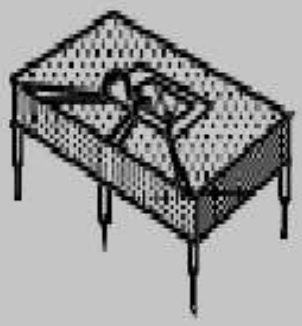
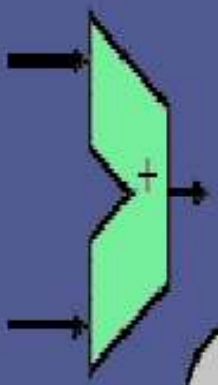


UNDERSTANDING

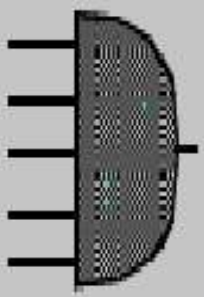
VLSI DESIGN

SYSTEM

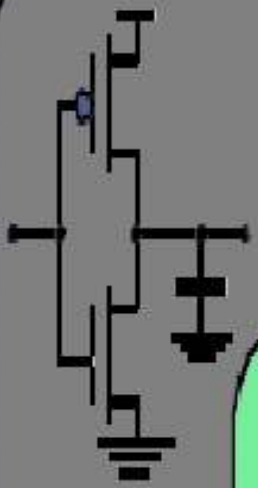
MODULE



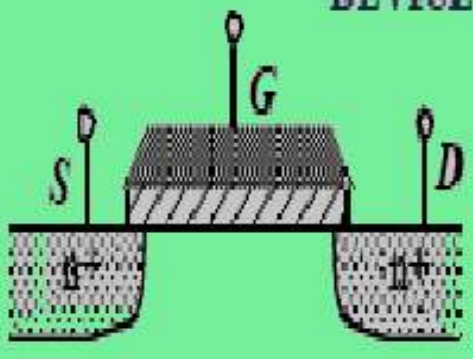
GATE



CIRCUIT



DEVICE



- RITU RAJ LAMSAL

UNDERSTANDING

VLSI DESIGN

- RITU RAJ LAMSAL (M.TECH (VLSI DESIGN)
(Gold Medalist)
Asst.Prof.VSB Engineering College

Few words

"THERE IS NO VIRTUE IN NOT KNOWING, WHAT CAN BE KNOWN"

"KNOWING IS NOT DOING, DOING IS DOING"

Welcome to this edition of the book *UNDERSTANDING VLSI DESIGN*. This book is written especially for those who are beginners and seeking their knowledge in the field of VLSI DESIGN .it is no doubt that many standard text and materials are available in the field of VLSI design, but my effort in bringing this book is to provide the fundamentals of the topics which is very hard to cover in single text book. the field of VLSI design is huge and has no depth many more is yet to come in future, in this context what I tried here is to gather the topics which gives the starting momentum for the readers to understand the terms and philosophy behind the VLSI design. in general the text book are written which focuses on some major issues but what I sincerely feel is until we have sound knowledge in fundamentals it is hard to follow the core topics. This text is especially designed to give information in various aspects of VLSI design.
My teaching experience and the interaction with the students made me to know how the things can be put in simplicity so that they get interest in the topic, this is what I have tried in this text. what I believe is once you started knowing the things ,I bet you go ahead .my sincere advice to the readers here is please have a patience, go thoroughly with the topics, Know the things and yes now you are ready to open the standard text book of your choice and topics, I am hopeful you wont find it difficult .
During the preparation of this text, I have research many textbook as reference; some materials are really the best way to understand the basics of the topics, which I have incorporated in this text. I have great respects for those authors, which have done such a great job.
Here is how the flow of this book starts.

Chapter 1 is introduction chapter, which deals with various information related to VLSI design. It gives the fundamental aspect of VLSI design. Before beginning to the other topic I request you to understand thoroughly what is VLSI design. The cover page of this text it self shows the abstraction of VLSI design.

Chapter 2 of this text gives you the idea related to the manufacturing process of integrated circuit for VLSI design. Obviously, to this date CMOS technology is the choice for VLSI design so I have incorporated various cmos process technology in this chapter.

Chapter 3 is related to the physics of the mos device, their characteristics and various design consideration.

Chapter 4 gives the necessary fundamentals to start writing the programming language code for VLSI design circuits. i have chosen the verilog hardware description language which is widely used in industries these days.

Chapter 5 is related to the design of small hardware modules using CMOS technology. Programmable logic device and ASIC design is also introduced.

Chapter 6 is intended for the test that has to be done while and after manufacturing the IC Chips. Various test methods and related topics are covered in this chapter.
Finally, I have included few exercise to recall the chapters.

I have always believed in learning so I look forward for your valuable suggestion to make this text better. I will be very much thankful for pointing and rectifying any mistakes that I might have committed in this text.

Last but not the least I take pride and pleasure in bringing this book in front of you, and here goes my sincere thanks to my publisher for the effort in bringing this book,thanks are also due to every individual who has supported me on this project. Finally I am thankful to my wife for her great encouragement and patience during this project. I dedicate this book to all my family members.

I hope you will enjoy reading this text as much as I enjoyed writing this.
Wish you all the best.

With warm regards
Ritu Raj Lamsal
Email: lamsal.raj@gmail.com

CONTENTS

CHAPTER 1.

CHAPTER 2

CHAPTER 3

MOS TRANSISTOR THEORY

CHAPTER 4

VERILOG HDL

CHAPTER 5

CHAPTER 6

CHAPTER1

- INTRODUCTION TO VLSI DESGIN
  - Historical prospective
  - VLSI design flow
  - Design hierarchy
  - Semiconductor technology
    - Bipolar circuit
    - Mos circuit
    - Pchannel MOSFET
    - Mos transistor operation mode
    - Various form of pull up
    - Mos transistor layers
    - Inverter design in nmos
    - Why mos device in VLSI design
    - Various mos transistor

1. INTRODUCTION TO VLSI SYSTEMS

1.1 Historical Perspective

The electronics industry has achieved a phenomenal growth over the last two decades, mainly due to the rapid advances in integration technologies, large-scale systems design - in short, due to the advent of VLSI. The number of applications of integrated circuits in high-performance computing, telecommunications, and consumer electronics has been rising steadily, and at a very fast pace. Typically, the required computational power (or, in other words, the intelligence) of these applications is the driving force for the fast development of this field. The current leading-edge technologies (such as low bit-rate video and cellular communications) already provide the end-users a certain amount of processing power and portability. This trend is expected to continue, with very important implications on VLSI and systems design. One of the most important characteristics of information services is their increasing need for very high processing power and bandwidth (in order to handle real-time video, for example). The other important characteristic is that the information services tend to become more and more personalized (as opposed to collective services such as broadcasting), which means that the devices must be more intelligent to answer individual demands, and at the same time they must be portable to allow more flexibility/mobility. As more and more complex functions are required in various data processing and telecommunications devices, the need to integrate these functions in a small system/package is also increasing. The level of integration as measured by the number of logic gates in a monolithic chip has been steadily rising for almost three decades, mainly due to the rapid progress in processing technology and interconnect technology. Table 1.1 shows the evolution of logic complexity in integrated circuits over the last three decades, and marks the milestones of each era. Here, the numbers for circuit complexity should be interpreted only as representative examples to show the order-of-magnitude. A logic block can contain anywhere from 10 to 100 transistors, depending on the function. State-of-the-art examples of ULSI chips, such as the DEC Alpha or the INTEL Pentium contain 3 to 6 million transistors.

ERA	DATE	COMPLEXITY
Single transistor	1959	less than 1
Unit logic (one gate)	1960	1
Multi-function	1962	2 - 4
Complex function	1964	5 - 20
Medium Scale Integration	1967	20 - 200 (MSI)
Large Scale Integration	1972	200 - 2000 (LSI)
Very Large Scale Integration	1978	2000 - 20000(VLSI)
Ultra Large Scale Integration	1989	20000 - ? (ULSI)

Table-1.1: Evolution of logic complexity in integrated circuits.

The most important message here is that the logic complexity per chip has been (and still is) increasing exponentially. The monolithic integration of a large number of functions on a single chip usually provides:

- Less area/volume and therefore, compactness
- Less power consumption
- Less testing requirements at system level
- Higher reliability, mainly due to improved on-chip interconnects
- Higher speed, due to significantly reduced interconnection length
- Significant cost savings

Therefore, the current trend of integration will also continue in the foreseeable future. Advances in device manufacturing technology, and especially the steady reduction of minimum feature size (minimum length of a transistor or an interconnect realizable on chip) support this trend. . At 1980 a minimum feature size of 0.3 microns was expected around the year 2000. The actual development of the technology, however, has far exceeded these expectations. A minimum size of 0.25 microns was readily achievable by the year 1995. As a direct result of this, the integration density has also exceeded previous expectations - the first 64 Mbit DRAM, and the INTEL Pentium microprocessor chip containing more than 3 million transistors were already available by 1994, pushing the envelope of integration density.

It can be observed that in terms of transistor count, logic chips contain significantly fewer transistors in any given year mainly due to large consumption of chip area for complex interconnects. Memory circuits are highly regular and thus more cells can be integrated with much less area for interconnects. Generally speaking, logic chips such as microprocessor chips and digital signal processing (DSP) chips contain not only large arrays of memory (SRAM) cells, but also many different functional units. As a result, their design complexity is considered much higher than that of memory chips, although advanced memory chips contain some sophisticated logic functions. The design complexity of logic chips increases almost exponentially with the number of transistors to be integrated. This is translated into the increase in the design cycle time, which is the time period from the start of the chip development until the mask-tape delivery time. However, in order to make the best use of the current technology, the chip development time has to be short enough to allow the maturing of chip manufacturing and timely delivery to customers. As a result, the level of actual logic integration tends to fall short of the integration level achievable with the current processing technology. Sophisticated computer-aided design (CAD) tools and methodologies are developed and applied in order to manage the rapidly increasing design complexity.

1.2 VLSI Design Flow

The design process, at various levels, is usually evolutionary in nature. It starts with a given set of requirements. Initial design is developed and tested against the requirements. When requirements are not met, the design has to be improved. If such improvement is either not possible or too costly, then the revision of requirements and its impact analysis must be considered. The Y-chart (first introduced by D. Gajski) shown in Fig. 1.1 illustrates a design flow for most logic chips, using design activities on three different axes (domains) which resemble the letter Y.

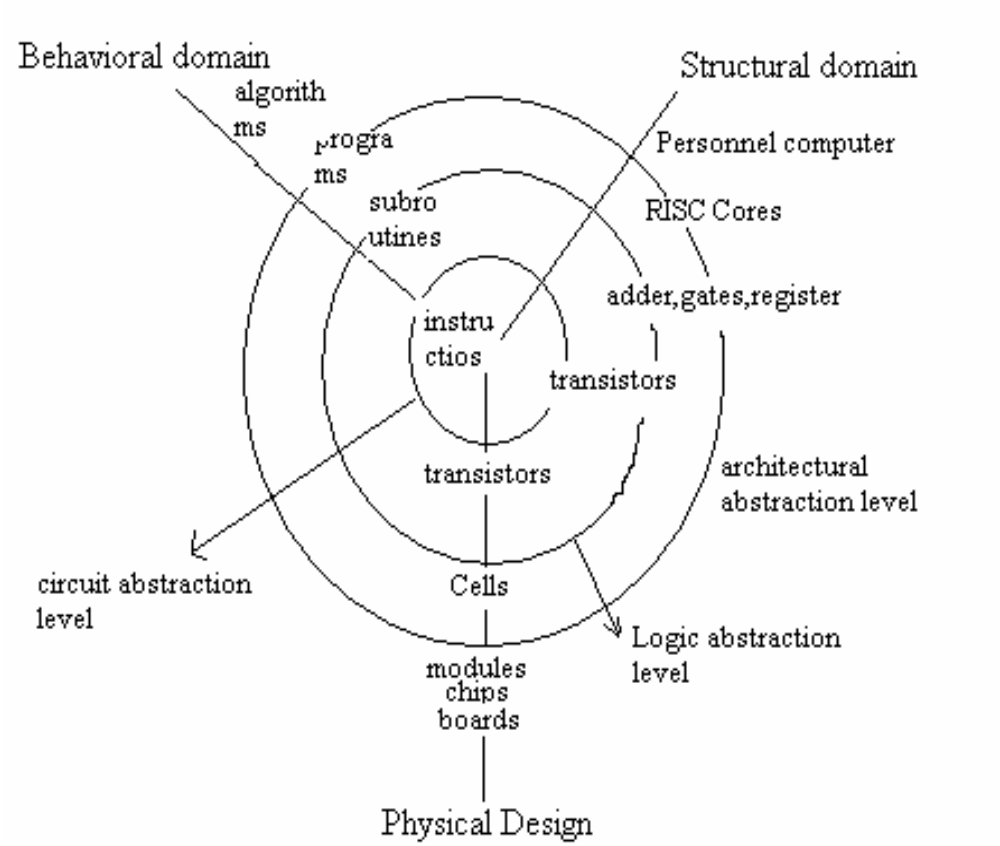


Figure-1.1: Typical VLSI design flow in three domains (Y-chart representation).

The Y-chart consists of three major domains, namely:

- behavioral domain,
- structural domain,
- Physical (geometrical layout) domain.

The design flow starts from the algorithm that describes the behavior of the target chip. The corresponding architecture of the processor is first defined. It is mapped onto the chip surface by floor planning.

The next design evolution in the behavioral domain defines finite state machines (FSMs) which are structurally implemented with functional modules such as registers and arithmetic logic units (ALUs). These modules are then geometrically placed onto the chip surface using CAD tools for automatic module placement followed by routing, with a goal of minimizing the interconnect area and signal delays.

The third evolution starts with a behavioral module description. Individual modules are then implemented with leaf cells. At this stage the chip is described in terms of logic gates (leaf cells), which can be placed and interconnected by using a cell placement & routing program. The last evolution involves a detailed Boolean description of leaf cells followed by a transistor level implementation of leaf cells and mask generation. In standard-cell based design, leaf cells are already pre-designed and stored in a library for logic design use. The simplified version of the VLSI design flow is shown in figure 1.2.

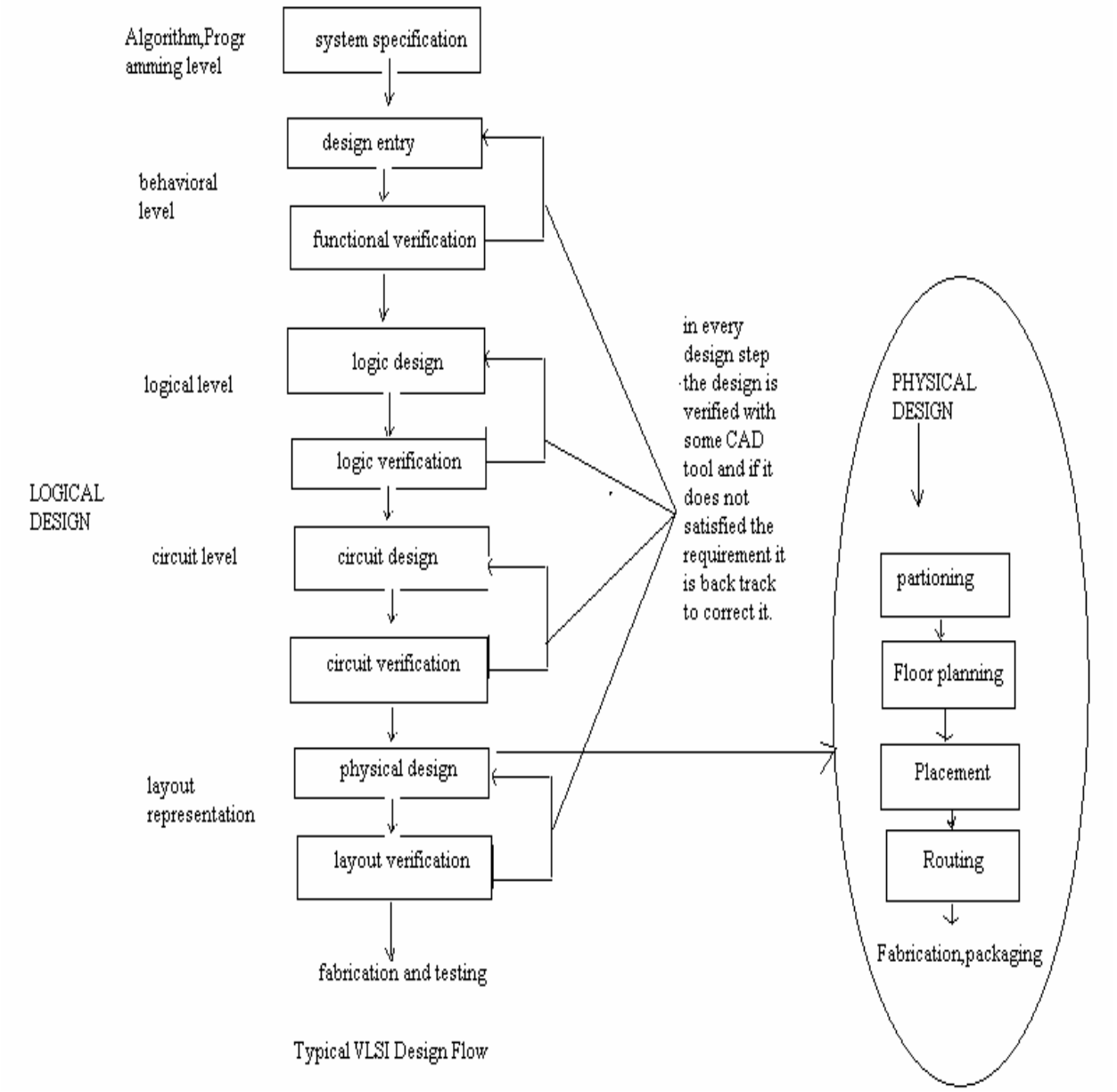


Figure-1.2: A more simplified view of VLSI design flow.

Figure 1.2 provides a more simplified view of the VLSI design flow, taking into account the various representations, or abstractions of design - behavioral, logic, circuit and mask layout. Note that the verification of design plays a very important role in every step during this process. The failure to properly verify a design in its early phases typically causes significant and expensive re-design at a later stage, which ultimately increases the time-to-market.

Although the design process has been described in linear fashion for simplicity, in reality there are many iterations back and forth, especially between any two neighboring steps, and occasionally even remotely separated pairs. Although top-down design flow provides an excellent design process control, in reality, there is no truly unidirectional top-down design flow. Both top-down and bottom-up approaches have to be combined. For instance, if a chip designer defined an architecture without close estimation of the corresponding chip area, then it is very likely that the resulting chip layout exceeds the area limit of the available technology. In such a case, in order to fit the architecture into the allowable chip area, some functions may have to be removed and the design process must be repeated. Such changes may require significant modification of the original requirements. Thus, it is very important to feed forward low-level information to higher levels (bottom up) as early as possible.

In the following, we will examine design methodologies and structured approaches which have been developed over the years to deal with both complex hardware and software projects. Regardless of the actual size of the project, the basic principles of structured design will improve the prospects of success. Some of the classical techniques for reducing the complexity of IC design are: Hierarchy, regularity, modularity and locality.

1.3 Design Hierarchy

The use of hierarchy or “divide and conquer” technique involves dividing a module into sub- modules and then repeating this operation on the sub-modules until the complexity of the smaller parts becomes manageable. This approach is very similar to the software case where large programs are split into smaller and smaller sections until simple subroutines, with well-defined functions and interfaces can be written. In Section 1.2, we have seen that the design of a VLSI chip can be represented in three domains. Correspondingly, a hierarchy structure can be described in each domain separately. However, it is important for the simplicity of design that the hierarchies in different domains can be mapped into each other easily.

As an example of structural hierarchy, Fig. 1.3 shows the structural decomposition of a CMOS four-bit adder into its components. The adder can be decomposed progressively into one- bit adders, separate carry and sum circuits, and finally, into individual logic gates. At this lower level of the hierarchy, the design of a simple circuit realizing a well-defined Boolean function is much easier to handle than at the higher levels of the hierarchy.

In the physical domain, partitioning a complex system into its various functional blocks will provide a valuable guidance for the actual realization of these blocks on chip. Obviously, the approximate shape and size (area) of each sub-module should be estimated in order to provide a useful floorplan. Figure 1.3 shows the hierarchical decomposition of a four-bit adder in physical description (geometrical

layout) domain, resulting in a simple floorplan. This physical view describes the external geometry of the adder, the locations of input and output pins, and how pin locations allow some signals (in this case the carry signals) to be transferred from one sub-block to the other without external routing. At lower levels of the physical hierarchy, the internal mask

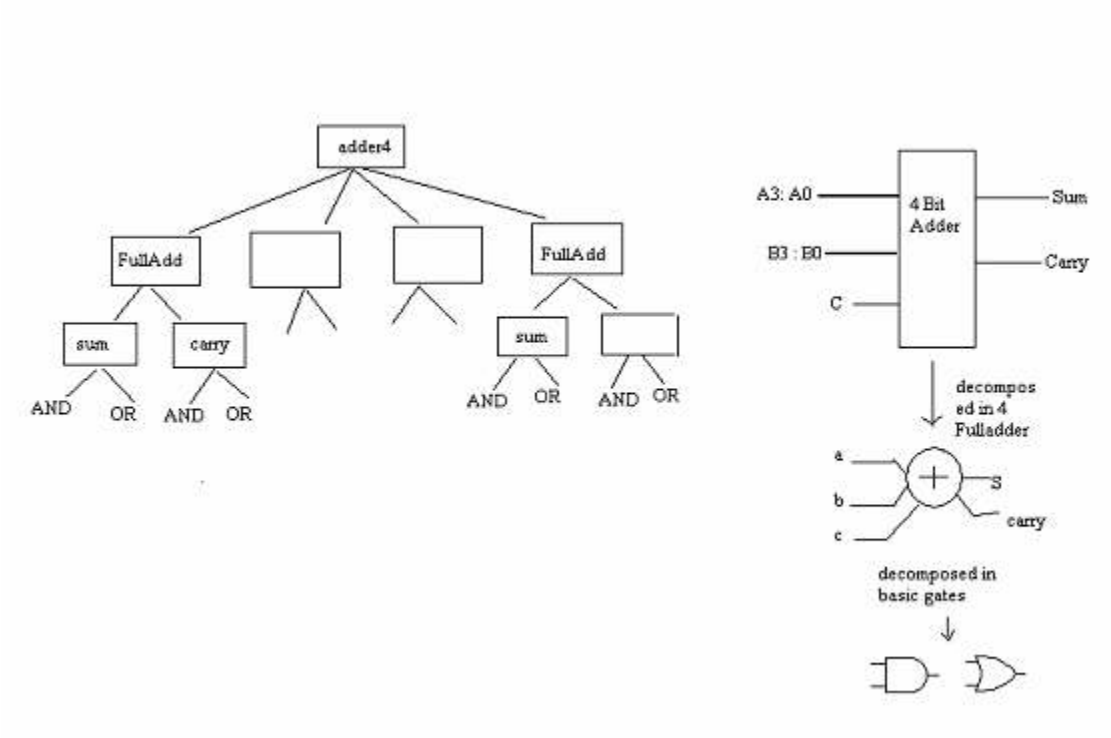


Figure-1.3: Structural decomposition of a four-bit adder circuit, showing the hierarchy down to gate level.

.

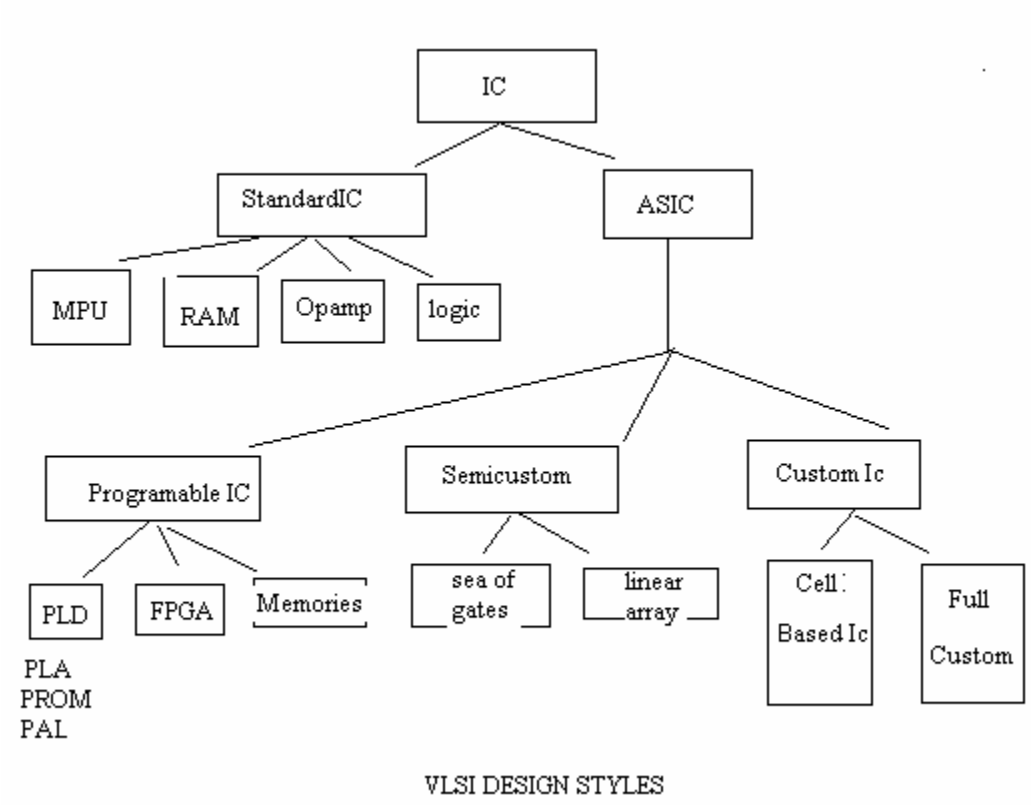


Figure-1.4: Overview of VLSI design styles.

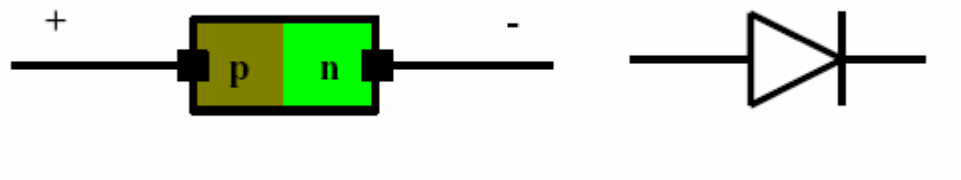
1.4 Semiconductor technology

Semiconductors can be made from crystalline silicon into which impurities have been introduced:

_ A pentavalent atom implant such as phosphorous gives free electrons, creating an n-type region.

_ A trivalent atom implant such as boron gives free holes, creating a p-type region.

The junction of an n-type and a p-type region in a single crystalline lattice creates a diode which only conducts if it is forward biased with the p-type region (the anode) more positive than the n-type region (the cathode).



A light emitting diode has the additional property that it glows when current is flowing through it. It is prudent to limit this current to a few milli-Amps by means of a kilo ohm series resistor.

Digital switching

Most digital logic is based on the idea of switching signals between a high voltage (which we will usually treat as being 5V, although modern systems more commonly use 3.3V or less) and a low voltage (0V, or ground). The sense may be determined by current flowing or not (as in bipolar circuits) or by the presence or absence of charge (as in MOS circuits). A logic function takes some input signals and computes an output function using pull-up and pull-down circuits
Which may be passive (always switched on) or active (selectively switched).

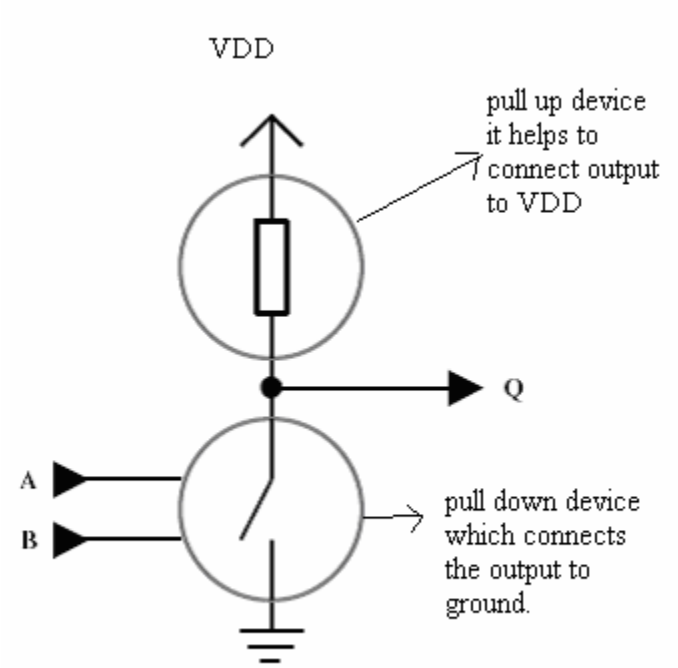


figure 1.5 Passive pull-up and active pull-down

Passive pull-up and active pull-down

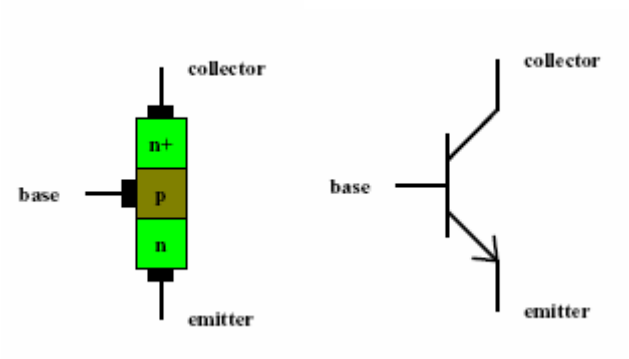
The figure 1.5 shows a circuit with an active pull-down and a passive pull-up. The pull-down can be thought of as a remote-control switch, usually made from transistors but possibly relays or valves.

A further complexity with MOS circuits is that the charge on wires persists after they have ceased to be driven; this means that the wires have a memory (typically lasting a thousandth of a second or so) of the last value driven on them.

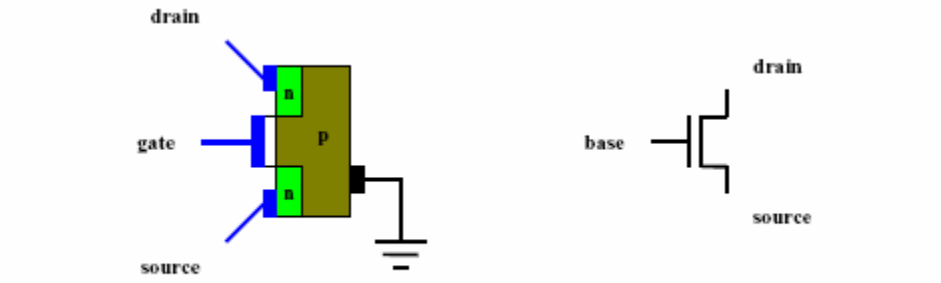
1.4.1 Bipolar circuits

A bipolar transistor is formed by a sandwich of n-type, p-type and n-type regions in a single crystalline lattice. It can be thought of two diodes connected anode-to-anode such that a current through the forward biased diode overwhelms the reverse biased diode.

npn bipolar transistor:
A small current flowing from the base to the emitter of an npn transistor induces a large current from the collector to the emitter. A pnp transistor has the opposite polarity.These can be used to construct a NAND gate using transistor-transistor logic (TTL).



1.4.2 MOS circuits

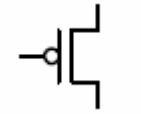


n-channel enhancement mode metal-oxide-semiconductor field-effect transistor

An enhancement mode, n-channel, metal-oxide-silicon field-effect transistor (nMOS FET) is formed on a crystal of p-type silicon. Two n-type regions (known as diffusion) lie on either side of a region of the p-type substrate which is covered by a thick layer of insulating silicon dioxide (or oxide) and a metal plate.
When the gate is positive with respect to the source, an n-type channel is formed under the gate and current is conducted from drain to source. Even when turned on, a MOS transistor has a resistance of about 10 kΩ.

The construction of the transistor is symmetric with respect to the source and drain - the labels merely indicate the relative voltages. This contrasts with the different processing used to make the collector and emitter of a bipolar transistor.

1.4.3 A p-channel MOSFET



pmos mosfet has the opposite polarity and conducts when its gate is low. However, the resistance of a p-type channel is about 2½ times that of an n-type channel of the same size. In integrated circuits, the metal gate is replaced by one made from polycrystalline silicon (or polysilicon) for ease of fabrication.

1.4.4 The nMOS transistor operates in three modes:

_ **Off** when Vgs < Vt
_ **saturated** when Vgs > Vt and Vds > Vgs – Vt
_ **Linear** when Vgs > Vt and Vds < Vgs – Vt

Where Vt is the threshold voltage (= 0.2 Vdd = 1V for a 5V system)
Note that, even when the transistor is turned on, the source voltage can not rise above the gate voltage less the threshold voltage.
The threshold voltage can be adjusted by implanting further impurities into the channel regions.It can even bemade negative (Vt = -0.8 Vdd = -4V), giving a depletion mode nMOS FET which always conducts. This can be used as a compact way of making a resistor.

1.5 Various form of PULL UP

.There are three ways that the pull-up could be made:

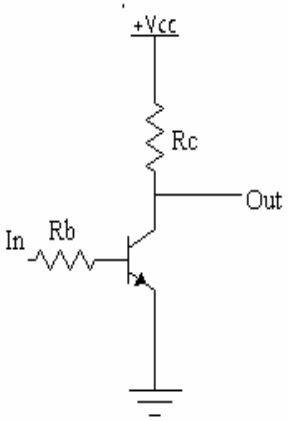
- **A resistor** – using polysilicon (which is the most resistive material available in a MOS process) this would have to be several hundred times the size of the pull-down transistor.
- An enhancement mode transistor with its gate wired high – this could never pull the output above Vdd – Vt.

- A depletion mode transistor with its gate wired to its source is used in practice.

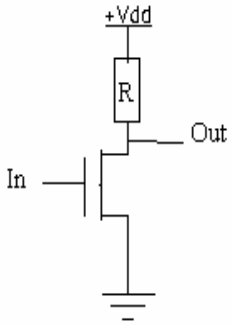
Depletion type transistor has channel implanted during maufacturing.so it is normally ON device

Nmos Depletion type

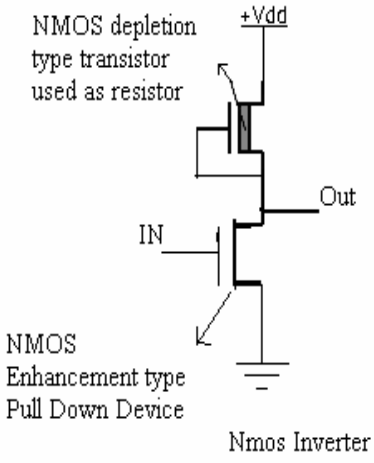
Figure 1.6 shows the various form of pull up device used to construct the inverter circuit



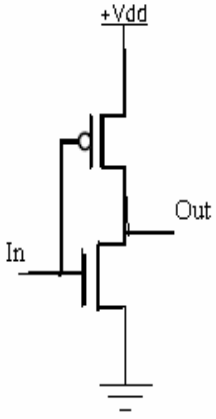
Inverter using NPN transistor.here resistor Rc acts as pull up device,and transistor acts as pull down device.when Input is high transistor conducts collector current flows all the drop takes place at Rc and output is pull down to ground,when input is low transistor is off ,no current flows at collector terminal so no drop in Rc hence ouput is high.



Nmos inverter with resistor pull up



Nmos Inverter



CMOS Inverter

Figure 1.6 Various form of pull up to realize the inverter circuit

1.6 MOS Transistor Layers:
There are several layers in an nMOS chip:

- a p-type substrate
- paths of n-type diffusion
- a thin layer of silicon dioxide
- paths of polycrystalline silicon
- a thick layer of silicon dioxide
- paths of metal (usually aluminium)
- a further thick layer of silicon dioxide With contact cuts through the silicon dioxide where connections are required.

The three layers carrying paths can be considered as independent conductors that only interact where polysilicon crosses diffusion to form a transistor. These tracks can be drawn as stick diagrams with

_ Diffusion in green
_ Polysilicon in red
_ Metal in blue

Using black to indicate contacts between layers and yellow to mark regions of implant in the channels of depletion mode transistors.refer figure 1.7.

With CMOS, there are two types of diffusion: n-type is drawn in green and p-type in brown. These are on the same layers in the chip and must not meet. In fact, the method of fabrication required that they be kept relatively far apart.
Modern CMOS processes usually support more than one layer of metal. Two are common and three or more are often available.
Actually, these conventions for colours are not universal; in particular, industrial (rather than academic) systems tend to use red for diffusion and green for polysilicon. Moreover, a shortage of coloured pens normally means that both types of diffusion in CMOS are coloured green and the polarity indicated by drawing a circle round p-type transistors or simply inferred from the
context. Coloring for multiple layers of metal are even less standard.
There are three ways that an nMOS inverter might be drawn:

1.7 Inverter designs in nMOS

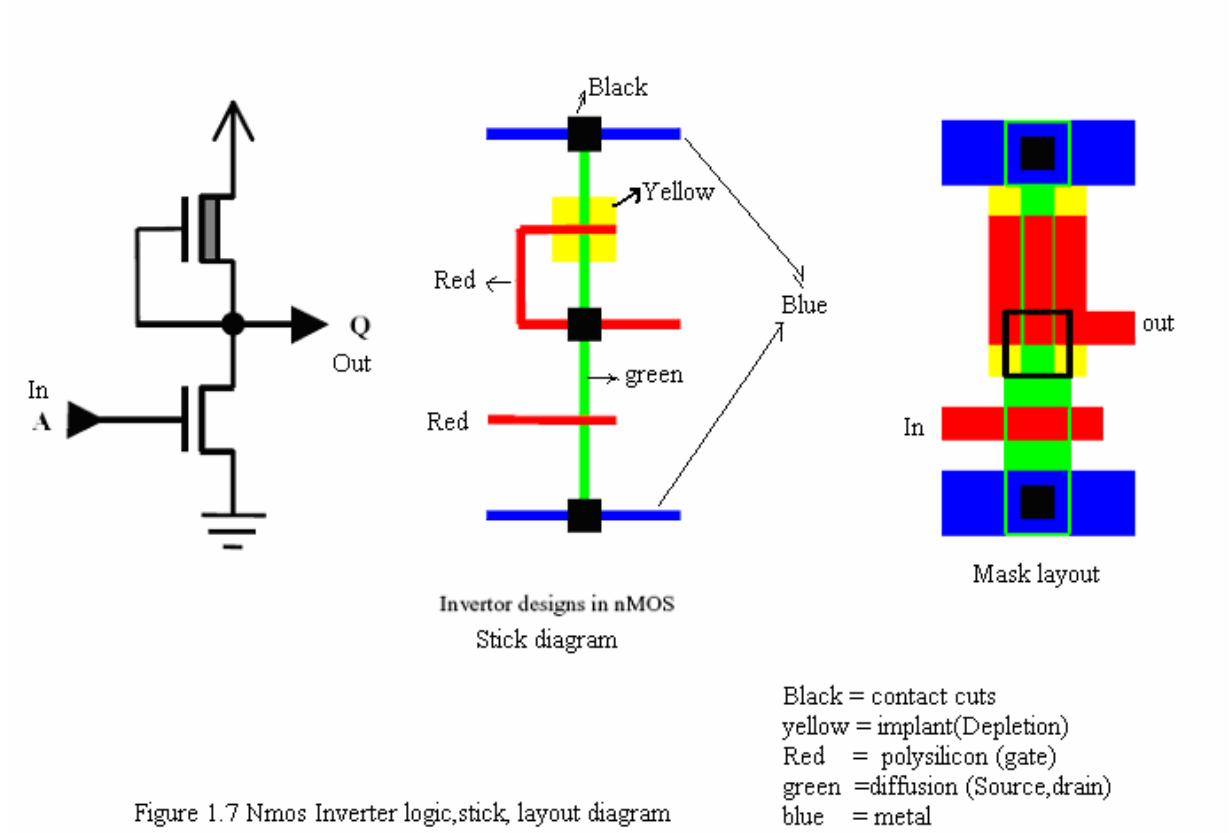


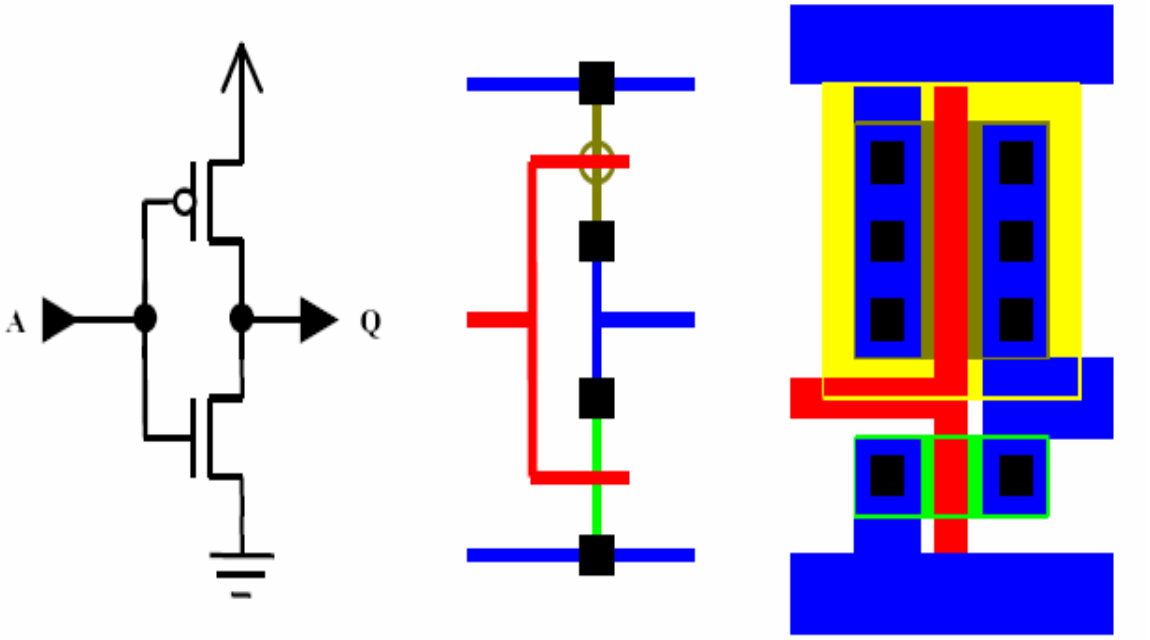
Figure 1.7 Nmos Inverter logic,stick, layout diagram

The three different representations are useful in different contexts:
_ a circuit diagram – used to plan the logic of the system;
_ a stick diagram – used to plan the topology of a layout, committing signals to particular layers; and
_ layout – final decisions of sizes

The equivalent pictures in CMOS are:

Inverter designs in CMOS

The equivalent pictures in CMOS are:



Invertor designs in CMOS

This layout shows the input arriving through polysilicon on the left and the output leaving through metal on the right. A second layer of metal might be used to allow connections above and below the invertor with a third layer left free to run other, quite separate, signals (such as a global clock) across the top of the invertor.The following design runs power and ground in the second metal layer and signals in the first, with the polysilicon hidden underneath it.
We will discuss about the layout diagram more in details later in this text.

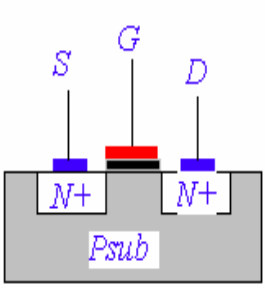
1.8 WHY MOS transistor in VLSI DESIGN:

As the name VLSI suggest that there are millions of transistors inside a single chip. Each transistor needs a power to operate and it dissipates some power. Consider a single transistor which dissipates power in microwatts, now if you combine those millions of transistor the total power dissipation will be huge in watts. If you look at the Mos transistor structure the gate of the mos transistor is isolated by the insulator hence it has very high input impedance which leads the mos device to consume less power. Also the Mos device provides good logic than bipolar device. The packing density (i.e number of transistor in given area) of Mos Technology is more than Bipolar Junction technology. However bipolar transistor also proves superior to mos in some aspects like current delivering, transconductance, operating frequency etc. so there is a need of fabricating both the mos device and bipolar device in single technology called bicmos process to achieve advantage of both device. The mos device is place at the input side for low power consumption and good logic level and bipolar device at the output side for driving large load. The comparison between bipolar and cmos technology can be summarized as follows.

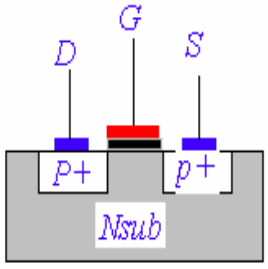
CMOS technology	Bipolar technology
▪ Low static Power dissipation	high power dissipation
▪ High input impedance	low input impedance
▪ Scalable threshold voltage	
▪ High noise margin	low voltage swing logic
▪ High packing density	low packing density
▪ High delay sensitivity to load	low delay sensitive to load
▪ (Fan–out limitations)	
▪ Low output drive current	high output drive current
▪ Low Transconductance	high Transconductance
▪ Bidirectional capability	
▪ (Drain and source are interchangeable)	high Ft at low current
▪ A near ideal switching device	essentially unidirectional

1.9 Various mos transistor:

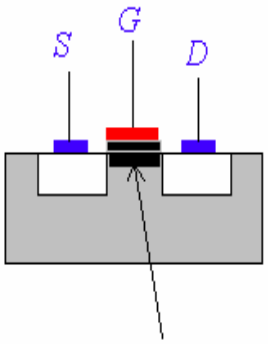
So to get the advantage of both technology BICMOS process is introduced but with increase in cost. Generally Bicmos process technology is used for mixed signal design where both analog and digital parts are fabricated in a single chip.



Nmos Transistor(Enhancement) is formed on ptype substrate,source and drain region are formed using diffusion of N+ type material.positive voltage Vgs is required to create a channel drain is connected to power and source to ground



Pmos transistor is formed in Nsubstrate,source and diffusion region are formed with heavily doped Ptype material(p+),Negative Voltage Vgs is applied to form the channel.source is connected to power and drain to ground



depletion type transistor has Channel already Present while maufacturing,

(note there is no physical distinguish between source and drain if you connect the terminal to ground for nmos it is source other one is drain.the connection is opposite to pmos

PMOS AND NMOS TRANSISTORS

CHAPTER 2

- CMOS PROCESS TECHNOLOGY
- Overview of silicon semiconductor technology
- Basic cmos technology
- Nwell process
- Pwell process
- Twin-tub cmos process
- Silicon on insulator (SOI) process
- Interconnect
- Circuits element
- Electrically alterable ROM
- Bicmos process
- Latchup and prevention
- Layout design rules
- Physical design of basic gates
- Cmos inverter,nand,nor layout design
- CAD tools

Chapter 2

The aim of this chapter is to make readers familiar with the technology and manufacturing process of semiconductor device especially CMOS Technology.

2.0 Overview of silicon semiconductor technology:

Silicon technology is the term which refers to the process and technology to manufacture the semiconductor device (also integrated circuit (IC)).as we know that pure semiconductor materials like silicon, germanium are added with pentavalent and trivalent material to make ntype and ptype semiconductor materials. Semiconductor device are composed of materials like ntype,ptype,substrate,oxide etc,whatever the scale of integration be it small scale or very large scale integration the silicon has to under go various process to form a device. Several steps are required to process the silicon to realize a physical semiconductor device. Some of the basic process involved in manufacturing the IC is as follows.

Wafer preparation
Oxidation
Epitaxy
Deposition
Ion implantation
Diffusion

Wafer preparation:

Wafers are the base material for manufacturing the integrated chips. Wafers are thin silicon cylindrical disks cut from the silicon ingot.(it resembles like a cdrom disk).The diameter of the wafer range from 75mm to 230 mm and the thickness less than 1 mm. A single wafer may house number of chips depending upon the complexity and size of the chip.(refer figure 1.0)

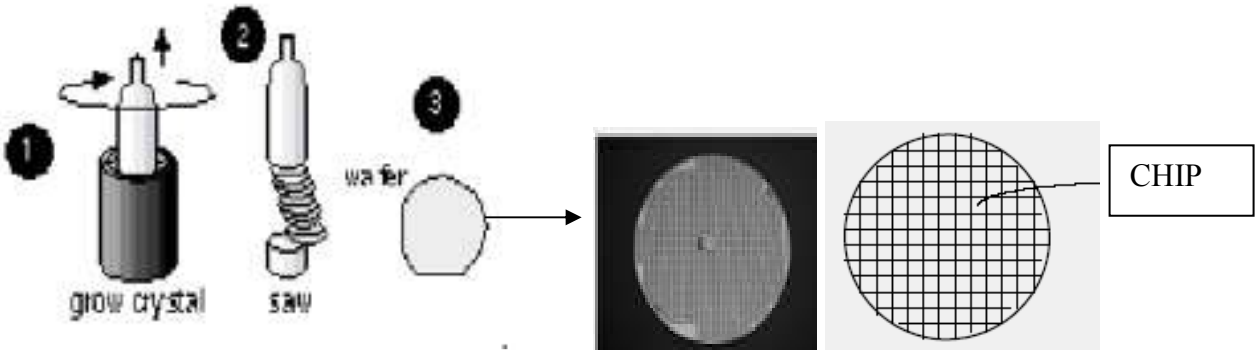


Figure 1.0 crystal growth wafer formation and chip

The molten silicon is added with impurities to get desired electrical properties .single crystal is grown towards the rotation and the growth direction of the molten silicon produce the silicon ingot (figure 1.0). The wafers are then sliced by the diamond saw. The rate of the growth varies from 30 mm to 180mm/hr.The wafer is polished flat in one of the surface, to make it scratch free.

Oxidation:

Oxidation is the process of forming oxide layer that is silicon dioxide sio2. Oxidation process is carried out using wet oxidation or dry oxidation process. in wet oxidation process the wafer is placed in the moisture (water vapor) with the temperature around 1000 degree centigrade . The reaction takes place and oxide layer is formed. in dry oxidation the wafer is heated in pure oxygen with the temperature around 1200 degree centigrade.

In the process of oxidation, some silicon is consumed. Since the volume of sio2 is double than the silicon it extends vertically equal in source and drain region of mos transistor and is known as field oxide. The figure 1.1 shows the oxidation and photoresist process.

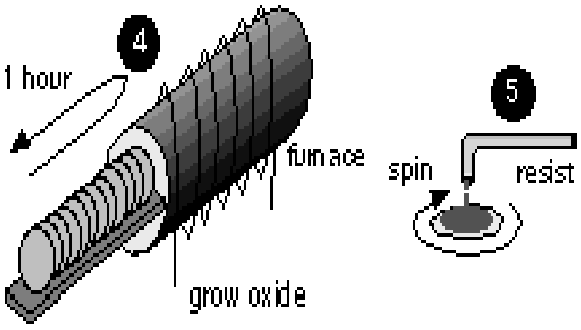


Figure 1.1 oxide formation and photoresist for mask

Epitaxy: the semiconductor device is formed with n type and p type materials with different doping concentration and proportion. Epitaxy is one of the method to create thin film of dopant materials which may form a thin layer of desire type and property.

Ion Implantation:

This is the process by which highly energized acceptor or donor atoms are impinged on the silicon substrate .these atoms travels below the silicon surface with varying concentration, with the silicon heated around 800 degree centigrade the diffusion process takes place i.e. the impurities from higher concentration region comes towards lower concentration region and equal distribution of impurities occurs on that region.

Deposition:

This process is carried out to evaporate the dopants from the region to reduce the concentration, to completely remove the dopants,or to drive the impurities from the surface of the silicon to the bulk.

Beside this basic process various maskings are used. The masking is used to create several structures like polysilicon gate,diffusion region and thinand thick oxide regions.The mask can be of materials like photoresist, sio2,polysilicon and silicon nitride.

2.1 Basic cmos technology

cmos refers to the combination of nmos and pmos transistor which are build in same substrate.so in any cmos technology the goal is to fabricate two transistor nmos and pmos in same substrate(Bulk).There are mainly four basic cmos process technology which is used to manufacture cmos devices. They are

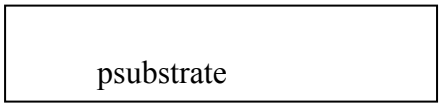
- n-well process
- p-well process
- twin-tub process
- silicon on insulator

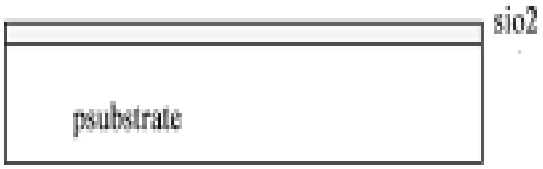
2.1 Nwell process:

In nwell process, the starting material is p substrate. A pure silicon wafer is doped with p type impurities to form p substrate. Now the various processes are done in this wafer to form the cmos device. These processes are given below.

Wafer and oxide formation:

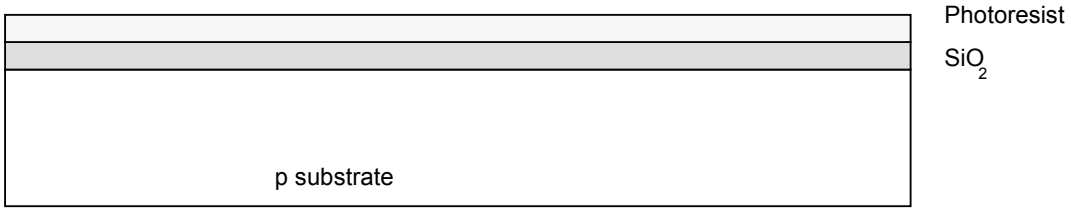
Start with blank wafer (typically p-type where NMOS is created) . Grow SiO2 on top of Si wafer .the wafer is heated to 900 – 1200 C with H_2O or O_2 in oxidation furnace to create this oxide layer.





Photoresist mask and etching

Add photoresist layer over the oxide layer. Photoresist is a light-sensitive organic polymer which Property changes when exposed to light. photoresist materials gets polymerized and becomes harder when exposed to ultra violet rays. Expose photoresist to Ultra-violate (UV) light through the n-well mask, Strip off exposed photoresist with chemicals. Etch oxide with hydrofluoric acid (HF) which Only attacks oxide where photoresist has been exposed .Strip off remaining photoresist



NWELL FORMATION:

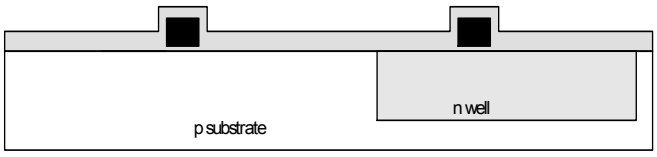
- Remove oxide and photoresist layer where n-well should be built (4 to 5 micron deep). .nwell is created by Implantation or diffusion of n dopants into exposed wafer to form n-well. Diffusion is achieved by placing wafer in furnace with arsenic-rich gas and heating until As(arsenic) atoms diffuse into exposed Silicon. Ion Implantation is carried out with following steps:
  - ✓ Blast wafer with beam of As ions
  - ✓ Ions are blocked by SiO2 and only enters where si is exposed
  - ✓ SiO2 shields (or masks) areas which remain p-type
-
-

POLYSILICON PATTERN FOR GATE STRUCTURE:

Deposit very thin layer of gate oxide(< 20 Å).

- Polysilicon is deposited using chemical vapour deposition for gate structure.

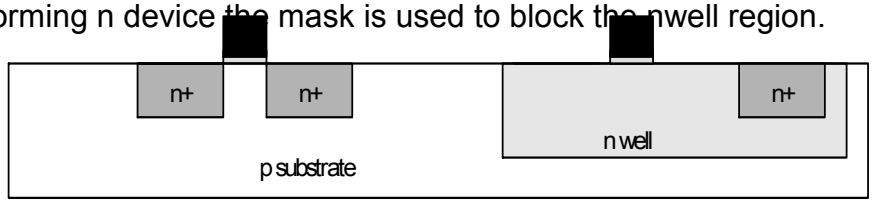
Use gate-oxide/Polysilicon and masking to expose where n+ dopant should be diffused or implanted



N+diffusion:

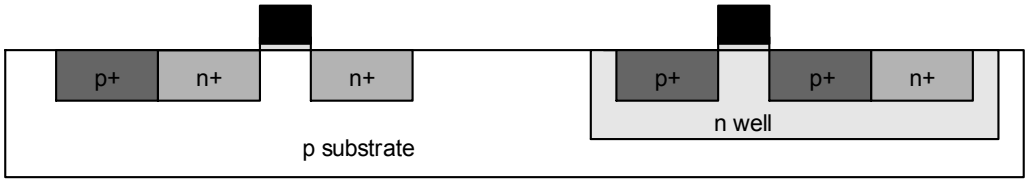
N+ diffusion is formed in p substrate to form source and drain region and this becomes the nmos transistor.while masking is done to cover the nwell region

where pmos transistor is to be formed.To create n device, Pattern oxide and form n+ regions. This is Self-aligned process where gate blocks n-dopants. While forming n device the mask is used to block the nwell region.



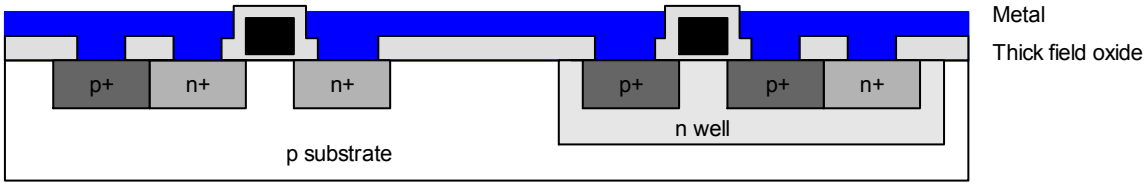
P+DIFFISION:

The mask opposite to the N+ process is used to prevent the ndevice region and p+ diffusion is formed in nwell region by diffusion or impalantation method.this P+ diffusion region creates the sorce and drain region for pmos transistor which resides in Nwell.



CONTACTCUTS AND METTALIZATION:

Now we need to wire together the devices, Cover chip with thick field oxide (FO).Etch oxide where contact cuts are needed. place aluminium or copper over whole wafer. Pattern to remove excess metal, leaving wires.hence two transistor nmos and pmos is formed in p substrate and this device is known as cmos device.



Summary of nwell process:

The steps in nwell process can be summarized as

- Start with p substrate
- create n well region in p substrate (4-5 micron deep)
- define nmos and pmos active area(gate, source and drain region)
- field and gate oxidation(Thinox region)
- form and pattern Polysilicon for gate structure.

- P+ diffusion
- N+ diffusion
- Contact cuts,metallization,bonding pads and packaging.

2.2 Pwell process:

The steps for p well process are similar to nwell process .in pwell process the starting substrate is n type .pwell is created in this nsubstrate.in pwell N+ diffusion region is created by diffusion or ion implantation method which defines source and drain region.the polysilicon is patterned which forms a gate structure.so nmos device is created in pwell. pmos device is formed in n substrate. The basic steps are summarized as follows.

- Start with n type substrate
- Define the area in which deep pwell diffusions are to take place the depth of the well is around 4 to 5 micron.
- Define the nmos and pmos active areas(diffusion region).
- Define thinox region.
- Form and pattern polysilicon for gate region
- N+diffusion
- P+diffuson
- Contact cuts
- Metallization ,bonding pads and packazing.

2.3 Twin-Tub (Twin-Well) CMOS Process:

This is advance cmos process. in this process both nmos and pmos device are formed in separate nwell and p well, hence the name twin tub .nmos device is formed in Ptub and pmos device is formed in Ntub. Generally, the starting material is a n+ or p+ substrate, with a lightly doped epitaxial layer on top. This epitaxial layer provides the actual substrate on which the n-well and the p-well are formed. Since two independent doping steps are performed for the creation of the well regions, the dopant concentrations can be carefully optimized to produce the desired device characteristics. This technology provides the basis for separate optimization of the nMOS and pMOS transistors, thus making it possible for threshold voltage, body effect and the channel transconductance of both types of transistors to be tuned independently. In the conventional n-well CMOS process, the doping density of the well region is typically about one order of magnitude higher than the substrate, which, among other effects, results in unbalanced drain parasitics.this helps to prevent the latchup problem which occurs due to the formation of parasitic components,we will discuss latchup phenomenon later in this text.The twin-tub process avoids this problem.

The process sequence is similar to nwell apart from the tub formation where both pwell and nwell are formed. Some of the basic steps for twin tub process are

- Tub formation(formation of separate Nwell and p well in a substrate)
- Thin oxide construction
- Source and drain implantations.
- Contact cut definition
- Metallization.

The figure (2.0) shows the cross section of twin tub cmos process where both n and p type transistor resides in p tub and n tub respectively.

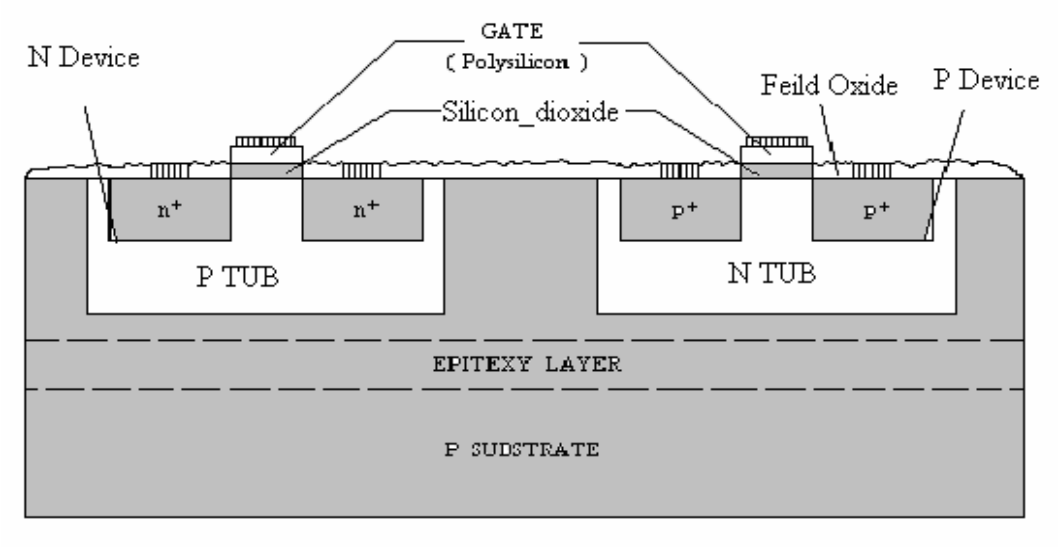


Figure-(2.0) : Cross-section of nMOS and pMOS transistors in twin-tub CMOS process.

2.4 Silicon-on-Insulator (SOI) CMOS Process

silicon on insulator is also an advance semiconductor technology where silicon is replaced as a substrate material**.**Rather than using silicon as the substrate material, technologists have sought to use an insulating substrate to improve process characteristics such as speed and latch-up susceptibility.(we will discuss Latchup later in this text). The SOI CMOS technology allows the creation of independent, completely isolated nMOS and pMOS transistors virtually side-by-side on an insulating substrate (for example: sapphire). The main advantages of this technology are the higher integration density (because of the absence of well regions), complete avoidance of the latch-up problem, and lower parasitic capacitances compared to the conventional n-well or twin-tub CMOS processes. A cross-section of nMOS and pMOS devices in created using SOI process is shown in Fig. (2.1). The SOI CMOS process is considerably more costly than the standard n-well CMOS process. Yet the improvements of device performance

and the absence of latch-up problems can justify its use, especially for deep-sub-micron devices.

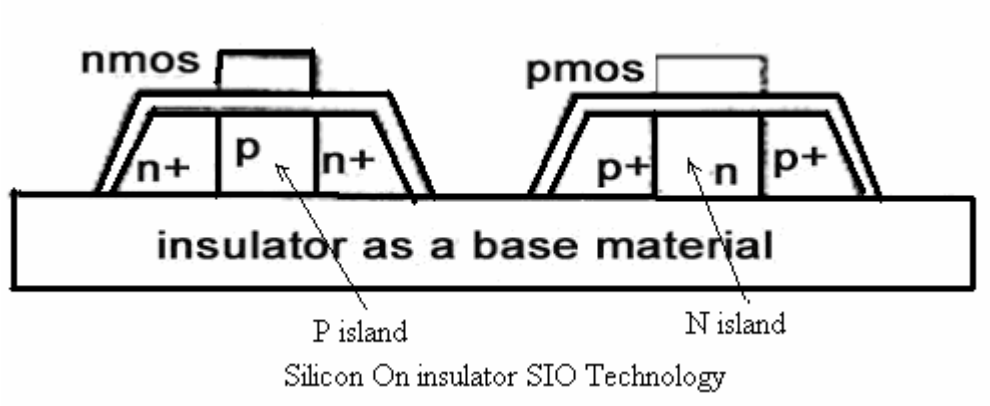
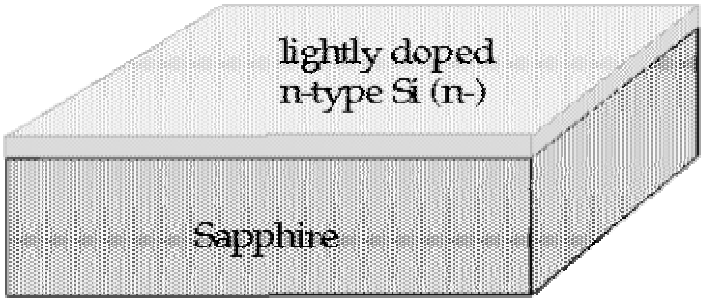


Figure 2.1 SOI process

The basic SOI process can be explain as follows .the figure numbers refers to the steps.

A thin film 7-8 micrometer of very lightly doped n type si is grown over an insulator. Sapphire or sio_2 is a commonly used insulator.
Anisotropic etching is required to etch away the silicon except where a diffusion area will be needed.

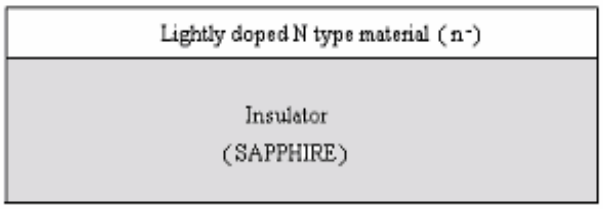
1)



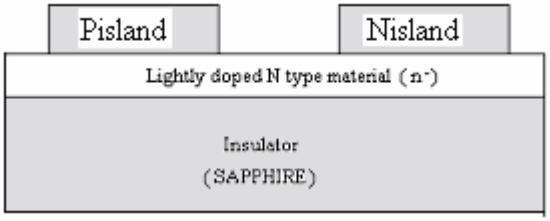
p and n island are formed by implanting p type and n type impurities.photoresist mask is used for forming island separately.
Thin gate oxide is grown all over the silicon using oxidation process.
Polysilicon film is deposited over the oxide; it is then pattern and mask to form gate structure in both P Island and N Island.
Now n+ diffusion and p+ diffusion process is carried to form source and drain structures in both the island. Contact cuts, metallization and packaging is done finally.

The steps can be summarized as below in the figure2.2.

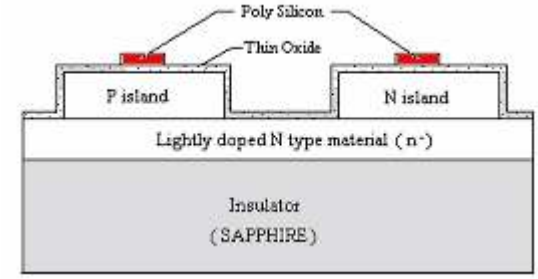
Step 1.Sapphire as an insulator



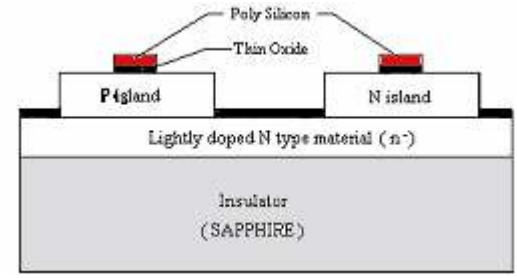
Step2: place lightly doped silicon (n^-) over sapphire.



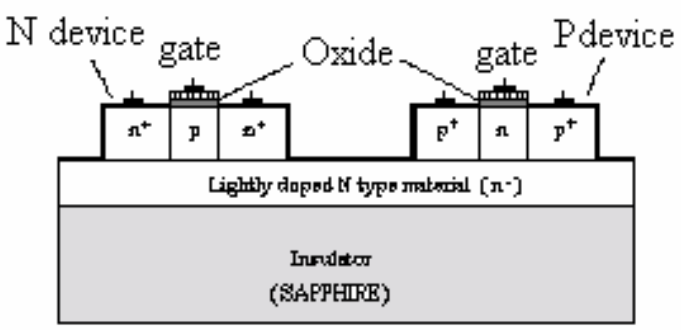
Step3: create P Island and N Island over the lightly doped silicon



Step4: cover the p and N Island with thin oxide and place the polysilicon for gate structure.



Step5: remove the thin oxide layer to create N and p diffusion region



Step6: source and drain region are formed in both island using n+ and p+ diffusion, oxidation and metallization is done for contacts .

Figure 2.2 fabrication steps of silicon on insulator process.

Advantages:

- Closer packing of p- and n-transistors, due to absence of wells.
- Absence of latch-up problems.
- Only “sidewall” areas of source and drain diffusions contribute to parasitic junction capacitance, faster devices.
- Leakage currents to substrate and adjacent devices almost eliminated.
- Enhanced radiation tolerance.

Disadvantages:

- No substrate diodes, inputs more difficult to protect.
- Device gains are lower, I/O structures must be larger.
- Density of contemporary digital processes is actually determined by number and density of metal interconnection layers.
- Sapphire and silicon on SiO_2 substrates are considerably more expensive.

Cmos process enhancements:

Many additional enhancement is added to cmos process for various reasons like, to increase the routablity, to provide high quality capacitor and resistors .to achieve this the process includes, Various metal layers (double,triple,quad) and Two or more poly layers.

3.interconnects:

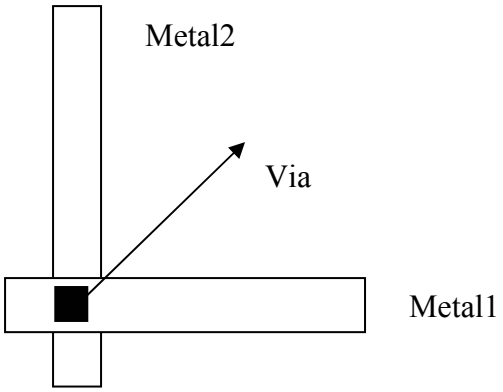
Due to the technology advancement, it is now, possible to house millions of transistors in a single chip. These transistors has to be interconnected to realize the circuit, this .is very cumbersome process hence to make it easy generally transistors are formed in one layer and interconnecting wires are formed in other layers. Generally 2 layers are used for interconnection,(now a days more than 2 layers are used for interconnection, routing) .there may be several case where different type of interconnection is required. The interconnection can be

- metal interconnection

- polysilicon/refractory metal interconnection
- local interconnection

Metal interconnect:

Signals like clock and power has to be connected for many device inside the IC and the problem is to interconnect the signals .the routing of signals is done using various metal layers. Normally metal1 is used for horizontal routing and metal2 is used for vertical routing. Aluminum is generally used as metal .more than two layers of metals are used theses days for routing the signals .the first layer of metal and the second layer of the metal is contacted using via .contact cuts are used to connect metal and poly or diffusion. Minimum separation is required for contact cut and via. Metal interconnection is especially used for global routing. The two metal processes can be briefly summarized as:



- The oxide below the first metal layer is deposited by atmoshpheric Chemical vapor deposition method.
- The second oxide layer between the two metal layer is applied in similar manner.
- Depending on the process removal of the oxide is done using plasma etching (via)

Polysilicon metal interconnect:

The gate of the transistor is made up of polysilicon material however for long interconnection the polysilicon introduces delay because of its internal resistance. The sheet resistance of doped polysilicon is around 20 to 40 ohms/square. the sheet resistance of the polysilicon can be reduced by combining it with refractory metal .combining polysilicon with slicide(sheet resistance 1 to 5 ohms/square) forms polycide which can be used as a gate material. This process reduces the complexity for interconnection of second layer as it can be used to interconnect instead of metal. This can be used for moderate distance interconnection.

Local interconnect:

Interconnection within the cell can be done using slicide, local interconnection allows direct interconnection between the polysilicon and diffusion thus avoiding metal interconnection (also avoids contacts hence consuming less space).the polysilicon and diffusion can be interconnected using local interconnection.

4 .circuit elements

Beside transistor, resistor and capacitors are also the major circuit element to realize the device. The general resistor is difficult to fabricate in cmos technology and also takes large space, hence the resistor in cmos technology are made up of polysilicon with varying doping concentration.

4.1 resistors and capacitors:

Polysilicon if left undoped is highly resistive material. This property is used to build resistors are in the range of tera ohms.10^{12} ohms.

For mixed signals a resistive material such as nichrome may be added to produce high value, high quality resistors. The resistor produced can have good accuracy .the resistor formed with this approach has excellent temperature stability and reliability.

Capacitors are essential part in dynamic memories and switch capacitor analog circuits. Small high value/area capacitors are required for memory elements where as good quality capacitors are needed for switch capacitor circuits. as we know if two conducting layer is separated by the insulator a capacitor is formed, with this concept the capacitor in cmos technology is built by adding additional polysilicon layer with oxide as insulator .so two polysilicon layer is sandwich with oxide layer in between to form the capacitor. refer figure(4.0)

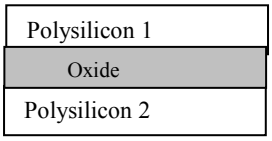
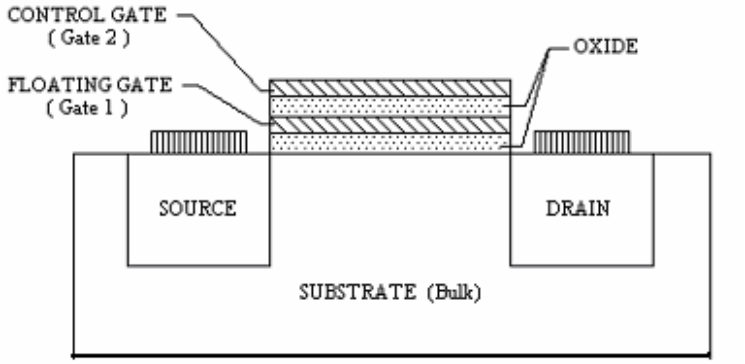


Figure 4.0 structure of capacitor in cmos technology.

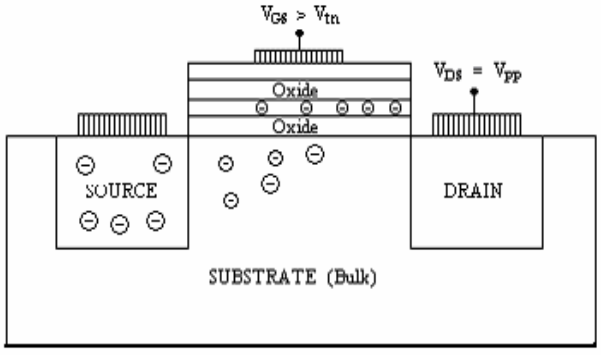
As we know dynamic memory device uses large number of capacitor so the packing density must be large. For this, there is a need to increase capacitance per area. This is achieved by forming a capacitor with trench structure.

5. Electrically alterable ROM :(EPROM and EEPROM Technology)

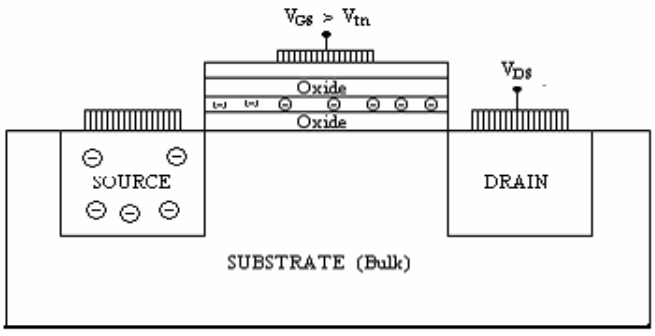
An EPROM transistor looks like a normal MOS transistor except it has a second, floating, gate (gate1 in Figure a). Applying a programming voltage V_{PP} (usually greater than 12 V) to the drain of the n- channel EPROM transistor programs the EPROM cell. A high electric field causes electrons flowing toward the drain to move so fast that they "jump" across the insulating gate oxide where they are trapped on the bottom of floating gate. We say these energetic electrons are hot and the effect is known as hot-electron injection or avalanche injection.



figure(a) structure of EPROM



When programing Voltage Vpp is applied at drain terminal the electrons gets trapped in floating gate before reaching to drain.



This figure shows the normal application voltage Vgs and Vds. the trapped electrons in floating gate raise the threshold voltage of the device and the device is in off state.

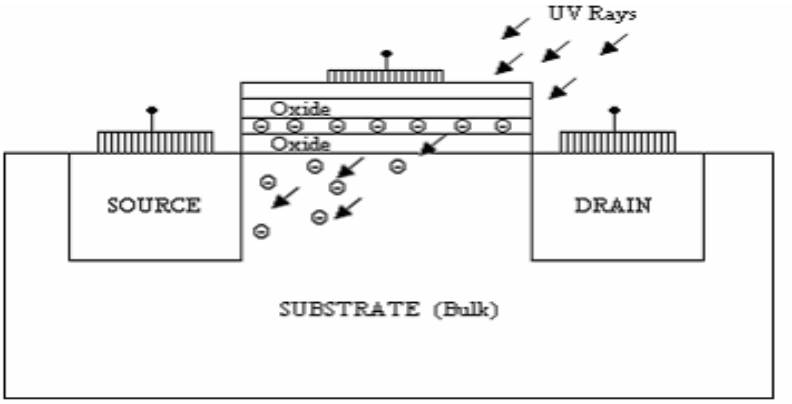


Figure 5.0 EPROM technology

EPROM technology is sometimes called floating-gate avalanche MOS (FAMOS). The figure 5.0 is described as (a) With a high (> 12 V) programming voltage, V_{PP}, applied to the drain, electrons gain enough energy to "jump" onto the floating gate (gate1). (b) Electrons stuck on gate1 raise the threshold voltage so that the transistor is always off for normal operating voltages. (c) Ultraviolet light provides enough energy for the electrons stuck on gate1 to "jump" back to the bulk, allowing the transistor to operate normally. Electrons trapped on the floating gate raise the threshold voltage of the n- channel EPROM transistor. Once programmed, an n- channel EPROM device remains off even with VDD applied to the top gate. An unprogrammed n- channel device will turn on as normal with a top-gate voltage of VDD. The programming voltage is applied either from a special programming box or by using on-chip charge pumps. Exposure to an ultraviolet (UV) lamp will erase the EPROM cell. An absorbed light quantum gives an electron enough energy to jump from the floating gate. To erase a part we place it under a UV lamp the time to erase may take up to 1 hour. The manufacturer provides a software program that checks to see if a part is erased. The packages get hot while they are being erased, so that windowed option is available with only ceramic packages, which are more expensive than plastic packages. Programming an EEPROM transistor is similar to programming an UV-erasable EPROM transistor, but the erase mechanism is different. In an EEPROM transistor, an electric field is also used to remove electrons from the floating gate of a programmed transistor. This is faster than using a UV lamp and the chip does not have to be removed from the system. If the part contains circuits to generate both program and erase voltages, it may use ISP (in system programming).

6. BICMOS Process:

Bicmos process is the fabrication technology where both the cmos device and bipolar transistors (Bipolar Junction Transistors) are fabricated in a single

substrate. First, we need to know that what the advantage is doing this. So before going to the technology process let us recall the advantages of bipolar device over Cmos and vice Versa. The advantage and disadvantage of bipolar device over Cmos is briefly summarized below.

Advantage of bipolar device:

- high current drive capability
- higher Transconductance
- higher operation frequency
- faster than cmos

The disadvantage of bipolar transistor over mos transistor are

- high power consumption
- less packing density (number of transistor/unit area)
- poor logic level (less voltage swing)

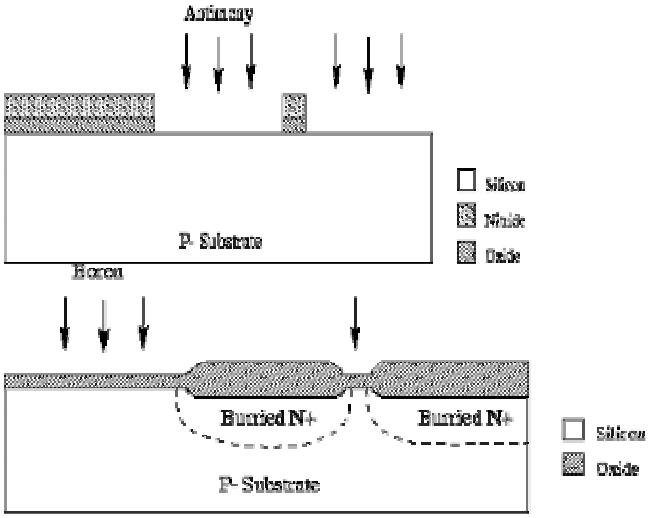
Bicmos process is the process technology where bipolar transistors and mos transistors are fabricated in a single substrate to achieve higher performance that is advantage of both mos and bipolar transistor. Usually in mixed signal design (analog and digital in a single chip) bicmos technology is preferred. For analog circuit bipolar transistor has superior performance than mos device. For digital circuit logic cmos provides excellent result.

6.1 BiCMOS Process Flow:

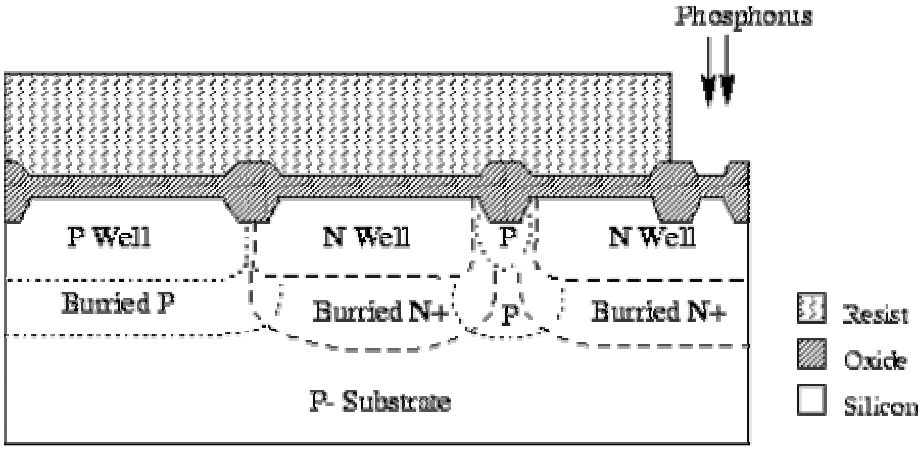
bicomos process is carried out by taking a substrate and fabrication nmos and pmos device along with the bipolar device in the same substrate.the steps below shows the fabrication of nmos,pmos and npn bipolar device in a single substrate.

the basic bicmos process flow can be summarised as follows:

- start up with a lightly-doped P-type wafer.
- form the buried N+ layer by ion implantation of antimony into the respective mask pattern.
- Form p+ buried layer by ion implantation of boron .

➢ 

- create n epitaxial layer.
- Form pwell.
- Form nwell.
- Form n base ion impalnt.
- Form collector region with n+ .

➢ 

- form p-base ion implantaion .
- form gate structure by patterning polysilicon.
- form emitter with polysilicon.

- form n+diffusion region (source,drain) in pwell.
- form p+ diffusion region (source,drain)in Nwell.
- contact cuts and mettalization.

The final diagram for bicmos process is shown in figure 6.0.

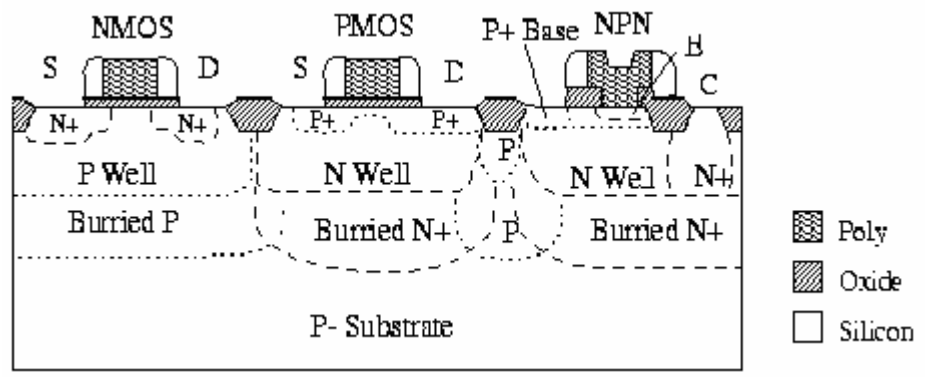


Figure 6.0: bipolar device in bicmos process

7. Latchup and prevention

Latch up in Bulk CMOS

A latchup is unintensional phenomenon which creates the low impedence path between the power supply rails (between VDD and VSS in Cmos) in electronic component due to the formation of parasitic structure (transistor,resistor) and triggering a parasitic structure, which then acts as a short circuit, disrupting proper functioning of the part and possibly even leading to its destruction due to overcurrent. the figure 7.0 shows that the nwell cmos inverter.parasitic PNP transistor is formed in nwell and parasitic transistor NPN is formed in substrate .the substrate and well also constitutes parasitic resistors. so there are two transistor and resistor in the form of parasitic components. The parasitic structure is usually an equivalent of a thyristor (or SCR), a PNPN structure which acts as a PNP and an NPN transistor stacked next to each other. During a latchup when one of the transistors is conducting, the other one begins conducting too. . They both keep each other in saturation for as long as the structure is forward-biased and some current flows through it - which usually means until a power-down.

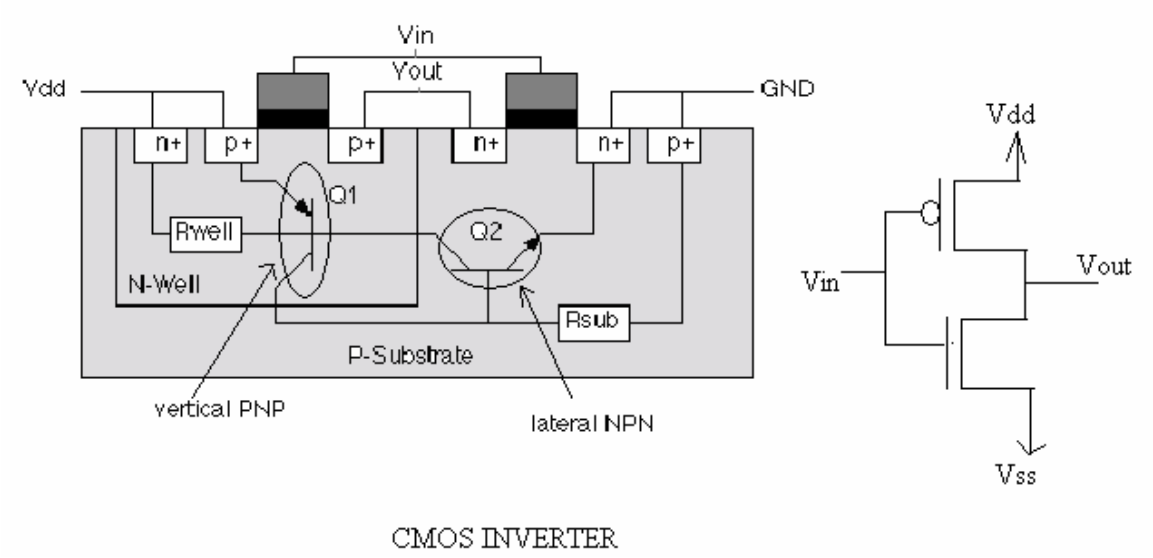
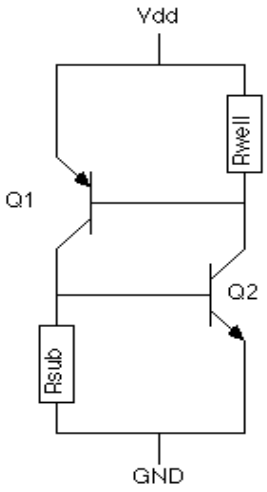
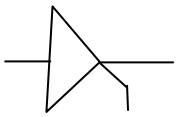


Figure 7.0 formations of parasitic components in bulk cmos.

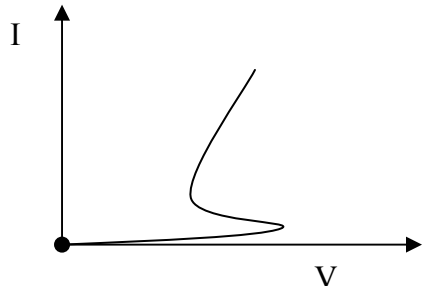
The SCR parasitic structure is formed as a part of the totem-pole PMOS and NMOS transistor pair on the output drivers of the gates.

A byproduct of the Bulk CMOS structure is a pair of parasitic bipolar transistors. The collector of each BJT is connected to the base of the other transistor in a positive feedback structure. A phenomenon called latchup can occur when (1) both BJT's conduct, creating a low resistance path between Vdd and GND and (2) the product of the gains of the two transistors in the feedback loop, b1 x b2, is greater than one. The result of latchup is at the minimum a circuit malfunction, and in the worst case, the destruction of the device. Latchup may begin when Vout drops below GND due to a noise spike or an improper circuit hookup (Vout is the base of the lateral NPN Q2). If sufficient current flows through Rsub to turn on Q2 (I Rsub > 0.7 V), this will draw current through Rwell. If the voltage drop across Rwell is high enough, Q1 will also turn on, and a self-sustaining low resistance path between the power rails is formed. If the gains are such that b1 x b2 > 1, latchup may occur. Once latchup has begun, the only way to stop it is to reduce the current below a critical level, usually by removing power from the circuit.





Formation of SCR



Current, voltage (V,I) relation of SCR
Transition from high resistance to low resistance

figure 7.1 parasitic components adds up to form a scr like structure and behaves the same.

The latchup does not have to happen between the power rails; it can happen at any place where the required parasitic structure exists. A spike of positive or negative voltage on an input or output pin of a digital chip, exceeding the rail voltage by more than a diode drop, is a common cause of latchup. Another cause is the supply voltage exceeding the absolute maximum rating, often from a transient spike in the power supply, leading to a breakdown of some internal junction. This frequently happens in circuits which use multiple supply voltages that do not come up in the proper order after a power-up, leading to voltages on data lines exceeding the input rating of parts that have not yet reached a nominal supply voltage.

Another common cause of latchup is ionizing radiation.

It is possible to design chips that are latchup-resistant, where a layer of insulating oxide (called a *trench*) surrounds both the NMOS and the PMOS transistors. This breaks the parasitic SCR structure between these transistors. Such parts are important in the cases where the proper sequencing of power and signals cannot be guaranteed (e.g., in hot swap devices). Most silicon-on-insulator devices are inherently latchup-resistant.

Another possibility for a latchup prevention is the *Latchup Protection Technology* circuit. When a latchup is detected, the LPT circuit shuts down the chip and holds it powered-down for a preset time.

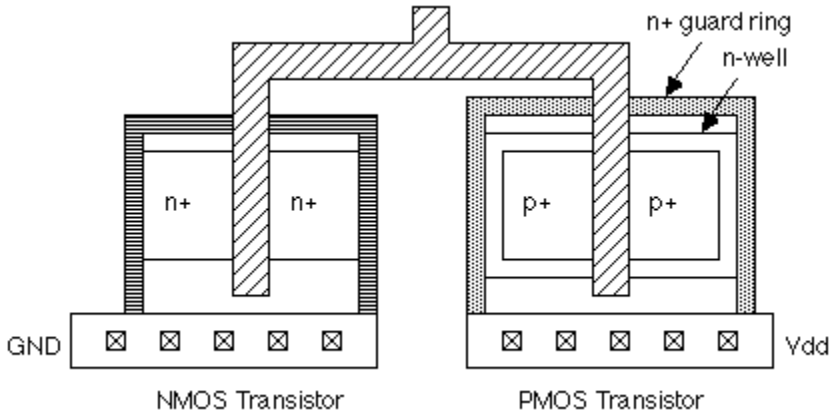
The most likely place for latchup to occur is in pad drivers, where large voltage transients and large currents are present.

Preventing Latchup:

Fab /Design Approaches

1. Reduce the gain product b1 x b1
    - move n-well and n+ source/drain farther apart increases width of the base of Q2 and reduces gain beta2 > also reduces circuit density
    - buried n+ layer in well reduces gain of Q1
2. Reduce the well and substrate resistances, producing lower voltage drops
    - higher substrate doping level reduces Rsub
    - reduce Rwell by making low resistance contact to GND

    3. Create Guard rings around p- and/or n-well, with frequent contacts to the rings, reduces the parasitic resistances.



4. Using twin tub structures (twin tub process)

5. Silicon on insulator process also is latch up free.

8.0 Layout Design Rules

The physical mask layout of any circuit to be manufactured using a particular process must conform to a set of geometric constraints or rules, which are generally called layout design rules. These rules usually specify the minimum allowable line widths for physical objects on-chip such as metal and polysilicon interconnects or diffusion areas, minimum feature dimensions, and minimum allowable separations between two such features. If a metal line width is made too small, for example, it is possible for the line to break during the fabrication process or afterwards, resulting in an open circuit. If two lines are placed too close to each other in the layout, they may form an unwanted short circuit by

merging during or after the fabrication process. The main objective of design rules is to achieve a high overall yield and reliability while using the smallest possible silicon area, for any circuit to be manufactured with a particular process.

Note that there is usually a trade-off between higher yield which is obtained through conservative geometries, and better area efficiency, which is obtained through aggressive, high- density placement of various features on the chip. The layout design rules which are specified for a particular fabrication process normally represent a reasonable optimum point in terms of yield and density. It must be emphasized, however, that the design rules do not represent strict boundaries which separate "correct" designs from "incorrect" ones. A layout which violates some of the specified design rules may still result in an operational circuit with reasonable yield, whereas another layout observing all specified design rules may result in a circuit which is not functional and/or has very low yield. To summarize, we can say, in general, that observing the layout design rules significantly increases the probability of fabricating a successful product with high yield.

The design rules are usually described in two ways :

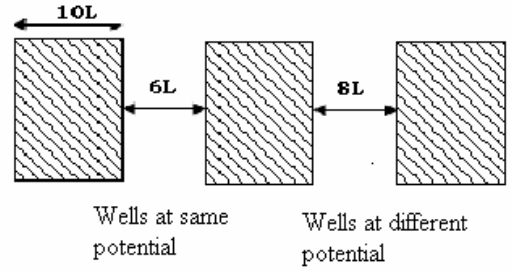
- Micron rules, in which the layout constraints such as minimum feature sizes and minimum allowable feature separations, are stated in terms of absolute dimensions in micrometers, or,
- Lambda rules, which specify the layout constraints in terms of a single parameter (L) and, thus, allow linear, proportional scaling of all geometrical constraints.

Lambda-based layout design rules were originally devised to simplify the industry- standard micron-based design rules and to allow scaling capability for various processes. It must be emphasized, however, that most of the submicron CMOS process design rules do not lend themselves to straightforward linear scaling. The use of lambda-based design rules must therefore be handled with caution in sub-micron geometries.

Lambda based Cmos Design Rules:

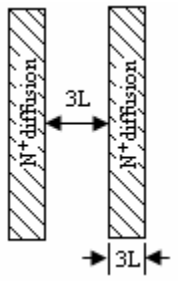
Nwell rule:

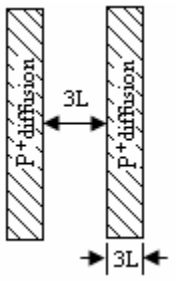
Nwell width of 10 L ,space between well if it is in same potential is 8L and if it is in different potential it is 6L.



Diffusion Rule:

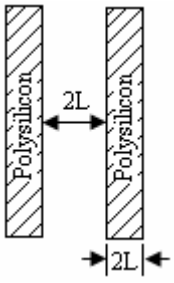
Minimum active area(diffusion)width	3 L
Minimum active area spacing	3 L

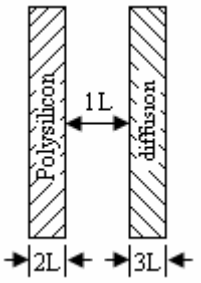


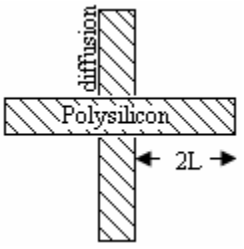


Poly Rule:

Minimum poly width	2 L
Minimum poly spacing	2 L
Minimum gate extension of poly over active	2 L
Minimum poly-active edge spacing (poly outside active area)	1 L
Minimum poly-active edge spacing (poly inside active area)	3 L

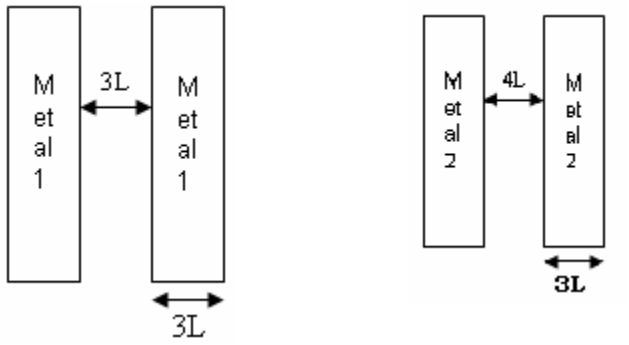






Metal Rule:

Minimum metal1 width	3 L,
Minimum metal1 spacing	3 L
Minimum metal2 width	3 L,
Minimum metal2 spacing	4 L



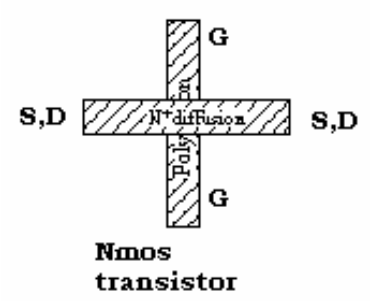
Contact cuts and via rule:

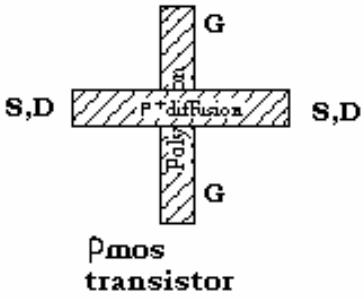
Poly contact size	2 L
Minimum poly contact spacing	2 L
Minimum poly contact to poly edge spacing	1 L
Minimum poly contact to metal edge spacing	1 L
Minimum poly contact to active edge spacing	3 L
Active contact size	2 L
Minimum active contact spacing (on the same active region)	2 L
Minimum active contact to active edge spacing	1 L
Minimum active contact to metal edge spacing	1 L
Minimum active contact to poly edge spacing	3 L
Minimum active contact spacing (on different active regions)	6 L

Physical Design:

The physical structure of the cmos device is drawn as layout diagram, which meets the design rules and helps to produce correctly working device. The layout diagram is the physical geometrical representation of the device. We will discuss the layout diagram of basic gates. The layout diagram consists of

number of rectangles which represents Metal,Polysilicon,Diffusion,Contact Cuts,Via,etc.and has to obey the design rules.
Now let us see how the transistor is formed using layout diagram.
Whenever the polysilicon crosses the diffusion transistor is formed.





Polysilicon defines for the gate region and diffusion defines for source and drain regions.remember that source and drain region cannot be physically distinguish until they are connected to power rails.for nmos if the diffusion contact is connected to Vdd that is drain ,and the other end connected to ground is source, for Cmos it is just opposite.

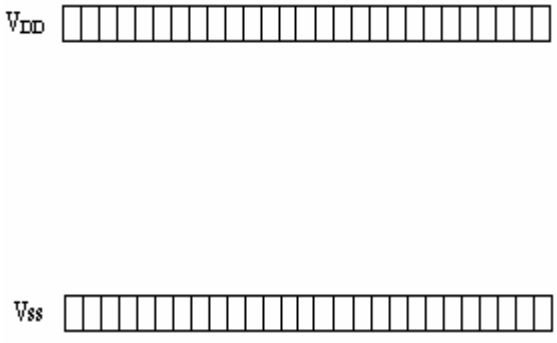
8.1 CMOS Inverter Layout Design:

In the following, the mask layout design of a CMOS inverter will be examined step-by-step. The circuit consists of one nMOS and one pMOS transistor.

First, we need to create the individual transistors according to the design rules. Assume that we attempt to design the inverter with minimum-size transistors. The width of the active area is then determined by the minimum diffusion contact size (which is necessary for source and drain connections) and the minimum separation from diffusion contact to both active area edges. The width of the polysilicon line over the active area (which is the gate of the transistor) is typically taken as the minimum poly width . Then, the overall length of the active area is simply determined by the following sum: (minimum poly width) + 2 x (minimum poly-to- contact spacing) + 2 x (minimum spacing from contact to active area edge). The pMOS transistor must be placed in an n-well region, and the minimum size of the n- well is dictated by the pMOS active area and the minimum n-well overlap over n+. The distance between the nMOS and the pMOS transistor is determined by the minimum separation between the n+ active area and the n-well . The polysilicon gates of the nMOS and the pMOS transistors are usually aligned. The final step in the mask layout is the local interconnections in metal, for the output node and for the VDD and GND contacts. Notice that in order to be biased properly, the n-well region must also have a VDD contact.

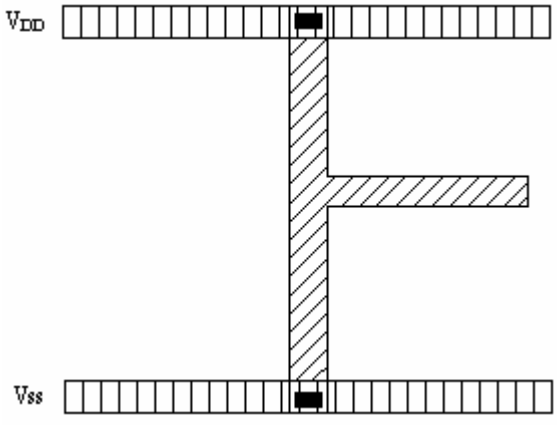
Step 1.

Draw two supply rails Vdd and Vss.



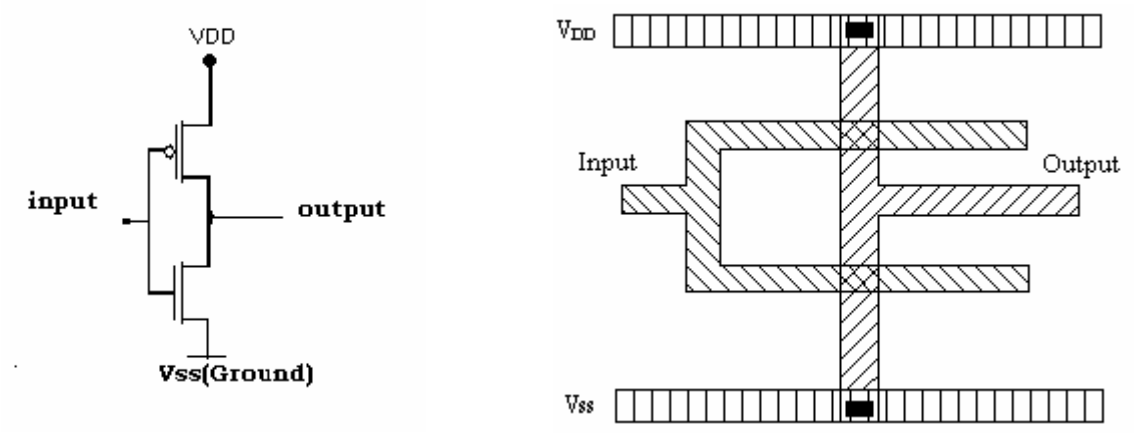
Step 2.

Draw vertical diffusion between rails.



Step3.

Now polysilicon is crossed with diffusion to form two transistor one pmos and other,nmos .



Another design style can be made if the polysilicon is placed vertically between the rails.

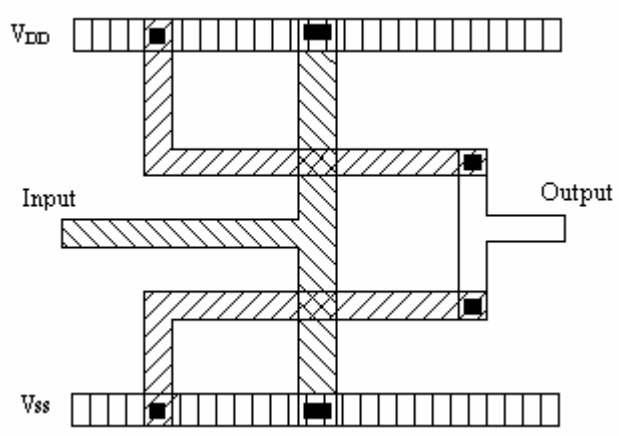


Figure-8.1: Complete mask layout of the CMOS inverter.

Layout of CMOS NAND and NOR Gates

The mask layout designs of CMOS NAND and NOR gates follow the general principles examined earlier for the CMOS inverter layout. Figure below 8.3 and 8.4 shows the sample layouts of a two- input NAND gate and a two-input NOR gate, using single-layer polysilicon and single-layer metal. Here, the p-type diffusion area for the pMOS transistors and the n-type diffusion area for the nMOS transistors are aligned in parallel to allow simple routing of the gate

signals with two parallel polysilicon lines running vertically. Also notice that the two mask layouts show a very strong symmetry, due to the fact that the NAND and the NOR gate are have a symmetrical circuit topology.

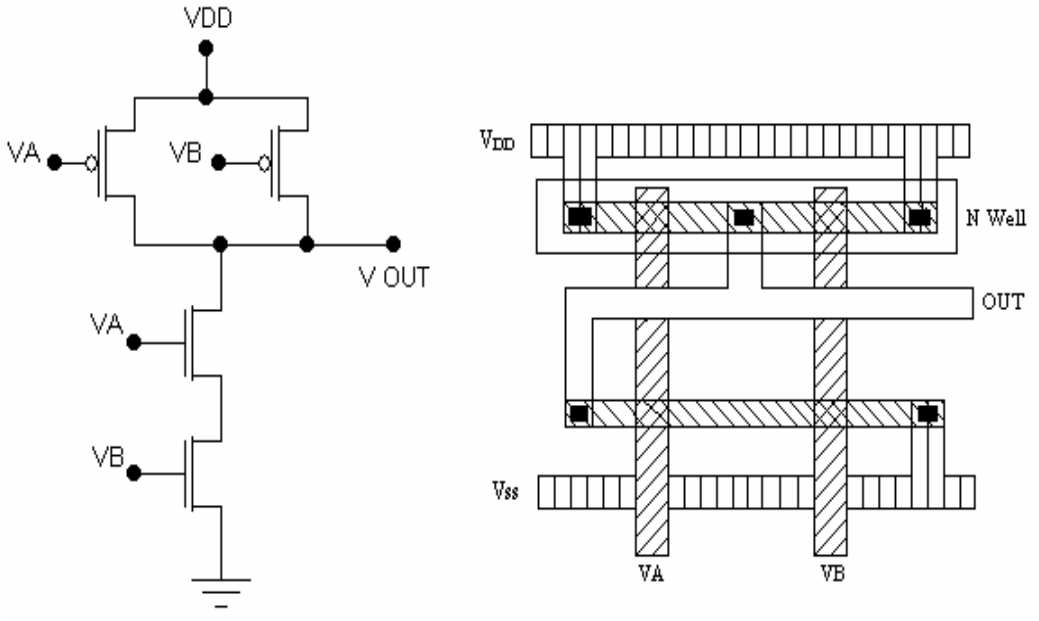


Figure-8.3: Sample layouts of a CMOS NAND gate

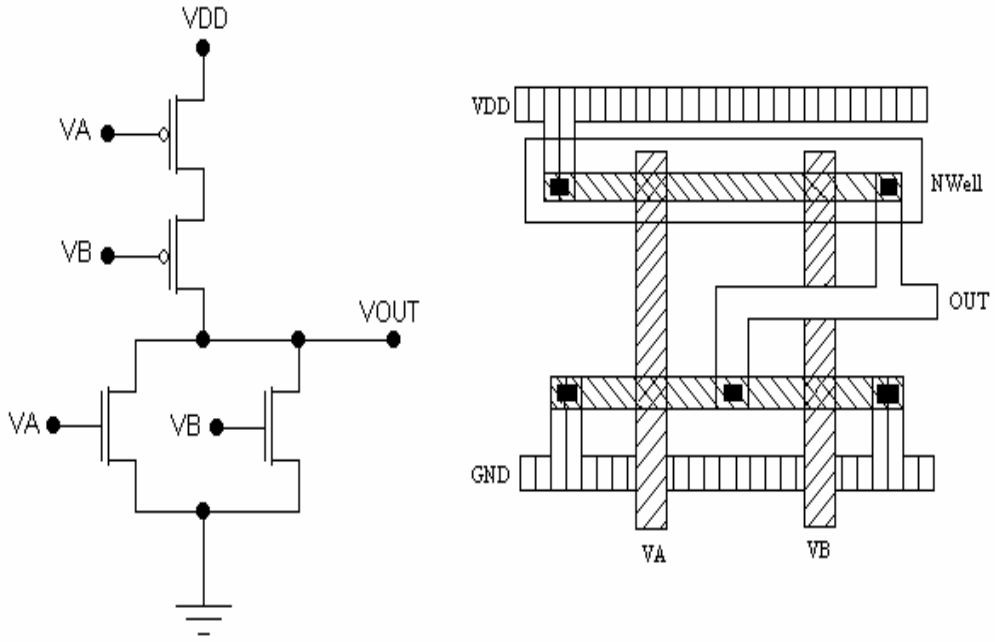


Figure-8.4: Sample layouts of a CMOS NAND gate

Computer added design tools (CAD Tools)

VLSI design is now rich with many cad tools further adding ease for the designer in several phases of the design. Some of the cad tools used in VLSI design is:

Design entry tools:

VLSI design starts with the design specification and transforming it in to some specific representation. This entry of specification can be textual or graphic form. VHDL and Verilog HDL are two commonly used textual representation. Schematic diagrams are used for graphical representation. Almost all the cad tools have editor which supports both the method for design entry. some cad tools support state diagram for sequential logic.

Functional verification (Simulation tools):

These tools are used to verify the functionality of the design. Designs are simulated using cad tools; the result is verified viewing wave form or list of the desired output. Modelsim is one of the popular simulation tools.

Synthesis tools:

This tools converts higher level of abstraction to lower level of abstraction. this tools incorporated design translation engine to translate various level of abstraction in to gate level logic. Optimization engine optimizes the logic reducing redundant logic and technology mapping is done for implementing in target device (fpga implementation) .various vendors have their own synthesis tools

Design rule checker tools:

DRC. Is used to verify if the design has violated any design rule and also may suggest the correct. To realize the physical device the design rule has to be strictly followed. The DRC checks the space and width for various structures and verifies if it has been violated with the given technology.

CHAPTER 3

- Mos transistor
- Mos transistor structure
- Nmos enhancement transistor operation
- Mos current voltage relation
- Threshold voltage
- Bodyeffect and its impact on threshold voltage
- Second order effects
- Mos models
- CMOS inverter
- Noise margin
- Pass transistor
- Cmos transmission gate
- Cmos tristate inverter
- Cmos rise and fall time
- Cmos power dissipation

CHAPTER 3

The aim of this chapter is to make readers familiar with Cmos transistor, their operation and characteristics.

3. Mos transistor:

A mos transistor (or device) has four terminals: gate, source, drain, and a fourth terminal substrate or bulk. Gate and body are conducting materials .the gate and body (bulk or substrate) are separated by a silicon dioxide which is good insulator. There are fundamentally two different type of mos transistor. They are nmos (n channel) and pmos (p channel) transistor. Cmos transistor is also mos transistor family which contains both nmos and pmos transistor .the figure (3.0) below shows n mos and pmos transistor.

Mos transistor structure and symbols:

- Four terminal device: gate, source, drain, body
- Gate – oxide – body stack looks like a capacitor
  - Gate and body are conductors (body is also called the substrate)
  - SiO_2 (oxide) is a "good" insulator (separates the gate from the body
  - Called metal–oxide–semiconductor (MOS) capacitor, even though gate is mostly made of poly-crystalline silicon (polysilicon)

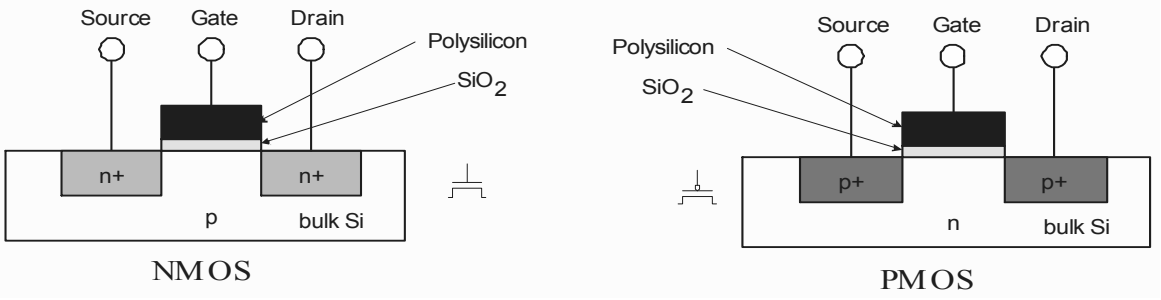


figure 3.0. nmos,pmos transistors and symbols

3.1 classification of mos transistor:

General classification of mos transistor is shown in figure 3.1 Nmos transistor can be enhancement type or depletion type. In enhancement type of transistor there is no conducting channel present initially hence it is normally called off device. Certain voltage known as threshold voltage (to be discussed later) is required to form the channel to bring transistor in to conduction. In depletion type of device initially the conducting channel is present (at the time of manufacturing ion implantation is done to form the channel) and is normally called on device.

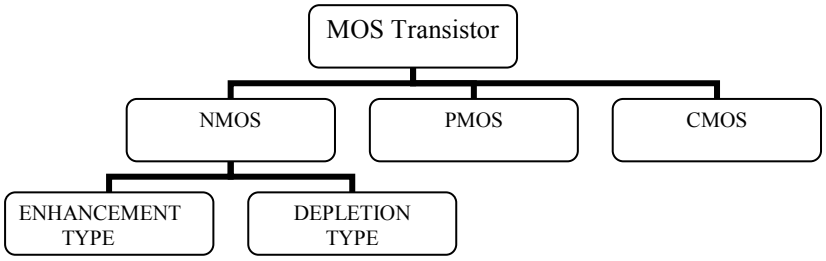


Figure 3.1 classification of mos transistor

3.2 MOS transistor structure

3.2.1 Mos transistors structure and operation

The operation of an mos transistor is considerably easier to explain than that of a bipolar transistor. An mos transistor is nothing more than a voltage-controlled switch. It has three connection points: a source, a drain, and a gate . A cross section of the metal-oxide-silicon sandwich that forms the transistor is shown in figure (3.2) below

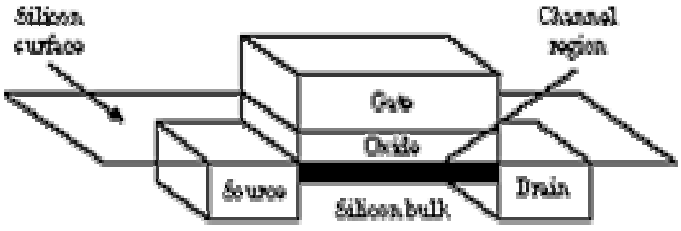


Figure (3.2) mos transistor structure

The bottommost material layer is made of silicon, an insulating oxide layer sits on top of it, and the topmost layer is the metal gate. (More modern integrated circuit processes have replaced the metal layer with a material called polycrystalline silicon, but the older “metal gate” terminology still holds.) The source and drain

regions contain silicon material with a large excess of electrons separated by the slightly positively charged bulk silicon. The source and drain are called diffusion regions because of the chemical process used to create them it means that source and drain region are formed by diffusion process. Negatively charged ions (atoms with extra valence electrons) are placed onto the silicon surface and are diffused into the surface by heating the silicon material. The materials of the source and drain are identical. Hence source and drain terminals are interchangeable in mos transistor. To distinguish source and drain terminal node remember the lower potential is source and other one is drain for nmos i.e drain is connected to Vdd and source is connected to Vss (Ground) and for pmos source is connected to Vdd and drain to Vss (Ground).

3.2.2 Operation:

The electrical behavior of the transistor is generally as follows. When a positive voltage is placed on the gate, electrons from the silicon bulk (substrate) are attracted to the transistor channel, an initially nonconducting region between the source and drain very close to the silicon surface. When the gate voltage becomes sufficiently positively charged, enough electrons are pulled into the channel from the bulk to establish a charged path between the source and the drain. Electrons flow across the transistor channel, and the voltage-controlled switch is conducting. If a 0 or very small voltage is placed on the gate, no electrons (or at least very few) are attracted to the channel. The source and drain are disconnected, no current flows across the channel, and the switch is not conducting. Because they are made from materials with different affinities for electrons, the two transistor types behave quite differently. The transistor operation described above is actually for the nmos transistor. The bulk is positively charged, while the diffusion is negatively charged. The transistor switch is “closed” (conducting) when a logic 1 is placed on its gate and “open” (nonconducting) when the gate is connected to a logic 0. The pmos transistor is complementary. The diffusion regions are positively charged and the silicon bulk is negatively charged. A pmos transistor behaves in a complementary way: it is “closed” (conducting) when a logic 0 is placed on the gate and is “open” (nonconducting) when a logic 1 is placed there.
The symbols for the two different kinds of transistors make it easy to remember how they operate. An nmos transistor conducts when the gate voltage is asserted in positive logic. The pmos transistor conducts when the gate is asserted in negative logic. This is why there is a polarity bubble on the gate of the pmos transistor’s symbol. figure (3.3) below shows the nmos and pmos switching condition. Nmos conducts(on) when gate input is ‘1’ and pmos conducts when gate input is’0’.the dim portion in figure shows that transistor is off.

Figure (3.3 a) is for nmos and (b) for pmos transistor.

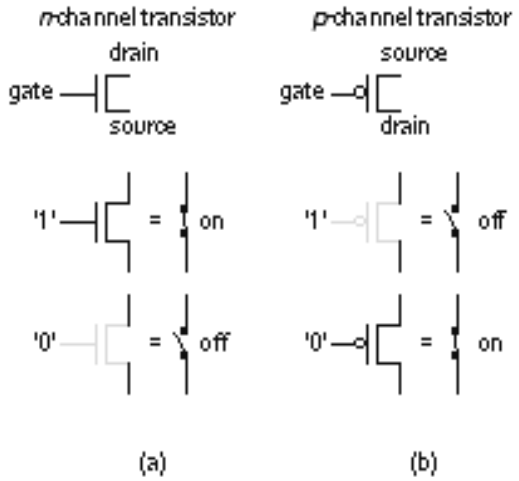


Figure (3.3) switching of nmos and pmos transistor.

3.3 Nmos Enhancement type of transistor modes of operation:

As we know that enhancement type of device (transistor) does not have conducting channel at the beginning so to make the device functional we have to create a channel between source and drain. The physical structure of nmos transistor is shown in the figure (3.4). Since there are n+ and p region in source drain, and substrate the formation of back to back parasitic diode prevents any current to flow, while the device is in neutral form (no external voltages applied).

Now if we look at the transistor gate and below gate region we can conclude that it forms a structure similar to capacitor. Gate material (aluminium, polysilicon) is one conductor and bulk (substrate) is other conductor. In-between these conductors the insulator in the form of silicon dioxide is present. This structure is also known as mos capacitor. As per the fundamental of the capacitor when one conductor is positively charged negative charge is induced in other conductor. Now let us see what happens to this mos capacitor when different gate voltage is applied. The application of different gate voltage results in 3 modes of operation. They are accumulation, depletion and inversion. This is the basic fundamental working principle of mos transistor.The voltage at gate terminal is required for the transistor to operate. Depending upon the voltage applied at the gate terminal the transistor enters in to these 3 modes. Refer figure (3.4).

Accumulation mode:

When Vgs is negative (Vgs<<Vt) positive charges are attracted near the source and drain below the gate .this is known as accumulation mode since more no

positive charges are induced due to the negative voltage applied at the gate terminal. This prevents the formation of channel.

Depletion mode:

When the gate terminal voltage is small positive voltage which is equal to threshold voltage vt (threshold voltage is the minimum voltage which is required for the transistor to conduct) the positive charge at the top of the substrate near source and drain gets repelled and electrons tries to come up but due to insufficient voltage at the gate terminal it cannot completely move up hence a gap comes in to existence known as depletion layer where no charge carriers are present.

Inversion mode:

When the gate voltage is sufficiently positive (vgs>vt) the negative charge are attracted and moves up ,positive charge are repelled further and goes down hence a ntype layer is created which establishes the conducting path between source and drain and referred as channel. Since the channel is now inverted originally from p type to n type it is known as inversion. Once the channel is created the path is established for the electrons in source terminal to reach drain terminal however to pull the electrons from the source to drain terminal additional force is required this additional force is derived from the other terminal voltage vds (drain to source voltage).now depending upon the vds the transistor enters in different region of operation .cutoff,nonsaturation(linear) and saturation region.

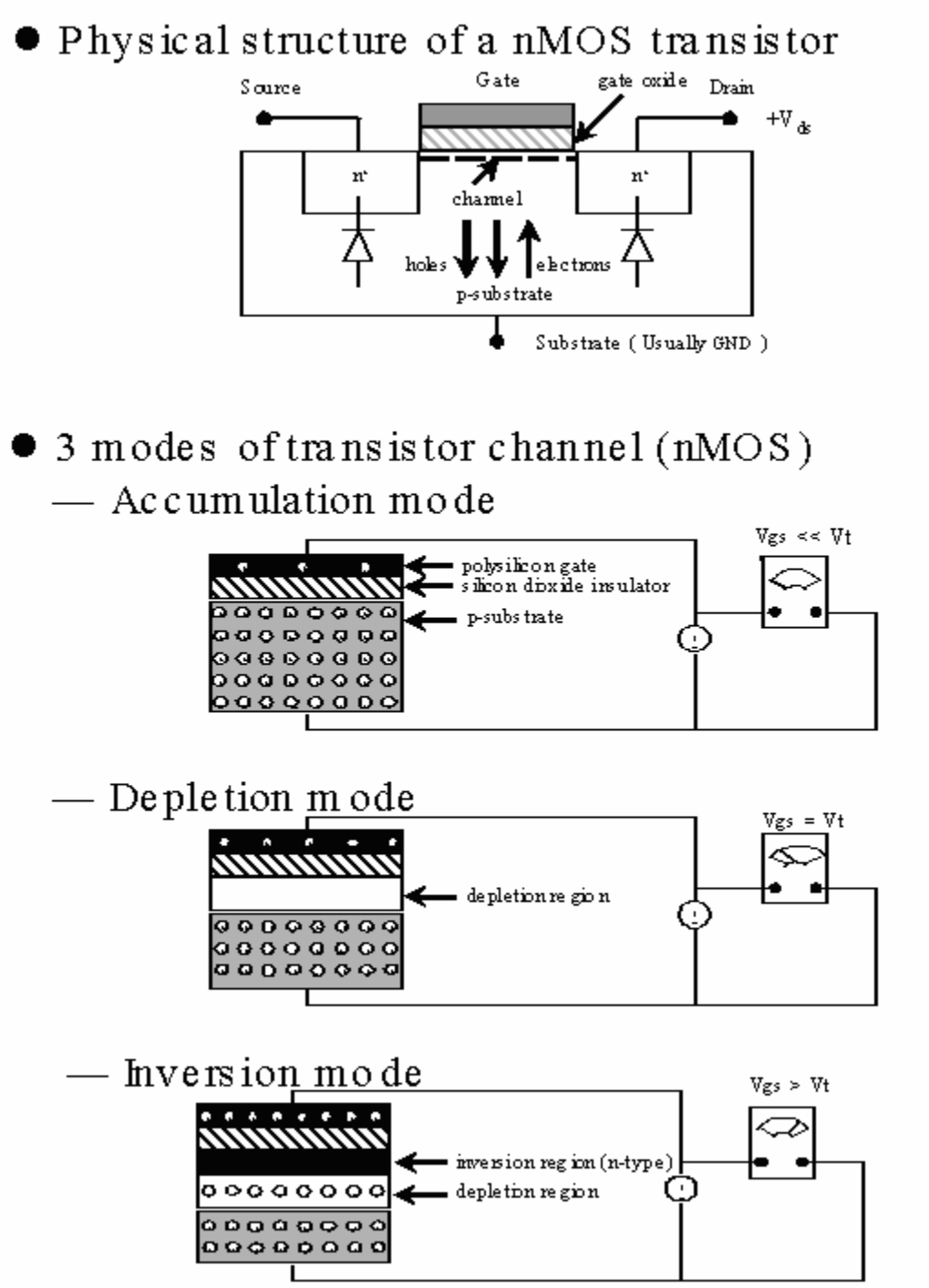


Figure (3.4) effect of Vgs in Nmos Transistor channel area (MOS Capacitor)

As we have discussed earlier that enhancement type of transistor can be operated in various region depending upon the applied voltage between drain and source terminal of course Vgs also. These regions are as follows.

Cutoff region:

when the gate to source voltage Vgs is less than threshold voltage Vt no current flows from drain to source (no flow of electron from source to drain as the conducting path is not established) .there is no effect of Vds when Vgs<<Vt and this region is known as cutoff region.(when Vgs is small positive but less than Vt a very small amount of current flows due to a weak inversion and known as subthreshold current).Vt for nmos enhancement type of transistor is approximately 0.2Vdd .collectively subthreshold region is also termed in cutoff region. So the current voltage relation at cutoff region is

When gate to source voltage is less than Vt i.e Vgs<Vt

Ids = 0

Nonsaturation (linear, triode) region:

This region is defined as when Vgs > Vt and Vds <Vgs – Vt. in this region the current Ids varies almost linearly with the Vds.

The relation between current and voltage in this region is

- **$Ids=\beta((vgs-vt).vds-vds^2/2)$**

Since in this region Vds is small so the quadratic terms can be neglecting then the overall effect will be linear.

In this region the gate voltage will drop along with the channel length therefore there is less gate voltage at the drain end compare to the source end. that

Saturation region:

When Vds becomes more positive i.e Vds>vgs-vt the more drop takes place along the length of the channel and due to insufficient gate voltage near the drain end the formation of inversion layer is seized ,this condition is known as Pinch Off, how ever the current completes its path as diffusion process after the pinch off occurs.

Saturation region starts when Vds is greater than or equal to (Vgs-Vt).for our convenience let us assume Vds=Vgs-Vt ,then substituting Vds=Vgs-Vt in above equation we get

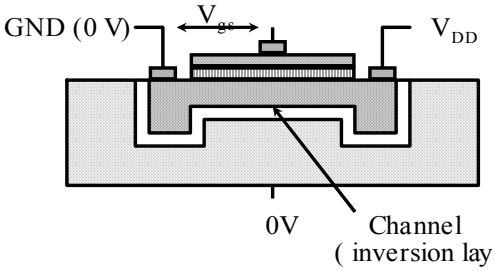
- **$Ids=\beta/2(Vgs-Vt)^2$**

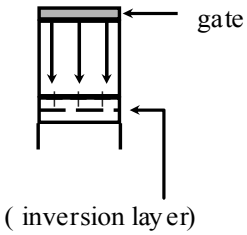
In this region current Ids is independent of Vds and is constant. i.e increase in Vds doesn't increase the current Ids rather it is constant .refer the V I characteristics curve.(due to channel length modulation a small variation is possible we will neglect this until we see second order effects later in this text.)

The figure.(3.5) and (3.6) shows the three region and the relation of current and voltage in various regions.

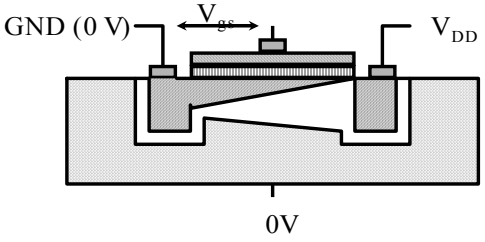
The operation for pmos transistors is similar to nmos except that the voltage polarity has to be altered. The gate to source voltage should be negative unlike

1. $V_{gs} > V_t$, $V_{ds} = 0$

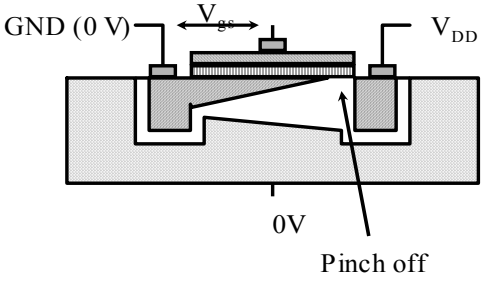




2. $V_{gs} > V_t$, $V_{gs} - V_{ds} = V_{gd} > V_t$



3. $V_{gs} > V_t$, $V_{gs} - V_{ds} = V_{gd} < V_t$



- I-V characteristic of nMOS

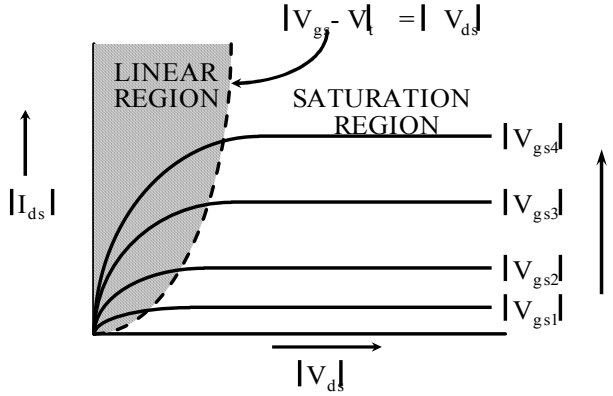


Figure (3.5) nmos various operation regions

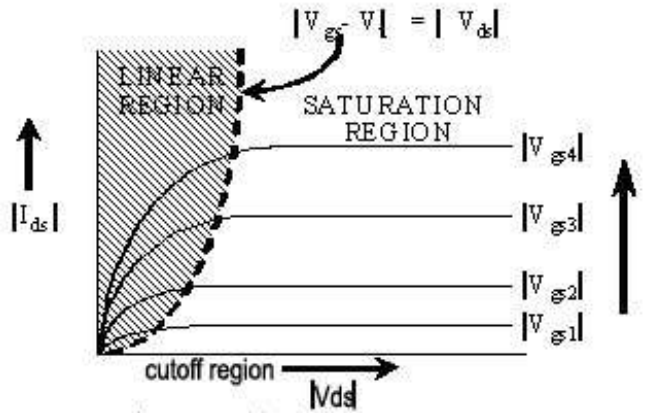
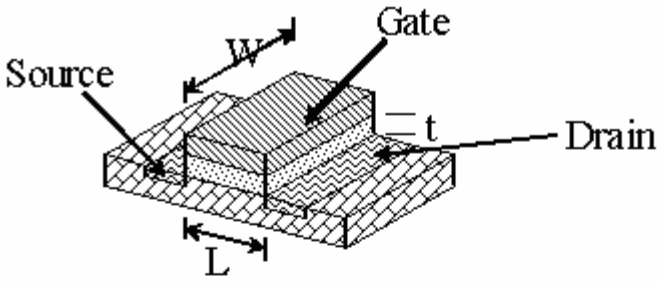


Figure (3.6) nmos enhancement transistor V, I characteristics curve

3.5 Mos current volage equations (Mos device equations)

Let us derive the relation of current and voltage for nmos enhancement type of transistor. The derivation is self explanatory.



I_{DS} VS. V_{DS} Relationship

$$I_{DS} = \frac{\text{charge induced in channel}(Q_c)}{\text{electron transit time}(\tau)}$$

$$\text{Transit time } \tau_{DS} = \frac{\text{length of channel}(L)}{\text{velocity}(\upsilon)}$$

$$\text{Velocity } \upsilon = \mu E_{DS}$$

where μ is electron or hole mobility

E_{DS} is electric field

$$E_{DS} = \frac{V_{DS}}{L}$$

$$\tau_{DS} = \frac{L^2}{\mu V_{DS}}$$

In Nonsaturation region the voltage along with the length of the channel varies therefore the average electric field between the gate and channel is thus ((Vgs-Vt)-Vds/2)/ t. then the expression for charge can be written as QC=Eg.ε W.L
Where ε is permittivity. W and L are width and length of the gate.

Eg: Average electric field (gate to channel)

ε: permittivity of insulator between gate and channe

$$Eg = \frac{(Vg-Vt)-V_{DS}/2}{t}$$; Assume average channel voltage $V_{DS}/2$

t: oxide thickness

$$Qc = \frac{\varepsilon WL}{t}\left[(Vgs-Vt)-\frac{V_{DS}}{2}\right]$$

$$I_{DS} = \frac{\mu\varepsilon}{t}\frac{W}{L}\left[(Vgs-Vt)-\frac{V_{DS}}{2}\right]V_{DS}$$

$$= \mu Cox\frac{W}{L}\left[(Vgs-Vt)-\frac{V_{DS}}{2}\right]V_{DS}$$

$$Cox: \text{gate capacitance/unit area} = \frac{\varepsilon}{t}$$

$$\text{Total gate capacitance } C_g = CoxWL$$

$$\text{Let } K=\mu Cox \text{ or } \beta = K\frac{W}{L} = \frac{\mu CoxW}{L}$$

$$\Rightarrow I_{DS} = K\frac{W}{L}\left[(Vgs-Vt)V_{DS} - \frac{V_{DS}^2}{2}\right]$$

$$= \beta\left[(Vgs-Vt)V_{DS} - \frac{V_{DS}^2}{2}\right]$$

Hence in non saturation region the relation of Ids and Vds, Vgs is

- **$Ids=\beta((Vgs-Vt).Vds-Vds^2/2)$**

In non saturation If Vds is small voltage than the quadratic term $Vds^2/2$can be ignored than the current has linear relation with voltage.

Saturation region:

We know the saturation starts when Vds >= Vgs-Vt

for simplicity let us take Vds=(Vgs-Vt) substituting this value of Vds in the relation

Ids=β((vgs-vt).vds-vds^2/2)

And solving to find Ids we get

- **Ids=β/2(vgs-vt)^2.**

Summary Of the mos equation:

$$I_{ds} = \begin{cases} 0 & V_{gs} < V_t \quad \text{Cut off region} \\ \beta\left(V_{gs} - V_t - \frac{V_{ds}}{2}\right)V_{ds} & V_{ds} < V_{dsat} \quad \text{Non saturation region} \\ \frac{\beta}{2}\left(V_{gs} - V_t\right)^2 & V_{ds} > V_{dsat} \quad \text{Saturation region} \end{cases}$$

3.6 Threshold voltage:Vt.

Threshold voltage is defined as the minimum voltage that is required for the mos transistor to conduct. (The origin of threshold voltage is due to the charge trapped in silicon substrate interface below the gate region. to neutralize this charge carrier additional voltage is required). Threshold voltage depends on number of parameters like

- Gate material (conductor)
- Gate insulator material(gate oxide)
- Insulator thickness(oxide thickness)
- Impurities t silicon substrate interface
- Voltage at source and substrate

Combining all the above parameter the threshold voltage equation can be written as:

Vt = Vtmos + Vfb

Where Vtmos is ideal threshold voltage where there is no work function difference between gate and substrate materials. And Vfb is flat band voltage.

Vtmos=2Φ_b +Qb/Cox where Φ_b is bulk potential which accounts for doping concentration of the substrate. Φ_b=KT/Q ln(Na/Ni).where k is Boltzman constant(1.38x10^{-23} joule/^{0}k).T is the temperature, and q is the electronic charge.

For room temperature around 300^{o}k the term KT/Q comes around 26 mv. Na and Ni are acceptor concentration and intrinsic doping concentration.

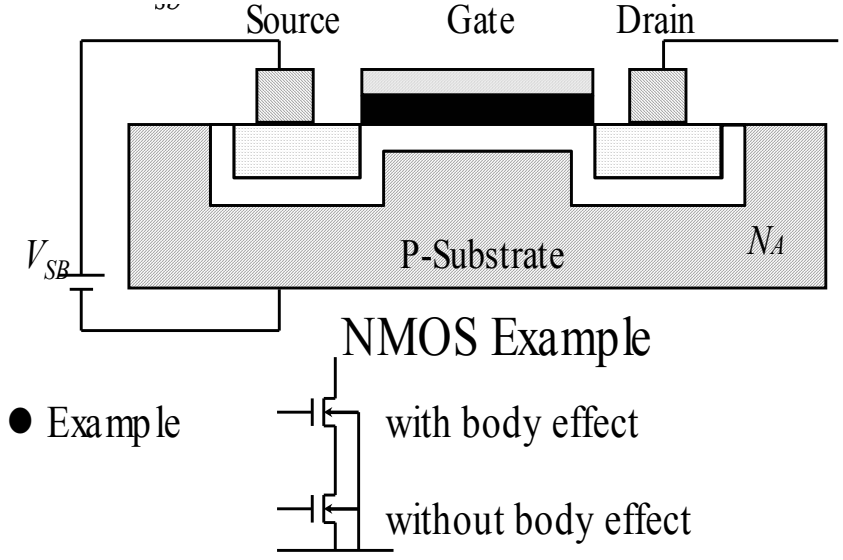
Qb is bulk charge and is equal to = $\sqrt{}$ (2 ε_{si}QNa 2$\Phi_{b.}$)

The flat band voltage is given by Vfb=Φms-Qfc/Cox, where Φms is the work function difference between gate material and substrate (Φm- Φs). Qfc is the surface charge accumulated due to the imperfection in silicon - oxide interface. Cox is oxide capacitance. Combining all this parameters we can write the final threshold voltage equation as

- Vt = Vtmos + Vfb
- 2Φ_b +Qb/Cox+ Φms-Qfc/Cox

2.7 Body Effect and its impact on threshold voltage:

Body effect is the term which refers to the increase in threshold voltage due to increase in source to substrate voltage (Vsb) .As we have noticed that the entire device in mos is made on a common substrate. Normally the source and substrate are connected so Vsb=0.how ever there may arise the situation that the device have to be connected in series for example nmos transistor are connected in series to form nand gate in this situation Vsb1=0v but Vsb2 ≠ 0v.so the bottom transistor do not have body effect since both source and substrate are biased to ground(common point).However as we move up to top transistor than the source and substrate are biased at different point hence the Vsb2 may not be equal to 0.this may cause the depletion region to increase further as shown in figure and the number of charge carriers trapped in this region may increase and the overall effect is increase in threshold voltage to hold the charge neutrality.



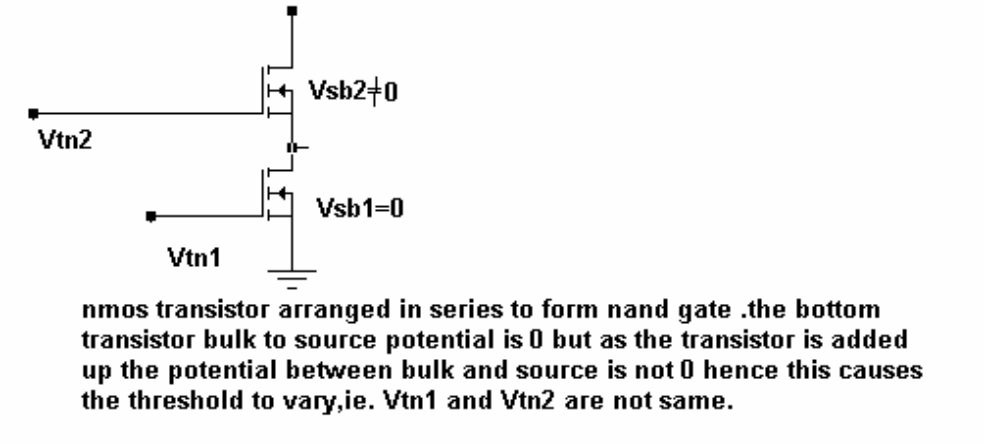


Figure 3.7 body effect

The change in threshold voltage due to body effect is given as

- $\Delta Vt = \gamma (Vsb)^{1/2}$

Where γ is constant and depends on substrate doping concentration, thickness of oxide. typical values for γ is in the range 0.4 to 1.2.

Now we have to modify the equation for threshold voltage considering the body effect (body effect is one of the Second Order effect to be discuss next in this text).the equations related to body effect is as follows.

$$V_t = V_{fb} + 2\phi_b + \frac{\sqrt{2\varepsilon_{si} q N_A (2\phi_b + |V_{sb}|)}}{C_{ox}}$$

$$V_t = V_{t0} + \gamma[\sqrt{(2\phi_b + |V_{sb}|)} - \sqrt{2\phi_b}]$$

$$\gamma = \frac{t_{ox}}{\varepsilon_{ox}} \sqrt{2\varepsilon_{si} q N_A} = \frac{1}{C_{ox}} \sqrt{2\varepsilon_{si} q N_A}$$

V_{fb} flat-band voltage ε_{ox}: permitivity of gate oxide

ϕ_b Fermi potential

V_{sb} substrate bias

V_{t0} V_t for V_{SB}=0

N_A substrate doping concentration

3.8 Second order effect:

while deriving the mos various DC equations (current voltage relation in various region) we have over simplified and made assumptions. However the accuracy lies in fact considering other effects that may come in to existence for several reasons. These second order effects has several side effects for the general equations derived earlier, as we have already seen the effect of body effect in threshold voltage and modified the equation. Some of the second order effects are

- **Body effect and threshold voltage**

  We have already discussed about the body effect and its impact on threshold voltage, so if you have any doubt still on your mind kindly refer section 3.6 and 3.7.

- **subthreshold current**

  we generally say that when Vgs < Vt there is no flow of current (Ids) however what about if Vgs>0 but less than Vt. when 0<Vgs<Vt ,it indicates a small positive voltage but which is less than threshold voltage there for it can be said it is insufficient to establish a complete inversion layer however a weak inversion layer is formed and a very small current flows between source and drain. This current is known as subthreshold current. The point to remember here is that even though the current Ids is very small it has an exponential

relation with the voltage Vds, Vgs. this current have both advantage as well as disadvantage. It can be utilized for low power circuits in the mean time it may affect the operation in circuits like dynamic charge storage nodes.

- **Channel length modulation**

It is observed that the length of the channel is not constant; while the device is operating in saturation the actual channel length is decreased. simplified equations that describe the behavior of an MOS device assume that carrier mobility is constant ,and do not take into account the variation in channel length in drain-to –source voltage ,V_{ds}'. for long channel length ,the influence of channel variation can be ignored. However for short channel device (as devices are scaled down), this variation should be taken into account.

When an MOS device is in saturation, the effective channel length actually is decreased such that

$$L_{Eff} = L - L_{short}$$

Where

$$L_{Short} = \sqrt{2}\,(\varepsilon_{si} / QN_A\ (V_{ds} \text{ ---}(V_{gs} - V_t))$$

The reduction in channel length increases the (W/L) ratio, there by increasing β as the drain voltage increase. Thus rather than appearing as a constant current source with infinite output impedance, the MOS device has finite output impedance. An approximation that takes this behavior into account 11 is represented by the following equation:

$$I_{ds} = k\,W / 2L\,(Vgs - V_t)^2(1+\lambda V_{ds})$$

Where k is the process gain factor με /tox and λ is an empirical channel length modulation factor having a value in the range 0.02v-1 to 0.005v-1. in the SPICE level 1 model 1λ is the parameter LAMBDA.

- **Drain punch through**

when the drain is at high enough potential with respect to the source the depletion region around the drain may get extended towards the source end and this may cause the current to flow from the source to drain irrespective of gate voltage. This condition is known as drain punch through; this effect can be utilized in I/O protection circuit to limit the voltage across internal nodes.

- **Mobility variation**

Mobility is related to the velocity with which electron or holes travels in applied electric field. The mobility, μ, describes the ease with which carriers drift in the substrate material .it is defined by

μ(Mobility) =average carrier drift velocity (V) /Electric Field (E)

If the velocity, V, is given in cm/sec, and the electric field, E, in V/cm, the mobility has the dimension cm^2/v-sec. The mobility may vary in a number of ways. Primarily, mobility varies according to the type of charge carrier. Electrons (negative –charge carriers) in silicon have a much higher mobility than holes (positive –charge carriers), (approximately electron has 2.5 times higher mobility than holes.) Resulting in n-devices having higher current- producing capability than the corresponding p-devices. Mobility decreases with increasing doping – concentration and increasing temperature. The temperature variation becomes less pronounced as the doping density increases.

- **impact ionization (Hot electrons)**

As the length of the gate of an MOS transistor is reduced, the electric field at the drain of a transistor in saturation increases (for a fixed drain voltage). For submicron gate length, the field can become so high that electrons are imparted with enough energy to become what is termed “hot.” These hot electrons impact the drain, dislodging holes that are then swept toward the negatively charged substrate and appear as a substrate current. This effect is known as impact ionization. Moreover, the electrons can penetrate the gate oxide causing a gate current. Eventually this can lead to degradation of the CMOS device parameters (threshold voltage, subthreshold current, and transconductance), which in turn can lead to the failure of circuits. While the substrate current may be used in a positive manner to estimate the severity of the hot-electron effect, it can lead to poor refresh times in dynamic memories , noise in mixed signal systems, and possibly latchup. Hot holes do not normally present a problem because of their lower mobility.

- **Fowler –Nordheim Tunneling**

When the gate oxide is very thin, a current can flow from gate to source or drain by electron tunneling through the gate oxide. This current is proportional to the area of the transistor as follows:

- $I_{FN} = C_1 WLE_{OX}^2 e^{-E_o / E_{ox}}$

Where $E_{OX} \approx V_{gs} / t_{OX}$ is the electric field across the gate oxide and

E_0 and C_1 are constants.

This effect limits the thickness of the gate oxide as processes are scaled. However, it is of great use in the electrically alterable programmable logic devices.

3.9 Mos Models:

The mos device can be modeled as DC model and Small signal AC Model. spice parameters can be used to model the device. The mos structure is shown in the figure below along with the parasitic capacitors from various regions. In small signal analysis we try to find out channel resistance and transconductance when the device is in nonsaturation and in saturation region.

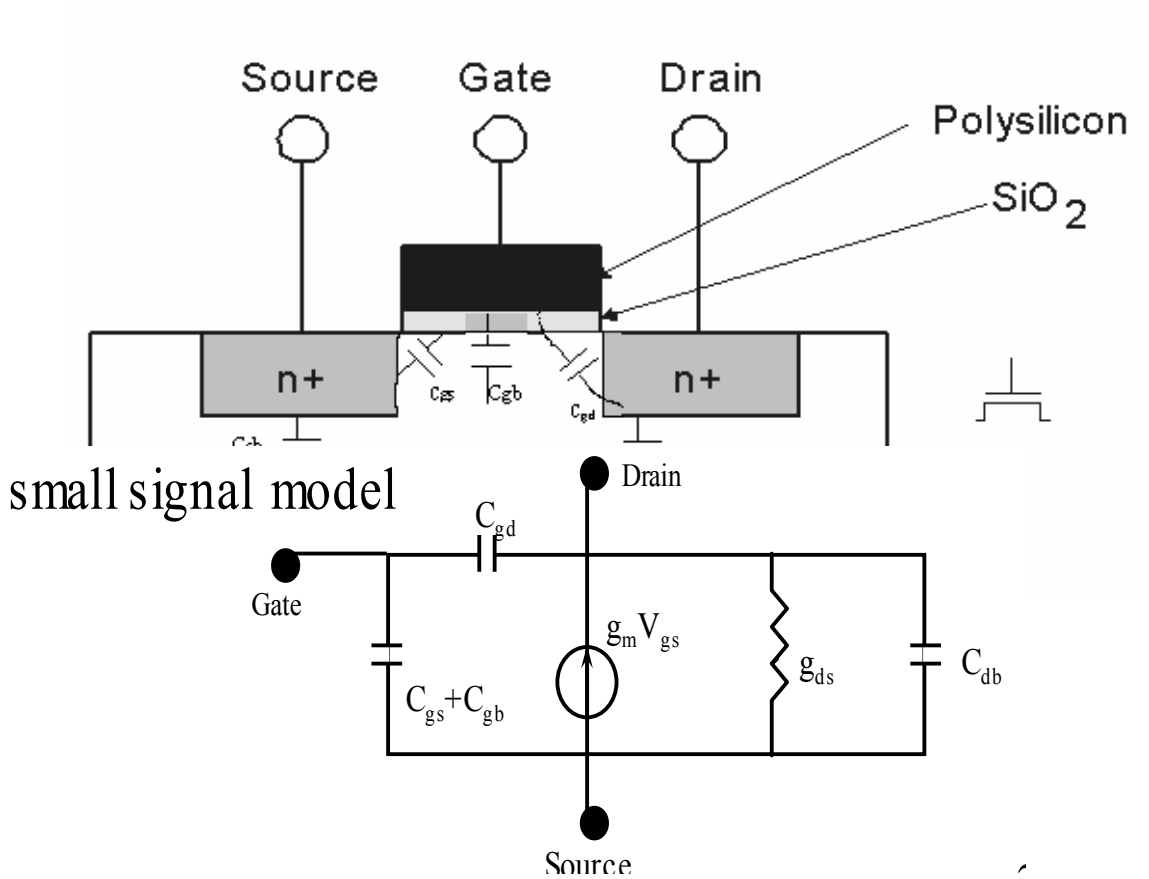


Figure 3.8 Mos device parasitic capacitance and small signal Model.

To find the channel resistance in nonsaturation (Linear) region let us look at the basic equation

- **Ids=β((vgs-vt).vds-vds^2/2)**

We know that output conductance is ∂ ids/ ∂ Vds.differentiating the above equation with respect to Vds we get output conductance as Gds=∂ ids/ ∂ Vds = β (vgs-vt).

Therefore channel resistance

- **$R_{clinear=}$ 1/β(vgs-vt).**

For saturation region we take the equation Ids=β/2(vgs-vt)^2.,and if we try to differentiate this equation with respect to Vds the result is 0 that means there is no change in output current with change in output voltage.hence it behaves as a constant current source in saturation region.

To determine the transconductance when the device is in linear region (Nonsaturation)

We know that transconductance g_m is defined as output current divided by input voltage.

Hence $\mathbf{g_m = \partial\ Ids/\ \partial\ Vgs}$.so differentiating the equation **Ids=β((vgs-vt).vds-vds^2/2)** with respect to Vgs we get

- $\mathbf{g_{m(linear)} = \partial\ Ids/\ \partial\ Vgs = \beta.vds}$

Similarly to find the transconductance in saturation we differentiate the equation **Ids=β/2(vgs-vt)^2** with respect to Vgs.

- $\mathbf{g_{m(sat)} = \partial\ Ids/\ \partial\ Vgs = \beta.(vgs-vt).}$

3.10 Complimentary Mos (Cmos) Inverter:

Cmos inverter is formed with interconnecting the gate terminals of both nmos and pmos transistor which are connected in series as shown in figure together .the input is applied to this common gate terminal.the output is taken from the common point drain of both the transistor.when input Vin is '0' pmos transistor conducts and nmos transistor remains off therefore Vout ="Vdd"(logic 1).simillarly when Vin='1' then pmos is off and nmos transistor is on therefore the output is pulldown to Vss or Low logic.

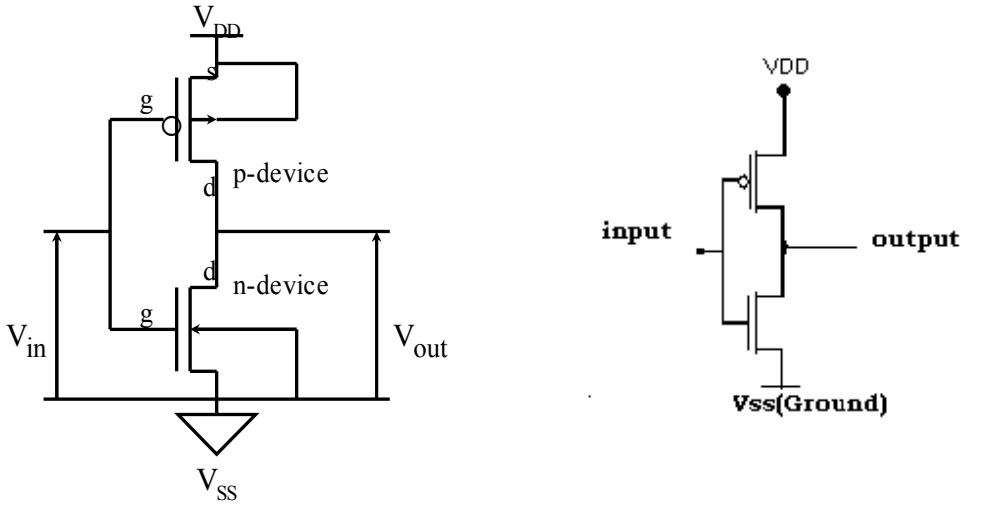


Figure 3.9 cmos inverter with substrate connection,and without substrate connection.

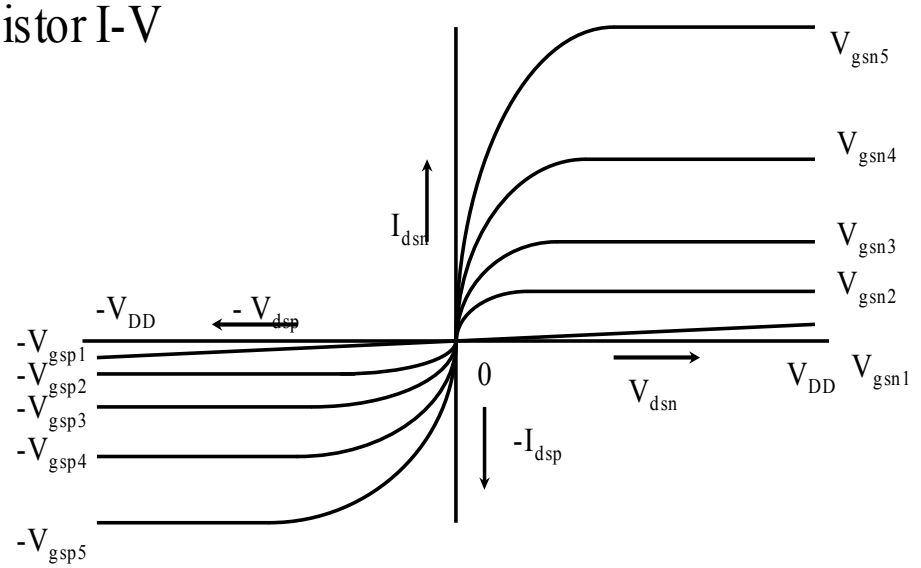
- Relations between voltages for the three regions of operation of a CMOS inverter

	CUTOFF	NONSATURATED	SATURATED
p-device	$V_{gsp} > V_{tp}$ $V_{in} > V_{tp} + V_{DD}$	$V_{gsp} < V_{tp}$ $V_{in} < V_{tp} + V_{DD}$ $V_{dsp} > V_{gsp} - V_{tp}$ $V_{out} > V_{in} - V_{tp}$	$V_{gsp} < V_{tp}$ $V_{in} < V_{tp} + V_{DD}$ $V_{dsp} < V_{gsp} - V_{tp}$ $V_{out} < V_{in} - V_{tp}$
n-device	$V_{gsn} < V_{tn}$ $V_{in} < V_{tn}$	$V_{gsn} > V_{tn}$ $V_{in} > V_{tn}$ $V_{dsn} < V_{gs} - V_{tn}$ $V_{out} < V_{in} - V_{tn}$	$V_{gsn} > V_{tn}$ $V_{in} > V_{tn}$ $V_{dsn} > V_{gs} - V_{tn}$ $V_{out} > V_{in} - V_{tn}$

Graphical representation of transfer characteristics (input/output relation) of Cmos Inverter:

To determine the transfer characteristics of the Cmos inverter we first look at the individual current voltage relation ship of the pmos and nmos transistor. In the figure below RHS from the current axis is for nmos and LHS is for pmos.

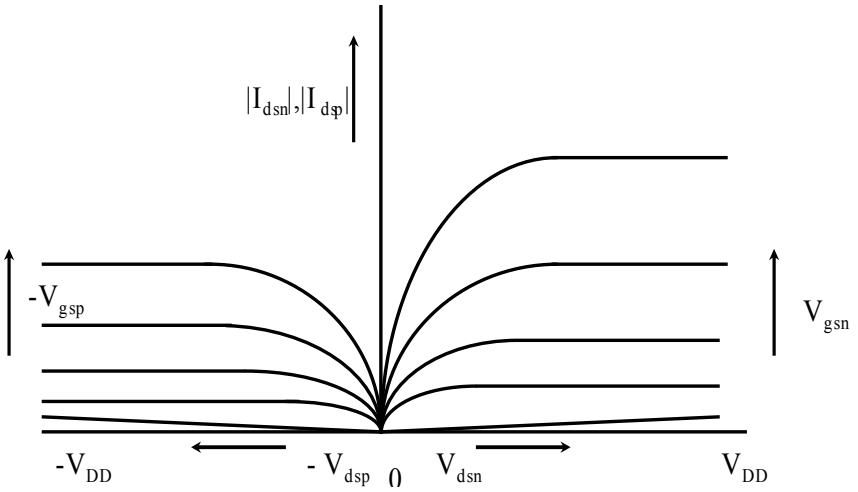
- transistor I-V



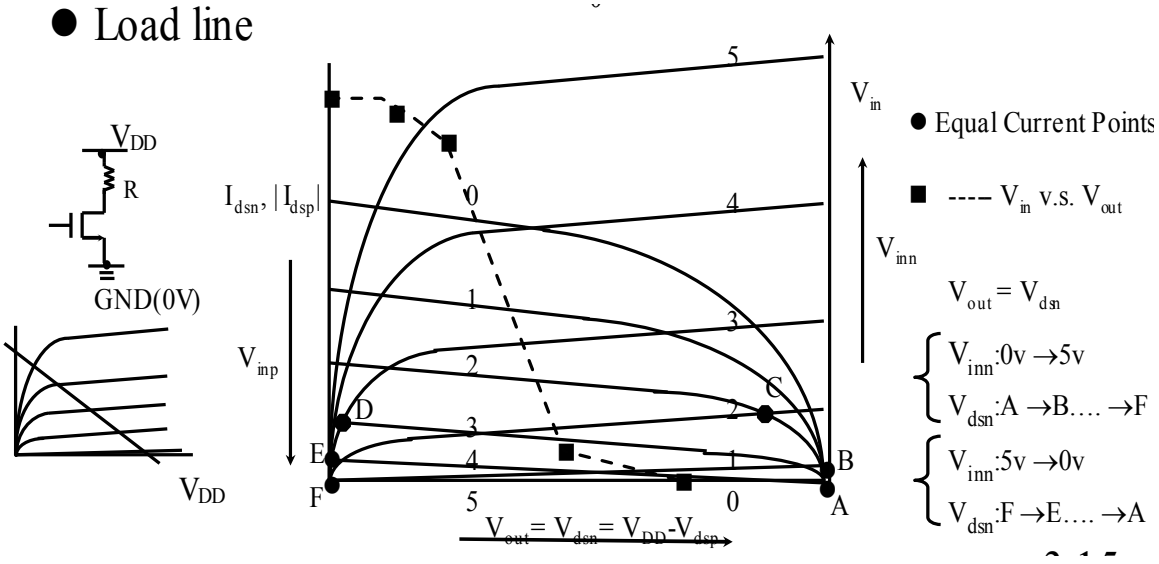
Pmos V-I characterstics

Nmos V-I characterstics

If we take the absolute value for idsp (the pmos plot is shifted upwards when we take the absolute value of current of p device) the plot looks as shown below.



now rotating the pmos v-I curve about the axis to right side we get intersection points for each curve (1-1,2-2,3-3 etc).this points are equal current points.



the input output transfer curve is determined now by the points of common intersections.substutiing Vinn=Vinp and Idsn=Idsp results in the transfer characterstics of Cmos Inverter.

Transfer characteristics Curve (Input /output curve)

Now we will see how the inverter output is related to the input.we will gradually apply the input from 0 volt to 5 volt (Vdd) and see how the output behaves.this input output relationship is know as the transfer characteristics curve X axis is input Vin and Y axis is output Vout.see the figure 3.11.when input Vin is 0 The output Vout is Vdd.There are five different regions on transfer characteristics curve namely Region A,B,C,D,E.The figure 3.11 shows these regions and the state of the nmos and pmos transistors.

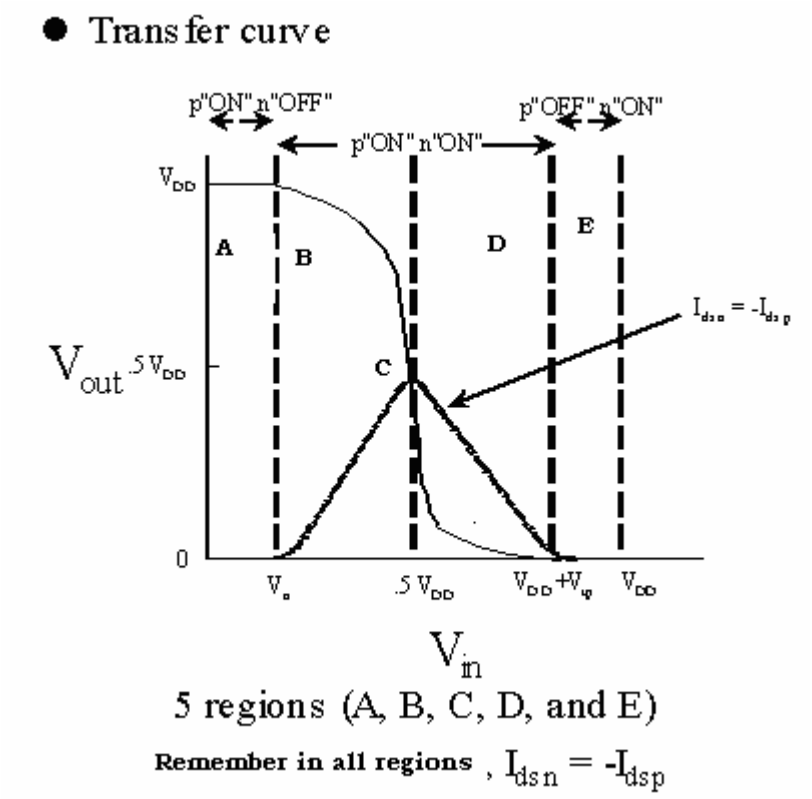


Figure 3.11 cmos inverter characteristics.

Region A : (0 < Vin < Vtn)

This region is defined when input Vin is between 0v and Vtn (threshold voltage of nmos transistor).in this region the pmos transistor is on and it conducts in triode (linear,nonsaturation) region where as nmos transistor is Off(Cut Off). Hence the output Vout = Vdd.

Region B:(Vtn ≤ Vin <Vdd / 2)

In this region as the input is greater than the threshold voltage of nmos transistor it starts conducting in saturation (observe that the Vds > Vgs-Vt) and pmos is conducting in linear region.

When the device conducts in saturation region it can be modeled as constant current source and when the device is conducting in linear region it can be modeled as resistive. The equivalent diagram for the region B can be drawn as pmos as resistive and nmos as constant current source. (Refer figure 3.12) since both the transistor are connected in series the current flowing in both transistor must be same. our aim now is to equate both the transistor current idsn= - idsp and derive Vin and Vout relations.

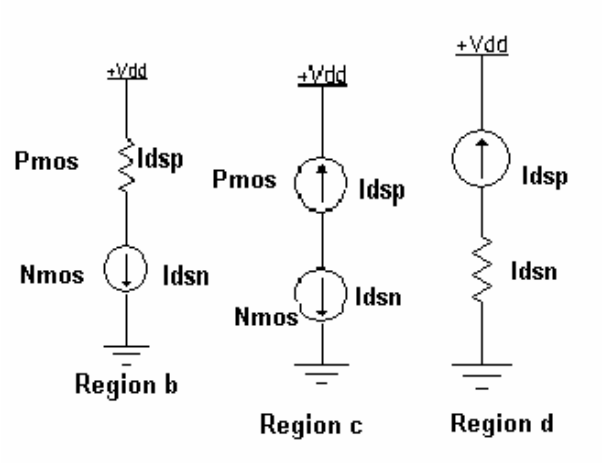
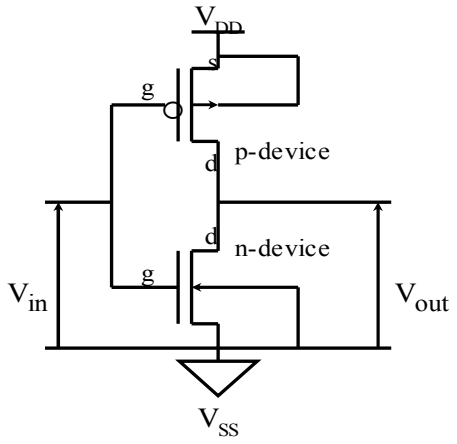


Figure 3.12 equivalent diagram of Cmos inverter in different region

Now let us recall the fundamental equations

- **$Ids=\beta((vgs-vt).vds-vds^2/2)$ nonsaturation (linear,triode) region**
- **$Ids=\beta/2(Vgs-Vt)^2$ saturation region**

Now for region B the nmos device is in saturation so writing the equation in saturation device for nmos**$Idsn = \beta n/2 (Vgs-Vtn)^2$** here Vgs for nmos transistor is Vin



Therefore **Idsn = βn/2 (Vin - Vtn)^2**

Now for pmos transistor it is in triode region the equation for this region is

Idsp= - βp((vgs$_p$-vtp).vds$_p$-vds$_p^2$/2)

For pmos vgs$_p$ = Vin –Vdd and Vdsp =(Vout - Vdd) refer the figure above.

Now substituting these values and equating both the currents we have

Region B: n: saturation p: nonsaturation

$$I_{dsn}=\frac{\beta_n}{2}(V_{in}-V_{tn})^2$$

$$I_{dsp}=-\beta_p[(\underbrace{V_{in}-V_{DD}}_{V_{gsp}}-V_{tp})(\underbrace{V_O-V_{DD}}_{V_{dsp}})-\frac{1}{2}(V_O-V_{DD})^2]$$

$$I_{dsn}=-I_{dsp}$$

$$\Rightarrow V_O=[(V_{in}-V_{tp})+[(V_{in}-V_{tp})^2-2(V_{in}-\frac{V_{bb}}{2}-V_{tp})V_{DD}-\frac{\beta_n}{\beta_p}(V_{in}-V_{tn})^2]^{1/2}$$

Region C:

In this region the input voltage is close to half of the Vdd (.5 Vdd) and both the transistor enters in to saturation region. This is the main region where the cmos inverter dissipates more power due to the flow of short circuit current between Vdd and Vss.since both the transistor are conducting in saturation there is direct short circuit between Vdd rail and Vss rail. The figure 3.14 shows the magnitude of current in various regions .in region 1 and 5 there is no current because Nmos

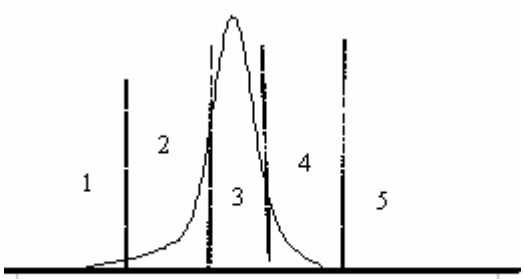


Figure 3.14 Magnitude of Current in various region of cmos inverter

and Pmos transistor are off respectively in these region. in region 2 and 4 small amount of current starts flowing ,however in region 3 more amount of current flows because both the transistor are in saturation.Now let us relate the current Idsn = -Idsp where both the device are in saturation.

Region C: Both the n and p-device are in saturation

$$I_{dsp}=\frac{\beta_p}{2}(V_{in}-V_{DD}-V_{tp})^2$$

$$I_{dsn}=\frac{\beta_n}{2}(V_{in}-V_{tn})^2$$

$$I_{dsp}=-I_{dsn}$$

$$\Rightarrow V_{in}=\frac{V_{DD}+V_{tp}+V_{tn}\sqrt{\frac{\beta_n}{\beta_p}}}{1+\sqrt{\frac{\beta_n}{\beta_p}}}$$

By setting $\beta_n = \beta_p$ and $V_{tn}=-V_{tp}$

$\Rightarrow V_{in}=\frac{V_{DD}}{2}=V_O=V_{inv}$ where V_{inv} is gate threshold voltage

Region D (.5Vdd < Vin < Vdd +Vtp)

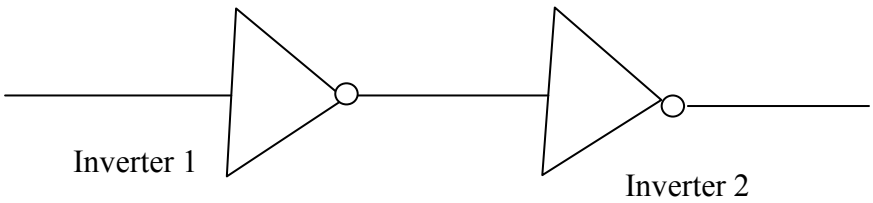
This region is similar to region B, except that here nmos transistor remains in saturation and pmos transistor conducts in nonsaturation region (just opposite to region B).the current equations now are altered. I encourage the reader to derive the input output relation in this region.

Region E Vin > Vdd+Vtp

In this region the nmos transistor is fully on and pmos transistor is now fully off because at least some negative voltage (threshold voltage for p mos transistor Vtp) is require for pmos to conduct .so the current in this region is Idsn = - idsp = 0.

3.11. Noise margin:

as we know that the digital logic device requires some voltage level to determine the logic high or logic low. in general terms we define this logic level as 1 and 0.but what actually logic 1 means and logic 0 means. To answer this questions the logic 1 and logic 0 can be defined as different level of voltage level. let us assume a positive logic where say +5 Volt is used to represent logic (1) High and 0 Volt for logic 0 or low. Now if we closely observe the transfer characteristics of the inverter curve we have seen that there are various voltage levels in between 0 and 5 volt. So what do we do with this different levels .the solution might be we can assume some range say 3 volt to 5 volt for logic 1 and 0 to 1.5 volt as logic 0,and ignored the range 1.5 to 3 volt as undefined logic. Yes this is what a noise margin is concerned. Also there might be the need when device are connected in cascade the output of first device is input to second device and so on.so noise margin is related to input output curve. For example figure below shows that two inverters are connected in series. The output of first inverter is input for second inverter.



Noise margin is defined as the maximum allowable noise at the input of the device so that the device correctly assumes the logic as high or low and perfumes the operation correctly. The figure 3.15 shows that there are two inverters in series. The output of the first inverter is input to the second inverter. There is a range for the output of the first inverter for logic 0 and logic 1.now how

much noise at the input side can second inverter allow for logic 1 or zero is the noise margin.

Before going further to noise margin let us see some parameters (Range for low and high logic) related to noise margin. Refer to the figure 3.15.

V_{OLMAX}: maximum Low output voltage (First inverter output for max low logic)

V_{ILMAX}: maximum Low input voltage (second inverter input for max low logic)

V_{OHMIN}: Minimum high output voltage (First inverter output for min high logic)

V_{IHMIN}: Minimum high input voltage (second inverter input for min high logic)

Noise margin is described in two parameters,

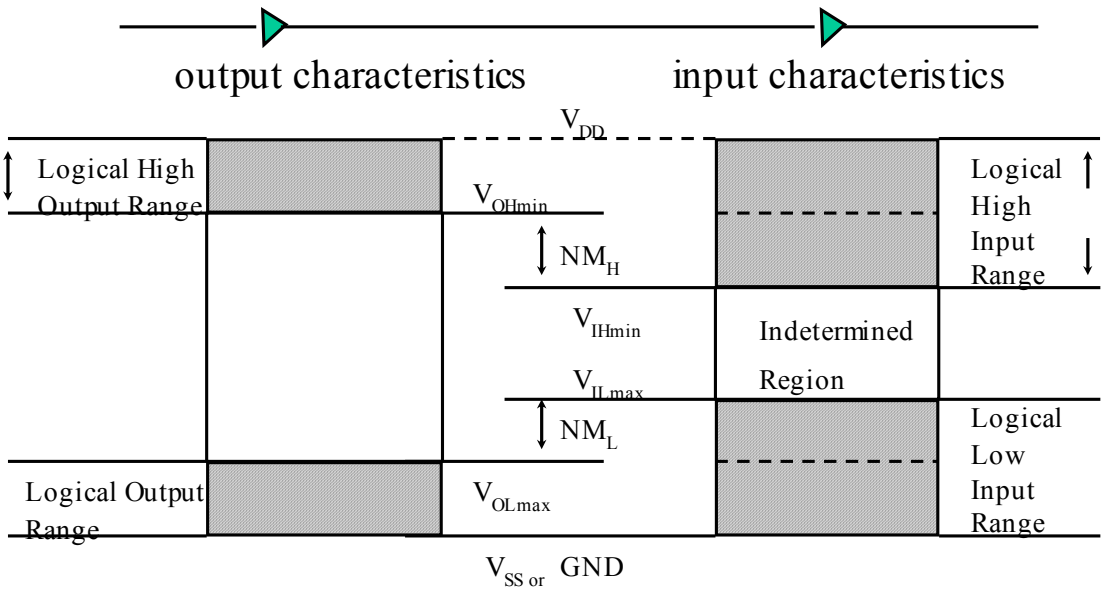
NM_L low noise margin (for logic 0).it is the modulas difference between the maximum Low input voltage and maximum Low output voltage.

- $NM_L = | V_{iL\ max} - V_{OL\ max} |$

NM_H High noise margin is defined as the modulas difference of Minimum high input voltage and Minimum high output voltage

$NM_H = | V_{IH\ min} - V_{O\ H\ min} |$.

The figure 3.15 shows the noise margin representation, the transfer characteristics curve has two points where the slope is unit and known as unity gain point. These two points determines the parameter four above parameters.V_{OL}, V_{IL}, V_{OH}, V_{IH}.



$$NM_L = | V_{ILmax} - V_{OLmax} |$$

$$NM_H = | V_{OHmin} - V_{IHmin} |$$

V_{IHmin} = Minimum HIGH input voltage

V_{ILmax} = Maximum LOW input voltage

V_{OHmin} = Minimum HIGH output voltage

V_{OLmax} = Maximum LOW output voltage

- CMOS inverter noise margin

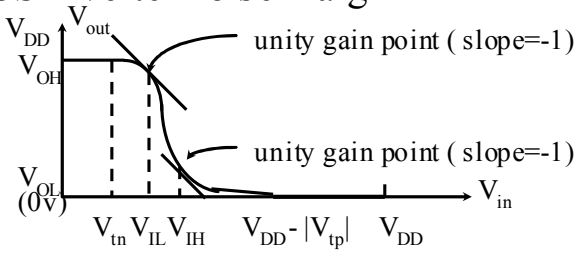


Figure 3.15 Noise margin and the characteristics curve.

3.12 PASS TRANSISTOR:

nmos and pmos transistors can be used to transmit the logic level from input to output hence the name pass transistor.nmos transistor degrades the high logic

value by an amount Vt(Threshold voltage) i.e out put is Vdd-Vtn at the output due to threshold voltage but passes logic zero without degradation so it is only suitable to pass logic zero not for logic 1. nmos transistor passes weak one and strong 0.similarly pmos transistor passes logic 1 as good and degrades logic 0 .now the question arises what if we want to pass both the logic 0 and 1 as good , the solution for this is if we combine both the transistor in parallel we are able to get good logic for both the logic. this combination is termed as transmission gate.

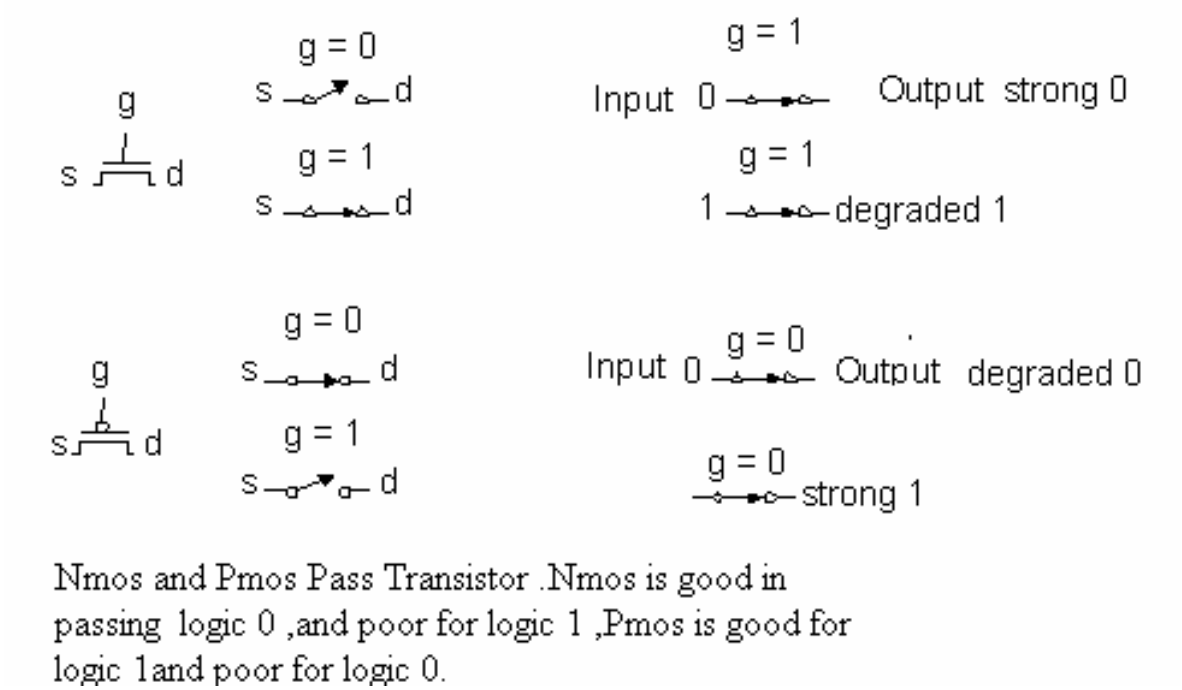
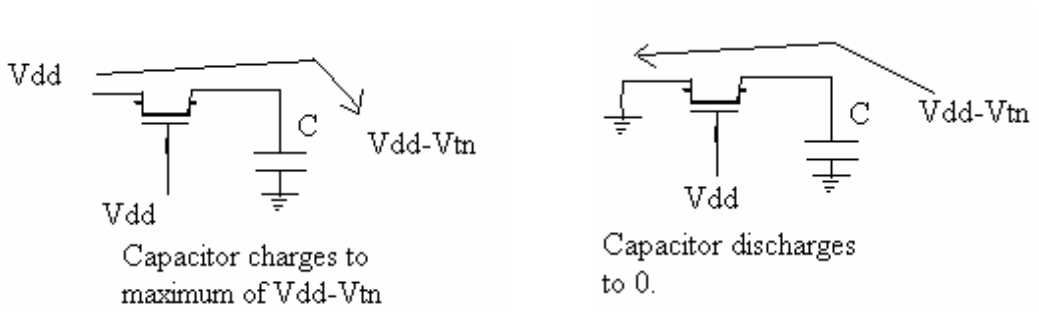


Figure 3.16 pass transistor

3.13. Cmos transmission gate

If pmos and nmos transistors are connected in parallel (refer figure 3.17) it forms the transmission gate. We will see the individual operation of nmos and pmos pass transistor .let us connect the load capacitor at the one terminal (either source or drain does not matter) and other terminal with Vdd, assume the capacitor is discharged initially, and the gate terminal is connected to Vdd. the capacitor starts charging towards Vdd but cannot reach Vdd (due to threshold voltage). The maximum voltage at the capacitor will be Vdd -Vtn. so nmos transistor degrades logic 1.now if the input source is made 0 the capacitor starts discharging towards 0.same approach can be made for pmos transistor also .if pmos transistor is connected instead of nmos transistor the capacitor is fully charged towards Vdd when gate is connected to 0,thus giving good logic for 1,however due to threshold voltage while discharging the capacitor voltage cannot fall completely to 0,thus providing poor logic for 0.



Now we will see what happens when we combine both the transistor .the goal of doing this is to get good logic for both logic levels 0 and 1.

The figure 3.17 below shows the structure and operation of the transmission gate. Transmission gates are used to pass both the logic 0 and logic 1 as good. Let us consider that g=0 and gb=1 then both the transistor are off so it does not allow the input to pass at the output. when g=1 and gb=0 both the transistor are on and input is connected to output. now if the input is logic 0 nmos transistor passes good logic for 0 and if the input is high pmos transistor passes good logic to output.

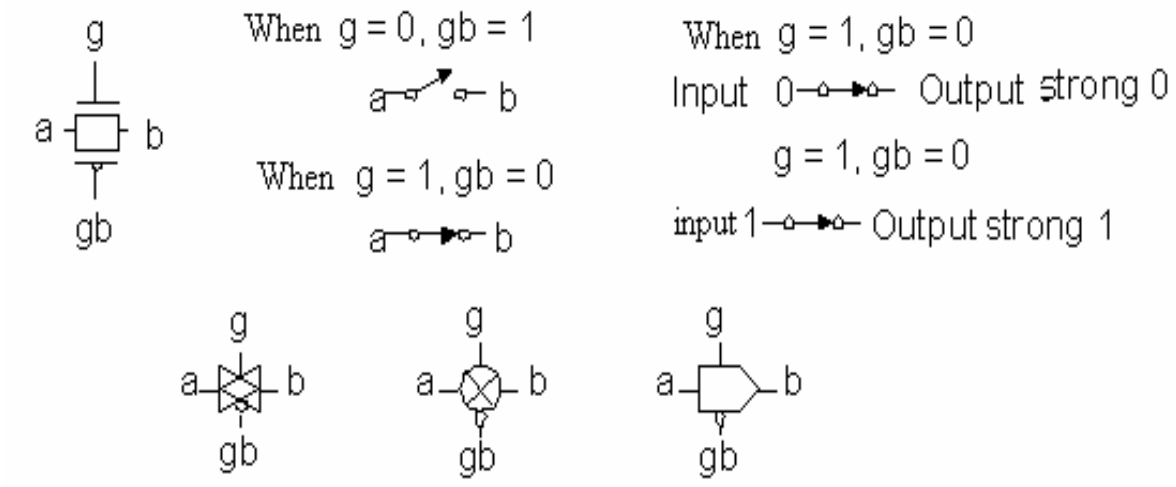


Figure 3.17-transmission gate.

3.14 Cmos tristate inverter:

Tristate refers to the logic level which has three states. Beside logic high and low (1 or 0) tristate device has third logic state which is neither high nor low, this state is known as high impedance state. Generally the devices are in this state when they are idle and help to save power. The control input alters the condition of device either to set in active mode where it operates as normal device or in tristate mode where it becomes idle and remains in high impedance state.

When the inverter is cascaded with transmission gate tristate inverter is formed as shown in the figure below. The control inputs clkn and clkp controls the inverter.clkn refers to negative and clkp refers to positive level. When clkp and clkn are asserted (enable) the inverter works normally however if the control input are not asserted than the inverter remains in high impedance state. the figure shows the cmos inverter cascaded with the transmission gate(figure 3.18 (a) .the link can be broken (figure b) and the transistor can be arranged in stack form as shown in figure3.18 (c).finally the symbol for tristate inverter is shown in figure d.

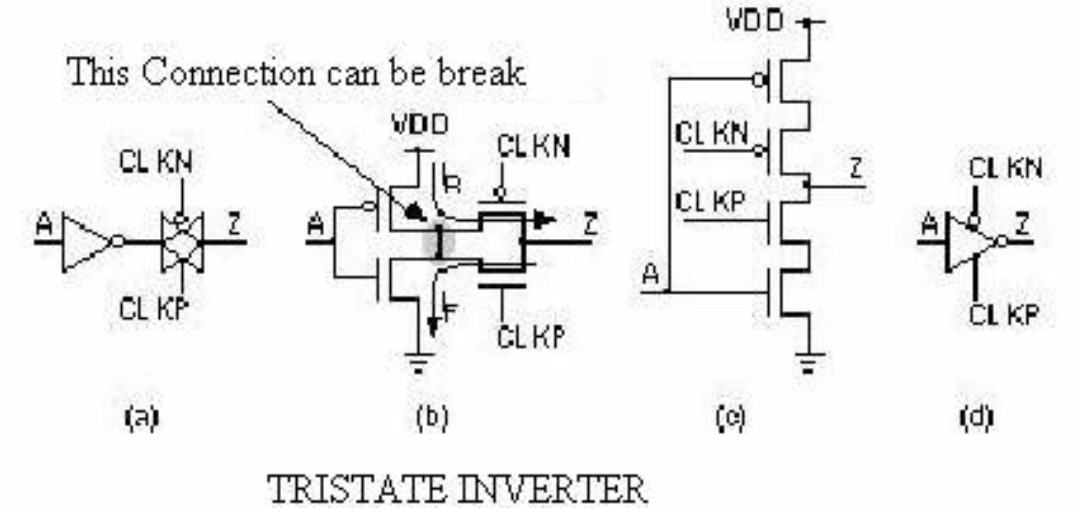


Figure 3.18 Tristate Inverter

3.15 Determination of Rise and fall time Of Cmos Inverter:

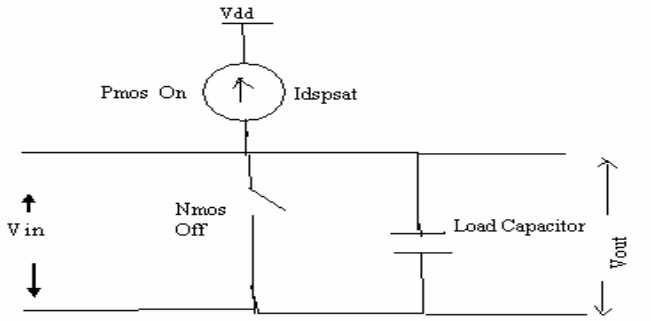
The speed of the mos transistor is determined by deriving the rise and fall time of the mos device. as we have seen already that certain time is required for the cmos inverter to transit from high state to low state and vice versa. Rise time T_r is defined as the time taken by the signal to reach 10% to 90% of its peak value. Similarly fall time T_f is defined as the time taken by the signal to fall from 90% to 10% of its peak value. Now our aim is to find rise and fall time so that we can determine the switching speed of the device. here I would like to put a simple derivation for finding the rise time, and in the similar manner a fall time can be calculate. When we apply logic 0 to the cmos inverter input the nmos transistor is off and Pmos Transistor is on. Let us assume that throughout the rise time the Pmos transistor remains in saturation. The figure shows the equivalent diagram. The figure below shows that the with the application low “0” input the pmos transistor is in saturation and the nmos transistor is fully off. The current I_{dspsat} flows from Vdd and the load capacitor starts charging towards Vdd. The out put voltage Vout = Charge stored in capacitor (Qc) / load capacitance (C_L)

$Vout=Q_c/C_L$ …… (A)

We know that **$Q_c=I_{dspsat}.T$,**

Now substituting the value of I_{dspsat} at saturation i.e.

$I_{dspsat}=Bp\,(V_{gs}-V_{tp})^2/2$,



Remember V_{tp} is the Threshold Voltage for Pmos and we are taking here absolute value of **.2V_{dd}**.
Substituting the value of Q_c and I_{dspsat} in equation (A) we get
Vout= Bp (V_{gs}-V_{tp})^2/2.T/C_L
Now we know the output is Vdd when input is "0"
So substituting **V_{out}=V_{dd}** and **V_{tp} =.2V_{dd}**
Also **Vgsp=V_{dd}** and T=T_r (rise time)
We approximately get the relation as

- **T_r ~ 3C_L/BpV_{dd}**

In general
T_r =K C_L/BpV_{dd}
Where K is constant and can vary from 3 to 4.

For fall time calculation when the input Vin is High ("Vdd")than the pmos transistor is off and nmos transistor is On. The capacitor now starts discharging towards ground"0".the fall time is calculated similarly and can be written as

- **Tf=kCL/Bn.vdd**

Now let us see that if rise time and fall time are same ,from the guess work we can say that fall time is less compared to rise time, because the mobility of holes is less than mobility of electrons. Pmos transistor is on for rise time and it constitutes of Holes. Where as nmos transistor is associated with fall time and has majority carriers as electron. Remember in mos transistor current constitute only majority carriers either by electrons or by holes unlike bipolar transistor where both electron and holes represents the current.

3.16 Cmos power dissipation:

Cmos circuits consume very less power compared to the Bipolar devices because of the insulated gate structure. Let us now see what are the factors which leads the cmos device to dissipate the power. There are major two factors which constitute the power dissipation for cmos circuit. These are Static and dynamic power dissipation.Static power means the power dissipated when the device remains in one state i.e when there is no transition of logic. Static dissipation occurs due to

- Reversed biased current .
- Subthreshold current

As we have seen the structure of cmos device which contains many junction and pwell ,nwell made up of p and n type material. This forms a reverse biased junction and causes the reverse biased current to flow. This current dissipates the power as power is the product of current and voltage. This reverse leakage current is given as.

$I_0 = I_s (e^{q v/ kT} - 1)$

Where reverse saturation current

Vt=kT/q= thermal voltage

I0 = leakage current

$$V_t = V_{t0} - \eta V_{ds} + \gamma \left(\sqrt{\phi_s + V_{sb}} - \sqrt{\phi_s} \right)$$

Also when Vgs<Vt a weak inversion takes place and a very small current flows in mos device known as subthreshold current. This current will also cause the power to dissipate.

$$I_{ds} = I_{ds0} e^{\frac{V_{gs} - V_t}{n v_T}} \left[1 - e^{\frac{-V_{ds}}{v_T}} \right]$$

So total power dissipation can be written as

$P_{Static} = \sum$ (leakage current X supply voltage)

The summation refers to the total number of transistor leakage current in the circuit or system.

Dynamic Power dissipation:

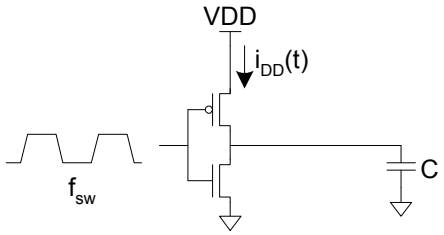
Dynamic power dissipation occurs when the logic is changing either from logic 0 to logic1 or logic1 to logic 0 .in this transition from one logic to another momentarily there is a flow of current from Vdd to Vss rail which is termed as **short circuit current**. Also to charge and discharge the load capacitor certain amount of current is required which is termed as **charging and discharging current**. This two current are the major factor which causes the major power dissipation factor in Cmos transistor.

Dynamic power dissipation occurs due to

- Charging and discharging of load capacitor
- Short circuit current.

Dynamic power dissipation: now let us derive the power dissipation due to charging and discharging of load capacitance. let us assume a switching frequency f_{sw} which is the frequency of charging and discharging the load capacitor. The time period is T then we can define average power as

$$P_{\text{dynamic}} = \frac{1}{T}\int_0^T i_{DD}(t)V_{DD}dt$$
$$= \frac{V_{DD}}{T}\int_0^T i_{DD}(t)dt$$
$$= \frac{V_{DD}}{T}\left[Tf_{\text{sw}}CV_{DD}\right]$$
$$= CV_{DD}{}^2 f_{\text{sw}}$$



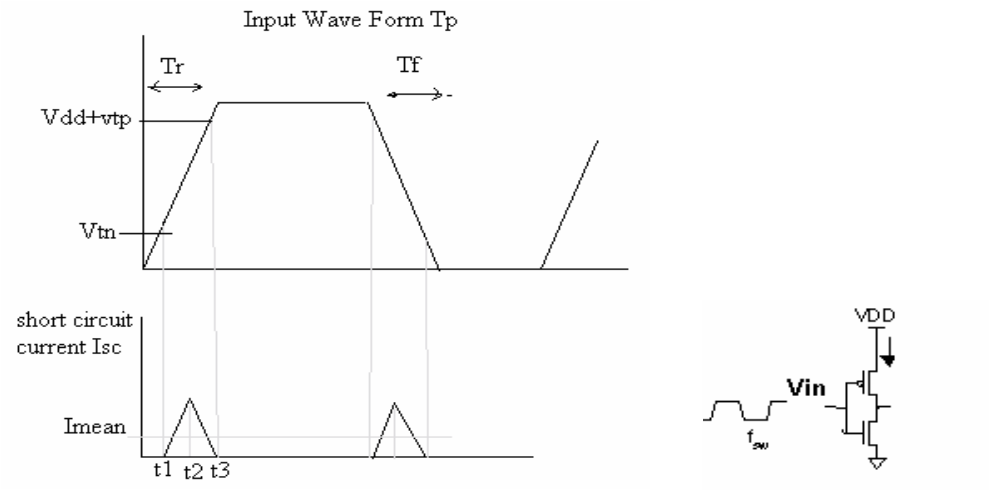
Hence the dynamic power dissipation due to charging and discharging of load capacitance depends upon the capacitance, the switching frequency and the supply voltage. To reduce the power dissipation due to charging and discharging current the most important thing is to reduce the power supply voltage because it has the quadratic effect(square term).more the logic transition more power dissipation so switching frequency can be reduced for low power. The capacitance can be also decreased to reduce dynamic power dissipation.

Short circuit power dissipation:

The short –circuit power dissipation is given by the product of current (mean value) and supply voltage

$P_{sc} = I_{mean} . V_{DD}$.

For the input wave form shown in figure.



$$I_{mean} = 2 * [1/T \int_{t1}^{t2} I(t)\,dt + 1/T \int_{t2}^{t3} I(t)\,dt]$$

Assuming that

$V_{tn} = -V_{tp}$ and $\beta_n = \beta_p = (\beta)$

and that the behavior is symmetrical around t_2

$$= 2 * 2/T \int_{t1}^{t2} \beta/2 \,(V_{in}(t) - V_t)^2\,dt,$$

With

$$V_{in}(t) = V_{DD}.\, t / t_r$$

$$t_1 = V_t . t_r / V_{DD}$$

$$t_2 = t_r /2 .$$

Thus for an inverter without load ,assuming that $t_r = t_f$ (t_{rf}),

$P_{sc} = \beta/12 \;\; (V_{DD} - 2V_t)^3 \, t_{rf} / t_p$.

Where **t_p** is period of the input wave form .The derivation shows that the short circuit current is dependent on **β** and the rise and fall time of input wave form

Total power dissipation In Cmos :

Now we can say that the total power dissipation in cmos is the sum of static power dissipation, short circuit power dissipation and dynamic(charging and discharging) power dissipation

Hence

$P_{total} = P_{static} + P_{dyn} + P_{shortckt}$

CHAPTER 4

- Basic concepts
- VLSI design flow
- Design Flow using Verilog
- Verilog HDL
- Verilog keywords
- Verilog data types
- Verilog operators
- Verilog gate primitives
- Verilog number and value set

- Gate delay
- Structural level modeling
- Switch level modeling
- Timing control and delay
- Coding style in verilog
- Continuous assignment statement
- Procedural assignment
- Sequential statements
- Example of behavioral modeling

- Verilog summary and quick recall

Chapter 4

The aim of this unit is to make the readers familiar with one of the Hardware description language VerilogHDL.

4.0 Basic concept:

Early days circuit designer followed, the classical design method (Schematic, manual) to design a circuit these method has now become obsolete due to sheer complexity of the design. With the emerge of VLSI design technology it is very difficult if not impossible to follow the old design methodology. Computer based design language are now very popular among the designer. with the availability of computer aided design tools designer find very comfortable working with this tools .circuit of large size and complexity are designed using the language called Hardware description language HDL (a software used to design the digital function).today integrated circuits (ICs) may contain as many as millions of gates which is virtually impossible to sketch in paper .the most commonly used HDL language are verilog HDL and VHDL (Very high speed integration circuit HDL)

4.1 VLSI Design Flow:

VLSI design starts with the design specification (ideas).after the conceptualization is done VLSI design goes in two more design phase. The first phase is logical design and the second one is Physical design.the figure 3.0 shows the VLSI design flow.

Logical Design:

In this phase the design specifications are enter in some design entry tools. This tool supports textual as well as graphical form. Textual representation is done with HDL languages (VHDL, VerilogHDL).graphical representation is in the form of schematic capture or state diagram .once the design is entered in this format it is validate for functional verification. Cad tools can verify the functionality of the design .the design can be written in various coding style. These include structural, behavioral, dataflow, gate and switch level (in verilog).

The implementation of the design can be in Programmable logic devices like FPGA and CPLDs or Building a chip.

After the verification of the design it can be synthesize to lower level of abstraction. Up to this phase it is referred as logical design.

Physical design:

Physical design refers to the actual mask layout of the chip, which has several phases .once the circuit is synthesize to gate level it is partitioned in smaller sub circuits, than floor planning is used to arrange this sub circuits in proper place such that the size of the chip is optimal also the routing of wires is minimum, after floor planning, Placement and routing are performed. Placement is used to arrange the cells in side the module and routing is used to interconnect each cells and modules .Finally mask layout is prepared and fabrication, packaging is done which we call Integrated circuit(IC).

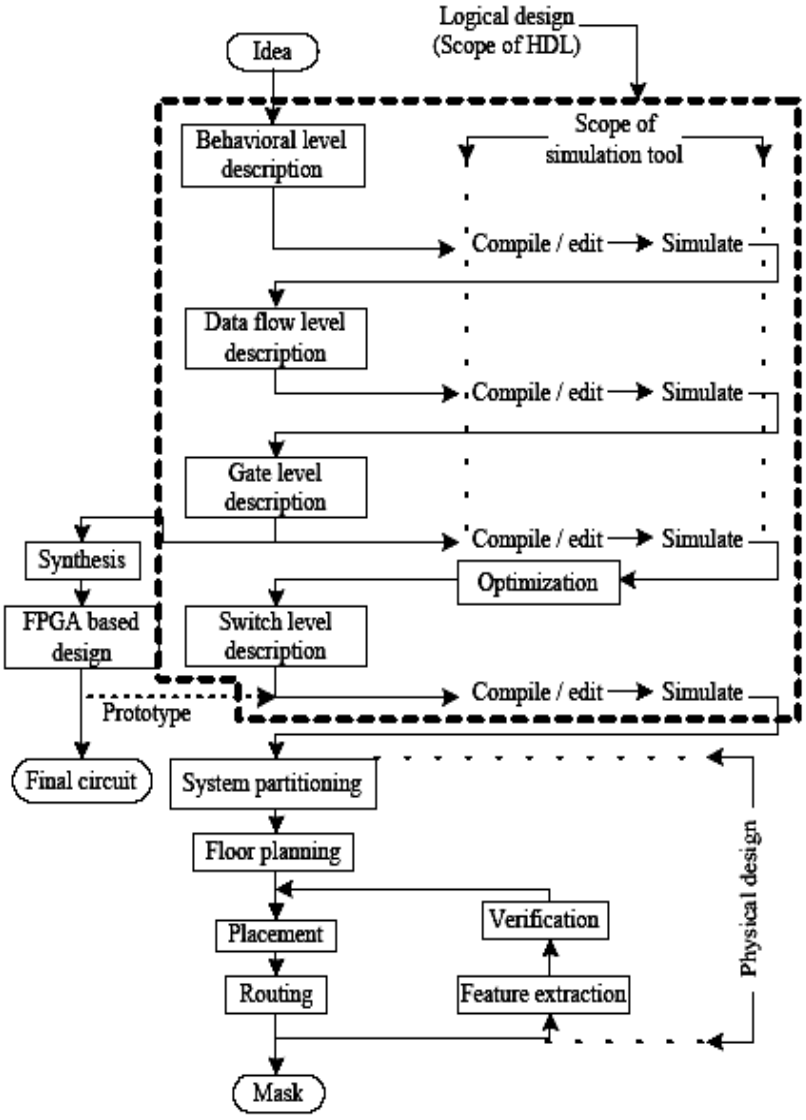
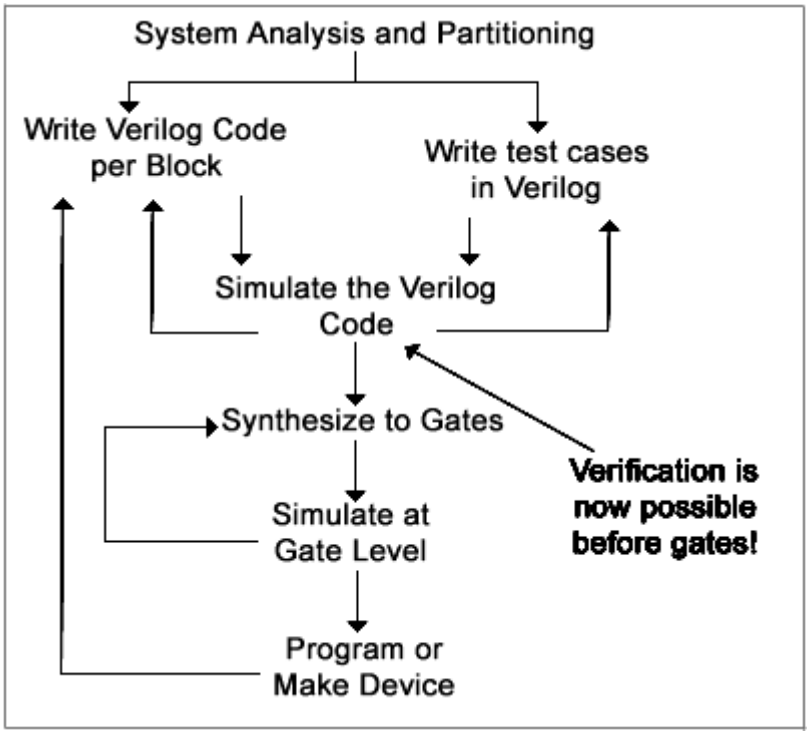


Figure 3.0 VLSI design flow

4.1.1 Design Flow using Verilog

The diagram below summarizes the high level design flow for an ASIC (ie. gate array, standard cell) or FPGA. In a practical design situation, each step described in the following sections may be split into several smaller steps, and parts of the design flow will be iterated, as errors are uncovered.



System-level Verification

As a first step, Verilog may be used to model and simulate aspects of the complete system containing one or more ASICs or FPGAs. This may be a fully functional description of the system allowing the specification to be validated prior to commencing detailed design. Alternatively, this may be a partial description that abstracts certain properties of the system, such as a performance model to detect system performance bottle-necks.

Verilog is not ideally suited to system-level modelling. This is one motivation for SystemVerilog, which enhances Verilog in this area.

RTL design and testbench creation

Once the overall system architecture and partitioning is stable, the detailed design of each ASIC or FPGA can commence. This starts by capturing the design in Verilog at the register transfer level, and capturing a set of test cases in Verilog. These two tasks are

complementary, and are sometimes performed by different design teams in isolation to ensure that the specification is correctly interpreted. The RTL Verilog should be synthesizable if automatic logic synthesis is to be used. Test case generation is a major task that requires a disciplined approach and much engineering ingenuity: the quality of the final ASIC or FPGA depends on the coverage of these test cases.

For today's large, complex designs, verification can be a real bottleneck. This provides another motivation for SystemVerilog - it has features for expediting test bench development. See the SystemVerilog section of Knowhow for more details.

RTL verification

The RTL Verilog is then simulated to validate the functionality against the specification. RTL simulation is usually one or two orders of magnitude faster than gate level simulation, and experience has shown that this speed-up is best exploited by doing more simulation, not spending less time on simulation.

In practice it is common to spend 70-80% of the design cycle writing and simulating Verilog at and above the register transfer level, and 20-30% of the time synthesizing and verifying the gates.

Look-ahead Synthesis

Although some exploratory synthesis will be done early on in the design process, to provide accurate speed and area data to aid in the evaluation of architectural decisions and to check the engineer's understanding of how the Verilog will be synthesized, the main synthesis production run is deferred until functional simulation is complete. It is pointless to invest a lot of time and effort in synthesis until the functionality of the design is validated.

4.2 VERILOG HARDWARE DISCRIPTION LANGUAGE:

Digital design using verilog starts with the design entity called module and ends with the endmodule. In between module and endmodule, various language constructs and modeling styles can be accommodated. We will now see how to write verilog module. Remember that verilog Is Case Sensitive for example a is not same as A.

4.2.1 Module and ports.

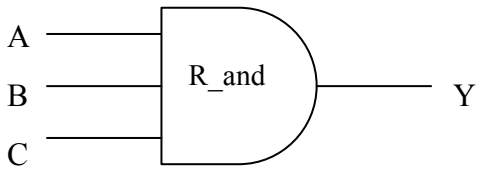
Every Verilog design consists of module. A module is a basic entity of the design. Module consists of design name and port list. Design can have one or more module instantiation.
The syntax for defining the module is:
module design_name (portlist);

input list;
output list;
inout list;
;
.
.
endmodule

Design name is identifier for the name of the design.portlist includes all the input and output of the design. **input** keyword is used for inputs. **output** key word is used for outputs.**inout** key word is used for bidirectional (input and output).

Let us look for few examples to define module and ports.

The figure below is 3 input and gate. here the gate name is r_and (to distinguish from and keyword in verilog because keywords cannot be used as identifier.)

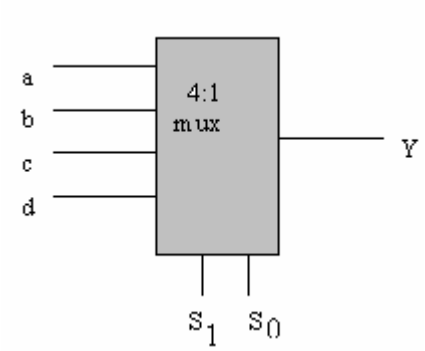


module r_and (A,B,C,Y); \\module design name and portlist
input A; \\direction
input B;
input C;
output Y;
..
..
endmodule

Verilog module and port definition for 3 input and gate; here design name is **r_and**. port list contains input and output so A,B,C,Y. where A,B,C are input and Y is output.Every verilog statement ends with semicolon. Module must be terminated with endmodule keyword. To write comments// sign is used. Note that verilog is case sensitive hence every keyword has to be written in small case. Instead of defining input /output separately they can be defined in one statement separated by comma as shown below.

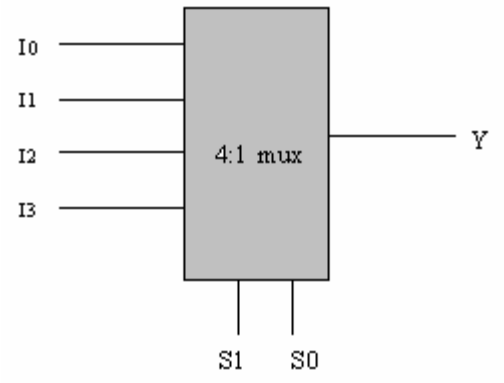
module r_and (A,B,C,Y);
input A,B,C;
output Y;

Now let us take a look how a 4:1 multiplexer module and ports can be define in verilog



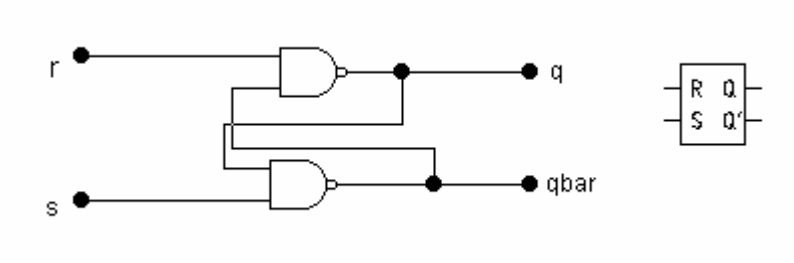
module mux4_1(a,b,c,d,s1,s0,y); \\ identifier cannot start with digit so mux4_1
Input a,b,c,d; \\input
Input s1,so;
Output y; \\ output
..
..
..
endmodule

In above example all the inputs and outputs are single bit, these are referred as scalars. Verilog provides the flexibility to define multi bit inputs and outputs. Multi bits are referred as vectors.
Now consider the input as I [0], I [1], I [2], I [3], s [1], s[2] ,here I is an array of length 4 and s is array of length 2.



Module mux4_1(I,S,Y); // I and s will be defined as vector
Input [3:0] I; // here I is vector of length 4 starting from I[0] to i[3].
Input [1:0] S; // s is vector of length 2.
Output Y;

Now let us examine the figure below. it is a basic sr latch. at first sight we might say that y and ybar are output but examine carefully they are output as well as input also.y and y bar are connected at the input side of nand gates along with other two inputs r and s.hence y and ybar has to be declared with inout keyword not output..



module sr_latch(r,s,q,qbar);
input r,s;
inout q,qbar;

4.2.2 Identifiers:

Identifiers are names that are given to elements such as modules, registers, ports, wires, instances, and procedural blocks. An identifier is any sequence of letters, digits, and the underscore (_) symbol except that:
the first character must not be a digit, and the identifier must be 1024 characters or less.

Verilog Identifier Naming Conventions

An identifier in a Verilog file must adhere to the following conventions. For more information see the IEEE Standard Description Language Based on the Verilog™ Hardware Description Language manual.

- Must begin with an alphabetic or underscore character (a-z, A-Z, or _)
  - Can contain alphanumeric (a-z, A-Z, 0-9), underscore (_), or dollar sign ($) characters
  - May use any character by escaping with a backslash (\) at the beginning of the identifier and terminating with a white space (a blank, tab, newline, or formfeed). For example, the identifier "reset*" is not acceptable but the identifier "\reset* " is acceptable.
  - Can be up to 1024 characters long
  - Cannot contain white space

  Note Identifiers are case sensitive.

Verilog is case sensitive, ie Upper and lower case letters are considered to be different. System tasks and system functions are identifiers that always start with the dollar symbol. Escaped identifiers allow for any printable ASCII character to be included in the name. Escaped identifiers begin with white space. The backslash ("\") character leads off the identifier, which is then terminated with white space. The leading backslash character is not considered part of the identifier. Examples of escaped identifiers include:

\flip-flop
\a+b

Escaped identifiers are used for translators from other CAD systems. These systems may allow special characters in identifiers. Escaped identifiers should not be used under normal circumstances.

Verilog Keywords

Verilog keywords are reserved words which are predefined .verilog is case sensitive language .keywords are written in small case.the below mention key words cannot be used to define as module name ,variable name or constant name.

always	endmodule	large	reg	tranif0
and	endprimitive	macromodule	release	tranif1
assign	endspecify	nand	repeat	tri
attribute	endtable	negedge	rnmos	tri0
begin	endtask	nmos	rpmos	tri1
buf	event	nor	rtran	triand
bufif0	for	not	rtranif0	trior
bufif1	force	notif0	rtranif1	trireg
case	forever	notif1	scalared	unsigned
casex	fork	or	signed	vectored
casez	function	output	small	wait
cmos	highz0	parameter	specify	wand
deassign	highz1	pmos	specparam	weak0
default	if	posedge	strength	weak1
defparam	ifnone	primitive	strong0	while

disable	initial	pull0	strong1	wire
edge	inout	pull1	supply0	wor
else	input	pulldown	supply1	xnor
end	integer	pullup	table	xor
endattribute	join	rcmos	task	
endcase	medium	real	time	
endfunction	module	realtime	tran	

White Space and Comments

White space is defined as any of the following characters: blanks, tabs, new lines, and form feeds. These are ignored except for when they are found in strings. There are two forms of comments. The single line comment begins with the two characters // and ends with a new-line. A block comment begins with the two characters /* and ends with the two characters */. Block comments may span several lines. However, they may not be nested.

4.2.3 Verilog data types:

Unlike other programming language verilog is used to model digital hardware logic circuits so it is rich in data types that can be supported. it supports signal, and time in addition to constant and variables etc.
Verilog has two families of fixed data types:**Nets and register**. Nets establishes structural connectivity, registers stores information temporarily.

Net (Signal Is referred as net in verilog)

Net data types are

Wire, tri , wand , wor, triand, trior, supply0 , suppl1, trireg

Example if p,q are internal node of any logic circuits we can define these signals as
Wire p,q; \\ indicates p and q are signals
tri p; \\ indicates p is tristate signal.
Supply0 and **supply1** refres to ground and power respectively.
Wand :a net that is connected to the output of multiple primitives. this models as wired and logic.
Trireg models a net which stores charge in node.

Verilog variable
Variable are the data which can change their value anytime throughout the program. To define variable the keyword reg, integer, real, realtime are used**.**
reg refers for register variable which is used only when the sequential statements are used to hold the data temporarily. Register variable are assigned values by procedural (sequential) statements within an always keyword or initial block. Register variables hold their value until an assignment statement changes them.

Types of register variables
Reg,integer,real,realtime,time

reg variable is used for temporary storage it does not corresponds to physical memory or hardware register.**reg** type data has default initial value of X (unknown) and until an unless specified the size is one bit..
Example of reg type
reg y; \\ y is a scalar variable which is 1 bit in size and default value is X
reg [3:0] y; \\y is vector and is 4 bit in size y[0],y[1],y[2],y[3].
reg [3:0] x , y ; \\ both x and y are 4 bit in size
reg [7:0] mem [3:0] \\corrosponds to memory name mem which size is 8*4 bit.
Integer variable:
The data type integer supports numeric computation in procedural statements. Integers are represented internally to the word length of the host machine (at least 32 bit).a negative integer is stored in 2s completed format. Integer type has initial default value of 0.
Integer x;
Integer y,z ;
Integer q [4:0];

Real data type

Real data type is denoted with keyword real. It is used to represents the real type of data .The default value is 0.0

Real pi = 4.4;
Real a,b,c;

Time data type
The data type time are used for time related computation in procedural code. Realtime data type is also supported by verilog and has default initial value of 0.0.

Verilog constant:

In verilog constants are defined with keyword **parameter**. Parameter deals only with unsigned numbers unlike integer which can be used for signed numbers.

Example
Parameter count=8;
Parameter sum=10;
Parameter one=1;

4.2.4 Verilog operators:

Verilog has sets of operators which can operate with different data types.operators are similar to c language operator.verilog unary operators operates on single operand..it operates bit wise. These are also known as reduction operator.the table below shows the list of operators.
Unary operators and their precedence:

Operator	Name
!	logical negation
~	bitwise unary negation
&	unary reduction and
~&	unary reduction nand
|	unary reduction or
~|	unary reduction nor
^	unary reduction xor
~^ ^~	unary reduction xnor
+	unary plus
-	unary minus

?: (conditional opearator) Trinary operator.
|| (logical or
&& (logical and
| (bitwise or) , ~| (bitwise nor)
^ (bitwise xor) , ^~ ~^ (bitwise xnor, equivalence)
& (bitwise and) , ~& (bitwise nand)
== (logical equal) , != (logical not equal)
< (less than) ,<= (lt or equal) ,> (greater than) , >= (gt or equal)
<< (shift left) , >> (shift right)
+ (addition) , - (subtraction)
* (multiply) , / (divide) , % (modulus)
Unary operators: !, ~ ,& ,~,& ,| ,~| , ^ , ~^ , ^~, +, -

Example of unary reduction operator
& 4'b1111 is 1'b1,
& 2'bx1 is 1'bx, \\ x is unknown
& 2'bz1 is 1'bx \\ z is high impedance

Unary reduction nand~&

~& 4'b1111 is 1'b0,
~& 2'bx1 is 1'bx

Binary operator operates on 2 operands
example: a&b \\a and b
a^v \\axor b
a+b \\a plus b

Ternary operator operates on 3 operands.

Example: the conditional statement

Y=S ? a : b \\if s='1'(true) then Y is assigned to a i.e (Y=a) ,if s=0(false) then Y=b;

some of the operators in verilog:

Arithmetic: bitwise

+ , - , * , / , %

Reduction operators

&,~&,|,^,~^,^~

Example

If A=0111
Starting from left, bits are operated bit by bit.
&A results to 0. from left to write first it ands 0 with 1,result is 0,this is anded with 1,result is 0,this is finally anded with 1 still the result is 0.
| A results to 1.0 is ored with 1, result is 1,this result is ored with 1, result is 1,and so on.
This unary operator compares bit by bit and determines the result.
Logical operator

! \\logical not
&& \\logical and
|| \\logical or
!= \\not equal to
=== case equality
Example

If (a==b)
If (a==b && b!=c)

Relational operators
<,>,<=,>=,
Example
if (a>b)
if (a<=c)
Shift operators:
<< \\left shift
>> \\right shift
Example
If a=0010
a<< 1 results in 0100 \\shifts left one bit position
a<<2 results in 1000 \\shifts 2 bit position to left

Conditional operator

? This is conditional operator.
Y = s ? a : b \\if s =’1’ (true)then y gets the value a ,if s=’0’(false) y gets the value ob b;

Concatenation operator:

{ }

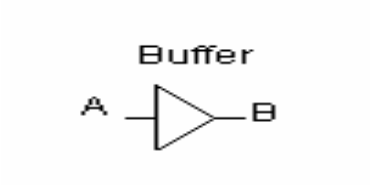
This operator is used to combine two or more operands to single operand.

Example
If a=01 and b=11 then
{a , b}results in 0111.

4.3 Verilog gate primitives:

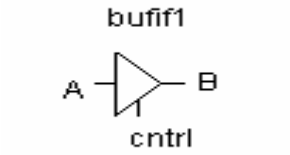
Verilog has 26 sets of primitives for modeling the functionality of combinational and switch level logic. The output terminals are listed first and the input terminal are listed last. Some of the primitives are:
Buffer primitives: buf, bufif0 , bufif1
Not Primitive: not ,notif0 , notif1
Transceiver (bidirectional buffer): Tranif0,Tranif1
Logic gates primitives: and ,Or , xor , nand ,nor
Mos switch primitives:Nmos,Pmos,Cmos
Resitive switch primitives:Rcmos(out,in,control), Rnmos, Rpmos

Buffer
The syntax is **buf (out,in)**



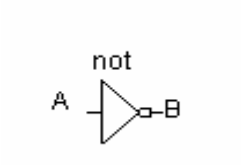

```
Module buffer (A,B);
Input A;
Output B;
buf (B,A);
endmodule
```

Bufif1
This is also a buffer primitive but it passes the input to the output only when control input cntrl is 1.



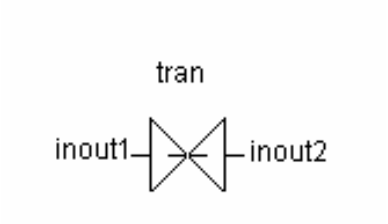

```
module bufferhigh (A,B,cntrl);
Input A,cntrl;
Output B;
bufif1  (B,A,cntrl);
endmodule
```
similarly the buffer which is control by low input is represented by
bufif0(out,in,cntrl)

Not Primitive: verilog gate primitive consist of inverter with or without control input the syntax is **not (out,in) ,notif0(out,in,cntrl) , notif1(out,in,cntrl)**




```
module notprim (A,B)
input A;
output B;
not (B,A)
endmodule
```

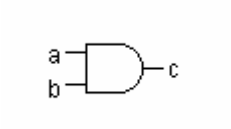
Transreciver (bidirectional buffer):
verilog provides the use of bidirectional buffer with or without control input.the syntax is given below.




```
tran(inout1,inout2)
tranif0(inout1,inout2,cntrl)
tranif1(inout1,inout2,cntrl)
```

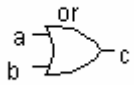
And primitive:

and gate primitive syntax is **and (out,input1,input2)**



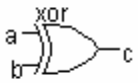

```
module andprim (a,b);
input a;
output b;
and(c,a,b);
endmodule
```

Or primitive:

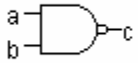


module orprim (a,b);
input a;
output b;
or (c,a,b);
endmodule

Xor primitive syntax is **xor (out,in1,in2);**

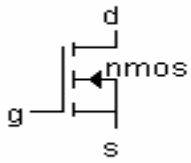


Nand syntax is **nand (out,in1,in2);**

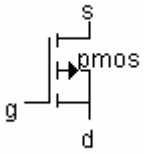


Nor

Mos switch primitives:verilog can be used to model at switch level where nmos ,pmos and cmos transistor cabn be used as a switch.
Nmos
nmos (out, ground, in); // instantiate nmos switch out is drain d,ground is source s,and in is gate g.



Pmos



pmos (out, power, in); // instantiate pmos switch

Cmos transistor primitive is represented by **cmos(out,in1,in2,out);**
Resitive switch primitives:the nmos,pmos and cmos transistor can also be used as a resistor .these resistive primitives are:
rcmos (out,in,ncontrol,pcontrol)
rnmos (out,in,control)
rpmos (out,in,control)

4.4 Verilog Number and value set:

verilog is rich in numbering system and values to be set .decimal, hexadecimal, octal, binary number systems can be used to represent the values of the variable. it supports unknown value x and high impedence value z.the following table gives the clear view how the number are represented in verilog and how they are stored.

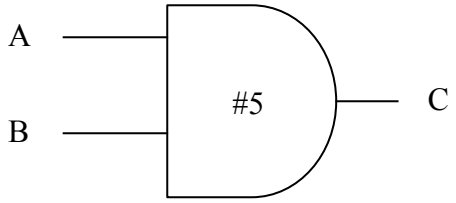
Expression	Meaning	value stored as
5'b00101	5 bit binary 00101	00101
8'b0	8 bit binary 00000000	00000000
8'b101	8 bit binary 00000101	00000101
8'd5	8 bit decimal 5	00000101
8'h9f	8 bit hex 9f	10011111
3'd1	3 bit decimal 1	001
4'bz	3 bit decimal z	zzzz
4'bx1	binary	xxx1
5'b11z	binary	0011z
15	32 bit decimal 15	0....01111 (32 bits)
'o5	32 bit octal 5	0....00101 (32 bits)

Notes:

- 1: Active high bit
- 0: Active low bit
- z: high impedance
- x: Uncertain/ Don't care
- If not mentioned, length is 32 bit and data type is decimal integer by default.
- If value is larger than the length, left most bits will be truncated
- If value is smaller,
  - 0's are filled to the left if left most bit is 0 or 1
  - 'z' are filled if left most bit is z
  - 'x' are filled if left most bit is x

4.5 Gate delays:

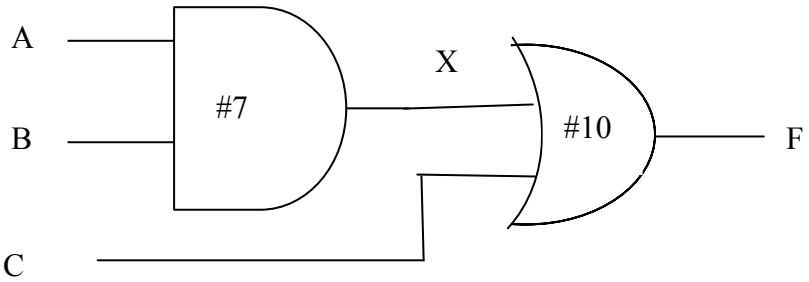
When the inputs are applied to the gate it takes certain amount of time to produce the output, we do not notice it because the delay is very small fraction of second. This delay is due to the internal resistance and parasitic effect along with load capacitance. This delay can be also termed as propagation delay from input to output.the figure below shows that the two input **and** gate which is having a delay of 5-time unit. This means that the gate produces the output after 5-time unit delay. **#** symbol is used to model gate delay .



Verilog code to model the gate above delay is

```
module r_and (A,B,C);      \\r_and is the name of design
Input A,B;
output C;
and #5 (C,A,B) ;   \\
endmodule
```

The figure below shows the and and or gate in cascade .the first gate and has delay of7 time unit and or gate has delay of 10 time unit.



Verilog cod

```
module and_or (A,B,C,F);
Input A,B,C;
output F;
Wire X;                  \\x is not input or output it is internal node so it declared wire
and  #7 ( X,A,B );
Or #10  (F,X,C);
endmodule;
```

gate delays can be further classified as a rise time delay, fall time, turn off delay.this delays are separated by comma .for instance let us assume a and gate with a and b as input and c as output with rise time delay #2,fall time delay #3 and turn off daly #5 unit..
verilog statement for this is:

```
module r_and (a,b,c);
Input a,b;
output C;
and # (2,3,5) (c,a,b);   \\here  rise ,fall and turn off time are separated by comma.
endmodule
```

Now at the end we will just see this gate delays can also have maximum, minimum and typical values associated with each rise, fall and turn off delay time.
Syntax for this is as follows observe carefully comma separates the rise, fall and turn off delay and semicolon separates maximum, minimum and typical.

```
 and # (4:2:3,5:3:4,5:8:6) (c,a,b);
```

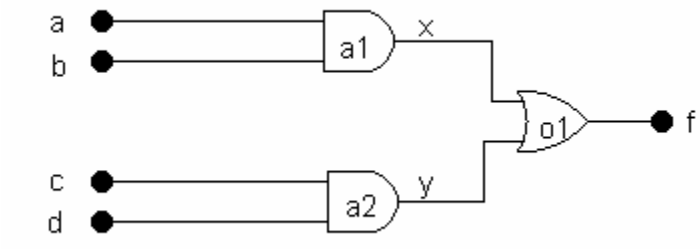
the gate has rise time specification as maximum of 4unit. Minimum of 2 unit and typical of 3 unit. Fall time maximum of 5 units, minimum of 3 units and typical of 4 unit. And so on for turn off delay .

4.6 Structural level modeling:

Structural level modeling in verilog uses various gate primitives and other primitives which are connected together to realize the logic circuit. Structural level modeling refers to the architecture of the logic circuit. Nets data type are used to connect intermediate nodes. Let us see a simple example,

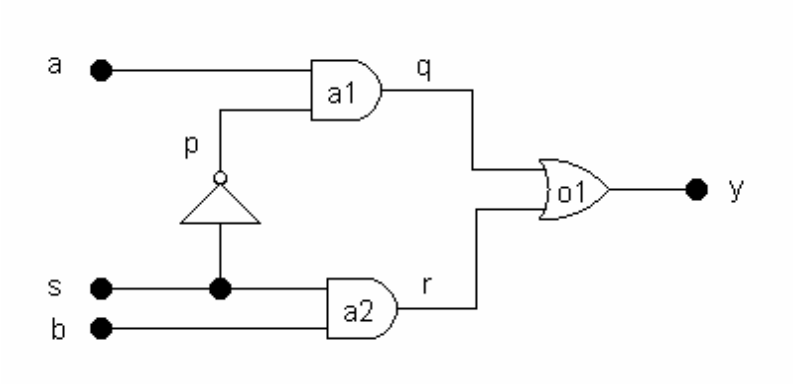
F= C.D+A.B

```
module example(a,b,c,d,f);
   input a;
   input b;
   input c,d;
   output f;
    wire x,y;
          and a1(x,a,b);
          and a2(y,c,d);
          or o1(f,x,y);
endmodule
```



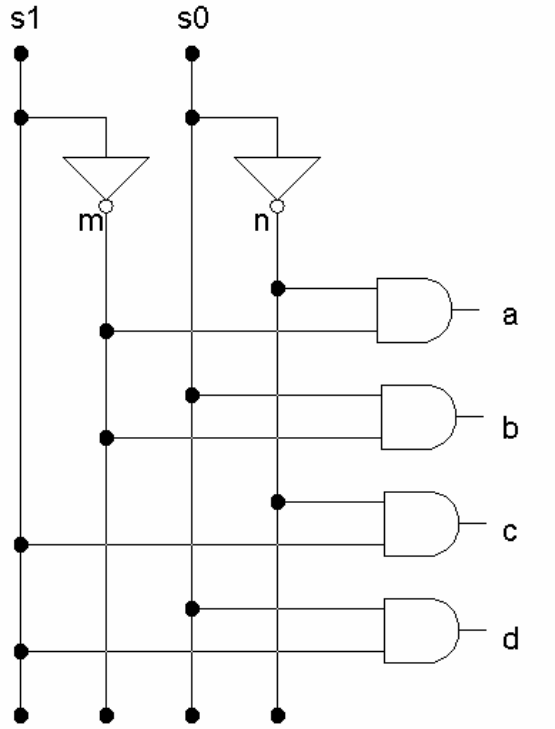
multiplexer2:1

```
module mux2_1(a,b,s,y);
input a,b,s;
output y;
wire p,q,r;
        not n1(p,s);
        and a1(q,a,p);
        and a2(r,s,b);
        or o1(y,q,r);
endmodule
```



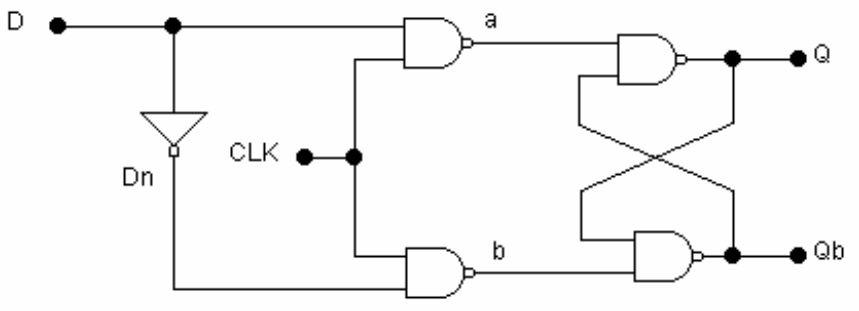
2 : 4 DECODER:

```
module decode24(s1,s0,a,b,c,d);
input s1,s0;
output a,b,c,d;
wire   m,n;
         not   (m,s1);
         not   (n,s0);
         and   (a,n,m);
         and   (b,m,s0);
         and   (c,n,s1);
         and   (d,s0,s1);
endmodule
```



D-FF:

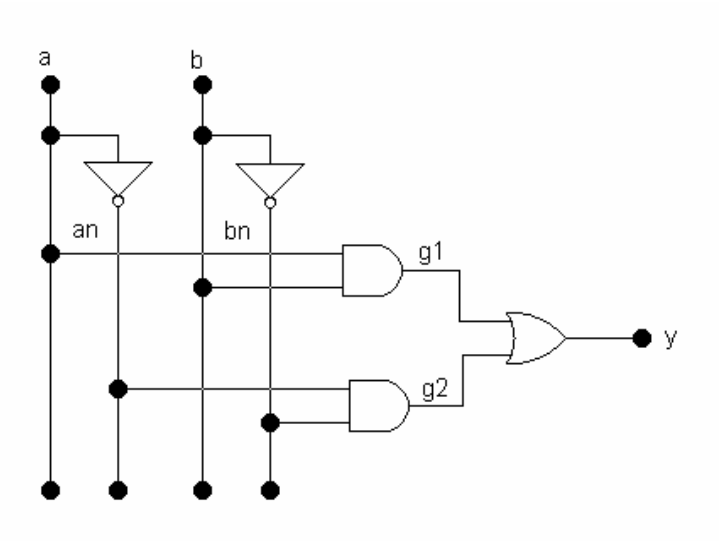
```
module DFF(D,CLK,Q,Qb);
input D,CLK;
inout Q,Qb;
         wire Dn,a,b;
         not    (Dn,D);
         nand  (a,D,CLK);
         nand  (b,Dn,CLK);
         nand  (Qb,b,Q);
         nand  (Q,a,Qb);
       endmodule
```



EQUALITY DETECTOR:

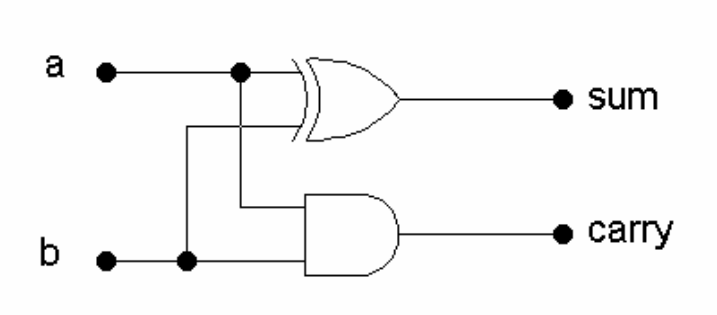
If a and b are equal the output is 1.

```
module equdetect(a,b,y);
input a,b;
wire an,bn,g1,g2;
output y;
        not(an,a);
        not(bn,b);
        and(g1,a,b);
        and(g2,an,bn);
        or(y,g1,g2);
endmodule
```



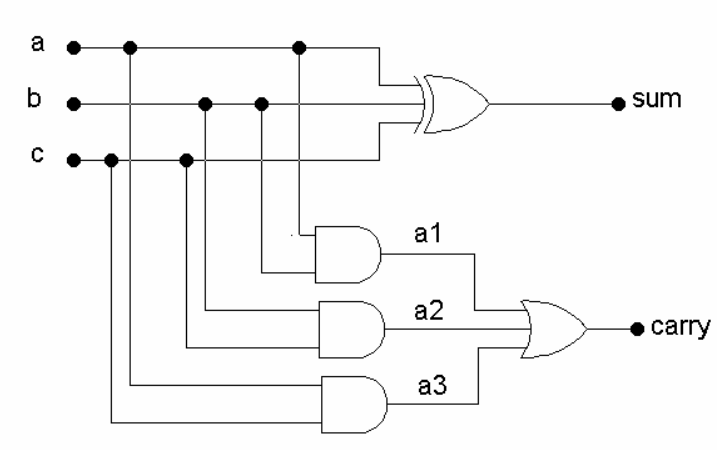
HALFADDER:

```
module halfadd(a,b,sum,carry);
input a,b;
output sum,cout;
and(carry,a,b);
xor(sum,a,b);
endmodule
```



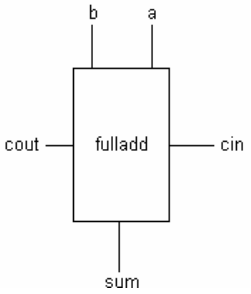
FULLADDER:

```
module fulladd(a,b,c,sum,carry);
input a,b,c;
output sum,carry;
wire a1,a2,a3;
xor(sum,a,b,c);
and(a1,a,b);
and(a2,b,c);
and(a3,c,a);
or(carry,a1,a2,a3);
endmodule
```

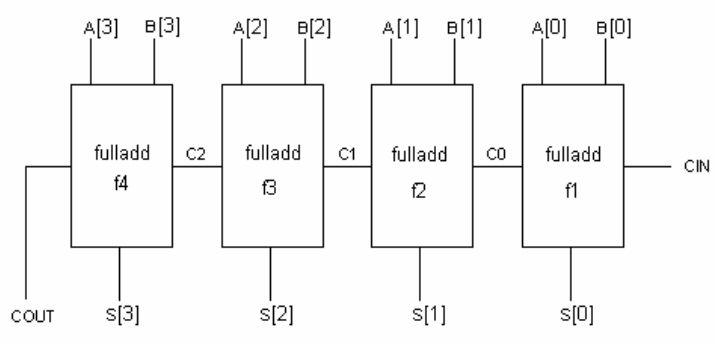


RIPPLE CARRY ADDER:

The four bit ripple carry adder is design using structural modeling as shown.



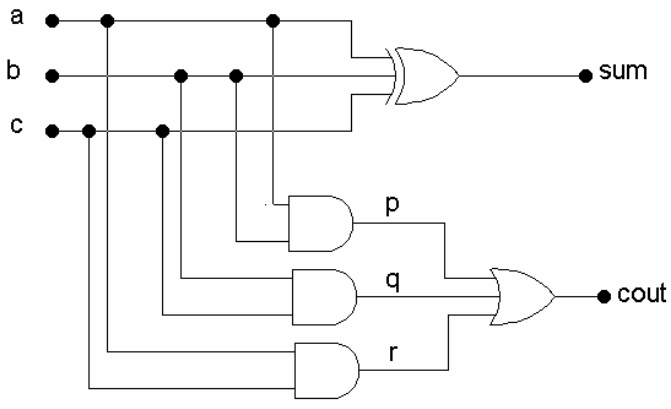

```
module RCA(CIN,A,B,S,COUT);
input[3:0]A,B;
output[3:0]S;
output COUT;
input CIN;
wire C0,C1,C2;
fulladd f1(CIN,A[0],B[0],S[0],C0);
fulladd f2(C0,A[1],B[1],S[1],C1);
fulladd f3(C1,A[2],B[2],S[2],C2);
fulladd f4(C2,A[3],B[3],S[3],COUT);
endmodule
```



In the above example, the four-bit ripple carry adder four-fulladd primitive is instantiated. This fulladd primitive is 1 bit full adder and it must be available in library or can be written and compile together so we have to write the code for fulladd primitive.

fulladd primitive

```
module fulladd(cin, a,b,sum,cout);
input a,b,cin;
output sum,cout;
wire p,q,r;
xor(sum,a,b,cin);
and(p,a,b);
and(q,cin,b);
and(r,a,cin);
or(cout,p,q,r);
endmodule
```



4.7 Switch-Level Modeling:

Verilog can be used to model the logic circuit at the switch level also. the mos transistors are used as switch. The keywords for switch level are: nmos, pmos, cmos, tran, tranif0, tranif1, rnmos, rpmos, rcmos, rtran, rtranif0, rtranif1, supply. **Switch level models are used to allow detailed construction of logical gates and functions and also to allow complex delay modeling to be used.**

MOSFET switches

There are six different transistor models used in Verilog, nmos, pmos and cmos and the corresponding three resistive versions rnmos, rpmos and rcmos. The cmos type of switches have two gates and so have two control signals.

Syntax: **keyword** unique_name (drain. source, gate);

e.g. **nmos** transistor1 (out, gnd, in);
cmos transistor2 (out, gnd, in_n, in_p);

Resistive devices reduce the signal strength which appears on the output by one level.

All the switches only pass signals from source to drain, incorrect wiring of the devices will result in high impedance outputs.

Transmission gates

Transmission gates are bi-directional and can be resistive or non-resistive.

Syntax: keyword unique_name (inout1, inout2, control);

e.g. **tranif0** trans_gate1 (net5, net8, cnt);
rtranif1 rtrans_gate2 (net5, net12, cnt);

Transmission gates tran and rtran are permanently on and do not have a control line. Tran can be used to interface two wires with separate drives, and rtran can be used to weaken signals.

Delays

Real transistors have resolution delays between the input and output. This is modeled in Verilog by specifying one or more delays for the rise, fall, turn-off and turn off time seperated by commas.

Syntax: keyword #(delay{s}) unique_name (node specifications);

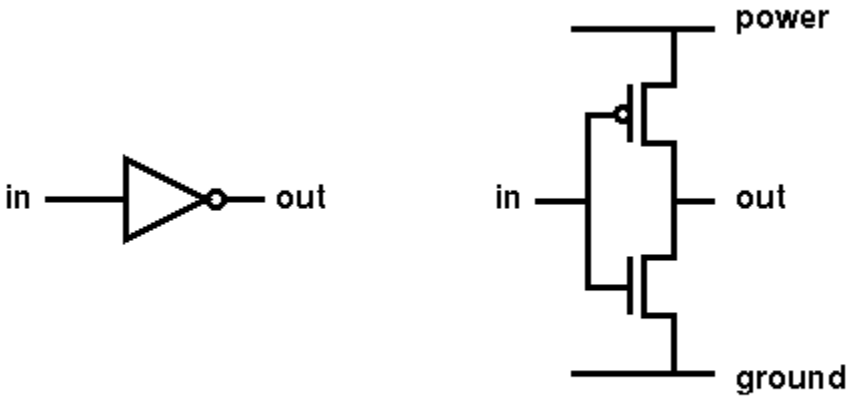
e.g. rnmos #(5,3,10) slow_nmos1 (out, gnd, in);

Note that the transmission gate primitives tran and rtran cannot be delayed because they **never** switch state.

Switch element	Number of delays	Specified delays
Switches	1	Rise, fall and turn-off times of equal length
	2	Rise and fall times
	3	Rise, fall and turn off
(r)tranif0, (r)tranif1	1	both turn on and turn off
	2	turn on, turn off
(r)tran	0	None allowed

This section provides two examples of implementing circuit designs at switch-level.

1. CMOS Inverter



Gate representation Switch–level model

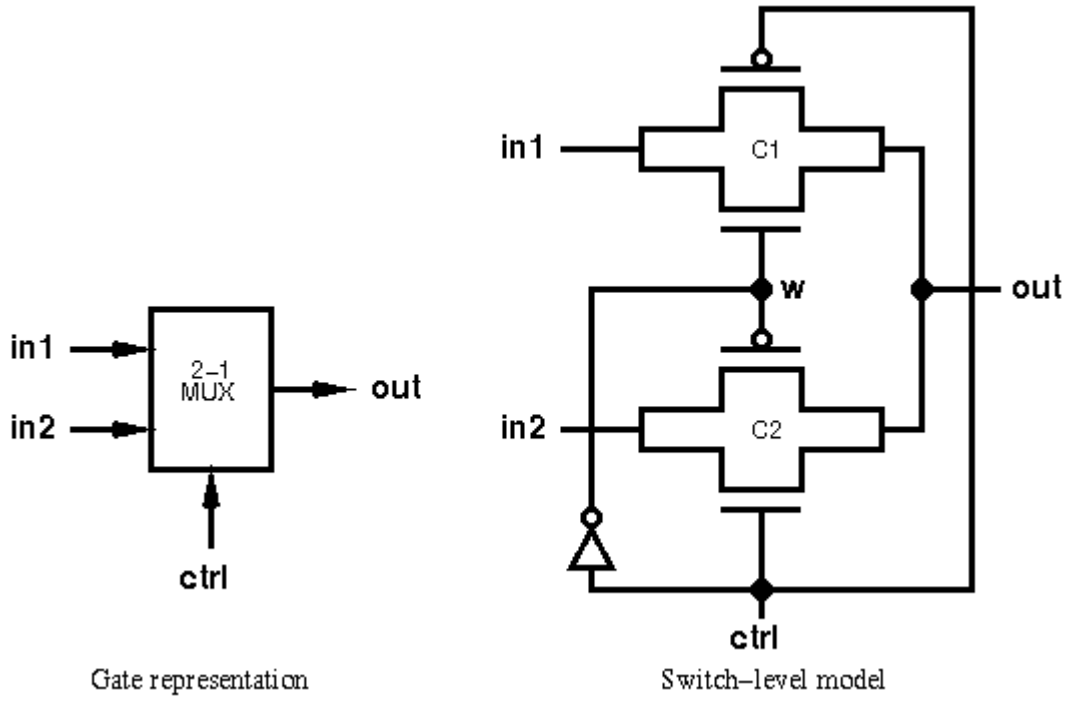
The switch-level description has a similar structure to that of the behavioural, functional or gate level codes, as in this example of a CMOS inverter:

// Switch-level description of a CMOS inverter

```
module inv_sw (out, in);

output  out;                    // inverter output
input   in;                     // inverter input

supply1 power;                  // "power" connected to Vdd
supply0 ground;                 // "ground" connected to Gnd

pmos (out, power, in);          // instantiate pmos switch
nmos (out, ground, in);         // instantiate nmos switch
endmodule
```

Connection to the Vdd power supply is made using the *supply1* net, declared as "power". Similarly, the "ground" terminal is connected to the *supply0* net. Using the *pmos* and *nmos* switch-level primitives the respective transistors are instantiated.

1-bit 2-1 Multiplexer



This circuit assigns the output **out** to either inputs **in1** or **in2** depending on the low or high values of **ctrl** respectively.

// Switch-level description of a 1-bit 2-1 multiplexer

```
// ctrl=0, out=in1; ctrl=1, out=in2

module mux21_sw (out, ctrl, in1, in2);

  output out;                                   // mux output
  input  ctrl, in1, in2;                        // mux inputs
  wire     w;                                   // internal wire

  inv_sw I1 (w, ctrl);                          // instantiate inverter
module

  cmos C1 (out, in1, w, ctrl);                  // instantiate cmos switches
  cmos C2 (out, in2, ctrl, w);
endmodule
```

An inverter is required in the multiplexer circuit, which is instantiated from the previously defined module.

Two transmission gates, of instance names C1 and C2, are implemented with the *cmos* statement, in the format cmos [instancename]([output],[input],[nmosgate],[pmosgate]). The instance name C1,C2 is optional.

4.8 Timing Controls and Delay

The statements within a sequential block are executed in order, but, in the absence of any delay, they all execute at the same simulation time--the current time step. In reality there are delays that are modeled using a timing control.

Timing Control

A **timing control** in verilog can be classified as:

- **Delay control**
- **Event control**.

A **delay control** delays an assignment by a specified amount of time. A **timescale compiler directive** is used to specify the units of time followed by the precision used to calculate time expressions,

```
`timescale 1ns/10ps // Units of time are ns. Round times to 10 ps.
```

Time units may only be s , ns , ps , or fs and the multiplier must be 1, 10, or 100. We can delay an assignment in two different ways:

- Sample the RHS immediately and then delay the assignment to the LHS.(**intra assignment delay**)
- Wait for a specified time and then assign the value of the LHS to the RHS.

Here is an example of the first alternative (an **intra-assignment delay**):

```
x = #1 y;          // intra-assignment delay
```

this assignment assigns the value of y to x by 1 unit time delay but

The second alternative is **delayed assignment**:

```
#1 x = y; // delayed assignment
```

These two alternatives are not the same. The intra-assignment delay is equivalent to the following code:

```
begin          // Equivalent to intra-assignment delay.
hold = y;      // Sample and hold y immediately.
#1;            // Delay.
x = hold;      // Assignment to x. Overall same as x = #1 y.
end
```

In contrast, the delayed assignment is equivalent to a delay followed by an assignment as follows:

```
begin          // Equivalent to delayed assignment.
#1;            // Delay.
x = y;         // Assign y to x. Overall same as #1 x = y.
end
```

```
assign sum= #5 a^b;   \\continious assignment expression for Half adder
assign cout=#5 a&b;
```

in the above example the value a^b is sampled immediately but it is assigned to sum after the delay of 5 unit time. the value assigned to a and b is exactly at the 0 simulation time Now let us see delayed version timing control

```
assign #5 sum=  a^b;
```

in this expression the statement is delayed 5 time unit first so the execution of a^b takes place only after 5 time unit delay and the value assigned to a and b is also at that delayed time.

Hence the expression **assign** sum**= #5** a**^**b**;** and **assign #5** sum**=** a**^**b**;**
Yields completely different value for sum.

Event control:

The other type of timing control, an **event control**, delays an assignment until a specified event occurs. Event is referred as the transition .for example if the clock

pulse is changing from 0 to 1or 1 to 0, than we can say event has occur .this transition can be used to control the timing .for example we may postponed the assignment until there is a transition from 0 to 1 at clock ,this is referred as **posedge** (Positive edge clock) in verilog.the event control timing control uses the keyword **always @** ,and the event name .this indicates that until that particular event happens do not execute the statements in that block.Here is the formal definition:

event_control ::= @ event_identifier | @ (event_expression)
event_expression ::= expression | event_identifier
| **posedge** expression | **negedge expression**
| event_expression **or** event_expression

(Notice there are two different uses of 'or' in this simplified BNF definition--the last one, in bold, is part of the Verilog language, a keyword.) A positive edge (denoted by the keyword posedge) is a transition from '0' to '1' or 'x' , or a transition from 'x' to '1 '. A negative edge (negedge) is a transition from '1' to '0' or 'x' , or a transition from 'x' to '0'. Transitions to or from 'z' do not count. Here are examples of event controls:

```
module delay_controls;
reg X, Y, Clk, Dummy;
always #1 Dummy=!Dummy;          // Dummy clock, just for graphics.

 // Examples of delay controls:
always
begin
#25 X=1;
#10 X=0;
#5;
end


// An event control:

always @(posedge Clk)
Y=X;                    // Wait for +ve clock edge.

always #10 Clk = !Clk;  // The real clock.
initial begin Clk = 0;
  $display("T   Clk X Y");
  $monitor("%2g",$time,,,Clk,,,,X,,Y);
  $dumpvars;
#100 $finish;
end
```

endmodule

```
T   Clk X Y
 0  0    x x
10  1    x x
20  0    x x
25  0    1 x
30  1    1 1
35  1    0 1
40  0    0 1
50  1    0 0
60  0    0 0
65  0    1 0
70  1    1 1
75  1    0 1
80  0    0 1
90  1    0 0
```

The dummy clock in delay controls helps in the graphical waveform display of the results (it provides a one-time-tick timing grid when we zoom in, for example). Figure below shows the graphical output from the Waves viewer in VeriWell (white is used to represent the initial unknown values). The assignment statements to 'X' in the always statement repeat (every 25 + 10 + 5 = 40 time ticks).

.

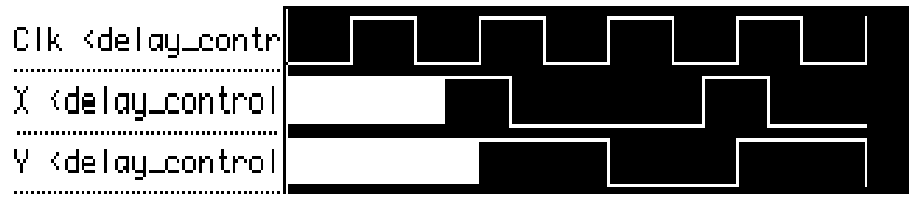


FIGURE Output from the module delay_controls

Events can be declared (as named events), triggered, and detected as follows:

```
module show_event;
reg clock;
always #10 clock = ~ clock;                    // We need a clock.
initial clock = 0;

event event_1, event_2;                        // Declare two named events.
always @(posedge clock) -> event_1;            // Trigger event_1.
always @ event_1
begin
$display("Strike 1!!"); -> event_2; end        // Trigger event_2.
always @ event_2
```

```
 begin
 $display("Strike 2!!");
 $finish;
end                                          // Stop on detection of event_2.
endmodule
```

4.9 Coding style in verilog:

The digital design using verilog hdl can be done in various coding styles. The coding style differs depending upon the complexity of the design. The design style can be of

- Structural modeling,
- Behavioral
- Switch level modeling.

Structural modeling

Structural modeling style defines the architecture of the design. Structural gate level modeling refers to the design model which is decomposed in gate level and the gate primitive are instantiated from the library. These primitives are interconnected to form the circuit. The designer writing codes in this style knows the architecture of the design .for example to design a full adder ,the designer draws the gate level circuit and starts coding it.now for 4 bit adder the designer cascades the 1 bit, 4 adders and defines the structure.in this way large design can be accomplished instantiating the primitives.we have already discuss some of the circuit designs using this style.

Behavioral modeling:

This modeling style describes the function of the design rather than the architecture. The function of the design can be verified in the form of verbal expression (yes,correct,No etc) or in the form truth table .Behavioral modeling can be data flow modeling or procedural modeling. Data flow modeling is used for concurrent operations in combinational logic circuit. Continuous assignments statements (assign) are used for this type of modeling. blocking and non blocking statement (**=** , **<=**) are used . We will discuss blocking and non blocking statements later in this text. The data can me made to flow in registers clocked by clock to perform the pipelined operation, so Register transfer level (RTL) also is part of data flow model.

Procedural modeling:

Basically Hardware description language are designed to handle concurrent activities which models the actual hardware. however there is also a need of sequential operation where the statements are executed sequentially, and in the order they are defined. The ordering of the statements is extremely important for sequential operation.The **sequential operation** in verilog is performed by using procedural assignment which includes **always** block. The statements like **if ,if else, case, loop**,etc are sequential statement and are present inside **always** block.switch level modeling is the lowest abstraction level modeling which uses nmos,pmos and cmos transistor and resistive transistor to function as a switch.the chart below shows the various coding styles in verilog.

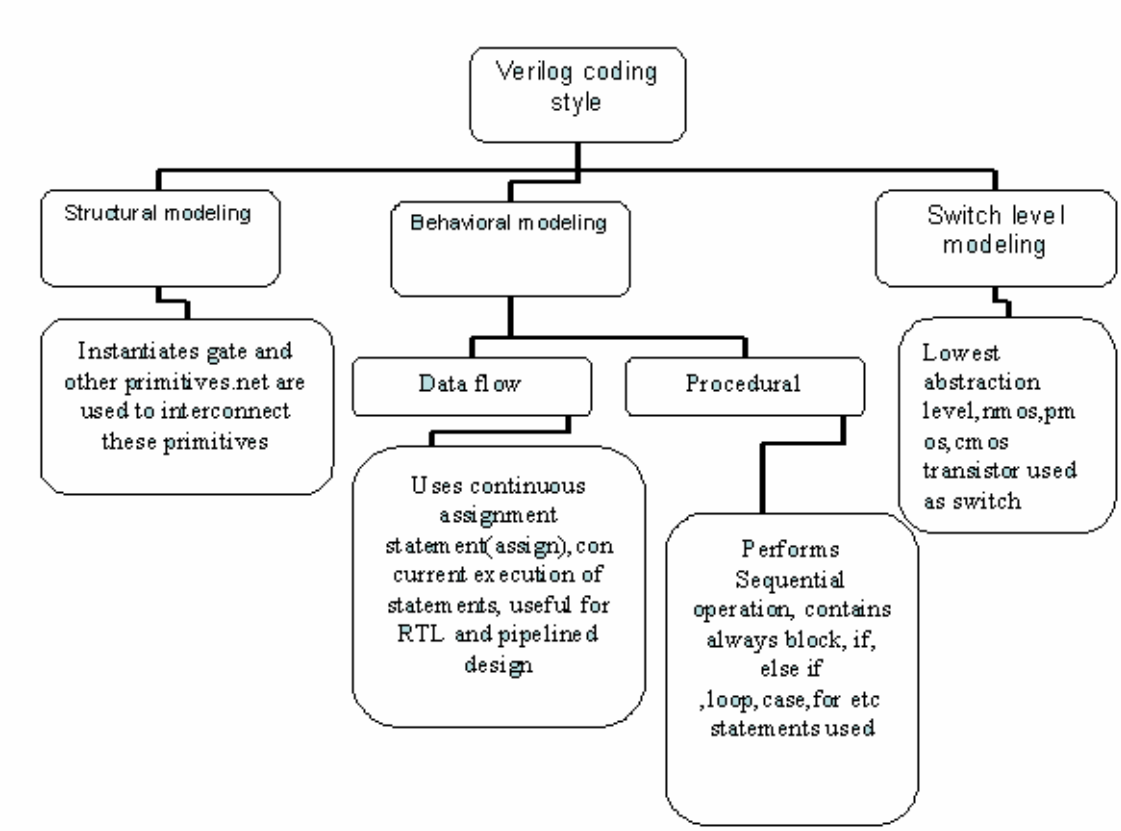


figure: chart showing various coding styles

4.9.1 Continuous assignment statements

These statements are concurrent .concurrent means the statements executes parallel and the ordering of the statement is not important.the right side of the expression is continiously assigned to the left hand side of the expression. the key word assign is used for continuous assignment statements.
For example let us take an example of half adder expression

```
assign sum= (a ^ b);
assign cout=(a & b);
```

The above statement indicates that the sum and carryout (cout) are evaluated concurrently, and the order of the statement does not matter. the statement can be written as

```
assign cout=(a & b);
assign sum= (a ^ b);
```

Now let us see few examples of continuous assignment statements:
Full adder using continuous assignment statements:

```
Module fulladdr(a,b,cin,sum,cout);
Input a,b,cin;
Output sum,cout;
assign sum=a^b^cin;
```

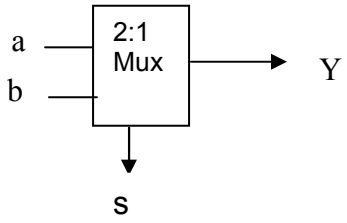
```
assign cout=( a &b) | ( b & cin) |( a & cin);
endmodule
```

Full adder using + operator and concatenation:

```
Module fulladdr(a,b,cin,sum,cout);
Input a,b,cin;
Output sum,cout;
assign {cout,sum} = a + b + cin ;
endmodule
```

2:1 multiplexer:

2 to 1 multiplexer is shown below.if s=0,Y=a else if s=1 ,Y=b;



The code can be written as

```
module mux21(a,b,s,Y);
input a,b,s;
output y;
assign Y=(a&~s) | ( b&s);
endmodule
```

Let us now see the 8-bit adder example using continuous assignment statements:

```
module adder(a, b, ci, sum, co);
input       ci;
input  [7:0] a;
input  [7:0] b;
output [7:0] sum;
output       co;
wire   [8:0] tmp;
assign tmp = a + b + ci;
assign sum = tmp [7:0];
assign co  = tmp [8];
endmodule
```

```
\\ Verilog code for an unsigned 8-bit greater or equal comparator.

module compar(a, b, cmp);
 input  [7:0] a;
```

```
input  [7:0] b;
output      cmp;
assign cmp = (a >= b) ?  1'b1 : 1'b0;
endmodule
```

4.9.2 Procedural Continuous Assignment:

Procedural statements refer to the sequential operation. Sequential statements like if, else if, case, loop, for, while etc are executed inside the procedural block and in the order they are kept. Always block is used to perform the repetitive execution inside the procedural block when any events occurs at the sensitivity list .**initial** block instead of **always** block is used for only one time execution of the procedural block. Remember all the input must be included in sensitivity list and out put must be defined with **reg** variable

The structure of procedural block

always @ (sensitivity list)

begin

If statement…..

Else if……

Case……

For…..

Loop….

end

4.9.3 Blocking and Non Blocking statements:

Procedure assignment can be evaluated in two ways: Blocking and nonblocking assignments. If execution of other statements is blocked after evaluation of right

hand side of a statement until assignment to left hand side is done, it is called blocking statement. That is, if you use a nonblocking statements in your code, right hand side of the statements to be executed without an timing delay in between will be evaluated first, and assigned later. Only if timing delay exists between two nonblocking assignments, assignment to LHS of one statements will be done before evaluation RHS of another statement. In blocking style of assignment, execution will be depending on the order in which statements are written. But in non-blocking statements, result will be independent of order. To understand it more clearly, consider these examples. "=" represents blocking and "<=" represent nonblocking in Verilog.

We will try to swap values in registers a and b, assume the value of a and b before the clock is **a = 0 and b = 1.**

```
always@ (posedge clk)
begin
   a = b;
   b = a;
end
// (1) both a and b becomes 1.
```

```
always@ (posedge clk)
begin
   a <= b;
   b <= a;
end
//(2)  a becomes 1 and b becomes 0.
```

Here, code (1) do not give us required result. Because when a=b is executed until b is assigned to a it doesn't execute next statement b=a.now when the first assignment is over the old value of a is lost and next statement b=a gets the value of b,so no swapping is done.

In code 2 both the statement is executed concurrently such that the current value of a and b are sampled and stored temporarily .now when the assignment is done the values are restored and the swapping operation is performed.

A **procedural continuous assignment statement** (sometimes called a quasicontinuous assignment statement) is a special form of the assign statement

that we use within a sequential block. For example, the following flip-flop model assigns to q depending on the clear, clr, and preset, pre, inputs.

```
module dff_clr_pre(q, d ,clear ,preset , clock);
output q;
 input d,clear,preset,clock;
reg q;
always @(clear or preset)
  If (clear) assign q = 0;                    // active-high clear
  else if(preset) assign q = 1;               // active-high preset
  else deassign q;
always @(posedge clock) q = d;
endmodule
```

We have now seen all of the different forms of Verilog assignment statements. The following skeleton code shows where each type of statement belongs:

```
module all_assignments

assignment (assign) statements          //... continuous assignments.

always                                  // beginning of procedure
  begin                                 // beginning of sequential block
  //... blocking procedural assignments.
  //... nonblocking procedural assignments.
  //... procedural continuous assignments.
  end
endmodule
```

Summary of different types of statement:

Verilog assignment statements.				
Type of Verilog assignment	**Continuous assignment statement**	**Procedural assignment statement**	**Nonblocking procedural assignment statement**	**Procedural continuous assignment statement**
Where it can occur	outside an always or initial statement, task, or function	inside an always or initial statement, task, or function	inside an always or initial statement, task, or function	always or initial statement, task, or function
Example	**wire** [31:0] DataBus; **assign** DataBus = Enable ? Data : 32'bz	**reg** Y; **always** @(posedge clock) Y = 1;	**reg** Y; **always** Y <= 1;	**always** @(Enable) **if**(Enable) **assign** Q = D; **else deassign** Q;

Valid LHS of assignment	net	register or memory element	register or memory element	net
Valid RHS of assignment	<expression> net, reg or memory element	<expression> net, reg or memory element	<expression> net, reg or memory element	<expression> net, reg or memory element

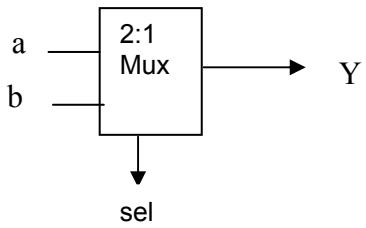
4.10 Sequential statements:

Now we shall see individual sequential statement and their uses.following that we will see few examples which are behavioral modeling styles.

If statement

If statement
The *if* statement in Verilog is a sequential statement that conditionally executes other sequential statements, depending upon the value of some condition. An if statement may optionally contain an *else* part, executed if the condition is false. Although the else part is optional, for the time being, we will code up if statements with a corresponding else rather than simple if statements. In order to have more than one sequential statement executed in an if statement, multiple statements are bracketed together using the begin..end keywords,
Let us take an example of 2:1 multiplexer to describe the if statement function

```
module mux2_1(a,b,sel,Y);
input a,b,sel;
output Y;
reg Y;
always @(sel or a or b)
begin
  if (sel == 1)
   Y = b;
else
Y = a;
end
endmodule
```



note**:** *To synthesize combinational logic using an always block, all inputs to the design must appear in the sensitivity list.*
If statements can be nested if you have more complex behavior to describe: here is a random example for nested if statement.

```
reg f, g;
always @(sel or sel_2 or a or b)
  if (sel == 1)
    begin
      f = a;
      if (sel_2 == 1)
        g = ~a;
      else
        g = ~b;
    end
  else
    begin
      f = b;
      if (sel_2 == 1)
        g = a & b;
      else
        g = a | b;
    end
```

Notice that the code is beginning to look a little bit confusing! In the code above, begin. .end blocks have only be used where they *must* be used, that is, where we have multiple statements. It is probably a good idea to use begin..end blocks throughout your Verilog code - you end up typing in a bit more Verilog but it's easier to read. Also, if you have to add more functionality to an always block later on (more sequential statement), at least the begin..end block is already in place.

Case statement:

Case statement is used instead of multiple if ,else if statement. The syntax for case statement is

Case(selection expression)

Selection option1: statement;

Selection option2: statement;

;

;

default :statement

endcase

example

Verilog Code for a 4-to- 1 MUX using a Case statement.

```
module mux (a, b, c, d, s, o);
input       a, b, c, d;
input  [1:0] s;
output      o;
reg         o;
always @(a or b or c or d or s)
begin
  case (s)
    2'b00   : o = a;
    2'b01   : o = b;
    2'b10   : o = c;
    default : o = d;
  endcase
        end
    endmodule
```

Loop statements:

Loop statements are used to control repeated execution of one or more statements. There are 4 types of looping statements in Verilog:these are

forever statement;

repeat (expression) statement;

while (expression) statement;

for (initial_assignment; expression; step_assignment) statement;

We can combine more than one statements using begin -- end block in a looping instruction. Looping statements should be used within a procedural block.

forever Loop:

The forever instruction continuously repeats the statement that follows it. Therefore, it should be used with procedural timing controls (otherwise it hangs the simulation). Consider this example:

```
initial
begin
  clk = 0;
```

```
  forever #5 clk = ~clk;
end
```

repeat Loop:

Repeats the following instruction for specified times. The number of executions is set by the expression or constant value. If expression evaluates to high impedance or un-known, then statement will not be executed.

```
initial
begin
  x = 0;
  repeat( 16 )
  begin
    #2 $display("y= ", y);
    x = x + 1;
  end
end
```

while Loop:

while loop repeats the statement until the expression returns true. If starts with false value, high impedance or unknown value, statement will not be executed.

```
initial
begin
  x = 0;
  while( x <= 10 )
  begin
    #2 $display("y= ", y);
    x = x + 1;
  end
end
```

for Loop:

Executes initial_assignment once when the loop starts, Executes the statement or statement group as long as the expression evaluates as true and executes the step_assignment at the end of each pass through the loop.

for(initial_assignment; expression; step_assignment) statement;

Syntax is similar to C language except that **begin--end** is used instead of **{--}** to combine more than one statements. Remember that we don't have ++ operator in Verilog.

```
for( i = 0; i <= 10; i++ )
  mem[i] = 0;
```

Some examples of procedural verilog codes:

Verilog code for an unsigned 8-bit adder/subtractor.

```
module addsub(a, b, oper, res);
input       oper;
input  [7:0] a;
input  [7:0] b;
output [7:0] res;
reg    [7:0] res;
always @(a or b or oper)
begin
  if (oper == 1'b0)
    res = a + b;
  else
    res = a - b;
   end

endmodule
```

Verilog code for a 4-to-1 MUX using an If statement.

```
module mux (a, b, c, d, s, o);
input       a,b,c,d;
input  [1:0] s;
output       o;
reg          o;
always @(a or b or c or d or s)
begin
  if (s == 2'b00)
    o = a;
  else if (s == 2'b01)
    o = b;
  else if (s == 2'b10)
    o = c;
  else
    o = d;
end
    endmodule
```

Verilog Code for a 4-to- 1 MUX using a Case statement.

```
module mux (a, b, c, d, s, o);
input       a, b, c, d;
input  [1:0] s;
output       o;
```

```
reg         o;
always @(a or b or c or d or s)
begin
  case (s)
    2'b00   : o = a;
    2'b01   : o = b;
    2'b10   : o = c;
    default : o = d;
  endcase
        end
    endmodule
```

Verilog code for a 3-to-8 decoder.

```
    module dec3to8 (sel, res);
    input  [2:0] sel;
    output [7:0] res;
    reg    [7:0] res;
    always @(sel or res)
    begin
      case (sel)
        3'b000  : res = 8'b00000001;
        3'b001  : res = 8'b00000010;
        3'b010  : res = 8'b00000100;
        3'b011  : res = 8'b00001000;
        3'b100  : res = 8'b00010000;
        3'b101  : res = 8'b00100000;
        3'b110  : res = 8'b01000000;
        default : res = 8'b10000000;
      endcase
    end
    endmodule
```

simple d flipflop:

```
module flop (clk, d, q);
input  clk, d;
output q;
reg    q;

always @(posedge clk)
begin
  q <= d;
end
    endmodule
```

Verilog code for a flip-flop with a negative-edge clock and asynchronous clear.

```
module flop (clk, d, clr, q);
input  clk, d, clr;
output q;
reg    q;
always @(negedge clk or posedge clr)
    begin
  if (clr)
     q <= 1'b0;
  else
     q <= d;
end
    endmodule
```

verilog code for the flip-flop with a positive-edge clock and synchronous set.

```
    module flop (clk, d, s, q);
    input  clk, d, s;
    output q;
    reg    q;
    always @(posedge clk)
    begin
      if (s)
        q <= 1'b1;
      else
        q <= d;
    end
    endmodule
```

Verilog code for a latch with a positive gate and an asynchronous clear.

```
    module latch (g, d, clr, q);
    input  g, d, clr;
    output q;
    reg    q;
always @(g or d or clr)
    begin
      if (clr)
        q <= 1'b0;
      else if (g)
        q <= d;
    end
    endmodule
```

Verilog code for a 4-bit latch with an inverted gate and an asynchronous preset.

```
module latch (g, d, pre, q);
input       g, pre;
input  [3:0] d;
output [3:0] q;
reg    [3:0] q;
always @(g or d or pre)
begin
  if (pre)
    q <= 4'b1111;
  else if (~g)
    q <= d;
end
endmodule
```

Verilog code for a 3-bit 1-of-9 Priority Encoder.

```
module priority (sel, code);
input  [7:0] sel;
output [2:0] code;
reg    [2:0] code;
always @(sel)
begin
  if (sel[0])
    code = 3'b000;
  else if (sel[1])
    code = 3'b001;
  else if (sel[2])
    code = 3'b010;
  else if (sel[3])
    code = 3'b011;
  else if (sel[4])
    code = 3'b100;
  else if (sel[5])
    code = 3'b101;
  else if (sel[6])
    code = 3'b110;
  else if (sel[7])
    code = 3'b111;
  else
    code = 3'bxxx;
end
    endmodule
```

Verilog code for a logical shifter.

```
module lshift (di, sel, so);
    input  [7:0] di;
input  [1:0] sel;
output [7:0] so;
reg    [7:0] so;
always @(di or sel)
begin
  case (sel)
    2'b00   : so = di;
    2'b01   : so = di << 1;
    2'b10   : so = di << 2;
    default : so = di << 3;
  endcase
end
    endmodule
```

Verilog Summary:

Verilog feature	Example

Comments	a = 0; // comment ends with newline
	/* This is a multiline or block comment */

Constants: string and numeric	**parameter** BW = 32 // local, use BW `define G_BUS 32 // global, use `G_BUS 4'b2 1'bx
Names (case-sensitive, start with letter or '_')	_12name A_name $BAD NotSame notsame
Two basic types of logic signals: wire and reg	**wire** myWire; **reg** myReg;
Use a continuous assignment statement with wire	**assign** myWire = 1;
Use a procedural assignment statement with reg	**always** myReg = myWire;
Buses and vectors use square brackets	**reg** [31:0] DBus; DBus[12] = 1'bx;
We can perform arithmetic on bit	**reg** [31:0] DBus; DBus = DBus + 2;

vectors	
Arithmetic is performed modulo 2^n	**reg** [2:0] R; R = 7 + 1; // now R = 0
Operators: as in C (but not **++** or **- -**)	
Fixed logic-value system	1, 0, x (unknown), z (high-impedance)
Basic unit of code is the module	**module** bake (chips, dough, cookies); **input** chips, dough; **output** cookies; assign cookies = chips & dough; **endmodule**
Ports	input or input/output ports are wire output ports are wire or reg

Procedures model things that happen at the same time
and may be sensitive to an edge,
posedge, **negedge**,
or to a level.
	always @rain sing; **always** @rain dance;
	always @(**posedge** clock) D = Q; // flop
	always @(a **or** b) c = a & b; // and gate

Sequential blocks model repeating things: **always**: executes forever **initial**: executes once only at start of simulation	**initial** born; **always** @alarm_clock **begin** : a_day metro=commute; bulot=work; dodo=sleep; **end**
Functions and tasks	**function** ... **endfunction** **task** ... **endtask**
Output	$display("a=%f",a);$dumpvars;$monitor(a)
Control simulation	$stop; $finish // sudden or gentle halt
Compiler directives	`timescale 1ns/1ps // units/resolution

Delay	#1 a = b; // delay then sample b
	a = #1 b; // sample b then delay

3.10 VERILOG QUICK RECALL;;;

Hardware Description Language (HDL)

- What is the need for Hardware Description Language?
- Model, Represent, And Simulate Digital Hardware
  - Hardware Concurrency
  - Parallel Activity Flow
  - Semantics for Signal Value And Time
- Special Constructs And Semantics
  - Edge Transitions
  - Propagation Delays
  - Timing Checks

VERILOG HDL

- Basic Unit – A module
- Module
  - Describes the functionality of the design
  - States the input and output ports
- Example: A Computer
  - Functionality: Perform user defined computations
  - I/O Ports: Keyboard, Mouse, Monitor, Printer

Verilog module

General definition

```
module module_name ( port_list );
    port declarations;
    ...
    variable declaration;
    ...
    description of behavior
endmodule
```

Example

```
module HalfAdder (A, B, Sum Carry);
input A, B;
output Sum, Carry;
assign Sum = A ^ B;
//^ denotes XOR
assign Carry = A & B;
// & denotes AND
endmodule
```

LEXICAL CONVENTIONS

- **Comments**

```
// Single line comment
/* Another single line comment */
/* Begins multi-line (block) comment
   All text within is ignored
  Line below ends multi-line comment
*/
```

- **Number**

  decimal, hex, octal, binary
  unsized decimal form
  size base form
  include underlines, +,-

- **String**

  " Enclose between quotes on a single line"

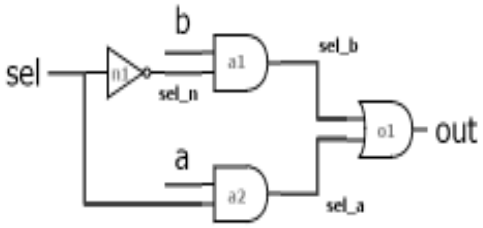
- **Identifier**
  - A ... Z
  - a ... z
  - 0 ... 9
  - Underscore
- Strings are limited to 1024 chars
- First char of identifier must not be a digit
- **Keywords**: See text.
- **Operators**: See text.

Verilog is case sensitive

VERILOG DESCRIPTION STYLE

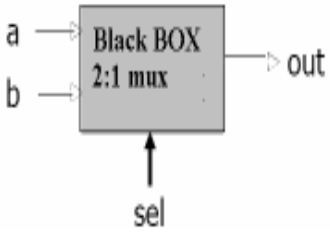
- **Structural**: Logic is described in terms of Verilog gate primitives
- Example:

```
not  n1(sel_n, sel);
and a1(sel_b, b, sel_b);
and a2(sel_a, a, sel);
or   o1(out, sel_b, sel_a);
```



- **Behavioral**: Algorithmically specify the behavior of the design
- Example:

```
if (select == 0) begin
    out = b;
end
else if (select == 1) begin
    out = a;
end
```



DATA FLOW MODELLING

- Uses continuous assignment statement
  - Format: assign [delay] net = expression;
  - Example: assign sum = a ^ b;
- **Delay**: Time duration between assignment from RHS to LHS
- All continuous assignment statements execute concurrently
- Order of the statement does not impact the design

- Delay can be introduced
  - Example: assign #2 sum = a ^ b;
  - "#2" indicates 2 time-units
  - No delay specified : 0 (default)
- Associate time-unit with physical time
  - `timescale time-unit/time-precision
  - Example: `timescale 1ns/100 ps
- Timescale

  `timescale 1ns/100ps
  - 1 Time unit = 1 ns
  - Time precision is 100ps (0.1 ns)
  - 10.512ns is interpreted as 10.5ns

Example:

```
`timescale 1ns/100ps
module HalfAdder (A, B, Sum, Carry);
  input A, B;
  output Sum, Carry;
  assign #3 Sum = A ^ B;
  assign #6 Carry = A & B;
endmodule
```

Behavioral Modelling

- **Example:**

```
module mux_2x1(a, b, sel, out);
  input a, a, sel;
  output out;
  always @(a or b or sel)
  begin
   if (sel == 1)
      out = a;
   else out = b;
  end
endmodule
```

Sensitivity List

- always statement : Sequential Block
- Sequential Block: All statements within the block are executed sequentially
- When is it executed?
  - Occurrence of an event in the sensitivity list
  - Event: Change in the logical value
- Statements with a Sequential Block: Procedural Assignments
- Delay in Procedural Assignments
  - Inter-Statement Delay
  - Intra-Statement Delay

- Inter-Assignment Delay
  - Example:

```
Sum = A ^ B;
#2 Carry = A & B;
```

  - Delayed execution
- Intra-Assignment Delay
  - Example:

```
Sum = A ^ B;
Carry = #2 A & B;
```

  - Delayed assignment

- Two Procedural Constructs
  - initial Statement
  - always Statement
- initial Statement : Executes only once
- always Statement : Executes in a loop
- Example:

```
...
initial begin
  Sum = 0;
  Carry = 0;
end
...
```

```
...
always @(A or B) begin
  Sum = A ^ B;
  Carry = A & B;
end
...
```

- Event Control
  - Edge Triggered Event Control
  - Level Triggered Event Control
- Edge Triggered Event Control

```
@ (posedge CLK) //Positive Edge of CLK
  Curr_State = Next_state;
```

@ negedge	@ posedge
1 → x	0 → x
1 → z	0 → z
1 → 0	0 → 1
x → 0	x → 1
z → 0	z → 1

- Level Triggered Event Control

```
@ (A or B) //change in values of A or B
  Out = A & B;
```

- Loop Statements
  - Repeat
  - While
  - For
- Repeat Loop
  - Example:

```
repeat (Count)
  sum = sum + 5;
```

  - If condition is a x or z it is treated as 0
- While Loop
  - Example:

```
while (Count < 10) begin
  sum = sum + 5;
  Count = Count +1;
end
```

  - If condition is a x or z it is treated as 0
- For Loop
  - Example:

```
for (Count = 0; Count < 10; Count = Count + 1) begin
  sum = sum + 5;
end
```

- if Statement
- Format:

```
if (condition)
   procedural_statement
else if (condition)
   procedural_statement
else
   procedural_statement
```

- Example:

```
if (Clk)
   Q = 0;
else
   Q = D;
```

- Case Statement
- Example 1:

```
case (X)
  2'b00: Y = A + B;
  2'b01: Y = A – B;
  2'b10: Y = A / B;
endcase
```

- Example 2:

```
case (3'b101 << 2)
  3'b100: A = B + C;
  4'b0100: A = B – C;
  5'b10100: A = B / C; //This statement is executed
endcase
```

- Net Types: Physical Connection between structural elements
- Register Type: Represents an abstract storage element.
- Default Values
  - Net Types : z
  - Register Type : x
- Net Types: wire, tri, wor, trior, wand, triand, supply0, supply1
- Register Types : reg, integer, time, real, realtime

- Net Type: Wire

  wire [msb : lsb] wire1, wire2, ...

  - Example

```
wire Reset; // A 1-bit wire
wire [6:0] Clear; // A 7-bit wire
```

- Register Type: Reg

  reg [msb : lsb] reg1, reg2, ...

  - Example

```
reg [ 3: 0 ] cla; // A 4-bit register
reg cla; // A 1-bit register
```

MEMORY MODELLING

- An array of registers

```
reg [ msb : lsb ] memory1 [ upper : lower ];
```

- Example

```
reg [ 0 : 3 ] mem [ 0 : 63 ];
// An array of 64 4-bit registers
reg mem [ 0 : 4 ];
// An array of 5 1-bit registers
```

RESTRICTION ON DATA TYPES

- Data Flow and Structural Modeling
  - Can use only ***wire*** data type
  - Cannot use ***reg*** data type
- Behavioral Modeling
  - Can use only ***reg*** data type (within initial and always constructs)
  - Cannot use ***wire*** data type

CHAPTER 5

- Mos transistor as a switch
- Basic logic design in cmos137
- How to realize gate structure with cmos logic

- Pass transistor
- D-Latch
- D-Flip Flop
- Programmable interconnect (antifuse)

- ASIC Design Flow
  - Types of ASIC
  - Full custom asic
  - Semicustom Asic
  - Standard cell base ASIC
  - Channel,channelless, structural gate array ASIC

  - Programmable logic devices
  - Xilinx FPGA

CHAPTER 5

The aim of this chapter is to make readers familiar with logic design with Cmos logic and introduction to application specific CHAPTER 6integrated circuit design.

5.1 Mos transistor as a switch:

Mos transistors can be used as switch. the figure below shows how nmos and pmos transistor acts as open and close switch .nmos transistor acts as open switch when low logic level “0” is applied at the gate terminial,and acts as close switch when high logic “1” is applied to the gate terminial.pmos transistor operates just opposite to that. So we can conclude that nmos transistor conducts when its gate terminal is at high logic and pmos transistor conducts when low logic is at the gate terminal.

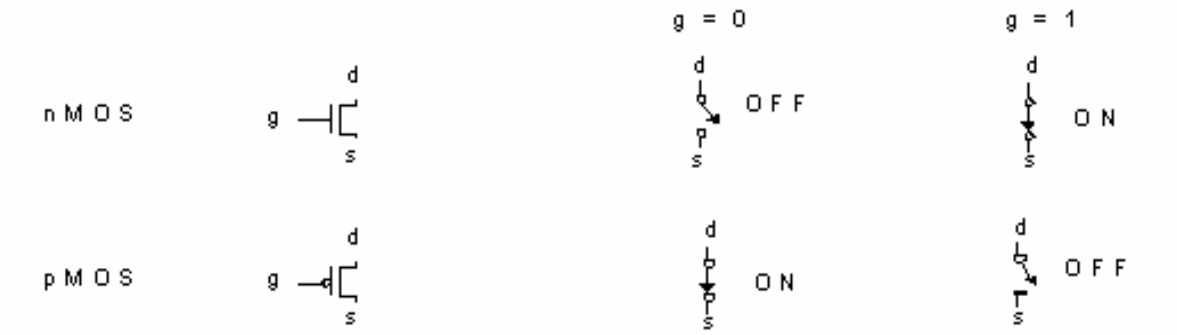


Figure 4.0 nmos and pmos transistors as switch

5.2 Basic logic gates in cmos:

Using the above property of the nmos and pmos transistor it is possible to construct various gate functions. Cmos uses both nmos and pmos transistor to realize gate function and other logic structure. The figure below shows the cmos inverter constructed from cmos logic. Refer figure 4.1.

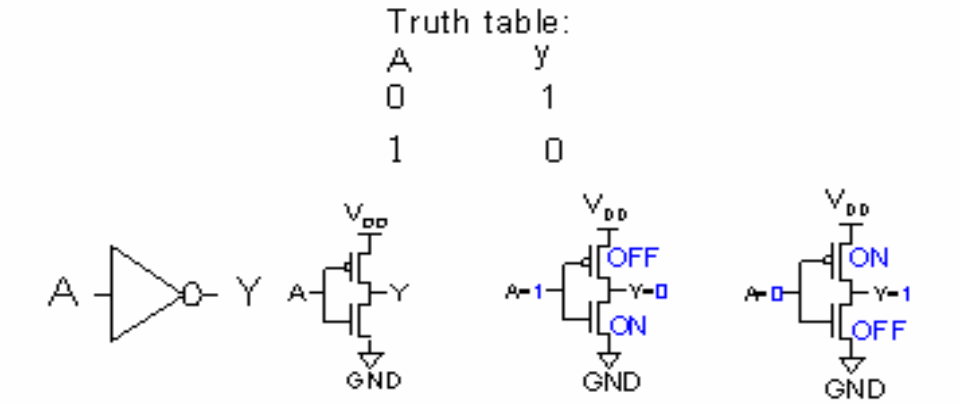


Figure 4.1 cmos inverter

5.2 .1 HOW TO REALIZE GATE STRUCTURE WITH MOS TRANSISTORS

Cmos design uses both pmos and nmos transistor to realize the gate structure. Each input corresponds to two transistors, one for nmos and one for pmos.nmos transistors forms pull down network and pmos transistors forms pull up network. pull down means it is the path to ground and results in low level logic at the output.pullup network connects to the supply and results in logic high at the output.
Refer figure 4.3

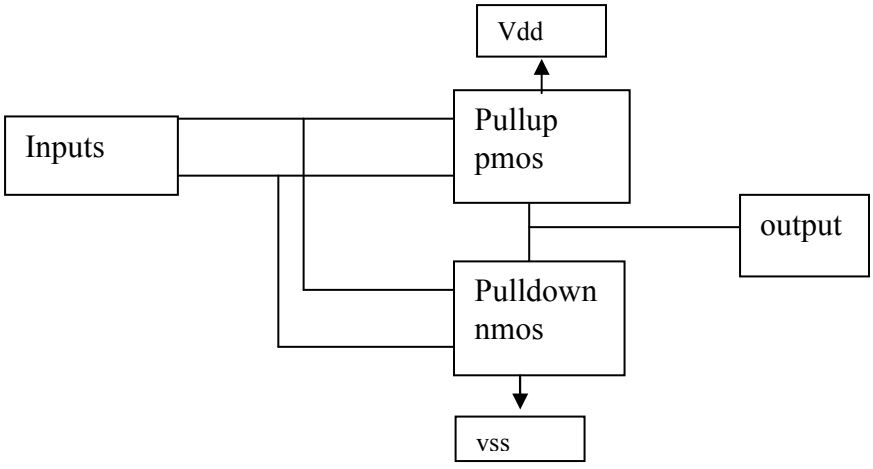


Figure 4.3 cmos logic structure

Here is the simple rule how you can draw the cmos logic structure;

For **and, nand** type of operation nmos transistors are connected in series and pmos transistors are connected in parallel.
For **or, nor** structure nmos transistors are connected in parallel and pmos transistors are connected in series.
For example for two input nand gate nmos transistors (pull down network) are connected in series and pmos transistors (pull up network) are connected in parallel. For nor gate it is vice versa. refer to the figure below.

Cmos Nand gate:

To construct 2 input cmos nand gate first find out the combination for nmos and pmos transistor.as mention earlier the nmos transistor are connected in series and pmos transistors are connected in parallel. The output is then connected through the junction of nmos and pmos network. when the input a and b both are “0” both pmos transistor conducts and output is pulled up or is high because pull up network is tied to vdd.when both the input are at high logic”1” nmos both nmos transistor conducts and connects the output to low logic as pulldown network is connected to ground.similarly remaining combination is shown in figure 4.4.

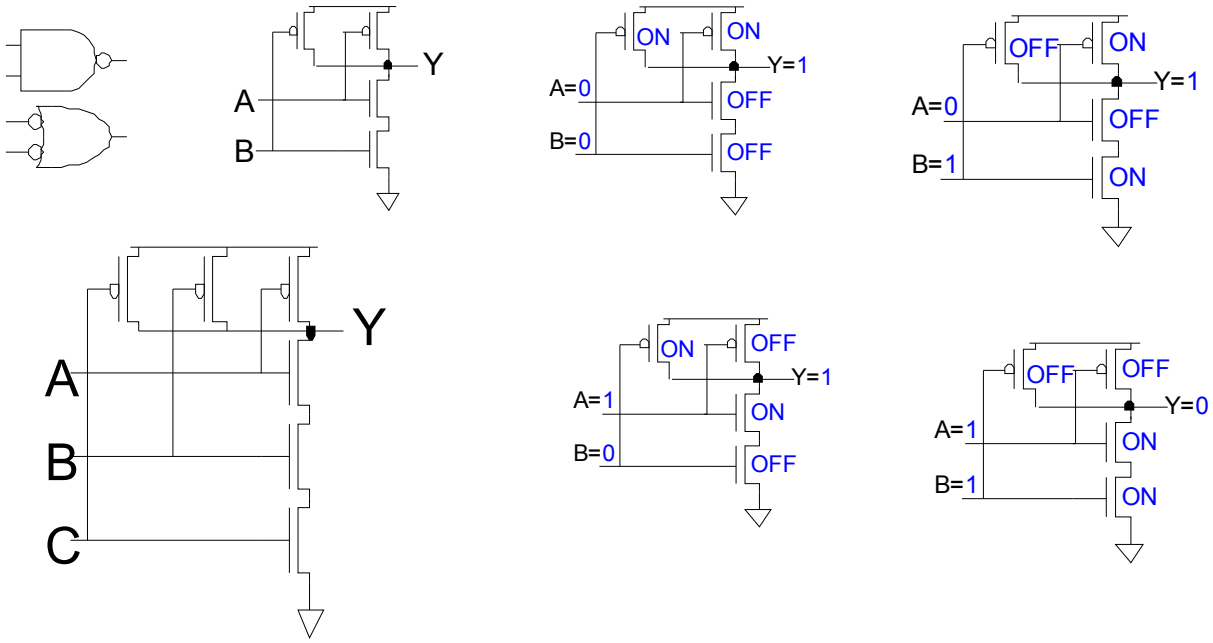


Figure 4.4 two and three input nand gate

CMOS NOR gate:
Similarly for nor gate nmos network is connected in parallel and pmos network in series. Applying similar approach readers are encouraged to verify the truth table for nor gate. Figure 4.5 below shows the two and three input nor gate.

Nor gate:

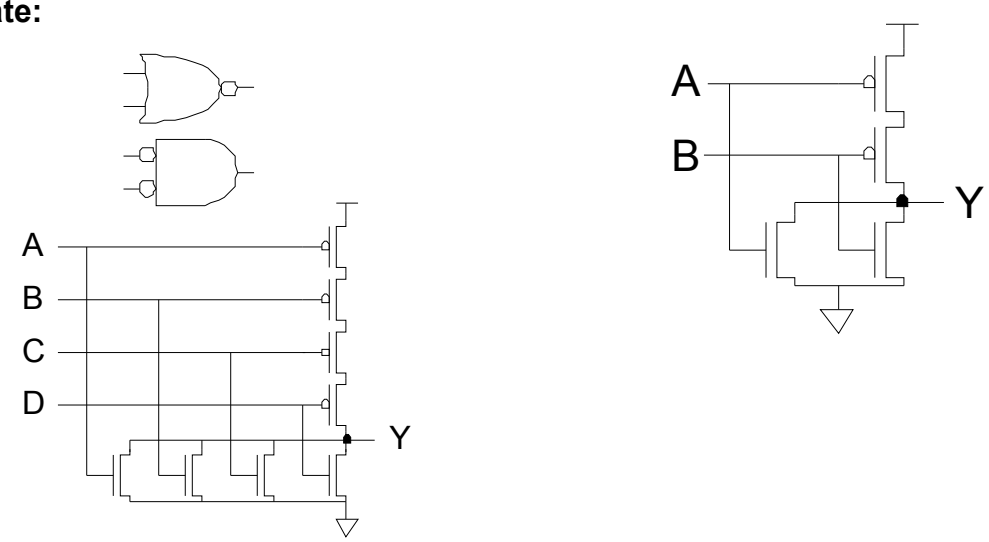


Figure 4.5 cmos two and three input nor gate.

CMOS Compound gates:

The above rules can be extended to form complex gates beside simple gates as described earlier. The example below shows the construction of and or invert structure (AOI).

$$Y = \overline{(A \bullet B) + (C \bullet D)}$$

First, determine the nmos and pmos network. Since input A is and with B so nmos transistor in series, likewise C and D is also (refer figure a) and so in series. now this structure are or hence they should be connected in parallel (figure b).now next for pmos it is just opposite to that of nmos (figure c and d).Now this nmos and pmos structure are combined together(figure 4.6) to get the desire function..

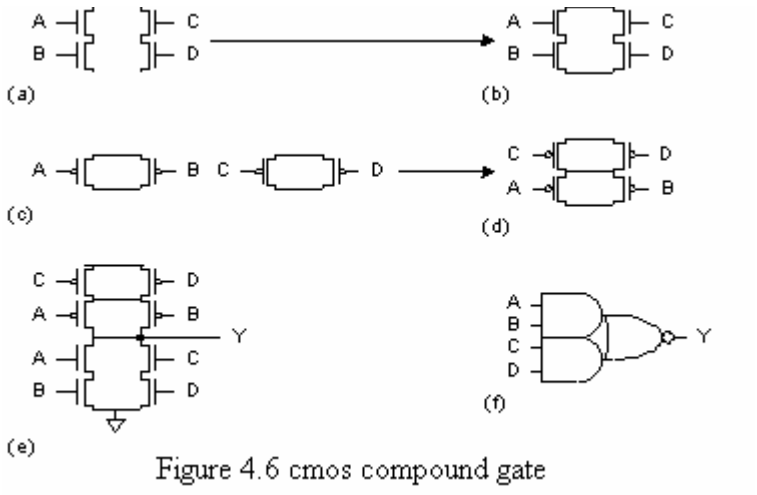


Figure 4.6 cmos compound gate

Some Examples of the compound gate structures Using Cmos Logic.

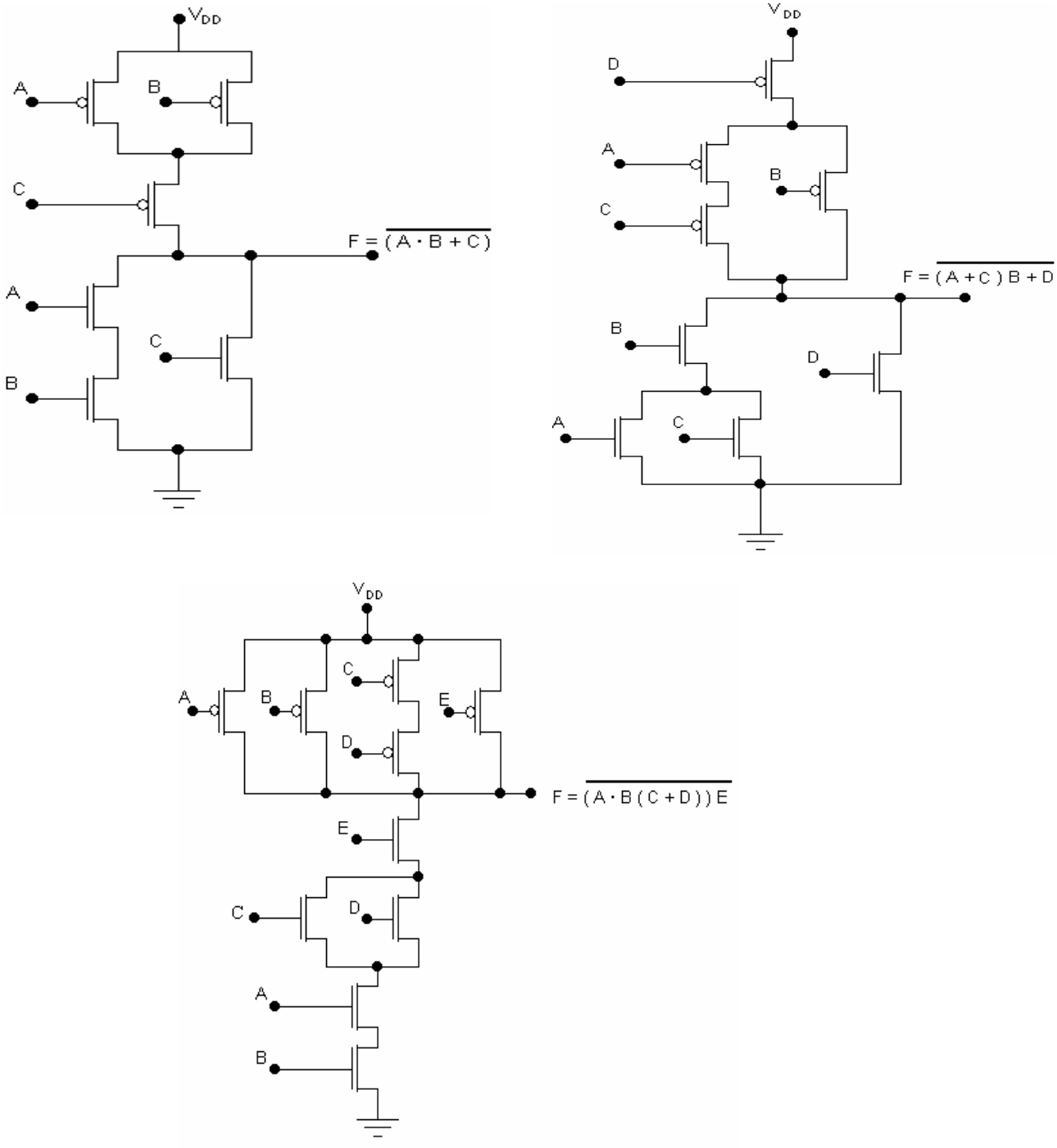


Figure 4.6 cmos compound gate structures

5.3 PASS TRANSISTOR:

nmos and pmos transistors can be used to transmit the logic level from input to output hence the name pass transistor.nmos transistor degrades the high logic value by an amount Vt (Threshold voltage) i.e out put is Vdd-Vtn at the output due to threshold voltage but passes logic zero without degradation so it is only suitable to pass logic zero not for logic 1. nmos transistor passes weak one and strong 0.similarly pmos transistor passes logic 1 as good and degrades logic 0 .now the question arises what if we want to pass both the logic 0 and 1 as good , the solution for this is if we combine both the transistor in parallel we are able to get good logic for both the logic. This combination is termed as transmission gate.

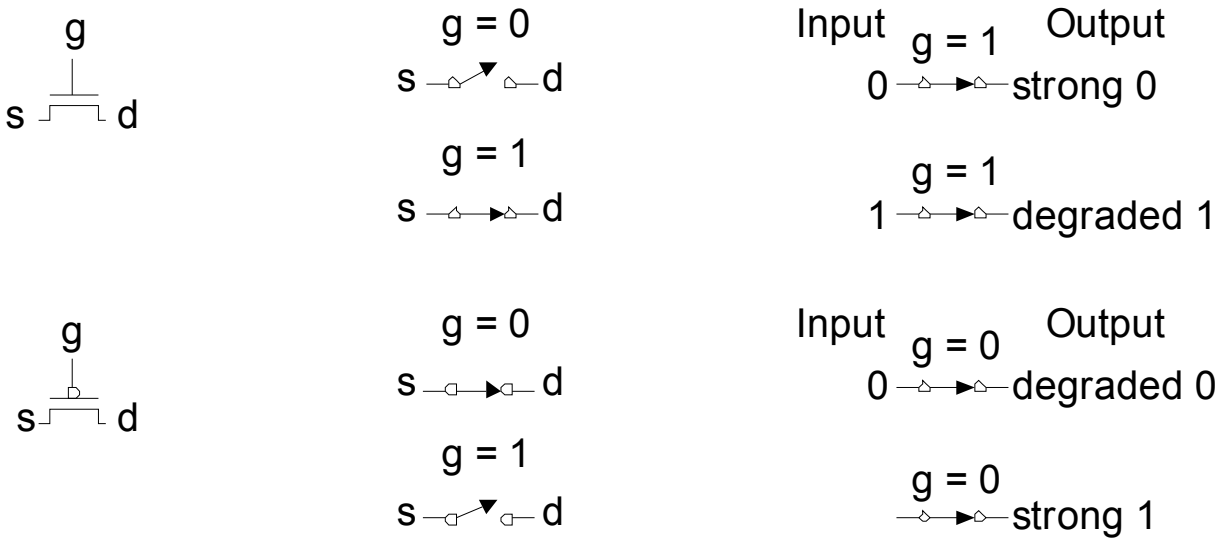


Figure 4.7 pass transistor

Transmission gates:

If pmos and nmos transistors are connected in parallel, (refer figure 4.8)it forms the transmission gate. The figure below shows the structure and operation of the transmission gate. Transmission gates are used to pass both the logic 0 and logic 1 as good. Let us consider that g=0 and gb=1 then both the transistor are off so it does not allow the input to pass at the output. when g=1 and gb=0 both the transistor are on and input is connected to output. now if the input is logic 0 nmos transistor passes good logic for 0 and if the input is high pmos transistor passes good logic to output.

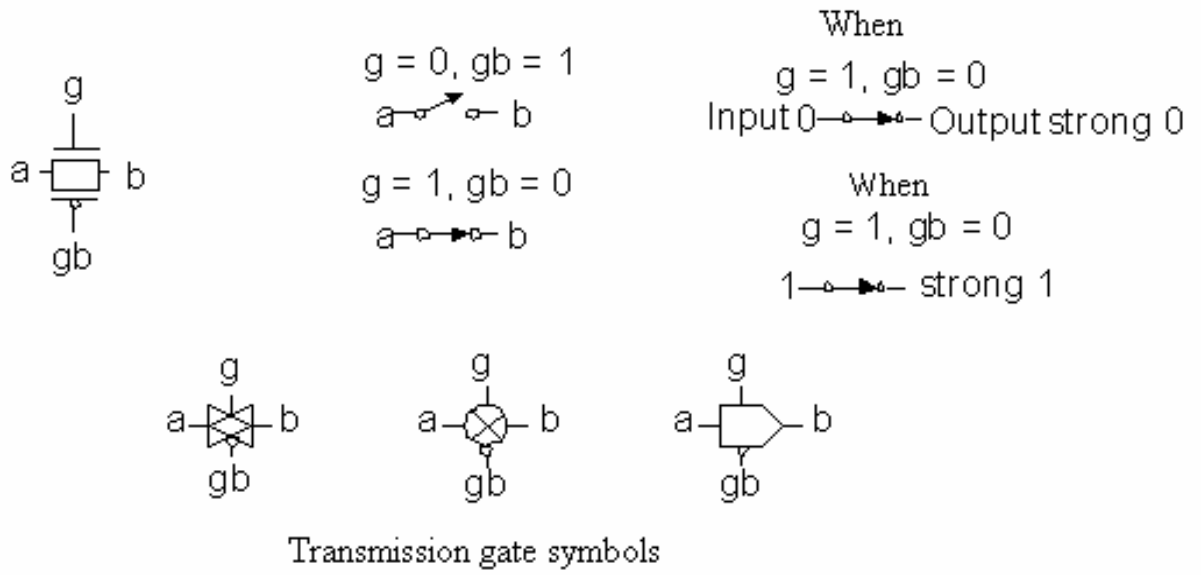


Figure 4.8 transmission gate .

multiplexer can be realized using transmission gates as shown below in figure 4.9 .figure below shows 2:1 multiplexer using two transmission gates.when s=0 the upper transmission gate is on while bottom transmission gate is off and D0 is passed at the output,similarly when s=1 then the bottom transmission gate is on while top transmission gate is off so D1 is passed at the output.

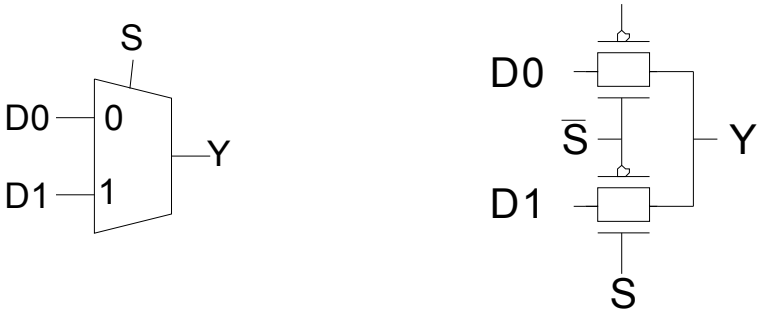


Figure 4.9 cmos multiplexer.

5.4 D LATCH:

D latch is constructed using transmission gates and inverters as shown in the figure 4.10 .latch is a memory device which passes the input at the output while maintaining the output until next input is applied.latch are level sensitive that is the output is changed during the level of clock When CLK = 1, latch is *transparent that means the output follows the input. whatever changes in input during clock=1 occurs at the output.*Q follows D (a buffer with a Delay) .When CLK = 0, the latch is *opaque that means* Q holds its last value independent of D

CMOS LATCH

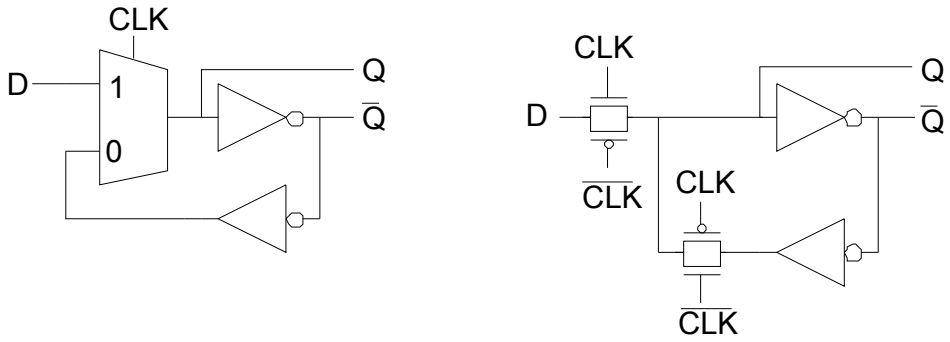


Figure 4.10 cmos latch

When clock is 1 the first transmission gate is on while other is off hence D is passed at the output Q .when clock =0 the first transmission gate is off and second transmission gate is on and the output loops through to provide the output.the operation of the latch is easily understood by the figure 4.11

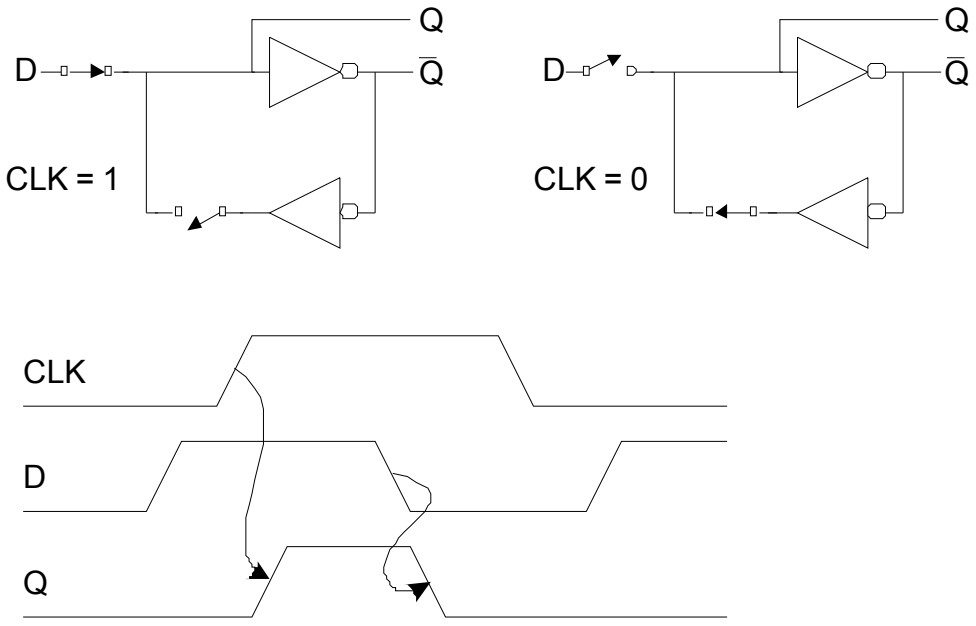


Figure 4.11 LATCH and its OPERATION

5.5 D FLIPFLOP:

Flip flops are constructed using latch.if two latches are connected in cascade for flip flop is formed.flip flops are edge sensitive unlike latch which are level sensitive.flip flopchanges it output at the rising or falling edge of the clock.a basic d flip flop is shown in the figure using two latch connected in seres as master and slave.

- When CLK rises, D is copied to Q
- At all other times, Q holds its value
- The master and slave latch clock are connected in opposite polarity such that when master is enabled slave is disabled and vice versa. The operation of the flip flop is easily understood by the figure 4.12.

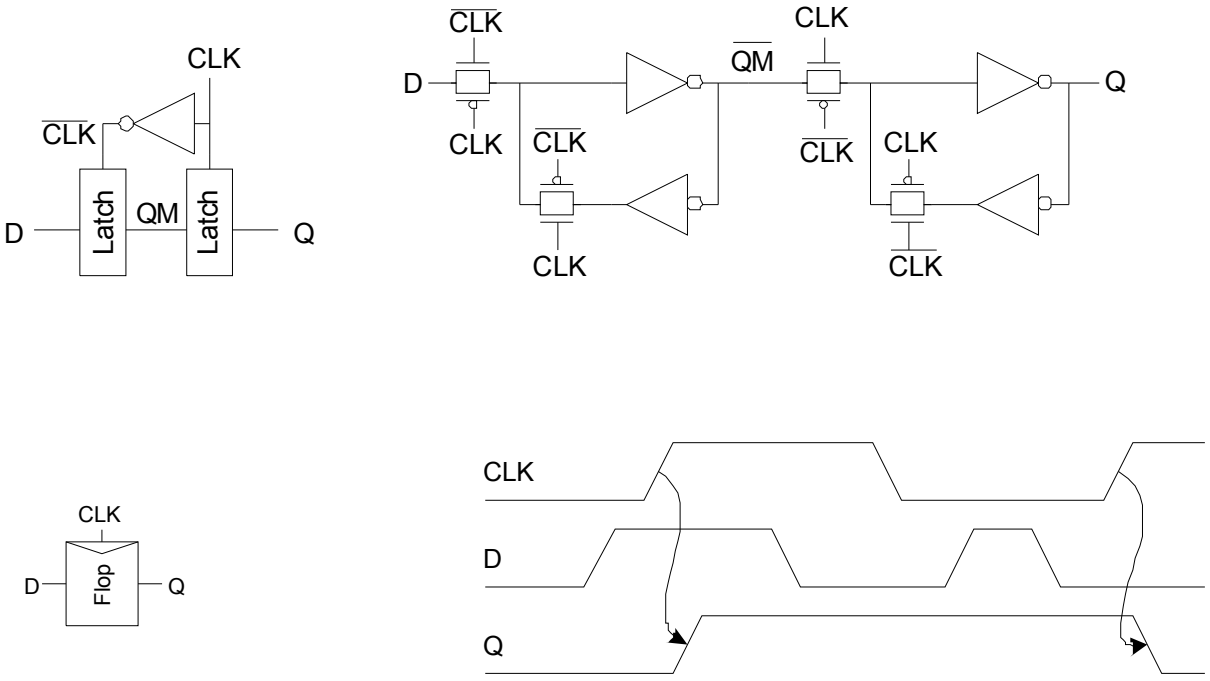
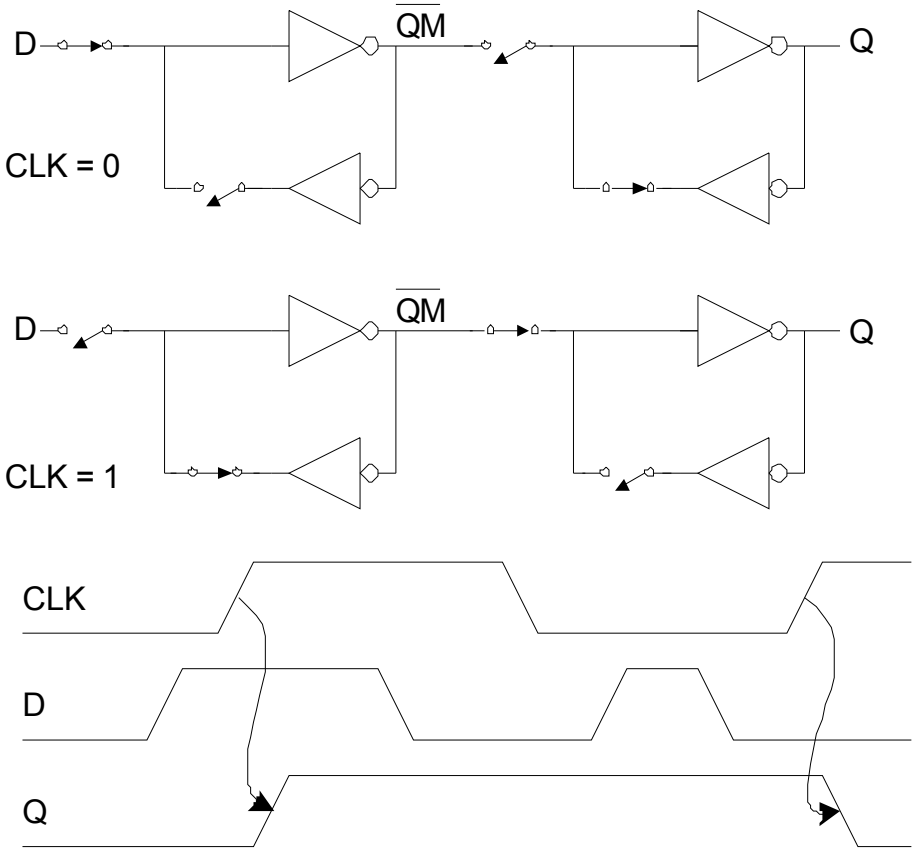


Figure 4.12 D flip flop

FLIPFLOP OPERATION:

when clock=0 input D is latch at the output as Dbar of first latch i.e QMbar=Dbar.now when the clock changes its state from 0 to 1 second latch is enabled and the switch is closed which connects the output of first latch to the input of second latch hence the output Q=D,in the mean time now the input side switch is open which blocks any input if available.

.



5.6 programmable interconnect:

Programmable logic devices are widely used these days. The benefits in using these device is it can be reprogrammed do realize various function. For programmable logic device the interconnection is programmed such that the desired connection can me made among the internal logic device. This interconnection is programmable that means we can make the internal connection among the macrocell inside by programming from outer source .the widely used programmable interconnection is antifuse.

The Antifuse

An antifuse is the opposite of a regular fuse—an antifuse is normally an open circuit until you force a programming current through it (about 5 mA).antifuse commonly in use are

- **Poly-diffusion antifuse**
- **Metal metal antifuse**

Poly-diffusion antifuse

In a poly–diffusion antifuse the high current density causes a large power dissipation in a small area, which melts a thin insulating dielectric between polysilicon and diffusion electrodes and forms a thin (about 20 nm in diameter), permanent and resistive silicon link. The programming process also drives dopant atoms from the poly and diffusion electrodes into the link, and the final level of doping determines the resistance value of the link. Actel calls its antifuse a programmable low-impedance circuit element (PLICE ').

Figure 4.13 shows a poly–diffusion antifuse with an oxide–nitride–oxide (ONO) dielectric sandwich of: silicon dioxide (SiO_2) grown over the n -type antifuse diffusion, a silicon nitride (Si_3N_4) layer, and another thin SiO_2 layer. The layered ONO dielectric results in a tighter spread of blown antifuse resistance values than using a single-oxide dielectric. The effective electrical thickness is equivalent to 10nm of SiO_2 (Si_3N_4 has a higher dielectric constant than SiO_2 , so the actual thickness is less than 10 nm). Sometimes this device is called a fuse even though it is an anti fuse, and both terms are often used interchangeably.

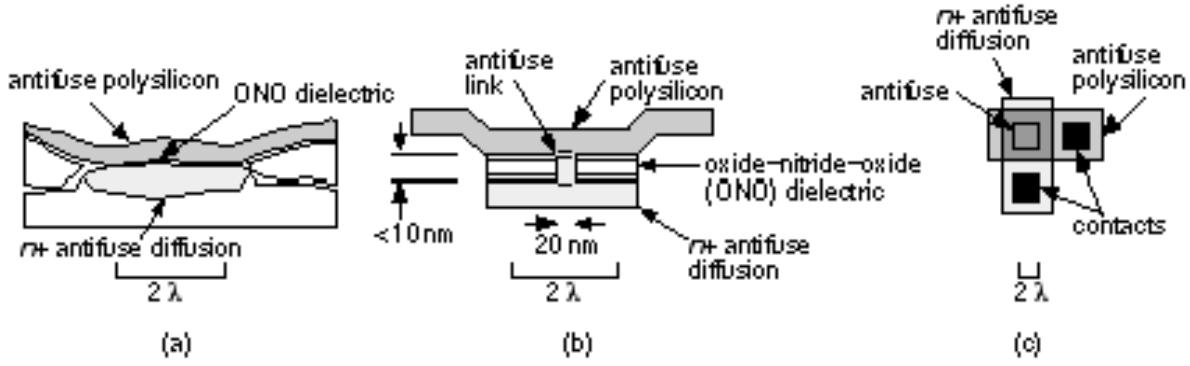


FIGURE 4.13 Actel antifuse. (a) A cross section. (b) A simplified drawing. The ONO (oxide–nitride–oxide) dielectric is less than 10 nm thick, so this diagram is not to scale. (c) From above, an antifuse is approximately the same size as a contact.

The fabrication process and the programming current control the average resistance of a blown antifuse, but values vary as shown in Figure 4.2 . In a particular technology a programming current of 5 mA may result in an average blown antifuse resistance of about 500 ohms. Increasing the programming current to 15 mA might reduce the average antifuse resistance to 100 ohms . Antifuse separate interconnect

wires on the FPGA chip and the programmer blows an antifuse to make a permanent connection. Once an antifuse is programmed, the process cannot be reversed. This is an OTP(one time programming) technology. An Actel 1010, for example, contains 112,000 antifuses we typically only need to program about 2 percent of the fuses on an Actel chip.

TABLE 4.1 Number of antifuses on Actel FPGAs.

Device	Antifuses
A1010	112,000
A1020	186,000
A1225	250,000
A1240	400,000
A1280	750,000

FIGURE 4.2 The resistance of blown Actel antifuses. The average antifuse resistance depends on the programming current. The resistance values shown here are typical for a programming current of 5 mA.

To design and program an Actel FPGA, designers iterate between design entry and simulation. When they are satisfied the design is correct they plug the chip into a socket on a special programming box, called an Activator , that generates the programming voltage. A PC downloads the configuration file to the Activator instructing it to blow the necessary antifuses on the chip. When the chip is programmed it may be removed from the Activator without harming the configuration data and the chip assembled into a system. One disadvantage of this procedure is that modern packages with hundreds of thin metal leads are susceptible to damage when they are inserted and removed from sockets. The advantage of other programming technologies is that chips may be programmed after they have been assembled on a printed-circuit board—a feature known as in-system programming (ISP).

The Actel antifuse technology uses a modified CMOS process. A double-metal, single-poly CMOS process typically uses about 12 masks—the Actel process requires an additional three masks. The n- type antifuse diffusion and antifuse polysilicon require an extra two masks and a 40 nm (thicker than normal) gate oxide (for the high-voltage transistors that handle 18 V to program the antifuses) uses one more masking step. Actel and Data General performed the initial experiments to

develop the PLICE technology and Actel has licensed the technology to Texas Instruments (TI).
The programming time for an ACT 1 device is 5 to 10 minutes. Improvements in programming make the programming time for the ACT 2 and ACT 3 devices about the same as the ACT 1. A 5-day work week, with 8-hour days, contains about 2400 minutes. This is enough time to program 240 to 480 Actel parts per week with 100 percent efficiency and no hardware down time. A production schedule of more than 1000 parts per month requires multiple or gang programmers.

Metal–Metal Antifuse:

Metal –metal antifuse uses two metal layers metal m1 and metal m2 separated with thin insulator. Figure 4.13 shows a Quick Logic metal–metal antifuse (Via Link ‘). The link is an alloy of tungsten, titanium, and silicon with a bulk resistance of about 500 mW cm.

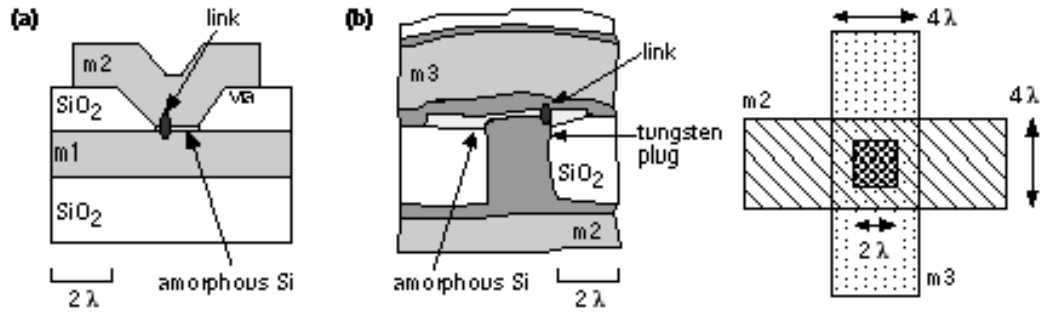
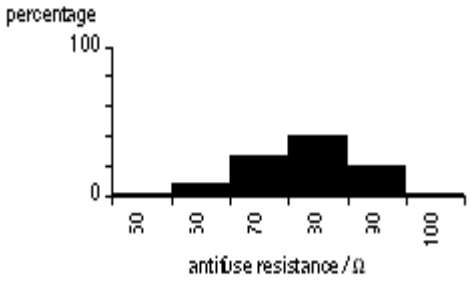


FIGURE 4.13 Metal–metal antifuse. (a) An idealized (but to scale) cross section of a QuickLogic metal–metal antifuse in a two-level metal process. (b) A metal–metal antifuse in a three-level metal process that uses contact plugs. The conductive link usually forms at the corner of the via where the electric field is highest during programming.

There are two advantages of a metal–metal antifuse over a poly–diffusion antifuse. The first is that connections to a metal–metal antifuse are direct to metal—the wiring layers. Connections from a poly–diffusion antifuse to the wiring layers require extra space and create additional parasitic capacitance. The second advantage is that the direct connection to the low-resistance metal layers makes it easier to use larger programming currents to reduce the antifuse resistance. For example, the antifuse resistance R 0.8/ I , with the programming current I in mA and R in W , for the QuickLogic antifuse. Figure 4.14 shows that the average QuickLogic metal–metal antifuse resistance is approximately 80 ohms (with a standard deviation of about 10 W) using a programming current of 15 mA as opposed to an average antifuse

resistance of 500 W (with a programming current of 5 mA) for a poly–diffusion antifuse.

FIGURE 4.14 Resistance values for the Quick Logic metal–metal antifuse. A higher programming current (about 15 mA), made possible partly by the direct connections to metal, has reduced the antifuse resistance from the poly–diffusion antifuse resistance values shown in Figure 4.12 .



The size of an antifuse is limited by the resolution of the lithography equipment used to makes ICs. The Actel antifuse connects diffusion and polysilicon, and both these materials are too resistive for use as signal interconnects. To connect the antifuse to the metal layers requires contacts that take up more space than the antifuse itself, reducing the advantage of the small antifuse size. However, the antifuse is so small that it is normally the contact and metal spacing design rules that limit how closely the antifuses may be packed rather than the size of the antifuse itself.
An antifuse is resistive and the addition of contacts adds parasitic capacitance. The intrinsic parasitic capacitance of an antifuse is small (approximately 1–2 fF in a 1 m m CMOS process), but to this we must add the extrinsic parasitic capacitance that includes the capacitance of the diffusion and poly electrodes (in a poly–diffusion antifuse) and connecting metal wires (approximately 10 fF). These unwanted parasitic elements can add considerable RC interconnect delay if the number of antifuses connected in series is not kept to an absolute minimum. Clever routing techniques are therefore crucial to antifuse-based FPGAs.
The long-term reliability of antifuses is an important issue since there is a tendency for the antifuse properties to change over time. There have been some problems in this area, but as a result we now know an enormous amount about this failure mechanism. There are many failure mechanisms in ICs—electro migration is a classic example—and engineers have learned to deal with these problems. Engineers design the circuits to keep the failure rate below acceptable limits and systems designers accept the statistics. All the FPGA vendors that use antifuse technology have extensive information on long-term reliability in their data books

5.7 ASIC DESIGN FLOW:
Asic refers to application specific integrated circuit which means that the ICs manufacture is used for special applications .for example ICs used in toys,

robots,etc.these ICs perform the application oriented task. the design flow is shown in figure 4.15.

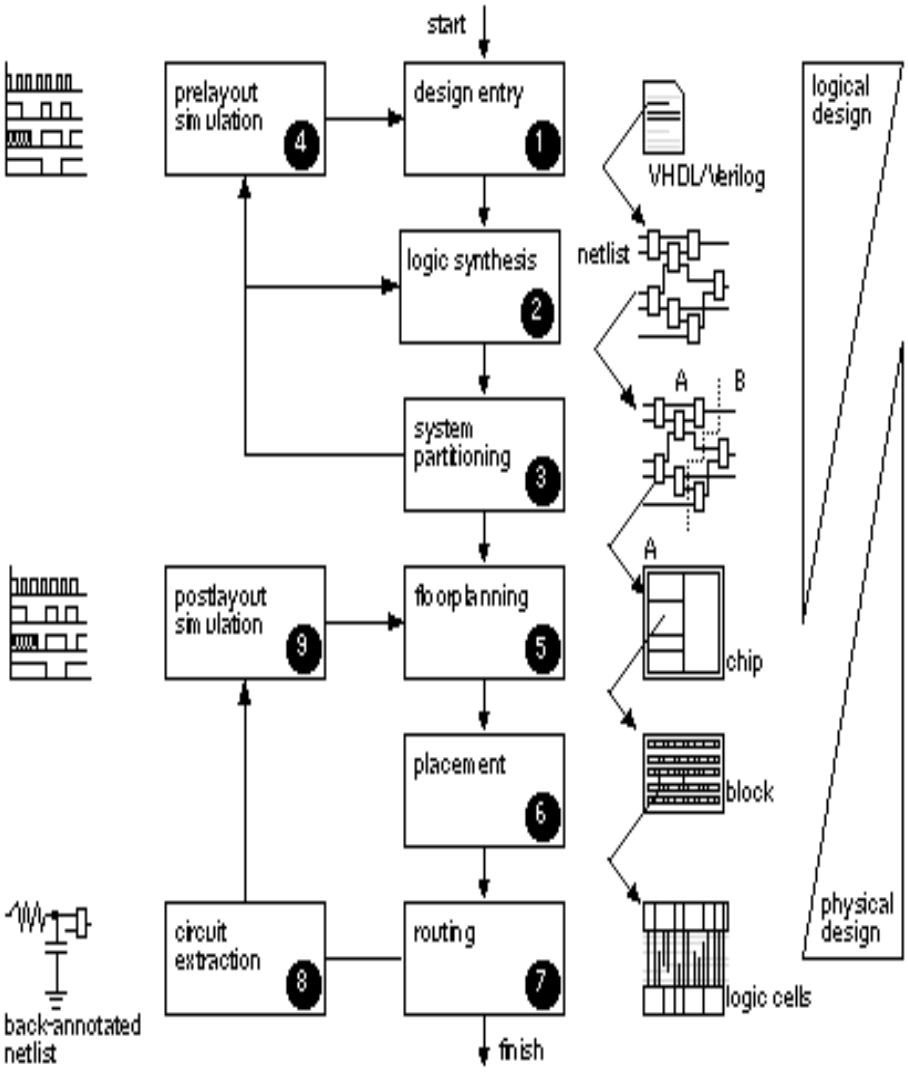


figure 4.15 Asic design flow.

1. Design entry. Enter the design into an ASIC design system, either using a hardware description language (HDL) or schematic entry . verilogHDL and VHDL are commonly used HDL language.
2. Logic synthesis. Use an HDL (VHDL or Verilog) and a logic synthesis tool (synthesis tools are CAD tools which are used to convert the design at higher level of abstraction tolower level of abstraction) to produce a netlist —a description of the logic cells and their connections.
3. System partitioning. Large circuit is difficult to handle as well as analyse so system portioning is to divide a large system into ASIC-sized pieces.
4. Prelayout simulation. Once the design is ready we have to check wheather it functions properly or not for that pre simulation is carried out.
5. Floorplanning. Floor planning is done to Arrange the blocks of the netlist on the chip. (It is similar to estimating the various rooms, kitchen, restroom etc

while building the house.)the chip contains several modules ,the module itself contains several cells,so the planning where to keep the modules is floor planning.

6. Placement. The module itself has so many cells inside ,the placement is done to Decide the locations of cells in a module(block).
7. Routing. The module ,cells within the chip has to be interconnected ,the process of this interconnection is known as routing. Global routing estimates and plans the area and length required for all modules and cells detailed routing is actual routing which optimizes the area and delay.
8. Extraction. Once the routing is over we have to determine the resistance and capacitance of the interconnect and also various parasitic that may come in to picture. the determination of this resistance and capacitance of the interconnect is called extraction.
9. Post layout simulation. After the extraction some more resistance and capacitance is added to the circuit so we have to verify once again either our design is still working or not.

5.8 Types of ASICs

ICs are made on a thin (a few hundred microns thick), circular silicon wafer , with each wafer holding hundreds of die (sometimes people use dies or dice for the plural of die). The transistors and wiring are made from many layers (usually between 10 and 15 distinct layers) built on top of one another. Each successive mask layer has a pattern that is defined using a mask similar to a glass photographic slide. The first half-dozen or so layers define the transistors. The last half-dozen or so layers define the metal wires between the transistors (the interconnect).

A full-custom IC includes some (possibly all) logic cells that are customized and all mask layers that are customized. A microprocessor is an example of a full-custom IC—designers spend many hours squeezing the most out of every last square micron of microprocessor chip space by hand. Customizing all of the IC features in this way allows designers to include analog circuits, optimized memory cells, or mechanical structures on an IC, for example. Full-custom ICs are the most expensive to manufacture and to design. The manufacturing lead time (the time it takes just to make an IC—not including design time) is typically eight weeks for a full-custom IC. These specialized full-custom ICs are often intended for a specific application, so we might call some of them full-custom ASICs.
We shall discuss full-custom ASICs briefly next, but the members of the IC family that we are more interested in are semicustom ASICs , for which all of the logic cells are predesigned and some (possibly all) of the mask layers are customized. Using predesigned cells from a cell library makes our lives as designers much, much easier. There are two types of semicustom ASICs that we shall cover: standard-cell–based ASICs and gate-array–based ASICs. Following this we shall describe the programmable ASICs , for which all of the logic cells are predesigned and none of the mask layers are customized. There are two types of programmable ASICs: the

programmable logic device and, the newest member of the ASIC family, the field-programmable gate array.

5.8.1 Full-Custom ASICs

In a full-custom ASIC an engineer designs some or all of the logic cells, circuits, or layout specifically for one ASIC. This means the designer abandons the approach of using pretested and precharacterized cells for all or part of that design. It makes sense to take this approach only if there are no suitable existing cell libraries available that can be used for the entire design. This might be because existing cell libraries are not fast enough, or the logic cells are not small enough or consume too much power. You may need to use full-custom design if the ASIC technology is new or so specialized that there are no existing cell libraries or because the ASIC is so specialized that some circuits must be custom designed. Fewer and fewer full-custom ICs are being designed because of the problems with these special parts of the ASIC. There is one growing member of this family, though, the mixed analog/digital ASIC, which we shall discuss next.

Bipolar technology has historically been used for precision analog functions. There are some fundamental reasons for this. In all integrated circuits the matching of component characteristics between chips is very poor, while the matching of characteristics between components on the same chip is excellent. Suppose we have transistors T1, T2, and T3 on an analog/digital ASIC. The three transistors are all the same size and are constructed in an identical fashion. Transistors T1 and T2 are located adjacent to each other and have the same orientation. Transistor T3 is the same size as T1 and T2 but is located on the other side of the chip from T1 and T2 and has a different orientation. ICs are made in batches called wafer lots. A wafer lot is a group of silicon wafers that are all processed together. Usually there are between 5 and 30 wafers in a lot. Each wafer can contain tens or hundreds of chips depending on the size of the IC and the wafer.
If we were to make measurements of the characteristics of transistors T1, T2, and T3 we would find the following:

- Transistors T1 will have virtually identical characteristics to T2 on the same IC. We say that the transistors match well or the tracking between devices is excellent.
- Transistor T3 will match transistors T1 and T2 on the same IC very well, but not as closely as T1 matches T2 on the same IC.
- Transistor T1, T2, and T3 will match fairly well with transistors T1, T2, and T3 on a different IC on the same wafer. The matching will depend on how far apart the two ICs are on the wafer.
- Transistors on ICs from different wafers in the same wafer lot will not match very well.

- Transistors on ICs from different wafer lots will match very poorly.

For many analog designs the close matching of transistors is crucial to circuit operation. For these circuit designs pairs of transistors are used, located adjacent to each other. Device physics dictates that a pair of bipolar transistors will always match more precisely than CMOS transistors of a comparable size. Bipolar technology has historically been more widely used for full-custom analog design because of its improved precision. Despite its poorer analog properties, the use of CMOS technology for analog functions is increasing. There are two reasons for this. The first reason is that CMOS is now by far the most widely available IC technology. Many more CMOS ASICs and CMOS standard products are now being manufactured than bipolar ICs. The second reason is that increased levels of integration require mixing analog and digital functions on the same IC: this has forced designers to find ways to use CMOS technology to implement analog functions. Circuit designers, using clever new techniques, have been very successful in finding new ways to design analog CMOS circuits that can approach the accuracy of bipolar analog designs.

5.8.2 Standard-Cell–Based ASICs

A cell-based ASIC (cell-based IC, or CBIC pronounced “sea-bick”) uses predesigned logic cells (AND gates, OR gates, multiplexers, and flip-flops, for example) known as standard cells . We could apply the term CBIC to any IC that uses cells, but it is generally accepted that a cell-based ASIC or CBIC means a standard-cell–based ASIC.

The standard-cell areas (also called flexible blocks) in a CBIC are built of rows of standard cells—like a wall built of bricks. The standard-cell areas may be used in combination with larger predesigned cells, perhaps microcontrollers or even microprocessors, known as megacells . Megacells are also called megafunctions, full-custom blocks, system-level macros (SLMs), fixed blocks, cores, or Functional Standard Blocks (FSBs).
The ASIC designer defines only the placement of the standard cells and the interconnect in a CBIC. However, the standard cells can be placed anywhere on the silicon; this means that all the mask layers of a CBIC are customized and are unique to a particular customer. The advantage of CBICs is that designers save time, money, and reduce risk by using a predesigned, pretested, and precharacterized standard-cell library . In addition each standard cell can be optimized individually. During the design of the cell library each and every transistor in every standard cell can be chosen to maximize speed or minimize area, for example. The disadvantages are the time or expense of designing or buying the standard-cell library and the time needed to fabricate all layers of the ASIC for each new design. Figure 4.20 shows a CBIC.
The important features of this type of ASIC are as follows:

- All mask layers are customized—transistors and interconnect.

- Custom blocks can be embedded.
- Manufacturing lead time is about eight weeks.

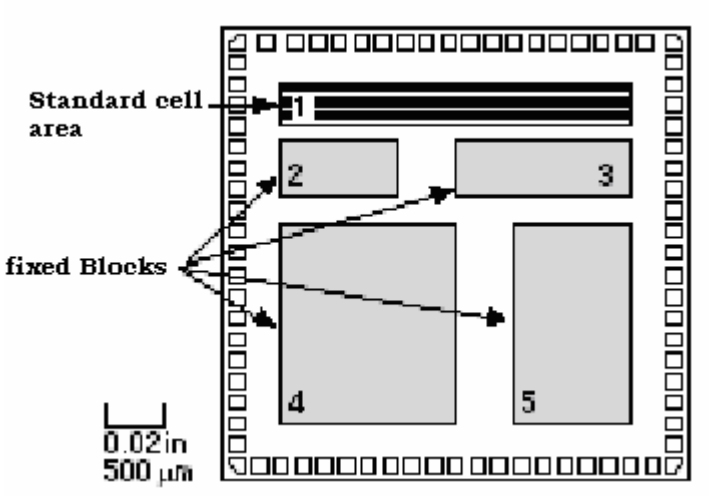


FIGURE 4.20 A cell-based ASIC (CBIC) die with a single standard-cell area (a flexible block) together with four fixed blocks.

Each standard cell in the library is constructed using full-custom design methods, but you can use these predesigned and precharacterized circuits without having to do any full-custom design yourself. This design style gives you the same performance and flexibility advantages of a full-custom ASIC but reduces design time and reduces risk.
Standard cells are designed to fit together like bricks in a wall. Figure 4.21 shows an example of a simple standard cell (it is simple in the sense it is not maximized for density—but ideal for showing you its internal construction). Power and ground buses (VDD and GND or VSS) run horizontally on metal lines inside the cells. Standard-cell design allows the automation of the process of assembling an ASIC. Groups of standard cells fit horizontally together to form rows. The rows stack vertically to form flexible rectangular blocks (which you can reshape during design). You may then connect a flexible block built from several rows of standard cells to other standard-cell blocks or other full-custom logic blocks. For example, you might want to include a custom interface to a standard, predesigned microcontroller together with some memory. The microcontroller block may be a fixed-size megacell, you might generate the memory using a memory compiler, and the custom logic and memory controller will be built from flexible standard-cell blocks, shaped to fit in the empty spaces on the chip.

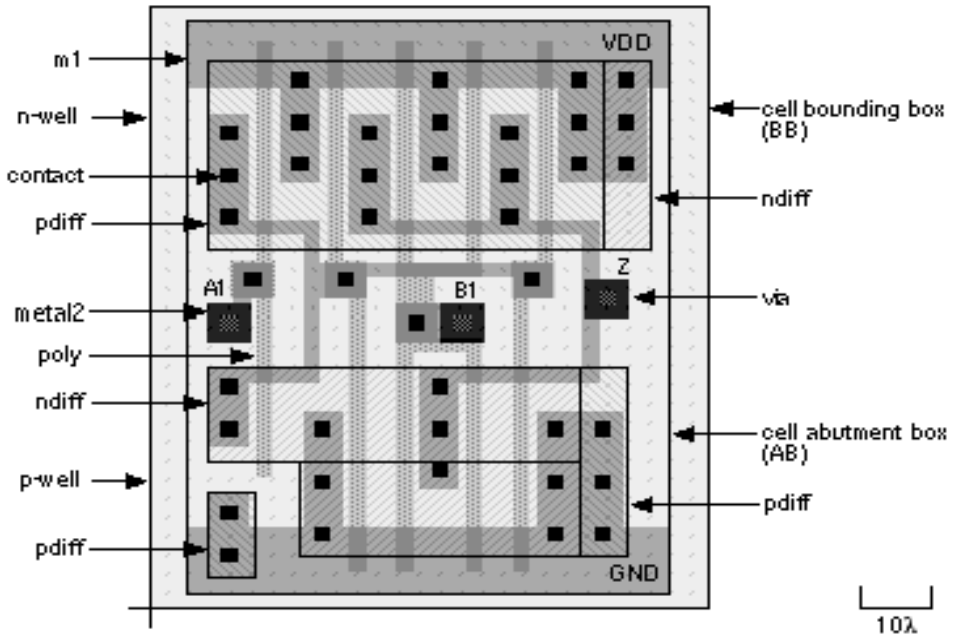


FIGURE 4.21 Looking down on the layout of a standard cell. This cell would be approximately 25 microns wide on an ASIC with l (lambda) = 0.25 microns (a micron is 10^{-6} m). Standard cells are stacked like bricks in a wall; the abutment box (AB) defines the "edges" of the brick. The difference between the bounding box (BB) and the AB is the area of overlap between the bricks. Power supplies (labeled VDD and GND) run horizontally inside a standard cell on a metal layer that lies above the transistor layers. Each different shaded and labeled pattern represents a different layer. This standard cell has center connectors (the three squares, labeled A1, B1, and Z) that allow the cell to connect to others. The layout was drawn using ROSE, a symbolic layout editor developed by Rockwell and Compass, and then imported into Tanner Research's L-Edit.

Both cell-based and gate-array ASICs use predefined cells, but there is a difference—we can change the transistor sizes in a standard cell to optimize speed and performance, but the device sizes in a gate array are fixed. This results in a trade-off in performance and area in a gate array at the silicon level. The trade-off between area and performance is made at the library level for a standard-cell ASIC. Modern CMOS ASICs use two, three, or more levels (or layers) of metal for interconnect. This allows wires to cross over different layers in the same way that we use copper traces on different layers on a printed-circuit board. In a two-level metal CMOS technology, connections to the standard-cell inputs and outputs are usually made using the second level of metal (metal2 , the upper level of metal) at the tops and bottoms of the cells. In a three-level metal technology, connections may be internal to the logic cell (as they are in Figure 4.21). This allows for more sophisticated routing programs to take advantage of the extra metal layer to route interconnect over the top of the logic cells. A connection that needs to cross over a row of standard cells uses a feedthrough. The term feedthrough can refer either to the piece of metal that is used to pass a signal through a cell or to a space in a cell

waiting to be used as a feedthrough—very confusing. Figure 1.4 shows two feedthroughs: one in cell A.14 and one in cell A.23.
In both two-level and three-level metal technology, the power buses (VDD and GND) inside the standard cells normally use the lowest (closest to the transistors) layer of metal (metal1). The width of each row of standard cells is adjusted so that they may be aligned using spacer cells . The power buses, or rails, are then connected to additional vertical power rails using row-end cells at the aligned ends of each standard-cell block. If the rows of standard cells are long, then vertical power rails can also be run in metal2 through the cell rows using special power cells that just connect to VDD and GND. Usually the designer manually controls the number and width of the vertical power rails connected to the standard-cell blocks during physical design. A diagram of the power distribution scheme for a CBIC is shown in Figure 4.22

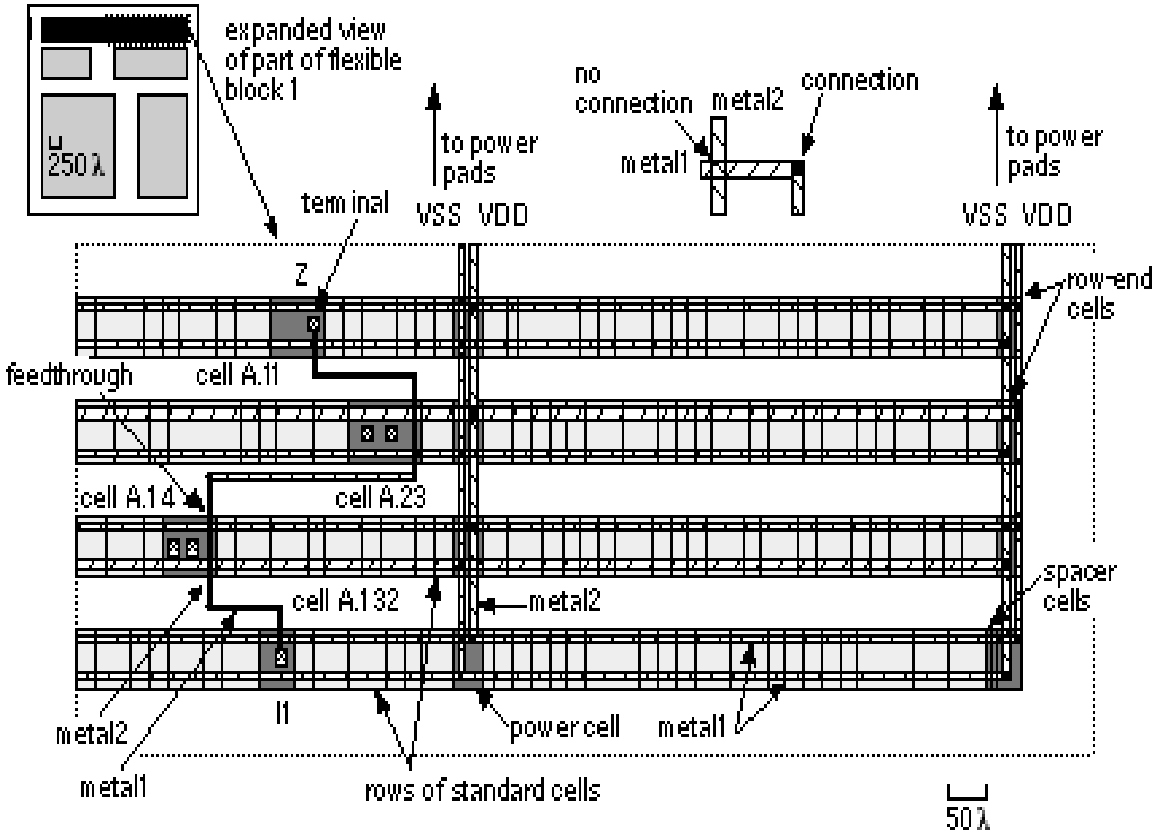


FIGURE 4.22 Routing the CBIC (cell-based IC) shown in Figure 1.2. The use of regularly shaped standard cells, such as the one in Figure 1.3, from a library allows ASICs like this to be designed automatically. This ASIC uses two separate layers of metal interconnect (metal1 and metal2) running at right angles to each other (like traces on a printed-circuit board). Interconnections between logic cells uses spaces (called channels) between the rows of cells. ASICs may have three (or more) layers of metal allowing the cell rows to touch with the interconnect running over the top of the cells.

All the mask layers of a CBIC are customized. This allows megacells (SRAM, a SCSI controller, or an MPEG decoder, for example) to be placed on the same IC with standard cells. Megacells are usually supplied by an ASIC or library company complete with behavioral models and some way to test them (a test strategy). ASIC library companies also supply compilers to generate flexible DRAM, SRAM, and ROM blocks. Since all mask layers on a standard-cell design are customized, memory design is more efficient and denser than for gate arrays.
For logic that operates on multiple signals across a data bus—a datapath (DP)—the use of standard cells may not be the most efficient ASIC design style. Some ASIC library companies provide a datapath compiler that automatically generates datapath logic. A datapath library typically contains cells such as adders, subtracters, multipliers, and simple arithmetic and logical units (ALUs). The connectors of datapath library cells are pitch-matched to each other so that they fit together. Connecting datapath cells to form a datapath usually, but not always, results in faster and denser layout than using standard cells or a gate array.
Standard-cell and gate-array libraries may contain hundreds of different logic cells, including combinational functions (NAND, NOR, AND, OR gates) with multiple inputs, as well as latches and flip-flops with different combinations of reset, preset and clocking options. The ASIC library company provides designers with a data book in paper or electronic form with all of the functional descriptions and timing information for each library element.

5.8.3 Gate-Array–Based ASICs

In a gate array (sometimes abbreviated to GA) or gate-array–based ASIC the transistors are predefined on the silicon wafer. The predefined pattern of transistors on a gate array is the base array, and the smallest element that is replicated to make the base array (like tiles on a floor) is the base cell (sometimes called a primitive cell). Only the top few layers of metal, which define the interconnect between transistors, are defined by the designer using custom masks. To distinguish this type of gate array from other types of gate array, it is often called a masked gate array (MGA). The designer chooses from a gate-array library of predesigned and precharacterized logic cells. The logic cells in a gate-array library are often called macros. The reason for this is that the base-cell layout is the same for each logic cell, and only the interconnect (inside cells and between cells) is customized, so that there is a similarity between gate-array macros and a software macro. Inside IBM, gate-array macros are known as books (so that books are part of a library), but unfortunately this descriptive term is not very widely used outside IBM.

We can complete the diffusion steps that form the transistors and then stockpile wafers (sometimes we call a gate array a prediffused array for this reason). Since only the metal interconnections are unique to an MGA, we can use the stockpiled wafers for different customers as needed. Using wafers prefabricated up to the metallization steps reduces the time needed to make an MGA, the turnaround time , to a few days or at most a couple of weeks. The costs for all the initial fabrication

steps for an MGA are shared for each customer and this reduces the cost of an MGA compared to a full-custom or standard-cell ASIC design.
There are the following different types of MGA or gate-array–based ASICs:

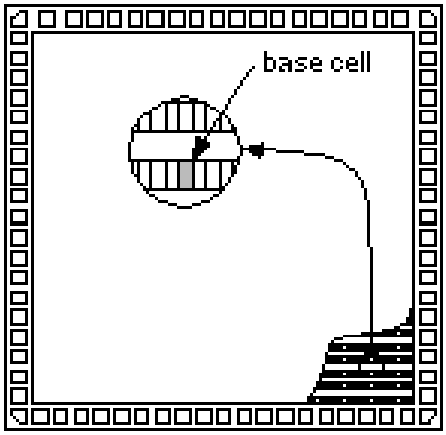
- Channeled gate arrays.
- Channelless gate arrays.
- Structured gate arrays.

The hyphenation of these terms when they are used as adjectives explains their construction. For example, in the term “channeled gate-array architecture,” the *gate array* is *channeled*, as will be explained. There are two common ways of arranging (or arraying) the transistors on a MGA: in a channeled gate array we leave space between the rows of transistors for wiring; the routing on a channelless gate array uses rows of unused transistors. The channeled gate array was the first to be developed, but the channelless gate-array architecture is now more widely used. A structured (or embedded) gate array can be either channeled or channelless but it includes (or embeds) a custom block.

❖ Channeled Gate Array

Figure 4.23 shows a channeled gate array . The important features of this type of MGA are:

- Only the interconnect is customized.
- The interconnect uses predefined spaces between rows of base cells.
- Manufacturing lead time is between two days and two weeks.



. FIGURE 4.23 A channeled gate-array die. The spaces between rows of the base cells are set aside for interconnect

A channeled gate array is similar to a CBIC—both use rows of cells separated by channels used for interconnect. One difference is that the space for interconnect

between rows of cells are fixed in height in a channeled gate array, whereas the space between rows of cells may be adjusted in a CBIC.

❖ Channelless Gate Array

Figure 4.24 shows a channelless gate array (also known as a channel-free gate array , sea-of-gates array , or SOG array). The important features of this type of MGA are as follows:Only some (the top few) mask layers are customized—the interconnect.

- Manufacturing lead time is between two days and two weeks.

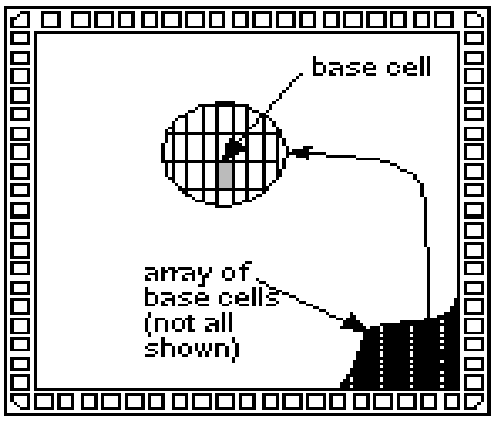


FIGURE 4.24 A channelless gate-array or sea-of-gates (SOG) array die. The core area of the die is completely filled with an array of base cells (the base array).

The key difference between a channelless gate array and channeled gate array is that there are no predefined areas set aside for routing between cells on a channelless gate array. Instead we route over the top of the gate-array devices. We can do this because we customize the contact layer that defines the connections between metal1, the first layer of metal, and the transistors. When we use an area of transistors for routing in a channelless array, we do not make any contacts to the devices lying underneath; we simply leave the transistors unused.
The logic density—the amount of logic that can be implemented in a given silicon area—is higher for channelless gate arrays than for channeled gate arrays. This is usually attributed to the difference in structure between the two types of array. In fact, the difference occurs because the contact mask is customized in a channelless gate array, but is not usually customized in a channeled gate array. This leads to denser cells in the channelless architectures. Customizing the contact layer in a channelless gate array allows us to increase the density of gate-array cells because we can route over the top of unused contact sites.

❖ Structured Gate Array

An embedded gate array or structured gate array (also known as masterslice or masterimage) combines some of the features of CBICs and MGAs. One of the disadvantages of the MGA is the fixed gate-array base cell. This makes the implementation of memory, for example, difficult and inefficient. In an embedded gate array we set aside some of the IC area and dedicate it to a specific function. This embedded area either can contain a different base cell that is more suitable for building memory cells, or it can contain a complete circuit block, such as a microcontroller.

Figure 4.25 shows an embedded gate array. The important features of this type of MGA are the following:

- Only the interconnect is customized.
- Custom blocks (the same for each design) can be embedded.
- Manufacturing lead time is between two days and two weeks.

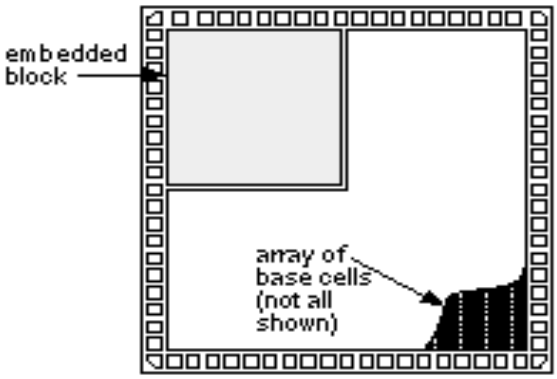


FIGURE 4.25 A structured or embedded gate-array die showing an embedded block in the upper left corner (a static random-access memory, for example). The rest of the die is filled with an array of base cells.

An embedded gate array gives the improved area efficiency and increased performance of a CBIC but with the lower cost and faster turnaround of an MGA. One disadvantage of an embedded gate array is that the embedded function is fixed. For example, if an embedded gate array contains an area set aside for a 32 k-bit memory, but we only need a 16 k-bit memory, then we may have to waste half of the embedded memory function. However, this may still be more efficient and cheaper than implementing a 32 k-bit memory using macros on a SOG array.
ASIC vendors may offer several embedded gate array structures containing different memory types and sizes as well as a variety of embedded functions. ASIC

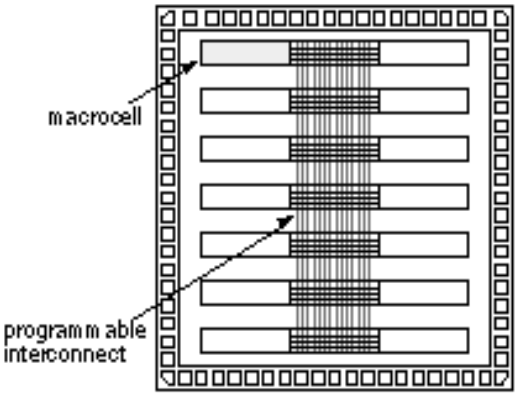
companies wishing to offer a wide range of embedded functions must ensure that enough customers use each different embedded gate array to give the cost advantages over a custom gate array or CBIC (the Sun Microsystems SPARCstation 1 described in Section 1.3 made use of LSI Logic embedded gate arrays—and the 10K and 100K series of embedded gate arrays were two of LSI Logic's most successful products).

5.9 Programmable Logic Devices

Programmable logic devices (PLDs) are standard ICs that are available in standard configurations from a catalog of parts and are sold in very high volume to many different customers. However, PLDs may be configured or programmed to create a part customized to a specific application, and so they also belong to the family of ASICs. PLDs use different technologies to allow programming of the device.
Figure 4.26 shows a PLD and the following important features that all PLDs have in common:

- No customized mask layers or logic cells
- Fast design turnaround
- A single large block of programmable interconnect
- A matrix of logic macrocells that usually consist of programmable array logic followed by a flip-flop or latch

FIGURE 4.26 A programmable logic device (PLD) die. The macrocells typically consist of programmable array logic followed by a flip-flop or latch. The macrocells are connected using a large programmable interconnect block.

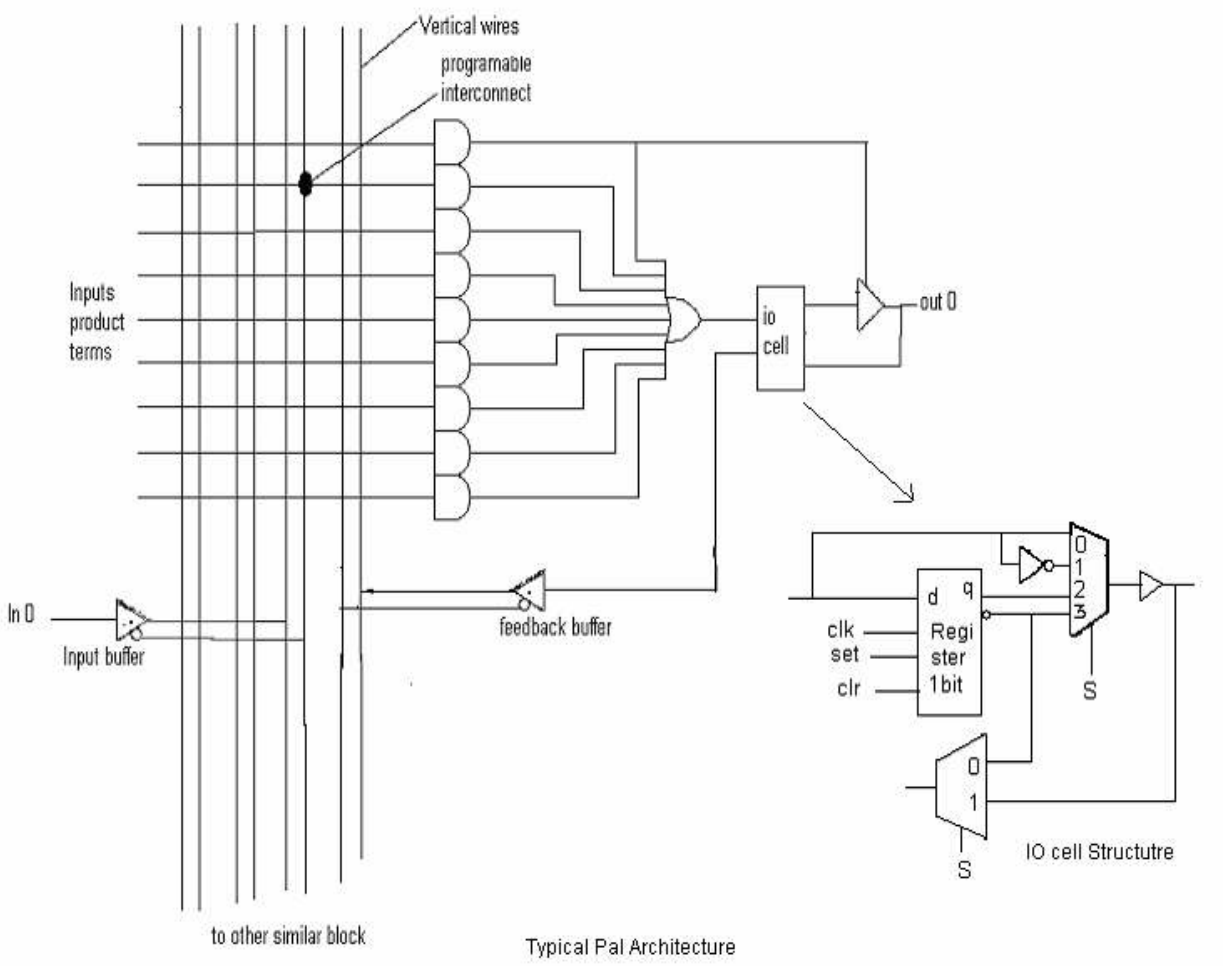


The simplest type of programmable IC is a read-only memory (ROM). The most common types of ROM use a metal fuse that can be blown permanently (a programmable ROM or PROM). An electrically programmable ROM , or EPROM , uses programmable MOS transistors whose characteristics are altered by applying a high voltage. You can erase an EPROM either by using another high voltage (an electrically erasable PROM , or EEPROM) or by exposing the device to ultraviolet light (UV-erasable PROM , or UVPROM).
There is another type of ROM that can be placed on any ASIC—a mask-programmable ROM (mask-programmed ROM or masked ROM). A masked ROM is

a regular array of transistors permanently programmed using custom mask patterns. An embedded masked ROM is thus a large, specialized, logic cell.
The same programmable technologies used to make ROMs can be applied to more flexible logic structures. By using the programmable devices in a large array of AND gates and an array of OR gates, we create a family of flexible and programmable logic devices called logic arrays . The company Monolithic Memories (bought by AMD) was the first to produce Programmable Array Logic (PAL ® , a registered trademark of AMD) devices that you can use, for example, as transition decoders for state machines. A PAL can also include registers (flip-flops) to store the current state information so that you can use a PAL to make a complete state machine.
Just as we have a mask-programmable ROM, we could place a logic array as a cell on a custom ASIC. This type of logic array is called a programmable logic array (PLA). There is a difference between a PAL and a PLA: a PLA has a programmable AND logic array, or AND plane , followed by a programmable OR logic array, or OR plane ; a PAL has a programmable AND plane and, in contrast to a PLA, a fixed OR plane.

- Depending on how the PLD is programmed, we can have an erasable PLD (EPLD), or mask-programmed PLD (sometimes called a masked PLD but usually just PLD). The first PALs, PLAs, and PLDs were based on bipolar technology and used programmable fuses or links. CMOS PLDs usually employ floating-gate transistors .

PAL Device 22V10:
PAL 22v10 (Programmable array logic) is programmable logic device. Generally PLD are implemented with AND-OR Plane, where both or one Plane can be programmed to realize the function. In PLA (programmable logic array) both the AND – OR plane can be programmed. For PAL device and plane is programmable and Or plane is fixed. The device 22v10 is a pal device which can be programmed to realize the function. This design approach has no of vertical lines for inputs, which can be connected to and plane by programming it. Each and or gate has variable number of product term that feed the or gate. the output of the or gate feeds to an io cell, which allows the registering of AND OR signals and also the feed back can be taken to AND OR plane.
22v10 PAL device has 12 inputs pin, 10 I/O pins all together 24 pin. The device has AND OR Plane with IO cell as shown in figure.



Typical Pal Architecture

IO structure consists of Register,4 to 1 multiplexer ,Tristate Buffer, and 2 to 1 multiplexer. The tristate buffer is used to enable the output. Alternatively this output pin may be used as an input to the array also.4 to 1 mux routes the true or complimented version of the product term or register the output .2 to 1 multiplexer may also select the register output. the register is provided with the global synchronous preset and asynchronous reset. Typical speed for 22v10 in high speed Cmos are

Clock to Output - 8ns, input to combinational output - 15ns, toglleing frequency with feedback-40megahertz.

The programming in PAL is done In 3 Technology :

Fusable Link, UV-erasable EPROM, EEPROM.

Fusable links are metal links such as platinum slicide or titanium tungsten .the links are blown when a certain current is allowed to pass from them .A programming voltage higher than the normal operating voltage is used to generate this large current which heat the wire and the link is broken.

UV erasable technology uses the floating gate structure for programming the device. we have discussed the EPROM technology in this text already.

5.9.1 Field-Programmable Gate Arrays

A step above the PLD in complexity is the field-programmable gate array (FPGA). There is very little difference between an FPGA and a PLD—an FPGA is usually just larger and more complex than a PLD. In fact, some companies that manufacture programmable ASICs call their products FPGAs and some call them complex PLDs . FPGAs are the newest member of the ASIC family and are rapidly growing in importance, replacing TTL in microelectronic systems. Even though an FPGA is a type of gate array, we do not consider the term gate-array–based ASICs to include FPGAs. This may change as FPGAs and MGAs start to look more alike.

Figure 4.27 illustrates the essential characteristics of an FPGA:

- None of the mask layers are customized.
- A method for programming the basic logic cells and the interconnect.
- The core is a regular array of programmable basic logic cells that can implement combinational as well as sequential logic (flip-flops).
- A matrix of programmable interconnect surrounds the basic logic cells.
- Programmable I/O cells surround the core.
- Design turnaround is a few hours.

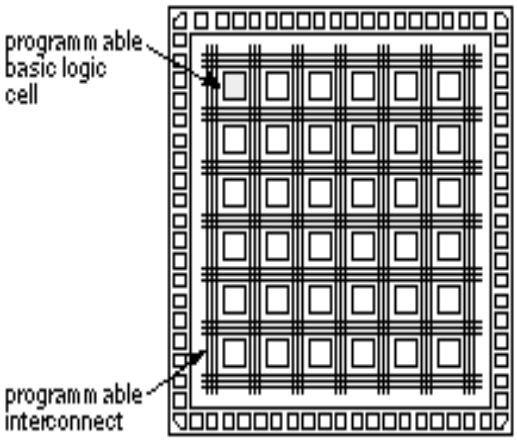


FIGURE 4.27 A field-programmable gate array (FPGA) die. All FPGAs contain a regular structure of programmable basic logic cells surrounded by programmable interconnect. The exact type, size, and number of the programmable basic logic cells varies tremendously.

5.10. Xilinx LCA

Xilinx LCA (a trademark, denoting logic cell array) basic logic cells, configurable logic blocks or CLBs , are bigger and more complex than the Actel or QuickLogic cells. The Xilinx LCA basic logic cell is an example of a coarse-grain architecture. The Xilinx CLBs contain both combinational logic and flip-flops.

❖ XC3000 CLB

The XC3000 CLB, shown in Figure 4.28 , has five logic inputs (A–E), a common clock input (K), an asynchronous direct-reset input (RD), and an enable (EC). Using programmable MUXes connected to the SRAM programming cells, you can independently connect each of the two CLB outputs (X and Y) to the output of the flip-flops (QX and QY) or to the output of the combinational logic (F and G).

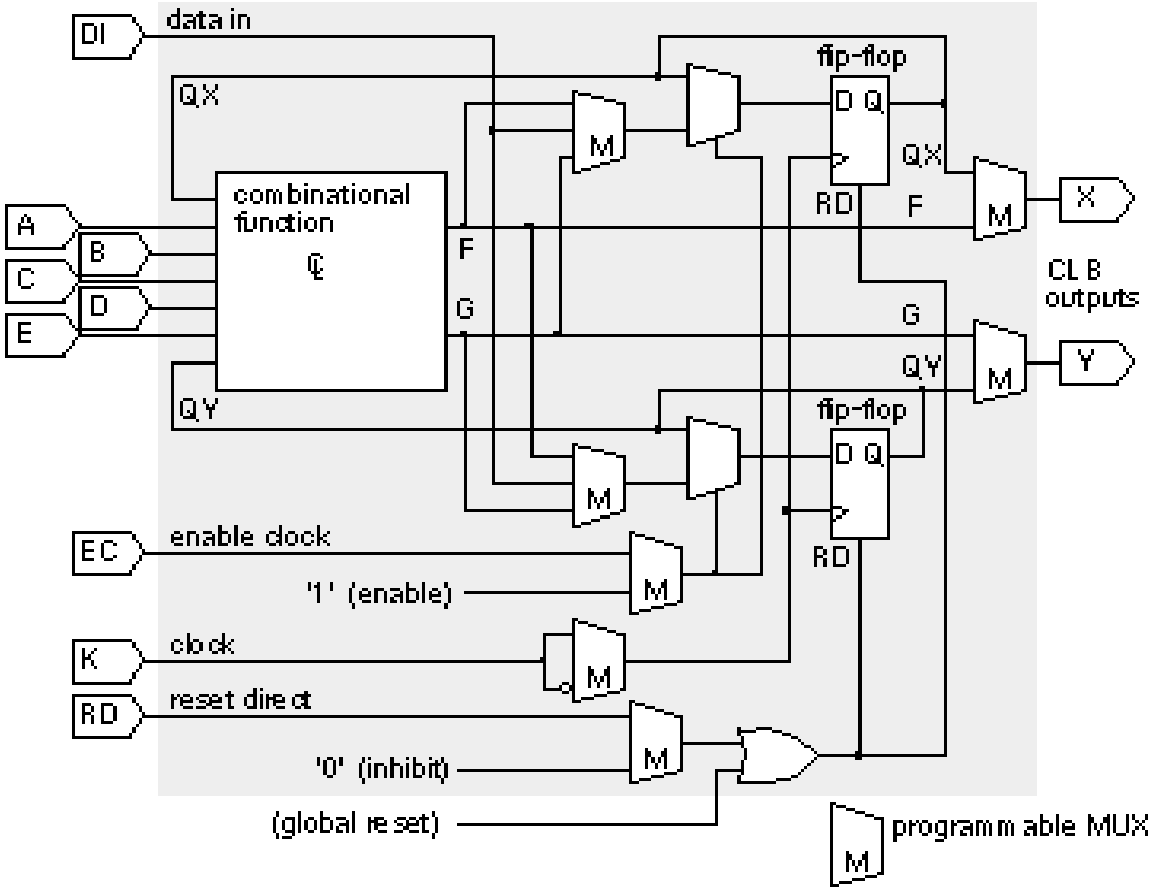


FIGURE 4.28 The Xilinx XC3000 CLB (configurable logic block). (Source: Xilinx.)

A 32-bit look-up table (LUT), stored in 32 bits of SRAM, provides the ability to implement combinational logic. Suppose you need to implement the function F = A · B · C · D · E (a five-input AND). You set the contents of LUT cell number 31 (with address '11111') in the 32-bit SRAM to a '1'; all the other SRAM cells are set to '0'. When you apply the input variables as an address to the 32-bit SRAM, only when ABCDE = '11111' will the output F be a '1'. This means that the CLB propagation delay is fixed, equal to the LUT access time, and independent of the logic function you implement.
There are seven inputs for the combinational logic in the XC3000 CLB: the five CLB inputs (A–E), and the flip-flop outputs (QX and QY). There are two outputs from the LUT (F and G). Since a 32-bit LUT requires only five variables to form a unique address ($32 = 2^5$), there are several ways to use the LUT:

- You can use five of the seven possible inputs (A–E, QX, QY) with the entire 32-bit LUT. The CLB outputs (F and G) are then identical.
- You can split the 32-bit LUT in half to implement two functions of four variables each. You can choose four input variables from the seven inputs (A–E, QX, QY). You have to choose two of the inputs from the five CLB inputs (A–E); then one function output connects to F and the other output connects to G.
- You can split the 32-bit LUT in half, using one of the seven input variables as a select input to a 2:1 MUX that switches between F and G. This allows you to implement some functions of six and seven variables.

CHAPTER 6

CMOS test methods

Need and Importance of the test

Manufacturing test principle

Sensitized path testing

Manufacturing and functional test

Faults

Stuck at fault models

Fault simulation

Automatic test pattern generator(ATPG)

Design for testability (DFT)

Various design approach for DFT

ADHOC test method

Scan test method

Built in self test

Boundary scan test

CHAPTER 6

6.0 CMOS TEST Methods:

Introduction:

Design of logic integrated circuits in CMOS technology is becoming more and more complex since VLSI is the interest of many electronic IC users and manufacturers. A common problem to be solved by both users and manufacturers is the testing of these ICs.

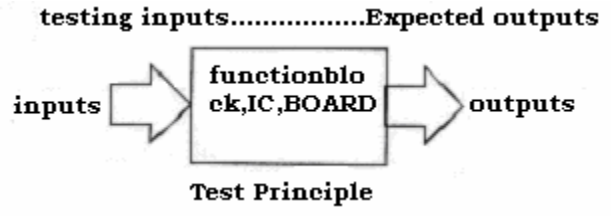


Figure 5.0 Testing of device

Testing can be expressed by checking if the outputs of a functional system (functional block, Integrated Circuit, Printed Circuit Board or a complete system) correspond to the inputs applied to it. If the test of this functional system is positive, then the system is good for use. If the outputs are different than expected, then the system has a problem: so either the system is rejected (Go/No Go test), or a diagnosis is applied to it, in order to point out and probably eliminate the problem's causes.

Testing is applied to detect faults after several operations: design, manufacturing, packaging and especially during the active life of a system, and thus since failures caused by wear-out can occur at any moment of its usage.6.1 Need and importance of the test:As we know that a wafer contains many die (chip) and it is very likely that not the entire chip works properly in that wafer. We define yield as the no of chips which are good divided by the total number of chip in that wafer. Chip manufacturing is a complex process hence if a small imperfection in materials, process technology, and other variation may result in bad die. it is then necessary to check each die individually to verify whether it is good or bad. The test of the chip can be done in different level. Starting from wafer level, packaged level, board level, system level and in the field.

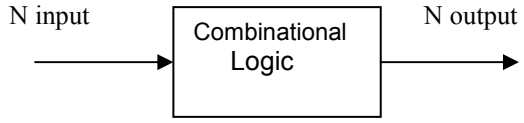
Detecting the fault in early stage is economic rather than detecting the fault at later stage. for example the cost of detecting the fault at various level is approximately given as

Wafer level	$ .01 - $.1
Packaged level	$0.1 - $1
Board level	$ 1 - $10
System level	$ 10 - $100
Field level	$ 100 - $1000

The above data shows that if we are able to detect the fault at wafer level we are saving much more cost.

6.2 Manufacturing test principles:

It is more convenient to talk about "test generation for combinational logic testing" in this section, and about "test generation for sequential logic testing" in the next section. Thus the solution to the problem of testing a purely combinational logic block is a good set of patterns detecting "all" the possible faults. The first idea to test an N input circuit would be to apply an N-bit counter to the inputs (controllability), then generate all the 2N combinations, and observe the outputs for checking (observability). This is called "exhaustive testing", and it is very efficient... but only for few- input circuits. When the input number increase, this technique becomes very time consuming. the table below shows the time needed to test the device exhaustively at 10 MHz.

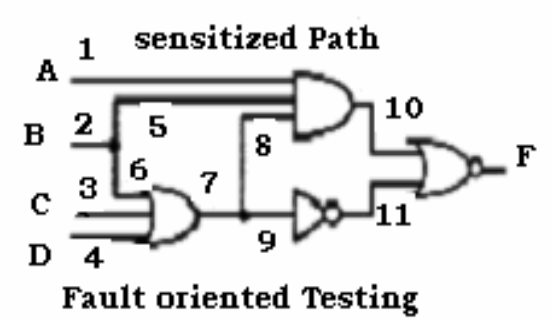


32 inputs	7 minutes
40 inputs	30 hours
64 input	58.5E8 century

6.3 Sensitized Path Testing

Most of the time, in exhaustive testing, many patterns do not occur during the application of the circuit. So instead of spending a huge amount of time searching for faults everywhere, the possible faults are first enumerated and a set of appropriate

vectors are then generated. This is called "single-path sensitization" and it is based on "fault oriented testing".



Fault oriented Testing

The basic idea is to select a path from the site of a fault, through a sequence of gates leading to an output of the combinational logic under test. The process is composed of three steps:

- Manifestation: gate inputs, at the site of the fault, are specified as to generate the opposite value of the faulty value (0 for SA1, 1 for SA0).
- Propagation: inputs of the other gates are determined so as to propagate the fault signal along the specified path to the primary output of the circuit. This is done by setting these inputs to "1" for AND/NAND gates and "0" for OR/NOR gates.
- Consistency: or justification. This final step helps finding the primary input pattern that will realize all the necessary input values. This is done by tracing backward from the gate inputs to the primary inputs of the logic in order to receive the test patterns.

Test mainly falls in two categories:

Manufacturing test

And functionality test.

6.4 Manufacturing test and functional test:

To result in good ics the chip has to under go mainly two type of test. They are functionality test and manufacturing test. Functionality test refers to the function that the chip is supposed to perform and manufacturing test refers to the correct fabrication process till the chip is packaged. Any defects can occur in the stage of manufacturing which leads the chip to malfunction or not working.

The manufacturing test is done to verify that each individual gate in the die are correctly fabricated and are in good state. This test is done when the chip is finally

packaged. Every gate is checked thoroughly and check that it is functioning properly or not. The need for this test is due to the manufacturing defects and process variations. The defects that are likely to found in this test are

- Short circuit between layers (metal to metal),
- Open circuit
- Thin oxide short with well or substrate.

The above defects leads to

- Nodes shorted to power or ground
- Nodes shorted to each other
- Inputs and outputs unconnected (open)

Test has to be done to verify that each gate and registers are functional and there is no manufacturing defects.test are normally carried out in Wafer level and bad chip(Die) are eliminated ,then it is again tested in package level.Manufacturing test is done in two level .first in production level and second at packaging level (after manufacturing).

ICs are tested at two stages during manufacture using **production tests** . First, the silicon die are tested after fabrication is complete at **wafer test** or **wafer sort**. Each wafer is tested, one die at a time, using an array of probes on a **probe card** that descend onto the bonding pads of a single die. The **production tester** applies signals generated by a **test program** and measures the IC **test response**. A test program often generates hundreds of thousands of different **test vectors** applied at a frequency of several megahertz over several hundred milliseconds. Chips that fail are automatically marked with an ink spot. Production testers are large machines that take up their own room and are very expensive (typically well over $1 million). Either the customer, or the IC manufacturer, or both, develops the test program.

A diamond saw separates the die, and the good die are bonded to a lead carrier and packaged. A second, **final test** is carried out on the packaged ASIC (usually with the same test vectors used at wafer sort) before the ASIC is shipped to the customer. The customer may apply a **goods-inward test** to incoming ICs if the customer has the resources and the product volume is large enough. Normally,

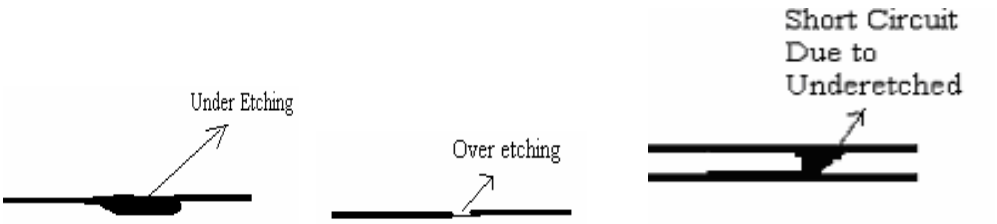
though, parts are directly assembled onto a bare **printed-circuit board** (**PCB** or **board**) and then the board is tested. If the board test shows that an IC is bad at this point, it is difficult to replace a surface-mounted component soldered on the board, for example. If there are several board failures due to a particular IC, the board manufacturer typically ships the defective chips back to the IC vendor. IC vendors have sophisticated **failure analysis** departments that take packaged ICs apart and can often determine the failure mechanism. If the IC production tests are adequate, failures are often due to the soldering process, electrostatic damage during handling, or other problems that can occur between the part being shipped and board test. If the problem is traced to defective IC fabrication, this indicates that the test program may be inadequate.

5.4.1 Functionality test:

Functionality test refers to whether the chip performs the correct function or not. For example the adder adds correctly or not, the counter counts correctly or not, the microprocessor if functioning properly or not. the functionality can be tested using various method .we can check the truth table , verbal description, description in high level language like c ,c++,hardware description language like vhdl,verilog hdl.these description can be verified using various simulation tools and cad tools.

6.5 Faults:

Fabrication of an IC Chip is a complicated process requiring hundreds of processing steps. These processing may introduce a **defect** that in turn may introduce a **fault**. Any problem during fabrication may prevent a transistor from working and may break or join interconnections. Two common types of defects occur in metallization: either underetching the metal (a problem between long, closely spaced lines), which results in a **bridge** or short circuit (**shorts**) between adjacent lines, or overetching the metal and causing **breaks** or open circuits (**opens**).



Defects may also arise after chip fabrication is complete—while testing the wafer, cutting the die from the wafer, or mounting the die in a package. Wafer probing, wafer saw, die attach, wire bonding, and the intermediate handling steps each have their own defect and failure mechanisms. Many different materials are involved in the packaging process that have different mechanical, electrical, and thermal properties, and these differences can cause defects due to corrosion, stress, adhesion failure, cracking, and peeling. Yield loss also occurs from human error—using the wrong mask, incorrectly setting the implant dose—as well as from physical

sources: contaminated chemicals, dirty etch sinks, or a troublesome process step. It is possible to repeat or **rework** some of the reversible steps (a lithography step, for example—but not etching) if there are problems. However, reliance on rework indicates a poorly controlled process.

6.5.1 Reliability:

It is possible for defects to be nonfatal but to cause failures early in the life of a product. We call this **infant mortality**. Most products follow the same kinds of trend for failures as a function of life. Failure rates decrease rapidly to a low value that remains steady until the end of life when failure rates increase again; this is called a **bathtub curve**. The end of a product lifetime is determined by various **wear out mechanisms** (usually these are controlled by an exponential energy process). Some of the most important wear out mechanisms in ASICs are hot-electron wear out, electronmigration, and the failure of antifuses in FPGAs.

We can measure the overall **reliability** of any product using the **mean time between failures** (**MTBF**) for a repairable product or **mean time to failure** (**MTTF**) for a fatal failure. We also use **failures in time** (**FITs**) where 1 FIT equals a single failure in 10^9 hours. We can sum the FITs for all the components in a product to determine an overall measure for the product reliability.
Case study of Sun SPARCstation 1
Suppose we have a system with the following components:

- Microprocessor (standard part) 5 FITs
- 100 TTL parts, 50 parts at 10 FITs, 50 parts at 15 FITs
- 100 RAM chips, 6 FITs

The overall failure rate for this system is 5 + 50 × 10 + 50 × 15 + 100 × 6 = 1855 FITs. Suppose we could reduce the component count using ASICs to the following:

- Microprocessor (custom) 7 FITs
- 9 ASICs, 10 FITs
- 5 SIMMs, 15 FITs

The failure rate is now 10 + 9 × 10 + 5 × 15 = 175 FITs, or about an order of 10 magnitude lower.

6.5.2 Fault Models

Table 1 shows some of the causes of faults. The first column shows the **fault level** —whether the fault occurs in the logic gates on the chip or in the package. The second column describes the **physical fault**. There are too many of these and we need a way to reduce and simplify their effects—by using a fault model.

There are several types of **fault model**. First, we simplify things by mapping from a physical fault to a **logical fault**. Next, we distinguish between those logical faults that degrade the device performance and those faults that are fatal and stop the device from working at all. There are three kinds of logical faults in Table 1 : a *degradation* fault, an *open-circuit* fault, and a *short-circuit* fault.

TABLE 1. Mapping physical faults to logical faults.

Fault level,	Physical fault ,	Logical fault: Degradation fault	Logical fault: Open-circuit fault	Logical fault: Short-circuit fault
Chip,	Leakage or short between package leads	•		•
	Broken, misaligned, or poor wire bonding		•	
	Surface contamination, moisture	•		
	Metal migration, stress, peeling		•	•
	Metallization (open or short)		•	•
Gate	Contact opens		•	
	Gate to S/D junction short	•		•
	Field-oxide parasitic device	•		•
	Gate-oxide imperfection, spiking	•		•
	Mask misalignment	•		•

A **degradation fault** may be a **parametric fault** or **delay fault** (**timing fault**). A parametric fault might lead to an incorrect switching threshold in a TTL/CMOS level converter at an input, for example. We can test for parametric faults using a production tester. A **delay fault** might lead to a critical path being slower than specification. Delay faults are much harder to test in production. An **open-circuit fault** results from physical faults such as a bad contact, a piece of metal that is missing or over etched, or a break in a polysilicon line. These physical faults all result in failure to transmit a logic level from one part of a circuit to another—an open circuit. A **short-circuit fault** results from such physical faults as: underetching of metal; spiking, pinholes or shorts across the gate oxide; and diffusion shorts. These faults result in a circuit being accidentally connected—a short circuit. Most short-circuit faults occur in interconnect; often we call these **bridging faults** (BF). A BF usually results from **metal coverage** problems that lead to shorts.

Physical Faults

Figure 5,2 shows the following examples of physical faults in a logic cell:

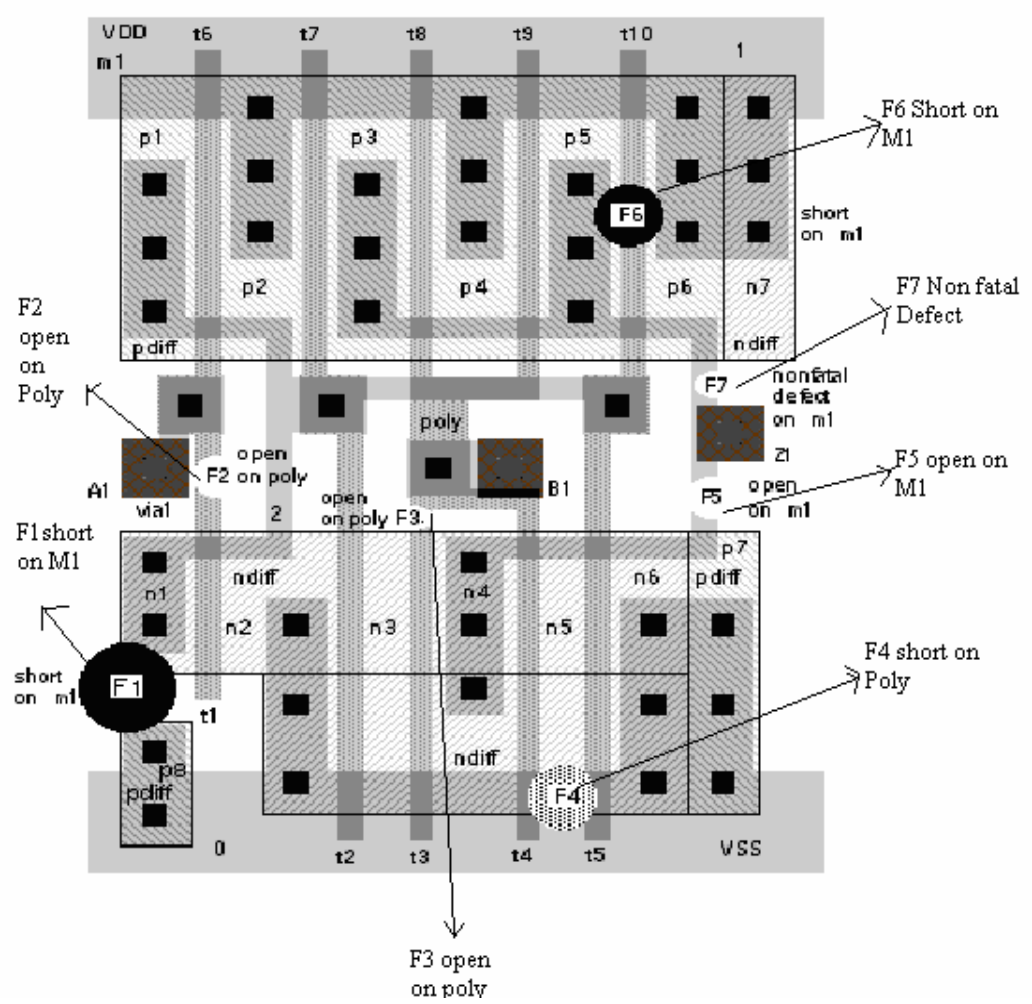


FIGURE 5.2 Defects and physical faults. Many types of defects occur during fabrication. Defects can be of any size and on any layer. Only a few small sample defects are shown here using a typical standard cell as an example. Defect density for a modern CMOS process is of the order of 1 cm $^{-2}$ or less across a whole wafer. The logic cell shown here is approximately 64 □ 32 □ ^{2} , or 250 □ m ^{2} for a □= 0.25 □ m process. We would thus have to examine approximately 1 cm $^{-2}$ /250 □ m ^{2} or 400,000 such logic cells to find a single defect.

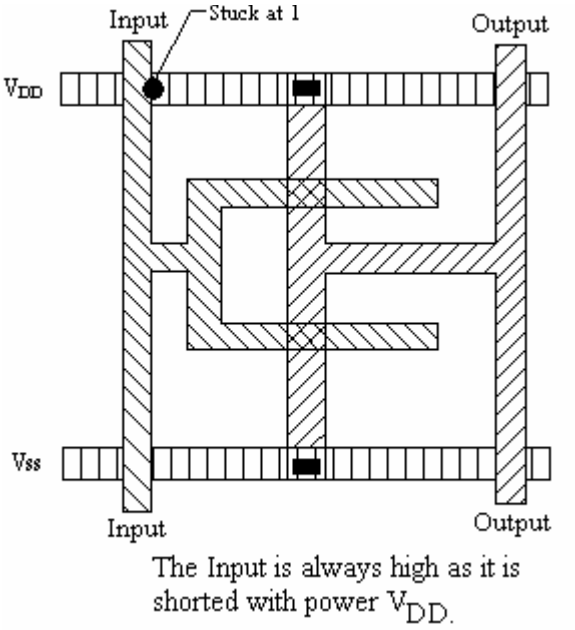
- F1 is a short between m1 lines and connects node n1 to VSS.
- F2 is an open on the poly layer and disconnects the gate of transistor t1 from the rest of the circuit.
- F3 is an open on the poly layer and disconnects the gate of transistor t3 from the rest of the circuit.
- F4 is a short on the poly layer and connects the gate of transistor t4 to the gate of transistor t5.
- F5 is an open on m1 and disconnects node n4 from the output Z1.
- F6 is a short on m1 and connects nodes p5 and p6.
- F7 is a nonfatal defect that causes necking on m1.

Once we have reduced the large number of physical faults to fewer logical faults, we need a model to predict their effect. The most common model is the **stuck-at fault model**.

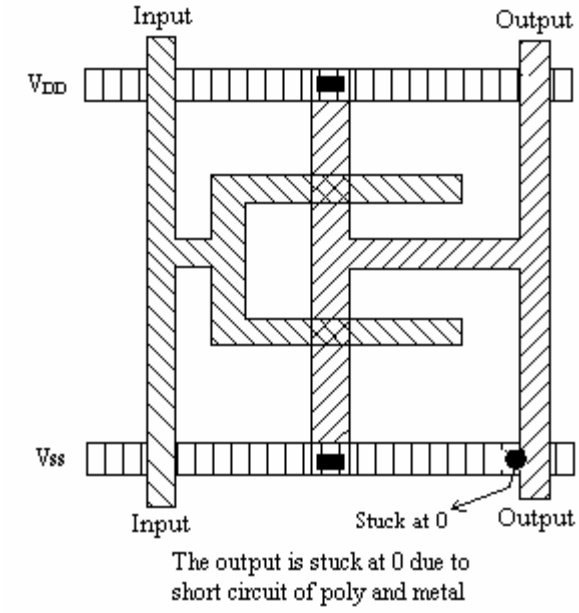
6.6 Stuck-at Fault Model:

*To deal with the existence of good and bad part it is necessary to have a fault model which describes that how a fault can occur and what is the impact of that fault in particular design.The **single stuck-at fault** (**SSF**) model assumes that there is just one fault in the logic we are testing. We use a single stuck-at fault model because a **multiple stuck-at fault model** that could handle several faults in the logic at the same time is too complicated to implement.*

There are other fault models. For example, we can assume that faults are located in the transistors using a **stuck-on fault** and **stuck-open fault** (or **stuck-off fault**). Fault models such as these are more realistic in that they more closely model the actual physical faults. However, in practice the simple SSF model has been found to work—and work well. We shall concentrate on the SSF model.
In the SSF model we further assume that the effect of the physical fault (whatever it may be) is to create only two kinds of logical fault. The two types of logical faults or **stuck-at faults** are:
stuck-at-1 fault (abbreviated to SA1 or s@1) this fault model assumes that the particular node is always connected to VDD (Power).



Stuck-at- zero fault (SA0 or s@0). This model assumes that the particular node is always connected to the ground.



We say that we **place faults** (**inject faults** , **seed faults** , or **apply faults**) on a node (or net), on an input of a circuit, or on an output of a circuit. The location at which we place the fault is the **fault origin**.
A **net fault** forces all the logic cell inputs that the net drives to a logic '1' or '0' . An **input fault** attached to a logic cell input forces the logic cell input to a '1' or '0' , but does not affect other logic cell inputs on the same net. An **output fault** attached to the output of a logic cell can have different strengths. If an output fault is a **supply-**

strength fault (or **rail-strength** fault) the logic-cell output node and every other node on that net is forced to a '1' or '0' —as if all these nodes were connected to one of the supply rails. An alternative assigns the same strength to the output fault as the drive strength of the logic cell. This allows contention between outputs on a net driving the same node. There is no standard method of handling **output-fault strength** , and no standard for using types of stuck-at faults. Usually we do not inject net faults; instead we inject only input faults and output faults. Some people use the term **node fault** —but in different ways to mean either a net fault, input fault, or output fault.

We usually inject stuck-at faults to the inputs and outputs, the pins, of logic cells (AND gates, OR gates, flip-flops, and so on). We do not inject faults to the internal nodes of a flip-flop, for example. We call this a **pin-fault model** and say the fault level is at the **structural level** , gate level, or cell level. We could apply faults to the internal logic of a logic cell (such as a flip-flop) and (the fault level would then be at the transistor level or switch level. We do not use transistor-level or switch-level fault models because there is often no need. From experience, but not from any theoretical reason, it turns out that using a fault model that applies faults at the logic-cell level is sufficient to catch the bad chips in a production test.

When a fault changes the circuit behavior, the change is called the **fault effect** . Fault effects travel through the circuit to other logic cells causing other fault effects. This phenomenon is **fault propagation** . If the fault level is at the structural level, the phenomenon is **structural fault propagation** . If we have one or more large functional blocks in a design, we want to apply faults to the functional blocks only at the inputs and outputs of the blocks. We do not want to place (or cannot place) faults inside the blocks, but we do want faults to propagate through the blocks. This is **behavioral fault propagation** .

Designers adjust the fault level to the appropriate level at which they think there may be faults. Suppose we are performing a fault simulation on a board and we have already tested the chips. Then we might set the fault level to the chip level, placing faults only at the chip pins. For ASICs we use the logic-cell level. You have to be careful, though, if you mix behavioral level and structural level models in a **mixed-level fault simulation**. You need to be sure that the behavioral models propagates faults correctly. In particular, if the behavioral model responds to faults on its inputs by propagating too many unknown 'X' values to its outputs, this will decrease the fault coverage, because the model is hiding the logic beyond it.

6.6.1 IDDQ Test

As we know that cmos logic virtually draws no current(a very small static current flows when the cmnos is in idle state), when the logic transition is not happening .this is the method to find if there is any short circuit (Bridging) between Vdd and ground.if there is any such short circuit than obviously the amount of current drawn will be much more larger than the normal current.An **IDDQ** (IDD stands for the supply current, and Q stands for quiescent) test is one of the first production tests

applied to a chip on the tester, after the chip logic has been initialized. Designers measure the resistance between VDD and GND pins. Providing there is not a short between VDD and GND, they connect the power supplies and measure the power-supply current. High supply current can result from bridging faults. We can declare that a supply current of more than a few miliamperes indicates a bad chip. This is exactly what is done in production test: Find the bad chips quickly, get them off the tester, and save expensive tester time.

6.7 Fault Simulation:

We use **fault simulation** after we have completed logic simulation to see what happens in a design when we deliberately introduce faults. In a production test we only have access to the package pins—the **primary inputs** (**PIs**) and **primary outputs** (**POs**). To test chip we must devise a series of sets of input patterns that will detect any faults. A **stimulus** is the application of one such set of inputs (a **test vector**) to the PIs of a chip. A typical ASIC may have several hundred PIs and therefore each test vector is several hundred bits long. A **test program** consists of a set of test vectors. Typical chip test programs require tens of thousands and sometimes hundreds of thousands of test vectors.

The **test-cycle time** is the period of time the tester requires to apply the stimulus, sense the POs, and check that the actual output is equal to the expected output. Suppose the test cycle time is 100 ns (corresponding to a test frequency of 10 MHz), in which case we might **sense** (or **strobe**) the POs at 90 ns after the beginning of each test cycle. Using fault simulation we mimic the behavior of the production test. The fault simulator deliberately introduces all possible faults into our ASIC, one at a time, to see if the test program will find them. For the moment we dodge the problem of how to create the thousands of test vectors required in a typical test program and focus on fault simulation.
As each fault is inserted, the fault simulator runs our test program. If the fault simulation shows that the POs of the faulty circuit are different than the PIs of the good circuit at any strobe time, then we have a **detected fault** ; otherwise we have an **undetected fault** . The list of **fault origins** is collected in a file and as the faults are inserted and simulated, the results are recorded and the faults are marked according to the result. At the end of fault simulation we can find the **Fault coverage**

Fault coverage = detected faults / detectable faults.

The number of detectable faults excludes any undetectable fault categories (untestable or redundant faults). Thus,

Detectable faults = faults – undetectable faults,
Undetectable faults = untested faults + redundant faults.

Serial Fault Simulation

Serial fault simulation is the simplest fault-simulation algorithm. We simulate two copies of the circuit, the first copy is a good circuit. We then pick a fault and insert it

into the faulty circuit. In test terminology, the circuits are called **machines** , so the two copies are a **good machine** and a **faulty machine** . We shall continue to use the term *circuit* here to show the similarity between logic and fault simulation (the simulators are often the same program used in different modes). We then repeat the process, simulating one faulty circuit at a time. Serial simulation is slow and is impractical for large ASICs.

Parallel Fault Simulation

Parallel fault simulation takes advantage of multiple bits of the words in computer memory. In the simplest case we need only one bit to represent either a '1' or '0' for each node in the circuit. In a computer that uses a 32-bit word memory we can simulate a set of 32 copies of the circuit at the same time. One copy is the good circuit, and we insert different faults into the other copies. When we need to perform a logic operation, to model an AND gate for example, we can perform the operation across all bits in the word simultaneously. In this case, using one bit per node on a 32-bit machine, we would expect parallel fault simulation to be about 32 times faster than serial simulation. The number of bits per node that we need in order to simulate each circuit depends on the number of states in the logic system we are using. Thus, if we use a four-state system with '1' , '0' , 'X' (unknown), and 'Z' (high-impedance) states, we need two bits per node.

Parallel fault simulation is not quite as fast as our simple prediction because we have to simulate all the circuits in parallel until the last fault in the current set is detected. If we use serial simulation we can stop as soon as a fault is detected and then start another fault simulation. Parallel fault simulation is faster than serial fault simulation but not as fast as concurrent fault simulation. It is also difficult to include behavioral models using parallel fault simulation.

Concurrent Fault Simulation

Concurrent fault simulation is the most widely used fault-simulation algorithm and takes advantage of the fact that a fault does not affect the whole circuit. Thus we do not need to simulate the whole circuit for each new fault. In concurrent simulation we first completely simulate the good circuit. We then inject a fault and resimulate a copy of only that part of the circuit that behaves differently (this is the **diverged circuit**). For example, if the fault is in an inverter that is at a primary output, only the inverter needs to be simulated—we can remove everything preceding the inverter.

Keeping track of exactly which parts of the circuit need to be diverged for each new fault is complicated, but the savings in memory and processing that result allow hundreds of faults to be simulated concurrently. Concurrent simulation is split into several chunks, you can usually control how many faults (usually around 100) are simulated in each chunk or **pass**. Each pass thus consists of a series of test cycles. Every circuit has a unique **fault-activity signature** that governs the divergence that occurs with different test vectors. Thus every circuit has a different optimum setting

for **faults per pass**. Too few faults per pass will not use resources efficiently. Too many faults per pass will overflow the memory.

Nondeterministic Fault Simulation

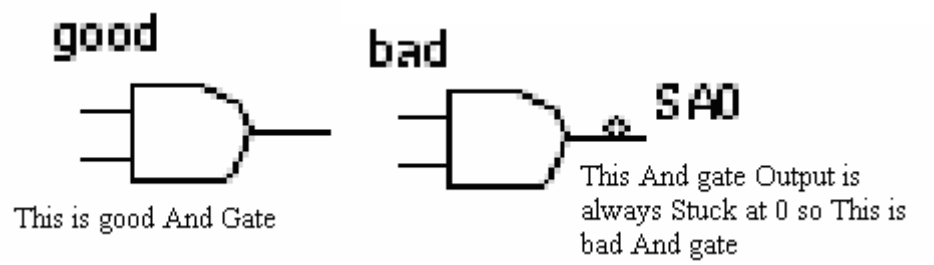
Serial, parallel, and concurrent fault-simulation algorithms are forms of **deterministic fault simulation**. In each of these algorithms, we use a set of test vectors to simulate a circuit and discover which faults we can detect. If the fault coverage is inadequate, we modify the test vectors and repeat the fault simulation. This is a very time-consuming process.

As an alternative, we give up trying to simulate every possible fault and instead, using **probabilistic fault simulation**, we simulate a subset or sample of the faults and extrapolate fault coverage from the sample.
In **statistical fault simulation,** we perform a fault-free simulation and use the results to predict fault coverage. This is done by computing measures of observability and controllability at every node.
We know that a node is not stuck if we can make the node toggle—that is, change from a '0' to '1' or vice versa. A **toggle test** checks which nodes toggle as a result of applying test vectors and gives a statistical estimate of **vector quality**, a measure of faults detected per test vector. There is a strong correlation between high-quality test vectors, the vectors that will detect most faults, and the test vectors that have the highest **toggle coverage**. Testing for nodes toggling simply requires a single logic simulation that is much faster than complete fault simulation.
We can obtain a considerable improvement in fault simulation speed by putting the high-quality test vectors at the beginning of the simulation. The sooner we can detect faults and eliminate them from having to be considered in each simulation, the faster the simulation will progress. We take the same approach when running a production test and initially order the test vectors by their contribution to fault coverage. This assumes that all faults are equally likely. Test engineers can then modify the test program if they discover vectors late in the test program that are efficient in detecting faulty chips estimate defect levels.

6.8 Automatic Test-Pattern Generation (ATPG):

When The Chip is manufactured it has to be tested whether it correctly performs the operation or not. Already we have seen that to check the functionality of the chip with many inputs is virtually impossible for all the combination of the input (Exhaustive method).generally to test the chip a combination of input vectors are created which is then applied at the input and observe its behavior at the output, if the expected output occurs than the chip can be called as good else it is bad. but the question arises how many such input vectors have to be created and how many

such test has to be conducted as the size and the density of the chip is growing more and more therefore there became the necessary for the designers to generate the input test vectors automatically which is termed as Automatic Test Pattern generation (ATPG).Historically, most ATPG approaches have been based on simulation .A five-valued logic form is commonly used to implement test generation algorithms (more advanced algorithms use up to 10 level logic). This consists of the states 1,0,D, $\overline{D}$, and X. 0 and 1 represent logical zero and logical one respectively. X represents the unknown or DON'T-CARE state. D represents a logic 1 in a good machine and a logic 0 in a faulty machine while $\overline{D}$ represents a logic 0 in good machine and a logic 1 in a faulty machine.the figure below shows the D calculus and truth table for logic gates which has 5 valued system.



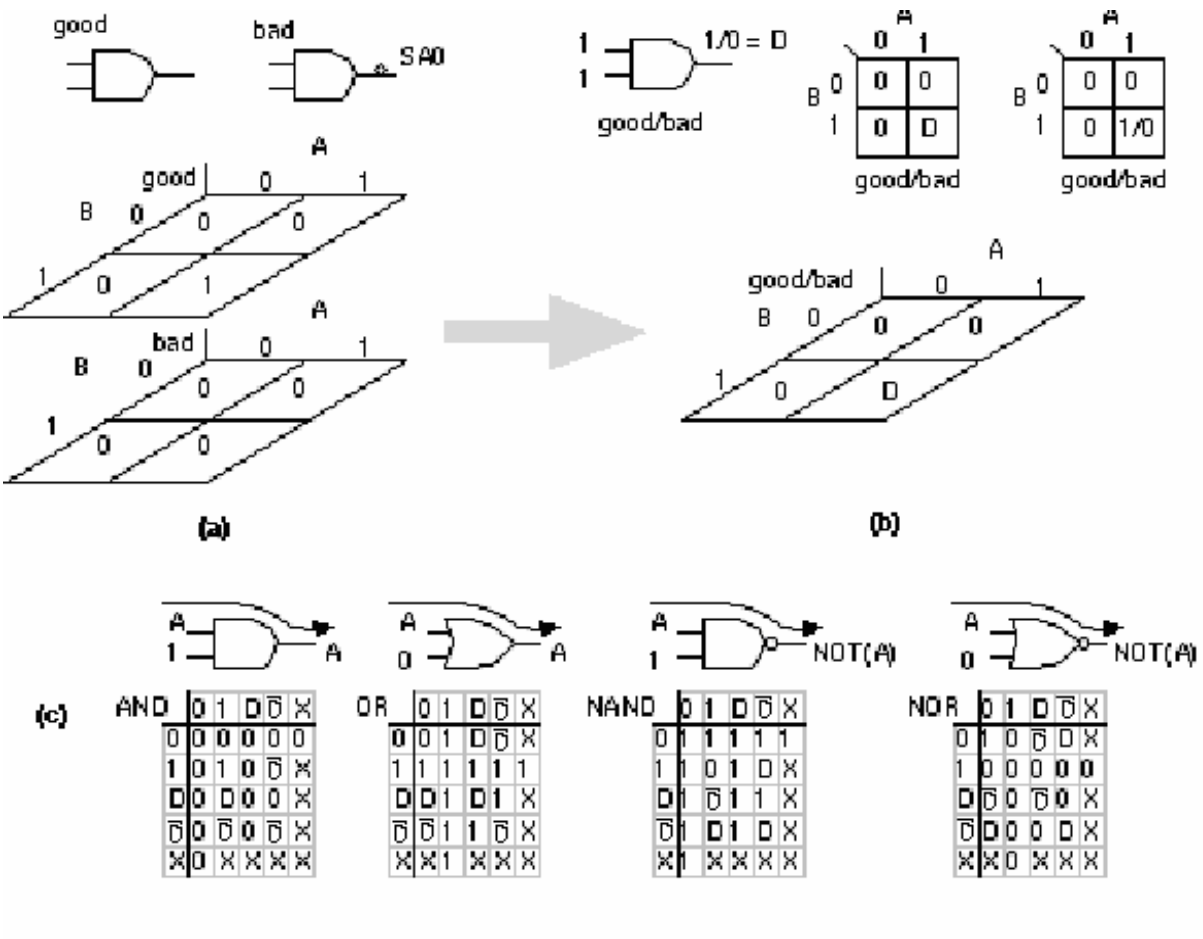


FIGURE 5.5 The D-calculus. (a) We need a way to represent the behavior of the good circuit and the bad circuit at the same time. (b) The composite logic value D (for detect) represents a logic '1' in the good circuit and a logic '0' in the bad circuit. We can also write this as D = 1/0. (c) The logic behavior of simple logic cells using the D-calculus. Composite logic values can propagate through simple logic gates if the other inputs are set to their enabling values.

The D-Calculus

Figure 5.5 (a) and (b) shows a shorthand notation, the **D-calculus** , for tracing faults. The D-calculus was developed by Roth [1966] together with an ATPG $\overline{D}$ algorithm, the **D-algorithm** . The symbol D (for detect) indicates the value of a node is a logic '0' in the good circuit and a logic '1' in the bad circuit. We can also write this as D = 0/1. In general $\overline{D}$ we write g/b, a **composite logic value**, to indicate a node value in the good circuit is g and b in the bad circuit (by convention we always write the good circuit value first and the faulty circuit value second). The complement of D is $\overline{D}$ = 1/0 ($\overline{D}$ is rarely written as D' since $\overline{D}$ is a logic value just like '1' and '0'). Notice that $\overline{D}$ does not mean *not* detected, but simply that we see a '0' in the good circuit and a '1' in the bad circuit. We can apply Boolean algebra to the composite logic values D and as shown in 5.5 (c). The composite values 1/1 and 0/0 are equivalent to '1' and '0' respectively. We use the unknown logic value 'X' to represent a logic value that is one of '0', '1', D, or , but we do not know or care which.

If we wish to **propagate** a signal from one or more inputs of a logic cell to the logic cell output, we set the remaining inputs of that logic cell to what we call the **enabling value** . The enabling value is '1' for AND and NAND gates and '0' for OR and NOR gates. Figure 5.5 (c) illustrates the use of enabling values. In contrast, setting at least one input of a logic gate to the **controlling value** , the opposite of the enabling value for that gate, forces or **justifies** the output node of that logic gate to a fixed value. The controlling value of '0' for an AND gate justifies the output to '0' and for a NAND gate justifies the output to '1'. The controlling values of '1' justifies the output of an OR gate to '1' and justifies the output of a NOR gate to '0'. To find controlling and enabling values for more complex logic cells, such as AOI and OAI logic cells, we can use their simpler AND, OR, NAND, and NOR gate representations.

Basic ATPG Algorithm

A basic algorithm to generate test vectors automatically is shown in Figure 5.6 . We detect a fault by first **activating** (or **exciting** the fault). To do this we must drive the faulty node to the opposite value of the fault. Figure 5.6 (a) shows a stuck-at-1 fault at the output pin, ZN, of the inverter U2 (we call this fault U2.ZN.SA1). To create a test for U2.ZN.SA1 we have to find the values of the PIs that will justify node U2.ZN to '0' . We work backward from node U2.ZN justifying each logic gate output until we reach a PI. In this case we only have to justify U2.ZN to '0' , and this is easily done by setting the PI A = '0'. Next we work forward from the fault origin and **sensitize** a path to a PO (there is only one PO in this example). This propagates the fault effect to the PO so that it may be **observed**. To propagate the fault effect to the PO Z, we set U3.A2 = '1' and then U5.A2 = '1'.

We can visualize fault propagation by supposing that we set all nodes in a circuit to unknown, 'X'. Then, as we successively propagate the fault effect toward the POs, we can imagine a wave of D's and D 's, called the **D-frontier** , that propagates from the fault origin toward the POs. As a value of D or D reaches the inputs of a logic cell

whose other inputs are 'X', we add that logic cell to the D-frontier. Then we find values for the other inputs to propagate the D-frontier through the logic cell to continue the process.

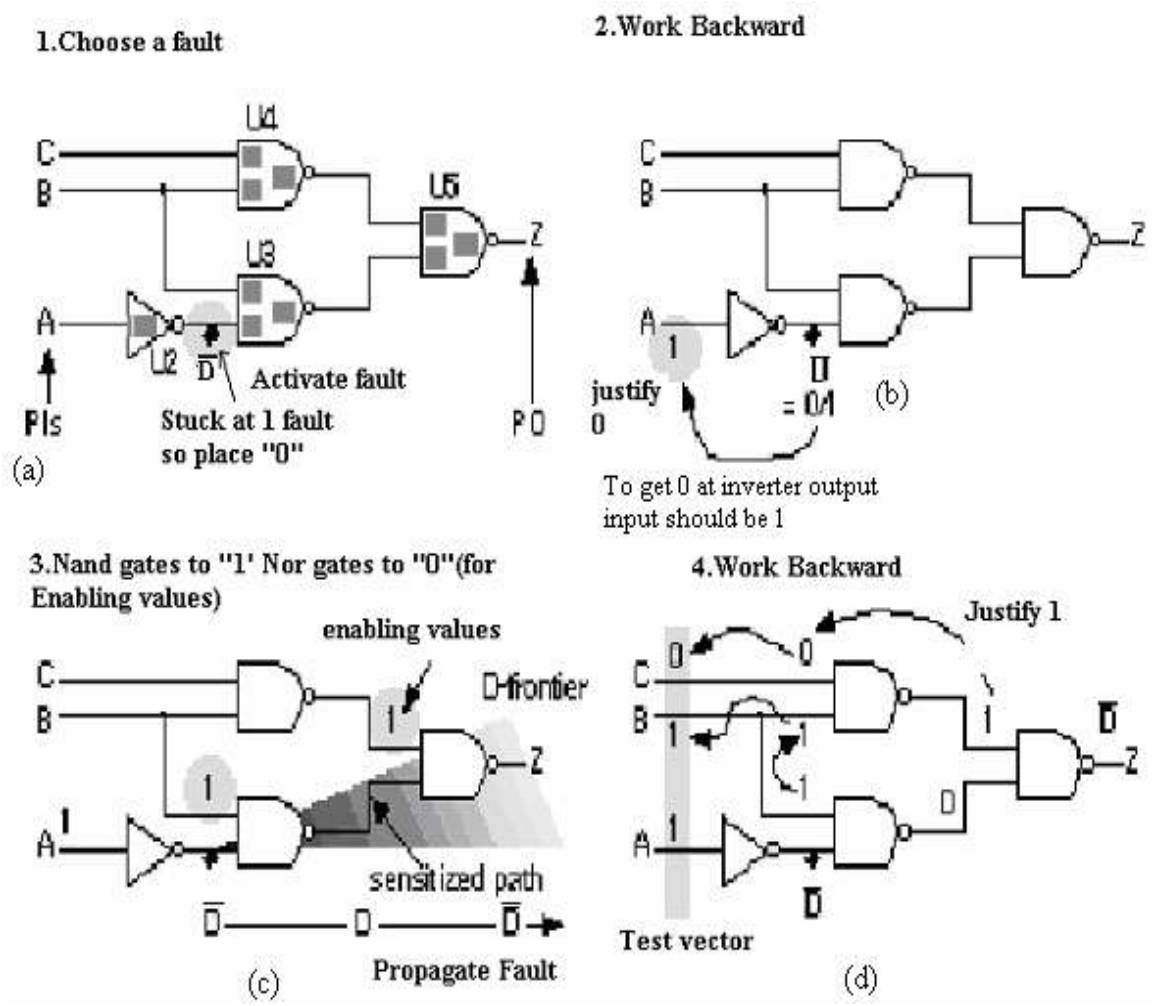
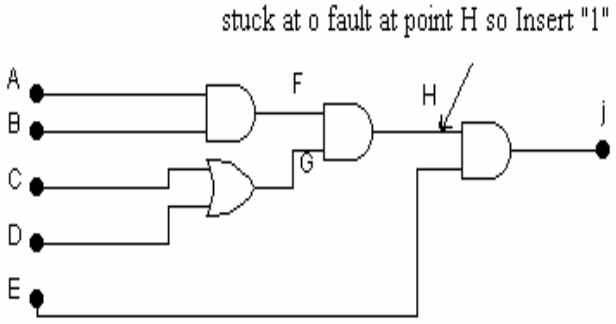


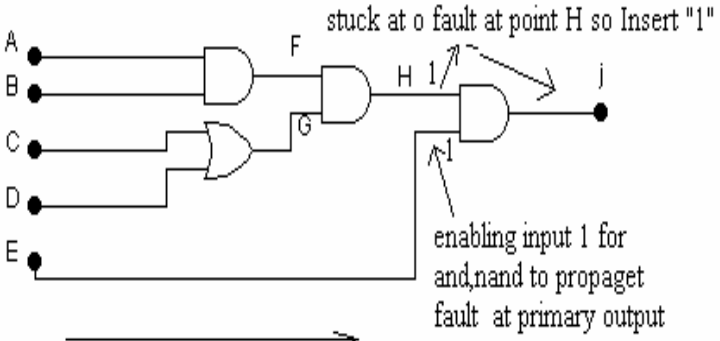
FIGURE 5.6 A basic ATPG (automatic test-pattern generation) algorithm for A'B + BC. (a) We activate a fault, U2.ZN stuck at 1, by setting the pin or node to '0', the opposite value of the fault. (b) We work backward from the fault origin to the PIs (primary inputs) by recursively justifying signals at the output of logic cells. (c) We then work forward from the fault origin to a PO (primary output), setting inputs to gates on a sensitized path to their enabling values. We propagate the fault until the D-frontier reaches a PO. (d) We then work backward from the PO to the PIs recursively justifying outputs to generate the sensitized path. This simple algorithm always works, providing signals do not branch out and then rejoin again.

Example of generating test Vector using D Algorithm.
Assume Stuck at falut 0 is to be analyzed at point H so the generation of test vector for this is given step by step.

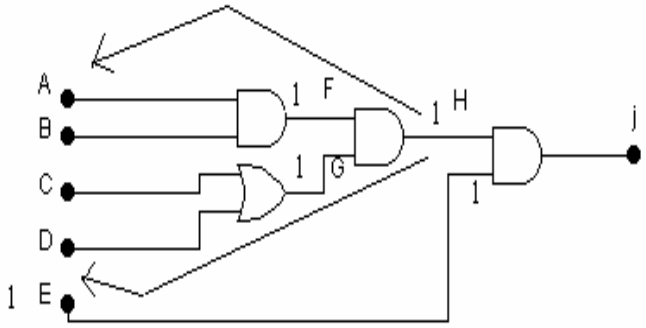


first insert the opposite value at the node where the fault is to be propagated.

Step 1

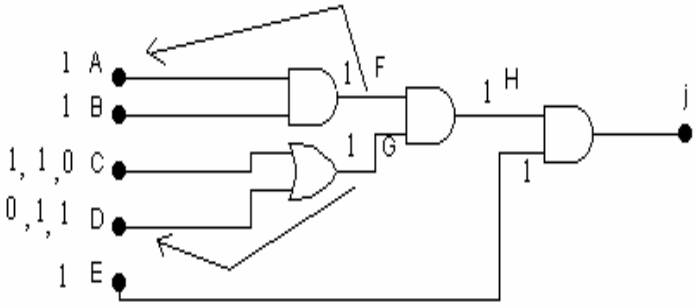


Step 2 propagating the fault from the point H to primary output



coming back to primary input from the point of fault.to have "1" at H it is required to have '1' both in F and G.

Step3,

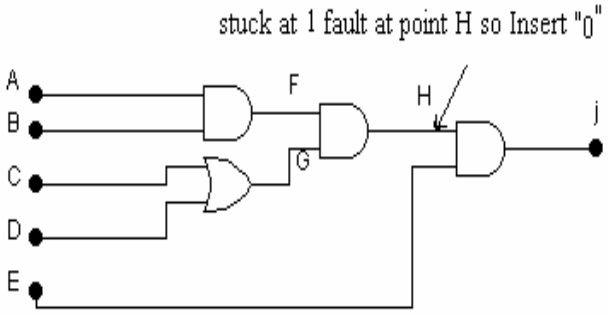


now to get 1 in F A and B must Have 1,1 similarly to get 1 in G, C and D can have any combination 0,1 or 1,1 or 1,0.

Step 4

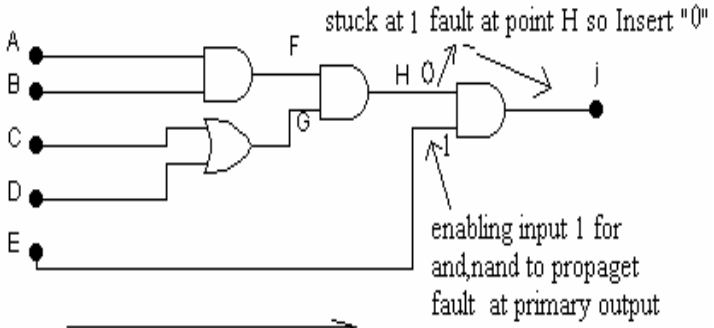
Hence the test vector for stuck at 0 in H node can be {1,1,0,1,1} or {1,1,1,1,1},or {1,1,0,1,1}

Assume Stuck at falut 1 is to be analyzed at point H so the generation of test vector for this is given step by step.

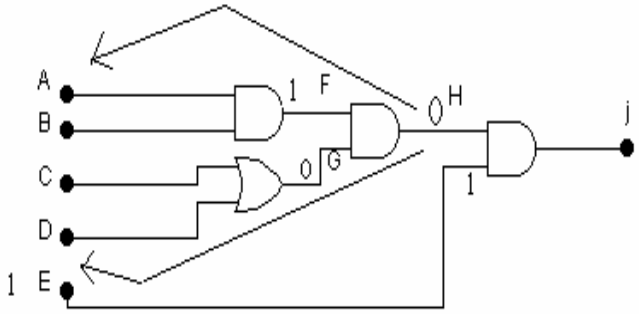


first insert the opposite value at the node where the fault is to be propagated.

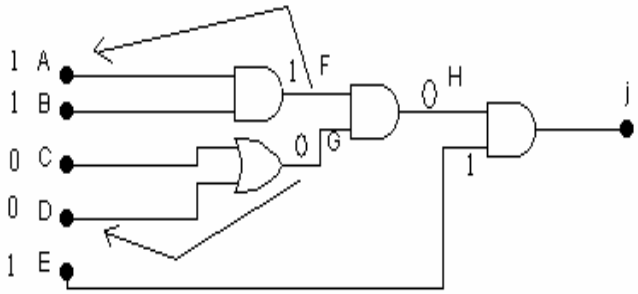
Step 1



Step 2 propagating the fault from the point H to primary output



coming back to primary input from the point of fault.to have "0" at H F and G can have {1,0},{0,1},{0,0} prcesiely if 1 input is 0 other is not important or dont care so it can be written as (0,X) ,(X,0).



now to get 1 in F A and B must Have 1,1 similarly to get 0 in G, C and D has to be 0,0.

Step3,

hence the test vector is{1,1,0,0,1}

similarly if you alter the combination the possible test vectors are:{0,1,X,X,1},{1,0,X,X,1},{0,0,X,X,1},{1,1,0,0,1}.

This basic algorithm of justifying and then propagating a fault works when we can justify nodes without interference from other nodes. This algorithm breaks down when we have **reconvergent fanout** . Figure 5.7) shows another example of justifying and propagating a fault in a circuit with reconvergent fanout. For direct comparison Figure 5.7(b) shows an irredundant circuit, similar to part (a), except the fault signal, B stuck at 1, branches and then reconverges at the inputs to gate U5. The reconvergent fanout in this new circuit breaks our basic algorithm. We now have two sensitized paths that propagate the fault effect to U5. These paths combine to produce a constant '1' at Z, the PO. We have a **multipath sensitization** problem.

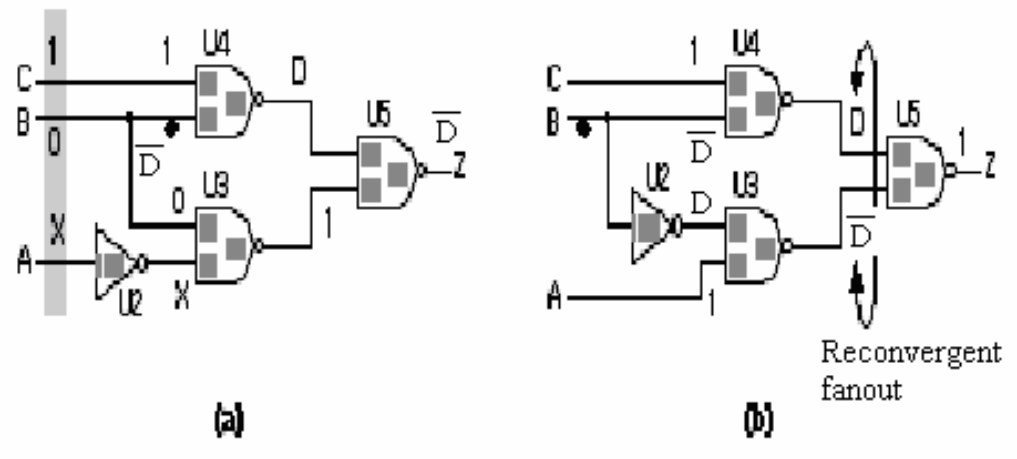


FIGURE 5.7 Reconvergent fanout. (a) Signal B branches and then reconverges at logic gate U5, but the fault U4.A1 stuck at 1 can still be excited and a path sensitized using the basic algorithm of Figure 5.6 . (b) Fault B stuck at 1 branches and then reconverges at gate U5. When we enable the inputs to both gates U3 and U4 we create two sensitized paths that prevent the fault from propagating to the PO (primary output). We can solve this problem by changing A to '0', but this breaks the rules of the algorithm illustrated in Figure 5.6 . The PODEM algorithm solves this problem.

The PODEM Algorithm

The **path-oriented decision making** (**PODEM**) algorithm solves the problem of reconvergent fanout and allows multipath sensitization]. The method is similar to the basic algorithm we have already described except PODEM will retry a step, reversing an incorrect decision. There are four basic steps that we label: ***objective*** **,** ***backtrace*** **,** ***implication*** **, and** ***D-frontier*** . These steps are as follows:

1. Pick an *objective* to set a node to a value. Start with the fault origin as an objective and all other nodes set to 'X'.
2. *Backtrace* to a PI and set it to a value that will help meet the objective.
3. Simulate the network to calculate the effect of fixing the value of the PI (this step is called *implication*). If there is no possibility of sensitizing a path to a PO, then retry by reversing the value of the PI that was set in step 2 and simulate again.
4. Update the *D-frontier* and return to step 1. Stop if the D-frontier reaches a PO.

Figure 5.8 shows example that uses the following iterations of the four steps in the PODEM algorithm:

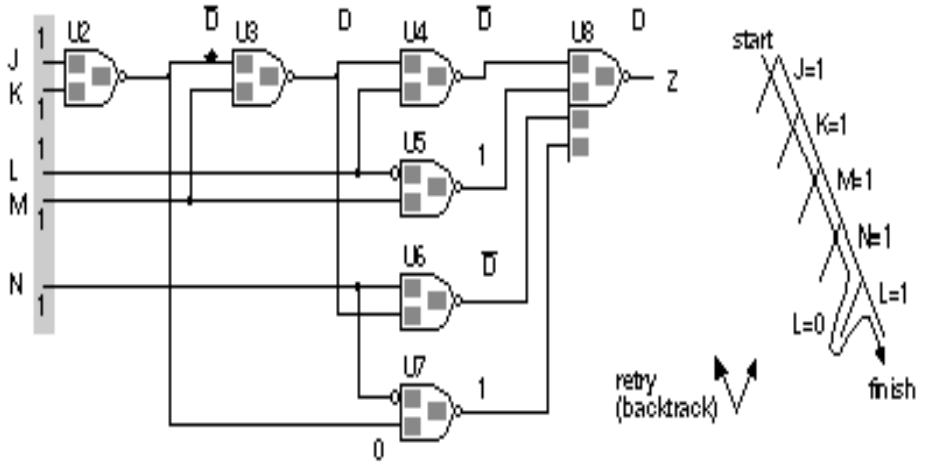
1. We start with activation of the fault as our objective, U3.A2 = '0'. We backtrace to J. We set J = '1'. Since K is still 'X', implication gives us no further information. We have no D-frontier to update.

2. The objective is unchanged, but this time we backtrace to K. We set K = '1'. Implication gives us U2.ZN = '1' (since now J = '1' and K = '1') and therefore U7.ZN = '1'. We still have no D-frontier to update.

3. We set U3.A1 = '1' as our objective in order to propagate the fault through U3. We backtrace to M. We set M = '1'. Implication gives us U2.ZN = '1' and U3.ZN = D. We update the D-frontier to reflect that U4.A2 = D and U6.A1 = D, so the D-frontier is U4 and U6.

4. We pick U6.A2 = '1' as an objective in order to propagate the fault through U6. We backtrace to N. We set N = '1'. Implication gives us U6.ZN = D . We update the D-frontier to reflect that U4.A2 = D and U8.A1 = D , so the D-frontier is U4 and U8.

5. We pick U8.A1 = '1' as an objective in order to propagate the fault through U8. We backtrace to L. We set L = '0'. Implication gives us U5.ZN = '0' and therefore U8.ZN = '0' (this node is Z, the PO). There is then no possible sensitized path to the PO Z. We must have made an incorrect decision, we retry and set L = '1'. Implication now gives us U8.ZN = D and we have propagated the D-frontier to a PO.



Iteration	Objective	Backtrace [1]	Implication	D-frontier
1	U3.A2 = 0	J = 1		
2	U3.A2 = 0	K = 1	U7.ZN = 1	
3	U3.A1 = 1	M = 1	U3.ZN = D	U4, U6
4	U6.A2 = 1	N = 1	U6.ZN = D	U4, U8
5a	U8.A1 = 1	L = 0	U8.ZN = 1	U4, U8
5b	Retry	L = 1	U8.ZN = D	A

[1] Backtrace is not the same as retry or backtrack.

FIGURE 5..8 The PODEM (path-oriented decision making) algorithm.

We can see that the PODEM algorithm proceeds in two phases. In the first phase, iterations 1 and 2 in Figure 5.8, the objective is fixed in order to activate the fault. In the second phase, iterations 3–5, the objective changes in order to propagate the fault. In step 3 of the PODEM algorithm there must be at least one path containing unknown values between the gates of the D-frontier and a PO in order to be able to complete a sensitized path to a PO. This is called the **X-path check** .

You may wonder why there has been no explanation of the backtrace mechanism or how to decide a value for a PI in step 2 of the PODEM algorithm. The decision tree shown in Figure 5.8 shows that it does not matter. PODEM conducts an implicit binary search over all the PIs. If we make an incorrect decision and assign the wrong value to a PI at some step, we will simply need to retry that step. Texts, programs, and articles use the term *backtrace* as we have described it, but then most use the term **backtrack** to describe what we have called a retry, which can be confusing. I also did not explain how to choose the objective in step 1 of the PODEM algorithm. The initial objective is to activate the fault. Subsequently we select a logic gate from the D-frontier and set one of its inputs to the enabling value in an attempt to propagate the fault.

We can use intelligent procedures, based on *controllability* and *observability*, to guide PODEM and reduce the number of incorrect decisions. PODEM is a development of the D-algorithm, and there are several other ATPG algorithms that

are developments of PODEM. One of these is **FAN** (**fanout-oriented test generation**) that removes the need to backtrace all the way to a PI, reducing the search time [Fujiwara and Shimono, 1983; Schulz, Trischler, and Sarfert, 1988]. Algorithms based on the D-algorithm, PODEM, and FAN are the basis of many commercial ATPG systems.

6.9 Design for Testability (DFT)

. *Controllability and Observability:*

Chips are manufactured in such a way that it can be tested for its correct functioning or not. Various design approaches has been discovered to make the chip testable. This design approach is known as Design for Testability (DFT)
For the chip to be tested fully it is necessary that all its input and output pin including internal nodes can be controlled and observed at any time externally. The node or pins which can be direct control by the external input is known as controllability and the node or pin which can be observed externally is known as observability.

In order for an ATPG system to provide a test for a fault on a node it must be possible to both control and observe the behavior of the node. There are both theoretical and practical issues involved in making sure that a design does not contain buried circuits that are impossible to observe and control. A software program that measures the **controllability** (with three *I'* s) and **observability** of nodes in a circuit is useful in conjunction with ATPG software.

There are several different measures for controllability and observability. **Combinational controllability** is defined separately from **sequential controllability** . We also separate **zero-controllability** and **one-controllability**. For example, the **combinational zero-controllability** for a two-input AND gate, Y = AND (X_1 , X_2), is recursively defined in terms of the input controllability values as follows:

$$CC0\ (Y) = \min \{ CC0\ (X_1), CC0\ (X_2) \} + 1 . \qquad \ldots.1$$

We choose the minimum value of the two-input controllability values to reflect the fact that we can justify the output of an AND gate to '0' by setting any input to the control value of '0'. We then add one to this value to reflect the fact that we have passed through an additional level of logic. Incrementing the controllability measures for each level of logic represents a measure of the **logic distance** between two nodes.

We define the **combinational one-controllability** for a two-input AND gate as

$$CC1\ (Y) = CC1(X_1) + CC1\ (X_2) + 1 . \quad (..2)$$

This equation reflects the fact that we need to set all inputs of an AND gate to the enabling value of '1' to justify a '1' at the output. Figure 5.9 (a) illustrates these definitions.

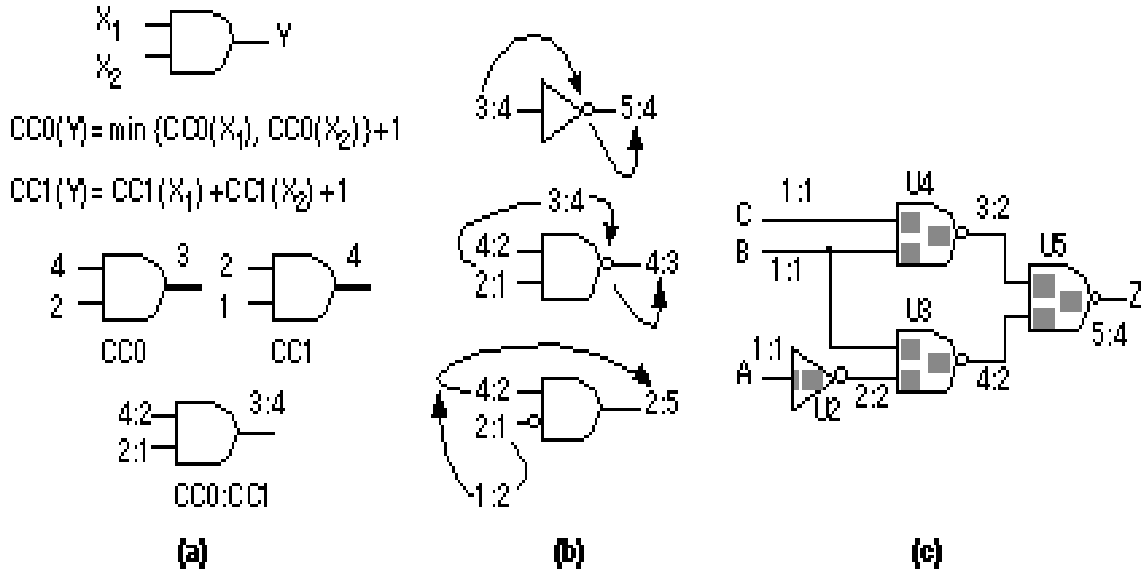


FIGURE 5.9 Controllability measures. (a) Definition of combinational zero-controllability, CC0, and combinational one-controllability, CC1, for a two-input AND gate. (b) Examples of controllability calculations for simple gates, showing intermediate steps. (c) Controllability in a combinational circuit.

An inverter, Y = NOT (X), reverses the controllability values:

$CC1(Y) = CC0(X) + 1$ and $CC0(Y) = CC1(X) + 1$ …3

Since we can construct all other logic cells from combinations of two-input AND gates and inverters we can use Eqs. 1 – 5 to derive their controllability equations. When we do this we only increment the controllability by one for each primitive gate. Thus for a three-input NAND with an inverting input, Y = NAND (X_1, X_2, NOT (X_3)):

$CC0(Y) = CC1(X_1) + CC1(X_2) + CC0(X_3) + 1$,

$CC1(Y) = \min\{ CC0(X_1), CC0(X_2), CC1(X_3) \} + 1$. (…4)

For a two-input NOR, Y = NOR (X_1,X_2) = NOT (AND (NOT (X_1), NOT (X_2)):

$CC1(Y) = \min\{ CC1(X_1), CC1(X_2) \} + 1$,

$CC0(Y) = CC0(X_1) + CC0(X_2) + 1$. (…5)

Figure 5.9 (b) shows examples of controllability calculations. A bubble on a logic gate at the input or output swaps the values of CC1 and CC0. Figure 5.9 (c) shows

how controllability values for a combinational circuit are calculated by working forward from each PI that is defined to have a controllability of one.
We define observability in terms of the controllability measures. The **combinational observability** , OC (X_1), of input X_1 of a two-input AND gate can be expressed in terms of the controllability of the other input CC1 (X_2) and the combinational observability of the output, OC (Y):

$$OC(X_1) = CC1(X_2) + OC(Y) + 1 \quad (\ldots 6)$$

If a node X_1 branches (has fanout) to nodes X_2 and X_3 we choose the most observable of the branches:

$$OC(X_1) = \min \{ O(X_2) + O(X_3) \} \quad (\ldots 7)$$

Figure 5.10 (a) and (b) show the definitions of observability. Figure 5.10 (c) illustrates calculation of observability at a three-input NAND; notice we sum the CC1 values for the other inputs (since the enabling value for a NAND gate is one, the same as for an AND gate). Figure 5.10 (d) shows the calculation of observability working back from the PO which, by definition, has an observability of zero.

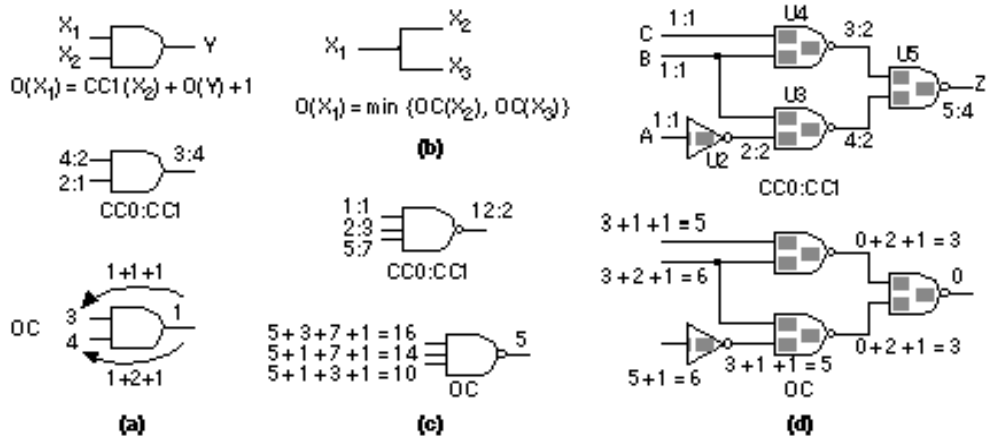


FIGURE 5.10 Observability measures. (a) The combinational observability, $OC(X_1)$, of an input, X_1, to a two-input AND gate defined in terms of the controllability of the other input and the observability of the output. (b) The observability of a fanout node is equal to the observability of the most observable branch. (c) Example of an observability calculation at a three-input NAND gate. (d) The observability of a combinational network can be calculated from the controllability measures, CC0:CC1. The observability of a PO (primary output) is defined to be zero.

Sequential controllability and observability can be measured using similar equations to the combinational measures except that in the sequential measures (SC1, SC0, and OS) we measure logic distance in terms of the layers of sequential logic, not the layers of combinational logic.

6.10 Various design approaches for Design for testable:

Design for testability can be classified as

- AD-HOC based methods
- Scan Methods
- Self test Method.

6.11 AD-HOC Test Methods

AD-Hoc approach refers to collection of various ideas which helps the design to be testable .This section provides a set of practical Design for Testability guidelines classified into three types: **test generation, test application and avoiding timing problems.**

6.11.1Adding internal nodes to Improve Controllability and Observability:

All "design for test" methods ensure that a design has enough observability and controllability to provide for a complete and efficient testing. When a node has difficult access from primary inputs or outputs (pads of the circuit), a very efficient method is to add internal pads acceding to this kind of node in order, for instance, to control block B2 and observe block B1 with a probe. It is easy to observe block B1 by adding a pad just on its output, without breaking the link between the two blocks. The control of the block B2 means to set a 0 or a 1 to its input, and also to be transparent to the link B1-B2. The logic functions of this purpose are a NOR- gate, transparent to a zero, and a NAND-gate, transparent to a one. By this way the control of B2 is possible across these two gates. Another implementation of this cell

is based on pass-gates multiplexers performing the same function, but with less transistors than with the NAND and NOR gates (8 instead of 12).

The simple optimization of observation and control is not enough to guarantee a full testability of the blocks B1 and B2. This technique has to be completed with some other techniques of testing depending on the internal structures of blocks B1 and B2.

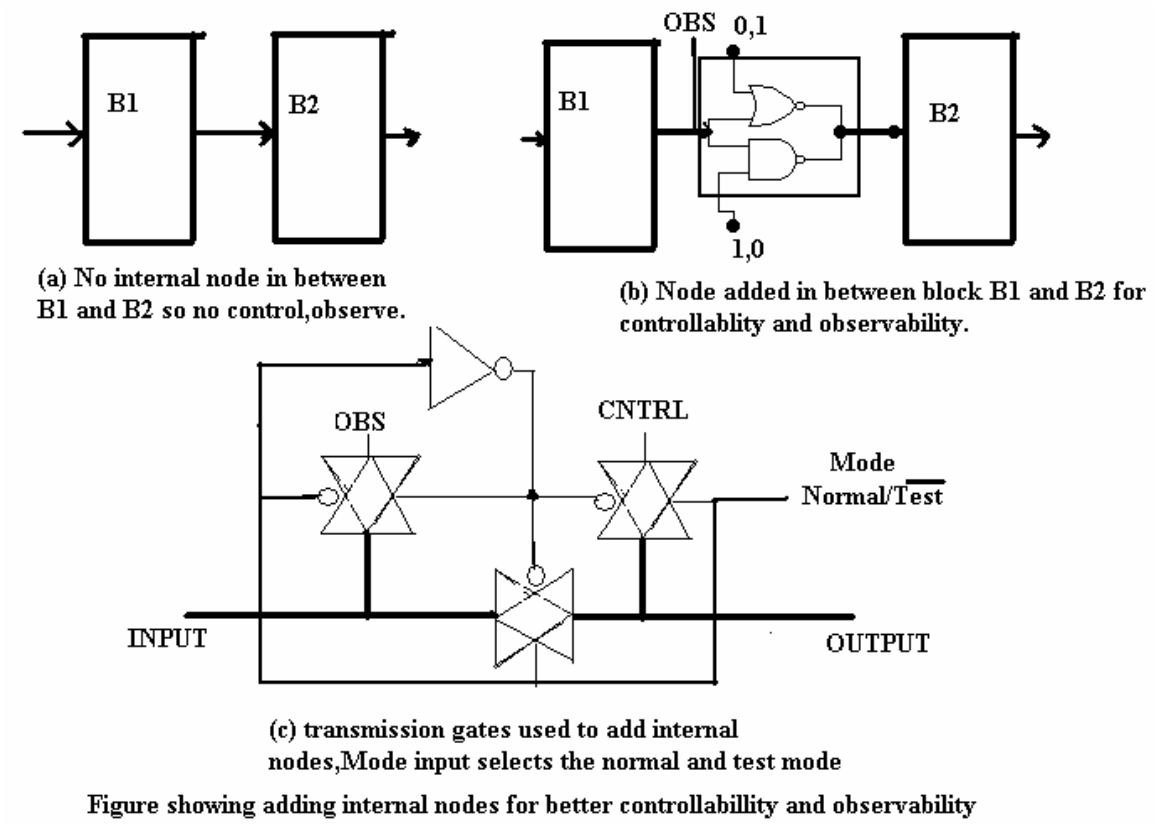


Figure-5.11 Internal Nodes added to Improve Controllability and Observability.

6.11.2 Use Multiplexers

This technique is an extension of the precedent, while multiplexers are used in case of limitation of primary inputs and outputs.

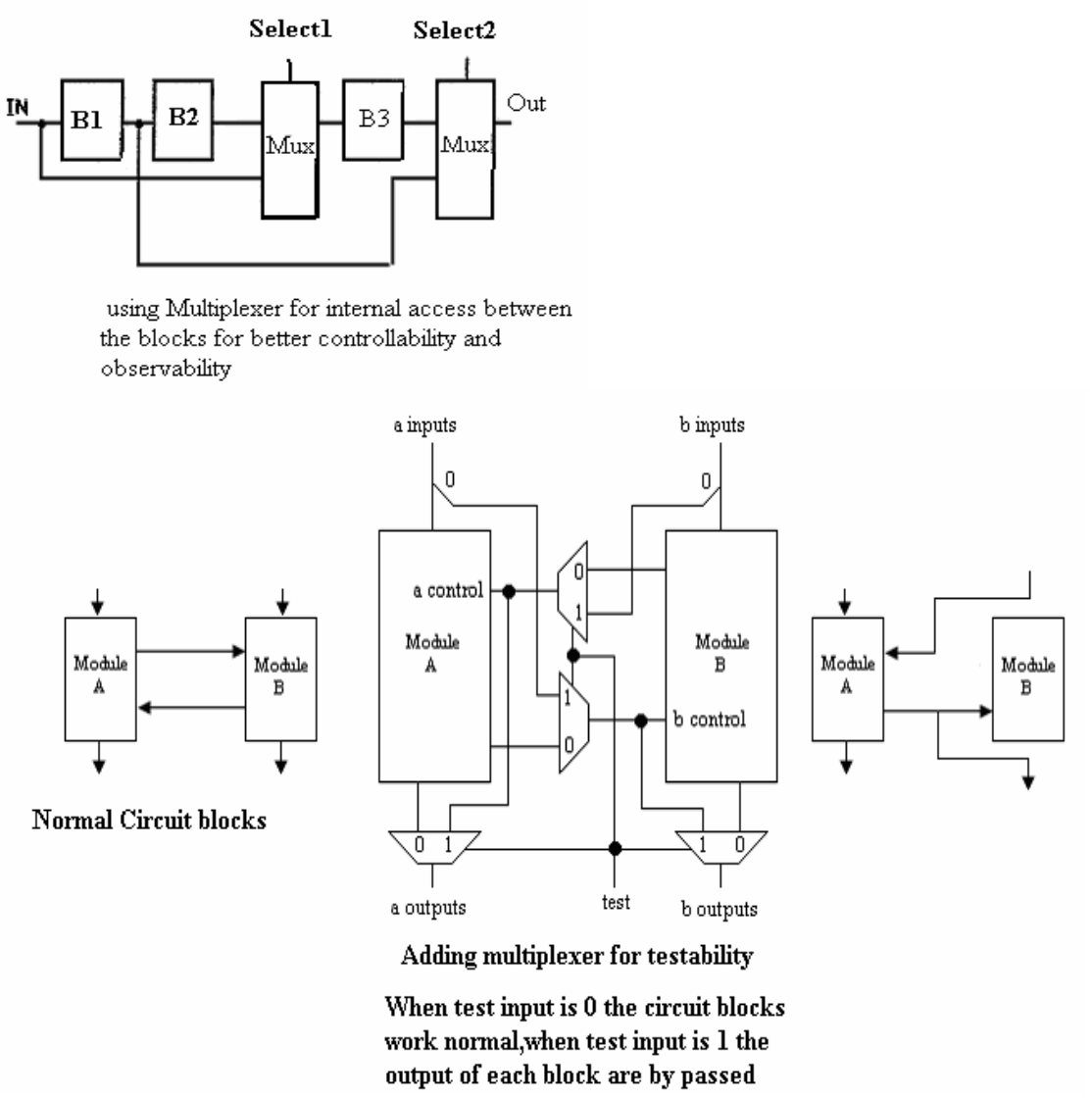


Figure-5.12 Adding multiplexer For DFT

In this case the major penalties are extra devices and propagation delays due to multiplexers. Demultiplexers are also used to improve observability. Using multiplexers and demultiplexers allows internal access of blocks separately from each other, which is the basis of techniques based on partitioning or bypassing blocks to observe or control separately other blocks.

6.11.3 Partition Large Circuits

Partitioning large circuits into smaller sub-circuits reduces the test-generation effort. The test- generation effort for a general purpose circuit of n gates is assumed to be

proportional to somewhere between n2 and n3. If the circuit is partitioned into two sub-circuits, then the amount of test generation effort is reduced correspondingly.

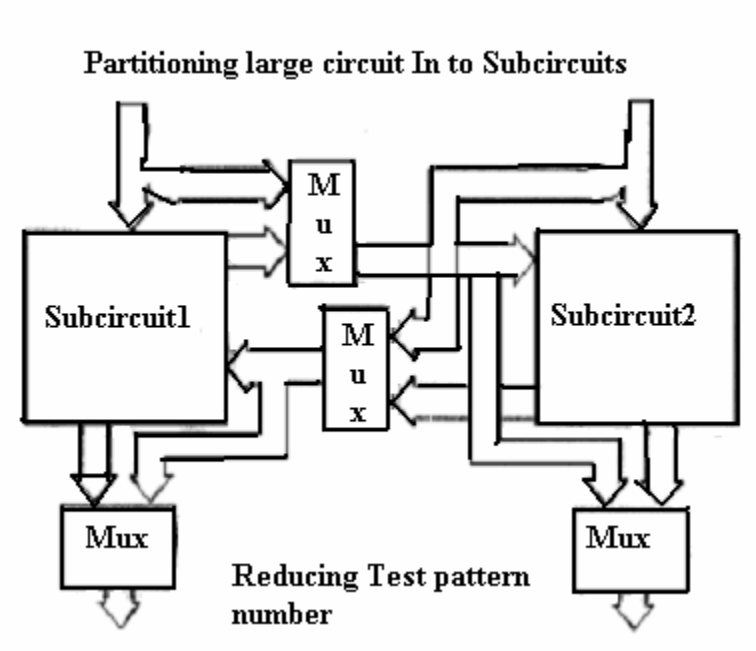


Figure-5.13: Partitioning large circuit in to subcircuits.

For example the SN7480 full adder exhaustive testing requires 512 tests, while a full test after partitioning into four sub-circuits, for SA0 and SA1 faults, requires 24 tests. Logical partitioning of a circuit should be based on recognizable sub-functions and can be achieved physically by incorporating some facilities to isolate and control clock lines, reset lines and power supply lines. The multiplexers can be massively used to separate sub-circuits without changing the function of the global circuit.

6.11.4 Divide Long Counter Chains

Based on the same principle of partitioning, the counters are sequential elements that need a large number of vectors to be fully tested. The partitioning of a long counter corresponds to its division into sub-counters.

The full test of a 16-bit counter requires the application of 216 + 1 = 65537 clock pulses. If this counter is divided into two 8-bit counters, then each counter can be tested separately, and the total test time is reduced 128 times (27). This is also useful if there are subsequent requirements to set the counter to a particular count for tests associated with other parts of the circuit : pre-loading facilities.

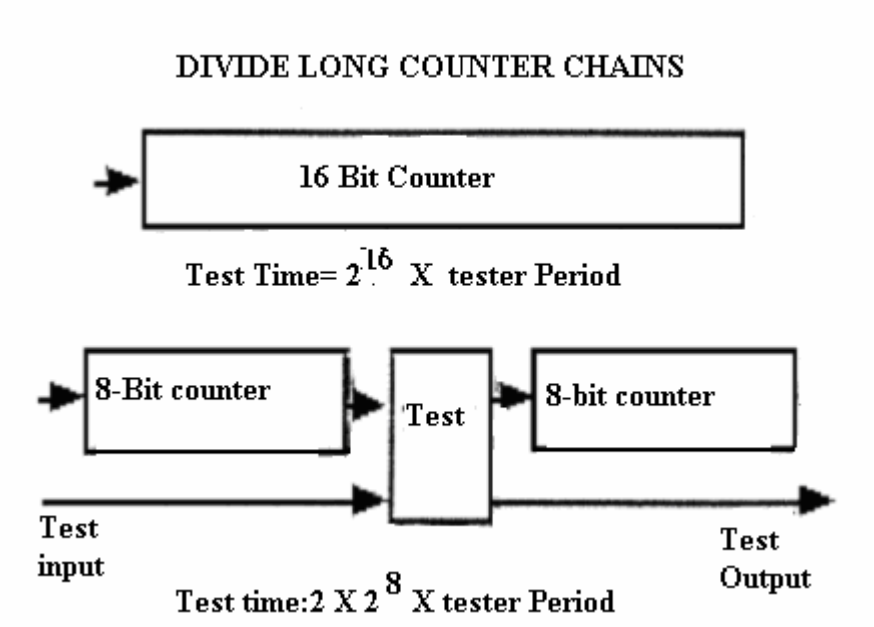


Figure-5.14: Dividing long Counter chains:

6.11.5 Use Bused Structure

This approach is related, by structure, to partitioning technique. It is very useful for microprocessor-like circuits. Using this structure allows the external tester the access of three buses, which go to many different modules.

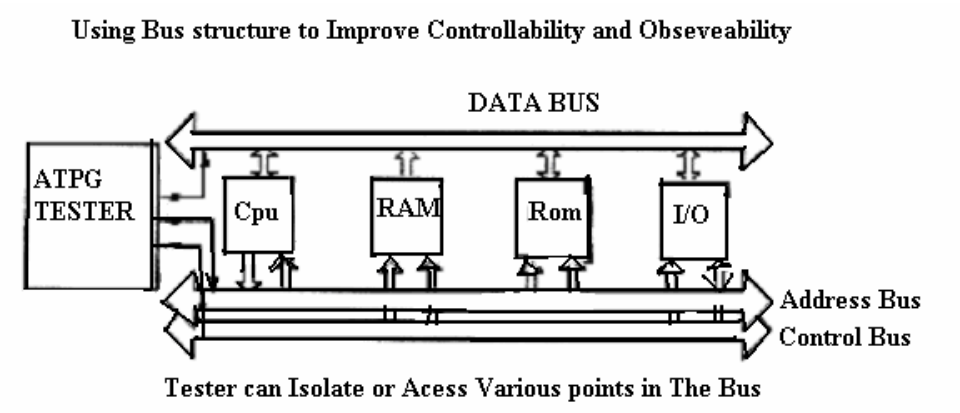


Figure-5.15: Bus structure:

The tester can then disconnect any module from the buses by putting its output into a high- impedance state. Test patterns can then be applied to each module separately.

6.11.6 Separate Analog and Digital Circuits

Testing analog circuit requires a completely different strategy than for digital circuit. Also the sharp edges of digital signals can cause cross-talk problem to the analog lines, if they are close to each other.

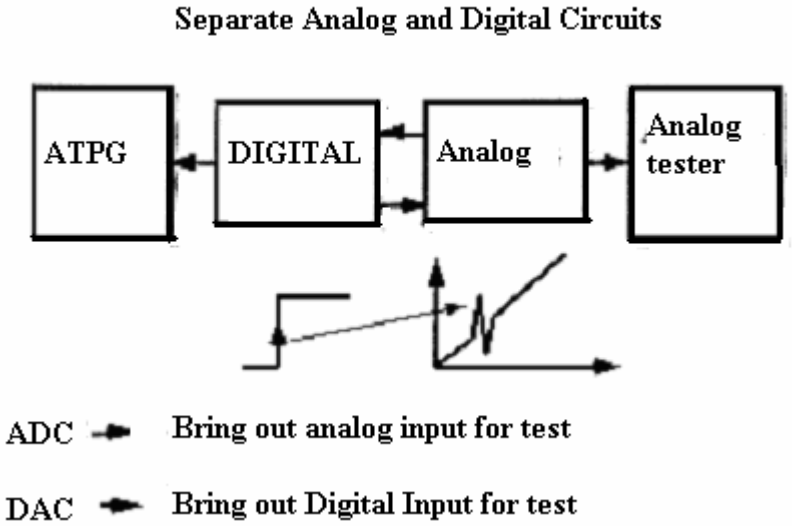


Figure-5.16: separation of analog and digital Circuit.

If it is necessary to route digital signals near analog lines, then the digital lines should be properly balanced and shielded. Also, in the cases of circuits like Analog-Digital converters, it is better to bring out analog signals for observation before conversion. For Digital-Analog converters, digital signals are to be brought out also for observation before conversion.

6.11.7 Bypassing Techniques

Bypassing a sub-circuit consists in propagating the sub-circuit inputs signals directly to the outputs. The aim of this technique is to bypass a sub-circuit (part of a global circuit) in order to access another sub-circuit to be tested. The partitioning technique is based on bypassing technique and they both use multiplexers to perform two different methods.

In the bypassing technique sub-circuits can be then tested exhaustively, by controlling multiplexers in the whole circuit. To speed-up the test, some sub-circuits are tested simultaneously if the propagation paths are associated with other disjoint or separated sub- circuits.

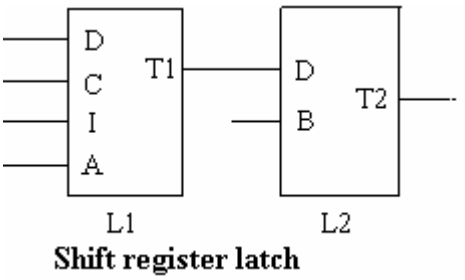
6.12 Scan Test Methods:

Sequential logic poses a very difficult ATPG problem. Consider the example of a 32-bit counter with a final carry. If the designer included a reset, we have to clock the counter 2^{32} (approximately $4 \square 10^{9}$) times to check the carry logic. Using a 1 MHz tester clock this requires $4 \square 10^{3}$ seconds, 1 hour, or (at approximately $0.25 per second) $1,000 of tester time. Consider a 16-bit state machine implemented using a one-hot state register with 16 D flip-flops. If the designer did not include a reset we have a very complicated initialization problem. A sequential ATPG algorithm must consider over 2000 states when constructing sequential test vectors. In an ad hoc approach to testing we could construct special reset circuits or create manual test vectors to deal with these special situations, one at a time, as they arise. Instead we can take a **structured test** approach (also called **design for test**, though this term covers a wider field).

We can automatically generate test vectors for combinational logic, but ATPG is much harder for sequential logic. Therefore the most common sequential structured test approach converts sequential logic to combinational logic. In full-scan design we replace every sequential element with a scan flip-flop. The result is an internal form of boundary scan and, if we wish, we can use the IEEE 1149.1 TAP to access (and the boundary-scan controller to control) an internal-scan chain.

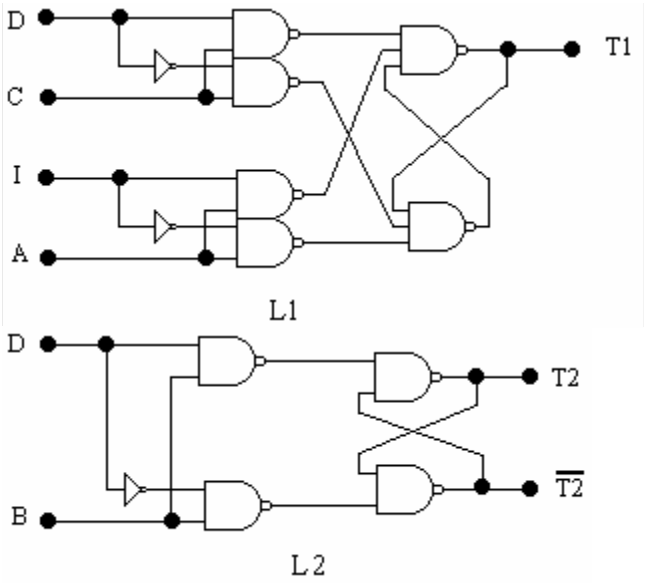
6.12.1 Scan Path

The goal of the scan path technique is to reconfigure a sequential circuit, for the purpose of testing, into a combinational circuit. Since a sequential circuit is based on a combinational circuit and some storage elements, the technique of scan path consists in connecting together all the storage elements to form a long serial shift register. Thus the internal state of the circuit can be observed and controlled by shifting (scanning) out the contents of the storage elements. The shift register is then called a scan path. The popular approach of scan based technique is **Level Sensitive Scan design (LSSD**). The basic building block of LSSD is **Shift Register latch (SRL).**SRL consists of two Latches L1 and L2.



Shift register latch

The internal diagram for L1 and L2 is shown in figure



L1 consists of data port D which is normal data and is enabled by Control signal c , the serial input data port is I which is used in test mode and is controlled by control signal A. remember A and C cannot be enable simultaneously. The data is passed from T1 to T2 when the enable signal B is asserted.

In normal operation D is the normal input and T2 output is normal output. Shift Register latches are connected in series by using the T2 output and I input of successive Latch as shown in figure 5.17 for serial scan. The serial input data is given to the first set of SRL and the output from the Latch QA1 is connected to the Input (I) of second test of SRL, similarly the QA2 is connected to input of Third SRL and so on. In this manner the chain is created with this registers and finally the output is obtained which has passed through points QB1,QB2,QB3 etc.

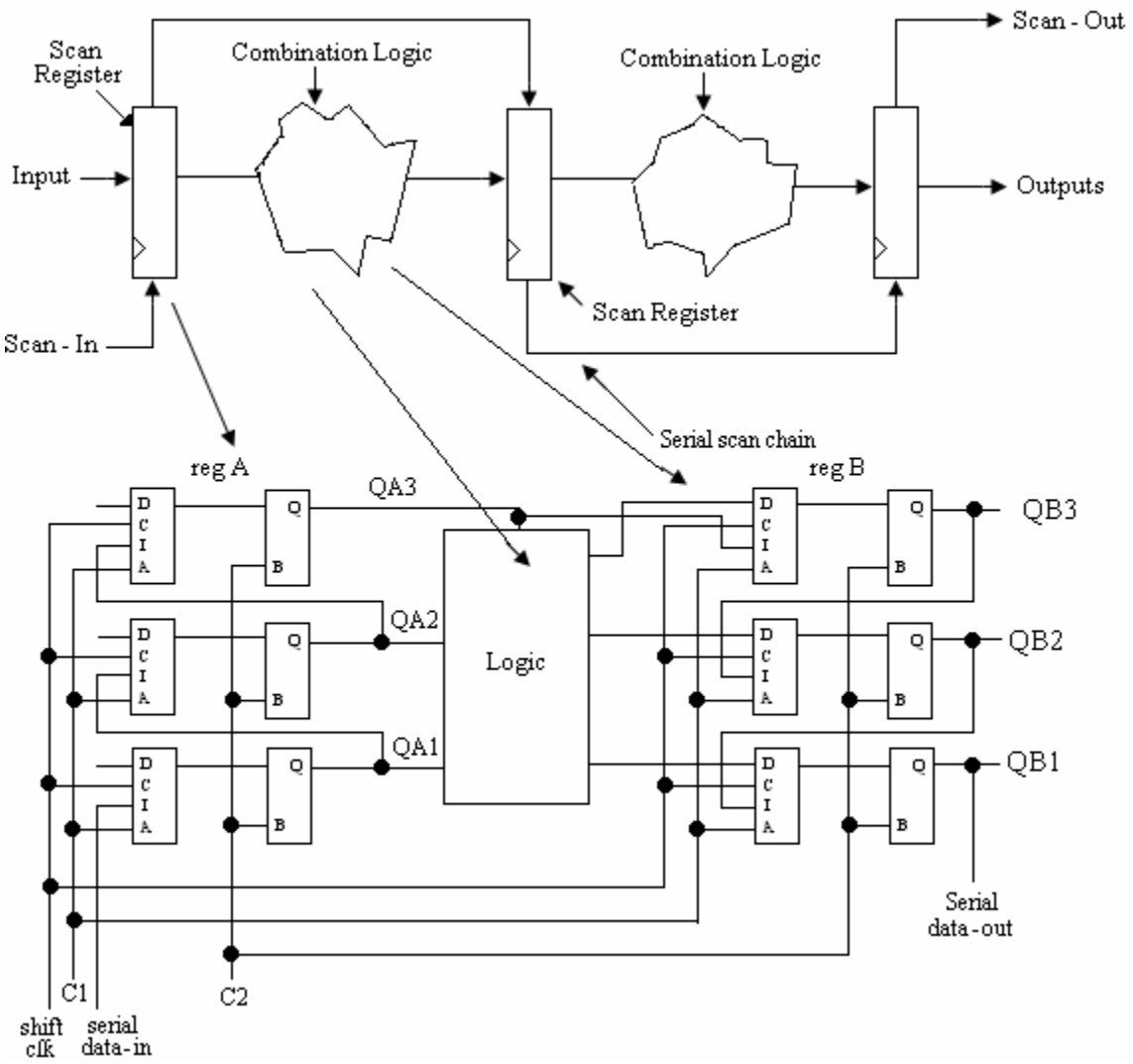


Figure-5.17 Serial Scan test method.

The storage elements can either be D, J-K, or R-S types of flip-flops, but simple latches cannot be used in scan path. However, the structure of storage elements is slightly different than classical ones. Generally the selection of the input source is achieved using a multiplexer on the data input controlled by an external mode signal. This multiplexer is integrated into the D-flip-flop, in our case; the D-flip-flop is then called MD-flip-flop (multiplexed-flip-flop).

The sequential circuit containing a scan path has two modes of operation : a normal mode and a test mode which configure the storage elements in the scan path.

In the normal mode, the storage elements are connected to the combinational circuit, in the loops of the global sequential circuit, which is considered then as a finite state machine.

In the test mode, the loops are broken and the storage elements are connected together as a serial shift register (scan path), receiving the same clock signal. The input of the scan path is called scan-in and the output scan-out. Several scan paths can be implemented in one same complex circuit if it is necessary, though having several scan-in inputs and scan-out outputs.

A large sequential circuit can be partitioned into sub-circuits, containing combinational sub-circuits, associated with one scan path each. Efficiency of the test pattern generation for a combinational sub-circuit is greatly improved by partitioning, since its depth is reduced.

Before applying test patterns, the shift register itself has to be verified by shifting in all ones i.e. 111...11, or zeros i.e. 000...00, and comparing.

The method of testing a circuit with the scan path is as follows:

1. Set test mode signal, flip-flops accept data from input scan-in
2. Verify the scan path by shifting in and out test data
3. Set the shift register to an initial state
4. Apply a test pattern to the primary inputs of the circuit
5. Set normal mode, the circuit settles and can monitor the primary outputs of the circuit
6. Activate the circuit clock for one cycle
7. Return to test mode
8. Scan out the contents of the registers, simultaneously scan in the next pattern

6.13 BUILT IN SELF TEST:

This approach uses the technique whereby the circuit itself generates the test vector and verifies the correct operation. The widely used technique for self test is Signature analysis or cyclic redundancy check.

Signature analysis and Built in Self test (BILBO):

Signature analysis technique uses the pseudo random sequence generator (PRSG) to generate the input test vectors which are then applied to the input of the combinational logic circuit .the output coming from the combinational logic circuit is

than analyzed with the signature analyzer.

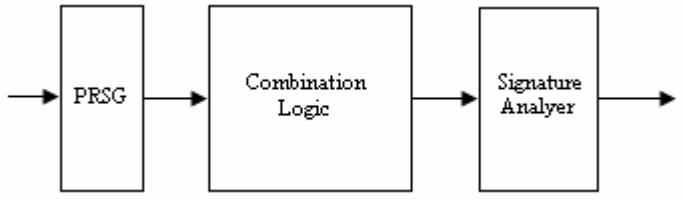


Figure 5.18 Built In self Test

A pseudo random sequence generator is constructed using linear feed back shift register (LFSR).Linear Feed back Shift register consists of number of 1 bit register which are connected in series to form shift register. Pseudo random sequence generator implements a polynomial of some length N. to form the PRSG the output of the certain register are feed to the XOR gate and the output of XOR gate is given to the input of LFSR.Depending upon the output taken from the register and xored it ,different polynomial is formed. For n bit Register the LFSR will cycle through 2n-1 states before repeating the sequence.

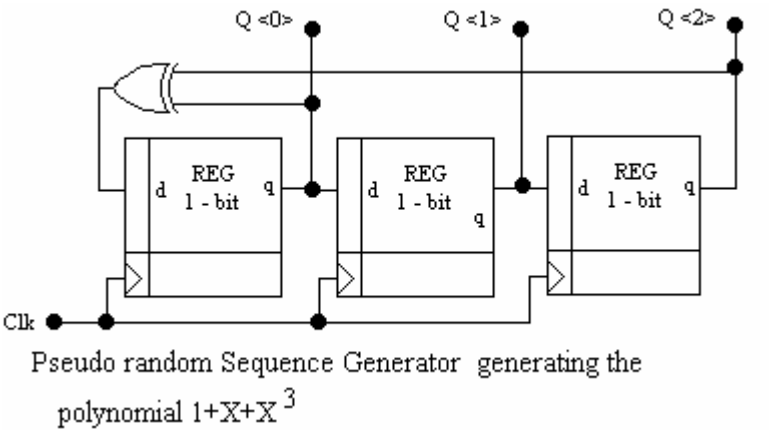


Figure 5.19 PRSG constructed from LFSR

a signature analyzer is constructed by cyclically adding the output of the circuit to the shift register or LFSR if successive blocks are to be tested in like manner. Signature analysis can be merged with Scan technique to form a structure known as Built in Logic Block observation (BILBO).
The BILBO structure consisting of 3 register is shown in figure (5.20). This block is capable of performing following operation depending upon the mode.

Mode	C0	C1	
A	0	0	scan Mode
B	0	1	Reset
C	1	0	PRSG or Signature analyzer
D	1	1	Parallel Registers.

When C0 and C1 is 0,0 (MODE A) this block acts as scan Register, when both C0 and C1 are 1 (MODE D) it acts as parallel register.in mode C it acts as PRSG or signature analyzer.

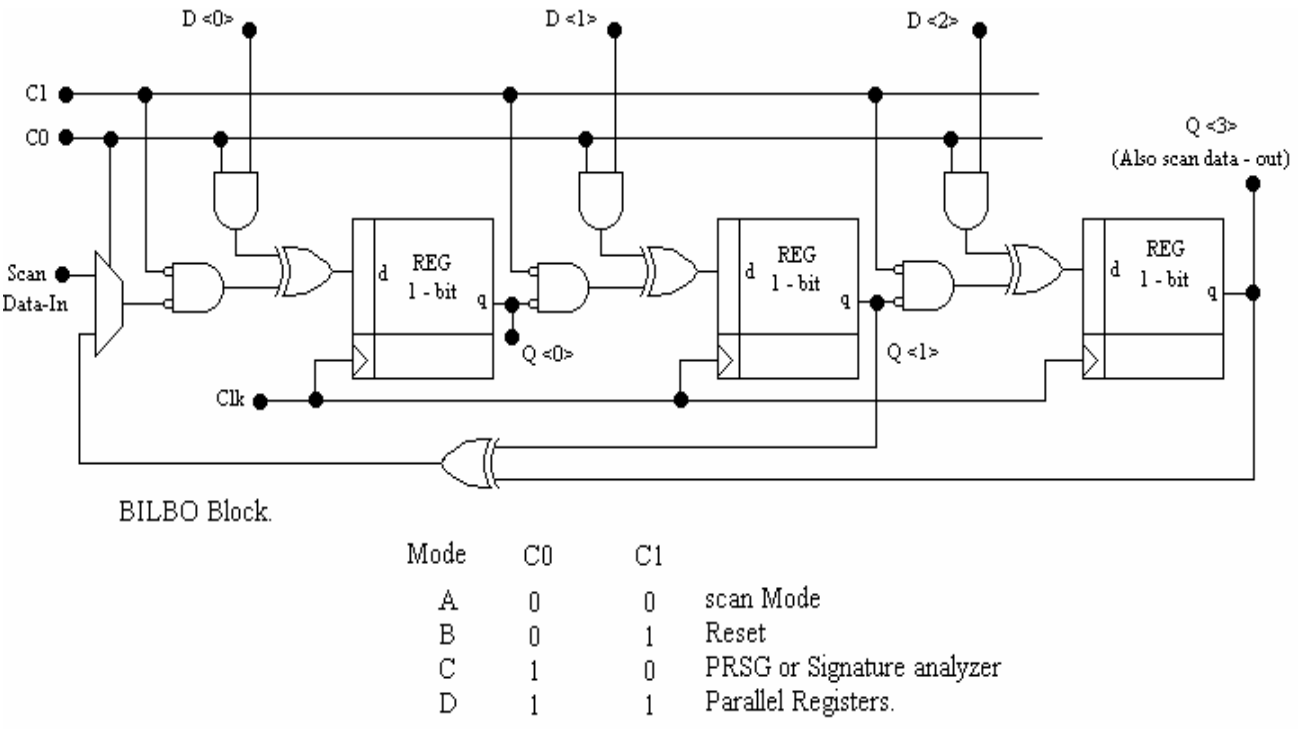


Figure 5.20 BILBO showing various Modes of operatrion.

6.14 Boundary Scan Test (BST)

Boundary Scan Test (BST) is a technique involving scan path and self-testing techniques to resolve the problem of testing boards carrying VLSI integrated circuits and/or surface mounted devices (SMD).

Printed circuit boards (PCB) are becoming very dense and complex, especially with SMD circuits, that most test equipment cannot guarantee a good fault coverage. It is possible to test ICs in dual-in-line packages (DIPs) with 0.1 inch (2.5 mm) lead spacing on low-density boards using a **bed-of-nails tester** with probes that contact test points underneath the board. Mechanical testing becomes difficult with board trace widths and separations below 0.1 mm or 100 mm, package-pin separations of 0.3 mm or less, packages with 200 or more pins, surface-mount packages on both sides of the board, and multilayer boards.

Boundary-scan test (BST) is a method for testing boards using a four-wire interface (five wires with an optional master reset signal). A good analogy would be the RS-232 interface for PCs. The BST standard interface was designed to test boards, but it is also useful to test ASICs. The BST interface provides a standard means of communicating with test circuits on-board an ASIC. We do need to include extra circuits on an ASIC in order to use BST. This is an example of increasing the cost and complexity (as well as potentially reducing the performance) of an ASIC to reduce the cost of testing the ASIC and the system.

BST consists in placing a scan path (shift register) adjacent to each component pin and to interconnect the cells in order to form a chain around the border of the circuit. The BST circuits contained on one board are then connected together to form a single path through the board.

The boundary scan path is provided with serial input and output pads and appropriate clock pads which make it possible to :

- Test the interconnections between the various chip
- Deliver test data to the chips on board for self-testing
- Test the chips themselves with internal self- test

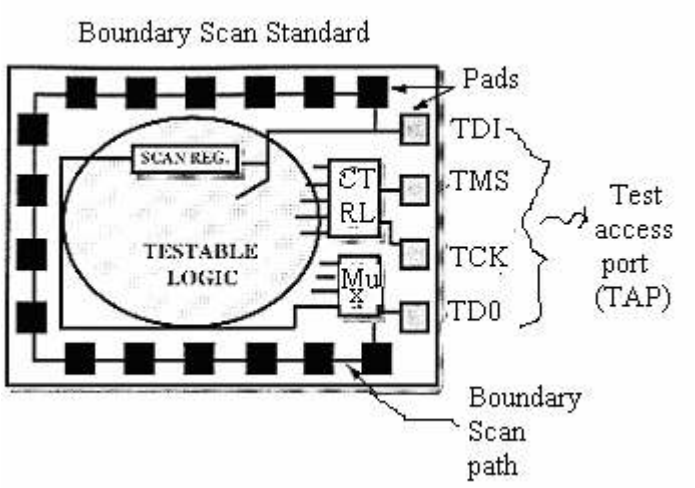


Figure-5.21: Boundary scan test standard.

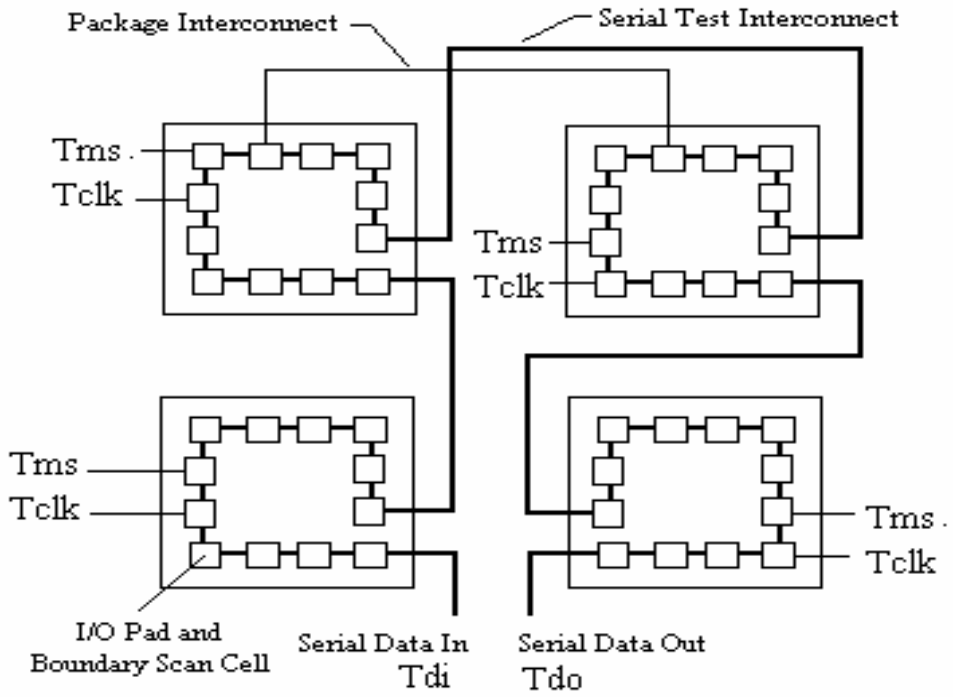
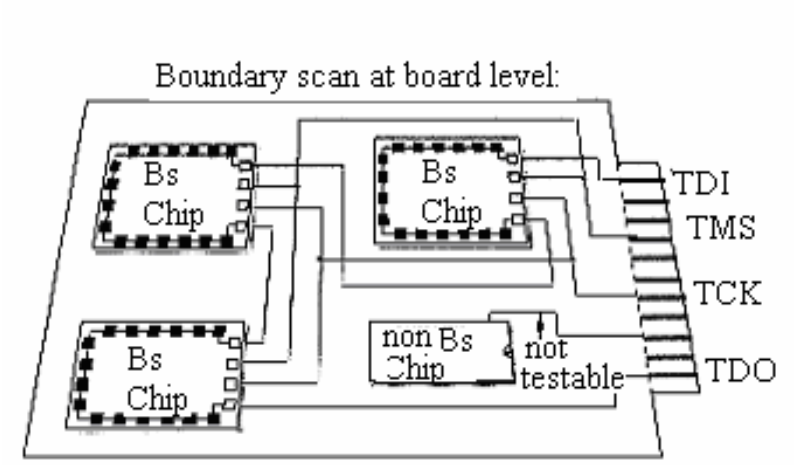


Figure-5.22: Boundary scan test for multiple Chips in PCB.



- Figure-5.23: connecting various chips for boundary scan.

The advantages of Boundary scan techniques are as follows :

- No need for complex testers in PCB testing
- Test engineers work is simplified and more efficient
- Time to spend on test pattern generation and application is reduced
- Fault coverage is greatly increased.

BS Techniques are grouped by the IEEE Standard Organization in a "standard test access port and boundary scan architecture", namely IEEE P1149.1-1990. The Joint Test Action Group (JTAG), formed basically in 1986 at Philips, is an international committee composed of IC manufacturers who have set the technical development of the IEEE P1149 standard and promoted its use by all sectors of electronics industry.The IEEE 1149 is a family of overall testability bus standards, defined by the Joint Test Action Group (JTAG), formed basically in 1986 at Philips. JTAG is an international committee composed of European and American IC manufacturers. The "standard Test Access Port and Boundary Scan architecture", namely IEEE P1149.1 accepted by the IEEE standard committee in February1990, is the first one of this family. Several other ongoing standards are developed and suggested as drafts to the technical committee of the IEEE 1149 standard in order to promote their use by all sectors of electronics industry.

In 1985 a group of European manufacturers formed the **Joint European Test Action Group** (**JETAG**) to study board testing. With the addition of North American companies, JETAG became the **Joint Test Action Group** (**JTAG**) in 1986. The JTAG 2.0 test standard formed the basis of the **IEEE Standard 1149.1 Test Port and Boundary-Scan Architecture** [IEEE 1149.1b, 1994], approved in February 1990 and also approved as a standard by the American National Standards Institute (ANSI) in August 1990 [Bleeker, v. d. Eijnden, and de Jong, 1993; Maunder and Tulloss, 1990; Parker, 1992]. The IEEE standard is still often referred to as JTAG, although there are important differences between the last JTAG specification (version 2.0) and the IEEE 1149.1 standard.

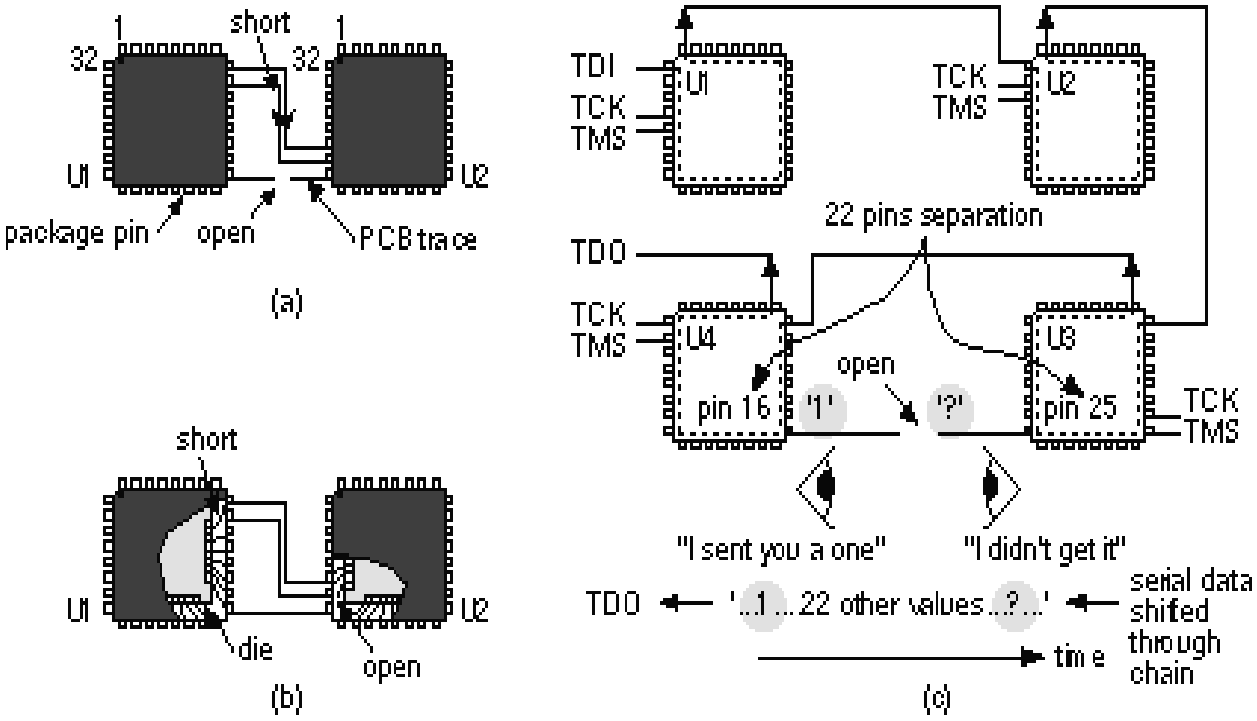
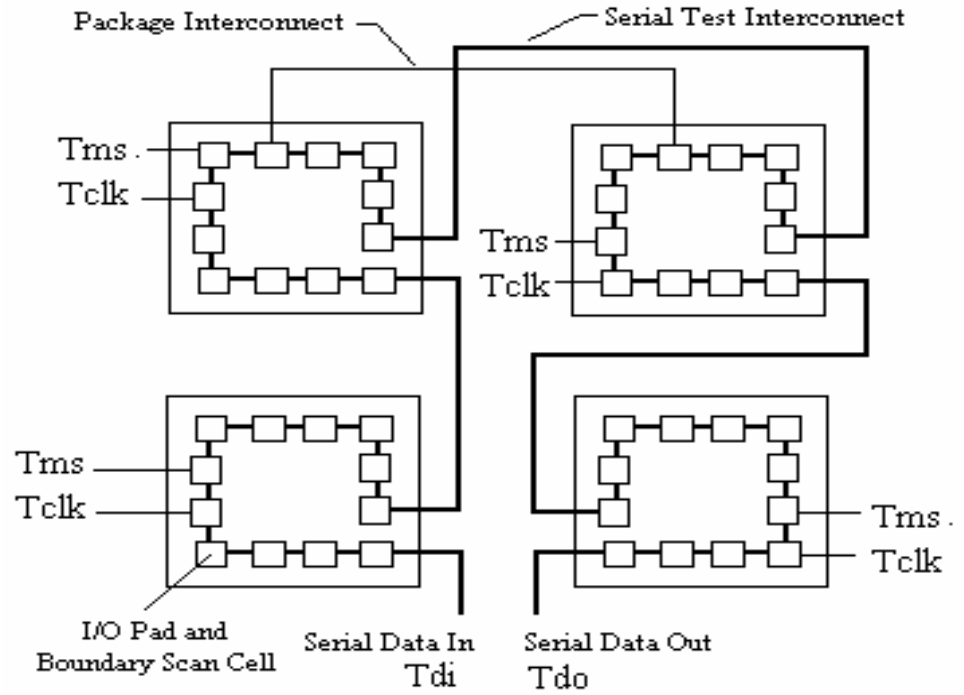


FIGURE 5.24. IEEE 1149.1 boundary scan. (a) Boundary scan is intended to check for shorts or opens between ICs mounted on a board. (b) Shorts and opens may also occur inside the IC package. (c) The boundary-scan architecture is a long chain of shift registers allowing data to be sent over all the connections between the ICs on a board.

Figure 5.24 (a) illustrates failures that may occur on a PCB due to shorts or opens in the copper traces on the board. Less frequently, failures in the ASIC package may also arise from shorts and opens in the wire bonds between the die and the package frame (Figure 5.24 b). Failures in an ASIC package that occur during ASIC fabrication are caught by the ASIC production test, but stress during automated handling and board assembly may cause package failures. Figure 5.24 (c) shows how a group of ASICs are linked together in boundary-scan testing. To detect the failures shown in Figure 5.24 (a) or (b) manufacturers use boundary scan to test every connection between ASICs on a board. During boundary scan, test data is loaded into each ASIC and then driven onto the board traces. Each ASIC monitors its inputs, captures the data received, and then shifts the captured data out. Any defects in the board or ASIC connections will show up as a discrepancy between expected and actual measured continuity data.

In order to include BST on an ASIC, we add a special logic cell to each ASIC I/O pad. These cells are joined together to form a chain and create a boundary-scan shift register that extends around each ASIC. The input to a boundary-scan shift register is the **test-data input** (**TDI**). The output of a boundary-scan shift register is

the **test-data output** (**TDO**). These boundary-scan shift registers are then linked in a serial fashion with the boundary-scan shift registers on other ASICs to form one long boundary-scan shift register. The boundary-scan shift register in each ASIC is one of several **test-data registers** (**TDR**) that may be included in each ASIC. All the TDRs in an ASIC are connected directly between the TDI and TDO ports. A special register that decodes instructions provides a way to select a particular TDR and control operation of the boundary-scan test process.



Controlling all of the operations involved in selecting registers, loading data, performing a test, and shifting out results are the **test clock** (**TCK**) and **test-mode select** (**TMS**). The boundary-scan standard specifies a four-wire test interface using the four signals: TDI, TDO, TCK, and TMS. These four dedicated signals, the **test-access port** (**TAP**), are connected to the TAP controller inside each ASIC. The TAP controller is a state machine clocked on the rising edge of TCK, and with state transitions controlled by the TMS signal. The **test-reset input signal** (**TRST*** , **nTRST** , or **TRST** —always an active-low signal) is an optional (fifth) dedicated interface pin to reset the TAP controller.

Normally the boundary-scan shift-register cells at each ASIC I/O pad are transparent, allowing signals to pass between the I/O pad and the core logic. When an ASIC is put into boundary-scan test mode, we first tell the TAP controller which TDR to select. The TAP controller then tells each boundary-scan shift register in the appropriate TDR either to capture input data, to shift data to the neighboring cell, or to output data.

There are many acronyms in the IEEE 1149.1 standard (referred to as " **dot one** "); Table below provides a list of the most common terms.

TABLE :Boundary-scan terminology.

Acronym	**Meaning**	**Explanation**

BR	Bypass register	A TDR, directly connects TDI and TDO, bypassing BSR
BSC	Boundary-scan cell	Each I/O pad has a BSC to monitor signals
BSR	Boundary-scan register	A TDR, a shift register formed from a chain of BSCs
BST	Boundary-scan test	Not to be confused with BIST (built-in self-test)
IDCODE	Device-identification register	Optional TDR, contains manufacturer and part number
IR	Instruction register	Holds a BST instruction, provides control signals
JTAG	Joint Test Action Group	The organization that developed boundary scan
TAP	Test-access port	Four- (or five-)wire test interface to an ASIC
TCK	Test clock	A TAP wire, the clock that controls BST operation
TDI	Test-data input	A TAP wire, the input to the IR and TDRs
TDO	Test-data output	A TAP wire, the output from the IR and TDRs
TDR	Test-data register	Group of BST registers: IDCODE, BR, BSR
TMS	Test-mode select	A TAP wire, together with TCK controls the BST state
TRST* or nTRST	Test-reset input signal	Optional TAP wire, resets the TAP controller (active-low)

5.14.1 BST Cells

Figure 5.25 shows a **data-register cell** (**DR cell**) that may be used to implement any of the TDRs. The most common DR cell is a **boundary-scan cell** (**BS cell** , or **BSC**), or **boundary-register cell** (this last name is not abbreviated to BR cell, since this term is reserved for another type of cell)

A BSC contains two sequential elements. The **capture flip-flop** or **capture register** is part of a shift register formed by series connection of BSCs. The **update flip-flop** , or **update latch** , is normally drawn as an edge-triggered D flip-flop, though it may be a transparent latch. The inputs to a BSC are: **scan in** (**serial in** or **SI**); **data in** (**parallel in** or **PI**); and a control signal, **mode** (also called **test / normal**). The BSC outputs are: **scan out** (**serial out** or **SO**); **data out** (**parallel out** or **PO**). The BSC in Figure 14.2 is **reversible** and can be used for both chip inputs and outputs. Thus data_in may be connected to a pad and data_out to the core logic or vice versa.

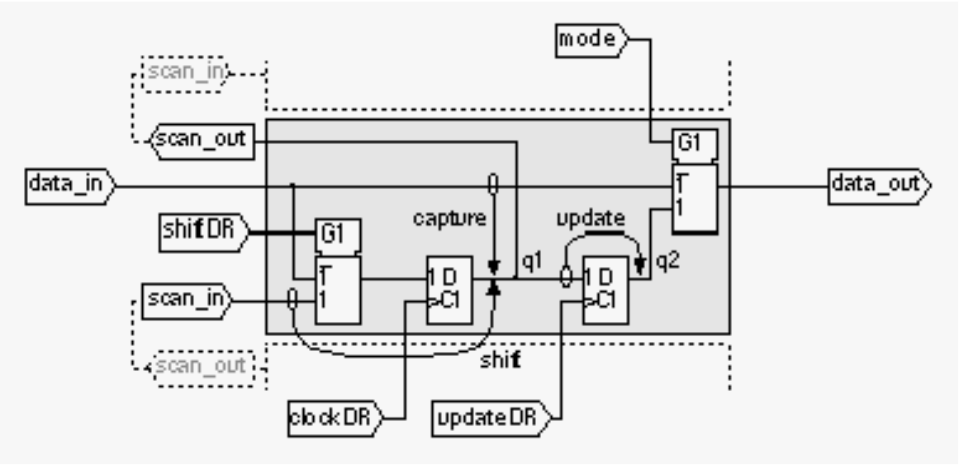


FIGURE 5.25 A DR (data register) cell. The most common use of this cell is as a boundary-scan cell (BSC).

The IEEE 1149.1 standard shows the sequential logic in a BSC controlled by the gated clocks: clockDR (whose positive edge occurs at the positive edge of TCK) and updateDR (whose positive edge occurs at the negative edge of TCK). The IEEE 1149.1 schematics illustrate the standard but do not define how circuits should be implemented. The function of the circuit (and its model) follows the IEEE 1149.1 standard and many other published schematics, but this is not necessarily the best, or even a safe, implementation. For example, as drawn here, signals clockDR and updateDR are gated clocks—normally to be avoided if possible. The update sequential element is shown as an edge-triggered D flip-flop but may be implemented using a latch.

Figure 5.26 shows a **bypass-register cell** (**BR cell**). The BR inputs and outputs, scan in (serial in, SI) and scan out (serial out, SO), have the same names as the DR cell ports, but DR cells and BR cells are not directly connected.

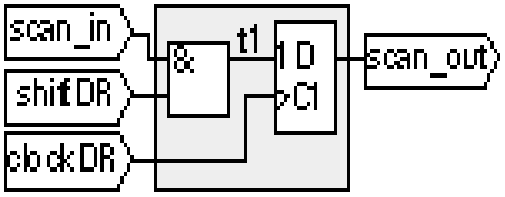


FIGURE 5.26 A BR (bypass register) cell.

Figure 5.27 shows an **instruction-register cell** (**IR cell**) The IR cell inputs are: scan_in , data_in ; as well as clock, shift, and update signals (with names and functions similar to those of the corresponding signals in the BSC). The reset signals are nTRST and reset_bar (active-low signals often use an asterisk, reset* for example, but this is not a legal VHDL name). The two LSBs of data_in must

permanently be set to '01' (this helps in checking the integrity of the scan chain during testing). The remaining data_in bits are status bits under the control of the designer. The update sequential element (sometimes called the **shadow register**) in each IR cell may be set or reset (depending on reset_value). The IR cell outputs are: data_out (the instruction bit passed to the instruction decoder) and scan_out (the data passed to the next IR cell in the IR).

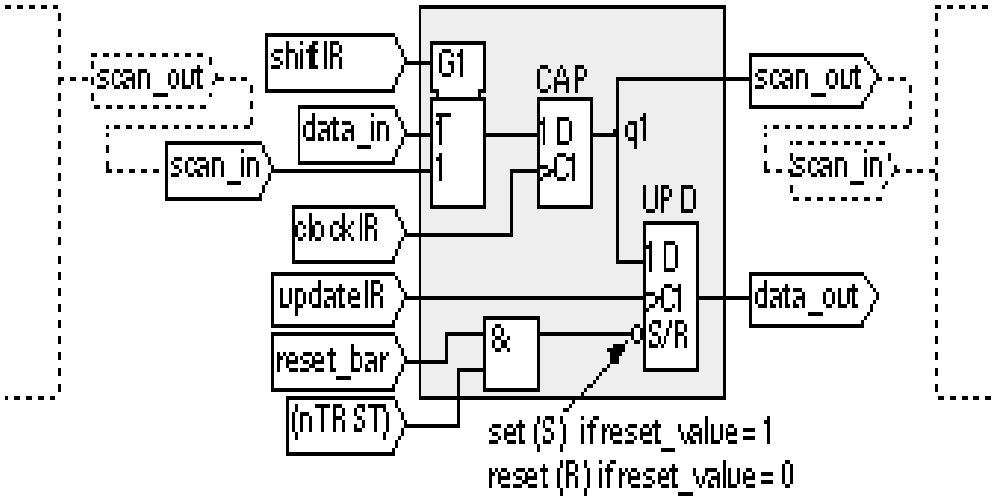


FIGURE 5.27 An IR (instruction register) cell.

6.14.2 BST Registers

Figure 5.28 shows a **boundary-scan register** (**BSR**), which consists of a series connection, or chain, of BSCs. The BSR surrounds the ASIC core logic and is connected to the I/O pad cells. The BSR monitors (and optionally controls) the inputs and outputs of an ASIC. The direction of information flow is shown by an arrow on each of the BSCs . The control signal, mode , is decoded from the IR. Signal mode is drawn as common to all cells for the BSR , but that is not always the case.

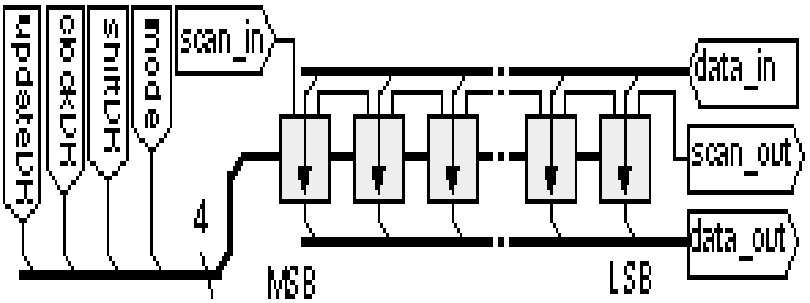


FIGURE 5.28 A BSR (boundary-scan register).

Figure 5.29 shows an **instruction register** (**IR**), which consists of at least two IR cells connected in series. The IEEE 1149.1 standard specifies that the IR cell is reset to '00...01' (the optional IDCODE instruction). If there is no IDCODE TDR, then the IDCODE instruction defaults to the BYPASS instruction.

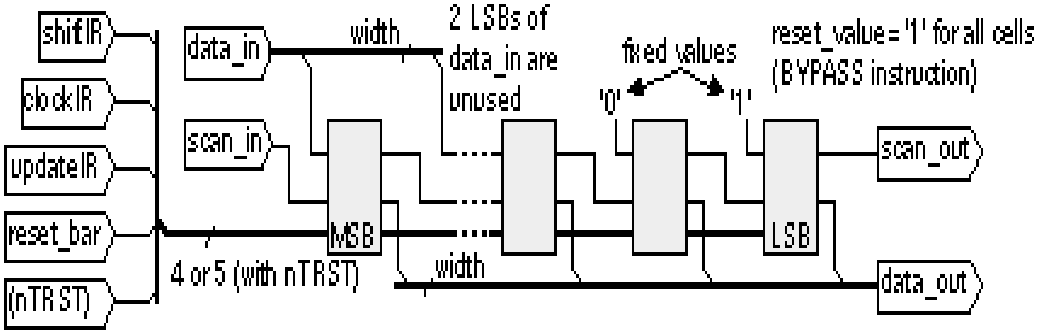


FIGURE 5.29 An IR (instruction register).

6.14.3 Boundary scan Instructions

1. **EXTEST** , external test. Drives a known value onto each output pin to test connections between ASICs.

2. **SAMPLE/PRELOAD** (often abbreviated to SAMPLE). Performs two functions: first sampling the present input value from input pad during capture; and then preloading the BSC update register output during update (in preparation for an EXTEST instruction, for example).

3. **IDCODE** . An optional instruction that allows the **device-identification register** (IDCODE) to be shifted out. The IDCODE TDR is an optional register that allows the tester to query the ASIC for the manufacturer's name, part number, and other data that is shifted out on TDO. IDCODE defaults to the BYPASS instruction if there is no IDCODE TDR.

4. **BYPASS** . Selects the single-cell bypass register (instead of the BSR) and allows data to be quickly shifted between ASICs.

The IEEE 1149.1 standard predefines additional optional instructions and also defines the implementation of custom instructions that may use additional TDRs.

6.14.4 TAP Controller

Test Access Port Controller is a 16 state finite machine .Figure 5.30 shows the TAP controller finite-state machine. The 16-state diagram contains some symmetry: states with suffix '_DR' operate on the data registers and those with suffix '_IR' apply to the instruction register. All transitions between states are determined by the TMS (test mode select) signal and occur at the rising edge of TCK , the boundary-scan clock. An optional active-low reset signal, nTRST or TRST* , resets the state machine to the initial state, Reset . If the dedicated nTRST is not used, there must be a power-on reset signal (POR)—not an existing system reset signal.
The outputs of the TAP controller are not shown in Figure 5.30, but are derived from each TAP controller state. The TAP controller operates rather like a four-button digital watch that cycles through several states (alarm, stopwatch, 12 hr / 24 hr, countdown timer, and so on) as you press the buttons. Only the shaded states in Figure 5.30 are the ASIC core logic; the other states are intermediate steps. The pause states let the controller jog in place while the tester reloads its memory with a new set of test vectors.

When nTRST = 0, the state machine comes to the reset state and if TMS='0' it enters to run idle mode. Now if TMS ='1' for 1 cycles (TCK) the FSM enters in to select DR state, in this state if TMS is set to 0 it goes to capture_DR state ,if set to 1 it goes to Select_IR state .in this manner the FSM traverse to these states depending upon the TMS,nTRST and TCK.

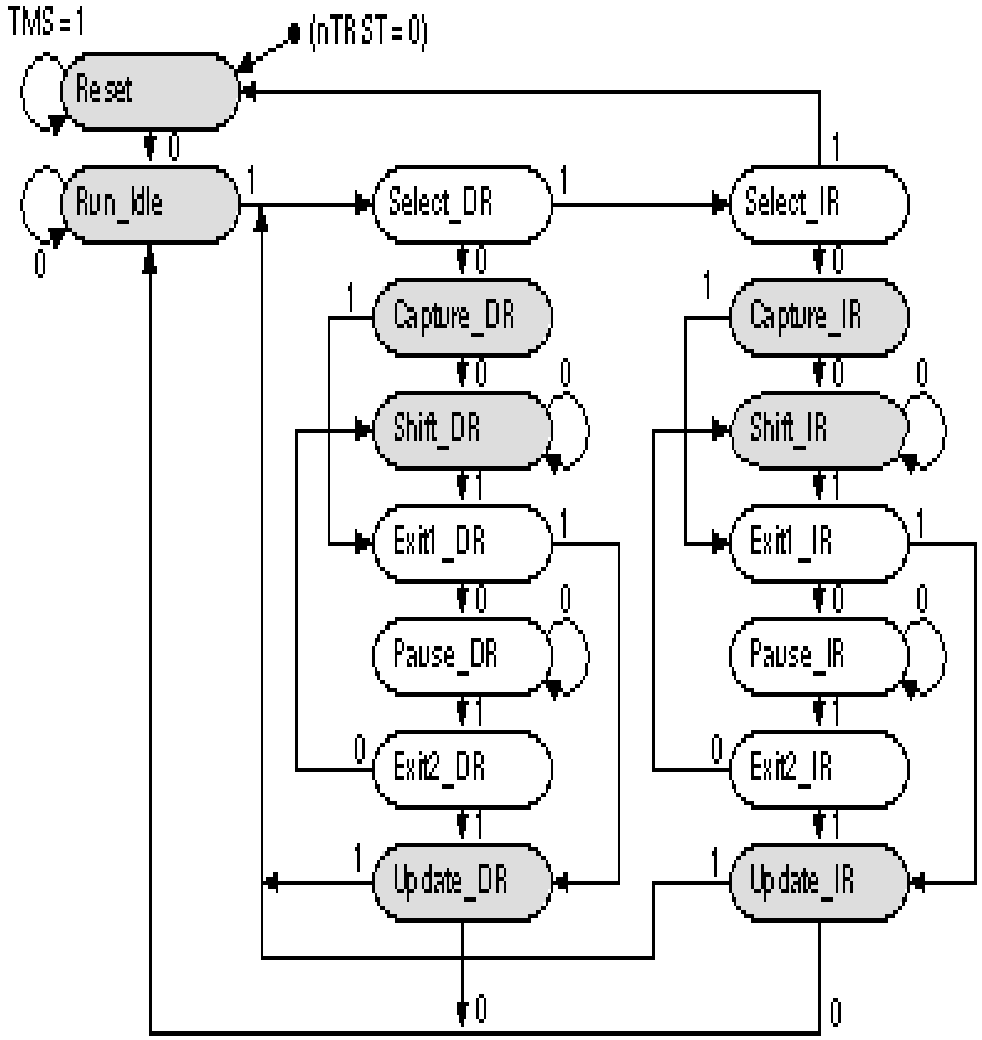


Figure 5.30 TAP Controller

Exercise

Chapter 1.

1. Classify the chips according to the gate density.
2. Describe VLSI Design flow and abstraction level.
3. Explain the design hierarchy.
4. Briefly summarize semiconductor technology.
5. Distinguish Bipolar and MOS transistor.
6. Draw stick diagram for inverter.
7. What do you understand by pull up and pull down device.
8. What are the various form of pull up device?

Chapter 2

1. Briefly explain about silicon semiconductor technology.
2. Name the various cmos process technology.
3. Explain pwell and nwell cmos process technology.
4. Explain twin tub cmos process in details.
5. Explain the SOI technology and its advantage over well based process technology.
6. Can resistor and capacitor be formed with mos technology if yes how?
7. Explain the construction and working principle of EPROM.
8. Explain bicmos process and its advantage.
9. What do you understand by layout design rule ,explain few rules for cmos lambda based design rule.
10. What is physical design; draw the mask layout diagram for
(a) Nmos Inverter
(b) Cmos inverter
(c) mos NAND gate
(d) Cmos NOR gate

11. Explain how VLSI design can be implemented with CAD Tools.

CHAPTER 3

1. Classify the types of Mos transistor.
2. What is the difference between enhancement and depletion type of transistor.
3. How nmos transistor differs from pmos transistor.
4. Explain the construction of enhancement and depletion type of transistor
5. Explain the working principle of enhancement and depletion type transistor.
6. Explain the accumulation, depletion and inversion mode in mos structure.
7. Derive the relation of current and voltage for mos transistor.
8. Explain the region of operation of nmos enhancement type of transistor.
9. What is threshold voltage; explain the equations related to threshold voltage.
10. Define body effect and its impact on threshold voltage.
11. Expalin various second order effects in detail.
12. Explain mos ac model, and derive channel resistance and transconductance in linear and saturation region.
13. Draw the cmos logic for inverter circuit.
14. Expalin the operation of cmos inverter.
15. Draw the equivalent circuit diagram for cmos inverter in different region.
15. Plot the cmos inverter transfer characteristics and describe all the regions with suitable equations.
16. Explain noise margin in detail.
17. Derive the expression for rise and fall time for cmos inverter.
18. Explain the various types of power dissipation in cmos .what is the total power dissipation in terms of equation.
19. Explain the working of transmission gate, why it is superior to pass transistor.
20. What is tristate inverter?

Chapter 4

1. What is HDL?how it differs from normal programming language?
2. Define module and port?
3. What is identifier, how can it be created?
4. Expalin various gate primitives in verilog.
5. Explain Gate delay with appropriate example.
6. What do you understand by structural level modeling?
7. Write the structural verilog code for

(a) Full adder,
(b) D flip-flop,

(c) 2 to 4 decoder,
(d) 4 to 1 multiplexer,
(e) 4 to 2 priority encoder
(f) 4 bit ripple carry adder
(g) 2 bit equality detector

8. Expalin the various operators used in verilog.
9. Expalin the switch level modeling with example.
10. Write the switch level code for cmos transmission gate.
11. What is behavioral modeling?
12. Explain how sequential operation is performed in verilog.
13. What are blocking and non blocking statements?
14. Expalin various conditional statements in verilog.
15. What is the use of initial, always block in verilog.
16. Write the behavioral code for
(A) 3 to 8 decoder
(b) 4 to 1 multiplexer
(c) d latch and flip flop
(d) 4 bit adder /subtractor.
(e) A simple 8 bit ALU which can perform addition, subtraction, multiplication, and, or, invert, xor, buffer function

CHAPTER 5

1. Draw the cmos logic for
   - Inverter,
   - AND gate ,
   - NAND gate,
   - NOR gate,
   - Xor gate

2. Draw the cmos logic for
   - **F = (A.B+C.D)'**
   - **F = (A+B.C)'**
   - **F = ((A.B+C).D)'**
   - **F = ((A+B).(C+D))'**
3. Design 2 to 1 Multiplexer using cmos logic. And explain its operation.
4. Explain D latch using cmos Logic with suitable diagrams.

5. Explain D flip-Flop using Cmos logic and its operation.
6. What are the various chip design options?
7. Explain ASIC design in detail.
8. What are the types of ASIC?
9. What do you mean by PLD?
10. Explain in detail
    - Full custom ASIC ,
    - Standard cell based ASIC ,
    - Gate Array based Asic,
11. Explain ASIC Design Flow In detail.
12. Explain the architecture of XILINX 3000 FPGA.

CHAPTER 6

1. What is the need and importance of testing the chip?
2. What is manufacturing and functional test?
3. What are the level at which chips are undergone for test.
4. What is fault?
5. What is stuck at fault, explain stuck at 0, stuck at 1, and stuck open and short faults.
6. can combinational circuit behave as sequential if stuck at fault occur if yes explain with suitable example.
7. What are the different types of fault simulation?
8. What is reliability and yield?
9. Explain the concept of ATPG with suitable example.
10. What is reconvergent fan out in ATPG and how to avoid it?
11. What is design for testability, controllability and observability?
12. What is scan based test explain in detail.
13. What are the various approaches that are used for DFT.
14. What is PRSG? How it can be useful for testing the chip.
15. Explain built in self test in detail.
16. Expalin boundary scan test in detail.