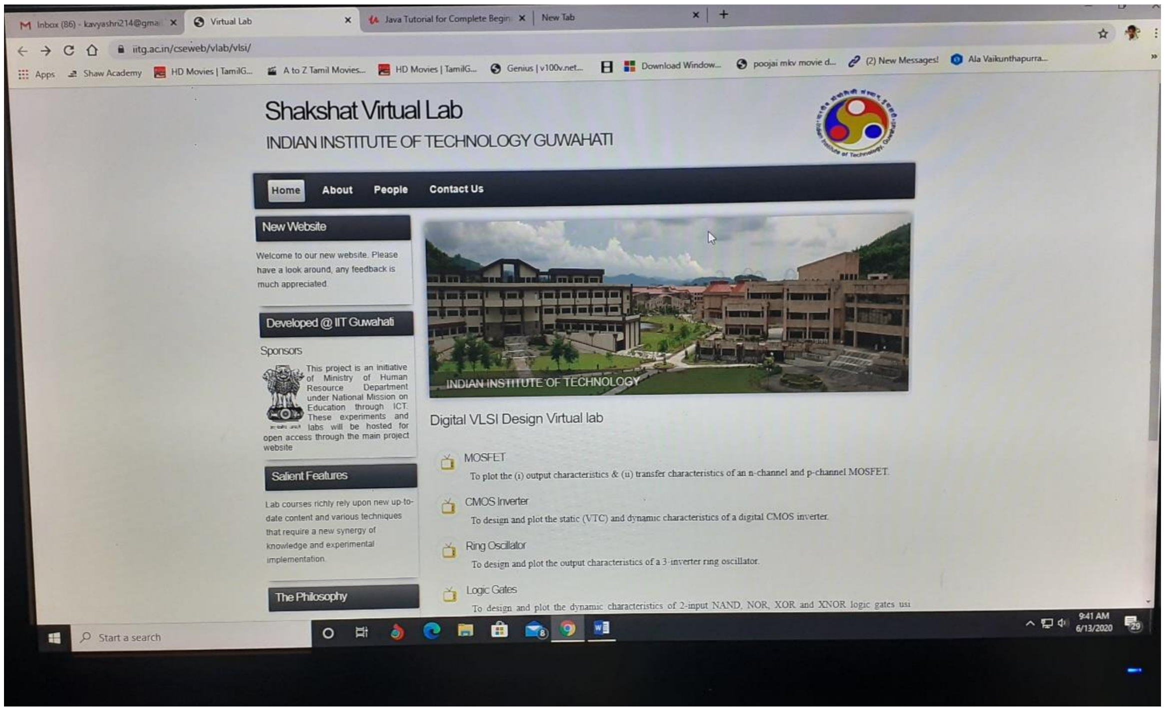
DAILY ASSESSMENT

Date:	**13-6-2020**	**Name:**	**Kavyashree m**
Course:	**VLSI**	**USN:**	**4al15ec036**
Topic:	**Digital VLSI Design Virtual lab**	**Semester & Section:**	**8th A**
Github Repository:	**kavya**		

FORENOON SESSION DETAILS



Shakshat Virtual Lab

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

New Website

Welcome to our new website. Please have a look around, any feedback is much appreciated.

Developed @ IIT Guwahati

Sponsors

This project is an initiative of Ministry of Human Resource Department under National Mission on Education through ICT. These experiments and labs will be hosted for open access through the main project website

Salient Features

Lab courses richly rely upon new up-to-date content and various techniques that require a new synergy of knowledge and experimental implementation.

The Philosophy

Digital VLSI Design Virtual lab

MOSFET
To plot the (i) output characteristics & (ii) transfer characteristics of an n-channel and p-channel MOSFET.

CMOS Inverter
To design and plot the static (VTC) and dynamic characteristics of a digital CMOS inverter.

Ring Oscillator
To design and plot the output characteristics of a 3-inverter ring oscillator.

Logic Gates
To design and plot the dynamic characteristics of 2-input NAND, NOR, XOR and XNOR logic gates usi

Digital VLSI Design Virtual lab

MOSFET

The aim of this experiment is to plot (i) the output characteristics and, (ii) the transfer characteristics of an n-channel and p-channel MOSFET.

Introduction

The metal–oxide–semiconductor field-effect transistor (MOSFET) is a transistor used for amplifying or switching electronic signals. In MOSFETs, a voltage on the oxide-insulated gate electrode can induce a conducting channel between the two other contacts called source and drain. The channel can be of n-type or p-type, and is accordingly called an nMOSFET or a pMOSFET. Figure 1 shows the schematic diagram of the structure of an nMOS device before and after channel formation.

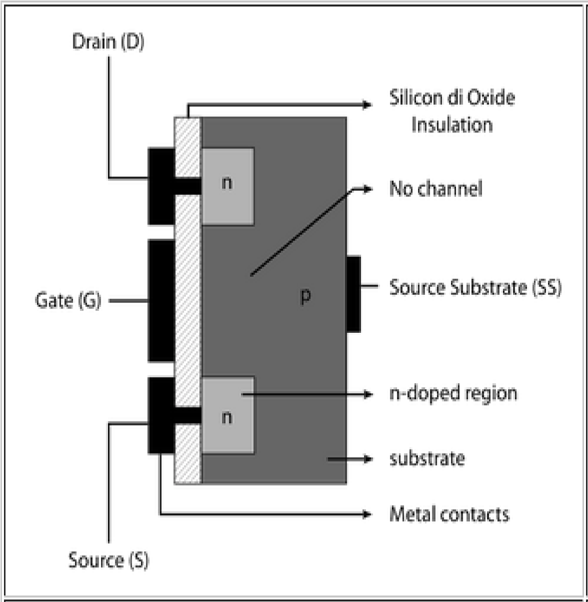


Fig. (1a): nMOSFET before channel formation

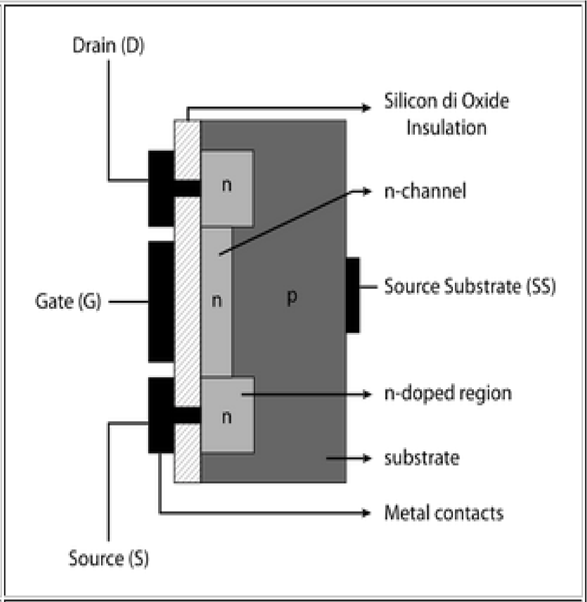


Fig. (1b): nMOSFET structure after channel formation

Figure 2 shows symbols commonly used for MOSFETs where the bulk terminal is either labeled (B) or implied (not drawn).

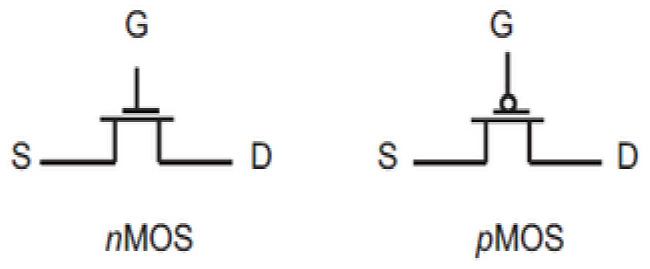


Fig. (2): Circuit symbols for nMOS and pMOS respectively

Output Characteristics

MOSFET output characteristics plot I_D versus V_{DS} for several values of V_{GS}.

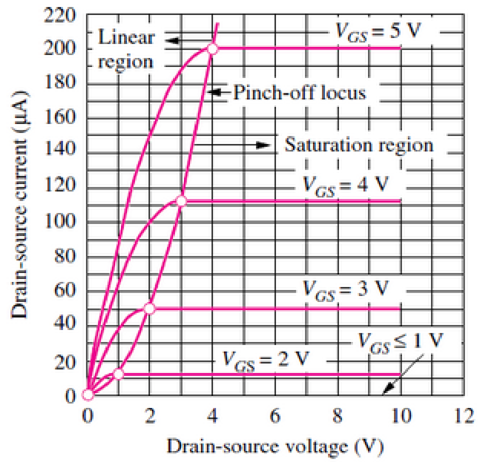


Fig. (3a): Output Characteristics of nMOS transistor

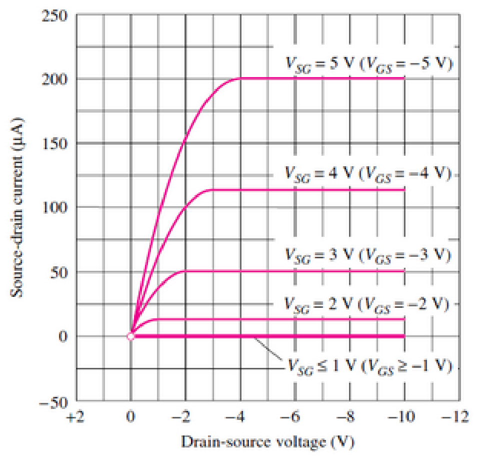


Fig. (3b): Output Characteristics of pMOS transistor

The characteristics of an nMOS transistor can be explained as follows. As the voltage on the top electrode increases further, electrons are attracted to the surface. At a particular voltage level, which we will shortly define as the threshold voltage, the electron density at the surface exceeds the hole density. At this voltage, the surface has inverted from the p-type polarity of the original substrate to an n-type inversion layer, or inversion region, directly underneath the top plate as indicated in Fig. 1(b). This inversion region is an extremely shallow layer, existing as a charge sheet directly below the gate. In the MOS capacitor, the high density of electrons in the inversion layer is supplied by the electron–hole generation process within the depletion layer. The positive charge on the gate is balanced by the combination of negative charge in the inversion layer plus negative ionic acceptor charge in the depletion layer. The voltage at which the surface inversion layer just forms plays an extremely important role in field-effect transistors and is called the threshold voltage V_{tn}. The region of output characteristics where V_{GS}tn and no current flows is called the cutt-off region. When the channel forms in the nMOS (pMOS) transistor, a positive (negative) drain voltage with respect to the source creates a horizontal electric field moving the electrons (holes) toward the drain forming a positive (negative) drain current coming into the transistor. The positive current convention is used for electron and hole current, but in both cases electrons are the actual charge carriers. If the channel horizontal electric field is of the same order or smaller than the vertical thin oxide field, then the inversion channel remains almost uniform along the device length. This continuous carrier profile from drain to source puts the transistor in a bias state that is equivalently called either the non-saturated, linear, or ohmic bias state. The drain and source are effectively short-circuited. This happens when $V_{GS} > V_{DS} + V_{tn}$ for nMOS transistor and $V_{GS} < V_{DS} + V_{tp}$ for pMOS transistor. Drain current is linearly related to drain-source voltage over small intervals in the linear bias state.

But if the nMOS drain voltage increases beyond the limit, so that VGS < VDS + Vtn, then the horizontal electric field becomes stronger than the vertical field at the drain end, creating an asymmetry of the channel carrier inversion distribution shown in Figure 4.

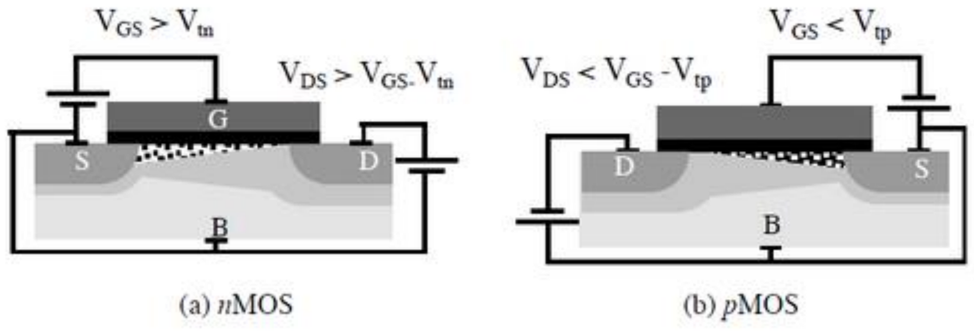


Fig. 4: Channel pinchoff for (a) nMOS and (b) pMOS transistor devices.

If the drain voltage riseswhile the gate voltage remains the same, then VGD can go below the threshold voltage in the drain region. There can be no carrier inversion at the drain-gate oxide region, so the inverted portion of the channel retracts from the drain, and no longer "touches" this terminal. The pinched-off portion of the channel forms a depletion region with a high electric field. The n-drain and p-bulk form a pn junction. When this happens the inversion channel is said to be "pinched-off" and the device is in the saturation region. The characteristics can be loosely modelled by the following equations.

$$I_D = \begin{cases} 0, & \text{Cut-off state} \\ \dfrac{\mu_n \varepsilon_{ox}}{T_{ox}} \dfrac{W}{L} \left[(V_{GS} - V_{tn}) V_{DS} - \dfrac{V_{DS}^2}{2} \right], & \text{Ohmic state} \\ \dfrac{\mu_n \varepsilon_{ox}}{2T_{ox}} \dfrac{W}{L} (V_{GS} - V_{tn})^2, & \text{Saturation state} \end{cases}$$

Transfer Characteristics

The transfer characteristic relates drain current (I_D) response to the input gate-source driving voltage (V_{GS}). Since the gate terminal is electrically isolated from the remaining terminals (drain, source, and bulk), the gate current is essentially zero, so that gate current is not part of device characteristics. The transfer characteristic curve can locate the gate voltage at which the transistor passes current and leaves the OFF-state. This is the device threshold voltage (V_{tn}). Figure 5 shows measured input characteristics for an nMOS and pMOS transistor with a small 0.1V potential across their drain to source terminals.

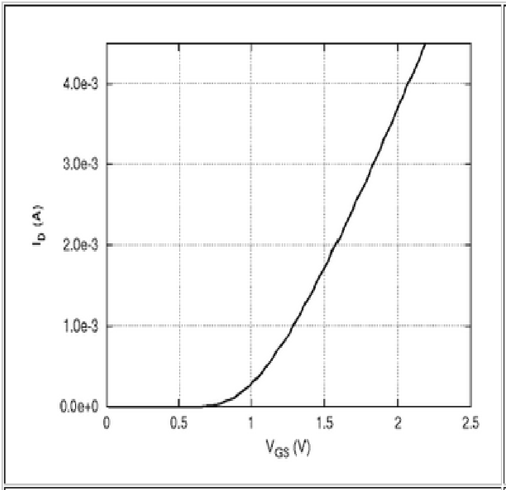


Fig. (5a): Transfer characteristics of nMOS transistor

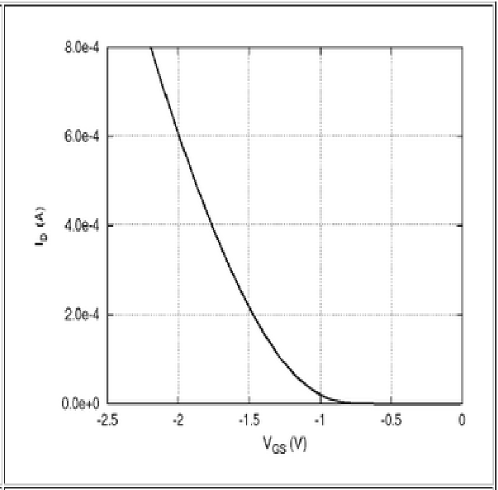


Fig. (5b): Transfer characteristics of pMOS transistor

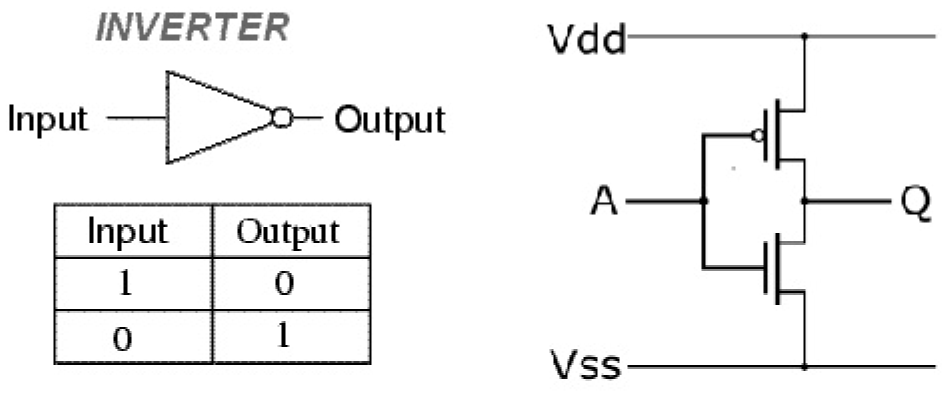
The transistors are in their non-saturated bias states. As V_{GS} increases for the nMOS transistor in Figure 5a, the threshold voltage is reached where drain current elevates. For V_{GS} between 0V and 0.7V, I_D is nearly zero indicating that the equivalent resistance between the drain and source terminals is extremely high. Once V_{GS} reaches 0.7V, the current increases rapidly with V_{GS} indicating that the equivalent resistance at the drain decreases with increasing gate-source voltage. Therefore, the threshold voltage of the given nMOS transistor is about $V_{tn} \approx 0.7V$. The pMOS transistor input characteristic in Figure 5b is analogous to the nMOS transistor except the I_D and V_{GS} polarities are reversed.

CMOS

The aim of this experiment is to design and plot the static (VTC) and dynamic characteristics of a digital CMOS inverter.

Introduction

The inverter is universally accepted as the most basic logic gate doing a Boolean operation on a single input variable. Fig.1 depicts the symbol, truth table and a general structure of a CMOS inverter. As shown, the simple structure consists of a combination of an pMOS transistor at the top and a nMOS transistor at the bottom.



Input	Output
1	0
0	1

Fig.1: Symbol, circuit structure and truth table of a CMOS inverter

CMOS is also sometimes referred to as complementary-symmetry metal–oxide–semiconductor. The words "complementary-symmetry" refer to the fact that the typical digital design style with CMOS uses complementary and symmetrical pairs of p-type and n-type metal oxide semiconductor field effect transistors (MOSFETs) for logic functions. Two important characteristics of CMOS devices are high noise immunity and low static power consumption. Significant power is only drawn while the transistors in the CMOS device are switching between on and off states. Consequently, CMOS devices do not produce as much waste heat as other forms of logic, for example transistor-transistor logic (TTL) or NMOS logic, which uses all n-channel devices without p-channel devices.

Inverter Static Characteristics (VTC)

Digital inverter quality is often measured using the Voltage Transfer Curve (VTC), which is a plot of input vs. output voltage. From such a graph, device parameters including noise tolerance, gain, and operating logic-levels can be obtained.

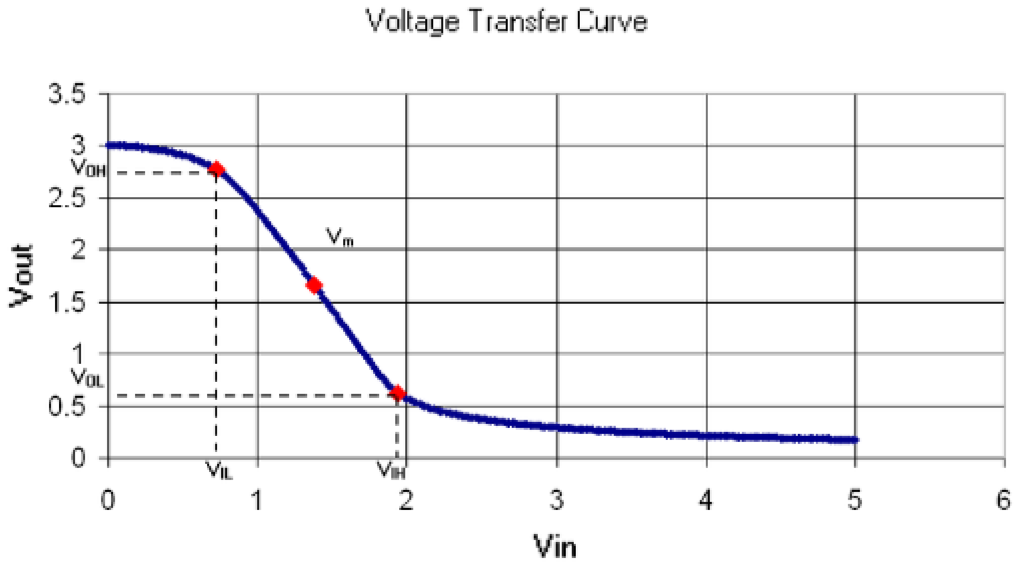


Fig.2: Voltage Transfer Curve for a typical 20 μm Inverter

Ideally, the voltage transfer curve (VTC) appears as an inverted step-function - this would indicate precise switching between on and off - but in real devices, a gradual transition region exists. The VTC indicates that for low input voltage, the circuit outputs high voltage; for high input, the output tapers off towards 0 volts. The slope of this transition region is a measure of quality - steep (close to -Infinity) slopes yield precise switching. The tolerance to noise can be measured by comparing the minimum input to the maximum output for each region of operation (on / off). This is more explicitly shown in the fig.3.

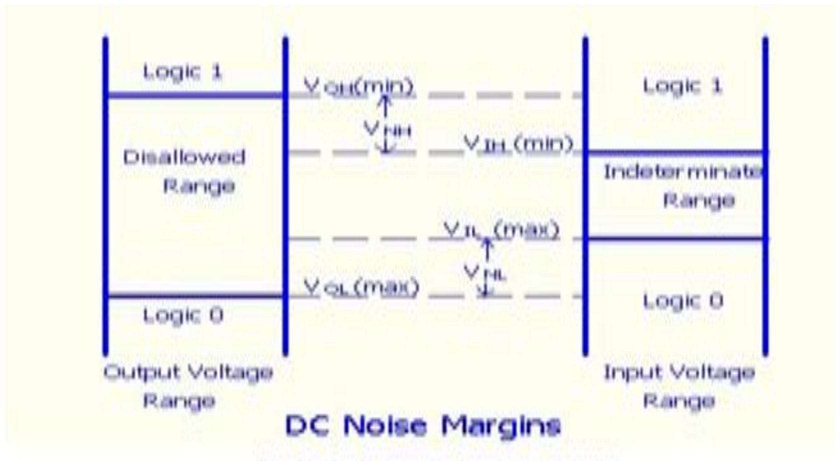


Fig.3: Definition of noise margin

Noise margin : is a parameter intimately related to the transfer characteristics. It allows one to estimate the allowable noise voltage on the input of a gate so that the output will not be affected. Noise margin (also called noise immunity) is specified in terms of two parameters - the low noise margin N_L, and the high noise margin N_H . Referring to above figure, N_L is defined as the difference in magnitude between the maximum LOW input voltage recognized by the driven gate and the maximum LOW output voltage of the driving gate. That is, $N_L = |V_{IL} - V_{OL}|$. Similarly, the value of N_H is the difference in magnitude between the minimum HIGH output voltage of the driving gate and the minimum HIGH input voltage recognizable by the driven gate. That is, $N_{MH} = |V_{OH} - V_{IH}|$. Where $V_{IH}|$: minimum HIGH input voltage, V_{IL}: maximum LOW input voltage, V_{OH}: minimum HIGH output voltage, V_{OL}: maximum LOW output voltage.

Inverter Dynamic Characteristics

Fig.4 shows the dynamic characteristics of a CMOS inverter. The following are some formal definitions of temporal parameters of digital circuits. All percentages are of the steady state values.

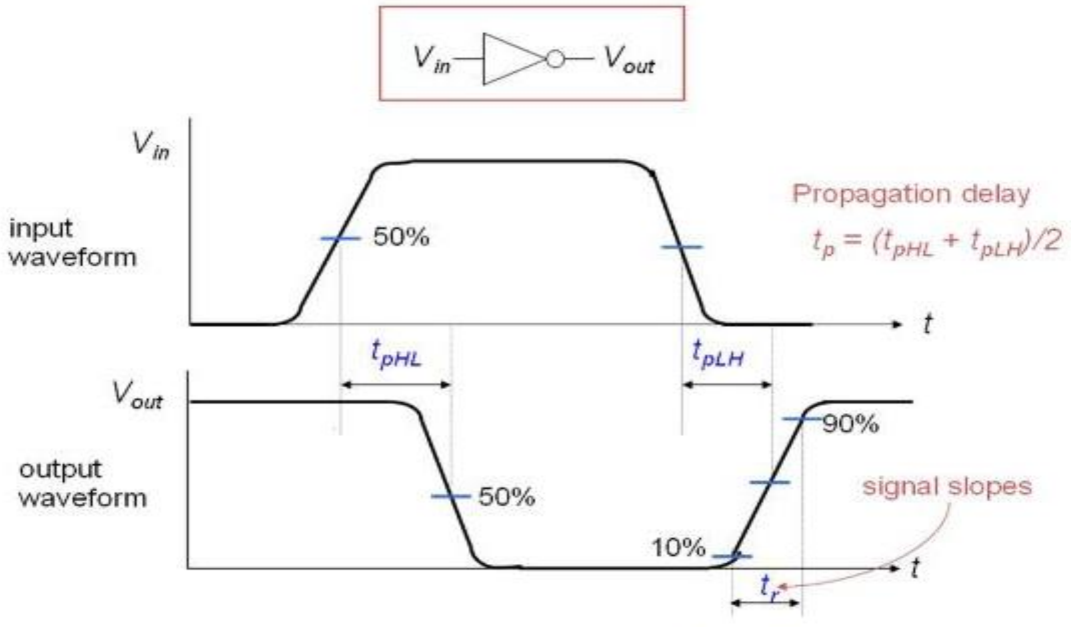


Fig.4: Dynamic characteristics of CMOS inverter

Rise Time (t_r) : Time taken to rise from 10% to 90%.

Fall Time (t_f): Time taken to fall from 90% to 10%

Edge Rate (t_{rf}): (t_r + t_f)/2.

High-to-Low propagation delay (t_{pHL}): Time taken to fall from V_{OH} to 50%.

Low-to-High propagation delay (t_{pLH}): Time taken to rise from 50% to V_{OL}.

Propagation Delay (t_p): (t_{pHL} + t_{pLH})/2.

Contamination Delay (t_{cd}): Minimum time from the input crossing 50% to the output crossing 50%

RING OSCILLATOR

The aim of this experiment is to design and plot the output characteristics of 3-inverter and 5-inverter ring oscillator.

Introduction

A ring oscillator is a device composed of an odd number of NOT gates whose output oscillates between two voltage levels, representing true and false. A schematic diagram of a simple three inverter ring oscillator is shown in Fig.1.

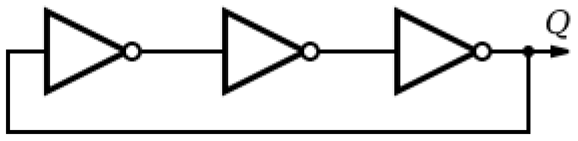


Fig.1: A 3-inverter ring oscillator.

The NOT gates, or inverters, are attached in a chain; the output of the last inverter is fed back into the first. Because a single inverter computes the logical NOT of its input, it can be shown that the last output of a chain of an odd number of inverters is the logical NOT of the first input. This final output is asserted a finite amount of time after the first input is asserted; the feedback of this last output to the input causes oscillation. A real ring oscillator only requires power to operate; above a certain threshold voltage, oscillations begin spontaneously. To increase the frequency of oscillation, two methods may be used. Firstly, the applied voltage may be increased; this increases both the frequency of the oscillation and the power consumed, which is dissipated as heat.

Operation

To understand the operation of a ring oscillator, one must first understand gate delay. In a physical device, no gate can switch instantaneously; in a device fabricated with MOSFETs, for example, the gate capacitance must be charged before current can flow between the source and the drain. Thus, the output of every inverter of a ring oscillator changes a finite amount of time after the input has changed. From here, it can be easily seen that adding more inverters to the chain increases the total gate delay, reducing the frequency of oscillation. The switching frequency at each gate is inversely proportional to both the number of gates in the ring and the gate delay of each individual gate. A typical simulation output for a ring oscillator is shown in Fig. 2.

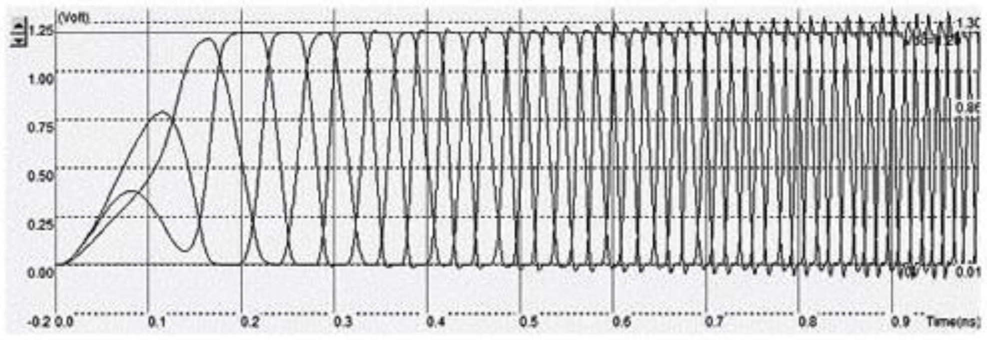


Fig.2: A typical output waveform of a 3-inverter ring oscillator.

The simulation in Fig.2 shows the "warm-up" of the inverter circuit followed by a stable frequency oscillation. The main problem of this type of oscillator is the very strong dependence of the output frequency with virtually all process parameters (like W, L etc) and operating conditions. As an example, the power supply voltage V_{DD} has a very significant importance on the oscillating frequency. The output frequency of a 3-inverter ring oscillator can be written as 1/(6×inverter delay). Thus the propagation delay of an inverter circuit can be obtained by measuring the time period of the oscillator.

LOGIC GATES

The aim of this experiment is to design and plot the dynamic characteristics of 2-input NAND, NOR, XOR and XNOR gates based on CMOS static logic.

Introduction

Static logic is a design methodology in integrated circuit design where there is at all times some mechanism to drive the output either high or low. For example, in many of the popular logic families, such as TTL and traditional CMOS, there is always a low-impedance path between the output and either the supply voltage or the ground. The most widely used logic style is static CMOS. A static CMOS gate is a combination of two networks, called the pull-up network (PUN) and the pull-down network (PDN). The function of the PUN is to provide a connection between the output and VDD anytime the output of the logic gate is meant to be 1 (based on the inputs). Similarly, the function of the PDN is to connect the output to VSS when the output of the logic gate is meant to be 0 (based on the inputs). The PUN and PDN networks are constructed in a mutually exclusive fashion such that, one and only one of these networks is conducting in the steady state.

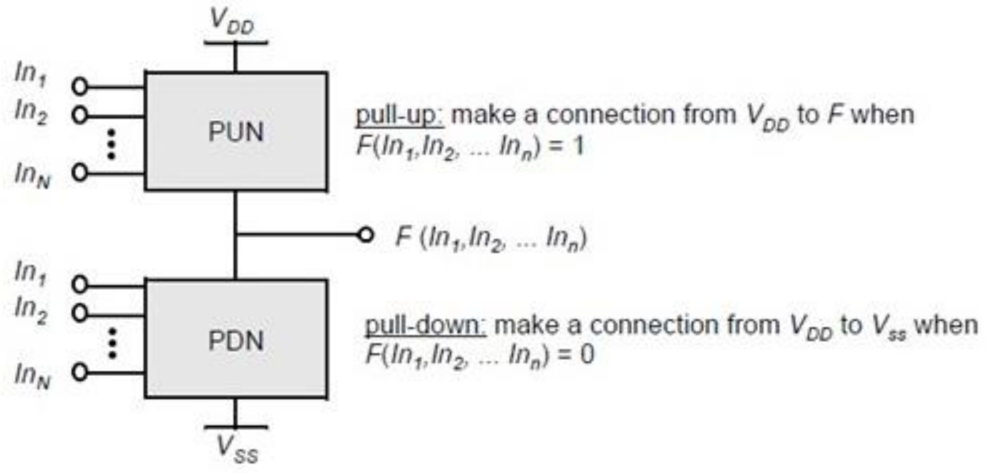
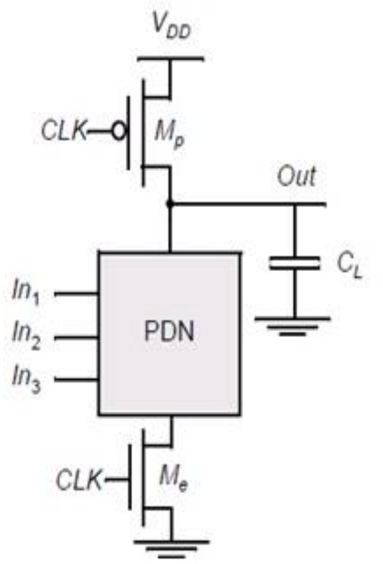

V_{DD}
In_1
In_2
In_N
PUN
pull-up: make a connection from V_{DD} to F when $F(In_1, In_2, \ldots In_n) = 1$
$F(In_1, In_2, \ldots In_n)$
PDN
pull-down: make a connection from V_{DD} to V_{ss} when $F(In_1, In_2, \ldots In_n) = 0$
V_{SS}

Fig.1: Complementary logic gate as a combination of a PUN (pull-up network) and a PDN (pull-down network).

Dynamic logic is a design methodology in integrated circuit design in that it uses a clock signal in its implementation of combinational logic circuits. In dynamic logic, there is not always a mechanism driving the output high or low. In the most common version of this concept, the output is driven high or low during distinct parts of the clock cycle. Dynamic logic requires a minimum clock rate fast enough that the output state of each dynamic gate is used before it leaks out of the capacitance holding that state. The basic construction of a dynamic logic gate is shown in fig.2. The PDN (pull-down network) is constructed exactly as in complementary CMOS. The operation of this circuit is divided into two major phases: precharge and evaluation, with the mode of operation determined by the clock signal CLK.



Phase	Clk	Inputs	Outputs
Precharge	Low	Don't care	High
Evaluation	High	Valid	Valid

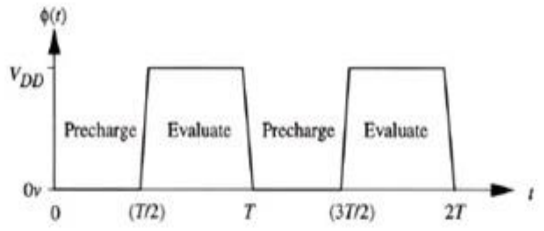


Fig.2: Basic concepts of a dynamic gate: Two-phase operation and clocking sequence for precharge/evaluate logic.

Precharge: When CLK = 0, the output node Out is precharged to V_{DD} by the PMOS transistor M_p. During that time, the evaluate NMOS transistor M_e is off, so that the pull-down path is disabled. The evaluation FET eliminates any static power that would be consumed during the precharge period (this is, static current would flow between the supplies if both the pulldown and the precharge device were turned on simultaneously).

Evaluation: For CLK = 1, the precharge transistor M_p is off, and the evaluation transistor M_e is turned on. The output is conditionally discharged based on the input values and the pull-down topology. If the inputs are such that the PDN conducts, then a low resistance path exists between Out and GND and the output is discharged to GND. If the PDN is turned off, the precharged value remains stored on the output capacitance C_L, which is a combination of junction capacitances, the wiring capacitance, and the input capacitance of the fan-out gates. During the evaluation phase, the only possible path between the output node and a supply rail is to GND. Consequently, once Out is discharged, it cannot be charged again till then next precharge operation. The inputs to the gate can therefore make at most one transition during evaluation.

Static Logic Design of NAND, NOR, XOR and XNOR Gates

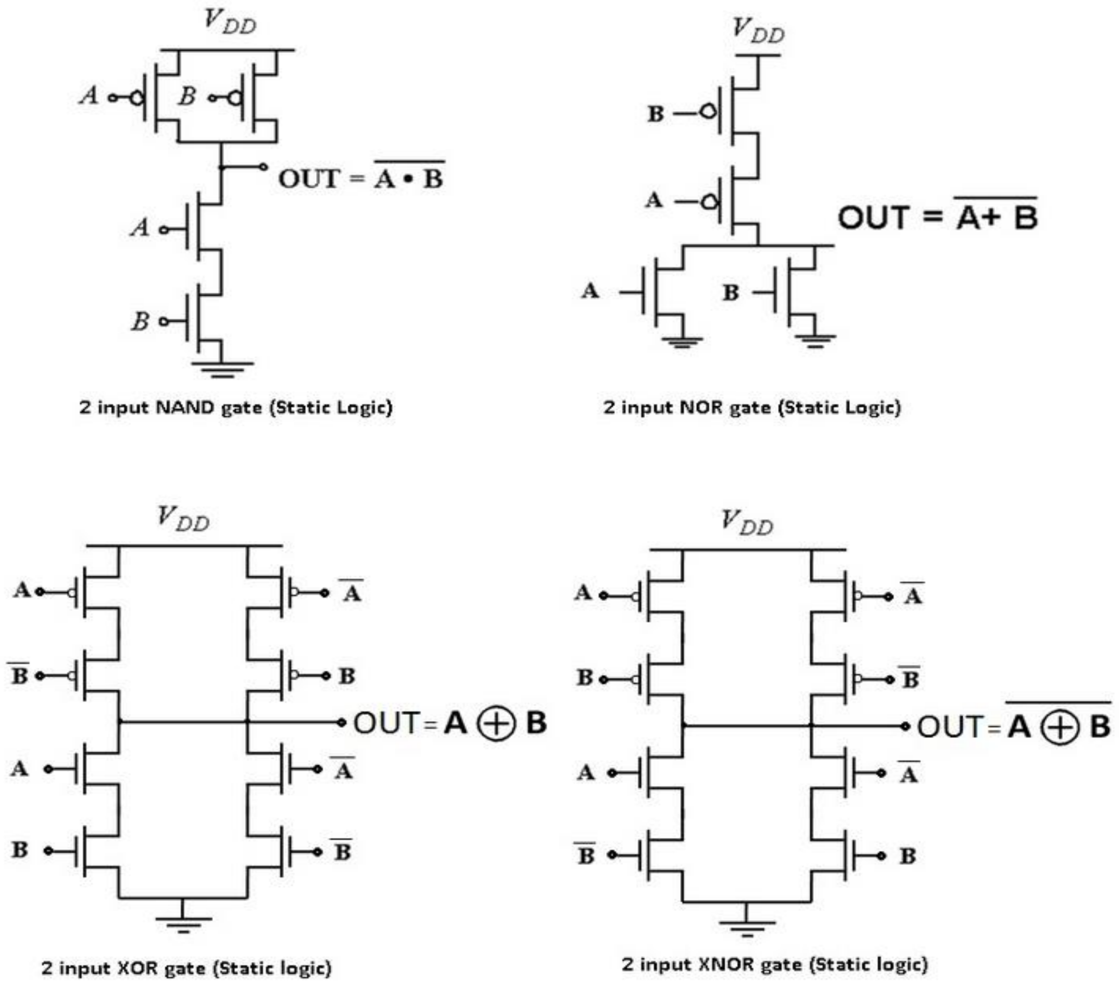

V_{DD}
A
B
OUT = $\overline{A \bullet B}$
2 input NAND gate (Static Logic)
OUT = $\overline{A + B}$
2 input NOR gate (Static Logic)
OUT= $A \oplus B$
$\overline{A}$
$\overline{B}$
2 input XOR gate (Static logic)
OUT= $\overline{A \oplus B}$
2 input XNOR gate (Static logic)

Fig.3: The circuit diagram for 2-input NAND, NOR, XOR and XNOR gates in CMOS static logic.

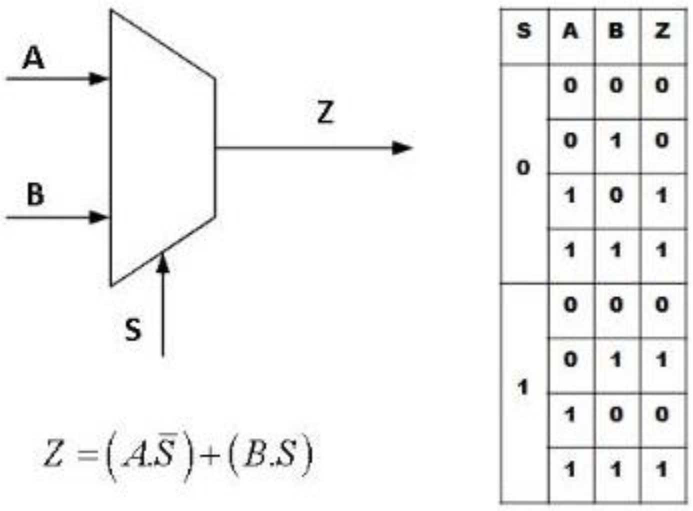
In order to design 2-input NAND, NOR, XOR and XNOR gates for equal rise and fall time, it is necessary to first design an inverter with equal rise and fall time. This involves compensating for the difference in electron and hole mobilities. For silicon material, the electron mobility is about 2.5 to 3 times greater than the hole mobility. Therefore, to have equal rise tand fall time in an inverter, we must choose the W/L ration of pMOS as 2.5 times greater than that of the nMOS transistor. After performing this task, we need to size the transistors of each gate under worst case conditions (of input combination) for charging and discharging resistances R_c and R_d. (In every gate circuit, the PUN provides maximum ON resistance for rise time and the PDN provides maximum ON resistance for fall time.) For a NAND gate, the worst case charging corresponds to an input combination where only one of the pMOS is ON and discharging takes place only when both nMOS' are turned ON. i.e. in the worst case, $R_c/R_d=1/2$. Thus, in order to equalize both currents (considering also the mobility defferences), we must have (W/L)p=(2.5*2)(W/L)n. This can be achieved in a 180nm technology by choosing $W_n=0.18$ µm and $W_p=0.90$ µm. Similary in case of a NOR gate, (W/L)p must be equal to (2.5*0.5)(W/L)n which can be achieved by taking $W_n=0.36$µm and $W_p=0.45$µm. For XOR and XNOR gates, worst case R_c/R_d ratio is equal to one. Therefore, (W/L)p must be equal to (2.5*1)(W/L)n for both gates.

MULTIPLEXER

The aim of this experiment is to design and plot the characteristics of a 4x1 digital multiplexer using pass transistor and transmission gate logic.

Introduction

A multiplexer or mux is a combinational circuits that selects several analog or digital input signals and forwards the selected input into a single output line. A multiplexer of 2^n inputs has n selected lines, are used to select which input line to send to the output.

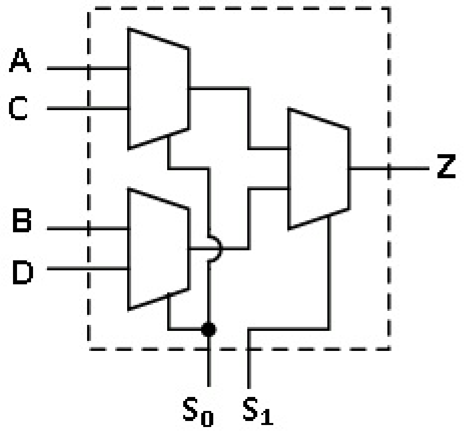


$$Z=(A.\bar{S})+(B.S)$$

S	A	B	Z
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

Fig.1: The schematic diagram, boolean equation and the truth table of a 2:1 multiplexer with inputs ***A*** and ***B***, select input ***S*** and the output ***Z***.

Figure 2 shows how a 4:1 MUX can be constructed out of two 2:1 MUXs.



$$Z = \left(A.\overline{S}_0.\overline{S}_1\right) + \left(B.\overline{S}_0.S_1\right) + \left(C.S_0.\overline{S}_1\right) + \left(D.S_0.S_1\right)$$

Fig.2: Implementation of 4:1 MUX using 2:1 MUXs

Design using pass-transistor logic

A multiplexer can be designed using various logics. Fig.3 shows how a 2:1 MUX is implemented using a pass-transistor logic.GS.

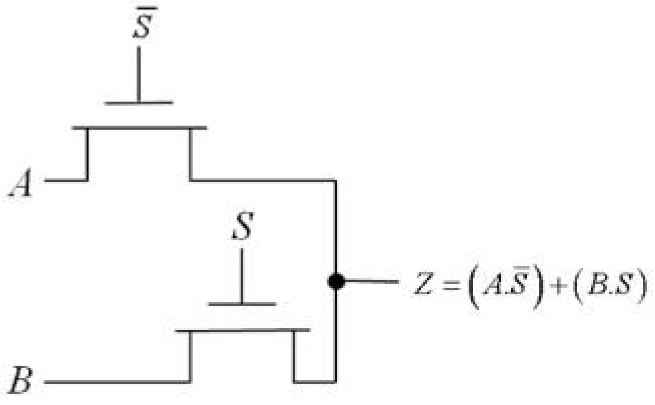


Fig.3. Design of a 2:1 MUX using pass-transistor logic

The pass-transistor logic attempts to reduce the number of transistors to implement a logic by allowing the primary inputs to drive gate terminals as well as source-drain terminals. The implementation of a 2:1 MUX requires 4 transistors (including the inverter required to invert S), while a complementary CMOS implementation would require 6 transistors. The reduced number of devices has the additional advantage of lower capacitance.

Design using transmission gate logic

A transmission gate is an electronic element and good non mechanical relay built with CMOS technology. It is made by parallel combination of nMOS and pMOS transistors with the input at the gate of one transistor (C) being complementary to the input at the gate () of the other. The symbol of a transmission gate is shown below in fig.4.

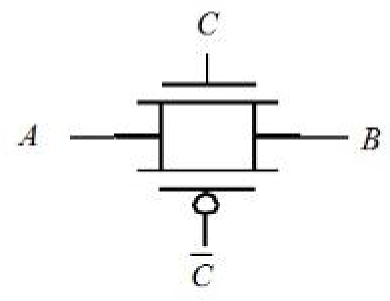


Fig.4: Symbol for tranmission gate

The transmission gate acts as a bidirectional switch controlled by the gate signal C. When C=1, both MOSFETs are on, allowing the signal to pass through the gate. In short, A=B, if C=1. On the other hand, C=0, places both transistors in cut-off, creating an open circuit between nodes A and B. Fig.5 shows the implementation of a 2:1 MUX using transmission gate logic.

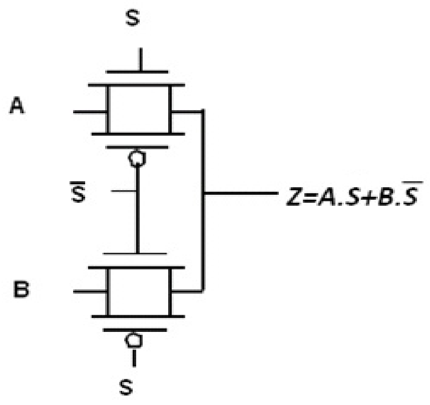


Fig.5: Circuit diagram of a 2:1 MUX using transmission gate logic

Here, the transmission gates selects input A or B on the basis of the value of the control signal S. When S=0, Z=A and when S=1, Z=B.

LATCHES

The aim of this experiment is to design and plot the characteristics of a positive and negative multiplexers based latches .

Introduction

A bistable circuit- a circuit having two stable states that represent 0 and 1, can be designed using a positive feedback. The basic idea is shown in fig.1, which shows two inverters connected in cascade along with the voltage-transfer characteristic typical of such a circuit.

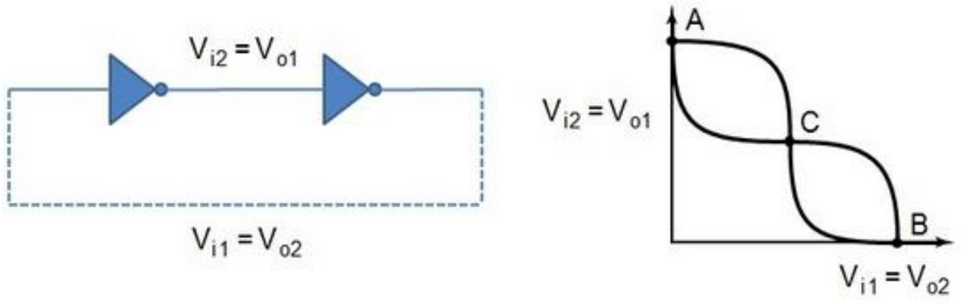


Fig.1: Two cascaded inverters along with their superimposed VTCs.

The above circuit has only three possible operation points (A, B, and C), as demonstrated on the combined VTC. Out of these, A and B are the only stable operating points, and C is a metastable point; therefore, the name bistable. The circuit serves as a memory, storing either a 1 or a 0 corresponding to positions A and B. We can change the state of such a circuit by cutting the feedback loop or by overpowering the feedback loop. The first is called a multiplexer based Latch and it realizes the following multiplexer equation:

$$Q = \overline{Clk}.Q + Clk.D$$

MUX based Latches

Fig.2 shows an implementation of positive and negative static latches based on multiplexers. For a negative latch input D is selected when the CLK is 0 whereas when the CLK is high, output is held. This is reversed for a positive latch.

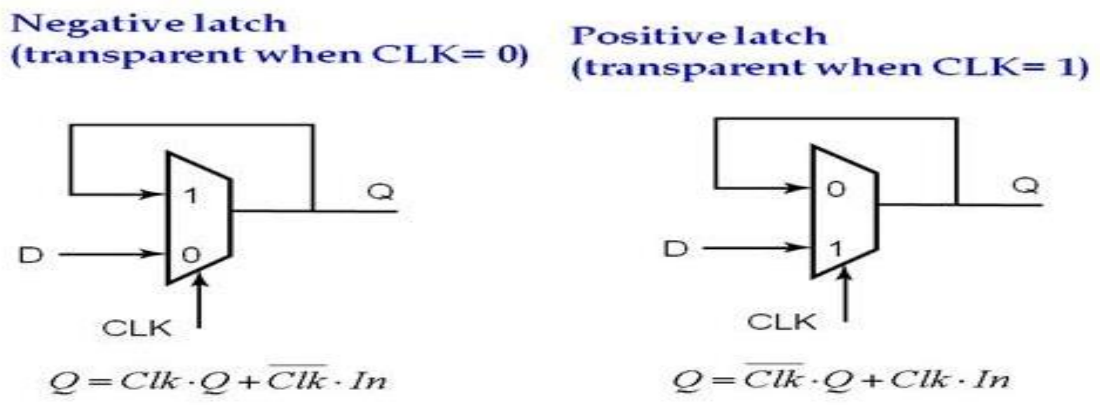

Negative latch
(transparent when CLK= 0)
Positive latch
(transparent when CLK= 1)
1
0
Q
D
CLK
0
1
Q
D
CLK
$Q = Clk \cdot Q + \overline{Clk} \cdot In$
$Q = \overline{Clk} \cdot Q + Clk \cdot In$

Fig.2: Negative and positive latches based on multiplexers.

A transistor-level implementation of a positive latch based on multiplexers is shown in Fig.3. When the CLK is high, the bottom transmission gate is on and the latch is transparent- that is, the D input is copied to the Q output. During this phase, the feedback loop is open, since the top transmission gate is off.

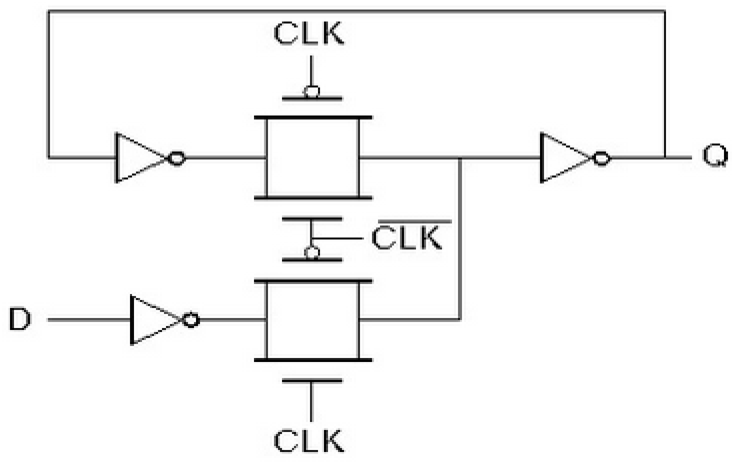


Fig.3: Positive built by using transmission gates

REGISTERS

The aim of this experiment is to design and plot the characteristics of a master-slave positive and negative edge triggered registers based on multiplexers.

MUX based Registers

A register consists of cascading a negative latch (master stage) with a positive one (slave stage). Fig. 1 shows a multiplexer-latch based implementation of register. On the low phase of the clock, the master stage is transperent, and the D input is passed to the master stage output, Q_M. During this period, the slave stage is in the hold mode, keeping its previous value by using feedback. On the rising edge of the clock, the master stage stops sampling the input, and the slave stage starts sampling. During the high phase of the clock, the slave stage samples the output of the master stage (Q_M), while the master stage remains in a hold mode. Since Q_M is constant during the high phase of the clock, the output Q makes only one transistion per cycle. The value of Q is the value of D right before the rising edge of the clock, achieving the positive edge-triggered effect.

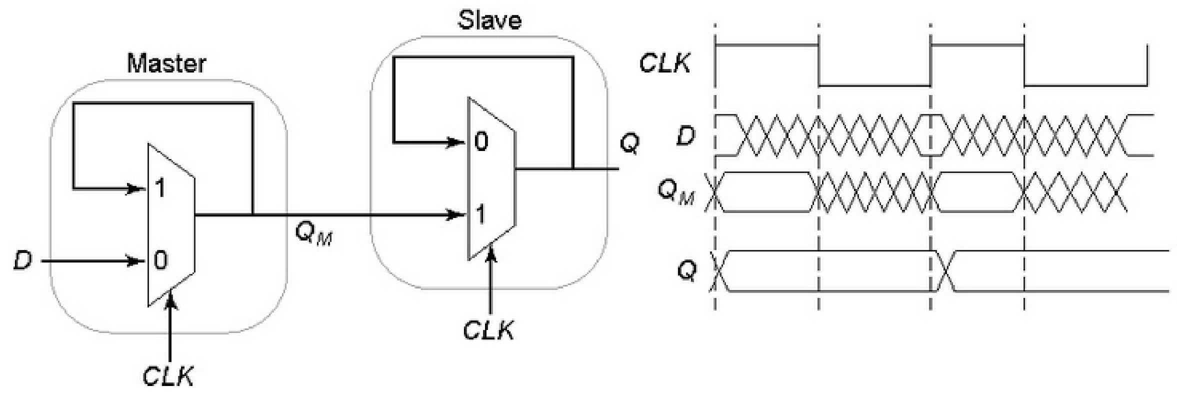


Fig. 1: Positive edge-triggered register and its waveforms.

A negative edge-triggered register can be constructed by using the same principle by simply switching the order of the positive and negative latches (i.e., placing the positive latch first). Fig. 2 shows the transmission gate based implementation of the above register.

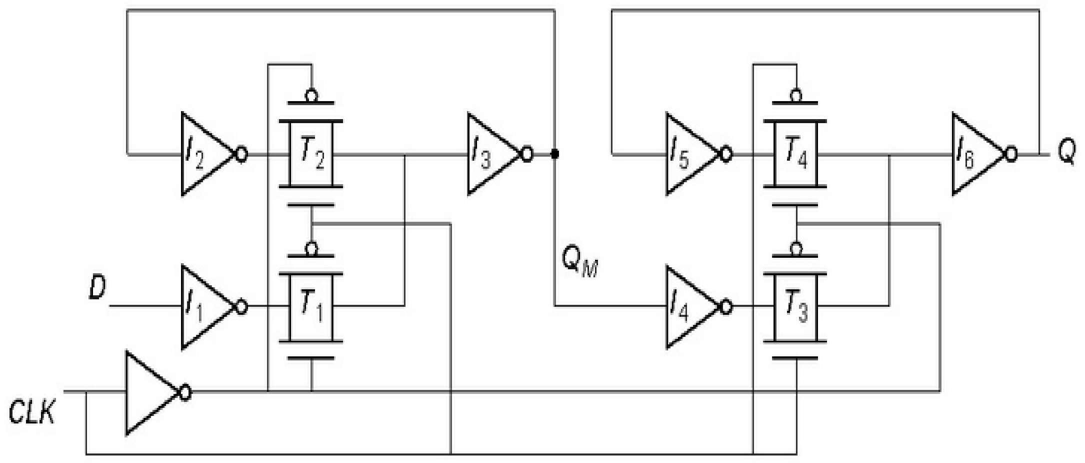
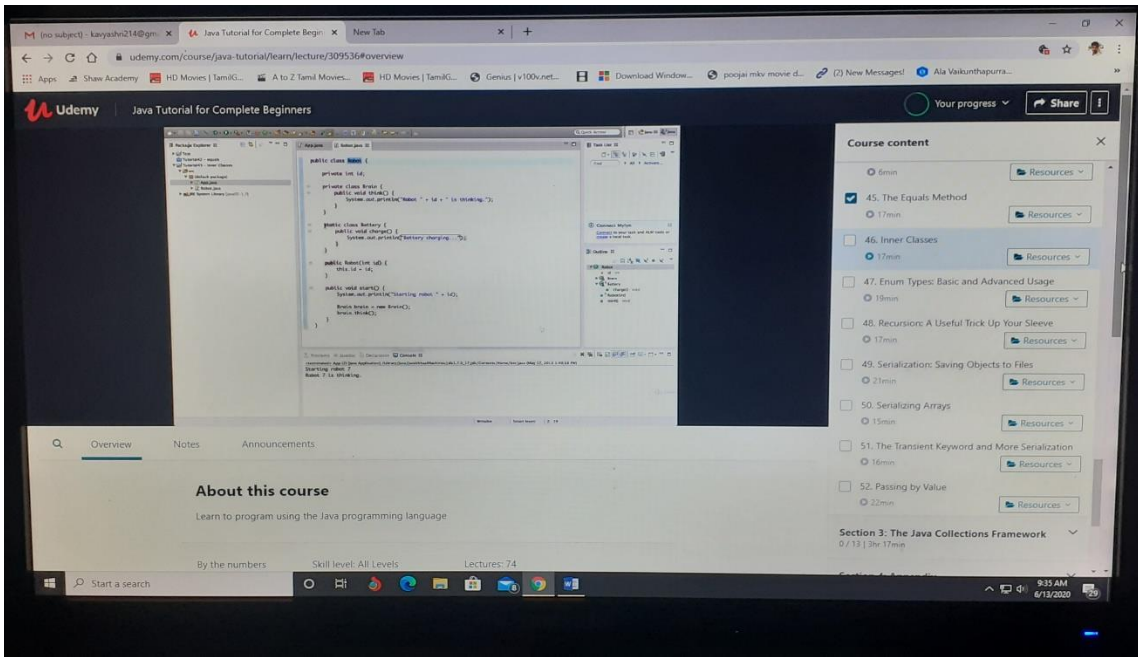


Fig. 2: Master-slave positive edge-triggered register using multiplexers.

AFTERNOON SESSION DETAILS

Date:	13-6-2020	Name:	Kavyashree m
Course:	Java	USN:	4al15ec036
Topic:	The equals method,inner classes,enum types,recursion,serialization:saving objects to files,serializing arrays,the transient keyword and more serialization,passing by value	Semester & Section:	8th A
Github Repository:	kavya		

Image of session



```
public class Robot {

    private int id;

    private class Brain {
        public void think() {
            System.out.println("Robot " + id + " is thinking.");
        }
    }

    static class Battery {
        public void charge() {
            System.out.println("Battery charging...");
        }
    }

    public Robot(int id) {
        this.id = id;
    }

    public void start() {
        System.out.println("Starting robot " + id);

        Brain brain = new Brain();
        brain.think();
    }
}
```

```
Starting robot 7
Robot 7 is thinking.
```

Course content

- 6min
- ✓ 45. The Equals Method 17min
- 46. Inner Classes 17min
- 47. Enum Types: Basic and Advanced Usage 19min
- 48. Recursion: A Useful Trick Up Your Sleeve 17min
- 49. Serialization: Saving Objects to Files 21min
- 50. Serializing Arrays 15min
- 51. The Transient Keyword and More Serialization 16min
- 52. Passing by Value 22min

Section 3: The Java Collections Framework
0 / 13 | 3hr 17min

About this course

Learn to program using the Java programming language

By the numbers | Skill level: All Levels | Lectures: 74

The equals method

The equals method for class Object implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values x and y, this method returns true if and only if x and y refer to the same object.

Following is the declaration for java.lang.Object.equals() method public boolean equals

Parameters

obj − the reference object with which to compare.

Return Value

This method returns true if this object is the same as the obj argument; false otherwise.

Example

The following example shows the usage of lang.Object.equals() method.

```
package com.tutorialspoint;

public class ObjectDemo {

   public static void main(String[] args) {

      // get an integer, which is an object
      Integer x = new Integer(50);

      // get a float, which is an object as well
      Float y = new Float(50f);

      // check if these are equal,which is
      // false since they are different class
      System.out.println("" + x.equals(y));

      // check if x is equal with another int 50
      System.out.println("" + x.equals(50));
   }
}
```

Enum types

The Enum in Java is a data type which contains a fixed set of constants.The Enum in Java is a data type which contains a fixed set of constants.It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY) , directions (NORTH, SOUTH, EAST, and WEST), season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colors (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc. According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.

Java Enums can be thought of as classes which have a fixed set of constants (a variable that does not change). The Java enum constants are static and final implicitly. It is available since JDK 1.5.Enums are used to create our own data type like classes. The enum data type (also known as Enumerated Data Type) is used to define an enum in Java. Unlike C/C++, enum in Java is more powerful. Here, we can define an enum either inside the class or outside the class.Java Enum internally inherits the Enum class, so it cannot inherit any other class, but it can implement many interfaces. We can have fields, constructors, methods, and main methods in Java enum.

Points to remember for Java Enum

- Enum improves type safety
- Enum can be easily used in switch
- Enum can be traversed
- Enum can have fields, constructors and methods
- Enum may implement many interfaces but cannot extend any class because it internally extends Enum class.

RECURSION

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

It makes the code compact but complex to understand.

Syntax:

```
returntype methodname(){
//code to be executed
methodname();//calling same method
}
```

Java Recursion Example 1: Infinite times

```
public class RecursionExample1 {
static void p(){
System.out.println("hello");
p();
}

public static void main(String[] args) {
p();
}
}
```

Output:

```
hello
hello
...
java.lang.StackOverflowError
```

Serialization

Java provides a mechanism, called object serialization where an object can be represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object.

After a serialized object has been written into a file, it can be read from the file and deserialized that is, the type information and bytes that represent the object and its data can be used to recreate the object in memory

Serializing an Object

The ObjectOutputStream class is used to serialize an Object. The following SerializeDemo program instantiates an Employee object and serializes it to a file.

When the program is done executing, a file named employee.ser is created. The program does not generate any output, but study the code and try to determine what the program is doing.

Example

```
import java.io.*;
public class SerializeDemo {

   public static void main(String [] args) {
      Employee e = new Employee();
      e.name = "Reyan Ali";
      e.address = "Phokka Kuan, Ambehta Peer";
      e.SSN = 11122333;
      e.number = 101;

      try {
         FileOutputStream fileOut =
         new FileOutputStream("/tmp/employee.ser");
         ObjectOutputStream out = new ObjectOutputStream(fileOut);
         out.writeObject(e);
         out.close();
         fileOut.close();
         System.out.printf("Serialized data is saved in /tmp/employee.ser");
      } catch (IOException i) {
         i.printStackTrace();
      }
   }
}
```

Serializing Arrays

Arrays in Java are serializable - thus Arrays of Arrays are serializable too.The objects they contain may not be, though, so check that the array's content is serializable - if not, make it so.

Here's an example, using arrays of ints.

```
public static void main(String[] args) {

    int[][] twoD = new int[][] { new int[] { 1, 2 },
            new int[] { 3, 4 } };

    int[][] newTwoD = null; // will deserialize to this

    System.out.println("Before serialization");
    for (int[] arr : twoD) {
        for (int val : arr) {
            System.out.println(val);
        }
    }

    try {
        FileOutputStream fos = new FileOutputStream("test.dat");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(twoD);

        FileInputStream fis = new FileInputStream("test.dat");
        ObjectInputStream iis = new ObjectInputStream(fis);
        newTwoD = (int[][]) iis.readObject();

    } catch (Exception e) {

    }

    System.out.println("After serialization");
    for (int[] arr : newTwoD) {
        for (int val : arr) {
            System.out.println(val);
        }
    }
}
```

Output:

```
Before serialization
1
2
```

```
3
4
After serialization
1
2
3
4
```

Transient keyword

transient is a variables modifier used in serialization. At the time of serialization, if we don’t want to save value of a particular variable in a file, then we use transient keyword. When JVM comes across transient keyword, it ignores original value of the variable and save default value of that variable data type. transient keyword plays an important role to meet security constraints. There are various real-life examples where we don’t want to save private data in file. Another use of transient keyword is not to serialize the variable whose value can be calculated/derived using other serialized objects or system such as age of a person, current date, etc.

Practically we serialized only those fields which represent a state of instance, after all serialization is all about to save state of an object to a file.

```
// A sample class that uses transient keyword to
// skip their serialization.
class Test implements Serializable
{
    // Making password transient for security
    private transient String password;

    // Making age transient as age is auto-
    // computable from DOB and current date.
    transient int age;

    // serialize other fields
    private String username, email;
    Date dob;

    // other code
}
```

Pass by Value

It is a process in which the function parameter values are copied to another variable and instead this object copied is passed. This is known as call by Value.

Java is always Pass by Value and not pass by reference, we can prove it with a simple example.

Let’s say we have a class Balloon like below.

```
package com.journaldev.test;

public class Balloon {

    private String color;

    public Balloon(){}

    public Balloon(String c){
        this.color=c;
    }
    public String getColor() {
            return color;
    }

    public void setColor(String color) {
            this.color = color;
    }
}
```