

Experiment No: 01

Experiment Name: Introduction to DSCH3 and Microwind Software.

Objective: The objective is to be introduced and learn details about how to design circuits with the software DSCH3 and Microwind.

Theory:

DSCH3:

The DSCH3 application is a simulator and logic editor. Before beginning microelectronics design, DSCH3 is utilized to validate the architecture of the logic circuit. DSCH3 provides a user-friendly environment for the hierarchical logic design and rapid simulation with delay analysis, enabling the design and validation of complicated logic structures. In the guidebook, several strategies for low-power design are discussed. DSCH3 additionally has 8051 and 18f64 symbols, models, and assembly support. DSCH3 additionally offers a SPICE interface. The picture below depicts the DSCH3 User Interface.

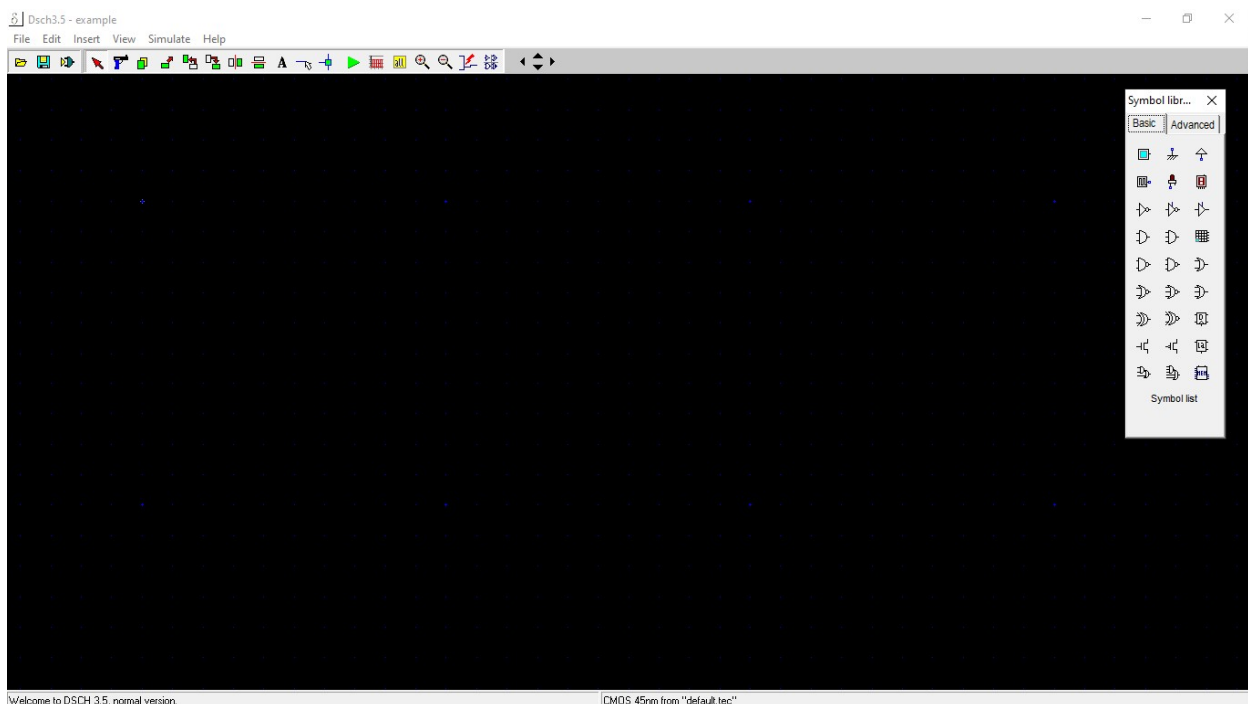


Fig: DSCH3 User Interface

Key features of DSCH3 typically include:

- User-friendly environment for rapid design of logic circuits.
- Supports hierarchical logic design.
- Added a tool on fault analysis at the gate level of digital. Faults: Stuck-at-1, stuck-at-0. The technique allows the injection of a single stuck-at-fault at the circuit nodes.
- Improved interface between DSCH and Winspice.

- Handles both conventional pattern-based logic simulation and intuitive on-screen mouse-driven simulation.
- Built-in extractor which generates a SPICE netlist from the schematic diagram (Compatible with PSPICETM and WinSpiceTM).
- Generates a VERILOG description of the schematic for layout conversion.
- Immediate access to symbol properties (Delay, fanout).
- Model and assembly support for 8051 and PIC 16F84 microcontrollers.
- Sub-micron, deep-submicron, and nanoscale technology support.
- Supported by a vast symbol library

Microwind:

Microwind is a software tool primarily used for the design and simulation of digital and analog integrated circuits at the transistor level. It provides a platform for designing and analyzing various aspects of integrated circuits, including logic gates, analog circuits, memory cells, and more. The tool is commonly used in educational settings and research environments for teaching and prototyping purposes.

Key features of Microwind typically include:

- Microwind allows users to design and simulate digital circuits, including logic gates, flip-flops, and other digital components.
- The software supports the design and simulation of analog circuits, such as amplifiers, filters, and other analog components.
- Microwind is often used for Complementary Metal-Oxide-Semiconductor (CMOS) technology, which is widely used in the fabrication of integrated circuits.
- The tool often includes features for generating physical layouts of circuits based on the designed schematics.
- Microwind provides simulation capabilities to analyze the performance of the designed circuits under different conditions.
- Microwind 3.1 editor: We may edit boxes using a palette of layers. We may cut, paste, duplicate, generate a layout matrix, and use the layout editor to insert contacts, MOS devices, pads, complex contacts, and paths in one click.
- Microwind 3.1 2D viewer: Use the 2D viewer to see the cross-section of the fabricated integrated circuits.
- Microwind3.1 3D viewer: Use this 3D viewer to see the step-by-step fabrication of any portion of layout.
- Interactive 3D views: Impressive animated 3D views of the layout are now available in Microwind3.1 through the new command "3D view of the IC".

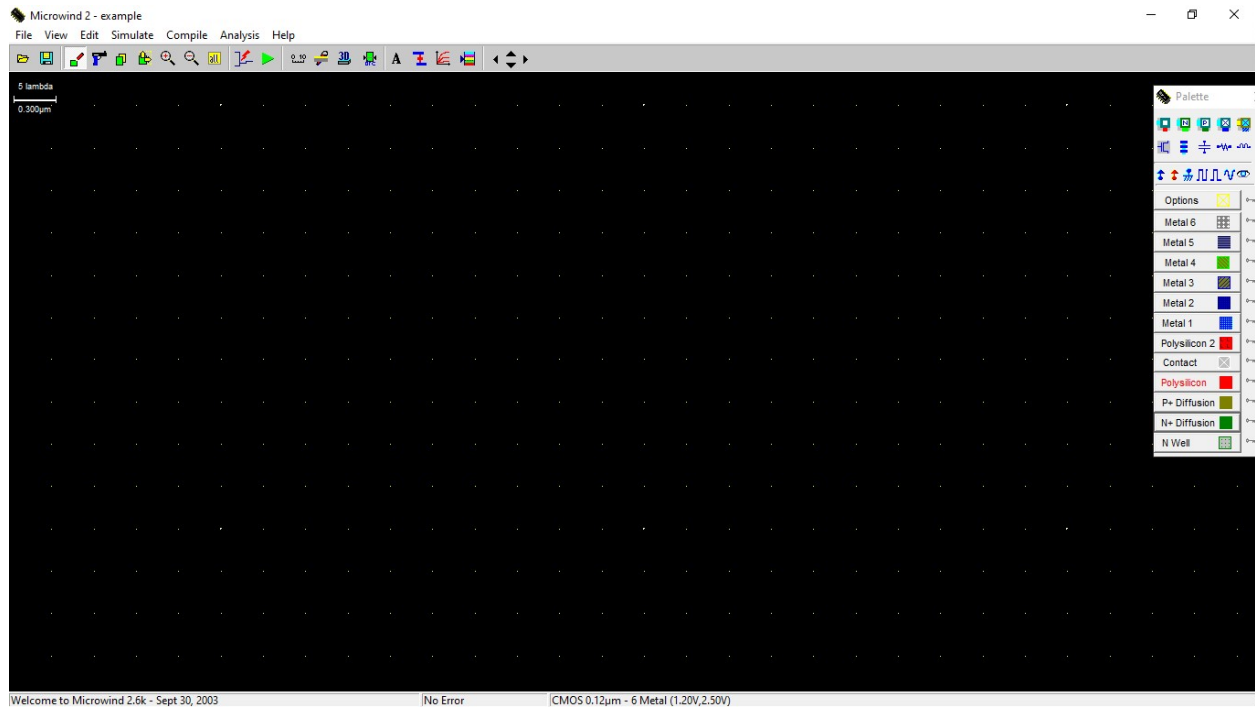


Figure: Microwind user interface.

Discussion: We have been introduced to and learned details from DSCH3 and Microwind that is very helpful for further design.

Experiment No: 02

Experiment Name: Implementation of schematic diagram and layout design of NOT, NAND, and NOR gate.

Objective: To learn to draw a schematic diagram and layout design of NOT, NAND, and NOR gates using DSCH3 and Microwind.

Theory: NOT, NAND and NOR gates are fundamental building blocks in digital logic design, and each serves a specific purpose in processing binary information. The purpose of this lab is to talk about Digital Logic Circuits. By the end of this lab, you will understand the functions and operations of different logic gates.

NOT Gate:

The NOT gate, also known as an inverter, is designed to produce the logical negation of its input. The output of a NOT gate is the opposite (complement) of its input. If the input is high, the output is low, and vice versa.

Symbol, Truth Table, and CMOS circuit of NOT Gate

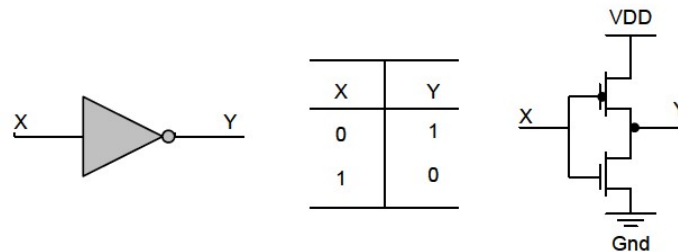


Figure: Symbol, Truth Table, and CMOS circuit of NOT Gate.

NAND Gate:

The NAND gate is designed to perform the logical operation of the negation of the AND operation. The output of a NAND gate is high (1) only when at least one of its inputs is low (0). In other words, it produces a low output only when all of its inputs are high. The output is the inverse of the AND gate. Here, $OUT = (A.B)'$.

Symbol, Truth Table, and CMOS circuit of NAND Gate

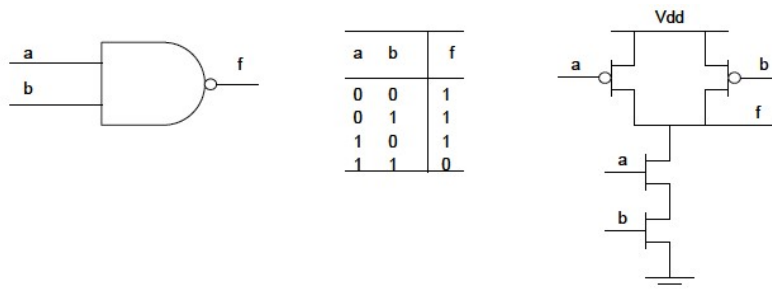


Figure: Symbol, Truth Table and CMOS circuit of NAND Gate.

NOR Gate

The NOR gate is designed to perform the logical operation of the negation of the OR operation. The output of a NOR gate is low (0) only when at least one of its inputs is high (1). It produces a high output only when all of its inputs are low. The output is the inverse of the OR gate. Here, $OUT = (A + B)'$.

Symbol, Truth Table and CMOS circuit of NOR Gate

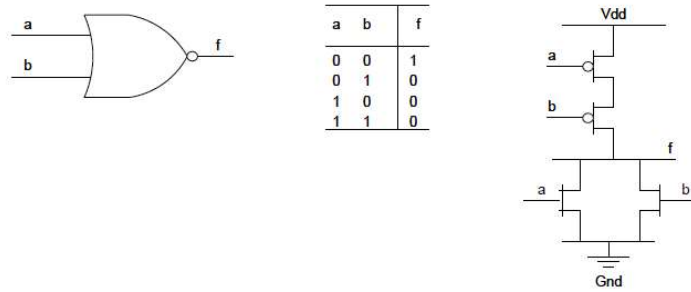


Figure: Symbol, Truth Table and CMOS circuit of NOR Gate.

Layout design of NOT, NAND, and NOR gate

Procedure:

1. At first we took p and n diffusion.
2. Then add corresponding connectors to the diffusion.
3. Then we added poly from p type diffusion to n-type diffusion.
4. We added metal to connect them.
5. Vdd+ and Vss- are added for power supply.
6. Drawing the rails of and ground rails above and below Vdd+
7. Connecting the nWell to Vdd
8. Checking the design using DRC for any design rule violation and correcting the design in case of error, again running the DRC and checking for errors. Or running the DRC after each change in the layout.
9. Checking for Electrical connections to be valid.
10. Adding inputs and outputs to the design; also adding virtual capacitance at the output in the design.

NOT Gate:

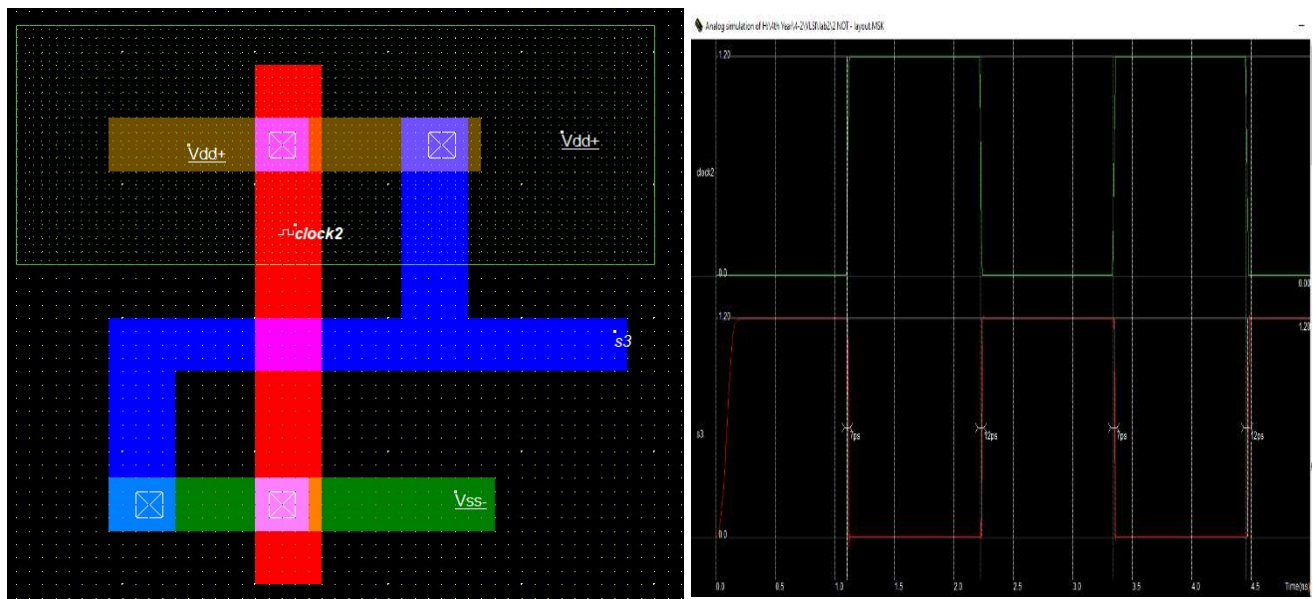


Figure: Not Gate Layout Design & Output

NAND Gate:

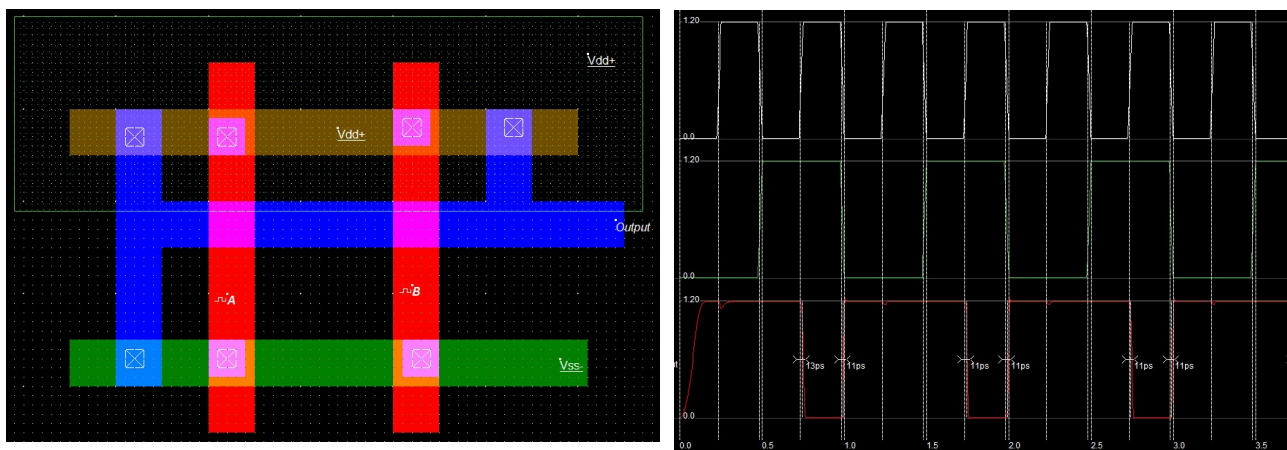


Figure: NAND Gate layout design & output

NOR Gate:

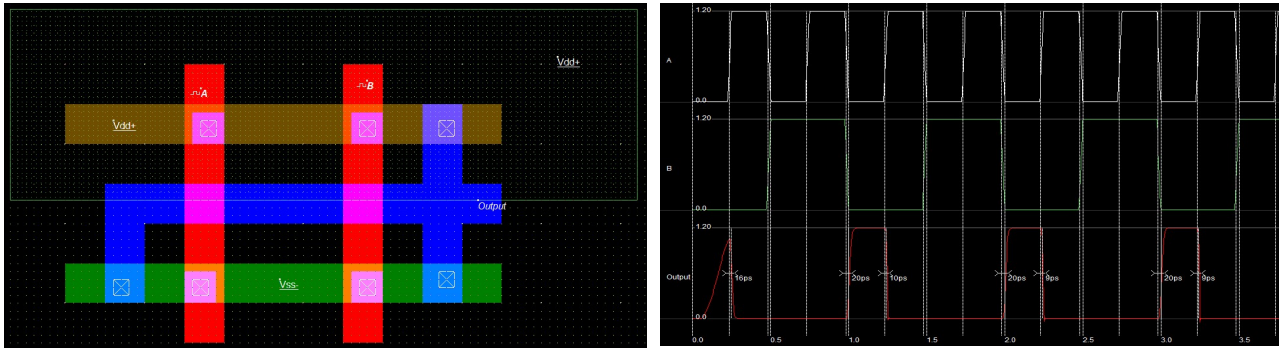


Figure: NAND Gate layout design & output

Discussion: We have successfully implemented and learned details about the schematic and layout diagram of the NOT, NAND, and NOR gates using DSCH3 and Microwind.

Experiment No: 03

Experiment Name: Draw the circuit design and layout design for the expression $(A.B+C)'$.

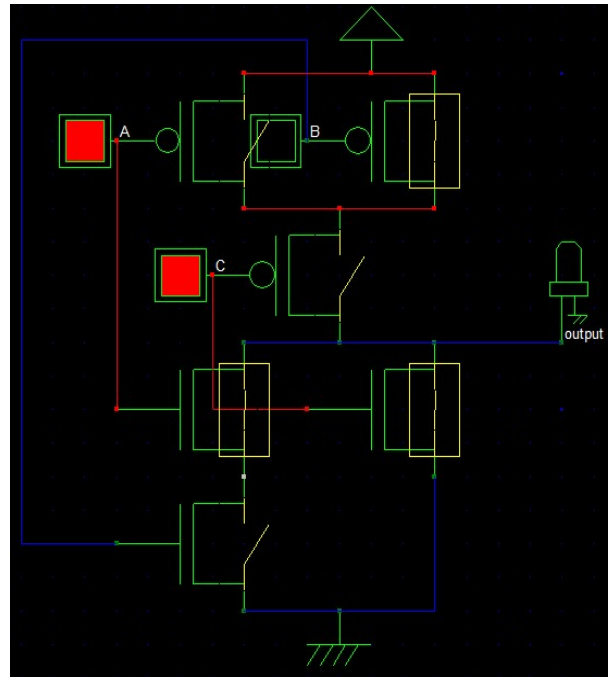
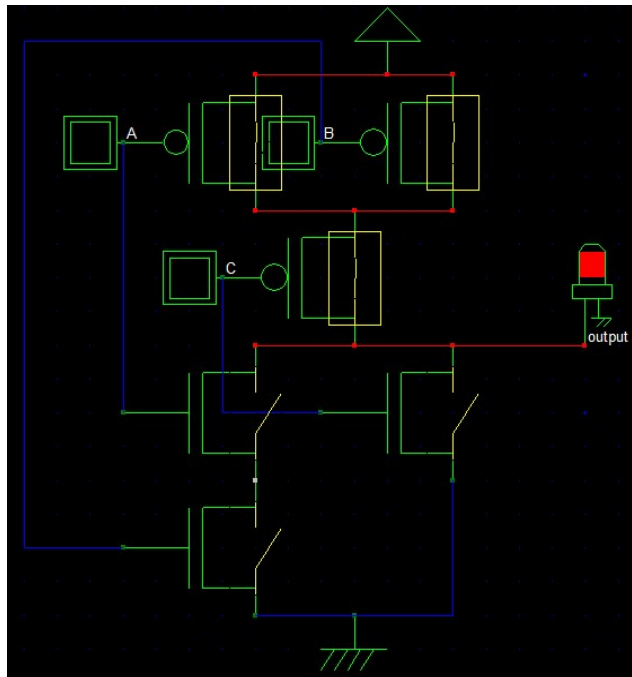
Objective: The objective is to learn draw a schematic diagram and layout design of $(A.B+C)'$ using DSCH3 and Microwind.

Theory: To design $(AB+C)'$, A and B are connected in parallel for the pull-up circuit, and C is associated with the endpoint of B and C in series. In the case of a pull-down circuit, A and B are connected in series, and C is associated with them in parallel. The drain of the pull-down network is connected to the drain of the pull-up network. Vdd is connected with the pull-up, and Vss is connected with the pull-down network. Output is from the between the pull-up and the pull-down network. The truth table for the expression is:

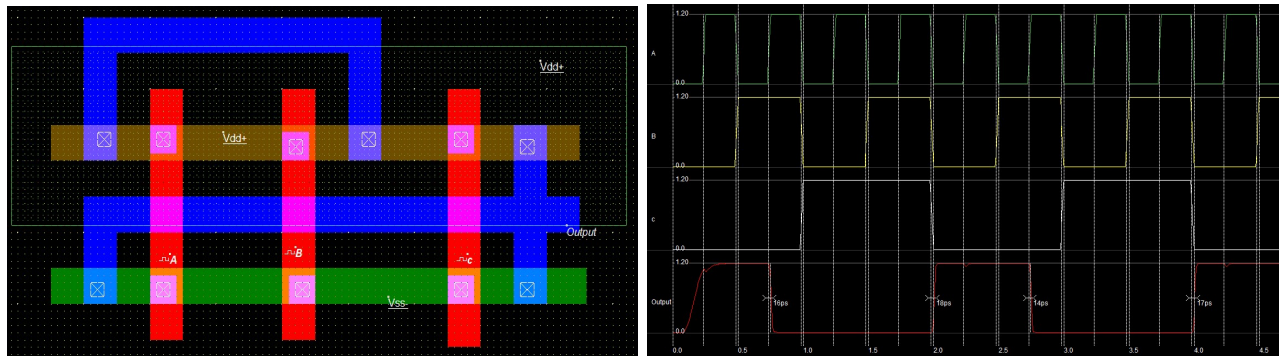
A	B	C	$A.B+C$	$(A.B+C)'$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

Circuit Diagram:

Schematic Diagram



Layout Design



Discussion: We have successfully shown the output of schematic diagram and layout design for the experiment.

Experiment No: 04

Experiment Name: Draw the circuit design and layout design for the expression $(AB+C+DE)'$.

Objective: The objective to learn and draw the schematic diagram and the layout design of $(AB+C+DE)'$ using DSCH3 and Microwind software.

Theory: To create a truth table for the expression $(AB+C+DE)'$, we need to consider all possible combinations of truth values for the variables A, B, C, D, and E. The expression is complemented, so we'll determine the output for each combination when the expression is false (0). The truth table for the expression is:

A	B	C	D	E	$(AB+C+DE)'$
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	0

Circuit Diagram:

Schematic Diagram

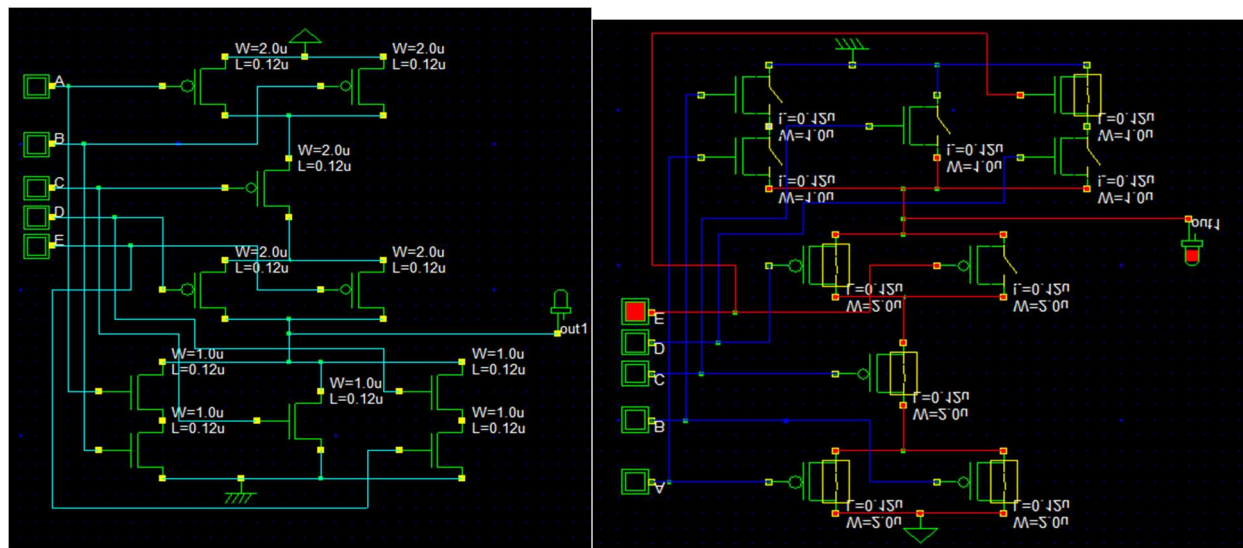


Figure: Circuit Diagram and its input-output sample

Layout Design

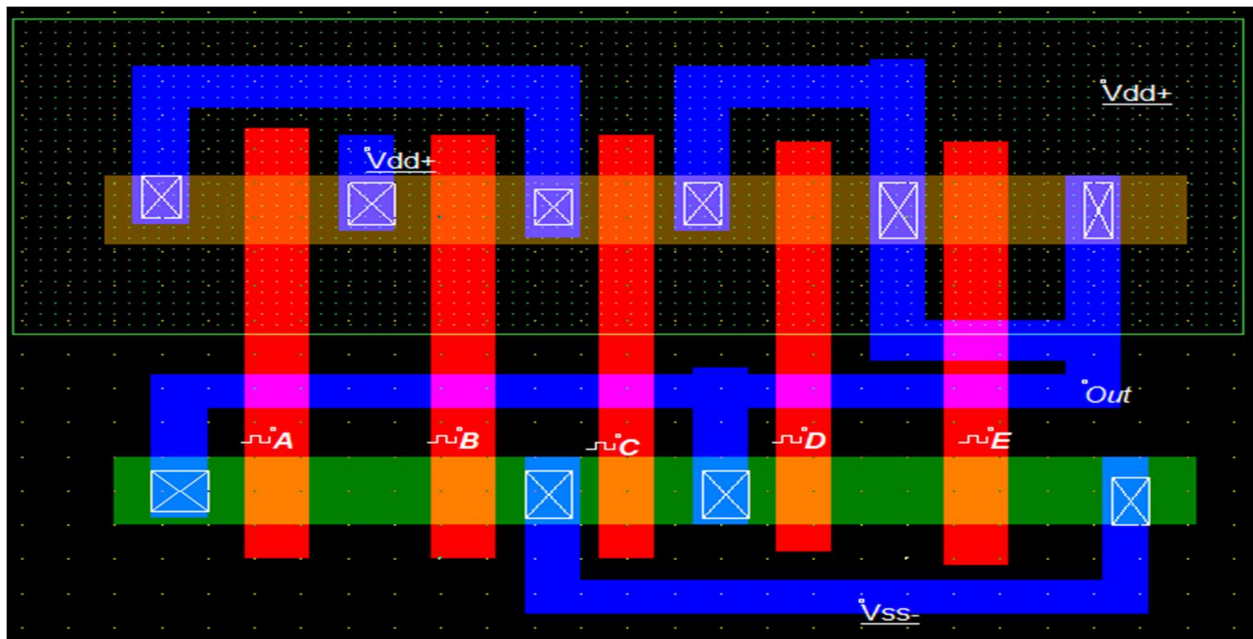


Figure: Layout Design for the expression

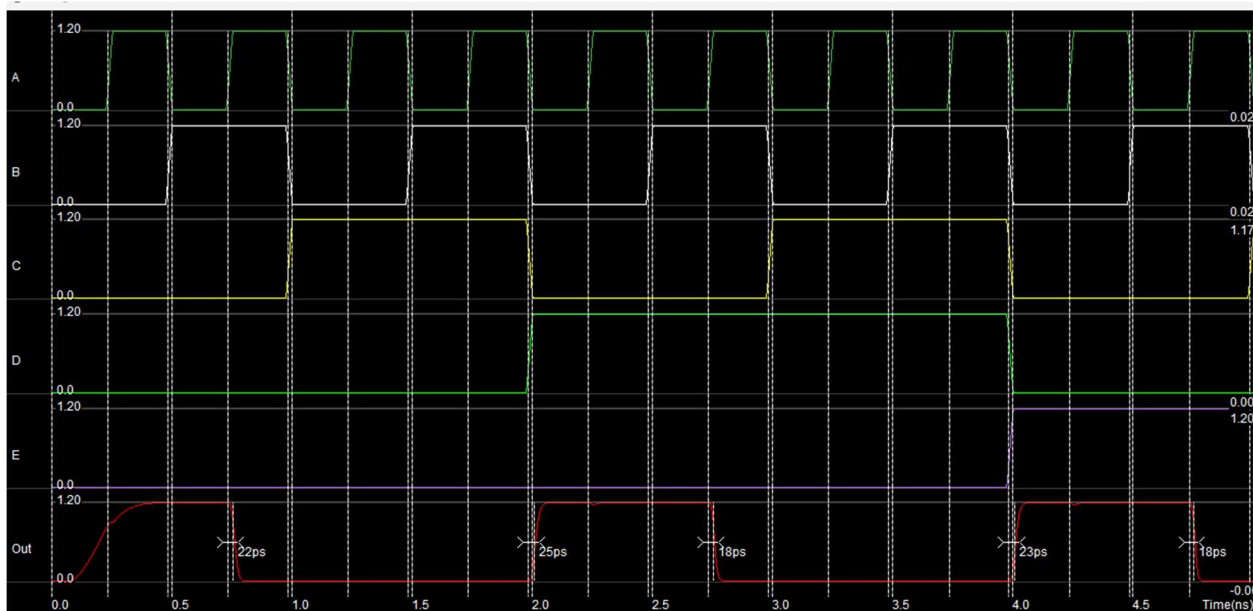


Figure: Output of the Layout Design

Discussion: We have successfully drawn the circuit and show the proper output properly. Then we successfully drew the layout for the experiment.

Experiment No: 05

Experiment Name: Implement the circuit design and layout design for XOR and XNOR gates.

Objective: The objective is to learn and draw the schematic diagram and the layout design of XOR and XNOR using DSCH3 and Microwind software.

Theory: XOR gate is a digital logic gate that gives an accurate (1 or HIGH) output when the number of true inputs is odd. An XOR gate implements an exclusive or from mathematical logic; that is, true output results if one, and only one, of the inputs to the gate is true. If both inputs are false (0/LOW) or both are true, a false output results. XOR represents the inequality function, i.e., the output is true if the inputs are dissimilar; otherwise, the output is false. And the opposite for the XNOR circuit.

The algebraic expressions $A \oplus B$ or $A.B' + A'.B$, all represent the XOR gate with inputs A and B . The behavior of XOR is summarized in the truth table shown below.

A	B	XOR	XNOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Symbol

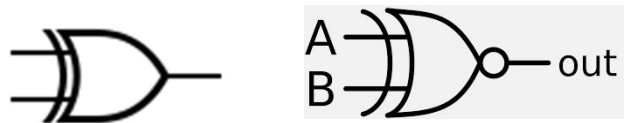


Fig.: Symbols of XOR & XNOR and their truth table.

Circuit Diagram:

Schematic Diagram

XOR:

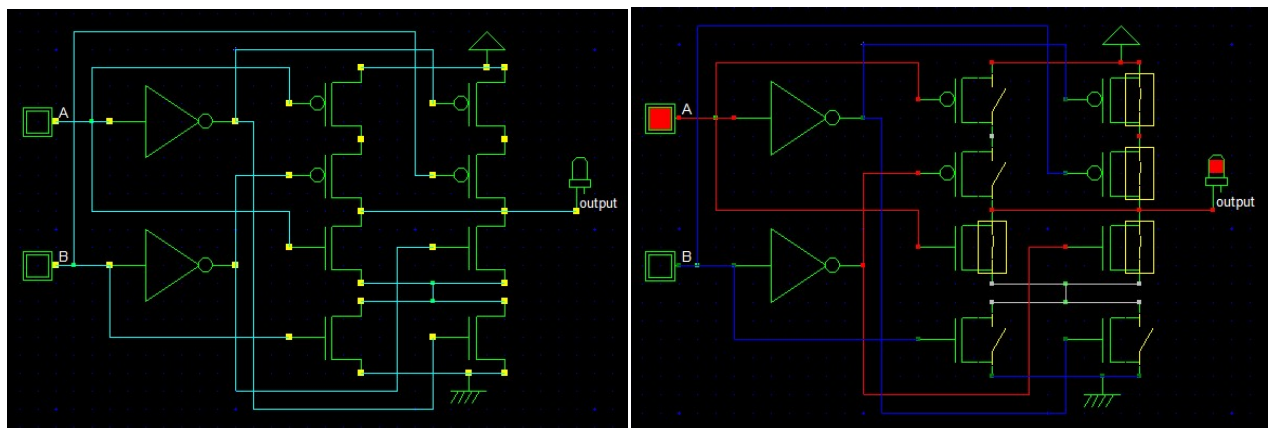


Fig: Circuit Diagram and its input-output sample

XNOR:

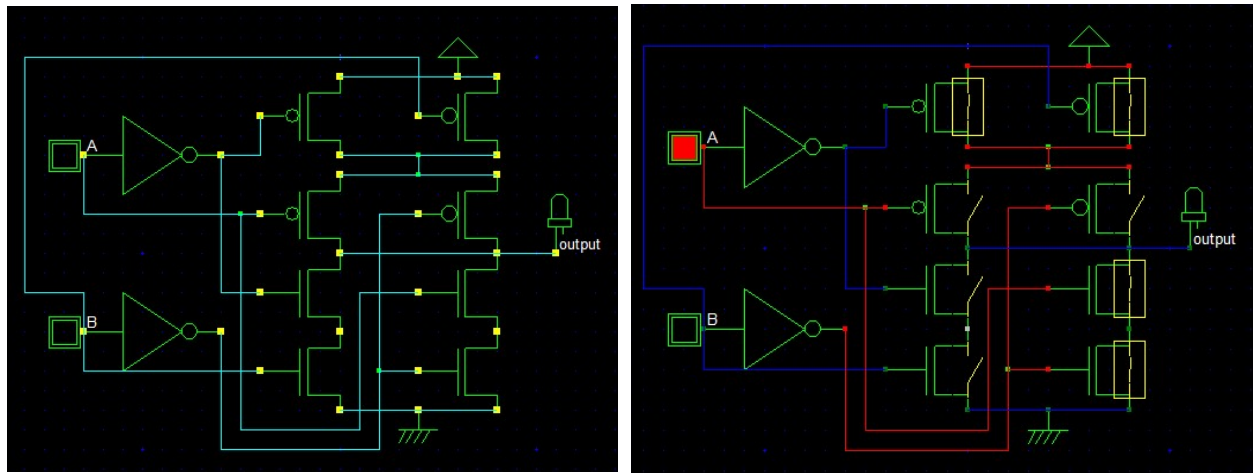
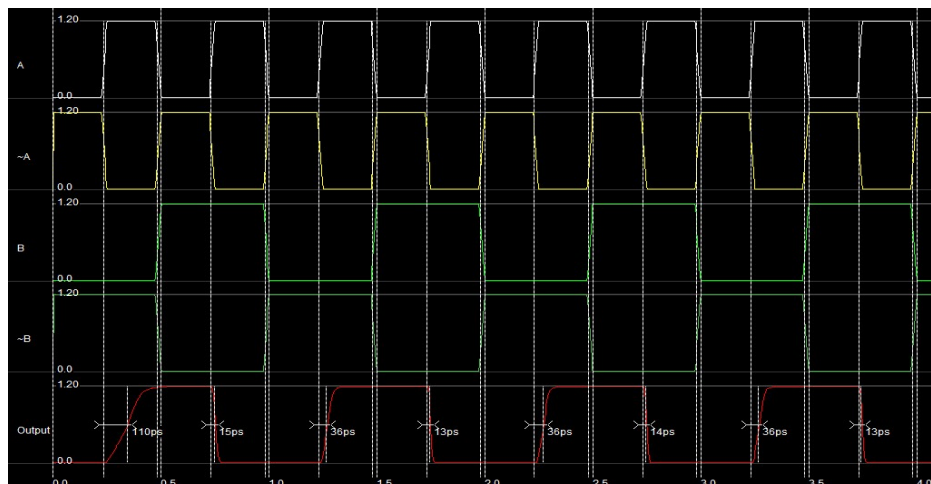
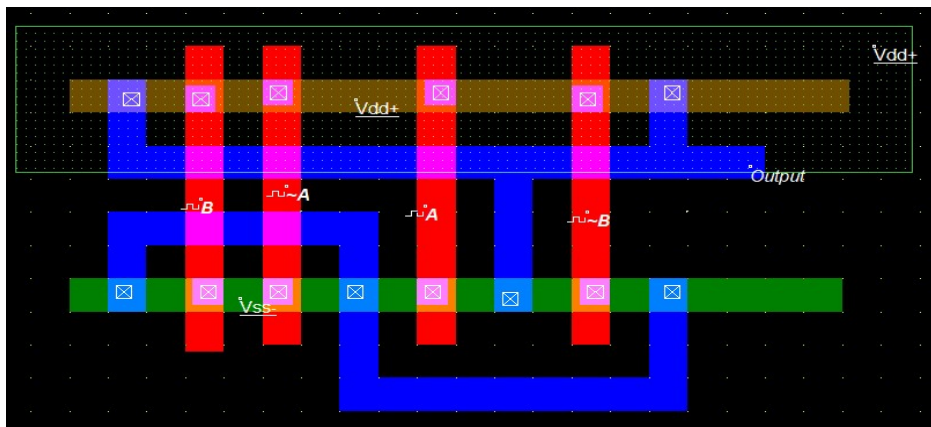


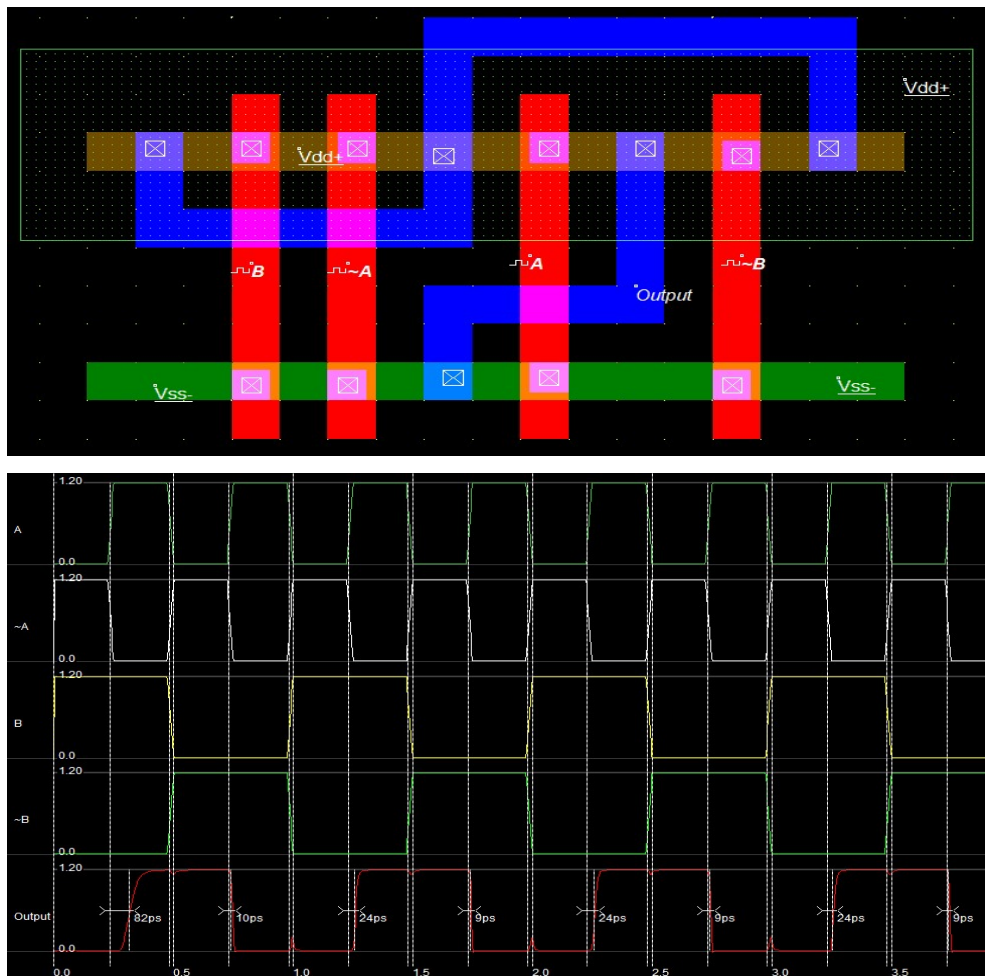
Fig: Circuit Diagram and its input output sample

Layout Design

XOR:



XNOR:



Discussion: We have successfully completed this experiment and show the output of the circuit and their layout.

Experiment No: 06

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

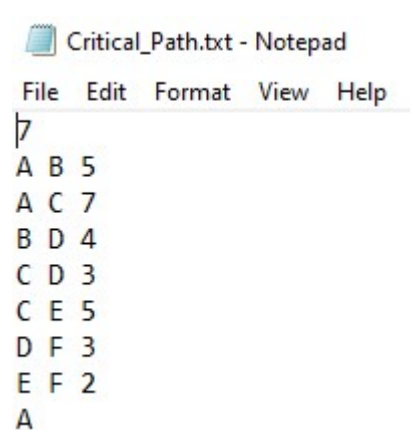
Objective: The objective is to learn about critical path algorithms and find the critical path in a graph.

Theory:

Critical path algorithm

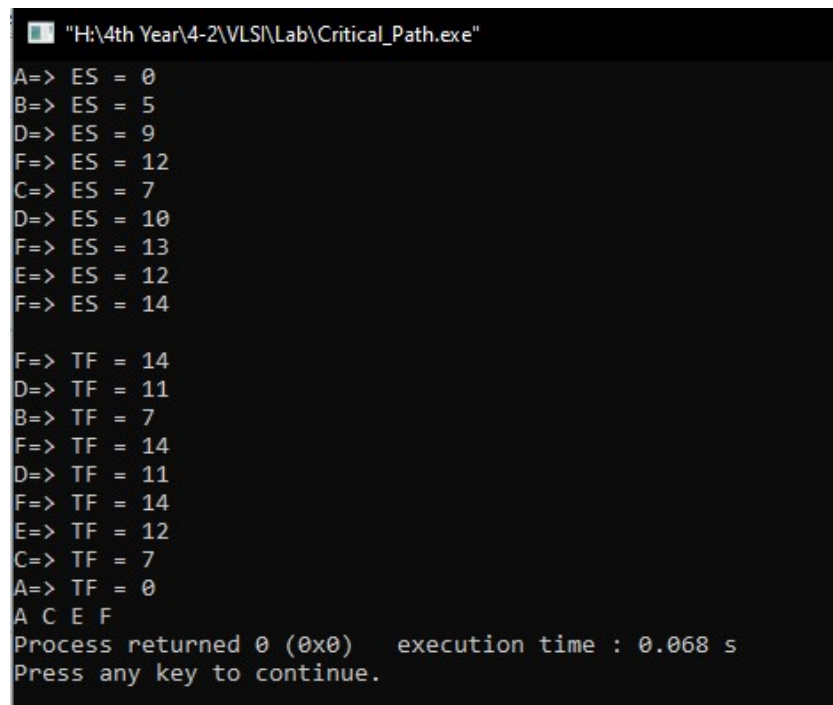
- The algorithm starts with finding the node's earliest possible start time, going through the network.
- Then, the latest possible start time for each node is found by going back through the network.
- Nodes with an equal earliest possible start time and the latest possible start time are on the critical path.

Input:



```
File Edit Format View Help
7
A B 5
A C 7
B D 4
C D 3
C E 5
D F 3
E F 2
A
```

Output:



```
"H:\4th Year\4-2\VLSI\Lab\Critical_Path.exe"
A=> ES = 0
B=> ES = 5
D=> ES = 9
F=> ES = 12
C=> ES = 7
D=> ES = 10
F=> ES = 13
E=> ES = 12
F=> ES = 14

F=> TF = 14
D=> TF = 11
B=> TF = 7
F=> TF = 14
D=> TF = 11
F=> TF = 14
E=> TF = 12
C=> TF = 7
A=> TF = 0
A C E F
Process returned 0 (0x0)   execution time : 0.068 s
Press any key to continue.
```

Discussion: We have successfully implemented a critical path algorithm to find a critical path in a graph and then shown the output successfully.

Experiment No: 07

Experiment Name: Write a program to implement the Left Edge Channel Routing Algorithm.

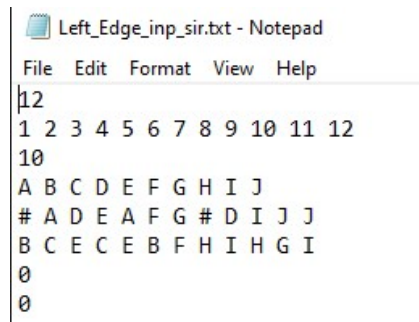
Objective: The objective is to learn about the left-edge channel routing algorithm.

Theory: The left edge and dogleg algorithm comes under detailed routing, determining the exact route and layers for each net.

Algorithm

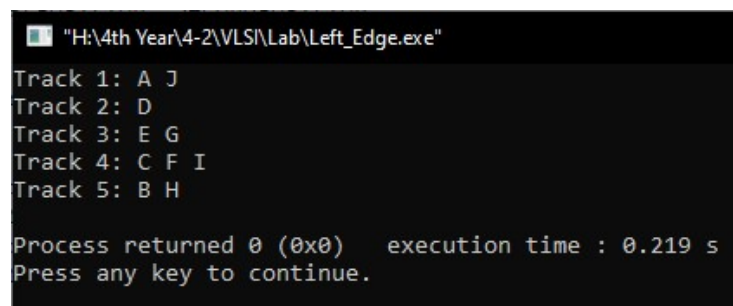
1. The algorithm sweeps the channel from left to right.
2. If the pin is the first pin on a net, that net is assigned its lone horizontal wire segment immediately.
3. The track assignment is the bottommost empty track is assigned to the net.
4. When the last pin on a net is encountered, the net's track is marked as empty, and it can be reused by another net farther to the right. The vertical wire segments that connect the pins to the horizontal segment and the necessary vias can be added separately after the assignment of horizontal segments is complete.

Input:



```
Left_Edge_inp_sir.txt - Notepad
File Edit Format View Help
12
1 2 3 4 5 6 7 8 9 10 11 12
10
A B C D E F G H I J
# A D E A F G # D I J J
B C E C E B F H I H G I
0
0
```

Output:



```
"H:\4th Year\4-2\VLSI\Lab\Left_Edge.exe"
Track 1: A J
Track 2: D
Track 3: E G
Track 4: C F I
Track 5: B H

Process returned 0 (0x0)   execution time : 0.219 s
Press any key to continue.
```

Discussion: From this problem, we could write such a program that implements the Left Edge Algorithm on the channel the routing problem and shown the output.

Experiment No: 08

Experiment Name: Write a program to implement the Kernighan-Lin (KL) Algorithm.

Objective: The objective is to learn and implement the Kernighan-Lin (KL) algorithm.

Theory: The Kernighan–Lin algorithm is a heuristic algorithm for finding partitions of graphs. The algorithm has significant practical application in the layout of digital circuits and components in the electronic design automation of VLSI.

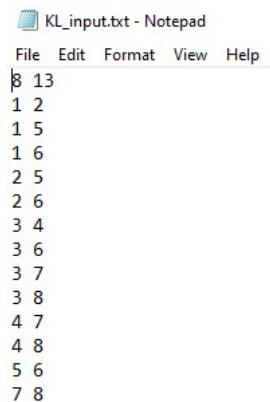
The input to the algorithm is an undirected graph $G = (V, E)$ with vertex set V , edge set E , and numerical weights on the edges in E . The algorithm aims to partition V into two disjoint subsets, A and B , of equal (or nearly equal) size to minimize the sum T of the weights of the subset of edges that cross from A to B . If the graph is unweighted, then instead, the goal is to minimize the number of crossing advantages; this is equivalent to assigning weight one to each edge. The algorithm maintains and improves a partition, in each pass using a greedy algorithm to pair up vertices of A with B so that moving the paired vertices from one side to the other will improve the partition. After matching the vertices, it then performs a subset of the pairs chosen to have the best overall effect on the solution quality T . Given a graph with n vertices, each algorithm pass runs in time $O(n^2 \log n)$.

Algorithm

Iterate as long as the cut size improves:

1. Find a pair of vertices that result in the most significant decrease in cut size if exchanged.
2. Exchange the two vertices (potential move).
3. “Lock” the vertices.
4. If no improvement is possible, and still some vertices are unlocked, then exchange vertices that result in the smallest increase in cut size.

Input:



```
KL_input.txt - Notepad
File Edit Format View Help
8 13
1 2
1 5
1 6
2 5
2 6
3 4
3 6
3 7
3 8
4 7
4 8
5 6
7 8
```

Output

```
"H:\4th Year\4-2\VLSI\Lab\KL Algo.exe"
Initial Partition:
A: [ 1 2 3 4 ]
B: [ 5 6 7 8 ]

Precess: 1
2 => 3, 6
A: [ 1 2 4 6 ]
B: [ 3 5 7 8 ]

Precess: 2
6 => 4, 5
A: [ 1 2 5 6 ]
B: [ 3 4 7 8 ]

Precess: 3
-6 => 1, 7
A: [ 2 5 6 7 ]
B: [ 1 3 4 8 ]

Precess: 4
-2 => 2, 8
A: [ 5 6 7 8 ]
B: [ 1 2 3 4 ]

The Solution of Partition: (MaxGain = 6, For Pair <4,5>)
A: [ 1 2 5 6 ]
B: [ 3 4 7 8 ]

Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

Discussion: From this problem, we could implement the Kernighan-Lin Algorithm on graph partitioning and shown the output successfully.

Experiment No: 09

Experiment Name: Write a program to implement the Quine-McClusky(QM) Method Algorithm.

Objective: The objective is to learn and implement the Quine-McClusky (QM) Method algorithm.

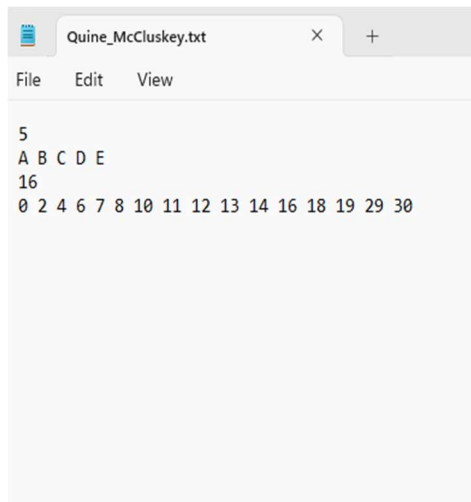
Theory: The quine-McCluskey method also called the tabulation method is a very useful and convenient method for simplification of the Boolean functions for a large number of variables (greater than 4). This method is useful over K-map when the number of variables is larger for which K-map formation is difficult. This method uses prime implicants for simplification.

In this method, we construct multiple tables according to the question and at the last, we make a prime implicant table which is used to obtain essential prime implicants which are present in the simplified boolean expression. This method requires prior knowledge of decimal to binary representation and the basics of boolean algebra. It is a suitable method for a large number of input variables which can be easily solved by this method but the computation complexity is high. Majorly, this method includes the use of minterms, and prime implicants and obtains essential prime implicants which are further used in the simplified boolean functions.

Algorithm

1. Arrange the given minterms according to the number of ones present in their binary representation in ascending order.
2. Take the minterms from the continuous group if there is only a one-bit change to make their pair.
3. Place the '-' symbol where there is a bit change accordingly and keep the remaining bits the same.
4. Repeat steps 2 to 3 until we get all prime implicants (when all the bits present in the table are different).
5. Make a prime implicant table that consists of the prime implicants (obtained minterms) as rows and the given minterms (given in problem) as columns.
6. Place '1' in the minterms (cell) which are covered by each prime implicant.
7. Observe the table, if the minterm is covered by only one prime implicant then it is an essential to prime implicant.
8. Add the essential prime implicants to the simplified boolean function.

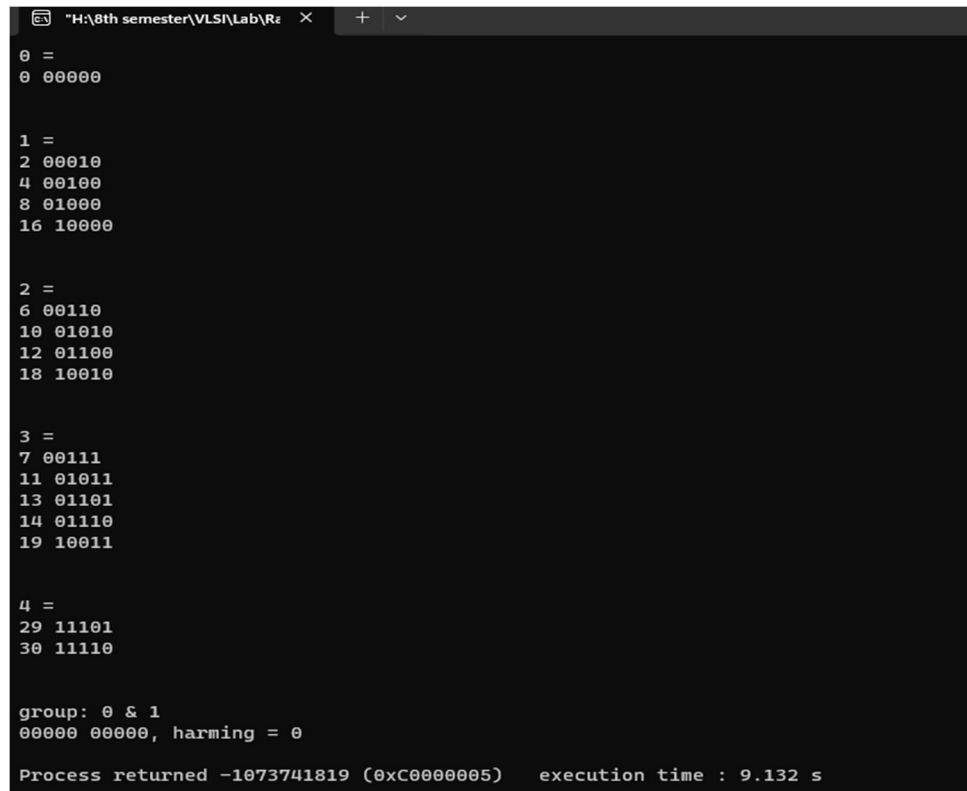
Input:



```
Quine_McCluskey.txt
File Edit View

5
A B C D E
16
0 2 4 6 7 8 10 11 12 13 14 16 18 19 29 30
```

Output:



```
"H:\8th semester\VLSI\Lab\Re X + -
0 =
0 00000

1 =
2 00010
4 00100
8 01000
16 10000

2 =
6 00110
10 01010
12 01100
18 10010

3 =
7 00111
11 01011
13 01101
14 01110
19 10011

4 =
29 11101
30 11110

group: 0 & 1
00000 00000, harming = 0

Process returned -1073741819 (0xC0000005) execution time : 9.132 s
```

Discussion: From this problem, we could implement the Quine-McCluskey(QM) Method Algorithm and show the output successfully.