RV College of Engineering®, Bengaluru-560059

(Autonomous Institution affiliated to VTU, Belagavi)

Department of Electronics and Communication Engineering

Academic Year:2022-23

Digital VLSI Design

18EC54

Laboratory Manual and Observation Book

(Autonomous Scheme 2018)

RV College of Engineering®, Bengaluru

(*Autonomous institution affiliated to VTU, Belagavi*)

Department of Electronics and Communication Engineering

Laboratory Certificate

This is to certify that Mr./Ms_____ _____ has satisfactorily completed the course of Experiments in Practical _____prescribed by the Department during the year_____

Name of the Candidate:_____

USN No.:_____Semester:_____

.

Marks	
Maximum	**Obtained**
50	

Marks in Words

Signatures:

Staff in-charge Head of the Department

Date:

RV College of Engineering®, Bengaluru

(*Autonomous institution affiliated to VTU, Belagavi*)

Department of Electronics and Communication Engineering

.

Digital VLSI Design (18EC54)

Scheme of Conduction and Evaluation

CLASS: V SEMESTER CIEMARKS: 50

YEAR: 2022-23 SEE: 3Hrs SEEMARKS: 50

Note:

1. Out of 10 experiments, for 9 experiments manual will be provided. Each of these would also include practice experiments. Last experiment is a case study and is compulsory.

2. Practice questions: Students should design in advance and practice the lab.

Exp	Title	Page	Duration In Hrs	Max. Marks	Marks Obt.
1.a	Realize CMOS Logic-universal gates	5	2.5	10	
1.b	Practice question: Realize XOR/XNOR gates				
2.a	Realization of CMOS-adder circuits	19	2.5	10	
2.b	Practicequestion:Realize4-bitadder/subractor				
3.a	MOS device Characterization	27	2.5	10	
3.b	Practice question: Plot g_mVs V_{gs} for NMOS/P-MOS				
4.a	Inverter Static Characteristics	43	2.5	10	
4.b	Practice question: Plot the Voltage Transfer Characteristic graph of CMOS inverter and calculate the switching voltage for the given specification				
5.a	Sequential Circuit Design using Master-Slave configuration	53	2.5	10	
5.b	Practice question: Realize 4-bit Ring counter and Johnson counter				
6	Inverter layout and post simulation	59	2.5	10	
7.a	CMOS NAND and NOR gates layout and post simulation	71	2.5	10	
7.b	Practice question: Realize AND/OR gates				
8.a	Common source single stage amplifier and Differentail amplifier	77	2.5	10	
8.b	Practice question :Realize Op-amp circuit				
9.a	Synthesis of Serial Adder	89	2.5	10	
9.b	Practice question: Perform PnR for serial adder				
10	**Case Study:** ASIC design flow using Innovas. (Students should learn the concept and produce the relevant document).	112	2.5	10	
	Total record marks obtained :			80	
	Record Marks :			40	
	Lab Test :			10	
	Final Assessment :			50	

Vision

Imparting quality technical education through interdisciplinary research, innovation and teamwork for developing inclusive & sustainable technology in the area of Electronics and Communication Engineering.

Mission

- To impart quality technical education to produce industry-ready engineers with a research outlook.
- To train the Electronics & Communication Engineering graduates to meet future global challenges by inculcating a quest for modern technologies in the emerging areas.
- To create centre of excellence in the field of Electronics & Communication Engineering with industrial and university collaborations.
- To develop entrepreneurial skills among the graduates to create new employment opportunities.

CO-PO Mapping:

CO/PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	3	2	-	3	-	-	-	1	-	-	2
CO2	3	2	3	2	3	-	1	-	-	-	-	2
CO3	3	3	2	2	3	-	-	-	-	1	-	1
CO4	1	1	3	3	3	-	2	1	-	1	-	1

L: Low-1; M:Medium-2; H:High-3;

General Guidelines

- The students are required to strictly follow the scheme of conduction of the experiments
- The students are here by advised to prepare for the experiments well in advance and the timing of the lab slots are to be effectively utilized for the computational and analysis aspects of the experiments
- The students are required to bring in their Lab Record duly complete in all respects, without which the students will not allowed to so the experiments in the lab.
- Submission of the Lab Records complete in all respects is to be done in the next subsequent Lab class.
- Students must maintain strict academic discipline in the Laboratory.
- Use of external hard disks (thumb drives) in the Lab without prior permission of the Lab-in-charge will be viewed very seriously
- CIE Marks will be awarded based on the conduction, Analysis, Result & Inference with necessary attendance requirements

Sl.No	Criteria	Excellent	Good	Average	Max Score
Data sheet					
A	Problem statement	9-10	6-8	1-5	**10**
B	Design & specifications	9-10	6-8	1-5	**10**
C	Expected output	9-10	6-8	1-5	**10**
Record					
D	Simulation/ Conduction of the experiment	14-15	11-13	1-10	**15**
E	Analysis of the result.	14-15	11-13	1-10	**15**
Viva					**40**
Total					**100**
Scale down to 10 marks					

Preface

This laboratory complements the course **18EC54: Digital VLSI Design**. The lab manual details basic CMOS integrated Circuit design, simulation, and testing techniques. Several tools from the Cadence Development System have been integrated in to the lab to teach students the idea of Computer Aided Design (CAD) and to make the VLSI experience more practical.

To fully appreciate the material in this lab course, the student should have a minimal back ground with the following computer systems, equipment, and circuit analysis techniques. Students should be familiar with the UNIX operating system. Previous experience using a SPICE-like circuit simulator is also important. This course does not explain the various SPICE analyses and assumes the student is capable of configuring the appropriate SPICE analysis to obtain the desired information from the circuit. Finally, the student should have general familiarity with active circuit ”hand” analysis. All of these prerequisites are satisfied by having credit for **18EC33** and **18EC34**.

The lab manual develops the concepts of Integrated Circuit design in a bottom-up approach. First, the basic devices of CMOS circuit design, the NMOS and PMOS transistors, are introduced and characterized. Then, one or more transistors are combined into a simple **Digital sub circuits**, such as Inverter and combinational circuits, and **Analog sub circuits**, such as Differential pairs, and their characteristics are analyzed. Finally, these sub circuits are connected to form larger digital circuits such as Sequential circuits and analog circuits such as operational transconductance amplifiers, and the idea of design methodologies is developed. Continuing with the bottom-up approach, these circuits can be combined to form systems such as filters or data converters (not currently covered in this course).Notably, the Digital circuit design, like NOR gate and D-Flip flop, gives a flavor of the *standard cell design methodology*.

Finally, to appreciate the *standard cell design methodology*, synthesis of serial adder and a case study on Place and Route (PnR) process (a part of ASIC flow) is included as part of the experiment.

Contents

Preface **vii**

1 Realize CMOS Logic-universal gates **5**
1.1 Objective 5
1.2 Introduction 5
1.3 Invoking Cadence 5
1.3.1 Common Procedure 5
1.3.2 Attaching Technology Lib to User Lib 6
1.4 Universal Gates Design 7
1.4.1 Inverter 8
1.4.2 NOR Gate 12
1.4.3 NAND Gate 14
1.5 Observations 16

2 Realization of CMOS-adder circuits **19**
2.1 Objective 19
2.2 Introduction 19
2.3 Theory 19
2.3.1 CMOS Full adder 19
2.3.2 4-bit parallel Ripple carry adder 20
2.4 Lab conduction Procedure 21
2.4.1 CMOS Full adder 21
2.4.2 4-bit ripple carry adder 23
2.5 Observations 24

3 MOS device Characterization **27**
3.1 Objective 27
3.2 Introduction 27
3.3 Theory 27
3.3.1 The MOS Transistor under Static Conditions 27
3.3.2 MOSFET Capacitance 29
3.4 Lab conduction Procedure 30
3.4.1 Schematic Creation 30
3.4.2 Analysis setup 31
3.5 Observations 39
3.5.1 Id Vs Vds 39
3.5.2 Id Vs Vgs 39
3.5.3 Vth variation 39
3.5.4 Resistance plot 40
3.5.5 CV plot 40

4 Inverter Static Characteristics 43
4.1 Objectives 43
4.2 Introduction 43
4.3 Theory 43
4.3.1 Switching Threshold 43
4.3.2 Noise Margins 44
4.4 Lab conduction Procedure 44
4.4.1 Switching voltage of Inverter 45
4.5 Observations 50
4.5.1 Switching voltage 50
4.5.2 Noise Margin 50

5 Sequential Circuit Design using Master-Slave configuration 53
5.1 Objective 53
5.2 Introduction 53
5.3 Theory 53
5.3.1 Clocked Latches or Level-sensitive Flip-flops 53
5.3.2 Master-Slave Edge-Triggered Register 54
5.4 Lab conduction Procedure 54
5.4.1 Clocked D Latch 54
5.4.2 Positive Edge-triggered Master-slave D Flip flop 56
5.5 Observations 56

6 Inverter layout and post simulation 59
6.1 Objective 59
6.2 Introduction 59
6.3 Theory 59
6.3.1 Integrated circuit layout 59
6.3.2 Physical Verification 60
6.3.3 Parasitic Extraction 60
6.4 Lab conduction Procedure 61
6.4.1 Inverter using gpdk180 Technology 61
6.4.2 User Library modification 61
6.4.3 Design Schematic creation 61
6.4.4 Test schematic creation 62
6.4.5 Pre-Layout simulation 62
6.4.6 Design Layout creation 63
6.4.7 Physical Verification 66
6.4.8 Parasitic Extraction 67
6.4.9 Configuration Cell View 67
6.4.10 Post-Layout simulation 69
6.5 Observation 69

7 NOR/NAND gates layout and post simulation 71
7.1 Objective 71
7.2 Introduction 71
7.3 Theory 71
7.4 Lab conduction Procedure 72
7.5 Observation 72

8 Common source single stage amplifier and Differential amplifier 77
8.1 Objective 77
8.2 Introduction 77
8.3 Theory 77
8.3.1 CS amplifier with current source load 77
8.3.2 Differential pair with active mirror load 78
8.4 Lab conduction Procedure 79
8.4.1 CS Amplifier 79
8.4.2 Differential Amplifier 83
8.5 Observation 86

9 Synthesis of Serial Adder **89**
9.1 Objective 89
9.2 Introduction 89
9.3 Theory 89
9.3.1 Basics of RTL coding 90
9.3.2 Overview of Synthesis process 90
9.3.3 Overview of the contents of.lib and .sdc files 91
9.3.4 Finite State Machine 92
9.3.5 Physical Design 94
9.4 Lab conduction Procedure 95
9.4.1 Directory structure 96
9.4.2 Codes 96
9.4.3 Compilation, Elaboration and Simulation of RTL code 99
9.4.4 Synthesize RTL code 101
9.4.5 Compilation, Elaboration and Simulation of Synthesized Net list. 103
9.5 Observation 104

10 Case Study: ASIC Design flow 112
10.1 Objective 112
10.2 Introduction 112
10.3 Theory: Typical Design Flow 112
10.4 Observation 112

Introduction

Introduction

This particular chapter will introduce students to the computer system and software used throughout the lab course. It is necessary for the student to understand some basic UNIX commands used during lab, thus facilitating the conduction of lab with ease. The following will help in understanding

1. The *Login* and *Logout* process out of UNIX work Station

2. Teaches the basic operating system commands used to perform file management, printing, and various other tasks.

3. About Cadence Tool and it's invocation in sever.

Procedures

Logging-In/Logging-Out

In order to use the lab, there needs 2 login steps. As illustrated in Fig.1

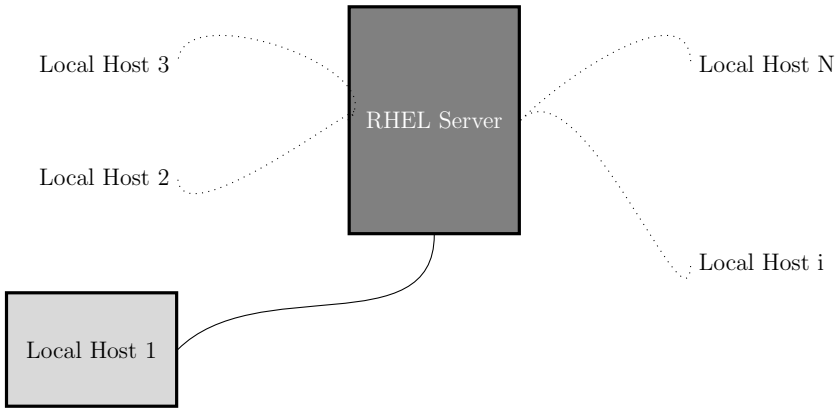


Figure 1: Login and Logout procedure

To use the UNIX server machine, you must first login into the local host system. Then you need to use `ssl` protocol to login into the Cadence server.

> Note: Login using the logon ID and password obtained from the lab instructor.

The User id of the Server login will usually have the following format:

User ID: `vlsi<#>`
Password: `rvvlsi<#>`

The <#> field is usually given by the lab instructor and it has to be followed throughout the lab. The number field usually takes the value from 1 to 25.

For instance, consider the user belong to "A1" group/batch and '1st' user, then the user has the following ID associated to access the server.

User ID: `vlsi1`
Password: `rvvlsi1`

Based on the *User Id*, open a terminal (by right clicking your mouse) to access the server through it's IP Address, as shown in Fig.2

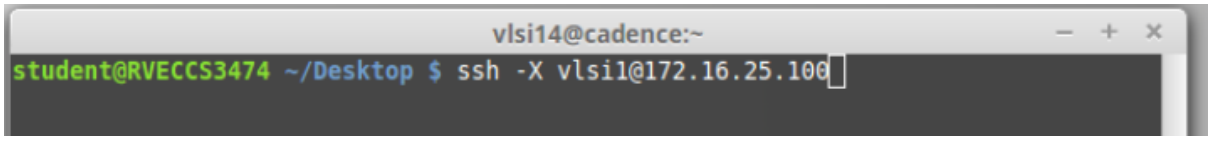


Figure 2: Access to Cadence Work station / server

After pressing "Enter" key, the user will be asked for password.

Note: The password that you enter will not be visible, so don't get panic.

The next section will help to get familiarize with some basic UNIX commands. As this would help you to debug much easier, it is advisable to go through it once.

Using the UNIX Operating System

Using the UNIX operating system is similar to using other operating systems such as DOS. UNIX commands are issued to the system by typing them in a "shell" or "xterm". UNIX commands are case sensitive so be careful when issuing a command, usually they are given in lower-case.

The following list summarizes all the basic commands required to manage the data files you will be creating in this lab course. All UNIX commands are entered from the shell or xterm window. Do not use UNIX commands for modifying, deleting, or moving any Cadence data files.

Note: The command "&" tells UNIX to execute the command and return the prompt to the active shell.

Cadence Tool and it's invocation in server

The Cadence Development System consists of a bundle of software packages such as schematic editors, simulators, and layout editors. This software manages the development process for analog, digital, and mixed-mode circuits. In this course, we will strictly use the tools associated with analog circuit design.

All the Cadence design tools are managed by a software package called the **Design Framework II**. This program supervises a common database which holds all circuit information including schematics, layouts, and simulation data.

From the Design Framework II, also known as the "framework", we can invoke a program called the **Library Manager** which governs the storage of circuit data. We can access libraries and the components of the libraries called **cells**.

Table 2: Common UNIX Commands

Commands	Comments
`ls [ -la]`	Lists files in the current directory. "l" lists with properties and "a" also lists hidden files (ones beginning with a ".").
`cd XXXX`	Changes the current directory to XXXX.
`cd ..`	Changes the current directory back one level.
`cp XXXX YYYY`	Copies the file XXXX to YYYY.
`mv XXXX YYYY`	Move file XXXX to YYYY. Also used for rename
`rm XXXX`	Deletes the file XXXX
`mkdir XXXX`	Creates the directory XXXX in the current directory.
`lp -dXXXX YYYY`	Prints the textfile or postscript file YYYY to the printer named XXXX, where XXXX can be either "ipszac" or "hpszac".
`gedit XXXX&`	Starts the gedit text editor program and loads file XXXX.
`top`	Check available processes and memory usage.
`quota – v`	Check for disk space available.
`who|grep my_name`	Display the terminal where I am connected.

Also, from the framework we can invoke the schematic entry editor. The editor is used to draw circuit diagrams and draw circuit symbols.

A program called **Layout L/XL** is used for creating integrated circuit layouts. The layout is used to create the masks which are used in the integrated circuit fabrication process.

Finally, circuit simulation is handled through an interface called **Analog Design Environment**. This interface can be used to invoke various simulators including HSPICE, Spectre, and Verilog. We will be using the SpectreS simulator in this course.

Now we will look into the set of commands that would lead to invoke a Cadence tool known as **Virtuoso**, as shown in Fig.3

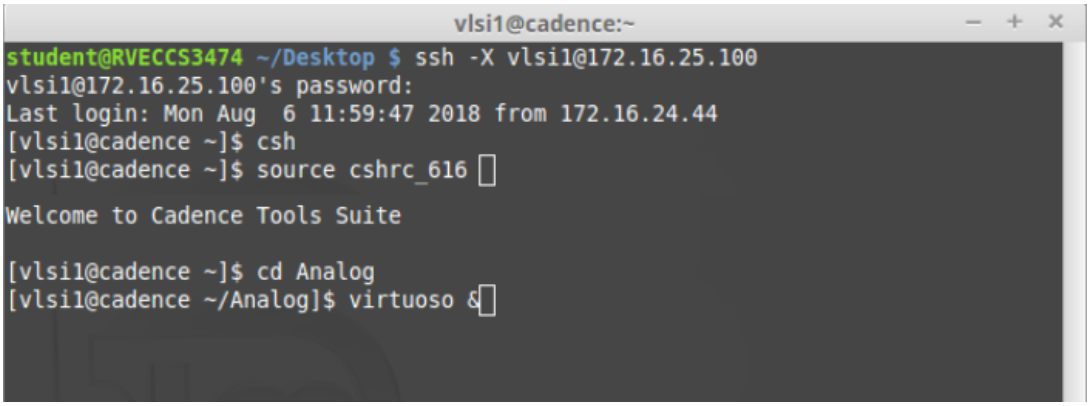


Figure 3: Invoking Virtuoso

Data sheet

The data sheet should include the following information, as shown in Fig.4

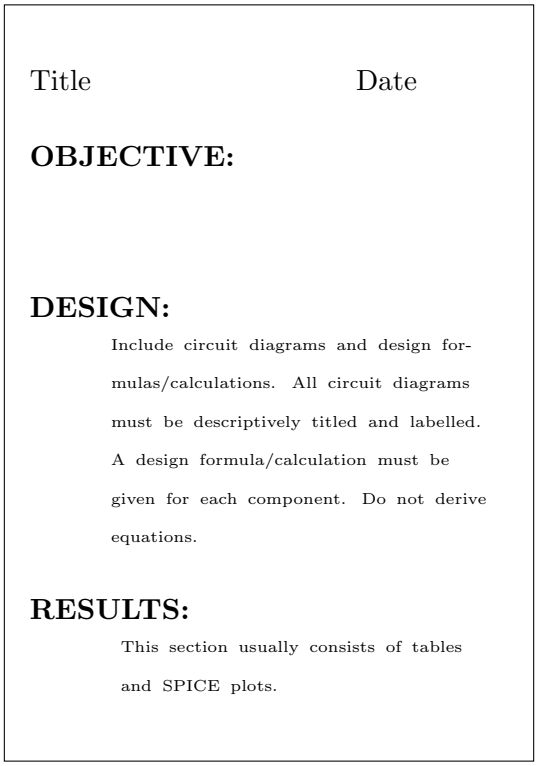


Figure 4: Data Sheet

Experiment 1

Realize CMOS Logic-universal gates

1.1 Objective

To implement CMOS based Inverter, NAND and NOR gates and verify the design using the following analysis:

i DC Analysis

ii Transient Analysis

1.2 Introduction

This lab provides students an overview of the Cadence Development System, using which students will

1. Built a CMOS inverter schematic, Nand and Nor gates,
2. Perform Transient and DC analysis,.

As this is the first experiment, the following section will give us the procedure that are common across all the experiments in the lab. Followed by which, theoretical concepts are given to extract the static parameters from the results obtained.

1.3 Invoking Cadence

Follow the steps shown in Fig.3. Note, as stated, the commands that are marked "#" are executed only for the first experiment/time. For the sake of clarity the commands are repeated below that a user normally uses [For the first time commands, please refer Fig.3]

1.3.1 Common Procedure

1. Open a Terminal window by right clicking the mouse.
2. Now change the shell from Bash to C shell using the following command

   ```
   csh
   ```

3. Now set the UNIX environmental variables to access the Cadence package

   ```
   source cshrc
   ```

4. Now change the directory to "Analog" [Note: The folder name is case sensitive, so type in the folder name with capital "A" and then type "nalog" with small letters].

   ```
   cd Analog/
   ```

5. Now invoke cadence analog package *virtuoso* by typing the following command. Note, use "&" to enable switching between the tool window and active prompt window.

   ```
   virtuoso &
   ```

   This will load Cadence. The Command Interpreter Window (CIW) will now load as shown in Fig.1.1.

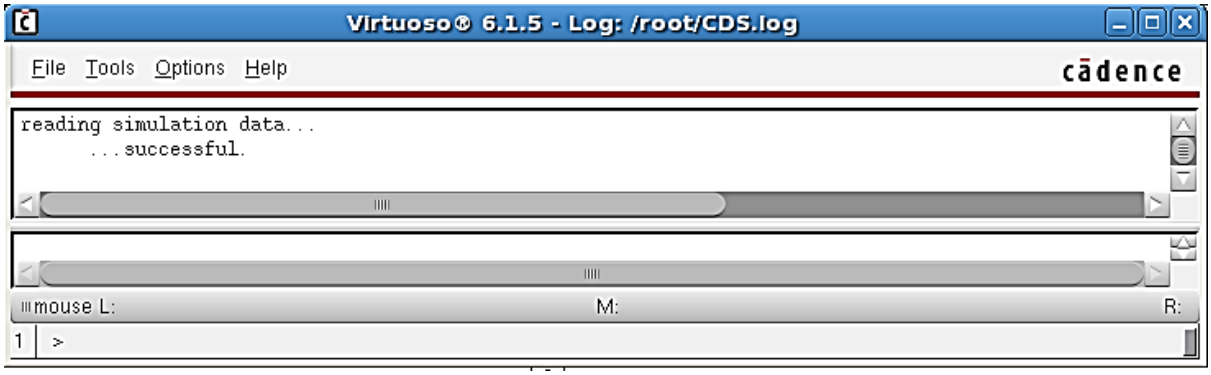


Figure 1.1: Virtuoso CIW

Recommendation:
Keep the CIW insight, from the CIW, you can access all Cadence tolls and functionalities

- view prompts,
- view error and informational messages,
- start specific tools,
- run SKILL command

1.3.2 Attaching Technology Lib to User Lib

A User library is nothing but a library that holds all your designs and test environments. In order simulate, we need the simulator to understand the transistor's parameters, design rules. This is achieved by Attaching the Technology Library to the User Library.
From the CIW, select
. `Tools→Library Manager`
- to load the Library Manager (Fig.1.2). The Library Manager stores all designs in a hierarchal manner. A library is a collection of cells. For example, if you had a digital circuits library named Digital, it will have several cells included in it. These cells will be `inverters, nand gates, nor gates, multiplexers,` etc. Each cell has different views. These views will in general be things such as `symbols, schematics, or layouts`.

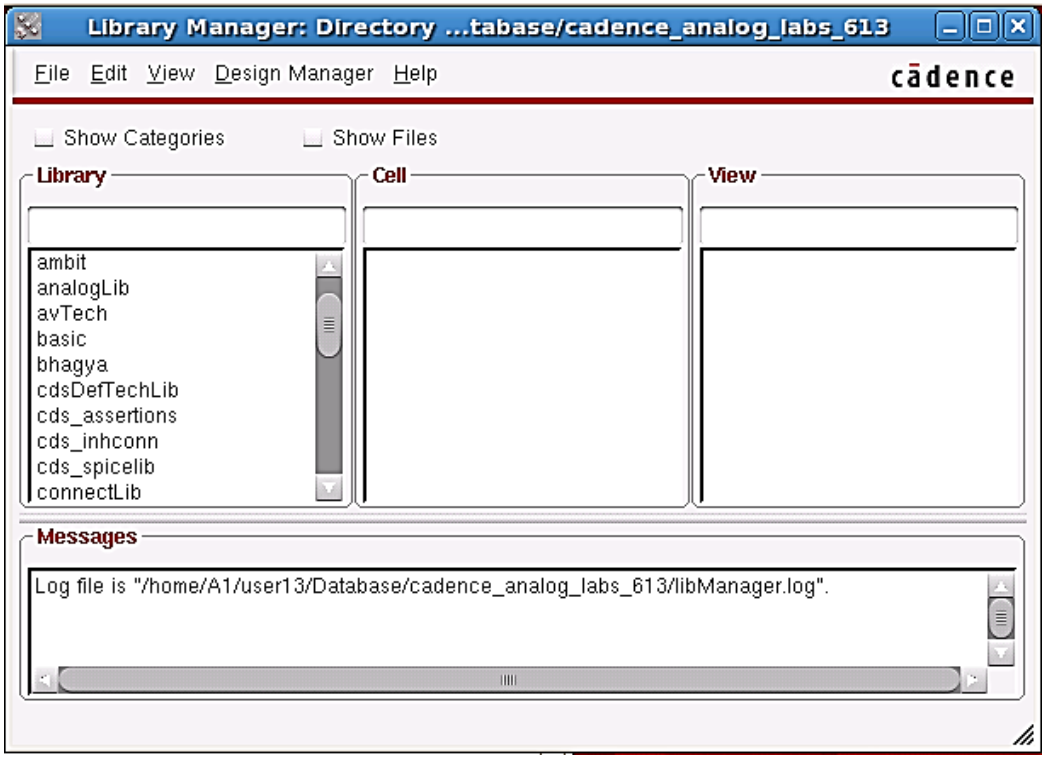


Figure 1.2: Library Manager

The first thing you need to do to start a design is create a library to store the cells you will be designing in this lab. Let's call this `VLSI_Lab`. From the Library Manager select

`File` → `New` → `Library`.

Name the library `VLSI_Lab` and select OK. In the next window that appears select `Attach to an existing techfile` (Fig.1.3) and select OK. In the next window make sure that `gpdk045` is selected and select OK.

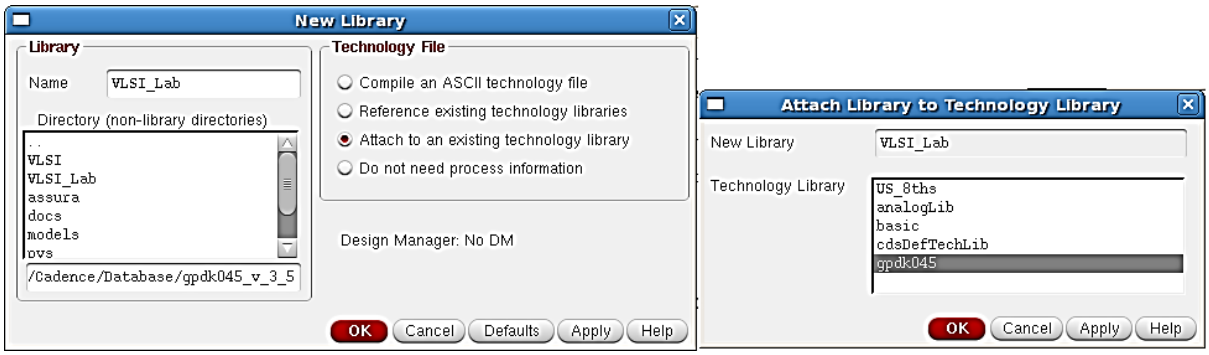


Figure 1.3: Library creation and Attachment of Tech Library

1.4 Universal Gates Design

Mostly, the process of designing and verifying any logic gate involves the creation of **2** *Cells - one* for Designing and *other* for Testing. The following section will help us to utilize both the Cells to achieve the task of design and verification of Universal gates.

1.4.1 Inverter

The first circuit we will design is a simple inverter.

Design Schematic Creation

1. Select which library you want to put the cell into, in this case `VLSI_Lab`,
2. then select `File` → `New` → `Cellview"`
3. Name your cell INVERTER.
   The tool you want to use here is Virtuoso Schematic Editor as seen in Fig.1.4

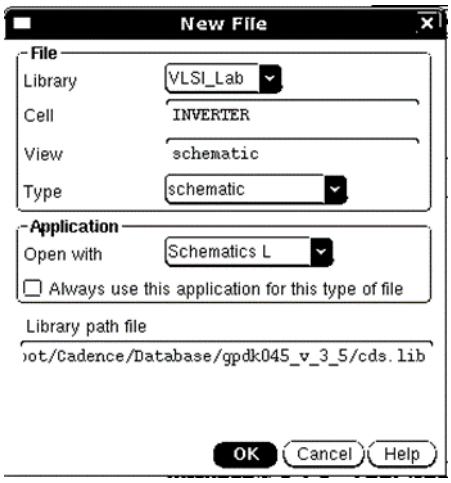


Figure 1.4: Creating a new design cell view for Inverter

4. After selecting OK, the schematic window opens.
5. We wish to add two transistors so that we can make an inverter. To do this we need to add an instance. You can do this by either clicking `Add` → `Instance` or by pressing `i` on the keyboard. A window titled `Add Instance` should pop up. Make sure that the library `gpdk045` is selected.
6. Select `nmos1v` form cell and then `symbol` form view.
7. Go back to the schematic and select where you would like to add the NMOS transistor.
8. Go back to the `Add Instance` and select `pmos1v`. Add this transistor to your schematic.
9. Hit ESC to exit the Add Instance mode.
10. Connect components together using wires. You can select `Add` → `Wire` or use the `w` hotkey.
11. Pins identify the inputs and outputs of the schematic. Click `Add` → `Pin` or use the `p` hotkey.
    Pin names and directions must be consistent between the symbol, schematic, and layout. The name uniquely identifies the pin while the direction indicates the usage of the pin.

12. To change the properties of a device use `Edit` → `Properties` → `Objects` or use the `g` hotkey.
    Try changing the width of the PMOS transistor from its default value, say `120n`, to a variable `wp`. When finished, your schematic should resemble Fig.1.5.

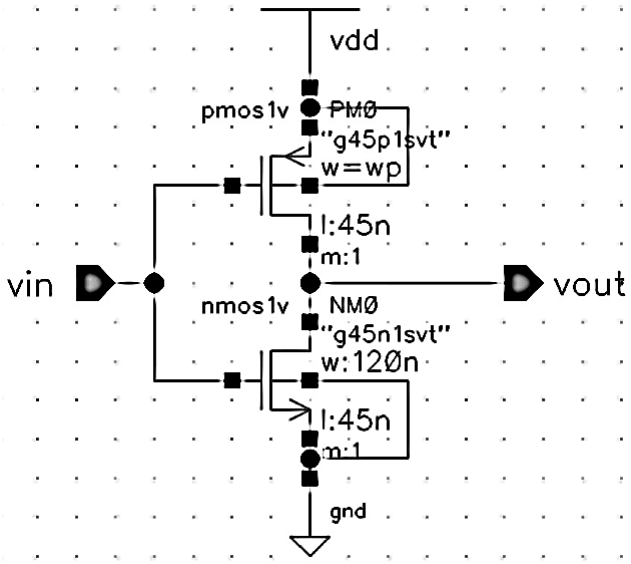


Figure 1.5: Inverter Schematic

13. Select `File` → `Check and Save` to save your schematic and make sure that there are no errors or warnings.

Design Symbol Creation

Once the schematic for the circuit is done, a symbol view must be created. This is the view which is used when an instance of the circuit is put into another schematic (e.g. a test bench circuit).

1. Create a "default symbol" by clicking: `Create` → `Cell View` → `From Cell View`

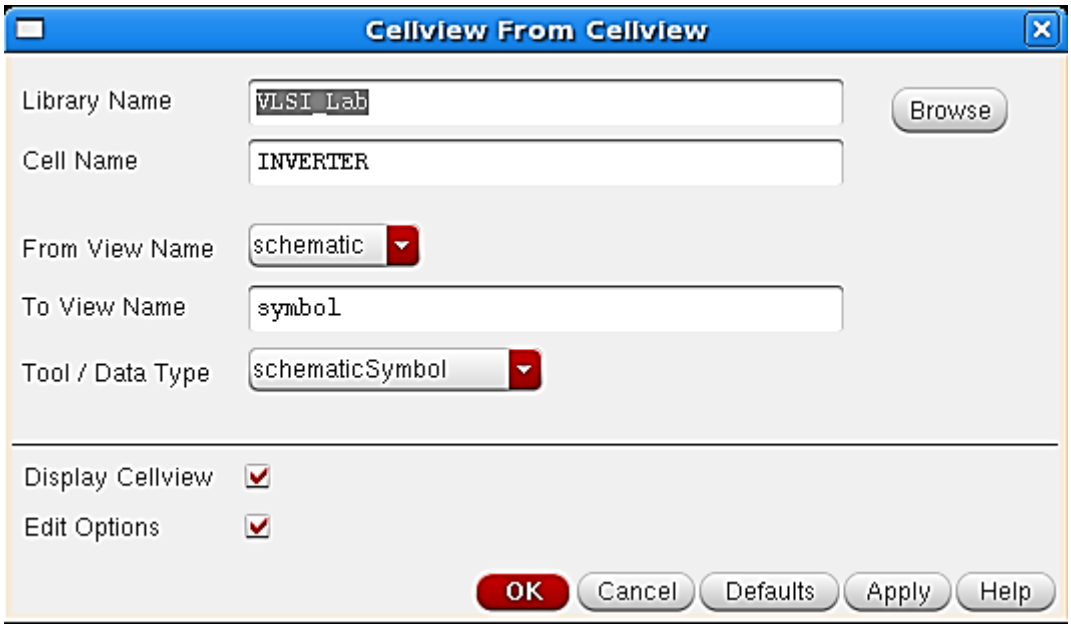


Figure 1.6: Inverter Symbol Creation

2. Use the polygon and other options to create a final symbol that should resemble the Fig.1.7

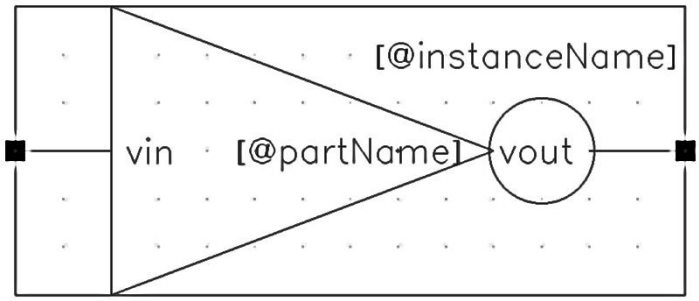


Figure 1.7: Inverter Symbol

Test Schematic Creation

To test the inverter, we need to create a new schematic cell view called `INVERTER_TEST`. To simulate the design, add the inverter symbol, signal sources, power supplies, and loads as illustrated in Fig.1.8. Follow the steps given in Section.1.4.1 for Test schematic creation.

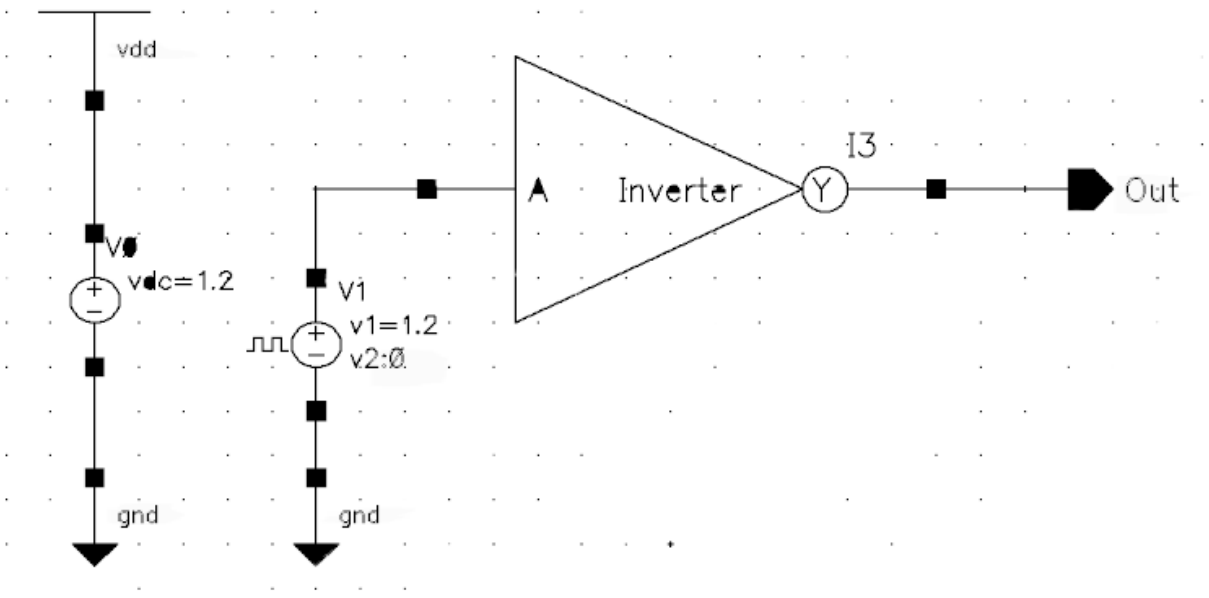


Figure 1.8: Inverter Test Schematic

Table 1.1: Library elements

Library	Cell Name	Cell View	Specification
AnalogLib	`vdd`	Symbol	-
AnalogLib	`gnd`	Symbol	-
AnalogLib	`vdc`	Symbol	DC voltage: 1.2 V
AnalogLib	`vpulse`	Symbol	Voltage 1: 0; Voltage 2: 1.2 V; Period: 20n s; Pulse width: 10n s;
VLSI_lab	`INVERTER`	-	

Pre-Layout Simulation and Analysis

Now we will do 2 analysis to ensure the working of inverter.

1. Start the simulator environment by selecting `Launch` → `ADE L` from the Test schematic.
2. In the Analog Design Environment (ADE), Select `Setup` → `Simulator/Directory/Host` and verify that `spectre` is the simulator.
3. Next we need to configure the environment to run our first simulation.
4. **Transient Analysis**
   (a) In the ADE, Select `Analyses` → `Choose`.
   (b) Select analysis section as `trans`, set the `stop time` as **200n**,
   (c) Click on the enable button and then click apply.
5. **DC Analysis**

(a) In the ADE, Select `Analyses` → `Choose`.

(b) Select `dc` and then `Component Parameter`.

(c) Select `Select Component` and then click on the desired voltage source in the schematic to sweep.
In this case we want to sweep the input voltage source which is out in Fig.1.8.

(d) Select `dc` as the variable to sweep when the popup window opens.

(e) We wish to sweep the source from the lower potential to higher potential, so input **0** into `Start` and **1** into `Stop`. Select OK.

1.4.2 NOR Gate

The procedure discussed in `Inverter` Cell design (Section.1.4.1) would be followed here for the design of 2-input CMOS NOR gate. Since most of the steps will be same, only the most important difference in the process steps will be discussed below.

Design Schematic Creation

1. Select which library you want to put the cell into, in this case `VLSI_Lab`, and then `File` → `New` → `Cellview`.

2. Name your cell `Nor2`

3. Now follow the same procedure that was followed for building the inverter to build the Nor gate. When finished, your schematic should resemble Fig.1.9.

4. Select `File` → `Check and Save` to save your schematic and make sure that there are no errors or warnings.

Design Symbol Creation

1. From the Schematic editor window, select `Create` → `Cell View` → `From Cell View`

2. Follow the same procedure as before, such that the final symbol should resemble as shown in Fig.1.10

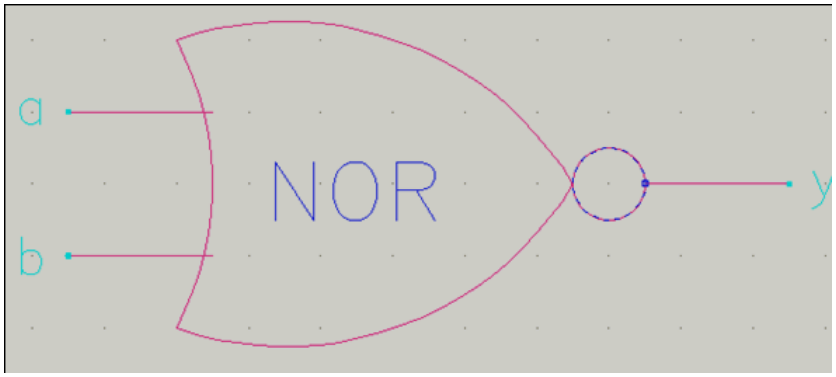


Figure 1.10: 2 input NOR Symbol

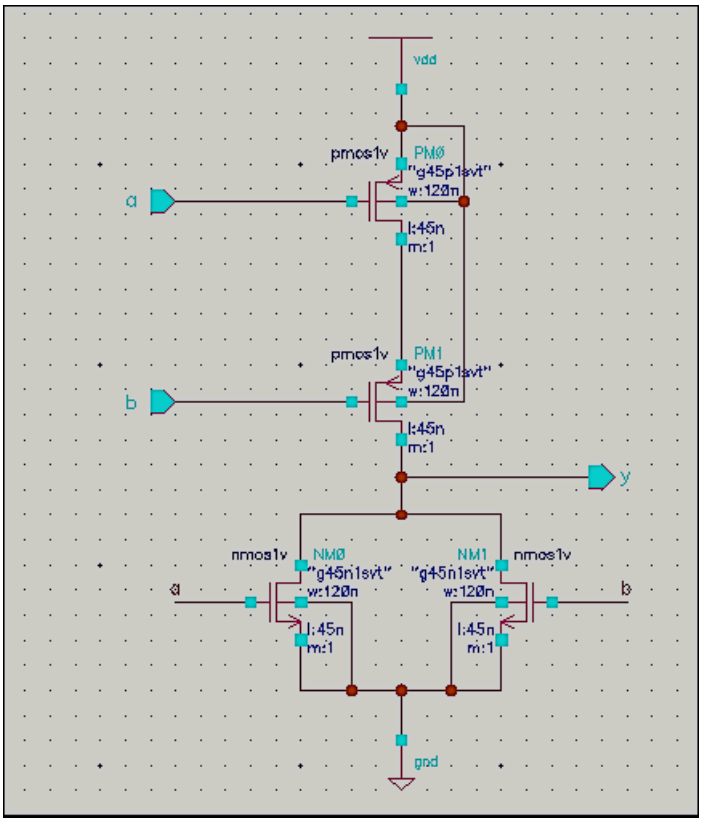


Figure 1.9: 2 input NOR gate Schematic

Test Schematic Creation

Prepare the Test Circuit schematic, create a new schematic cell view called `Nor2_TEST`. The final schematic should resemble as shown in Fig.1.11

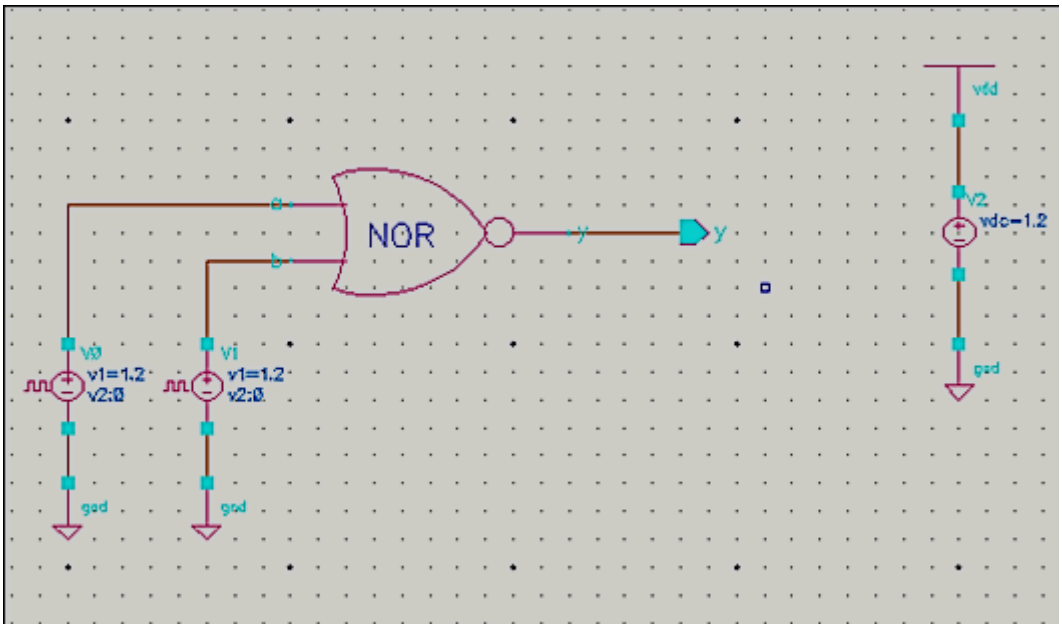


Figure 1.11: 2 input NOR Test Schematic

Pre-Layout Simulation and Analysis

Perform only `trans` Analysis as before.
Transient Analysis

1. In the ADE, Select `Analyses` → `Choose`.
2. Select analysis section as `trans`, set the `stop time` as **200n**,
3. Click on the enable button and then click apply.

1.4.3 NAND Gate

The procedure discussed in `Inverter` Cell design (Section.1.4.1) would be followed here for the design of 2-input CMOS NAND gate

Design Schematic Creation

1. Select which library you want to put the cell into, in this case `VLSI_Lab`, and then `File` → `New` → `Cellview`.
2. Name your cell `Nand2`
3. Now follow the same procedure that was followed for building the inverter to build the Nor gate. When finished, your schematic should resemble Fig.1.12.

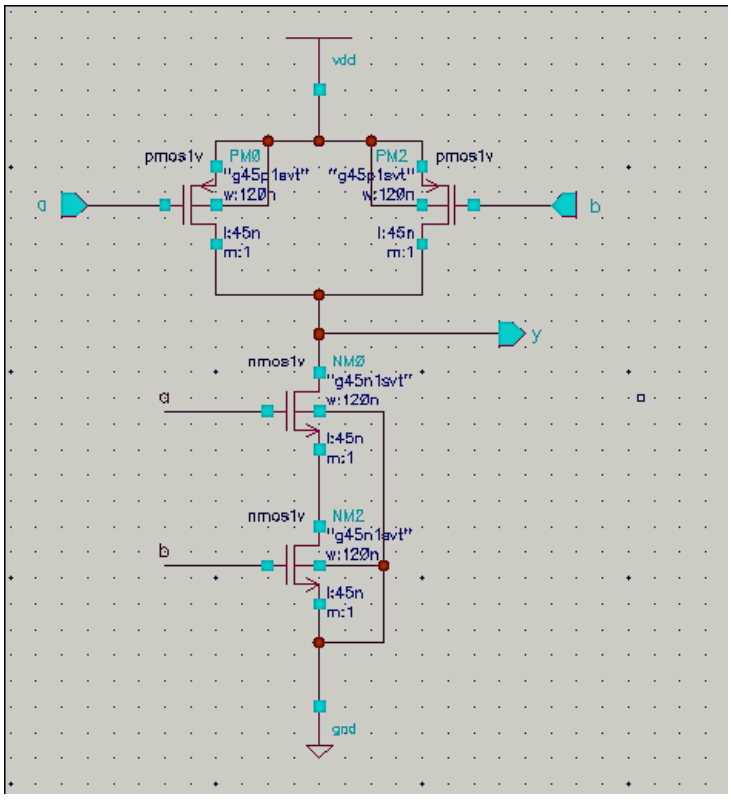


Figure 1.12: 2 input NAND gate Schematic

4. Select `File` → `Check and Save` to save your schematic and make sure that there are no errors or warnings.

Design Symbol Creation

1. From the Schematic editor window, select `Create` → `Cell View` → `From Cell View`
2. Follow the same procedure as before, such that the final symbol should resemble as shown in Fig.1.13

Figure 1.13: 2 input NAND Symbol

Test Schematic Creation

Prepare the Test Circuit schematic, create a new schematic cell view called `Nand2_TEST`. The final schematic should resemble as shown in Fig.1.14

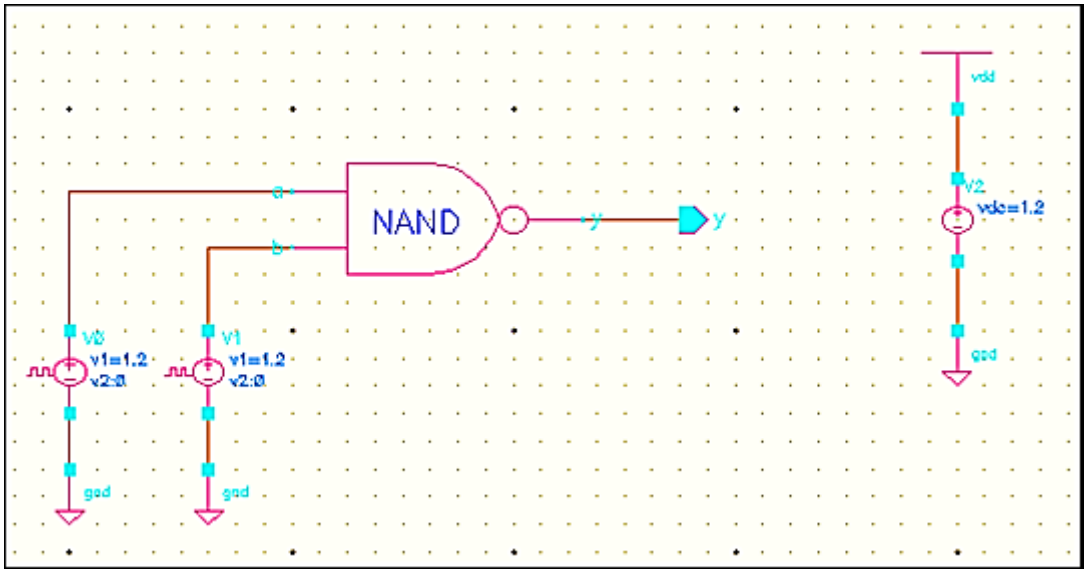


Figure 1.14: 2 input NAND Test Schematic

Pre-Layout Simulation and Analysis

Perform only `trans` Analysis as before.

Transient Analysis

1. In the ADE, Select `Analyses` → `Choose`.
2. Select analysis section as `trans`, set the `stop time` as **200n**,
3. Click on the enable button and then click apply.

1.5 Observations

- DC sweep output graph showing the inverter was simulated.
- Verify the functionality of all the gates through Transient analysis.

Part B

Practice question: Realize XOR/XNOR gates and perform functional verification

.

SI	Criteria	Max Marks	Marks Obtained
Data Sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation / Conduction	15	
E	Analysis of the Result	15	
	Viva	40	
	Total	100	
	Scale to 10 Marks		

Staff Signature

Experiment 2

Realization of CMOS - adder circuits

2.1 Objective

To design a CMOS Full adder circuit and build a 4-bit parallel adder using ripple structure.

2.2 Introduction

Addition forms the basis for many processing operations, from counting to multiplication to filtering. As a result, adder circuits that add two binary numbers are of great interest to digital system designers. An extensive, almost endless, assortment of adder architectures serve different speed/area requirements.

This experiment begins with a full adder for single-bit addition. It then considers a basic simplest structure of Carry-Propagate Adders (CPAs) for the addition of multi-bit words.

2.3 Theory

The theory section contains two sub sections. One gives an overview of full adder structure using CMOS transistors. Other one builds a CPA structure using the full adder that was built as its primitive element.

2.3.1 CMOS Full adder

The basic structure of full adder, as shown in Fig.2.1, contains three inputs (A, B, carry in[C_{in}]) and two outputs (sum S, carry out [C_{out}]).

The carry-out is equivalent to a carry-in to the next more significant column of a multi-bit adder, so it can be described as having double the weight of the other bits. If multiple adders are to be cascaded, each must be able to receive the carry-in.

The truth table for the full adder is given in Table.2.1. For a full adder, it is sometimes useful to define Generate (G), Propagate (P) and perhaps Kill (K) signals. The adder

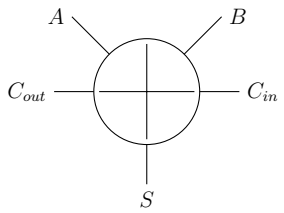


Figure 2.1: Full adder structure.

generates a carry when C_{out} is true independent of C_{in}, so $G = A \cdot B$. The adder kills a carry when C_{out} is false independent of C_{in}, so $K = \overline{A} \cdot \overline{B} = \overline{A + B}$. The adder propagates a carry, i.e., produces a carry-out if and only if it receives a carry-in, when exactly one input is true: $P = A \oplus B$.

From the truth table, the full adder logic can be expressed as:

$$\begin{aligned} S &= A\overline{BC_{in}} + \overline{A}B\overline{C_{in}} + \overline{AB}C_{in} + ABC_{in} \\ &= (A \oplus B) \oplus C_{in} = P \oplus C_{in} \qquad (2.1) \\ C_{out} &= AB + AC_{in} + BC_{in} \\ &= MAJ(A, B, C_{in}) \qquad (2.2) \end{aligned}$$

Table 2.1: Truth Table for Full adder

Inputs			Int. sig.			Outputs	
A	B	C_{in}	G	P	K	C_{out}	S
0	0	0	0	0	1	0	0
		1				0	1
0	1	0	0	1	0	0	1
		1				1	0
1	0	0	0	1	0	0	1
		1				1	0
1	1	0	1	0	0	1	0
		1				1	1

2.3.2 4-bit parallel Ripple carry adder

An N-bit adder when constructed by cascading N full adders, is called **carry ripple adder**. The carry-out of bit i, C_i, is the carry-in to bit $i + 1$. This carry is said to have twice the weight of the sum S_i. The delay of the adder is set by the time for the carries to ripple through the N stages and the delay should be minimized. As shown in the Fig.2.2, this is the simplest architecture among the other CPAs architecture.

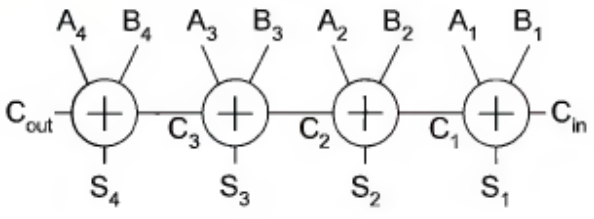


Figure 2.2: 4-bit carry-ripple adder

One of the most serious drawbacks of this adder is that the delay increases linearly with the bit length. Each full adder has to wait for the carry out of the previous stage to output steady-state result. Therefore even if the adder has a value at its output terminal, it has to wait for the propagation of the carry before the output reaches a correct value.

2.4 Lab conduction Procedure

This section primarily focuses on the implementation of the Full adder and 4-bit ripple adder.

2.4.1 CMOS Full adder

Design Schematic Creation

In carry-ripple adders, the critical path goes from C_{in} to C_{out} through many full adders, so the extra delay computing S is unimportant. Thus the adder with transistor sizes are optimized to favor the critical path using a number of techniques. But we will not address any of optimization technique here.

Based on the specification given in Table.2.2 realize the design schematic shown in Fig.2.3

Table 2.2: Design Schematic Library elements

Library	Cell Name	Instance Name	Specification
AnalogLib	`vdd`	-	-
gpdk045	`pmos1v`	PM#	Total width: 120n m; Length: 45n m
AnalogLib	`gnd`	-	-
gpdk045	`nmos1v`	NM#	Total width: 120n m; Length: 45n m
VLSI_lab	`INVERTER`	I#	-

- denotes number ranging from 0 to 12

Symbol

Perform the same steps as we did in Experiment 1 for the creation of symbol. No need to change the shape, as the default shape would suffice the identity of the design.

Test Schematic

To make you become familiar with tool flow, the schematic is not shown here. But its the same step that you followed in Experiment 1.

Based on the Table.2.3 try to rig up the test schematic.

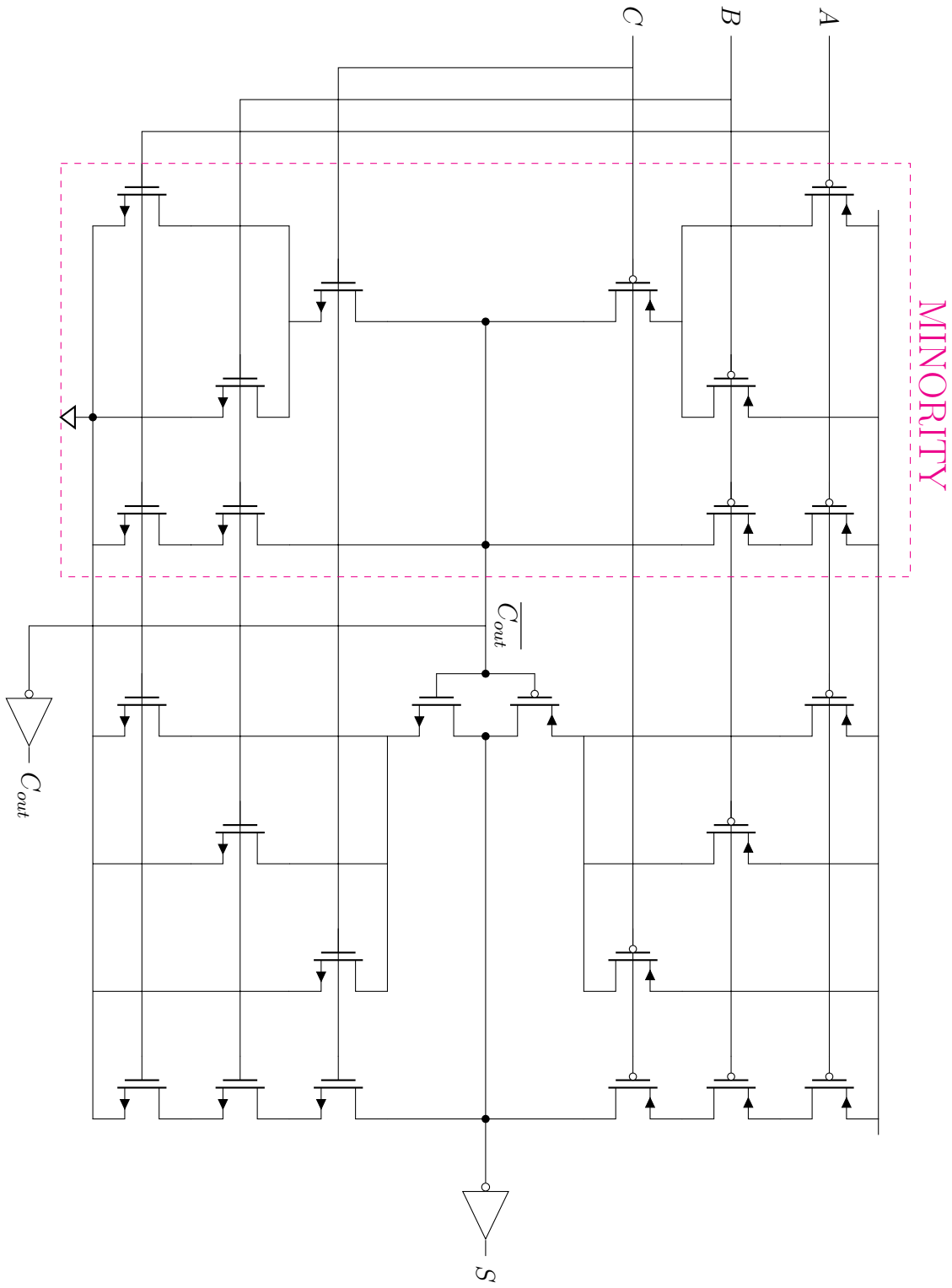


Figure 2.3: Full Adder Schematic

Table 2.3: Test Schematic Library elements

Library	Cell Name	Instance Name	Specification
AnalogLib	`vdd`	-	-
AnalogLib	`gnd`	-	-
VLSI_lab	`FAdd`	I#	-
AnalogLib	`vdc`	V0	DC voltage: 1.2 V
AnalogLib	`vpulse`	V1	Voltage 1: 0; Voltage 2: 1.2 V; Period: 20n s; Pulse width: 10n s;
AnalogLib	`vpulse`	V2	Voltage 1: 0; Voltage 2: 1.2 V; Period: 40n s; Pulse width: 20n s;
AnalogLib	`vpulse`	V3	Voltage 1: 0; Voltage 2: 1.2 V; Period: 60n s; Pulse width: 30n s;

Pre-Layout Simulation

Perform `trans` analysis as before

1. In the ADE, Select `Analyses` → `Choose`.
2. Select analysis section as `trans`, set the `stop time` appropriately, such that you could observe all the combinations of input shown in Table.2.1
3. Click on the enable button and then click apply.

2.4.2 4-bit ripple carry adder

Design Schematic & Symbol

The ripple carry adder is constructed by cascading full adders (FA) blocks in series. One full adder is responsible for the addition of two binary digits at any stage of the ripple carry. The carryout of one stage is fed directly to the carry-in of the next stage.

Based on Fig.2.4, try to build a 4-bit carry-ripple adder Schematic, using the full adder from your library.

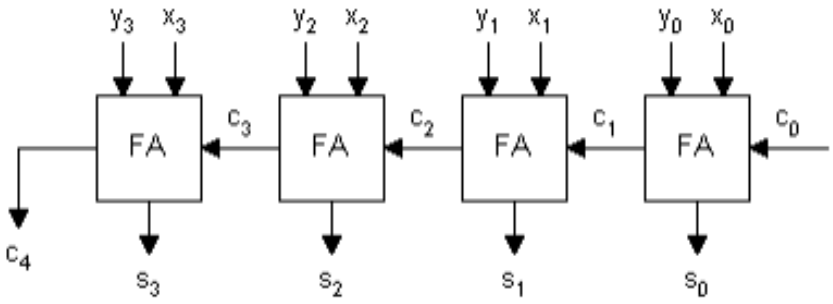


Figure 2.4: 4-bit carry-ripple adder Schematic

Also, create a symbol for the design schematic based on the default setting - follow the procedure steps given in Experiment 1.

Test Schematic

Since we need to measure maximum and minimum delay across C_4 (based on Fig.2.4), it requires 2 circuit conditions.

For computing the *maximum delay*, built the appropriate test schematic, such that all the inputs are X_i and Y_i are connected to `vdd` and `gnd`, and C_0 is excited with `vpulse`. This circuit condition would ensure Propagate condition across all the Full adder, thus measures the maximum across C_4 (C_{out})

For computing the *minimum delay*, C_0 is deactivated using `gnd`, and all the inputs (X_i and Y_i) are retained as it is, except X_4 and Y_4 which are excited with `vpulse`. This circuit condition would ensure that C_4 produce edge quickly.

Pre-Layout Simulation

Perform `trans` analysis as before

1. In the ADE, Select `Analyses` → `Choose`.
2. Select analysis section as `trans`, set the `stop time` appropriately, such that you could observe delay across C_4.
3. Click on the enable button and then click apply.

2.5 Observations

- Check the functionality of full adder against the Table.2.1.
- Calculate the propagation delay of sum and carry for full adder circuit when the inputs switches from $0 \rightarrow 1$ and vice-versa.

Part B

Practice question: Realize 4-bit adder/subtractor

.

SI	Criteria	Max Marks	Marks Obtained
Data Sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation / Conduction	15	
E	Analysis of the Result	15	
	Viva	40	
	Total	100	
	Scale to 10 Marks		

Staff Signature

Experiment 3

MOS device Characterization

3.1 Objective

To understand and measure MOS transistor model parameters.

3.2 Introduction

In this lab we will review basic transistor operation and learn how the short-channel MOS transistor parameters differ from many of the long channel MOS parameters. As we know many of the electrical parameters, such as V_{T0}, λ, K_n (process parameter), and γ (body effect) are constants; but the assumption is valid only for long channel MOS transistors. This experiment shows that these parameters are not constant and helps in understanding what is known as **second-order effect**, which is due to *very high electric-field* between source and drain and reduction in the thickness of the SiO_2, due to scaling. Also we will provide an insight of some of the SPICE model parameters relate to the physical structure and electrical equations of the device.

The section 3.3 gives the necessary theatrical background about the I-V characteristics of short channel MOS transistors, along with the comparative differences with the long-channel device characteristics, and the intrinsic capacitive model of MOS transistor.

Note: The I-V characteristics, and intrinsic capacitive models of the MOSFET are studied.

The section 3.4 in general gives an idea of what and how to measure various electrical model parameters, in reference with previous section 3.3.

3.3 Theory

This section helps you in understanding the static and dynamic models of MOS transistor. Thus we have two sub-sections to address these model behaviour.

3.3.1 The MOS Transistor under Static Conditions

In the derivation of the *static model of the MOS* transistor, we concentrate on the NMOS device. All the arguments made for NMOS are equally valid for PMOS devices as well, but under slightly different environment.

The focus in this *static study* will be on the second-order effects in the sub-micron MOS transistors. We will discuss about these effects with a comparative study on long channel transistors

1. Velocity Saturation
2. Subthreshold Conduction
3. Drain-induced barrier lowering or DIBL

Velocity Saturation

The behavior of transistors with very short channel lengths deviates considerably from the resistive and saturated models of long channel transistors. The main cause for this deficiency is the velocity saturation effect. The electron velocity is related to the electric field through a parameter called the mobility μ_n and its given by

$$v_n = -\mu_n \eta(x) = \mu_n \frac{dV}{dx} \tag{3.1}$$

states that the velocity of the carriers is proportional to the electrical field $\eta(x)$, and the carrier mobility μ_n is constant.

However, at high-field strengths, the carriers fail to follow this linear model. In fact, when the electrical field along the channel reaches a critical value η_c, the velocity of the carriers tends to saturate - due to scattering effects (collisions suffered by the carriers). This is illustrated in Fig.3.1.

The velocity as a function of the electrical field, plotted in Fig.3.1, can be roughly approximated by the following expression:

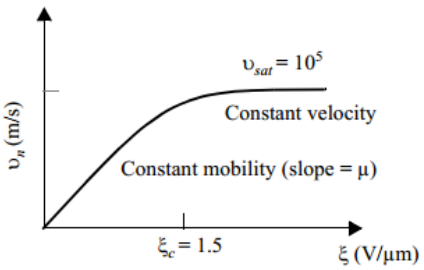


Figure 3.1: Velocity-saturation effect

$$v = \begin{cases} \dfrac{\mu_n \eta}{1 + \eta/\eta_c} & \text{for } \eta \le \eta_c \\ v_{sat} & \text{for } \eta \ge \eta_c \end{cases} \tag{3.2}$$

The continuity requirement between the two regions dictates that $\eta_c = 2v_{sat}/\mu_n$. Thus the modified expression of the drain currents in the resistive region and saturated region are given in Eq.(3.3) and Eq.(3.4) respectively.

$$I_{DSAT} = v_{sat} C_{OX} W \left[(V_{GS} - V_T) - V_{DSAT} \right] \tag{3.3}$$

$$I_{DSAT} = k(V_{DSAT}) \mu_n C_{OX} \frac{W}{L} \left[(V_{GS} - V_T) V_{DSAT} - \frac{V_{DSAT}^2}{2} \right] \tag{3.4}$$

k is a measure of the degree of velocity saturation, since V_{DS}/L can be interpreted as the average field in the channel. In case of long-channel devices (large values of L) or small values of V_{DS}, k approaches 1, whereas for short-channel devices, k is smaller than 1, which means that the delivered current is smaller than what would be normally expected and hence $V_{DSAT} < V_{GS} - V_T$. The device enters saturation before V_{DS} reaches $V_{GS} - V_T$. Short-channel devices therefore experience an extended saturation region, and tend to operate more often in saturation conditions than their long-channel counterparts, as is illustrated in Fig.3.2.

The saturation current I_{DSAT} displays a linear dependence with respect to the gate source voltage V_{GS}, which is in contrast with the squared dependence in the long channel device. This reduces the amount of current a transistor can deliver for a given control voltage. On the other hand, reducing the operating voltage does not have such a significant effect in submicron devices as it would have in a long-channel transistor

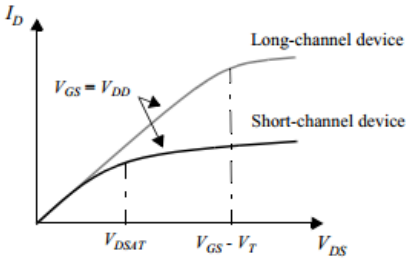


Figure 3.2: Short-channel devices display an extended saturation region due to velocity-saturation.

Subthreshold Conduction

It is the effect where MOS transistor is partially conducting for voltages below the threshold voltage. This effect is called **subthreshold** or **weak-inversion conduction**.

Drain-induced barrier lowering or DIBL

This effect causes the threshold potential to be a function of the operating voltages. Since a part of the region below the gate is already depleted (by the source and drain fields), a smaller threshold voltage suffices to cause strong inversion. In other words, V_{T0} decreases with L for short-channel devices. A similar effect can be obtained by raising the drain-source (bulk) voltage, as this increases the width of the drain-junction depletion region. Consequently, the threshold decreases with increasing V_{DS}.

3.3.2 MOSFET Capacitance

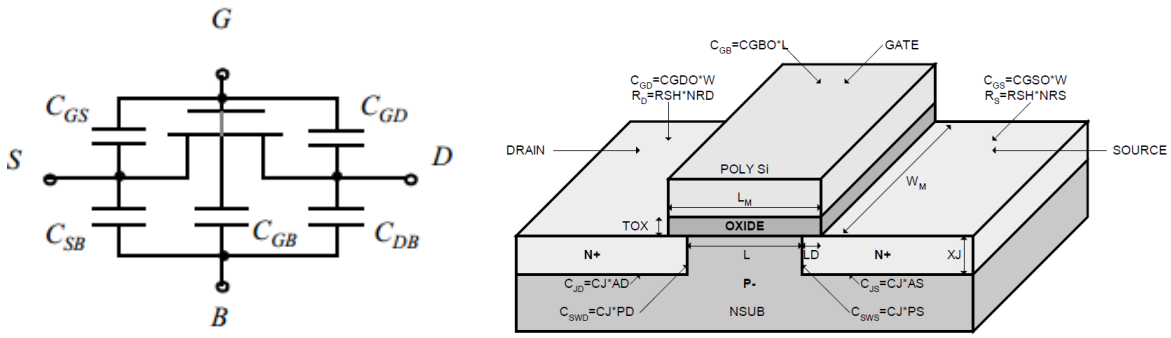
In order to examine the transient (AC) response of MOSFETs and digital circuits consisting of MOSFETs, the study of the nature and the amount of parasitic capacitances associated with the MOS transistor is important.

The dynamic response of a MOSFET transistor is a sole function of the time it takes to (dis)charge the parasitic capacitances that are intrinsic to the device, and the extra capacitance introduced by the interconnecting lines.

They originate from three sources:

Table 3.1: Intrinsic Capacitance

Source of origin	Capacitance	w.r.t applied voltage
Basic MOS structure	Overlap capacitance (C_{GD}, C_{GS})	Fixed
Channel charge	Channel capacitance (C_{GC})	Varies
Depletion regions of reverse-biased pn-junctions of drain and source	Parasitic capacitance (C_{SB}, C_{DB})	Varies



(a) MOSFET capacitance model. (b) Physical structure of a MOSFET

Figure 3.3: MOSFET Capacitance model

Another classification on capacitance can be made based on the MOS capacitance model shown in Fig.3.3a.

Table 3.2: MOSFET capacitance model

Gate capacitance, C_{gg}	(C_{GD}, C_{GS})	Overlap Cap
	+	
	C_{GC}	Channel Cap
Parasitic capacitance C_{SB}, C_{DB}	C_j	Bottom-plate cap
	+	
	C_{jsw}	Side-wall cap

Fig.3.3b is a three-dimensional cross-sectional view of a MOSFET. The gate of the MOS transistor is isolated from the conducting channel by the gate oxide that has a capacitance per unit area equal to $C_{OX} = \epsilon_{OX}/T_{OX}$. The total value of this capacitance is called the **gate capacitance**, C_{gg} and can be decomposed into *two* elements, each with a different behavior.

1. One part of C_{gg} contributes to the channel charge.(Channel Capacitance, C_{GC})

2. Another part is solely due to the topological structure of the transistor.(Overlap Capacitance, C_{GS}, C_{GD})

3.4 Lab conduction Procedure

For this experiment, since MOS transistor itself is design element, only test cell is needed to be designed.

3.4.1 Schematic Creation

Create a schematic with the following specification shown in Fig.3.4.

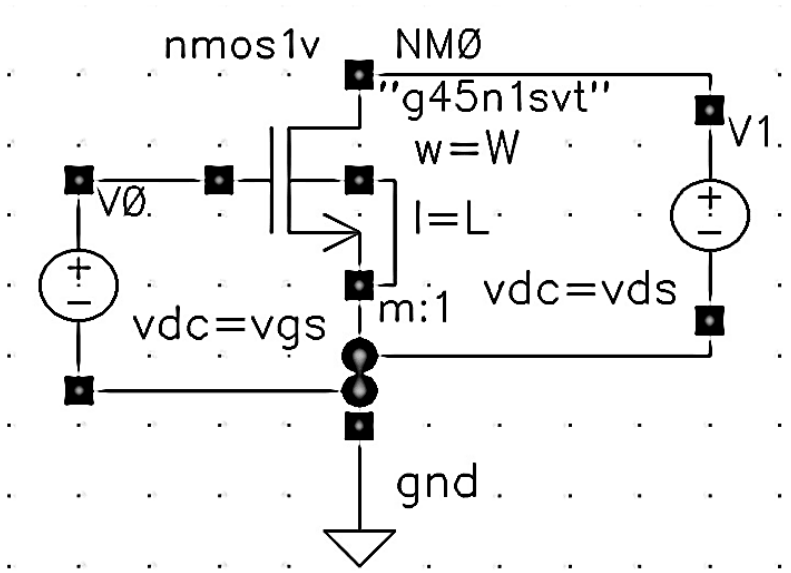


Figure 3.4: NMOS schematic for characterization

Set the appropriate values to the components shown in Table.3.3.

Table 3.3: Library elements

Library	Cell Name	Instance Name	Specification
AnalogLib	`vdc`	V0	DC voltage: Vgs V
AnalogLib	`vdc`	V1	DC voltage: Vds V
gpdk045	`nmos1v`	NM0	Total width: W; Length: L

3.4.2 Analysis setup

For this experiment only `dc` Analysis is required to plot the following graphs:

1. Id Vs Vds
2. Id Vs Vgs
3. Vth variation
4. Resistance plot
5. CV plot

To provide the following default values to the variables defined in test cell, choose `Variables` → `Copy from cellview` in ADE window.

L (m)	45n	Vgs(V)	0.6
W (m)	120n	Vds (V)	0.6

Id vs Vds plot

To exercise short channel effects, provide `L= 45n` and `W = 120n` for **short channel**, and for **long channel** to provide conventional plot, provide `L=6u` and `W = 9u` to mimic long channel MOS transistor.

1. Analysis setup

   (a) Select `dc` analysis

   (b) Check `save DC operating point`.

   (c) Under sweep variable, select `component parameter` to choose `Vds` as DC voltage parameter.

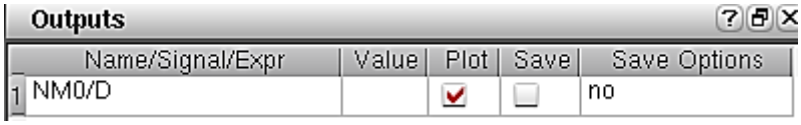
   (d) Provide sweep range : `start-stop` from 0 to 1.2.

2. Output definition

   (a) Select `Outputs` → `To be Plotted` → `Select on Schematic`

   (b) Go to schematic and select the D (drain) terminal to plot the current.

   (c) Press Ecs key.



3. Plot

   (a) Click the `Run` button.

   (b) To plot Id for different values of Vgs, parametric analysis is required.
   Go to ADE window, select `Tools` → `Parametric Analysis ...` and define the flowing.

Variable	Value	Sweep?	Range Type	From	To	Step Mode	Total Steps
vgs	.5	☑	From/To	0	1	Auto	11

At the end, one should observe the following **similar** graphs as shown in Fig.3.5.

Note: The Fig.3.5 is for the illustration purpose. You will be getting a different plot

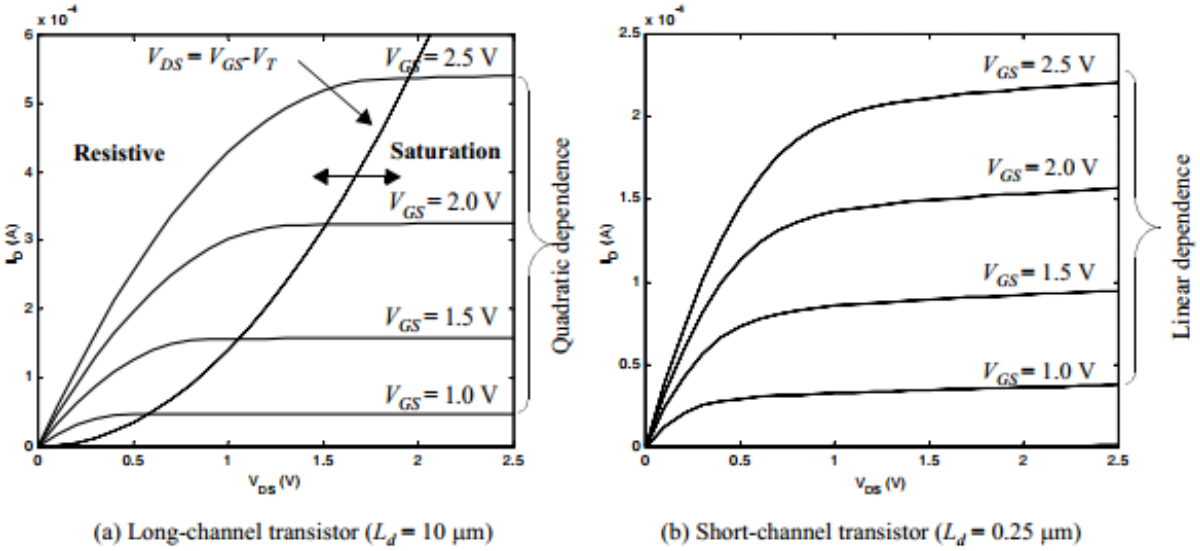


Figure 3.5: $I_D - V_{DS}$ characteristics of long- and a short-channel NMOS transistors in a 0.25 um CMOS technology. The (W/L) ratios of both transistors is identical and equals 1.5.

Id vs Vgs plot

To exercise short channel effects, provide `L= 45n` and `W = 120n` for **short channel**, and for **long channel** to provide conventional plot, provide `L=6u` and `W = 9u` to mimic long channel MOS transistor.

1. Analysis setup
   (a) Select `dc` analysis
   (b) Check `save DC operating point`.
   (c) Under sweep variable, select `component parameter` to choose `Vgs` as DC voltage parameter.
   (d) Provide sweep range : `start-stop` from 0 to 1.2.

2. Output definition
   (a) For the output definition, retain the previous setup as in `Output definition`.
3. Plot
   (a) Click the `Run` button.
   (b) In the waveform window change the scale of Id (y-axis) to log scale.by right-clicking the y-axis and check the `log scale` option.

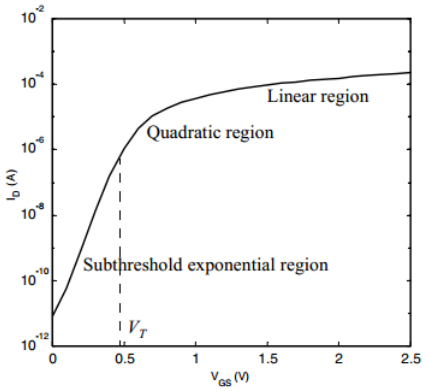
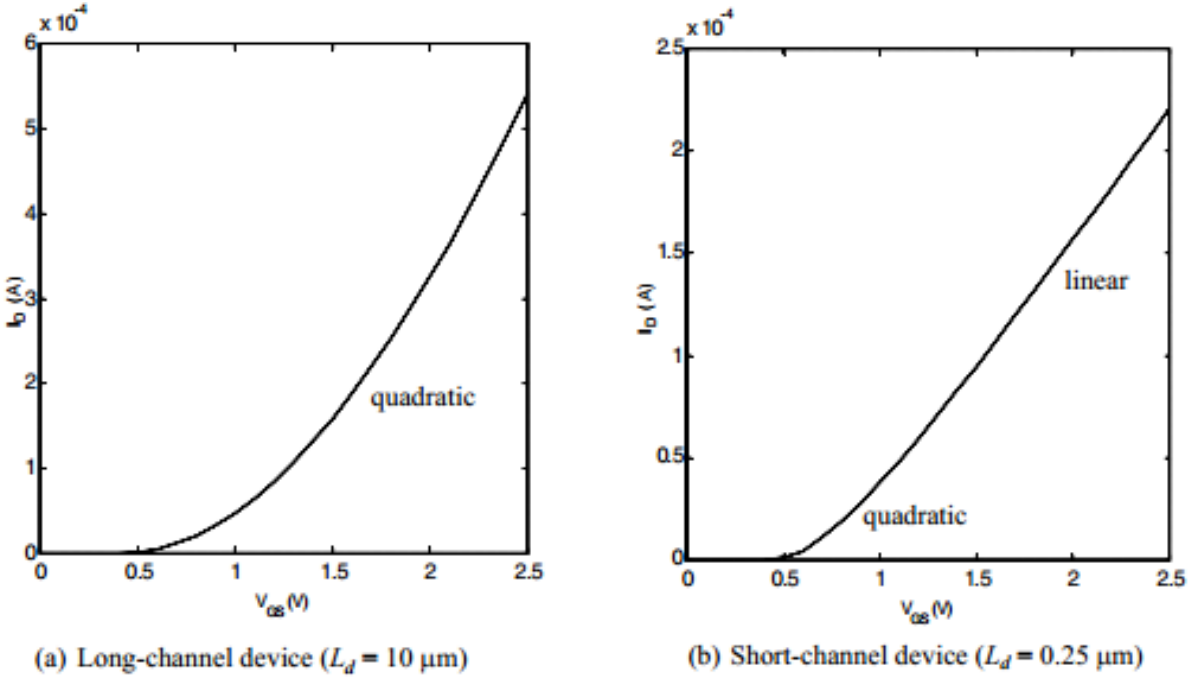


Figure 3.6: I_D versus V_{GS} (on logarithmic scale).

At the end, one should observe the following **similar** graphs as shown in Fig.3.7.

Note: The Fig.3.7 and Fig.3.6 is for the illustration purpose. You will be getting a different plot



(a) Long-channel device (L_d = 10 µm) (b) Short-channel device (L_d = 0.25 µm)

Figure 3.7: NMOS transistor $I_D - V_{GS}$ characteristic for long and short-channel devices (0.25 um CMOS technology). W/L= 1.5 for both transistors and $V_{DS} = 2.5V$.

Vth variation plot

From now, use the shorter channel spec given in above. Here there are two plots, one showing **Vth variation with Vds** and other showing **Vth variation with L**.

Note

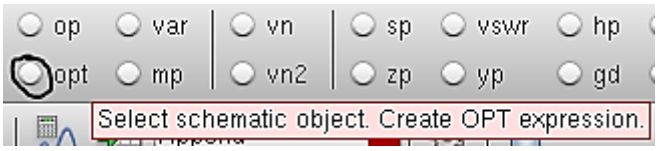
Make sure that you didn't close the ADE window, before performing the following procedure. In case, if you close it, then run a *dummy DC analysis*. This ensures that the database is built, from which we can plot the waveform defined through

`Output definition.`

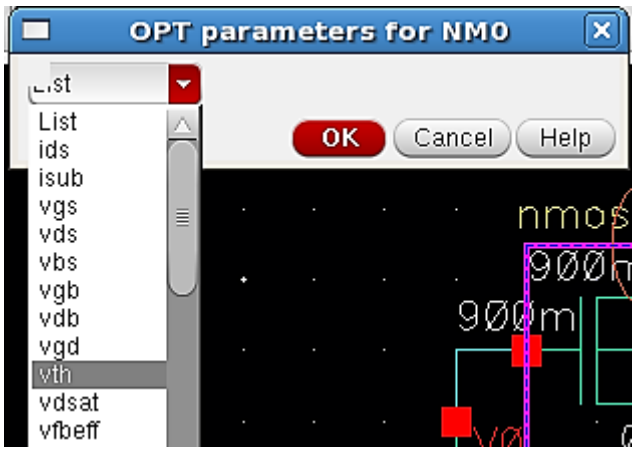
Vth variation with Vds

Use $L = 45n\ m$ and $W = 120n\ m$ for the plot.

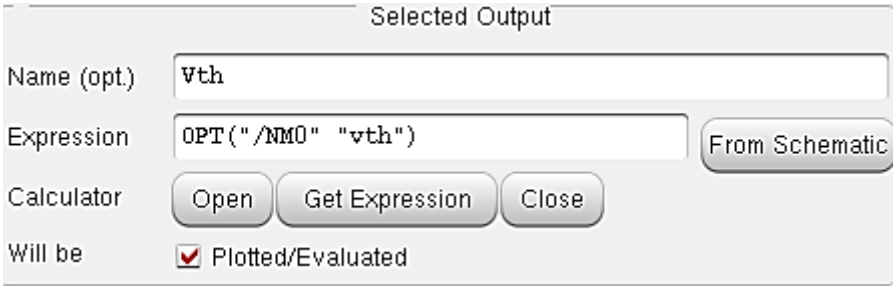
1. Analysis setup
   (a) Select `dc` analysis
   (b) Check `save DC operating point`.
   (c) Under sweep variable, select `component parameter` to choose `Vds` as DC voltage parameter.
   (d) Provide sweep range: `start-stop` from 0 to 1.0.
2. Output definition
   (a) Select `Outputs` → `Setup`
   (b) Enter `Vth` for the `Name(opt)`. label.
   (c) Click `open` which opens calculator window.
   (d) Select `OPT` radio button.



   (e) Navigate to Test schematic and select the MOS instance.
   (f) Select `Vth` from the List window.



   (g) Go to calculator window, where the buffer is filled with an expression.
   (h) To copy the expression in the output setup window, click on `Get Expression` button in the output setup window.



(i) Click the Add button to get added to the output section in the ADE window.

3. Plot

   (a) Click the `Run` button.

   (b) To plot `Vth vs Vds`, parametric analysis is done with the following specification.

Variable	Value	Sweep?	Range Type	From	To	Step Mode	Total Steps
vds	.5	☑	From/To	0	1	Auto	11

Vth variation with L

Use $W = 120n\ m, V_{DS} = V_{GS} = 0.6$(default) for the plot.

1. Analysis setup

   (a) Select `dc` analysis

   (b) Check `save DC operating point`.

   (c) Under sweep variable, select `component parameter` to choose `Vgs` as DC voltage parameter.

   (d) Provide sweep range : `start-stop` from 0 to 1.0.

2. Output definition

   (a) Since Vth is plotted against L, the `vth` output expression defined earlier can be retained.

3. Plot

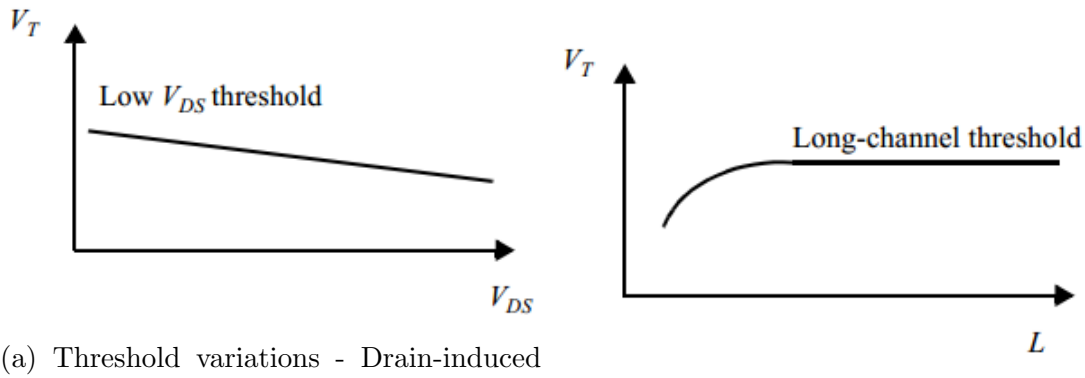
   (a) Click the `Run` button.

   (b) To plot Vth vs L, parametric analysis is done with the following specification.

Variable	Value	Sweep?	Range Type	From	To	Step Mode	Step Size
L	45n	☑	From/To	45n	10u	Linear Steps	500n

At the end, one should observe the following **similar** graphs as shown in Fig.3.8a and Fig.3.8b.

> Note: The Fig.3.8a and Fig.3.8b is for the illustration purpose. You will be getting a different plot



(a) Threshold variations - Drain-induced barrier lowering (for low L).

(b) Classical short channel effect.

Figure 3.8: Threshold voltage variation

Note

Fig.3.8b shows a classical short-channel effect - decrease of V_T with decrease of L.

In newer processes (like 180 nm technology), there is so-called **halo** or **pocket implant** , where substrate/body/channel is more heavily doped near source/drain junctions - this is done in order to suppress DIBL (decrease of Vt with increase of Vds voltage).

When L gets shorter, halo regions overlap, leading to effectively higher substrate doping, and thus higher Vt. This is called a **reverse short-channel effect** - Vt increase with decrease of L. This effect is shown in Fig.3.9

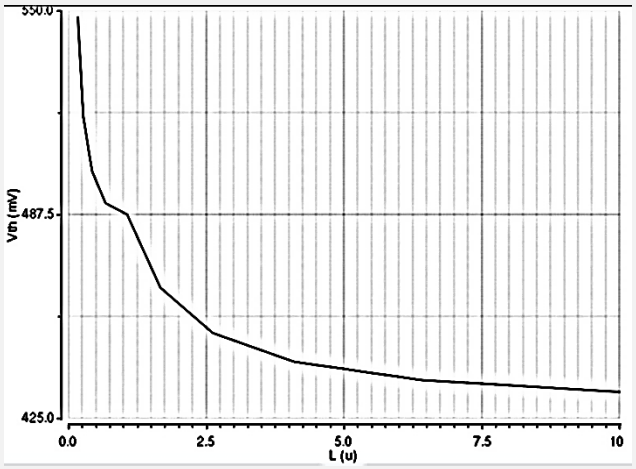


Figure 3.9: Reverse short channel effect

Resistance plot

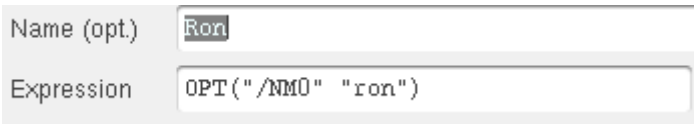
Use $L = 45n\ m$ and $W = 120n\ m$ for the plot.

1. Analysis setup
   (a) Retain the previous setup.

2. Output definition

   (a) Select `Outputs` → `Setup`

   (b) Enter `Ron` for the `Name(opt).` label and `OPT("/NM0" "ron")`.



3. Plot

   (a) Click the `Run` button.

   (b) To plot Ron vs Vgs, parametric analysis is done with the following specification.

Variable	Value	Sweep?	Range Type	From	To	Step Mode	Total Steps
vgs	.5	☑	From/To	0	1	Auto	11

At the end, one should observe the following **similar** graphs as shown in Fig.3.10a.

CV plot

Use $L = 45n\ m$ and $W = 120n\ m$ for the plot.

1. Analysis setup

   (a) Retain the previous setup.

2. Output definition

   (a) Select `Outputs` → `Setup`

   (b) Enter `Cgs` for the `Name(opt).` label and `OPT("/NM0" "cgg")`.

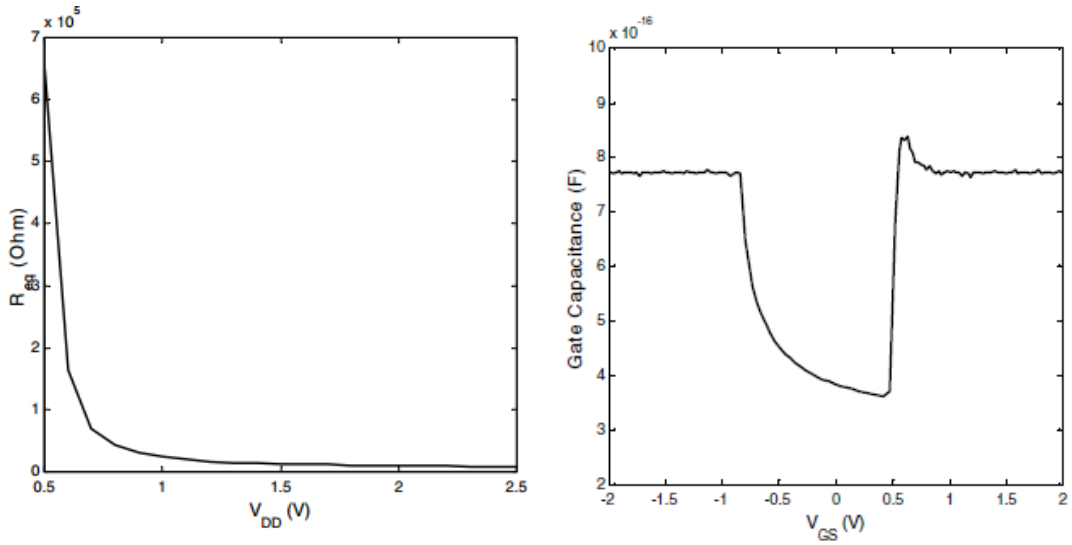
3. Plot

   (a) Click the `Run` button.

   (b) To plot Cgg vs Vgs, parametric analysis is done with the same specification as above.

At the end, one should observe the following **similar** graphs as shown in Fig.3.10b.

Note: The Fig.3.10b is for the illustration purpose. You will be getting a different plot



(a) Resistance plot

(b) CV plot

3.5 Observations

The observation should be made for NMOS as well as PMOS devices. For PMOS parameter extraction, one can use similar procedure that was used for NMOS.

3.5.1 Id Vs Vds

From plots, you might observe the following

- Velocity-saturation causes the device to saturate for substantially smaller values of V_{DS} in short channel device.
- Based on the plot shown in Fig.3.5, measure the values of I_D at $V_{DS} = 1.2V$ for both $L = 45n\ m$ & $W = 120n\ m$ and $L = 6\mu\ m$ & $W = 9u\ m$.

3.5.2 Id Vs Vgs

- Observe the short-channel device shows that I_{DSAT} has linear dependency with respect to V_{GS}.
- The log-scale plot will show the conduction of I_D even below V_T. From this plot, measure I_D at $V_{GS} = 0$ for both $L = 45n\ m$ & $W = 120n\ m$ and $L = 6\mu\ m$ & $W = 9u\ m$.

3.5.3 Vth variation

- Based on plot shown in Fig.3.8a, measure the values of V_T at $V_{DS} = 0V, 0.6V, 1.2V$.
- Based on plot shown in Fig.3.9, measure the values of V_T at $L = 45nm, 180nm, 1\mu m, 5\mu m, 10\mu m$.

3.5.4 Resistance plot

- Measure the minimum and maximum channel resistances for $L = 45n\ m$ and $W = 120n\ m$

3.5.5 CV plot

- For $L = 45n\ m$ and $W = 120n\ m$, measure the capacitance values at -1V and 1V with respect to x-axis. Also identify the minimum value capacitance using `ymin` function from the calculator and identify its corresponding x-axis value.

Part B

Practice question: Plot g_m Vs V_{gs} for NMOS/PMOS

.

SI	Criteria	Max Marks	Marks Obtained
	Data Sheet		
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
	Record		
D	Simulation / Conduction	15	
E	Analysis of the Result	15	
	Viva	40	
	Total	100	
	Scale to 10 Marks:		

Staff Signature

Experiment 4

Inverter Static Characteristics

4.1 Objectives

To design a simple CMOS inverter, and extract the static characteristics using DC analysis.

4.2 Introduction

As we are already familiar in building CMOS Logic design, now it's time to characterize it (as done for MOSFETs in the last experiment). This lab provides students how to perform DC analysis to extract important static parameters, such as Noise margin, Switching threshold voltage.

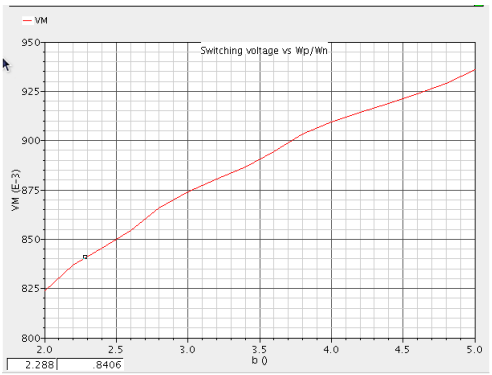
4.3 Theory

This section will provide you the basic understanding of the static parameters that are extracted from the given logic gate.

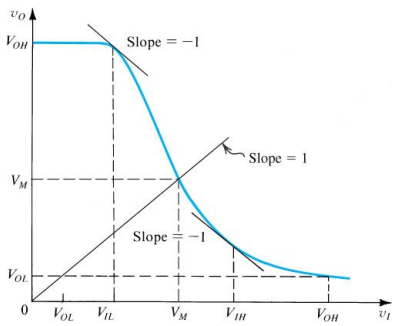
4.3.1 Switching Threshold

The switching threshold, V_M, is defined as the input voltage where $V_{in} = V_{out}$. In this region, both PMOS and NMOS are always saturated, since their $V_{DS} = V_{GS}$.

A plot of V_M Vs W_p/W_n ratio, see Fig.4.1a, shows that the V_M is relatively insensitive to variations in the device ratio.



(a) V_M versus W_p/W_n ratio (L = 45 nm , $V_{DD} = 1.8V$)



(b) Typical Voltage Transfer Characteristic (VTC) of a logic inverter

Figure 4.1: Inverter Characteristics

But by definition, the switching voltage has to be at $(V_{DD}+|V_{SS}|)/2$, which is exactly at the mid value of DC potential that drives the circuit. Since the resistance across the PMOS channel is more than the NMOS channel resistance, it takes more time to charge the capacitive load. This particular phenomenon, would drive the value of $V_M < (V_{DD}+|V_{SS}|)/2$.

Now to have equal resistance, we have to satisfy the following equation

$$\frac{K_p^{'}}{K_n^{'}} = 1 = \frac{\frac{W_p}{L_p}\mu_p C_{OX}}{\frac{W_n}{L_n}\mu_n C_{OX}} \approx \frac{\frac{W_p}{L_p}\mu_p}{\frac{W_n}{L_n}(1 \rightarrow 3)\mu_p} \tag{4.1}$$

As noted in Eq.4.1, the mobility of PMOS, μ_p, is assumed to be less than NMOS, μ_n, by a factor $1 \rightarrow 3$.

> Note: In nano-meter technologies $mu_p \geq mu_n$, which is due to less scattering in High electric field.

So from Eq.4.1, one can deduce the following W_p/W_n ratio

$$\begin{aligned} &\Rightarrow W_p/L_p \approx 1 \rightarrow 3W_n/L_n \\ &\quad \Rightarrow W_p \approx 1 \rightarrow 3W_n \quad \text{since } L_p \approx L_n \end{aligned} \tag{4.2}$$

This means that small variations of the ratio (e.g., making it 3 or 2.5 or even 2) do not disturb the transfer characteristic that much. It is therefore an accepted practice in industrial designs to set the width of the PMOS transistor to values smaller than those required for exact symmetry. For example, setting the ratio to 3, 2.5, and 2 yields switching thresholds of 0.89 V, 0.85 V, and 0.825 V, respectively.

4.3.2 Noise Margins

The static operation of a logic-circuit family is characterized by the Voltage Transfer Characteristic (VTC) of its basic inverter. Fig.4.1b shows such a VTC and defines its four parameters; V_{OH}, V_{OL}, V_{IH}, and V_{IL}. Note that V_{IH} and V_{IL} are defined as the points at which the slope of the VTC is - 1. Also indicated is the definition of the threshold voltage V_M, or $V_{T,inv}$ as we shall frequently call it, as the point at which $V_O = V_I$.

The robustness of a logic-circuit family is determined by its ability to reject noise, and the parameters that attributes are called as the noise margins, NM_H and NM_L,

$$\begin{aligned} NM_H &= V_{OH} - V_{IH} \\ NM_L &= V_{IL} - V_{OL} \end{aligned} \tag{4.3}$$

An ideal inverter is one for which $NM_H = NM_L = (V_{DD}+|V_{SS}|)/2$, where V_{DD} and V_{SS} is the +ve and -ve power-supply voltages respectively. Usually, V_{SS} will be connected to GND, and hence $V_{SS} = 0$. Further, for an ideal inverter, the threshold voltage $V_M = V_{DD}/2$.

4.4 Lab conduction Procedure

This section would guide you through the procedure of extracting the following parameters, based on `dc` analysis.

1. Switching voltage of Inverter
2. Noise Margin calculation

4.4.1 Switching voltage of Inverter

Design Schematic Modification

Since the inverter is already designed in Experiment 1, we will only do some modification to extract the parameter and retain the `symbol` of it.

1. Open the `Schematic` view of your Inverter cell, say `INVERTER`, from the `CIW` using `Tools` → `Library Manager` option
2. Once the schematic is open, right click on `pmos1v` and select `properties` (or) select `pmos1v` and press the keyboard shortcut `Q`
3. Change the `Width` parameter

`width`	Wp

> Note: No need to modify the properties of `nmos1v`. So retain it's `width: 120n`

Test Schematic Creation

Retain the Test schematic as before. For ease, it is replicated in Fig.4.2.

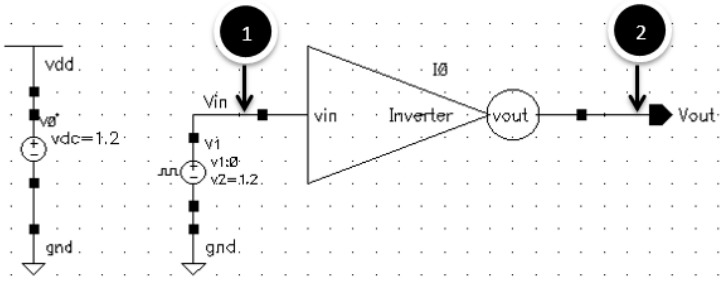


Figure 4.2: Inverter test schematic

Per-Layout Simulation

There are two steps involved in the simulation

libel=i Computation of V_M for the default transistor sizes

liibel=ii Plot V_M to optimize V_M value using parametric analysis on `Wp`

Computation of V_M for the default transistor sizes

1. Analysis setup

(a) Copy the variable defined for `pmos1v` on to ADE
`Variables` → `Copy From Cellview`

(b) Specify the value of the variable `Wp:  120n`

(c) Select `dc` analysis

(d) Check `save DC operating point`.

(e) Select `Select Component` and then click on the desired voltage source in the schematic to sweep.
In this case we want to sweep the input voltage source.

(f) Select `dc` as the variable to sweep when the popup window opens.

(g) We wish to sweep the source from the lower potential to higher potential, so input **0** into `Start` and **1** into `Stop`. Select OK.

2. Run and plot the simulation based on your previous experience. You should endup getting the plot as shown in Fig.4.3.

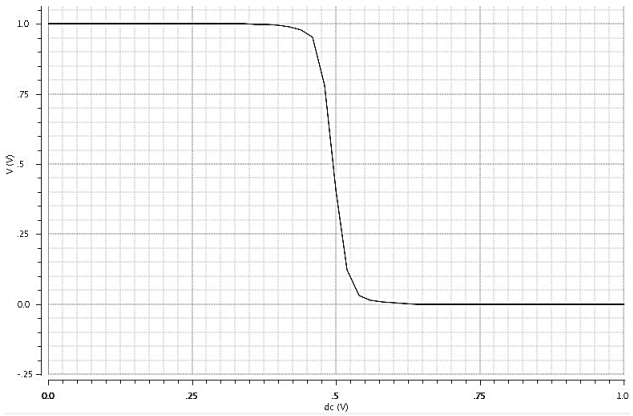


Figure 4.3: Inverter Voltage Transfer Characteristic (VTC) plot

3. Computation of V_M

(a) To access the calculator, select `Tools` → `Calculator` in the ADE.

(b) In the calculator window, select `wave` radio button and go back to the wave window and select the wave.

(c) Now from the `special function`, select `cross`. Mirror the setting as shown in Fig.4.4.

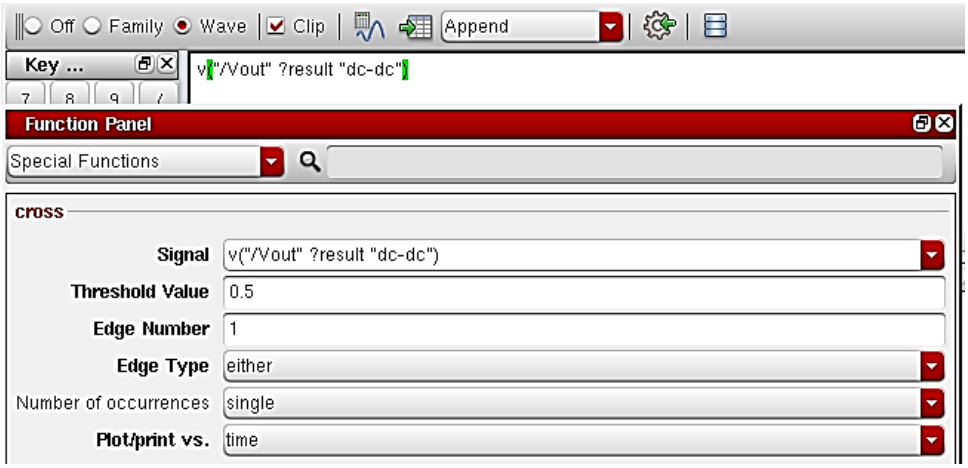


Figure 4.4: Calculator showing cross function for V_M

(d) Click Ok. Click the `Evaluate buffer` button.

V_M Plot

When you observe the V_M value (computed in the previous step), you might note that the output crosses the midpoint of the supply voltage (i.e., 0.5 V) when the input is at $V_{IN} = V_M \approx 495.1mV$. As stated earlier, we need to make the channel resistance of both the MOS transistors same to bring the $V_M = 0.5V$.

Now to perform this task, we need a parametric analysis, which varies the width of PMOS `Wp`.

Note

Make sure that you didn't close the ADE window, before performing the following procedure. In case, if you close it, then run a *dummy DC analysis*. This ensures that the database is built, from which we can plot the waveform defined through `Output definition`.

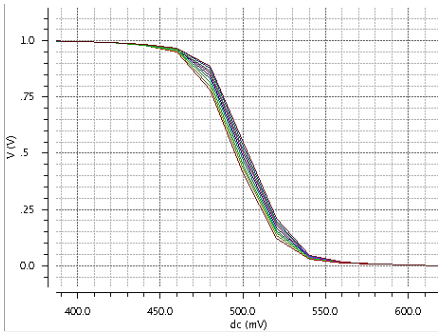
1. Now to perform the parametric analysis, in the ADE window, click `Tools` → `Parametric Analysis`. Enter the following details as shown below.

Variable	Value	Sweep?	Range Type	From	To	Step Mode	Step Size	Inclusion List	Exclusion List
wp	120n	☑	From/To	120n	160n	Linear Steps	5n		

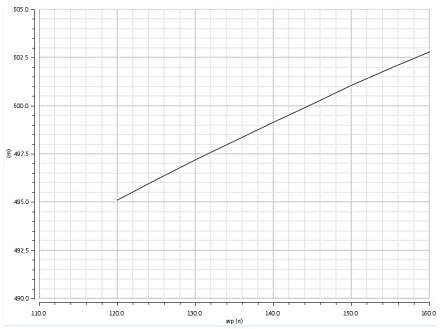
2. Go to ADE window, select `outputs` → `Setup` and define `VM` for `Name (opt)` and click on `open`.

3. This should open the calculator window. Select `vs` radio button, as shown in Fig.4.5, then go to test schematic and select the node 2 as shown in Fig.4.2.

Figure 4.5: DC swept button



(a) Parametric Result



(b) V_M Vs W_p plot

Figure 4.7: Plots for V_M

4. Press `ESC` and return the calculator window.

5. Under special function, select `cross` function and specify the same setting as shown in Fig.4.4. Click OK.

6. Switch back to the `setting output` window and click `Get Expression` to reflect the setting as in Fig.4.6.

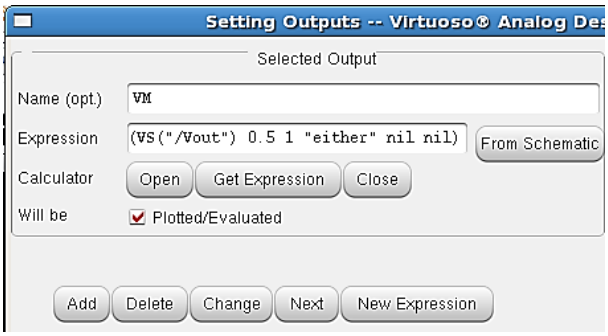


Figure 4.6: Switching voltage expression for W_p varitions

7. Now when you click on the play button, it provides us with two plots, as shown in Fig.4.7a and Fig.4.7b.

Noise Margin calculation

Based on the observation given in Section.4.5.1, use W_p value calculated for $V_M = V_{DD}/2$ in the following procedure.

Since this doesn't require any Schematic or Output Expression, we will directly get into *Pre-Layout simulation* step.

1. Pre-Layout simulation

   (a) In the ADE window, define the value of `Wp` calculated above.

   (b) Run the simulation. This should bring us VTC plot again, but with V_M at $V_{DD}/2$.

   (c) From the DC plot, select `Tools` → `Calculator`.

(d) In the calculator window, select `wave` radio button and go back to the wave window and select the wave.

(e) From `special function` menu, select `deriv` function. This should result in Fig.4.8.

Figure 4.8: Calculator showing `deriv` function in the buffer

(f) Click `Evaluate buffer`, which results in Fig.4.9.

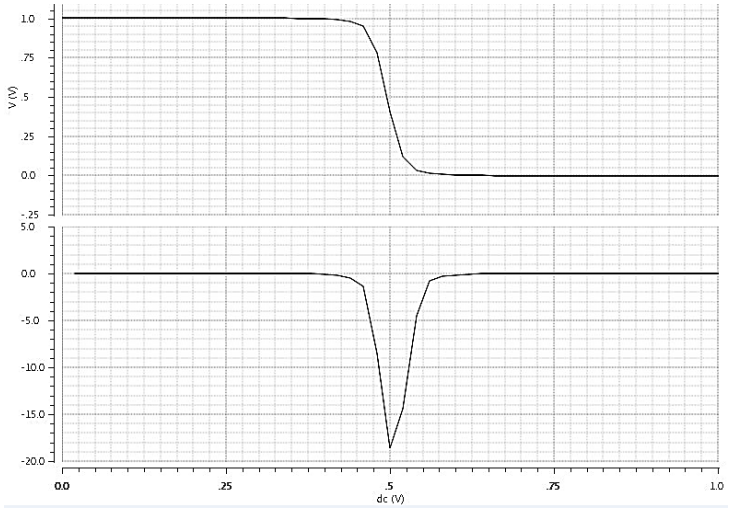


Figure 4.9: DC Plot with derive function

(g) Now from the `special function`, select `cross`. Mirror the setting as shown in Fig.4.10.

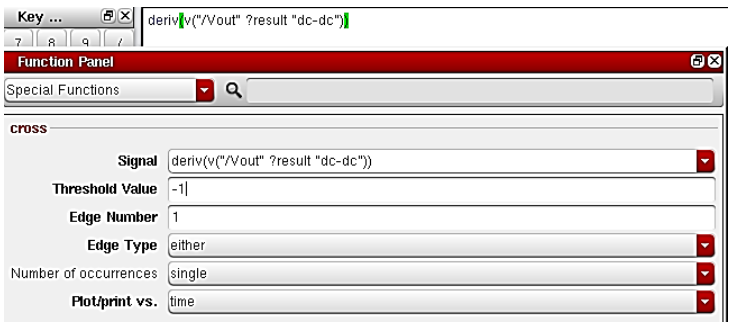


Figure 4.10: Calculator showing cross function for slope calculation

(h) Click Ok. Click the `Evaluate buffer` button. Note down the V_{IL} value.

(i) Just change the `Edge Number` to **2** and click `Evaluate buffer` to measure V_{IH}.

4.5 Observations

4.5.1 Switching voltage

From plots, you might observe the following

- Calculate the W_p for $V_M = V_{DD}/2$.
- Derive the conclusion from the Switching Threshold section and draw the inverter circuit specifying the widths of PMOS and NMOS, which are normalized with respect to NMOS width.

4.5.2 Noise Margin

- Calculate Noise Margin:
  $NM_L =$ _____; $NM_H =$ _____

Part B

Practice question: Plot the VTC graph of CMOS inverter and calculate W_p for $V_M = 0.6 \times V_{DD}$

.

SI	Criteria	Max Marks	Marks Obtained
	Data Sheet		
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
	Record		
D	Simulation / Conduction	15	
E	Analysis of the Result	15	
	Viva	40	
	Total	100	
	Scale to 10 Marks:		

Staff Signature

Experiment 5

Sequential Circuit Design using Master-Slave configuration

5.1 Objective

To design the basic CMOS sequential circuits namely D Latch, D Master-Slave flip-flop, and verify its functional behaviour.

5.2 Introduction

This experiment discusses the CMOS implementation of the most important sequential building blocks. As you might know from ADDC course, a static sequential circuit is built by making use of feedback. As the name suggest us that it doesn't require any refreshing circuit to hold it's Data.

In this lab we will be studying 2 basic variants of static sequential circuits. Their functional behaviour is verified using transient analysis.

5.3 Theory

There are basically 2 variants of static sequential circuits, namely:

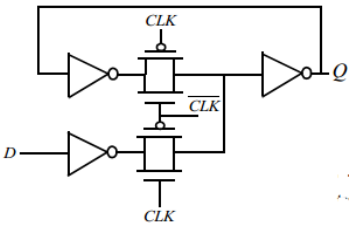
1. Clocked Latch (or) Level-sensitive Flip-flop
2. Master-Slave Edge-Triggered Flip-flop

5.3.1 Clocked Latches or Level-sensitive Flip-flops

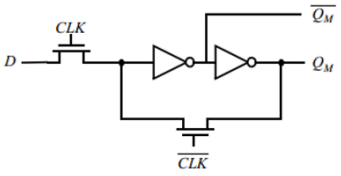
We are going to design *Clocked D Latch* circuit in this lab. There are many approaches for constructing latches. One very common technique involves the use of transmission gate multiplexers.

A transistor level implementation of a positive latch based on multiplexers is shown in Fig.5.1a.

It is possible to reduce the clock load to two transistors by using implement multiplexers using NMOS only pass transistor as shown in Fig.5.1b. The advantage of this approach is the reduced clock load of only two NMOS devices. While attractive for its simplicity, the use of NMOS only pass transistors results in the passing of a degraded high voltage of $V_{DD} - V_{Tn}$ to the input of the first inverter. This impacts both noise margin and the switching performance, especially in the case of low values of V_{DD} and high values of V_{Tn}. It also causes static power dissipation in first inverter. Since the maximum input-voltage to the inverter equals $V_{DD} - V_{Tn}$, the PMOS device of the inverter is never turned off, resulting is a static current flow.



(a) Positive latch built using TGs



(b) Positive latch built using NMOS pass transistors

Figure 5.1: Latch using TGs and nmos Pass transistors

Fig.5.2 shows the non-overlapping clock signals that are used for both the types of latches which are discussed above.

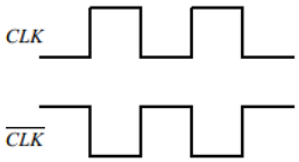


Figure 5.2: Non-overlapping Clock inputs

5.3.2 Master-SlaveEdge-Triggered Register

The most common approach for constructing an edge-triggered register is to use a master-slave configuration, as shown in Fig.5.3. The register consists of cascading a negative latch (master stage) with a positive latch (slave stage).

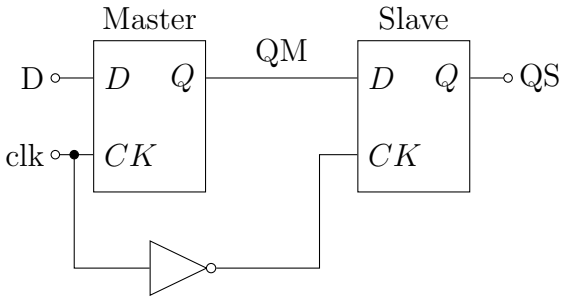


Figure 5.3: Positive edge-triggered register based on a master-slave configuration.

The output of this flip flop appear at the negative edge of the clock, even when the data is latched during the positive edge. So the output is always postponed.

5.4 Lab conduction Procedure

In this section we will built and verify the functionality using transient analysis of the following static sequential logic circuits.

1. Clocked D Latch
2. Positive Edge-triggered Master-slave D Flip flop

5.4.1 Clocked D Latch

Both the positive and negative of latches are to be realized.

Design Schematic and Symbol

With the experiences gained from the previous experiments, built positive and negative latches using Transmission gates (Refer Fig.5.1a)

Test Schematic

Use the Test schematic, shown in Fig.5.4, along with the simulation environmental setup table (Table.5.1) to test both the latches.

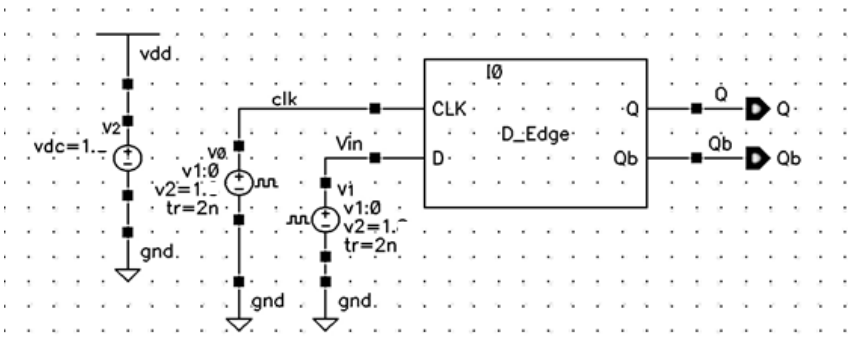


Figure 5.4: Test schematic for Sequential logics

Table 5.1: Library elements

Library	Cell Name	Instance Name	Specification
AnalogLib	`vdc`	V2	DC voltage: 1.2 V
AnalogLib	`vpulse`	V1(drives D)	Voltage 1: 0 V; Voltage 2: 1.2 V; Period: 80n s; Pulse width: 40n s; Rise time: 2n s; Fall time: 2n s; Delay = 0n s
AnalogLib	`vpulse`	V0(drives clk)	Voltage 1: 0 V; Voltage 2: 1.2 V; Period: 40n s; Pulse width: 20n s; Rise time: 2n s; Fall time: 2n s; Delay = 12n s
<Your Lib>	<Your Design>	I0	-

Pre-Layout Simulation

1. Analysis setup
   (a) In the ADE window, Select `tran` analysis
   (b) Provide `stop time` as 300n.
   (c) Check `moderate` as Accuracy parameter.
   (d) Click `OK`.
2. Output definition

(a) Select `Outputs` → `To be Plotted` → `Select on Schematic`

(b) Go to schematic and select the `Vin`, `clk`, `Q`, and `Qb` nets.

(c) Press `Ecs` key.

3. Plot

(a) Click the `Run` button

5.4.2 Positive Edge-triggered Master-slave D Flip flop

Design Schematic and Symbol

Based on Fig.5.3, built a Master-Slave positive Edge triggered flip-flop and it's corresponding symbol.

Test Schematic

Use the Test schematic, shown in Fig.5.4, along with the simulation environmental setup table (Table.5.1) to test both the latches.

Pre-Layout Simulation

Follow the steps used in the previous procedure to perform transient analysis.

5.5 Observations

- Built the characteristic table for sequential circuits given in this experiment.
- Verify the functionality of the sequential circuits built.

Part B

Practice question: Realize 4-bit Ring Counter and Johnson counter

.

SI	Criteria	Max Marks	Marks Obtained
Data Sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation / Conduction	15	
E	Analysis of the Result	15	
	Viva	40	
	Total	100	
	Scale to 10 Marks		

Staff Signature

Experiment 6

Inverter layout and post simulation

6.1 Objective

To build layout for the inverter, extract RC parameters, and perform post simulation to calculate the propagation delay.

6.2 Introduction

In this lab we will create physical layout of an inverter. In general, it is always advisable to create the schematic of a design before creating the layout, because errors in layout are difficult to fix and sometimes even to detect, whereas it is much easier to modify errors in schematics. Even though we had our Inverter designed in Exp.1, in this experiment, we use `gpdk180` technology.

6.3 Theory

6.3.1 Integrated circuit layout

Integrated circuit layout, also known **IC layout**, **IC mask layout**, or **mask design**, is the representation of an integrated circuit in terms of *planar geometric shapes* which correspond to the patterns of metal, oxide, or semiconductor layers that make up the components of the integrated circuit.

When using a standard process - where the interaction of the many chemical, thermal, and photographic variables are known and carefully controlled - the behaviour of the final integrated circuit depends largely on the positions and interconnections of the geometric shapes. Using a computer-aided layout tool, the layout engineer—or layout technician—places and connects all of the components that make up the chip such that they meet certain criterion—typically: performance, size, density, and manufacturability. This practice is often subdivided between two primary layout disciplines: Analog and Digital.

The generated layout must pass a series of checks in a process known as physical verification. The most common checks in this verification process are

1. Design Rule Check (DRC),
2. Layout Versus Schematic (LVS),
3. Parasitic Extraction,
4. Antenna Rule Check, and
5. Electrical Rule Check (ERC).

When all verification is complete, the data is translated into an industry standard format, typically **GDSII**, and sent to a semiconductor foundry. The process of sending this data to the foundry is called **tapeout**, due to the fact the data used to be shipped out on a magnetic tape. The foundry converts the data into another format and uses it to generate the photomasks used in a photolithographic process of semiconductor device fabrication.

6.3.2 Physical Verification

Out of the verification processes that are listed, we will focus on the first 3 processes here.

Before we discuss about the verification, we should know about design rules. Thus we start off with a brief note about design rules followed by design verification.

Design Rules

Design Rules are a series of parameters provided by semiconductor manufacturers that enable the designer to verify the correctness of a mask set. Design rules are specific to a particular semiconductor manufacturing process. A design rule set specifies certain geometric and connectivity restrictions to ensure sufficient margins to account for variability in semiconductor manufacturing processes, so as to ensure that most of the parts work correctly.

DRC and LVS Design Rule Check (DRC) and Layout Versus Schematic (LVS) are verification processes. Reliable device fabrication at modern deep-submicrometer (0.13 μm and below) requires strict observance of transistor spacing, metal layer thickness, and power density rules.

DRC exhaustively compares the physical netlist against a set of "foundry design rules" (from the foundry operator), then flags any observed violations.

The LVS process confirms that the layout has the same structure as the associated schematic; this is typically the final step in the layout process. The LVS tool takes as an input a schematic diagram and the extracted view from a layout. It then generates a netlist from each one and compares them. Nodes, ports, and device sizing are all compared. If they are the same, LVS passes and the designer can continue. LVS tends to consider transistor fingers to be the same as an extra-wide transistor. Thus, 4 transistors in parallel (each 1 μm wide), a 4-finger 1 μm transistor, and a 4 μm transistor are viewed as the same by the LVS tool.

6.3.3 Parasitic Extraction

In electronic design automation, parasitic extraction is calculation of the parasitic effects in both the designed devices and the required wiring interconnects of an electronic circuit: detailed device parameters, parasitic capacitances, parasitic resistances and parasitic inductances, commonly called parasitic devices, parasitic components, or simply parasitics.

The major purpose of parasitic extraction is to create an accurate analog model of the circuit, so that detailed simulations can emulate actual digital and analog circuit responses. Digital circuit responses are often used to populate databases for signal delay and loading calculation such as: timing analysis; circuit simulation; and signal integrity analysis. Analog circuits are often run in detailed test benches to indicate if the extra extracted parasitics will still allow the designed circuit to function.

6.4 Lab conduction Procedure

This section would guide you to

1. Build the Inverter Layout
2. Perform post layout simulation

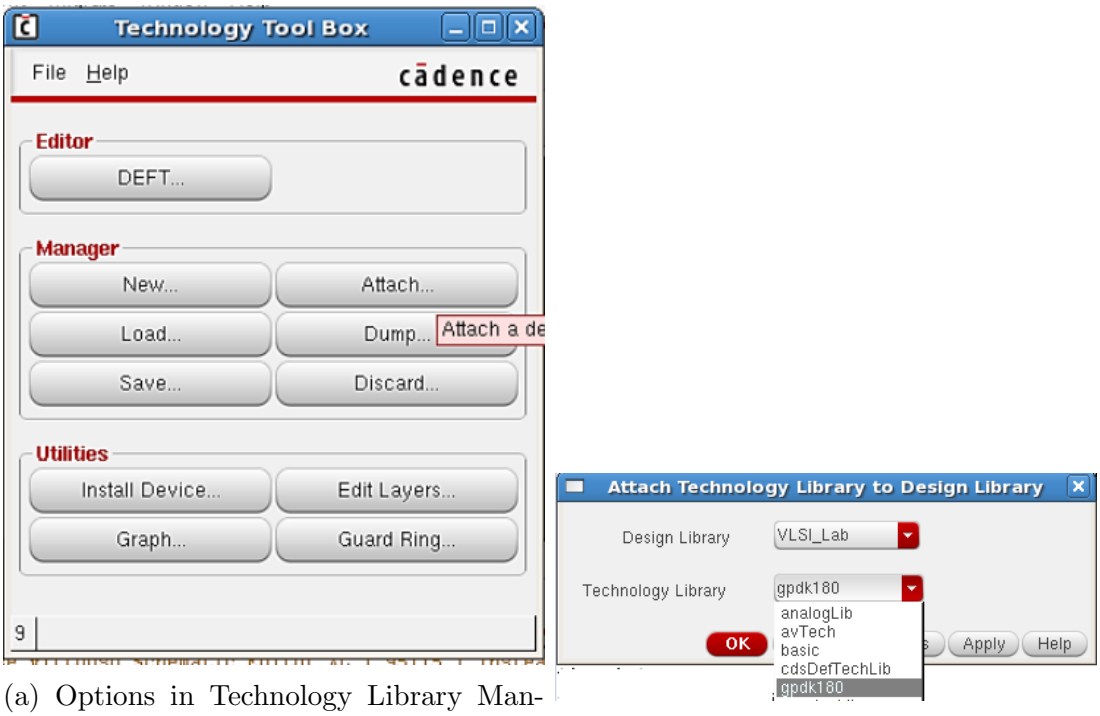
6.4.1 Inverter using gpdk180 Technology

Since we are going to design the layout using `gpdk180` technology, it necessary to create the inverter cell again.

6.4.2 User Library modification

Since we are going to use `gpdk180` technology, the user library (say, `VLSI_Lab`) has to be reattached with `gpdk180` from `gpdk045` technology.

1. In CIW, go to `Tools` → `Technology File Manager ...`. You would end up having a window as shown in Fig.6.1a.



(a) Options in Technology Library Manager

(b) Change the attached Technology Lib

Figure 6.1: Technology Library Manager

2. Select `gpdk180` from the options as shown in Fig.6.1b

6.4.3 Design Schematic creation

Build the design schematic of Inverter cell using the specification from Table.6.1.

Table 6.1: Design Schematic Library elements

Library	Cell Name	Instance Name	Specification
AnalogLib	`vdd`	-	-
gpdk180	`pmos`	PM0	Total width: 2u m; Length: 180n m
AnalogLib	`gnd`	-	-
gpdk180	`nmos`	NM0	Total width: 2u m; Length: 180n m

6.4.4 Test schematic creation

Build the test schematic based on Fig.6.2 using the specification given in Table.1.1.

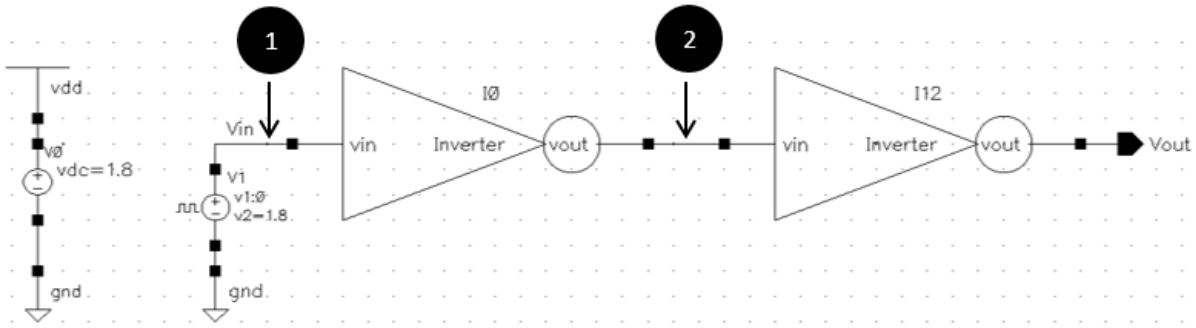


Figure 6.2: Inverter Test Schematic

Note:The 2^{nd} Inverter serves as a capacitive load for the 1^{st} Inverter

6.4.5 Pre-Layout simulation

In order to compute the propagation delay, t_{cd}, we need to perform `transient` analysis.

1. Analysis setup
   (a) In the ADE, Select `Analyses` → `Choose`.
   (b) Select analysis section as `trans`, set the `stop time` as **40n**,
   (c) Click on the enable button and then click apply.
2. Output definition
   (a) Select `Outputs` → `To be Plotted` → `Select on Schematic`
   (b) Go to schematic and select the output terminal of the 1^{st} Inverter (Marked as **2** in Fig.6.2) to plot.
   (c) Press `Ecs` key.
3. Plot

(a) Click the `Run` button in ADE

4. Computation of propagation delay between input and output pin.

(a) From the plot window, go to `Tools` → `Calculator`

(b) Select `delay` from the `Function Panel`. You end up having options as shown in Fig.6.3

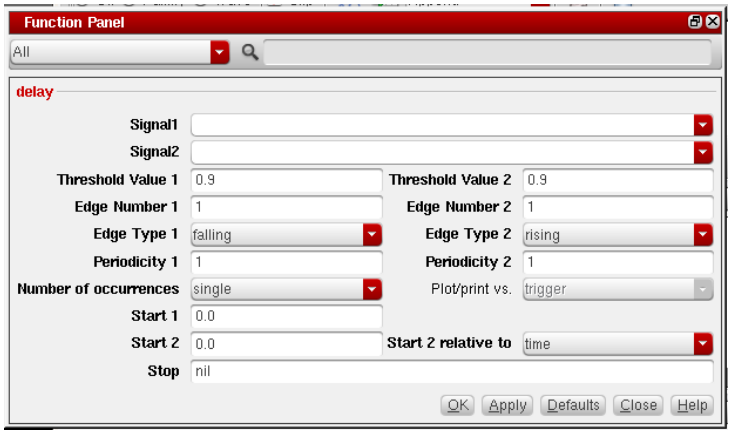


Figure 6.3: Options in Delay function

(c) Specify the following entries and replicate the other values as shown in Fig.6.3

Parameter	Comment
Signal1	Specify the input signal of the 1^{st} inverter
Signal2	Specify the output signal of the 1^{st} inverter

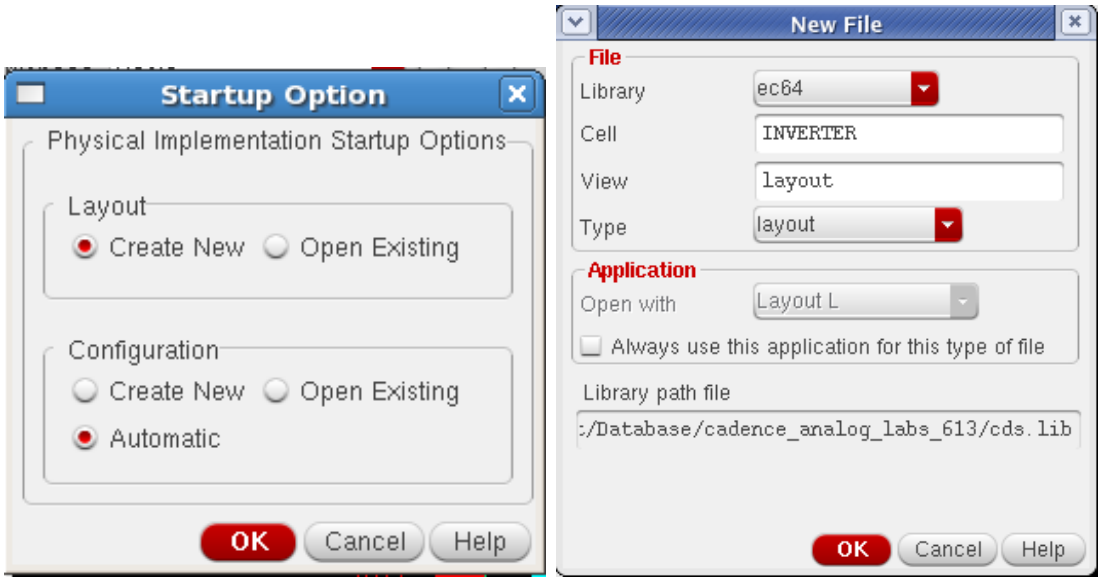
(d) Click `OK` and in the `Calculator` window, click `Evaluate`

6.4.6 Design Layout creation

After the pre-layout simulation, it is time to move onto the layout of the circuit.

1. From the *Schematic Editor* of Design cell (Not the Test Schematic) select `Launch` → `Layout XL`.

2. A `Startup Option` window pop-ups, in which select, `Create New` in the Layout option and `Automatic` in the Configuration. Select Layout as the Type (Fig.6.4a).



(a) Startup Option (b) Creating a layout cell view

Figure 6.4: Layout creation

3. After clicking OK in Fig.6.4b, Virtuoso Layout XL editor should open as well as the Layer Selection Window (LSW) (Fig.6.5).

Short Notes on MOSFET Layout

The layout consists of rectangles, instances, and pins. A rectangle is used to create gate, diffusion, and metal regions for the transistor. The gate region is created by drawing a rectangle with poly (drw) or "r" on the keyboard. The diffusions for a transistor are created by drawing a rectangle with the active (drw) layer. The intersection of poly and active regions defines the size (length and width) of the transistors. In order to define whether a transistor is NMOS or PMOS, nselect (drw) or pselect (drw) needs to surround the active area.
Note your Schematic editor window displays `Navigator & Property` window along its side. You could note that the Navigator window shows the instance's names that are available in the schematic editor along with a Red cross symbol. These symbols turns into Green correct symbol when the corresponding instances are added into the Layout Editor.

4. From the *Layout Editor*, select `Connectivity` → `Generate` → `All From Source...`
5. Accept the default settings of `Generate Layout` window by clicking `OK`.
6. Use `SHIFT + key \f"` to get the layered view of the transistors.
7. As stated early, you can use `Rectangle` or `Path` option for interconnect.
   But, it is strongly recommended to use `Path` for wiring.
   Use `Rectangle` wherever 'Via' or 'Power rails' comes.
8. `Substrate contacts` and `vias` between layers of metal can be drawn by selecting `Create` → `Contact`, or by using the `o` hotkey.

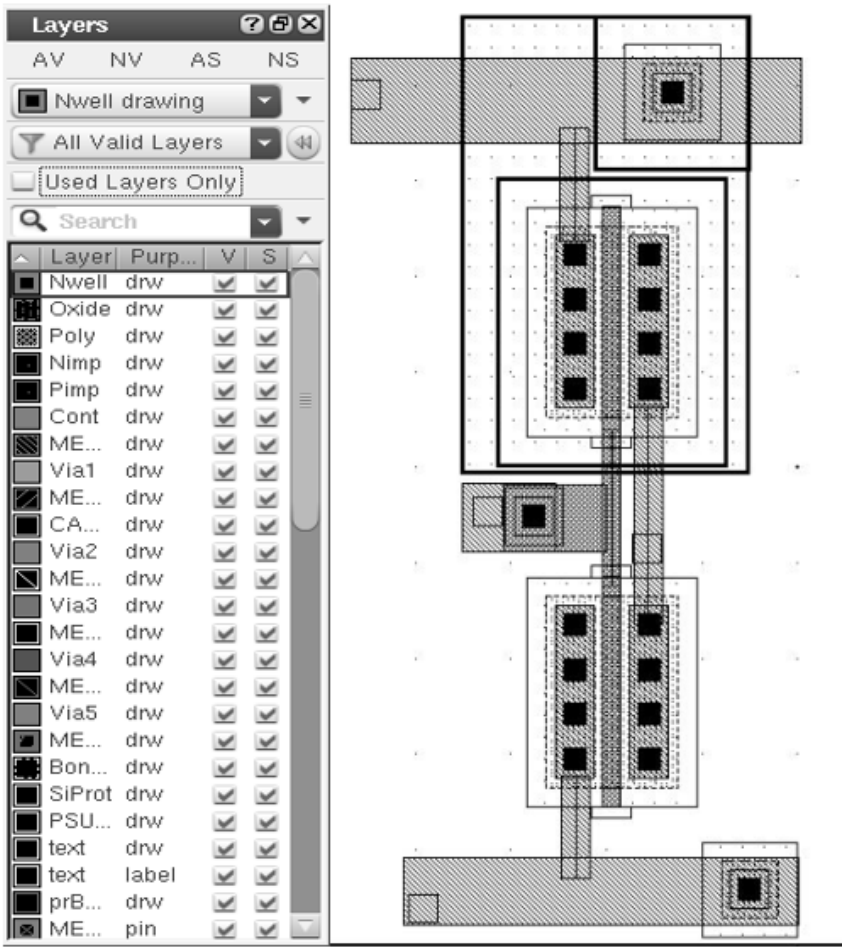


Figure 6.5: LSW & Inverter Layout

9. To add a pin, select `Create` → `Pin`.
   `Terminal Name` should be same as the name of the pin in the schematic.
   Make sure that the `Display Pin Name` option is selected so that the pin name will appear in the layout.
   `Pin Type` should be the same as the metal layer that it is connecting to.

10. After completing the above steps, you should obtain a layout of the inverter which resembles the layout shown in Fig.6.5

6.4.7 Physical Verification

There are basically 2 important verification need to be done, before further processing of Design cell.

Design Rule Check (DRC)

1. Select `Assura` → `Run DRC`.

2. Provide a Run Name, say for example, `run1`.

3. Make sure that the Technology option is ticked and `gpdk180` option is selected.

4. Select `OK` to run the DRC.

5. The total number of errors will show up in the `Error Layer Window` as seen in Fig.6.6.

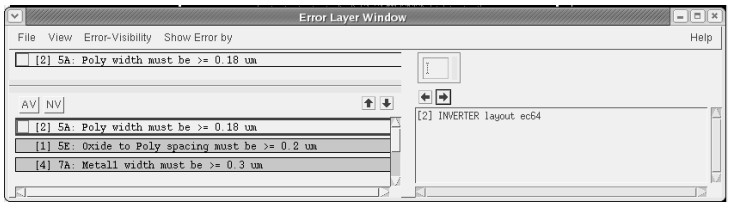


Figure 6.6: Error Layer Window

6. Before going further we have to reduce the number of errors to 0.
   Navigate through the errors and adjust the layout to fix these errors.

7. Rerun DRC until you have no design rule errors

Layout Versus Schematic (LVS)

1. Select on `Assura` → `Run LVS`.

2. Click `Run`.

3. If there is mismatch between Layout and Schematic, a Window pops up with Error.

4. Adjust the layout to match, rerun DRC, and LVS until the netlists match.

6.4.8 Parasitic Extraction

1. Once DRC is completed, Select `Assura` → `Run RCX`.

2. Click `OK` again to extract the layout parasitics to make the extracted netlist. This step will create another View, `av_extracted` as shown in Fig.6.7.

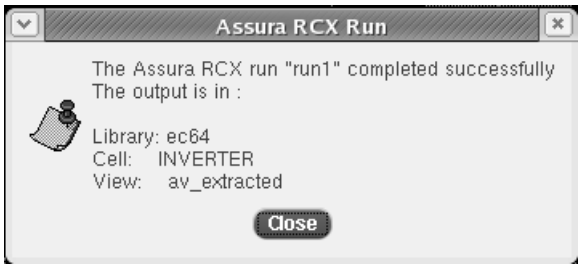


Figure 6.7: RCX Extraction

6.4.9 Configuration Cell View

Before moving on to the Post-Layout simulation, we need to create `config` view for the **Test cell**.

1. From the CIW, select `File` → `New` → `Cellview`

2. Replicate the settings shown in Fig.6.8.

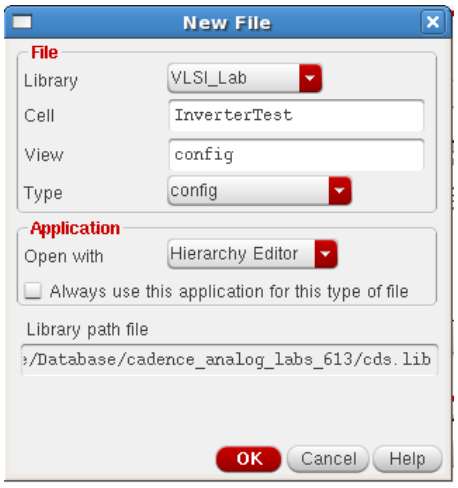


Figure 6.8: Configuration view Creation for Test cell

3. Click `OK`, which should pop up 2 windows: 1. Hierarchy Editor and 2. New Configuration.

4. Click `Use Template` option and select `spectre`, as shown in Fig.6.9.

5. After consecutive clicks on `OK`, you end up having `Table View` tab filled with all cells used in the Test schematic.

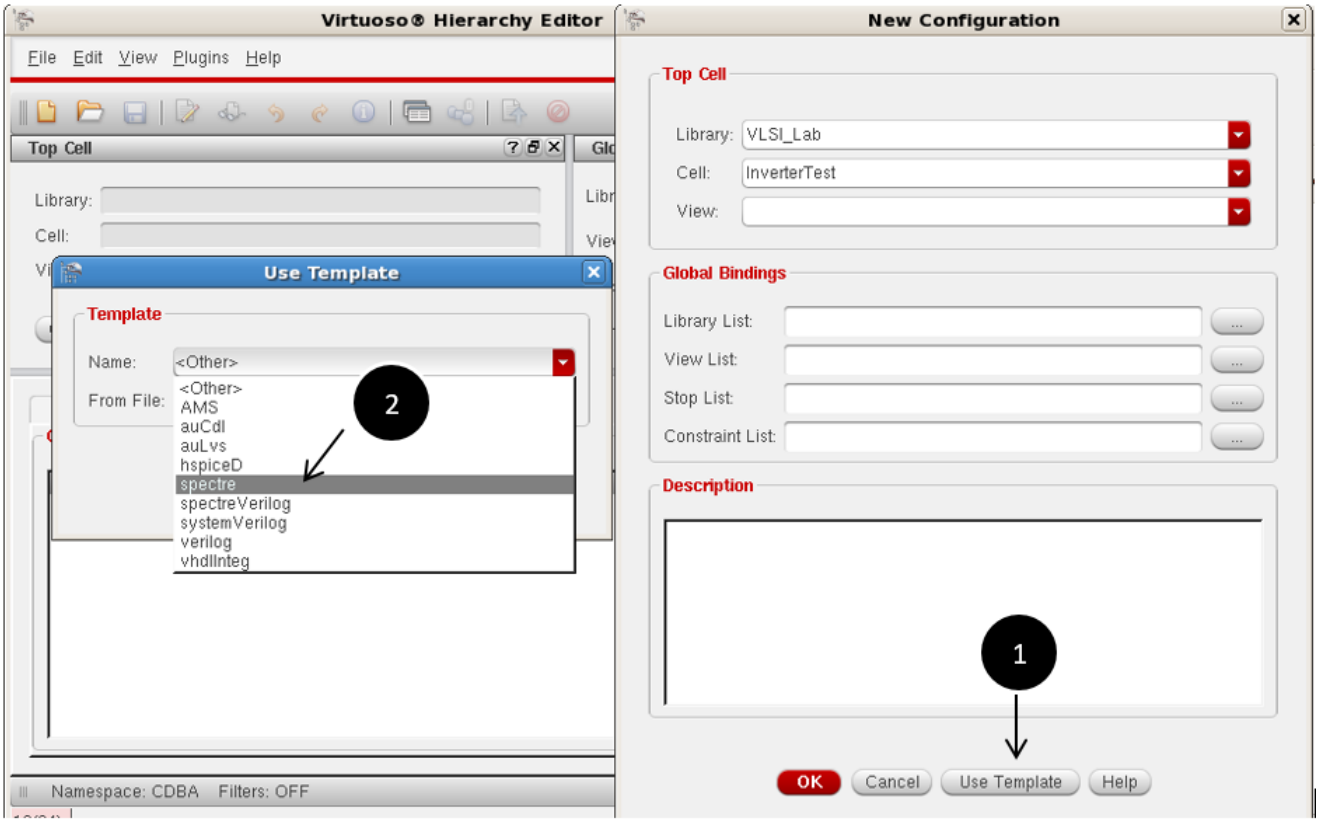


Figure 6.9: Hierarchy Editor for Test cell

6. Right click on the Inverter cell and change the `View Found` to `av_extracted`, as shown in Fig.6.10.

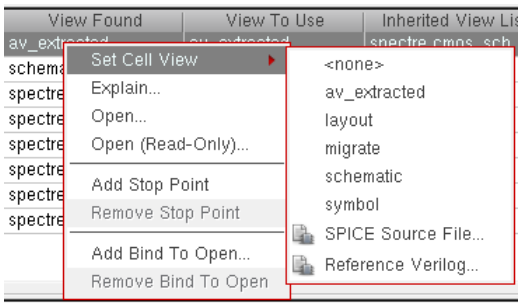


Figure 6.10: Changing the view of Inverter cell

7. Click `Save` to make the change of view to be effective

8. Click `Open` in the Hierarchy editor, which should open the Schematic of Test cell.

> Note: The Inverter symbol in the Test cell now points to `av_extracted` view. You can change it to point it back to `schematic` view by changing the view in the Hierarchy editor.

6.4.10 Post-Layout simulation

1. Perform the same procedure followed in Section.6.4.5 to evaluate the propagation delay.

6.5 Observation

- Measure the timing responses of both pre- and post-layout propagation results.
- Comment the reason for the differences in their reading.

.

SI	Criteria	Max Marks	Marks Obtained
Data Sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation / Conduction	15	
E	Analysis of the Result	15	
	Viva	40	
	Total	100	
	Scale to 10 Marks		

Staff Signature

Experiment 7

NOR/NAND gates layout and post simulation

7.1 Objective

To build layout for two input NAND & NOR gates, extract RC parameters, and perform post simulation to calculate the propagation delay

7.2 Introduction

This experiment is intended to learn how a standard cell library is designed for ASIC(Application Specific Integrated Circuits) design of digital system.

The standard cells, typically have a common height but variable widths. As shown in Fig.7.1, there are strict rules that are used while designing it.

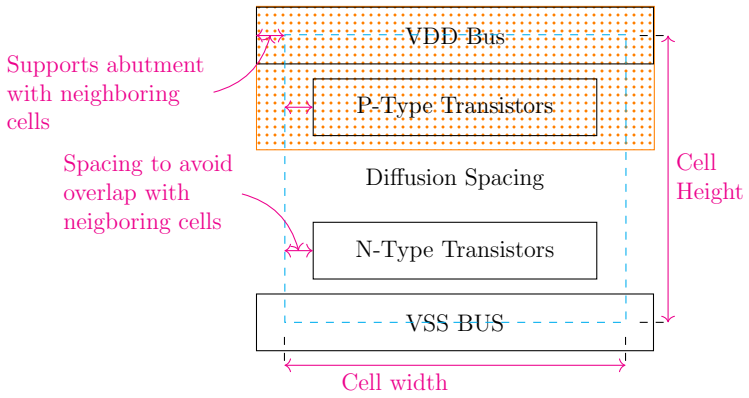


Figure 7.1: Typical layout of standard cell

Nevertheless, this lab will NOT follow those strict procedures, but just the same way followed in the previous experiment.

7.3 Theory

For a 2-input NAND (or) NOR gate, there will be a total of 6 possible output transitions that can happen when a single or both inputs makes transition.

So one has to compute 6 propagation delay for different input conditions. The computation of these delay should show us the delay dependence on the input pattern. In the Fig.7.2, a 2 input NAND gate is considered, showing such dependence delay for the different input conditions.

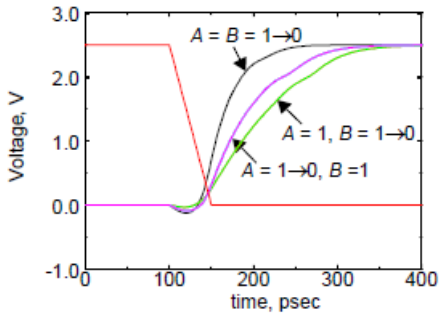


Figure 7.2: 2-input NAND gate showing the delay dependence on input patterns

7.4 Lab conduction Procedure

In this section, the layout of both NAND & NOR gates are realized using `gpdk180` technology library.

Based on the previous experiment, one has to perform the several steps, which are summarized as follows

1. Design Schematic creation
2. Test Schematic creation
3. Pre-Layout simulation
4. Parasitic extraction
5. Configuration view creation
6. Post-Layout simulation

7.5 Observation

- Compute the delay for different input patterns given below

Input Pattern	Pre-Layout Delay	Post-Layout Delay
$A = B = 0 \rightarrow 1$		
$A = 1, B = 0 \rightarrow 1$		
$A = 0 \rightarrow 1, B = 1$		
$A = B = 1 \rightarrow 0$		
$A = 1, B = 1 \rightarrow 0$		
$A = 1 \rightarrow 0, B = 1$		

Part B

Practice question: Realize AND/OR gates

.

SI	Criteria	Max Marks	Marks Obtained
Data Sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation / Conduction	15	
E	Analysis of the Result	15	
	Viva	40	
	Total	100	
	Scale to 10 Marks		

Staff Signature

Experiment 8

Common source single stage amplifier and Differential amplifier

8.1 Objective

To build a common source and differential pair amplifiers, and perform large signal and small signal analysis to measure its parameters.

8.2 Introduction

Amplification is an essential function in most analog (and many digital) circuits. We amplify an analog signal because it may be too small - to drive a load, or to overcome the noise of a subsequent stage, or to provide logical levels to a digital circuit. Amplification also plays a critical role in feedback systems.

In this lab, we study the large signal and the small-signal characteristics of Common Source (CS) amplifier and differential amplifier.

8.3 Theory

This section provides a brief overview of CS amplifier with current source load, thus possibly increasing the headroom for the output voltage swing than compared to other topologies. This is followed by a discussion about the differential amplifier, with current mirror load, providing a single ended output.

8.3.1 CS amplifier with current source load

As stated before, with current source load, the amplifier gain increases and also we can achieve a large output swing. The gain of the circuit given in Fig.8.1 is,

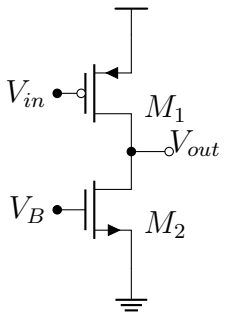


Figure 8.1: PMOS CS amplifier

$$A_v = -g_{m1}\,(r_{o1} \parallel r_{o2}) \tag{8.1}$$

The key point is that the output impedance and the minimum required $|V_{DS}|$ of M_1 are less strongly coupled than the value and voltage drop of a resistor. The voltage $|V_{DS,min}| = |V_{GS2} - V_{TH2}|$ can be reduced to even a few hundred millivolts by simply increasing the widths of M_2. If r_{o2} is not sufficiently high, the length and the width of M_2 can be increased to achieve a smaller λ while maintaining the same overdrive voltage. The penalty is the large capacitance introduced by M_2 at the output node.

8.3.2 Differential pair with active mirror load

This section describes the behaviour of the differential pair for large signal and small signal input conditions, thus studying the DC and AC conditions required for biasing and amplification respectively.

The Differential amplifier along with large-signal differential behavior is depicted in Fig.8.2

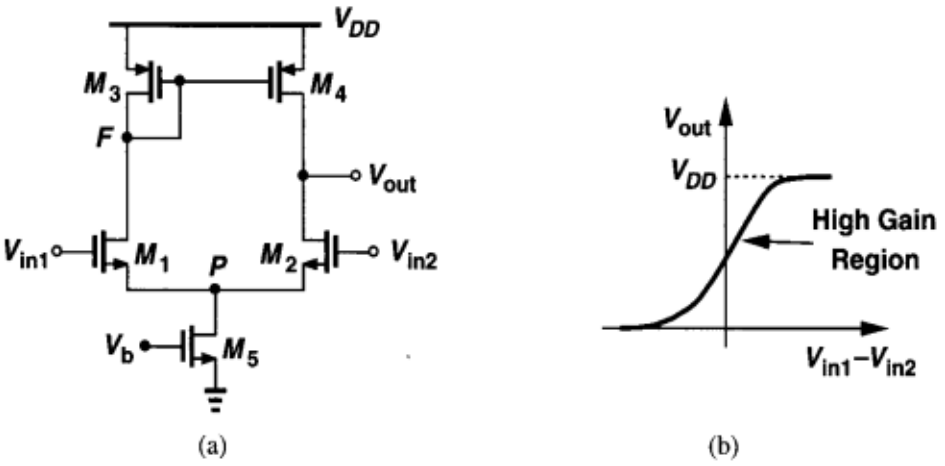


Figure 8.2: Differential amplifier with differential large-signal behavior

Also one can understand the Common-mode behavior with the following set of Large-signal graphs from the Fig.8.3.

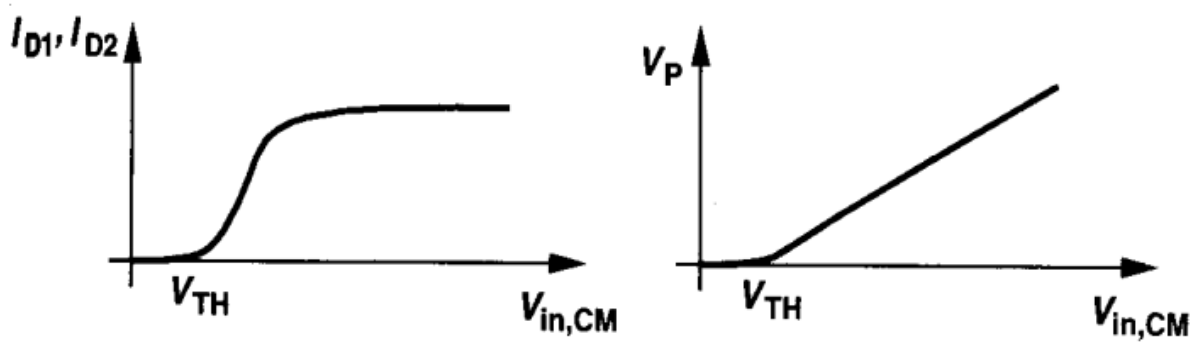


Figure 8.3: Large-signal Common mode behavior of Differential Amplifier

8.4 Lab conduction Procedure

This section helps you to build the following analog circuit blocks

1. Common Source (CS)

2. Differential Amplifier

8.4.1 CS Amplifier

Design Schematic

1. Built the design schematic based on Fig.8.4 and Table.8.1.

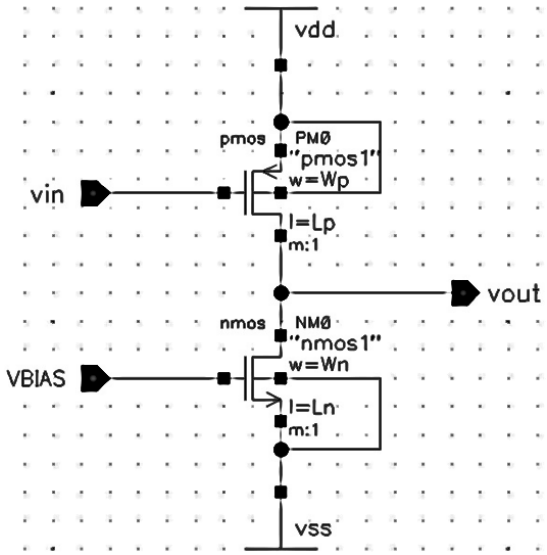


Figure 8.4: CS Amplifier

Table 8.1: Device dimensions for CS Amplifier

Device	Library	W	L
`pmos`	`gpdk180`	$50\mu m$	$1\mu m$
`nmos`	`gpdk180`	$10\mu m$	$1\mu m$

Test Schematic

1. Built the Test schematic based on Fig.8.5 and Table.8.2.

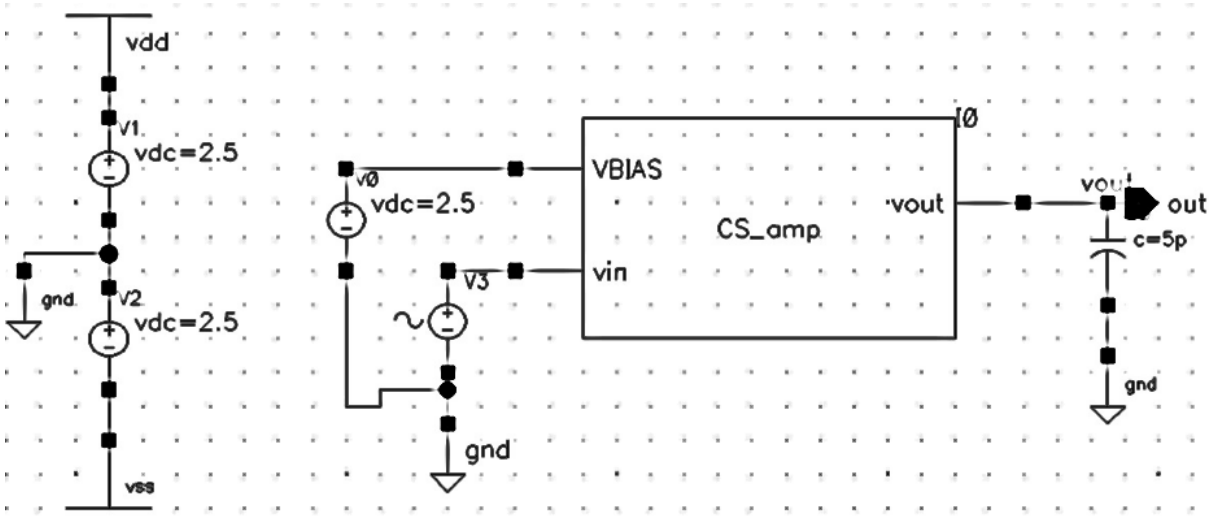


Figure 8.5: Test Schematic for CS Amplifier

Table 8.2: Library elements for CS amplifier

Library	Cell Name	Instance Name	Specification
AnalogLib	`vdc`	V0,V1,V2	DC voltage: 2.5 V
AnalogLib	`vsin`	V3	No need to set any parameters now
AnalogLib	`cap`	C0	5 pF
<Your Lib>	<Your Design>	I0	-

2. Note that the bias voltage is chosen to be at 2.5 V, so that the bias current set around 5mA through M_2.

Pre-Layout simulation

Under this section three analyses are performed.

1. DC Analysis

   (a) To plot the VTC of CS amplifier, choose the `vsin` cell's `dc` parameter and sweep between -2.5 to +2.5 voltage range.

   (b) The response of the DC analysis is shown in Fig.8.6.

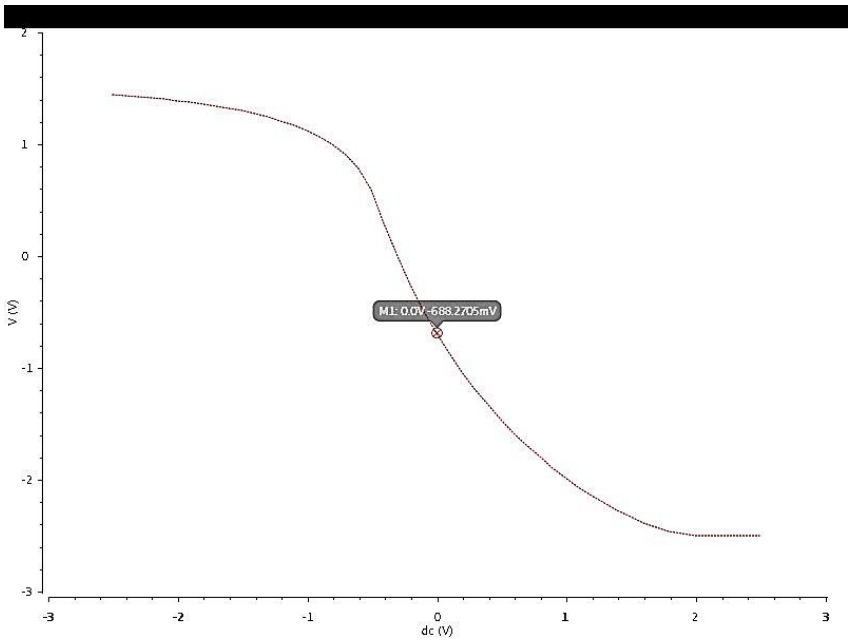


Figure 8.6: DC plot of CS Amplifier

This plot helps us to determine two important parameters: 1. DC operating point, and 2. Allowable input swing.
Here, in this design, the input is biased at, $V_{GS} = (V_{DD} + |V_{SS})/2 = 0$. Next the linear segment has to be identified, so that one can determine the input signal range that can be applied over the bias point to achieve linear amplification.

`vsin`	`DC voltage`	0

2. AC Analysis
   This analysis helps us to identify DC gain and the 3-dB frequency.

   (a) In the CS test schematic, set `ac magnitude` of `vsin` to **1**. Also, retain it's `dc voltage` to 0V.

   Although a 1V peak signal would saturate the real circuit, in the `AC` analysis mode SPICE works with the linearized circuit and is happy to report output voltage amplitudes of 100V or whatever is indicated by the small-signal gain. We use 1V peak to make calculating gain ratios easy: if $v_{in} = 1$, the v_{out}/v_{in} is just whatever voltage is seen at v_{out}.

   (b) In CIW, choose `ac` analysis, select sweep variable as `Frequency`, and mirror the settings shown in Fig.8.7 for specifying sweep parameters.

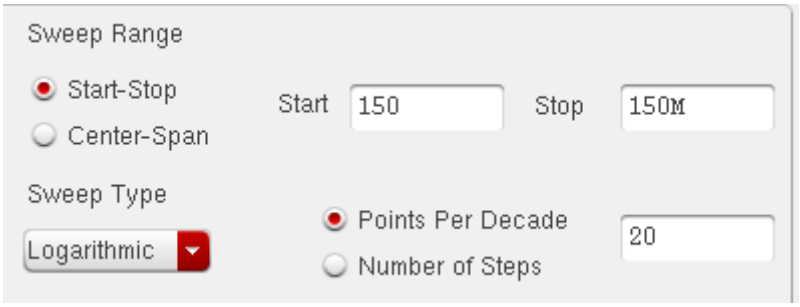


Figure 8.7: Sweep range for AC analysis

(c) Click `Run` and you would get a similar plot as shown in Fig.8.8

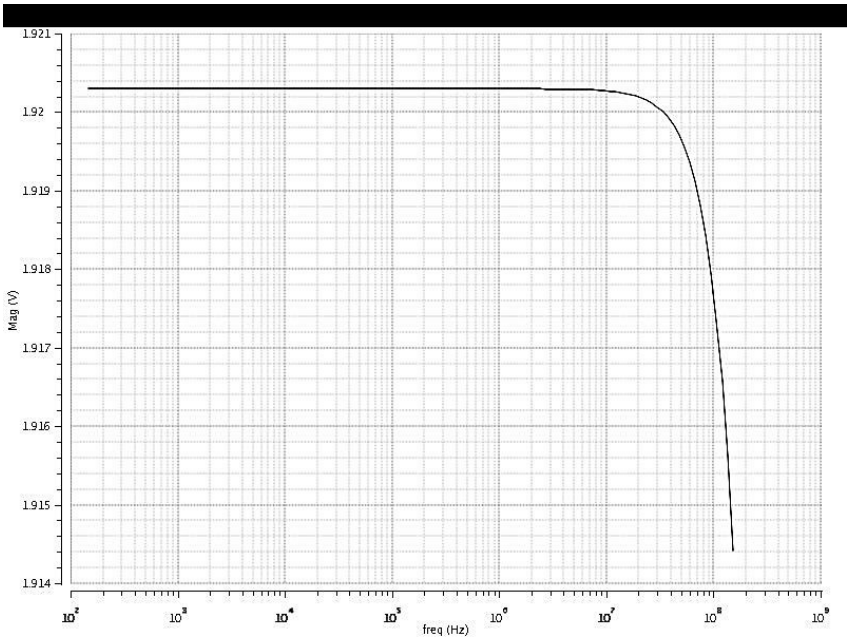


Figure 8.8: AC plot for CS amplifier

(d) To measure the 3-dB frequency, Go back to the ADE window, click on `Results` → `Direct Plot` → `AC Magnitude & Phase`.

(e) Go to the schematic window. Click on the `vout` net. Then hit the `Esc` key to finish the selection.

(f) A plot similar to the one shown in Fig.8.8 will be shown.

3. Transient Analysis

(a) Modify `vsin`: `Amplitude` to 5m V.

(b) Also from Fig.8.8, it is clear that the bandwidth extends beyond $10^6 Hz$, so it is safe to provide `frequency` to $1kHz$.

(c) Go back to the ADE, choose `tran` analysis and provide `stop time` as **5m**.

8.4.2 Differential Amplifier

Design Schematic

1. Built the design schematic based on Fig.8.9 and Table.8.3.

Table 8.3: Device dimensions for Differential Amplifier

Cell Name	Instance Name	W	L
`nmos`	M_1, M_2	$3\mu m$	$1\mu m$
`pmos`	M_3, M_4	$15\mu m$	$1\mu m$
`nmos`	M_5, M_6	$4.5\mu m$	$1\mu m$

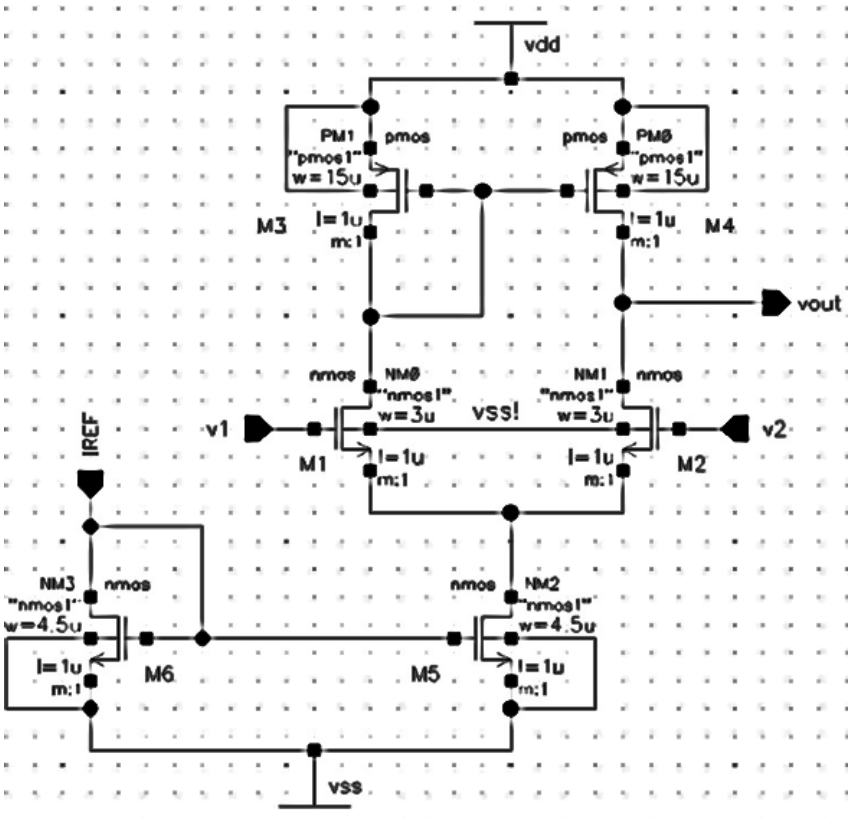


Figure 8.9: Differential Amplifier

Test Schematic

1. Built the Test schematic based on Fig.8.5 and Table.8.2.

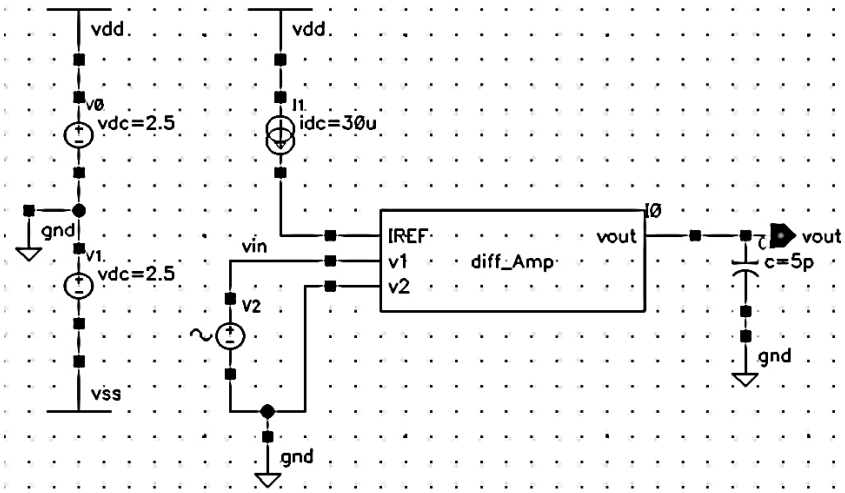


Figure 8.10: Test Schematic for Differential Amplifier

Table 8.4: Library elements for Differential amplifier

Library	Cell Name	Instance Name	Specification
AnalogLib	`vdc`	V0,V1	DC voltage: 2.5 V
AnalogLib	`vsin`	V2	No need to set any parameters now
AnalogLib	`cap`	C0	5 pF
AnalogLib	`idc`	I0	30u
<Your Lib>	<Your Design>	I0	-

The input is biased at a common mode voltage,$V_{ICM} = (V_{DD} + |V_{SS})/2 = 0$ V and the differential input is tagged in terms of `vsin` cell between the inverting and non-inverting inputs V1 and V2.

Pre-Layout simulation

Under this section three analyses are performed.

1. DC Analysis
   Since the input common mode level is fixed to 0 V, in this analysis we will focus only on the differential input range.

   (a) To achieve large-signal analysis for differential input, choose the `vsin` cell's `dc` parameter and sweep between -2.5 to +2.5 voltage range.

   (b) The response of the DC analysis is shown in Fig.8.11.

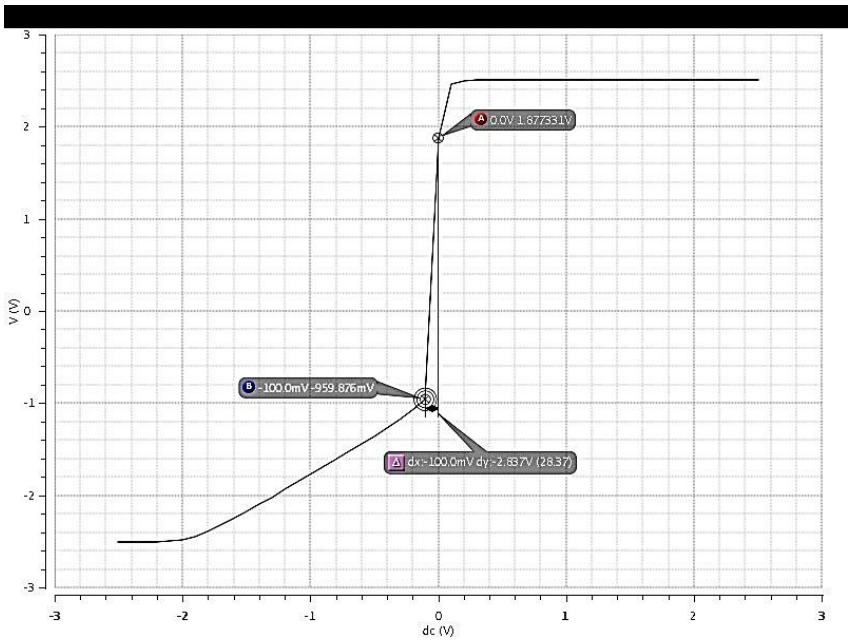


Figure 8.11: DC plot of Differential Amplifier

2. AC Analysis
   This analysis helps us to identify DC gain and the 3-dB frequency.

   (a) As stated earlier, set `ac magnitude` of `vsin` to **1**. Also, retain it's `dc voltage` to 0V.

   (b) In CIW, choose `ac` analysis, select sweep variable as `Frequency`, and mirror the settings shown in Fig.8.7 for specifying sweep parameters.

   (c) Click `Run` and you would get a similar plot as shown in Fig.8.12

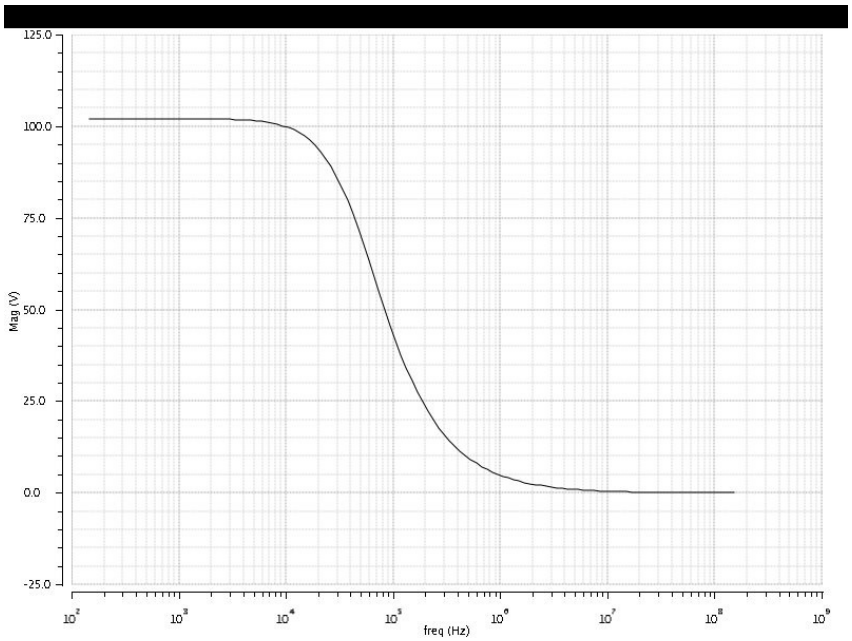


Figure 8.12: AC plot for Differential amplifier

(d) To measure the 3-dB frequency, Go back to the ADE window, click on `Results` → `Direct Plot` → `AC Magnitude & Phase`.

(e) Go to the schematic window. Click on the `vout` net. Then hit the `Esc` key to finish the selection.

(f) A plot similar to the one shown in Fig.8.12 will be shown.

3. Transient Analysis

(a) Modify `vsin`: `Amplitude` to 5m V.

(b) Also from Fig.8.12, it is clear that the bandwidth extends beyond $10^6 Hz$, so it is safe to provide `frequency` to $1kHz$.

(c) Go back to the ADE, choose `tran` analysis and provide `stop time` as **5m**.

8.5 Observation

- DC Analysis

1. Measure the output Q-point

- AC Analysis

1. From the plots shown in Fig.8.8 and Fig.8.12, Use `cross` function to read the *3dB frequency* and *unity-gain frequency.*

- Transient Analysis

1. Using `calculator`'s `peak-to-peak` function, compute the swing of the output graph
2. Calculate the gain using the expression, $A_v = \frac{V_{out,PP}}{V_{in,PP}}$, where $V_{in,PP} \approx 10mV$
   Verify the result with the AC analysis
3. Verify that the output swings across output Q-point measured from DC Analysis

Part B

Practice question: Realize Op-amp circuit

Cascade Differential amplifier with CS amplifier as the output stage, as shown in Fig.8.13 and perform all the analysis.

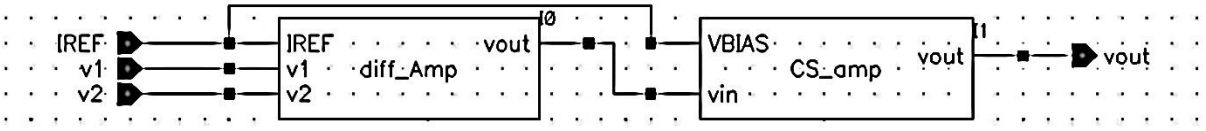


Figure 8.13: Two-stage Op-amp

.

SI	Criteria	Max Marks	Marks Obtained
	Data Sheet		
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
	Record		
D	Simulation / Conduction	15	
E	Analysis of the Result	15	
	Viva	40	
	Total	100	
	Scale to 10 Marks		

Staff Signature

Experiment 9

Synthesis of Serial Adder

9.1 Objective

- To write an RTL code for a Serial Adder in Verilog and verify it's function behavior using **nclaunch** tool.
- To synthesize and generate netlist file for the verified verilog code using **Genus** tool with
  1. `.lib` standard cell library, and
  2. `.sdc` constraint file
- To perform functional verification again on the generated netlist file.
- To realize physical design (Place and Route (PnR)) using Encounter tool and perform physical verification.
- To perform power analysis.

9.2 Introduction

An example of synchronous sequential design is a Finite State Machine (FSM) system that preforms computation based on the current state and current input.

In this lab, you will realize a serial adder, which then synthesized and verified by post-synthesis functional verification. These verified synthesized netlist is taken further down the Application Specific Integrated Circuit (ASIC) design flow to PnR process. The PnR process done with the `.lef` as a part of self-study component in Part B. Since this is case study, the students need to perform this with little help from this manual.

9.3 Theory

In this section, we will try to understand the following concepts - which are essential for this lab.

1. Basics of Register Transfer Level (RTL) coding
2. Overview of Synthesis process
3. Overview of the contents of `.lib` and `.sdc` files
4. About FSMs and it's implementation in Verilog.
5. About PnR tool flow.

9.3.1 Basics of RTL coding

In digital circuit design, RTL is a design abstraction which models a synchronous digital circuit in terms of the flow of digital signals (data) between hardware registers, and the logical operations performed on those signals.

Register-transfer-level abstraction is used in Hardware Description Languages (HDLs) like Verilog and VHDL to create high-level representations of a circuit, from which lower-level representations and ultimately actual wiring can be derived. Design at the RTL level is typical practice in modern digital design.

A synchronous circuit consists of two kinds of elements: registers (Sequential logic) and combinational logic. Registers (usually implemented as D flip-flops) synchronize the circuit's operation to the edges of the clock signal, and are the only elements in the circuit that have memory properties. Combinational logic performs all the logical functions in the circuit and it typically consists of logic gates. This pictorially represented in the Fig.9.1.

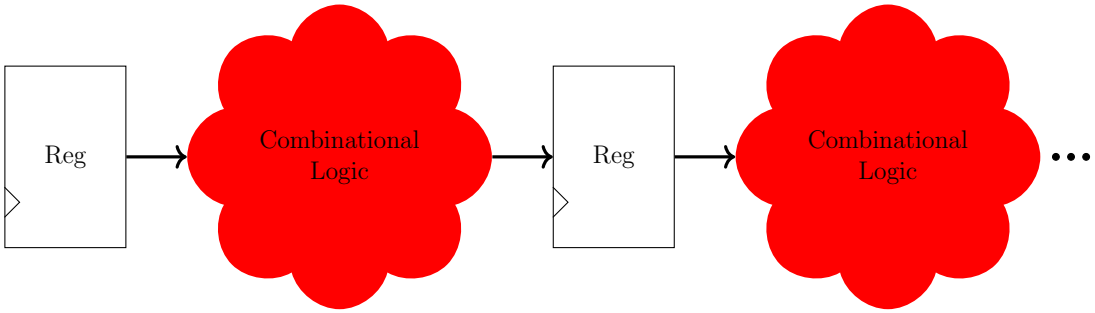
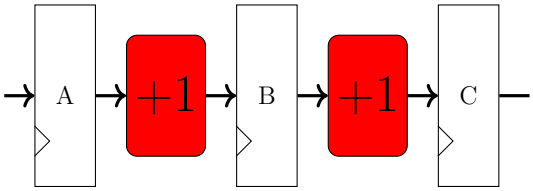


Figure 9.1: A typical RTL Design representation

As an example, consider the following *code snippet* and it's representation below

```
wire A_in, B_in, C_in;
reg A_out, B_out, C_out;
always @( posedge clk )
begin
 A_out <= A_in;
 B_out <= B_in;
 C_out <= C_in;
end
assign B_in = A_out + 1;
assign C_in = B_out + 1;
```

(a) RTL code



(b) Implementation

Figure 9.2: RTL example

9.3.2 Overview of Synthesis process

Logic synthesis is a process by which an abstract form of desired circuit behaviour, typically at RTL is termed into a design implementation in terms of logic gates typically by a program called a **synthesis** tool.

The following Fig.9.3 shows the Input-Output of Synthesis process - guiding us to understand the usage of `.lib` file that we generated in Exp.7.

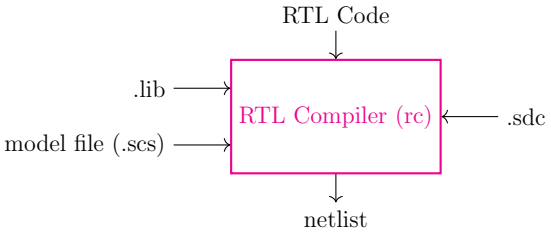


Figure 9.3: Input-Output of Synthesis flow

Here the model file contains information about the MOS transistors and it's associated parameters.

9.3.3 Overview of the contents of `.lib` and `.sdc` files

Apart from the model file (`.scs`) and the glsrtl file, one has to provide the timing information of the standard cell through `.lib` file and the constraint file in `.sdc` format. Here only a brief information about these files are provided. For more information, use the YouTube link † given below.

General `.lib` content

We are already familiar with the contents of `.lib` file, from our previous experiment. So here our focus will be on the general content that will be available in the commercial standard cell library. In general, it contains bank of cells which perform different functions and each bank contains cells that performs the same function. As shown in Fig.9.4, the library not only contains different bank of cells performing functions like NAND, NOR, DFF, ADDER, etc, but each bank has a set of cells that performs the same function like NOT function which are built using different circuit families. In short, it contains information about the delay, leakage currents, functional table and power dissipation of each cells.

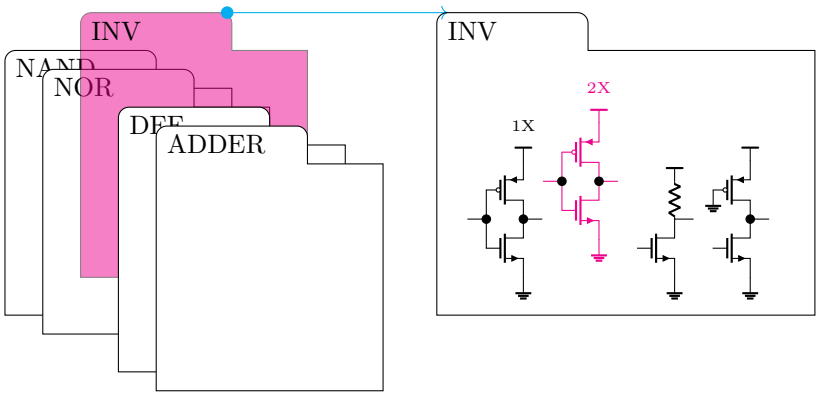


Figure 9.4: Cells inside LIB file

†https://www.youtube.com/playlist?list=PLXnaDu1KFWvY1BljNS-mP70i4cf6Tl7cb

Note: In our `.lib` file, we have only 4 cells performing 4 different functions and all the cells belong to *cmos logic family*.

General `.sdc` content

An Synopsys Design Constraints (SDC) file contains constrains written in TCL based format. These constrains help the synthesizer to narrow down it's search for proper cell from the bank of cells having same functionality (as shown in the Fig.9.4 `INV` cells). One can group the commands into following categories

1. Basic Constraints commands
   Specifies the units for capacitance, resistance, time, voltage, current and power.

2. Object Access Constraints commands
   Specify how to access objects in a design instance.
   Object can be a cell, a block, a port, a pin, or anything else in the design.

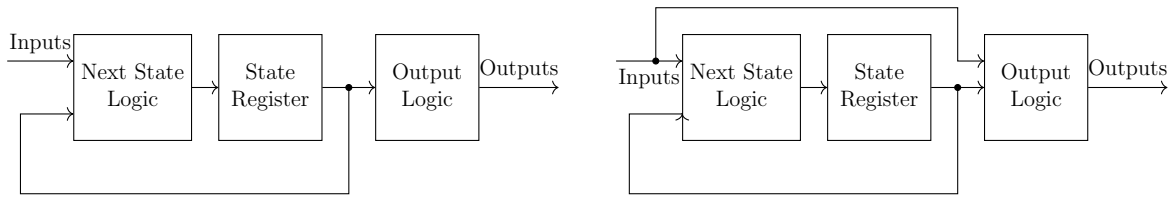
3. Timing Constraints commands
   Related to timing specifications of the design

4. Environmental constraints
   Used to setup the environment of the design under analysis

5. Multi-Voltage Commands
   Used when multi-voltage islands are present in the design

9.3.4 Finite State Machine

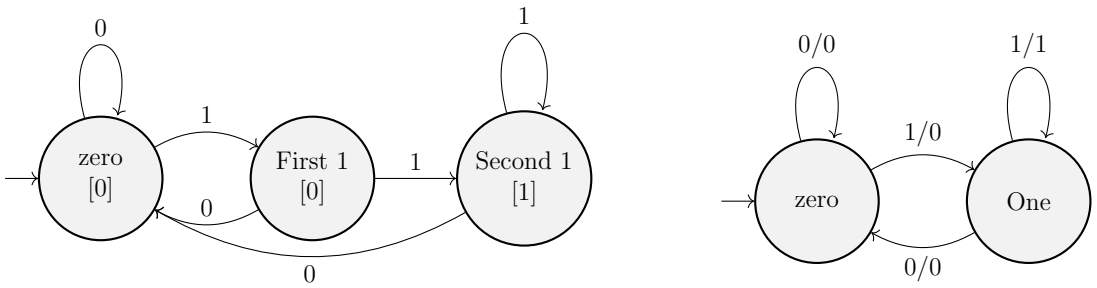
Designing a synchronous Finite State Machine (FSM) is a common task for a digital logic engineer. A finite state machine can be divided in to two types: **Moore** and **Mealy** state machines. Fig.9.5a has the general structure for Moore and Fig.9.5b has general structure for Mealy. The current state of the machine is stored in the state memory, a set of n flip-flops clocked by a single clock signal (hence "synchronous" state machine). The state vector (also current state, or just state) is the value currently stored by the state memory. The next state of the machine is a function of the state vector in Moore; function of state vector and the inputs in Mealy.



(a) Moore State Machine (b) Mealy State Machine

Figure 9.5: Finite State Machines

To appreciate the differences that exist between both the machines, let's consider an FSM that detect two or more 1s in the given sequence. The corresponding state diagram for the machines are shown in Fig.9.6.



(a) Moore State Diagram (b) Mealy State Diagram

Figure 9.6: State diagrams that detect 2 or more one's in sequence

Now let's focus on the verilog coding style for both the type of FSM.

Moore Verilog FSM

The FSMs are in general follow behavioral modeling approach using procedural statements.

As shown in Fig.9.5a, the output is a function of present state alone. This results in higher number of states than the Mealy machine, as shown in Fig.9.6a.

Now let's see how to code the Moore machine, shown in Fig.9.6a, in verilog. The following code snippet shows the realization of states and it's transition to other states.

```
always @(in or state)
  case (state)

        zero: begin // last input was a zero
          out = 0;
          if (in) next_state = First1;
          else next_state = zero;
          end

        First1: begin // we've seen one 1
          out = 0;
          if (in) next_state = Second1s;
          else next_state = zero;
          end

        Second1s: begin // we've seen at least 2 ones
          out = 1;
          if (in) next_state = Second1s;
          else next_state = zero;
          end

        default: begin // in case we reach a bad state
          out = 0;
          next_state = zero;
  endcase

// Implement the state register
```

```
always @(posedge clk)
  if (reset) state <= zero;
  else state <= next_state;
```

Mealy Verilog FSM

As shown in Fig.9.5b, the output is a function of both present input and present state. This results in less number of states than the Moore machine counter part, as shown in Fig.9.6b.

The following code snippet shows the realization of states and it's transition to other states for Mealy machine.

```
always @(in or state)
  case (state)
        zero: begin // last input was a zero
          if (in) next_state = one;
          else next_state = zero;
          out = 0;
        end

        one: // we've seen one 1
          if (in) begin
            next_state = one;
            out = 1;
          end
          else begin
            next_state = zero;
            out = 0;
          end
        endcase

always @(posedge clk)
        if (reset) state <= zero;
        else state <= next_state;
```

9.3.5 Physical Design

Physical design is process of transforming netlist into layout which is manufactureable [GDS]. Physical design process is often referred as Place and Route (PnR) / Automatic Place & Route (APR). Main steps in physical design are placement of all logical cells, Clock Tree Synthesis (CTS) and routing. During this process of physical design, the timing, power, design and technology constraints have to be met. Further the design might require optimization w.r.t area, power and performance.

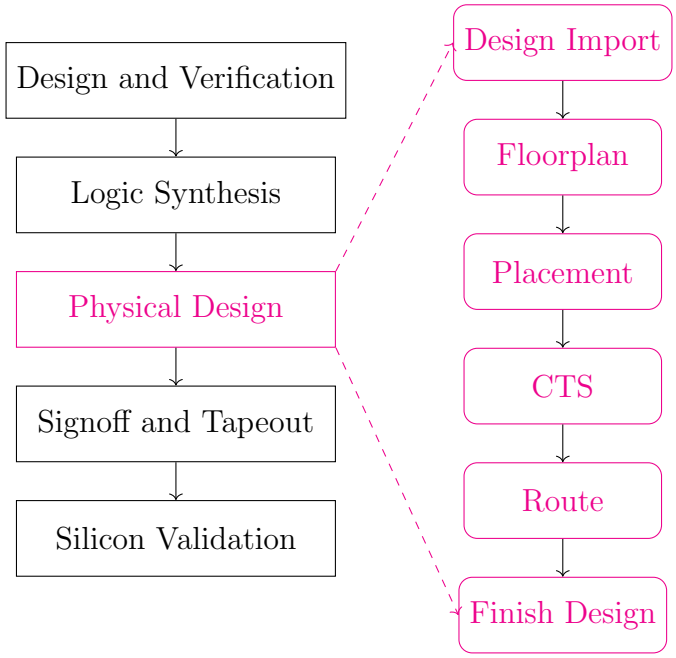


Figure 9.7: Physical Design

Floorplan

Floorplan is one of the critical & important step in Physical design. Quality of your Chip / Design implementation depends on how good is the Floorplan. A good floorplan can be make implementation process (place, CTS, route & timing closure) cake walk. On similar lines, a bad floorplan can create all kind issues in the design (congestion, timing, noise, ir, routing issues). A bad floorplan will blow up the area, power & affects reliability, life of the IC and also it can increase overall IC cost (more effort to closure, more LVTs/ULVTs)

Before staring of Floorplan, it is better to have basic design understanding, data flow of the design, integration guidelines of any special analog hard IPs in the design. And for block/partition level designs, understanding the placement & IO interactions of the block in Full chip will help in coming up with good floorplan.

Routing

In general, the power strips are layered out and then the signals are routed with appropriate pitch sizes that matches the pin location of our standard cells.

9.4 Lab conduction Procedure

One can summarize the Synthesis flow as

1. Compile - Elaborate - Simulate [RTL code]
   - `nclaunch` (Native Compiler) from Cadence
2. Synthesize RTL code with Constraint - Generate [Synthesized code]
   - `genus` from Cadence

3. Compile - Elaborate - Simulate [Synthesized code]
    - `nclaunch` from Cadence

and the same is reflected in Fig.9.8 as Part A.

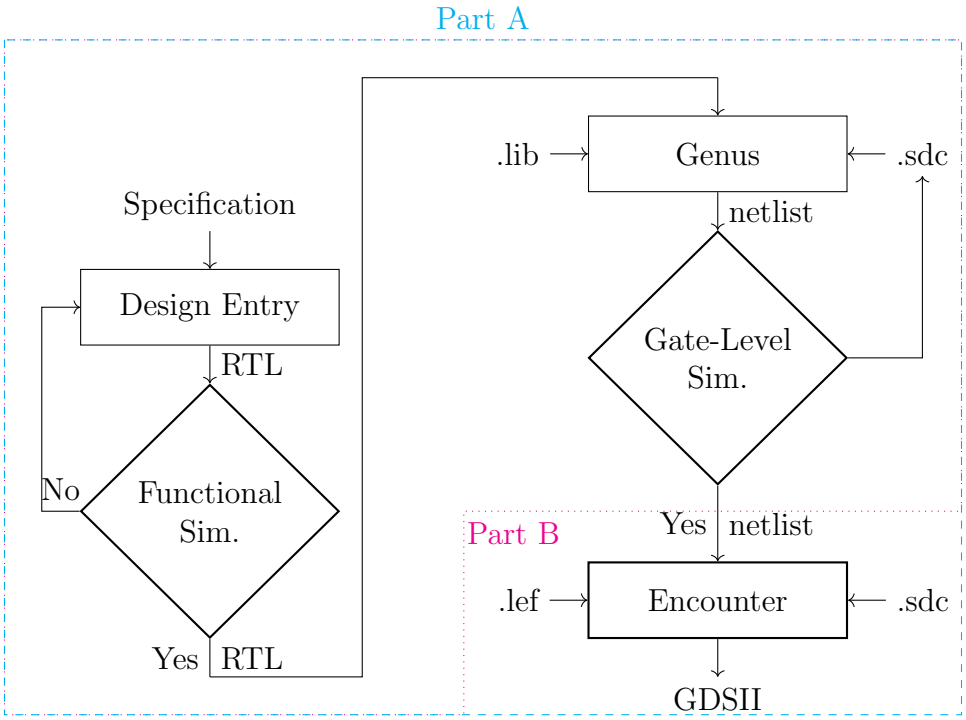


Figure 9.8: Methodology

9.4.1 Directory structure

You will be using the directory structure shown in Fig.9.9 to perform this experiment.

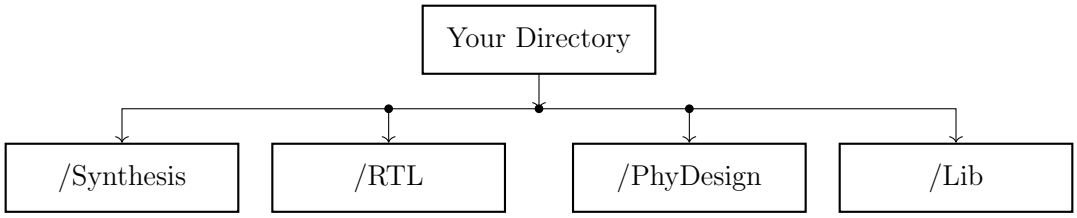


Figure 9.9: Directory Structure followed during this Experiment

9.4.2 Codes

Now create the Design file and the corresponding Testbench file written in Verilog under `RTL/` folder.

Note

The code snippet of Half-Adder, Full-Adder and Test bench is shown here just to *exercise the synthesis flow*. But you should rather be in a position to do any design

using this flow.

Code for Serial Adder

```
module serial_adder (A, B, reset, clock, sum);
input [7:0] A, B;
input reset, clock;
output [7:0] sum;
reg [3:0] count;
reg s, y, Y;

wire [7:0] qa, qb, sum;
wire run;
parameter G=0, H=1;

shiftreg A1(A, reset, 1'b1, 1'b0, clock, qa);
shiftreg B1(B, reset, 1'b1, 1'b0, clock, qb);
shiftreg C1(8'b0, reset, run, s, clock, sum);

always@(qa, qb, y)  // full adder fsm
begin
case(y)
  G: begin
        s = qa[0] ^ qb[0];
        if (qa[0] & qb[0])
          Y = H;
        else
          Y = G;
        end
  H: begin
        s = qa[0] ~^ qb[0];
        if(~ qa[0] & ~qb[0])
          Y = H;
        else
          Y= G;
        end
default: Y= G;
endcase
end

always@(posedge clock)
if (reset)
        y= G;
else
        y= Y;

always@(posedge  clock)
if(reset)
        count = 8;
else if(run) count = count-1;
assign run = | count;
```

```
endmodule

module shiftreg(R, L, E, W, clock, q);
parameter n = 8;
input[n-1:0] R;
input L, E, W, clock;
output [n-1:0] q;
reg [n-1:0] q;
integer k;

always@(posedge clock)
if (L)
        q <= R;
else if (E)
  begin
        for(k = n-1; k>0; k = k-1)
          q[k-1] <= q[k];
          q[n-1] <= W;
  end
endmodule
```

Test Bench for Serial Adder

```
module serial_adder_tb;
reg [7:0] A, B;
reg reset, clock;
wire [7:0] sum;
serial_adder s1(A,B,reset, clock, sum);

initial
clock = 0;

always
#5 clock = ! clock;

initial
  begin
        A = 8'hA0;
        B = 8'h1F;
        reset = 1;
        #10 reset = 0;
        #100 $finish;
  end

endmodule
```

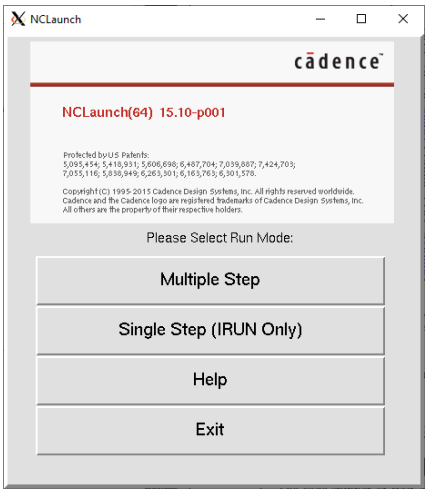
9.4.3 Compilation, Elaboration and Simulation of RTL code

With all files ready, we can start the 1^{st} flow as summarized at the beginning of this section.

1. Go to `RTL/` folder and invoke the **nclaunch** tool

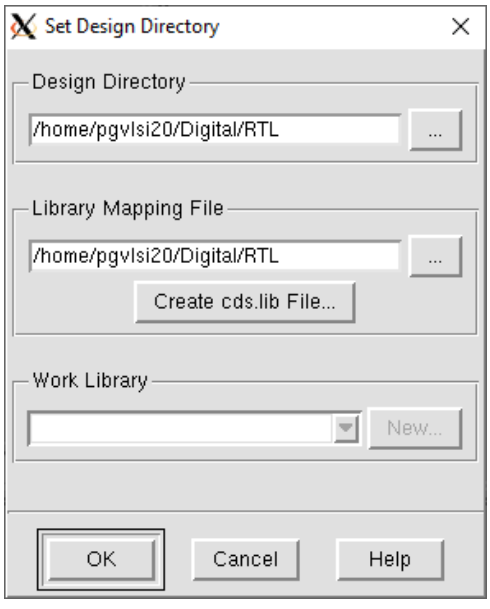
   ```
   nclaunch &
   ```

2. **If appears**: Select `Multiple Step`

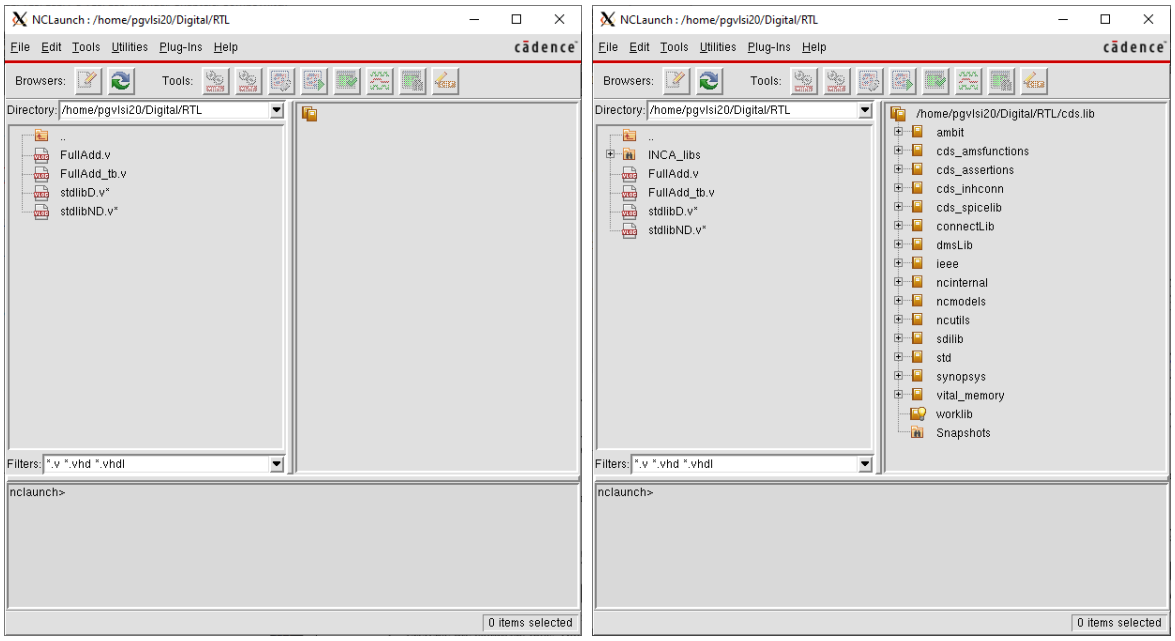


3. **Conditional Steps**: If the right side of the `NCLaunch` window is empty, (or) if the directory is **not** pointing to your current directory, then perform the step given below.

   (a) Go to `File` → `Set Design Directory`. In the sub-window, click on `Create cds.lib File`. Click on `Save` and `Ok` in the next consecutive sub-windows that pops-up.



(b) The above step should result as shown in Fig on the right

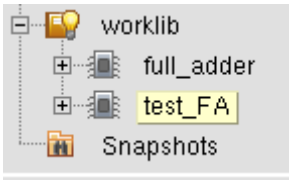


(a) Empty Right side sub-window (b) After creating *cds.lib* file

Figure 9.10: NCLaunch Interface

4. *Compile* both the Design and Test verilog files using 

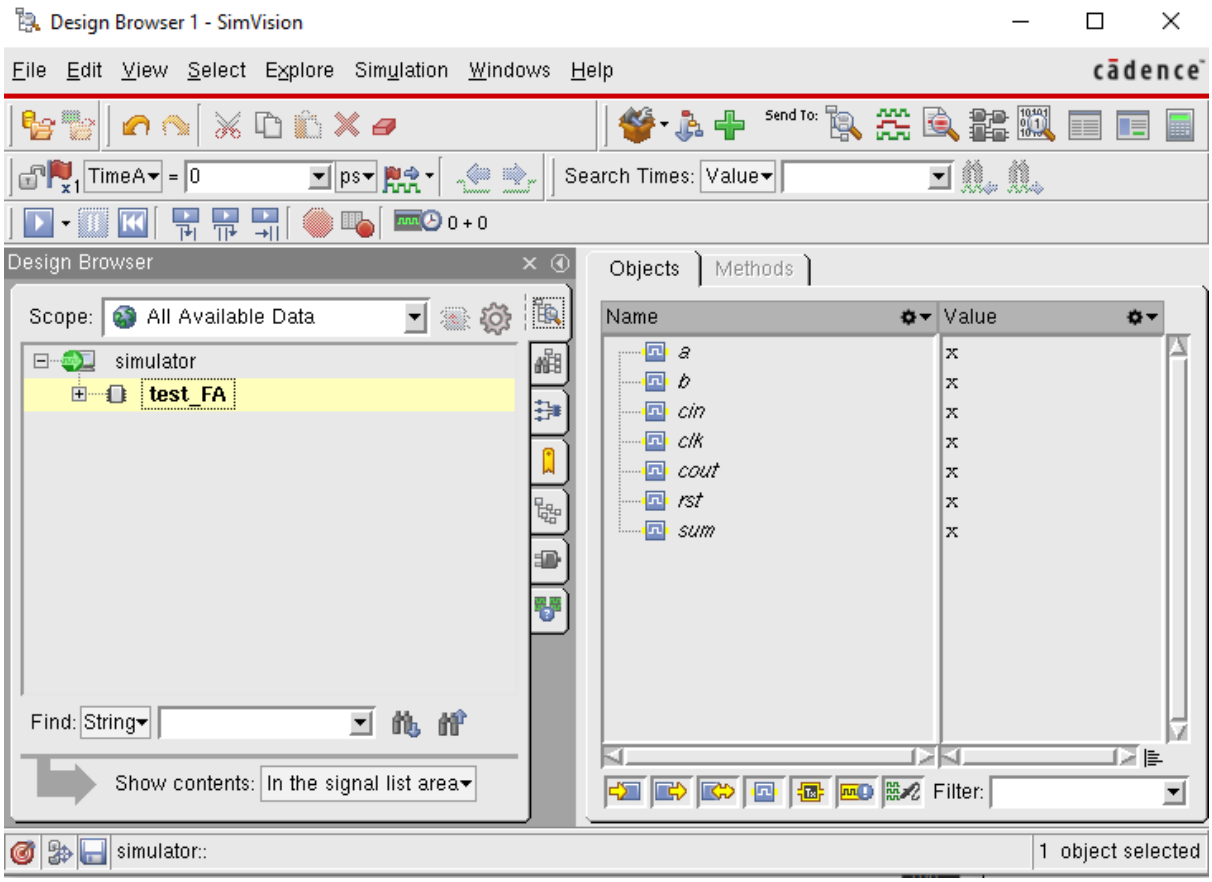
5. Next *Elaborate* the top module (which in our case, `serialAdder_tb.v`) by first expanding the `worklib` and selecting it, which then elaborated using  .



6. Next to *Simulate*, expand the `Snapshots` and select the top module (`Worklib.serialAdder_tb:mo` and then use the  to *invoke* the simulator.

7. This should bring the following 2 windows (Design Browser and Console).

8. Select `Design Browser` window and click on the top module name (`serialAdder_tb`), this click should list all the signals on the right side window as shown



9. Select all the signals on the right side using shift key and then launch graph window using  icon. This should bring the graph window.

10. Click on play icon, which results in *transient waveform* for a period of `65n` secs (as specified in Testbench).

11. After Verification, close all the windows.

9.4.4 Synthesize RTL code

After verifying the RTL code in the previous step, we use *only our Design verilog* file for the synthesis process.

1. Copy the Design verilog file from `RTL/` to `Synthesis` folder using `cp` command.

   ```
   cp SerialAdder.v ../Synthesis/
   ```

2. Invoke the Synthesis tool using the following command

   ```
   genus -legacy_ui -gui
   ```

3. Next, we have to Load Libraries and Designs and finally Synthesize the design inside the `genus_legacy` environment

   (a) Setting the library and HDL paths:

```
set_attr init_lib_search_path {../Lib/}
set_attr init_hdl_search_path {../RTL/}
```

(b) Loading the library

```
set_attr library {slow.lib}
```

(c) Reading the design:

```
read_hdl SerialAdder.v
```

(d) Elaborating the design:

```
elaborate
```

(e) Reading constraints:

```
read_sdc constraints.sdc
```

(f) Synthesize to generic gates and then map to technology library

```
syn_generic
syn_map
```

Note: After execution of each above commands, you can visually see the changes that happens in the `gui` window.

4. Generating Reports inside the `genus_legacy` environment
Use the `report_*` command to write out the results.

(a) To generate timing report use:

```
report_timing
```

(b) To dump out the power report use:

```
report_power
```

(c) To report Quality of Report (QOR) use:

```
report_qor
```

5. Writing Output Files again inside the `genus_legacy` environment

(a) To write out Synthesized netlist:

```
write_hdl > SerialAdder_netlist.v
```

(b) To generate final SDC file run:

```
write_sdc > SerialAdder_sdc.sdc
```

(c) Write out SDF file:

```
write_sdf -timescale ns -nonegchecks -recrem split
-edges check_edge > delays.sdf
```

- `timescale`: Used to mention the time unit
- `nonegchecks`: Used to ignore the negative timing checks
- `recrem`: Used to split out the recrem (recovery-removal) timing check to separate checks for recovery and removal

- `edges`: Specifies the edges values
- `check_edge`: Keeps edge specifiers on timing check arcs but does not add edge specifiers on combinational arcs.

> **Note**
>
> The commands listed above will generate netlist, SDF and SDC in the synthesis directory. If you want, you can specify required directory to save the output files.

6. Exiting the Software inside the `genus_legacy` environment

   (a) To close Genus use the following command:

   ```
   exit
   ```

9.4.5 Compilation, Elaboration and Simulation of Synthesized Netlist

Now it's time to perform post-simulation on Synthesized netlist, to verify whether the netlist that we generated using the 4 standard cells is capable of reproducing the same Full Adder function or not.

Most of the steps that we performed in the Sec.9.4.3 is same here, except few changes in them. So the manual will present only those changes to be done before redoing the steps that where presented in Sec.9.4.3.

1. Copy the generated netlist file (`SerialAdder_netlist.v`) back to `RTL` folder from the `Synthesis` folder

2. Make the following changes in the copied netlist file's **Header** section.

   ```
   'timescale 1ns/10ps
   ```

9.5 Observation

- One has to fine tune `create_clock` parameter such that there exist a positive time slag of 100 ps and identify the speed of the circuit.

Part B

Practice question: Perform Back end flow

Place and Route

We will be using **Innovus** tool[†] to perform PnR. One can summarize the steps that are needed for proper execution of PnR, based on Fig.9.7 , as follows

1. Import the Design
2. Floorplan the Design
3. Power Plan
4. Create Power Rails with Special Route
5. Running Clock Tree Synthesis
6. Routing the Nets
7. Extraction and Timing Analysis
8. Running Physical Verification
9. Running and Viewing Power Analysis

Import the Design

In this section, you import a gate-level netlist and libraries into the Innovus Implementation System.

1. Using the directory structure shown in Fig.9.9, navigate into the `PhyDesign/` folder
2. Start the Innovus Implementation System by entering:

   ```
   innovus &
   ```

3. To import a gate-level netlist, timing constraints, and libraries, choose `File` → `Import Design`.
   - Under `Files:` label, select the *netlist* file generated from the 9.4.4
   - Under `Technology/Physical Libraries:` section, choose `LEF:` option and navigate to the `gpdk.lef`[‡].
   - Under `Power:` section, specify `Power Nets:`, as **VDD** and `Ground Nets:` as **GND**
   - Under `Analysis Configuration` section, load `serialAdd.view` from the present directory.

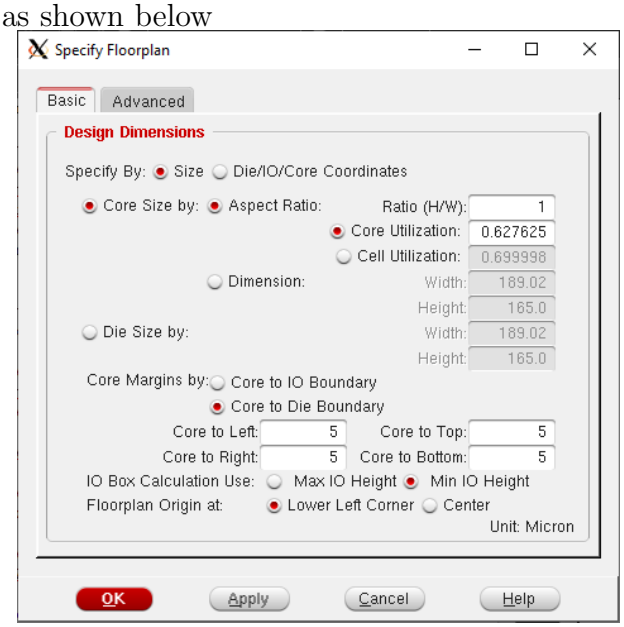
[†]Previously the tool is called as *Encounter*

[‡]This is the post-processed file from the Exp.6

4. The result above step can be visualized by using *zoom-fit* option in the Innovus main window.

Floorplan the Design

1. From the main window, select `Floorplan` → `Specify Floorplan` and reflect the following settings as shown below

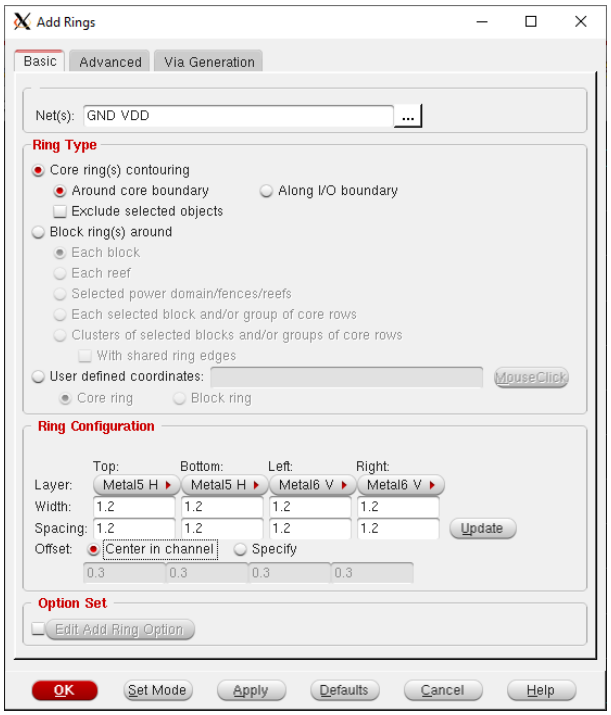


Power Plan

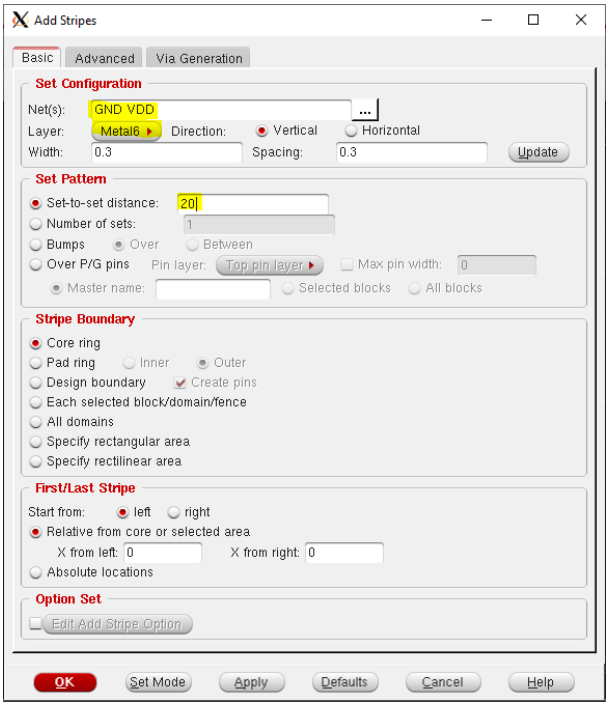
1. Again from the main window, select `Power` → `Power Planning` → `Add Ring...` and reflect the following settings as shown in Fig.(a)

Note: Under `Nets` section, click on browse botton and select the `VDD` and `GND` nets and `add`.

2. Again from the main window, select `Power` → `Power Planning` → `Add Strip...` and reflect the following settings as shown in Fig.(b)



Add Ring



Add Strip

Create Power Rails with Special Route

1. From the main window, select `Route` → `Special Route`

2. Under `Nets` section, click on browse botton and select the `VDD` and `GND` nets and `add`.

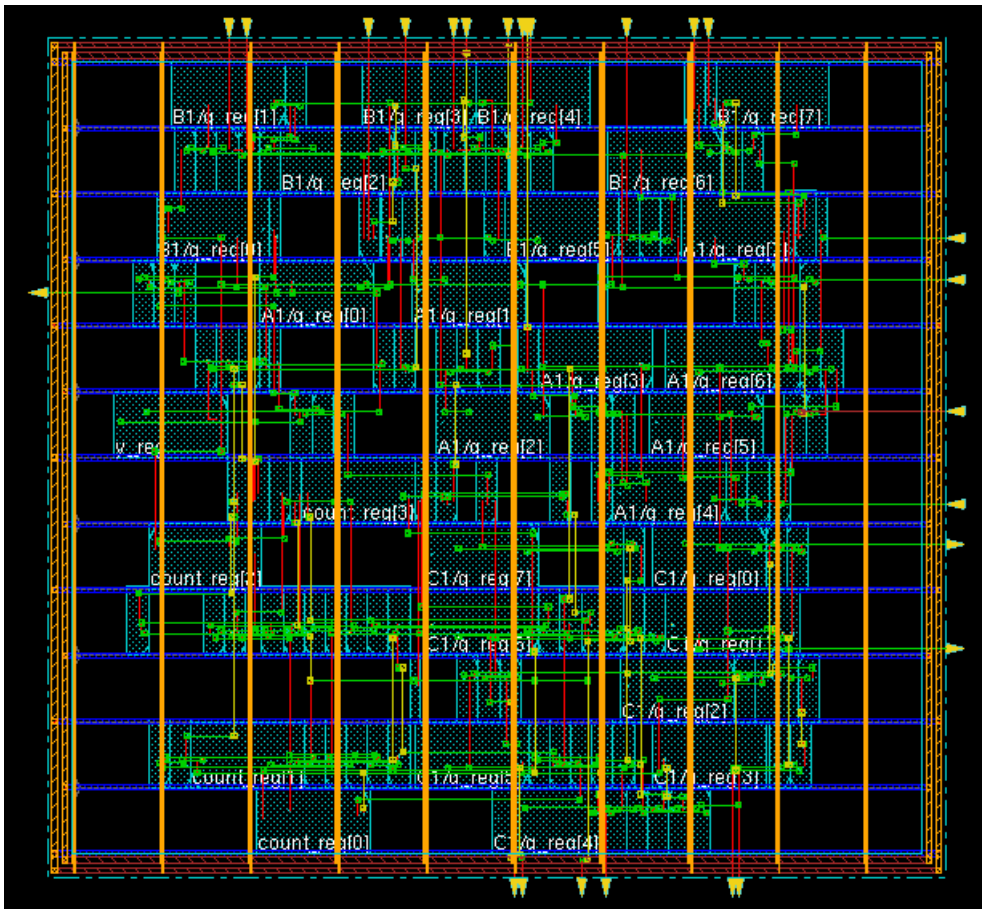
3. Under `SRoute` section, deselect all the options, **except** `Follow Pins`

4. Click OK.

Placement of Standard cells

1. Form the Main window, select `Place` → `Place Standard Cell...`.

2. Click `ok` with the default option.



Running Clock Tree Synthesis

Until now, we have assumed an ideal clock. Now we have all sequential elements placed, so we have to provide them with a real clock signal. Now you might have a question in your mind: "*Why not just route the clock net to all sequential elements, just like any other net?*". If it's not taken care, the clock signals arriving to different registers/flip-flops will have skew, jitter, slew, etc. This also causes implications to signal integrity, area and power.

1. Generate the clock tree by running the command:

```
create_ccopt_clock_tree -name clk -source [get_ports clock]\
                        -no_skew_group
```

2. Create a clock tree by running the command:

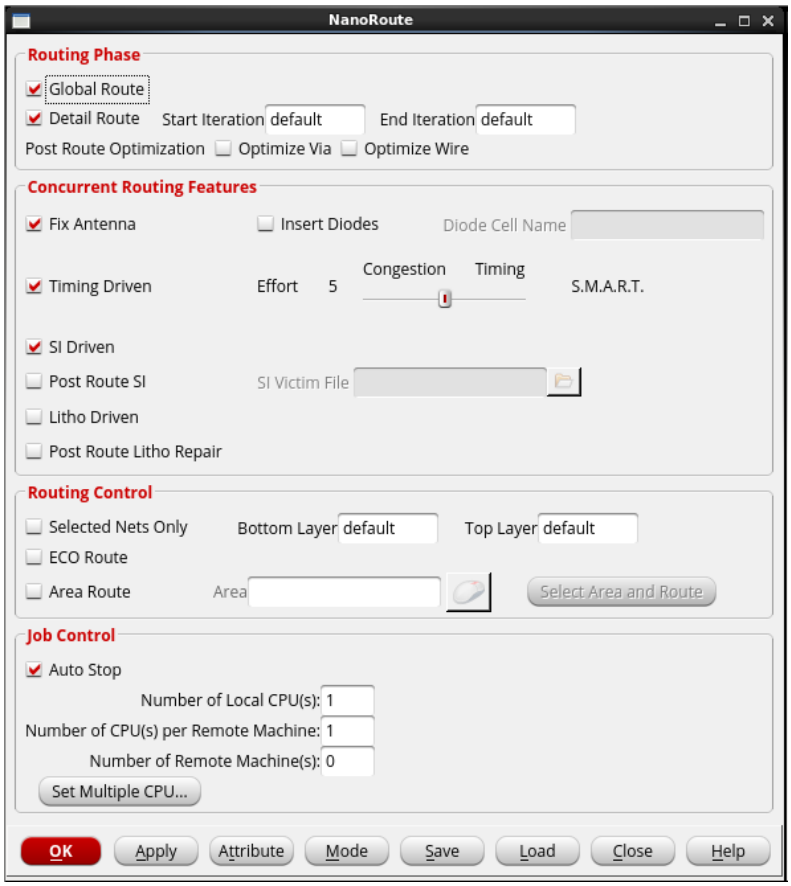
```
ccopt_design
```

   If you see error message about the clock net being not completely routed. Ignore this error as later on, when you run the NanoRoute tool for the remaining nets, this error will be fixed.

Routing the Nets

1. To route the nets, choose `Route` → `NanoRoute` → `Route`

2. Reflect the setting shown below:



3. Click OK.

Extraction and Timing Analysis

1. Run RC extraction on the routed design by selecting `Timing` → `Extract RC`.

2. Unselect all options except `Save SPEF to`.

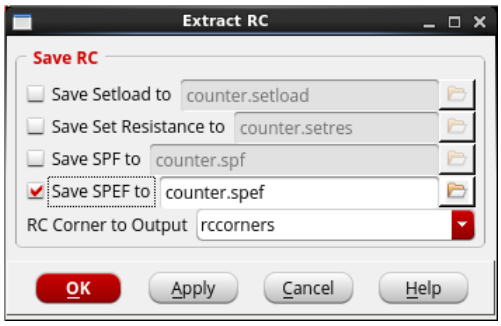
3. Click OK.

4. Set the timing analysis mode by running the following commands:

```
setAnalysisMode -analysisType onChipVariation
```

5. Run setup and hold timing analysis by running the following commands:

```
timeDesign -postRoute
```



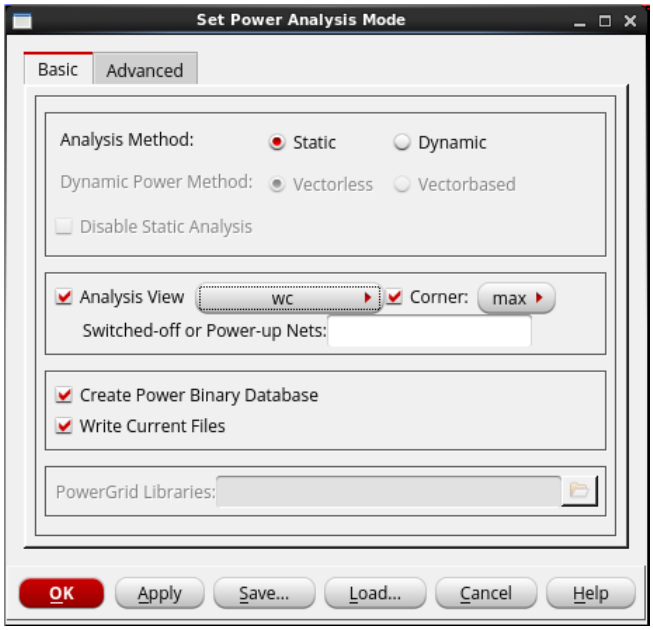
Running Physical Verification

In this section, you run physical verification commands in the Innovus system.

1. Verify Geometry
   (a) Choose `Verify` → `Verify Geometry`
   (b) The Verify Geometry form is displayed.
   (c) Are there any violations?
   Answer:
2. Verify Connectivity
   (a) Choose `Verify` → `Verify Connectivity`.
   (b) The Verify Connectivity form is displayed.
   (c) Are there any violations?
   Answer:

Running and Viewing Power Analysis

1. To display the power analysis setup form, choose `Power` *to* `Power Analysis` *to* `Setup`.



.

SI	Criteria	Max Marks	Marks Obtained
Data Sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation / Conduction	15	
E	Analysis of the Result	15	
	Viva	40	
	Total	100	
	Scale to 10 Marks		

Staff Signature

Experiment 10

Case Study: ASIC Design flow

10.1 Objective

1. To write a verilog code for the given combinational / sequential system
2. To simulate the design using Cadence NC Launch
3. To synthesize the design using Cadence RTL compiler and report area & power
4. To complete physical design using Cadence Encounter Digital Implementation

10.2 Introduction

This lab deals with the ASIC(Application Specific Integrated Circuits) design of a digital system, starting from writing a verilog code till physical design / backend.

10.3 Theory: Typical Design Flow

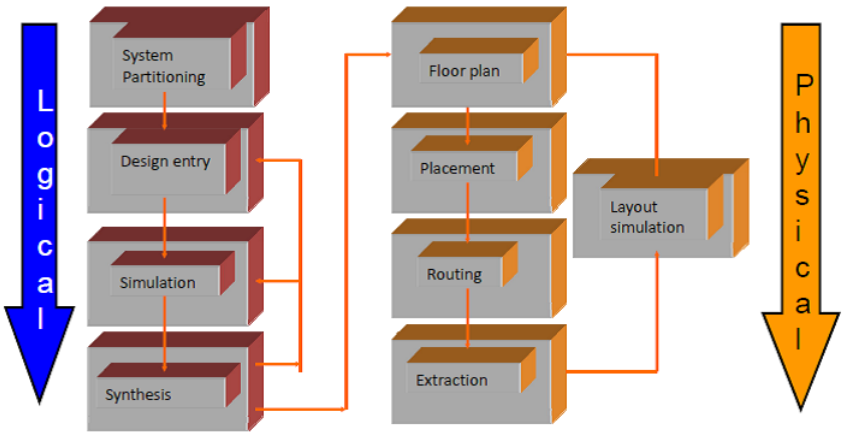


Figure 10.1: ASIC Flow

10.4 Observation

1. Verify the functionality of the given system using Cadence NCLaunch
2. Synthesize the verilog code using RTL compiler and report area & power

3. Complete the physical design using Cadence Encounter Digital Implementation System

Glossary

ADE Analog Design Environment. 11, 14, 15, 23, 24, 31, 34, 36, 46–48, 55, 62, 63, 82, 86

APR Automatic Place & Route. 94

ASIC Application Specific Integrated Circuit. 89

CAD Computer Aided Design. vii

CIW Command Interpreter Window. 6, 61, 67, 81, 85

CPA Carry-Propagate Adder. 19, 20

CS Common Source. 77, 79–81, 87

CTS Clock Tree Synthesis. 94, 95

DRC Design Rule Check. 59, 60, 66

ERC Electrical Rule Check. 59

FSM Finite State Machine. 89, 92, 93

HDL Hardware Description Language. 90

LSW Layer Selection Window. 64

LVS Layout Versus Schematic. 59, 60, 66

PnR Place and Route. vii, 89, 94, 105

QOR Quality of Report. 102

RTL Register Transfer Level. 89, 90

SDC Synopsys Design Constraints. 92

VTC Voltage Transfer Characteristic. 43, 44, 46, 48, 71, 80