# VLSI Technology (4361102) - Summer 2025 Solution

Milav Dabgar

May 12 , 2025

## Question 1(a) [3 marks]

### State importance of scaling

**Solution**

Scaling is crucial for advancing semiconductor technology and improving device performance.

**Table 1.** Scaling Benefits

| Scaling Benefits | Description |
|---|---|
| **Device Size** | Reduces transistor dimensions for higher density |
| **Speed** | Faster switching due to shorter channel length |
| **Power** | Lower power consumption per operation |
| **Cost** | More chips per wafer, reducing cost per function |

- **Technology advancement**: Enables Moore's Law continuation
- **Performance boost**: Higher frequency operation possible
- **Market competitiveness**: Smaller, faster, cheaper products

**Mnemonic**

"Small Devices Speed Progress Cheaply"

## Question 1(b) [4 marks]

### Compare Planar MOSFET and FINFET

**Solution**

FinFET technology addresses limitations of planar MOSFET at smaller nodes.

**Table 2.** Planar MOSFET vs FinFET

| Parameter | Planar MOSFET | FinFET |
|---|---|---|
| **Structure** | 2D flat channel | 3D fin-shaped channel |
| **Gate Control** | Single gate | Tri-gate/multi-gate |
| **Short Channel Effects** | High at small nodes | Significantly reduced |
| **Leakage Current** | Higher subthreshold leakage | Much lower leakage |

- **Scalability**: FinFET enables sub-22nm technology nodes
- **Power efficiency**: FinFET offers better power-performance ratio
- **Manufacturing**: FinFET requires more complex fabrication

**Mnemonic**

"Fins Control Current Better Than Flat"

# Question 1(c) [7 marks]

## Draw and Explain VDS-ID AND VGS-ID characteristics of N channel MOSFET

**Solution**

N-channel MOSFET characteristics show device behavior in different operating regions.
**Diagram:**

**Figure 1.** MOSFET Characteristics

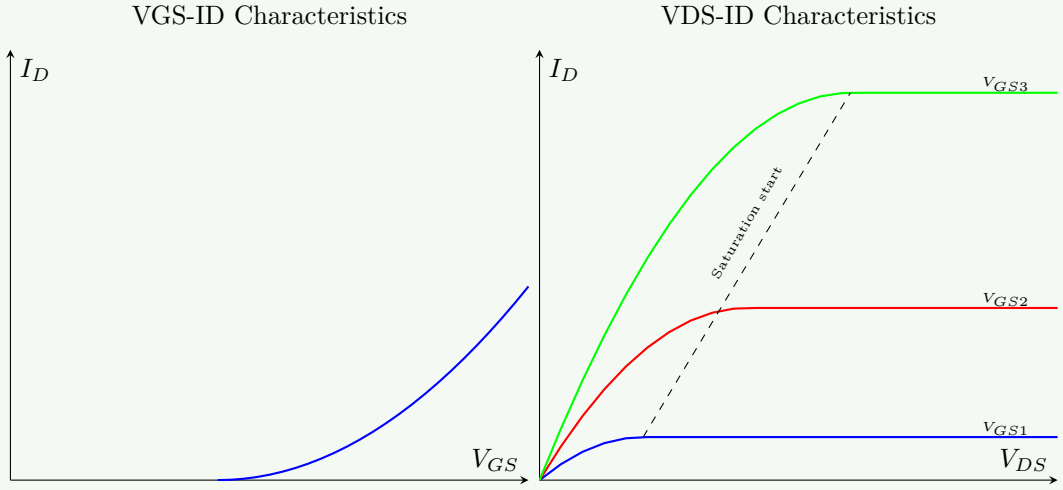



**Table 3.** MOSFET Operating Regions

| Region | Condition | Current Equation |
|---|---|---|
| **Cutoff** | $V_{GS} < V_T$ | $I_D = 0$ |
| **Linear** | $V_{DS} < (V_{GS} - V_T)$ | $I_D \propto V_{DS}$ |
| **Saturation** | $V_{DS} \geq (V_{GS} - V_T)$ | $I_D \propto (V_{GS} - V_T)^2$ |

- **Cutoff**: No current flows, acts as open switch.
- **Linear/Triode**: Current increases linearly with $V_{DS}$, acts as resistor.
- **Saturation**: Current is constant, independent of $V_{DS}$, acts as current source.

**Mnemonic**

"Threshold Gates Linear Saturation"

**OR**

## (c) Explain different condition of MOS under external bias [7 marks]

**Answer:** External bias creates different charge distributions affecting MOS capacitor behavior.

**Diagram:**

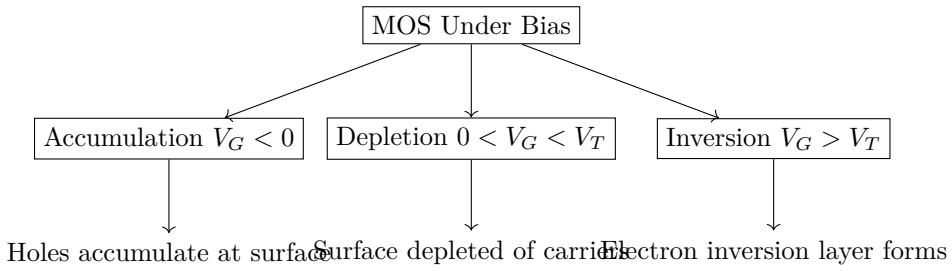



**Figure 2.** MOS Bias Modes

| Bias Condition | Surface State | Capacitance |
|---|---|---|
| **Accumulation** | Majority carriers at surface | High ($C_{ox}$) |
| **Depletion** | No mobile carriers | Medium |
| **Inversion** | Minority carriers form channel | High ($C_{ox}$) |

- **Flat band voltage**: No charge separation exists
- **Energy band bending**: Determines carrier distribution
- **Surface potential**: Controls inversion layer formation

**Mnemonic**

[Mnemonic] "Accumulate, Deplete, then Invert"

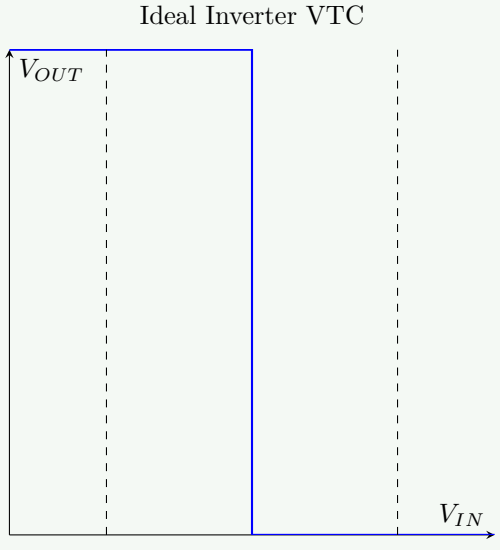
# Question 2(a) [3 marks]

## Draw voltage transfer characteristic of ideal inverter

**Solution**

Ideal inverter provides sharp transition between logic levels with infinite gain.
**Diagram:**

**Figure 3.** Ideal Inverter VTC




- **Sharp transition**: Infinite slope at switching point
- **Noise margins**: $NMH = V_{OH} - V_{IH}$, $NML = V_{IL} - V_{OL}$
- **Perfect logic levels**: $V_{OH} = V_{DD}$, $V_{OL} = 0V$

**Mnemonic**

"Sharp Switch, Perfect Levels"

## Question 2(b) [4 marks]

### Explain noise immunity and noise margin

**Solution**

Noise immunity measures circuit's ability to reject unwanted signal variations.

**Table 4.** Noise Margin Parameters

| Parameter | Definition | Formula |
|---|---|---|
| **NMH** | High-level noise margin | $V_{OH} - V_{IH}$ |
| **NML** | Low-level noise margin | $V_{IL} - V_{OL}$ |
| **Noise Immunity** | Ability to reject noise | $Min(NMH, NML)$ |

- **Logic threshold levels**: $V_{IH}$ (input high), $V_{IL}$ (input low)
- **Output levels**: $V_{OH}$ (output high), $V_{OL}$ (output low)
- **Better immunity**: Larger noise margins provide better protection
- **Design goal**: Maximize noise margins for robust operation

**Mnemonic**

"Margins Protect Against Noise"

## Question 2(c) [7 marks]

### Describe inverter circuit with saturated and linear depletion load nMOS inverter

**Solution**

Depletion load nMOS inverters use depletion transistor as active load resistor.
**Diagram:**

**Figure 4.** Depletion Load Inverter

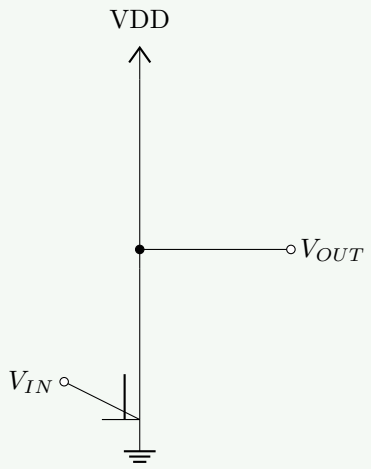


**Table 5.** Load Types

| Load Type | Gate Connection | Operation |
|---|---|---|
| **Saturated Load** | $V_G = V_D$ | Always in saturation |
| **Linear Load** | $V_G = V_{DD}$ | Can operate in linear region |

- **Depletion device**: Conducts with $V_{GS} = 0$, acts as current source
- **Load line analysis**: Determines operating point intersection
- **Power consumption**: Always conducting, higher static power
- **Switching speed**: Faster pull-down than pull-up

**Mnemonic**

"Depletion Loads Drive Outputs"

**OR**

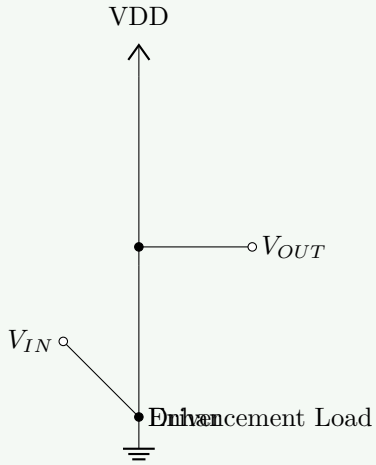
## Question 2(a) [3 marks]

### Draw and explain enhancement load inverter

**Solution**

Enhancement load inverter uses enhancement MOSFET as load with special biasing.
**Diagram:**

**Figure 5.** Enhancement Load Inverter




- **Bootstrap connection**: Gate connected to drain for load
- **Limited output high**: $V_{OUT(max)} = V_{DD} - V_T$
- **Threshold loss**: Enhancement load causes voltage drop

**Mnemonic**

"Enhancement Loses Threshold"

## Question 2(b) [4 marks]

### List the advantages of CMOS inverter

**Solution**

CMOS technology offers superior performance compared to NMOS inverters.

**Table 6.** CMOS Advantages

| Advantage | Benefit |
|---|---|
| **Zero static power** | No current path in steady state |
| **Rail-to-rail output** | Full $V_{DD}$ and 0V output levels |
| **High noise immunity** | Large noise margins |
| **Symmetric switching** | Equal rise and fall times |

- **Power efficiency**: Only dynamic power during switching
- **Scalability**: Works well at all technology nodes
- **Fan-out capability**: Can drive multiple inputs
- **Temperature stability**: Performance less sensitive to temperature

**Mnemonic**

"CMOS Saves Power Perfectly"

# Question 2(c) [7 marks]

## Draw and Explain operating mode of region for CMOS Inverter

**Solution**

CMOS inverter operation involves five distinct regions based on input voltage.
**Diagram:**

**Figure 6.** CMOS Operation Regions

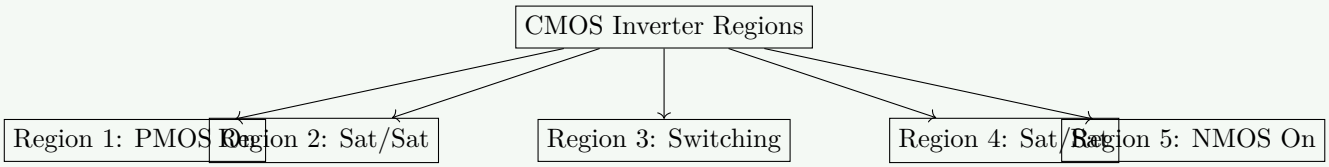



**Table 7.** CMOS Operating Regions

| Region | NMOS State | PMOS State | Output |
|---|---|---|---|
| **1** | OFF | Linear | $V_{OH} \approx V_{DD}$ |
| **2** | Saturation | Saturation | Transition |
| **3** | Saturation | Saturation | $V_{DD}/2$ |
| **4** | Saturation | Saturation | Transition |
| **5** | Linear | OFF | $V_{OL} \approx 0V$ |

- **Switching threshold**: VTC crosses $V_{DD}/2$ at region 3
- **Current flow**: Only during transition regions 2,3,4
- **Noise margins**: Regions 1 and 5 provide immunity
- **Gain**: Maximum in region 3 (switching point)

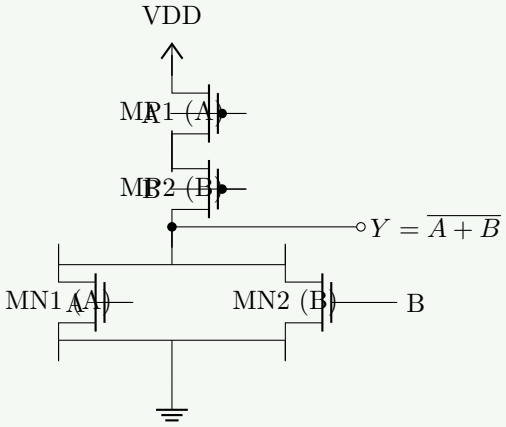
# Question 3(a) [3 marks]

## Draw two input NOR gate using CMOS

**Solution**

CMOS NOR gate implements De Morgan's law using complementary networks.
**Diagram:**

**Figure 7.** CMOS NOR2 Gate




- **Pull-up network**: Series PMOS transistors (A AND B both low for high output)
- **Pull-down network**: Parallel NMOS transistors (A OR B high for low output)
- **Logic function**: $Y = (A + B)' = A' \cdot B'$

**Mnemonic**

"Series PMOS, Parallel NMOS"

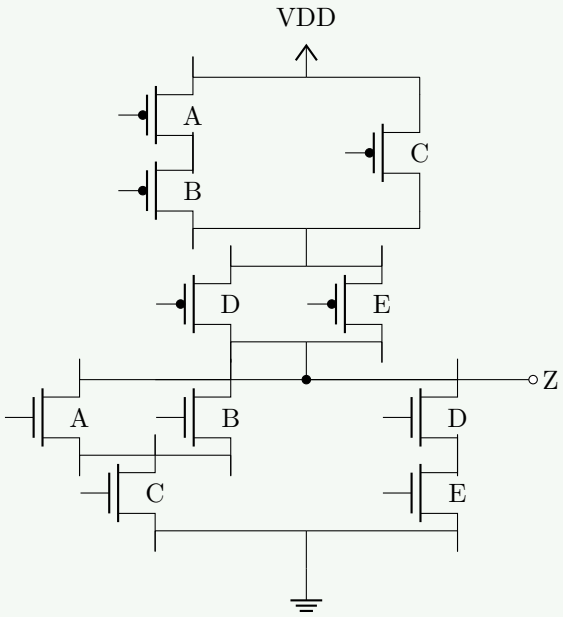
# Question 3(b) [4 marks]

## Implement Boolean function Z= [(A+B)C+DE]' using CMOS

**Solution**

Complex CMOS logic uses AOI (AND-OR-Invert) structure for efficient implementation.
**Diagram:**

**Figure 8.** CMOS Implementation of $Z = [(A + B)C + DE]'$




- **AOI structure**: Efficient single-stage implementation
- **Dual networks**: Complementary pull-up and pull-down
- **Logic optimization**: Fewer transistors than separate gates

**Mnemonic**

"AOI Inverts Complex Logic Efficiently"

## Question 3(c) [7 marks]

**Draw and explain CMOS NAND2 gate with the parasitic device capacitances**

**Solution**

Parasitic capacitances in CMOS gates affect switching speed and power consumption.
**Diagram:**

**Figure 9.** CMOS NAND2 with Parasitics

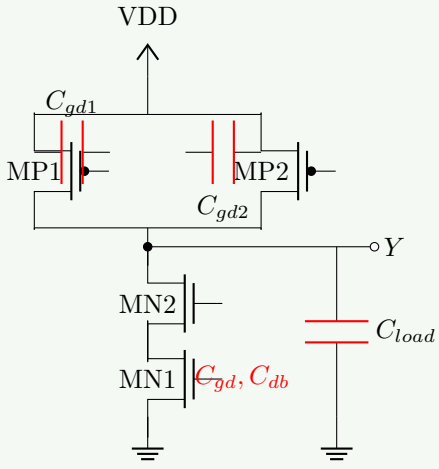



**Table 8.** Parasitic Capacitances

| Capacitance | Location | Effect |
|---|---|---|
| **Cgs** | Gate-Source | Input capacitance |
| **Cgd** | Gate-Drain | Miller effect |
| **Cdb** | Drain-Bulk | Output loading |
| **Csb** | Source-Bulk | Source loading |

- **Switching delay**: Parasitic capacitances slow transitions
- **Power consumption**: Charging/discharging parasitic caps
- **Miller effect**: Cgd creates feedback, slows switching
- **Layout optimization**: Minimize parasitic capacitances

**Mnemonic**

"Parasitics Slow Gates Down"

**OR**

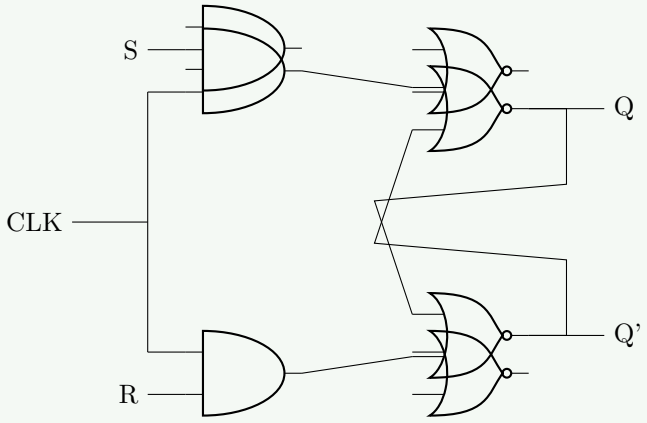
## Question 3(a) [3 marks]

### Draw and explain NOR based Clocked SR latch using CMOS

**Solution**

Clocked SR latch uses NOR gates with clock enable for synchronous operation.
**Diagram:**

**Figure 10.** Clocked SR Latch




- **Clock control**: S and R effective only when CLK = 1
- **Transparent mode**: Output follows input when clock active
- **Hold mode**: Output maintains state when clock inactive
- **Basic building block**: Foundation for flip-flops

**Mnemonic**

"Clock Controls Transparent Latching"

## Question 3(b) [4 marks]

### Implement Boolean function Z=[AB+C(D+E)]' using CMOS

**Solution**

This function implements inverted sum-of-products using AOI logic structure.
**Logic Analysis:** $Z = [AB + C(D + E)]' = [AB + CD + CE]'$
**Diagram:**

**Figure 11.** CMOS Implementation of $Z = [AB + C(D + E)]'$

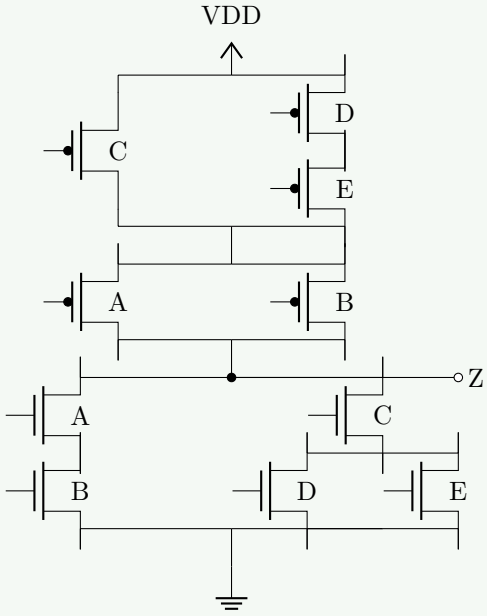



**Table 9.** Boolean Terms

| Term | Inputs | Function |
|---|---|---|
| **Term 1** | A, B | AB |
| **Term 2** | C, D | CD |
| **Term 3** | C, E | CE |
| **Output** | All terms | (AB + CD + CE)' |

- **AOI implementation**: Single stage, efficient design
- **Transistor count**: Fewer than separate gate implementation
- **Performance**: Fast switching, low power

**Mnemonic**

"Three AND Terms Feed One NOR"

# Question 3(c) [7 marks]

## Differentiate AOI and OAI Logic with example

**Solution**

AOI and OAI are complementary logic families for efficient CMOS implementation.

**Table 10.** AOI vs OAI

| Parameter | AOI (AND-OR-Invert) | OAI (OR-AND-Invert) |
|---|---|---|
| **Structure** | AND gates → OR → Invert | OR gates → AND → Invert |
| **Function** | $(AB + CD + ...)'$ | $((A + B)(C + D)...)'$ |
| **PMOS Network** | Series-parallel | Parallel-series |
| **NMOS Network** | Parallel-series | Series-parallel |

**AOI Example:** $Y = (AB + CD)'$
- PMOS: Series A-B in parallel with Series C-D
- NMOS: Parallel A,B in series with Parallel C,D

**OAI Example:** $Y = ((A + B)(C + D))'$
- PMOS: Parallel A,B in series with Parallel C,D
- NMOS: Series A-B in parallel with Series C-D
- **Design choice**: Select based on Boolean function form
- **Optimization**: Minimizes transistor count and delay
- **Duality**: AOI and OAI are De Morgan duals

**Mnemonic**

"AOI ANDs then ORs, OAI ORs then ANDs"

## Question 4(a) [3 marks]

### Define: 1) Regularity 2) Modularity 3) Locality

**Solution**

Design hierarchy principles essential for managing VLSI complexity and ensuring successful implementation.

**Table 11.** Design Principles

| Principle | Definition | Benefit |
|---|---|---|
| **Regularity** | Repeated use of similar structures | Easier layout, testing |
| **Modularity** | Breaking design into smaller blocks | Independent design, reuse |
| **Locality** | Interconnections mostly local | Reduced routing complexity |

- **Design efficiency**: Principles reduce design time and effort
- **Verification**: Modular approach simplifies testing
- **Scalability**: Enables larger, more complex designs

**Mnemonic**

"Regular Modules Stay Local"

## Question 4(b) [4 marks]

### Implement SR latch (NAND gate) using CMOS inverter

**Solution**

SR latch using NAND gates provides set-reset functionality with active-low inputs.
**Diagram:**

**Figure 12.** NAND SR Latch

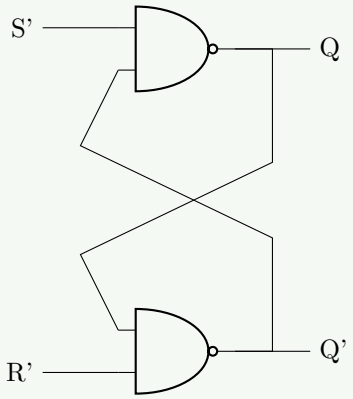



**Table 12.** SR Latch Truth Table

| S' | R' | Q | Q' | State |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | Set |
| 1 | 0 | 0 | 1 | Reset |
| 1 | 1 | Q | Q' | Hold |
| 0 | 0 | 1 | 1 | Invalid |

- **Cross-coupled structure**: Provides memory function
- **Active-low inputs**: S' = 0 sets, R' = 0 resets
- **Forbidden state**: Both inputs low simultaneously

**Mnemonic**

"Cross-Coupled NANDS Remember State"

# Question 4(c) [7 marks]

## Explain VLSI design flow

**Solution**

VLSI design flow follows systematic steps from specification to fabrication.
**Diagram:**

**Figure 13.** VLSI Design Flow

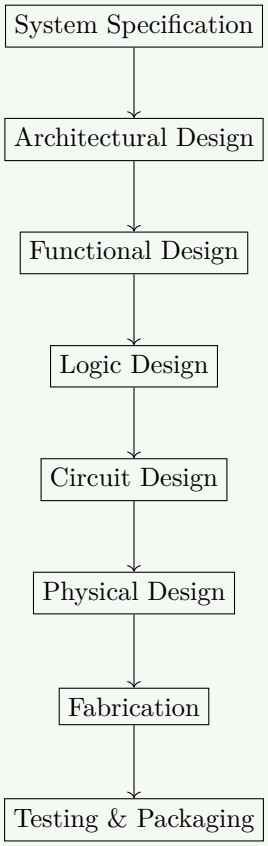



**Table 13.** Design Steps

| Level | Activities | Output |
|---|---|---|
| **System** | Requirements analysis | Specifications |
| **Architecture** | Block-level design | System architecture |
| **Logic** | Boolean optimization | Gate netlist |
| **Circuit** | Transistor sizing | Circuit netlist |
| **Physical** | Layout, routing | GDSII file |

- **Design verification**: Each level requires validation
- **Iteration**: Feedback loops for optimization
- **CAD tools**: Automation essential for complex designs
- **Time-to-market**: Efficient flow reduces design cycle

**Mnemonic**

"System Architects Love Circuit Physical Fabrication"

**OR**

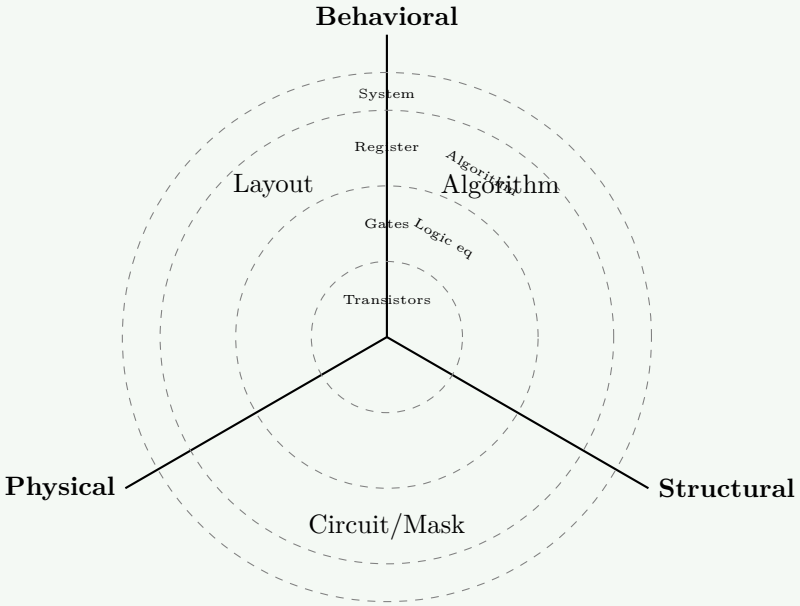
## Question 4(a) [3 marks]

**Draw and explain Y-chart**

**Solution**

Y-chart represents three design domains and their abstraction levels in VLSI design.
**Diagram:**

**Figure 14.** Gajski-Kuhn Y-Chart




- **Three domains**: Behavioral (function), Structural (components), Physical (geometry)
- **Abstraction levels**: System → Algorithm → Gate → Circuit → Layout
- **Design methodology**: Move between domains at same abstraction level

**Mnemonic**

"Behavior, Structure, Physics at All Levels"

# Question 4(b) [4 marks]

## Implement clocked JK latch (NOR gate) using CMOS inverter

**Solution**

JK latch eliminates forbidden state of SR latch with toggle capability.
**Diagram:**

**Figure 15.** Clocked JK Latch

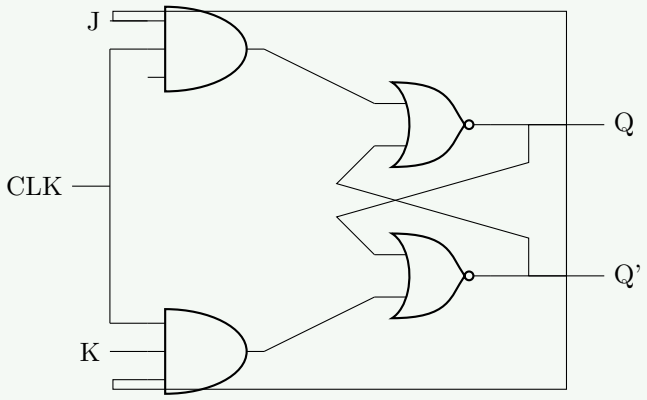



**Table 14.** JK Latch Truth Table

| J | K | Q(next) | Operation |
|---|---|---|---|
| 0 | 0 | Q | Hold |
| 0 | 1 | 0 | Reset |
| 1 | 0 | 1 | Set |
| 1 | 1 | Q' | Toggle |

- **Toggle mode**: J=K=1 flips output state
- **Clock enable**: Active only when CLK=1
- **Feedback**: Uses current output to enable inputs

**Mnemonic**

"JK Toggles, No Forbidden State"

## Question 4(c) [ Explain the terms Lithography, Etching, Deposition, Oxidation, Ion implantation, Diffusion marks]

**7**

**Solution**

Semiconductor fabrication processes essential for creating integrated circuits.

**Table 15.** Fabrication Processes

| Process | Purpose | Method |
|---|---|---|
| **Lithography** | Pattern transfer | UV exposure through masks |
| **Etching** | Material removal | Wet/dry chemical processes |
| **Deposition** | Layer addition | CVD, PVD, sputtering |
| **Oxidation** | Insulator growth | Thermal/plasma oxidation |
| **Ion Implantation** | Doping introduction | High-energy ion bombardment |
| **Diffusion** | Dopant distribution | High-temperature spreading |

- **Pattern definition**: Lithography creates device features
- **Selective removal**: Etching removes unwanted material
- **Layer building**: Deposition adds required materials
- **Doping control**: Implantation and diffusion create junctions
- **Quality control**: Each step affects final device performance

**Mnemonic**

"Light Etches Deposited Oxides, Ions Diffuse"

# Question 5(a) [3 marks]

### Implement 2 input XNOR gate using Verilog

**Solution**

XNOR gate produces high output when inputs are equal.

**Listing 1.** Verilog Code for XNOR Gate

```
module xnor_gate(
    input a, b,
    output y
);
    assign y = ~(a ^ b);
endmodule
```

- **Logic function**: $Y = (A \oplus B)' = A'B' + AB$
- **High Output**: When both inputs are 0 or both are 1
- **Applications**: Equality comparison, parity checking

**Mnemonic**

"XNOR Equals Equal Inputs"

# Question 5(b) [4 marks]

### Implement Encoder (8:3) using CASE statement in Verilog

**Solution**

Priority encoder converts 8-bit input to 3-bit binary output using case statement for behavioral modeling.

**Listing 2.** 8:3 Priority Encoder using CASE

```
module encoder_8to3(
    input [7:0] in,
    output reg [2:0] out
);
    always @(*) begin
        case(in)
            8'b00000001: out = 3'b000;
            8'b00000010: out = 3'b001;
            8'b00000100: out = 3'b010;
            8'b00001000: out = 3'b011;
            8'b00010000: out = 3'b100;
            8'b00100000: out = 3'b101;
            8'b01000000: out = 3'b110;
            8'b10000000: out = 3'b111;
            default: out = 3'b000;
        endcase
    end
endmodule
```

- **Behavioral Modeling**: Case statement describes function conceptually

- **Combinational Logic**: `always @(*)` block used
- **Default case**: Handles undefined states (e.g., all zeros)

**Mnemonic**

"One Hot Input, Binary Output"

# Question 5(c) [7 marks]

### Explain CASE statement in Verilog with suitable examples

**Solution**

CASE statement provides multi-way branching based on expression value, typically used in combinational logic (decoders, muxes) and state machines.

**Syntax:**

```
case (expression)
    value1: statement1;
    value2: statement2;
    default: default_statement;
endcase
```

**Example 1: 4:1 Multiplexer**

```
module mux_4to1(
    input [1:0] sel,
    input [3:0] in,
    output reg out
);
    always @(*) begin
        case(sel)
            2'b00: out = in[0];
            2'b01: out = in[1];
            2'b10: out = in[2];
            2'b11: out = in[3];
        endcase
    end
endmodule
```

**Benefits:**

- **Readability**: Clearer than multiple `if-else` statements
- **Synthesis**: Efficiently maps to multiplexers or ROMs
- **Parallel Evaluation**: Hardware checks all conditions simultaneously

**Table 16.** CASE Statement Variants

| Variant | Syntax | Usage |
|---|---|---|
| **case** | `case(expr)` | Exact bit matching (0, 1, x, z) |
| **casex** | `casex(expr)` | Treats 'x' and 'z' as don't care |
| **casez** | `casez(expr)` | Treats 'z' as don't care |

**Mnemonic**

"CASE Chooses Actions Systematically Everywhere"