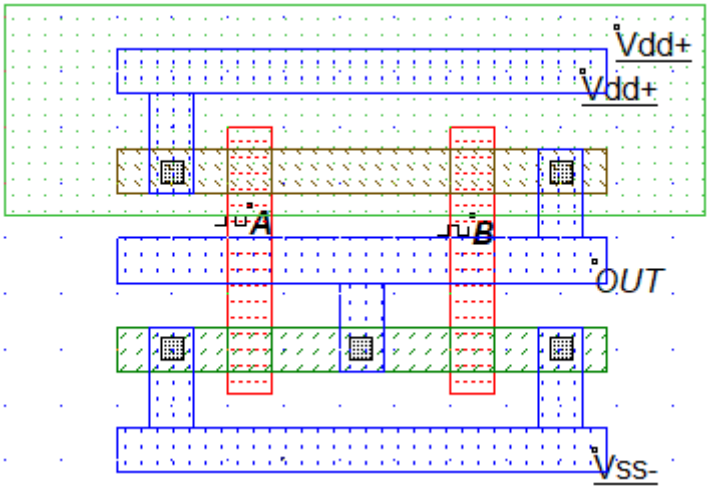


Figure 16: Layout of CMOS NOR

Result & Discussion: From this experiment we could implemented and design a CMOS NOT, NAND, NOR gate using simulator DSCH and show its layout using Microwind.

Experiment No: 03

Experiment Name: Design $\overline{AB+C}$ and $\overline{((A+B)C)}$ using CMOS in DSCH3 simulator and show its layout using Microwind.

Objectives: The objective of this experiment is to design $\overline{AB+C}$ and $\overline{((A+B)C)}$ using DSCH and draw its layout on Microwind.

Theory: $\overline{AB+C}$ and $\overline{((A+B)C)}$ circuit shown below which is built from six transistors and an output. p channel MOS constructs the pull-up section and n channel MOS constructs pull-down sections. There is also three inputs and outputs which are verified with the truth tables.

CMOS circuit for $\overline{AB+C}$:

a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

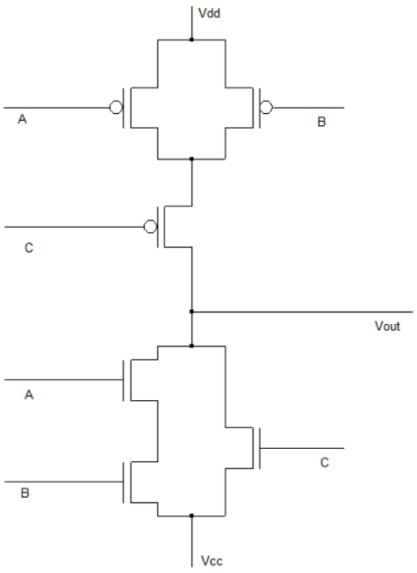
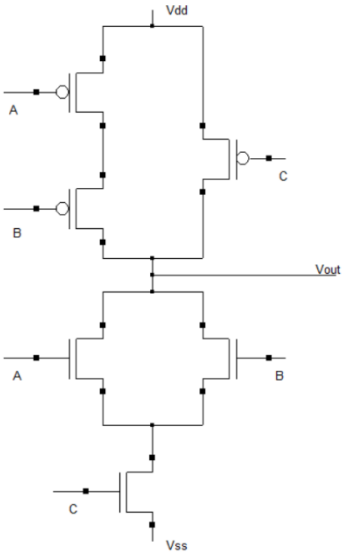


Figure 17: Symbol of Truth Table and CMOS Circuit of (AB+C)'

CMOS circuit for $\overline{((A+B)C)}$:



a	b	c	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Figure 18: Symbol, of Truth Table and CMOS Circuit of (((A+B)C))'

Working procedure:

- Execute DSCH3.exe.
- Drag and drop three pMOS transistor from the tool bar for drawing pull up circuit.
- Also drag and drop three nMOS transistor from the tool bar for drawing pull down circuit.
- Similarly selecting supply and ground symbols from *Symbol Library* and dragging them to the pull up and pull-down circuit.
- Connecting all symbols as shown in the figure, we can use *add a line* command to connect different nodes of these symbols
- Adding input symbol (A,B,C) to the input and LED symbol to the output of the circuit from symbols library to complete the circuit.

Schematic diagram of the CMOS $\overline{AB+C}$:

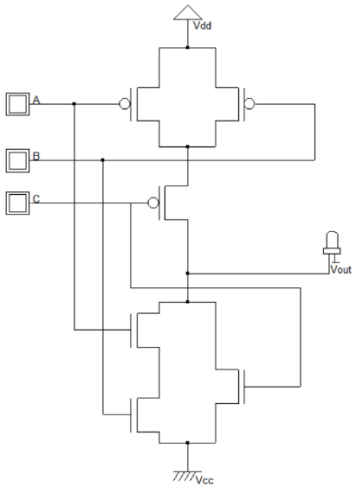


Figure 19: (AB+C)' schematic diagram in DSCH3

The logical simulation of $\overline{AB+C}$ from the true table:

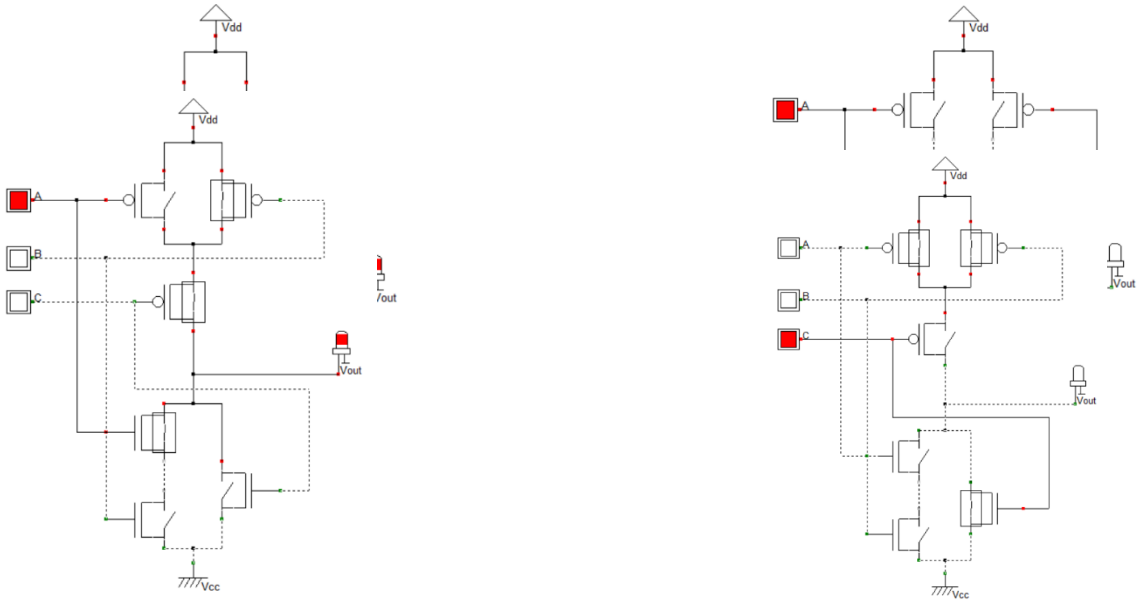


Figure 20: Logical simulation of (AB+C)' in DSCH3

Schematic diagram of the CMOS $\overline{((A+B)C)}$:

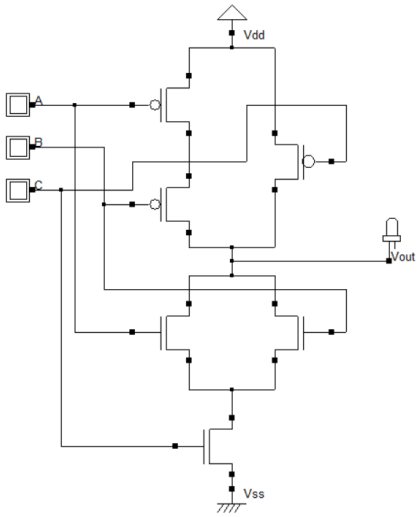


Figure 21: ((A+B)C)' schematic diagram in DSCH3

The logical simulation of $\overline{((A+B)C)}$ from the true table:

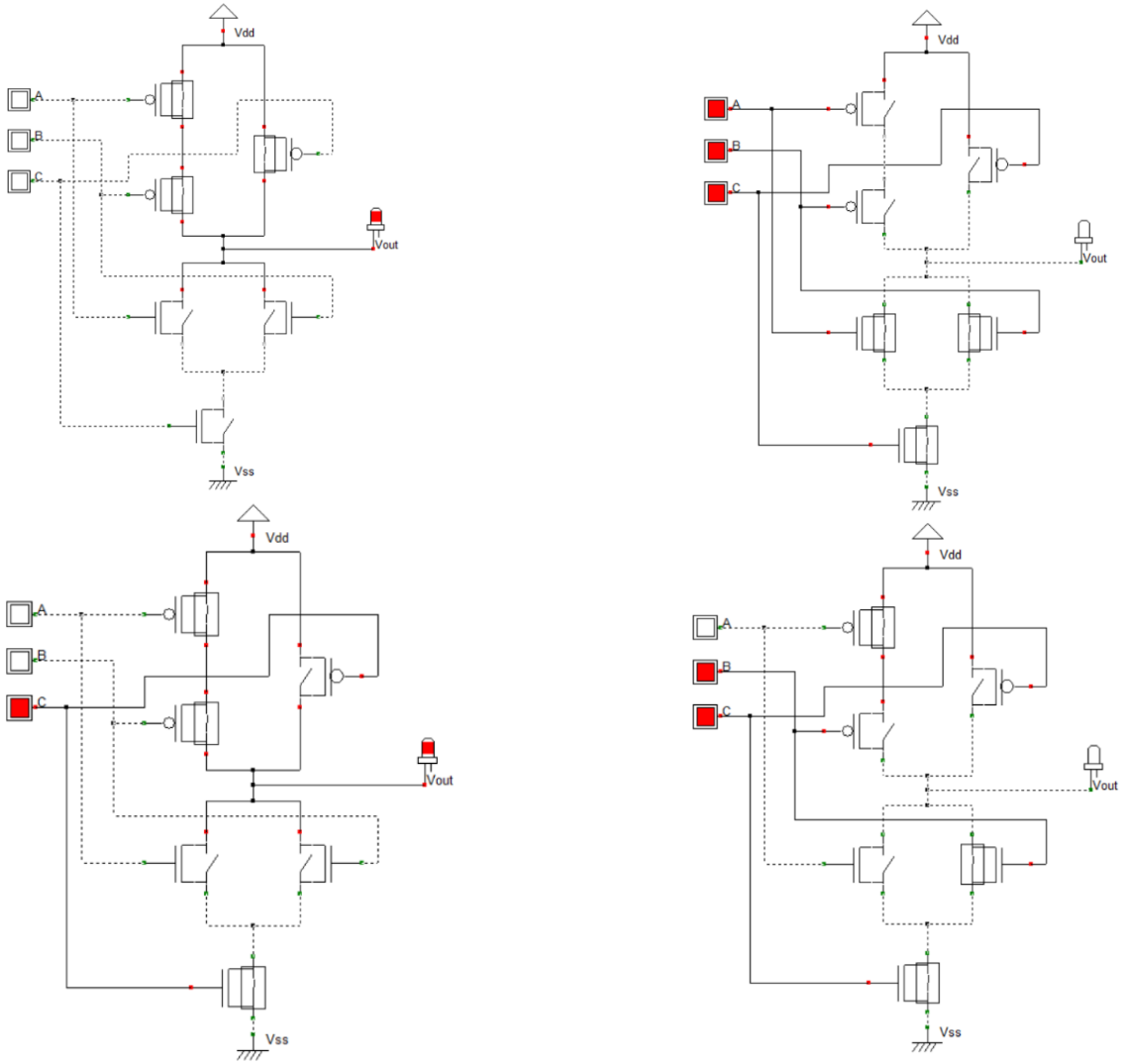


Figure 22: Logic simulation of ((A+B)C)' in DSCH3

Layout of CMOS $\overline{AB+C}$ and $\overline{((A+B)C)}$ in Microwind:

For designing the CMOS $\overline{AB+C}$ and $\overline{((A+B)C)}$ circuit we used Euler Path Theorem from the above and designed the stick diagram layout below. To design the layout we used following tools from the microwind:

$V_{dd}+/V_{ss}-$

P+ Diffusion

N+ Diffusion

Polysilicon

P+/N+ Contact Metal

Euler Path: For $\overline{AB+C}$ and $\overline{((A+B)C)}$ circuit

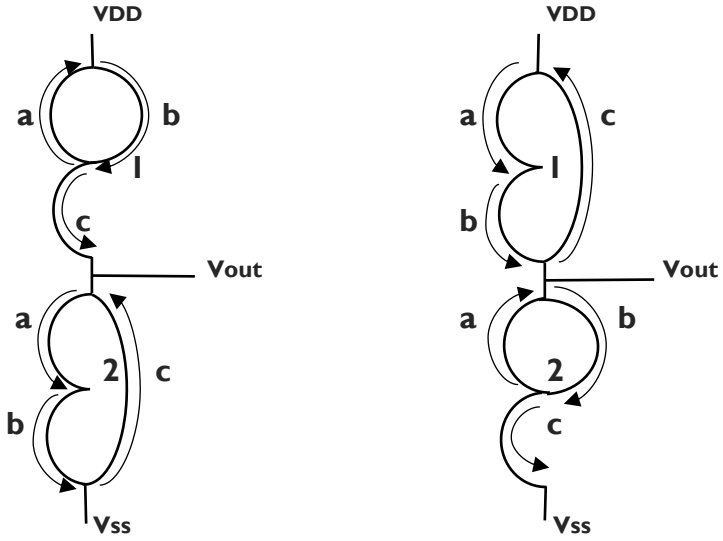


Figure 23: Euler path for $\overline{AB+C}$ and $\overline{((A+B)C)}$ circuit

The layout of CMOS $\overline{AB+C}$ looks like below:

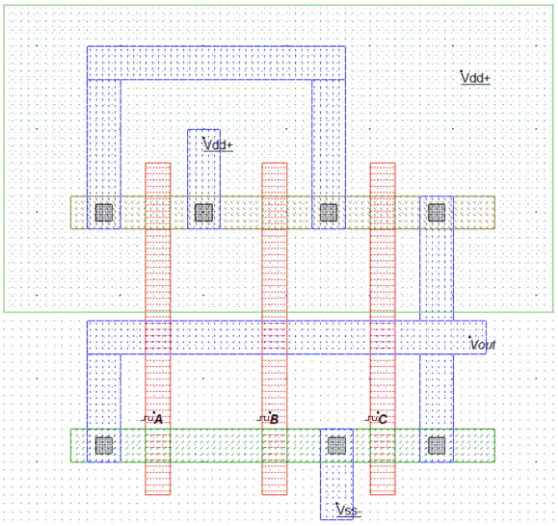
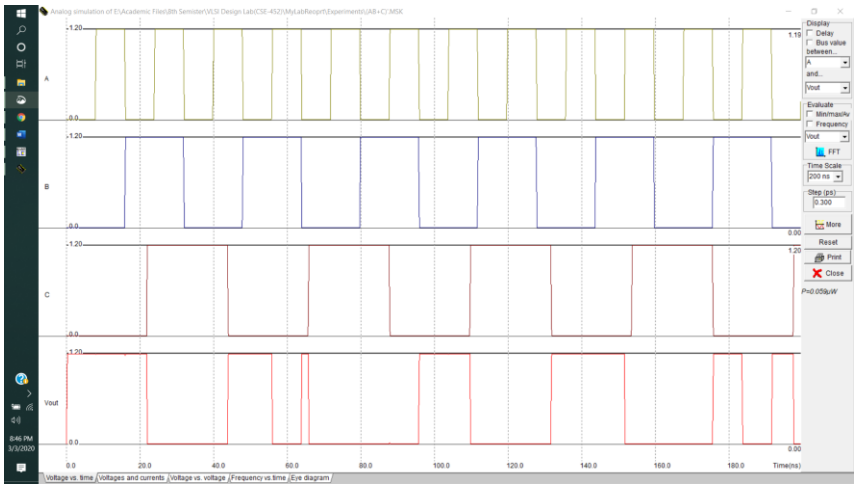


Figure 24 : Layout of CMOS (AB+C)' in Microwind

Result:



The layout of CMOS $(\overline{(A+B)C})$ looks like follows:

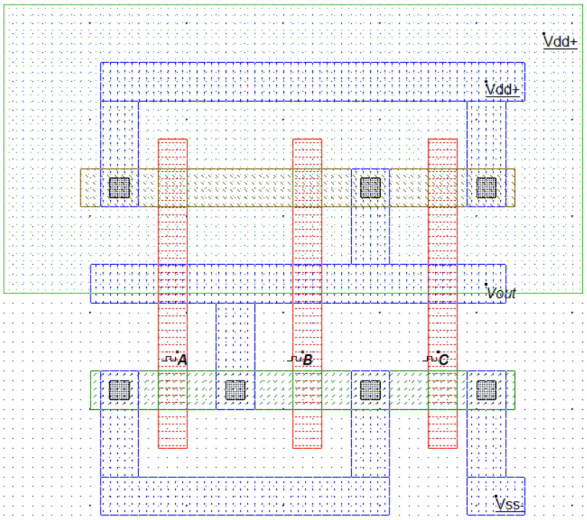


Figure 25 : Layout of ((A+B)C)' CMOS in Microwind

Result:

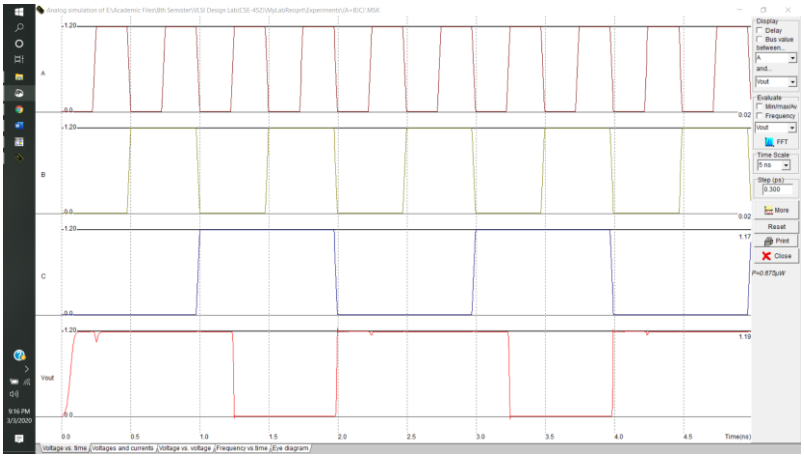


Figure 26 : Simulation of ((A+B)C)' CMOS in Microwind

Discussion: From the above experiment we could completed the stick diagram drawing and its simulation process successfully. We could know about several features of Microwind and learnt how to work with this simulation process.

Experiment No: 04

Experiment Name: Design XOR and XNOR gates using CMOS in DSCH simulator and show layout using MICROWIND.

Objectives: The objective of this experiment is to implement XOR and XNOR gate using DSCH and Microwind.

Theory: The structure of a CMOS logic gate is based on complementary networks of nchannel and p-channel MOS circuits. Recall that the pMOS switch is good at passing logic signal '1', while nMOS switches are good at passing logic signal '0'.

XOR gate: The two-input XOR gate shown on the left is built from twelve transistors. The XOR function is built using AND/OR inverted logic (AOI logic). The function created by the nchannel MOS network is equivalent to (A|~B) and (~A|B). The p-channel MOS network gives the function where all AND functions are transformed into OR, and vice versa. In other words, the pMOS network realizes the function (A & ~B)|(~A & B).

XOR (Exclusive-OR)

$\bar{x}\,y + x\,\bar{y}$

A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	0

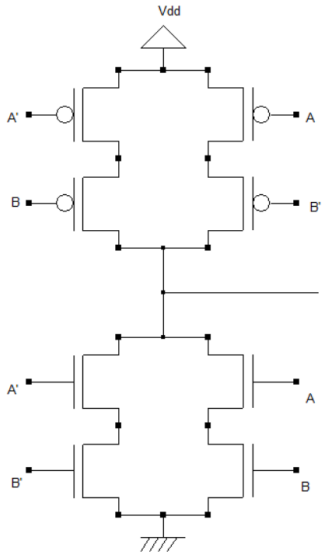
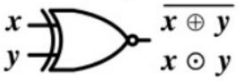


Figure: Symbol, Truth Table and CMOS Circuit XOR gate

XNOR gate: The XNOR gate symbol is shown below. The XNOR circuit is usually an exact copy of the XOR gate, except that the role of the B and ~B signals are opposite in the transmission gate structures. Removing the last inverter is a poor alternative as the output signal is no longer amplified. Adding a supplementary inverter would increase the propagation delay of one stage.

XNOR (Exclusive-NOR)
(Equivalence)



$\bar{x}\,\bar{y} + x\,y$

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

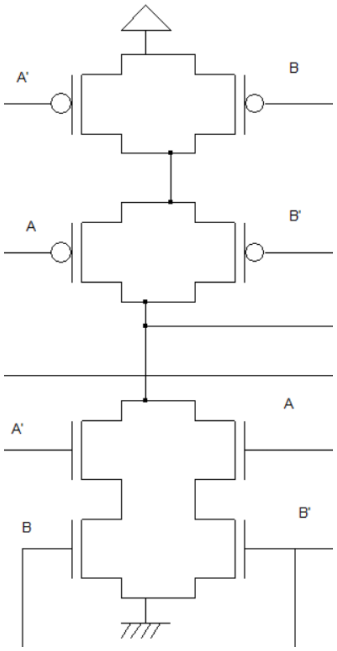


Figure: Symbol, Truth Table and CMOS Circuit XOR gate

Working procedure:

- Opening dsch3.exe.
- Selecting the nNMOS transistor from Symbol Library on top right and dragging it to the main screen.
- Selecting the nNMOS transistor from Symbol Library on top right and dragging it to the main screen.
- Similarly selecting supply and ground symbols from Symbol Library and dragging them to the main screen.
- Connecting all symbols as shown in the figure, We can use Add a line command to connect different nodes of these symbols
- Adding a Button Symbol to the input and Light symbol to the output of the circuit from Symbols library.
- This completes schematic entry.

XOR and XNOR Circuit Design on DSCH3: -

The circuit diagram will look like as follows-

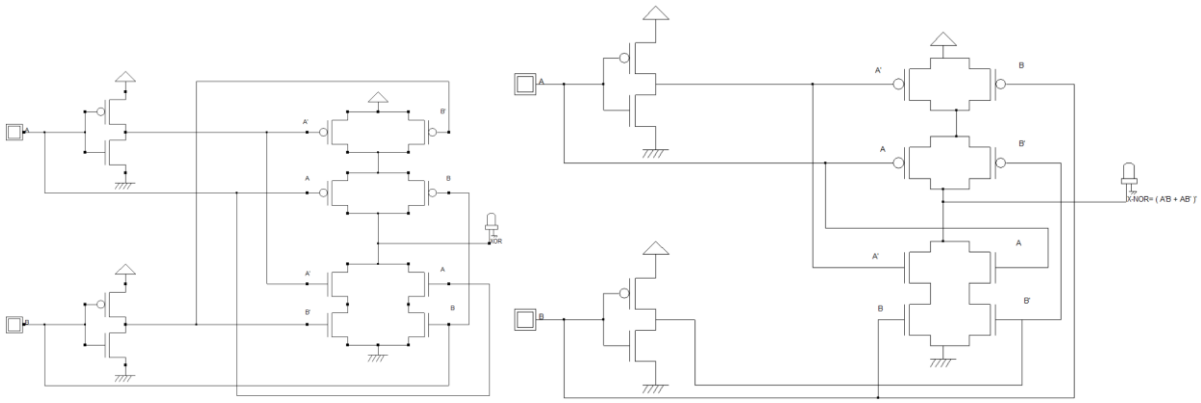


Figure: XOR and XNOR gate schematic in DSCH3

State transitions of following circuit for some input is shown below-

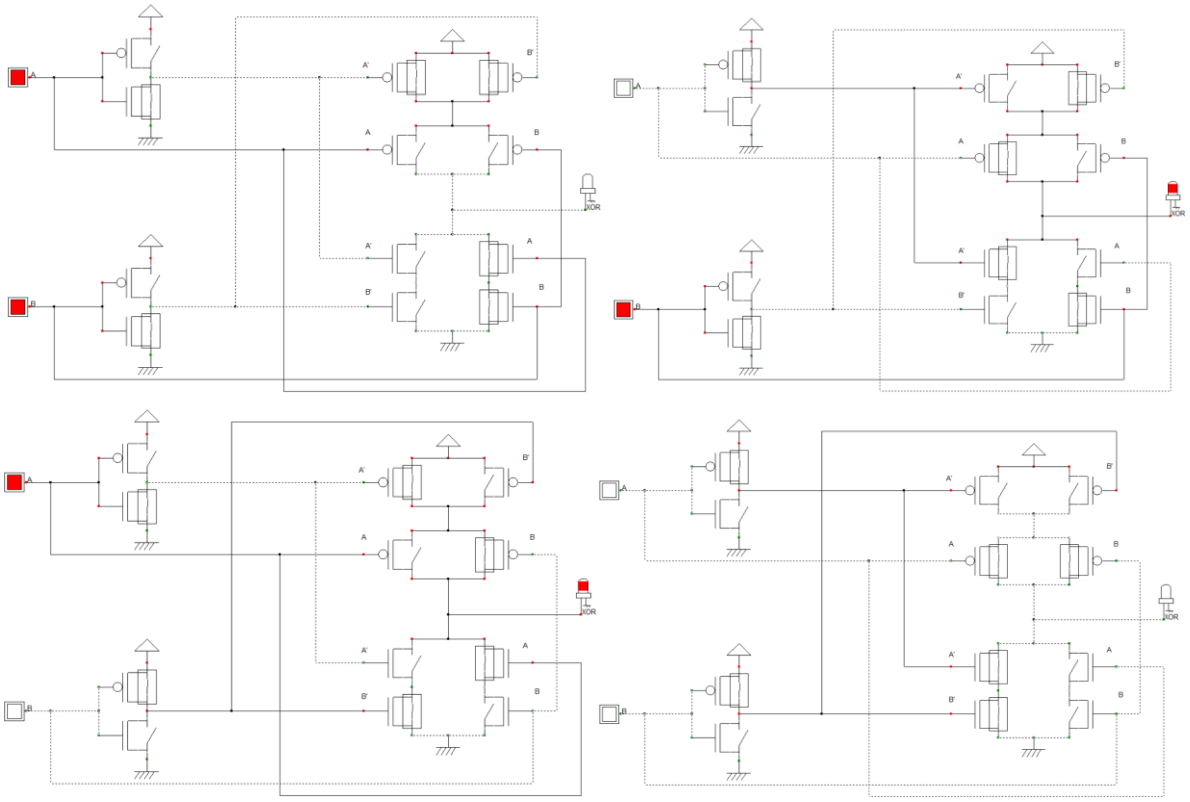


Figure: State transition of XOR in DSCH

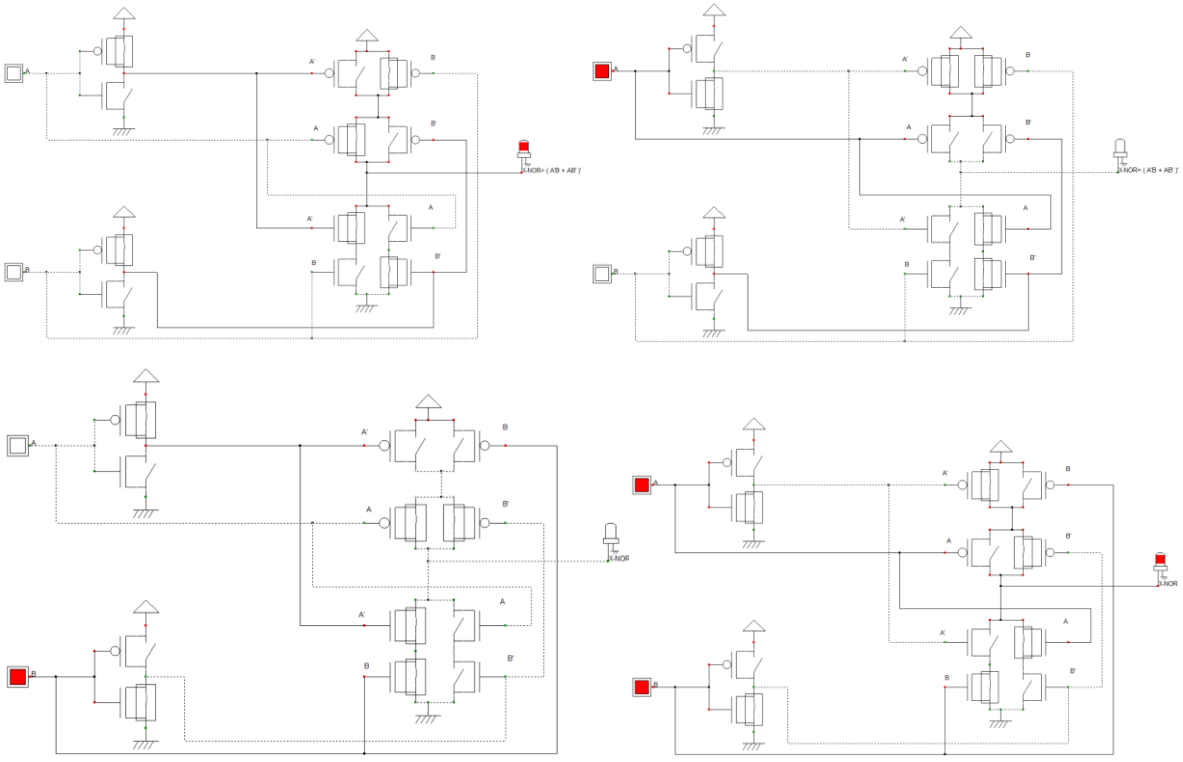
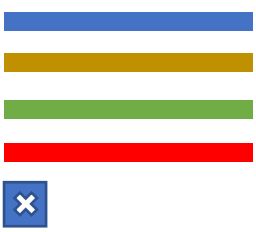


Figure: State transition of XNOR in DSCH

Layout of XOR and XNOR gate in Microwind:

For designing the CMOS XOR and XNOR circuit we used Euler Path Theorem from the below and designed the stick diagram layout below. To design the layout we used following tools from the microwind:



$V_{dd}+/V_{ss}-$

P+ Diffusion

N+ Diffusion

Polysilicon

P+/N+ Contact Metal

Euler Path: for XOR and XNOR circuit is given below.

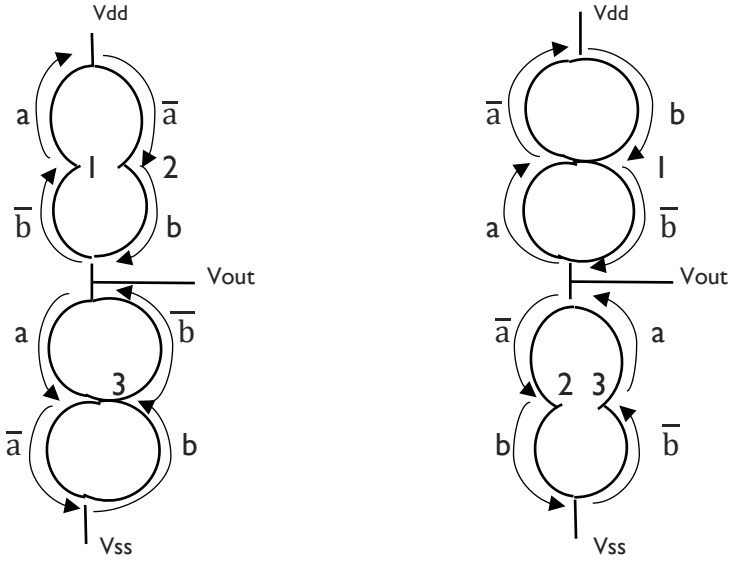


Figure 27: Euler path for XOR and XNOR circuit

The layout of CMOS XOR and XNOR gate look like below:

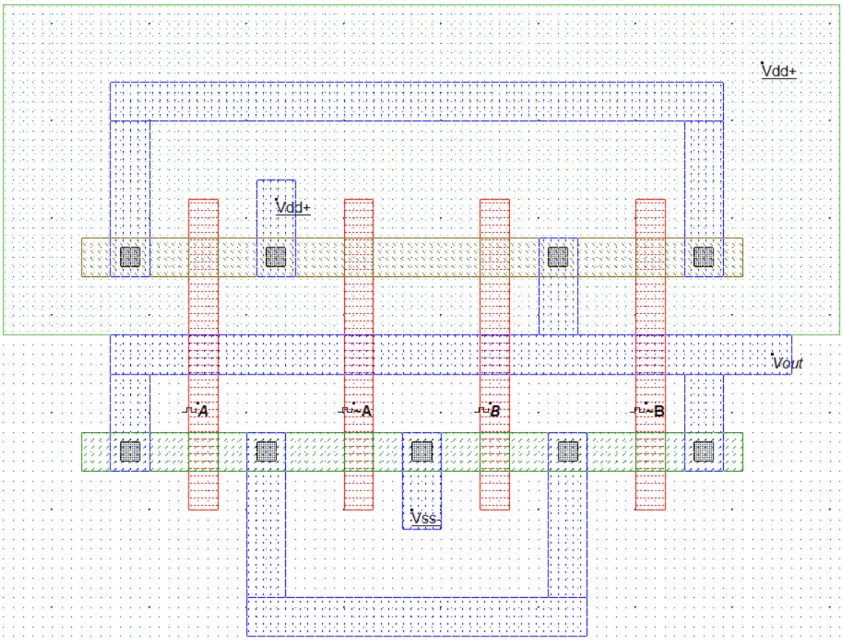


Figure 28 : Layout of CMOS XOR in Microwind

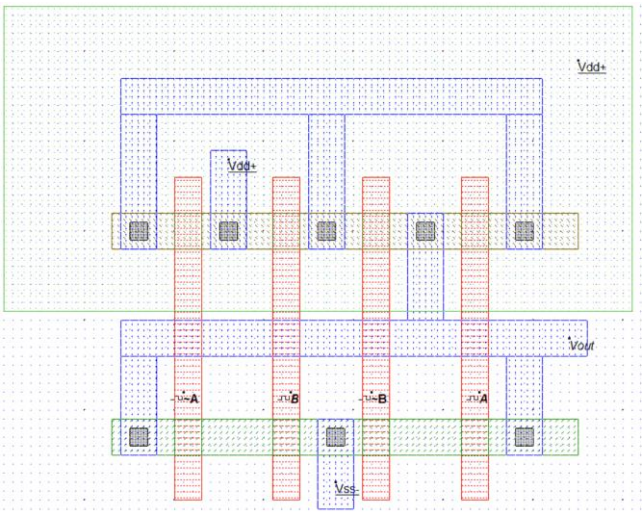


Figure 29 : Layout of CMOS XNOR in Microwind

Result:

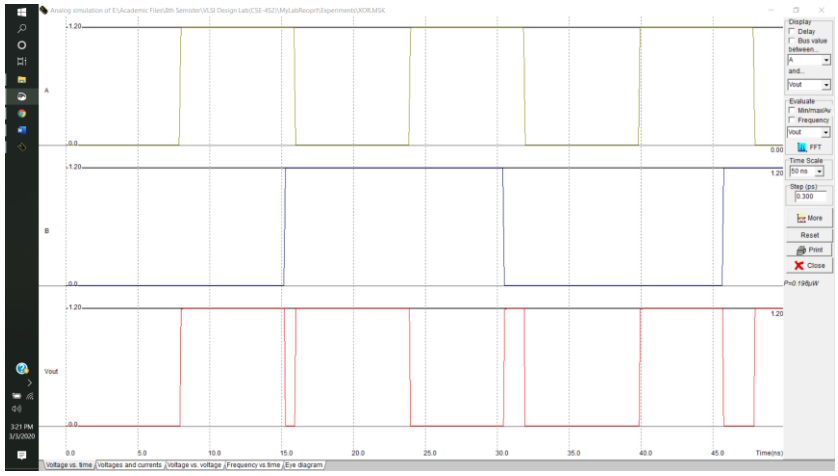


Figure 30 : Simulation of CMOS XOR in Microwind

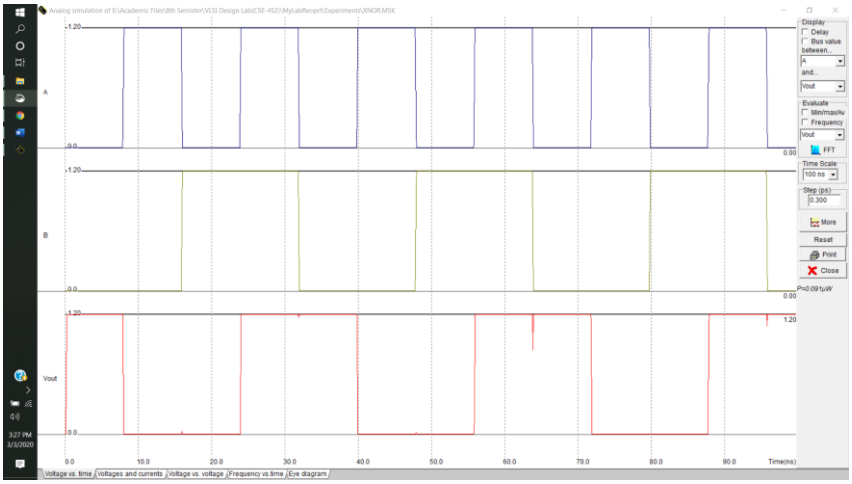


Figure 31 : Simulation of CMOS XOR in Microwind

Discussion: From the above experiment we could completed the stick diagram drawing and its simulation process successfully. We could know about several features of Microwind and learnt how to work with this simulation process.

Experiment No: 05

Experiment Name: Design $(\overline{AB + CD})$using CMOS in DSCH3 simulator and show layout using MICROWIND.

Objectives: The objective of this experiment is to design $(\overline{AB + CD})$ using DSCH3 and draw its layout on Microwind.

Theory: $(\overline{\mathrm{AB} + \mathrm{CD}})$ circuit shown below which is built from transistors and an output. p channel MOS constructs the pull-up section and n channel MOS constructs pull-down sections. There is also four inputs and outputs which are verified with the truth tables.

CMOS circuit for $(\overline{AB + CD})$:

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

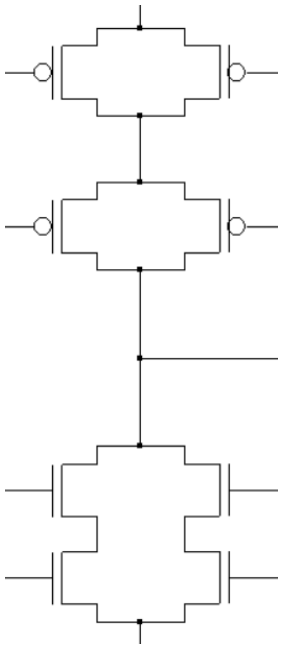


Figure 32: Symbol of Truth Table and CMOS Circuit of $(\overline{AB + CD})$

Working procedure:

- Execute DSCH3.exe.
- Drag and drop three pMOS transistor from the tool bar for drawing pull up circuit.
- Also drag and drop three nMOS transistor from the tool bar for drawing pull down circuit.
- Similarly selecting supply and ground symbols from *Symbol Library* and dragging them to the pull up and pull-down circuit.
- Connecting all symbols as shown in the figure, we can use *add a line* command to connect different nodes of these symbols
- Adding input symbol (A,B,C,D) to the input and LED symbol to the output of the circuit from symbols library to complete the circuit.

Schematic diagram of the CMOS $(\overline{AB+CD})$

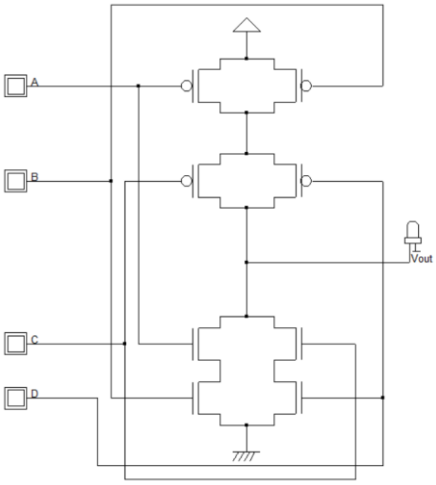


Figure 33: $(\overline{AB+CD})$schematic diagram in DSCH3

The logical simulation of $(\overline{AB+CD})$from the true table:

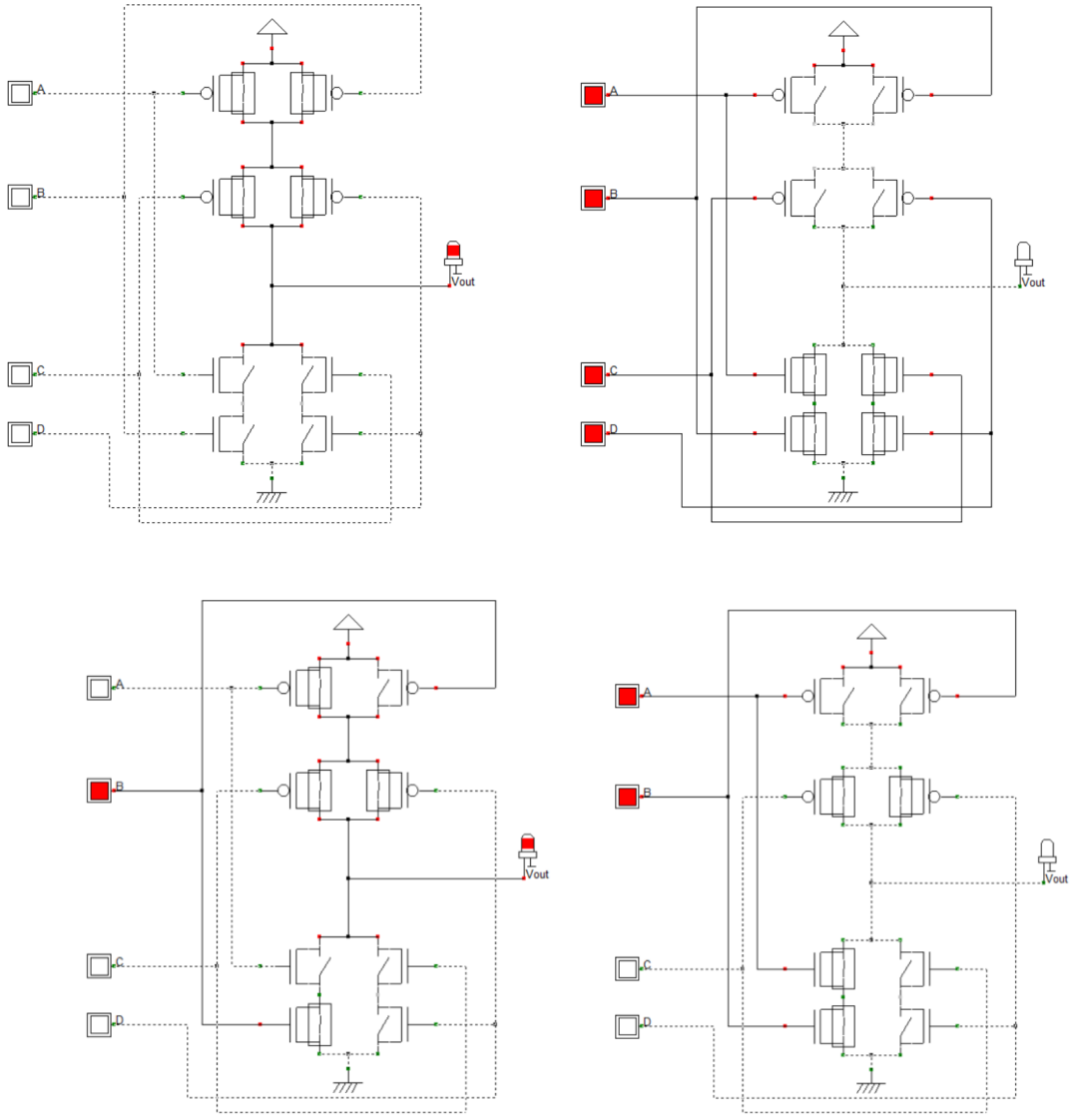


Figure 34: Logical simulation of (AB+C)' in DSCH3

Layout of CMOS $(\overline{AB + CD})$ in Microwind:

For designing the CMOS $(\overline{AB + CD})$ circuit we used Euler Path Theorem from the above and designed the stick diagram layout below. To design the layout we used following tools from the microwind:

$V_{dd}+/V_{ss}-$

P+ Diffusion

N+ Diffusion

Polysilicon

P+/N+ Contact Metal

Euler Path: for $(\overline{AB + CD})$ circuit

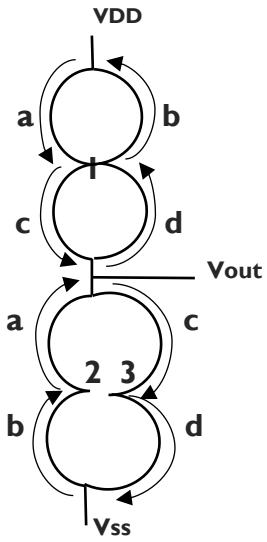


Figure 35: Euler path for $(\overline{AB + CD})$ circuit

The layout of CMOS$(\overline{AB + CD})$ looks like below:

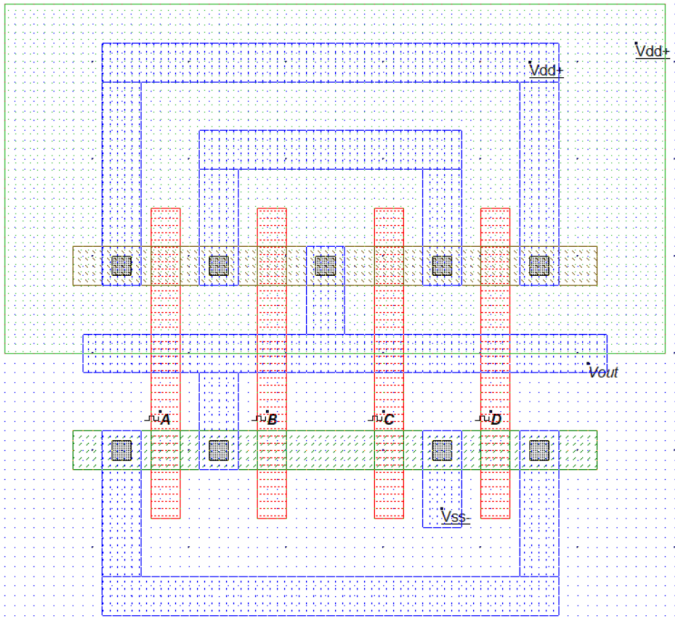


Figure 36 : Layout of CMOS $(\overline{AB + CD})$ in Microwind

Result:

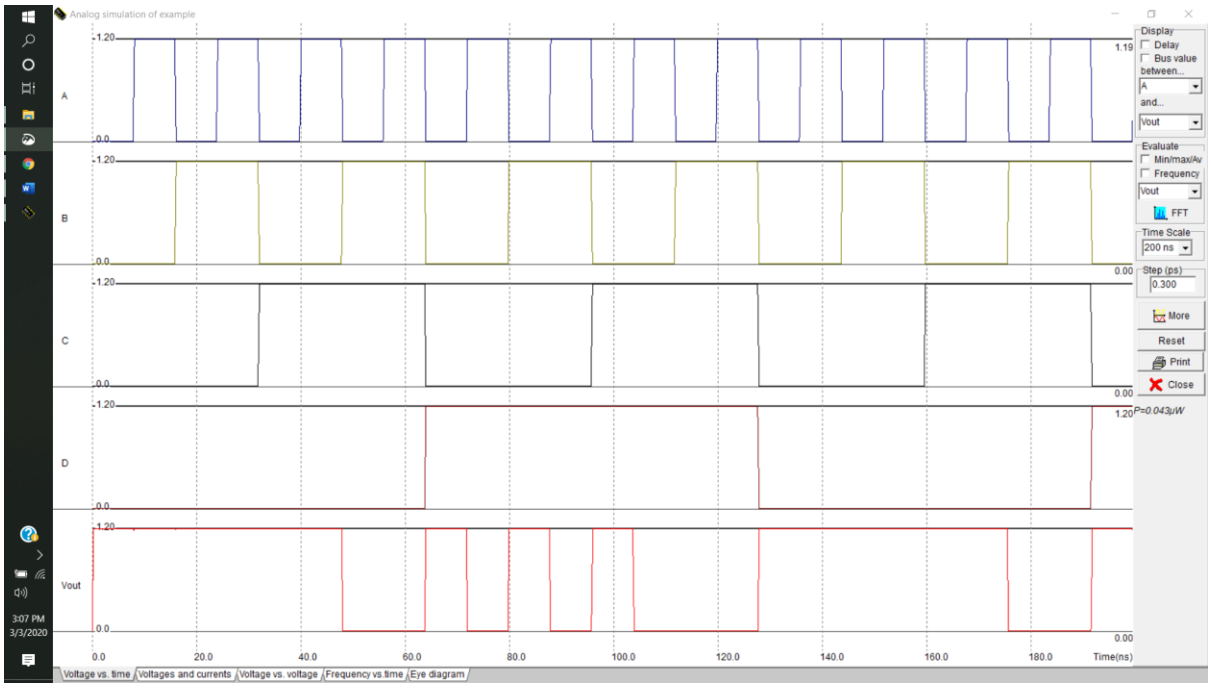


Figure 37: Simulation of CMOS $(\overline{AB + CD})$ circuit in Microwind

Discussion: From the above experiment we could completed the stick diagram drawing and its simulation process successfully. We could know about several features of Microwind and learnt how to work with this simulation process.

Part Two

Problem Solve by C/C++ Programming

Experiment No: 01

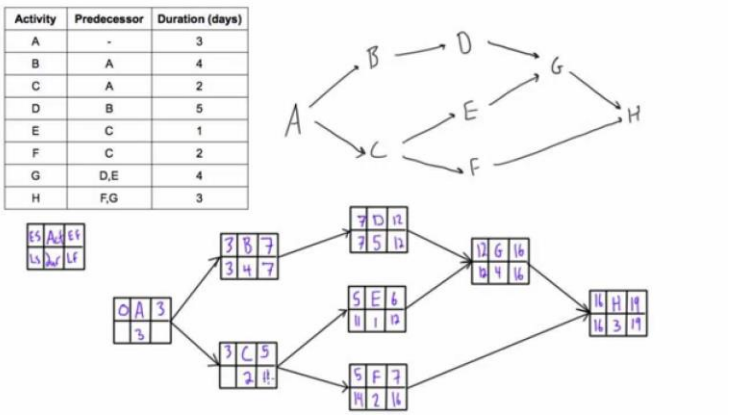
Experiment Name: Write a program to implement the Critical Path Algorithm to find the critical path in a graph.

Objectives: Our main objective is to implement the critical path algorithm to find the critical path in a graph.

Theory:

1. Convert combinational logic circuit into combinational network where each gate presents a node and connection/wire presents edge with delay.
2. The algorithm starts with finding earliest possible start time for each node going through the network.
   - from input to output.
   - from left to right
3. The earliest possible finish time for each node is found by going backward through the network output to input. Left to right.
4. Nodes which have equal earliest possible start time and finish time are on critical path.

Activity	Predecessor	Duration (days)
A	-	3
B	A	4
C	A	2
D	B	5
E	C	1
F	C	2
G	D,E	4
H	F,G	3



Result:

Task ID	Predecessors	Duration
A(start)	_	0
B	A	10
C	A	20
D	B,C	30
E	B,C	20
F	E	40
G	D,F	20
H(Finish)	G	0

Discussion: From this program we could implement critical path algorithm by using C++ programming languages.

Experiment No: 02

Experiment Name: Write a program to implement the Left Edge Algorithm on channel routing problem.

Objectives: Our aim is to write such a program that implement the Left Edge Algorithm on channel routing problem.

Theory:

- **Step 1:** Build the Vertical Constraint Graph (VCG) for the input channel routing problem
- **Step 2:** Place horizontal segments (choose nets)
  - that do not have ancestors in the VCG and
  - whose horizontal segments do not overlap) and update the VCG
- **Step 3:** Repeat Step 2 until all the horizontal segments have been placed

Result:

Input:

N: maximum number of pins

List of top net list, top-list {…….}

List of bottom net list, bottom-list {…….}

Example:

11

O B D E B F G O D O O

A C E C E A F H O H G

Output for example 2

A J

D

E G

C F I

B H

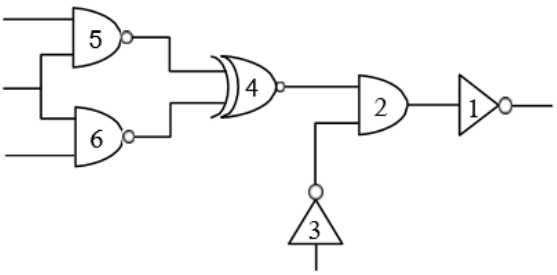
Discussion: From this problem we could able to write such a program that implement the Left Edge Algorithm on channel routing problem.

Experiment No: 03

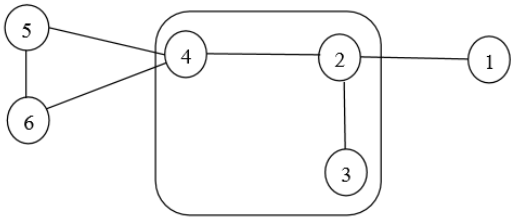
Experiment Name: Write a program to implement the Kernighan-Lin Algorithm on graph partitioning.

Objectives: Our main objective is to write a program to implement the Kernighan-Lin Algorithm on graph partitioning.

Theory:



(a) A circuit to be partitioned



(b) Its corresponding graph

Step 1: Initialization.

Let the initial partition be a random division of vertices into the partition

A= {2,3,4} and B= {1,5,6}.

A'= A = {2,3,4} and B' = B = {1,5,6}.

Step 2: Compute D − values.

$D1 = E1 - I1 = 1 - 0 = +1$

$D2 = E2 - I2 = 1 - 2 = -1$

$D3 = E3 - I3 = 0 - 1 = -1$

$D4 = E4 - I4 = 2 - 1 = +1$

$D5 = E5 - I5 = 1 - 1 = +0$

$D6 = E6 - I6 = 1 - 1 = +0$

Step 3:

Compute *gains.*

g21 = D2 + D1 − 2c21 = (−1) + (+1) − 2(1) = −2

g25 = D2 + D5 − 2c25 = (−1) + (+0) − 2(0) = −1

g26 = D2 + D6 −2c26 = (−1) + (+0) −2(0) = −1

g31 = D3 + D1 − 2c31 = (−1) + (+1) − 2(0) = +0

g35 = D3 + D5 − 2c35 = (−1) + (+0) − 2(0) = −1

g36 = D3 + D6 −2c36 = (−1) + (+0) − 2(0) = −1

g41 = D4 + D1 − 2c41 = (+1) + (+1) − 2(0) = +2

g45 = D4 + D5 − 2c45 = (+1) + (+0) − 2(1) = −1

g46 = D4 + D6 − 2c46 = (+1) + (+0) − 2(1) = −1

The largest g value is g41. (a1, b1) is (4, 1), the gain

g41 = g1 = 2, and

$A' = A' - \{4\} = \{2,3\}$, $B' = B' - \{1\} = \{5, 6\}$.

Both A' and B' are not empty; then we update the D values in the next step and repeat the procedure from Step 3.

Step 4: Update D−values of nodes connected to (4,1).

The vertices connected to (4,1) are vertex (2) in set A' and vertices (5,6) in set B'. The new D−values for vertices of A' and B' are given by

$D_2' = D2 + 2c24 - 2c21 = -1 + 2(1 - 1) = -1$

$D_5' = D5 + 2c51 - 2c54 = +0 + 2(0 - 1) = -2$

$D_6' = D6 + 2c61 - 2c64 = +0 + 2(0 - 1) = -2$

To repeat Step 3, we assign $D_i = D_i'$ and then recompute the gains:

g25 = D2 + D5 − 2c25 = (−1) + (−2) − 2(0) = −3

g26 = D2 + D6 − 2c26 = (−1) + (−2) − 2(0) = −3

g35 = D3 + D5 −2c35 = (−1) + (−2) − 2(0) = −3

g36 = D3 + D6 − 2c36 = (−1) + (−2) − 2(0) = −3

All the g values are equal, so we arbitrarily choose g36, and hence the pair (a2, b2) is (3, 6),

g36 = g2 = −3,

$A' = A' - \{3\} = \{2\}$,

$B' = B' - \{6\} = \{5\}$.

The new D−values are:

$D_2' = D2 + 2c23 - 2c26 = -1 + 2(1 - 0) = 1$

$D_5' = D5 + 2c56 - 2c53 = -2 + 2(1 - 0) = 0$

The corresponding new gain is:

g25 = D2 + D5 − 2c52 = (+1) + (0) − 2(0) = +1

Therefore, the last pair (a3, b3) is (2,5) and the corresponding gain is g25 = g3 = +1.

Step 5: Determine k.

We see that g1 = +2, g1 + g2 = −1, and

g1 + g2 + g3 =0.

The value of k that results in maximum G is 1.

Therefore, X = {a1} = {4} and Y = {b1} = {1}.

The new partition that results from moving X to B and Y to A is, A = {1, 2, 3} and

B = {4,5,6}.

The entire procedure is repeated again with this new partition as the initial partition.

Verify that the second iteration of the algorithm is also the last, and that the best solution obtained is *A* = {1, 2, 3} and B = {4, 5, 6}.

Result:

Discussion: From this problem we could able to implement the Kernighan-Lin Algorithm on graph partitioning.