VLSI DESIGN

22EC2211R/A/E

Academic Year: 2024 - 25

SKILL WORKBOOK

Course Co-ordinator

Dr. Ngangbam Phalguni Singh & Dr. Vijay Rao Khumbare

STUDENT ID:

STUDENT NAME:

ACADEMIC YEAR: 2024 - 25

Vision

To be a globally renowned university.

Mission

To impart quality higher education and to undertake research and extension with emphasis on application and innovation that cater to the emerging societal needs through all-round development of the students of all sections enabling them to be globally competitive and socially responsible citizens with intrinsic values.

****

Department of Electronics and Communication

Vision

To evolve into a globally recognized department in the frontier areas of Electronics & Communication Engineering.

Mission

To produce graduates having professional excellence.

To carry out quality research having social & industrial relevance.

To provide technical support in transforming the learners into entrepreneurs.

****

Table of Contents

A.Y. 2024-25LAB/SKILL CONTINUOUS EVALUATION

S.No	Date	Experiment Name	Pre-Lab (10M)	In-Lab (25M)			Post-Lab (10M)	Viva Voce (5M)	Total (50M)	Faculty Signature
				Program/Procedure (5M)	Data and Results(10M)	Analysis & Inference(10M)				
1.										
2.										
3.										
4.										
5.										
6.										
7.										
8.										
19.										
10.										
11.										
12.										
13.										

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Introduction

LT Spice

Creating and simulating circuits in LTspice involves several steps. Below is a beginner-friendly tutorial to get you started:

Step 1: Download and Install LTspice

1. Visit Analog Devices LTspice page.
2. Download the appropriate version for your operating system (Windows or Mac).
3. Install LTspice by following the on-screen instructions.

Step 2: Getting Familiar with the Interface

- **Toolbar**: Contains tools for placing components, wires, and labels.
- **Schematic Editor**: The workspace where you design circuits.
- **Netlist Viewer**: Displays the textual representation of your circuit.
- **Simulation Window**: Used for viewing the results of your circuit simulations.

Step 3: Creating a Basic Circuit

Here's how to design a simple **RC Circuit**:

1. Open a New Schematic

- Go to File > New Schematic.

2. Add Components

- Press the **"R"** key to place a resistor.
- Press the **"C"** key to place a capacitor.
- Press the **"G"** key to place a ground node (mandatory for simulation).
- Right-click to set component values (e.g., R = 1kΩ, C = 1µF).

3. Add Voltage Source

- Press **F2** or click the "Component" icon to open the component library.
- Search for **"voltage"** and place it.
- Set it as a sinusoidal source by right-clicking it and entering parameters like amplitude and frequency (e.g., AC Amplitude = 1V, Freq = 1kHz).

4. Connect Components

- Use the **"wire" tool** (hotkey: F3) to connect components.

Step 4: Configure the Simulation

1. Go to Simulation > Edit Simulation Cmd.
2. Choose a simulation type:
    - **Transient Analysis**: For time-domain response (e.g., 0 to 10ms with step size 0.01ms).
    - **AC Analysis**: For frequency response (e.g., sweep 10Hz to 1MHz).
3. Click OK, then place the simulation command on the schematic.

Step 5: Run the Simulation

1. Click the **"Run"** button (the running man icon).
2. Use the probe tool (cursor will turn into a voltage probe) to click on nodes to view voltage waveforms.
3. For current, click on component leads (e.g., the resistor or capacitor).

Step 6: Analyze the Results

- **Voltage/Current Graphs**: Observe how voltages and currents change over time or frequency.
- Use cursors to measure specific values (e.g., peak voltage, frequency at -3dB).

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Tips for Common Tasks

- **Adding Models**: Right-click a component to add manufacturer-specific SPICE models.
- **Parametric Sweeps**: Use .step commands to sweep parameter values like resistance or frequency.
- **Custom Libraries**: Import third-party SPICE models (right-click the schematic and use .include commands).
- **Exporting Data**: Right-click the waveform window and export data to CSV for further analysis.

Example Circuit: Low-Pass RC Filter

1. Build a circuit with:
    - Voltage source (V1) set to AC Amplitude = 1V, Freq = 1kHz.
    - Resistor (R1) = 1kΩ.
    - Capacitor (C1) = 1µF.
2. Set up an **AC Analysis**:
    - Frequency sweep: 10Hz to 1MHz.
3. Simulate and observe the cutoff frequency where amplitude drops by -3dB.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

SPICE commands are essential for defining and configuring your circuit simulations. Here's an overview of the most commonly used SPICE commands in LTspice:

---

1. Basic SPICE Commands

These commands are used to define components, models, and configurations in your circuit.

Command	Purpose	Example
.MODEL	Defines a model for a component like a diode, transistor, etc.	.MODEL D1 D (IS=1n RS=1)
.PARAM	Defines a parameter for parametric simulations.	.PARAMRval=1k
.SUBCKT	Defines a subcircuit (reusable circuit block).	.SUBCKT INV A Y Vin=1V
.INCLUDE	Includes an external file containing models or subcircuits.	.INCLUDE mymodels.lib
.LIB	Includes a library file for SPICE models.	.LIB standard.lib
.NODESET	Sets an initial guess for node voltages to aid in convergence.	.NODESET V(n1)=2
.IC	Specifies initial conditions for capacitors or inductors.	.IC V(n1)=2

---

2. Simulation Commands

These commands define the type of simulation to be performed.

Command	Purpose	Example
.TRAN	Performs a transient (time-domain) simulation.	.TRAN 1u 10m
.AC	Performs an AC (frequency-domain) simulation.	.AC DEC 10 10 1Meg
.DC	Performs a DC sweep simulation over a parameter (e.g., voltage or resistance).	.DC V1 0 10 0.1
.OP	Computes the operating point of the circuit.	.OP
.NOISE	Performs a noise analysis simulation.	.NOISE V(out) V1 DEC 10

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Command	Purpose	Example
		1 1Meg
.TF	Computes the transfer function of the circuit.	.TF V(out) Vin
.FOUR	Computes the Fourier series of a transient signal.	.FOUR 10k V(out)
.STEP	Sweeps a parameter (e.g., R, V) over a range.	.STEP PARAM Rval 1k 10k 1k

---

3. Advanced Simulation Options

Use these commands to fine-tune your simulations.

Command	Purpose	Example
.OPTIONS	Sets simulation options like accuracy or timestep.	.OPTIONS ABSTOL=1n VNTOL=1u
.MEAS	Measures specific simulation results (e.g., peak voltage, delay).	.MEAS TRAN V_{peak} MAX V(out)
.SAVE	Saves only specific nodes or currents in the output file.	.SAVE V(n1) I(V1)
.TEMP	Sets the temperature for the simulation.	.TEMP 25
.WAVE	Writes simulation results to a WAV file (for audio simulations).	.WAVE out.wav 44.1k V(out)

---

4. Source Definitions

Voltage and current sources can be defined with parameters for different signal types.

Type	Command Example	Description
DC Source	V1 n1 n2 DC 5	A constant DC voltage of 5V.
AC Source	V1 n1 n2 AC 1 SIN(0 1 1k)	A sinusoidal AC voltage with 1V amplitude.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Type	Command Example	Description
Pulse Source	V1 n1 n2 PULSE(0 5 1u 1u1u 10u 20u)	A pulse signal with rise/fall times.
Piecewise Linear	V1 n1 n2 PWL(0 0 1u 5 2u 0)	A piecewise linear signal.

---

5. Examples of Common Commands

Transient Analysis Example

.TRAN 1u 10m

Simulates the circuit from 0 to 10ms with a timestep of 1µs.

AC Analysis Example

.AC DEC 10 1 1Meg

Performs a decade sweep from 1Hz to 1MHz, plotting 10 points per decade.

DC Sweep Example

.DC V1 0 10 0.1

Sweeps the DC voltage of V1 from 0V to 10V in steps of 0.1V.

Parametric Sweep Example

.PARAM Rload=1k

.STEP PARAM Rload 1k 10k 1k

Sweeps the value of Rload from 1kΩ to 10kΩ in steps of 1kΩ.

Fourier Analysis Example

.FOUR 1k V(out)

Performs a Fourier analysis of V(out) with a fundamental frequency of 1kHz.

- Use .MEAS commands to extract specific results from your simulations, like delay, peak voltage, or power dissipation.
- Add comments in your netlist using * at the beginning of a line.

- Combine commands for complex simulations (e.g., .STEP with .TRAN**).e**

Example MOS circuit designs with SPICE code you can use in LTspice for simulation:

Here are five examples of CMOS circuit designs with SPICE code you can use in LTspice for simulation:

---

1. CMOS Inverter

A basic CMOS inverter switches the output between VDDV_{DD} and 00 based on the input voltage.

```
* CMOS Inverter

M1 out in VDD VDD PMOS W=10u L=1u

M2 out in 0 0 NMOS W=5u L=1u

VDD VDD 0 DC 5

Vin in 0 PULSE(0 5 0 1n 1n 10n 20n)

Cload out 0 10f

.TRAN 1n 100n

.END
```

Explanation:

- **M1 and M2**: PMOS and NMOS transistors.
- **Vin**: Input pulse toggling from 0V to 5V.
- **Cload**: Load capacitor to simulate switching behavior.
- **VDD**: Supply voltage is 5V.

---

2. CMOS NAND Gate

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

A 2-input CMOS NAND Gate generates an output that is high unless both inputs are high.

```
* CMOS NAND Gate

M1 out in2 VDD VDD PMOS W=10u L=1u

M2 out in1 VDD VDD PMOS W=10u L=1u

M3 out in1 0 0 NMOS W=5u L=1u

M4 out in2 out 0 NMOS W=5u L=1u

VDD VDD 0 DC 5

Vin1 in1 0 PULSE(0 5 0 5n 5n 50n 100n)

Vin2 in2 0 PULSE(0 5 0 10n 5n 50n 100n)

Cload out 0 10f

.TRAN 1n 200n

.END
```

Explanation:

- **M1 and M2**: PMOS transistors in parallel.
- **M3 and M4**: NMOS transistors in series.
- **Vin1 and Vin2**: Pulses represent two inputs.
- **Cload**: Simulates capacitive load at the output.

---

3. CMOS NOR Gate

A 2-input CMOS NOR Gate outputs high only when both inputs are low.

```
* CMOS NOR Gate

M1 out in1 VDD VDD PMOS W=10u L=1u
```

```
M2 out in2 out VDD PMOS W=10u L=1u

M3 out in1 0 0 NMOS W=5u L=1u

M4 out in2 0 0 NMOS W=5u L=1u

VDD VDD 0 DC 5

Vin1 in1 0 PULSE(0 5 0 5n 5n 50n 100n)

Vin2 in2 0 PULSE(0 5 0 10n 5n 50n 100n)

Cload out 0 10f

.TRAN 1n 200n

.END
```

Explanation:

- **M1 and M2**: PMOS in series for NOR logic.
- **M3 and M4**: NMOS in parallel for NOR logic.
- **Vin1 and Vin2**: Input signals.

---

4. CMOS Ring Oscillator

A ring oscillator demonstrates an oscillatory output using a series of CMOS inverters.

```
* CMOS Ring Oscillator

VDD VDD 0 DC 5

Cload1 n1 0 10f

Cload2 n2 0 10f

Cload3 n3 0 10f

XINV1 n1 n2 VDD 0 CMOS_INVERTER
```

```
XINV2 n2 n3 VDD 0 CMOS_INVERTER

XINV3 n3 n1 VDD 0 CMOS_INVERTER

.SUBCKT CMOS_INVERTER in out VDD 0

M1 out in VDD VDD PMOS W=10u L=1u

M2 out in 0 0 NMOS W=5u L=1u

.ENDS

.TRAN 1n 1u

.END
```

Explanation:

- **3 inverters**: Connected in a ring to form feedback.
- **Cload**: Simulates delay and stabilizes oscillations.
- **VDD**: Provides power to the ring oscillator.

---

5. CMOS Differential Amplifier

A differential amplifier is used in analog VLSI for amplification of the difference between two signals.

```
* CMOS Differential Amplifier

M1 out1 in1 net1 VDD PMOS W=10u L=1u

M2 out2 in2 net1 VDD PMOS W=10u L=1u

M3 out1 net2 0 0 NMOS W=5u L=1u

M4 out2 net2 0 0 NMOS W=5u L=1u

M5 net1 0 0 0 NMOS W=10u L=1u
```

```
VDD VDD 0 DC 5

Vin1 in1 0 PULSE(0 1 0 1n 1n 10n 20n)

Vin2 in2 0 PULSE(0 1 10n 1n 1n 10n 20n)

I1 VDD net1 DC 10u

.TRAN 1n 200n

.END
```

Explanation:

- **M1 and M2**: PMOS transistors for load.
- **M3 and M4**: NMOS differential pair.
- **M5**: Current source for biasing.
- **Vin1 and Vin2**: Differential input signals.

How to Use These SPICE Codes in LTspice

1. Copy the code into a text file.
2. Save the file with a .asc or .net extension.
3. Open the file in LTspice and run the simulation.
4. Probe nodes (e.g., out) to observe the simulation results.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Exp1: Design and Analysis of CMOS AND Gate

Aim/Objective:

Design and Analysis of CMOS AND Gate using LT Spice tool

Pre-Requisites: LTSpice Version 24.0.12

Pre-Lab Task:

1. What is the truth table of a 2-Input AND Gate? How does it translate into a CMOS circuit?

2. Explain the role of the pull-up and pull-down networks in CMOS logic. Why are PMOS transistors used in the pull-up and NMOS transistors in the pull-down?

3. What happens to the output of the CMOS AND Gate for each combination of inputs (00, 01, 10, 11)? How do the transistors behave in each case?

4. What simulation model can be used to analyze the performance of a CMOS AND Gate in LTSpice?

5. What are the main sources of power dissipation in CMOS circuits? How can they affect the performance of the AND Gate??

In-Lab Task:

I. Tools

1. LTSpice
2. Waveform Viewer (Integrated in LT Spice)
3. Computer with Linux OS and required LT Spice software setup

II. Procedure

1. Open LTSpice

- Launch LT Spice on your computer and create a new schematic (`File -> New Schematic`).

2. Add Components

- Add NMOS and PMOS Transistors:

  - Press `F2` to open the component library.

  -Search for `NMOS` and `PMOS` and place two of each in the schematic.

  -Arrange the PMOS transistors in parallel (for the pull-up network) and the NMOS transistors in series (for the pull-down network).

- Add Voltage Sources:

  - Place a DC voltage source (`F2 -> Voltage`) and set it as V_{DD} (e.g., 5V or 3.3V).

  - Place two additional voltage sources for the inputs (A) and (B).

  - Set these voltage sources as `PULSE` for transient analysis or `DC` for steady-state simulations.

- Add Ground:

  -Use the ground symbol (`F2 -> GND`) and connect it to the sources of the NMOS transistors and the negative terminals of the voltage sources.

3. Connect the Circuit

- Pull-Up Network (PUN):

- Connect the sources of both PMOS transistors to V_{DD}.
- Connect the drains of both PMOS transistors together to form the output node.
- Connect the gates of the PMOS transistors to the inputs (A) and (B).

- Pull-Down Network (PDN):
  - Connect the sources of the NMOS transistors to ground.
  - Connect the drains of the NMOS transistors and to the output node.
  - Connect the gates of the NMOS transistors to (A) and (B).
- Output Node:
  - Label the output node as `Out` using the wire labeling tool.

4. Set Component Properties

- Double-click on the transistors to set their properties (e.g., W/L ratio) or use default values.

- Set the voltage sources:

- V_{DD}: DC voltage (e.g., 5V).

- Inputs A and B: Use `PULSE` settings to generate square wave inputs for transient analysis.

5. Configure Simulation

- Go to `Simulate -> Edit Simulation Cmd`.
- 2. Choose:

  - Transient Analysis:
  - Set appropriate time parameters (e.g., `Stop time: 100ns`).
  - DC Sweep Analysis:
  - Sweep the input voltage from 0 V to V_{DD} to verify the truth table.
- Place the simulation command on the schematic.

6. Run the Simulation

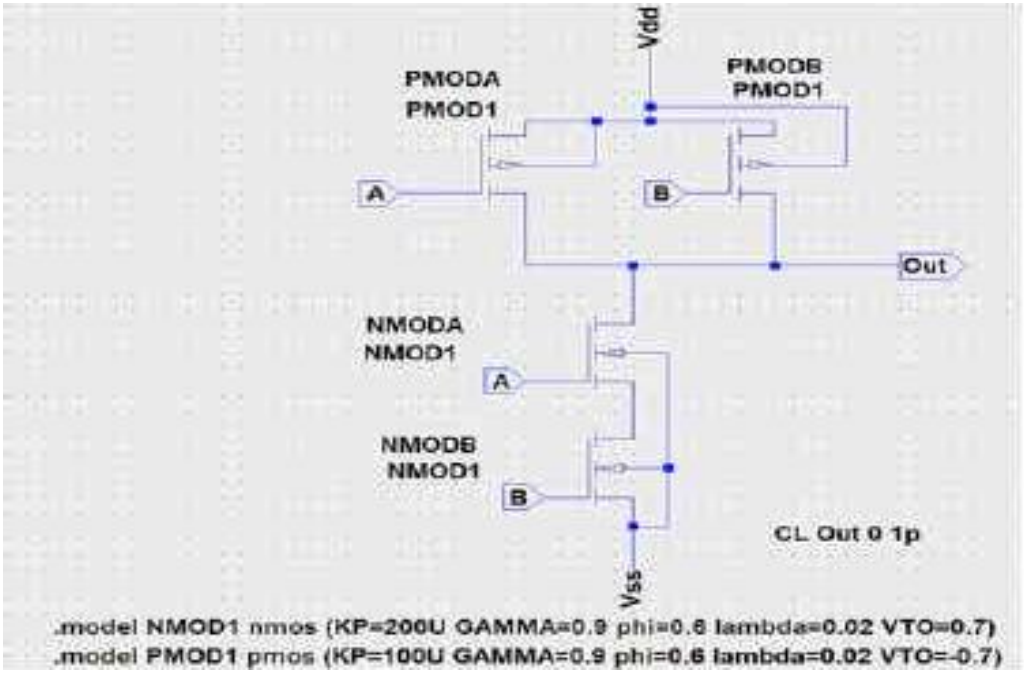
- Press the `Run` button or go to `Simulate -> Run`.

- Use the probe tool to observe the output at the `Out` node and verify its behavior against the input signals A and B.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

7. Analyze Results

- Truth Table Verification:
  - Ensure the output is HIGH (logic 1) only when both inputs A and B are HIGH.
- Propagation Delay:
  - Measure the delay between the input change and the corresponding output change.
- Power Consumption:
  - Measure the current through V_{DD} for static and dynamic analysis.



8. Save and Document

- Save the schematic and simulation results for documentation and future reference.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Post Lab Activities:

1. Does the simulated output match the expected truth table of an AND Gate? Discuss any discrepancies.

2. What is the rise time, fall time, and propagation delays of the CMOS AND Gate? How do these affect the circuit performance?

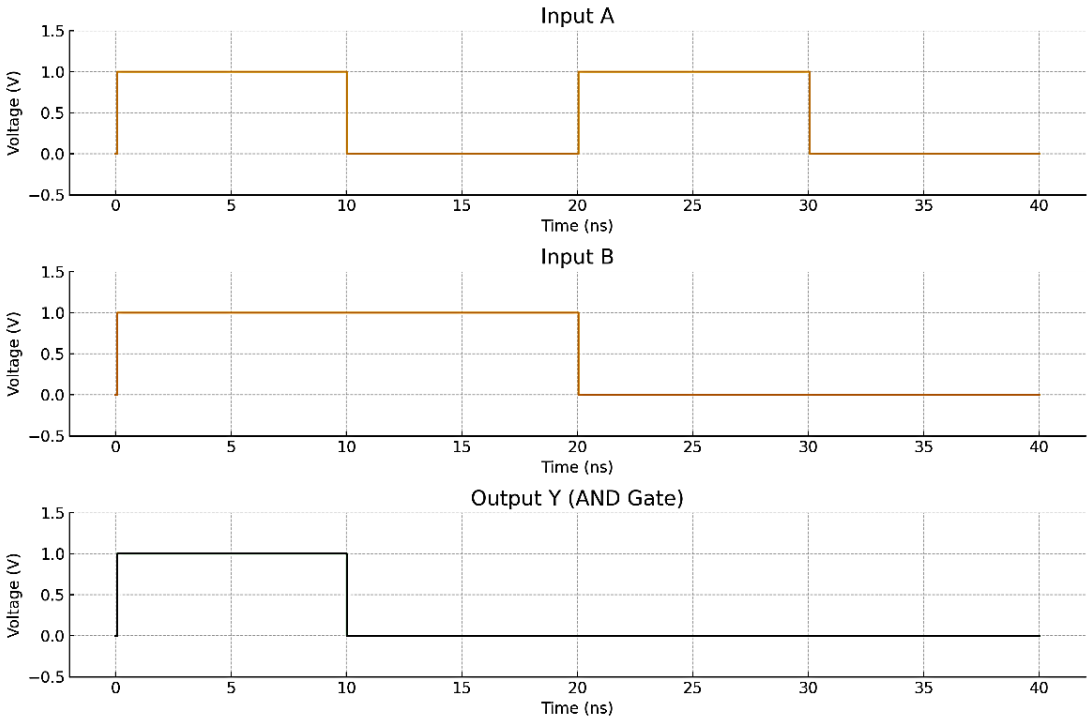
3. Calculate the static and dynamic power dissipation of the circuit. How does it vary with the input switching frequency?

4. Explain the behaviour of the pull-up and pull-down networks for each input combination. How do these networks ensure correct logic operation?

5. What design modifications could be made to improve the performance (speed, power efficiency) of the CMOS AND Gate?

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

RESULTS:



Output and Conclusion:

Add extra page if required

Evaluator Remark (if Any):	Marks Secured:_____out of 50
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Exp2: Design and Analysis of XOR Gate

Aim/Objective:

Design and Analysis of CMOS XOR Gate using LT Spice tool.

Pre-Requisites: Cadence Tool

Pre-Lab Task:

1. What is the truth table of a 2-Input XOR Gate? How does it differ from AND and OR Gates in terms of functionality?

2. Explain how the pull-up and pull-down networks are structured in a CMOS XOR Gate. Why is the design more complex than a basic AND or ORGate?

3. Describe the operation of a CMOS XOR Gate for each input combination. What transistor states are responsible for the high (VDD) and low (0V) outputs?

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

4. What parameters (e.g., voltage levels, delays, power consumption) should be analyzed in the LTSpice simulation of a CMOS XOR Gate?

5. How does the complexity of the XOR Gate affect its power dissipation compared to simpler gates like AND or OR?

In-Lab:

Tools

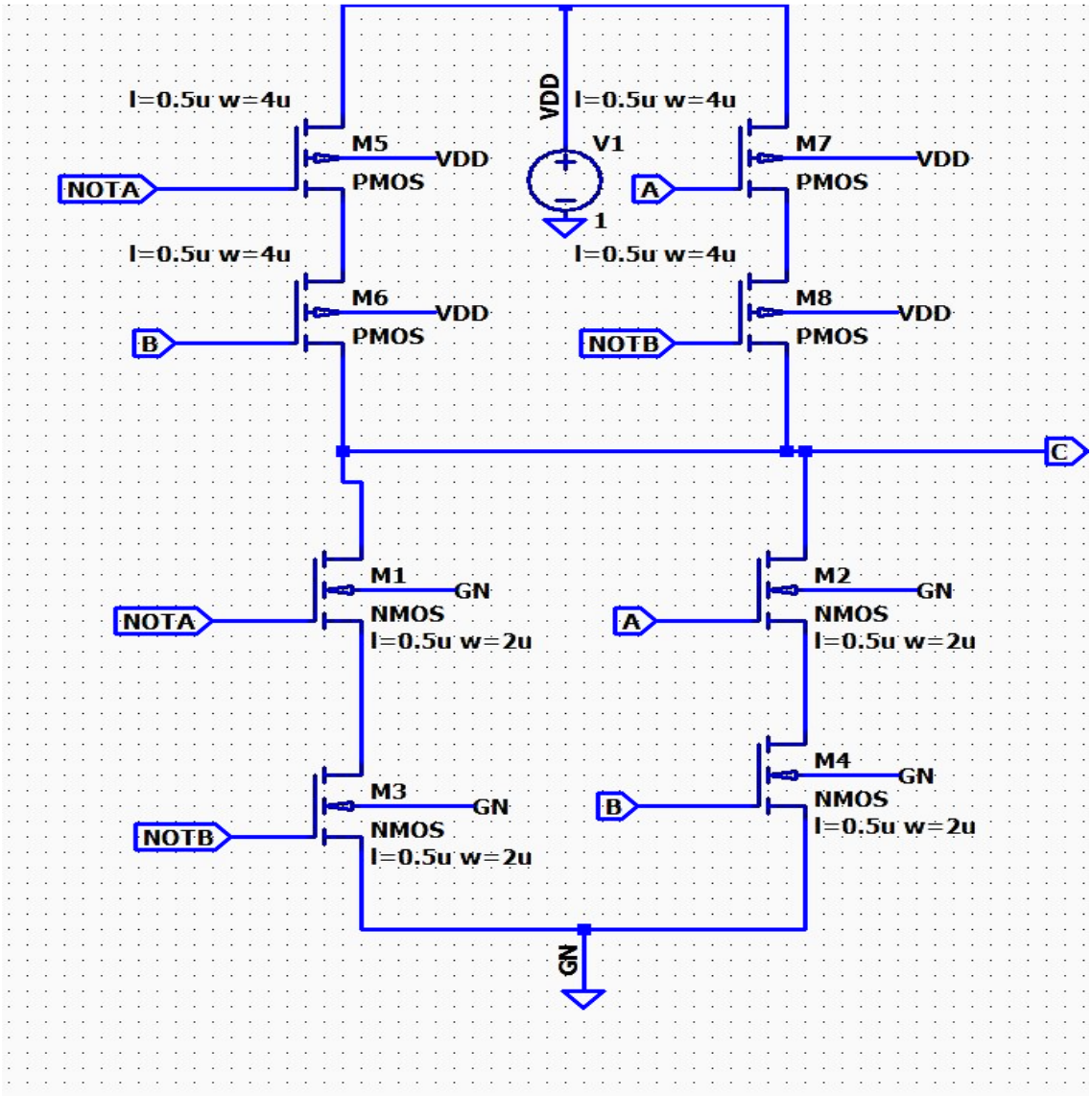
1. LTSpice
2. Waveform Viewer (Integrated in LT Spice)
3. Computer with Linux OS and required LT Spice software setup

Procedure

1. Open LTSpice:
   - Create a new schematic (File -> New Schematic).
2. Add Components:
   - Place four NMOS and four PMOS transistors.
   - Add voltage sources for V_{DD}, A and B.
   - Place a ground symbol and connect it appropriately.
3. Build the Circuit:
   - Connect the transistors in a configuration that realizes the XOR logic:
   - Use series and parallel combinations of PMOS and NMOS transistors.
   - Connect the gates of the transistors to A and B.

- Label the output node as Y.

4. Set Parameters:
   - Set the voltage of V_{DD} (e.g., 5V).
   - Define input signals (A and B) as PULSE sources to create square waves for transient analysis.
5. Simulation:
   - Go to Simulate -> Edit Simulation Cmd.
   - Select Transient Analysis:
   - Set a time range (e.g., Stop time: 100ns).
   - Place the simulation command on the schematic.
   - Run the simulation to observe the output waveform.
6. **DC Sweep:**
   - For verifying the truth table, perform a DC sweep by varying the input voltages (A and B)



Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

and observe the output voltage

Post Lab Activities:

1. Does the simulated output match the expected truth table for an XOR Gate? Explain any discrepancies.

2. What are the rise time, fall time, and propagation delays of the CMOS XOR Gate? How do these compare to simpler gates?

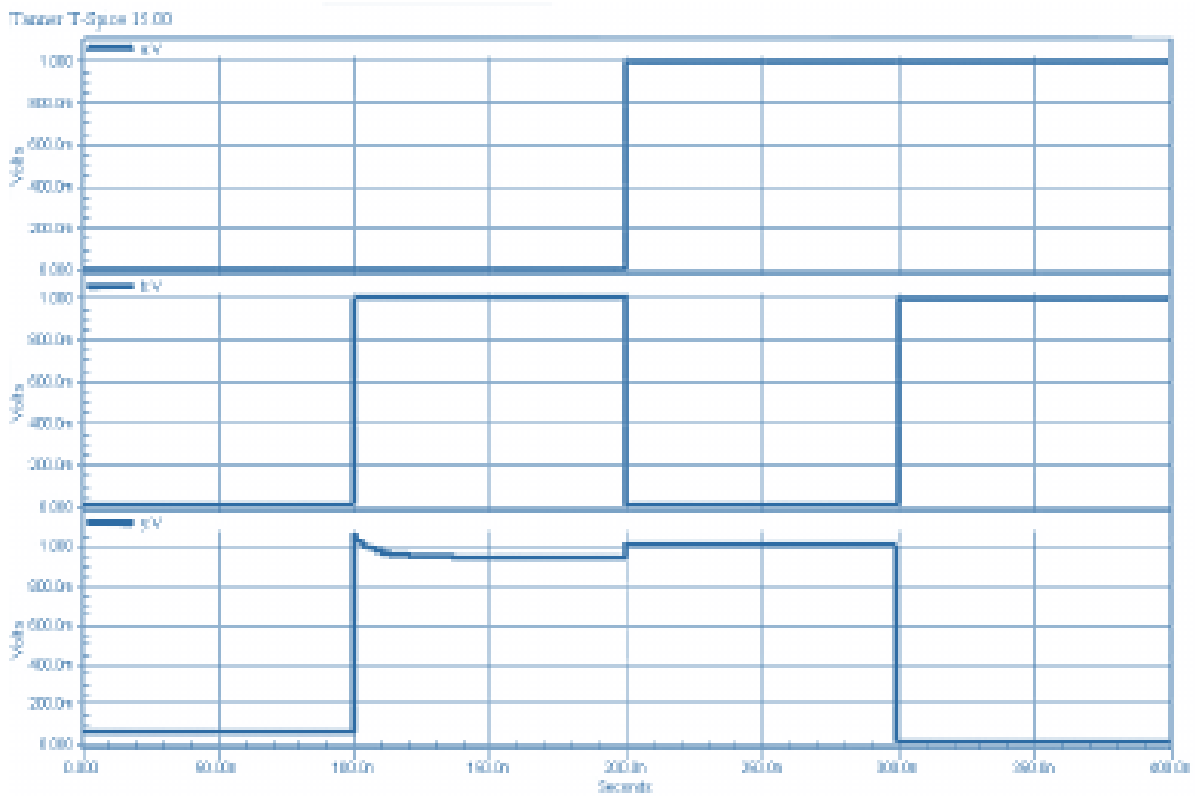
3. How does the power dissipation of the XOR Gate compare with that of simpler gates like AND and OR? Discuss the reasons for any differences?

4. Analyze the operation of the pull-up and pull-down networks for each input combination. How do these networks ensure correct logic functionality?

5. Discuss how the increased complexity of the XOR Gate impacts its design, power, and delay characteristics?

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Expected Results:



Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Observations and Output:

Add Extra Sheets

CONCLUSION:

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Exp3: Design and Analysis of 2x1 Multiplexer

1. Aim/Objective: To design a TG 2x1 multiplexer using LTSPIC and transient response.

2. Description:

A Transmission Gate (TG) is a type of switch that uses complementary MOSFETs (both NMOS and PMOS) to control the flow of signals. When used in a 2x1 multiplexer (MUX) configuration, the transmission gate allows the selection of one of two input signals based on a control signal, and it passes the selected input to the output.

The Transmission Gate 2x1 Multiplexer is a basic digital circuit that has two data inputs, a control signal, and one output. Based on the control signal, either of the two input signals is passed to the output.

A 2x1 multiplexer has:

- Two inputs: A and B
- One control (select) input: S
- One output: Y

The behavior of the multiplexer is as follows:

- When S=0, the output Y is equal to A.
- When S=1, the output Y is equal to B.

In this design, Transmission Gates (TGs) are used to pass the input signals to the output. The TG consists of:

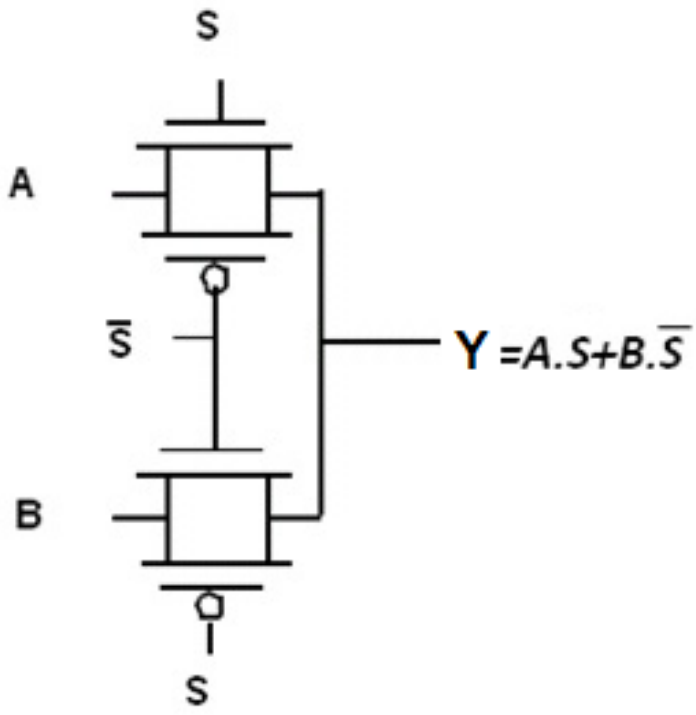
- PMOS transistor controlled by S.
- NMOS transistor controlled by the complement of S (S').

When S=0, the PMOS is turned ON and the NMOS is turned OFF, passing input A to the output. When S=1, the PMOS is turned OFF and the NMOS is turned ON, passing input B to the output.

The *truth table* for a TG 2x1 Multiplexer:

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

The symbols below can be used to represent a TG 2x1 Multiplexer.



Boolean equation: The Boolean equation for a TG 2x1 Multiplexer is Y =A.S + B. ~S

Pre-Requisites: LTSPICE

Pre-SKILL Task:

1. Derive the Boolean expression for a 2x1 multiplexer. How does the control signal (S) determine the output Y?

2. Explain how a transmission gate operates. Why is it necessary to use both NMOS and PMOS transistors in a TG?

3. What are the advantages of using a transmission gate for signal routing compared to a traditional pass transistor?

4. Construct the truth table for a 2x1 multiplexer with inputs A, B, control S, and output Y.

In-SKILL Task:

Procedure

Open LTSpice: Start a new project in LTSpice.

Place Components:

- Place the PMOS and NMOS transistors. These will form the transmission gate.
- Connect the PMOS transistor between VDD and the output node, and the NMOS between the output node and ground.
- Place two input voltage sources A and B for the multiplexer inputs.
- Place a control signal SSS to control the gate terminals of the transistors.

Connect the Circuit:

- PMOS Source connected to VDD and its drain connected to the output node.
- NMOS Source connected to ground and its drain connected to the output node.
- The gate of PMOS is controlled by the control signal S.
- The gate of NMOS is controlled by S' (complement of S).

Create the Control Logic:

- Use an inverter or a NOT Gate to generate the complement of the control signal SSS (i.e., S') for the NMOS Gate.

Configure the Inputs:

- Place voltage sources to represent inputs A and B, typically with square waves or other signal forms to test the functionality of the multiplexer.

Set Parameters:

- Choose appropriate values for transistors' sizes (e.g., W /L ratios for NMOS and PMOS) to ensure proper switching behavior.
- Set the control signal S to alternate between 0 and 1 to switch between inputs A and B.

Run Transient Simulation:

- Set up a transient simulation to observe the output. You can use a square wave for S and test the response of the multiplexer to alternating inputs A and B.

Observe the Output:

- Use the probe tool to measure the output and verify that it correctly reflects the behavior of the multiplexer, switching between A and B based on the value of S.

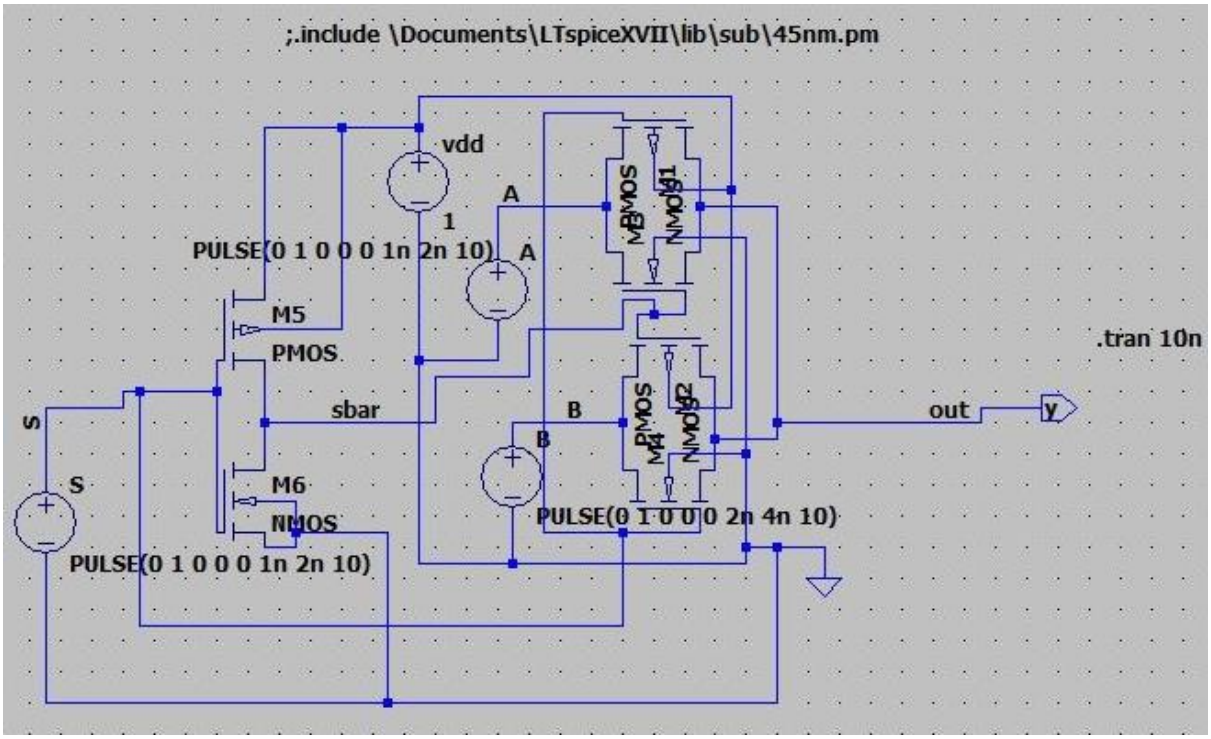


Figure 1– TG 2x1 Multiplexer

Post SKILL Activities:

1. What is the purpose of the complementary control signals used in the transmission gate multiplexer design?

2. What would happen if the complementary control signals were not synchronized (i.e., not perfectly opposite)?

3. What was the impact of the transmission gate's threshold voltage on the multiplexer's performance?

RESULTS:

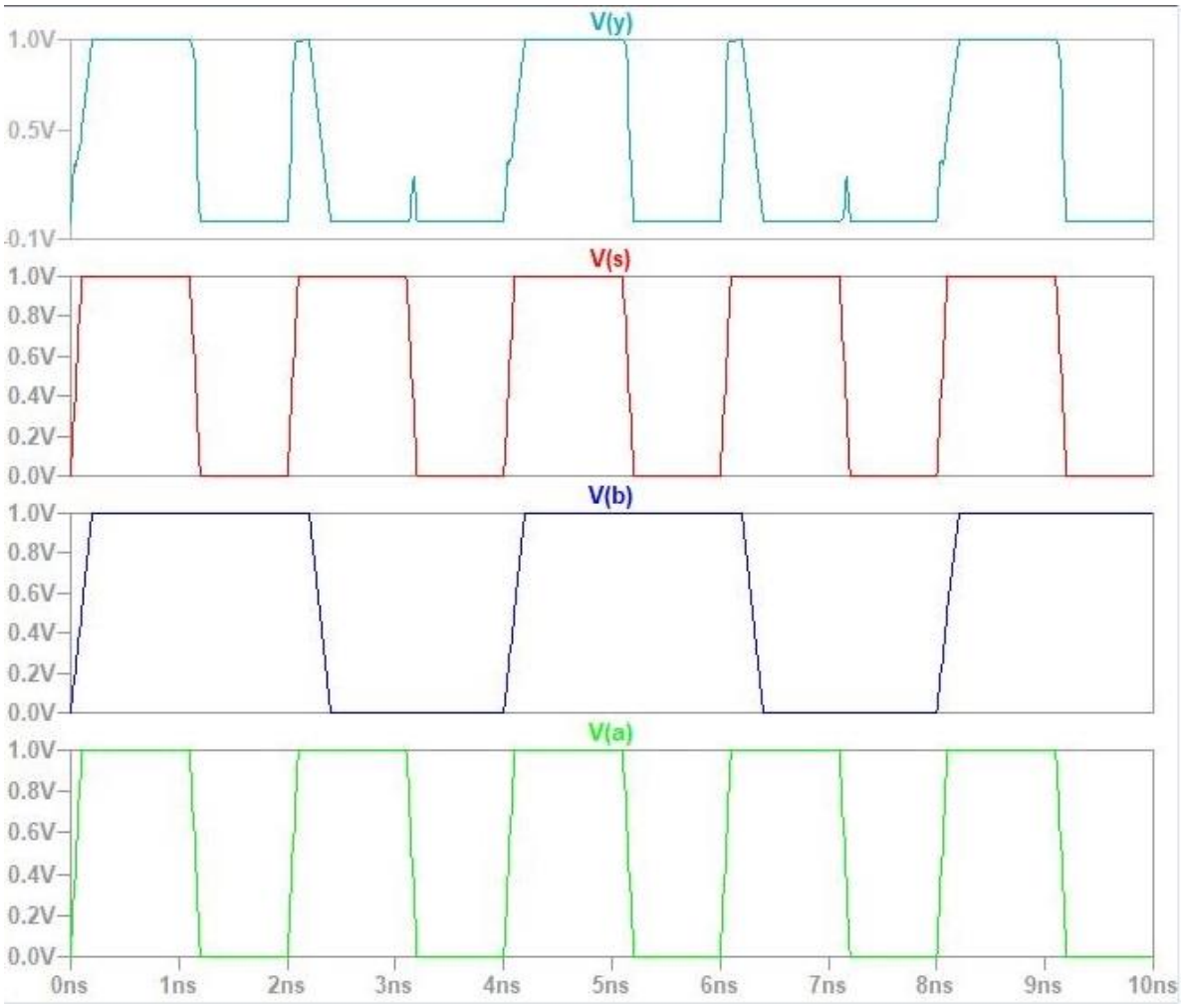


Figure 2– TG 2x1 Multiplexer Transient response

Viva Questions

1. What is the primary role of a transmission gate in a 2x1 multiplexer?

(A) To amplify the signal

(B) To act as a bidirectional switch

(C) To store the input signal

(D) To convert analog signals to digital

2. Which transistors are used in a transmission gate?

(A) Only PMOS

(B) Only NMOS

(C) Both NMOS and PMOS

(D) Bipolar Junction Transistors (BJTs)

3. What is the Boolean expression for the output Y of a 2x1 multiplexer with inputs A and B and control S?

(A) $Y=A+B$

(B) $Y=S\cdot A+\bar{S}\cdot B$

(C) $Y=\bar{S}\cdot A+S\cdot B$

(D) $Y=A\cdot B$

4. Why is the complement of the control signal (S) required in a transmission gate multiplexer?

(A) To control the PMOS transistor

(B) To ensure proper switching between inputs

(C) To save power

(D) To invert the input signals

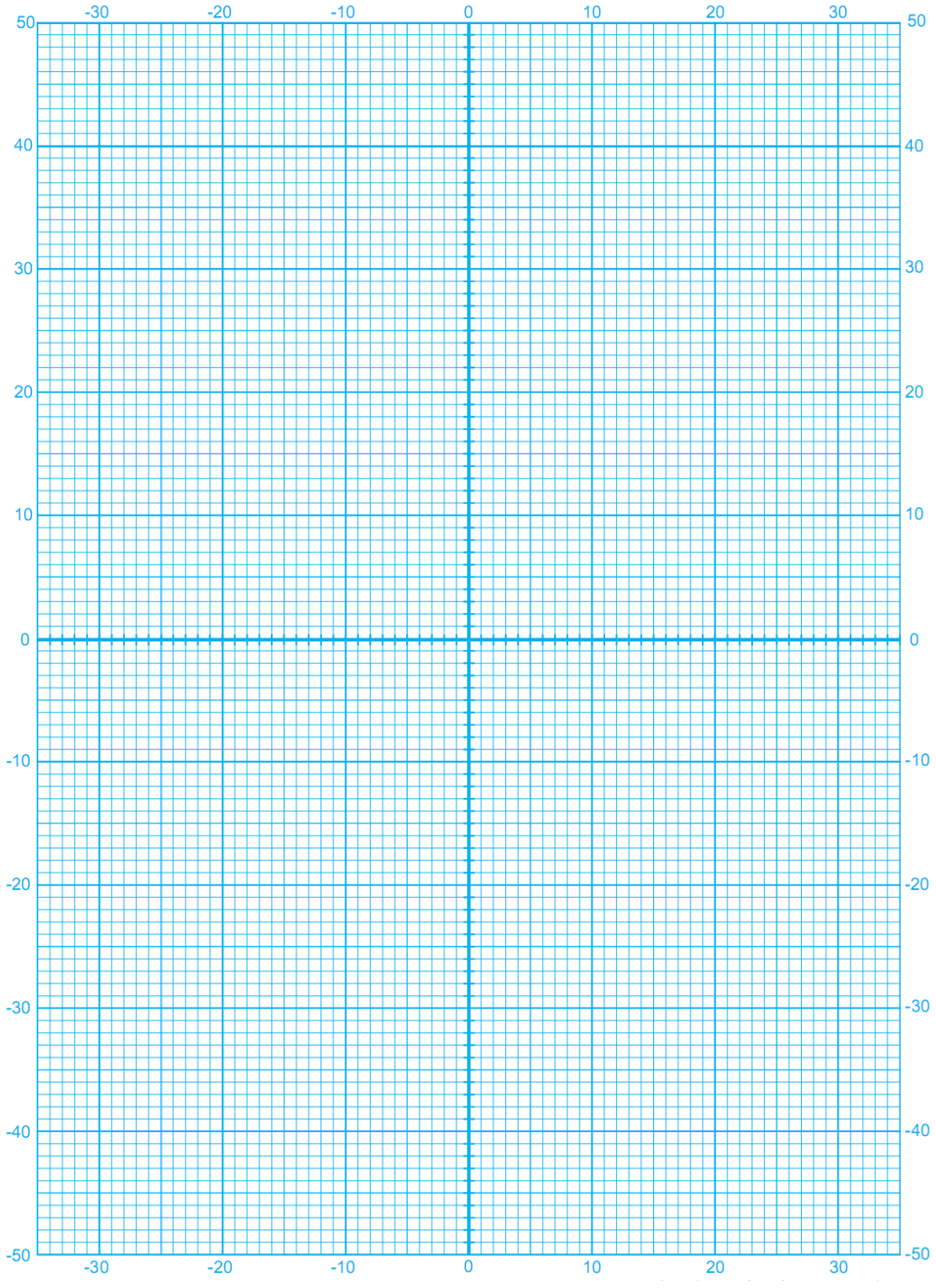
5. What happens if the control signal (S) is 1 in a 2x1 TG multiplexer?

(A) Input AAA is passed to the output

(B) Input BBB is passed to the output

(C) The output is always 1

(D) The output is always 0


-30 -20 -10 0 10 20 30
50 40 30 20 10 0 -10 -20 -30 -40 -50

CONCLUSION:

The TG 2x1 Multiplexer was successfully simulated and analyzed using the Cadence tool. The experiment provided a detailed understanding of the transient response.

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Exp4: Design and Analysis of 4x1 Multiplexer

Aim/Objective: To Design a CMOS 2:1 multiplexer combinational circuit and use the module to design a 4:1 CMOS MUX

Description:

Creating a 4:1 multiplexer using two 2:1 multiplexer involves cascading the 2:1 MUXes in a specific arrangement. While I can guide you through the conceptual design, performing the process in Cadence would require the actual software and access to its tools. Here's a conceptual breakdown of how you might design a 4:1 MUX using 2:1 MUX components in Cadence:

Design Steps:

1. Arrangement of 2:1 MUX:

   Start by arranging two 2:1 MUX blocks in a cascading configuration to create a 4:1 MUX.

2. Input and Select Line Arrangement:

   Assign two input lines (let's call them A and B) to the first 2:1 MUX.

   For the second 2:1 MUX, use two other input lines (C and D).

3. Select Line Configuration:

   Use a select line S as a control input to choose between the outputs of the first and second 2:1 MUX.

4. Interconnection of 2:1 MUXes:

   Connect the output of the first 2:1 MUX to one input of the second 2:1 MUX.

   Connect the output of the second 2:1 MUX to the other input of the second 2:1 MUX.

5. Output Selection:

   The output of the final 4:1 MUX is determined by the select line S:

   When S = 0, the output is taken from the first 2:1 MUX.

   When S = 1, the output is taken from the second 2:1 MUX.

Pre-Requisites: CADENCE Virtuoso Analog Design Environment

Pre-Lab:

1. Define the function of a multiplexer (MUX) and its applications in digital circuits.

2. Explain the role of a 4:1 MUX and how it differs from a 2:1 MUX in terms of input selection.

3. Describe the hierarchical approach of designing a 4:1 MUX using two 2:1 MUXes. How are the 2:1 MUX blocks interconnected to achieve the 4:1 MUX functionality?

4. Describe the cascading arrangement of the 2:1 MUXes and how they are interconnected to yield the 4:1 MUX output.

In-Lab:

- Circuit:

Procedure:

Results:

Analysis and Inferences:

Sample VIVA-VOCE Questions (In-Lab):

1. Detail the logic involved in controlling the selection of output from the individual 2:1 MUXes.

Ans:

2. Analyze the transient behavior in a CMOS MUX design. How do transitions between different inputs or select lines affect the dynamic power consumption of the MUX?

Ans:

3. Relate between average power dissipated by one 2:1 MUX module and the entire 4:1 MUX module.

Ans:

4. How can you design a CMOS 8:1 MUX using CMOS 2:1 MUX.

Ans:

Post lab:

1. To investigate input-output delay of 4:1 MUX. Find the input which suffers maximum delay while transferring its data to output.
2. To design dynamic 4:1 mux logic for reduced power dissipation.

Procedure:

Results:

Analysis and Inferences:

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Exp5: Design and Analysis of D flip-flop

Aim/Objective: To Design D, T flip flops and perform transient analysis

Description:

The circuit has a single input D, which is directly connected to the S input of the latch. The input variable D is also inverted and connected to the R input of the latch. It can be seen from the gate-level schematic that the output Q assumes the value of the input D when the clock is active, i.e., for CK = "1." When the clock signal goes to zero, the output will simply preserve its state. Thus, the CK input acts as an enable signal which allows data to be accepted into the D-latch.

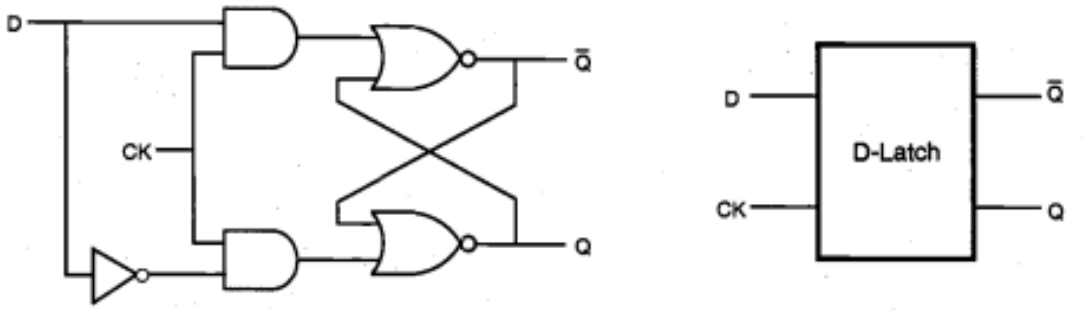


Fig. 4. Gate level implementation of D-Latch

Pre-Requisites: SPECTRE Fundamentals in Virtuoso Design Environment – Cadence Inc.,

Pre-Lab:

1. What is a level-triggered D flip-flop, and how does it differ from edge-triggered flip-flops?

Ans:

2. Explain the basic operation of a level-triggered CMOS D flip-flop with the help of a schematic diagram.

Ans:

3. What are the advantages of using level-triggered flip-flops in digital circuit design?

Ans:

In-Lab:

Circuit:

Procedure:

Results:

Analysis and Inferences:

Sample VIVA-VOCE Questions (In-Lab):

1. Discuss the importance of master-slave latches in the implementation of level-triggered D flip-flops.

Ans:

2. Describe the setup and hold time requirements for stable operation of a level-triggered D flip-flop.

Ans:

3. How does the clock-to-Q delay affect the performance of a level-triggered flip-flop, and how can it be minimized?

Ans:

4. Explain the concept of data transparency in level-triggered flip-flops and its significance in digital systems.

Ans:

Post-Lab:

1. To design edge triggered D-Flipflops
2. To design higher level muxes using lower level muxes.

Procedure:

Results:

Analysis and Inferences:

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Exp6: Design and Analysis of JK Flip-flop

Aim/Objective: To design and analyse JK Flip Flop using LT Spice

Description:

The name JK flip-flop is termed from the inventor ***Jack Kilby*** from ***Texas Instruments.*** Due to its versatility, they are available as IC packages. The major applications of JK flip-flop are Shift registers, storage registers, counters and control circuits. Inspite of the simple wiring of D type flip flop, JK flip-flop has a toggling nature. This has been an added advantage. Hence, they are mostly used in counters and PWM generation, etc. Here we are using NAND Gates for demonstrating the JK flip flop.

Whenever the clock signal is LOW, the input is never going to affect the output state. The clock has to be high for the inputs to get active. Thus, JK flip-flop is a controlled Bi-stable latch where the clock A

JK flip-flop is a versatile digital circuit that can operate in three modes: set, reset, and toggle, depending on its inputs. Implementing this in LTSpice involves creating the circuit using basic logic gates or subcircuits. Signal is the control signal.

Circuit Diagram:

The JK Flip-Flop can be implemented using:

- Four NAND Gates for the basic flip-flop structure.
- Additional NAND Gates for input logic (to implement J and K inputs).

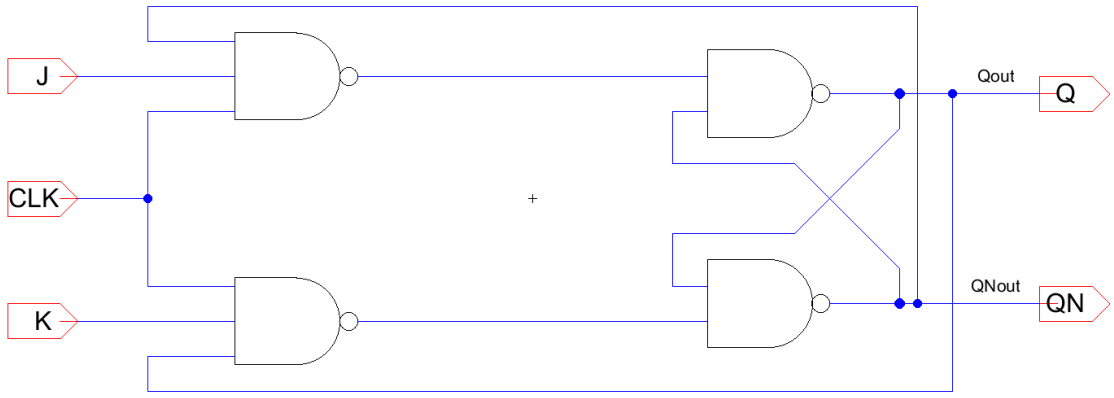
A basic JK Flip-Flop circuit diagram can be built as follows:

- Two cross-coupled NAND Gates for the flip-flop.
- One NAND Gate each for J and K input logic (feeding the flip-flop).

- Clock signal applied to control when the inputs are processed.

JK Flip-Flop Truth Table

J	K	Clock	Q (Next State)	$\overline{Q}$ (Next State)
0	0	Rising	No Change	No Change
0	1	Rising	0	1
1	0	Rising	1	0
1	1	Rising	Toggle	Toggle



Pre-Requisites: LTSPICE

Pre Skill-Task

1. What is a flip-flop in VLSI and what are they various types of flip-flops?
2. What is the difference between a latch and a flip-flop?
3. What are the key design parameters for flip-flops in VLSI?
4. What is the role of the clock signal in a flip-flop?
5. What is setup and hold time? Why are they important?
6. What are the design considerations for JK Flip-Flops in circuits?

In-SKILL Task:

Procedure:

- Set Up LTSpice: Open LTSpice and create a new schematic.
- Add Basic Components:
- Logic gates: NAND gates are needed to build a JK Flip-Flop.
- Power supply: Use voltage sources (e.g., V_{dd}) to power the circuit.
- Clock signal: Use a pulse voltage source to simulate the clock input.
- Resistors and capacitors: As needed for pull-up/down or timing.

Configure Components:

- Use Voltage-controlled switches or define transistor models to mimic the NAND gate functionality.
- Set the Clock Input (CLK) as a pulse signal.
- Adjust V_{DD}, rise time, fall time, high/low time, and period as needed.
- Label the inputs as J,K, and CLK.
- Label the outputs as Q and Q^1

Simulation:

- Go to ***Simulate*** **>** ***Edit Simulation*** Cmd.
- Select ***Transient Analysis*** and set the simulation time (e.g., 200ns).
- Place the simulation command on the schematic.

Probe Signals: After simulation, click on the output nodes (Q and Q^1) to view the waveforms. Verify the JK Flip-Flop behavior for different combinations of J, K, and CLK.

Post SKILL Activities:

1. What are the inputs and outputs of a JK Flip-Flop?

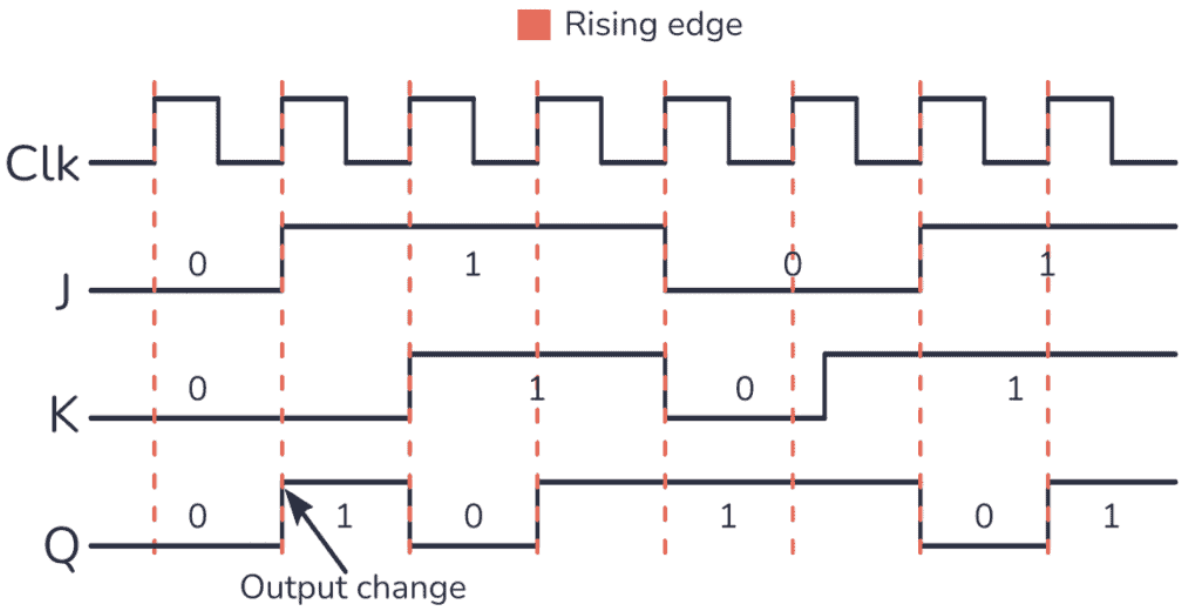
2. What happens when J=0 and K=0?

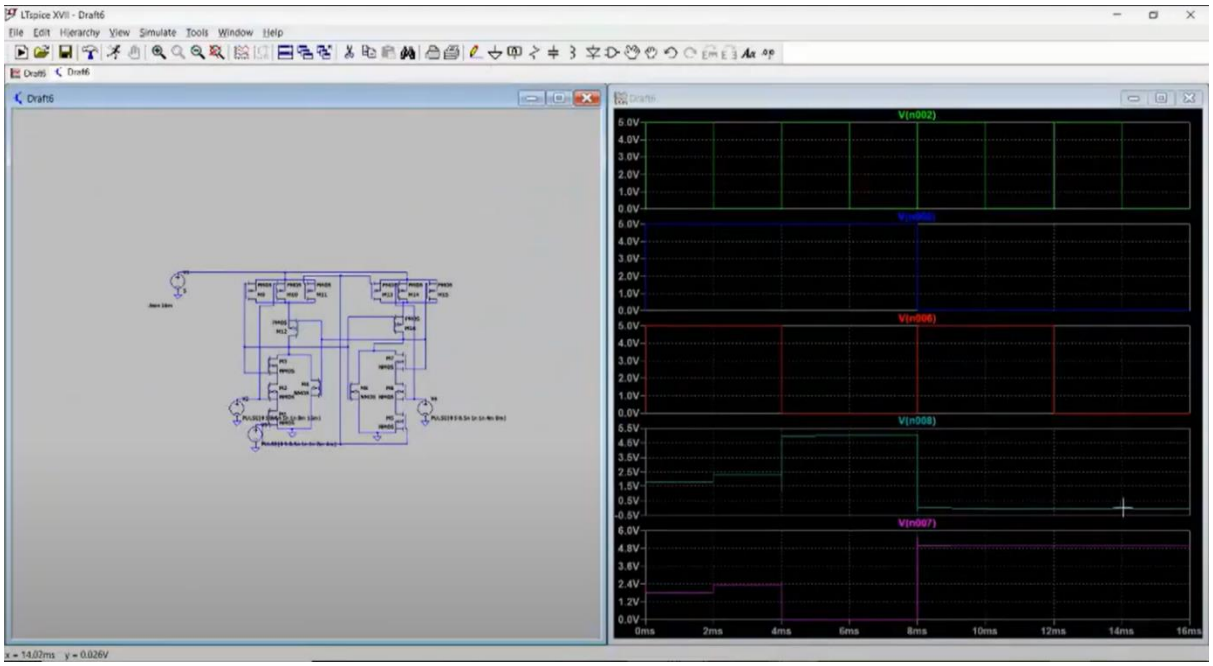
3. What happens when J=1 and K=1?

4. What are race-around conditions in JK Flip-Flops?

5. What is the difference between synchronous and asynchronous JK Flip-Flop operation?

Results:





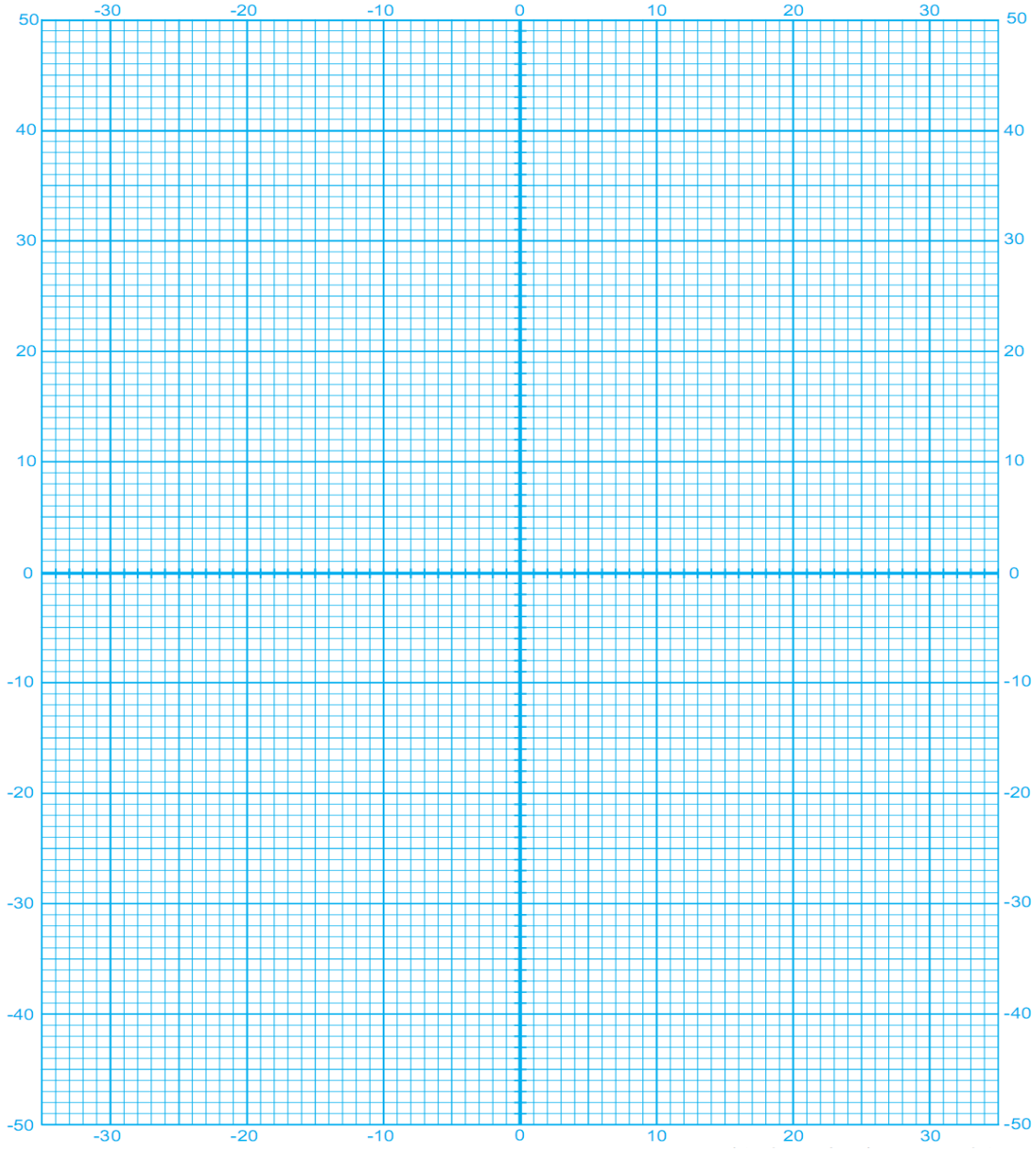
Viva Questions:

1. What is the characteristic equation of a JK Flip-Flop?
   A. $Q_{next} = J + \overline{K} \cdot Q$
   B. $Q_{next} = J \cdot \overline{Q} + \overline{K} \cdot Q$
   C. $Q_{next} = J \cdot K + Q$
   D. $Q_{next} = J \cdot Q + K \cdot \overline{Q}$
2. How is the race-around condition avoided in JK Flip-Flops?
   A. By using asynchronous inputs
   B. By using a latch instead of a flip-flop
   C. By using edge-triggered or Master-Slave configuration
   D. None of the above
3. How many states are possible in a JK Flip-Flop?
   A. 1
   B. 2

C. 3

D. 4

4. Which of the following is NOT an application of a JK Flip-Flop?
   A. Counters
   B. Frequency Division
   C. Multiplication
   D. Shift Registers
5. If a JK Flip-Flop operates in toggle mode, what is its output frequency relative to the clock frequency?
   A. Same as the clock frequency
   B. Twice the clock frequency
   C. Half the clock frequency
   D. One-fourth the clock frequency



Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Exp7: Design and Analysis of SRAM Cell

Aim/Objective:

To design and analyze a 6-transistor Static Random-Access Memory (SRAM) cell using LTspice and observe its behavior during read and write operations. The goal is to understand how an SRAM cell stores data and how it can be accessed through word and bit lines.

Description:

Static Random-Access Memory (SRAM) is a type of semiconductor memory that uses flip-flops to store data. The 6T SRAM cell consists of four transistors that form a cross-coupled latch for data storage and two additional access transistors for reading and writing operations. In this experiment, the design of a basic SRAM cell (6T) is simulated using LTspice, and the effects of the read and write operations are studied.

The key components of an SRAM cell are:

- Two PMOS transistors that form the data storage latch.
- Two NMOS transistors that control the access to the storage node.
- Two access transistors (NMOS) that control data reading and writing to/from the bit lines.

The SRAM cell will be analyzed using transient analysis to observe its dynamic behavior, focusing on how the stored data is read and written when toggling the word line and bit lines.

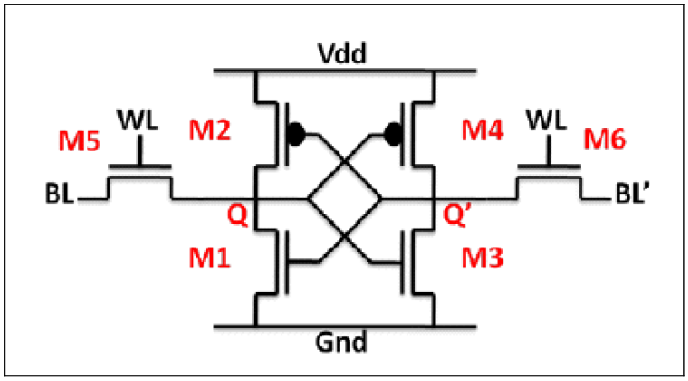
Pre-Requisites: LTspice

Pre-Lab Task:

1. Review the basic properties of NMOS and PMOS transistors and how they function in digital circuits.
2. Learn about the .tran (transient) command for time-domain simulation.

3. Learn about the .op (DC operating point) command to check the initial state of the circuit.

In-Lab:



After placing the transistors and connections, you need to set up the simulation parameters.

1. DC Operating Point:
    - For a DC operating point analysis, use a .op command to check the biasing and current flow.
2. Transient Analysis:
    - Add a transient analysis to observe how the SRAM cell stores and retrieves data. Set up a pulse for the word line (WL) and bit lines (BL, BL_).
    - Use the .tran command for transient analysis. Set a time step that allows you to observe the behavior of the SRAM cell in action.
3. AC Analysis (Optional):
    - You may also perform an AC analysis to see the stability of the SRAM cell at various frequencies.
4. Logic Simulation:
    - The bit lines (BL, BL_) should switch depending on the data being stored and accessed.
    - Toggle the WL (word line) to simulate a read or write operation.

- Write Operation: Set the bit lines to the desired logic level (high or low).

Write NET List Program for LTspice:

Obtained Wave Form :

Paste your Output waveform obtained from EDA Tool(In Colour Print)

Inference and Analysis:

Sample VIVA-VOCE Questions (In-Lab):

1. What is the function of the cross-coupled transistors in the SRAM cell?
2. Explain the role of access transistors in the SRAM cell.
3. Why do we use both PMOS and NMOS transistors in the SRAM design?
4. What is the significance of the word line (WL) in SRAM operation?
5. How can the performance of an SRAM cell be improved or optimized?

Post-Lab:

1. Check the waveform of the word line (WL), bit line (BL), and bit line complement (BL_).
2. Observe the read and write cycles to ensure that the SRAM cell is storing and retrieving data correctly.
3. Analyze how changes in the size of the transistors or the timing of the pulse signals affect the behavior of the SRAM cell.
4. Modify transistor sizes and observe the impact on the stability and power consumption of the SRAM.

Post Lab Work and Result:

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Exp8: Design and Analysis of SRAM array

Aim/Objective:

To design and analyze an SRAM array using LTspice and observe its functionality during read and write operations. The goal is to simulate the behavior of an array of 6T SRAM cells and understand how multiple bits are stored and accessed using word and bit lines.

Description:

SRAM (Static Random-Access Memory) is used in digital systems for high-speed data storage. An SRAM cell typically consists of six transistors (6T), and when extended into an array, it forms a memory block capable of storing multiple bits of data. In this experiment, we will design an SRAM array where each cell is a 6T SRAM unit. The operation of the SRAM array will be analyzed by performing read and write operations, with the array being organized in rows and columns, with word lines (WL) and bit lines (BL, BL_) to control access.

The main steps involved:

- Design an SRAM cell (6T) and replicate it to form an array.
- Implement word lines (WL) and bit lines (BL, BL_) to control data access.
- Simulate read and write operations using LTspice.
- Observe and analyze the behavior of the array.

In this experiment, we will simulate the behavior of an SRAM array of size 2x2 (2 rows and 2 columns) and analyze how data is written and read from the array.

Pre-Requisites: LTspice Tool

Pre-Lab:

1. Design a 6T SRAM cell in LTspice, ensuring correct connections of PMOS, NMOS, bit lines, and word lines.

2. Test the basic SRAM cell design using transient analysis.
3. Learn how word lines (WL) and bit lines (BL, BL_) work in controlling access to the SRAM array.

In-Lab:

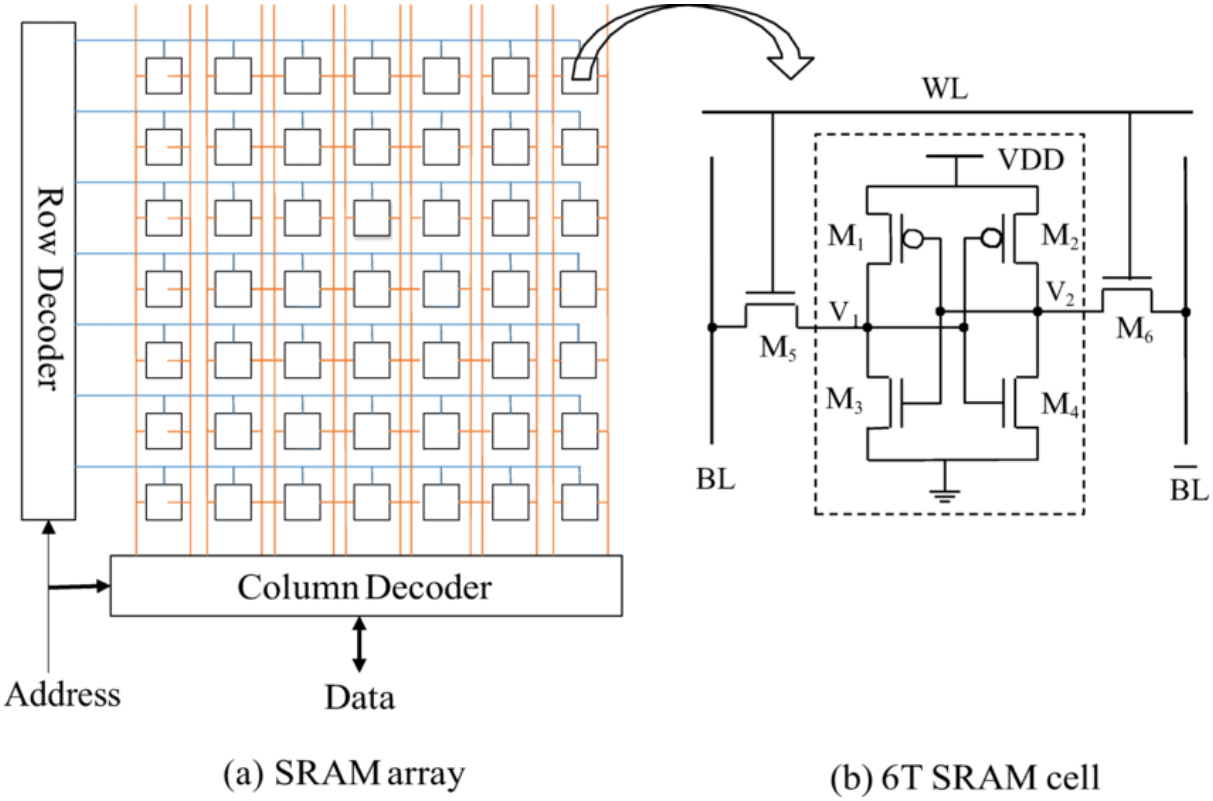


Fig8.1: 8*8 SRAM array 6T SRAM Cell Design

Observation Table

Operation	**Word Line (WL)**	**Bit Line (BL)**	**Bit Line Complement (BL_)**	**Cell Output (Stored Data)**	**Remarks**
Write Operation	Active (High)	High or Low	Low or High	Written Data (1 or 0)	Data written into the cell
Read Operation	Active (High)	Reflects Stored Data	Complement of Bit Line	Stored Data (1 or 0)	Data read from the cell

Operation	Word Line (WL)	Bit Line (BL)	Bit Line Complement (BL_)	Cell Output (Stored Data)	Remarks
Idle State	Inactive (Low)	Stable (No Change)	Stable (No Change)	Retained Data (1 or 0)	SRAM cell holds the data

- Write Operation: The bit line is set to the desired value (high or low), and the word line is activated (high), writing the data to the SRAM cell.
- Read Operation: The word line is activated, and the bit lines reflect the stored value (1 or 0) from the SRAM cell.
- Idle State: When the word line is low, the SRAM cell retains its stored data, and there is no activity on the bit lines.

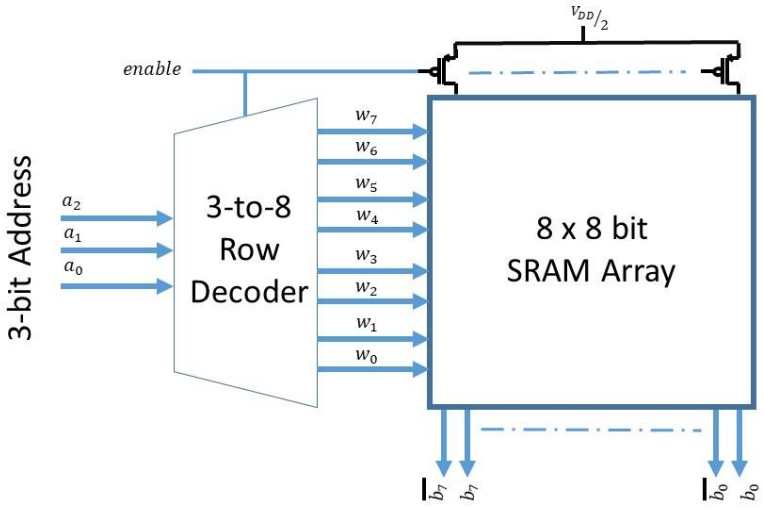


Fig8.2.: 8*8 SRAM array 6T SRAM Cell Design

Obtained Wave Form :

Paste your Output waveform obtained from EDA Tool(In Colour Print)

Inference and Analysis:

Sample VIVA-VOCE Questions (In-Lab):

1. What is the purpose of the cross-coupled transistors in the 6T SRAM cell?
2. How do the word lines (WL) and bit lines (BL, BL_) control the read/write operations in an SRAM array?
3. What would happen if the word line (WL) is kept high during idle state?
4. Explain how a 2x2 SRAM array is organized in terms of word lines and bit lines.
5. What factors affect the performance of an SRAM array in terms of speed and power consumption?

Post-Lab:

1. Check that when a write operation is performed (WL high, BL set to 1 or 0), the corresponding SRAM cell stores the correct value.
2. Verify that during a read operation (WL high), the bit lines reflect the stored data (BL reflects stored value, BL_ its complement).
3. Test the behavior of multiple cells in the array and ensure that each cell operates independently.
4. Observe any interference or crosstalk between cells in the array and analyze the performance.

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Exp9: Design and Analysis of LUT2

Aim/Objective:

To design and simulate a CMOS circuit implementing a Look-Up Table (LUT) using the LTspice tool, and to analyze its operation as a combinational logic block that stores and retrieves predefined logic outputs based on input combinations.

Description:

A Look-Up Table (LUT) is a fundamental building block in digital logic design, particularly in Field Programmable Gate Arrays (FPGAs). It stores predefined outputs corresponding to a set of input combinations and is implemented using CMOS logic gates. The CMOS implementation provides low power consumption and high reliability.

In this experiment:

- A 2-input LUT will be designed using CMOS Gates to implement a logic function (e.g., AND, OR, XOR, etc.).
- The inputs will be binary signals, and the output will be determined by the stored logic function.
- The LUT will be analyzed by applying all possible input combinations and verifying the output.

Theory :

Look-up tables are known in the literature as a method of implementing an arbitrary binary logic function. A truth table for a general 2-input 1-output combinational logic function is shown in Figure 9.1. In a combinational logic expression, the output is uniquely determined by the current input bits. Therefore, a truth table like that shown in Figure 9.1(a) can fully characterize the function. A look-up table (LUT) is a direct implementation of the truth table. Figure 9.1(b) shows a LUT implementation for the truth table shown in Figure 9.1(a). There is a one-to-one correspondence between the rows of the truth table and the rows of the LUT.

For any given input bit-pattern, only one of the paths from input fi to output Z has all of its transistors ON. Referring to Figure 9.1(b), the intermediate nodes of rows 0 and 2 have the same voltage independent of the logic values of A and B. The same is true for the intermediate nodes of rows 1 and 3. That means the right-most transistors of rows 0 and 1 can be shared with those of rows 2 and 4, respectively. The result is shown in Figure 9.1(c). The ON path of a binary LUT carries binary information (i.e. 0 or 1) in the form of voltage or current signal to the output node. The main application of this is in an intelligent memory that could perform data mining, pattern recognition, or image processing.

Pre-Requisites: Cadence Tool

Pre-Lab:

1. Study the concept of a Look-Up Table and how it stores logic functions.

2. Understand the truth table representation of combinational logic circuits.

3. Select a simple logic function (e.g., F = A XOR B) for implementation in the LUT.

4. Prepare the truth table for the chosen logic function.

In-Lab:

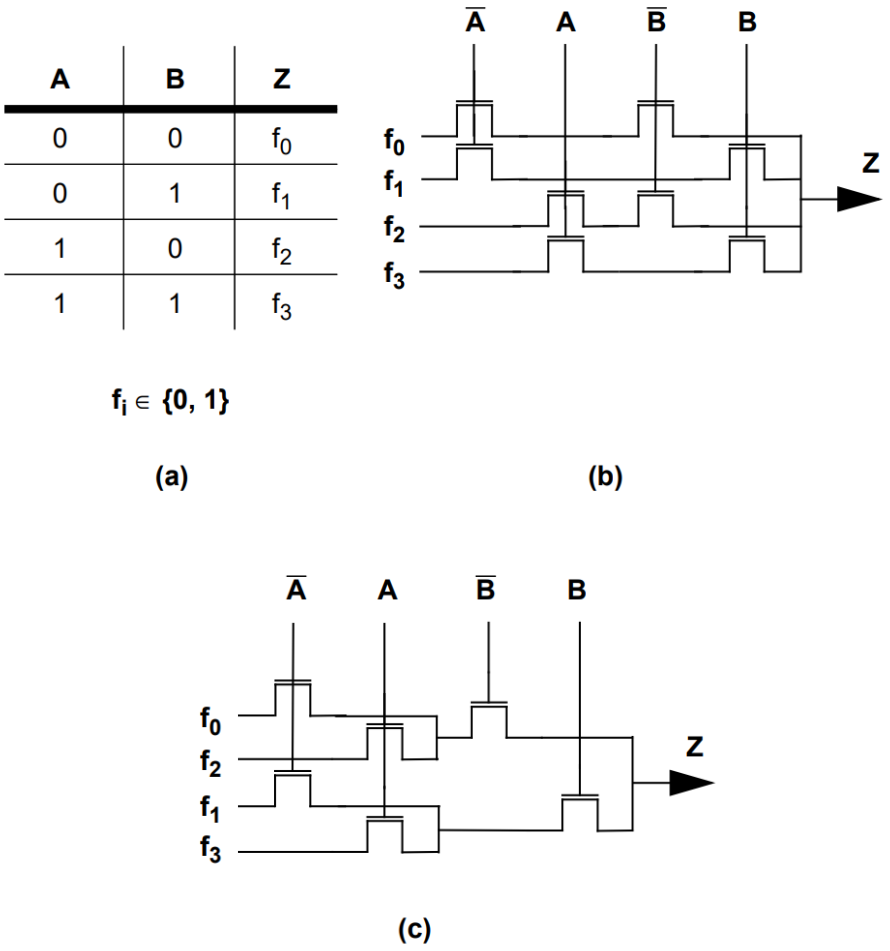


Figure 9. 1. Example (a) A truth table for a general 2-input logic function, (b) A LUT implementation of the function, (c) A LUT implementation with reduced transistor count

In-Lab Procedure

1. Setup in LTspice:
    - Open LTspice and create a new schematic.
    - Place the required components:
        - NMOS and PMOS transistors to implement the chosen logic gates.

- Voltage sources for power supply (VDD and VSS).
- Input voltage sources (pulse generators or DC sources) for the logic inputs.

- Wire the components to form the CMOS circuit implementing the LUT.

2. Apply Inputs:
   - Add pulse sources to the input nodes to simulate all possible combinations of input signals (e.g., 00, 01, 10, 11 for a 2-input LUT).
3. Simulation Setup:
   - Add a transient analysis directive (e.g., .tran 1u 100u) to observe the circuit's behavior over time.
   - Label the input and output nodes for easy identification in the waveform viewer.
4. Run the Simulation:
   - Run the transient simulation.
   - Observe the output waveform and verify it matches the expected truth table for the chosen logic function.
5. Analyze Power Consumption :
   - Use the LTspice probe tool to measure current through VDD and calculate the power consumption of the circuit.

Obtained Wave Form :

Paste your Output waveform obtained from EDA Tool(In Colour Print)

Inference and Analysis:

Sample VIVA-VOCE Questions (In-Lab):

Post-Lab:

1. Compare the simulated output waveforms with the expected truth table.
2. Identify any discrepancies and troubleshoot the circuit if necessary.
3. Analyze the timing behavior of the LUT, including propagation delay and switching characteristics.
4. Suggest ways to optimize the CMOS circuit in terms of power consumption or speed.
5. Discuss the scalability of the circuit for implementing larger LUTs.

Post Lab Work and Result:

Add extra sheet if need more space

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Exp10: Design and Analysis of LUT4

Aim/Objective:

To design and analyze a 4-input Look-Up Table (LUT4) using CMOS logic in LTspice, simulate its operation, and verify its functionality as a digital logic component capable of implementing any 4-input combinational logic function.

Description:

A 4-input Look-Up Table (LUT4) is a digital logic component that can store and produce outputs for all possible combinations of four inputs. It is widely used in Field Programmable Gate Arrays (FPGAs) for implementing combinational logic functions.

The LUT4 is essentially a small memory array that uses CMOS-based multiplexers to select the appropriate output based on the input combination. With 4 inputs (A, B, C, D), the LUT4 can implement any logic function by storing the desired output values (truth table) for the 24=162^4 = 1624=16 input combinations.

In this experiment:

- A CMOS-based LUT4 will be designed using a combination of inverters and multiplexers.
- The truth table for a chosen logic function will be stored in the LUT4, and its behavior will be validated using LTspice.

Pre-Requisites: LTSpiceTool

Pre-Lab:

1. Choose a logic function (e.g., F = A·B+C·DA \cdot B + C \cdot DA·B+C·D) and create its truth table with all 16 input combinations.
2. Plan how to map the truth table into the LUT4 structure.
3. Practice designing simple CMOS Gates and MUX circuits in LTspice.
4. Learn how to use pulse sources to generate input combinations for the LUT.

In-Lab Procedure

1. Design the LUT4 in LTspice:
    - Open LTspice and create a new schematic.
    - Implement the LUT4 structure using:
        - CMOS inverters for generating complement signals.
        - CMOS 2-to-1 multiplexers for selecting the stored output based on inputs.
    - The LUT4 will require multiple levels of MUX (hierarchical design) to handle 16 possible output values.
2. Configure Inputs and Truth Table:
    - Use voltage sources or pulse sources to generate all 4 input signals (A, B, C, D).
    - Configure the LUT4 circuit with the truth table for the chosen logic function by connecting appropriate outputs to each MUX level.
3. Set Up Simulation:
    - Add a transient analysis directive (e.g., .tran 1u 100u) to observe the dynamic behavior of the LUT4.
    - Label all input and output nodes for easy identification in the simulation results.
4. Run Simulation:
    - Simulate the LUT4 circuit and verify that the output matches the truth table for all 16 input combinations.
    - Use the LTspice waveform viewer to analyze the circuit's response.

Obtained Wave Form :

Paste your Output waveform obtained from EDA Tool(In Colour Print)

Inference and Analysis:

Sample VIVA-VOCE Questions (In-Lab):

1. What is a LUT4, and why is it important in digital systems?
2. Explain how a MUX is used to implement a LUT.
3. What is the significance of truth tables in LUT design?
4. How does the propagation delay affect LUT performance?
5. What are the advantages of CMOS technology in LUT implementation?

Post-Lab:

1. Compare the output waveforms with the expected truth table values.
2. Verify that the LUT4 produces correct outputs for all input combinations.
3. Measure the propagation delay of the LUT4 (time taken for the output to stabilize after input changes).
4. Observe any glitches or anomalies in the output waveforms and analyze their causes.
5. Suggest optimizations to reduce the delay or power consumption of the LUT4.
6. Discuss the scalability of the design for higher input LUTs (e.g., LUT5 or LUT6).

Post Lab work and Analysis:

Add Extra sheet

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Exp11: Design and Analysis of Combinational Logic Block (CLB)

Aim/Objective:

To design and analyze a Combinational Logic Block (CLB) using CMOS technology in LTspice, and to verify its functionality by implementing a specific combinational logic function such as a multiplexer, decoder, or arithmetic unit.

Description:

A Combinational Logic Block (CLB) is a fundamental building block of digital circuits, especially in FPGA architectures. It performs logic functions based on current input values without any memory or feedback. Common implementations of CLBs include multiplexers (MUX), decoders, adders, and other arithmetic or logical operations.

In this experiment:

- A CMOS-based CLB will be designed to implement a chosen logic function (e.g., 4-to-1 MUX, full adder, or 2-bit comparator).
- The design will be simulated using LTspice, and its behavior will be analyzed for all possible input combinations.

Pre-Requisites: LTspice Tool

Pre-Lab Tasks

1. Understand the concepts of combinational logic and its difference from sequential logic.
2. Review truth tables, Karnaugh maps (K-maps), and Boolean algebra for simplifying logic functions.
3. Select a combinational logic function such as:
    - 4-to-1 multiplexer
    - 2-to-4 decoder
    - Full adder
    - 2-bit magnitude comparator

In-Lab:

1. Design the CLB in LTspice:
    - Open LTspice and create a new schematic.
    - Implement the CLB using CMOS transistors:
        - Use CMOS Gates (e.g., NAND, NOR) to realize the simplified Boolean expressions.
        - For more complex functions, use hierarchical design with modular sub-circuits.
    - Label all input and output nodes for clarity.
2. Apply Inputs:
    - Configure input sources (pulse generators or voltage sources) to simulate all possible input combinations for the chosen CLB function.
3. Run Simulation:
    - Add a transient analysis directive (e.g., .tran 1u 100u) to simulate the dynamic behavior of the circuit.
    - Run the simulation and observe the output waveforms for each input combination.
4. Validate Results:
    - Verify that the simulated outputs match the expected truth table or logic function.
5. Performance Analysis:
    - Measure propagation delay, power consumption, and switching characteristics of the CLB.
    - Use LTspice probes to analyze current and voltage at different nodes.

Obtained Wave Form /Obseravtions:

Paste your Output screenshot obtained from EDA Tool(In Colour Print)

Inference and Analysis:

Sample VIVA-VOCE Questions (In-Lab):

1. **What is a Combinational Logic Block (CLB), and how is it used in digital systems?**

2. **How do CMOS transistors implement basic logic gates?**

3. **What are the primary factors affecting the performance of a CLB?**

4. **Why are truth tables important in the design of CLBs?**

5. **How would you optimize a CMOS circuit for low power consumption?**

Post-Lab:

1. Compare the simulated outputs with the truth table for the chosen logic function.
2. Calculate the propagation delay and observe any glitches in the output.
3. Discuss the power consumption of the circuit and factors influencing it.

Post Lab Result and Outcome:

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Exp12: Design and Analysis of Half Adder CLB

Aim/Objective:

To design and analyze a Half Adder using CMOS logic in LTspice, simulate its operation, and verify its functionality in performing basic arithmetic addition of two binary bits.

Description:

A Half Adder is a basic combinational logic circuit that adds two binary inputs (A and B) and generates two outputs:

1. **Sum (S)**: Represents the least significant bit (LSB) of the addition.
2. **Carry (C)**: Represents the carry-out bit when both inputs are 1.

The Half Adder uses an XOR Gate to calculate the Sum and an AND Gate to calculate the Carry. In CMOS implementation, the XOR and ANDGates are constructed using NMOS and PMOS transistors. The functionality of the Half Adder can be tested for all input combinations using LTspice.

Logic Equations

- **Sum (S):** S=A⊕BS = A \oplus BS=A⊕B
- **Carry (C):** C=A·BC = A \cdot BC=A·B

Truth Table

A	B	Sum (S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Pre-Requisites: LTspice Tool

Pre-Lab:

1. Understand how XOR and ANDGates are implemented using CMOS transistors.
2. Study the pull-up and pull-down networks in CMOS circuits.
3. Sketch the circuit diagram for the Half Adder, showing the interconnections between XOR and ANDGates.

In-Lab Procedure

Design the Half Adder in LTspice:

Create a new schematic in LTspice.

Implement the XOR Gate using CMOS transistors:

Combine NMOS and PMOS transistors to construct a complementary logic circuit for

$A \oplus B$

Implement the AND Gate using CMOS transistors:

Design a pull-up and pull-down network for

$A \cdot B$

Connect the XOR Gate output to the Sum (S) node and the AND Gate output to the Carry (C) node.

Apply Inputs:

Use pulse voltage sources to generate binary input combinations for A and B.

Configure the pulses to toggle between 0 V (logic 0) and 5 V (logic 1).

Simulate the Circuit:

Add a transient analysis directive (e.g., .tran 1u 100u) to observe the circuit's behavior over time.

Simulate the circuit and record the output waveforms for Sum (S) and Carry (C).

Verify Outputs:

Compare the simulated outputs with the expected truth table values for all input combinations.

Performance Analysis:

Measure propagation delay and observe any glitches in the output signals.

Analyze the power consumption using LTspice's built-in measurement tools.

Obtained Wave Form :

Paste your Output screenshot obtained from EDA Tool(In Colour Print)

Sample VIVA-VOCE Questions (In-Lab):

1. What is the function of a Half Adder, and where is it used?
2. Explain the difference between a Half Adder and a Full Adder.
3. How is an XOR Gate implemented using CMOS?
4. What are the advantages of CMOS technology in logic circuit design?
5. What factors influence the propagation delay in a CMOS circuit?

Post-Lab:

1. Explore ways to reduce transistor count or improve circuit speed and efficiency.
2. Measure propagation delay for both Sum and Carry outputs.
3. Analyze power dissipation and suggest possible optimizations.

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Annexure: Suggested Projects to implement using LTspice:

Digital Circuits

1. Design and Simulation of a Low-Power 4-Bit Arithmetic Logic Unit (ALU)
2. High-Speed CMOS Full Adder Design for Arithmetic Applications
3. Implementation of a Low-Leakage SRAM Cell for VLSI Applications
4. Design and Optimization of a Dynamic Logic XOR Gate
5. Simulation of Energy-Efficient Flip-Flop Architectures for Low-Power VLSI

Analog Circuits

6. Design of a Low-Power Operational Amplifier for Mixed-Signal Applications
7. Implementation and Simulation of a High-Speed Comparator for ADCs
8. Design of a Low-Noise Differential Amplifier for Biomedical Applications
9. Simulation of a CMOS Voltage-Controlled Oscillator (VCO) for PLLs
10. Design of a Low-Dropout (LDO) Voltage Regulator for IoT Devices

Mixed-Signal Circuits

11. Design of a Delta-Sigma Modulator for Analog-to-Digital Conversion
12. Simulation of a Pipeline ADC Using CMOS Technology
13. Implementation of a Digital-to-Analog Converter (DAC) with Minimum INL/DNL Errors
14. Low-Power CMOS Phase-Locked Loop (PLL) Design for Clock Generation
15. Mixed-Signal Simulation of a Capacitance-to-Digital Converter

Power and Performance Optimization

16. Design of Energy-Efficient Clock Distribution Networks in VLSI Systems
17. Simulation of Power-Gated Circuits for Reduced Static Power Consumption
18. Design of High-Speed Multipliers with Low Switching Power
19. Optimization of Power Delivery Network (PDN) for VLSI Chips
20. Simulation of Adaptive Body Bias Circuits for Dynamic Power Management


KL
KL